MVME197LEIG/D1A1
November 1993

Supplement to MVME197LE Single Board Computer Installation Guide (MVME197LEIG/D1)

The attached pages are replacements for the corresponding pages in the manual. Place this page behind the title page of the manual as a record of this change. Please replace the pages according to the following table:

Replace Old	With New
ix / x through xv / xvi, 1-3 / 1-4, 1-7 / 1-8, 1-9 / 1-10, 3-1 / 3-2 through 3-17 / 3-18, 4-13 / 4-14, A-1 / A-2, A-3 / A-4, IN-1 / IN-2 through IN-5 / IN-6	ix / x through xv / xvi, 1-3 / 1-4 1-7 / 1-8, 1-9 / 1-10, 3-1 / 3-2 through 3-21 / 3-22, 4-13 / 4-14, A-1 / A-2, A-3 / A-4, IN-1 / IN-2 through IN-5 / IN-6

- ❏ A vertical bar (|) in the margin of a replacement page indicates a text change or addition.
- ❏ The supplement number is shown at the bottom of each replacement page.

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

Contents

CHAPTER 2 HARDWARE PREPARATION AND INSTALLATION

CHAPTER 3 DEBUGGER GENERAL INFORMATION

List of Figures

List of Tables

Features

These are some of the major features of the MVME197LE single board computer:

❏ MC88110 RISC Microprocessor

❏ 32 or 64 megabytes of 64-bit Dynamic Random Access Memory (DRAM) with error correction

❏ 1 megabyte of Flash memory

❏ Six status LEDs (FAIL, RUN, SCON, LAN, SCSI, and VME)

❏ 8 kilobytes of Static Random Access Memory (SRAM) and Time of Day (TOD) clock with Battery Backup RAM (BBRAM)

❏ Two push-button switches (ABORT and RESET)

❏ 128 kilobytes of BOOT ROM

❏ Six 32-bit tick timers for periodic interrupts

❏ Watchdog timer

❏ Eight software interrupts

❏ I/O

- SCSI Bus interface with Direct Memory Access (DMA)
- Four serial ports with EIA-232-D buffers
- Centronics printer port
- Ethernet transceiver interface

❏ VMEbus interface

- VMEbus system controller functions
- VMEbus interface to local peripheral bus (A24/A32, D8/D16/D32 BLT (D8/D16/D32/D64))(BLT = Block Transfer)
- Local peripheral bus to VMEbus interface (A24/A32, D8/D16/D32 BLT (D16/D32/D64))
- VMEbus interrupter
- VMEbus interrupt handler
- Global CSR for inter-processor communications
- DMA for fast local memory - VMEbus transfers (A16/A24/A32, D16/D32 BLT (D16/D32/D64))

Specifications

The specifications for the MVME197LE are listed in Table 1-1.

Table 1-1. MVME197LE Specifications

Characteristics	Specifications
Power requirements	+5 Vdc (± 2.5%), 4 A (typical), 5 A (maximum) +12 Vdc (± 2.5%), 100 mA (maximum) -12 Vdc (± 2.5%), 100 mA (maximum)
Operating temperature	0° to 55° C at point of entry of forced air (approximately 490 LFM)
Storage temperature	-40° to 85° C
Relative humidity	5% to 90% (non-condensing)
Physical dimensions:	Double-high VMEboard
PC board	
Height	9.187 inches (233.35 mm)
Width	6.299 inches (160.00 mm)
Thickness	0.063 inch (1.60 mm)
PC board with connectors and front panel	
Height	10.309 inches (261.85 mm)
Width	7.4 inches (188.00 mm)
Thickness	0.80 inch (20.32 mm)
Board connectors:	
P1 connector	96-pin connector which provides the interface to the VMEbus signals.
P2 connector	96-pin connector which provides the interface to the extended VMEbus signals and other I/O signals.
J1 connector	20-pin connector which provides the interface to the remote reset, abort, the LEDs, and three general purpose I/O signals.
J2 connector	249-pin connector which provides the interface to the MC88110 address, data, and control signals to and from the mezzanine expansion.

Block Diagram

Figure 1-1 is a general block diagram of the MVME197LE.

Data Bus Structure

The data bus structure is arranged to accommodate the various 8-bit, 16-bit, 32-bit, and 64-bit devices that reside on the module. Refer to the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* and to the user's guide for each device to determine its port size, data bus connection, and any restrictions that apply when accessing the device.

