Sun™ StorEdge™ A7000 Diagnostics Reference Manual

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
650 960-1300 Fax 650 969-9131

Part No. 805-4891-10
September 1998, Revision A

Send comments about this document to: `docfeedback@sun.com`

Contents

Tables

Preface

Sun StorEdge A7000 Diagnostics Reference Manual provides information describing the various types of diagnostics provided with the Sun StorEdge A7000 Intelligent Storage Server System. The information provided includes:

- A description of the testing philosophy used to design the diagnostics.
- Descriptions of individual diagnostic tests.
- Descriptions of diagnostic-related messages.
- Procedures for running the diagnostics on your system.

How This Book is Organized

Chapter 1 "Diagnostics Overview" provides an overview of the diagnostics available for testing your computer system. It identifies the different types of diagnostics and their functions.

Chapter 2 "Built-in Self Tests (BISTs)" describes the individual tests included in the Built-in Self Tests (BISTs) executed during the system power up or reset sequence or through the command line interface.

The BISTs are used to test the functionality of various processor and adaptor board components:

- CPUs and cache memory
- DRAM and SRAM
- Timers and interrupts
- Real-time clock
- Counter I/O controllers (CIO)
- Ethernet interface and LAN controller
- VME interrupter
- NVRAM memory

- Error detection and correction logic
- SCSI interface logic

In addition, all memory tests are applied to DRAM on both the processor board and the external memory boards.

This section also provides descriptions of BIST-related messages and commands, and procedures for modifying the testing environment.

Chapter 3 “Extended Diagnostics” describes the Extended Diagnostics. It includes procedures for executing these diagnostics in both automatic and interactive modes.

Chapter 4 “The Diagnose Program” describes a system console application program. The Diagnose program provides a graphical interface to the system diagnostics. Topics discussed include:

- Displaying a graphical representation of the system
- Selecting components for testing
- Starting the Diagnose program
- Interpreting test results

Typographic Conventions

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output.	Use the `help` command to display a list of commands. `Automatic test enabled`
`AaBbCc123`	What you type, when contrasted with on-screen computer output.	`ROM >>`**`test`** *`lvl...lvl`*
AaBbCc123	Book titles, new words or terms, words to be emphasized. Command-line variable; replace with a real name or value.	Read Chapter 2 in the *Console Operations Manual*. `ROM >>`**`readconfig slot`** *`n`* **`all`**
`|`	In command line syntax and system output displays, a vertical line indicates that a choice can be made between the values.	In the following example, either `enabled` or `disabled` will be displayed: `autotest {enabled|disabled}`
`[ ]`	In command line syntax or system output examples, brackets indicate optional values. If several values are placed inside brackets, any or none of them can be entered or displayed. Brackets are also used in system prompts to enclose the response choices.	In the following example, displaying the slot number is optional: `[Slot` *`n`*`:]`
`( )`	In command line syntax or system output examples, parenthesis around several values indicate that the enclosed values *must* be entered. If the values are separated by a vertical line then only one of the values must entered. If they are not separated by a vertical line, then all the values must be entered.	In the following example, either **`set`** or **`reset`** must be entered: (**`set`**|**`reset`**) In the following example, all the values must be entered: (*bus chan target lun*)

TABLE P-1 Typographic Conventions *(Continued)*

Typeface or Symbol	Meaning	Examples
`...`	In command line syntax and running text, a horizontal ellipsis indicates repetition or omission.	In the following example, one or more levels (*lvl*) can be entered or displayed: `levels selected` *lvl...lvl*
`< >`	In examples of command input, an item surrounded by a greater than and less than sign must be replaced with an action. The greater than and less than signs are omitted when performing the action.	`<CR>` means press the Return key.
`Utilities`	An underlined letter in a menu title is a menu shortcut (mnemonic). This means that simultaneously pressing the Alt key and the key for that letter makes that menu appear.	For example, holding down the Alt key while pressing the u key would display the Utilities menu.
`Open`	An underlined letter in a menu is another type of mnemonic. After a menu is displayed, you can select a menu item by pressing the key for the underlined letter in that menu item.	For example, press o when the `File` menu is displayed to select the `Open` item.
Click	Press and immediately release a mouse button in one smooth motion, without moving the mouse.	
Double click	Press and release a mouse button twice in rapid succession.	
Drag	Press and continue to hold down a mouse button while moving the mouse.	
Select	Click on something to highlight it.	

Related Documentation

TABLE P-2 Related Documentation

Type	Title
User interface	*Sun StorEdge A7000 ROM Monitor Reference Manual*
Programming guide	*Sun StorEdge A7000 Online Exerciser User's Guide*
Diagnostic reference	*Sun StorEdge A7000 Online Exerciser Reference Manual*
Diagnostic reference	*Sun StorEdge A7000 External Interrupt and Clock Cable Diagnostic Reference Manual*
Diagnostic reference	*Sun StorEdge A7000 VME Quad Block Mux Channel Diagnostics Reference Manual*
Diagnostic reference	*Sun StorEdge A7000 VME SCSI Controller Diagnostics Reference Manual*
Diagnostic reference	*Sun StorEdge A7000 MEMORY CHANNEL IV System Diagnostic Reference Manual*
Diagnostic reference	*Sun StorEdge A7000 VME Dual Channel ESCON Diagnostics Reference Manual*

Sun Documentation on the Web

The `docs.sun.com` web site enables you to access Sun technical documentation on the Web. You can browse the `docs.sun.com` archive or search for a specific book title or subject at:

`http://docs.sun.com.`

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email your comments to us at:

`smcc-docs@sun.com`.

Please include the part number of your document in the subject line of your email.

CHAPTER 1

Diagnostics Overview

Introduction

System diagnostics are provided to verify the integrity of the Sun StorEdge A7000 Intelligent Storage Server System and to detect and isolate any hardware that is causing system malfunctions. Built-in tests of the hardware are run during the system power up sequence. The diagnostics run automatically and normally require no operator intervention.

Levels of Testing

The system diagnostics provide the following discrete levels of testing:

Hardware Confidence Tests

When the board set is powered up or reset, it automatically runs a sequence of hardware confidence tests. These tests are used to initialize and test the hardware on the processor and adaptor boards.

If the hardware confidence tests run successfully, the ROM Monitor starts execution and writes its start and copyright message to the operator's console.

As the confidence tests are executed, they are trapped by LED codes on the adaptor board and milestone characters displayed on the operator's console.

A failure in one of the hardware confidence tests indicates a serious malfunction that must be resolved before further system operation is possible. If a failure is detected, an error code is flashed in the LEDs on the adaptor board and no more milestone characters are displayed on the operator's console. If an error causes the ROM Monitor to hang, the LEDs display a code representing the operation in progress when the failure occurred.

Note – A DIAGNOSTIC MODE is provided for debugging and troubleshooting the board set. This mode is enabled by turning switch 4 on the adaptor board to the ON position. When this switch is ON, it starts the ROM Monitor with instruction cache disabled upon power up or reset. The normal mode of operation is with the switch in the OFF position. A message is displayed during the initialization sequence describing the status (enabled or disabled) of the instruction cache. Refer to the *Sun StorEdge A7000 Service Manual* for adaptor board switch information.

TABLE 1-1 shows the relationship between the testing operations, LED codes, and milestone characters for the processor board hardware confidence tests. The character is displayed before the specified operation is performed.

TABLE 1-1 Hardware Confidence Tests LED Codes

LED Code	Milestone	Testing Operation
		Initializes the IOPORT outputs (TS80 bit ports).
0xff		All processors set their processor status registers (psrs).
0x01		
0x02		
0x03		Initializes the parity inject registers. CPUs 1, 2, and 3 enable their instruction caches. CPUs 1, 2, and 3 go to spin in their ROM code.
0x04		CPU 0 initializes the SCCs (SCC1 and SCC2).
0x05		
0x06	a	Reads the first word in SRAM.
0x07	b	Checks the first word in SRAM.
0x08	c	Tests SRAM with two data patterns (0xc8c8c8c8 and 0x00000000).
0x09	d	Performs an SRAM address test.
0x0a	e	CPU 0 initializes cache (invalidates primary and secondary caches and sets up instruction, data MMU, and cache control registers).

TABLE 1-1 Hardware Confidence Tests LED Codes *(Continued)*

LED Code	Milestone	Testing Operation
0x0b	f	CPU 0 enables instruction cache (`enable_my_inst_cache`).
0x0c		
0x0d	g	Zeros the SRAM.
0x0e		
0x0f	h	Zeros the BSS area.
0x10	i	Tests the first megabyte of DRAM with two data patterns (0xc8c8c8c8 and 0x00000000).
0x11	j	Performs DRAM address test (first megabyte only).
	k	Initializes LBBA and control register.
	l	Tests first word in each bank of processor board memory.
0x12	m	Tests the ability of the free running counter to increment.
0x13	n	Branches to the C environment.
	o	Initializes some global variables.
0x14		
0x15		
0x16		
0x17	p	Clears the rest of Adaptor board DRAM.
0x18		
0x19	q	Initializes the real time clock chip address variables.
0x1a		Initializes global `slot_has_board[ ]` for all present expansion memory boards.
0x1b		Initializes the node state check block (Load `node_state` structure into DRAM from NVRAM and verify checksum).
0x1c		Initializes the global variables to their starting conditions. Initializes the error log buffers in DRAM
0x1d		Initializes the error log buffers in NVRAM.

TABLE 1-1 Hardware Confidence Tests LED Codes *(Continued)*

LED Code	Milestone	Testing Operation
0x1e	r	Initializes and enables CPU 0 interrupts.
0x1f	s	Initializes the serial channel for interrupt-driven I/O.
	t	Prints the ROM Monitor banner. Initializes the SCSI chips. Starts the other CPUs and sends them to `cpu_table`. Clears processor board memory. Clears expansion memory, if present. Queries the SCSI Bus. Compares the status of the SCSI configuration with the stored SCSI configuration. Starts Node Control Flow (displays `ROM >>` prompt).

Built-in Self Tests (BISTs)

The Built-in Self Tests, or BISTs, reside in ROM on the processor board and provide additional built-in testing for the CPUs, memory, and processor board functions and interfaces. These tests run automatically during a power up or reset sequence unless automatic testing is disabled in the NVRAM or the operator initiates a bypass sequence. The Built-in Self Tests may also be run using the ROM Monitor `test` command. The type of testing is determined by either the parameters in NVRAM or the arguments entered with the `test` command.

The Built-in Self Tests support the following groups of tests:

- Core logic tests (core)
- Quick memory tests (qmem)
- Long memory tests (lmem)
- Loopback tests (lpbk)
- Cache memory tests (cache)
- Error correction code tests (ecc and edramecc)
- SCSI interface tests (scsi)

Note – All memory testing is automatically applied to Sun-supplied external memory boards.

The ROM Monitor `test` command also provides a `slots` test group. These tests run Extended Diagnostics on specific boards.

Refer to Chapter 2 for additional BISTs information.

Extended Diagnostics

The Extended Diagnostics reside in the header partition of the system disk. These diagnostics allow you to test additional boards configured on the system backplane.

Like the Built-in Self Tests, the Extended Diagnostics also run during a power up or reset sequence if they are configured in NVRAM. An operator initiated bypass sequence, the disabling of automatic testing, or the failure of the core group of Built-in Self Tests will inhibit running the Extended Diagnostics. If automatic testing is enabled and the Built-in Self Tests execute successfully, the ROM Monitor calls the specified Extended Diagnostics.

Extended Diagnostics may also be run using the ROM Monitor `test` command or in interactive mode. Refer to Chapter 3 for additional information about Extended Diagnostic execution.

The following Extended Diagnostics are described in separate manuals:

- External Interrupt and Clock Cable
- VME Quad Block Mux Channel
- VME SCSI Controller
- VME Dual Channel ESCON
- MEMORY CHANNEL IV System

Online Exerciser Tests

Online Exerciser tests are used to verify the operation of the Sun StorEdge A7000 Intelligent Storage Server System under normal operating conditions. Unlike the other diagnostics, which run during a power up or reset sequence, the Online Exerciser tests run under the software operating system.

The Online Exerciser contains groups of tests used to exercise both the software and the hardware under normal operation. This diagnostic tool can be configured to test any type of hardware including remote hosts configured with Ethernet on a TCP/IP network.

Online Exerciser tests are described in the *Sun StorEdge A7000 Online Exerciser Reference Manual*.

CHAPTER 2

Built-in Self Tests (BISTs)

Introduction

The ROM Monitor Built-in Self Tests (BISTs) reside in ROM on the processor board and provide built-in testing for the CPUs, memory, and processor board functions and interfaces. If an error is detected, these tests identify the source of the error and, in many cases, the board causing the error. This provides faster fault isolation and a lower mean time to repair for the types of faults detected with the Built-in Self Tests.

The Built-in Self Tests (BISTs) are used to verify the functionality of the various processor and adaptor board components:

- CPUs and cache memory
- DRAM and SRAM
- Timers and interrupts
- Real-time clock
- Counter I/O controllers (CIO)
- Ethernet interface and LAN controller
- VME interrupter
- NVRAM memory
- Error detection and correction logic
- SCSI interface

The Built-in Self Tests are grouped into the following test categories:

- Core logic (core) tests
- Quick memory (qmem) tests
- Long memory (lmem) tests
- Loopback (lpbk) tests
- Cache memory (cache) tests
- Error correction code (ecc and edramecc) tests
- SCSI interface (scsi) tests

Note – The memory tests are also applied to the Sun-supplied expansion memory boards.

Each test group mnemonic may be used as an option with the ROM Monitor `test` command.

This chapter contains the following topics:

Test Execution

The Built-in Self Tests are executed automatically during a processor board power up or reset sequence unless the operator initiates a bypass sequence or automatic testing is disabled in the NVRAM. The group of tests executed is determined by the parameters in NVRAM. You can modify the default test selection with the ROM Monitor `setconfig tlevel` command.

These tests can also be executed using the ROM Monitor `test` command. Either specify the level of testing on the command line with the `test` command or use the `setconfig tlevel` command to change the NVRAM parameters prior to executing the `test` command.

Refer to *Automatic Testing Procedures* on page 2-4 for directions on using ROM Monitor commands to manipulate automatic testing options. The individual ROM Monitor commands are described in the *Sun StorEdge A7000 ROM Monitor Reference Manual*.

Error Reporting and Fault Isolation

If a test fails during the automatic execution of the Built-in Self Tests, the ROM Monitor displays a message on the operator's console identifying the board reporting the error. This message is displayed in one of the following formats:

```
**********************************************************
Card in slot n (cardtype) failed diagnostics
**********************************************************
**********************************************************
Card in slot n (cardtype) failed loopback diagnostics
**********************************************************
**********************************************************
Extended Memory Diagnostic Failure (can't isolate)
**********************************************************
```

Variable	Description
n	Identifies the slot location of the board reporting the error.
cardtype	Identifies the type of board reporting the error: `CPU` or `MEMORY`.

The first two messages indicate the error is on the specified CPU or memory board. In the case of expansion memory testing, the memory algorithms encountered an error and attempted to isolate the failure by determining the location of the failing address with respect to the memory configuration.

The third message indicates the memory algorithms encountered an error but the memory card configuration can not be determined. This type of problem may be caused by a hardware error or because the failing algorithm is not memory card specific (for example, an address line failure).

Note – The identification of the failing board is a recommendation from the ROM Monitor. In some cases, faulty logic on one board may cause another board to fail. When this occurs, it is impossible for the ROM Monitor to accurately pinpoint the source of the failure.

Automatic Testing Procedures

Use the following procedures to perform these operations:

- Bypass automatic testing
- Enable or disable automatic testing
- Select automatic testing levels
- Save changes to NVRAM
- Run the Built-in Self Tests with the `test` command

All the commands used in this section are described in the *Sun StorEdge A7000 ROM Monitor Reference Manual*.

Bypassing Automatic Testing

If automatic testing is enabled, the ROM Monitor displays the following prompt on the operator's console during a power up or reset sequence:

```
Automatic test enabled, 3 seconds to break
```

Press any key within three seconds to bypass running the Built-in Self Tests.

To permanently bypass automatic testing on every power up or reset sequence, use the monitor `setconfig autotest` command. Refer to *Enabling or Disabling Automatic Testing* on page 2-5 for additional information.

Note – When autotest is disabled, only the hardware confidence tests are executed.

Bypassing automatic testing leaves the system in ROM Monitor mode (`ROM >>`).

Enabling or Disabling Automatic Testing

Automatic testing on power up or reset is enabled and disabled with the ROM Monitor `setconfig autotest` command.

Enabling Automatic Testing

Enter the following command line to enable automatic testing on power up or reset:

```
ROM >> setconfig autotest enable
```

Disabling Automatic Testing

Enter the following command line to disable automatic testing:

```
ROM >> setconfig autotest disable
```

Displaying the Autotest State

Enter the following command line to display the current autotest state:

```
ROM >> readconfig autotest
```

The monitor displays the current state in the following format:

```
autotest: {enabled|disabled}
```

Selecting Automatic Testing Levels

Automatic testing is done at different levels, any of which can be enabled or disabled with the ROM Monitor `setconfig tlevel` command. Levels can be enabled and disabled on the same command line by using a plus (+) to enable the level and a minus (-) to disable the level. Levels not specified remain in their current states. More than one level can be specified on the command line.

