MVME135DIAG/D2

MVME135 Diagnostic Firmware User's Manual

MOTOROLA

MOTOROLA

MVME135DIAG/D2

JUNE 1988

MVME135 DIAGNOSTIC FIRMWARE USER'S MANUAL

Second Edition

First Edition April 1987

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
SCOPE

This diagnostic manual contains information about the operation and use of the MVME135 Diagnostic Firmware Package, hereafter referred to as the **diagnostics**. Chapters 3 and 4 give the user guidance in setting up the system and invoking the various utilities and tests. Chapter 5 describes utilities available to the user. Chapters 6 through 14 are guides to using each test. Chapter 15 provides the Final Assembly Tests (FAT) for the MVME135.

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 2
OVERVIEW OF DIAGNOSTIC FIRMWARE

The MVME135 diagnostic firmware package consists of two 64K x 8 EPROMs which are plugged into the MVME135 module. These two EPROMs (which may also contain 135Bug) contain a complete diagnostic monitor along with a battery of utilities and tests for exercise, test, and debug of hardware in the MVME135 environment.

The diagnostics are menu-driven for ease of use. The **HE** command (explained fully in Chapter 4) displays a menu of all available diagnostic functions (i.e., the tests and utilities). Each test has a subtest menu which may be called using the **HE** command. In addition, some utilities have subfunctions and as such have subfunction menus.

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 3
SYSTEM START-UP

Plug the MVME135 diagnostic ROMs into sockets U56 (even byte) and U54 (odd byte), **CAREFULLY OBSERVING THE PIN 1 ORIENTATION.**

Be certain that jumper block J7 has a jumper pin installed between pins 2 and 3 to accommodate 27512-type devices.

Plug in the MVME135 under test. Attach the cable from the RS-232 terminal to the "SER PORT 1" connector on the MVME135 front panel. Engage power to the chassis. The prompt should appear following the banner (refer to Chapter 4).

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 4
DIAGNOSTIC MONITOR

The tests described herein are called via a common diagnostic monitor, hereafter called **monitor**. This monitor is menu driven and provides input/output facilities, command parsing, error reporting, interrupt handling, and a multi-level directory.

4.1 MONITOR START-UP

When the monitor is first brought up, following power up or pushbutton reset, the following line is displayed on the diagnostic video display terminal:

```
MVME135 Debugger/Diagnostics    Version x.y - mm/dd/yy
<Confidence Test Fail Information>
<FPC Status>
<PMMU Status>
Local Memory Size is z Meg
135Bug>
```

Where x.y is the version number and revision level of the 135Bug package being used and mm/dd/yy is the date of issue of the package. < Confidence Test Fail Information> is only printed out if the power-up confidence test fails. A fail code, indicating the mode of failure is displayed (a list of fail codes and their meaning is given in an appendix of the 135Bug Manual). < FPC Status> is the results of a floating point coprocessor check done at each power-up or reset. < PMMU Status> is the results of a check for an installed PMMU, also done at power-up or reset. The local memory size, z, will be either 1Mb or 4Mb.

At the prompt, enter **SD** to switch to the diagnostics directory. Command **SD** is explained in detail in section 4.5. The prompt should now read **135Diag>** .

4.2 COMMAND ENTRY AND DIRECTORIES

Entry of commands is made when the prompt **135Diag>** appears. The name of the command is entered before pressing **RETURN**. Multiple commands may be entered. If a command expects parameters and another command is to follow it, separate the two with an exclamation mark "!". For instance, to invoke the commands **MT A** and **MT B**, the command line would read **MT A ! MT B**. Spaces are not required but are shown here for legibility.

Most 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 **BUS** and **MMU**. These commands are used to refer to a set of lower level commands.

To call up a particular bus test, one would enter (on the same line) **BUS A**. This command would cause the monitor to find the **BUS** subdirectory, and then to execute the command **A** from that subdirectory.

The diagram in Figure 4-1 is provided to illustrate the directory structure.

Single Level Commands	HE	Help
	LE	Loop on Error
Two Level Commands	CIO A	Z8Ø36 Counter/Timer
		A Post Reset Initiate
	MMU A	Memory Management Board
		A RP Register
Three Level Commands	MMU Z Ø	Memory Management Board
		Z 16-bit Bus
		Ø User - Prog Space
		1 Page - Desc Modify Bit

FIGURE 4-1. MONITOR DIRECTORY STRUCTURE

4.3 HELP - Command HE

On line documentation has been provided in the form of a **HELP** command (syntax: **HE**). This command will display a menu of the top level directory if no parameters are entered, or a menu of each subdirectory if the name of that subdirectory is entered. For example, to bring up a menu of all the bus tests, one would enter **HE BUS**. When a menu is too long to fit on the screen, it will pause until the operator presses **RETURN**.

4.4 SELF-TEST - Prefix ST

The monitor provides an automated test mechanism called self test. Entering **ST** before a command will cause the monitor to run the tests included in an internal self test directory. **ST** without any parameters will run most of the MVME135 diagnostics, including those that expect special test configurations (i.e., a second MVME135, a RS-232C loopback connector, etc). Clearly, this should not be done in most situations. To facilitate testing a MVME135 in combination with only a MVME2Ø4, two Final Assembly Tests (FATs) have been provided: FATPKGØ for a MVME135 and FATPKG1 for a MVME136. Refer to the Final Assembly Test chapter for details.

To run most of the SIO tests, one would enter **ST SIO**. Entering **ST MMU Z** would cause the monitor to run the three subtests (16-bit bus) for **MMU Z**.

Each test that is not included in the self test chain for that particular command is listed in the section pertaining to the command (i.e., refer to the Bus Test chapter for the **BUS** commands that are not in the self test chain).

4.5 SWITCH DIRECTORIES - Command SD

To leave the diagnostic directory (and disable the diagnostic tests), enter **SD**. At this point, only the commands for 135Bug will function. To return to the diagnostic directory, the command **SD** is entered again. When in the 135Bug directory, the prompt will read **135Bug>** . When in the diagnostic directory, the prompt will read **135Diag>** . The purpose of this feature is to allow the end user to access 135Bug without the diagnostics being visible.

4.6 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 will be incremented. If one 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. **DE** will display the results of a particular test if the name of that test follows **DE**. Only non-zero values are displayed.

4.7 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 will reset 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 BUS A** will clear the error counter associated with **BUS A**.

4.8 NON-VERBOSE MODE - Prefix NV

Some diagnostics will display a substantial amount of data if an error is displayed. To avoid the necessity of watching the scrolling display, a mode is provided that suppresses all messages except "PASSED" or "FAILED". This mode is called "nonverbose" and is invoked prior to calling a command by entering **NV**. **NV ST SIO** would cause the monitor to run the SIO self test, but only show the names of the subtests and the results (pass/fail).

4.9 LOOP CONTINUOUS MODE - Prefix LC

To repeat a test or series of tests endlessly, the prefix **LC** is entered. This loop will include everything on the command line. To break the loop, press **BREAK**. Certain tests disable the **BREAK** key interrupt, so pressing the **ABORT** or **RESET** buttons may become necessary.

4.10 LOOP ON ERROR MODE - Prefix LE

Occasionally, when a scope or logic analyzer is in use, it becomes desirable to repeat a test endlessly at the point where an error is detected. **LE** accomplishes that for most of the tests. To invoke **LE**, enter it before the test that is to run in "loop on error" mode.

4.11 DISPLAY PASS COUNT - Command DP

A count of the number of passes in loop continuous mode is kept by the monitor. This count is displayed with other information at the conclusion of each pass (refer to section 4.9). To display this information without using **LC**, enter **DP**.

4.12 ZERO PASS COUNT - Command ZP

Invoking this command resets the pass counter (described in section 4.11) to zero. This is frequently desirable before typing in a command that invokes the loop continuous mode. Entering this command on the same line as **LC** will result in the pass counter being reset every pass.

CHAPTER 5
UTILITIES

The monitor is supplemented by several utilities that are separate and distinct from the monitor itself and the diagnostics.

5.1 WRITE LOOP - Command WL.size

The "WL.size" command invokes a streamlined write of specified size to the specified memory location. This command is intended as a technician aid for debug once specific fault areas are identified. The write loop is very short in execution so that measuring devices such as oscilloscopes may be utilized in tracking failures.

The size of the command may be specified as B for byte, W for word, or L for longword.

The command requires two parameters: target address and data to written. The address and data are both hexadecimal values and must be preceded by a Ø if the first digit is other than Ø to 9, i.e., $FF would be entered as **OFF**. To write $ØØ out to address $FFFBØØ3Ø, one would enter "WL.B ØFFFBØØ3Ø ØØ". Omission of either or both parameters will cause prompting for the missing values.

5.2 READ LOOP - Command RL.size

The "RL.size" command invokes a streamlined read of specified size from the specified memory location. This command is intended as a technician aid for debug once specific fault areas are identified. The read loop is very short in execution so that measuring devices such as oscilloscopes may be utilized in tracking failures.

The size of the command may be specified as B for byte, W for word, or L for longword.

The command requires one parameter: target address. The address is a hexadecimal value. To read from address $FFFBØØ3Ø, one would enter "RL.B FFFBØØ3Ø". Omission of the parameter will cause prompting for the missing value.

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 6
VMEbus AND VSB TESTS

6.1 GENERAL DESCRIPTION

This section details the diagnostics provided to test the VMEbus and VSB interfaces. Table 6-1 lists the VMEbus and VSB tests.

The bus tests are divided into groups zero through five. These groupings correspond to the type of testing done. In type-Ø tests, the UUT (Unit Under Test) performs data transfers through various bus interfaces (local bus, VMEbus, VSB) to slaves. This mainly tests the buffers going to the bus. No other bus master is allowed to use the bus being tested while a test is running.

The type-1 tests verify the functioning of bus hardware used for purposes other than normal data transfers. This includes interrupt circuitry, bus-related control, and status registers, etc. For some of these tests an Aux (Auxiliary bus-master) is required. The other MVME135 board in the chassis is used for this purpose.

All of the type-2 tests require an Aux board. In these tests, the UUT performs data transfers to slaves over the various interfaces. While this is going on, the Aux attempts to access the same slave that the UUT is accessing by writing/reading to it over the same interface that the UUT is using. The activity of the Aux and the UUT are synchronized so that they are both accessing the same slave at the same time. This tests the timing chains and circuitry involved with back-to-back accesses with another master (i.e., can the Aux take the bus from the UUT, can the UUT take the bus back from the Aux again?).

In the type-3 tests, the UUT accesses slaves using the various interfaces. While the UUT is accessing a slave, the Aux attempts to access the same slave using a different interface.

The type-4 tests have the UUT doing accesses to a slave while at the same time the Aux does accesses to some other slave using the same interface that the UUT is using.

The type-5 tests, referred to as independence tests, consist of accesses to a slave over a particular interface during simultaneous accesses to a different slave over a different interface by the Aux.

Some tests require the simultaneous execution of two bus masters, the UUT and the Aux. In these situations, the UUT must force the Aux to execute code for the test. To do this, first the UUT downloads the code that the Aux is to execute from the UUT EPROM into the Aux's on-board DRAM. The UUT then writes a vector into the Aux's vector table which points at the·code which was downloaded. The UUT then sets up a "packet" (predefined data structure for passing information from UUT to Aux and vice-versa) and uses the MVMEØ5Ø to interrupt the Aux master with a VMEbus level 7 interrupt. The Aux vectors into the downloaded code and is instructed to read a field in the packet to get a command number. Using the command number, the Aux indexes into a jump table and jumps to the code for the particular test.

6.2 HARDWARE CONFIGURATION

The following hardware is required to perform these tests.

MVME135 CPU - Board being tested (referred to as the " UUT").
MVME135 CPU - Auxiliary bus master (" Aux") used in some tests.
MVME2Ø4 RAM Board - Addressed at $ØØAØØØØØ (VMEbus) and $ØØCØØØØØ (VSB).
MVMEØ5Ø System Module - Used to generate VMEbus interrupts.
Video Display Terminal.

See Chapter 15 (Final Assembly Tests) for a more thorough description of system setup requirements, including jumper placements, etc.

TABLE 6-1. VMEbus AND VSB TESTS

Monitor Command	Title	Section
BUS Ø.A	VME Write/Read	6.3
BUS Ø.B	VME Byte/Word/Long	6.4
BUS Ø.C	VME Misalignment	6.5
BUS Ø.D	VME Short I/O	6.6
BUS Ø.E	VSB Write/Read	6.7
BUS Ø.F	VSB Byte/Word/Long	6.8
BUS Ø.G	VSB Misalignment	6.9
BUS Ø.H	Local Dram base Ø Write/Read	6.1Ø
BUS Ø.I	Local Dram base $FFEØØØØØ Write/Read	6.11
BUS Ø.J	Local Dram VME Write/Read	6.12
BUS 1.A	SYSFAIL IRQ	6.13
BUS 1.B	LBTO BERR	6.14
BUS 1.C**	Local Dram Parity BERR	6.15
BUS 1.D	VME Timeout BERR	6.16
BUS 1.E	Local Dram VMSK*	6.17
BUS 1.F	VME Slave BERR	6.18

TABLE 6-1. VMEbus AND VSB TESTS (cont.)

Monitor Command	Title	Section
BUS 1.G	VME Interrupt Handler	6.19
BUS 1.H	VME IRQ Mask	6.2Ø
BUS 1.I	VME Broadcast IRQ	6.21
BUS 1.J	VSB Parity BERR	6.22
BUS 1.K	VSB Timeout BERR	6.23
BUS 1.L	VSB Write Error	6.24
BUS 1.M	VSB Interrupt Handler	6.25
BUS 1.N	VSB Interrupt Mask	6.26
BUS 2.A	VME vs VME Arbiter	6.27
BUS 2.B	VSB vs VSB Arbiter	6.28
BUS 2.C	VME vs VME TAS Arbiter	6.29
BUS 3.A	VME vs VSB Arbiter	6.3Ø
BUS 3.B	VME vs Local Arbiter (SHARED)	6.31
BUS 3.C	VSB vs Local Arbiter (DUAL-PORTED)	6.32
BUS 4.A	VME MPCSR/MPCSR	6.33
BUS 4.B	VME Local/Local	6.34
BUS 5.A	VSB/VME Independence	6.35
BUS 5.B	VME/VSB Independence	6.36
BUS 5.C	Local/VME Independence	6.37
BUS 5.D	Local/VSB Independence	6.38

** Not in self-test chain.

6.3 VME WRITE/READ BUS Ø.A

6.3.1 Description

This command is a write/read test to RAM via VMEbus. The test writes a pattern and then reads it back, comparing the data read to the expected value.

6.3.2 Command Input

```
135Diag> BUS Ø.A
```

6.3.3 Response/Messages

After entering this command, the following should be printed out:

```
Ø.A     VME Write/Read .......................Running ----------->
```

If an error occurs, the address of the errant data will be displayed along with the expected and read values.

```
Ø.A     VME Write/Read .......................Running ----------->
      Addr=ØØAØ1234      Expect=ØØØØ5678      Read=FFFFFFFF
```

(actual values depend on the error, each is 8 hex digits)

```
 .... FAILED
```

If no errors are encountered, then the word "PASSED" will be printed out.

```
Ø.A     VME Write/Read .......................Running -----------> PASSED
```

6.4 VME BYTE/WORD/LONG BUS Ø.B

6.4.1 Description

This command is a test that writes a pattern of bytes to VMEbus memory and then reads back the data in both words and longwords to verify the byte/word/long addressing.

6.4.2 Command Input

```
135Diag> BUS Ø.B
```

6.4.3 Response/Messages

After entering this command, the following should be printed out:

```
Ø.B     VME Byte/Word/Long ...................Running ----------->
```

If an error occurs, the address of the errant data will be displayed along with the expected and read values.

```
Ø.B     VME Byte/Word/Long ...................Running ----------->
      Addr=ØØAØ1234      Expect=ØØØØ4567      Read=FFFFFFFF
```

(actual values depend on the error, each is 8 hex digits)

```
 .... FAILED
```

If no errors are encountered, then the word "PASSED" will be printed out.

```
Ø.B     VME Byte/Word/Long ...................Running -----------> PASSED
```

6.5 VME MISALIGNMENT BUS Ø.C

6.5.1 Description

This command tests the ability of the VMEbus interface to write/read longwords to/from odd addresses. The test also writes bytes and word out to odd addresses.

6.5.2 Command Input

```
135Diag> BUS Ø.C
```

6.5.3 Response/Messages

After entering this command, the following should be printed out:

```
Ø.C     VME Misalignment ....................Running ----------->
```

If an error occurs, the address of the errant data will be displayed along with the expected and read values.

```
Ø.C     VME Misalignment ....................Running ----------->
      Addr=ØØAØØØØ4      Expect=ØØØØ4567      Read=FFFFFFFF
```

(actual values depend on the error, each is 8 hex digits)

```
 .... FAILED
```

If no errors are encountered, then the word "PASSED" will be printed out.

```
Ø.C     VME Misalignment ....................Running -----------> PASSED
```

6.6 VMEbus SHORT I/O BUS Ø.D

6.6.1 Description

This command tests the VMEbus short I/O addressing space access by testing a bit in the MVME2Ø4 control/status register. The location of the register is within the short I/O addressing space. This test preserves the contents of the control/status register and restores it before returning control to the diagnostic monitor.

6.6.2 Command Input

```
135Diag> BUS Ø.D
```

6.6.3 Response/Messages

After entering this command, the following should be displayed:

```
Ø.D     VME Short I/O ........................Running ----------->
```

If the bit in the MVME2Ø4 control/status register cannot be set, then the error is reported in the following manner:

```
Ø.D     VME Short I/O ........................Running ----------->
        Addr=FFFFBEØ1       Expect=ØØØØØØØ4       Read=ØØØØØØØØ
 .... FAILED
```

If the bit does get set by the test, then the test passes.

```
Ø.D     VME Short I/O ........................Running -----------> PASSED
```

6.7 VSB WRITE/READ — BUS Ø.E

6.7.1 Description

This command is a write/read test to RAM via the VSB bus. The test writes a pattern and then reads it back, comparing the data read to the expected value.

6.7.2 Command Input

```
135Diag> BUS Ø.E
```

6.7.3 Response/Messages

After entering this command, the following should be printed out:

```
Ø.E     VSB Write/Read .......................Running ----------->
```

If an error occurs, the address of the errant data will be displayed along with the expected and read values.

```
Ø.E     VSB Write/Read .......................Running ----------->
      Addr=ØØCØ1234      Expect=ØØØØ5678      Read=FFFFFFFF
```

(actual values depend on the error, each is 8 hex digits)

```
 .... FAILED
```

If no errors are encountered, then the word "PASSED" will be printed out.

```
Ø.E     VSB Write/Read .......................Running -----------> PASSED
```

6.8 VSB BYTE/WORD/LONG BUS Ø.F

6.8.1 Description

This command is a test that writes a pattern of bytes to the VSB bus memory and then reads back the data in both words and longwords to verify the byte/word/long addressing.

6.8.2 Command Input

```
135Diag> BUS Ø.F
```

6.8.3 Response/Messages

After entering this command, the following should be printed out:

```
Ø.F     VSB Byte/Word/Long ...................Running ------------>
```

If an error occurs, the address of the errant data will be displayed along with the expected and read values.

```
Ø.F     VSB Byte/Word/Long ...................Running ------------>
        Addr=ØØCØ1234      Expect=ØØØØ4567      Read=FFFFFFFF
```

(actual values depend on the error, each is 8 hex digits)

```
 .... FAILED
```

If no errors are encountered, then the word "PASSED" will be printed out.

```
Ø.F     VSB Byte/Word/Long ...................Running ------------> PASSED
```

6.9 VSB MISALIGNMENT **BUS Ø.G**

6.9.1 Description

This command tests the ability of the VSB bus interface to write/read longwords to/from odd addresses. This test also writes bytes and words out to odd addresses.

6.9.2 Command Input

```
135Diag> BUS Ø.G
```

6.9.3 Response/Messages

After entering this command, the following should be printed out:

```
Ø.G     VSB Misalignment .....................Running ----------->
```

If an error occurs, the address of the errant data will be displayed along with the expected and read values.

```
Ø.G     VSB Misalignment .....................Running ----------->
      Addr=ØØCØ1234       Expect=ØØØØ4567      Read=FFFFFFFF
```

(actual values depend on the error, each is 8 hex digits)

```
 .... FAILED
```

If no errors are encountered, then the word "PASSED" will be printed out.

```
Ø.G     VSB Misalignment .....................Running -----------> PASSED
```

6.10 LOCAL DRAM LOW ADDRESS WRITE/READ — BUS 0.H

6.10.1 Description

The local DRAM should appear at $0 on the local bus regardless of the VMEbus address specified by the mapping switches if the OPT0 bit in control register CNT5 is set.

This command is a write/read test to local DRAM at $0. The test writes a pattern and then reads it back, comparing the data read to the expected value.

6.10.2 Command Input

```
135Diag> BUS 0.H
```

6.10.3 Response/Messages

After entering this command, the following should be printed out:

```
0.H     Local DRAM low address Write/Read.....Running ----------->
```

If an error occurs, the address of the errant data will be displayed along with the expected and read values.

```
0.H     Local DRAM low address Write/Read.....Running ----------->
      Addr=00001234      Expect=00005678      Read=FFFFFFFF
```

(actual values depend on the error, each is 8 hex digits)

```
 .... FAILED
```

If no errors are encountered, then the word "PASSED" will be printed out.

```
0.H     Local DRAM low address Write/Read.....Running -----------> PASSED
```

6.11 LOCAL DRAM HIGH ADDRESS WRITE/READ BUS Ø.I

6.11.1 Description

The local DRAM should appear at $FFEØØØØØ on the local bus regardless of the VMEbus address specified by the mapping switches or the state of the OPTØ bit in control register CNT5.

This command is a write/read test to local DRAM at $FFEØØØØØ via the local bus. The test writes a pattern and then reads it back, comparing the data read to the expected value.

6.11.2 Command Input

```
135Diag> BUS Ø.I
```

6.11.3 Response/Messages

After entering this command, the following should be printed out:

```
Ø.I     Local DRAM High address Write/Read....Running ----------->
```

If an error occurs, the address of the errant data will be displayed along with the expected and read values.

```
Ø.I     Local DRAM High address Write/Read....Running ----------->
      Addr=FFEØ1234       Expect=ØØØØ5678       Read=FFFFFFFF
(actual values depend on the error, each is 8 hex digits)
 .... FAILED
```

If no errors are encountered, then the word "PASSED" will be printed out.

```
Ø.I     Local DRAM High address Write/Read....Running -----------> PASSED
```

6.12 LOCAL DRAM VME WRITE/READ BUS Ø.J

6.12.1 Description

The local DRAM should appear on the VMEbus at the address specified by the mapping switches. This command is a write/read test to local DRAM at that address. The OPTØ bit in CNT5 is cleared to assure that the local DRAM is not accessed via the local bus. The test writes a pattern and then reads it back, comparing the data read to the expected value.

6.12.2 Command Input

```
135Diag> BUS Ø.J
```

6.12.3 Response/Messages

After entering this command, the following should be printed out:

```
Ø.J     Local DRAM VME Write/Read.............Running ------------>
```

If an error occurs, the address of the errant data will be displayed along with the expected and read values.

```
Ø.J     Local DRAM VME Write/Read.............Running ------------>
     Addr=ØØØØ1234       Expect=ØØØØ5678       Read=FFFFFFFF
(actual values depend on the error, each is 8 hex digits)
 .... FAILED
```

If no errors are encountered, then the word "PASSED" will be printed out.

```
Ø.J     Local DRAM VME Write/Read.............Running ------------> PASSED
```

6.13 SYSFAIL IRQ — BUS 1.A

6.13.1 Description

This command tests the SYSFAIL interrupt request logic. The intent of the test is to verify that the MVME135 is able to generate the SYSFAIL signal as well as recognize it as an interrupt source when the signal is generated elsewhere.

The implementation of the test is as follows:

Step 1: Enable SYSFAIL IRQ.

Step 2: Assert SYSFAIL by clearing BDFAIL bit in CNT5 control register.

Step 3: After a short time delay, check local copy of STAT2. If the interrupt occurred, then a bit in the local copy will have been cleared in the interrupt handler. The interrupt handler negates SYSFAIL.

Step 4: If the interrupt did not occur or the SYSFAIL bit in STAT2 is still clear, then the test fails and an error message is printed.

6.13.2 Command Input

```
135Diag> BUS 1.A
```

6.13.3 Response/Messages

After entering this command, the following should be displayed:

```
1.A   SYSFAIL IRQ ............................Running ----------->
```

If the interrupt never occurs or the SYSFAIL bit does not get set by the interrupt handler, then the following error message will be printed out:

```
1.A   SYSFAIL IRQ ............................Running ----------->
      Addr=FFFBØØ32      Expect=ØØØØØØCF      Read=ØØØØØØDF
 .... FAILED
```

NOTE: If the "Expect" and "Read" values are identical, then the SYSFAIL bit did not get set by the interrupt handler.

If the test passes, then the display will appear as follows:

```
1.A   SYSFAIL IRQ ...........................Running ------------> PASSED
```

6.14 LBTO BERR — BUS 1.B

6.14.1 Description

This command tests the Local Bus Time-Out error circuitry and checks the status bit LCLERR* in the MVME135 register STAT2. The test has been implemented in the following manner:

Step 1: Attempt to read from a non-existent location that will be decoded as being in the local resource map.

