Origin2000™ and Onyx2™ Power-On Diagnostics

Document Number 108-0161-002

Contributors

Written by Darrin Goss
Illustrated by Darrin Goss
Edited by Mary Geisert
Production by Carlos Miqueo
Engineering contributions by Curt McDowell

Origin2000 and Onyx2 Power-On Diagnostics
Document Number 108-0161-002

Silicon Graphics, Inc.
Mountain View, California

Contents

Figures

Tables

About This Guide

This document describes the power-on diagnostics (POD) for the Origin2000 and Onyx2 series computer systems. It is written for System Support Engineers (SSEs) to use as a reference document in training and on-site.

The information contained in this document is organized into the following chapters:

- Chapter 1, “Introduction,” provides an overview of and introduction to the power-on diagnostics. This Chapter introduces the various POD modes, the IP27 PROM tests, the BaseIO PROM tests, the debug DIP switches, how to send input to the power-on diagnostics, and how the power-diagnostics return output.
- Chapter 2, “Running the Power-On Diagnostics,” describes how you can run the power-on diagnostics automatically as part of the boot sequence and manually from the POD prompt or BaseIO command prompt.
- Chapter 3, “IP27 PROM Tests,” describes the power-on diagnostic tests that are included in the IP27 PROM.
- Chapter 4, “BaseIO PROM Tests,” describes the power-on diagnostic tests that are included in the BaseIO PROM.
- Appendix A, “Node Board LED Values,” decodes all Node board LED values that the boot process and power-on diagnostics use.
- Appendix B, “diag_rc Values,” decodes all diag_rc values that the power-on diagnostics return.

Note: This document supports PROM version 6.11; the POD mode *version* command shows the PROM version that is loaded on your system.

Notational Conventions

This document uses the following notational conventions:

- Information displayed on the screen is shown in `Courier` type.
- Commands that you should enter are shown in **`Courier bold`** type.
- Commands in text are shown in *italic* type.
- Variables are shown in *italic* type.

Terminology

This document uses the following terms interchangeably:

- Bridge, Bridge ASIC, and Bridge chip
- HUB, HUB ASIC, and HUB chip
- Crossbow, XBOW, XBOW ASIC, and XBOW chip

Chapter 1

Introduction

This chapter provides an overview of the power-on diagnostics (POD) for Origin2000 and Onyx2 systems. The power-on diagnostics test the components necessary to load the operating system or the micro-diagnostic kernel (MDK). These diagnostics run as part of the normal boot sequence; you can also run several of these diagnostics from the IP27 PROM POD-mode command line or the BaseIO PROM command monitor.

1.1 Overview of Power-On Diagnostics

The power-on diagnostics (POD) are programmable read-only memory (PROM) resident routines that run automatically during the boot sequence. You can also run several of the power-on diagnostics manually from the IP27 PROM POD mode command line or the BaseIO PROM command monitor.

Power-on diagnostics focus on the hardware needed to boot the operating system or the micro-diagnostic kernel (MDK). The routines that the power-on diagnostics use are stored in the IP27 PROM and the BaseIO PROM.The POD diagnostics run in four modes: no diags mode, normal mode, heavy mode, and manufacturing mode. Table 1-1 describes the modes and the DIP switch settings you can use to select the modes. Refer to Section 1.2, “Modifying Power-On Diagnostics Execution,” for more information about how to set the DIP switches.

Table 1-1 POD Modes

Mode	Description[a]
No Diags	This mode performs minimal testing; only the initialization functions of the power-on diagnostics (for example, clearing memory) are performed. (This mode also checks the memory configurations and the real-time clock.) To set this mode, put DIP switch 1 in the On position and DIP switch 2 in the Off position.
Normal	This mode performs tests to ensure that the operating system or MDK can run. Because of the time constraints of the normal boot process, this mode performs a subset of the testing done by heavy mode. To select this mode, put both DIP switches in the Off position.
Heavy	This mode performs more extensive and time-consuming tests than normal mode. Use this mode the first time that you power on a system in the field or when the normal mode detects errors. To select this mode, put DIP switch 1 in the Off position and DIP switch 2 in the On position.
Manufacturing	This mode provides more detailed status and failure information than heavy mode. This mode may run more extensive tests than heavy mode. (Normally, this mode is used only in the manufacturing environment.) To select this mode, put both DIP switches in the On position.

a. The DIP switches described in this table are located on a blue block next to the module system controller (MSC) keyswitch. The DIP switches are numbered from the left, so DIP switch number 1 is the first switch on the left and DIP switch number 2 is the second switch from the left. Refer to Section 1.2, “Modifying How the Power-On Diagnostics are Run.”