The following example enables core logic and quick memory testing and disables long memory testing:

```
ROM >> setconfig tlevel +core +qmem -lmem
```

To display the current autotest levels, enter the following command line:

```
ROM >> readconfig tlevel
```

The ROM Monitor displays the selected levels in the following format:

```
autotest levels selected: lvl ... lvl
```

Variable	Description
lvl	Specifies the enabled level: `core`, `lmem`, `qmem`, `slots`, `lpbk`, `cache`, `ecc`, `edramecc`, `scsi`.

Note – The `slots` test level is used to automatically run extended diagnostics on additional boards configured in the system including VMEbus Controllers and MEMORY CHANNEL boards.

Saving Automatic Testing Changes

The automatic testing changes made with the ROM Monitor commands are not effective beyond the current session unless they are saved to NVRAM. To save the changes, enter the ROM Monitor `nvram` command in the following format and respond to the prompt as shown:

```
ROM >> nvram accept
Are you sure you want to save the configuration? (y/n) y
nvram modified
ROM >>
```

Using the `test` Command

The Built-in Self Tests (BISTs) may be executed from the ROM Monitor at any time by entering the `test` command. Either enter the `test` command after selecting the desired test groups with the ROM Monitor `setconfig tlevel` command or enter the test groups and the `test` command on the same command line as follows:

```
ROM >> test lvl ... lvl
```

Variable	Description
lvl	Specifies the enabled level: `core`, `lmem`, `qmem`, `slots`, `lpbk`, `cache`, `ecc`, `edramecc`, `scsi`.

Start Messages

If automatic testing is enabled, the ROM Monitor displays the following messages to identify the start of testing and the selected testing levels:

```
Slot n: Running Built In Tests
Test levels selected: lvl...lvl
Testing CPU Card
```

Variable	Description
n	Specifies the processor board slot number.
lvl	Specifies the enabled testing levels.

Prior to executing each individual Built-in Self Test, the ROM Monitor displays a start message on the operator's console identifying the test under execution. The start message is displayed in the following format:

```
Starting testname test:
```

Variable	Description
n	Specifies the processor board slot number.
lvl	Specifies the enabled testing levels.
testname	Identifies the name of the test being executed.

Pass and Error Count Messages

After executing all the selected test groups, the ROM Monitor displays the number of passes through the test list and the number of errors encountered in the following format:

```
PASSCOUNT x,     ERRORS y
```

Variable	Description
x	Specifies the number of passes made through the test list.
y	Specifies the number of errors encountered during *x* passes.

Core Logic Tests (core)

The core logic group of tests are used to test the operation of the CPUs, Counter I/O controllers (CIO), Ethernet interface and controller, VME interrupts, EPROM, NVRAM and other processor board components and functions. There is also a test in this group that verifies the Parameter PROMs on all configured expansion memory boards.

This group includes the following tests:

EPROM Checksum Test

During this test, the contents of EPROM are read and a checksum value is calculated. This calculated value is compared to the checksum value stored in the last location in EPROM.

Once started, this test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
CPU n, Read checksum aaaaaaaa calculated eeeeeeee
```

Variable	Description
n	Specifies the number of the CPU under test.
aaa...aaa	Specifies the checksum value read from EPROM.
eee...eee	Specifies the expected checksum value calculated by the program.

If the checksum values are equal, the test appends `...PASS` to the message. If the checksum values are different, the test appends `...FAIL` to the message.

Parameter PROM Checksum Test

During this test, the contents of the Parameter PROM are read and a checksum value is calculated. This calculated value is compared to the checksum value stored in the last four locations in the Parameter PROM.

Once started, this test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
CPU n, calc checksum eeeeeeee, stored aaaaaaaa
```

Variable	Description
n	Specifies the number of the CPU under test.
eee...eee	Specifies the expected checksum value calculated by the program.
aaa...aaa	Specifies the checksum value read from Parameter PROM.

If the checksum values are equal, the test appends `...PASS` to the message. If the checksum values are different, the test appends `...FAIL` to the message.

DRAM Parity Test

This test verifies the operation of the DRAM parity detection logic on the processor board. The test uses the parity fault injection lines to force a parity error, accesses DRAM, and verifies that the parity error interrupt from the TS80 is handled correctly.

Once started, the test displays a start message on the operator's console in the standard format.

If the test passes, `...PASS` is displayed on the operator's console below the start message. If the test fails, `...FAIL` is displayed on the operator's console and the test displays one of the following error messages:

- If the test is unable to force a parity error, it displays:

```
DRAM Parity test: did not detect PE forcing bad (loc) parity
```

Variable	Description

loc	Specifies one of the following messages identifying the location (byte and bus) where the parity fault was injected:
	`Local Data Bus D24-31`
	`Local Data Bus D16-23`
	`Local Data Bus D8-15`
	`Local Data Bus D0-7`

- If the test was unable to clear the forced parity error condition, it displays:

```
DRAM Parity test: Couldn't clear force bad (loc) parity
Testing bank (n), addr (aaaaaaaa), PIN bits are (yyyyyyyy)
```

Variable	Description

loc	Specifies one of the following messages identifying the location (byte and bus) where the parity fault was injected:
	`Local Data Bus D24-31`
	`Local Data Bus D16-23`
	`Local Data Bus D8-15`
	`Local Data Bus D0-7`

Variable	Description
n	Specifies the bank of DRAM under test. There are four banks.
aaaaaaaa	Specifies the address used during the parity access.
yyyyyyyy	Specifies the input port values read from the TS80. This contains the error status fed into interrupt 31.

SRAM Parity Test

This test verifies the operation of the CMMU parity error detection logic on the processor board. The test forces a parity error on the I/O bus, accesses SRAM, and verifies that the parity error interrupt from the TS80 is handled correctly.

Once started, the test displays a start message on the operator's console in the standard format.

If the test passes, `...PASS` is displayed on the operator's console below the start message. If the test fails, `...FAIL` is displayed on the operator's console and the test displays one of the following error messages:

- If the test is unable to force a parity error, it displays:

```
No data access exception detected forcing bad (loc) parity
Test addr (aaaaaaaa), PIN bits are (yyyyyyyy)
```

Variable	Description

loc	Specifies one of the following messages identifying the location (byte and bus) on the I/O bus where the parity fault was injected:
	`I/O Data Bus D0-7`
	`I/O Data Bus D8-15`
	`I/O Data Bus D16-23`
	`I/O Data Bus D24-31`

aaaaaaaa	Specifies the address used during the parity access.
yyyyyyyy	Specifies the input port value read from the TS80 PIN (Port INput) register.

■ If the TS80 status on the input port is incorrect, the test displays:

```
Bad status on ts80 (pin is (xxxxxxxx), exp (yyyyyyyy)),
forcing bad (loc) parity
```

Variable	Description
xxxxxxxx	Specifies the address used during the parity access.
yyyyyyyy	Specifies the input port value read from the TS80 PIN (Port INput) register.

loc	Specifies one of the following messages identifying the location (byte and bus) on the I/O bus where the parity fault was injected:
	`I/O Data Bus D0-7`
	`I/O Data Bus D8-15`
	`I/O Data Bus D16-23`
	`I/O Data Bus D24-31`

■ If the error recurs after it has been cleared, the test displays:

```
SRAM Parity test: Couldn't clear force bad (loc) parity
```

Variable	Description

loc	Specifies one of the following messages identifying the location (byte and bus) on the I/O bus where the parity fault was injected:
	`I/O Data Bus D24-31`
	`I/O Data Bus D16-23`
	`I/O Data Bus D8-15`
	`I/O Data Bus D0-7`

TS80 Cross Processor Interrupt Test

This test is used to verify the ability of the CPUs to interrupt one another. Each CPU generates a cross processor interrupt to each of the other CPUs. Each configured CPU becomes the master and its ability to interrupt all of the other configured CPUs is verified.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message for each configured CPU:

```
CPU x interrupting: CPU 0, CPU 1, CPU 2, CPU 3
```

Variable	Description
x	Specifies the number of the master CPU generating the interrupt.

Note – As each CPU is used to field the interrupt, the CPU number is appended to the message.

If the test executes successfully, `...PASS` is appended to each message.

If an error is detected, the test appends `...FAIL` to the message. It then displays:

```
TS80 X-cpu int test No interrupt. cpu x couldn't int y
```

Variable	Description
x	Specifies the number of the master CPU generating the interrupt.
y	Specifies the number of the CPU to be interrupted.

This test fails if no interrupt occurs or the wrong processor is interrupted.

TS80 External Interrupt Test

This test is used to check each of the eight external interrupts using the internal loopback capability of the TS80. The interrupts are tested with all CPUs.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message for each configured CPU:

```
CPU n testing xint 0, 1, 2, 3, 4, 5, 6, 7
```

Variable	Description
n	Specifies the number of the CPU used to field the interrupts.

Note – As each external interrupt is used, the interrupt number is appended to the message.

If the test executes successfully, `...PASS` is appended to each message. If an error is detected, the test appends `...FAIL` to the message. It then displays:

```
TS80 xint test No interrupt, cpu a, int i
```

Variable	Description
a	Specifies the number of the CPU selected to field the interrupt.
i	Specifies the expected interrupt.

TS80 Interrupt CPU Test

This test verifies the ability of each local timer to generate an interrupt to each CPU. Each timer is loaded with a specific time period and started. The test verifies that an interrupt is sent to the CPU when the count expires. Each CPU is tested using each timer.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message for each configured CPU:

```
Cpu = x, Timer: 0, 1, 2, 3, 4, 5
```

Variable	Description
x	Specifies the selected CPU.

Note – The timer number is appended to the message as each timer is used.

If the test passes, `...PASS` is appended to each message.

If an error is detected, the message `...FAIL` is appended to the message and the test displays one of the following error messages:

```
TS80 int cpu test no int from cpu x. timer y
TS80 int cpu test int from cpu c sb x. timer y
```

Variable	Description
x	Specifies the selected CPU.
y	Specifies the selected timer.
c	Specifies the CPU generating the interrupt.

TS80 Timer Interrupt Test

This test verifies the ability of each timer to time out when the count reaches the end of a specified time period. The timer is loaded with a specific period and started. The test then polls the count-complete flag until the timer times out. This test is performed using all the CPUs.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message for each configured CPU:

```
CPU: x, timer 0, 1, 2, 3, 4, 5
```

Variable	Description
x	Specifies the selected CPU.

Note – The timer number is appended to the message as each timer is tested.

If the test is successful, `...PASS` is appended to each message. If an error is detected, the test appends `...FAIL` to the message and displays one of the following error messages:

```
TS80 Timer int test Timer x timeout flag not reset
TS80 Timer int test Timer x count complete not flagged
```

Variable	Description
x	Specifies the number of the timer under test when the error occurred.

Real Time Clock Increment Test

This test verifies the ability of the Real-time clock to increment. The test reads the Real-time clock, waits one second according to the CIO timer, and verifies that the Real-time clock value incremented.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
CPU n: First read of seconds = xx, second read = yy
```

Variable	Description
n	Specifies the selected CPU.
xx	Specifies the count received the first time the Real-time clock was read.
yy	Specifies the count received the second time the Real-time clock was read.

If the test executes successfully, `...PASS` is appended to the message. If an error is detected, the test appends `...FAIL` to the message and displays one of the following error messages:

```
RTC increment illegal seconds value
RTC increment test clock stopped
RTC increment test clock incremented too fast
```

Note – This test will still pass if the first and second values read from the clock are two seconds apart. This is caused by the delays associated with executing the code on top of the one second delay.

NVRAM Memory Test

This test verifies the integrity of the Nonvolatile Random Access Memory (NVRAM) on the processor board. The test data area is filled with a data pattern that is written into NVRAM. The data is read back from NVRAM and verified.

Once started, this test displays a start message on the operator's console in the standard format.

If the test executes successfully, `...PASS` is displayed on the operator's console below the start message. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays an error message in the following format:

```
Nvram memory failure, addr x, is a, sb e
```

Variable	Description
x	Specifies the failing address in NVRAM.
a	Specifies the data value read from the NVRAM location.
e	Specifies the expected contents of the NVRAM location.

This test also performs an NVRAM battery check. If the battery level is low, an appropriate message is displayed.

CIO Interrupt Test

This test performs an interrupt test on a register in each of the Counter I/O controllers (CIO) on the processor board. The test enables interrupts in the master control register and sets the IE bit in the control/status register under test. The test then sets the IP bit to force an interrupt. The CPU then executes a wait loop. When the loop terminates, the test verifies that the interrupts have been serviced.

Once started, this test displays a start message on the operator's console in the standard format. The test then displays the following message for each CIO tested:

```
CPU x: Testing CIO n: Port A  Port B
```

Variable	Description
x	Specifies the number of the selected CPU.
n	Specifies the CIO under test.

Note – The port name is appended to the message as each port is tested.

If the test passes, ...PASS is appended to each message. If an error is detected, ...FAIL is appended to the message and the test displays the following error message:

```
CIO interrupt no interrupt received cpu x  n  y
```

Variable	Description
x	Specifies the selected CPU.
n	Specifies the failing CIO (0 or 1).
y	Specifies the failing port.

Ethernet Interrupt Test

This test verifies the ability of the LAN controller to generate an interrupt. The Ethernet controller executes a NOP (No Operation) command and the test verifies that an interrupt was received.

Once started, this test displays a start message on the operator's console in the standard format.

If the test executes successfully, ...PASS is displayed on the operator's console below the start message. If an error is detected, the test displays ...FAIL on the operator's console. The test then displays one of the following error messages:

- If the test is unable to initialize the network interface, it displays:

```
Couldn't initialize network interface
```

- If the Ethernet controller fails to execute the NOP command, the test displays:

```
Ethernet controller failed to execute NOP command
```

- If no interrupt is received from the LAN controller, the test displays:

```
Lan Controller Failed to generate interrupt
```

Ethernet Controller Chip Test

This test verifies the ability of the Ethernet controller to execute a DIAGNOSE command. The Ethernet controller executes a DIAGNOSE command and the test verifies that the command executed successfully.

Once started, this test displays a start message on the operator's console in the standard format.

If the test executes successfully, ...PASS is displayed on the operator's console below the start message. If an error is detected, the test displays ...FAIL on the operator's console. The test then displays one of the following error messages:

- If the test is unable to initialize the network interface, it displays:

```
Couldn't initialize network interface
```

- If the Ethernet controller fails to execute the DIAGNOSE command, the test displays:

```
Ethernet controller failed to execute diagnose command
```

- If the DIAGNOSE command fails, the test displays:

```
Ethernet controller reports failure of diagnose command
```

Ethernet Internal Loopback Test

This test verifies the ability of the LAN controller to send and receive packets internally.

Once started, this test displays a start message on the operator's console in the standard format.

If the test executes successfully, `...PASS` is displayed on the operator's console below the start message. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays one of the following error messages:

- If the test is unable to initialize the network interface, it displays:

```
Couldn't initialize network interface (status stat)
```

Variable	Description
stat	specifies the controller status information.

- If the test is unable to configure internal loopback mode, it displays:

```
Couldn't configure 82596 for 82596 internal mode
```

- If an error occurs when receiving a frame, the test displays:

```
Error receiving frame (status stat)
```

Variable	Description
stat	Specifies the controller status information.

- If the test times out waiting to receive a loopback frame, it displays:

```
Timed out waiting to receive loopback frame
```

- If the test detects an address error in the received packet, it displays:

```
Destination Address error in received packet
```

■ If a data error is detected, the test displays:

```
Received data != transmitted data
Data in byte x is y, sb z
```

Variable	Description
x	Specifies the failing data byte.
y	Specifies the actual data received.
z	Specifies the expected data value.

Ethernet External Loopback Test

This test verifies the ability of the LAN controller to send and receive packets through the Ethernet Serial Interface chip.

Once started, this test displays a start message on the operator's console in the standard format.

If the test executes successfully, `...PASS` is displayed on the operator's console below the start message. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays one of the following error messages:

■ If the test is unable to initialize the network interface, it displays:

```
Couldn't initialize network interface (status stat)
```

Variable	Description
stat	Specifies the controller status information.

■ If an error occurs when receiving a frame, the test displays:

```
Error receiving frame (status stat)
```

Variable	Description
stat	Specifies the controller status information.

- If the test is unable to configure external loopback mode, it displays:

```
Couldn't configure 82596 for 82501 external mode
```

- If the test times out waiting to receive a loopback frame, it displays:

```
Timed out waiting to receive loopback frame
```

- If the test detects an address error in the received packet, it displays:

```
Destination Address error in received packet
```

- If a data error is detected, the test displays:

```
Received data != transmitted data
Data in byte x is y, sb z
```

Variable	Description
x	Specifies the failing data byte.
y	Specifies the actual data received.
z	Specifies the expected data value.

VME Interrupt Test

This test verifies the operation of the VME interrupter. The following conditions are tested:

- A VME interrupt can be generated for each of the seven interrupt positions. This tests both the interrupt connections and the VME IRQ level bits in the CIO.
- The acknowledge interrupt mechanism functions properly.
- The interrupt vector functions using a different vector data pattern for each interrupt.

Once started, this test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
CPU n testing vme interrupt: 1, 2, 3, 4, 5, 6, 7
```

Variable	Description
n	Specifies the number of the selected CPU.