MC88110 MPU

The MVME197LE is based on the MC88000 family and uses one MC88110 RISC (Reduced Instruction Set Computer) microprocessor unit. Refer to the *MC88110 Second Generation RISC Microprocessor User's Manual* for more information.

BOOT ROM

A socket for a 32-pin PLCC / CLCC ROM / EPROM referred to as BOOT ROM or DROM (Download ROM) is provided. It is organized as a 128K x 8 device, but as viewed from the processor it looks like a 16K x 64 memory. This memory is mapped starting at location $FFF80000, but after a local reset it is also mapped at location 0, providing a reset vector and bootstrap code for the processor. The DR0 bit in the General Control Register (GCR) of the PCCchip2 must be cleared to disable the BOOT ROM memory map at 0.

FLASH Memory

Up to 1MB of FLASH memory is available on the board. FLASH memory works like EPROM, but can be erased and reprogrammed by software. It is organized as 32 bits wide, but to the processor it looks as 64 bits wide. It is mapped at location $FF800000. Reads can be of any size, including burst transfers, but writes are always 32 bits wide, regardless of the size specified for the transfer. For this reason, software should only use 32-bit write transfers. This memory is controlled by the BusSwitch, and the memory size, access time, and write enable capability can be programmed via the ROM Control Register (ROMCR) in the BusSwitch. The FLASH memory can be accessed from the processor bus only. It is not accessible from the local peripheral bus or VMEbus.

Onboard DRAM

The MVME197LE onboard DRAM (2 banks of 32MB memory, one optionally installed) is sized at 32MB using 1M x 4 devices and configured as 256 bits wide. The DRAM is four-way interleaved to efficiently support cache burst cycles. The DRAM is controlled by the DCAM and ECDM, and the map decoders in the DCAM can be programmed through the I2Cbus interface in the ECDM to accommodate different base address(es) and sizes. The onboard DRAM is not disabled by a local peripheral bus reset. Refer to the *DCAM* and *ECDM* chapters in the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer’s Reference Guide* for detailed programming information.

Battery Backup RAM and Clock

The MK48T08 RAM and clock chip is used on the MVME197LE. This chip provides a time of day clock, oscillator, crystal, power fail detection, memory write protection, 8KB of RAM, and a battery in one 28-pin package. The clock provides seconds, minutes, hours, day, date, month, and year in BCD 24-hour format. Corrections for 28-, 29-, (leap year) and 30-day months are automatically made. No interrupts are generated by the clock. The MK48T08 is an 8-bit device; however the interface provided by the PCCchip2 supports 8-, 16-, and 32-bit accesses to the MK48T08. Refer to the *PCCchip2* chapter in the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer’s Reference Guide* and to the MK48T08 data sheet for detailed programming information.

VMEbus Interface

The local peripheral bus to VMEbus interface, the VMEbus to local peripheral bus interface, and the local-VMEbus DMA controller functions on the MVME197LE are provided by the VMEchip2. The VMEchip2 can also provide the VMEbus system controller functions. Refer to the *VMEchip2* chapter in the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer’s Reference Guide* for detailed programming information.

I/O Interfaces

The MVME197LE provides onboard I/O for many system applications. The I/O functions include serial ports, a printer port, an Ethernet transceiver interface, and a SCSI mass storage interface.

Serial Port Interface

The CD2401 serial controller chip (SCC) is used to implement the four serial ports. The serial ports support the standard baud rates (110 to 38.4K baud). Serial port 4 also supports synchronous modes of operation.

The four serial ports are different functionally because of the limited number of pins on the I/O connector. Serial port 1 is a minimum function asynchronous port. It uses RXD, CTS, TXD, and RTS. Serial ports 2 and 3 are full function asynchronous ports. They use RXD, CTS, DCD, TXD, RTS, and DTR. Serial port 4 is a full function asynchronous or synchronous port. It can operate at synchronous bit rates up to 64k bits per second. It uses RXD, CTS, DCD, RTS, and DTR. It also interfaces to the synchronous clock signal lines. Refer to the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for drawings of the serial port interface connections.