Step 2: If the local bus time-out bus error does not occur or status bit LCLERR* does not get cleared, then report the error.

6.14.2 Command Input

```
135Diag> BUS 1.B
```

6.14.3 Response/Messages

After entering this command, the following should be displayed:

```
1.B   LBTO BERR .............................Running ----------->
```

If the bus error does not occur or the status bit LBTO does not get cleared, then the error will be reported as shown:

```
1.B   LBTO BERR .............................Running ----------->
      Addr=FFFBØØ32     Expect=ØØØØØØDE     Read=ØØØØØØDF
 .... FAILED
```

If the bus error occurs as planned, and the LBTO bit does get cleared, then the test passes.

```
1.B   LBTO BERR .............................Running -----------> PASSED
```

6.15 LOCAL DRAM PARITY **BUS 1.C**

6.15.1 Description

This command tests the parity generation and checking circuitry associated with the local dynamic RAM (including the parity error BERR generation circuitry).

NOTE: Parity generation/checking for the local DRAM is an option on the MVME135. In order to take advantage of this option, the local DRAM must be configured to operate with one wait state. This can be done by installing a jumper pin between pins 2 and 3 of jumper block J6. **THE TEST WILL NOT RUN PROPERLY UNLESS THIS JUMPER IS INSTALLED.**

This test does not appear in the standard self-test sequence (reference the **ST** command) because of the necessity to make a jumper change before running.

The test has been implemented in the following manner:

Step 1: Assert OPTØ so that the on-board DRAM is mapped to $Ø on the local bus.

Step 2: For each byte of the targeted long-word, do the following...

- Enable parity error reporting by local DRAM.
- Enable write-wrong-parity (WWP) bit in CNT1.
- Do byte write to target address (with wrong parity).
- Disable WWP bit.
- Do byte read from target address.
- Check for BERR .
- Check for STAT2's RAMERR* bit asserted.

Step 3: If the parity fault bus error does not occur or status bit RAMERR* does not get cleared, then report the error.

Step 4: Negate OPTØ so that local DRAM is accessed via VMEbus and repeat steps 2 and 3.

6.15.2 Command Input

```
135Diag> BUS 1.C
```

6.15.3 Response/Messages

After entering this command, the following should be displayed:

```
1.C   Local DRAM Parity ......................Running ----------->
```

If the bus error does not occur or the status bit LBTO does not get cleared, then the error will be reported as shown:

```
1.C   Local DRAM Parity ......................Running ----------->
      Addr=FFFBØØ32       Expect=ØØØØØØDE       Read=ØØØØØØDF
 .... FAILED
```

If the bus error occurs as planned, and the LBTO bit does get cleared, then the test passes.

```
1.C   Local DRAM Parity ......................Running -----------> PASSED
```

6.16 VME TIMEOUT BERR BUS 1.D

6.16.1 Description

This command tests the ability of the MVME135 to terminate a VMEbus cycle if no slave responds by means of the VMEbus timeout generator (part of the MVME135's System Controller circuitry). This test also tests the VBER* bit in the MVME135's STAT2 register.

The test has been implemented in the following manner:

Step 1: Attempt to read from a location that will be decoded as being mapped to the VMEbus but where it is known that no VMEbus slave will respond.

Step 2: If the VMEbus timeout bus error does not occur or status bit VBER* does not get cleared, then report the error.

6.16.2 Command Input

```
135Diag> BUS 1.D
```

6.16.3 Response/Messages

After entering this command, the following should be displayed:

```
1.D   VME Timeout BERR .......................Running ----------->
```

If the bus error does not occur or the status bit VBER* does not get cleared, then the error will be reported as shown:

```
1.D   VME Timeout BERR .......................Running ----------->
      Addr=FFFBØØ32      Expect=ØØØØØØD7      Read=ØØØØØØDF
 .... FAILED
```

If the bus error occurs as planned, and the VBER* bit does get cleared, then the test passes.

```
1.D   VME Timeout BERR .......................Running -----------> PASSED
```

6.17 LOCAL DRAM VMSK* BUS 1.E

6.17.1 Description

This command tests the functioning of the VMSK* bit in CNT2. When this bit is asserted the local DRAM should be inaccessible from the VMEbus. This test is in two parts. In the first part of the test the local CPU attempts to access the local DRAM via the VMEbus. In the second part of test, the other MVME135 attempts to access the local DRAM of the MVME135 under test. This test requires that the Aux MVME135 be present in the chassis.

The test has been implemented in the following manner:

Step 1: Assert VMSK* bit in CNT2.

Step 2: Local Access

- Enable parity error reporting by local DRAM.
- Negate OPTØ bit in CNT5 (no local image of DRAM at $Ø).
- Access local DRAM via VMEbus.
- VMEbus Timeout circuit should issue BERR.
- VBER* bit in STAT2 should be asserted.

Step 3: Remote Access

- Cause Aux to attempt access of UUT's local DRAM.
- Aux should receive VMEbus timeout Bus Error.

6.17.2 Command Input

```
135Diag> BUS 1.E
```

6.17.3 Response/Messages

After entering this command, the following should be displayed:

```
1.E   Local DRAM VMSK* ......................Running ----------->
```

If the bus error does not occur or the status bit VBER* does not get cleared, then the error will be reported as shown:

```
1.E   Local DRAM VMSK* ......................Running ----------->
      Addr=FFFBØØ32      Expect=ØØØØØØD7      Read=ØØØØØØDF
 .... FAILED
```

If the bus error occurs as planned, and the VBER* bit does get cleared, then the test passes.

```
1.E   Local DRAM VMSK* .......................Running -----------> PASSED
```

6.18 VME SLAVE BERR — BUS 1.F

6.18.1 Description

This command tests the ability of the MVME135 to recover from a bus error returned by an accessed VMEbus slave. This situation is created in this test by setting up the MVME2Ø4 memory card to signal a BERR if a parity fault is detected and then causing an intentional parity fault. This command also tests the VBER* bit in the MVME135's STAT2 register.

The test has been implemented in the following manner:

Step 1: Enable Write-Wrong-Parity (WWP) bit in MVME2Ø4 control/status register.

Step 2: Write to the MVME2Ø4 via VMEbus.

Step 3: Disable Write-Wrong-Parity (WWP) bit.

Step 4: Read from the same location (also via VMEbus). The resulting parity error should cause a VMEbus bus error.

Step 5: If a VMEbus bus error does not occur, or the VBER* bit in STAT2 does not get cleared, then report the error.

6.18.2 Command Input

```
135Diag> BUS 1.F
```

6.18.3 Response/Messages

After entering this command, the following should be displayed:

```
1.F   VME Slave BERR ........................Running ----------->
```

If the bus error does not occur or the status bit VBER* does not get cleared, then the error will be reported as shown:

```
1.F   VME Slave BERR ........................Running ----------->
      Addr=FFFBØØ32     Expect=ØØØØØØD7     Read=ØØØØØØDF
 .... FAILED
```

If the bus error occurs as planned, and the VBER* bit does get cleared, then the test passes.

```
1.F   VME Slave BERR ........................Running -----------> PASSED
```

6.19 VME IRQ HANDLER — BUS 1.G

6.19.1 Description

This command is a test of the MVME135's ability to respond to interrupts of different levels from VMEbus. This test requires that the Aux MVME135 be present in the chassis.

The following steps are performed for levels one through seven.

Step 1: Issue VMEbus interrupt by writing to the control register of the MVMEØ5Ø.

Step 2: Wait in a timeout loop for the interrupt to occur. If the interrupt does occur within the time interval, then proceed to the next interrupt level.

Step 3: If the interrupt did not occur before the timeout expired, then the test fails and an error message is displayed.

6.19.2 Command Input

```
135Diag> BUS 1.G
```

6.19.3 Response/Messages

After entering this command, the following should be displayed:

```
1.G   VME IRQ Handler ........................Running ----------->
```

If an interrupt does not occur in the time allowed, then the test fails.

```
1.G   VME IRQ Handler ........................Running ----------->
      Tested Level=Ø4   Vector=FF
 .... FAILED
```

If all seven interrupts occur during their respective time intervals, then the test passes.

```
1.G   VME IRQ Handler ........................Running -----------> PASSED
```

6.2Ø VME IRQ MASK BUS 1.H

6.2Ø.1 Description

This command tests the VMEbus interrupt request mask mechanism. Each level of interrupt has a corresponding mask bit in register CNT2. The test first disables all of the interrupts, then proceeds to issue all seven levels (from 7 down to 1) of interrupts. None of the interrupt requests should be permitted. If any of the requests generate interrupts, then the test fails. This test does not use the auxiliary MVME135.

6.2Ø.2 Command Input

```
135Diag> BUS 1.H
```

6.2Ø.3 Response/Messages

After entering this command, the following should be displayed:

```
1.H   VME IRQ Mask ..........................Running ----------->
```

If an interrupt occurs, then the mask is not functioning and the test fails.

```
1.H   VME IRQ Mask ..........................Running ----------->
      Tested Level=Ø4   Vector=FF
 .... FAILED
```

If all seven interrupts are masked, that is they do not interrupt the MVME135, then the test passes.

```
1.H   VME IRQ Mask ..........................Running -----------> PASSED
```

6.21 VME BROADCAST IRQ BUS 1.I

6.21.1 Description

This command verifies the broadcast interrupt feature built into the MVME135 which uses the VMEbus level 1 interrupt line to generate a timer interrupt. The MVME135 should be able to generate a broadcast interrupt to all bus masters as well as be able to respond correctly to a broadcast interrupt.

This test requires that the Aux MVME135 be present in the chassis.

The test has been implemented in the following manner:

Step 1: Mask VMEbus level 1 interrupts in control register CNT2. Set up CIO to provide a local interrupt on a transition of the broadcast interrupt line.

Step 2: Use the MVMEØ5Ø to interrupt the Aux MVME135 and tell it to start looking for the broadcast interrupt.

Step 3: Write to the BRIRQ* bit in control register CNT3, then sit in a loop waiting for the interrupt.

Step 4: If the broadcast interrupt does not occur, or the MPIRQ* bit in CNT1 will not return to its negated state after the interrupt has been serviced, then report the error.

Step 5: Check the Aux MVME135. If it did not get the broadcast interrupt, then report the error.

6.21.2 Command Input

```
135Diag> BUS 1.I
```

6.21.3 Response/Messages

After entering this command, the following should be displayed:

```
1.I   VME Broadcast IRQ ......................Running ----------->
```

If the broadcast interrupt does not occur , then the error will be reported as shown:

```
1.I   VME Broadcast IRQ ......................Running ----------->
      Addr=FFFBØØØE     Expect=ØØØØØØ6F     Read=ØØØØØØEF
 .... FAILED
```

If the status bit MPIRQ* will not return to its inactive state after the interrupt has been serviced, then the error will be reported as shown:

```
1.I   VME Broadcast IRQ ......................Running ----------->
      Addr=FFFBØØØE      Expect=ØØØØØØ6F      Read=ØØØØØØ6F
 .... FAILED
```

If the CIO device gives an interrupt before the broadcast interrupt is issued, then the error will be reported as shown:

```
1.I   VME Broadcast IRQ ......................Running ----------->
     Spurious CIO Int
 .... FAILED
```

If the Aux MVME135 reports that it never received the broadcast interrupt, then the error will be reported as shown:

```
1.I   VME Broadcast IRQ ......................Running ----------->
     Aux Not Interrupted
 .... FAILED
```

If the broadcast interrupt occurs as planned, and the MPIRQ* bit does return to its inactive state, then the test passes.

```
1.I   VME Broadcast IRQ ......................Running -----------> PASSED
```

6.22 VSB PARITY BERR BUS 1.J

6.22.1 Description

This command tests the ability of the MVME135 to respond to a bus error resulting from an access to a VSB slave. This situation is created in this test by setting up the MVME2Ø4 memory card to signal a BERR if a parity fault is detected and then causing an intentional parity fault during a VSB access to the MVME2Ø4. This command also tests the VSBERR* bit in the MVME135's STAT2 register and several bits in the VSB gate array control/status register.

The test has been implemented in the following manner:

Step 1: Enable Write-Wrong-Parity (WWP) bit in MVME2Ø4 control/status register.

Step 2: Write to the MVME2Ø4 (with wrong parity).

Step 3: Disable Write-Wrong-Parity (WWP) bit.

Step 4: Read from the same location via the VSB. The resulting parity error should cause a VSB bus error. The bus error's occurrence should be reflected in the STAT2 register of the MVME135 and in the CSR of the VSB gate array. The ASACKØ* and ASACK1* bits in the VSB gate array CSR should both be clear, indicating the bus error occurred on a longword access over the VSB.

Step 5: If a VSB bus error does not occur, or if either of the two status bits does not return to its inactive state after the bus error has been serviced, then report the error.

6.22.2 Command Input

```
135Diag> BUS 1.J
```

6.22.3 Response/Messages

After entering this command, the following should be displayed:

```
1.J   VSB Parity BERR .......................Running ------------>
```

If the bus error does not occur then the error will be reported as shown:

```
1.J    VSB Parity BERR ........................Running ------------>
       Addr=FFFB0032       Expect=000000DB       Read=000000DF
       Addr=FFFA0000       Expect=00001800       Read=0000F800
 .... FAILED
```

If the bus error occurs, but a status bit refuses to return to its inactive state after the bus error has been serviced, then the error report shown above will occur with the "Expect" value identical to the "Read" value.

If the bus error occurs as planned, and the status bits change state correctly, then the test passes.

```
1.J    VSB Parity BERR ........................Running ------------> PASSED
```

6.23 VSB TIMEOUT BERR BUS 1.K

6.23.1 Description

This command tests the ability of the VSB gate array to terminate a VSB bus cycle if no slave responds by means of a VSB timeout BERR. This command also checks for proper error reporting in the VSB gate array's control/status register and in the MVME135's STAT2 register.

The test has been implemented in the following manner:

Step 1: Access invalid VSB memory (past end of MVME2Ø4).

Step 2: VSB timeout should occur, causing VSB bus error.

Step 3: VSB CSR's VSBTMO bit should be asserted, with both ASACK* lines high (to indicate no slave responded). Also STAT2's VSBERR* bit should be asserted.

6.23.2 Command Input

```
135Diag> BUS 1.K
```

6.23.3 Response/Messages

After entering this command, the following should be displayed:

```
1.K   VSB Timeout BERR .......................Running ----------->
```

If the bus error does not occur then the error will be reported as shown:

```
1.K   VSB Timeout BERR .......................Running ----------->
      Addr=FFFBØØ32      Expect=ØØØØØØDB      Read=ØØØØØØDF
      Addr=FFFAØØØØ      Expect=ØØØØ78ØØ      Read=ØØØØF8ØØ
 .... FAILED
```

If the bus error occurs, but a status bit refuses to return to its inactive state after the bus error has been serviced, then the error report shown above will occur with the "Expect" value identical to the "Read" value.

If the bus error occurs as planned, and the status bits change state correctly, then the test passes.

```
1.K   VSB Timeout BERR .......................Running -----------> PASSED
```

6.24 VSB WRITE ERROR BUS 1.L

6.24.1 Description

This command tests the read-only mode of the VSB gate array by writing to a read-only bus and checking for error report.

The test has been implemented in the following manner:

Step 1: Set up VSB for read only.

Step 2: Write to MVME2Ø4 memory via VSB.

Step 3: Cycle should redirect to VME automatically.

Step 4: BERR should occur (no VMEbus memory mapped at VSB address). BERR handler copies VSB CSR contents and MVME135's STAT2 register.

Step 5: Check contents of registers copied in Step 4 above. WRERR* in VSB CSR should be asserted, but VSBERR* should not (no VSB bus error should have occurred, only the redirection). STAT2 should show VME bus error at address of VSB write, but no VSB bus error.

6.24.2 Command Input

```
135Diag> BUS 1.L
```

6.24.3 Response/Messages

After entering this command, the following should be displayed:

```
1.L    VSB Write Error .......................Running ----------->
```

If a VSB bus error occurs instead of a VMEbus error, then the error will be reported as shown:

```
1.L    VSB Write Error .......................Running ----------->
       Addr=FFFBØØ32      Expect=ØØØØØØD7      Read=ØØØØØØDB
       Addr=FFFAØØØØ      Expect=ØØØØFØØØ      Read=ØØØØ78ØØ
  .... FAILED
```

If the VMEbus BERR occurs, but a status bit refuses to return to its inactive state after the bus error has been serviced, then the error report shown above will occur with the "Expect" value identical to the "Read" value.

If the bus error occurs as planned, and the status bits change state correctly, then the test passes.

```
1.L   VSB Write Error ........................Running ------------> PASSED
```

6.25 VSB IRQ HANDLER BUS 1.M

6.25.1 Description

This test verifies that the MVME135 will be able to receive and respond to VSB interrupts issued by another master on the VSB.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Interrupt Aux MVME135 to start test. Aux masks VSB interrupts to itself and then writes the VSBIRQ* bit in CNT3 to put a VSB IRQ onto the VSB.

Step 2: Wait a short length of time. If the UUT does not receive the interrupt, then the test fails and an error message will be printed out.

Step 3: UUT signals Aux to remove interrupt.

6.25.2 Command Input

```
135Diag> BUS 1.M
```

6.25.3 Response/Messages

After entering this command, the following should be displayed:

```
1.M   VSB IRQ Handler ........................Running ----------->
```

If the interrupt does not occur, the following error message will be displayed:

```
1.M   VSB IRQ Handler ........................Running ----------->
      Tested Level=Ø4   Vector=47
 .... FAILED
```

If the interrupt occurs once, then the test passes.

```
1.M   VSB IRQ Handler ........................Running -----------> PASSED
```

6.26 VSB IRQ MASK **BUS 1.N**

6.26.1 Description

This test verifies that the MVME135 will be able to mask out VSB interrupts issued by another master on the VSB by clearing the VSBIEN bit in control register CNT1.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Mask VSB interrupts using the VSBIEN bit.

Step 2: Interrupt Aux MVME135 to start test. Aux masks VSB interrupts to itself and then writes the VSBIRQ* bit in CNT3 to put a VSB IRQ onto the VSB.

Step 3: Wait a short length of time. If the interrupt occurs, then the test fails and an error message will be printed out.

Step 4: Signal Aux to remove interrupt.

6.26.2 Command Input

```
135Diag> BUS 1.N
```

6.26.3 Response/Messages

After entering this command, the following should be displayed:

```
1.N   VSB IRQ Mask ..........................Running ----------->
```

If the interrupt occurs, the following error message will be displayed:

```
1.N   VSB IRQ Mask ..........................Running ----------->
      Tested Level=Ø4   Vector=47
 .... FAILED
```

If the timeout expires and no interrupt occurs, then the test passes.

```
1.N   VSB IRQ Mask ..........................Running -----------> PASSED
```

6.27 VMEbus VS VMEbus ARBITER BUS 2.A

6.27.1 Description

This command is a test that verifies the ability of the MVME135 bus arbiter to handle the situation of two bus masters attempting to access VMEbus simultaneously.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize the two bus masters via semaphore.

Step 2: The two bus masters will each write a known pattern to memory via VMEbus.

Step 3: The target memory will be examined and compared against the expected patterns.

6.27.2 Command Input

```
135Diag> BUS 2.A
```

6.27.3 Response/Messages

After entering this command, the following should be displayed:

```
2.A    VME vs VME Arbiter ....................Running ----------->
```

If the memory contains anything other than the expected values, then an error message will be printed out:

```
2.A    VME vs VME Arbiter ....................Running ----------->
       Addr=00A00004       Expect=00004567      Read=FFFFFFFF
 .... FAILED
```

If the memory contains the expected values, that is the patterns that both bus masters attempted to write, then the test completes successfully.

```
2.A    VME vs VME Arbiter ....................Running -----------> PASSED
```

6.28 VSB VS VSB ARBITER BUS 2.B

6.28.1 Description

This command is a test that verifies the ability of the MVME135 bus arbiter to handle the situation of two bus masters attempting to access VSB simultaneously.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize the two bus masters via semaphore.

Step 2: The two bus masters will each write a known pattern to memory via VSB.

Step 3: The target memory will be examined and compared against the expected patterns.

6.28.2 Command Input

```
135Diag> BUS 2.B
```

6.28.3 Response/Messages

After entering this command, the following should be displayed:

```
2.B   VSB vs VSB Arbiter ....................Running ----------->
```

If the memory contains anything other than the expected values, then an error message will be printed out:

```
2.B   VSB vs VSB Arbiter ....................Running ----------->
      Addr=00C00004      Expect=00004567      Read=FFFFFFFF
 .... FAILED
```

If the memory contains the expected values, that is the patterns that both bus masters attempted to write, then the test completes successfully.

```
2.B   VSB vs VSB Arbiter ....................Running -----------> PASSED
```

6.29 VME VS VME TAS ARBITER BUS 2.C

6.29.1 Description

This command is a test that verifies the ability of the MVME135 bus arbiter to handle the situation of two bus masters attempting to access VMEbus simultaneously using read-modify-write cycles. The MC68020 Test-And-Set instruction requires that the bus be accessed using a read-modify-write cycle.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize the two bus masters via semaphore.

Step 2: MVME135 under test writes a known pattern (with bit 7 of each byte clear) to the target memory via VMEbus.

Step 3: The two bus masters simultaneously access the memory using the TAS instruction. If TAS returns wrong condition code indicating the location was already "set", then the test fails.

Step 4: Examine memory. Pattern written there should be altered by TAS to have bit 7 of each location set. Fail if unexpected value read.

6.29.2 Command Input

```
135Diag> BUS 2.C
```

6.29.3 Response/Messages

After entering this command, the following should be displayed:

```
2.C   VME vs VME TAS Arbiter .................Running ----------->
```

If a location accessed by the TAS instruction was already set, or a location read during the pattern checking afterward contains an unexpected value, then test fails and the display will appear as follows:

```
2.C   VME vs VME TAS Arbiter .................Running ----------->
      Addr=00C00000       Expect=00000080      Read=FFFFFFFF
 .... FAILED
```

If the read-modify-cycles are arbitrated correctly and the bus masters access the memory correctly, then the test will pass and the display will appear as follows:

```
2.C   VME vs VME TAS Arbiter .................Running ------------> PASSED
```

6.30 VSB VS VMEbus ARBITER **BUS 3.A**

6.30.1 Description

This command is a test that verifies the ability of the MVME135 bus arbiter to handle the situation of one bus master accessing a VMEbus resource while another bus master simultaneously accesses a VSB resource.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize the two bus masters via semaphore.

Step 2: The two bus masters will each write a known pattern to memory. The UUT accesses the MVME204 via VSB. The Aux board accesses the MVME204 via the VMEbus.

Step 3: The target memory will be examined and compared against the expected patterns.

6.30.2 Command Input

```
135Diag> BUS 3.A
```

6.30.3 Response/Messages

After entering this command, the following should be displayed:

```
3.A   VSB vs VME Arbiter .....................Running ----------->
```

If the memory contains anything other than the expected values, then an error message will be printed out:

```
3.A   VSB vs VME Arbiter .....................Running ----------->
      Addr=00A00004      Expect=00004567      Read=FFFFFFFF
 .... FAILED
```

If the memory contains the expected values, that is the patterns that both bus masters attempted to write, then the test completes successfully.

```
3.A   VSB vs VME Arbiter .....................Running -----------> PASSED
```

6.31 VME VS LOCAL ARBITER FOR A SHARED RESOURCE BUS 3.B

6.31.1 Description

This command verifies the ability of the MVME135's system controller circuitry to arbitrate two bus masters accessing the same slave via different interfaces. In this test the UUT writes to its local DRAM via the local bus while the Aux simultaneously accesses the same DRAM using the VMEbus.

The UUT's DRAM is a shared resource. For each access to the UUT's DRAM, the Aux must request and be granted access to the UUT's local bus by the local processor. The local processor of the UUT must wait until the Aux's access is complete before resuming its own accesses to the DRAM.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize the two bus masters via semaphore.

Step 2: The two bus masters will each write a known pattern to memory. The UUT accesses the first half of its local DRAM via the local bus. The Aux board accesses the second half of the UUT's DRAM via the VMEbus.

Step 3: The target memory will be examined and compared against the expected patterns.

6.31.2 Command Input

```
135Diag> BUS 3.B
```

6.31.3 Response/Messages

After entering this command, the following should be displayed:

```
3.B   VME vs Local (Shared) Arbiter ..........Running ----------->
```

If the memory contains anything other than the expected values, then an error message will be printed out:

```
3.B   VME vs Local (Shared) Arbiter ..........Running ----------->
      Addr=00000004      Expect=00004567      Read=FFFFFFFF
 .... FAILED
```

If the memory contains the expected values, that is the patterns that both bus masters attempted to write, then the test completes successfully.

```
3.B   VME vs Local (Shared) Arbiter ..........Running ------------> PASSED
```

6.32 VME VS LOCAL ARBITER FOR A DUAL-PORTED RESOURCE BUS 3.C

6.32.1 Description

This command verifies the ability of the MVME135's system controller circuitry to arbitrate two bus masters accessing the same slave via different interfaces. In this test the UUT writes to its Multi-Processor CSR via its local bus while the Aux simultaneously accesses the UUT's MPCSR using the VMEbus.