Note – The VME interrupt number is appended to the message as each interrupt is tested.

If the test executes successfully, `...PASS` is appended to the message. If an error is detected, the test appends `...FAIL` to the message and displays one of the following error messages:

- If an expected VME interrupt fails to occur, the test displays:

```
No VME Interrupt on level p
```

Variable	Description
p	Specifies the failing interrupt level.

- If the interrupt vector is incorrect, the test displays:

```
Bad vector read from IACK cycle, is a, sb e
```

Variable	Description
p	Specifies the failing interrupt level.
a	Specifies the vector data pattern received.
e	Specifies the expected vector data pattern.

VME Interrupter Busy Test

This test verifies the VME Interrupter Busy signal can be asserted and removed on request.

Once started, this test displays a start message on the operator's console in the standard format.

If the test executes successfully, `...PASS` is displayed on the operator's console below the start message. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays one of the following error messages:

- If the VME Interrupter Busy signal is asserted when it should not be asserted, the test displays:

```
VME interrupter busy is asserted (0) but shouldn't be
```

- If the test is unable to assert the VME Interrupter Busy signal, it displays:

```
VME interrupter busy cannot be asserted (0)
```

- If the test is unable to remove the VME Interrupter Busy signal, it displays:

```
VME interrupter busy can not be de-asserted (1)
```

SCC Internal Loopback Test

This test verifies the ability of the Serial Communications Controller (SCC) to transmit and receive data internally.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message as each port is tested:

```
port: n
```

Variable	Description
n	Specifies the number of the port under test.

If the port tests successfully, `....PASS` is appended to the message. If the port is reserved as the console port, `....RESERVED (console port)` is appended to the message. If an error is detected, `....FAIL` is appended to the message and the test displays one of the following error messages:

- If a timeout occurs while waiting to receive data, the test displays:

```
Port n: SCC a Channel b: timeout
```

Variable	Description
n	Specifies the number of the port under test.
a	Specifies the SCC under test.
b	Specifies the channel under test.

- If an incorrect character is received, the test displays:

```
Port n: SCC a Channel b: read back char number x - y
```

Variable	Description
n	Specifies the number of the port under test.
a	Specifies the SCC under test.
b	Specifies the channel under test.
x	Specifies the number of the failing character.
y	Specifies the failing character.

SCSI Chip Regs Test

This test verifies the following SCSI chip registers:

- icode
- istat
- dmastat
- stat0
- stat2
- ID

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
CPU n reading icode istat dmastat stat0 stat2 ID
```

Variable	Description
n	Specifies the number of the selected CPU.

If the test executes successfully, `...PASS` is appended to the message.

Note – As each register is tested, the register name is appended to the message.

This test does not produce error messages. A failure may result in a parity error indication.

Expansion Memory Parameter PROM Checksum Test

This test checks each configured memory board for the following conditions:

- The board contains a Sun StorEdge A7000 Parameter PROM.
- The Parameter PROM identifies the board as a Sun StorEdge A7000 memory board.
- The PROM checksum value is correct.

All Sun StorEdge A7000 boards designed by Sun Microsystems, Inc. contain a Parameter PROM with the same standard format. The data is protected with a two's complement checksum stored in the last four bytes of the PROM. PROMs of the same size are not required. The size is determined by the first locations in the PROM.

Once started, this test displays a start message on the operator's console in the standard format. If the test executes successfully, the test displays `...PASS` on the operator's console below the start message. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays one of the following error messages:

- If the Parameter PROM indicates the incorrect PROM type, the test displays:

```
Memory board (n) invalid prom type
Prom type expected (eeeeeeee), Prom type read=(aaaaaaaa)
```

Variable	Description
n	Specifies the number of the memory board.
eeeeeeee	Specifies the expected hexadecimal prom type value.
aaaaaaaa	Specifies the hexadecimal prom type value read from PROM.

This message indicates either a bad device or the wrong device is installed in the PROM socket.

- If the Parameter PROM indicates the incorrect board type, the test displays:

```
Memory board (n) invalid board type
Board type expected=(eeeeeeee), Prom type read=(aaaaaaaa)
```

Variable	Description
n	Specifies the number of the memory board.
eeeeeeee	Specifies the expected board type value.
aaaaaaaa	Specifies the board type value read from PROM.

This message indicates that a bad device is installed in the PROM socket.

- If the incorrect checksum value is read from the Parameter PROM, the test displays:

```
Memory board (n) bad biprom checksum
Checksum calculated=(eeeeeeee), Checksum read=(aaaaaaaa)
```

Variable	Description
n	Specifies the number of the memory board.
eeeeeeee	Specifies the two's complement checksum value calculated by the test.
aaaaaaaa	Specifies the two's complement checksum value read from PROM.

This message indicates that a bad device is installed in the PROM socket.

Watchdog Abort Test

This test forces an interrupt from CIO1 (timers 1 and 2) to simulate a watchdog timeout. The watchdog timeout should cause an abort interrupt. The resulting interrupt is handled through the interrupt 31 handler.

Once started, this test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
CPU n: Testing CIO 1:
```

Variable	Description
n	Specifies the number of the selected CPU.

If the test executes successfully, `...PASS` is appended to the message. If an error is detected, `...FAIL` is appended to the message and the test displays the following error message:

```
Watchdog abort no interrupt received cpu n CIO 1 Port p
```

Variable	Description
n	Specifies the number of the selected CPU.
p	Specifies the failing port.

System Bus Parity Test

This test is used to verify the operation of the 410 System Bus data parity error detection logic. The test sets the 410 System Bus parity error inject bit (Bit 6) in the processor control register to force a parity error. The test then reads the processor board memory and verifies that a parity error was detected on the CPU 0 data bus and reported by a Non-Maskable Interrupt (NMI). The test then writes to the processor board memory and verifies that a parity error detected in the 410 System Bus data field inhibits writing the data and generates a data access exception.

Once started, the test displays a start message on the operator's console in the standard format. If the test executes successfully, `...PASS` is displayed on the operator's console below the start message. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays one of the following error messages:

- If the test forces a parity error on a read operation but no parity error is detected on the System Data Bus, it displays:

```
No PE detected after forcing bad System Data Bus parity on a read
```

- If the test forces a parity error on a write operation but no parity error is detected on the System Data Bus, it displays:

```
No PE detected after forcing bad System Data Bus parity on a write
```

- If memory fails to inhibit a write operation when a forced parity error occurs, the test displays:

```
memory failed to inhibit write when forced PE occurred
```

Quick Memory Tests (qmem)

The quick memory tests provide a quick verification of the DRAM on the processor board and the expansion memory boards. This group includes the following tests:

- DRAM Data=Address Test—page 2-33
- DRAM Address Line Test—page 2-35
- VMEbus Data=Address Test—page 2-36
- Expansion Memory Parity Test—page 2-38
- Expansion Memory Interleave Test—page 2-42
- Multi Data Size Test—page 2-43

DRAM Data=Address Test

This test writes the address of a location as data into the same location. The test writes the locations from the starting address to the ending address and verifies the data written in the same sequence to ensure that patterns at higher addresses do not corrupt data at lower addresses.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message identifying the memory test range and number of CPUs configured:

```
DRAM DATA=ADDR aaaaaaaa-bbbbbbbb with x cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test range.
bb...bb	Specifies the ending address of the memory test range.
x	Specifies the number of CPUs configured.

The program then displays the following message for each configured CPU to identify the memory test range associated with each CPU:

```
cpu n testing aaaaaaaa bytes starting at bbbbbbbb
```

Variable	Description
n	Specifies the CPU number.
aa...aa	Specifies the number of bytes in the memory test range assigned to this CPU.
bb...bb	Specifies the starting address of the memory test range assigned to this CPU.

Prior to filling memory with the data pattern, the test displays `fill` below the start messages. After the memory test range is filled with data, the test appends `chk` to the message and begins reading and verifying data. If the test executes successfully, the program then appends `...PASS` to the message.

If an error is detected, the test appends `...FAIL` to the message and displays the following error message:

```
DATA=ADDRESS test cpu x exp: ee, rcvd: aa, at addr bbbbbbbb
```

Variable	Description
x	Specifies the number of the CPU used for testing purposes.
ee	Specifies the expected data pattern.
aa	Specifies the actual data received.
bbbbbbbb	Specifies the failing memory address.

DRAM Address Line Test

This test verifies that none of the DRAM address lines affect any other address lines. The test writes a pattern to an address and writes a different pattern to an address that has a one bit variation from the first address. If the data in the first address changes to the second pattern, the address lines may be tied together. The test performs this operation for all locations in the test range. The test makes two passes through the memory range. First, with most address bits as zeros, the test walks a one through the address range. Second, with most of the address bits as ones, the test walks a zero through the address range.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
DRAM address line size, from aaaaaaaa-bbbbbbbb
```

Variable	Description
size	Specifies the address size under test: BYTE, WORD or LONG. The message is repeated once for each size.
aa...aa	Specifies the starting address of the memory test range.
bb...bb	Specifies the ending address of the memory test range.

If the test executes successfully, ...PASS is appended to the message. If an error is detected, ...FAIL is appended to the message and the test displays one of the following error messages:

- If the error is associated with stuck or shorted address lines, the test displays:

```
DRAM addr line size test: Addr lines Aaa and Abb tied
```

Variable	Description
size	Specifies the address size under test when the error was detected.
aa	Specifies one of the failing address lines.
bb	Specifies the other failing address line.

- If the error is associated with a hole in memory or a data bit problem, the test displays:

```
DRAM address line size test: bad mem at dddddddd: exp ee: rcvd: aa
```

Variable	Description
size	Specifies the address size under test when the error was detected.
dd...dd	Specifies the failing memory location.
ee	Specifies the expected data pattern read from memory.
aa	Specifies the actual data pattern received from memory.

VMEbus Data=Address Test

This test writes the address of a location as data into the same location. The test writes the locations from the starting address to the ending address and verifies the data written in the same sequence to ensure that patterns at higher addresses do not corrupt data at lower addresses.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message identifying the VMEbus address test range and number of CPUs configured:

```
VBUS DATA=ADDR aaaaaaaa-bbbbbbbb with x cpus
```

Variable	Description
aa...aa	Specifies the starting address of the VMEbus address test range.
bb...bb	Specifies the ending address of the VMEbus address test range.
x	Specifies the number of CPUs configured.

The program then displays the following message for each configured CPU to identify the VMEbus address test range associated with each CPU:

```
cpu n testing aaaaaaaa bytes starting at bbbbbbbb
```

Variable	Description
n	Specifies the CPU number.
aa...aa	Specifies the number of bytes in the VMEbus address test range assigned to this CPU.
bb...bb	Specifies the starting address of the VMEbus address test range assigned to this CPU.

Prior to filling memory with the data pattern, the test displays `fill` below the start messages. After the VMEbus address test range is filled with data, the test appends `chk` to the message and begins reading and verifying data. If the test executes successfully, the program then appends `...PASS` to the message.

If an error is detected, the test appends `...FAIL` to the message and displays the following error message:

```
DATA=ADDRESS test cpu x exp: ee, rcvd: aa, at addr bbbbbbbb
```

Variable	Description
x	Specifies the number of the CPU used for testing purposes.
ee	Specifies the expected data pattern.
aa	Specifies the actual data received.
bbbbbbbb	Specifies the failing VMEbus address.

Expansion Memory Parity Test

This test verifies the ability of the expansion memory to detect and report the following conditions:

- Parity errors made over the VMEport
- Parity errors made on the Local Bus address bus
- Parity errors made on the Local Bus control bus
- Parity errors made on the Local Bus data bus

The test contains the following subtests:

- Memory Board Address Bus Parity
- Memory Board Control Bus Parity
- Memory Board CSR Data Parity

Once started, the test displays a start message on the operator's console in the standard format. If the test passes, `...PASS` is displayed on the operator's console. If an error is detected, the test displays `...FAIL` on the operator's console followed by an error message identifying the type of failure and the test or subtest where the error occurred.

Error messages are displayed in the following format:

header
`board: (`*n*`), control: (`*cccccccc*`), status:(`*ssssssss*`),address:(`*aaaaaaaa*`)`
string

Variable	Description
header	Specifies the name of the test or subtest where the error occurred.
n	Specifies the number of the memory board under test.
cccccccc	Specifies the contents of the memory board control register.
ssssssss	Specifies the contents of the memory board status register.
aaaaaaaa	Specifies the test address.
string	Specifies an optional message used to provide additional error information.

The memory board uses the expansion bus parity input to report any detected errors to the CPU. The error is latched in the CPU and used to generate the parity error interrupt. If the interrupt is expected but not detected, the following error message is displayed:

```
local expansion bus parity error failed to generate an interrupt
```

Status Error Messages

The following error messages are associated with the failure to set or reset individual bits in the memory board status register:

- If the expansion memory board under test does not contain any internal status indicating an error when an error is expected, the test displays:

```
failed to set interrupt status
```

- If interrupt status is present when no error was expected, the test displays:

```
set interrupt status
```

- If the VME BERR status is set but not expected, the test displays:

```
set VME BERR status
```

- If the VME parity interrupt status on the memory board under test is not set, the test displays:

```
failed to set VME parity interrupt status
```

- If the VME parity interrupt status is set when no error was expected, the test displays:

```
set VME parity interrupt status
```

- If the control bus parity interrupt status is set unexpectedly, the test displays:

```
set local control bus parity interrupt status
```

- If the control bus parity interrupt status is not set when expected, the test displays:

```
failed to set local control bus parity interrupt status
```

- If the address bus parity interrupt status is set unexpectedly, the following error message is displayed:

```
set local address bus parity interrupt status
```

- If the address bus parity interrupt status is not set when expected, the test displays:

```
failed to set local address bus parity interrupt status
```

- If the byte field in the status register on the expansion memory board is not set when expected, the test displays:

```
failed to set byte [abcd] parity error status
```

Variable	Description
abcd	Specifies the byte under test when the error occurred.

- If the byte field in the status register is set incorrectly, the test displays:

```
set byte [abcd] parity error status
```

Variable	Description
abcd	Specifies the byte under test when the error occurred.

CSR Data Parity Error Messages

During the CSR Data Parity subtest, faults are inserted for read accesses to the memory CSRs. Any reads will result in errors detected by the CPU, and not the memory board. These reads should result in parity errors detected by both the CMMU and the Local Bus.

The following error messages are specific to this subtest:

- If the CMMU does not detect the parity error and fault the CPU, the test displays:

```
CMMU failed to fault on parity error
```

- If the TS80 input port does not contain the correct parity error status, which should indicate a data parity error on the Local Bus and not an expansion bus parity error, the test displays:

```
Bad int status (ts80 pin is (aaaaaaaa), exp (eeeeeeee)),
forcing bad CSR parity
```

Variable	Description
aaaaaaaa	Specifies the actual hexadecimal value of the TS80 input port.
eeeeeeee	Specifies the expected hexadecimal value of the TS80 input port.

Expansion Memory Interleave Test

This test checks all possible interleave combinations on the expansion memory boards. The boards are set into each interleave combination and tested individually to determine that the correct interleaving occurs when writing to memory.

The data patterns are designed to identify which board should receive the pattern. For example, board 1 uses data pattern BD01BD01, board 2 uses data pattern BD02BD02, and so forth.

Once started, this test displays a start message in the standard format. The test then displays a message identifying the interleave combination to be tested. This message is repeated each time the interleave combination changes.

If the test executes correctly, PASS is displayed on the operator's console below the start messages. If an error is detected, the test displays FAIL on the operator's console, followed by an error message displayed in the following format:

```
Board Interleave Test
board: (n), ipos: (pos), isize: (size),
exp: (eeeeeeee), rcvd: (aaaaaaaa), at addr (bbbbbbbb)
interleaved addr (hhhhhhhh)
```

Variable	Description
n	Specifies the number of the failing memory board.
pos	Specifies the interleave position.
size	Specifies the interleave size under test.
eeeeeeee	Specifies the hexadecimal value expected from memory
aaaaaaaa	Specifies the hexadecimal value received from memory.
bbbbbbbb	Specifies the address accessed.
hhhhhhhh	Specifies the interleave address.