All four serial ports use EIA-232-D drivers and receivers located on the main board, and all the signal lines are routed to the I/O connector. The configuration headers are located on the MVME712X transition board. An external I/O transition board such as the MVME712X should be used to convert the I/O connector pinout to industry-standard connectors.

The interface provided by the PCCchip2 allows the 16-bit CD2401 to appear at contiguous addresses; however, accesses to the CD2401 must be 8 or 16 bits. 32-bit accesses are not permitted. Refer to the CD2401 data sheet and to the *PCCchip2* chapter in the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for detailed programming information.

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

Printer Interface

The MVME197LE has a Centronics-compatible printer interface. The printer interface is provided by the PCCchip2. Refer to the *PCCchip2* chapter in the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for detailed programming information and for drawings of the printer port interface connections.

Ethernet Interface

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

Every MVME197LE module is assigned an Ethernet Station Address. This address is $08003E2XXXXX, where XXXXX is the unique 5-nibble number assigned to the board (i.e., every MVME197LE has a different value for XXXXX).

The Ethernet Station Address is displayed on a label attached to the VMEbus P2 connector. In addition, the eight bytes including the Ethernet address are stored in the configuration area of the BBRAM, with the two lower bytes of those set to 0. That is, 08003E2XXXXX0000 is stored in the BBRAM. At an address of $FFFC1F2C, the upper four bytes (08003E2X) can be read. At an address of $FFFC1F30, the lower four bytes (XXXX0000) can be read. Refer to the BBRAM, TOD Clock memory map description later in this chapter. The MVME197LE debugger has the capability to retrieve or set the Ethernet address.

If the data in the BBRAM is lost, the user should use the number on the VMEbus P2 connector label to restore it. Refer to Appendix A (*Configure and Environment Commands*) in this guide or to the *MVME197BUG 197Bug Debugging Package User's Manual*.

The Ethernet transceiver interface is located on the MVME197LE main module, and the industry standard connector is located on the MVME712X transition module.

Support functions for the 82596CA are provided by the PCCchip2. Refer to the *82596CA LAN Coprocessor User's Manual* and to the *PCCchip2* chapter in the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for detailed programming information.

SCSI Interface

The MVME197LE provides for mass storage subsystems through the industry-standard SCSI bus. These subsystems may include hard and floppy disk drives, streaming tape drives, and other mass storage devices. The SCSI interface is implemented using the NCR 53C710 SCSI I/O controller.

Support functions for the 53C710 are provided by the PCCchip2. Refer to the *NCR 53C710 SCSI I/O Processor Data Manual* and to the *PCCchip2* chapter in the *MVME197LE, MVME197DP, and MVME197SP Single Board Computers Programmer's Reference Guide* for detailed programming information.

SCSI Termination

The system configurer must ensure that the SCSI bus is terminated properly. On the MVME197LE, the terminators are located on the P2 transition board. The +5V power to the SCSI bus termination resistors is provided by the P2 transition board.

DEBUGGER GENERAL INFORMATION 3

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 and user friendliness, and yet stresses portability and ease of maintenance. This member of the M88000 firmware family is implemented on the MVME197LE Single Board Computer, 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 microcomputers. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 197Bug includes commands for display and modification of memory, breakpoint and tracing capabilities, a powerful assembler / disassembler useful for patching programs, and a self-test at power-up feature which verifies the integrity of the system. Various 197Bug routines that handle I / O, data conversion, and string functions are available to user programs through the TRAP #496 handler. The TRAP #496 handler is accessible through any of the trap exception instructions TB0, TB1, TBND, and TCND, with trap vector #496.

197Bug consists of three parts:

❏ A command-driven user-interactive software debugger, described in a later chapter (*Using the 197Bug Debugger*) and hereafter referred to as “the debugger”.

❏ A command-driven diagnostic package for the MVME197LE hardware, described in the *MVME197BUG 197Bug Diagnostic Firmware User’s Manual* and 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 the user is in the debugger directory, then the debugger prompt “**197-Bug>**” is displayed and the user has all of the debugger commands at his or her disposal. If in the diagnostic directory, then the diagnostic prompt "**197-Diag>**" is displayed and the user has all of the diagnostic commands at his disposal as well as all of the debugger commands. The user may switch between directories by using the Switch Directories (**SD**) command, or may examine the commands in the particular directory that the user is currently in by using the Help (**HE**) command.