The UUT's MPCSR is a dual-ported resource and can be accessed by the local processor and an external bus master in back-to-back cycles. The Aux MVME135 need not wait until the access by the UUT's local processor is complete before beginning its own access.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Start Aux board in loop reading from ID byte of UUT's MPCSR.

Step 2: UUT loops while writing/reading MP Comm byte from its own MPCSR. If a read gives a different value then that written, then signal error and end test.

Step 3: Stop Aux. If Aux shows any read errors then signal error and end test. Otherwise, start Aux in loop writing/reading from UUT's MP Comm byte.

Step 4: UUT loops reading from its own ID byte. If a read gives wrong value then signal error and end test.

Step 5: Stop Aux. If Aux shows any read errors then signal error and end test. Otherwise, test passes.

6.32.2 Command Input

```
135Diag> BUS 3.C
```

6.32.3 Response/Messages

After entering this command, the following should be displayed:

```
3.C   VME vs Local (dual-ported) Arbiter .....Running ----------->
```

If the UUT reads an unexpected value from the MP Comm byte or the ID byte, then an error message will be printed out:

```
3.C    VME vs Local (dual-ported) Arbiter .....Running ------------>
       Addr=FFFBØØ61       Expect=ØØØØØØ67       Read=ØØØØØØFF
 .... FAILED
```

If the Aux reads an unexpected value from the UUT's MP Comm byte or the UUT's ID byte, then an error message will be printed out:

```
3.C    VME vs Local (dual-ported) Arbiter .....Running ------------>
-Error during Aux read-
 .... FAILED
```

If both the UUT and the Aux read the expected values while looping, then the test completes successfully.

```
3.C    VME vs Local (dual-ported) Arbiter .....Running ------------> PASSED
```

6.33 VME MPCSR/MPCSR — BUS 4.A

6.33.1 Description

The purpose of this test is to prove that the UUT and the Aux can simultaneously access each other's MPCSRs via the VMEbus without interfering with each other.

This test is a type-4 bus diagnostic, that is, two masters simultaneously accessing different slaves via the same interface.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Start Aux in loop writing/reading from MP Comm byte in UUT's MPCSR.

Step 2: UUT loops while writing/reading MP Comm byte in Aux's MPCSR. If a read gives a different value then that written, then signal error and end test.

Step 3: Stop Aux. If Aux shows any read errors then signal error. Otherwise, test passes.

6.33.2 Command Input

```
135Diag> BUS 4.A
```

6.33.3 Response/Messages

After entering this command, the following should be displayed:

```
4.A   VME MPCSR/MPCSR ......................Running ----------->
```

If the UUT reads an unexpected value from the Aux's MP Comm byte, then an error message will be printed out:

```
4.A   VME MPCSR/MPCSR ......................Running ----------->
      Addr=FFFFC219      Expect=00000055      Read=000000FF
 .... FAILED
```

If the Aux reads an unexpected value from the UUT's MP Comm byte, then an error message will be printed out:

```
4.A    VME MPCSR/MPCSR ........................Running ------------>
-Error during Aux read-
 .... FAILED
```

If both the UUT and the Aux read the expected values while looping, then the test completes successfully.

```
4.A    VME MPCSR/MPCSR ........................Running ------------> PASSED
```

6.34 VME LOCAL/LOCAL NON-INTERFERENCE TEST BUS 4.B

6.34.1 Description

The purpose of this test is to prove that bus activity carried out by the UUT will not interfere with simultaneous bus activity carried out by the Aux (using the same bus) and vice-versa.

In this test the UUT accesses a particular slave (the Aux board's local DRAM) using the VMEbus at the same time the Aux is doing accesses to a different slave (the UUT's local DRAM) over the same interface (the VMEbus).

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize with the other bus master through a semaphore.

Step 2: Both bus masters try to write to different slaves via VMEbus at the same time.

Step 3: Check patterns written by both bus masters.

6.34.2 Command Input

```
135Diag> BUS 4.B
```

6.34.3 Response/Messages

After entering this command, the following should be displayed:

```
4.B   VME Local/Local .......................Running ----------->
```

If the memory contains anything other than the expected values, then an error message will be printed out:

```
4.B   VME Local/Local .......................Running ----------->
      Addr=ØØØØØØØ4      Expect=ØØØØ4567     Read=FFFFFFFF
 .... FAILED
```

If the memory contains the expected values, that is the patterns that both bus masters attempted to write, then the test completes successfully.

```
4.B   VME Local/Local .......................Running -----------> PASSED
```

6.35 VSB/VME INDEPENDENCE TEST BUS 5.A

6.35.1 Description

The purpose of this test is to prove that bus activity carried out by the UUT will not interfere with simultaneous bus activity carried out by the Aux (using a different bus) and vice-versa.

In this test the UUT accesses the MVME214's memory using the VSB while at the same time the Aux is doing accesses to the UUT's local DRAM over the VMEbus.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize with the other bus master through a semaphore.

Step 2: Both bus masters try to write to different slaves via different interfaces at the same time.

Step 3: Check patterns written by both bus masters.

6.35.2 Command Input

```
135Diag> BUS 5.A
```

6.35.3 Response/Messages

After entering this command, the following should be displayed:

```
5.A   VSB/VME Independence ..................Running ----------->
```

If the memory contains anything other than the expected values, then an error message will be printed out:

```
5.A   VSB/VME Independence ..................Running ----------->
      Addr=00000004      Expect=00004567      Read=FFFFFFFF
 .... FAILED
```

If the memory contains the expected values, that is the patterns that both bus masters attempted to write, then the test completes successfully.

```
5.A   VSB/VME Independence ..................Running -----------> PASSED
```

6.36 VME/VSB INDEPENDENCE TEST BUS 5.B

6.36.1 Description

The purpose of this test is to prove that bus activity carried out by the UUT will not interfere with simultaneous bus activity carried out by the Aux (using a different bus) and vice-versa.

In this test the UUT accesses the Aux's local DRAM over the VMEbus while at the same time the Aux is doing accesses to the MVME214's memory using the VSB.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize with the other bus master through a semaphore.

Step 2: Both bus masters try to write to different slaves via different interfaces at the same time.

Step 3: Check patterns written by both bus masters.

6.36.2 Command Input

```
135Diag> BUS 5.B
```

6.36.3 Response/Messages

After entering this command, the following should be displayed:

```
5.B    VME/VSB Independence ...................Running ----------->
```

If the memory contains anything other than the expected values, then an error message will be printed out:

```
5.B    VME/VSB Independence ...................Running ----------->
       Addr=00C00004       Expect=00004567       Read=FFFFFFFF
 .... FAILED
```

If the memory contains the expected values, that is the patterns that both bus masters attempted to write, then the test completes successfully.

```
5.B    VME/VSB Independence ...................Running -----------> PASSED
```

6.37 LOCAL/VME INDEPENDENCE TEST BUS 5.C

6.37.1 Description

The purpose of this test is to prove that bus activity carried out by the UUT will not interfere with simultaneous bus activity carried out by the Aux (using a different bus) and vice-versa.

In this test the UUT accesses it's own on-board DRAM using the local bus while at the same time the Aux is doing accesses to the MVME214's memory using the VMEbus.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize with the other bus master through a semaphore.

Step 2: Both bus masters try to write to different slaves via different interfaces at the same time.

Step 3: Check patterns written by both bus masters.

6.37.2 Command Input

```
135Diag> BUS 5.C
```

6.37.3 Response/Messages

After entering this command, the following should be displayed:

```
5.C   Local/VME Independence .................Running ----------->
```

If the memory contains anything other than the expected values, then an error message will be printed out:

```
5.C   Local/VME Independence .................Running ----------->
      Addr=00000004       Expect=00004567       Read=FFFFFFFF
 .... FAILED
```

If the memory contains the expected values, that is the patterns that both bus masters attempted to write, then the test completes successfully.

```
5.C   Local/VME Independence .................Running -----------> PASSED
```

6.38 LOCAL/VSB INDEPENDENCE TEST BUS 5.D

6.38.1 Description

The purpose of this test is to prove that bus activity carried out by the UUT will not interfere with simultaneous bus activity carried out by the Aux (using a different bus) and vice-versa.

In this test the UUT accesses it's own on-board DRAM using the local bus while at the same time the Aux is doing accesses to the MVME214's memory using the VSB.

This test requires that the Aux MVME135 be present in the chassis.

The test is implemented in the following manner:

Step 1: Synchronize with the other bus master through a semaphore.

Step 2: Both bus masters try to write to different slaves via different interfaces at the same time.

Step 3: Check patterns written by both bus masters.

6.38.2 Command Input

```
135Diag> BUS 5.D
```

6.38.3 Response/Messages

After entering this command, the following should be displayed:

```
5.D   Local/VSB Independence .................Running ----------->
```

If the memory contains anything other than the expected values, then an error message will be printed out:

```
5.D   Local/VSB Independence .................Running ----------->
      Addr=00000004       Expect=00004567       Read=FFFFFFFF
 .... FAILED
```

If the memory contains the expected values, that is the patterns that both bus masters attempted to write, then the test completes successfully.

```
5.D   Local/VSB Independence .................Running -----------> PASSED
```

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 7
MC68020 (ON-CHIP) CACHE TESTS

7.1 GENERAL DESCRIPTION

This section details the diagnostics provided to test the MC68020 cache. Table 7-1 lists the MC68020 cache diagnostic tests.

TABLE 7-1. MC68020 CACHE DIAGNOSTIC TESTS

Monitor Command	Title	Section
CA20 F	Basic Caching	7.3
CA20 G	Unlike Function Codes	7.4
CA20 H	Disable Test	7.5
CA20 I	Clear Test	7.6

7.2 HARDWARE CONFIGURATION

The following hardware is required to perform these tests.

MVME135 CPU - Board being tested.
VME chassis.
Video Display Terminal.

7.3 BASIC CACHING TEST CA2Ø F

7.3.1 Description

This command tests the basic caching function of the MC68Ø2Ø microprocessor. The test caches a program segment that resides in RAM, freezes the cache, changes the program segment in RAM, then reruns the program segment. If the cache is functioning correctly, the cached instructions will be executed. Failure is detected if the MC68Ø2Ø executes the instructions that reside in RAM; any cache misses will cause an error.

The process is first attempted in supervisor mode for both the initial pass through the program segment and the second pass. It is then repeated, using user mode for both the initial pass and the second pass. A bit is included in each cache entry for distinguishing between supervisor and user mode. If this bit is stuck or inaccessible, the cache will miss during one of these two tests.

7.3.2 Command Input

```
135Diag> CA2Ø F
```

7.3.3 Response/Messages

After entering this command, the following should be printed out:

```
F     Basic Caching .........................Running ----------->
```

If there are any cache misses during the second pass through the program segment, then the test fails and the display will appear as follows:

```
F     Basic Caching .........................Running ----------->
2 CACHE MISSES!
CACHED IN SUPV MODE, RERAN IN SUPV MODE
 .... FAILED
```

If there are no cache misses during the second pass, then the test passes.

```
F     Basic Caching .........................Running -----------> PASSED
```

7.4 UNLIKE FUNCTION CODES TEST CA2Ø G

7.4.1 Description

This command tests the ability of the on-chip cache to recognize function codes. Bit 2 of the function code is included in the tag for each entry. This provides a distinction between supervisor and user modes for the cached instructions. To test this mechanism, a program segment that resides in RAM is cached in supervisor mode. The cache is then frozen and the program segment in RAM is changed. When the program segment is executed a second time in user mode, there should be no cache hits due to the different fn. codes. Failure is detected if the MC68Ø2Ø executes the cached instructions.

After the program segment has been cached in supervisor mode and rerun in user mode, the process is repeated, caching in user mode and rerunning in supervisor mode. Again, the cache should miss during the second pass through the program segment.

7.4.2 Command Input

```
135Diag> CA2Ø G
```

7.4.3 Response/Messages

After entering this command, the following should be printed out:

```
G    Unlike fn. Codes ......................Running ----------->
```

If there are any cache hits during the second pass through the program segment, then the test fails and the display will appear as follows:

```
G    Unlike fn. Codes ......................Running ----------->
5 CACHE HITS!
CACHED IN SUPV MODE, RERAN IN USER MODE
 .... FAILED
```

If there are no cache hits during the second pass, then the test passes.

```
G    Unlike fn. Codes ......................Running -----------> PASSED
```

7.5 DISABLE TEST **CA2Ø H**

7.5.1 Description

In the MC68Ø2Ø cache control register ("CACR") a control bit is provided to enable the cache. When this bit is clear, the cache should never hit, regardless of whether the address and function codes match a tag. To test this mechanism, a program segment is cached from RAM. The cache is frozen to preserve its contents, then the enable bit is cleared. The program segment in RAM is then changed and rerun. There should be no cache hits with the enable bit clear. Failure is declared if the cache does hit.

7.5.2 Command Input

```
135Diag> CA2Ø H
```

7.5.3 Response/Messages

After entering this command, the following should be printed out:

```
H     Disable Test .........................Running ----------->
```

If there are any cache hits during the second pass through the program segment, then the test fails and the display will appear as follows:

```
H     Disable Test .........................Running ----------->
1 CACHE HIT!
CACHED IN SUPY MODE, RERAN IN SUPY MODE
 .... FAILED
```

If there are no cache hits during the second pass, then the test passes.

```
H     Disable Test .........................Running -----------> PASSED
```

7.6 CLEAR TEST CA2Ø I

7.6.1 Description

A control bit is included in the MC68Ø2Ø Cache Control register ("CACR") to clear the cache. Writing a one to this bit invalidates every entry in the on-chip cache. To test this function, a program segment in RAM is cached and then frozen there to preserve it long enough to assert the cache clear control bit. The program segment in RAM is then modified and rerun with the cache enabled. If the cache hits, the clear is incomplete and failure is declared.

7.6.2 Command Input

```
135Diag> CA2Ø I
```

7.6.3 Response/Messages

After entering this command, the following should be printed out:

```
I     Clear Test ...........................Running ----------->
```

If there are any cache hits during the second pass through the program segment, then the test fails and the display will appear as follows:

```
I     Clear Test ...........................Running ----------->
58 CACHE HITS!
CACHED IN SUPY MODE, RERAN IN SUPY MODE
 .... FAILED
```

If there are no cache hits during the second pass, then the test passes.

```
I     Clear Test ...........................Running ----------->  PASSED
```

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 8
CIO (COUNTER/TIMER) TESTS

An important feature of the MVME135 is a combination parallel interface and counter/timer, hereafter referred to as the CIO. The CIO is intended to provide two parallel input/output ports and three linkable 16-bit timers with interrupt capability.

8.1 GENERAL DESCRIPTION

The Z8Ø36 CIO is a Zilog product that features three parallel input/output ports and three 16-bit counter/timers, two of which can be linked together to form one 32-bit counter/timer. Two of the parallel ports (A and B) are dedicated to control functions on the MVME135. Use of the counter/timers is user definable, as they are not dedicated.

The five tests (refer to Table 8-1) included for the CIO are organized according to complexity. All of the tests are included in the self-test sequence. A detailed description explains the format used to dump the contents of the CIO registers following an error.

TABLE 8-1. CIO DIAGNOSTIC TESTS

Monitor Command	Title	Section
CIO A	Post Reset Initialization	8.3
CIO B	Registers	8.4
CIO C	Count Down	8.5
CIO D	Interrupts	8.6
CIO F	Linked Timers 4 MHz clock	8.7

8.2 HARDWARE CONFIGURATION

The following hardware is required to perform these tests.

MVME135 CPU - Board being tested.
Video Display Terminal.

8.3 POST RESET INITIALIZATION TEST CIO A

8.3.1 Description

This test resets the CIO and checks most of the registers. Some registers reset to a known value, while others are not affected by the reset. The Current Count registers are not examined during this test.

8.3.2 Command Input

```
135Diag> CIO A
```

8.3.3 Response/Messages

The monitor will respond with:

```
A     Post Reset Init ........................Running ------------>
```

If an error occurs, the test will display the contents of the CIO and an error message. Refer to section 8.8 for a description of the CIO register dump. The display should appear as follows:

```
A     Post Reset Init ........................Running ------------>
CIO CONTENTS:   (see section 8.8)
       :
       :
*** ERROR: CIO REG. DIDN'T RESET
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
A     Post Reset Init ........................Running ------------> PASSED
```

8.4 REGISTERS TEST — CIO B

8.4.1 Description

This test exercises the CIO registers by writing data to each and then reading it back, much like a memory test. Some registers have bits that do not latch data, therefore, they are not tested in this manner. Other registers have bits that are set or cleared by writing a code to the upper three bits (all of the command/status registers). Failure is declared when a bit cannot be set or cleared.

8.4.2 Command Input

```
135Diag> CIO B
```

8.4.3 Response/Messages

The monitor will respond with:

```
B     Registers ............................Running ----------->
```

If an error occurs, the test will display the contents of the CIO and an error message. Refer to section 8.8 for a description of the CIO register dump. The display should appear as follows:

```
B     Registers ............................Running ----------->
CIO CONTENTS:   (see section 8.8)
       :
       :
*** ERROR: BAD CIO REG
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
B     Registers ............................Running -----------> PASSED
```

8.5 COUNT DOWN TEST CIO C

8.5.1 Description

This test exercises the three counter/timers in the CIO. Failure is declared if the timer cannot be triggered, if it does not count down to zero in a specified time interval, or if the Interrupt Pending bit in its command/status register is not set when the count reaches zero.

8.5.2 Command Input

```
135Diag> CIO C
```

8.5.3 Response/Messages

The monitor will respond with:

```
C     Count Down ............................Running ----------->
```

If an error occurs, the test will display the contents of the CIO and an error message. Refer to section 8.8 for a description of the CIO register dump. The display should appear as follows:

```
C     Count Down ............................Running ----------->
CIO CONTENTS:   (see section 8.8)
       :
       :
*** ERROR: CIO CTR/TIMER FAILED
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
C     Count Down ............................Running -----------> PASSED
```

8.6 INTERRUPTS TEST — CIO D

8.6.1 Description

This test verifies the ability of each of the three counter/timers to interrupt when its count reaches zero. Failure is declared (on a per counter/timer basis) if either an interrupt does not occur within a given time interval or if more than one interrupt occurs.

8.6.2 Command Input

```
135Diag> CIO D
```

8.6.3 Response/Messages

The monitor will respond with:

```
D     Interrupts ............................Running ----------->
```

If an counter/timer does not generate an interrupt within the given time, the test will display the contents of the CIO and "INTERRUPT MISSING". Refer to section 8.8 for a description of the CIO register dump. The display should appear as follows:

```
D     Interrupts ............................Running ----------->
CIO CONTENTS:   (see section 8.8)
        :
        :
*** ERROR: INTERRUPT MISSING
 .... FAILED
```

If an counter/timer generates more than one interrupt or the interrupt cannot be cleared, the test will display the contents of the CIO and "EXCESSIVE INTERRUPTS". Refer to section 8.8 for a description of the CIO register dump. The display should appear as follows:

```
D     Interrupts ............................Running ----------->
CIO CONTENTS:   (see section 8.8)
        :
        :
*** ERROR: EXCESSIVE INTERRUPTS
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
D     Interrupts ............................Running ------------> PASSED
```

8.7 LINKED TIMERS 4 MHZ CLOCK — CIO F

8.7.1 Description

This test will check the linking of all three timers of the CIO running with an internal clock (4 MHz). Among the functions tested are reloading, borrowing across the link, and interrupts out of Timer 3.

The basic algorithm used in this test is as follows:

Step 1: Initialize CIO to known state.

Step 2: Program the CIO to link the three timers.

Step 3: Trigger and wait for all three counters to count down and rollover.

Step 4: Reload counters and verify loading takes place.

Step 5: Trigger counters and verify interrupt out of the last stage on rollover (counter 3).

8.7.2 Command Input

```
135Diag> CIO F
```

8.7.3 Response/Messages

The monitor will respond with:

```
F     Linked Timers, 4 MHz clk ...............Running ----------->
```

If an error occurs, one of several messages may be printed out. The basic form will be as follows:

```
F     Linked Timers, 4 MHz clk ...............Running ----------->
(error message)
 .... FAILED
```

Error messages that can be printed are as follows:

TIMERS 3 AND 2 NOT LINKED
Indicates IP set on Timer 2 because of count rollover with no IP set on Timer 3.

TIMERS 2 AND 3 NOT COUNTING
Indicates IP set on Timer 1 because of count rollover with no IP set on Timer 2.

TIMERS DID NOT TRIGGER
Indicates no IP set on Timer 1.

TIMER #1 NOT LOADED, nnnn
Failure during step 4 with nnnn being the value read out of Timer 1.
Expected is $1ØØ.

TIMER #2 NOT LOADED, nnnn
Failure during step 4 with nnnn being the value read out of Timer 2.
Expected is 4.

TIMER #3 NOT LOADED, nnnn
Failure during step 4 with nnnn being the value read out of Timer 3.
Expected is 3.

TIMED OUT, EXPECTED 3 INTERRUPTS GOT n
Failure during step 5 with n being the number of interrupts received.

If no errors occur, then the display will appear as follows:

```
F     Linked Timers, 4 MHz clk ..............Running ------------> PASSED
```

8.8 DESCRIPTION OF CIO DUMP FORMAT

To aid in the diagnosis of CIO problems, all of the tests display the contents of the CIO registers when reporting errors. The labels and offsets displayed will be cryptic to anyone attempting to use these tests without the Z8Ø36 Technical Manual. Have the Technical Manual for the Z8Ø36 nearby when using the CIO tests.

To reduce the amount of confusion in reading the display, each register has been assigned a label. The number in parentheses is the offset from the base address. The value to the immediate right of the colon is the content of the register.

Figure 8-1 is a sample display of the CIO registers' contents. Table 8-2 lists the label, offset, and description of each register.

```
CIO CONTENTS:
 MICR(ØØ):  Ø2    MCCR(Ø1):  F4    PAIV(Ø2):  ØF    PBIV(Ø3):  ØF
 CTIV(Ø4):  ØF    PCDP(Ø5):  FØ    PCDD(Ø6):  FØ    PCSC(Ø7):  FØ
 PACS(Ø8):  ØØ    PBCS(Ø9):  ØØ  C1CSR(ØA):  ØØ  C2CSR(ØB):  ØØ
C3CSR(ØC):  ØØ    PADR(ØD):  FF    PBDR(ØE):  58    PCDR(ØF):  ØF
CT1CM(1Ø):  12   CT1CL(11):  34   CT2CM(12):  56   CT2CL(13):  78
CT3CM(14):  9A   CT3CL(15):  BC   CT1PM(16):  ØØ   CT1PL(17):  1Ø
CT2PM(18):  5Ø   CT2PL(19):  1F   CT3PM(1A):  49   CT3PL(1B):  ØØ
CT1MO(1C):  Ø4   CT2MO(1D):  Ø4   CT3MO(1E):  Ø4    CTCV(1F):  FF
 PAMO(2Ø):  ØØ    PAHS(21):  ØØ    PADP(22):  ØØ    PADD(23):  ØØ
 PASC(24):  ØØ    PAPP(25):  ØØ    PAPT(26):  ØØ    PAPM(27):  ØØ
 PBMØ(28):  ØØ    PBHS(29):  ØØ    PBDP(2A):  ØØ    PBDD(2B):  ØØ
 PBSC(2C):  ØØ    PBPP(2D):  ØØ    PBPT(2E):  ØØ    PBPM(2F):  ØØ
```

FIGURE 8-1. SAMPLE CIO REGISTER DUMP

TABLE 8-2. CIO REGISTERS

Label	Offset	MVME135 Address	Section
MICR	$ØØ	$FFFBØØØØ	Master Interrupt Control
MCCR	$Ø1	$FFFBØØØ1	Master Configuration Control
PAIV	$Ø2	$FFFBØØØ2	Port A Interrupt Vector
PBIV	$Ø3	$FFFBØØØ3	Port B Interrupt Vector
CTIV	$Ø4	$FFFBØØØ4	Counter/Timer Interrupt Vector
PCDP	$Ø5	$FFFBØØØ5	Port C Data Polarity
PCDD	$Ø6	$FFFBØØØ6	Port C Data Direction
PCSC	$Ø7	$FFFBØØØ7	Port C Special I/O Control
PACS	$Ø8	$FFFBØØØ8	Port A Command/Status
PBCS	$Ø9	$FFFBØØØ9	Port B Command/Status
C1CSR	$ØA	$FFFBØØØA	Counter/Timer 1 Command/Status
C2CSR	$ØB	$FFFBØØØB	Counter/Timer 2 Command/Status
C3CSR	$ØC	$FFFBØØØC	Counter/Timer 3 Command/Status
PADR	$ØD	$FFFBØØØD	Port A data Register
PBDR	$ØE	$FFFBØØØE	Port B data Register
PCDR	$ØF	$FFFBØØØF	Port C data Register
CT1CM	$1Ø	$FFFBØØ1Ø	Counter/Timer 1 Current Count, MSB
CT1CL	$11	$FFFBØØ11	Counter/Timer 1 Current Count, LSB
CT2CM	$12	$FFFBØØ12	Counter/Timer 2 Current Count, MSB
CT2CL	$13	$FFFBØØ13	Counter/Timer 2 Current Count, LSB
CT3CM	$14	$FFFBØØ14	Counter/Timer 3 Current Count, MSB
CT3CL	$15	$FFFBØØ15	Counter/Timer 3 Current Count, LSB
CT1PM	$16	$FFFBØØ16	Counter/Timer 1 Preload, MSB*
CT1PL	$17	$FFFBØØ17	Counter/Timer 1 Preload, LSB*
CT2PM	$18	$FFFBØØ18	Counter/Timer 2 Preload, MSB*
CT2PL	$19	$FFFBØØ19	Counter/Timer 2 Preload, LSB*
CT3PM	$1A	$FFFBØØ1A	Counter/Timer 3 Preload, MSB*
CT3PL	$1B	$FFFBØØ1B	Counter/Timer 3 Preload, LSB*
CT1MO	$1C	$FFFBØØ1C	Counter/Timer 1 Mode Specification
CT2MO	$1D	$FFFBØØ1D	Counter/Timer 2 Mode Specification
CT3MO	$1E	$FFFBØØ1E	Counter/Timer 3 Mode Specification
CTCV	$1F	$FFFBØØ1F	Counter/Timer Current Vector
PAMO	$2Ø	$FFFBØØ2Ø	Port A Mode Specification
PAHS	$21	$FFFBØØ21	Port A Handshake Specification
PADP	$22	$FFFBØØ22	Port A Data Path Polarity

TABLE 8-2. CIO REGISTERS (cont.)