This test is bypassed under the following conditions:

- No expansion memory boards are configured; the test displays the following message:

```
Bypassed for this configuration
No expansion memory board configured.
```

- Only one expansion memory board is configured; the test displays the following message:

```
Bypassed for this configuration
Only 1 expansion memory board configured.
```

- The system contains expansion memory boards of different sizes; the test displays the following message:

```
Bypassed for this configuration
Require two expansion memory boards of equal size.
```

Multi Data Size Test

This test verifies the Local Bus/410 System Bus interface logic. The test transfers data to and from the adaptor board memory using various data sizes. The memory test area is filled with data of one size and read and verified using a different data size. The test uses the following data size combinations:

Fill Data Size	Check Data Size
long	byte
byte	long
long	word
word	long
long	double
double	long
byte	word
word	byte
byte	double
double	byte
word	double
double	word

The following data sizes are used for testing:

- 1 word = 2 bytes (16 bits)
- 1 long = 4 bytes (32 bits)
- 1 double = 8 bytes (64 bits)

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
Multi Data Size aaaaaaaa-bbbbbbbb with n cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test area.
bb...bb	Specifies the ending address of the memory test area.
n	Specifies the number of CPUs configured.

Before filling memory, the test displays a message identifying the data size to be used. For example, `bfill` indicates memory will be filled with bytes. Before reading and verifying the data, the test also displays a message identifying the data size to be used. For example, `wchk` indicates the data will be read and verified as words. These messages are paired for each of the data size combinations. For example, if words are written and bytes are read, the test displays (`wfill-bchk`). The first half of the message is displayed before the data is written and the second half is displayed before the data is read. If the test executes successfully, `...PASS` is appended to the test messages. If an error is detected, the test appends `...FAIL` to the test messages and displays the following error message:

```
Data size test cpu n exp: exp, rcvd: act, at addr: addr
```

Variable	Description
n	Specifies the CPU under test.
exp	Specifies the expected data.
act	Specifies the actual data read from memory.
addr	Specifies the failing memory address.

Long Memory Tests (lmem)

The long memory tests provide a more thorough check of the DRAM on the processor board and external memory boards. The total amount of memory to be tested is divided between the available CPUs. All CPUs test their allotted memory ranges at the same time.

This test group includes the following individual tests:

- DRAM Memory March Test—page 2-45
- DRAM Random Test—page 2-47
- DRAM Byte Strobe Test—page 2-48
- DRAM Data Bit Test—page 2-50
- Multi CPU Fairness Test—page 2-51
- Multi CPU VMEbus Fairness Test—page 2-52

DRAM Memory March Test

This test writes data patterns throughout a range of memory. The data is then read and verified from the ending address to the starting address. As each location is verified, the data pattern is complemented and rewritten. The test then reads and verifies the complemented data in each location from the starting address to the ending address.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
DRAM MEM MARCH aaaaaaaa-bbbbbbbb with x cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test range.
bb...bb	Specifies the ending address of the memory test range.
x	Specifies the number of CPUs configured.

The program then displays the following message for each configured CPU to identify the memory test range associated with the CPU:

```
cpu n testing aaaaaaaa bytes starting at bbbbbbbb
```

Variable	Description
n	Specifies the CPU number.
aa...aa	Specifies the number of bytes in the memory test range assigned to this CPU.
bb...bb	Specifies the starting address of the memory test range assigned to this CPU.

Prior to writing data into the memory range, the test displays `fill` below the start messages. After the data is written into memory, the test appends `chk down` to the message and begins verifying the written data pattern from ending address to starting address. As each location is verified, the data is complemented and rewritten. The test then appends `chk up` to the message and verifies the complemented data pattern from the starting address to the ending address.

If the test executes successfully, the program appends `...PASS` to the message. If an error is detected, the test appends `...FAIL` to the message and displays the following error message:

```
MEM MARCH test cpu x exp: ee, rcvd: aa, at addr yyyyyyyy
```

Variable	Description
x	Specifies the CPU selected for the testing operations.
ee	Specifies the expected data pattern.
aa	Specifies the actual data pattern received.
yyyyyyyy	Specifies the failing memory address.

DRAM Random Test

This test writes pseudo random data patterns to memory and verifies they can be read back correctly. The address range under test is filled with a random sequence of data in a specified size (byte, word, long word). The random number generator is reseeded with the same seed to generate the same sequence of random data. The test reads the data already written into memory and verifies that the data read equals the data pattern generated by the second call to the random number generator.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
DRAM random  size from aaaaaaaa-bbbbbbbb with x cpus
```

Variable	Description
size	Specifies the size of data under test: `BYTE`, `WORD` or `LONG`. The start message is repeated for each data type.
aa...aa	Specifies the starting address of the memory test range.
bb...bb	Specifies the ending address of the memory test range.
x	Specifies the number of CPUs configured.

The program then displays the following message for each configured CPU to identify the memory test range associated with the CPU:

```
cpu n testing aaaaaaaa bytes starting at bbbbbbbb
```

Variable	Description
n	Specifies the CPU number.
aa...aa	Specifies the number of bytes in the memory test range assigned to this CPU.
bb...bb	Specifies the starting address of the memory test range assigned to this CPU.

Prior to filling the memory test range with data, the test displays `fill` below the start messages. After the data is written, the test appends `chk` to the message and begins reading and verifying the data.

If the test runs successfully, `...PASS` is appended to the message and the next test is executed. If an error is detected, the test appends `...FAIL` to the message and displays the following error message:

```
DRAM size random test cpu x exp: ee, rcvd: aa, at addr bbbbbbbb
```

Variable	Description
size	Specifies the size of the data under test: `BYTE`, `WORD` or `LONG`.
x	Specifies the number of the CPU used for testing.
ee	Specifies the expected data value.
aa	Specifies the actual data read from memory.
bb...bb	Specifies the address of the failing memory location.

DRAM Byte Strobe Test

This test verifies that data written as long words can be read as bytes and data written as bytes can be read as long words without corrupting the data. The test first writes long words of a random data pattern to each location in the memory range under test. The data is read back as bytes and verified. The test then fills the memory test range with selected bytes of data. The new data is read back as long words and verified.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
DRAM byte strobe: aaaaaaaa-bbbbbbbb with x cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test range.
bb...bb	Specifies the ending address of the memory test range.
x	Specifies the number of CPUs configured.

The program then displays the following message for each configured CPU to identify the memory test range associated with the CPU:

```
cpu n testing aaaaaaaa bytes starting at bbbbbbbb
```

Variable	Description
n	Specifies the CPU number.
aa...aa	Specifies the number of bytes in the memory test range assigned to this CPU.
bb...bb	Specifies the starting address of the memory test range assigned to this CPU.

The test displays `rand fill` below the start message and fills memory with long words of the random data pattern. The test then appends `chk` to the message and reads the data back as bytes and verifies that the data read equals the data written. Next, the test appends `fil1-4` to the message and loads the memory test range with selected bytes of data. The new data pattern is read back as bytes (`bchk`) and long words (`wchk`).

If the test executes successfully, `...PASS` is appended to the message and the next test is started. If an error is detected, the test appends `...FAIL` to the message and displays the following error message:

```
BYTE STROBE test cpu x exp: ee, rcvd: aa, at addr bbbbbbbb
```

Variable	Description
x	Specifies the number of the CPU used for testing.
ee	Specifies the expected data value.
aa	Specifies the actual data read from memory.
bb...bb	Specifies the address of the failing memory location.

DRAM Data Bit Test

This test writes a data pattern with alternate bits set to each location in the memory range under test. The data is read from memory and verified against the data written. The test then complements the data pattern and repeats the write, read, and verify sequence for the complemented pattern. This test is performed for data sized as bytes, words, and long words.

Once started, this test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
DRAM data bit size aaaaaaaa-bbbbbbbb with x cpus
```

Variable	Description
size	Specifies the size of the data under test: BYTE, WORD or LONG. The start message is repeated for each type of data tested.
aa...aa	Specifies the starting address of the memory test range.
bb...bb	Specifies the ending address of the memory test range.
x	Specifies the number of CPUs configured.

The program then displays the following message for each configured CPU to identify the memory test range associated with the CPU:

```
cpu n testing aaaaaaaa bytes starting at bbbbbbbb
```

Variable	Description
n	Specifies the CPU number.
aa...aa	Specifies the number of bytes in the memory test range assigned to this CPU.
bb...bb	Specifies the starting address of the memory test range assigned to this CPU.

Prior to filling memory with the first data pattern, the program displays `pat1` below the start message. The test then appends `chk` to the message and reads and verifies the data in each location. After the data is verified, the test appends `pat2` to the message, complements the data pattern and rewrites it. Finally, the test appends `chk` to the message and verifies the complemented data. If the test runs successfully, `...PASS` is appended to the message and test execution is completed.

If an error is detected, the test appends ...FAIL to the message and displays the following error message:

```
DRAM size data bit test cpu x exp: ee, rcvd: aa, at addr bbbbbbbb
```

Variable	Description
x	Specifies the number of the CPU used for testing.
size	Specifies the size of data under test when the error occurred: BYTE, WORD or LONG.
ee	Specifies the expected data value.
aa	Specifies the actual data value received from memory.
bb...bb	Specifies the address of the failing memory location.

Multi CPU Fairness Test

This test verifies the ability of multiple CPUs to access memory fairly. The test runs in both on board DRAM and the expansion memory boards.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
CPU n, aaaaaa, [CPU n, aaaaaa, ...]
```

Variable	Description
n	Specifies the CPU number.
aaaaaa	Specifies the base address of expansion memory.

The message is expanded to include each configured CPU.

If the test executes successfully, PASS is displayed on the operator's console below the message. If an error is detected, the test displays FAIL on the operator's console. The test then displays one of the following error messages:

- If a processor is unable to access the test location, the test displays:

```
CPU n  Mbus access unfair
```

Variable	Description
n	Specifies the CPU under test.

- If an address error is detected, the test displays:

```
CPU n: Write a, Lower Bound l, Upper Bound u
```

Variable	Description
n	Specifies the CPU under test.
a	Specifies the address of the test location.
l	Specifies the lower test boundary.
u	Specifies the upper test boundary.

Multi CPU VMEbus Fairness Test

This test verifies the ability of multiple CPUs to access memory fairly across the VMEbus. The test runs in both on board DRAM and the expansion memory boards.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
CPU n, aaaaaa, [CPU n, aaaaaa,...]
```

Variable	Description
n	Specifies the CPU number.
aaaaaa	Specifies the base address of expansion memory.

The message is expanded to include each configured CPU.

If the test executes successfully, PASS is displayed on the operator's console below the message. If an error is detected, the test displays FAIL on the operator's console. The test then displays one of the following error messages:

- If a processor is unable to access the test location, the test displays:

```
CPU n  Mbus access unfair
```

Variable	Description
n	Specifies the CPU under test.

- If an address error is detected, the test displays:

```
CPU n: Write a, Lower Bound l, Upper Bound u
```

Variable	Description
n	Specifies the CPU under test.
a	Specifies the address of the test location.
l	Specifies the lower test boundary.
u	Specifies the upper test boundary.

Loopback Tests (lpbk)

Loopback tests are tests requiring some type of external connection, including special connectors and, in the case of the Ethernet test, a connection to the network/transceiver.

This group includes the following tests:

- Ethernet Network Loopback Test—page 2-54
- SCC External Loopback Test—page 2-56

Ethernet Network Loopback Test

This test verifies the ability of the LAN controller to send and receive packets across the Ethernet network.

Once started, the test displays a start message on the operator's console in the standard format.

If the test executes successfully, PASS is displayed on the operator's console below the start message. If an error is detected, the test displays FAIL on the operator's console. The test then displays one of the following error messages:

- If the test is unable to communicate across the network interface, it displays:

```
<<<<<< Check Transceiver/Network Connections >>>>>>
```

- If the test is unable to initialize the network interface, it displays:

```
Couldn't initialize network interface (status stat)
```

Variable	Description
stat	Specifies the controller status information.

- If an error occurs when receiving a frame, the test displays:

```
Error receiving frame (status stat)
```

Variable	Description
stat	Specifies the controller status information.

- If the test is unable to configure network loopback mode, it displays:

```
Couldn't configure 82596 for Transceiver mode
```

- If the test times out waiting to receive a loopback frame, it displays:

```
Timed out waiting to receive loopback frame
```

- If the test detects an address error in the received packet, it displays:

```
Destination Address error in received packet
```

- If a data error is detected, the test displays:

```
Received data != transmitted data
Data in byte x is y, sb z
```

Variable	Description
x	Specifies the failing data byte.
y	Specifies the actual data received.
z	Specifies the expected data value.

- If too many errors occur, the test displays:

```
External Loopback too many errors (nnn)
```

Variable	Description
nnn	Specifies the number of errors detected.

■ If no frames were received, the test displays:

```
Error: No frames received
```

SCC External Loopback Test

This test verifies the ability of the Serial Communications Controller (SCC) to transmit and receive data through an external loopback connection. The data is transmitted from the transmit pin on the serial connector port to the receive pin. The test also verifies the ability to set and clear modem signals and verifies that the correct SCC interrupt is generated for each signal.

Once started, this test displays a start message on the operator's console in the standard format. The test then displays the following message as each port is tested:

```
port: n
```

Variable	Description
n	Specifies the number of the port under test.

If the test executes correctly, `...PASS` is appended to the message. If an error is detected, the test appends `...FAIL` to the message and displays one of the following error messages:

■ If the test is unable to communicate across the loopback cable, it displays:

```
<<<<<< Check Loopback Cable Connections >>>>>>
port: n...FAIL
```

Variable	Description
n	Specifies the number of the port under test.

■ If a timeout occurs while waiting to receive data, the test displays:

```
Port n: SCC a Channel b: timeout
```

Variable	Description
n	Specifies the number of the port under test.
a	Specifies the SCC under test.
b	Specifies the channel under test.

- If an incorrect character is received, the test displays:

```
Port n: SCC a Channel b: read back char number x - y
```

Variable	Description
n	Specifies the number of the port under test.
a	Specifies the SCC under test.
b	Specifies the channel under test.
x	Specifies the number of the failing character.
y	Specifies the failing character.

- If the test is unable to set or reset a modem signal bit, it displays:

```
Port n: SCC a Channel b: failed to {set|reset} s status bit
```

Variable	Description
n	Specifies the number of the port under test.
a	Specifies the SCC under test.
b	Specifies the channel under test.
s	Specifies the failing modem signal bit.

- If the interrupt associated with a modem signal bit fails to occur when the bit is set, the test displays:

```
Port n: SCC a Channel b: External Status Interrupt - i 0 to 1 not
detected.
```

Variable	Description
n	Specifies the number of the port under test.
a	Specifies the SCC under test.
b	Specifies the channel under test.
i	Specifies the failing interrupt.

- If the interrupt associated with a modem signal bit fails to clear when the bit is reset, the test displays:

```
Port n: SCC a Channel b: External Status Interrupt - i 1 to 0 not
detected.
```

Variable	Description
n	Specifies the number of the port under test.
a	Specifies the SCC under test.
b	Specifies the channel under test.
i	Specifies the failing interrupt.

Cache Memory Tests (cache)

The cache memory tests verify the interface between the processor board memory and the 410 CMMUs across the 410 System Bus in both burst and nonburst modes. The tests also verify the interface between the adaptor board memory and the 410 CMMUs across the Local Bus and the 410 System Bus in both burst and nonburst modes. These tests also verify the coherency of cache memory on the processor boards. The following cache memory tests are available:

- 88410 Ptag Verification Test—page 2-60
- 88410 Mtag/Cache Array Verification Test—page 2-61
- Flush Control Register Test—page 2-63
- Processor Memory Non-Burst Retry Test—page 2-66
- Processor Memory Burst Retry Test—page 2-68
- Adaptor Memory Non-Burst Retry Test—page 2-70
- Adaptor Memory Burst Retry Test—page 2-72
- Critical Word Transfer Test—page 2-74

During the set up of all the cache memory tests, secondary cache is invalidated. If one or more busy bits in the processor status register remain set, the following message is displayed:

```
Timeout while waiting for busy bits to go away - status at fff80808
```

Variable	Description
status	Specifies the contents of the processor board status register when the timeout occurred.
`fff80808`	Specifies the address of the status register.

After the timeout, control returns to the test under execution which reports the error with the following message:

```
Can't invalidate all secondary cache, a busy bit still on...
```

These tests run with instruction caches disabled. After the test exits, instruction caches are enabled.