The power-on diagnostics are divided into two sections: IP27 tests and BaseIO tests. Table 1-2 lists the IP27 PROM tests by the functional unit that they test. Table 1-3 lists the BaseIO PROM tests by the functional unit that they test. The “Page” column in the tables indicates the page on which you can find a detailed description of the test.

Table 1-2 IP27 PROM Tests

Functional Unit	Test	Failure LED Pattern [a]	Failure Message	Failing FRU	Page
R10000 processors	Register test	0x81 (FLED_CP1) 	n / a	Node board	3-2
	Primary instruction cache test	0x83 (FLED_ICACHE) 	n / a	Node board	3-3

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table 1-2 (continued) IP27 PROM Tests

Functional Unit	Test	Failure LED Pattern [a]	Failure Message	Failing FRU	Page
R10000 processors (continued)	Primary data cache test	0x84 (FLED_DCACHE) 	n/a	Node board	3-3
	Interrupt test	0x90 (FLED_HUBINTS) 	`RSLT hub_int_diag FAIL`	Node board	3-4

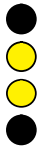
a. = Illuminated LED = 0
= Unilluminated LED = 1

Table 1-2 (continued) IP27 PROM Tests

Functional Unit	Test	Failure LED Pattern [a]	Failure Message	Failing FRU	Page
Secondary cache	Secondary cache test	0x85 (FLED_SCACHE) 	`RSLT cache_test_s FAIL`	Node board	3-5
HUB	GClk logic test	n/a	`RSLT rt_clock_test FAIL`	Node board	3-6
	BTE test	n/a	`RSLT hub_bte_diag FAIL`	Node board	3-7
PROM	PROM checksum test	n/a	`RSLT prom_checksum FAIL`	Node board	3-8

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table 1-2 (continued) IP27 PROM Tests

Functional Unit	Test	Failure LED Pattern [a]	Failure Message	Failing FRU	Page
Memory and directory	Memory configuration test	0x98 (FLED_NOMEM)  or 0xa9 (FLED_BADMEM) 	`No Memory Found`, `*** Mixed standard and premium memory`, or `nasid` *node_number*`: disabling bank` *bank_number*	DIMM	3-9
	Backdoor directory space test	n / a	`RSLT dirtest FAIL` or `MEMORY FAILURE`	DIMM	3-10
	Memory test	n / a	`RSLT memtest FAIL` or `MEMORY FAILURE`	DIMM	3-12
CrayLink	Local link connection test	n / a	`RSLT chk_local_link FAIL`	Midplane, Node board, or Router	3-14

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table 1-2 (continued) IP27 PROM Tests

Functional Unit	Test	Failure LED Pattern [a]	Failure Message	Failing FRU	Page
CrayLink (continued)	HUB send-data error test	n/a	`RSLT hub_send_data_error FAIL`	Router, Node board, or midplane	3-15
	HUB error detection test	n/a	`RSLT test_hub_error_detection FAIL`	Node board, Router, or midplane	3-16
	Router send-data error test	n/a	`RSLT rtr_send_data_error FAIL`	Node board, Router, or midplane/cable	3-17
	Router error detection test	n/a	`RSLT test_rtr_error_detection FAIL`	Router, Node board, or midplane/cable	3-18
	Get chipid test	n/a	`RSLT get_chipid FAIL`	Node board or Router	3-19
XBOW	XBOW test	n/a	`RSLT xbow_sanity FAIL`	Midplane or Node board	3-20
Bridge	Bridge test	n/a	`RSLT bridge_sanity FAIL`	BaseIO board	3-22
PCI	PCICONFIG test	n/a	`RSLT io6config_space FAIL`	BaseIO board	3-24
	PCI bus test	n/a	`RSLT pcibus_sanity FAIL`	BaseIO board	3-25
Serial port	Serial programmed I/O test	n/a	`RSLT serial_pio FAIL`	BaseIO board	3-27
	Serial DMA test	n/a	`RSLT serial_dma FAIL`	BaseIO board	3-29