Label	Offset	MVME135 Address	Section
PADD	$23	$FFFBØØ23	Port A Data Direction
PASC	$24	$FFFBØØ24	Port A Special I/O Control
PAPP	$25	$FFFBØØ25	Port A Pattern Polarity
PAPT	$26	$FFFBØØ26	Port A Pattern Translation
PAPM	$27	$FFFBØØ27	Port A Pattern Mask
PBMO	$28	$FFFBØØ28	Port B Mode Specification
PBHS	$29	$FFFBØØ29	Port B Handshake Specification
PBDP	$2A	$FFFBØØ2A	Port B Data Path Polarity
PBDD	$2B	$FFFBØØ2B	Port B Data Direction
PBSC	$2C	$FFFBØØ2C	Port B Special I/O Control
PBPP	$2D	$FFFBØØ2D	Port B Pattern Polarity
PBPT	$2E	$FFFBØØ2E	Port B Pattern Translation
PBPM	$2F	$FFFBØØ2F	Port B Pattern Mask

* Timer Constant

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 9
MEMORY MANAGEMENT UNIT (MMU) TESTS

9.1 GENERAL DESCRIPTION

This section details the diagnostics provided to test the memory management hardware in the system. This refers to the **MC68851 PMMU** (Paged Memory Management Unit). The diagnostics will configure themselves for the PMMU without explicit configuration by the user. Table 9-1 lists the memory management unit diagnostic tests.

TABLE 9-1. MEMORY MANAGEMENT UNIT DIAGNOSTICS

Monitor Command	Title	Section
MMU A	PR Registers	9.3
MMU B	TC Registers	9.4
MMU C	Super_Prog Space	9.5
MMU D	Super_Data Space	9.6
MMU E	Write/Mapped-Read Pages	9.7
MMU F	Read Mapped ROM	9.8
MMU G	Fully Filled ATC	9.9
MMU H	User_Data Space	9.1Ø
MMU I	User_Prog Space	9.11
MMU J	Indirect Page	9.12
MMU K	Page-Desc Used-Bit	9.13
MMU L	Page-Desc Modify-Bit	9.14
MMU M	Segment-Desc Used-Bit	9.15
MMU P	Invalid Page	9.16
MMU Q	Invalid-Segment	9.17
MMU R	Write-Protect Page	9.18
MMU S	Write-Protect Segment	9.19
MMU T	Write-Access-Level	9.2Ø
MMU U	Read-Access-Level	9.21
MMU V	Upper-Limit Violation	9.22
MMU W	Lower-Limit Violation	9.23
MMU X	Prefetch on Invalid-Page Boundary	9.24
MMU Y	Modify - Bit and Index	9.25
MMU Z	Sixteen-Bit Bus	9.26
MMU Z Ø	User-Prog Space	9.26.1

TABLE 9-1. MEMORY MANAGEMENT UNIT DIAGNOSTICS (cont.)

Monitor Command	Title	Section
MMU Z 1	Page-Desc Modify-Bit	9.26.2
MMU Z 2	Indirect Page	9.26.3
MMU Ø	Read/Modify/Write Cycle	9.27

9.2 HARDWARE CONFIGURATION

The following hardware is required to perform these tests.

Memory Management Unit (MC68851)
MVME135 CPU - Host for MMU being tested.
MVME2Ø4 RAM Board - Addressed at $AØØØØØ (VMEbus) and $CØØØØØ (VSB).
Video Display Terminal.

9.3 RP REGISTER MMU A

9.3.1 Description

This command tests the root pointer register by doing a walking bit through it.

9.3.2 Command Input

```
135Diag> MMU A
```

9.3.3 Response/Messages

After entering this command, the display should read as follows:

```
A     RP Register ...........................Running ------------>
```

If the root pointer register fails to latch correctly, then the test fails and the following error message is printed out:

```
A     RP Register ...........................Running ------------>
      Expect=ØØØØØØ1Ø       Read=FFFFFFFF
 .... FAILED
```

If the walk-a-bit test is successful, then the root pointer register test passes.

```
A     RP Register ...........................Running ------------> PASSED
```

9.4 TC REGISTER　　　　MMU B

9.4.1 Description

This command tests the translation-control register by attempting to clear and then set the Initial Shift (IS) bit.

9.4.2 Command Input

```
135Diag> MMU B
```

9.4.3 Response/Messages

After entering this command, the display should read as follows:

```
B       TC Register ...........................Running ----------->
```

If the bit cannot be cleared and set, then the test fails.

```
B       TC Register ...........................Running ----------->
        Expect=ØØØØ8Ø1Ø       Read=ØØØØØØØØ
 .... FAILED
```

If the bit gets cleared and set, then the test passes.

```
B       TC Register ...........................Running -----------> PASSED
```

9

9.5 SUPER_PROG SPACE MMU C

9.5.1 Description

This command enables the MMU and lets it do a table walk in supervisor program space. The test has been implemented in the following manner:

Step 1: Put the function code in the MC68Ø2Ø DFC register.

Step 2: Load the address of function code table into the root pointer register.

Step 3: Set the E bit in the translation control (TC) register.

Step 4: Let the MMU do a table walk for the next instruction; a NOP.

Step 5: Clear the E bit in the TC register.

9.5.2 Command Input

```
135Diag> MMU C
```

9.5.3 Response/Messages

After entering this command, the display should read as follows:

```
C     Super_Prog Space ......................Running ------------>
```

If a bus error occurs, then the test fails.

```
C     Super_Prog Space ......................Running ------------>
(bus error: CPU registers dumped to screen here)
 .... FAILED
```

If the table walk does not cause a bus error, then the test passes.

```
C     Super_Prog Space ......................Running ------------> PASSED
```

9.6 SUPER_DATA SPACE MMU D

9.6.1 Description

This command enables the MMU and lets it do a table walk twice in supervisor program space, then once in supervisor data space. The two walks in supervisor program space are necessary to fetch the instruction that accesses the supervisor data space and prefetch the next one. The test has been implemented in the following manner:

Step 1: Put the function code in the MC68020 DFC register.

Step 2: Load the address of function code table into the root pointer register.

Step 3: Set the E bit in the translation control (TC) register.

Step 4: Let the MMU do a table walk for the next instruction, which will cause the access to supervisor data space via a read (TST.B). Prefetching will cause access to the next location, in supervisor program space.

Step 5: Clear the E bit in the TC register.

9.6.2 Command Input

```
135Diag> MMU D
```

9.6.3 Response/Messages

After entering this command, the display should read as follows:

```
D     Super_Data Space ......................Running ----------->
```

If a bus error occurs, then the test fails.

```
D     Super_Data Space ......................Running ----------->
(bus error: CPU registers dumped to screen here)
 .... FAILED
```

If the table walk does not cause a bus error(s), then the test passes.

```
D     Super_Data Space ......................Running -----------> PASSED
```

9.7 WRITE/MAPPED-READ PAGES MMU E

9.7.1 Description

This command is a test that writes data with the MMU disabled, then verifies that the data can be read with the MMU enabled. The test has been implemented in the following manner:

Step 1: Fill two pages with a pattern.

Step 2: Enable the MMU.

Step 3: Check RAM where the two pages were written. Report all discrepancies.

9.7.2 Command Input

```
135Diag> MMU E
```

9.7.3 Response/Messages

After entering this command, the display should read as follows:

```
E     Write/Mapped-Read Pages ................Running ------------>
```

If a bus error occurs or data read does not equal that expected, then the test fails. A bus error generates a dump of the MC68020 MPU registers' contents. Data corruption generates a message that indicates where the error occurred, then a map of the table walk will be displayed. Refer to section 9.28 for an explanation of the map format.

```
E     Write/Mapped-Read Pages ................Running ------------>
      Addr=00000000       Expect=00000001       Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the reading of the two pages does not generated a bus error, and the patterns read match the expected, then the test passes.

```
E     Write/Mapped-Read Pages ................Running ------------> PASSED
```

9.8 READ MAPPED ROM — MMU F

9.8.1 Description

This command tests some of the upper MMU address lines by attempting to access the ROM. Both supervisor program and supervisor data function codes are used to test two separate paths through the translation table. The test has been implemented in the following manner:

Step 1: Set up a pointer to the ROM that will be PC relative. All PC relative accesses will use supervisor program space and will be software transparent.

Step 2: Set up a pointer to the ROM that will use virtual addressing. Accesses using this pointer will be to supervisor data space.

Step 3: Enable the MMU.

Step 4: For each location in the ROM, read the ROM via both pointers. The data read should be identical.

NOTE: The table walking for the supervisor data space takes a much different path from that used for supervisor program space. If the data does not match, then the test fails. Display the physical address, the expected data, and the read data.

Step 5: Disable the MMU.

9.8.2 Command Input

```
135Diag> MMU F
```

9.8.3 Response/Messages

After entering this command, the display should read as follows:

```
F     Read Mapped ROM .......................Running ----------->
```

If the data read via the two pointers ever differs, then the test fails.

```
F     Read Mapped ROM .......................Running ----------->
      Addr=00100000      Expect=00000001      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the test is able to read every ROM location via both paths, then it passes.

```
F      Read Mapped ROM .......................Running ------------> PASSED
```

9.9 FULLY FILLED ATC **MMU G**

9.9.1 Description

This command tests the Address Translation Cache by verifying that all the entries in the translation cache can hold a page descriptor.

For the **MMB851** this is done by filling the ATC, then accessing the data that caused the caching to see if the addresses were cached correctly. This method of testing is used because the ATC cannot be directly read. The best that can be done is to read enough data to fill it and check each item read against a list of what was originally at each location. The test has been implemented in the following manner:

Step 1: Create a list of the contents of the first word of each RAM page.

Step 2: Enable the MMU.

Step 3: Compare the contents of the list against the first words in each page to see if the caching worked.

Step 4: The test fails if any of the comparisons reveal bad address caching.

For the **PMMU** this is done by filling the ATC with locked descriptors, and then verifying that each descriptor is resident in the cache. This is implemented as follows:

Step 1: The lock bit is set in the first 63 page descriptors.

Step 2: The first word in each of those pages is read, creating an entry for each page in the ATC.

Step 3: The **LW** (Lock Warning) bit in the **PCSR** register is checked, and if it is not set, an error is flagged.

Step 4: The PMMU **PTEST** instruction is used to verify that the page descriptors for each of the 63 pages reside in the ATC.

9.9.2 Command Input

```
135Diag> MMU G
```

9.9.3 Response/Messages

After entering this command, the display should read as follows:

```
G    Fully Filled ATC .......................Running ----------->
```

If a word in the list does not match the corresponding word at the beginning of a page, then the test fails.

```
G     Fully Filled ATC .......................Running ------------>
      Addr=00000000       Expect=00000001       Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If every word in the list matches the first word of each page, then the test passes.

```
G     Fully Filled ATC .......................Running ------------> PASSED
```

9.1Ø USER_DATA SPACE   MMU H

9.1Ø.1 Description

This command tests the function code signal lines connecting into the MMU by accessing user data space. This causes the MMU to read the function code and do a table walk as a part of its translation. The test has been implemented in the following manner:

Step 1: Write a pattern out to an area that is mapped to user data space for diagnostic purposes.

Step 2: Enable the MMU.

Step 3: Read the area where the pattern was written to, using the function code for user data space. The test fails if the pattern does not match that written out.

9.1Ø.2 Command Input

```
135Diag> MMU H
```

9.1Ø.3 Response/Messages

After entering this command, the display should read as follows:

```
H     User_Data Space .......................Running ----------->
```

If the pattern written out does not match that read, the test fails.

```
H     User_Data Space .......................Running ----------->
      Addr=ØØØØØØØØ       Expect=ØØØØØØØ1       Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the pattern written out matches the one read, the test passes.

```
H     User_Data Space .......................Running -----------> PASSED
```

9.11 USER_PROG SPACE MMU I

9.11.1 Description

This command tests the function code signal lines connecting into the MMU by accessing user program space. This causes the MMU to read the function code and do a table walk as a part of its translation. The test has been implemented in the following manner:

Step 1: Write a pattern out to an area that is mapped to user program space for diagnostic purposes.

Step 2: Enable the MMU.

Step 3: Read the area where the pattern was written to, using the function code for user program space. The test fails if the pattern does not match that written out.

9.11.2 Command Input

```
135Diag> MMU I
```

9.11.3 Response/Messages

After entering this command, the display should read as follows:

```
I     User_Prog Space .......................Running ----------->
```

If the pattern written out does not match that read, the test fails.

```
I     User_Prog Space .......................Running ----------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the pattern written out matches the one read, the test passes.

```
I     User_Prog Space .......................Running -----------> PASSED
```

9.12 INDIRECT PAGE MMU J

9.12.1 Description

This command tests the ability of the MMU to handle an indirect descriptor. The test has been implemented in the following manner:

Step 1: Modify the descriptor for the first RAM page to point to the descriptor for the next RAM page.

Step 2: Write a known value into the first location of the second RAM page and the complement of that value into the first location of the first RAM page.

Step 3: Enable the MMU.

Step 4: Read the first location of the first virtual RAM page. This should address the first location in the second physical RAM page due to the indirect.

Step 5: If the value read is not the value written out to the second RAM page in Step 2, then the test fails.

Step 6: Disable the MMU.

9.12.2 Command Input

```
135Diag> MMU J
```

9.12.3 Response/Messages

After entering this command, the display should read as follows:

```
J     Indirect Page .........................Running ----------->
```

If the value that was supposedly read from the first virtual page in Step 4 does not match the value written in Step 2 to the second physical page, then the test fails.

```
J     Indirect Page .........................Running ----------->
      Addr=00000000       Expect=00000001       Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the value matches, then the indirect mechanism has functioned correctly and the test passes.

```
J     Indirect Page .........................Running -----------> PASSED
```

9.13 PAGE-DESC USED-BIT **MMU K**

9.13.1 Description

This command tests the ability of the MMU to set the Used-bit in a page descriptor when the page gets accessed. The test has been implemented in the following manner:

Step 1: Clear the Used-bit in a page descriptor.

Step 2: Enable the MMU.

Step 3: Read from the page.

Step 4: Examine the page descriptor. If the Used-bit is not set, then the test fails.

9.13.2 Command Input

```
135Diag> MMU K
```

9.13.3 Response/Messages

After entering this command, the display should read as follows:

```
K      Page-Desc Used-Bit ......................Running ----------->
```

If the Used-bit does not get set by the access in Step 3, then the test fails.

```
K      Page-Desc Used-Bit ......................Running ----------->
       Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the Used-bit does get set, then the test passes.

```
K      Page-Desc Used-Bit ......................Running -----------> PASSED
```

9.14 PAGE-DESC MODIFY-BIT MMU L

9.14.1 Description

This command tests the ability of the MMU to set the Modify-bit in a page descriptor when the page is written. The test has been implemented in the following manner:

Step 1: Clear the Modify-bit in a page descriptor.

Step 2: Enable the MMU.

Step 3: Write from the page.

Step 4: Examine the page descriptor. If the Modify-bit is not set, then the test fails.

9.14.2 Command Input

```
135Diag> MMU L
```

9.14.3 Response/Messages

After entering this command, the display should read as follows:

```
L     Page-Desc Modify-Bit ..................Running ----------->
```

If the Modify-bit does not get set by the access in Step 3, then the test fails.

```
L     Page-Desc Modify-Bit ..................Running ----------->
      Addr=ØØØØØØØØ      Expect=ØØØØØØØ1      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the Modify-bit does get set, then the test passes.

```
L     Page-Desc Modify-Bit ..................Running -----------> PASSED
```

9.15 SEGMENT-DESC USED-BIT MMU M

9.15.1 Description

This command tests the ability of the MMU to set the Used-bit in a segment descriptor when the corresponding segment is accessed. The test has been implemented in the following manner:

Step 1: Clear the Used-bit in a segment descriptor.

Step 2: Enable the MMU.

Step 3: Read from an address mapped to that segment.

Step 4: Check the Used-bit in the segment descriptor. If it has not been set, the test fails.

9.15.2 Command Input

```
135Diag> MMU M
```

9.15.3 Response/Messages

After entering this command, the display should read as follows:

```
M    Segment-Desc Used-Bit ..................Running ------------>
```

If the Used-bit does not get set by the access in Step 3, then the test fails.

```
M    Segment-Desc Used-Bit ..................Running ------------>
     Addr=00000000     Expect=00000001     Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the Used-bit does get set, then the test passes.

```
M    Segment-Desc Used-Bit ..................Running ------------> PASSED
```

9.16 INVALID PAGE   MMU P

9.16.1 Description

This command tests the ability of the MMU to detect an invalid page and generate bus error when access is attempted to that page. The invalid page is intentionally declared that way for test purposes. The test has been implemented in the following manner:

Step 1: Modify the descriptor for a RAM page to make it invalid.

Step 2: Enable the MMU.

Step 3: Attempt to read from the page. This should generate a bus error.

Step 4: If no bus error occurred, then the test fails.

9.16.2 Command Input

```
135Diag> MMU P
```

9.16.3 Response/Messages

After entering this command, the display should read as follows:

```
P      Invalid Page .........................Running ----------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
P      Invalid Page .........................Running ----------->
       Addr=00000000       Expect=00000001       Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
P      Invalid Page .........................Running -----------> PASSED
```

9.17 INVALID SEGMENT MMU Q

9.17.1 Description

This command tests the ability of the MMU to detect an invalid segment and generate bus error when access is attempted to that segment. The invalid segment is intentionally declared that way for test purposes. The test has been implemented in the following manner:

Step 1: Modify the descriptor for a RAM segment to make it invalid.

Step 2: Enable the MMU.

Step 3: Attempt to read from the page in the segment. This should generate a bus error.

Step 4: If no bus error occurred, then the test fails.

9.17.2 Command Input

```
135Diag> MMU Q
```

9.17.3 Response/Messages

After entering this command, the display should read as follows:

```
Q     Invalid Segment ........................Running ----------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
Q     Invalid Segment ........................Running ----------->
      Addr=00000000       Expect=00000001       Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
Q     Invalid Segment ........................Running -----------> PASSED
```

9.18 WRITE-PROTECT PAGE MMU R

9.18.1 Description

This command tests the page write-protect mechanism in the MMU. If the MMU is functioning correctly, then attempting a write to a protected page will cause a bus error. The test has been implemented in the following manner:

Step 1: Set the WP bit in the descriptor for the first RAM page.

Step 2: Enable the MMU.

Step 3: Attempt to write to the protected page.

Step 4: If a bus error does not occur, then the test fails.

9.18.2 Command Input

```
135Diag> MMU R
```

9.18.3 Response/Messages

After entering this command, the display should read as follows:

```
R     Write-Protect Page ....................Running ----------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
R     Write-Protect Page ....................Running ----------->
      Addr=ØØØØØØØØ      Expect=ØØØØØØØ1      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
R     Write-Protect Page ....................Running -----------> PASSED
```

9.19 WRITE-PROTECT SEGMENT MMU S

9.19.1 Description

This command tests the segment write-protect mechanism in the MMU. If the MMU is functioning correctly, then attempting a write to a protected segment will cause a bus error. The test has been implemented in the following manner:

Step 1: Set the WP bit in the descriptor for the first RAM segment.

Step 2: Enable the MMU.

Step 3: Attempt to write to page in the protected segment.

Step 4: If a bus error does not occur, then the test fails.

9.19.2 Command Input

```
135Diag> MMU S
```

9.19.3 Response/Messages

After entering this command, the display should read as follows:

```
S    Write-Protect Segment .................Running ----------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
S    Write-Protect Segment .................Running ----------->
     Addr=ØØØØØØØØ      Expect=ØØØØØØØ1      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
S    Write-Protect Segment .................Running -----------> PASSED
```

9.2Ø WRITE-ACCESS-LEVEL **MMU T**

9.2Ø.1 Description

This command is used to test the write access level mechanism of the MMU. Attempted writes to a segment protected in this manner should cause a bus error. For the **MMB851**, the test has been implemented as follows:

Step 1: Set the Write-Access-Level (WAL) bit in the descriptor for the first user segment.

Step 2: Enable the MMU.

Step 3: Attempt to write to the segment using User Data function code. This should cause a bus error.

Step 4: If a bus error does not occur, then the test fails.

For the **PMMU**, the test is implemented as follows:

Step 1: Eight access levels are enabled, therefore an initial shift of the upper 3 bits of the address is specified in the **TC** -*Translation Control*- register of the PMMU.

Step 2: The access level bits of the test address are set to the lowest priority (level 7).

Step 3: The **WAL** field is set up for levels 6 down to Ø while a write cycle is tried at the test address. A bus error should occur in each access.

9.2Ø.2 Command Input

```
135Diag> MMU T
```

9.2Ø.3 Response/Messages

After entering this command, the display should read as follows:

```
T     Write-Access-Level ....................Running ----------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
T     Write-Access-Level ....................Running ----------->
      Addr=ØØØØØØØØ      Expect=ØØØØØØØ1      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
T     Write-Access-Level .....................Running -----------> PASSED
```

9.21 READ-ACCESS-LEVEL **MMU U**

9.21.1 Description

This command is used to test the read access level mechanism of the MMU. Attempted reads to a segment protected in this manner should cause a bus error. For the **MMB851**, the test is implemented as follows:

Step 1: Set the Read-Access-Level (RAL) bit in the descriptor for the first user segment.

Step 2: Enable the MMU.

Step 3: Attempt to read to the segment using User Data function code. This should cause a bus error.

Step 4: If a bus error does not occur, then the test fails.

For the **PMMU**, the test is implemented as follows:

Step 1: Eight access levels are enabled, therefore an initial shift of the upper 3 bits of the address is specified in the **TC** -*Translation Control*- register of the PMMU.

Step 2: The access level bits of the test address are set to the lowest priority (level 7).

Step 3: The **RAL** field is set up for levels 6 down to Ø while a read cycle is tried at the test address. A bus error should occur in each access.

9.21.2 Command Input

```
135Diag> MMU U
```

9.21.3 Response/Messages

After entering this command, the display should read as follows:

```
U     Read-Access-Level ......................Running ----------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
U     Read-Access-Level ......................Running ----------->
      Addr=ØØØØØØØØ      Expect=ØØØØØØØ1      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
U     Read-Access-Level .....................Running -----------> PASSED
```

9.22 UPPER-LIMIT VIOLATION MMU V

9.22.1 Description

This command tests the capability of the MMU to detect when a logical address exceeds the upper limit of a segment. This condition is called an upper limit violation and should cause a bus error. The test has been implemented in the following manner:

Step 1: Modify the descriptor for a segment to lower the upper limit to where it permits access to only the first page.

Step 2: Attempt access to the second page.

Step 3: This should cause a bus error. If no bus error occurs, then the test fails.

9.22.2 Command Input

```
135Diag> MMU V
```

9.22.3 Response/Messages

After entering this command, the display should read as follows:

```
V     Upper-Limit Violation ..................Running ----------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
V     Upper-Limit Violation ..................Running ----------->
      Addr=ØØØØØØØØ       Expect=ØØØØØØØ1       Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
V     Upper-Limit Violation ..................Running -----------> PASSED
```

9.23 LOWER-LIMIT VIOLATION **MMU W**

9.23.1 Description

This command tests the capability of the MMU to detect when a logical address exceeds the lower limit of a segment. This condition is called an lower limit violation and should cause a bus error. The test has been implemented in the following manner:

Step 1: Modify the descriptor for a segment to set the lower limit to where it permits access to only the second page.

Step 2: Attempt access to the first page.

Step 3: This should cause a bus error. If no bus error occurs, then the test fails.

9.23.2 Command Input

```
135Diag> MMU W
```

9.23.3 Response/Messages

After entering this command, the display should read as follows:

```
W     Lower-Limit Violation .................Running ----------->
```

If the MMU does not cause the CPU to take a bus error exception, then the test fails.