88410 Ptag Verification Test

This test verifies the copy of the ptags (processor tags) that are on the MC88410 (Secondary Cache Controller). The test places the 410 in diagnostic mode and writes a data pattern to the 256 ptags. The contents of the ptags are read back and verified. The test also verifies the Secondary Cache Memory Arrays (MC62110 chips) by writing a data pattern and reading it back for comparison. Each configured CPU tests its own 410. When test execution is completed, the tags are left in an invalid state.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
88410 Ptag Verification from set 0 to set 127, n cpus
```

Variable	Description
n	Specifies the number of CPUs configured.

If the test executes successfully, `...PASS` is displayed on the operator's console below the start message. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays one of the following error messages:

- If the data read from the ptags does not match the data written, the test displays:

```
88410 Ptag Verification test, cpu n got a
DATA ACCESS EXCEPTION when reading ptag at address xxxddaay
```

Variable	Description
n	Specifies the number of the failing CPU.
xxx	Specifies do not care bits.
dd	Specifies the ptag data.
aa	Specifies the ptag address (pointer).
y	Specifies do not care bits.

- If the data read from the cache memory arrays does not match the data written, the test displays:

```
88410 Ptag Verification test, cpu n
exp: e, rcvd: a, at addr: x in the MC62110 memory arrays
```

Variable	Description
n	Specifies the number of the failing CPU.
e	Specifies the expected data.
a	Specifies the actual data.
x	Specifies the failing address.

88410 Mtag / Cache Array Verification Test

This test verifies the mtags (main tags) that are on the MC88410 (Secondary Cache Controller). The test places the 410 in diagnostic mode and writes a data pattern to the 16384 mtag entries. The contents of the mtags are read back and verified. The test also verifies the Secondary Cache Memory Arrays (MC62110 chips) by writing a data pattern and reading it back for comparison. Each configured CPU tests its own 410. When test execution is completed, the tags are left in an invalid state.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
88410 Mtag/Cache Array Verification from mtag entry 0 to mtag entry
16383 n cpus
```

Variable	Description
n	Specifies the number of CPUs configured.

As a data pattern is written, the test displays a message in the following format:

```
mtag data patt: pat1 - memory data patt: pat2
```

Variable	Description
pat1	Specifies one of the following mtag data patterns: 0xfffc 0x0000 0xaaa8 0x5554
pat2	Specifies one of the following memory data patterns: 0x00000000 0xffffffff 0x69696969 0x5a5a5a5a

If the test executes successfully, ...PASS is displayed on the operator's console. If an error is detected, the test displays ...FAIL on the operator's console. The test then displays one of the following error messages:

- If the data read from the mtags does not match the data written, the test displays:

```
88410 Mtag/Cache Array Verification test, cpu n got a
DATA ACCESS EXCEPTION when reading mtag at address addr
```

Variable	Description
n	Specifies the number of the failing CPU.
addr	Specifies the failing address as follows: Bits 0-3 are do not care bits Bits 4-17 specify the mtag address (pointer) Bits 18-31 specify the mtag data

- If the data read from the cache memory arrays does not match the data written, the test displays:

```
88410 Mtag/Cache Array Verification test, cpu n
exp: e, rcvd: a, at addr: x in the MC62110 memory arrays
```

Variable	Description
n	Specifies the number of the failing CPU.
e	Specifies the expected data.
a	Specifies the actual data.
x	Specifies the failing address.

Flush Control Register Test

This test verifies the cache flushing logic on the processor board.

The following operations are performed during test execution:

- Data cache is disabled.
- CPU 0 writes a background data pattern to processor board memory.
- With data cache enabled, CPU 0 reads one byte from each of the six lines causing the lines to be cached and to change from invalid to shared (or exclusive unmodified). A read burst should occur on the bus.
- CPU 0 modifies one byte in each line causing the lines to change from shared to exclusive modified and making the main memory (processor board memory) data stale.
- CPU 0 then issues a `FLUSH ALL` command using the flush control register. It then issues an `INVALIDATE ALL` command also in the flush control register. By this time main memory should have been updated with the new data.
- CPU 1 then reads a byte from the first line with its cache disabled.
- The test then verifies that the data read by CPU 1 is the same data CPU 0 modified the line with before updating main memory.
- CPU 1 repeats the read operation on a different line.
- CPUs 2 and 3 perform the same read operation on different lines.

These operations are repeated until each CPU has flushed its cache by performing the steps associated with CPU 0 in the above procedure. While one CPU is flushing cache, the other CPUs are reading the data from main memory and comparing it.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
Flush Control Register from aaaaaaaa-bbbbbbbb with n cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test area.
bb...bb	Specifies the ending address of the memory test area.
n	Specifies the number of CPUs configured.

If only one CPU is present, the test displays the following message and testing is bypassed:

```
Bypassing test, only one cpu available
```

As each CPU flushes its cache, the test displays a message in the following format:

```
cpu n testing its flush logic...
```

Variable	Description
n	Specifies the CPU under test.

If the test executes successfully, `...PASS` is displayed on the operator's console. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays one of the following error messages:

- If a data mismatch occurs while the other CPUs read and compare the data in main memory, the test displays:

```
Flush Control Register Test
cpu n exp: exp, rcvd: act, at addr: addr
```

Variable	Description
n	Specifies the number of the failing CPU.
exp	Specifies the expected data value.
act	Specifies the actual data read.
addr	Specifies the failing memory address.

- If the busy bits are on when the test attempts to issue the `FLUSH ALL` command, it displays:

```
Can not issue a FLUSH ALL cmd to node n - Flush Busy bit is on
0xstatus at 0xaddr
```

Variable	Description
n	Specifies the number of the failing node.
status	Specifies the contents of the processor board status register.
addr	Specifies the address of the processor board status register.

- If a timeout occurs after the test issues the FLUSH ALL command, it displays:

```
Time out while waiting for busy bits to go away after
a FLUSH ALL cmd to node n flush control register - 0xstatus at 0xaddr
```

Variable	Description
n	Specifies the number of the failing node.
status	Specifies the contents of the processor board status register.
addr	Specifies the address of the processor board status register.

- If the busy bits are on when the test attempts to issue the INVALIDATE ALL command, it displays:

```
Can not issue an INVALIDATE ALL cmd to node n - Flush Busy bit is on
0xstatus at 0xaddr
```

Variable	Description
n	Specifies the number of the failing node.
status	Specifies the contents of the processor board status register.
addr	Specifies the address of the processor board status register.

- If a timeout occurs after the test issues the INVALIDATE ALL command, it displays:

```
Time out while waiting for busy bits to go away after
an INVALIDATE ALL cmd to node n flush control register - 0xstatus at
0xaddr
```

Variable	Description
n	Specifies the number of the failing node.
status	Specifies the contents of the processor board status register.
addr	Specifies the address of the processor board status register.

Processor Memory Non-Burst Retry Test

This test verifies the interface between the processor board memory and the 410 CMMUs. Data transfers are made across the 410 System Bus in nonburst mode. The test also tests the snooping and retry mechanisms of the 410 CMMUs as well as cache coherency.

The following operations are performed during test execution:

- Data cache is disabled.
- CPU 0 writes a background data pattern to processor board memory.
- With data cache enabled, CPU 0 reads one byte from each of the six lines, causing the lines to be cached and to change from invalid to shared (or exclusive unmodified). A read burst should occur on the bus.
- CPU 0 modifies one byte in each line, causing the lines to change from shared to exclusive modified and making the main memory (processor board memory) data stale.
- CPU 1 then reads a byte from the first line. This causes CPU 0 to copy back that line to the processor board memory and forces CPU 1 to be retried on the read (nonburst).
- The test verifies that the data CPU 1 reads on the retry is the same data CPU 0 modified the line with before updating main memory.
- CPU 1 repeats the read and retry operations on a different line.
- CPUs 2 and 3 perform the same read and retry operations on different lines to verify their retry mechanisms.

These operations are repeated until each CPU has performed the setup operations associated with CPU 0 in the above procedure. While one CPU is performing the setup, the other CPUs are trying to read the data from main memory and compare it.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
Processor Memory Non-Burst Retry from aaaaaaaa-bbbbbbbb with n cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test area.
bb...bb	Specifies the ending address of the memory test area.
n	Specifies the number of CPUs configured.

If only one CPU is present, the test displays the following message and testing is bypassed:

```
Bypassing test, only one cpu available
```

As a CPU starts the setup operations, the test displays a message in the following format:

```
cpu n will retry the others...
```

Variable	Description
n	Specifies the number of the CPU performing the setup operation.

If the test executes successfully, `...PASS` is displayed on the operator's console. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays the following error message:

```
Processor Memory Non-Burst Retry Test
cpu n exp: exp, rcvd: act, at addr: addr
```

Variable	Description
n	Specifies the number of the failing CPU.
exp	Specifies the expected data value.
act	Specifies the actual data read.
addr	Specifies the failing memory address.

Processor Memory Burst Retry Test

This test verifies the interface between the processor board memory and the 410 CMMUs. Data transfers are made across the 410 System Bus in burst mode. The test also tests the snooping and retry mechanisms of the 410 CMMUs as well as cache coherency.

The following operations are performed during test execution:

- Data cache is disabled.
- CPU 0 writes a background data pattern to processor board memory.
- With data cache enabled, CPU 0 reads one byte from each of the six lines, causing the lines to be cached and to change from invalid to shared (or exclusive unmodified). A read burst should occur on the bus.
- CPU 0 modifies one byte in each line, causing the lines to change from shared to exclusive modified and making the main memory (processor board memory) data stale.
- CPU 1 then reads a byte from the first line with its cache enabled. This causes CPU 0 to copy back that line to the processor board memory and forces CPU 1 to be retried on the read (burst).
- The test verifies that the data CPU 1 reads on the retry is the same data CPU 0 modified the line with before updating main memory.
- CPU 1 repeats the read and retry operations on a different line.
- CPUs 2 and 3 perform the same read and retry operations on different lines to verify their retry mechanisms.

These operations are repeated until each CPU has performed the setup operations associated with CPU 0 in the above procedure. While one CPU is performing the setup, the other CPUs are trying to read the data from main memory and compare it.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
Processor Memory Burst Retry from aaaaaaaa-bbbbbbbb with n cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test area.
bb...bb	Specifies the ending address of the memory test area.
n	Specifies the number of CPUs configured.

If only one CPU is present, the test displays the following message and testing is bypassed:

```
Bypassing test, only one cpu available
```

As a CPU starts the setup operations, the test displays a message in the following format:

```
cpu n will retry the others...
```

Variable	Description
n	Specifies the number of the CPU performing the setup operation.

If the test executes successfully, `...PASS` is displayed on the operator's console. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays the following error message:

```
Processor Memory Burst Retry test
cpu n exp: exp, rcvd: act, at addr: addr
```

Variable	Description
n	Specifies the number of the failing CPU.
exp	Specifies the expected data value.
act	Specifies the actual data read.
addr	Specifies the failing memory address.

Adaptor Memory Non-Burst Retry Test

This test verifies the interface between the adaptor board memory and the 410 CMMUs. Data transfers are made across the Local Bus and 410 System Bus in nonburst mode. The test also tests the snooping and retry mechanisms of the 410 CMMUs as well as cache coherency.

The following operations are performed during test execution:

- Data cache is disabled.
- CPU 0 writes a background data pattern to adaptor board memory.
- With data cache enabled, CPU 0 reads one byte from each of the six lines, causing the lines to be cached and to change from invalid to shared (or exclusive unmodified). A read burst should occur on the bus.
- CPU 0 modifies one byte in each line, causing the lines to change from shared to exclusive modified and making the main memory (adaptor board memory) data stale.
- CPU 1 then reads a byte from the first line. This causes CPU 0 to copy back that line to the adaptor board memory and forces CPU 1 to be retried on the read (nonburst).
- The test verifies that the data CPU 1 reads on the retry is the same data CPU 0 modified the line with before updating main memory.
- CPU 1 repeats the read and retry operations on a different line.
- CPUs 2 and 3 perform the same read and retry operations on different lines to verify their retry mechanisms.

These operations are repeated until each CPU has performed the setup operations associated with CPU 0 in the above procedure. While one CPU is performing the setup, the other CPUs are trying to read the data from main memory and compare it.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
Adaptor Memory Non-Burst Retry from aaaaaaaa-bbbbbbbb with n cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test area.
bb...bb	Specifies the ending address of the memory test area.
n	Specifies the number of CPUs configured.

If only one CPU is present, the test displays the following message and testing is bypassed:

```
Bypassing test, only one cpu available
```

As a CPU starts the setup operations, the test displays a message in the following format:

```
cpu n will retry the others...
```

Variable	Description
n	Specifies the number of the CPU performing the setup operation.

If the test executes successfully, `...PASS` is displayed on the operator's console. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays the following error message:

```
Adaptor Memory Non-Burst Retry test
cpu n exp: exp, rcvd: act, at addr: addr
```

Variable	Description
n	Specifies the number of the failing CPU.
exp	Specifies the expected data value.
act	Specifies the actual data read.
addr	Specifies the failing memory address.

Adaptor Memory Burst Retry Test

This test verifies the interface between the adaptor board memory and the 410 CMMUs. Data transfers are made across the Local Bus and 410 System Bus in burst mode. The test also tests the snooping and retry mechanisms of the 410 CMMUs as well as cache coherency.

The following operations are performed during test execution:

- Data cache is disabled.
- CPU 0 writes a background data pattern to adaptor board memory.
- With data cache enabled, CPU 0 reads one byte from each of the six lines, causing the lines to be cached and to change from invalid to shared (or exclusive unmodified). A read burst should occur on the bus.
- CPU 0 modifies one byte in each line, causing the lines to change from shared to exclusive modified and making the main memory (adaptor board memory) data stale.
- CPU 1 then reads a byte from the first line with its cache enabled. This causes CPU 0 to copy back that line to the adaptor board memory and forces CPU 1 to be retried on the read (burst).
- The test verifies that the data CPU 1 reads on the retry is the same data CPU 0 modified the line with before updating main memory.
- CPU 1 repeats the read and retry operations on a different line.
- CPUs 2 and 3 perform the same read and retry operations on different lines to verify their retry mechanisms.

These operations are repeated until each CPU has performed the setup operations associated with CPU 0 in the above procedure. While one CPU is performing the setup, the other CPUs are trying to read the data from main memory and compare it.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
Adaptor Memory Burst Retry from aaaaaaaa-bbbbbbbb with n cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test area.
bb...bb	Specifies the ending address of the memory test area.
n	Specifies the number of CPUs configured.

If only one CPU is present, the test displays the following message and testing is bypassed:

```
Bypassing test, only one cpu available
```

As a CPU starts the setup operations, the test displays a message in the following format:

```
cpu n will retry the others...
```

Variable	Description
n	Specifies the number of the CPU performing the setup operation.

If the test executes successfully, `...PASS` is displayed on the operator's console. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays the following error message:

```
Adaptor Memory Burst Retry test
cpu n exp: exp, rcvd: act, at addr: addr
```

Variable	Description
n	Specifies the number of the failing CPU.
exp	Specifies the expected data value.
act	Specifies the actual data read.
addr	Specifies the failing memory address.

Critical Word Transfer Test

This test verifies the proper translation of a nonaligned or aligned 64-bit transfer on the 410 System Bus into two 32-bit transfers on the Local Bus.

The following operations are performed during test execution:

- Data cache is disabled.
- CPU 0 writes a background data pattern to the adaptor board memory.
- With data cache enabled, CPU 0 reads one word from each of the 24 lines, causing the lines to be cached and to change from invalid to shared (or exclusive unmodified). A read burst should occur on the bus.
- CPU 0 modifies one word in each line, causing the lines to change from shared to exclusive modified and making the main memory data stale.
- CPU 1 then reads word 0 from the first line with its cache enabled. This causes CPU 0 to copy back that line and forces CPU 1 to be retried on the read (burst).
- The test verifies that the data CPU 1 reads on the retry is the same data CPU 0 modified the line with before updating main memory.
- CPU 1 repeats the read and retry operations with word 1 from the next line. CPU 1 repeats the operations with a different word until it has walked a word through every position on the line. (8 words/line)
- CPUs 2 and 3 perform the same read and retry operations on different lines to test the critical word mechanism on the Local Bus.

