Cover

MVME197BUG
Diagnostic Firmware
(Part 1 of 2)

34 pages
1/8″ spine

® ™

36 - 84 pages
3/16″ & 1/4″ spine

® ™

86 - 100 pages
5/16″ spine

® ™

102 - 180 pages
3/8″ - 1/2″ spine

® ™

182 - 308 pages
5/8″ - 1 1/8″ spine
2 lines allowed

MVME197BUG Diagnostic Firmware (Part 1 of 2)

MVME197BUG Diagnostic Firmware (Part 1 of 2)

V197DIAA1/UM1

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

Preface

This manual provides general information and a description of the diagnostic firmware for the MVME197BUG (197Bug) Debugging Package.

Use of the MVME197 debugger, the debugger command set, use of the one-line assembler/ disassembler, and system calls for the Debugging Package are all described in the *MVME197BUG (197Bug) Debugging Package User's Manual*.

Note

This document is bound in two parts:

Part 1 (*this volume, V197DIAA1/UM1*) contains Chapters 1 and 2, and the first half of Chapter 3 (pages 3-1 through 138, covering test suites RAM, RAMCDIx, RAMCD, ECDM, DCAM, BSW, XCAx, and MK48T0x).

Part 2 (V197DIAA2/UM1) contains the rest of Chapter 3 (pages 3-139 through 3-302, covering test suites PPC2, CD2401, VME2, LANC, and NCR), Chapter 4, and Appendix A.

The table of contents and index are duplicated in Parts 1 and 2.

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 found in the following pages.

Manual Terminology

Throughout this manual, a convention has been maintained whereby data and address parameters are preceded by a character which specifies the numeric format, as follows:

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

For example, "12" is the decimal number twelve, and "$12" is the decimal number eighteen. Unless otherwise specified, all address references are in hexadecimal throughout this manual.

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

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

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

Data and address sizes are defined as follows:

- ❏ A *byte* is eight bits, numbered 0 through 7, with bit 0 being the least significant.
- ❏ A two-byte is 16 bits, numbered 0 through 15, with bit 0 being the least significant. For the MVME197 and other RISC modules, this is called a *half-word*.
- ❏ A four-byte is 32 bits, numbered 0 through 31, with bit 0 being the least significant. For the MVME197 and other RISC modules, this is called a *word*.
- ❏ An eight-byte is 64 bits, numbered 0 through 63, with bit 0 being the least significant. For the MVME197 and other RISC modules, this is called a *double-word*.

Throughout this document, it is assumed that the MPU on the MVME197/MVME297 module series is always programmed with *big-endian byte ordering*, as shown below. Any attempt to use *small-endian byte ordering* will immediately render the MVME197Bug debugger unusable.

BIT			**BIT**
63 56	**55 48**	**47 40**	**39 32**
ADR0	ADR1	ADR2	ADR3
31 24	**23 16**	**15 08**	**07 00**
ADR4	ADR5	ADR6	ADR7

The terms *control bit* and *status bit* are used extensively in this document. The term *control bit* is used to describe a bit in a register that can be set and cleared under software control. The term *true* is used to indicate that a bit is in the state that enables the function it controls. The term *false* is used to indicate that the bit is in the state that disables the function it controls. In all tables, the terms 0 and 1 are used to describe the actual value that should be written to the bit, or the value that it yields when read. The term *status bit* is used to describe a bit in a register that reflects a specific condition. The status bit can be read by software to determine operational or exception conditions.

The following conventions are used in this document:

bold

is used for user input that you type just as it appears. Bold is also used for commands, options and arguments to commands, and names of programs, directories, and files.

italic

is used for names of variables to which you assign values. Italic is also used for comments in screen displays and examples.

`courier`

is used for system output (e.g., screen displays, reports), examples, and system prompts.

<RETURN>

represents the Carriage Return or Enter key.

CTRL

represents the Control key. Execute control characters by holding down the control key while pressing the letter key, e.g., **CTRL-d**.

Related Documentation

The following publications are applicable to the MVME197 module series and may provide additional helpful information. If not shipped with this product, they may be purchased by contacting your Motorola sales office.

Document Title	Motorola Publication Number
MVME197LE Single Board Computer User's Manual	MVME197LE
MVME197LE Single Board Computer Support Information	SIMVME197LE
MVME197DP and MVME197SP Single Board Computers User's Manual	MVME197
MVME197DP and MVME197SP Single Board Computers Support Information	SIMVME197
MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide	MVME197PG
MVME197BUG 197Bug Debugging Package User's Manual	MVME197BUG
MVME712M Transition Module and P2 Adapter Board User's Manual	MVME712M
MVME712-12, MVME712-13, MVME712A, MVME712AM, and MVME712B Transition Module and LCP2 Adapter Board User's Manual	MVME712A
MC88110 Second Generation RISC Microprocessor User's Manual	MC88110UM
MC68040 Microprocessor User's Manual	MC68040UM
MC88410 Secondary Cache Controller User's Manual	MC88410UM

Notes

1. **The support information manuals (SIMVME197LE and SIMVME197) contain: the connector interconnect signal information, parts lists, and the schematics for the specific board indicated.**

2. **Although not shown in the above list, each Motorola Computer Group manual publication number is suffixed with characters which represent the revision level of the document, such as "/D2" (the second revision of a manual); a supplement bears the same number as the manual but has a suffix such as "/A1" (the first supplement to the manual).**

To further assist your development effort, Motorola has collected user's manuals for each of the peripheral controllers used on the MVME197 module series and other boards from the suppliers. This bundle includes manuals for the following:

68-1X7DS for use with the MVME197 series of Single Board Computers.

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

The following non-Motorola publications may also be of interest and may be obtained from the sources indicated. The VMEbus Specification is contained in ANSI/IEEE Standard 1014-1987.

ANSI/IEEE Std 1014-1987 Versatile Backplane Bus: VMEbus	The Institute of Electrical and Electronics Engineers, Incorporated Publication and Sales Department 345 East 47th Street New York, New York 10017-2633 Telephone: 1-800-678-4333
ANSI Small Computer System Interface-2 (SCSI-2), Draft Document X3.131-198X, Revision 10c	Global Engineering Documents P.O. Box 19539 Irvine, California 92713-9539 Telephone (714) 979-8135

Contents

Figures

Tables

197BUG GENERAL INFORMATION 1

Overview of M88000 Firmware

The firmware for the M88000-based (88K) series of board and system level products has a common genealogy, deriving from the BUG firmware currently used on all Motorola M68000-based (68K) CPU modules.

The M88000 firmware family provides:

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

This member of the M88000 Firmware family is implemented on all of the MVME197 series of Single Board Computers, and is known as the MVME197BUG, or just 197Bug.

Description of 197Bug

The 197Bug package, MVME197BUG, is a powerful evaluation and debugging tool for systems built around the MVME197 series of RISC-based (Reduced Instruction Set Computing) microcomputers.

197Bug consists of three parts:

- ❏ A command-driven user-interactive software debugger, described in the *MVME197BUG 197Bug Debugging Package User's Manual*. It is hereafter referred to as "the debugger", "197Bug", or just "Bug"
- ❏ A command-driven diagnostic package for the MVME197 hardware, described in this manual. It is hereafter referred to as "the diagnostics"
- ❏ A user interface which accepts commands from the system console terminal

When using 197Bug, the user operates out of either:

- ❏ The debugger directory or
- ❏ The diagnostic directory

If you are in the debugger directory, then the debugger prompt "**197-Bug>**" is displayed and you have all of the debugger commands at your disposal. If you are in the diagnostic directory, then the diagnostic prompt "**197-Diag>**" is displayed and you have all of the diagnostic commands at your disposal as well as all of the debugger commands.

You may switch between directories by using the Switch Directories (**SD**) command (refer to the *MVME197BUG 197Bug Debugging Package User's Manual*), or may examine the commands in the particular directory that you are currently in by using the Help (**HE**) command (refer to the *MVME197BUG 197Bug Debugging Package User's Manual*).

Because 197Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. The flow of control in 197Bug is explained in the following section. When a command is entered, 197Bug executes the command and the prompt reappears. However, if a command is entered which causes execution of user target code (e.g., "**GO**"), then control may or may not return to 197Bug, depending on the outcome of the user program.

197Bug Implementation

FLASH-Based Debugger

197Bug is contained in the FLASH memory devices located on the MVME197 board. The FLASH devices are electrically re-writable and may be reprogrammed without removing the physical devices from the MVME197 board. This allows you to incorporate updated versions of the 197Bug as they become available by simply loading the newer version into the FLASH memory and overwriting the older version.

The **PFLASH** command (refer to the *MVME197BUG 197Bug Debugging Package User's Manual*) describes how to reprogram the FLASH memory contents. The executable code is checksummed at every power-on or reset firmware entry. You are cautioned against reprogramming of the FLASH memory contents unless rechecksum precautions are taken. Refer to the **CS** command description in the *MVME197BUG 197Bug Debugging Package User's Manual* for checksum information.

WARNING

Reprogramming any portion of FLASH memory will erase everything currently contained in FLASH, including the debugger. A valid version of 197Bug must be transferred from RAM into the FLASH during FLASH reprogramming in order for the debugger to operate.

The *197Bug Debugger Command Set* chapter of the *MVME197BUG 197Bug Debugging Package User's Manual* describes the command set of the FLASH-based debugger.

BOOT ROM

A subset of 197Bug is also programmed into the BOOT ROM, which is an EPROM or One-Time Programmable ROM on the MVME197 module. This scaled-down 197Bug is referred to as the "BootBug", or "197BBug".

When the MVME197 module is reset, control is first given to the code which resides in the BOOT ROM. During normal operation, the BOOT ROM passes control quickly to the debugger residing in the FLASH memory.

It is possible to prevent control from being passed to the FLASH-based debugger and to continue execution of the BOOT ROM code (or BootBug). This may be done by performing a "double-button RESET". Refer to the *Double-Button Reset* section later in this chapter.

The BootBug, due to its limited size, does not support the entire command set of the FLASH-based debugger, but contains enough functionality to enable downloading of object code (by means of the VMEbus, serial port, SCSI bus or the network) and reprogramming of the FLASH memory. Some versions of the BootBug may not contain both network commands (**NIOT**, **NIOP**) and disk/tape commands (**IOT**, **IOP**) because of ROM-space constraints.

The following table lists the debugger commands of the BootBug.

Table 1-1. BootBug Debugger Commands

Command Mnemonic	Command Title	Command Line Syntax
BC	Block of Memory Compare	**BC** *RANGE DEL ADDR* [**; B**|**H**|**W**]
BF	Block of Memory Fill	**BF** *RANGE DEL data* [*DEL increment*] [**; B**|**H**|**W**]
BM	Block of Memory Move	**BM** *RANGE DEL ADDR* [**; B**|**H**|**W**]
BS	Block of Memory Search	**BS** *RANGE DEL TEXT* [**;B**|**H**|**W**] or **BS** *RANGE DEL data DEL* [*mask*] [**;B**|**H**|**W**,**N**,**V**]

Table 1-1. BootBug Debugger Commands (Continued)

Command Mnemonic	Command Title	Command Line Syntax
BV	Block of Memory Verify	**BV** *RANGE DEL data* [*DEL increment*] [**;B**|**H**|**W**]
CS	Checksum	**CS** *RANGE* [**;B**|**H**|**W**]
DC	Data Conversion	**DC** *EXP* | *ADDR* [**;**[**B**] [**O**] [**A**]]
HE	Help on Command(s)	**HE** [COMMAND]
IOP	I/O Physical (Direct Disk Access)	**IOP**
IOT	I/O "TEACH" for Configuring Disk Controller	**IOT** [**;**[**A**|**F**|**H**|**T**]
LO	Load S-Records from Host	**LO** [*PORT*] [*DEL ADDR*] [**;**[**X**][**C**][**T**] [=*text*]
MD	Memory Display	**MD**[**S**] *ADDR* [**:***COUNT*|*DEL ADDR*] [**;** [**B**|**H**|**W**]
MM	Memory Modify	**MM** *ADDR* [**;**[**B**|**H**|**W**] [**A**] [**N**]]
MS	Memory Set	**MS** *ADDR DEL* {*Hexadecimal Number*}/{**'***String***'**}
NIOP	Network I/O Physical	**NIOP**
NIOT	Network I/O Teach	**NIOT** [**;**[**H**]|[**A**]]
NOPF	Port Detach	**NOPF** [*PORT*]
PF	Port Format	**PF** [*PORT*]

Table 1-1. BootBug Debugger Commands (Continued)

Command Mnemonic	Command Title	Command Line Syntax
PFLASH	Program FLASH Memory	**PFLASH** *SSADDR SEADDR DSADDR* [*IEADDR*] [**;**[**A** ǀ **R**] [**X**]] **PFLASH** *SSADDR:COUNT DSADDR* [*IEADDR*] [**;**[**B** ǀ **H** ǀ **W**] [**A** ǀ **R**] [**X**]]
SET	Set Time and Date	**SET** [*mmddyyhhmm*] ǀ [<**+**/**-***CAL*:>**;C**]
TIME	Display Time and Date	**TIME** [**;C** ǀ **L** ǀ **O**]
TM	Transparent Mode	**TM** [*PORT*] [*DEL ESCAPE*]
VE	Verify S-records Against Memory	**VE** [*n*] [*ADDR*] [**;X** ǀ **C**] [=*text*]

Detailed descriptions of these commands may be found in the *MVME197BUG 197Bug Debugging Package User's Manual*.

The BootBug contains two additional commands that are not found in the command set of the FLASH-based debugger. These are the **SETUP** command and the **EXEC** command. The following table lists these two commands.

Table 1-2. SETUP and EXEC Commands

Command Mnemonic	Command Title	Command Line Syntax
EXEC	Execute User Program	**EXEC** [*ADDR*]
SETUP	Setup System Parameters	SETUP

Before using some of the features of the BootBug, some parameters may need to be defined. Some examples are the SCSI ID, the Ethernet address, the clock speed of the board, and the mapping of the VMEbus. The **SETUP** command has been provided for this purpose. Run this command and answer the prompts to be sure the board is configured properly before using any SCSI, VME, or Ethernet I/O.

Setup System Parameters SETUP

SETUP allows configuring certain parameters that are necessary for some I/O operations (SCSI, VME, and Ethernet). When this command is executed, the operator is prompted for input after displaying the default value, if any is available.

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

```
197-BBug>setup
MPU Clock Speed       = "4000"?
Ethernet Address      = 08003E21F959?
Local SCSI Identifier = "07"?
VME Slave Enable #1 [Y/N]               = N?
VME Slave Starting Address              = 00000000?
VME Slave Ending Address                = 0000FFFF?
VME Slave Address Translation Address   = 00000000?
VME Slave Address Translation Select    = 00000000?
VME Slave Control                       = 0000?
VME Master Enable [Y/N]                 = Y?
VME Master Starting Address             = 40000000?
VME Master Ending Address               = 4FFFFFFF?
VME Master Address Translation Address  = 00000000?
VME Master Address Translation Select   = 00000000?
VME Master Control                      = 0D?
197-BBug>
```

Execute User Program EXEC [*ADDR*]

The **EXEC** command is used to start code execution at a particular address. Execution is transferred to the specified address (“ADDR”).

Installation and Start-up

Even though the MVME197Bug flash memory devices are installed on the MVME197 module, for 197Bug to operate properly with the MVME197, follow this set-up procedure.

CAUTION

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

1. Turn all equipment power OFF. Refer to the board specific MVME197 User's Manual for selecting the configuration switch settings required for the user's particular application.
2. Refer to the set-up procedure for the user's particular chassis or system for details concerning the installation of the MVME197.
3. Connect the terminal which is to be used as the 197Bug system console to the default debug EIA-232-D port at serial port 1 on backplane connector P2 through an MVME712X transition module. Refer to the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for some possible connection diagrams.

   Set up the terminal as follows:

   – eight bits per character
   – one stop bit per character
   – parity disabled (no parity)
   – baud rate 9600 baud (default baud rate of the MVME197 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 197Bug debugger.

Note

In order for high-baud rate serial communication between 197Bug and the terminal to work, the terminal must do some form of handshaking. If the terminal being used does not do hardware handshaking via the CTS line, then it must do XON/XOFF handshaking. If you get garbled messages and missing characters, then you should check the terminal to make sure XON/XOFF handshaking is enabled.

4. If it is desired to connect devices (such as a host computer system and/or a serial printer) to the other EIA-232-D port connectors (marked SERIAL PORTS 2, 3, and 4 on the MVME712X transition module), connect the appropriate cables and configure the port(s) as detailed in step 3 above.

   After power-up, this(these) port(s) can be reconfigured by programming the MVME197 CD2401 Serial Controller Chip (SCC), or by using the 197Bug **PF** command.

Note that the MVME197 also contains a parallel port. To use a parallel device, such as a printer, with the MVME197, connect it to the "printer" port at P2 through an MVME712X transition module. Refer to the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for some possible connection diagrams.

5. Power up the system. 197Bug executes some self-checks and displays the debugger prompt "**197-Bug>**" (if 197Bug is in Board Mode). However, if the **ENV** command has put 197Bug in System Mode, the system performs a self-test and enters the Menu mode. Refer to the **ENV** and **MENU** commands in the *MVME197BUG 197Bug Debugging Package User's Manual*.

   All self-tests are executed unless masked by the Diagnostic Mask command. If a test fails, any error messages are preserved in the Diagnostic error message buffer and execution continues to the next test after issuing a CRLF to preserve the test failure banner. At the conclusion of the self-test, a list of any failed tests will remain on the display and the Display Error Messages (DEM) command may be used from the 197-Diag> prompt to see any detailed error information resulting from the execution of the self-tests.