```
W     Lower-Limit Violation .................Running ----------->
      Addr=00000000       Expect=00000001       Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the MMU does cause the CPU to take a bus error exception, then the test passes.

```
W     Lower-Limit Violation .................Running -----------> PASSED
```

9.24 PREFETCH ON INVALID-PAGE BOUNDARY MMU X

9.24.1 Description

This command tests to see if the MC68020 will ignore a bus error that occurs as the result of a prefetch. The MMU will signal a bus error if a prefetch operation crosses a page boundary into an invalid page. The MC68020 is to ignore such bus errors. The test has been implemented in the following manner:

Step 1: Invalidate the second page mapped to user Program space. This page is shared with User Data space.

Step 2: Insert a trap instruction at the last location of the previous page (this page is still valid).

Step 3: Point to a special trap handler that will check for the bus error by examing some flags.

Step 4: Enable the MMU.

Step 5: Branch to the address in the first page of the trap instruction, leaving Supervisor mode and entering User mode.

Step 6: The MC68020 should fetch the operating word at the end of the valid page, then attempt to prefetch the next word, which will cross the page boundary into the invalid page.

Step 7: If the MC68020 takes a bus error exception, then the test fails. Once bus error exception processing completes, control passes to the special trap handler.

Step 8: Once in the special trap handler, the stack is cleaned up (leaving the MC68020 in Supervisor mode), and a test is performed to determine if the MC68020 executed a bus error exception.

Step 9: If the bus error occurred, then the test fails.

9.24.2 Command Input

```
135Diag> MMU X
```

9.24.3 Response/Messages

After entering this command, the display should read as follows:

```
X     Prefetch On Inv-Page ...................Running ----------->
```

If a bus error occurs, then the test fails.

```
X     Prefetch On Inv-Page ..................Running ------------>
      Addr=ØØØØØØØØ      Expect=ØØØØØØØ1      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the prefetching does not cause the MC68Ø2Ø to take a bus error exception, then the test passes.

```
X     Prefetch On Inv-Page ..................Running ------------> PASSED
```

9.25 MODIFY-BIT AND INDEX **MMU Y**

9.25.1 Description

This command tests the capability of the MMU to set the Modify-Bit in a page descriptor of a page which has a index field greater than zero (not a page-Ø) when the page is written.

9.25.2 Command Input

```
135Diag> MMU Y
```

9.25.3 Response/Messages

After entering this command, the display should read as follows:

```
Y     Modify-Bit & Index ....................Running ----------->
```

If the Modify-bit does not get set by the write, then the test failed.

```
Y     Modify-Bit & Index ....................Running ----------->
      Addr=ØØFØØØØØ      Expect=ØØØØØØ1Ø      Read=ØØØØØØØØ
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the Modify-bit does get set, the test passed.

```
Y     Modify-Bit & Index ....................Running -----------> PASSED
```

9.26 SIXTEEN-BIT BUS **MMU Z Ø**

This command is used to run the following tests with the MMU set for 16-bit bus size. The command must be followed by a numeral indicating which sub-test is to be executed.

9.26.1 User-Program Space

9.26.1.1 Description

This command is used in conjunction with "MMU Z" to test the capability of the MMU to access User Program space in 16-bit mode.

9.26.1.2 Command Input

```
135Diag> MMU Z Ø
```

9.26.1.3 Response/Messages

After entering this command, the display should read as follows:

```
Ø     User-Prog Space ........................Running ----------->
```

The conditions determining the success of this test are fully described in section 9.11, as will as the error messages. If the test fails, the display will appear as shown:

```
Ø     User-Prog Space ........................Running ----------->
      Addr=ØØØØØØØØ      Expect=ØØØØØØØ1      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the test passes, then the display will appear as follows:

```
Ø     User-Prog Space ........................Running -----------> PASSED
```

9

9.26.2 PAGE-DESC MODIFY-BIT — MMU Z 1

9.26.2.1 Description

This command is used in conjunction with "MMU Z" to test the ability of the MMU to set the Modify bit in a page descriptor when the page gets written to. This test operates exactly like the test described in section 9.14; the only difference is that the MMU is set to 16-bit mode before executing the test described in that section.

9.26.2.2 Command Input

```
135Diag> MMU Z 1
```

9.26.2.3 Response/Messages

After entering this command, the display should read as follows:

```
1      Page-Desc Modify-Bit ..................Running ----------->
```

The conditions determining the success of this test are fully described in section 9.14, as will as the error messages. If the test fails, the display will appear as shown:

```
1      Page-Desc Modify-Bit ..................Running ----------->
       Addr=00000000       Expect=00000001       Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the test passes, then the display will appear as follows:

```
1      Page-Desc Modify-Bit ..................Running -----------> PASSED
```

9.26.3 INDIRECT PAGE MMU Z 2

9.26.3.1 Description

This command is used in conjunction with "MMU Z" to handle an indirect descriptor. This test operates exactly like the test described in section 9.12; the only difference is that the MMU is set to 16-bit mode before executing the test described in that section.

9.26.3.2 Command Input

```
135Diag> MMU Z 2
```

9.26.3.3 Response/Messages

After entering this command, the display should read as follows:

```
2     Indirect Page ..........................Running ----------->
```

The conditions determining the success of this test are fully described in section 9.12, as will as the error messages. If the test fails, the display will appear as shown:

```
2     Indirect Page ..........................Running ----------->
      Addr=00000000      Expect=00000001      Read=FA445221
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the test passes, then the display will appear as follows:

```
2     Indirect Page ..........................Running -----------> PASSED
```

9.27 READ/MODIFY/WRITE CYCLE — MMU Ø

9.27.1 Description

This tests performs the Test-And-Set instruction in three modes to verify that the MMU functions correctly during read/modify/write cycles.

The message "hit page" is displayed. The MMU is turned on and a write access is performed to cache the address translation for that location. The first TAS is then done to verify that a hit page can be accessed. No bus error should result from this. The test will fail if either a bus error occurs or the location (in RAM) written to does not contain $8Ø afterward.

The MMU is shut off and the message "missed page" displayed. The MMU is turned on, flushing the Address Translation Cache (ATC). A TAS is then attempted. As the ATC was just flushed, the access should cause a table walk and a single bus error. An error is declared if other than on bus error occurred.

If the two previous phases completed successfully, the message "unmodified page" is displayed and the final phase begun. The Modified bit for a particular page is cleared, then a read from that page is performed to cache its address translation. Next, a TAS is attempted to that location. A bus error should occur to allow the MMU time to set the Modified bit. An error is declared if the Modified bit was not set or if other than one bus error occurred.

9.27.2 Command Input

```
135Diag> MMU Ø
```

9.27.3 Response/Messages

After entering this command, the display should read as follows:

```
Ø     R/M/W Cycles ..........................Running ----------->
                    Hit page .........
```

If the access to the "hit page" causes a bus error, or the location written to does not contain $8Ø, then the test fails and the table walk is displayed.

```
Ø     R/M/W Cycles ...........................Running ------------>
                  Hit page .........
      Addr=XXXXXXXX      Expect=8ØØØØØØØ      Read=ØØØØØØØØ
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the access to the "missed page" does not cause a bus error, then the test fails and the table walk is displayed.

```
Ø     R/M/W Cycles ...........................Running ------------>
                  Hit page .........
                  Missed page ......
      Addr=XXXXXXXX      Expect=8ØØØØØØØ      Read=ØØØØØØØØ
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If the access to the "modified page" does not cause a bus error, or the Modified bit for the page does not get set, then the test fails and the table walk is displayed.

```
Ø     R/M/W Cycles ...........................Running ------------>
                  Hit page .........
                  Missed page ......
                  Modified page ....
      Addr=XXXXXXXX      Expect=8ØØØØØØØ      Read=ØØØØØØØØ
(map of table walk displayed here, refer to section 9.28)
 .... FAILED
```

If all three phases of the test complete successfully, then the test passes and the display will appear as follows:

```
Ø     R/M/W Cycles ...........................Running ------------>
                  Hit page .........
                  Missed page ......
                  Modified page .... PASSED
```

9.28 TABLE WALK DISPLAY FORMAT

Many of the MMU tests will display the supposed path through the translation table upon encountering an error. This section explains the format used to display that path and the meaning of the values shown. Figure 9-1 and Table 9-2 illustrate and describe a sample table walk display.

```
RP=ØØØØØØØØ   TC=11111111
    ----V-- Fc ------      ------ SegØ -----      ----- Seg1 ------
    22222222 33333333 --> 44444444 55555555 --> 66666666 77777777 ---+
                                                                      V
                                                          Page =  88888888
(shown only if previous page desc is an indirect)         Page =  99999999
```

FIGURE 9-1. SAMPLE TABLE WALK DISPLAY

TABLE 9-2. SAMPLE TABLE WALK DISPLAY

Value	Description
ØØØØØØØØ	Root Pointer register contents, address of Function Code Table.
11111111	Translation Control register contents.
22222222	Function Code Table entry status longword.
33333333	Function Code Table entry, address of Segment Ø Table.
44444444	Segment Ø Table entry, status longword.
55555555	Segment Ø Table entry, address of Segment 1 Table.
66666666	Segment 1 Table entry, status longword.
77777777	Segment 1 Table entry, address of Page descriptor.
88888888	Page descriptor longword. Can be indirect.
99999999	Page descriptor longword. Shown only if previous one is indirect.

For further information as to the meaning of these values, refer to the MMU User's Manual.

CHAPTER 1Ø
RAM TESTS

1Ø.1 GENERAL DESCRIPTION

This set of tests access random access memory (read/write) that may either reside on the MVME135 or on a MVME2Ø4 RAM board. Table 1Ø-1 lists the memory diagnostic tests.

TABLE 1Ø-1. MEMORY DIAGNOSTIC TESTS

Monitor Command	Title	Section
MT A	Set Function Code	1Ø.3
MT B	Set Starting Address	1Ø.4
MT C	Set Stop Address	1Ø.5
MT D	Set VMEbus Width	1Ø.6
MT E	March Address Test	1Ø.7
MT F	Walk A Bit Test	1Ø.8
MT G	Refresh Test	1Ø.9
MT H	Random Pattern Test	1Ø.1Ø
MT I	Program Test	1Ø.11
MT J	TAS Test	1Ø.12
MT K	Address Complement Test	1Ø.13
MT L	Byte/Word/Long Permutations Test	1Ø.14

1Ø.2 HARDWARE CONFIGURATION

The following hardware is required to perform these tests.

MVME135 CPU - Board being tested.
MVME2Ø4 - RAM board (may or may not be present).
Video Display Terminal.

1Ø.3 SET FUNCTION CODE MT A

1Ø.3.1 Description

This command allows the user to select the function code used in most of the memory tests. The exceptions to this are "TAS Test" and "Program Test".

1Ø.3.2 Command Input

```
135Diag> MT A [new value] < CR>
```

1Ø.3.3 Response/Messages

If the user supplied the optional new value, then the display will appear as follows:

```
135Diag>MT A [new value]
FUNCTION CODE=<new value>
135Diag>
```

If a new value was not specified by the user, then the old value will be displayed and the user will be allowed to enter a new value.

```
135Diag>MT A
FUNCTION CODE=<current value> ? [new value]<CR>
FUNCTION CODE=<new value>
135Diag>
```

This command may be used to display the current value without changing it by pressing **RETURN** without entering the new value.

```
135Diag>MT A
FUNCTION CODE=<current value> ? <CR>
FUNCTION CODE=<current value>
135Diag>
```

10.4 SET STARTING ADDRESS MT B

10.4.1 Description

This command allows the user to select the starting address used by all of the memory tests. Approximately 16K bytes of RAM is used by the MVME135 for stack and workspace. It is suggested that addresses in this area not be used for the starting address. Refer to the "Memory Requirements" section of the 135Bug User's Manual for more information concerning allocation of RAM by 135Bug.

10.4.2 Command Input

```
135Diag> MT B [new value] < CR>
```

10.4.3 Response/Messages

If the user supplied the optional new value, then the display will appear as follows:

```
135Diag>MT B [new value]
Start Addr=<new value>
135Diag>
```

If a new value was not specified by the user, then the old value will be displayed and the user will be allowed to enter a new value.

```
135Diag>MT B
Start Addr=<current value> ? [new value]<CR>
Start Addr=<new value>
135Diag>
```

This command may be used to display the current value without changing it by pressing **RETURN** without entering the new value.

```
135Diag>MT B
Start Addr=<current value> ? <CR>
Start Addr=<current value>
135Diag>
```

NOTE: If a new value is specified, it will be truncated to a longword boundary. If the new starting address is greater than the value of the stop address, the stop address will be ignored and the memory test(s) will test only one location at the address specified by the starting address. The start address is never allowed to be higher in memory than the stop address during memory test execution.

10.5 SET STOP ADDRESS MT C

10.5.1 Description

This command allows the user to select the stop address used by all of the memory tests.

10.5.2 Command Input

```
135Diag> MT C [new value] < CR>
```

10.5.3 Response/Messages

If the user supplied the optional new value, then the display will appear as follows:

```
135Diag>MT C [new value]
Stop Addr=<new value>
135Diag>
```

If a new value was not specified by the user, then the old value will be displayed and the user will be allowed to enter a new value.

```
135Diag>MT C
Stop Addr=<current value> ? [new value]<CR>
Stop Addr=<new value>
135Diag>
```

This command may be used to display the current value without changing it by pressing **RETURN** without entering the new value.

```
135Diag>MT C
Stop Addr=<current value> ? <CR>
Stop Addr=<current value>
135Diag>
```

NOTE: If a new value is specified, it will be truncated to a longword boundary. If the new stop address is smaller than the value of the starting address, the stop address will be ignored and the memory test(s) will test only one location at the address specified by the starting address. The stop address is never allowed to be lower in memory than the starting address during memory test execution.

10.6 SET VMEbus WIDTH — MT D

10.6.1 Description

This command is used to select either 16- or 32-bit VMEbus accesses during the MVME135 **MT** memory tests. The width is selected by entering zero for 16 bits or one for 32 bits.

10.6.2 Command Input

```
135Diag> MT D [new value: Ø for 16, 1 for 32] < CR>
```

10.6.3 Response/Messages

If the user supplied the optional new value, then the display will appear as follows:

```
135Diag>MT D [new value]
VMEbus (32=1/16=Ø) Width=<new value>
135Diag>
```

If a new value was not specified by the user, then the old value will be displayed and the user will be allowed to enter a new value.

```
135Diag>MT D
VMEbus (32=1/16=Ø) Width=<current> ? [new value]<CR>
VMEbus (32=1/16=Ø) Width=<new value>
135Diag>
```

This command may be used to display the current value without changing it by pressing **RETURN** without entering the new value.

```
135Diag>MT D
VMEbus (32=1/16=Ø) Width=<current> ? <CR>
VMEbus (32=1/16=Ø) Width=<new value>
135Diag>
```

1Ø.7 MARCH ADDRESS TEST MT E

1Ø.7.1 Description

This command performs a "march address" test from "STARTING ADDRESS" to "STOP ADDRESS". The march address test has been implemented in the following manner:

Step 1: All memory locations from STARTING ADDRESS up to STOP ADDRESS are cleared to Ø.

Step 2: Beginning at STOP ADDRESS and proceeding downward to STARTING ADDRESS, each memory location is checked for bits that did not clear and then the contents are changed to all F's (all the bits are set). This process will reveal address lines that are stuck high.

Step 3: Beginning at STARTING ADDRESS and proceeding upward to STOP ADDRESS, each memory location is checked for bits that did not set and then the memory location is again cleared to Ø. This process will reveal address lines that are struck low.

1Ø.7.2 Command Input

```
135Diag> MT E < CR>
```

1Ø.7.3 Response/Messages

After entering this command, the display should read as follows:

```
E     March Addr Test ........................Running ----------->
```

If an error is encountered, then the memory location and other related information will be displayed (refer to section 1Ø.15).

```
E     March Addr Test ........................Running ----------->
(error-related information)
 .... FAILED
```

If no errors are encountered, then the display will appear as follows:

```
E     March Addr Test ........................Running -----------> PASSED
```

10.8 WALK A BIT TEST MT F

10.8.1 Description

This command performs a "walking bit" test from "STARTING ADDRESS" to "STOP ADDRESS". The walking bit test has been implemented in the following manner:

Step 1: For each memory location, do the following:

a. Write out a 32-bit value with only the lower bit set.

b. Read it back and verify that the value written equals the one read. Report any errors.

c. Shift the 32-bit value to move the bit up one position.

d. Repeat the procedure (write, read, and verify) for all 32-bit positions.

10.8.2 Command Input

```
135Diag> MT F < CR>
```

10.8.3 Response/Messages

After entering this command, the display should read as follows:

```
F     Walk a Bit Test .......................Running ----------->
```

If an error is encountered, then the memory location and other related information will be displayed (refer to section 10.15).

```
F     Walk a Bit Test .......................Running ----------->
(error-related information)
 .... FAILED
```

If no errors are encountered, then the display will appear as follows:

```
F     Walk a Bit Test .......................Running -----------> PASSED
```

10

1Ø.9 REFRESH TEST MT G

1Ø.9.1 Description

This command performs a refresh test from "STARTING ADDRESS" to "STOP ADDRESS". This test fills the entire board with a pseudo-random pattern, delays for approximately 4Ø seconds, then examines each memory location, regenerating the random value previously written to it. The delay allows for the effects of faulty dynamic RAM refreshing to become evident. An error is declared when the data read from the target location differs from the data written to it.

1Ø.9.2 Command Input

```
135Diag> MT G < CR>
```

1Ø.9.3 Response/Messages

After entering this command, the display should read as follows:

```
G     Refresh ...............................Running ----------->
```

If an error is encountered, then the memory location and other related information will be displayed (refer to section 1Ø.15).

```
G     Refresh ...............................Running ----------->
(error-related information)
 .... FAILED
```

If no errors are encountered, then the display will appear as follows:

```
G     Refresh ...............................Running -----------> PASSED
```

1Ø.1Ø RANDOM PATTERN TEST — MT H

1Ø.1Ø.1 Description

This command performs a "random byte" test from "STARTING ADDRESS" to "STOP ADDRESS". The random pattern test has been implemented in the following manner:

Step 1: A register is loaded with the value $ECA8642Ø.

Step 2: For each memory location:

a. Copy the contents of the register to the memory location, one byte at a time.

b. Add $Ø2468ACE to the contents of the register.

c. Proceed to next memory location.

Step 3: Reload $ECA8642Ø into the register.

Step 4: For each memory location:

a. Compare the contents of the memory to the register to verify that the contents are good, one byte at a time.

b. Add $Ø2468ACE to the contents of the register.

c. Proceed to next memory location.

1Ø.1Ø.2 Command Input

```
135Diag> MT H < CR>
```

1Ø.1Ø.3 Response/Messages

After entering this command, the display should read as follows:

```
H    Random Pattern ........................Running ----------->
```

If an error is encountered, then the memory location and other related information will be displayed (refer to section 1Ø.15).

```
H    Random Pattern ........................Running ----------->
(error-related information)
 .... FAILED
```

If no errors are encountered, then the display will appear as follows:

```
H     Random Pattern ........................Running ------------> PASSED
```

10.11 PROGRAM TEST **MT I**

10.11.1 Description

This command moves a program segment into RAM and executes it. The implementation of this is as follows:

Step 1: The program is moved into the RAM, repeating it as many times as necessary to fill the available RAM (i.e., from "STARTING ADDRESS" to "STOP ADDRESS"-8).

> **NOTE:** Only complete segments of the program are moved. The space remaining from the last program segment copied into the RAM to "STOP ADDRESS"-8 is filled with NOP instructions. Attempting to run this test without sufficient memory (around 400 bytes) for at least one complete program segment to be copied will cause the error message "INSUFFICIENT MEMORY" to be printed.

Step 2: The last location, "STOP ADDRESS" receives an RTS instruction.

Step 3: Finally, the test performs a JSR to location "STARTING ADDRESS".

Step 4: The program itself performs a wide variety of operations, with the results frequently being checked and a count of the errors maintained. Errant locations are reported in the same fashion as any memory test failure (refer to section 10.15).

10.11.2 Command Input

```
135Diag> MT I
```

10.11.3 Response/Messages

After entering this command, the display should read as follows:

```
I     Program Test ..........................Running ----------->
```

If the operator has not allowed enough memory for at least one program segment to be copied into the target RAM, then the following error message will be printed. To avoid this, make sure that the Stop Address is at least 388 bytes ($00000184) greater that the Start Address.

```
I     Program Test ..........................Running ----------->
      Insufficient Memory PASSED
```

If the program (in RAM) detects any errors, then the location of the error and other information will be displayed (refer to section 1Ø.13).

```
I     Program Test ..........................Running ----------->
(error-related information)
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
I     Program Test ..........................Running -----------> PASSED
```

1Ø.12 TAS TEST MT J

1Ø.12.1 Description

This command performs a Test And Set (TAS) test from "STARTING ADDRESS" to "STOP ADDRESS". The implementation of this is as follows:

Step 1: For each memory location:

a. Clear the memory location to Ø.

b. "Test And Set" the location (should set upper bits only).

c. Verify that the location now contains $8Ø.

d. Proceed to next location (next byte).

1Ø.12.2 Command Input

```
135Diag> MT J
```

1Ø.12.3 Response/Messages

After entering this command, the display should read as follows:

```
J     TAS Test ..............................Running ----------->
```

If an error occurs, then the memory location and other information are displayed (refer to section 1Ø.15).

```
J     TAS Test ..............................Running ----------->
(error-related information)
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
J     TAS Test ..............................Running -----------> PASSED
```

10.13 ADDRESS COMPLEMENT TEST MT K

10.13.1 Description

This command performs an address complement test from "STARTING ADDRESS" to "STOP ADDRESS". The test creates a noisy environment by switching all address and data lines between memory accesses. The implementation of this is as follows:

Step 1: Write $AAAAAAAA to "STARTING ADDRESS".

Step 2: Write $55555555 to "STOP ADDRESS".

Step 3: Work in toward the middle of the test memory, alternately writing $AAAAAAAA to the top half and $55555555 to the bottom half.

Step 4: Once the middle of the test memory space is reached, the write portion of the test is finished.

Step 5: Check the written locations for the expected data.

10.13.2 Command Input

```
135Diag> MT K
```

10.13.3 Response/Messages

After entering this command, the display should read as follows:

```
K     Addr. Complement ......................Running ----------->
```

If an error occurs, then the memory location and other information are displayed (refer to section 10.15).

```
K     Addr. Complement ......................Running ----------->
(error-related information)
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
K     Addr. Complement ......................Running -----------> PASSED
```

10.14 BYTE/WORD/LONG PERMUTATIONS TEST MT L

10.14.1 Description

This command performs a test from "STARTING ADDRESS" to "STOP ADDRESS" which verifies that the memory can accommodate byte, word and longword writes and reads in any combination. The test also verifies that the bytes, words and longwords are organized in memory in a way consistent with the MC68000-family architecture. The implementation of this is as follows:

Step 1: Load a test pattern of $01234567 into a data register.

Step 2: Begin at "STARTING ADDRESS". Write the data as a longword. Read the data back as four bytes, two words and then as a longword. Compare with original data pattern in each case.

Step 3: Write the data as a pair of sixteen-bit words. Read the data back as four bytes, two words and then as a longword. Compare with original data pattern in each case.

Step 4: Write the data as four bytes. Read the data back as four bytes, two words and then as a longword. Compare with original data pattern in each case.

Step 5: Repeat the sequence of writes and reads every 260 bytes, until "STOP ADDRESS" is reached.

10.14.2 Command Input

```
135Diag> MT L
```

10.14.3 Response/Messages

After entering this command, the display should read as follows:

```
L     B/W/L Permutations ....................Running ----------->
```

If an error occurs, then the memory location and other information are displayed (refer to section 10.15).

```
L     B/W/L Permutations ....................Running ----------->
(error-related information)
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
L     B/W/L Permutations ....................Running -----------> PASSED
```

1Ø.15 MEMORY ERROR DISPLAY FORMAT

This section is included to describe the format used to display errors during memory tests E to L.

The error reporting code is designed to conform to two rules:

1. The first time an error occurs, headings are printed out prior to the printing of the values.
2. Upon 2Ø memory errors, the printing of error messages ceases for the remainder of the test.

Figure 1Ø-1 is included as an example of the display format.