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
Critical Word Transfer from aaaaaaaa-bbbbbbbb with n cpus
```

Variable	Description
aa...aa	Specifies the starting address of the memory test area.
bb...bb	Specifies the ending address of the memory test area.
n	Specifies the number of CPUs configured.

If only one CPU is present, the test displays the following message and testing is bypassed:

```
Bypassing test, only one cpu available
```

If the test executes successfully, `...PASS` is displayed on the operator's console below the start message. If an error is detected, the test displays `...FAIL` on the operator's console. The test then displays the following error message:

```
Critical Word Transfer test
cpu n exp: exp, rcvd: act, at addr: addr
```

Variable	Description
n	Specifies the number of the failing CPU.
exp	Specifies the expected data value.
act	Specifies the actual data read.
addr	Specifies the failing memory address.

Error Correction Code Tests (ecc)

The error correction code tests verify the memory error detection and correction logic on the processor board. The following error correction code tests are available:

- Error Correction Code Test—page 2-78
- Error Correction Code Exception Test—page 2-82

Each 32-bit value in the processor memory has seven associated error correction code (ecc) check bits. These bits are used to detect and correct single bit errors in each 32-bit word. The ecc check bits can also detect double bit errors although the 32-bit data value can not be corrected.

The processor board memory is tested by writing 32-bit data values to memory with the IDT49C465 Error Detection and Correction Unit in Normal mode. The Error Detection and Correction Unit is then placed in ECC INJECT mode and the seven least significant bits of the next memory write will be held as the ecc code for future reads while in Inject mode.

The error correction code tests use a table of 128 data values. The index to this table (0 to 127) is the ecc value for each data word in the table. The following data values are used for testing:

TABLE 2-1 Data and ECC Values for the Error Correction Code Tests

Data Values				ECC Bits
0x010202fd	0x03040505	0x0f101111	0x010202f2	0x00 to 0x03
0x03040506	0x010202fe	0x010202f1	0x0f101112	0x04 to 0x07
0x2223241e	0xff0000fc	0xff0000f3	0x8283847d	0x08 to 0x0b
0xff0000ff	0x2223241d	0x8283847e	0xff0000f0	0x0c to 0x0f
0x2324251f	0xff0000fa	0xff0000f5	0x8283847b	0x10 to 0x13
0xff0000f9	0x2223241b	0x8384857f	0xff0000f6	0x14 to 0x17
0x010202fb	0x02030404	0x3f404142	0x010202f4	0x18 to 0x1b
0x03040500	0x010202f8	0x010202f7	0x3f404141	0x1c to 0x1f
0x1f202120	0x7e7f8081	0x04050607	0x8182837d	0x20 to 0x23
0x08090a07	0xff000100	0x07080908	0x00010200	0x24 to 0x27
0x06070806	0x01020302	0x7f80817c	0x06070809	0x28 to 0x2b
0x01020301	0x06070805	0x1e1f201e	0x7f80817f	0x2c to 0x2f
0x06070800	0x01020304	0x0d0e0f10	0x1e1f201b	0x30 to 0x33

TABLE 2-1 Data and ECC Values for the Error Correction Code Tests *(Continued)*

Data Values				ECC Bits
0x05060703	0x06070803	0x7e7f807b	0x1112130f	0x34 to 0x37
0x07080902	0x1011120d	0x04050601	0x8182837b	0x38 to 0x3b
0x1011120e	0x07080901	0x81828378	0x04050602	0x3c to 0x3f
0xff0000f1	0x8283847f	0x2223241c	0xff0000fe	0x40 to 0x43
0x8283847c	0xff0000f2	0xff0000fd	0x2223241f	0x44 to 0x47
0x3e3f4041	0x010202f0	0x010202ff	0x02030400	0x48 to 0x4b
0x010202f3	0x0e0f100f	0x02030403	0x010202fc	0x4c to 0x4f
0x3e3f403f	0x010202f6	0x010202f9	0x03040501	0x50 to 0x53
0x010202f5	0x3e3f403c	0x02030405	0x010202fa	0x54 to 0x57
0xff0000f7	0x8384857e	0x2324251d	0xff0000f8	0x58 to 0x5b
0x8384857d	0xff0000f4	0xff0000fb	0x2324251e	0x5c to 0x5f
0x7f80817e	0x1e1f201f	0x06070804	0x01020300	0x60 to 0x63
0x06070808	0x05060708	0x01020303	0x06070807	0x64 to 0x67
0xfdff0000	0x07080909	0x1d1e1f20	0x08090a06	0x68 to 0x6b
0x0708090a	0x00010202	0x7e7f8080	0x7f808182	0x6c to 0x6f
0x04050603	0x81828379	0x07080900	0x1011120f	0x70 to 0x73
0x8182837a	0x04050600	0x1011120c	0x07080903	0x74 to 0x77
0x1112130e	0x7e7f807a	0x06070802	0x05060702	0x78 to 0x7b
0x1e1f201a	0x1112130d	0x05060701	0x06070801	0x7c to 0x7f

Error Correction Code Test

This test verifies the correct operation of the processor memory error detection and correction logic by manipulating the ecc check bits and memory data bits. The test verifies that the correct response is received for the following conditions:

Testing Condition	Expected Result
Correct ECC Bits Injected	No ECC Errors
1 Check Bit Changed	Correct Data
1 Data Bit Changed	Corrected Data
2 Check Bits Changed	Uncorrected Data
2 Data Bits Changed	Uncorrected Data
1 Check Bit and 1 Data Bit Changed	Uncorrected Data

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
Error Correction Code Test executing on cpu n
```

Variable	Description
n	Specifies the CPU used for testing.

If the test executes successfully, `CPU` *n* `...PASS` is displayed on the operator's console below the start message. If an error is detected, `CPU` *n* `...FAIL` is displayed on the operator's console. The test then displays the following error message:

```
Error Correction Code Test
data_value data_write data_diff ecc_v ecc_w ecc_d data_read0 data_read1
   0xorig       0xact       0xdiff     0xvv   0xww  0xdd    0xread0       0xread1     [type]
```

Variable	Description
orig	Specifies the original 32-bit test data pattern.
act	Specifies the actual data pattern written to two sequential memory words.
diff	Specifies the difference between the original and actual data patterns.
vv	Specifies the ecc check bits for the original test data pattern.

Variable	Description
ww	Specifies the ecc check bits injected.
dd	Specifies the difference between the original and injected check bits.
read0	Specifies the first of two words read back from memory.
read1	Specifies the second (odd) of two words read back from memory.
type	Specifies the transfer type used for reading the data: `wd` (word), `hw` (halfword), `ch` (character). If the data read back for all three transfer types shows a consistent error, one line of data error information is displayed and this field is blank. If the data read back for a specific transfer type shows an error that is not consistent across the transfer types, the test displays one line of error information for each transfer type.

Correct ECC Bits Injected

The test performs the following operations for this testing condition:

- Writes data values to memory in Normal mode.
- Enters Inject mode.
- Writes the correct ecc value to memory for each data value.
- Reads the data back and verifies that no ecc errors occurred.

This testing condition is checked with the following data values:

Data Value	Description
`0x00000000`	All zeros data value
`0xffffffff`	All ones data value
`0x010202fd`	All zeros ecc value (`0x00`)
`0x06070801`	All ones ecc value (`0x7f`)
`0x00000001` to `0x80000000`	32 data values, each with 1 bit set
`ecc_table`	128 data values from the test table which produce ecc values from `0x00` to `0x7f`

One Check Bit Changed

The test performs the following operations for this testing condition:

- Writes data values to memory in Normal mode.
- Enters Inject mode.
- Writes ecc values with one bit changed to memory.
- Reads the data back and verifies that the data returned equals the data written.

This testing condition is checked with the following data values:

Data Value	Description
`0x010202fd`	All zeros ecc value (`0x00`). Change each ecc bit (`0x01` to `0x40`) for a total of 7 tests.
`ecc_table`	128 data values with ecc values from `0x00` to `0x7f`. Change each ecc bit for each data value for a total of 128*7 tests.

One Data Bit Changed

The test performs the following operations for this testing condition:

- Writes data values with one bit changed to memory in Normal mode.
- Enters Inject mode.
- Writes the ecc value for the original data word to memory.
- Reads the data back and verifies that corrected data was received.

This testing condition is checked with the following data values:

Data Value	Description
`0x00000000`	All zeros data value. Change each data bit from 0 through 31 for a total of 32 tests.
`ecc_table`	128 data values from the test table with ecc values from `0x00` to `0x7f`. Change each data bit from 0 through 31 for a total of 128*32 tests.

Two Check Bits Changed

The test performs the following operations for this testing condition:

- Writes data values to memory in Normal mode.
- Enters Inject mode.
- Writes ecc values with two bits changed to memory.
- Reads the data back and verifies that the data returned equals the data written and no error correction was performed.

This testing condition is checked with the following data values:

Data Value	Description
`0x010202fd`	All zeros ecc value (`0x00`). Change two ecc bits (`0x60` to `0x03`) for a total of 21 tests.

Two Data Bits Changed

The test performs the following operations for this testing condition:

- Writes data values with two bits changed to memory in Normal mode.
- Enters Inject mode.
- Writes the ecc value for the original data word to memory.
- Reads the data back and verifies that the data returned equals the data written and no error correction was performed.

This testing condition is checked with the following data values:

Data Value	Description
`0x00000000`	All zeros data value. Change two data bits (`0x80000001` to `0x00000003`) for a total of 496 tests.

One Check Bit and One Data Bit Changed

The test performs the following operations for this testing condition:

- Writes data values with one bit changed to memory in Normal mode.
- Enters Inject mode.
- Writes the ecc value with one bit changed to memory.
- Reads the data back and verifies that the data returned equals the data written and no error correction was performed.

This testing condition is checked with the following data values:

Data Value	Description
`0x00000000`	All zeros data value. Change one data bit from 0 through 31 in combination with one ecc bit for a total of 32*7 tests.

Error Correction Code Exception Test

This test verifies that a level 3 trap occurs if a multiple ecc error is detected. The test performs the following operations:

- Writes data values with two bits changed to memory in Normal mode.
- Enters Inject mode.
- Writes the ecc value for the original data word to memory.
- Reads the data back and verifies that a level 3 trap occurs.

This testing condition is checked with the following data values:

Data Value	Description
`0x00000000`	All zeros data value. Change two data bits (`0x80000001`).

Once started, the test displays the following message:

```
Error Correction Code Exception Test executing on cpu n
```

Variable	Description
n	Specifies the CPU used for testing.

If the test executes successfully, the test displays `CPU` *n* `...PASS` on the operator's console. If an error is detected, the test displays `CPU` *n* `...FAIL` on the operator's console followed by one of the following error messages:

- If the expected error status is not present, the test displays:

```
Error Correction Code Exception Test
Expected ECC error status not present, STS_REG = 0xssssssss
```

Variable	Description
ssssssss	Specifies the contents of the processor board status register after the failed operation.

- If the expected level 3 trap did not occur, the test displays:

```
Error Correction Code Exception Test
Expected ECC error (trap 3) did not occur
```

EDRAM Error Correction Code Tests (edramecc)

The EDRAM error correction code tests verify the memory error detection and correction logic on the Expansion DRAM (EDRAM) board. The following error correction code test is available:

- EDRAM Error Correction Code Test

EDRAM Error Correction Code Test

This test verifies the correct operation of the memory error detection and correction logic on the EDRAM board by manipulating the ecc check bits and memory data bits. Testing is first performed across the Local Bus and then across the VMEbus. The test verifies that the correct response is received for the conditions listed in TABLE 2-2

TABLE 2-2 ECC Testing Conditions

Testing Condition	Expected Result
Correct ECC Bits Injected	No ECC Errors
1 Check Bit Changed	Correct Data
1 Data Bit Changed	Corrected Data

Once started, the test displays a start message on the operator's console in the standard format. The test then displays the following message:

```
EDRAM Error Correction Code Test executing on cpu n Base Address 0xaddr
```

Variable	Description
n	Specifies the CPU used for testing.
addr	Specifies the base address of the EDRAM board.

If the test executes successfully, CPU *n* ...PASS is displayed on the operator's console below the start message. If an error is detected, CPU *n* ...FAIL is displayed on the operator's console. The test then displays one of the following error messages depending on the error location:

```
EDRAM Error Correction Code Test
data_value data_write data_diff ecc_v ecc_w ecc_d data_read0 data_read1
    0xorig       0xact      0xdiff     0xvv  0xww  0xdd    0xread0     0xread1     [type]


EDRAM Error Correction Code Test
data_value data_write data_diff ecc_v ecc_w ecc_d data_read2 data_read3
    0xorig       0xact      0xdiff     0xvv  0xww  0xdd    0xread2     0xread3     [type]
```

Variable	Description
orig	Specifies the original 32-bit test data pattern.
act	Specifies the actual data pattern written to four sequential memory words.
diff	Specifies the difference between the original and actual data patterns.
vv	Specifies the ecc check bits for the original test data pattern.
ww	Specifies the ecc check bits injected.
dd	Specifies the difference between the original and injected check bits.
read0	Specifies the first of four words read back from memory (bank 0).
read1	Specifies the second of four words read back from memory (bank 1).
read2	Specifies the third of four words read back from memory (bank 2).
read3	Specifies the fourth of four words read back from memory (bank 3).
type	Specifies the transfer type used for reading the data: wd (word), hw (halfword), ch (character). If the data read back for all three transfer types shows a consistent error, one line of data error information is displayed and this field is blank. If the data read back for a specific transfer type shows an error that is not consistent across the transfer types, the test displays one line of error information for each transfer type.

Correct ECC Bits Injected

The test performs the following operations for this testing condition:

- Writes data values to memory in Normal mode.
- Enters Inject mode.
- Writes the correct ecc value to memory for each data value.
- Reads the data back and verifies that no ecc errors occurred.

This testing condition is checked with the following data values:

Value	Description
`0x00000000`	All zeros data value
`0xffffffff`	All ones data value
`0x010202fd`	All zeros ecc value (`0x00`)
`0x06070801`	All ones ecc value (`0x7f`)
`0x00000001` to `0x80000000`	32 data values, each with 1 bit set
`ecc_table`	128 data values from the test table, which produce ecc values from `0x00` to `0x7f`

One Check Bit Changed

The test performs the following operations for this testing condition:

- Writes data values to memory in Normal mode.
- Enters Inject mode.
- Writes ecc values with one bit changed to memory.
- Reads the data back and verifies that the data returned equals the data written.

This testing condition is checked with the following data values:

Value	Description
`0x010202fd`	All zeros ecc value (`0x00`). Change each ecc bit (`0x01` - `0x40`) for a total of 7 tests.
`ecc_table`	128 data values with ecc values from `0x00` to `0x7f`. Change each ecc bit for each data value for a total of 128*7 tests.

One Data Bit Changed

The test performs the following operations for this testing condition:

- Writes data values with one bit changed to memory in Normal mode.
- Enters Inject mode.
- Writes the ecc value for the original data word to memory.
- Reads the data back and verifies that corrected data was received.

This testing condition is checked with the following data values:

Value	Description
`0x00000000`	All zeros data value. Change each data bit from 0 through 31 for a total of 32 tests.
`ecc_table`	128 data values from the test table with ecc values from `0x00` to `0x7f`. Change each data bit from 0 through 31 for a total of 128*32 tests.

SCSI Interface Tests (`scsi`)

The SCSI interface tests are used to verify the operation of the SCSI interface on the processor adaptor board. The following SCSI interface test is available:

- Bus Master Test

Bus Master Test

This test verifies the operation of the SCSI interface on the processor adaptor board by performing the following operations:

- Loading the SCSI Bus with memory write, read, and compare activity.
- Injecting errors during SCSI disk activity.
- Loading the SCSI Bus and Local Bus with memory write, read, and compare activity.

This test contains three separate routines:

- `scsi_talk`
- Error Inject Test
- `lbus_talk`

Testing operations are performed in the following sequence:

- The `scsi_talk` routine is executed by CPU 0.
- The Error Inject Test is executed by CPU 0.
- If more than one CPU is available, the `scsi_talk` routine is executed by CPU 0 while the other configured CPUs (1-3) execute the `lbus_talk` routine.

Once started, this test displays a start message on the operator's console in the standard format. If test execution is successful, the following message is displayed on the operator's console below the messages displayed by each routine.

```
CPU n ...PASS
```

Variable	Description
n	Specifies the CPU used for testing.

If an error is detected, the test displays the following message on the operator's console. The test then displays an error message associated with the failing routine.

```
CPU n ...FAIL
```

Variable	Description
n	Specifies the CPU used for testing.

scsi_talk

The `scsi_talk` routine is used to create extensive SCSI disk activity. Once started, `scsi_talk` displays the following message:

```
scsi_talk - device chan targ lun executing on cpu 0
```

Variable	Description
chan	Specifies the channel address.
targ	Specifies the target address.
lun	Specifies the logical unit number.

This routine performs the following operations:

- Reads the disk layout to adaptor board memory.
- Reads the disk layout to processor board memory.
- Compares the adaptor and processor boards' memory buffers. If an error is detected, the test displays the following error message:

```
Bus Master Test: scsi_talk
Disk layout compare failure v_addr = 0xvaddr, g_addr = 0xgaddr
```

Variable	Description
vaddr	Specifies the processor board buffer address.
gaddr	Specifies the adaptor board buffer address.

If an error occurs while getting the disk layout, the test displays one of the following error messages:

- If the disk header partition is in Little Endian format, the test displays:

```
Bus Master Test: scsi_talk
Little Endian format header partition detected
WARNING: This is an unreleased and unsupported Header Partition
Image
```

- If the diagnostic partition is too small, the test displays:

```
Bus Master Test: scsi_talk
Diag Partition too small
```

`scsi_talk` then performs the following operations:

- Writes the layout from adaptor board memory to the diagnostic partition. If an error is detected while writing to the disk, the test displays:

```
Bus Master Test: scsi_talk
Disk Write Failure chan targ lun, memory addr = 0xaaaaaaaa
```

Variable	Description
chan	Specifies the channel address.
targ	Specifies the target address.
lun	Specifies the logical unit number.
aaaaaaaa	Specifies the memory address.