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table 1-3 BaseIO PROM Tests

Functional Unit	Test	Failure Message	Failing FRU	Page
SCSI	SCSI SSRAM test	`RSLT scsi_ram FAIL`	BaseIO board	4-3
	SCSI DMA test	`RSLT scsi_dma FAIL`	BaseIO board	4-4
	SCSI self-test	`RSLT scsi_controller`	BaseIO board	4-5
Ethernet	Comprehensive Ethernet test	`RSLT enet_all FAIL`	BaseIO board	4-8
	Ethernet SSRAM test	`RSLT enet_ssram FAIL`	BaseIO board	4-11
	TX clock test	`RSLT enet_tx_clk FAIL`	BaseIO board	4-13
	PHY register test	`RSLT enet_phy_reg FAIL`	BaseIO board	4-14
	NIC test	`RSLT enet_nic FAIL`	BaseIO board	4-16
	IOC3 internal loopback DMA test	`RSLT enet_ioc3_loop FAIL`	BaseIO board	4-17
	PHY chip internal loopback test	`RSLT enet_phy_reg FAIL`	BaseIO board	4-20
	Twister chip internal loopback test	`RSLT enet_tw_loop FAIL`	BaseIO board	4-23
	External loopback DMA test	`RSLT enet_ext_loop FAIL`	BaseIO board	4-24
	External loopback DMA (10 Mb/s) test	`RSLT enet_10_ext_loop FAIL`	BaseIO board	4-27
	External loopback DMA (100 Mb/s) test	`RSLT enet_100_ext_loop FAIL`	BaseIO board	4-29
	XTALK stress test	`RSLT enet_xtalk_stress FAIL`	BaseIO board	4-31

1.2 Modifying Power-On Diagnostics Execution

The Module System Controller (MSC) NVRAM includes two sets of debug switch values: eight physical switch values (numbered 1 through 8) and eight virtual debug switch values (numbered 9 through 16).

You can view and change the debug switch settings with the MSC *dbg* command, the POD mode *dips* command, or the POD mode *dbg* command. (Refer to the *IP27PROM Technical Reference Manual*, publication number 108-0170-001, for more information about these commands.)

Table 1-4 describes the function of each physical switch.

Table 1-4 DIP Switch Assignments

Switch	Description[a]
1	Selects the diagnostic mode (with switch 2)
2	Selects the diagnostic mode (with switch 1) Switches 1 and 2 set the mode as follows: Turning switch 1 Off and switch 2 Off selects “normal” mode. Turning switch 1 Off and switch 2 On selects “heavy” mode. Turning switch 1 On and switch 2 Off selects “no diags” mode. Turning switch 1 On and switch 2 On selects “manufacturing” mode. For more information about the diagnostic modes, refer to Table 1-1.
3	Selects the boot information level When switch 3 is set to On, the PROM code displays detailed information messages during the boot sequence. When switch 3 is set to Off, the PROM code displays only normal boot status messages.
4	Selects the boot stop point (with switch 5)
5	Selects the boot stop point (with switch 4) Switches 4 and 5 set the boot stop point as follows: Turning switch 4 Off and switch 5 Off sets the boot stop point to “never.” This setting causes the boot process to completely boot the operating system. Turning switch 4 Off and switch 5 On sets the boot stop point to “local.” This setting causes the boot process to stop at the point where it would normally enter the BaseIO PROM code. All Nodes enter cached (Cac) POD mode. This setting affects all modules in the system. Turning switch 4 On and switch 5 Off sets the boot stop point to “global.” This setting causes the boot process to stop at the point where it normally would jump to the BaseIO PROM code. Then, the master Node enters cached (Cac) POD mode, and the slave Nodes enter the slave loop. This setting affects all modules in the system. Turning switch 4 On and switch 5 On sets the boot stop point to “memoryless.” This setting causes the boot process to stop as soon as the system is able to enter POD mode. All Nodes enter dirty exclusive (Dex) POD mode.