```
FC  TEST ADDR    1Ø987654321Ø987654321Ø987654321Ø  EXPECTED  READ
 5  ØØØ1ØØØØ     -----------------------X--------  ØØØØØ1ØØ  ØØØØØØØØ
 5  ØØØ1ØØØ4     -------------------X-------X----  FFFFEFFF  FFFFFEFF
```

FIGURE 1Ø-1. SAMPLE MEMORY ERROR DISPLAY

Each line displayed consists of five items: function code, test address, graphic bit report, expected data, and read data. The test address, expected data, and read data are displayed in hexadecimal. The graphic bit report shows a letter "X" at each errant bit position and a dash ("-") at each "good" bit position.

The heading used for the graphic bit report is intended to make the bit position easy to determine. Each numeral in the heading is the one's digit of the bit position. For example, the leftmost "bad" bit at test address $1ØØØ4 has the numeral 2 over it since this is the second "2" from the right, the bit position is read "12" (base 1Ø).

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 11
MC68681 DUART (SIO) TESTS

11.1 GENERAL DESCRIPTION

This section details the diagnostics needed to test the DUART device and its associated circuitry. Table 11-1 lists the serial I/O port diagnostic tests.

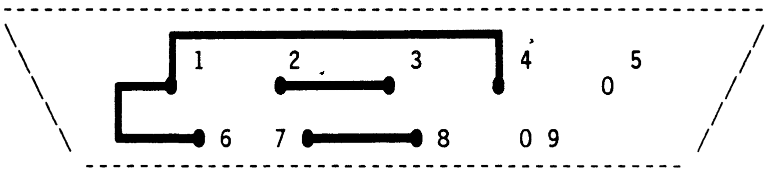
TABLE 11-1. MC68681 DUART (SIO) TESTS

Monitor Command	Title	Section
SIO A	Data Formats Test	11.3
SIO B	Baud Rate Test	11.4
SIO C	TX/RX Ready IRQ Test	11.5
SIO D	TX/RX Ready Test	11.6
SIO E	FIFO Full Test	11.7
SIO F	BREAK Test	11.8
SIO G	SIO Timer Test	11.9
SIO H	Normal Mode Test	11.1Ø
SIO I	Handshake Line Test	11.11

11.2 HARDWARE CONFIGURATION

The following hardware is required to perform these tests.

MVME135 CPU - Board being tested.
Video Display Terminal.
Male RS-232 loop back connector (see Figure 11-1 for configuration).



Connect	
From	**To**
2 RD	3 TD
7 RTS	8 CTS
1 DCD	4 DTR
4 DTR	6 DSR

FIGURE 11-1. LOOP BACK CONNECTOR WIRING DIAGRAM

11.3 DATA FORMATS TEST SIO A

11.3.1 Description

This test ensures that data can be sent and received in all formats at 96ØØ baud. The algorithm for this test is as follows:

```
For not parity, even parity, odd parity:
   For one stop bit, two stop bits:
      For 5-, 6-, 7-, 8-bit data widths:

        Send/Receive full data range ($ØØ-$FF).

      Next data width.
   Next number of stop bits.
Next parity mode
```

11.3.2 Command Input

```
135Diag> SIO A
```

11.3.3 Response/Messages

The monitor will respond with:

```
A     Data Formats Test ......................Running ----------->
```

Progress of the test will be output as the different steps of the test runs.

If no errors occur, then the display will appear as follows:

```
A     Data Formats Test ......................Running ----------->
Parity : None
One Stop Bit ... Two Stop Bits ...
Parity : Even
One Stop Bit ... Two Stop Bits ...
Parity : Odd
One Stop Bit ... Two Stop Bits ... PASSED
```

11.4 BAUD RATE TEST — SIO B

11.4.1 Description

This test verifies that data can be transmitted at all set 2 baud rates (refer to the MC68681 DUART Specification). A set of complement characters is transmitted at each baud rate on port 2 in local loopback mode. Only channel B is tested.

11.4.2 Command Input

```
135Diag> SIO B
```

11.4.3 Response/Messages

The monitor will respond with:

```
B     Baud Rate Test ........................Running ------------>
```

If no errors occur, then the display will appear as follows:

```
B     Baud Rate Test ........................Running ------------> PASSED
```

11.5 TX/RX READY IRQ TEST SIO C

11.5.1 Description

This test verifies that the DUART interrupt circuitry is working properly and that the MVME135 can properly respond to a DUART IRQ. This test is performed in the internal loopback mode and tests the TXRDY/RXRDY IRQ functions for each port. The test is performed on port 2 only and generates interrupts for both TXRDY and RXRDY. Four characters are used during this test, $55, $AA, $FF and $ØØ. The DUART IRQ generates a level 5 auto-vectored IRQ to the CPU, which should receive 8 IRQ's per port during this test (4 for TX, 4 for RX).

11.5.2 Command Input

```
135Diag> SIO C
```

11.5.3 Response/Messages

The monitor will respond with:

```
C     TX/RX Ready IRQ Test ..................Running ------------>
```

If no errors occur, then the display will appear as follows:

```
C     TX/RX Ready IRQ Test ..................Running ------------> PASSED
```

Example Failure Messages:

```
- DATA COMPARE ERROR ON PORT 1 -
          (Received data was not what was expected)

- RX FORMAT ERROR ON PORT x ! STATUS RECEIVED=xxxxxxxx (binary)
          (Indicates defective port in DUART, bad status on receive)

- TX IRQ TIMEOUT ON PORT x !
          (No TX IRQ Action detected, probably no IRQ generated)

- RX IRQ TIMEOUT ON PORT x !
          (TXRDY generated IRQs but RXRDY did not; defective DUART)

- IRQ COUNT = $x -
          (Wrong number of IRQs detected; bad DUART or noisy IRQ line)

- RX STATUS ERROR MASK = xxxxxxxx

- RX DATA ERROR MASK = xxxxxxxx
- RX PORT ERROR MASK = xxxxxxxx
        (Displayed at end of test if any of these errors
         occurred on any port, bit Ø = port 1, bit 1 = port 2)
```

11.6 TX/RX READY TEST SIO D

11.6.1 Description

This test checks the operation of both the TXRDY and TXRDY status bits of the DUART's port status registers and ensures that they reflect the proper status and that the interrupt status register bits match. This test is performed in the internal loopback mode. For port 2 the transmitter and receiver are enabled. Two characters ($55 and $AA) are transmitted with the appropriate status checks both before and after transmission, before and after reception and when reading the character to verify both states of the status lines. If errors are detected, messages will indicate the failed function and the port number.

11.6.2 Command Input

```
135Diag> SIO D
```

11.6.3 Response/Messages

The monitor will respond with:

```
D     TX/RX Ready Test .......................Running ------------>
```

If no errors occur, then the display will appear as follows:

```
D     TX/RX Ready Test .......................Running ------------> PASSED
```

11.7 FIFO FULL TEST SIO E

11.7.1 Description

This test checks the FIFO full status bit function in the port status register and in the interrupt status register of the DUART. This test is performed in the internal loopback mode. For port 2 three characters are transmitted and FIFO full status is checked to be active in both registers. A fourth character is transmitted to fill the holding register. After the transmitter is empty, the first character transmitted is read and verified and the status bits are verified as still active. The second character is then read and verified and status is checked to be inactive (FIFO not full). The the received characters are read and verified.

11.7.2 Command Input

```
135Diag> SIO E
```

11.7.3 Response/Messages

The monitor will respond with:

```
E     FIFO Full Test ........................Running ----------->
```

If no errors occur, then the display will appear as follows:

```
E     FIFO Full Test ........................Running -----------> PASSED
```

11.8 BREAK TEST SIO F

11.8.1 Description

This test checks the BREAK generation and detection functions of the DUART. It is performed in the internal loopback mode for port 2. A START BREAK command is issued to the port and status is checked in the port status register for BREAK DETECTED and in the interrupt status register for CHANGE-IN-BREAK. Status is cleared with the RESET ERROR STATUS and the RESET BREAK CHANGE INTERRUPT commands. A STOP BREAK command is then issued and the ISR is checked to have detected the change in BREAK.

11.8.2 Command Input

```
135Diag> SIO F
```

11.8.3 Response/Messages

The monitor will respond with:

```
F     BREAK Test ............................Running ------------>
```

If no errors occur, then the display will appear as follows:

```
F     BREAK Test ............................Running ------------> PASSED
```

11.9 SIO TIMER TEST SIO G

11.9.1 Description

This test checks the TIMER function of the DUART. It verifies loadability and readability of the counter/timer registers by writing a test value into the register, starting and stopping the counter, and then verifying the value (masking the least-significant nibble). The counter is then allowed to timeout and the Counter/Timer Ready bit in the interrupt status register is checked to be active. A STOP COUNTER command is issued and the C/TR bit is checked to be inactive.

11.9.2 Command Input

```
135Diag> SIO G
```

11.9.3 Response/Messages

The monitor will respond with:

```
G     SIO Timer Test ........................Running ------------>
```

If no errors occur, then the display will appear as follows:

```
G     SIO Timer Test ........................Running ------------> PASSED
```

11.1Ø EXTERNAL TX/RX TEST SIO H

11.1Ø.1 Description

This test is the first "external" DUART test. The full 8-bit character set is transmitted through the external loopback cable assembly on port 2. The characters are transmitted and received at 19.2K baud. The test is the first check of the RS232 drivers on the board. All checks are done at the receiving side of the port and error messages are displayed accordingly even though the transmitting port (or driver/receiver) may be at fault. Only the receive and transmit lines are checked in this test. The external loopback connector is required on the port 2 card edge connector for the test to pass.

11.1Ø.2 Command Input

```
135Diag> SIO H
```

11.1Ø.3 Response/Messages

The monitor will respond with:

```
H     External TX/RX Test ....................Running ------------>
```

If no errors occur, then the display will appear as follows:

```
H     External TX/RX Test ....................Running ------------> PASSED
```

11.11 HANDSHAKE LINE TEST SIO I

11.11.1 Description

This test checks for proper assertion and detection of the RS-232 handshake line signals implemented on the MVME135. A special loopback connector is required to run the test successfully. This connector routes the handshake output lines back to the corresponding handshake inputs on the DUART. The level detection as well as the change-of-state detection functions of the DUART are tested. The basic algorithm is as follows:

Step 1: All handshake outputs are negated and change-of-state status is cleared. The handshake line inputs are verified to be inactive and change-of-state status is verified to be clear.

Step 2: All handshake outputs are asserted. The handshake line inputs are verified to be active and change-of-state status is verified to be "true" for the appropriate inputs.

Step 3: All handshake outputs are negated. The handshake line inputs are verified to be inactive and change-of-state status is verified to be "true" for the appropriate inputs.

11.11.2 Command Input

```
135Diag> SIO I
```

11.11.3 Response/Messages

The monitor will respond with:

```
I     Handshake Line Test ...................Running ----------->
```

If no errors occur, then the display will appear as follows:

```
I     Handshake Line Test ...................Running ----------->
Inactive/No Change-of-State -
Active/Change-of-State -
Inactive/Change-of-State - PASSED
```

CHAPTER 12
DYNAMIC RAM MEZZANINE BOARD TESTS

12.1 GENERAL DESCRIPTION

This section details the diagnostics supplied for testing the DRAM which is physically located on the mezzanine board attached to the MVME135/136. The same diagnostic tests are used for both the 1-Megabyte mezzanine board (MVME135/136) and for the 4-megabyte mezzanine board (MVME135A/136A). The tests will be able to determine the size of the mezzanine present. Table 12-1 lists the DRAM mezzanine board diagnostic tests.

TABLE 12-1. DYNAMIC RAM MEZZANINE BOARD TESTS

Monitor Command	Title	Section
MEZZ A	Row March Test	12.3
MEZZ B	March Address Test	12.4
MEZZ C	Walk a bit Test	12.5
MEZZ D	Random Pattern Test	12.6
MEZZ E	Refresh Test	12.7
MEZZ F	Program Test	12.8
MEZZ G	TAS Test	12.9
MEZZ H	Address Complement Test	12.10
MEZZ I	Byte/Word/Long Permutations Test	12.11
MEZZ J	Write/Read CSR Test	12.12
MEZZ K	Write-Wrong Parity Test	12.13
MEZZ L	Parity March Test	12.14

Most of these tests do not terminate when an error is encountered. Normally a test will run to completion and any errors are logged for display at the end of the test. Refer to section 12.15 for details of the error map display.

The Self Test command sequence (invoked by ST MEZZ) consists of tests MEZZ A through MEZZ J. Two tests, MEZZ K and MEZZ L, are not included in the self-test sequence but must be invoked explicitly from the command line. These two tests are excluded from the self-test sequence because they require special jumper positioning (refer to section 12.2 below).

NOTE: All MEZZ tests will work in the same chassis configuration as the bus tests, FAT tests, etc. Since in this configuration the unit under test uses the on-board DRAM for stack, vector table and variables, the mezzanine memory tests only test a part of the mezzanine RAM and do not disturb 135Bug's space. The starting address for all mezzanine RAM tests is defined to be just past the area used by 135Bug. Test address ranges are from $FFE03000 to $FFEFFFFF for 1-megabyte mezzanine boards and from $FF803000 to $FFBFFFFF for 4-megabyte mezzanine boards.

12.2 HARDWARE CONFIGURATION

The following hardware is required to perform these tests.

MVME135 CPU - Unit under test (UUT)
Video Display Terminal.

Switch settings for the MVME135/136 under test:

S3:	OFF: 1, 2	ON: 3, 4, 5, 6, 7, 8
S4:	OFF: None	ON: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Special configuration for parity tests:

Jumper block J6 must be jumpered from pins 2 to 3 before attempting to run either of the two parity tests (MEZZ K and MEZZ L). Other MEZZ tests do not require specific positioning of this jumper. Refer to section 2.3.1.4 of the MVME135/136 Multiprocessor User's Manual for a more thorough explanation of jumper J6's effect on the operation of the parity circuitry.

12.3 ROW MARCH TEST MEZZ A

12.3.1 Description

This first test is a quick check of the mezzanine board to verify that every row of the memory devices is uniquely addressable. The first location written to is $3ØØØ past the beginning address of the mezzanine, in order to skip over 135Bug's work space. The next location written to is the first address plus $ØØØØØ1Ø4. The last address written to is the last longword of the mezzanine (i.e., $FFEFFFFF for a 1-megabyte mezzanine).

Three passes are made through the test address range. During the first pass, a background pattern of zeros is written to each target address. On the second pass, the target address is checked to contain zero, then, if so, it is written to $FFFFFFFF. On the third pass, the target address is checked to contain $FFFFFFFF. Redundant addressing will cause a write to one address to put the same data at another address also. An error is declared when the data read from a location differs from the data previously written to it.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

12.3.2 Command Input

```
135Diag> MEZZ A
```

12.3.3 Response/Messages

After entering this command, the following should be printed out:

```
A      Row March .............................Running ----------->
```

If an error occurs, a map of the mezzanine board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 12.15.

```
A      Row March .............................Running ----------->
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
A     Row March .............................Running -----------> PASSED
```

12.4 MARCH ADDRESS TEST MEZZ B

12.4.1 Description

This test verifies that every longword on the mezzanine is uniquely addressable. The first location written to is $3ØØØ past the beginning address of the mezzanine, in order to skip over 135Bug's work space. The next location written to is the first address plus four. The last address written to is the last longword of the mezzanine (i.e., $FFEFFFFF for a 1-megabyte mezzanine).

Three passes are made through the test address range. During the first pass, a background pattern of zeros is written to each target address. On the second pass, the target address is checked to contain zero, then, if so, it is written to $FFFFFFFF. On the third pass, the target address is checked to contain $FFFFFFFF. Redundant addressing will cause a write to one address to put the same data at another address also. An error is declared when the data read from a location differs from the data previously written to it.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

12.4.2 Command Input

```
135Diag> MEZZ B
```

12.4.3 Response/Messages

After entering this command, the following should be printed out:

```
B     March Addr. Test .......................Running ----------->
```

If an error occurs, a map of the mezzanine board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 12.15.

```
B     March Addr. Test .......................Running ----------->
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
B     March Addr. Test .......................Running ------------> PASSED
```

12.5 WALK A BIT TEST MEZZ C

12.5.1 Description

This test checks the data lines by "walking" a bit through several locations. The first pattern written is $00000001. $00000002 is next and is followed by $00000004. The last pattern written is $80000000. The first location written to is $3000 past the beginning address of the mezzanine, in order to skip over 135Bug's work space. The next location written to is the first address plus $00000104. This value is repeatedly added to the target address to exercise the address lines. An error is declared when the data read from a location differs from the data just written to it. The test completes when the target address no longer falls in the range assigned to the board (i.e., for a 1-megabyte mezzanine this is $FFE03000 to $FFEFFFFF).

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.10 of this manual for instructions on invoking the loop-on-error mode.

12.5.2 Command Input

```
135Diag> MEZZ C
```

12.5.3 Response/Messages

After entering this command, the following should be printed out:

```
C     Walk a bit Test .......................Running ----------->
```

If an error occurs, a map of the mezzanine board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 12.15.

```
C     Walk a bit Test .......................Running ----------->
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
C     Walk a bit Test .......................Running -----------> PASSED
```

12.6 RANDOM PATTERN TEST — MEZZ D

12.6.1 Description

This test fills the entire memory board with a pseudo-random pattern, then examines each memory location, regenerating the random value previously written to it. An error is declared when the data read from the target location differs from that expected. If any errors occur, then the test completes a map of the mezzanine board and displays the devices suspected. Refer to section 12.15.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

12.6.2 Command Input

```
135Diag> MEZZ D
```

12.6.3 Response/Messages

After entering this command, the following should be printed out:

```
D     Random Pattern ........................Running ----------->
```

If an error occurs, a map of the mezzanine board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 12.15.

```
D     Random Pattern ........................Running ----------->
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
D     Random Pattern ........................Running -----------> PASSED
```

12.7 REFRESH TEST MEZZ E

12.7.1 Description

This test fills the entire memory board with a pseudo-random pattern, delays for approximately 4Ø seconds, then examines each memory location, regenerating the random value previously written to it. The delay allows for the effects of faulty dynamic refreshing to become evident. An error is declared when the data read from the target location differs from that expected. If any errors occur, then the test completes a map of the mezzanine board and displays the devices suspected. Refer to section 12.15.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

12.7.2 Command Input

```
135Diag> MEZZ E
```

12.7.3 Response/Messages

After entering this command, the following should be printed out:

```
E     Refresh ..............................Running ----------->
```

If an error occurs, a map of the mezzanine board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 12.15.

```
E     Refresh ..............................Running ----------->
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
E     Refresh ..............................Running -----------> PASSED
```

12.8 PROGRAM TEST **MEZZ F**

12.8.1 Description

This command moves a program segment into RAM and executes it. The implementation of this is as follows:

Step 1: The program is moved into the RAM, repeating it as many times as necessary to fill the test address range. Only complete segments of the program are moved. The space remaining from the last program segment copied into the RAM to the end of the mezzanine is filled with NOP instructions.

Step 2: The last location receives an RTS instruction.

Step 3: Finally, the test performs a JSR to the first location of test RAM, which is the start address of the first copy of the program.

Step 4: The program itself performs a wide variety of operations, with the results frequently being checked and a count of the errors maintained. Errant locations are reported in the same fashion as for the other mezzanine tests (refer to section 12.15).

12.8.2 Command Input

```
135Diag> MEZZ F
```

12.8.3 Response/Messages

After entering this command, the display should read as follows:

```
F     Program Test ..........................Running ----------->
```

If the program (in RAM) detects any errors, then the error information will be logged and displayed at the end of the test (refer to section 12.15).

```
F     Program Test ..........................Running ----------->
(error-related information)
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
F     Program Test ..........................Running -----------> PASSED
```

12.9 TAS TEST MEZZ G

12.9.1 Description

This test verifies that the mezzanine DRAM operates correctly when accessed with a read/modify/write cycle. First the entire test address range is written with a background pattern of zeros. Then, starting at the base address ($FFE03000 for a 1-megabyte MVME135), the following steps are executed: a Test-And-Set is attempted (changing the byte to $80), then the byte is compared to $80. An error is declared when the data read from the target location differs from that expected. After the TAS and comparison have been made, the target location pointer is incremented by $00000104. If this value is within the test address range ($FFE03000 to $FFFEFFFF for a 1-megabyte MVME135), the test continues for this new location. If any errors occur, then the test completes a map of the mezzanine board and displays the devices suspected. Refer to section 12.15.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.10 of this manual for instructions on invoking the loop-on-error mode.

12.9.2 Command Input

```
135Diag> MEZZ G
```

12.9.3 Response/Messages

After entering this command, the following should be printed out:

```
G     TAS Test ..............................Running ----------->
```

If an error occurs, a map of the mezzanine board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 12.15.

```
G     TAS Test ..............................Running ----------->
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
G     TAS Test ..............................Running -----------> PASSED
```

12.1Ø ADDRESS COMPLEMENT TEST MEZZ H

12.1Ø.1 Description

This test creates a noisy environment by rapidly switching the address and data lines, starting at the extremes and working in towards the middle. $55555555 is written to the lower half of the board, while $AAAAAAAA is written to the upper half. The first location written to is the base ($FFEØ3ØØØ for 1-megabyte, $FF8Ø3ØØØ for 4-megabyte). The next location written to is the last longword ($FFEFFFFC for 1-megabyte, $FFBFFFFC for 4-megabyte). Next, $55555555 is written to the base plus four, then $AAAAAAAA is written to the next-to-last longword. The sequence terminates when the middle of the board is reached. Another pass through the test memory address range is then made to verify that the data was written without being disturbed by noise. An error is declared when the data read from a location differs from the data previously written to it.

This test is not designed to accommodate the loop-on-error mode of the diagnostic monitor. Prefixing the invocation of this command with "LE" will have no effect on the operation of the test.

12.1Ø.2 Command Input

```
135Diag> MEZZ H
```

12.1Ø.3 Response/Messages

After entering this command, the following should be printed out:

```
H     Addr. Complement .......................Running ----------->
```

If an error occurs, a map of the mezzanine board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 12.15.

```
H     Addr. Complement .......................Running ----------->
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
H     Addr. Complement .......................Running -----------> PASSED
```

12.11 BYTE/WORD/LONG PERMUTATIONS TEST MEZZ I

12.11.1 Description

This test verifies that the mezzanine can accommodate byte, word and longword writes and reads in any combination. The test also verifies that the bytes, words and longwords are organized in memory in a manner consistent with the MC68ØØØ architecture. The test is implemented as follows:

Step 1: Load a test pattern of $Ø1234567 into a data register.

Step 2: Begin at the base address. Write the data as a longword. Read the data back as four bytes, two words and then as a longword. Compare with original data pattern.

Step 3: Write the data as a pair of sixteen-bit words. Read the data back as four bytes, two words and then as a longword. Compare with original data pattern in each case.

Step 4: Write the data as four bytes. Read the data back as four bytes, two words and then as a longword. Compare with original data pattern in each case.

Step 5: Repeat the sequence of writes and reads every 26Ø bytes, until the end of the mezzanine is reached.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

12.11.2 Command Input

```
135Diag> MEZZ I
```

12.11.3 Response/Messages

After entering this command, the following should be printed out:

```
I     B/W/L Permutations ....................Running ----------->
```

If an error occurs, a map of the mezzanine board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 12.15.

```
I     B/W/L Permutations ....................Running ------------>
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
I     B/W/L Permutations ....................Running ------------> PASSED
```

12.12 WRITE/READ CSR TEST MEZZ J

12.12.1 Description

This test verifies that the control register bits that control the operation of the DRAM mezzanine board are both writable and readable. Patterns are written to control register CNT1 of the MVME135/136 so that the WWP and PAREN* bits are each written both to logic one and logic zero. Intermediate reads check that the written values are actually retained by the CSR. Failure is declared if they are not.

If a read or write to CNT1 results in a bus error the test will also fail. This could be caused by faulty on-board decoding of the control/status registers.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

12.12.2 Command Input

```
135Diag> MEZZ J
```

12.12.3 Response/Messages

After entering this command, the following should be printed out:

```
J    Write/Read CSR .......................Running ----------->
```

If the MVME135 cannot access control register CNT1, one or two bus errors may occur.

```
J    Write/Read CSR .......................Running ----------->
(one or two bus errors may cause register dumps here)
 .... FAILED
```

If the data read from the CSR is different from the expected data, then the test fails and the display will appear as follows.

```
J    Write/Read CSR .......................Running ----------->
     CSR addr=FFFBØØØE Expt=x1Øxxxxx Read=xØØxxxxx
 .... FAILED
```

Note that the error message only displays the value of bits that were actually tested. Other bits which are in the CSR but are not tested (i.e., are not read/writable) are designated by "x".

If no errors occur, the test passes and the display should appear as follows.

```
J     Write/Read CSR ........................Running -----------> PASSED
```

12.13 WRITE-WRONG PARITY TEST MEZZ K

12.13.1 Description

This test is a primary exercise of the parity generation logic. Results of this test are valid only if jumper block J6 on the MVME135/136 has been jumpered from pins 2 to 3 to allow parity operation of the mezzanine board.

The initial target address is $3ØØØ past the start of the mezzanine ($FFEØ3ØØØ for a 1-megabyte mezzanine, $FF8Ø3ØØØ for 4 megabytes). Starting with that address, the following steps are performed. Write-Wrong-Parity (WWP) and Enable-Parity-Error-Reporting (PAREN*) bits are asserted in CNT1. Next, a test pattern (incremented for each location) is written to the target location (the corresponding parity bit should be set to the wrong state). The data is read back from the target location, with a bus error expected. A failure of the DRAM device containing the parity bit is logged if the bus error does not occur. The target location pointer is then incremented by $ØØØØØ4Ø1, and the test continues throughout the address range of the mezzanine board.

This test assumes that control register CNT1 of the MVME135 may be written and read without causing a bus error. This may be verified by running the Write/Read CSR Test (MEZZ J) before running this test.

If any errors occur, a physical map of the mezzanine board will be displayed to indicate the components suspected. Refer to section 12.15.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

12.13.2 Command Input

```
135Diag> MEZZ K
```

12.13.3 Response/Messages

After entering this command, the following should be printed out:

```
K   Write-Wrong Parity ....................Running ----------->
```

In the event of failure, a map of the mezzanine board will be displayed to indicate which components are suspected of being faulty. Refer to section 12.15.