Because 197Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. 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.

Comparison With M68000-Based Firmware

Those users who have used one or more of Motorola's other debugging packages will find 197Bug very similar, after making due allowances for the architectural differences between the M68000 and M88000 CPU architectures. These are primarily reflected in the instruction mnemonics and addressing modes of the assembler / disassembler, and in the use of registers instead of the stack for the passing of arguments to or from the TRAP #496 handler. Some effort has also been made to make the interactive commands more consistent. For example, delimiters between commands and arguments may now be commas or spaces interchangeably.

197Bug Implementation

FLASH-Based Debugger

197Bug is contained in the FLASH memory devices located onboard the MVME197LE module. The FLASH devices are electrically re-writable and may be reprogrammed without removing the physical devices from the MVME197LE module. This allows the user 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. Users 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 MVME197LE module. This scaled-down 197Bug is referred to as the "BootBug", or "197BBug".

When the MVME197LE 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.

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* [*increment*] [**;** **B**|**H**|**W**]
BM	Block of Memory Move	**BM** *RANGE DEL ADDR* [**;** **B**|**H**|**W**]

Command Mnemonic	Command Title	Command Line Syntax
BS	Block of Memory Search	**BS** *RANGE DEL TEXT* [**;B** | **H** | **W**] or **BS** *RANGE DEL data DEL* [*mask*] [**;B** | **H** | **W**,**N**,**V**]
BV	Block of Memory Verify	**BV** *RANGE DEL data* [*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** [**;**[**H**] [**A**]]
LO	Load S-Records from Host	**LO** [*n*] [*ADDR*] [**;X** | **C** | **T**] [=*text*]
MD	Memory Display	**MD**[**S**] *ADDR*[**:***COUNT* | *ADDR*] [**;** [**B** | **H** | **W** | **S** | **D** | **DI**]]
MM	Memory Modify	**MM** *ADDR*[**;**[[**B** | **H** | **W** | **S** | **D**][**A**] [**N**]] | [**DI**]]
MS	Memory Set	**MS** *ADDR* {*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*]
PFLASH	Program FLASH Memory	**PFLASH** *SSADDR SEADDR DSADDR* [*IEADDR*] [**;**[**A** | **R**] [**X**]] **PFLASH** *SSADDR:COUNT DSADDR* [*IEADDR*] [**;**[**B** | **H** | **W**] [**A** | **R**] [**X**]]
SET	Set Time and Date	**SET** *mmddyyhhmm*

Command Mnemonic	Command Title	Command Line Syntax
TIME	Display Time and Date	**TIME** [**;C** | **L** | **O**]
TM	Transparent Mode	**TM** [*n*] [*ESCAPE*]
VE	Verify S-records Against Memory	**VE** [*n*] [*ADDR*] [**;X** | **C**] [=*text*]

Detailed descriptions of these commands may be found in the *197Bug Debugger Command Set* chapter of 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.

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 MVME197LE module, for 197Bug to operate properly with the MVME197LE, follow this set-up procedure.

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

1. Turn all equipment power OFF. Refer to the *Hardware Preparation and Installation* chapter in this 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 MVME197LE.
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 MVME197LE 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 the user gets garbled messages and missing characters, then the user 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 MVME197LE CD2401 Serial Controller Chip (SCC), or by using the 197Bug **PF** command.

   Note that the MVME197LE also contains a parallel port. To use a parallel device, such as a printer, with the MVME197LE, 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. However, you could also use a module such as the MVME335 for a parallel port connection.

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 tries to autoboot. Refer to the **ENV** and **MENU** commands. They are listed in Table 4-1.

If the confidence test fails, the test is aborted when the first fault is encountered. If possible, an appropriate message is displayed, and control then returns to the menu.

Autoboot

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

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

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

Following this message there is 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 the user wants to gain control without Autoboot, the <BREAK> key or the software 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 Appendix A for more details.

Caution

Although streaming tape can be used to autoboot, the same power supply must be connected to the streaming tape drive, controller, and the MVME197LE. 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 MVME197LE 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 ROMboot code is installed, a user-written routine is given control (if the routine meets the format requirements). One use of ROMboot might be resetting SYSFAIL* on an unintelligent controller module. The **NORB** command disables the function.