- Reads the layout from the diagnostic partition to the adaptor board buffer. If an error is detected while reading from the disk, the test displays:

```
Bus Master Test: scsi_talk
Disk Read Failure chan targ lun, memory addr = 0xaaaaaaaa
```

Variable	Description
chan	Specifies the channel address.
targ	Specifies the target address.
lun	Specifies the logical unit number.
aaaaaaaa	Specifies the memory address.

- Compares the adaptor and processor boards' buffers. If a buffer comparison error is detected, the test displays:

```
Bus Master Test: scsi_talk
Adaptor board layout wr/rd/cmp failure,
v_addr=0xvaddr,g_addr=0xgaddr
```

Variable	Description
vaddr	Specifies the memory address on the processor board.
gaddr	Specifies the memory address on the adaptor board.

- Writes the layout from processor board memory to the diagnostic partition. If an error is detected while writing to the disk, the test displays:

```
Bus Master Test: scsi_talk
Disk Write Failure chan targ lun, memory addr = 0xaaaaaaaa
```

Variable	Description
chan	Specifies the channel address.
targ	Specifies the target address.
lun	Specifies the logical unit number.
aaaaaaaa	Specifies the memory address.

- Reads the layout from the diagnostic partition to the processor board buffer. If an error is detected while reading from the disk, the test displays:

```
Bus Master Test: scsi_talk
Disk Read Failure chan targ lun, memory addr = 0xaaaaaaaa
```

Variable	Description
chan	Specifies the channel address.
targ	Specifies the target address.
lun	Specifies the logical unit number.
aaaaaaaa	Specifies the memory address.

- Compares the adaptor and processor boards' buffers. If a buffer comparison error is detected, the test displays:

```
Bus Master Test: scsi_talk
Processor board layout wr/rd/cmp failure,
v_addr=0xvaddr,g_addr=0xgaddr
```

Variable	Description
vaddr	Specifies the memory address on the processor board.
gaddr	Specifies the memory address on the adaptor board.

- Writes 2048 bytes from adaptor board memory to the diagnostic partition. If an error is detected while writing to the disk, the test displays:

```
Bus Master Test: scsi_talk
Disk Write Failure chan targ lun, memory addr = 0xaaaaaaaa
```

Variable	Description
chan	Specifies the channel address.
targ	Specifies the target address.
lun	Specifies the logical unit number.
aaaaaaaa	Specifies the memory address.

- Reads 2048 bytes from the diagnostic partition to adaptor board memory. If an error is detected while reading from the disk, the test displays:

```
Bus Master Test: scsi_talk
Disk Read Failure chan targ lun, memory addr = 0xaaaaaaaa
```

Variable	Description
chan	Specifies the channel address.
targ	Specifies the target address.
lun	Specifies the logical unit number.
aaaaaaaa	Specifies the memory address.

- Compares the 2048 byte write buffer to the read buffer. If a data comparison error is detected, the test displays:

```
Bus Master Test: scsi_talk
Adaptor board wr/rd/cmp fail, g_waddr=0xwaddr,g_raddr=0xraddr
```

Variable	Description
vaddr	Specifies the write memory address on the adaptor board.
gaddr	Specifies the read memory address on the adaptor board.

- Repeats the write/read/compare operations eight times, incrementing the buffer address by one byte each time.
- Writes 2048 bytes from processor board memory to the diagnostic partition. If an error is detected while writing to the disk, the test displays:

```
Bus Master Test: scsi_talk
Disk Write Failure chan targ lun, memory addr = 0xaaaaaaaa
```

Variable	Description
chan	Specifies the channel address.
targ	Specifies the target address.
lun	Specifies the logical unit number.
aaaaaaaa	Specifies the memory address.

- Reads 2048 bytes from the diagnostic partition to processor board memory. If an error is detected while reading from the disk, the test displays:

```
Bus Master Test: scsi_talk
Disk Read Failure chan targ lun, memory addr = 0xaaaaaaaa
```

Variable	Description
chan	Specifies the channel address.
targ	Specifies the target address.
lun	Specifies the logical unit number.
aaaaaaaa	Specifies the memory address.

- Compares the 2048 byte write buffer to the read buffer. If a data comparison error is detected, the test displays:

```
Bus Master Test:scsi_talk
Processor board wr/rd/cmp fail, v_waddr=0xwaddr,v_raddr=0xraddr
```

Variable	Description
vaddr	Specifies the write memory address on the processor board.
gaddr	Specifies the read memory address on the processor board.

- Repeats the write / read / compare operations eight times, incrementing the buffer address by one byte each time.

Error Inject Test

The Error Inject Test uses CPU 0 to perform the following error injection operations during SCSI disk activity. Once started, this test displays the following message:

```
Error Inject Test executing on cpu 0
```

The Error Inject Test performs the following operations:

- Executes a normal write of one sector to the diagnostic partition and verifies that normal termination occurred. The test displays the following milestone message:

```
SCSI_WRITE, CTRL_REG = 0xset, Expect No Error
```

Variable	Description
set	Specifies the control register setting for this operation.

If an error occurs, the test displays:

```
Bus Master Test: Error Inject Test
SCSI_WRITE, CTRL_REG = 0xcccccccc, GOT UNEXPECTED ERRORS
```

Variable	Description
cccccccc	Specifies the contents of the control register.

- Executes a normal read of one sector to the diagnostic partition and verifies that normal termination occurred. The test displays the following milestone message:

```
SCSI_READ, CTRL_REG = 0xset, Expect No Errors
```

Variable	Description
set	Specifies the control register setting for this operation.

If an error occurs, the test displays:

```
Bus Master Test: Error Inject Test
SCSI_READ, CTRL_REG = 0xcccccccc, STS_REG = 0xssssssss,
GOT UNEXPECTED ERRORS
```

Variable	Description
cccccccc	Specifies the contents of the control register.
ssssssss	Specifies the contents of the status register.

- Compares the processor board write and read buffers. If a comparison error occurs, the test displays:

```
Bus Master Test: Error Inject Test
write/read/compare failure, CTRL_REG = 0xcccccccc
```

Variable	Description
cccccccc	specifies the contents of the control register.

- Reads one sector from the diagnostic partition with the LOCAL BUS CONTROL PARITY ERROR INJECT bit set in the processor control register and displays the following milestone message:

```
SCSI_READ, CTRL_REG = 0xset, Expect Local Bus Control Parity Error
```

Variable	Description
set	specifies the control register setting for this operation.

- Verifies that the LOCAL BUS CONTROL PARITY ERROR bit is set in the processor status register. If an error is detected, the test displays:

```
Bus Master Test: Error Inject Test
SCSI_READ, CTRL_REG = 0xcccccccc, STS_REG = 0xssssssss,
DID NOT GET EXPECTED ERRORS
```

Variable	Description
cccccccc	Specifies the contents of the control register.
ssssssss	Specifies the contents of the status register.

- Reads one sector from the diagnostic partition with the LOCAL BUS ADDRESS PARITY ERROR INJECT bit set in the processor control register and displays the following milestone message:

```
SCSI_READ, CTRL_REG = 0xset, Expect Local Bus Address Parity Error
```

Variable	Description
set	Specifies the control register setting for this operation.

- Verifies that the LOCAL BUS ADDRESS PARITY ERROR bit is set in the processor status register. If an error occurs, the test displays:

```
Bus Master Test: Error Inject Test
SCSI_READ, CTRL_REG = 0xcccccccc, STS_REG = 0xssssssss,
DID NOT GET EXPECTED ERRORS
```

Variable	Description
cccccccc	Specifies the contents of the control register.
ssssssss	Specifies the contents of the status register.

- Writes one sector to the diagnostic partition with the 410 SYSTEM BUS PARITY ERROR INJECT bit set in the processor control register and displays the following milestone message:

```
SCSI_WRITE, CTRL_REG = 0xset, Expect 410 System Bus Parity Error
```

Variable	Description
set	Specifies the control register setting for this operation.

- Verifies that the SCSI LOCAL BUS PARITY ERROR bit is set in the SCSI status. If an error occurs, the test displays:

```
Bus Master Test: Error Inject Test
SCSI_WRITE,CTRL_REG=0xcccccccc, DID NOT GET EXPECTED ERRORS
```

Variable	Description
cccccccc	Specifies the contents of the control register.

- Writes one sector to the diagnostic partition with the processor control register set to INJECT MODE and displays the following milestone message:

```
SCSI_WRITE, CTRL_REG = 0xset, Expect Multi-bit ECC Error
```

Variable	Description
set	Specifies the control register setting for this operation.

- Verifies that the SCSI LOCAL BUS PARITY ERROR bit is set in the SCSI status and the ECC MULTI-BIT ERROR bit is set in the processor status register. If an error is detected, the test displays one of the following error messages:

```
Bus Master Test: Error Inject Test
SCSI_WRITE, CTRL_REG = 0xcccccccc, INJECT ECC ERR,
DID NOT GET EXPECTED ERRORS

Bus Master Test: Error Inject Test
SCSI_WRITE INJECT ECC ERR,
DID NOT GET EXPECTED ERRORS, STS_REG = 0xssssssss
```

Variable	Description
cccccccc	Specifies the contents of the control register.
ssssssss	Specifies the contents of the status register.

lbus_talk

The lbus_talk routine is executed on the other available CPUs (1-3) while CPU 0 executes the scsi_talk routine. Once started, this routine displays the following message:

```
lbus_talk executing on cpu n
```

Variable	Description
n	Specifies the CPU executing the routine.

The lbus_talk routine performs a write/read/compare activity on the Local Bus to create Local Bus activity. If an error is detected, the test displays one of the following error messages:

- If a long word data comparison error occurs, the test displays:

```
Bus Master Test: lbus_talk
long compare error on cpu c
```

Variable	Description
c	Specifies the CPU executing lbus_talk.

- If a halfword data comparison error occurs, the test displays:

```
Bus Master Test: lbus_talk
halfword compare error on cpu c
```

Variable	Description
c	Specifies the CPU executing lbus_talk.

- If a byte data comparison error occurs, the test displays:

```
Bus Master Test: lbus_talk
byte compare error on cpu c
```

Variable	Description
c	Specifies the CPU executing `lbus_talk`.

CHAPTER 3

Extended Diagnostics

The Extended Diagnostics extend the testing philosophy of the Built-in Self Tests (BISTs) to additional boards connected to the backplane. If an error is detected, these tests identify the source of the error and, in many cases, the board causing the error. This provides faster fault isolation and a lower mean time to repair for the types of faults detected with these diagnostics. Once installed, the Extended Diagnostics reside in the header partition of the system disk.

The following Extended Diagnostics are currently available. An individual diagnostic manual is provided for each diagnostic.

- External Interrupt and Clock Cable
- VME Quad Block Mux Channel
- VME SCSI Controller
- MEMORY CHANNEL IV System
- VME Dual Channel ESCON

This chapter contains the following topics:

Test Execution

Extended Diagnostics are executed either automatically or in an interactive environment. During a power up or reset sequence, the ROM Monitor checks to determine whether any extended diagnostics are configured. If so, the ROM Monitor automatically runs them against the appropriate slots after the Built-in Self Tests (BISTs) are executed. This type of automatic testing is bypassed under the following conditions:

- An operator-initiated bypass sequence is started
- Automatic testing is disabled in NVRAM
- The extended diagnostics are not configured
- The BISTs `core` level tests fail

Automatic execution of extended diagnostics may also be initiated by selecting the `slots` testing level with the ROM Monitor `test` command. Refer to *Automatic Testing Procedures* in Chapter 2 for additional information.

Interactive mode allows the operator to select individual tests and execution options related to extended diagnostics. Refer to *Running in Interactive Mode*—page 3-8 for a description of the procedures used to run extended diagnostics in interactive mode.

Note – In interactive mode, the Escape key interrupts the extended diagnostic and returns control to the Interactive Mode Options Menu. No environment variable changes made under the control of the extended diagnostic are preserved when the escape sequence is initiated.

Start Messages

If Extended Diagnostics are configured to run during the automatic testing sequence, the following message indicates the start of testing:

```
Testing backplane cards
```

As each diagnostic is started, a start message is displayed in the following format. The last two lines of the start message are displayed only for the MEMORY CHANNEL IV System diagnostic.

```
Slot s: Loading extended image imagename
Extended name Diagnostic
Diagnostic Revision: rev
Slot: s, Board CSR Addr: 0xaaaaaaaa - 0xbbbbbbbb
[Board Id: id, Major Rev: maj, Minor Rev: min]
[On Board Memory Size: size, Node Id: 0x
```

Variable	Description
s	Specifies the slot number associated with this diagnostic.
imagename	Specifies the name of the diagnostic image configured to this slot.
name	Specifies the name of the extended diagnostic.
rev	Specifies the diagnostic revision level.
aaaaaaaa	Specifies the starting address of the memory address range assigned to the board in this slot.
bbbbbbbb	Specifies the ending address of the memory address range assigned to the board in this slot.
id	Specifies the board ID for MEMORY CHANNEL boards.
maj	Specifies the board major revision for MEMORY CHANNEL boards.
min	Specifies the board minor revision for MEMORY CHANNEL boards.
size	Specifies the size of the on-board memory for MEMORY CHANNEL boards.
n	Specifies the Node ID number for MEMORY CHANNEL boards.

If the extended Diagnostic executes successfully, PASS is displayed below the start message.

When running in interactive mode, the extended diagnostic start message is displayed in the following format:

```
Extended name Diagnostic
Diagnostic Revision: rev
Board CSR Addr: 0xaaaaaaaa - 0xbbbbbbbb
Copyright year Sun Microsystems, Inc.
```

Variable	Description
name	Specifies the name of the extended diagnostic.
rev	Specifies the diagnostic revision level.
aaaaaaaa	Specifies the starting address of the memory address range assigned to the board in this slot.
bbbbbbbb	Specifies the ending address of the memory address range assigned to the board in this slot.
year	Specifies the year the diagnostic was copyrighted.

The program then displays the Interactive Mode Options Menu (see *Interactive Mode Options*—page 3-10).

Once test execution is started in interactive mode, the MEMORY CHANNEL IV System diagnostic displays the following additional information:

```
Board Id: id, Major Rev: maj, Minor Rev: min
On Board Memory Size: size, Node Id: 0xn
```

Variable	Description
id	Specifies the board ID.
maj	Specifies the board major revision.
min	Specifies the board minor revision.
size	Specifies the size of the onboard memory.
n	Specifies the Node ID number.

Note – If a board has two CSR addresses, the address range is displayed for both addresses.

Error Reporting

When Extended Diagnostics run as part of the automatic testing sequence, a failure is indicated by an error header message identifying the failing diagnostic and test, a diagnostic-specific error message, and a card failure message.

The error header message identifies the failing diagnostic and test in the following format:

```
Error[x-y]:Slot n: CPU 0: name: Test t [s]: testname
```

Variable	Description
x-y	Specifies error numbering information, where *x* is the number of this error and *y* is an index into the error log pointing to the last entry to be inspected. For example, [2-1] indicates that this is error 2 and the last error log entry inspected was 1.
n	Specifies the slot number of the board running the diagnostic when the failure occurred.
name	Specifies the mnemonic of the diagnostic reporting the failure.
t	Specifies the number of the failing test.
s	Specifies the letter of the failing subtest, if applicable.
testname	Specifies the name of the failing test.

The diagnostic-specific error message is unique to the failing program.

The failing board is identified with the following message:

```
************************************************************
Card in slot n failed extended diagnostic: image
************************************************************
```

Variable	Description
n	Identifies the slot location of the board reporting the error.
image	Identifies the name of the failing extended diagnostic image.

Note – Error messages during autotest are logged and can be displayed with the ROM Monitor `errorlog` or `nvlog` command.

If an extended diagnostic fails in interactive mode, several lines of error information are reported. The action taken when an error occurs is determined by the execution options in effect when the error was detected. The following is an example of an extended diagnostic error message. The format will vary with the type of diagnostic.

```
Extended diagname Diagnostic Failed     Time: hh:mm:ss
Board CSR Addr: 0xaaaaaaaa - 0xbbbbbbbb [Interrupt Priority: i]
[VME Base Addr: 0xcccccccc, MC Base Addr: 0xdddddddd]
[Local Base Addr:0xeeeeeeee, On Board Memory Size:0xmm, Node Id:0xn]
Error: Slot 0, CPU 0: name: Test t [s]: testname
error message
```

Variable	Description
diagname	Specifies the name of the failing diagnostic.
hh:mm:ss	Specifies the elapsed execution time when the error occurred.
aaaaaaaa	Specifies the starting address of the memory address range assigned to this board.
bbbbbbbb	Specifies the ending address of the memory address range assigned to this board.
i	Specifies the interrupt priority level. The priority level is displayed only if interrupts are enabled. Not all extended diagnostics display the interrupt priority level.
cccccccc	Specifies the VME Bus base address, if applicable to this diagnostic.
eeeeeeee	Specifies the Local Bus base address, if applicable to this diagnostic.
mm	Specifies the size of the on-board memory, if applicable to this diagnostic.
n	Specifies the Node ID number, if applicable to this diagnostic.
name	Specifies the mnemonic of the diagnostic reporting the failure.
t	Specifies the number of the failing test.
s	Specifies the letter of the failing subtest, if applicable.
testname	Specifies the name of the failing test.
error message	Specifies a diagnostic-specific error message associated with this failure.