Autoboot

Autoboot is a 197Bug software routine that provides an independent mechanism for booting an operating system. This autoboot routine automatically scans for controllers and devices in a specified sequence until a valid bootable device containing a boot media is found or the list is exhausted. If a valid bootable device is found, a boot from that device is started. The controller scanning sequence goes from the lowest controller Logical Unit Number (LUN) detected to the highest LUN detected.

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

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

Following this message there is approximately a thirty-second 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, the <BREAK> key or the ABORT or RESET switches can be pressed.

Autoboot is controlled by parameters contained in the **ENV** command. These parameters allow the selection of specific boot devices and files, and allow programming of the Boot delay. Refer to the **ENV** command in the *MVME197BUG 197Bug Debugging Package User's Manual* for more details.

CAUTION

Although streaming tape can be used to autoboot, the same power supply must be connected to the streaming tape drive, controller, and the MVME197. At power-up, the tape controller will position the streaming tape to load point where the volume ID can correctly be read and used.

If, however, the MVME197 loses power but the controller does not, and the tape happens not to be at load point, the sequences of commands required (attach and rewind) cannot be given to the controller and autoboot will not be successful.

ROMboot

This function is configured/enabled by the Environment (**ENV**) command and executed at power-up (optionally also at reset) or by the **RB** command assuming there is valid code in the flash memories (or optionally elsewhere on the module or VMEbus) to support it. If user-supplied ROMboot code is installed, the routine is given control (if the routine meets the format requirements). The **NORB** command disables the function.

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

1. Power must have just been applied (but the **ENV** command can change this to also respond to any reset).
2. The user's routine must be located within the MVME197 ROM memory map (but the **ENV** 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. The user's routine must pass a checksum test, which ensures that this routine was really intended to receive control at power-up.

For complete details on how to use ROMboot, refer to the *MVME197BUG 197Bug Debugging Package User's Manual*.

Network Boot

Network Auto Boot is a 197Bug software routine that provides a mechanism for booting an operating system using a network (local Ethernet interface) as the boot device. The Network Auto Boot routine automatically scans for controllers and devices in a specified sequence until a valid bootable device containing a boot media is found or the list is exhausted. If a valid bootable device is found, a boot from that device is started. The controller scanning sequence goes from the lowest controller Logical Unit Number (LUN) detected to the highest LUN detected.

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

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

Following this message there is approximately a thirty-second 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 Network Boot, you can press the <BREAK> key or the ABORT or RESET switches.

Network Auto Boot is controlled by parameters contained in the **NIOT** and **ENV** commands. These parameters allow the selection of specific boot devices, systems, and files, and allow programming of the Boot delay. Refer to the **NIOT** and **ENV** commands in the *MVME197BUG 197Bug Debugging Package User’s Manual* for more details.

Restarting the System

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

Reset

Pressing and releasing the MVME197 front panel reset switch initiates a system reset (if the board is the System Controller). A system reset also occurs if power is removed from the MVME197 module and then reapplied. Reset is used to restore the system to a known state. The debugger environment (refer to the ENV command description in the *MVME197BUG 197Bug Debugging Package User’s Manual*) is restored to the user-selected defaults, which had been stored to NVRAM and were preserved through the reset.

COLD and WARM reset modes are available and are selected using the **RESET** command. By default, 197Bug is in COLD mode (refer to the **RESET** command description in the *MVME197BUG 197Bug Debugging Package User's Manual*).

During COLD reset, a total system initialization takes place, as if the MVME197 had just been powered up:

- ❏ All static variables (including disk device and controller parameters) are restored to their default states
- ❏ The breakpoint table and offset registers are cleared
- ❏ The target registers are invalidated. Input and output character queues are cleared
- ❏ Onboard devices are reset
- ❏ The first two serial ports are reconfigured to their default state

During WARM reset, the 197Bug variables and tables are preserved, as well as the target state registers and breakpoints.

Reset must be used if the processor ever halts, or if the 197Bug environment is ever lost (vector table is destroyed, stack corrupted, etc.).

Double-Button Reset

Immediately after reset, control is given to the BootBug. During normal operation, the BootBug quickly passes control to the full debugger which resides in FLASH memory. This code then initializes the board and the debugger environment from the user-specified defaults. In some cases, however, it is desirable to not follow this normal flow after reset. This may be the case if the FLASH memory has been corrupted or if the user-specified parameters in the NVRAM prevent proper board initialization.

It is possible to prevent control from being passed to the FLASH-based debugger and to continue execution of the BOOT ROM code (or BootBug). This may be done by performing a "double-button RESET". Press both the **ABORT** and **RESET** push-buttons simultaneous and then release the RESET push-button but continue to press the ABORT push-button for approximately 2 seconds before releasing.

The BootBug banner will appear and a prompt message will query the user whether to transfer control to the FLASH-based debugger (with the factory-programmed defaults for board initialization and debugger environment):

```
Copyright Motorola Inc. 1993, All Rights Reserved
MVME197 Boot Debugger Release Version 0.5 - 08/09/93
Continue in Debugger with Double Button Reset? (N/Y)?
```

If the answer is “Y”, control will be passed to the FLASH version of the debugger and the user-selected defaults, stored in NVRAM, will not be used. This can be helpful if invalid parameters have inadvertently been written to NVRAM, thus interfering with normal start-up.

If the answer is “N”, then control is not passed to the FLASH-based debugger. Instead the BootBug initializes the board and issues its own prompt:

```
197-BBug>
```

It is helpful to remain in the BootBug if the FLASH has been mis-programmed and normal RESET causes the board to hang.

The BootBug, due to its limited size, does not support the entire command set of the FLASH-based debugger, but contains enough functionality to enable downloading of object code by means of the VMEbus, serial port, SCSI bus, or the network and reprogramming of the FLASH memory. Some versions of the BootBug may not contain both network commands (**NIOT** and **NIOP**) and disk/tape commands (**IOT** and **IOP**) because of ROM-space constraints.

Abort

Abort is invoked by pressing and releasing the ABORT switch on the MVME197 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 following:

❏ Instruction Pointer (IP)

❏ Status register

❏ 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 IP, register contents, etc., help to pinpoint the malfunction.

Pressing and releasing the ABORT switch causes the following:

- ❏ Interrupt to the microprocessor
- ❏ The target registers, reflecting the machine state at the time the ABORT switch was pressed, are displayed on the screen
- ❏ Any breakpoints installed in the user code are removed
- ❏ The breakpoint table remains intact
- ❏ Control is returned to the debugger

Break

A “Break” is generated by pressing and releasing the BREAK key on the terminal keyboard. Break does not generate an interrupt. The only time break is recognized is when characters are sent or received by the console port.

Break performs the following:

- ❏ Removes any breakpoints in the user code
- ❏ Keeps the breakpoint table intact
- ❏ Takes a snapshot of the machine state if the function was entered using SYSCALL
- ❏ Allows access to the snapshot for diagnostic purposes

Many times it is desired to terminate a debugger command prior to its completion, for example, the display of a large block of memory. Break normally allows the user to terminate the command.

SYSFAIL* Assertion/Negation

Upon a reset/power up condition the debugger asserts the VMEbus SYSFAIL* line (refer to the VMEbus specification). SYSFAIL* stays asserted if any of the following has occurred:

- ❏ Confidence test failure
- ❏ NVRAM checksum error
- ❏ Local memory configuration status
- ❏ Self test (if system mode) has completed with error
- ❏ MPU clock speed calculation failure

After debugger initialization is done and any of the above situations has not occurred, the SYSFAIL* line is negated. This indicates to the user or VMEbus masters the state of the debugger. In a multi-computer configuration, other VMEbus masters could view the pertinent control and status registers to determine which CPU is asserting SYSFAIL*. SYSFAIL* assertion/negation is also affected by the **ENV** command. Refer to the *MVME197BUG 197Bug Debugging Package User's Manual*.

MPU Clock Speed Calculation

The clock speed of the microprocessor is calculated and checked against a user definable parameter housed in NVRAM (refer to the **ENV** command). If the check fails, a warning message is displayed. The calculated clock speed is also checked against known clock speeds and tolerances.

Memory Requirements

The program portion of 197Bug is approximately 1 megabyte of code, consisting of download, debugger, and diagnostic packages and contained entirely in the flash memory. The flash memory on the MVME197 is mapped starting at location $FF800000.

197Bug requires a minimum of 64K bytes of contiguous read/write memory to operate. The **ENV** command controls where this block of memory is located. Regardless of where the onboard RAM is located, the first 64K bytes is used for 197Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME197 is reset, the target IP is initialized to the address corresponding to the beginning of the user space, and the target stack pointers are initialized to addresses within the user space, with the target Pseudo Stack Pointer (R31) set to the top of the user space.

Terminal Input/Output Control

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

Note

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

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

When observing output from any 197Bug 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 197Bug but may be changed by the user using 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

197Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 197Bug consist of:

- ❏ Command-level disk operations
- ❏ Disk I/O system calls (only via one of the TRAP #496 instructions) for use by user programs
- ❏ Defined data structures for disk parameters

Parameters such as the:

- ❏ Address where the module is mapped
- ❏ Type of device
- ❏ Number of devices attached to the controller module

are kept in tables by 197Bug. 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 197Bug as a storage area divided into logical blocks. By default, the logical block size is set to 256 bytes for every block device in the system. The block size can be changed on a per device basis with the **IOT** command.

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

When a disk transfer is requested:

- ❏ The start and size of the transfer is specified in blocks
- ❏ 197Bug translates this into an equivalent sector specification
- ❏ Sector specification is passed on to the controller to initiate the transfer

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

Device Probe Function

A device probe with entry into the device descriptor table is done whenever a specified device is accessed. This happens when system calls:

- ❏ .DSKRD
- ❏ .DSKWR
- ❏ .DSKCFIG
- ❏ .DSKFMT
- ❏ .DSKCTRL

or debugger commands:

- ❏ **BH**
- ❏ **BO**

- ❏ **IOC**
- ❏ **IOP**
- ❏ **IOT**
- ❏ **MAR**
- ❏ **MAW**

are used.

The device probe mechanism utilizes the SCSI commands **Inquiry** and **Mode Sense**. If the specified controller is non-SCSI, the probe simply returns a status of **device present and unknown**. The device probe makes an entry into the device descriptor table with the pertinent data. After an entry has been made, the next time a probe is done it simply returns with **device present** status (pointer to the device descriptor).

Disk I/O via 197Bug Commands

These following 197Bug commands are provided for disk I/O. Detailed instructions for their use are found in the *MVME197BUG 197Bug Debugging Package User's Manual*. When a command is issued to a particular controller LUN and device LUN, these LUNs are remembered by 197Bug so that the next disk command defaults to use the same controller and device.

IOI (Input/Output Inquiry)

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

IOP (Physical Input/Output to Disk)

The **IOP** command allows the user to:

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

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

IOT (Input/Output Teach)

The **IOT** command allows the user to:

- ❏ Change any configurable parameters

- ❑ Change device attributes
- ❑ See the controllers available in the system

IOC (Input/Output Control)

The **IOC** command allows the user to send command packets as defined by the particular controller directly. **IOC** can also be used to look at the resultant device packet after using the IOP command.

BO (Bootstrap Operating System)

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

BH (Bootstrap and Halt)

BH reads an operating system or control program from a specified device into memory, and then returns control to 197Bug. It is used as a debugging tool.

Disk I/O via 197Bug System Calls

All operations that actually access the disk are done directly or indirectly by 197Bug TRAP #496 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 do disk I/O operations:

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

Refer to the *MVME197BUG 197Bug Debugging Package User's Manual* for information on using these and other system calls.

To perform a disk operation, 197Bug 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 operations:

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

Refer to the descriptions in the *System Calls* chapter in the *MVME197BUG 197Bug 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 be given vary from controller to controller. The disk driver module for the particular hardware module (on- or off-board) must take the standardized packet given to a trap function and create a new packet which is specifically tailored for the disk drive controller it is sent to. Refer to documentation on the particular controller module for the format of its packets, and for using the **IOC** command.

Default 197Bug Controller and Device Parameters

197Bug initializes the parameter tables for a default configuration of controllers and devices. If the system needs to be configured differently than this default configuration (for example, to use a drive that is different from the default), then the user must change these tables.

There are three ways to change the parameter tables:

- ❑ Using **BO** or **BH**. When you invoke one of these commands, the configuration area of the disk is read and the parameters corresponding to that device are rewritten according to the parameter information contained in the configuration area. This is a temporary change. If a cold-start reset occurs, then the default parameter information is written back into the tables.
- ❑ Using **IOT**. You can use this command to manually reconfigure the parameter table for any controller and/or device that is different from

the default. This is also a temporary change and is overwritten if a cold-start reset occurs.

- ❏ Obtain the source. You may change the configuration files and rebuild 197Bug so that it has different defaults. Changes made to the defaults are permanent until changed again.

Disk I/O Error Codes

197Bug returns an error code if an attempted disk operation is unsuccessful.

Network I/O Support

The Network Boot Firmware provides the capability to boot the CPU through the ROM debugger using a network (local Ethernet interface) as the boot device.

The booting process is executed in two distinct phases:

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

The following figure depicts the various modules (capabilities) and the dependencies of these modules that support the overall network boot function. They are described in the following paragraphs.

Physical Layer Manager Ethernet Driver

This driver manages/surrounds the Ethernet controller chip or board. Management is in the scope of the reception of packets, the transmission of packets, receive buffer flushing, and interface initialization.

This module ensures that the packaging and unpackaging of Ethernet packets is done correctly in the Boot PROM.

UDP/IP Protocol Modules

The Internet Protocol (IP) is designed for use in interconnected systems of packet-switched computer communication networks. The Internet protocol provides for transmitting of blocks of data called datagrams (hence User Datagram Protocol, or UDP) from sources to destinations, where sources and destinations are hosts identified by fixed length addresses.

The UDP/IP protocols are necessary for the TFTP and BOOTP protocols. TFTP and BOOTP require a UDP/IP connection.

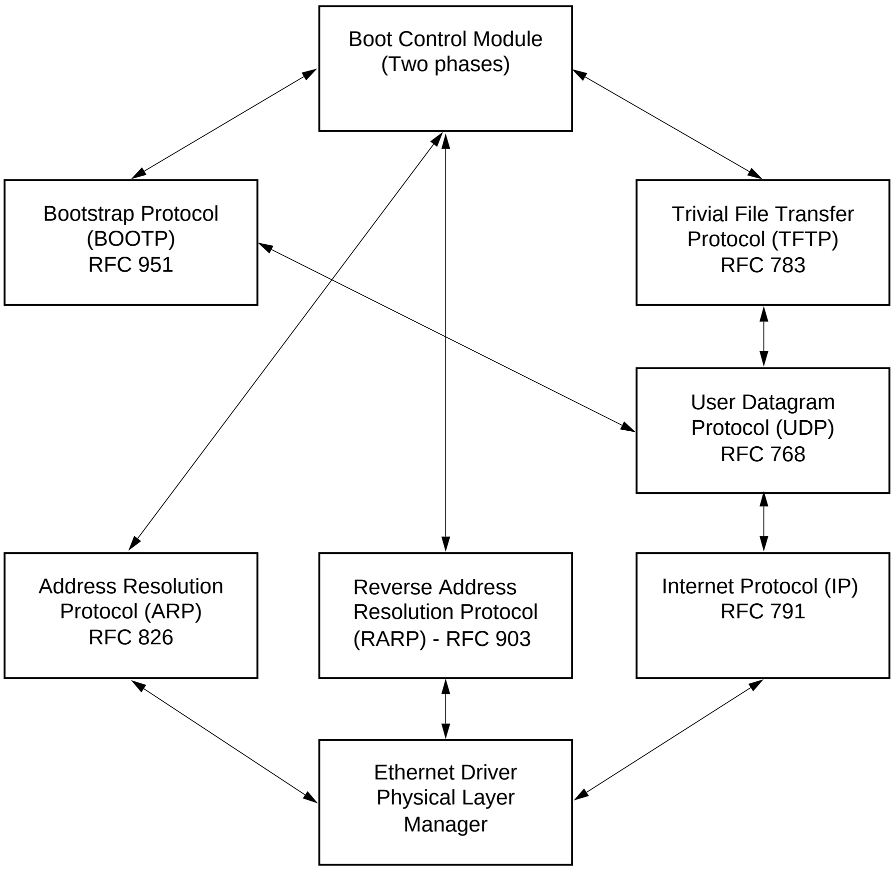

Figure 1-1. Network Boot Modules

RARP/ARP Protocol Modules

The Reverse Address Resolution Protocol (RARP) basically consists of an identity-less node broadcasting a “whoami” packet onto the Ethernet, and waiting for an answer. The RARP server fills an Ethernet reply packet up with the target's Internet Address and sends it.

The Address Resolution Protocol (ARP) basically provides a method of converting protocol addresses (e.g., IP addresses) to local area network addresses (e.g., Ethernet addresses). The RARP protocol module supports systems which do not support the BOOTP protocol (next paragraph).

BOOTP Protocol Module

The Bootstrap Protocol (BOOTP) basically allows a diskless client machine to discover its own IP address, the address of a server host, and the name of a file to be loaded into memory and executed.

TFTP Protocol Module

The Trivial File Transfer Protocol (TFTP) is a simple protocol to transfer files. It is implemented on top of the Internet User Datagram Protocol (UDP or Datagram) so it may be used to move files between machines on different networks implementing UDP. The only thing it can do is read and write files from/to a remote server.