Table 1-4 (continued) DIP Switch Assignments

Switch	Description[a]
6	Selects the default environment When switch 6 is set to On, the PROM code ignores all PROM log environment variables and all BaseIO NVRAM settings; instead, the PROM code uses the system defaults. When switch 6 is set to Off, the PROM code uses all PROM log environment variables and all BaseIO NVRAM settings. It is useful to set switch 6 to On if any of the variable storage devices contain data that prevents the system from booting.
7	Specifies whether or not to bypass the first BaseIO board in the system If switch 7 is set to On, the PROM code bypasses the first BaseIO board that it finds in the system and tries to boot from the second BaseIO board. If switch 7 is set to Off, the PROM code tries to boot from the first BaseIO board. It is useful to set switch 7 to On if the first BaseIO board in the system is not working; this enables you to boot the system without removing the failing BaseIO board.
8	Specifies whether or not to bypass the global master If switch 8 is set to On, the Node that would normally become the global master becomes the slave Node; the next CPU in line becomes the global master.

a. You should set all switches to Off for normal operation.

1.3 Console Input to the Power-On Diagnostics

The power-on diagnostics can interact with different types of consoles, depending on how your system is configured. Table 1-5 describes the three console types that are available.

Table 1-5 Consoles From Which the Power-On Diagnostics Can Receive Input

POD Console	Description
IOC3 UART	If a BaseIO board is available with a terminal connected to a port marked for the system console, POD expects input from the IOC3 UART on the BaseIO board. (LED pattern: 0x80 0x8c)
MSC UART	If no IOC3 is available and the MSC is responsive, POD expects input from the alternate diagnostic console that is connected to the MSC UART. (LED pattern: 0x80 0xbc)
Netuart	If none of the physical UARTs are available, POD expects input from a netuart. A netuart emulates a physical UART to enable one HUB to talk to another. When a CPU uses a netuart, it monitors shared memory locations and interrupt lines that it uses for communication. From any CPU that has a UART and is at the POD prompt, you can use the *talk* command to communicate with CPUs that are using a netuart.

POD mode checks for the consoles in the order shown in Table 1-5. When a CPU is at the POD prompt, the LED pattern for the selected console blinks at 0.5 Hz on the LEDs for the CPU.

1.4 Output From the Power-On Diagnostics

The power-on diagnostics send output to the Node board LEDs and to a console.

1.4.1 Node Board LEDs

Each Node board has two side-by-side columns of eight LEDs. While the system is booting, the Node board LEDs indicate the progress of the boot. If the system crashes while it is booting, you can use the value shown by the LEDs to determine where in the boot process the crash occurred. The left column of LEDs indicates the status of CPU A on the Node board; the right column of LEDs indicates the status of CPU B.

The columns of LEDs have the most significant bit at the top. An illuminated LED indicates a 0 value for that position; an unilluminated LED indicates a 1 value. Figure 1-1 shows several examples of LED values.