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

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 MVME197LE 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 software routine contained in the 197Bug that provides a mechanism for booting an operating system using a network (local Ethernet interface) as the boot device. The Network Auto Boot routine automatically scans for controllers and devices in a specified sequence until a valid bootable device containing a boot media is found or the list is exhausted. If a valid bootable device is found, a boot from that device is started. The controller scanning sequence goes from the lowest controller Logical Unit Number (LUN) detected to the highest LUN detected. (Refer to Appendix C for default LUNs).

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

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

Following this message there is 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 software ABORT or RESET switches.

Network Auto Boot is controlled by parameters contained in the **NIOT** and **ENV** commands. These parameters allow the selection of specific boot devices, systems, and files, and allow programming of the Boot delay. Refer to the **ENV** command in Appendix A (*Configure and Environment Commands*) and to the **NIOT** command description in the *MVME197BUG 197Bug Debugging Package User's Manual* for more details.

Restarting the System

The user 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 MVME197LE front panel reset switch initiates a system reset. A system reset also occurs if power is removed from the MVME197LE module and then reapplied. Reset is used to restore the system to a known state. The debugger environment (refer to Appendix A of this guide or 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 MVME197LE had just been powered up. All static variables (including disk device and controller parameters) are restored to their default states. The breakpoint table and offset registers are cleared. The target registers are invalidated. Input and output character queues are cleared. Onboard devices (timer, serial ports, etc.) are reset, and the *first* two serial ports are reconfigured to their default state.

During WARM reset, the 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.

Abort

Abort is invoked by pressing and releasing the ABORT switch on the MVME197LE 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 Instruction Pointer (IP), status register, and format/vector information). For this reason, abort is most appropriate when terminating a user program that is being debugged. Abort should be used to regain control if the program gets caught in a loop, etc. The target IP, register contents, etc., help to pinpoint the malfunction.

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

Break

A “Break” is generated by pressing and releasing the BREAK key on the terminal keyboard. Break does not generate an interrupt. The only time break is recognized is when characters are sent or received by the console port. Break removes any breakpoints in the user code and keeps the breakpoint table intact. Break also takes a snapshot of the machine state if the function was entered using SYSCALL. This machine state is then accessible to the user 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 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

❑ NVRAM low battery condition

❑ local memory configuration status

❑ self test (if system mode) has completed with error

❑ MPU clock speed calculation failure

After debugger initialization is done and 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 Appendix A (*Configure and Environment Commands*).

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 MVME197LE is mapped starting at location $FF800000.

197Bug requires a minimum of 64KB of contiguous read / write memory to operate.

The **ENV** command controls where this block of memory is located. Regardless of where the onboard RAM is located, the first 64KB is used for 197Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME197LE 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, and defined data structures for disk parameters.

Parameters such as the address where the module is mapped and the type and number of devices attached to the controller module are kept in tables by 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.

Appendix B (*Disk/Tape Controller Data*) contains a list of the controllers presently supported, as well as a list of the default configurations for each controller.

Blocks Versus Sectors

The logical block defines the unit of information for disk devices. A disk is viewed by 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, which is then passed on to the controller to initiate the transfer. If the conversion from blocks to sectors yields a fractional sector count, an error is returned and no data is transferred.

Device Probe Function

A device probe with entry into the device descriptor table is done whenever a specified device is accessed; i.e., when system calls .DSKRD, .DSKWR, .DSKCFIG, .DSKFMT, and .DSKCTRL, and debugger commands **BH**, **BO**, **IOC**, **IOP**, **IOT**, **MAR**, and **MAW** are used.

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

Disk I/O via 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 197Bug command **IOI** allows the user 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 197Bug command **IOP** allows the user to read or write blocks of data, or to format the specified device in a certain way. **IOP** creates a command packet from the arguments specified by the user, and then invokes the proper system call function to carry out the operation.

IOT (Input/Output Teach)

The 197Bug command **IOT** allows the user to change any configurable parameters and attributes of the device. In addition, it allows the user to see the controllers available in the system.

IOC (Input/Output Control)

The 197Bug command **IOC** 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)

The 197Bug command **BO** reads an operating system or control program from the specified device into memory, and then transfers control to it.