Network Boot Control Module

The "control" capability of the Network Boot Control Module is needed to tie together all the necessary modules (capabilities) and to sequence the booting process. The booting sequence consists of two phases: the first phase is labeled "address determination and bootfile selection" and the second phase is labeled "file transfer". The first phase will utilize the RARP/BOOTP capability and the second phase will utilize the TFTP capability.

Network I/O Error Codes

197Bug returns an error code if an attempted network operation is unsuccessful. Refer to Appendix H of the *MVME197BUG 197Bug Debugging Package User's Manual* for an explanation of network I/O error codes.

Multiprocessor Support

The MVME197 dual-port RAM feature makes the shared RAM available to remote processors as well as to the local processor. This can be done by either of the following two methods. Either method can be enabled/disabled by the **ENV** command as its Remote Start Switch Method.

Multiprocessor Control Register (MPCR) Method

A remote processor can initiate program execution by the local MVME197 MPU which is running the debugger by issuing a remote **GO** command using the Multiprocessor Control Register (MPCR). The MPCR, located at shared RAM location of $3000 offset from the base address of the debugger RAM work area (normally $0), contains one of two words used to control communication between processors.

The MPCR contents are organized as follows:

$3000	*	N/A	N/A	N/A	(MPCR)

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

– Status set (by the MVME197 MPU)

– Commands set (by the remote processor)

The status codes that may be returned from the MVME197 MPU are:

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

The user can only program FLASH memory by the MPCR method. Refer to the **.PFLASH** system call in the *MVME197BUG 197Bug Debugging Package User's Manual* for a description of the FLASH memory program control packet structure.

The command codes that may be set by the remote processor are:

ASCII	G	(HEX 47)	Use Go Direct (**GD**) logic specifying the MPAR address.
ASCII	P	(HEX 50)	Program FLASH memory. The MPAR is set to the address of the FLASH memory program control packet.
ASCII	B	(HEX 42)	Install breakpoints using the Go (**G**) logic.

The Multiprocessor Address Register (MPAR), located in shared RAM location of $3004 offset from the base address of the debugger RAM work area, contains the second of two words used to control communication between processors. The MPAR contents specify the address at which execution for the remote processor is to begin if the MPCR contains a G or B.

The MPAR is organized as follows:

$3004	*	*	*	*	(MPAR)

At power up, the debug monitor self-test routines initialize RAM, including the memory locations used for multi-processor support ($3000 through $3007).

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 a terminal is connected, the MPCR is polled for the same purpose while the serial port is being polled for user input.

Even if no terminal is connected to the MVME197’s console port, the MPCR is still polled to see whether an external processor wishes to re-direct execution of the MVME197’s MPU, possibly to another program which has been loaded into RAM.

An ASCII G placed in the MPCR by a remote processor indicates that the Go Direct type of transfer is requested. An ASCII B in the MPCR indicates that breakpoints are to be armed before control is transferred (as with the **GO** command).

In either sequence, an E is placed in the MPCR to indicate that execution is underway just before control is passed to the new address. (Any remote processor could examine the MPCR contents.)

If the code being executed is to reenter the debug monitor, a TRAP #496 call using function $0063 (SYSCALL .RETURN) returns control to the monitor with a new display prompt. Note that every time the debug monitor returns to the prompt, an R is moved into the MPCR to indicate that control can be transferred once again to a specified RAM location.

GCSR Method

A remote processor can also re-direct program execution by the MVME197 MPU by issuing a remote **GO** command using the VMEchip2 Global Control and Status Register (GCSR). The remote processor places the MVME197 execution address in general purpose registers 0 and 1 (GPCSR0 and GPCSR1). The remote processor then sets bit 8 (SIG0) of the VMEchip2 LM/SIG register. This causes the MVME197 to install breakpoints and begin execution. The result is identical to the MPCR method (with status code B) described in the previous section.

The GCSR registers are accessed in the VMEbus short I/O space. Each general purpose register is two bytes wide, occurring at an even address. The general purpose register number 0 is at an offset of $8 (local bus) or $4 (VMEbus) from the start of the GCSR registers. The local bus base address for the GCSR is $FFF40100. The VMEbus base address for the GCSR depends on the group select value and the board select value programmed in the Local Control and Status Registers (LCSR) of the MVME197. The execution address is formed by reading the GCSR general purpose registers in the following manner:

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

The address appears as:

GPCVSR0	GPCSR1

Diagnostic Facilities

Included in the 197Bug package is a complete set of hardware diagnostics intended for testing and troubleshooting of the MVME197. To use the diagnostics, you must switch directories to the diagnostic directory. If in the debugger directory, you can switch to the diagnostic directory by entering the debugger command Switch Directories (**SD**). The diagnostic prompt ("**197-Diag>**") should appear. Refer to the *Diagnostic Firmware* chapter for complete descriptions of the diagnostic routines available and instructions. 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.

DIAGNOSTIC FIRMWARE 2

Scope

This diagnostic guide contains information about the MVME197 Diagnostic Firmware Package, hereafter referred to as "the diagnostics". This chapter first describes how to invoke the diagnostic utilities and tests. The *Utilities* section then describes the utilities.

The diagnostic tests are described in the *Test Descriptions* chapter.

Overview of Diagnostic Firmware

The MVME197 diagnostic firmware package resides in flash memory which has been programmed on the MVME197. These flash memory devices (which also contain 197Bug) contain a complete diagnostic monitor along with a battery of utilities and tests for exercise, test, and debug of hardware in the MVME197 environment. The diagnostics are menu driven for ease of use. The Help (**HE**) command displays a menu of the diagnostic functions; i.e., the tests and utilities. Several tests have a sub-test menu which may be called using the **HE** command. In addition, some utilities have subfunctions, and as such have subfunction menus.

System Start-up

Refer to the *197Bug General Information* chapter for information on system start-up.

Design Requirements

The design requirements for the diagnostic firmware are as follows.

Assembly Language

Low-level hardware interface code is written in assembly language to control the hardware exactly. Where possible, the C programming language is used to speed coding and improve readability and portability.

Bug Interface

The diagnostic package is combined with the 197Bug, but the interface between these programs is minimal and is well defined. This facilitates independent development and modification.

Compatibility

Although the MVME197 diagnostic package contains a totally new test set, the user interface is similar to existing diagnostic packages. A user familiar with a previous package would be able to use this test set without study.

Menu Driven

The user interface is menu driven. A Help (**HE**) command is available for each test or set of tests in the package.

Diagnostic Monitor

The diagnostic tests are called, commands are input, and results are reported by a common diagnostic monitor (the system monitor used for 197Bug), hereafter called *monitor*. The monitor is command line driven and provides input/output facilities, command parsing, error reporting, interrupt handling, and a multi-level directory for menu selection.

Monitor Start-up

When the monitor is first brought up, either by powering up or by pushing the RESET switch, it displays a message on the diagnostic video display terminal (port 1 terminal).

```
Copyright Motorola Inc. 1988 - 199x, All Rights Reserved

MVME197 Debugger/Diagnostics Release Version X.X - XX/XX/XX
COLD Start

Local Memory Found =XXXXXXXX (&XXXXXXXXX)

MPU Clock Speed =XXMhz

197-Bug>
```

If after a delay, the monitor begins to display test result messages on the bottom line of the screen in rapid succession, the board is in the Bug system mode. If this is not the desired mode of operation, then press the ABORT switch. When the menu is displayed, enter a **3** to go to the system debugger. The environment may be changed by using the Set Environment to Bug/Operating System (**ENV**) command. Refer to the *MVME197BUG 197Bug Debugging Package User's Manual* and the *197Bug General Information* chapter of this guide for details of Bug operation in the system mode.

At the “197-Bug>” prompt, enter **SD** to switch to the diagnostics directory. The Switch Directories (**SD**) command is described elsewhere in this chapter. The prompt should now read “197-Diag>”.

Command Entry and Directories

Commands may be entered when the prompt “197-Diag>” appears. The name (mnemonic) for the command is entered before pressing the carriage return <**CR**>. Multiple commands may be entered. If a command expects parameters and another command is to follow it, separate the two with a semicolon (**;**).

For instance, to invoke the command **RAM ADR** after the command **VME2 REGB**, the command line would read **VME2 REGB ; RAM ADR**. Spaces are not required but are shown here for legibility.

Several commands may be combined on one line. Several commands consist of a command name that is listed in a main (root) directory and a subcommand that is listed in the directory for that particular command. In the main directory are commands like **RAM** and **VME2**. These commands are used to refer to a set of lower level commands. To call up a particular **RAM** test, enter (on the same line) **RAM ADR**. This command causes the monitor to find the **RAM** subdirectory, and then to execute the command (test) **ADR** from that subdirectory.

Examples:

Root-Level Commands:

HE	Help
DE	Display Error Counters

Subdirectory-Level Commands:

RAM ADR	Random Access Memory Tests (DIR), Memory Addressing test
VME2 REGB	VMEchip2 Tests (DIR), Register Walking Bit test

The **RAM** and **VME2** directories in these examples are test group names. If the first part of a command is a test group name, any number of tests from that test group may be entered after the test group name so long as spaces are entered between each test name (and the bug's input buffer size limit is not exceeded).

For instance, to invoke the **RAM** subcommands **CODE** and **BTOG**, the command line would read **RAM CODE BTOG**.

Utilities

In addition to individual or sets of tests, the diagnostic package provides the utilities listed in the next table and described on the following pages. These utilities are root-level commands to the diagnostic monitor and do not require a preceeding test group name.

Table 2-1. Diagnostic Utilities

Mnemonic	Description
AE	All Errors Mode
AEM	Append Error Messages Mode
CEM	Clear Error Messages
CF	Test Group Configuration (cf) Parameters Editor
DE	Display Error Counters
DEM	Display Error Messages
DP	Display Pass Count
HE	Help
HEX	Help Extended
LA	Loop Always Mode
LC	Loop-Continue Mode
LE	Loop-On-Error Mode
LF	Line Feed Suppression Mode
LN	Loop Non-Verbose Mode
MASK	Display/Revise Self Test Mask
NV	Non-Verbose Mode
SD	Switch Directories
SE	Stop-On-Error Mode

Table 2-1. Diagnostic Utilities (Continued)

Mnemonic	Description
ST	Self Test
ZE	Clear (Zero) Error Counters
ZP	Zero Pass Count

All Errors Mode - Command AE

This command requests that diagnostic tests report all errors which can be found. All **RAM** tests support this mode. The command will continue testing after the first error is encountered and report errors to the extent limit set by the size of the error message buffer.

Append Error Messages Mode - Command AEM

This command allows you to accumulate error messages in the internal error message buffer of the diagnostic monitor. The **AEM** command sets the internal append error messages flag of the diagnostic monitor. When the internal append error messages flag is clear, the diagnostic error message buffer is erased (cleared of all character data) before each test is executed. The duration of this command is for the life of the command line being parsed by the diagnostic monitor. The default of the internal append error messages flag is clear. The internal flag is not set until it is encountered in the command line by the diagnostic monitor.

Clear Error Messages - Command CEM

This command clears the internal error message buffer of the diagnostic monitor.

Test Group Configuration (cf) Parameters Editor - Command CF

The **cf** parameters control the operation of all tests in a test group. For example, the **RAM** test group has parameters such as starting address, ending address, parity enable, etc. At the time of initial execution of the diagnostic monitor, the default configuration parameters are copied from the firmware into the debugger work page. Here you can modify the configuration parameters using the **CF** command. When you invoke the **CF** command you are prompted with a brief parameter description and the current value of the parameter. You may enter a new value for that parameter, or a carriage return to proceed to the next configuration parameter.

You may also specify one or more test groups as argument(s) immediately following the **CF** command. If no arguments follow the **CF** command, the parameters for all test groups are listed for possible change.

Display Error Counters - Command DE

Each test or command in the diagnostic monitor has an individual error counter. As errors are encountered in a particular test, that error counter is incremented. If you were to run a self-test or just a series of tests, the results could be broken down as to which tests passed by examining the error counters.

To display all errors, enter **DE**. **DE** displays the results of a particular test if the name of that test follows **DE**. Only nonzero values are displayed.

Display Error Messages - Command DEM

This command displays (dumps) the internal error message buffer of the diagnostic monitor.

Display Pass Count - Command DP

A count of the number of passes in Loop-Continue (**LC**) mode is kept by the monitor. This count is displayed with other information at the conclusion of each pass. To display this information without using **LC**, enter **DP**.

Help - Command HE

On-line documentation has been provided in the form of a Help command (syntax: **HE** *[command name]*). This command displays a menu of the top level directory and test group names if no parameters are entered, or a menu of the subdirectory if the name of a subdirectory is entered. (The top level directory lists `(DIR)` after the name of each command that has a subdirectory).

For example, to bring up a menu of all the memory tests, enter **HE RAM**. When a menu is too long to fit on the screen, it pauses until the operator presses the carriage return, <**CR**>, again. To review a description of an individual test, enter the full name, i.e., **HE RAM CODE** displays information on the RAM Code Execution/Copy test routine.

The Help screen is shown in the following figure.

Help Extended - Command HEX

The **HEX** command goes into an interactive, continuous mode of the **HE** command. The syntax is **HEX<CR>**. The prompt displayed for **HEX** is the?. You may then type the name of a directory, or command. Type QUIT to exit.

```
197-Diag>he
AE       All Errors Mode
AEM      Append Error Messages Mode
BSW      BusSwitch Tests (DIR)
CEM      Clear Error Messages
CF       Configuration Editor
DCAM     DRAM Controller and Address Multiplexer (DCAM) Tests (DIR)
DE       Display Errors
DEM      Display Error Messages
DP       Display Pass Count
ECDM     Error Correction and Data Multiplexer (ECDM) Tests (DIR)
HE       Help on Tests/Commands
HEX      Help Extended
LA       Loop Always Mode
LANC     LAN Coprocessor (Intel 82596) Tests (DIR)
LC       Loop Continuous Mode
LE       Loop on Error Mode
LF       Line Feed Mode
LN       Loop Non-Verbose Mode
MASK     Self Test Mask
NCR      NCR 53C710 SCSI I/O Processor Test (DIR)
NV       Non-Verbose Mode
PCC2     PCCchip2 Tests (DIR)
RAM      Random Access Memory Tests (DIR)
RAMCD    Generic Memory Tests with Data Cache Only (DIR)
RAMCDI0  Generic Memory Tests with Instruction/Data Cache, MPU0 (DIR)
RAMCDI1  Generic Memory Tests with Instruction/Data Cache, MPU1 (DIR)
RAMCDI2  Generic Memory Tests with Instruction/Data Cache, MPU2 (DIR)
RAMCDI3  Generic Memory Tests with Instruction/Data Cache, MPU3 (DIR)
RTC      Self Test Mask
SE       Stop on Error Mode
ST       Self Test (DIR)
ST2401   CD2401 Serial Self-Tests (DIR)
VME2     VME2Chip2 Tests (DIR)
XCA0     External Cache (0) Tests
XCA1     External Cache (1) Tests
XCA2     External Cache (2) Tests
XCA3     External Cache (3) Tests
ZE       Zero Errors
ZP       Zero Pass Count
197-Diag>
```

Figure 2-1. Help Screen

Loop Always Mode - Prefix LA

To endlessly repeat a test or series of tests, enter the prefix **LA**. The **LA** command modifies the way that a failed test is endlessly repeated. The **LA** command has no effect until a test failure occurs, at which time, if the **LA** command has been previously encountered in the user command line, the failed test is endlessly repeated.

To break the loop, press the BREAK key on the diagnostic video display terminal. Certain tests disable the BREAK key interrupt, so pressing the ABORT or RESET switches on the MVME197 front panel may become necessary.

Loop-Continue Mode - Prefix LC

To endlessly repeat a test or series of tests, enter the prefix **LC**. This loop includes everything on the command line. To break the loop, press the BREAK key on the diagnostic video display terminal. Certain tests disable the BREAK key interrupt, so pressing the ABORT or RESET switches on the MVME197 front panel may become necessary.

Loop-On-Error Mode - Prefix LE

Occasionally, when an oscilloscope or logic analyzer is in use, it becomes desirable to endlessly repeat a test (loop) while an error is detected. The **LE** command modifies the way that a failed test is endlessly repeated. The **LE** command has no effect until a test failure occurs, at which time, if the **LE** command has been previously encountered in the user command line, the failed test is re-executed as long as the previous execution returned failure status.

To break the loop, press the BREAK key on the diagnostic video display terminal. Certain tests disable the BREAK key interrupt, so pressing the ABORT or RESET switches on the MVME197 front panel may become necessary.

Line Feed Suppression Mode - Prefix LF

The **LF** command sets the internal line feed mode flag of the diagnostic monitor. The default state of the internal line feed mode flag is clear which causes the executing test title/status line(s) to be terminated with a line feed character (scrolled). The duration of the **LF** command is the life of the user command line in which it appears. The line feed mode flag is normally used by the diagnostic monitor when executing a system mode self test. Although rarely invoked as a user command, the **LF** command is available to the diagnostic user.

Loop Non-Verbose Mode - Prefix LN

The **LN** command modifies the way that a failed test is endlessly repeated. The **LN** command has no effect until a test failure occurs, at which time, if the **LN** command has been previously encountered in the user command line, further printing of the test title and pass/fail status is suppressed. This is useful for more rapid execution of the failing test; i.e., the **LN** command contributes to a "tighter" loop.

Display/Revise Self Test Mask - Command MASK

The syntax is: **MASK** [*TEST NAME*]

where *TEST NAME* is the name of a diagnostic test.