```
K     Write-Wrong Parity .....................Running ------------>
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
K     Write-Wrong Parity .....................Running ------------> PASSED
```

12.14 PARITY MARCH TEST MEZZ L

12.14.1 Description

This test is a more thorough exercise of the parity generation logic than the Write-Wrong-Parity test. As with the Write-Wrong-Parity test, the results of this test are valid only if jumper block J6 on the MVME135/136 has been jumpered from pins 2 to 3 to allow parity operation of the mezzanine board.

The parity march test executes in the following manner. First, control register CNT1 is initialized to enable parity error reporting by asserting the PAREN* bit. Each location of the board is then initialized to zero to create a background pattern. Since the MVME135/136 uses an even parity scheme, this pattern should should result in all parity bits being written to logic zero by the parity-generation circuitry. Starting $3ØØØ past the start of mezzanine memory, the following steps are then performed.

A location (byte) is read. If the corresponding parity bit did not get written to zero when zeros were written to the test address range, then the read will cause a parity-fault bus error to occur and an error will be logged. Next the test pattern $6B is written to the test location. This pattern should cause the location's parity bit to be written to a logic one. The target location pointer is then incremented by one and the test continues throughout the range of memory on the mezzanine board. Once the board has been checked using the test pattern $6B, another check is performed using the test pattern $96 (same steps).

This test assumes that control register CNT1 of the MVME135 may be written and read without causing a bus error. This may be verified by running the Write/Read CSR Test (MEZZ J) before running this test.

If any errors occur, a physical map of the mezzanine board will be displayed to indicate the components suspected. Refer to section 12.15.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

12.14.2 Command Input

```
135Diag> MEZZ L
```

12.14.3 Response/Messages

After entering this command, the following should be printed out:

```
L     Parity March .........................Running ----------->
```

If a bus error occurs, the test fails and the display will appear as follows.

```
L     Parity March .........................Running ----------->
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
L     Parity March .........................Running -----------> PASSED
```

12.15 MEZZANINE PHYSICAL MAP DISPLAY

Most of the tests for the DRAM mezzanine will display a map of the board if an error occurs. This map is an aid to determining which components are faulty.

When an error is detected, the test will log the address of the error, the expected and read data. This is used to tag bad components. When the test completes, the map is printed out line by line. A translation is done to determine which devices are bad and each faulty component is indicated by showing its location descriptor and corresponding data bit position on the map. DRAM devices which are assumed good appear blank on the map.

Below the map, a line containing information about the most recent data error (address, expected and read data) is displayed. If errors exist only in the parity DRAMs, this line will not be displayed.

Sample display for 1-megabyte mezzanine:

```
C    Walking-Bit ...........................Running ----------->
+-------+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       |       |U08/D06|       |
+-------+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+-------+
address = FFE04FF0, expected 80000000, read 80000040
 .... FAILED
```

The sample display shown above would indicate that the device at U8 is faulty.

Sample display for 4-megabyte mezzanine:

```
C     Walking-Bit ..........................Running ----------->
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+
                |       |U4Ø/PAR|       |       |
                +-------+-------+-------+-------+
 .... FAILED
```

The sample display shown above would indicate that the device at U4Ø (which contains parity bits) is faulty.

CHAPTER 13
DYNAMIC RAM MEMORY BOARD TESTS - MVME224

13.1 GENERAL DESCRIPTION

This section details the diagnostics needed to test the MVME224 module using 135Bug. These tests are intended for both the 8-Megabyte and 4-Megabyte versions of the MVME224. The test software checks to see which version of the board is being tested and makes the necessary adjustments transparent to the user.

The expected address range of the MVME224 being tested is as follows:

8-Megabyte MVME224 - VMEbus: \$ØØ6ØØØØØ-\$ØØDFFFFF VSB: \$Ø16ØØØØØ-\$Ø1DFFFFF
4-Megabyte MVME224 - VMEbus: \$ØØ6ØØØØØ-\$ØØ9FFFFF VSB: \$Ø16ØØØØØ-\$Ø19FFFFF

Table 13-1 lists the MVME224 diagnostic tests.

TABLE 13-1. DYNAMIC MEMORY BOARD TESTS - MVME224

Monitor Command	Title	Section
V224 E	VME Tests	13.3
V224 E Ø	Row March Test	13.6
V224 E 1	March Address Test	13.7
V224 E 2	Walk a Bit Test	13.8
V224 E 3	Random Pattern Test	13.9
V224 E 4	Refresh Test	13.1Ø
V224 E 5	Program Test	13.11
V224 E 6	TAS Test	13.12
V224 E 7	Address Complement Test	13.13
V224 E 8	Byte/Word/Long Permutations Test	13.14
V224 E 9	Write/Read CSR Test	13.15
V224 E A	Write-Wrong Parity Test	13.16
V224 E B	Parity March Test	13.17
V224 X	VSB Tests	13.4
V224 X 1	March Address Test	13.7
V224 X 2	Walk a Bit Test	13.8
V224 E 3	Random Pattern Test	13.9
V224 X 4	Program Test	13.11

TABLE 13-1. DYNAMIC MEMORY BOARD TESTS - MVME224 (cont.)

Monitor Command	Title	Section
V224 X 5	TAS Test	13.12
V224 X 6	Address Complement Test	13.13
V224 X 7	Byte/Word/Long Permutations Test	13.14
V224 X 8	Write-Wrong Parity Test	13.16
V224 DE	Display V224 Error Log	13.19
V224 ZE	Clear V224 Error Log	13.2Ø

13.2 HARDWARE CONFIGURATION

The following hardware is required to perform these tests.

MVME135 CPU - Test station.
MVME224 RAM - Unit under test: addressed at $ØØ6ØØØØØ for VMEbus,
addressed at $Ø16ØØØØØ for VSB.
VSB backplane ribbon cable for MVME135 and MVME224
Video Display Terminal.

Switch settings for the MVME224 under test:

S1:	OFF: 2, 3	ON: 1, 4-8
S2:	OFF: 6, 7	ON: 1-5, 8
S3:	OFF: 1	ON: 2-8

13.3 VME TESTS

These commands (V224 E Ø through V224 E B) test the MVME224 via the VMEbus. The command input is as follows:

```
135Diag>  V224 E x (where " x" is replaced with Ø through B)
```

13.4 VSB Tests

These commands (refer to Table 13-1) test the MVME224 via the VSB bus. The command input is as follows:

```
135Diag>  V224 X x (where " x" is replaced with 1 through 8)
```

13.5 ERROR DISPLAY

Errors from the MVME224 tests will be logged according to device-at-fault. At the end of each test, the error log will be displayed if errors have occurred. Refer to section 13.18 for more information including typical displays.

Normally, the error log is cleared at the beginning of every test. In Loop-Continuous mode, however, the error log is not cleared before tests in order to accumulate errors from multiple tests in the same log.

To clear the error log before beginning tests in the Loop-Continuous mode use the **V224 ZE** (clear error log) command. Refer to section 13.2Ø for more information about the **V224 ZE** command.

At any time, the contents of the error log may be displayed using the **V224 DE** (display errors) command. Refer to section 13.19 for more information about the **V224 DE** command.

13.6 ROW MARCH TEST **V224 E Ø**

13.6.1 Description

This first test is a quick check of the board to verify that every row of the memory devices is uniquely addressable. This test is only run using the VMEbus. The first location written to is $ØØ6ØØØØØ. The next location written to is the first address plus $ØØØØØ1Ø4. The last address written to is the last longword of the board.

Three passes are made through the test address range. During the first pass, a background pattern of zeroes is written to each target address. On the second pass, the target address is checked to contain zero, then, if so, it is written to $FFFFFFFF. On the third pass, the target address is checked to contain $FFFFFFFF. Redundant addressing will cause a write to one address to put the same data at another address also. An error is declared when the data read from a location differs from the data previously written to it.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

13.6.2 Command Input

```
135Diag> V224 E Ø
```

13.6.3 Response/Messages

After entering this command, the following should be printed out:

```
Ø      Row March ..............................Running ----------->
```

If an error occurs, a map of the board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 13.18.

```
Ø      Row March ..............................Running ----------->
(map and error information displayed here, refer to section 13.18)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
Ø     Row March .............................Running -----------> PASSED
```

13.7 MARCH ADDRESS TEST

V224 E 1
V224 X 1

13.7.1 Description

This test verifies that every longword on the MVME224 is uniquely addressable. The first address written is the first longword of the board. The next location written to is the first address plus four. The last address written to is the last longword of the board.

Three passes are made through the test address range. During the first pass, a background pattern of zeros is written to each target address. On the second pass, the target address is checked to contain zero, then, if so, it is written to $FFFFFFFF. On the third pass, the target address is checked to contain $FFFFFFFF. Redundant addressing will cause a write to one address to put the same data at another address also. An error is declared when the data read from a location differs from the data previously written to it.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

13.7.2 Command Input

```
135Diag> V224 E 1 (for VMEbus access)
135Diag> V224 X 1 (for VSB access)
```

13.7.3 Response/Messages

After entering this command, the following should be printed out:

```
1     March Addr. Test .......................Running ----------->
```

If an error occurs, a map of the MVME224 board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 13.18.

```
1     March Addr. Test .......................Running ----------->
(map and error information displayed here, refer to section 13.18)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
1     March Addr. Test ......................Running -----------> PASSED
```

13.8 WALK A BIT TEST

V224 E 2
V224 X 2

13.8.1 Description

This test checks the data lines by "walking" a bit through several locations. The first pattern written is $ØØØØØØØ1. $ØØØØØØØ2 is next and is followed by $ØØØØØØØ4. The last pattern written is $8ØØØØØØØ. The first location written to is the first longword of the board. The next location written to is the first address plus $ØØØØØ1Ø4. This value ($1Ø4) is repeatedly added to the target address to exercise the address lines. An error is declared when the data read from a location differs from the data just written to it. The test completes when the target address no longer falls in the range assigned to the board.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

13.8.2 Command Input

```
135Diag> V224 E 2 (for VMEbus access)
135Diag> V224 X 2 (for VSB access)
```

13.8.3 Response/Messages

After entering this command, the following should be printed out:

```
2     Walk a bit Test .......................Running ----------->
```

If an error occurs, a map of the MVME224 board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 13.18.

```
2     Walk a bit Test .......................Running ----------->
(map and error information displayed here, refer to section 13.18)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
2     Walk a bit Test .......................Running -----------> PASSED
```

13.9 RANDOM PATTERN TEST

V224 E 3
V224 X 3

13.9.1 Description

This test fills the entire memory board with pseudo-random patterns, then examines each memory location, regenerating the random value previously written to it. An error is declared when the data read from the target location differs from that expected. If any errors occur, then the test completes a map of the MVME224 board and displays the devices suspected. Refer to section 13.18.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

13.9.2 Command Input

```
135Diag> V224 E 3 (for VMEbus access)
135Diag> V224 X 3 (for VSB access)
```

13.9.3 Response/Messages

After entering this command, the following should be printed out:

```
3     Random Pattern .........................Running ------------>
```

If an error occurs, a map of the board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 12.15.

```
3     Random Pattern .........................Running ------------>
(map and error information displayed here, refer to section 12.15)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
3     Random Pattern .........................Running ------------> PASSED
```

13.1Ø REFRESH TEST V224 E 4

13.1Ø.1 Description

This test fills the entire memory board with pseudo-random patterns, delays for approximately 4Ø seconds, then examines each memory location, regenerating the random value previously written to it. The delay allows for the effects of faulty dynamic refreshing to become evident. An error is declared when the data read from the target location differs from that expected. If any errors occur, then the test completes a map of the board and displays the devices suspected. Refer to section 13.18.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking to loop-on-error mode.

13.1Ø.2 Command Input

```
135Diag> V224 E 4
```

13.1Ø.3 Response/Messages

After entering this command, the following should be printed out:

```
4    Refresh ...............................Running ----------->
```

If an error occurs, a map of the board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 13.18.

```
4    Refresh ...............................Running ----------->
(map and error information displayed here, refer to section 13.18)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
4    Refresh ...............................Running -----------> PASSED
```

13.11 PROGRAM TEST

V224 E 5
V224 X 4

13.11.1 Description

This command moves a program segment into RAM and executes it. The implementation of this is as follows:

Step 1: The program is moved into the RAM, repeating it as many times as necessary to fill the test address range. Only complete segments of the program are moved. The space remaining from the last program segment copied into the RAM to the end of the test memory space is filled with NOP instructions.

Step 2: The last location receives an RTS instruction.

Step 3: Finally, the test performs a JSR to the first location of test RAM, which is the start address of the first copy of the program.

Step 4: The program itself performs a wide variety of operations, with the results frequently being checked and a count of the errors maintained. Errant locations are reported in the same fashion as for the other MVME224 tests (refer to section 13.18).

13.11.2 Command Input

```
135Diag> V224 E 5 (for VMEbus access)
135Diag> V224 X 4 (for VSB access)
```

13.11.3 Response/Messages

After entering this command, the display should read as follows:

```
5 (4)    Program Test .......................Running ----------->
```

If the program (in RAM) detects any errors, then the error information will be logged and displayed at the end of the test (refer to section 13.18).

```
5 (4)    Program Test .......................Running ----------->
(error-related information)
 .... FAILED
```

If no errors occur, then the display will appear as follows:

```
5 (4)    Program Test .......................Running ------------> PASSED
```

13.12 TAS TEST

V224 E 6
V224 X 5

13.12.1 Description

This test verifies that the MVME224 operates correctly when accessed with a read/modify/write cycle. First the entire test address range is written with a background pattern of zeros. Then, starting at the base address the following steps are executed: a Test-And-Set is attempted (changing the byte to $8Ø), then the byte is compared to $8Ø. An error is declared when the data read from the target location differs from that expected. After the TAS and comparison have been made, the target location pointer is incremented by $ØØØØØ1Ø4. If this value is within the test address range the test continues for this new location. If any errors occur, then the test completes a map of the board and displays the devices suspected. Refer to section 13.18.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

13.12.2 Command Input

```
135Diag> V224 E 6 (for VMEbus access)
135Diag> V224 X 5 (for VSB access)
```

13.12.3 Response/Messages

After entering this command, the following should be printed out:

```
6 (5)    TAS Test ..........................Running ----------->
```

If an error occurs, a map of the board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 13.18.

```
6 (5)    TAS Test ..........................Running ----------->
(map and error information displayed here, refer to section 13.18)
.... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
6 (5)    TAS Test ...........................Running -----------> PASSED
```

13.13 ADDRESS COMPLEMENT TEST

V224 E 7
V224 X 6

13.13.1 Description

This test creates a noisy environment by rapidly switching the address and data lines, starting at the extremes and working in towards the middle. $55555555 is written to the lower half of the board, while $AAAAAAAA is written to the upper half. The first location written to is the base (first longword) of the board. The next location written to is the last longword. Next, $55555555 is written to the base plus four, then $AAAAAAAA is written to the next-to-last longword. The sequence terminates when the middle of the board is reached. Another pass through the test memory address range is then made to verify that the data was written without being disturbed by noise. An error is declared when the data read from a location differs from the data previously written to it.

This test is not designed to accommodate the loop-on-error mode of the diagnostic monitor. Prefixing the invocation of this command with "LE" will have no effect on the operation of the test.

13.13.2 Command Input

```
135Diag> V224 E 7 (for VMEbus access)
135Diag> V224 X 6 (for VSB access)
```

13.13.3 Response/Messages

After entering this command, the following should be printed out:

```
7 (6)    Addr. Complement ....................Running ----------->
```

If an error occurs, a map of the board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 13.18.

```
7 (6)    Addr. Complement ....................Running ----------->
(map and error information displayed here, refer to section 13.18)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
7 (6)    Addr. Complement ....................Running -----------> PASSED
```

13.14 BYTE/WORD/LONG PERMUTATIONS TEST

V224 E 8
V224 X 7

13.14.1 Description

This test verifies that the MVME224 can accommodate byte, word, and longword writes and reads in any combination. The test also verifies that the bytes, words and longwords are organized in memory in a manner consistent with the MC68ØØØ architecture. The test is implemented as follows:

Step 1: Load a test pattern of $Ø1234567 into a data register.

Step 2: Begin at the base address. Write the data as a longword. Read the data back as four bytes, two words and then as a longword. Compare with original data pattern.

Step 3: Write the data as a pair of sixteen-bit words. Read the data back as four bytes, two words and then as a longword. Compare with original data pattern in each case.

Step 4: Write the data as four bytes. Read the data back as four bytes, two words, and then as a longword. Compare with original data pattern in each case.

Step 5: Repeat the sequence of writes and reads every 26Ø bytes, until the end of the board is reached.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

13.14.2 Command Input

```
135Diag> V224 E 8 (for VMEbus access)
135Diag> V224 X 7 (for VSB access)
```

13.14.3 Response/Messages

After entering this command, the following should be printed out:

```
8 (7)    B/W/L Permutations ..................Running ------------>
```

If an error occurs, a map of the board will be displayed to indicate which components are suspected of being faulty. The most recent expected and read errant data will be displayed beneath. Refer to section 13.18.

```
8 (7)    B/W/L Permutations ..................Running ----------->
(map and error information displayed here, refer to section 13.18)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
8 (7)    B/W/L Permutations ..................Running -----------> PASSED
```

13.15 WRITE/READ CSR TEST V224 E 9

13.15.1 Description

This test verifies that the control register bits that control the operation of the MVME224 board are both writable and readable. Patterns are written to the MVME224's control/status register so that the WWP and PBEN bits are each written both to logic one and logic zero. Intermediate reads check that the written values are actually retained by the CSR. Failure is declared if they are not.

If a read or write to the CSR results in a bus error the test will also fail. This could be caused by faulty decoding of the control/status register.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

13.15.2 Command Input

```
135Diag> V224 E 9
```

13.15.3 Response/Messages

After entering this command, the following should be printed out:

```
9     Write/Read CSR ........................Running ----------->
```

If the MVME135 cannot access the MVME224's CSR, one or two bus errors may occur.

```
9     Write/Read CSR ........................Running ----------->
(one or two bus errors may cause register dumps here)
 .... FAILED
```

If the data read from the CSR is different from the expected data, then the test fails and the display will appear as follows.

```
9     Write/Read CSR ........................Running ----------->
      CSR addr=FFFFBEØD Expt=xxxxxxØØ Read=xxxxxx1Ø
 .... FAILED
```

Note that the error message only displays the value of bits that were actually tested. Other bits which are in the CSR but are not tested (i.e., are not read/writable) are designated by "x".

If no errors occur, the test passes and the display should appear as follows.

```
J     Write/Read CSR ........................Running -----------> PASSED
```

13.16 WRITE-WRONG PARITY TEST

V224 E A
V224 X 8

13.16.1 Description

This test is a primary exercise of the parity generation logic. Starting with the base address, the following steps are performed. Write-Wrong-Parity (WWP) and Enable-Parity-Error-Reporting (PBEN) bits are asserted in the MVME224's CSR. Next, a test pattern (incremented for each location) is written to the target location (the corresponding parity bit should be set to the wrong state). The data is read back from the target location, with a bus error expected. A failure of the DRAM device containing the parity bit is logged if the bus error does not occur. The target location pointer is then incremented by $ØØØØØ4Ø1, and the test continues throughout the address range of the memory board.

This test assumes that the MVME224's control/status register may be written and read without causing a bus error. This may be verified by running the Write/Read CSR Test before running this test.

If any errors occur, a physical map of the board will be displayed to indicate the components suspected. Refer to section 13.18.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

13.16.2 Command Input

```
135Diag> V224 E A (for VMEbus access)
135Diag> V224 X 8 (for VSB access)
```

13.16.3 Response/Messages

After entering this command, the following should be printed out:

```
A (8)    Write-Wrong Parity .................Running ----------->
```

In the event of failure, a map of the board will be displayed to indicate which components are suspected of being faulty. Refer to section 13.18.

```
A (8)    Write-Wrong Parity ..................Running ------------>
(map and error information displayed here, refer to section 13.18)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
A (8)    Write-Wrong Parity ..................Running ------------> PASSED
```

13.17 PARITY MARCH TEST V224 E B

13.17.1 Description

This test is a more thorough exercise of the parity generation logic than the Write-Wrong-Parity test. Each bit in the DRAM devices containing the parity bits is written both to a high and a low logic state and is verified to be independent of the other bits within the DRAM device.

The parity march test executes in the following manner. First, the MVME224 is initialized to enable parity error reporting by asserting the PBEN bit in the CSR. Each location of the board is then initialized to zero to create a background pattern. Since the MVME224 uses an even parity scheme, this pattern should result in all parity bits being written to logic zero by the parity-generation circuitry. Starting at the first byte of the board, the following steps are then performed. A location (byte) is read. If the corresponding parity bit did not get written to zero when zeros were written to the test address range, then the read will cause a parity-fault bus error to occur and an error will be logged. Next the test pattern $6B is written to the test location. This pattern should cause the location's parity bit to be written to logic one. The target location pointer is then incremented by one and the test continues throughout the range of memory on the board. Once the board has been checked using the test pattern $6B, another check is performed using the test pattern $96 (same steps).

This test assumes that the CSR of the MVME224 may be written and read without causing a bus error. This may be verified by running the Write/Read CSR Test before running this test.

If any errors occurred, a physical map of the board will be displayed to indicate the components suspected. Refer to section 13.18.

This test is designed for loop-on-error operation. If an error occurs in loop-on-error mode the test will execute in a tight loop, causing the error to be repeated to facilitate analysis with an oscilloscope or other diagnostic instrument. Refer to section 4.1Ø of this manual for instructions on invoking the loop-on-error mode.

13.17.2 Command Input

```
135Diag> V224 E B
```

13.17.3 Response/Messages

After entering this command, the following should be printed out:

```
B     Parity March .........................Running ----------->
```

If a bus error occurs, the test fails and the display will appear as follows.

```
B     Parity March ...........................Running ----------->
(map and error information displayed here, refer to section 13.18)
 .... FAILED
```

If no errors occur, the test passes and the display should appear as follows.

```
B     Parity March ...........................Running -----------> PASSED
```

13.18 MVME224 PHYSICAL MAP DISPLAY

Most of the tests for MVME224 will display a map of the module if an error occurs. This map is an indication of which components were determined faulty by the test(s). When an error is detected, the test will log the address of the error, the expected and read data. This is used to tag bad components. When the test completes, the map is printed out line by line. A translation is done to determine which devices contained bad bits and these are indicated on the map by filling in the reference designator ("U-number") of the device instead of a device outline. Devices which were deemed good by the test will appear only as an outline. Below the map, a line containing the error information of the LAST error to occur (address, expected and read data) is displayed.

The 8-megabyte version of the MVME224 will display a map containing four rows of nineteen devices. The 4-megabyte version of the MVME224 will display a map containing two rows of nineteen devices.

The map of the MVME224 is arranged as if the viewer were looking at the board with the front panel, switches, and LEDs on the left and the P1 and P2 connectors on the right. The arrangement of devices in the map is as follows:

```
UØ2 UØ3 UØ4 UØ5 UØ6 UØ7 UØ8 UØ9 U1Ø U11 U12 U13 U14 U15 U16 U17 U18 U19
U27 U28 U29 U3Ø U31 U32 U33 U34 U35 U36 U37 U38 U39 U4Ø U41 U42 U43 U44
U52 U53 U54 U55 U56 U57 U58 U59 U6Ø U61 U62 U63 U64 U65 U66 U67 U68 U69
U76 U77 U78 U79 U8Ø U81 U82 U83 U84 U85 U86 U87 U88 U89 U9Ø U91 U92 U93
```

On the 4-megabyte version of the board, only the top two rows of the array will be populated with DRAM devices.

Sample display for 8-megabyte MVME224:

```
Ø     Row March .............................Running ----------->

( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) U1Ø ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) U6Ø ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
     last failed @ test-adr : ØØDCØØ7C expt FFFFFFFF read FFFDFFFF
 ..... FAILED
```

The above display indicated that the devices at U1Ø and U6Ø are suspect.

Sample display for 4-megabyte MVME224:

```
Ø     Row March ..............................Running ------------>

( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) U35 ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
     last failed @ test-adr : ØØ9CØØ3C expt FFFFFFFF read FDFFFFFF
 ..... FAILED
```

The above display indicates that the device at U35 is suspect.

13.19 DISPLAY MVME224 ERROR LOG V224 DE

13.19.1 Description

As mentioned in the previous section, errors from the MVME224 tests will be logged according to device-at-fault. At the end of each test, the error log will be displayed if errors have occurred.

At any time, however, the contents of the error log may be displayed by entering **V224 DE** (display errors) from the 135Diag prompt. The error log will contain errors accumulated during the last test.

If the error log is empty (i.e., no errors occurred) the message "No Errors" will be displayed.

13.19.2 Command Input

```
135Diag> V224 DE
```

13.19.3 Response/Messages

After entering this command, a display similar to the following should be printed out for an 8-megabyte board that has logged errors:

```
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) U1Ø ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) U6Ø ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
     last failed @ test-adr : ØØDCØØ7C expt FFFFFFFF read FFFDFFFF
```

A display similar to the following should be printed out for a 4-megabyte board that has logged errors:

```
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) U35 ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )
     last failed @ test-adr : ØØ9CØØ3C expt FFFFFFFF read FDFFFFFF
```

If the error log is empty, the message "No Errors" will appear.

13.2Ø CLEAR MVME224 ERROR LOG V224 ZE

13.2Ø.1 Description

This command is used to clear the error logger of any logged failures prior to running a new set of tests in the loop-continuous mode.

The error log is normally cleared at the beginning of each test. In loop-continuous mode, however, the error log is not cleared before tests in order to accumulate errors for multiple tests in the same log.

13.2Ø.2 Command Input

```
135Diag> V224 ZE
```

13.2Ø.3 Response/Messages

After entering this command, no message is printed. The 135Diag prompt should reappear, indicating that the command has completed.