Refer to the specific extended diagnostic manual for error message descriptions.

Configuring Extended Diagnostics

Before you run extended diagnostics during autotest or with the `test` command, they must be configured in NVRAM to the appropriate boards and slots. A series of ROM Monitor `setconfig` commands are used to configure each extended diagnostic.

Note – The values supplied with the `setconfig` commands depend on the system configuration.

Enter the following series of commands to configure an extended diagnostic:

```
ROM >> setconfig slot n state newstate
ROM >> setconfig slot n type boardtype
ROM >> setconfig slot n scsr 1 address
ROM >> setconfig slot n eimage image
```

Variable	Description
n	Indicates the slot number of the board requiring the extended image.
newstate	Indicates the required state for this slot: `configured` or `deconfigured`.
boardtype	Indicates the mnemonic for this type of board: `cpu`, `mem`, `vme`, `mcss`, or `mcsm`.
address	Indicates the address of the Reset Control/Status Register (csr) on this board.
image	Indicates the extended diagnostic image associated with the board in this slot. Use the monitor `listimages` command to display the images on the system disk.

Note – Some extended diagnostics require additional configuration information. For example, the VME SCSI Controller diagnostic requires two CSR addresses when testing a Quad Channel SCSI-2 Controller. Refer to the individual diagnostic manual to determine whether an extended diagnostic requires additional information.

Repeat the sequence of commands for each extended diagnostic you want to configure, and then save the changes in NVRAM:

```
ROM >> nvram accept
Are you sure you want to save the configuration? (y/n) y
nvram modified
ROM >>
```

To display the configuration for a specific slot, enter the ROM Monitor `readconfig` command in the following format:

```
ROM >> readconfig slot n all
```

Variable	Description
n	Specifies the desired slot number.

Running in Interactive Mode

The extended diagnostic interactive mode allows you to manipulate the diagnostic operating environment and parameters. To enter interactive mode, use ROM Monitor commands to load and start the desired extended diagnostic image.

Note – Use Control-c to terminate an operation in interactive mode.

Use the Escape key to abort an extended diagnostic and return control to the Interactive Mode Options Menu.

Environment variable changes made under the control of the extended diagnostic may be preserved when the escape or Control-C sequence is initiated, depending on the diagnostic. Refer to the individual diagnostic manual to determine whether a specific diagnostic will preserve these changes.

Loading Extended Diagnostics

Two methods are available for loading Extended Diagnostics. The `loadimage` command is used to load the Extended Diagnostic from the default boot device. If loading from the default boot device fails, use the `get` command to load the Extended Diagnostic from the console. Use one of the following:

```
ROM >>loadimage diagname [bus chan target lun]
ROM >>get IPaddr imagename 0x3f600000
```

Variable	Description
diagname	Specifies the name of the diagnostic (`scsidiag`, for example).
bus	Specifies an index or the hexadecimal value of a valid VME short address as follows:

Index number	Description
0	On-board SCSI
1	VME Controller 0x8800
2	VME Controller 0x9000
3	VME Controller 0x9800
.	
.	
.	
15	VME Controller 0xf800

chan target lun	Specifies the channel, target, and logical unit of the disk containing the image to be loaded. If no channel, target, and logical unit are specified, the `loadimage` command defaults to loading from the default boot device (normally 0 6 0).
IPaddr	Specifies the IP address of the console. The Ethernet address must be manually loaded into the console's `arp` table. Refer to the `arp` man page for additional information.
imagename	Specifies the name of the diagnostic image to be loaded from the console disk (`scsi.image`, for example).

Note – The `get` command cannot be executed concurrently on multiple subsystems.

The Ethernet address must be manually loaded into the console's arp table. Refer to the `arp` man page for additional information.

Starting Extended Diagnostics

Once the diagnostic image has been loaded, enter the following command line to start the image and enter interactive mode:

```
ROM >> go 0 0x3f600000
```

Once the image is started, the Interactive Mode Options Menu is displayed automatically on the operator's console.

Interactive Mode Options

Once an extended diagnostic is started in interactive mode, it displays its start message followed by a menu of options. The interactive mode options menu is displayed on the operator's console in the following format:

```
Valid Options                  Current State
-------------                  -------------
a   -  Automatic testing.      (disabled)
c   -  Select reset CSR.       (0xaaaaaaaa)
d   -  Loop on diagnostic.     (1)
e   -  Extensive testing.      (enabled)
f   -  Loop on failing test.   (disabled)
h   -  Help.
i   -  Initialize Diag.
q   -  Quit.
r   -  Run.
s   -  Stop on error.          (enabled)
t   -  Select test range.      (disabled)
Enter your selection:
```

Variable	Description
aaaaaaaa	Specifies the board base address. The value displayed in the menu when the diagnostic is started is board dependent.

Enter the letter corresponding to the desired option and press the Return key. Some of the options toggle between enabled and disabled when they are selected. Other options prompt for additional information.

Note – Use `q` to exit the diagnostic and return to the ROM Monitor prompt (`ROM >>`) and use `h` to display help information related to the menu selections. The other option selections are described in the following sections.

Changing the CSR

Use the `c` option to modify the Control/Status Register (CSR) address (board base address). Once selected, this option displays the following prompt:

```
Enter board base address = 0xaaaaaaaa ;[cr,?,^,(hex)]?0x
```

Enter the desired hexadecimal board base address (`hex`) or one of the following responses:

Response	Description
`cr`	Press the Return key to select the default address displayed in the prompt (*aaaaaaaa*).
`?`	Displays help information for this prompt.
`^`	Redisplays the Interactive Mode Options Menu.

If the `c` option is not selected, the diagnostic uses the default CSR address for the specific controller. Typically, the CSR address is 0xfff40000 plus the Short I/O address of the controller.

Initializing a Diagnostic

Use the `i` option to enter the diagnostic initialization mode. The diagnostic will then begin displaying initialization prompts. Refer to the individual diagnostic manual for information about initialization prompts and prompt responses.

Selecting Diagnostic Tests

Three mutually exclusive options are associated with the selection of diagnostic tests: `a`, `e`, and `t`.

The `a` option enables or disables the selection of a predetermined set of tests selected for automatic testing mode.

The e option enables or disables the selection of all diagnostic tests.

The t option allows the selection of individual tests. Once selected, this option displays the following prompt:

```
Enter test selection(s) = tests ;
[cr, ?, ^, a, l, #(,#,...),#-#]?
```

Enter one or more individual test numbers (#), a range of test numbers (#-#), or one of the following responses. Use a space or comma between individual test numbers.

Response	Description
cr	Press the Return key to select the default tests displayed in the prompt (*tests*).
?	Displays help information for this prompt.
^	Redisplays the Interactive Mode Options Menu.
a	Selects all diagnostic tests.
l	Displays a list of the available tests for this diagnostic.

The selected tests are displayed in the following format:

```
selected test ttt: testname
```

Variable	Description
ttt	Specifies the number of the selected test.
testname	Specifies the name of the selected test.

Selecting Loop and Error Related Options

The following options are related to selecting diagnostic loop features and the desired action when an error occurs:

Option	Description
d	Use this option to specify the number of passes of the diagnostic to be run.
f	This option enables or disables the ability to loop on a failing test.
s	This option enables or disables terminating diagnostic execution when an error is detected.

If you select the d option, the following prompt is displayed:

```
Enter loop count = n ;[cr, ?, ^, 0, #]?
```

Enter the number of passes of the diagnostic to run (#) or one of the following responses:

Response	Description
cr	Press the Return key to select the default pass count displayed in the prompt (*n*).
?	Displays help information for this prompt.
^	Redisplays the Interactive Mode Options Menu.
0	Causes the program to loop indefinitely until Control-c is entered.

Running Diagnostic Tests

Once the diagnostic tests have been selected with the a, e, or t option, select the r option to start test execution.

CHAPTER 4

The Diagnose Program

Introduction

The Diagnose program is a system console application program for the Sun StorEdge A7000 Intelligent Storage Server System. Diagnose provides a graphical point-and-click environment for running diagnostics through the system console software. This program provides simple pass/fail information by changing the colors of the graphics displayed on the system console.

Perform the following operations to use Diagnose to test system components:

- Display a graphical representation of the system on the console
- Select a component for testing
- Start the Diagnose program
- Interpret the test results

This chapter contains the following topics:

Special Requirements

The Diagnose program requires the following software and revisions:

- Master Configuration Data (MCD), Revision 1.3 or higher, installed on the console
- ROM Monitor, Revision 2.77 or higher, installed on the Processor Boards
- E90DIAGS package, Revision 1.4 or higher, installed on the console and / or system disk

---

Note – Before using Diagnose, ensure that the subsystem containing the board or device to be tested is in ROM Monitor mode.

Displaying the System Components

Use the Master Configuration Data (MCD) program to display a graphical representation of the system configuration by performing the following operations:

1. Move the mouse pointer to the `Utilities` selection on the system console Application Menu Bar.
2. Press the left mouse button. Drag the pointer to the `MCD` selection and release the left mouse button to display the MCD (Master Configuration Data) Menu Bar.
3. Move the pointer to the `View` selection on the MCD Menu Bar and press the left mouse button. Drag the pointer to `Bus View` and release the left mouse button.

   A graphical view of the top level system connectivity appears.
4. Either select a subsystem for testing or select an individual subsystem component. To select a subsystem, follow the procedure in *Selecting a Subsystem*—page 4-3. To select an individual component, follow the procedure in *Selecting an Individual Subsystem Component*—page 4-3.

Selecting a Subsystem

To select a subsystem for testing:

1. Move the pointer to the graphic representing the subsystem you want to test.
2. Click the left mouse button.
3. Proceed to *Starting the Diagnose Program*—page 4-5 to start testing the selected subsystem.

Selecting an Individual Subsystem Component

You must expand the Busview before selecting an individual subsystem component for testing. To select an individual subsystem component:

1. Move the pointer to the graphic of the subsystem containing the component you want to test.
2. Double-click on the desired subsystem with the left mouse button. The Busview expands to show a diagram of the components residing in the selected subsystem. Use the scroll bar to display additional components.

   Either select a component for testing or select an individual device (tape or disk). To select an individual device, follow the procedure in *Selecting a Device*—page 4-4. To select an individual component, proceed to Step 3.
3. Move the pointer to the component you want to test.
4. Click the left mouse button on the selected component.
5. Follow the procedure in *Starting the Diagnose Program*—page 4-5 to start testing the selected component.

Note – If you select a SCSI-2 Controller for testing, all the configured devices on that controller are also tested.

There are different types of Cache Memory Modules. The components displayed in the expanded Busview for a specific system are determined by the system configuration.

Selecting a Device

You must expand the Busview a second time before selecting a device for testing. To select an individual device:

1. Move the pointer to the graphic of the component interfacing to the device you want to test.
2. Double-click the left mouse button on the component. The Busview expands to show the configured devices. Use the scroll bar to display additional devices.
3. After you display the devices on the selected component, move the pointer to the device you want to test.
4. Click the left mouse button on the selected device.
5. Follow the procedure in *Starting the Diagnose Program* to start testing the selected device.

Note – To test all the devices configured on a SCSI-2 Controller, select only the controller for testing (refer to *Selecting an Individual Subsystem Component*—page 4-3).

Starting the Diagnose Program

After you select a component, device, or subsystem for testing, perform the following steps to start the Diagnose program:

1. With the pointer on the selected component, press the right mouse button.
2. Drag the pointer down to the `Diag` selection.
3. Drag the pointer right to the `Diagnose` selection.
4. Release the right mouse button.

The Diagnose program runs the appropriate tests on the component. A start message similar to the following is displayed in the text box at the bottom of the Busview screen and the graphic representing the component under test changes to light blue:

```
Diagnose <13184> : <Started>
```

When Diagnose testing is complete, the following type of message is displayed:

```
Diagnose <13184> : <Completed>
```

If testing was successful, the graphic representing the component under test changes to clear. Refer to *Recognizing Failures* for descriptions of failure indications.

Recognizing Failures

Diagnose uses the colors red and yellow to indicate test failures. The color displayed is determined by the system component under test. Refer to one of the following paragraphs for specific failure indications.

Subsystem Testing Failure Indications

If you are testing a complete subsystem, Diagnose makes the following color changes:

- As a component in the subsystem is tested, its graphic turns light blue. If the component passes the tests, its graphic turns clear.
- If a processor board fails during testing, its graphic turns red and the graphic representing the subsystem also turns red. A processor board failure is fatal to subsystem operation.
- If a component other than the processor board fails during testing, its graphic turns red and the graphic representing the subsystem turns yellow. This indicates the subsystem is marginal and needs service.

Subsystem Component Testing Failure Indications

If you are testing any individual subsystem component, the graphic representing the component turns light blue when testing is started, clear if testing was successful, and red if an error is detected.

Device Testing Failure Indications

If you are testing an individual device, the graphic representing the device turns light blue when testing is started, clear if testing was successful, and red if an error is detected.

Diagnose Log Files

The Diagnose program maintains two types of log files, activity and error, in the `/usr/conslog/diagnose` directory on the System Console hard disk.

The activity log file (`activity.log`) records the date and time a diagnostic session begins and an error log reference, if a test failure occurs. Error log file names are derived from the subsystem and failing component mnemonics. For example, the error log `dsp1_cpu-0.errlog` is named for the subsystem (`dsp1`) and the failing component (`cpu-0`). Error logs are located in the `/usr/conslog/diagnose` directory.

The following are examples of Diagnose activity logs:

- An error occurred while testing a specific controller. The controller number (13) relates to an entry in the MCD database table. The error log (`dsp1_cpu-0.errlog`) contains all the diagnostic messages produced during the testing session.

```
/* DIAGNOSE CONTROLLER - CONTROLLER FAILED DIAGNOSTIC */
[Jul 22 1998 08:36:53] STARTED:  diagnose session on CONTROLLER 13
[Jul 22 1998 08:39:50] ERROR:    test failed - see dsp1_cpu-0.errlog
[Jul 22 1998 08:39:54] FAILED:   diagnose session on CONTROLLER 13
```

- An error occurred while testing a subsystem. Controller number 21 was under test when the error occurred. The error log (`io_m328-3.errlog`) contains all the diagnostic messages produced during the testing session. Both the subsystem and controller numbers relate to entries in the MCD database table.

```
/* DIAGNOSE SUBSYSTEM - CONTROLLER FAILED DIAGNOSTIC */
[Jul 22 1998 08:36:53] STARTED:diagnose session on SUBSYSTEM 3
[Jul 22 1998 08:36:53] STARTED:diagnose session on CONTROLLER 17
[Jul 22 1998 08:36:53] COMPLETED:diagnose session on CONTROLLER 17
[Jul 22 1998 08:36:53] STARTED:diagnose session on CONTROLLER 18
[Jul 22 1998 08:36:53] COMPLETED:diagnose session on CONTROLLER 18
[Jul 22 1998 08:36:53] STARTED:diagnose session on CONTROLLER 19
[Jul 22 1998 08:36:53] COMPLETED:diagnose session on CONTROLLER 19
[Jul 22 1998 08:36:53] STARTED:diagnose session on CONTROLLER 20
[Jul 22 1998 08:36:53] COMPLETED:diagnose session on CONTROLLER 20
[Jul 22 1998 08:36:53] STARTED:diagnose session on CONTROLLER 21
[Jul 22 1998 08:39:50] ERROR:test failed - see io_m328-3.errlog
[Jul 22 1998 08:39:54] FAILED:diagnose session on CONTROLLER 21
[Jul 22 1998 08:39:54] FAILED:diagnose session on SUBSYSTEM 3
```

Note – If diagnostics are run a second time and a different error occurs, the contents of the error log are replaced with the messages from the new diagnostic session.

Displaying a Log File

Use the Log Browser utility on the system console to display the contents of a log file by performing the following operations:

1. Move the pointer to the `Utilities` selection on the system console Application Menu Bar.
2. Press the left mouse button. Drag the pointer to the `Log Browser` selection and release the left mouse button.
3. The system console displays a window outline. Drag the outline to an appropriate place on the screen and click the left mouse button. The console then displays a Log Browser window.
4. Move the pointer to `File` and click the left mouse button. A menu of file-related options is displayed.
5. Click the left mouse button on `OPEN` to display the `Open File Dialog` box:
6. Move the pointer to the right of the asterisk (*) in the `Filter` box. Press the left mouse button. Drag the pointer to the left and highlight the contents of the `Filter` box. Type **`/usr/conslog/diagnose/*`** and press the Enter key.
7. The `File` box displays the Diagnose log files. Click the left mouse button on the file you want to open and then click the left mouse button on the `OK` square at the bottom of the window. Use the scroll bar on the right side to advance through the file.