MASK is used with a parameter to enable/disable a test from running under Self Test. If the user invokes **MASK** with no parameters, the currently disabled tests are displayed. When the **MASK** command is used on an MVME197 system, the mask values are preserved in non-volatile memory. This allows you to power down the system without disturbing the Self Test mask.

If you invoke the **MASK** command with a parameter, the parameter must be a specific test name (e.g., **MASK RAM ADR**).

The **MASK** command is a "toggle" command - if the specified test name mask was SET, it will be RESET; if it was RESET, it will be SET. After the toggle, the new Self Test mask is displayed.

If you invoke the **MASK** command with an invalid test name or a test directory (as opposed to a specific test name), 197Bug outputs an appropriate error message output.

If you invoke the **MASK** command with NO parameters, the current Self Test mask is displayed.

Non-Verbose Mode - Prefix NV

Upon detecting an error, the tests display a substantial amount of data. To avoid having to watch the scrolling display, 197Bug includes a mode that suppresses all messages except `PASSED` or `FAILED`. This mode is called non-verbose and is invoked prior to calling a command by entering **NV**. **NV ST** would cause the monitor to run the self-test, but show only the names of the subtests and the results (pass/fail).

Switch Directories - Command SD

To leave the diagnostic directory (and disable the diagnostic tests), enter **SD**. At this point, only the commands for 197Bug function. When in the 197Bug directory, the prompt reads “197-Bug>”. To return to the diagnostic directory, the command **SD** is entered again. When in the diagnostic directory, the prompt reads “197-Diag>”. The purpose of this feature is to allow you to access 197Bug without the diagnostics being visible.

Stop-On-Error Mode - Prefix SE

It is sometimes desirable to stop a test or series of tests at the point where an error is detected. **SE** accomplishes that for most of the tests. To invoke **SE**, enter it before the test or series of tests that is to run in Stop-On-Error mode.

Self Test - Command ST

The monitor provides an automated test mechanism called self test. This mechanism runs all the tests included in an internal self-test directory. The command **HE ST** lists the top level of the self test directory in alphabetical order.

Each test is described later in this manual.

When in system mode, the suite of tests that are run at system mode start up can be executed from Bug. This is done with the **ST** command. This command is useful for debugging board failures that may require toggling between the test suite and Bug. Upon completion of running the test suite, the Bug prompt is displayed, ready for other commands. For details on extended confidence test operation, refer to the *MVME197BUG 197Bug Debugging Package User’s Manual*.

Clear (Zero) Error Counters - Command ZE

The error counters originally come up with the value of zero, but it is occasionally desirable to reset them to zero at a later time. This command resets all of the error counters to zero. The error counters can be individually reset by entering the specific test name following the command. Example: **ZE VME2 TMRA** clears the error counter associated with **VME2 TMRA**.

Zero Pass Count - Command ZP

Invoking the **ZP** command resets the pass counter to zero. This is frequently desirable before typing in a command that invokes the Loop-Continue mode. Entering this command on the same line as **LC** results in the pass counter being reset every pass.

TEST DESCRIPTIONS 3

Detailed descriptions of 197Bug's diagnostic tests are presented in this chapter. The test groups are described in the order shown in the following table.

Table 3-1. Diagnostic Test Groups

Test Group	Description
RAM	Local RAM Tests (Instruction Cache only)
RAMCDI0	Local RAM Tests with Data & Instruction Cache (MPU0)
RAMCDI1	Local RAM Tests with Data & Instruction Cache (MPU1)
RAMCDI2	Local RAM Tests with Data & Instruction Cache (MPU2)
RAMCDI3	Local RAM Tests with Data & Instruction Cache (MPU3)
RAMCD	Local RAM Tests with Data Cache only
ECDM	Error Correcting Data Multiplexor ASIC Tests
DCAM	DRAM Controller and Address Multiplexor Utilities
BSW	BusSwitch ASIC Tests
XCA0	External Cache (MC88410/MC62110) Tests (MPU0)
XCA1	External Cache (MC88410/MC62110) Tests (MPU1)
XCA2	External Cache (MC88410/MC62110) Tests (MPU2)
XCA3	External Cache (MC88410/MC62110) Tests (MPU3)
RTC	MK48T0x Real Time Clock Tests
PCC2	Peripheral Channel Controller 2 Tests
ST2401	CD2401 Serial Port Tests
VME2	VME Interface ASIC VMEchip2 Tests

Table 3-1. Diagnostic Test Groups (Continued)

Test Group	Description
LANC	LAN Coprocessor (Intel 82596) Tests
NCR	NCR 53C710 SCSI I/O Processor Tests

Note **You may enter command names in either uppercase or lowercase.**

Local RAM (RAM) Tests

These sections describe the individual **RAM** tests.

Entering **RAM** without parameters causes all **RAM** tests, except those noted otherwise, to execute in the order shown in the table below.

To run an individual test, add that test name to the **RAM** command.

The test address range of these tests is determined by the **CF** parameters for the test group described in the following section. The individual tests are described in alphabetical order on the following pages.

Table 3-2. RAM Test Group

Mnemonic	Description
QUIK	Quick Write/Read
ALTS	Alternating Ones/Zeros *
PATS	Data Patterns *
ADR	Memory Addressing *
CODE	Code Execution/Copy
PERM	Permutations
RNDM	Random Data
BTOG	Bit Toggle
MARCH	March Address *
REF	Memory Refresh

* Indicated tests support the AE (All Errors) test mode during test operations. Refer to the **AE** command for details.

Prior to the execution of any **RAM** test, the error status registers of all ECDM devices are checked for previously detected errors. Normally, errors detected during this check are considered error cases and further testing is aborted. However, if the AE mode flag is set when the testing begins, this condition is reported if the test fails, but ignored otherwise.

Additionally, when testing is complete, these error status registers are checked again for errors detected by the hardware during testing.

Refer to the ***RAM*** *Common Test Error Messages* section of this chapter for details of error messages displayed when these errors are detected.

RAM Configuration Parameters **CF RAM**

Command Input

```
197-Diag>CF RAM
RAM Configuration Data:
Starting/Ending Address Enable [Y/N] =N ?
Starting Address =00010000 ?
Ending Address =01FFFFFC ?
Execution MPU =00000000 ?
Maximum Error Count (AE mode) =0000000A ?
Random Data Seed =12301983 ?
March Address Pattern =00000000 ?
Instruction (Code) Cache Enable [Y/N] =Y ?
Double Instruction Issue Disable [Y/N] =N ?
Branch Prediction Enable [N/Y] =N ?
Target Instruction Cache Enable [Y/N] =Y ?
Data Cache Enable [N/Y] =N ?
Data Cache Writethrough Mode Enable [N/Y] =N ?
Decoupled Cache Accesses Enable [Y/N] =N ?
Data Matching Disable [Y/N] =Y ?
Snoop Enable [N/Y] =N ?
Error Detection Enable [N/Y] =N ?
Error Correction Disable [N/Y] =N ?
197-Diag>
```

Description

User configurable test parameters are available for the **RAM** test group. Refer to Chapter 2 for information on using the **CF** command to set configuration parameters.

The **RAM** test parameters are listed and described below.

```
Starting/Ending Address Enable [Y/N] =N
```

This parameter must be set to Y for the next two parameters to be in effect. Otherwise, the default address range (all of local RAM not reserved by the debugger) will be used.

```
Starting Address =00010000 ?
```

This is the beginning address of the test range.

```
Ending Address =01FFFFFC ?
```

This is the ending address of the test range.

```
Execution MPU =00000000 ?
```

This value indicates the number of the processor which will execute the test.

```
Maximum Error Count (AE mode) =0000000A ?
```

This parameter sets the error count limit when the AE option flag is active.

```
Random Data Seed =12301983 ?
```

This value is used by the RANDOM and BTOG tests for data pattern generation.

```
March Address Pattern =00000000 ?
```

This value is used by the MARCH test as its beginning data pattern.

```
Instruction (Code) Cache Enable [Y/N] =Y ?
```

This parameter indicates whether or not to enable the specified MC88110 instruction cache for testing.

```
Double Instruction Issue Disable [Y/N] =N ?
Branch Prediction Enable [N/Y] =N ?
Target Instruction Cache Enable [Y/N] =Y ?
```

These parameters correspond to the MC88110 MPU Instruction Cache Control Register (ICTL) bits of the same name(s). Refer to the *MC88110 User's Manual* for more information.

```
Data Cache Enable [N/Y] =N ?
```

This parameter indicates whether or not to enable the specified MC88110 data cache for testing. The mode is decided by the next parameter.

```
Data Cache Writethrough Mode Enable [N/Y] =N ?
```

If this parameter is Y and the data cache is enabled, the caching mode for data is writethrough. Otherwise, the mode is copyback.

CF RAM

```
Decoupled Cache Accesses Enable [Y/N] =N ?
Data Matching Disable [Y/N] =Y ?
Snoop Enable [N/Y] =N ?
```

These parameters correspond to the MC88110 MPU Data Cache Control Register (DCTL) bits of the same name(s). Refer to the *MC88110 User's Manual* for more information.

Note

Snooping must be enabled for all testing with the data cache enabled on all products which utilize MC88410 external cache devices.

```
Error Detection Enable [N/Y] =N ?
```

This parameter enables an external error output to the BusSwitch to allow it to capture the address of the first error detected during testing.

```
Error Correction Disable [N/Y] =N ?
```

This parameter allows disabling single bit error (SBE) correction on DRAM read operations during testing.

Memory Addressing - ADR

RAM ADR

Command Input

```
197-Diag>RAM ADR
```

Description

This is the memory addressability test, the purpose of which is to verify addressing of memory in the range specified by the configuration parameters for the **RAM** test group. Addressing errors are sought by using a memory locations address as the data for that location. This test is coded to use only 32-bit data entities.

The test proceeds as follows:

1. A Location's Address is written to its location (n).
2. The next location (n+4) is written with its address complemented.
3. The next location (n+8) is written with the most significant (MS) 16 bits and least significant (LS) 16 bits of its address swapped with each other.
4. Steps 1, 2, and 3 above are repeated throughout the specified memory range.
5. The memory is read and verified for the correct data pattern(s) and any errors are reported.
6. The test is repeated using the same algorithm as above (steps 1 through 5) except that inverted data is used to ensure that every data bit is written and verified at both "0" and "1".

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      ADR: Addressability........................ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM      ADR: Addressability........................ Running ---> PASSED
```

RAM ADR

If the test fails, then the display appears as follows:

```
RAM      ADR: Addressability......................... Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =_______, Actual =_______
```

Alternating Ones/Zeros - ALTS

RAM ALTS

Command Input

```
197-Diag>RAM ALTS
```

Description

This test verifies addressing of memory in the range specified by the configuration parameters for the **RAM** test group. Data path errors are sought by using alternating data patterns for each successive test location. This test is coded to use only 32-bit data entities.

The test proceeds as follows:

1. Location (n) is written with data of all bits 0.
2. The next location (n+4) is written with all bits 1.
3. Steps 1 and 2 above are repeated throughout the specified memory range.
4. The memory is read and verified for the correct data pattern(s) and any errors are reported.

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      ALTS: Alternating Ones/Zeros................ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM      ALTS: Alternating Ones/Zeros................ Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM      ALTS: Alternating Ones/Zeros................ Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =_______, Actual =_______
```

Bit Toggle - BTOG

RAM BTOG

Command Input

```
197-Diag>RAM BTOG
```

Description

The **RAM** test directory configuration parameters determine the value of the global random data seed (32-bit) used by this test. This global random data seed is incremented as a result of executing this test.

This test uses the following test data pattern generation algorithm:

1. Random data seed is copied into a work register.
2. Work register data is shifted right one bit position.
3. Random data seed is added to work register using unsigned arithmetic.
4. Data in the work register may or may not be complemented.
5. Data in the work register is written to current memory location.

This test is coded to operate using the 64-bit data size only. Each 64-bit memory location in the specified memory range is written with the random test data pattern (upper 32 bits) and its complement (lower 32 bits). Each memory location in the specified memory range is then written with its initial test data pattern complemented. The memory under test is read back to verify that the complement test data is properly retained. Each memory location in the specified range is then written with its initial pattern. The memory under test is read back to verify that the test data was properly written.

Testing proceeds from the ending address to the starting address for each write/read pass of the test.

Note

As a result of executing this test, the random seed in the RAM test group configuration parameters is incremented by 1, so that subsequent runs of the test will produce a different set of patterns for the test unless the random data seed is set to the same value each time.

RAM BTOG

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      BTOG: Bit Toggle............................ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM      BTOG: Bit Toggle........................... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM      BTOG: Bit Toggle........................... Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =_______, Actual =_______
```

Code Execution/Copy - CODE

RAM CODE

Command Input

```
197-Diag>RAM CODE
```

Description

In this test, a routine is copied to the specified memory starting address location and then control is transferred to this routine. The routine in the memory under test copies itself to the next available memory address and executes the new copy. Each copy's starting address is "stacked" at the upper end of the test range prior to its execution.This process is repeated until test memory is exhausted, as specified by the configuration parameters. The code detects this and each copy is re-executed by retrieving the previous copy's address from the "stack" and jumping to it. This continues until the initial routine passes control back to the initiating test code in the FLASH.

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      CODE: Code Execution/Copy.................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM      CODE: Code Execution/Copy.................. Running ---> PASSED
```

The test failure mode is typified by a variety of events and/or non-events. Data and instruction access fault exceptions, as well as misaligned access exceptions, are typical in failure cases. Also, the non-display of the "PASSED" message above (or any other event) after a significant amount of time indicates that the test has failed. A hardware reset is required for the monitor to recover from this type of failure even if control of the system is not lost. Normal test time at 50 MHz is approximately 7-8 seconds for every 32M bytes of DRAM tested.

Note

This is normally the first test in the Self Test (ST) sequence which attempts to perform burst read operations from the DRAM. RAMCDIx (and RAMCD) tests do both write and read burst operations to/from the DRAM.

March Address - MARCH

RAM MARCH

Command Input

```
197-Diag>RAM MARCH
```

Description

This test verifies address independence of memory in the range specified by the configuration parameters for the **RAM** test group. Addressing errors are sought by writing data to each location immediately after it is verified so that if multiple addresses are decoded for a given test location, the data will have changed when any redundant decode location is verified. This test is coded to use only 32-bit data entities.

The test proceeds as follows:

1. All locations are initially written with the data pattern indicated in the March Address Pattern configuration parameter for this test group.
2. Each location in the test range from lowest to highest is then verified to be the March Address Pattern and is immediately written to the complement of that pattern.
3. Each location in the test range from highest to lowest is then verified to be the complement of the March Address Pattern and is immediately written to the original pattern.This final pattern is not verified but is intended to disturb any redundantly decoded locations.

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      MARCH: March Address........................ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM      MARCH: March Address........................ Running ---> PASSED
```

RAM MARCH

If the test fails, then the display appears as follows:

```
RAM      MARCH: March Address........................ Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =_______, Actual =_______
```

Data Patterns - PATS

RAM PATS

Command Input

```
197-Diag>RAM PATS
```

Description

If the test address range (test range) is less than 8 bytes, the test immediately returns pass status. The effective test range end address is reduced to the next lower 8-byte boundary if necessary. Memory in the test range is filled with all ones ($FFFFFFFF).

For each location in the test range, the following patterns are used:

- ❏ $00000000
- ❏ $01010101
- ❏ $03030303
- ❏ $07070707
- ❏ $0F0F0F0F
- ❏ $1F1F1F1F
- ❏ $3F2F2F2F
- ❏ $7F7F7F7F

Each location in the test range is, individually, written with the current pattern and the 1's complement of the current pattern. Each write is read back and verified. This test is coded to use only 32-bit data entities.

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      PATS: Patterns.............................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM      PATS: Patterns.............................. Running ---> PASSED
```

RAM PATS

If the test fails, then the display appears as follows:

```
RAM      PATS: Patterns.............................. Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =_______, Actual =_______
```

Note **During RAMCDIx testing, this test only verifies the MC88110 data cache in normal operation.**

Byte/Half/Word Permutations - PERM

RAM PERM

Command Input

```
197-Diag>RAM PERM
```

Description

This command performs a test which verifies that the memory in the test range can accommodate byte, half, and word writes and reads in any combination. The test range is the memory range specified by the **RAM** test group configuration parameters for starting and ending address. If the test address range (test range) is less than 16 bytes, the test immediately returns pass status. The effective test range end address is reduced to the next lower 16-byte boundary if necessary.

This test performs three data size test phases in the following order:

- ❑ 8 bits
- ❑ 16 bits
- ❑ 32 bits

Each test phase writes a 16-byte data pattern (using its data size) to the first 16 bytes of every 256-byte block of memory in the test range. The 256-byte blocks of memory are aligned to the starting address configuration parameter for the **RAM** test group.

The test phase then reads and verifies the 16-byte blocks using the following access modes:

- ❑ 8-bit
- ❑ 16-bit
- ❑ 32-bit

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      PERM: Permutations.......................... Running --->
```

RAM PERM

If all parts of the test are completed correctly, then the test passes.

```
RAM      PERM: Permutations.......................... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM      PERM: Permutations.......................... Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =_______, Actual =_______
```

Quick Write/Read - QUIK

RAM QUIK

Command Input

```
197-Diag>RAM QUIK
```

Description

Each pass of this test fills the test range with a data pattern by writing the current data pattern to each memory location from a local variable and reading it back into that same register. The local variable is verified to be unchanged only after the write pass through the test range. This test uses a first pass data pattern of 0, and $FFFFFFFF for the second pass. This test is coded to use only 32-bit data entities.

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      QUIK: Quick Write/Read...................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM      QUIK: Quick Write/Read...................... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM      QUIK: Quick Write/Read...................... Running ---> FAILED

Data Miscompare Error: Expected =________, Actual =________
```

Memory Refresh Testing - REF

RAM REF

Command Input

```
197-Diag>RAM REF
```

Description

The memory range and address increment is specified by the **RAM** test directory configuration parameters. (Refer to the **CF** command).