BH (Bootstrap and Halt)

The 197Bug command **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 are provided to allow user programs to do disk I/O:

.DSKRD	Disk read. System call to read blocks from a disk into memory.
.DSKWR	Disk write. System call to write blocks from memory onto a disk.
.DSKCFIG	Disk configure. This function allows 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 do disk I/O accept a generalized (controller-independent) packet format as an argument, and translate it into a controller-specific packet, which is then sent to the specified device. Refer to the system call descriptions found 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 (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 (refer to Appendix B). If the system needs to be configured differently than this default configuration (for example, to use a 70MB Winchester drive where the default is a 40MB Winchester drive), then these tables must be changed.

There are three ways to change the parameter tables:

- ❏ Using **BO** or **BH**. When the user invokes 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**. The user 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. The user 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 various modules (capabilities) and the dependencies of these modules that support the overall network boot function 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.

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.

Multiprocessor Support

The MVME197LE dual-port RAM feature makes the shared RAM available to remote processors as well as to the local processor. This can be done by either of the following two methods. Either method can be enabled / disabled by the **ENV** command as its Remote Start Switch Method (refer to Appendix A).

Multiprocessor Control Register (MPCR) Method

A remote processor can initiate program execution by the local MVME197LE 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 MVME197LE MPU)

– Commands set (by the remote processor)

The status codes that may be returned from the MVME197LE 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.

Even if no terminal is connected to the MVME197LE’s console port, the MPCR is still polled to see whether an external processor wishes to re-direct execution of the MVME197LE’s MPU, possibly to another program which has been loaded into RAM. If a terminal is connected, the MPCR is polled for the same purpose while the serial port is being polled for user input.

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

In either sequence, an E is placed in the MPCR to indicate that execution is underway just before control is passed to 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 redirect program execution by the MVME197LE MPU by issuing a remote **GO** command using the VMEchip2 Global Control and Status Register (GCSR). The remote processor places the MVME197LE 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 MVME197LE 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 peripheral bus) or $4 (VMEbus) from the start of the GCSR registers. The local peripheral 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 MVME197LE. 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 MVME197LE (refer to the *MVME197 Single Board Computer Diagnostic Firmware User's Manual*). In order to use the diagnostics, the user must switch directories to the diagnostic directory. If in the debugger directory, the user can switch to the diagnostic directory by entering the debugger command Switch Directories (**SD**). The diagnostic prompt ("**197-Diag>**") should appear. Refer to the *MVME197 Single Board Computer Diagnostic Firmware User's Manual* for complete descriptions of the diagnostic routines available and instructions on how to invoke them. Note that some diagnostics depend on restart defaults that are set up only in a particular restart mode. Refer to the documentation on a particular diagnostic for the correct mode.

Table 4-1. Debugger Commands (Continued)

Command Mnemonic	Command Title	Command Line Syntax
RL	Read Loop	**RL** *ADDR*;[**B** | **H** | **W**]
RM	Register Modify	**RM** [*REG*] [;[**S** | **D**]]
RS	Register Set	**RS** *REG* [*DEL EXP* | *DEL ADDR*][;[**S** | **D**]]
RUN	MPU Execution / Status	**RUN** [*MPU#*]
SD	Switch Directories	**SD**
SET	Set Time and Date	**SET** *mmddyyhhmm*
SYM	Symbol Table Attach	**SYM** [*ADDR*]
NOSYM	Symbol Table Detach	**NOSYM**
SYMS	Symbol Table Display / Search	**SYMS** [*symbol-name*] | [;**S**]
T	Trace	**T** [*COUNT*]
TA	Terminal Attach	**TA** [*port*]
TIME	Display Time and Date	**TIME** [;**C** | **L** | **O**]
TM	Transparent Mode	**TM** [*n*] [*ESCAPE*]
TT	Trace to Temporary Breakpoint	**TT** *ADDR*
VE	Verify S-records Against Memory	**VE** [*n*] [*ADDR*] [;**X** | **C**] [=*text*]
VER	Revision / Version Display	**VER** [;**E**]
WL	Write Loop	**WL** *ADDR*:*DATA*;[**B** | **H** | **W**]

CONFIGURE AND ENVIRONMENT COMMANDS

A

Configure Board Information Block

CNFG [;[**M**][**I**]]

This command is used to display and configure the board information block. This block is resident within the Non-Volatile RAM (NVRAM). Refer to the *MVME197LE Single Board Computer User's Manual* for the actual location. The board information block contains various elements detailing specific operation parameters of the hardware. The *MVME197LE Single Board Computer User's Manual* also describes the elements within the board information block, and lists the size and logical offset of each element. The **CNFG** command does *not* describe the elements and their use. The board information block contents are checksummed for validation purposes. This checksum is the last element of the block.