```
135Diag> V224 ZE
135Diag>
```

Invoking the Display Errors command (**V224 DE**) will show that the error log is empty:

```
135Diag> V224 DE
No Errors
```

THIS PAGE INTENTIONALLY LEFT BLANK.

CHAPTER 14
MP-CSR TESTS

The MVME135 is especially suitable for multi-processor applications. The architecture of the MVME135 includes a Multi-Processor Control and Status Register, afterward referred to as the MP-CSR. The MP-CSR provides a convenient vehicle for communication and shared control among multiple processors on the same bus.

14.1 GENERAL DESCRIPTION

The MVME135's MP-CSR is a set of registers which are dual ported to the MVME135's local bus and to the VMEbus. The MP-CSR is available to the local processor through its local map and to other bus masters via the VMEbus in the short I/O address range.

TABLE 14-1. MP-CSR TESTS

Monitor Command	Title	Section
MP A	MPØ - MP3 & MP_COMM TAS Test	15.3
MP B	SIG(L)(H) Priority Interrupt Test	15.4
MP C	Location Monitor (KING)3-Ø Tests	15.5
MP D	Watchdog Timer Tests	15.6
MP E	RESET & HOLD Test	15.7
MP F	HALT & HOLD Test	15.8
MP G	Lock Transfer Test	15.9
MP H	Global Reset Test	15.1Ø

14.2 HARDWARE CONFIGURATION

The following hardware is required to perform these tests.

MVME135 CPU - Board being tested (referred to as the " UUT").
MVME135 CPU - Auxiliary bus master (" Aux") used in some tests.
MVME2Ø4 RAM Board - Addressed at $ØØAØØØØØ (VMEbus) and $ØØCØØØØØ (VSB).
MVMEØ5Ø System Module - Used to generate VMEbus interrupts.
Video Display Terminal.

14.3 MPØ - MP3, AND MP_COMM TAS TEST — MP A

14.3.1 Description

This test calculates the VME Short I/O address for the MP-CSR per the ID switch, then verifies that MPØ through MP3 and the MP COMM byte can correctly execute TAS instructions locally and over VME.

14.3.2 Command Input

```
135Diag> MP A
```

14.3.3 Response/Messages

After entering this command, the following should be displayed:

```
A     MPØ - MP3, & MP_COMM TAS Test .........Running ------------>
```

If a TAS instruction does not function properly, the associated MP-CSR register address (local - FFFBØØ6X, or VME - FFFFCØØX) will be displayed along with the expected and read data as shown below:

```
A     MPØ - MP3, & MP_COMM TAS Test .........Running ------------>
      Addr=FFFXØØXX     Expect=XX            Read=XX
 .... FAILED
```

If no errors are encountered, the test completes successfully, and the following is displayed:

```
A     MPØ - MP3, & MP_COMM TAS Test .........Running ------------>  PASSED
```

14.4 SIGNAL LOW/HIGH PRIORITY INTERRUPT TEST MP B

14.4.1 Description

This command tests the two MP-CSR Signal bits (SIGLP and SIGHP) by verifying proper functionality when interrupts are masked and unmasked. The test has been implemented in the following manner:

Step 1: For SIGnal Low Priority, enable interrupt SLPIEN, set SIGLP, and verify interrupt occurs.

Step 2: Mask SLPIEN, set SIGLP, and verify interrupt does not occur.

Step 3: Download code to AUX board, enable interrupt, and verify interrupt occurs when AUX sets UUT's SIGLP bit.

Step 4: Repeat steps 1-3, for SIGnal High Priority, using the SHPIEN and SIGHP bits.

14.4.2 Command Input

```
135Diag> MP B
```

14.4.3 Response/Messages

After entering this command, the following should be displayed:

```
B     SIG(L) (H) Priority Interrupt Test......Running ----------->
```

If the interrupt did not occur for either SIGnal Low or High Priority, then the error will be reported as shown:

```
B     SIG(L) (H) Priority Interrupt Test......Running ----------->
      Low(High) Priority Signal Test
        Signal Interrupt Enable Test
      FAILED
```

If the interrupt occurred for either SIGnal Low or High Priority, with the interrupt masked, the error will be reported as shown:

```
B     SIG(L) (H) Priority Interrupt Test......Running ----------->
      Low (High) Priority Signal Test
        Signal Interrupt Enable Test
        Signal Interrupt Mask Test
      FAILED
```

If the interrupt did not occur for either SIGnal Low or High Priority, when activated by the AUX board, then the error will be displayed as shown:

```
B     SIG(L) (H) Priority Interrupt Test......Running ----------->
      Low(High) Priority Signal Test
        Signal Interrupt Enable Test
        Signal Interrupt Mask Test
        VME (AUX) Signal Interrupt Test
      FAILED
```

If no errors occur, then the display will appear as follows:

```
B     SIG(L) (H) Priority Interrupt Test......Running ----------->
      Low Priority Signal Test
        Signal Interrupt Enable Test
        Signal Interrupt Mask Test
        VME (AUX) Signal Interrupt Test
      High Priority Signal Test
        Signal Interrupt Enable Test
        Signal Interrupt Mask Test
        VME (AUX) Signal Interrupt Test
      PASSED
```

14.5 LOCATION MONITOR (KING)3-Ø TESTS MP C

14.5.1 Description

This test verifies that the Location Monitor bits KING through LMØ can be set, cleared and cause interrupts (LMØ only) properly, both locally and over VME. The test has been implemented in the following manner:

Step 1: For LM3 through LMØ verify that the bit can be cleared, set, and that it goes to zero (low true) on both the UUT and the AUX board on a broadcast cycle.

Step 2: Verify an interrupt occurs locally when LMIEN is enabled, and LMØ is set to zero.

Step 3: Verify that an interrupt occurs on both the UUT and AUX boards, when LMIEN is enabled, and the UUT does a LMØ broadcast cycle.

14.5.2 Command Input

```
135Diag> MP C
```

14.5.3 Response/Messages

After entering this command, the following should be displayed:

```
C     Location Monitor (KING)3-Ø Tests.......Running ----------->
```

If a particular LMx bit (x=3-Ø) can not be cleared, this message will be displayed:

```
C     Location Monitor (KING)3-Ø Tests.......Running ----------->
      LMx BIT DID NOT CLEAR
      FAILED
```

If a particular LMx bit (x=3-Ø) can not be set, this message will be displayed:

```
C     Location Monitor (KING)3-Ø Tests.......Running ----------->
      LMx BIT DID NOT SET
      FAILED
```

If a particular UUT LMx bit (x=3-Ø) does not go low on a broadcast cycle, this message will be displayed:

```
C     Location Monitor (KING)3-Ø Tests.......Running ----------->
      UUT LMx BIT DID NOT CLEAR ON BROADCAST CYCLE
      FAILED
```

If a particular AUX LMx bit (x=3-Ø) does not go low on a broadcast cycle, this message will be displayed:

```
C     Location Monitor (KING)3-Ø Tests.......Running ----------->
      AUX LMx BIT NOT CLEAR ON BROADCAST CYCLE
      FAILED
```

If a broadcast cycle causes an interrupt to occur, with LMIEN disabled, this message will be displayed:

```
C     Location Monitor (KING)3-Ø Tests.......Running ----------->
      LMx INTERRUPT OCCURRED
      FAILED
```

If a LMØ broadcast cycle does not cause an interrupt to occur on the AUX, when LMIEN is enabled, this message will be displayed:

```
C     Location Monitor (KING)3-Ø Tests.......Running ----------->
      NO AUX INTERRUPT ON LMØ BROADCAST CYCLE
      FAILED
```

If a LMØ broadcast cycle does not cause an interrupt to occur on the UUT, when LMIEN in enabled, this message will be displayed:

```
C     Location Monitor (KING)3-Ø Tests.......Running ----------->
      NO UUT INTERRUPT ON LMØ BROADCAST
      FAILED
```

If the UUT's LMØ bit does not go to zero on a LMØ broadcast cycle, this message will be displayed:

```
C     Location Monitor (KING)3-Ø Tests.......Running ------------>
      UUT BIT NOT LOW ON LMØ INTERRUPT
      FAILED
```

If no errors occur, then the display will appear as follows:

```
C     Location Monitor (KING)3-Ø Tests.......Running ------------>  PASSED
```

14.6 WATCH DOG TIMER TESTS MP D

14.6.1 Description

This command tests the four possible Watch Dog Timer modes, that can be selected in CNT5. Three of the four tests will cause a reset on the UUT. The test is returned to after a reset, using the WARMSTART flag to signal WDT test mode in the SYSINIT flow. The WDT modes in order of testing are: No WDT, WDT momentary local reset, WDT system reset, and WDT reset and hold. The test has been implemented in the following manner:

Step 1: With No WDT selected, setup and cause the timer to timeout. Verify a WDT Reset does not occur, but that the WDT bit is set, then cleared.

Step 2: Select WDT momentary local reset, setup RESTARTV, save appropriate registers, then setup and cause the timer to timeout. Verify a local, not a system reset occurs, and that the WDT bit is set.

Step 3: Select WDT system reset, setup RESTARTV, save appropriate registers, then setup and start the timer. Verify a reset occurs on both the UUT and the AUX when the timer times out, and that the WDT bit is set.

Step 4: Select WDT Reset and Hold, download a program to the AUX, setup RESTARTV, save appropriate registers, then cause the timer to timeout. The AUX program verifies the UUT gets into the HOLD state, then takes it out of R&H. The UUT then verifies a reset occurred and that the WDT bit was set.

14.6.2 Command Input

```
135Diag> MP D
```

14.6.3 Response/Messages

After entering this command, the following should be displayed:

```
D     WatchDog Timer Tests ..................Running ----------->
```

If a Reset occurred in the No WDT test, or the WDT bit did not set, then the error will be reported as shown:

```
D     WatchDog Timer Tests ..................Running ----------->
      WDT bit not set &/of RESET occurred
      FAILED
```

If a Reset did not occur in the local WDT test, or the WDT bit did not set, the error will be reported as shown:

```
D     WatchDog Timer Tests ..................Running ----------->
      NO WDT Local reset &/or WDT bit not set
      FAILED
```

If a WDT Local Reset caused the system to Reset, then the error will be displayed as shown:

```
D     WatchDog Timer Tests ..................Running ----------->
      SYSTEM RESET occurred during Local Reset Test
      FAILED
```

If the UUT was not Reset during the WDT System Reset test, or the WDT bit did not set, then the error will be displayed as shown:

```
D     WatchDog Timer Tests ..................Running ----------->
      NO SYSRESET &/or WDT bit not set
      FAILED
```

If the AUX board was not Reset during the WDT System Reset test, then the error will be displayed as shown:

```
D     WatchDog Timer Tests ..................Running ----------->
      AUX Board was not RESET
      FAILED
```

If the UUT did not Reset and Hold during the WDT R&H test, then the error will be displayed as shown:

```
D     WatchDog Timer Tests ..................Running ----------->
      WDT R&H Local Reset did not occur
      FAILED
```

If the UUT's R&H bit did not set, signaling the AUX to wake up the UUT, then the error will be displayed as shown:

```
D     WatchDog Timer Tests ...................Running ------------>
      AUX did not clear UUT R&H
      FAILED
```

If WDT timer is not working properly, then the error will be displayed as shown:

```
D     WatchDog Timer Tests ...................Running ------------>
      NO CIO TIMEOUT
      FAILED
```

If all four tests run successfully, then the display will appear as follows:

```
D     WatchDog Timer Tests ...................Running ------------> PASSED
```

14.7 RESET AND HOLD TEST MP E

14.7.1 Description

This test verifies that the R&H bit when set, causes a Reset and Hold on the UUT, until cleared. The test has been implemented in the following manner:

Step 1: Download and execute a program on the AUX board, which will set the UUT's R&H bit.

Step 2: Verify the UUT is in R&H by examining locations that will change on valid and invalid R&H conditions.

Step 3: AUX clears the R&H bit. This causes UUT system initialization and a return to this test.

14.7.2 Command Input

```
135Diag> MP E
```

14.7.3 Response/Messages

After entering this command, the following should be displayed:

```
E     RESET & HOLD Test ....................Running ----------->
```

If the AUX board was reset as well, then the following error message will be displayed:

```
E     RESET & HOLD Test ....................Running ----------->
      AUX Reset on UUT R&H
      FAILED
```

If the UUT Reset did not occur, then the following error message will be displayed:

```
E     RESET & HOLD Test ....................Running ----------->
      RESET did not occur
      FAILED
```

If the Hold did not occur, then the following error message will be displayed:

```
E     RESET & HOLD Test ....................Running ------------>
      HOLD did not occur
      FAILED
```

If no errors occur, then the display will appear as follows:

```
E     RESET & HOLD Test ....................Running ------------>  PASSED
```

14.8 HALT AND HOLD TEST MP F

14.8.1 Description

This test verifies that the local Halt and Hold bit in the MP-CSR holds the board in a H&H state, until cleared. The test has been implemented in the following manner:

Step 1: Download, then interrupt the AUX to execute a program.

Step 2: AUX sets the H&H bit, waits for the UUT to Halt, then sets and changes a location to be examined by UUT.

Step 3: AUX clears the R&H bit. This causes UUT to resume execution.

Step 4 UUT verifies only the second pattern change was recognized.

14.8.2 Command Input

```
135Diag> MP F
```

14.8.3 Response/Messages

After entering this command, the following should be displayed:

```
F     HALT & HOLD Test ......................Running ------------>
```

If the UUT was not Halted, then the following error message will be displayed:

```
F     HALT & HOLD Test ......................Running ------------>
      HOLD did not occur
      FAILED
```

If no errors occur, then the display will appear as follows:

```
F     HALT & HOLD Test ......................Running ------------>  PASSED
```

14.9 LOCK TRANSFER TEST MP G

14.9.1 Description

This test enables LocK TRansfer mode, then verifies the AUX is not able to set a location over the VMEbus. With LKTR disabled, verify the slave can then properly set the location.

Step 1: Download, then interrupt the AUX to execute a program.

Step 2: When AUX is ready, UUT enables LKTR, then sets a flag on the AUX to cause the Lock Out. This will actually stop the AUX program running out of DRAM.

Step 3: UUT verifies the AUX is not able to modify the first MVME2Ø4 location over VMEbus.

Step 4 After a short delay, UUT then disables LKTR, and verifies the AUX program resumes, and that the VMEbus location can be modified.

14.9.2 Command Input

```
135Diag> MP G
```

14.9.3 Response/Messages

After entering this command, the following should be displayed:

```
G     Lock Transfer Test....................Running ----------->
```

If the AUX was not prevented from executing with LKTR enabled, this message will be displayed:

```
G     Lock Transfer Test....................Running ----------->
      Did not Lock Out AUX Access
      FAILED
```

If the AUX did not resume execution, when the UUT disabled LKTR, this message will be displayed:

```
G     Lock Transfer Test....................Running ----------->
      No AUX access when Lock Out Disabled
      FAILED
```

14.8 HALT AND HOLD TEST **MP F**

14.8.1 Description

This test verifies that the local Halt and Hold bit in the MP-CSR holds the board in a H&H state, until cleared. The test has been implemented in the following manner:

Step 1: Download, then interrupt the AUX to execute a program.

Step 2: AUX sets the H&H bit, waits for the UUT to Halt, then sets and changes a location to be examined by UUT.

Step 3: AUX clears the R&H bit. This causes UUT to resume execution.

Step 4 UUT verifies only the second pattern change was recognized.

14.8.2 Command Input

```
135Diag> MP F
```

14.8.3 Response/Messages

After entering this command, the following should be displayed:

```
F     HALT & HOLD Test .....................Running ----------->
```

If the UUT was not Halted, then the following error message will be displayed:

```
F     HALT & HOLD Test .....................Running ----------->
      HOLD did not occur
      FAILED
```

If no errors occur, then the display will appear as follows:

```
F     HALT & HOLD Test .....................Running ----------->  PASSED
```

14.9 LOCK TRANSFER TEST MP G

14.9.1 Description

This test enables LocK TRansfer mode, then verifies the AUX is not able to set a location over the VMEbus. With LKTR disabled, verify the slave can then properly set the location.

Step 1: Download, then interrupt the AUX to execute a program.

Step 2: When AUX is ready, UUT enables LKTR, then sets a flag on the AUX to cause the Lock Out. This will actually stop the AUX program running out of DRAM.

Step 3: UUT verifies the AUX is not able to modify the first MVME2Ø4 location over VMEbus.

Step 4 After a short delay, UUT then disables LKTR, and verifies the AUX program resumes, and that the VMEbus location can be modified.

14.9.2 Command Input

```
135Diag> MP G
```

14.9.3 Response/Messages

After entering this command, the following should be displayed:

```
G     Lock Transfer Test....................Running ----------->
```

If the AUX was not prevented from executing with LKTR enabled, this message will be displayed:

```
G     Lock Transfer Test....................Running ----------->
      Did not Lock Out AUX Access
      FAILED
```

If the AUX did not resume execution, when the UUT disabled LKTR, this message will be displayed:

```
G     Lock Transfer Test....................Running ----------->
      No AUX access when Lock Out Disabled
      FAILED
```

If no errors occur, then the display will appear as follows:

```
G      Lock Transfer Test....................Running ------------>  PASSED
```

14.10 GLOBAL RESET TEST MP H

14.10.1 Description

This test verifies that the GLBRES bit in CNT4, causes a system reset when cleared (low true). The test has been implemented in the following manner:

Step 1: Do setup similar to Watch Dog Timer Reset Test. Set WARMFLAG to WATCH DOG, in order to return to this test on RESET.

Step 2: Clear GLBRES* bit, and wait for either timeout or return from RESET.

Step 3: Verify both the UUT and AUX has been RESET, meaning no timeout on UUT, and MP_COMM byte reset on AUX.

14.10.2 Command Input

```
135Diag> MP H
```

14.10.3 Response/Messages

After entering this command, the following should be displayed:

```
H     Global Reset Test ....................Running ----------->
```

If the AUX board was not reset, then the following error message will be displayed:

```
H     Global Reset Test ....................Running ----------->
      AUX Not Reset on Global Reset
      FAILED
```

If the UUT Reset did not occur, then following error message will be displayed:

```
H     Global Reset Test ....................Running ----------->
      RESET did not occur
      FAILED
```

If no errors occur, then the display will appear as follows:

```
H     Global Reset Test ....................Running ----------->  PASSED
```

CHAPTER 15
FINAL ASSEMBLY TESTS

15.1 GENERAL DESCRIPTION

The Final Assembly Tests are a set of board level diagnostics. They are intended for use by assembly line personnel in the production environment and do not print out error messages. The only results printed following each test are "PASSED" or "FAILED".

Table 15-1 lists the Final Assembly Tests available in 135Bug.

TABLE 15-1. FINAL ASSEMBLY TESTS

Monitor Command	Title	Section
FAT 135	Dual MVME135 Tests	15.3
FAT 136	Dual MVME135 + MMB851/MC68851 tests	15.4
FAT PKGØ	MVME135 package tests	15.5
FAT PKG1	MVME136 package tests	15.6
FAT 224	MVME224 Memory Board tests	15.7

15.2 HARDWARE CONFIGURATION

The package tests (PKGØ and PKG1) require only the MVME135, an external memory board (MVME2Ø4 or MVME224), an MVMEØ5Ø, and a serial loopback cable in the chassis in order to run. The FAT 135 and FAT 136 tests do more extensive testing of the VMEbus and VSB bus interfaces and require an additional MVME135 board and an MVMEØ5Ø to be in the chassis.

The system configuration necessary to run these tests follows. Switch settings are defined with zeros and ones. One represents a switch that is **OFF**, zero represents a switch that is **ON**. Individual switch positions within a dip switch are called out lowest-to-highest. First switch 1, then 2, 3, 4, 5, etc. In the description below, for example, S3 on the UUT should have switches 1 and 2 **OFF**, and 3, 4, 5, 6, 7, and 8 **ON**.

CHASSIS: MVME945

No IACK daisy chain jumper for slots 1, 2, or 5.
Install IACK and bus grant jumpers for slots 3 and 4.
Install VSB cable to connect slots 2, 3, 4, and 5.
All boards on VME request level 3.

SLOT 1: MVME050

S1=0101 0101
Base address = $FFFF1000

SLOT 2: MVME135 (UUT)

S3=1100 0000, S4=00 0000 0000
System controller
OPT0 enabled (DRAM at $0 in local map)
RAM array available via VMEbus @ $000,000
MPCSR available via VME short I/O @ $C000
VME request level 3 (J2:1-2,5-6,7-8,9-11,10-12,16-18)

The following is only for BUS 1.A (local DRAM parity test)

One wait state for local RAM (J6 jumpered from 2 to 3)

SLOT 3: EMPTY

SLOT 4: MVME224

S1=0110 0000, S2=0000 0110, S3=1000 0000
VME Base address = $00600000
VSB Base address = $01600000
CSR in VME short I/O space at $BE0D

--- OR ---

MVME204

S1=1011 1110, S2=0000 110
S3=0000 0000, S4=0110
S5=0000 0001, S6=0110
VME Base address = $00600000
VSB Base address = $01600000
CSR in VME short I/O space at $BE0D

SLOT 5: MVME135 (AUX)

S3=1101 0000, S4=10 0100 0100
NOT system controller
OPT0 disabled (DRAM not at $0 in local map)
VSB disabled
RAM array available via VMEbus @ $1,000,000
MPCSR available via VME short I/O @ $C200
VME request level 3 (J2:1-2,5-6,7-8,9-11,10-12,16-18)

15.3 FAT135 (SYSTEM TEST)

This set of tests perform a system level test of all MVME135/MVME2Ø4 system components: the MVME135, MVME2Ø4(-1/-2/-2F), and VME chassis. The following tests are called in nonverbose mode by command "FAT 135".

BUS:	all except 1.C (1.C requires a jumper change)
CA2Ø:	E, F
CIO:	All
MT:	E-J (Start = $4ØØØ)
SIO:	A, C, E, G, J-N

15.4 FAT136 (SYSTEM TEST)

This set of tests perform a system level test of all MVME136/MVME2Ø4 system components: the MVME136, MVME2Ø4, VME chassis, and MMU. The following tests are called in nonverbose mode by command "FAT 136".

BUS:	all except 1.C (1.C requires a jumper change)
CA2Ø:	E, F
CIO:	All
MMU:	All
MT:	E-J (Start = $4ØØØ)
SIO:	A, C, E, G, J-N

15.5 FATPKGØ (BOARD TEST)

This set of tests perform a self-test sequence to check all on-board devices and logic of the MVME135. The following tests are called in nonverbose mode by command "FAT PKGØ".

BUS:	Ø.A-Ø.J, 1.A, 1.B, 1.D, 1.F, 1.J-1.L
CA2Ø:	All
CIO:	All
MT:	E-J (start = $8ØØØ)
SIO:	All

15.6 FATPKG1 (BOARD TEST)

This set of tests perform a self-test sequence to check all on-board devices and logic of the MVME136. The following tests are called in nonverbose mode by command "FAT PKG1".

BUS:	Ø.A-Ø.J, 1.A, 1.B, 1.D, 1.F, 1.J-1.L
CA2Ø:	All
CIO:	All
MMU:	All
MT:	E-J (start = $8ØØØ)
SIO:	All

15.7 FAT224 (EXTERNAL MODULE TEST)

This set of tests verifies the proper functioning of a MVME224 dynamic RAM memory module installed in the system. The following tests are called in nonverbose mode by command "FAT 224".

V224: All

THIS PAGE INTENTIONALLY LEFT BLANK.

SUGGESTION/PROBLEM REPORT

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Inc.
Microcomputer Division
2900 S. Diablo Way
Tempe, Arizona 85282
Attention: Publications Manager
Maildrop DW164

Product: ______________________ Manual: ______________________

COMMENTS: ______________________

(For additional comments use other side)

Please Print

Name ______________________ Title ______________________

Company ______________________ Division ______________________

Street ______________________ Mail Drop ______________________

City ______________________ Phone ______________________

State ____________ Zip ____________ Country ______________________

For Additional Motorola Publications
Literature Distribution Center
616 West 24th Street
Tempe, AZ 85282
(602) 994-6561

Motorola Field Service Division/Customer Support
(800) 528-1908
(602) 438-3100

MOTOROLA

COMMENTS:

MOTOROLA INC.

Microcomputer Division
2900 South Diablo Way
Tempe, Arizona 85282
P.O. Box 2953
Phoenix, Arizona 85062

10095 PRINTED IN USA (6/88) MPS 1500

68NW9209E77B