First, the real time clock may be checked to see if it is running. If not running, the RTC is started for use by the test. (Recent versions of this test do not use the RTC for the refresh delays.) Second, each memory location to be tested has the data portion verified by writing/verifying all zeros, and all ones. Next, a data pattern is written to the test location. After all the data patterns are filled for all test locations, a refresh wait cycle is executed using the RTC or an on-board timer device. After the wait cycle, the data is read, and if the previously entered data pattern does not match the data pattern read in, a failure is indicated. If the data patterns match, then the test is passed. After the test completes, the RTC is returned to the mode it was in when the test started (if used by the test).

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      REF:  Memory Refresh....................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM      REF:  Memory Refresh....................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
RAM      REF:  Memory Refresh....................... Running ---> FAILED

(error message)
```

RAM REF

If any failures occur, the following error message(s) are displayed (more descriptive text follows):

If a data verification error occurs before the refresh wait cycle:

```
RAM/REF Test Failure Data:

Immediate Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

If a data verification error occurs following the refresh wait cycle:

```
RAM/REF Test Failure Data:

Unrefreshed Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

Random Data - RNDM

RAM RNDM

Command Input

```
197-Diag>RAM RNDM
```

Description

This test is coded to operate using the 64-bit data size only. Each 64-bit data operand is generated with the 32-bit random test pattern (initially from the **RAM** test group configuration parameter random seed) as the upper 32 bits and its complement as the lower 32 bits.

The test proceeds as follows:

1. A random pattern is written throughout the test range using the pattern generation algorithm described in the **RAM BTOG** test section.
2. The random pattern is verified.

Note

As a result of executing this test, the random seed in the RAM test group configuration parameters is incremented by 1, so that each run of the test will produce a different set of patterns for the test, unless the random data seed is set to the same value each time.

Response/Messages

After the command has been issued, the following line is printed:

```
RAM      RNDM: Random Data.......................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM      RNDM: Random Data.......................... Running ---> PASSED
```

RAM RNDM

If the test fails, then the display appears as follows:

```
RAM      RNDM: Random Data........................... Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

RAM Common Test Error Messages

The tests of the **RAM** (and **RAMCDIx**/**RAMCD**) test group(s) may encounter errors detected by the hardware, independent of the test which is being executed. Checks are made before each test, as well as after, to determine if this type of error has been detected. If an error is detected prior to actual test execution, testing is not performed (unless the AE mode flag is set) and the error is reported to the user as a pre-test error. Errors detected after testing has completed will be reported as post-test errors in addition to any errors detected by software during testing.These error messages are described below.

The following error message indicates that the ECC circuitry in one of the ECDM devices of DRAM sub-system 0 had reported an error prior to the beginning of the test. The error was reported by ECDM #3 as a single-bit error at data channel segment RD bit 6. This information is derived from the Syndrome code obtained from the SYNSTAT register. The ERLOG signal from the ECDM was asserted to the BusSwitch ASIC which registered the address which was being accessed when the error was detected by the ECDM. This address is only valid if the RAM Error Detection Disable CF parameter is set to 'N' and the BusSwitch PALINT register value is 30 (PALINT bit set).

Refer to the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for more information on the DRAM sub-system architecture and ECC error reporting. Refer to Figure A-1 in Appendix A for address to data channel mapping and to the appropriate schematics for determination of the failing data channel and device(s).

```
RAM   ADR: Memory Addressing....................... Running ---> FAILED

RAMCDI0/ADR Test Failure Data:
Pre-Test ECC Error Detected in Sub-System #0 - ECDM CSR $FFF01100 !!
##### Error Report for ECDM CSR at $FFF01100 #####
Error Detected in ECDM #03:
 MEMCON = 00 - ERSTAT = 42 - SYNSTAT = 98
 Syndrome Code = 98 - Error in ECDM Segment RD, Data Bit 6
BusSwitch PALINT Register = 30
Address Latched in BusSwitch PAL = $00008525
```

This display shows the error messages for a post test error. Note that the test may, or may not, detect DRAM errors depending upon whether error correction is enabled.

```
RAM   ADR: Memory Addressing....................... Running ---> FAILED

RAM/ADR Test Failure Data:
< pre-test errors would be displayed here, if detected >
Data Miscompare Error:Address =03FFF284,Expected =FFFFFFFF,Actual =FFFFFFBF

Post-Test ECC Error Detected in Sub-System #0 - ECDM CSR $FFF01100 !!
##### Error Report for ECDM CSR at $FFF01100 #####
Error Detected in ECDM #03:
 MEMCON = 00 - ERSTAT = 42 - SYNSTAT = 98
 Syndrome Code = 98 - Error in ECDM Segment RD, Data Bit 6
BusSwitch PALINT Register = 30
Address Latched in BusSwitch PAL = $03FFF284
```

This display indicates that an ECC check bit data error was detected by the hardware during testing. Data channel miscompare errors are normally not detected in this situation.

```
RAM   ADR: Memory Addressing....................... Running ---> FAILED

RAM/ADR Test Failure Data:
< pre-test errors would be displayed here, if detected >

Post-Test ECC Error Detected in Sub-System #0 - ECDM CSR $FFF01100 !!
##### Error Report for ECDM CSR at $FFF01100 #####
Error Detected in ECDM #x:
 MEMCON = 00 - ERSTAT = 42 - SYNSTAT = 80
 Syndrome Code = 80 - Error in ECDM Check Bit 7
```

This display shows the error messages when multi-bit errors are detected by the software and hardware during testing.

```
RAM   ADR: Memory Addressing....................... Running ---> FAILED

RAM/ADR Test Failure Data:
< pre-test errors would be displayed here, if detected >
Data Miscompare Error:Address =xxxxxxxx,Expected =xxxxxxxx,Actual =xxxxxxxx

Post-Test ECC Error Detected in Sub-System #0 - ECDM CSR $FFF01100 !!
##### Error Report for ECDM CSR at $FFF01100 #####
Error Detected in ECDM #x:
 MEMCON = 00 - ERSTAT = 80 - SYNSTAT = xx
 Syndrome Code = xx - Not Found in SBE Tables - Multi-Bit Error
```

Local RAM (RAMCDIx) Tests with Caching (Data and Instruction)

This section describes the **RAMCDIx** test groups.

The code for these tests is the same as the tests described in the **RAM** test group except that the code is executed with the MC88110 Instruction and Data caches both enabled. This forces both read and write burst operations to the local DRAM and provides a dynamic exercise of both the MPU caches and the MC88410 external cache controller and MC62110 cache RAM devices (if present). The “x” in the test name indicates which MPU (MPU 0 through MPU 3) in the current board configuration will be executing the tests.

Entering **RAMCDIx** (x = 0 through 3) without parameters causes all **RAMCDIx** tests to execute in the order shown in the table below. If the requested MPU does not exist in the system configuration, the tests are bypassed.

To run an individual test, add that test name to the **RAMCDIx** command.

The test address range of these tests is determined by the **CF** parameters for the test groups and are described in the following section. The individual test descriptions are provided in the **RAM** test group section of this manual.

Table 3-3. RAMCDIx Test Groups

Mnemonic	Description
QUIK	Quick Write/Read
ALTS	Alternating Ones/Zeros
PATS	Data Patterns
ADR	Memory Addressing
CODE	Code Execution/Copy
PERM	Permutations
RNDM	Random Data
BTOG	Bit Toggle
MARCH	March Address

Table 3-3. RAMCDIx Test Groups (Continued)

Mnemonic	Description
REF	Memory Refresh

Prior to the execution of any **RAMCDIx** test, the error status registers of all ECDM devices are checked for previously detected errors. Normally, if an error is detected, it is considered an error case and further testing is aborted. However, if the AE mode flag is set when the testing begins, this condition is reported if the test fails, but ignored otherwise. Additionally, when testing is complete, these error status registers are checked again for errors detected by the hardware during testing.

Refer to the ***RAM*** *Common Test Error Messages* section for descriptions of error messages associated with these pre- and post-test errors.

RAMCDIx Configuration Parameters **CF RAMCDIx**

Command Input

```
197-Diag>CF RAMCDI0
RAM Configuration Data:
Starting/Ending Address Enable [Y/N] =N ?
Starting Address =00010000 ?
Ending Address =01FFFFFC ?
Execution MPU =00000000 ?
Maximum Error Count (AE mode) =0000000A ?
Random Data Seed =12301983 ?
March Address Pattern =00000000 ?
Instruction (Code) Cache Enable [Y/N] =Y ?
Double Instruction Issue Disable [Y/N] =N ?
Branch Prediction Enable [N/Y] =N ?
Target Instruction Cache Enable [Y/N] =Y ?
Data Cache Enable [N/Y] =Y ?
Data Cache Writethrough Mode Enable [N/Y] =N ?
Decoupled Cache Accesses Enable [Y/N] =Y ?
Data Matching Disable [Y/N] =Y ?
Snoop Enable [N/Y] =Y ?
Error Detection Enable [N/Y] =N ?
Error Correction Disable [N/Y] =Y ?
197-Diag>
```

Description

User configurable test parameters are available for the **RAMCDIx** test groups. Refer to Chapter 2 for information on using the **CF** command to set configuration parameters.

The **RAMCDIx** test default **CF** parameters are listed in the command input block above. The execution MPU will vary according to which test command is entered. Of special interest are the MPU cache control parameters and the DRAM Error Detection/Correction parameters.

Refer to the ***RAM*** *Configuration Parameters* section for descriptions of each parameter.

Local RAM (RAMCD) Tests with Caching (Data only)

This section describes the **RAMCD** test group.

Only a single test is part of this group. It is the same as the test described in the **RAM** test group except that it is executed with the MC88110 Instruction cache disabled and Data cache enabled. This forces both read and write burst operations to the local DRAM for data operations only. This testing is performed to verify correct handling by the DCAM of mixed burst and non-burst cycles to the DRAM. The test is normally performed by MPU 0.

Entering **RAMCD** without parameters is the same as entering **RAMCD CODE** in this case.

The test address range is determined by the **CF** parameters for the test group and is described in the following section. The **CODE** test is described in the **RAM** test group section of this manual.

Table 3-4. RAMCD Test Group

Mnemonic	Description
CODE	Code Execution/Copy

Prior to the execution of the **RAMCD** test, the error status registers of all ECDM devices are checked for previously detected errors. Normally, if an error occurs, it is considered an error case and further testing is aborted. However, if the AE mode flag is set when the testing begins, this condition is reported if the test fails, but ignored otherwise. Additionally, when testing is complete, these error status registers are checked again for errors detected by the hardware during testing.

Refer to the ***RAM*** *Common Test Error Messages* section for description of error messages associated with these pre- and post-test errors.

RAMCD Configuration Parameters CF RAMCD

Command Input

```
197-Diag>CF RAMCD
RAM Configuration Data:
Starting/Ending Address Enable [Y/N] =N ?
Starting Address =00010000 ?
Ending Address =01FFFFFC ?
Execution MPU =00000000 ?
Maximum Error Count (AE mode) =0000000A ?
Random Data Seed =12301983 ?
March Address Pattern =00000000 ?
Instruction (Code) Cache Enable [Y/N] =N ?
Double Instruction Issue Disable [Y/N] =N ?
Branch Prediction Enable [N/Y] =N ?
Target Instruction Cache Enable [Y/N] =Y ?
Data Cache Enable [N/Y] =Y ?
Data Cache Writethrough Mode Enable [N/Y] =N ?
Decoupled Cache Accesses Enable [Y/N] =Y ?
Data Matching Disable [Y/N] =Y ?
Snoop Enable [N/Y] =Y ?
Error Detection Enable [N/Y] =N ?
Error Correction Disable [N/Y] =Y ?
197-Diag>
```

Description

User configurable test parameters are available for the **RAMCD** test group. Refer to Chapter 2 for information on using the **CF** command to set configuration parameters.

The **RAMCD** test default **CF** parameters are listed in the command input block above. The execution MPU will default to MPU 0.

Refer to the ***RAM*** *Configuration Parameters* section for descriptions of each parameter.

Error Correcting Data Multiplexer (ECDM) ASIC Tests

These tests check the ECDM devices on the MVME197 family of boards. The ECDM ASIC devices s are part of the hardware of the DRAM memory sub-systems of the MVME197 family. Multiple memory sub-systems may be present, and each memory sub-system will have four ECDM devices associated with it. In multi-processor configurations, any “slave” processors must be forced into an idle loop and prevented from accessing local DRAM since these tests may temporarily put the DRAM into states which prevent normal operation and may cause erroneous operation of the “slave” processors otherwise.

Normally, the **ECDM** tests will check all ECDM devices of all existing memory sub-systems. However, a configuration parameter (described later) is provided which allows the user to limit testing to a particular memory sub-system, rather than test the ECDM devices of all of the memory sub-systems present. Refer to the MAP command description for more information on multiple sub-system configurations.

Entering **ECDM** without parameters causes all **ECDM** tests to execute in the order shown in the table below with the exception of the MAP command which is a utility used to display the mapping of the DRAM sub-systems present in the system.

To run an individual test, add that test name to the **ECDM** command.

The individual tests are described in alphabetical order on the following pages.

Table 3-5. ECDM Test Group

Mnemonic	Description
MAP	DRAM Sub-System Mapping
REGS	Register Checks
CHKGEN	Checkbit Generation
CHKRAM	Checkbit DRAM Test
SBEC	SBE Control Options
SBEP	SBE Permutations
DBEC	DBE Control Options
DBEP	DBE Permutations
INITCK	INIT Function Check
I2C	I2C Bus Interface Check

The DRAM sub-system of the MVME197 family products operates on MC88110 cache line sized data operands at the DRAM interface. These “lines” are 32 bytes or 256 bits wide at the DRAM array. Four ECDM devices operate in parallel with each ECDM handling 64 bits of the total operand width at the DRAM interface. At the processor interface, each ECDM provides 16 bits of data to the 64-bit MC88110 data bus. The 256 bits of data from the DRAM interface is latched in the ECDM devices on reads and is passed to the processor in four segments with each ECDM providing 16 bits per 64-bit transfer. The address mapping of this multiplexing function is illustrated in Figure A-1 in Appendix A. This figure also contains the “line” data bit numbering scheme (255-0) used for these tests

The 64 bits handled by each ECDM comprise the ECC data word for that ECDM which is independent of the other ECDM devices in the sub-system. Each ECDM also handles 8 bits of check bit information associated with its 64-bit ECC data word. This 8 bits is referred to as the check byte for the ECDM’s data word. Figure A-1 in Appendix A indicates the ECC data word bit numbering (63-0) for each ECDM which is used to correlate syndrome code error information to “line” bit numbering for error reporting.

Check byte data is only accessible in a special read/write check byte mode which is mutually exclusive with normal DRAM data access. The black boxes in Figure A-1 in Appendix A indicate the locations where the check byte for each ECDM ECC sub-system may be read in this mode.

The term “line” in the descriptions of these tests will refer to a 32-byte section of memory which is aligned to a 32-byte boundary. The term “even line” will refer to a “line” which is 64-byte aligned and the term “odd line” will refer to a “line” which immediately follows an even line in the address space.

Each ECDM contains two line buffers which are used to return data to the processor on read operations. In sequential operations, the ECDMs will alternate between these two buffers to return data. One of the buffers is physically addressed at an even “line” address and the other at an odd “line” address. Each of these buffers have their own set of ECC circuitry for generating or processing check bit information. These diagnostic tests must verify the operation of both sets of buffers for all functions.

ECDM Configuration Parameters

CF ECDM

Command Input

```
197-Diag>CF ECDM
ECDM Configuration Data:
Restore Defaults (=0) =0000xxxx ?
Instruction Cache Enable [Y/N] =Y ?
Override Test Mode [Y/N] =N ?
Override ECDM CSR Address =FFF01100 ?
ECDM INIT DRAM Test Size =00008000 ?
197-Diag>
```

Description

User configurable test parameters are available for the **ECDM** test group. Refer to Chapter 2 for information on using the **CF** command to set configuration parameters.

The ECDM test parameters are listed and described below.

Note

All of the parameters are not initialized until the first ECDM command is executed. It is suggested that the MAP command be executed prior to changing any CF parameters using the CF ECDM command.

```
Restore Defaults (=0) =0000xxxx ?
```

This parameter when set to zero will force the restoring of the default values to all of the configuration parameters when the next ECDM command is executed (prior to execution). Once initialized, this parameter will reflect the address where all of the ECDM parameters are stored in local memory.

```
Instruction Cache Enable [Y/N] =Y ?
```

This parameter, when set to Yes, enables the instruction cache of the MC88110 during testing. When this parameter is set to No the instruction cache is disabled. This parameter only affects the processor which is running the tests.

CF ECDM

```
Override Test Mode [Y/N] =N ?
```

This parameter, when set to Yes, will test only the subsystem with the CSR which appears at the address given by the “Override ECDM CSR Address” (below). The default for testing is to test all subsystems found. If this parameter is set to No, then all subsystems will be tested.

```
Override ECDM CSR Address =FFF01100 ?
```

This parameter should be set to the CSR address of a memory subsystem to be tested when it is desired to test only a single memory subsystem and not all memory subsystems. This parameter is only valid when the “Override Test Mode” parameter described above is set to Yes.

```
ECDM INIT DRAM Test Size =00008000 ?
```

This parameter sets the size (in bytes) of the ECDM INIT Function Check test. Refer to the descriptive text for the INIT Function Check test for more information.