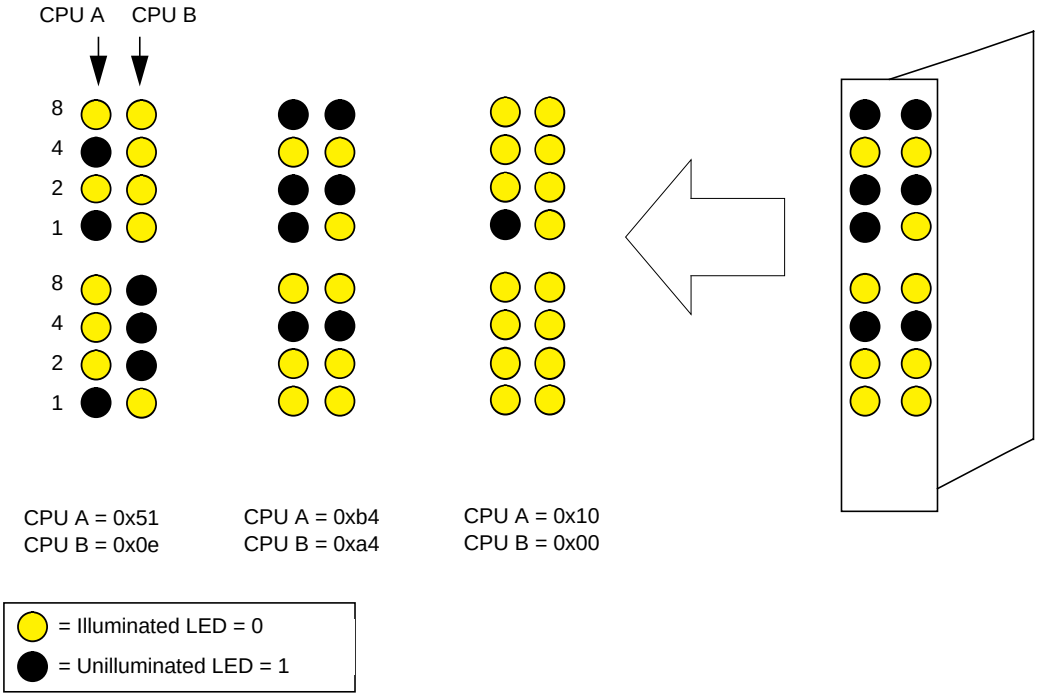


Figure 1-1 Reading the LED Patterns (Example)

Notice that the values for the LEDs are inverted from what you might expect. An illuminated LED has a value of 0, and an unilluminated LED has a value of 1.

Refer to Appendix A, “Node Board LED Values,” for a list of all LED patterns that the boot process and power-on diagnostics use.

1.4.2 Console Output

The power-on diagnostics also return failure messages to the alternate console port (ACP) on the MSC and to the system console. For more information about the messages, refer to the descriptions of the tests later in this document.

1.5 POD Mode

POD mode is an interactive mode that uses the command line interpreter that is in the PROM. You can use POD mode on a crashed system to:

- examine the contents of the CPU registers
- examine the contents of the support chip registers
- examine the contents of memory
- enable or disable Node board components (for example, CPUs and memory banks)
- manually run several of the IP27 power-on diagnostics

1.5.1 Entering POD Mode

The system enters POD mode under any of the following conditions:

- A power-on diagnostic fails.
- An unexpected interrupt occurs while the IP27 PROM code or BaseIO PROM code is executing.
- The boot sequence detects that the boot stop point debug DIP switches (switches 4 and 5) are set respectively to Off and On, On and Off, or On and On.

  Refer to Table 1-4 in Section 1.2, “Modifying Power-On Diagnostics Execution,” for more information about the debug DIP switches. The boot sequence stops and enters POD mode at the specified time.
- You issue a nonmaskable interrupt (NMI) at the PROM level.
- You issue two NMIs at the kernel level. The first takes the system to the PROM level; the second takes the system to the POD prompt.
- You type the *pod* command at the BaseIO command (PROM level) prompt (for example, >>**pod**).

1.5.2 Dirty Exclusive, Uncached, and Cached Modes

POD mode can run in three modes: dirty exclusive (Dex) mode, uncached (Unc) mode, or cached (Cac) mode.

1.5.2.1 Dirty Exclusive Mode

In dirty exclusive (Dex) mode, the power-on diagnostics require the fewest system resources to run. Dex mode does not require memory; it accesses the program code directly from the PROM and uses the data cache in the R10000 processors as the memory for its stack. Dex mode does not use the secondary cache. Nonmaskable interrupts and uncaught exceptions typically return you to the Dex mode prompt.

Dex mode is very slow because PROM instruction fetches are slow. Avoid performing long memory tests or flashing remote PROMs from Dex mode.

When you load or store data in memory or run memory tests in Dex mode, ensure that you do not access cached memory addresses.

1.5.2.2 Uncached Mode

In uncached (Unc) mode, the power-on diagnostics place the program code, data, and stack into main memory. This mode is similar to Cac mode, but Unc mode runs slower; however, program execution is quicker in Unc mode than in Dex mode.

1.5.2.3 Cached Mode

In cached (Cac) mode, the power-on diagnostics place the program code, data, and stack into the L2 cache. Cached mode is available only after memory has been probed and configured. POD execution is much quicker in Cac mode than in Dex mode or Unc mode because it runs from the L2 cache.

1.5.3 POD Mode Prompt

The POD mode prompt displays information about the current state of POD mode. An example POD mode prompt is:

```
3A 001: POD MSC Dex MDerr>
```

Table 1-6 uses this example to describe the components of the POD prompt.