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

```
197-Bug>cnfg
Board (PWB) Serial Number = "0000000xxxxx"
Board Identifier          = "MVME197LE       "
Artwork (PWA) Identifier  = "01-W3869B01A    "
MPU Clock Speed           = "5000"
Ethernet Address          = 08003E21EG7A
Local SCSI Identifier     = "07"
Optional Board 1 Artwork (PWA) Identifier    = "0           "
Optional Board 1 (PWB) Serial Number         = "0           "
Optional Board 2 Artwork (PWA) Identifier    = "0           "
Optional Board 2 (PWB) Serial Number         = "0           "
197-Bug>
```

Note that the parameters that are quoted are left-justified character (ASCII) strings padded with space characters, and the quotes (") are displayed to indicate the size of the string. Parameters that are not quoted are considered data strings, and data strings are right-justified. The data strings are padded with zeros if the length is not met.

In the event of corruption of the board information block, the command displays a question mark "?" for nondisplayable characters. A warning message (`WARNING: Board Information Block Checksum Error`) is also displayed in the event of a checksum failure.

Using the **I** option initializes the unused area of the board information block to zero.

Modification is permitted by using the **M** option of the command. At the end of the modification session, you are prompted for the update to Non-Volatile RAM (NVRAM). A **Y** response must be made for the update to occur; any other response terminates the update (disregards all changes). The update also recalculates the checksum.

Take caution when modifying parameters. Some of these parameters are set up by the factory, and correct board operation relies upon these parameters.

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

Set Environment to Bug/Operating System

ENV [;[**D**]]

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

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

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

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

The parameters to be configured are listed in the following table.

Table A-1. ENV Command Parameters

ENV Parameter and Options	Default	Meaning of Default
Bug or System Environment [B/S]	B	Bug is the standard mode of operation.
Field Service Menu Enable [Y/N]	N	Do not display the field service menu.
Remote Start Method Switch [G/M/B/N]	B	Use both the Global Control and Status Register (GCSR) in the VMEchip2, and the Multiprocessor Control Register (MPCR) in the shared RAM methods to pass and start execution of cross-loaded program.
Probe System for Supported Disk/Tape Controllers [Y/N]	Y	Accesses will be made to the VMEbus to determine the presence of supported controllers.
Negate VMEbus SYSFAIL* Always [Y/N]	N	Negate VMEbus SYSFAIL after the successful completion or entrance into the bug command monitor.
Local SCSI Bus Reset on Debugger Setup [Y/N]	Y	The local SCSI bus is reset on the debugger setup.
Local SCSI Bus Negotiations Type [A/S/N]	A	Use Asynchronous negotiations on the local SCSI bus.
Ignore CFGA Block on a Hard Disk Boot [Y/N]	Y	Enable the ignorance of the Configuration Area (CFGA) Block (hard disk only).
Auto Boot Enable [Y/N]	N	The Auto Boot function is disabled.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Auto Boot at power-up only [Y/N]	Y	Auto Boot is attempted at power-up reset only.
Auto Boot Controller LUN	00	LUN of a disk/tape controller module currently supported by the Bug. The default is $0.
Auto Boot Device LUN	00	LUN of a disk/tape device currently supported by the Bug. The default is $0.
Auto Boot Abort Delay	15	This is the time in seconds that the Auto Boot sequence will delay before starting the boot. The purpose of the delay is to allow the user the option of stopping the boot by use of the Break key. The time value is from 0 through 255 seconds.
Auto Boot Default String [NULL for an empty string]	<none>	The user may specify a string (filename) which is passed on to the code being booted. The maximum length of the string is 16 characters. The default is the null string.
ROM Boot Enable [Y/N]	N	The ROMboot function is disabled.
ROM Boot at power-up only [Y/N]	Y	ROMboot is attempted at power-up only.
ROM Boot Enable Search of VMEbus [Y/N]	N	VMEbus address space will not be accessed by ROMboot.

Index

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