Checkbit Generation - CHKGEN

ECDM CHKGEN

Command Input

```
197-Diag>ECDM CHKGEN
```

Description

This purpose of this test is to verify that, for all DRAM sub-systems resident in the system, the ECDM devices associated with each sub-system generate the expected check byte(s) when a single bit in a line of data (32 bytes/256 bits) is set in a background of zeros. This is verified for all bits (255 to 0) in both odd and even test lines to ensure that both ECDM line buffers and their associated EDAC circuitry is tested. The line under test is cleared and the generated check bytes verified to be zero before bit testing begins.

The expected check byte patterns for this test should match the syndrome patterns used for single bit error notification as if the set bit in the test line was in error. Refer to the ECDM section of the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for more information on the syndrome codes generated by the ECDM devices. Refer to Figure A-1 in Appendix A for test time bit numbering.

Response/Messages

After the command has been issued, the following line is printed:

```
ECDM     CHKGEN: Checkbit Generation................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
ECDM     CHKGEN: Checkbit Generation................. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
ECDM     CHKGEN: Checkbit Generation................. Running ---> FAILED

ECDM/CHKGEN Test Failure Data:
(error message)
```

ECDM CHKGEN

The following error messages are possible:

This error is displayed when any check bytes generated for the zero background pattern are found to be non-zero:

```
Error Verifying Zero Data CheckBytes
Address = xxxxxxxx, Expected = 00, Actual = xx
```

This error message is displayed if the expected check byte pattern is not detected in the expected location in the checkbit DRAM:

```
- Unexpected CHECK BIT pattern - Bit xxx:
Address = xxxxxxxx, Expected = xx, Actual = xx
```

This error message is displayed if the generation of the expected check byte pattern in the ECDM (one of four) targeted by the test disturbed the check byte(s) of another ECDM which should remain zero:

```
- Disturbed CHECK BIT pattern - Bit xxx:
Address = xxxxxxxx, Expected = 00, Actual = xx
```

Checkbit DRAM Test - CHKRAM

ECDM CHKRAM

Command Input

```
197-Diag>ECDM CHKRAM
```

Description

This test verifies that the DRAM used to store the checkbit information generated by the ECDMs as part of the ECC implementation can be written to and read from reliably. This is accomplished using a special mode of the ECDM which allows writing directly to this DRAM but prevents normal DRAM operation while active. A variety of patterns are written and verified for all four checkbytes at each line location of the DRAM. Check byte data is found in the odd-byte locations of the first double word of every line (32 bytes) of memory. Each line tested is re-initialized (to zero) after testing on that line is completed.

In multi-processor configurations, all slave processors are forced to execute an “idle” loop which prevents them from accessing the local DRAM while this test is executing.

Response/Messages

After the command has been issued, the following line is printed:

```
ECDM     CHKRAM: Checkbit DRAM Test.................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
ECDM     CHKRAM: Checkbit DRAM Test.................. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
ECDM     CHKRAM: Checkbit DRAM Test.................. Running ---> FAILED

ECDM/CHKGEN Test Failure Data:
(error message)
```

ECDM CHKRAM

Here, (error message) is one of the following:

This is the standard message when a checkbit DRAM data error is detected during testing:

```
- Check RAM Error Detected:
Address = xxxxxxxx, Expected = xxxxxxxx, Actual = xxxxxxxx
```

This message is displayed when an error is detected during the process of restoring the original check byte data to the test location:

```
- Check RAM Data Restore Error - Addr = $xxxxxxxx
--- Location Cleared to Re-Init Check Data ---
```

DBE Control Options - DBEC

ECDM DBEC

Command Input

```
197-Diag>ECDM DBEC
```

Description

This test verifies that, for all DRAM sub-systems resident in the system, each ECDM device associated with each sub-system can detect and report correctly double-bit (non-correctable) errors. It verifies that a double bit error in a particular line does not effect the data read from that line and that a read of any sized operand in the line will force the error to be reported exclusively in the ECDM expected.

This test also verifies that various control options available through the ECDM MEMCON register related to double-bit errors function as specified without affecting normal operations. These include the LOGNCER bit which enables the ERLOG signal to be generated when a non-correctable error is detected and the EERP bit which enables the ERLOG signal to be output to the BusSwitch. The address of these errors is verified to be latched in the BusSwitch PAL (Processor Address Latch) register when the ERLOG signal is received by the BusSwitch and it is verified to be the address read when the error was detected (if this function is enabled). Additionally, it is verified that if a non-correctable error is latched in the status registers that encountering another error of this type will not overwrite the original contents of the registers relating to non-correctable errors. Refer to *the MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for more information on ECDM MEMCON control functions.

All ECDMs and the ECC circuitry associated with both line buffers within each ECDM are tested independently.

Response/Messages

After the command has been issued, the following line is printed:

```
ECDM     DBEC: DBE Control Options.................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
ECDM     DBEC: DBE Control Options.................. Running ---> PASSED
```

ECDM DBEC

If any part of the test fails, then the display appears as follows:

```
ECDM     DBEC: DBE Control Options................... Running ---> FAILED

ECDM/DBEC Test Failure Data:
(error message)
```

Here, (error message) may be one of the following:

This message is displayed if the ECDM revision read in the ERSTAT register is not supported by the test code:

```
WARNING: Unsupported ECDM Revision Detected - xx !!!
```

This message is displayed if an error is detected in the ECDM Error Status registers prior to beginning the test checks:

```
Pre-Existing Error Detected Prior to Check:
< generic ECC error status message(s) here >
```

This message is displayed when forced double bit error data was modified unexpectedly by the ECC hardware when reading the error location:

```
Error Data NOT Corrected - ECDM 0x:
Address = xxxxxxxx, Expected = xxxx, Actual = xxxx
```

This message is displayed when the ERSTAT register value was not the expected value (error or no error):

```
Unexpected ERSTAT for ECDM 0x - Exp.= xx - Rd.= xx -
```

This message is displayed when the BusSwitch PAL register does not contain the expected value for the test conditions:

```
Address Latch Incorrect - Exp = $xxxxxxxx - Rd = $xxxxxxxx
```

ECDM DBEC

This message is displayed when the test expected the ERLOG signal to be asserted by the ECDM in response to a non-correctable error being detected and sent to the BusSwitch where it should have caused the PALINT interrupt status bit to be set:

```
Interrupt Status NOT Detected - PALINT = $xx -
```

This message is displayed when the error data location has been modified unexpectedly when the test error location was read with the indicated state of a specified bit in the MEMCOM register:

```
Error Data Modified <MEMCON bit status> - ECDM x
Address = xxxxxxxx, Expected = xxxx, Actual = xxxx
```

This message is displayed when unexpected error status was detected during read operations with the indicated state of a specified bit in the MEMCOM register:

```
<Incorrect/No> Error Status Detected in ECDM x w/<MEMCON status> - ERSTAT = xx
```

This message is displayed when an unexpected ERLOG signal must have triggered the PALINT interrupt status bit without the EERP bit in the ECDM MEMCON register being set:

```
PAL Interrupt Status Detected wo/EERP - PALINT = xx
```

One of the following messages may be displayed if data patterns are disturbed as a result of detecting a non-correctable error with a pending non-correctable error currently latched in the ERSTAT register:

```
DBE2 Error Data Modified on Read - ECDM x:
Address = xxxxxxxx, Expected = xxxx, Actual = xxx
```

```
DBE1 Error Data Modified w/Prior ERROR Latched - ECDM x
```

ECDM DBEC

This message is displayed when the BusSwitch PAL register does not contain the expected value after a read operation with the indicated state of a specified bit in the MEMCOM register:

```
BusSwitch PAL Error on ERSTAT/ERLOG Latch Test:
Address Latch Incorrect - Exp = $xxxxxxxx - Rd = $xxxxxxxx
```

This message is displayed when the error data location(s) under test could not be restored to their original data patterns after testing was completed. This restore is done with the EERO bit in the ECDMs cleared to disable the output of the NCER* signal to the DCAM which when asserted will prevent writes to lines which indicate non-correctable errors.

```
Data Not Restored Correctly After DBE<1/2> Verification - Addr. $xxxxxxxx
```

DBE Permutations - DBEP

ECDM DBEP

Command Input

```
197-Diag>ECDM DBEP
```

Description

This test verifies that, for all DRAM sub-systems resident in the system, that the ECDM devices associated each sub-system are capable of detecting and reporting correctly double-bit errors in all possible bit combinations in the ECC word as well as in any check bit positions associated with them. It will also verify that any type access into a line containing a double-bit error will result in that error being reported accurately. This is done for the ECC circuitry for both line buffers for all ECDM devices. The test is performed once with a background pattern of zeros and then with a pattern of all ones. Refer to Figure A-1 in Appendix A for bit number mapping information for this test.

Response/Messages

After the command has been issued, the following line is printed:

```
ECDM     DBEP: DBE Permutations..................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
ECDM     DBEP: DBE Permutations..................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
ECDM     DBEP: DBE Permutations..................... Running ---> FAILED

(error message)
```

Here, (error message) is one of the following:

This message is displayed if the ECDM revision read in the ERSTAT register is not supported by the test code:

```
WARNING: Unsupported ECDM Revision Detected - xx !!!
```

ECDM DBEP

This message is displayed if an error is detected in the line test buffer when verifying one of the background patterns prior to DBE testing:

```
Error Verifying Background Pattern
Address =xxxxxxxx, Expected =xxxxxxxx, Actual =xxxxxxxx
```

This message is displayed if any error is found when verifying that the error forced in the test line was detected and reported as expected:

```
Test Addr = $xxxxxxxx - Background = xxxx - ECDM #x
DBE <Data/Check> Bit1 = xxx - DBE <Data/Check> Bit2 = xx -
```

This message is displayed if the test line data (background pattern) could not be restored after a test iteration:

```
Data Not Restored Correctly After DBE Verification - Addr. $xxxxxxxx
```

This message is displayed if an error is detected in the ECDM Error Status registers prior to beginning the test checks:

```
Pre-Existing Error Detected Prior to Check:
< generic ECC error status message(s) here >
```

This message is displayed when the ERSTAT register value was not the expected value (error or no error):

```
Unexpected ERSTAT for ECDM 0x - Exp.= xx - Rd.= xx -
```

This message is displayed if an error is detected in the Error Status register of an ECDM not associated with the forced error:

```
Error Detected in Wrong ECDM - x - ERSTAT = xx
Expected ERROR in ECDM x
```

ECDM DBEP

This message is displayed when the BusSwitch PAL register does not contain the expected value for the test conditions:

```
Address Latch Incorrect - Exp = $xxxxxxxx - Rd = $xxxxxxxx
```

This message is displayed when the BusSwitch PAL register does not contain the expected value for the test conditions:

```
Interrupt Status NOT Detected - PALINT = $xx -
```

This message is displayed if verification errors are detected to indicate the type of read operations which were being performed when the errors were detected:

```
- Error Reading <8/16/32/64>-bit Operand -
```

I^2CBus Interface Check - I2C   ECDM I2C

Command Input

```
197-Diag>ECDM I2C
```

Description

This test is designed to verify that, for the first ECDM device associated with each DRAM sub-system, the I^2C control/status registers can be manipulated to verify that the DCAM registers can be accessed as well as the EEROM resident on the bus can be accessed. It also verifies that the user (non-protected) portion of the I^2C EEPROM can be written to.

Response/Messages

After the command has been issued, the following line is printed:

```
ECDM     I2C: I2C Bus Interface Check................ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
ECDM     I2C: I2C Bus Interface Check................ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
ECDM     I2C: I2C Bus Interface Check................ Running ---> FAILED

(error message)
```

Here, (error message) is similar to one or more of the following:

This message is displayed if the ECDM revision read in the ERSTAT register is not supported by the test code:

```
WARNING: Unsupported ECDM Revision Detected - xx !!!
```

ECDM I2C

This message is displayed if an error is detected in the ECDM I^{2}C bus interface registers while attempting to perform an I^{2}C bus read operation from the DCAM registers:

```
I2C Error Reading DCAM Image: <I2C Driver Error Code here>
```

This message is displayed if an error is detected in the ECDM I^{2}C bus interface registers while attempting to perform some I^{2}C bus operation:

```
< Protocol-specific message here>
Error in ECDM CSR @ $fff00x00 - I2CON = xx - I2STAT = xx
```

This message is displayed if the DCAM ID value in the register image read via the ECDM I^{2}C bus interface from the DCAM is not correct:

```
DCAM ID Value Incorrect in DCAM Image:
Address =xxxxxxxx, Expected =81, Actual =xx
```

This message is displayed if the CSR registers in the DCAM register image do not match the CSR address of the sub-system under test:

```
DCAM CSR Value(s) Incorrect in DCAM Image:
Address =xxxxxxxx, Expected =fff00x00, Actual =xxxxxxxx
```

This message is displayed if an error is detected in the ECDM I^{2}C bus interface registers while attempting to perform an I^{2}C bus read operation from the I^{2}C EEROM:

```
I2C Error Reading EEROM Protected Data: <I2C Driver Error Code here>
```

This message is displayed if the test encounters an unexpected or no error code when attempting an access to a non-existent I^{2}C address:

```
<No/Unexpected> I2C Error Reading From Bad I2C Address: 55
```

ECDM I2C

This message is displayed if the I^2C bus can not be cleared after the access to a non-existent address:

```
Error Attempting to Release I2C Bus After Test Access
Error in ECDM CSR @ $fff00x00 - I2CON = xx - I2STAT = xx
```

INIT Function Check - INITCK **ECDM INITCK**

Command Input

```
197-Diag>ECDM INITCK
```

Description

This test verifies that, for all DRAM sub-systems resident in the system, that the ECDM devices associated with each DRAM sub-system can be configured (by setting the MEMCON INIT bit) to cause an entire line of DRAM data to be initialized to zeros when a non-burst write of any size occurs. It also verifies that when any ECDM device's INIT bit is set that the INITSTAT bit in the ERSTAT register of all ECDM devices in the same sub-system will also be set.

Response/Messages

After the command has been issued, the following line is printed:

```
ECDM     INITCK: INIT Function Check................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
ECDM     INITCK: INIT Function Check................. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
ECDM     INITCK: INIT Function Check................. Running ---> FAILED

(error message)
```

Here, (error message) is one of the following:

This message is displayed if the ECDM revision read in the ERSTAT register is not supported by the test code:

```
WARNING: Unsupported ECDM Revision Detected - xx !!!
```

ECDM INITCK

This message is displayed if the ECDM revision read in the ERSTAT register is not supported by the test code:

```
WARNING: Unsupported ECDM Revision Detected - xx !!!
```

This message is displayed if the INITSTAT of a particular ECDM is not set when the INIT bit of another ECDM is set:

```
INITSTAT Not Set in ECDM #x When INIT bit Set in ECDM #x
```

This message is displayed if an error is detected when verifying a background pattern of all FFFF's prior to testing the INIT functions:

```
Background Data Verify Error:
Address =xxxxxxxx, Expected =ffffffff, Actual =xxxxxxxx
```

This message is displayed if an error is detected when verifying the data pattern produced by a byte write to the test address with the INIT bit set in the ECDM:

```
INIT Data Verify Error:
Address =xxxxxxxx, Expected =00000000, Actual =xxxxxxxx
```

DRAM Sub-System Mapping - MAP **ECDM MAP**

Command Input

```
197-Diag>ECDM MAP
```

Description

This utility displays mapping information for all DRAM sub-systems resident in the system. Possible sub-systems are numbered from 0 to 3, starting with the first sub-system in the DCAM "Disable In/Disable Out" chain. This is, generally, the left most physical sub-system when looking at the front panel(s) in its (their) upright position(s). Refer to the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide*. The command displays the sub-system number followed by its CSR Base address. This is followed by the starting address of the sub-system memory and its size in hex (and in megabytes).

This utility is most helpful in systems which contain multiple DRAM sub-systems such as systems with DRAM mezzanines. The debugger firmware dynamically maps the location of the DRAM sub-systems, placing the largest sub-system at the lowest physical address and, therefore, may not map the base board sub-system at the lowest address. Sub-systems of equal size are mapped as they are found in the chain (left most at the lowest address).

This utility can be used to track a DRAM failure address to the sub-system which contains the specific address.

Response/Messages

The following is the display for a 197LE with 32M on-board DRAM:

```
197-Diag>ecdm map
- DRAM Sub-System 0 - ECDM CSR Address = FFF01100 -
 - Starting Address = 00000000 - Size = 02000000 (32M) -
- DRAM Sub-System 1 - ECDM CSR Address = 00000000 -
 - Starting Address = 00000000 - Size = 00000000 (0M) -
- DRAM Sub-System 2 - ECDM CSR Address = 00000000 -
 - Starting Address = 00000000 - Size = 00000000 (0M) -
- DRAM Sub-System 3 - ECDM CSR Address = 00000000 -
 - Starting Address = 00000000 - Size = 00000000 (0M) -
197-Diag>
```

ECDM MAP

The following is the display for a 197LE with 32M on-board DRAM and a 512M DRAM mezzanine board mounted to the RIGHT of the CPU board:

```
197-Diag>ecdm map
- DRAM Sub-System 0 - ECDM CSR Address = FFF01100 -
 - Starting Address = 20000000 - Size = 02000000 (32M) -
- DRAM Sub-System 1 - ECDM CSR Address = FFF01200 -
 - Starting Address = 00000000 - Size = 20000000 (512M) -
- DRAM Sub-System 2 - ECDM CSR Address = 00000000 -
 - Starting Address = 00000000 - Size = 00000000 (0M) -
- DRAM Sub-System 3 - ECDM CSR Address = 00000000 -
 - Starting Address = 00000000 - Size = 00000000 (0M) -
197-Diag>
```

The following is the display for a 197LE with 32M on-board DRAM and a 512M DRAM mezzanine board mounted to the LEFT of the CPU board:

```
197-Diag>ecdm map
- DRAM Sub-System 0 - ECDM CSR Address = FFF01100 -
 - Starting Address = 00000000 - Size = 20000000 (512M) -
- DRAM Sub-System 1 - ECDM CSR Address = FFF01200 -
 - Starting Address = 20000000 - Size = 02000000 (32M) -
- DRAM Sub-System 2 - ECDM CSR Address = 00000000 -
 - Starting Address = 00000000 - Size = 00000000 (0M) -
- DRAM Sub-System 3 - ECDM CSR Address = 00000000 -
 - Starting Address = 00000000 - Size = 00000000 (0M) -
197-Diag>
```

Note that the DRAM starting addresses in the previous two examples did not change (since the largest sub-system is always mapped at the lowest address). However, the CSR addresses for the two sub-systems are different (due to the change in their physical position in the chain). If the size of the sub-systems had been equal, the starting addresses would also have changed since the firmware maps sub-systems of equal size in the order they are found in the chain (left most first).

Register Checks - REGS

ECDM REGS

Command Input

```
197-Diag>ECDM REGS
```

Description

The purpose of this test is to verify that, for all DRAM sub-systems resident in the system, that the ECDM device registers associated with each sub-system can be accessed using all available data sizes possible (8/16/32/64 bits). It will verify that the ID register value of all ECDM devices is correct for all data size read operations. It will also verify that certain hard-wired control bits in the ECDM are in their expected state and that no ECC error status information has been reported.

Response/Messages

After the command has been issued, the following line is printed:

```
ECDM     REGS: Register Checks....................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
ECDM     REGS: Register Checks....................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
ECDM     REGS: Register Checks....................... Running ---> FAILED

(error message)
```

Here, (error message) is one of the following:

This message is displayed if the ECDM revision read in the ERSTAT register is not supported by the test code:

```
WARNING: Unsupported ECDM Revision Detected - xx !!!
```

ECDM REGS

This message is displayed when a verify error has been detected in the ECDM register image after copying the registers to a test buffer using the indicated transfer size:

```
ECDM #x <MEMCON/ID/ERSTAT/SYNSTAT> Register Error:
Address =xxxxxxxx, Expected =xx, Actual =xx
Errors Detected on xx-bit Transfers
```

SBE Control Options - SBEC

ECDM SBEC

Command Input

```
197-Diag>ECDM SBEC
```

Description

This test verifies that, for all DRAM sub-systems resident in the system, each ECDM device associated with each sub-system can detect single-bit errors and correct and report them as specified by the control bits in the ECDM MEMCON register. It verifies that a single-bit error in a particular line does not effect the other data read from that line and that a read of any sized operand in the line will force the error to be reported exclusively in the ECDM expected.

This test also verifies that various control options available through the ECDM MEMCON register related to single-bit errors function as specified without affecting normal operations. These include the NOCOR bit which disables single-bit error correction for the ECDM, the LOGCER bit which enables the ERLOG signal to be generated when a correctable error is detected, and the EERP bit which enables the ERLOG signal to be output to the BusSwitch when errors are detected. The address of these errors is verified to be latched in the BusSwitch PAL (Processor Address Latch) register when the ERLOG signal is received by the BusSwitch and it is verified to be the address read when the error was detected (if this function is enabled). Additionally, it is verified that if a correctable error is latched in the status registers that encountering another error of this type will not overwrite the original contents of the registers relating to correctable errors. Refer to the ECDM section of the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for more information on ECDM MEMCON control bit functions.

All ECDMs and the ECC circuitry associated with both line buffers within each ECDM are tested independently.

Response/Messages

After the command has been issued, the following line is printed:

```
ECDM     SBEC: SBE Control Options.................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

ECDM SBEC

```
ECDM     SBEC: SBE Control Options................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
ECDM     SBEC: SBE Control Options................... Running ---> FAILED

(error message)
```

Here, (error message) is one of the following:

This message is displayed if the ECDM revision read in the ERSTAT register is not supported by the test code:

```
WARNING: Unsupported ECDM Revision Detected - xx !!!
```

This message is displayed if an error is detected in the ECDM Error Status registers prior to beginning the test checks:

```
Pre-Existing Error Detected Prior to Check:
< generic ECC error status message(s) here >
```

This message is displayed when the test data pattern which contains a single-bit error was not corrected by the ECC hardware when reading the error location:

```
Error Data NOT Corrected - ECDM 0x:
Address =xxxxxxxx, Expected =xxxx, Actual =xxxx
```

This message is displayed when the ERSTAT register value was not the expected value (error or no error) or the syndrome code generated by the ECC circuitry (in the SYNSTAT register) was not the expected value (indicated the wrong data bit in error):

ECDM SBEC

```
Unexpected <ERSTAT/SYNSTAT> for ECDM 0x - Exp.= xx - Rd.= xx -
```

This message is displayed when the BusSwitch PAL register does not contain the expected value for the test conditions:

```
Address Latch Incorrect - Exp = $xxxxxxxx - Rd = $xxxxxxxx
```

This message is displayed when the test expected the ERLOG signal to be asserted by the ECDM in response to a correctable error being detected and sent to the BusSwitch where it should have caused the PALINT interrupt status bit to be set:

```
Interrupt Status NOT Detected - PALINT = $xx -
```

This message is displayed when the error data location has been modified unexpectedly when the test error location was read with the indicated state of a specified bit in the MEMCOM register:

```
Error Data Modified w/<MEMCON bit status> - ECDM x
Address =xxxxxxxx, Expected =xxxx, Actual =xxxx+
```

This message is displayed when unexpected error status was detected during read operations with the indicated state of a specified bit in the MEMCOM register:

```
<Incorrect/No> Error Status Detected in ECDM x w/<MEMCON status> - ERSTAT = xx
```

This message is displayed when an unexpected ERLOG signal must have triggered the PALINT interrupt status bit without the EERP bit in the ECDM MEMCON register being set:

```
PAL Interrupt Status Detected wo/EERP - PALINT = xx
```

One of the following messages may be displayed if data patterns are disturbed as a result of read operations detecting correctable errors with a pending correctable error already latched in the ERSTAT register:

```
SBE2 Error Data Not Corrected - ECDM x:
Address =xxxxxxxx, Expected =xxxx, Actual =xxx
```

```
SBE1 Data NOT Corrected w/Prior ERROR Latched - ECDM x:
Address =xxxxxxxx, Expected =xxxx, Actual =xxx
```

This message is displayed when the syndrome code in the SYNSTAT register is not the expected value for the forced single-bit error data:

```
Unexpected SYNSTAT for ECDM x - Exp.= xx - Rd.= xx -
```

This message is displayed when the syndrome code in the SYNSTAT register is the expected value for the second forced error which should not override the previously latched error information:

```
SYNSTAT Overwritten By SBE1 Read
```

This message is displayed when the BusSwitch PAL register does not contain the expected value after a read operation with the indicated state of a specified bit in the MEMCOM register:

```
BusSwitch PAL Error on ERSTAT/ERLOG Latch Test:
Address Latch Incorrect - Exp = $xxxxxxxx - Rd = $xxxxxxxx
```

Permutations - SBEP

ECDM SBEP

Command Input

```
197-Diag>ECDM SBEP
```

Description

This test verifies that, for all DRAM sub-systems resident in the system, that the ECDM devices associated each sub-system are capable of detecting, correcting, and reporting correctly single-bit errors in all possible bit combinations in the ECC word as well as in any check bit positions associated with them. It will also verify that any type access into a line containing a single-bit error will result in that error being reported accurately. This is done for the ECC circuitry for both line buffers for all ECDMs. The test is performed once with a background pattern of zeros and then with a pattern of all ones. Refer to Figure A-1 in Appendix A for bit number mapping information for this test.

Response/Messages

After the command has been issued, the following line is printed:

```
ECDM     SBEP: SBE Permutations..................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
ECDM     SBEP: SBE Permutations..................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
ECDM     SBEP: SBE Permutations..................... Running ---> FAILED

(error message)
```

Here, (error message) is one of the following:

This message is displayed if the ECDM revision read in the ERSTAT register is not supported by the test code:

```
WARNING: Unsupported ECDM Revision Detected - xx !!!
```

ECDM SBEP

This message is displayed if an error is detected in the line test buffer when verifying one of the background patterns prior to SBE testing:

```
Error Verifying Background Pattern
Address =xxxxxxxx, Expected =xxxxxxxx, Actual =xxxxxxxx
```

This message is displayed if the data associated with the forced single-bit error in the test line was not corrected when read during the initial data check:

```
Error Data NOT Corrected - ECDM #x:
< generic ECC error status message(s) here >
Error Forced at Address xxxxxxxx -
```

This message is displayed if the forced single-bit error in the test line was not detected during the initial half-word read operation:

```
No Error Detected on Initial Halfword Read
Address =xxxxxxxx, Expected =xxxx, Actual =xxxx
< generic ECC error status message(s) here >
```

This message is displayed if any error is found when verifying that any data bit error forced in the test line was detected and reported as expected:

```
Test Addr = $xxxxxxxx - Background = xxxx - ECDM #x
LINE Data Bit = xxx - Half-Word Bit Mask = xxxx -
```

This message is displayed if any error is found when verifying that any check bit error forced in the test line was detected and reported as expected:

```
Test Addr = $xxxxxxxx - Background = xxxx - ECDM #x
Check Bit Mask = xx -
```

This message is displayed when the ERSTAT or SYNSTAT register value was not the expected value (error or no error):

ECDM SBEP

```
Unexpected <ERSTAT/SYNSTAT> for ECDM x - Exp.= xx - Rd.= xx -
```

This message is displayed if an error is detected in the Error Status register of an ECDM not associated with the forced error:

```
Error Detected in Wrong ECDM - x - ERSTAT = xx
Expected ERROR in ECDM x
```

This message is displayed when the BusSwitch PAL register does not contain the expected value for the test conditions:

```
Address Latch Incorrect - Exp = $xxxxxxxx - Rd = $xxxxxxxx
```

This message is displayed when the test expected the ERLOG signal to be asserted by the ECDM in response to a non-correctable error being detected and sent to the BusSwitch where it should have caused the PALINT interrupt status bit to be set:

```
Interrupt Status NOT Detected - PALINT = $xx -
```

This message is displayed if verification errors are detected to indicate the type of read operations which were being performed when the errors were detected:

```
- Error Reading <8/16/32/64>-bit Operand -
```

ECDM Common Test Error Messages

The ECDM test group uses certain error message formats extensively. Some of these are presented here and are referenced in other error message displays by their symbolic names indicated here.

This message section is the generic display when any unexpected ECC error is detected. It is usually proceeded by the reason for considering this a failure.

```
##### Error Report for ECDM CSR at $FFF01100 #####
Error Detected in ECDM #00:
 MEMCON = xx - ERSTAT = xx - SYNSTAT = xx
 Syndrome Code = xx - Error in ECDM Segment Rx, Data Bit x
            -or-   Error in ECDM Check Bit x
            -or-   Not Found in SBE Tables - Multi-Bit Error
< errors detected in other ECDMs here >
BusSwitch PALINT Register = 30
Address Latched in BusSwitch PAL = $00008525
.
```

The banner line indicates the ECDM CSR address of the DRAM sub-system under test. This is followed by error reports for each ECDM in the set which has detected any error. These reports include the values for the three main ECC control/status registers. Refer to the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for bit definitions for these registers. Syndrome status information evaluation is on the next line of each error report. The syndrome code from the SYNSTAT register is evaluated to one of three indications. Either the ECDM data segment and data bit are presented for correctable data channel errors, the check bit number is presented for single bit errors in the checkbit DRAMs, or the value indicates a non-correctable error. These error reports are followed by the BusSwitch PALINT register value which indicates whether the PALINT feature is enabled and whether it has detected an error signal from the ECDMs. If a signal is indicated, the value of the PALINT register in the BusSwitch is displayed indicating the processor address which was present on the bus at the time an error was detected (ECDM ERLOG signal asserted).

If multiple DRAM sub-systems exist in the system, they are tested one at a time and various error checks are performed prior to executing the test suite on each of the sub-systems.

The following error messages may be the top level routines which control testing of all the sub-systems resident.

This message is displayed if the diagnostic initialization code did not locate and log the ECDM CSR register under test during board start-up:

```
Invalid ECDM CSR Address - $xxxxxxxx
```

This message is displayed if the diagnostic initialization code found and logged the ECDM CSR register under test during board start-up but determined that the memory size associated with the sub-system was zero:

```
Sub_System x Has No Memory Assigned
```

This message is displayed if no valid sub-systems were found to be tested:

```
No Sub-Systems Tested !!
```

DRAM Controller and Address Multiplexer (DCAM) Utilities

These utilities allow the user to manipulate the internal registers of the DCAM ASIC. These are provided for use by trained personnel who are familiar with the operation of the DCAM.

Table 3-6. DCAM Test Group

Mnemonic	Description
RD	DCAM Register Display
RM	DCAM Register Modify

DCAM Configuration Parameters — CF DCAM

Command Input

```
197-Diag>cf dcam
DCAM Configuration Data:
Restore Defaults=0 =FFFFFFFF ?
ECDM CSR Base Address =FFF01100 ?
Diagnostic Control Structure Pointer (READ ONLY) =00000000 (READ ONLY) ?
LMEMINFO Pointer (READ ONLY) =00000000 (READ ONLY) ?
Scroll Count =0000000A ?
197-Diag>
```

Description

User configurable test parameters are available for the **DCAM** test group. Refer to Chapter 2 for information on using the **CF** command to set configuration parameters.

The DCAM configuration parameters are listed and described below.

```
Restore Defaults =0    =FFFFFFFF ?
```

This parameter is used to restore the default values to all of the configuration parameters. If you wish to restore the default values to the **DCAM CF** parameters, you should set this parameter to zero. After this is done, the **DCAM CF** parameters will be restored to the defaults the next time that any DCAM command is executed. (The restoration is actually done by the DCAM command just before execution, so that the DCAM command will act upon the restored default configuration values.)

After restoration, the value of this parameter will reflect the address in local memory where the **DCAM CF** parameters are actually stored.

```
ECDM CSR Base Address   =FFF01100 ?
```

This parameter points to the base address of the DCAM which controls the first memory sub-system found in the configuration. If multiple sub-systems are in the configuration, additional DCAM devices may be found at $FFF01200, $FFF01300, etc. (increments of $100 from the base address of the first DCAM).

The most probable reason to change this parameter is to look at the registers of the different DCAM devices that will exist in MVME197 systems that contains multiple memory sub-systems.

CF DCAM

```
Diagnostic Control Structure Pointer (READ ONLY) =00000000 (READ ONLY) ?
```

This is an internal parameter used for diagnostic development purposes.

```
LMEMINFO Pointer (READ ONLY) =00000000 (READ ONLY) ?
```

This is also an internal parameter used only for diagnostic development purposes.

```
Scroll Count =0000000A ?
```

This hexadecimal count is used to control the number of lines displayed to the console screen before pausing for user input (typing carriage return will display “Scroll Count” number of lines again.). This parameter is provided to prevent parts of the DCAM register display, which is rather lengthy, from scrolling off of the screen. This value may be increased for larger displays.

DCAM Register Display - RD

Command Input

```
197-Diag>DCAM RD
```

Description

This command allows you to examine (but not modify) the internal registers of the DCAM device whose base address is given by the current value of the "ECDM CSR Base Address" configuration parameter.

Refer to the *MVME197LE/DP/SP Single Board Computers Programmer's Reference Guide* for information on the meaning of the contents of various DCAM registers.

Response/Messages

After the command has been issued, the contents of the DCAM's registers will be listed out on the screen.

To prevent any of the display from scrolling off of the console screen before it can be read, only the number of lines given by the "Scroll Count" configuration parameter will be output before the display pauses and prompts for entry of a carriage return before continuing.

```
197-Diag>dcam rd
00-Chip ID =82 (READ -Version =04 (READ ONLY)
02-RSAR0(7-1)/DISRAM(0) =00
03-RSAR1(7-1)/Scrub1(0) =00
04-CASCSEL(7)/CASCLK(6-5)/PGMODE(4)/1BANK(3)/DRAMSZ(2-0) =D9
05-REFCNT7-0 =1C
06-REFTAIL4-1(7-4)/REFCNT11-8(3-0) =41
07-KILCACHE(7)/DISINVL(6)/RDTAIL5-1(5-1)/RTCLKSEL(0) =05
08-READACK7-1(7-1)/INTR(0) =05
09-*(7)/RDOE6-1(6-1)/SPLITRAS(0) =40
Press “RETURN” to continue
```

After entering a carriage return, more lines will be output. It may be necessary to enter several carriage returns before all of the DCAM registers have been displayed on the screen.

DCAM RD

The rest of the DCAM registers are shown below.

```
10-FECCKSL(7)/BREADOE6-1(6-1)/PCHGCLKSL(0) =A1
11-PCHG7-0 =02
12-SLECDM5-2(7-4)/FLECDM4-1(3-0) =04
13-*(7)/ERAMOE6-1(6-1)/ROECLKSL(0) =41
14-*(7)/RMWRMOE6-1(6-1)/RMWOE5(0) =08
15-CSRTAIL7-1(7-1)/*(0) =08
16-BWRTTL4-1(7-4)/RMWOE4-1(3-0) =28
17-SECCLKSL(7)/RMWOCKSL(6)/BWRITE5-1(5-1)/WRCLKSEL(0) =89
18-*(7-6)/RMW5-1(5-1)/*(0) =08
19-RMWTAIL7-1(7-1)/RMWTCSL(0) =40
20-CBRDOE3-1(7-5)/*(4)/CREADOE3-1(3-1)/BWRTCSL(0) =23
21-SCRUBCNT9-2 =00
22-SCRUBCNT17-10 =00
23-SCRUBCNT25-18 =00
24-*(7)/SCRUBCNT32-26 =00
25-*(7-5)/CBTAIL4-1(4-1)/CBTLCKSL(0) =04
26-ECDM_CSR7-6(7-6)/*(5-0) =00
27-ECDM_CSR15-8 =11
28-ECDM_CSR23-16 =F0
29-ECDM_CSR31-24 =FF
30-*(7-6)/BRDTAIL5-1(5-1)/*(0) =04
197-Diag>
```

DCAM Register Modify - RM

DCAM RM

Command Input

```
197-Diag>DCAM RM
```

Description

This command allows you to examine and modify the internal registers of the DCAM device whose base address is given by the current value of the “ECDM CSR Base Address” configuration parameter. A register editor allows you to change the contents of the registers. The command will then present you with the option of writing the changes back into the DCAM.

CAUTION

This command is intended for use by trained personnel. Changing DCAM register contents can cause the MVME197 board to malfunction. In general, you should never need to change any of the DCAM register contents.

Refer to the *MVME197LE/DP/SP Single Board Computers Programmer’s Reference Guide* for information on the meaning of the contents of various DCAM registers.

Response/Messages

The **DCAM RM** command invokes the first DCAM register, which will be listed and its contents displayed. A question-mark prompt will then be displayed:

```
00-Chip ID =82 (READ ONLY) ?
```

If the words, “READ ONLY”, appear after the register contents, then the register cannot be modified. The DCAM’s first few registers are like this. Enter carriage return to advance to the next register:

```
00-Chip ID =82 (READ ONLY) ? <CR>
01-Version =04 (READ ONLY) ? <CR>
02-RSAR0(7-1)/DISRAM(0) =00 ?
```

For modifiable registers, you may enter a value to be written to the register after the question-mark prompt.

Entering a carriage return causes the editor to scroll to the next register to be displayed. There are several special keystrokes to control scrolling:

- ❏ "=" (equals) followed by the carriage return will turn off scrolling. Subsequent carriage returns will redisplay the same register.
- ❏ "^" (up-caret) will set scrolling *backward*. Subsequent carriage returns will back up the editor to the previous register.
- ❏ "v" or "V" will set scrolling *forward*. This is the default when **DCAM RM** is invoked.
- ❏ "." (period) will exit the register editor.

Example:

```
197-Diag>
197-Diag>dcam rm
00-Chip ID =82 (READ ONLY) ?
01-Version =04 (READ ONLY) ?
02-RSAR0(7-1)/DISRAM(0) =00 ?
03-RSAR1(7-1)/Scrub1(0) =00 ?
04-CASCSEL(7)/CASCLK(6-5)/PGMODE(4)/1BANK(3)/DRAMSZ(2-0) =D9 ?
05-REFCNT7-0 =1C ? ^
04-CASCSEL(7)/CASCLK(6-5)/PGMODE(4)/1BANK(3)/DRAMSZ(2-0) =D9 ?
03-RSAR1(7-1)/Scrub1(0) =00 ? =
03-RSAR1(7-1)/Scrub1(0) =00 ?
03-RSAR1(7-1)/Scrub1(0) =00 ?
03-RSAR1(7-1)/Scrub1(0) =00 ? v
04-CASCSEL(7)/CASCLK(6-5)/PGMODE(4)/1BANK(3)/DRAMSZ(2-0) =D9 ?
05-REFCNT7-0 =1C ? .

OK to write to DCAM? (Y/N)? n
No Update Performed!

Re-edit DCAM Image? (Y/N)? n
197-Diag>
```

Notice that after the register editor has been exited (via the period),you are prompted whether to write the edited registers back into the DCAM, and then whether to enter another edit session.

BusSwitch ASIC (BSW) Tests

These sections describe the individual **BSW** tests. These tests check the correct operation of the BusSwitch.

Entering **BSW** without parameters causes all **BSW** tests to execute in the order shown in the table below.

To run an individual test, add that test name to the **BSW** command.

The individual tests are described in alphabetical order on the following pages.

Table 3-7. BSW Test Group

Mnemonic	Description
REGS	Register Checks
TMR1A	Timer 1 Counter
TMR1B	Timer 1 Free-Run
TMR1C	Timer 1 Clear on Compare
TMR1D	Timer 1 Overflow Counter
TMR1E	Timer 1 Interrupts
TMR2A	Timer 2 Counter
TMR2B	Timer 2 Free-Run
TMR2C	Timer 2 Clear on Compare
TMR2D	Timer 2 Overflow Counter
TMR2E	Timer 2 Interrupts
PCLK	Prescaler/Clock Accuracy
PADJ	Prescaler Clock Adjust
MERR	Memory Error Interrupt

Table 3-7. BSW Test Group (Continued)

Mnemonic	Description
XINT	External Interrupt
SPINT	Spurious Interrupt
VBR	Vector Base Register
CPINT	Cross Processor Interrupts
CPINTI	Dynamic CPINT Extensions: Internal
CPINTX	Dynamic CPINT Extensions: External
ISTR	Interrupt Steering

BSW Configuration Parameters CF BSW

Command Input

```
197-Diag>cf BSW
BSW Configuration Data:
Restore Defaults (=0) =00000000 ?
197-Diag>
```

Description

User configurable test parameters are available for the **BSW** test group. Refer to Chapter 2 for information on using the **CF** command to set configuration parameters.

The **BSW** test parameters are listed in the command input block above and described below.

```
Restore Defaults (=0) =00000000 ?
```

This parameter when set to zero will force the restoring of the default values to all of the configuration parameters. Once initialized, this parameter will reflect the address where all of the parameters are stored in local memory.

Note **There are no other configuration parameters for the BSW test group.**

Cross Processor Interrupts - CPINT

BSW CPINT

Command Input

```
197-Diag>BSW CPINT
```

Description

This test provides a static check of the BusSwitch Cross-Processor Interrupt functions. It performs a check of the CPINT register functions for the two supported processors.

This test uses a special MPU “perspective” bit (TCPU1) in the BusSwitch GCSR to enable access to and control of CPINT registers for each MPU as required. Refer to the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer’s Reference Guide* for more information on TCPU1 bit usage.

Cross-processor interrupt issuance and received status for both processors is checked statically. No NMI exception should be generated by this test.

This test forces MPU 0 to execute this function.

Response/Messages

After the command has been issued, the following line is printed:

```
BSW      CPINT: Cross Processor Interrupts........... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
BSW      CPINT: Cross Processor Interrupts........... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
BSW      CPINT: Cross Processor Interrupts........... Running ---> FAILED

(error message)
```

BSW CPINT

The following error messages are possible:

This error is displayed if the BusSwitch CPINT register for MPU 0 does not clear during register initialization prior to testing.

```
MPU 0 CPINT Register Not Cleared by ICLR Write
CPINT Status = 56 - BSW GCSR = 2
```

This error is displayed if a processor other than MPU 0 was executing upon entry to the test and the test code could not switch control to MPU 0 to execute the test.

```
cpint_init(SETUP): Could Not Switch to MPU 0
```

This error is displayed when the CINT bit in the CPINT register for MPU 0 does not set when MPU 0 has issued a CPINT signal to MPU 1. The CINT bit for MPU 0 should reflect MPU 1's INT bit status.

```
MPU 1 CPINT Bit (CINT) Not Set By Asserting SCPI From MPU 0
CPINT Status = 56 - BSW GCSR = 2
```

This error is displayed when the INT bit in the CPINT register for MPU 1 does not set when MPU 0 has issued a cross-processor interrupt signal to MPU 1.

```
MPU 1 CPINT Bit (INT) Not Set By Asserting SCPI From MPU 0
CPINT Status = 56 - BSW GCSR = 2
```

This error is displayed when the INT bit in the CPINT register for MPU 1 does not clear after setting its ICLR bit.

```
MPU 1 CPINT Register Not Cleared by MPU 1 ICLR Write
CPINT Status = 56 - BSW GCSR = 2
```

BSW CPINT

This error is displayed when the CINT bit in the CPINT register for MPU 1 does not set when MPU 1 has issued a CP interrupt signal to MPU 0. The CINT bit for MPU 1 should reflect MPU 0’s INT bit status.

```
MPU 0 CPINT Bit (CINT) Not Set By Asserting SCPI From MPU 1
CPINT Status = 56 - BSW GCSR = 2
```

This error is displayed when the INT bit in the CPINT register for MPU 0 does not set when MPU 1 has issued a CPINT signal to MPU 0.

```
MPU 0 CPINT Bit (INT) Not Set By Asserting SCPI From MPU 1
CPINT Status = 56 - BSW GCSR = 2
```

This error is displayed if the BusSwitch test is unable to return control to the indicated (original) processor after the test has completed.

```
cpint_init(DONE): Could Not Return Control to MPU “X”
```

Dynamic CPINT Extensions: Internal - CPINTI

BSW CPINTI

Command Input

```
197-Diag>BSW CPINTI
```

Description

This test provides a dynamic check of the internal BusSwitch Cross-Processor Interrupt functions.

Only MPU 0 and MPU 1 are checked by this test. This test is not performed on single processor boards. Cross-processor interrupts are issued and proper NMI exception handling is verified. Control of the test is switched to the sending/receiving MPU as required. Target (and idle) MPUs are forced into polling loops prior to the initiating MPU issuing a CPINT signal. Each MPU sends a CPINT signal to the other MPU (0 and 1 only) and verifies proper processing.

This test forces MPU 0 to begin execution of this test function.

Response/Messages

After the command has been issued, the following line is printed:

```
BSW      CPINTI: Dynamic CPInt Extensions: Internal.. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
BSW      CPINTI: Dynamic CPInt Extensions: Internal.. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
BSW      CPINTI: Dynamic CPInt Extensions: Internal.. Running ---> FAILED

(error message)
```

BSW CPINTI

The following error messages are possible:

This error is displayed if a processor other than MPU 0 was executing upon entry to the test and the test code could not switch control to MPU 0 to execute the test.

```
cpint_init(SETUP): Could Not Switch to MPU 0
```

This error is displayed if the BusSwitch test is unable to return control to the indicated (original) processor after the test has completed.

```
cpint_init(DONE): Could Not Return Control to MPU “X”
```

This error is displayed if the BusSwitch CPINT register for MPU 0 does not clear during register initialization prior to testing.

```
MPU 0 CPINT Register Not Cleared by ICLR Write
CPINT Status = 20 - BSW GCSR = XXXX
```

This error is displayed when the test could not switch control from one MPU to the other as required by the test.

```
Could not switch MPUs.
Could not switch from X to X
```

The test could not set one of the two MPU CPINT register Enable Interrupt bits, thus not allowing NMI exceptions to be posted when the corresponding MPU’s INT bit is set.

```
Could not set BSW_IEN
```

The test timed out waiting for the CPINT recipient (or idle) MPU(s) to acknowledge idle mode in preparation for CPINT issuance.

```
CPINT timed out while waiting for MPUs to go idle
CPINT Status = XX - BSW GCSR = XXXX
```

BSW CPINTI

The MPU is not MASTER when expected. This is possible if the MPU issuing the CPINT does not detect the proper system state prior to initiating a test sequence.

```
MPU is not MASTER when expected
CPINT Status = XX - BSW GCSR = XXXX
```

Dynamic CPINT Extensions: External - CPINTX BSW CPINTX

Command Input

```
197-Diag>BSW CPINTX
```

Description

This test provides a dynamic check of the external BusSwitch Cross-Processor Interrupt extensions (if resident).

This hardware is normally resident only on quad processor compatible boards. Cross-processor interrupts will be issued (and verified to be handled properly) from each resident MPU to all other resident MPUs.

This test is not performed on single processor boards. Cross-processor interrupts are issued and proper NMI exception handling is verified. Control of the test is switched to the sending/receiving MPU as required. Target (and idle) MPUs are forced into polling loops prior to the initiating MPU issuing a CPINT signal.

This test forces MPU 0 to begin execution of this function.

Response/Messages

After the command has been issued, the following line is printed:

```
BSW      CPINTX: Dynamic CPInt Extensions: External.. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
BSW      CPINTX: Dynamic CPInt Extensions: External.. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
BSW      CPINTX: Dynamic CPInt Extensions: External.. Running ---> FAILED

(error message)
```

BSW CPINTX

The following error messages are possible:

This error is displayed if a processor other than MPU 0 was executing upon entry to the test and the test code could not switch control to MPU 0 to execute the test.

```
cpint_init(SETUP): Could Not Switch to MPU 0
```

This error is displayed if the BusSwitch test is unable to return control to the indicated (original) processor after the test has completed.

```
cpint_init(DONE): Could Not Return Control to MPU “X”
```

This error is displayed if the BusSwitch CPINT register for MPU 0 does not clear during register initialization prior to testing.

```
MPU 0 CPINT Register Not Cleared by ICLR Write
CPINT Status = XX - BSW GCSR = XXXX
```

This error is displayed when the test could not switch control from one MPU to the other as required by the test.

```
Could not switch MPUs.
Could not switch from X to X
```

The test could not set the CPINT register Enable Interrupt bit for one of the MPUs to allow NMI exceptions to be generated when its INT bit is set.

```
Could not set BSW_IEN
```

The test timed out waiting for the CPINT recipient (or idle) MPU(s) to acknowledge idle mode in preparation for CPINT issuance.

```
CPINT timed out while waiting for MPUs to go idle
CPINT Status = XX - BSW GCSR = XXXX
```

BSW CPINTX

The MPU is not MASTER when expected. This is possible if the MPU issuing the CPINT does not detect the proper system state prior to initiating a test sequence.

```
MPU is not MASTER when expected
CPINT Status = XX - BSW GCSR = XXXX
```

Interrupt Steering Test - ISTR

BSW ISTR

Command Input

```
197-Diag>BSW ISTR
```

Description

This test verifies that maskable interrupts can be “steered” to a particular MPU (0 or 1 only) using the BusSwitch ISEL0 and ISEL1 register. This test is not executed for single processor boards.

Control of the test is switched to the target MPU and an interrupt is generated and verified to be directed to and handled by only the expected MPU. This is done for all seven interrupt levels on each MPU. The External Interrupt register is used to generate the required interrupts.

Response/Messages

After the command has been issued, the following line is printed:

```
BSW      ISTR: Interrupt Steering Tests............. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
BSW      ISTR: Interrupt Steering Tests............. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
BSW      ISTR: Interrupt Steering Tests............. Running ---> FAILED

(error message)
```

The following error messages are possible:

The test initializes the interrupt controller register to zero. If the interrupt controller does not clear then this error message is displayed.

```
Interrupt Control Register did not clear
Address = fff0006E, Expected = 0, Actual = X
```

BSW ISTR

The test sets the edge sensitive controller bit. If the bit does not set, then this error message is displayed.

```
E/L bit did not set
Address = fff0006E, Expected = 40, Actual = 0
```

The test sets the interrupt enable bit. If the interrupt enable bit cannot be set then this error message is displayed.

```
Interrupt Enable bit did not set
Address = fff0006E, Expected = 10, Actual = 00
```

The test generates conditions which should set the interrupt status bit. If the status bit did not set then this error message is displayed.

```
Interrupt Status bit did not set
Status: Expected = 20, Actual = 23
Vector: Expected = 0, Actual = 0
State : IRQ Level = 0, VBR = 5
```

This error is displayed when the test could not switch control from one MPU to the other as required by the test.

```
Could not switch MPUs.
Could not switch from X to X
```

The test found an incorrect vector returned from the exception handler. The exception vector did not indicate an external interrupt.

```
Incorrect Vector type
Status: Expected = 20, Actual = 23
Vector: Expected = 0, Actual = 0
State : IRQ Level = 0, VBR = 5
```

The test got an unexpected vector for the interrupt tested.

```
Unexpected Vector taken
Status: Expected = 20, Actual = 23
Vector: Expected = 0, Actual = 0
State : IRQ Level = 0, VBR = 5
```

The target MPU received an incorrect interrupt level value.

```
Incorrect Interrupt Level
Level: Expected = 3, Actual = 5
State: IRQ Level = 3, VBR = 5
```

The hardware did not generate the expected interrupt conditions.

```
Interrupt did not occur
Status: Expected = 20, Actual = 23
Vector: Expected = 0, Actual = 0
State : IRQ Level = 0, VBR = 5
```

Memory Error Interrupt - MERR

BSW MERR

Command Input

```
197-Diag>BSW MERR
```

Description

This test is designed to test the BusSwitch memory error (PALINT) interrupt functions. Read/write register bits are verified and interrupts are generated and verified for all interrupt levels.

Response/Messages

After the command has been issued, the following line is printed:

```
BSW      MERR: Memory Error Interrupt................ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
BSW      MERR: Memory Error Interrupt................ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
BSW      MERR: Memory Error Interrupt................ Running ---> FAILED

(error message)
```

The following error messages are possible:

The test initializes the interrupt controller register to zero. If the interrupt controller does not clear then this error message is displayed.

```
Interrupt Control Register did not clear
Address = fff0006D, Expected = 0, Actual = 23
```