Table 1-6 Components of the POD Mode Prompt

Prompt Component	Description
3A	The number in this field indicates the slot location of the Node board that contains the CPU (1 to 4) that is displaying the POD mode prompt. The letter in this field indicates which CPU on the Node board (A or B) is displaying the POD mode prompt.
001	This field displays the NASID value for the current Node, which is the Node ID.
POD	This field is always POD. It indicates that you are using POD mode.
MSC	This field indicates the UART on which the POD prompt is displayed. Possible values with matching descriptions are: MSC--The MSC alternate console port is displaying the POD prompt. Junk--A junk UART is displaying the POD prompt. IOC3--An IOC3 UART is displaying the POD prompt. Talk--A netuart is displaying the POD prompt.
Dex	This field indicates the mode in which POD mode is running. Possible values are Dex, Cac, and Unc.
MDerr	This field occasionally indicates errors or alternate operating modes. Possible values with matching descriptions are: MDerr--MD errors are pending. (Enter the *error* command to view the errors. Enter the *clear* command to clear the errors.) AltRegs--POD mode is using the kernel register set instead of the PROM register set.

1.5.4 More Information About POD Mode

Refer to the *IP27PROM Technical Reference Manual*, publication number 108-0170-001, for more information about POD mode and the commands that you can use in POD mode.

Chapter 2

Running the Power-On Diagnostics

This chapter describes how you can run the power-on diagnostics automatically as part of the boot sequence and manually from the POD prompt or the BaseIO command prompt.

2.1 Running Power-On Diagnostics During the Boot Sequence

The boot sequence occurs whenever you power up or reset the system. It uses code from the IP27 PROM and, optionally, the BaseIO PROM. The appropriate power-on diagnostics run automatically as the boot sequence discovers each hardware component in the system.

2.1.1 Boot Sequence in IP27 PROM Code

The code in the IP27 PROM performs the following actions.

Note: Steps 1 through 6 do not produce MSC or console output. Only the Node board LEDs are available to provide output during this phase of the boot process.

1. Initialize the CPU(s).
   - Set up the R10000 status registers.
   - Clear the register files.
   - Activate the Node board LEDs.
2. Test the CPU caches.
   - Test the instruction cache.
   - Test the data cache.
3. Set up Dex mode.
   - Start using the CPU primary data cache as memory.
   - Initialize the stack.
   - Execute the boot procedure.

4. Disable CPU A and/or CPU B.
   - Disable the CPUs indicated by the PROM DisableA and DisableB environment variables.

     Note: You can use Virtual Debug Switch 9 to override the disabling of the CPUs.
   - Keep the CPUs separated from the HUBs by turning off PI_CPU_ENABLE.
5. Arbitrate the local master CPU (one per Node board).
   - Check for the presence of another CPU on the Node board.
   - Time out if the other CPU does not respond.
   - Disable the other CPU if it does not respond.
6. Read the debug DIP switch settings from the module system controller (MSC).
7. Determine the initial console device and initialize it.

   Normally the initial console port is the alternate console port (ACP) on the MSC.

   Once the initial console device is initialized, the boot sequence can print detailed boot status information to the console device.
8. Record error information that may indicate the cause of a prior crash.
   - Save the state of HUB error registers in the cache.
   - Clear the state of HUB error registers.
   - Enable SYSAD error checking.
9. Initialize I/O.
   - Initialize the I/O interface (II) section of the HUB.
   - Initialize the Bridge.
   - Initialize the PCI bus.
10. Determine whether the system contains a BaseIO card with a console port. If it does, switch the console to the terminal that is connected to the BaseIO console port.
11. Display the PROM boot banner on the console.
    - Print the IP27 PROM version number and size.
    - Print the chip revisions.
    - Print the slot ID values and any CPUs that are in the system.
    - Print other miscellaneous information.
12. Configure local memory.
    - Initialize the DIMM control registers.
    - Probe for the amount and type (standard or premium) of memory.
    - Program the HUB memory configuration registers.
13. Display the Debug Switch settings that are set to other than the default; this information appears on the console.
14. Determine the serial number of the Node and send it to the other Nodes.

15. Configure the HUB clock speed.
16. Run the HUB self-test (if heavy mode or manufacturing mode is selected).
    - Run the built-in self test (BIST), which automatically reboots the system.
    - After the reboot, stop with a failure LED value if the test failed.
17. Perform basic memory tests.
    - Ensure that address 0 of each bank is accessible.
    - Disable any banks that are not accessible.
    - If bank 0 is bad, swap it with a good bank (use the SwapBank environment variable, which is set to 1 by default, to select the good bank).
18. Download the PROM to memory.
    - Perform a memory test on the lower portion of local memory by using a small portion of PROM data.
    - If the memory test passes, download a copy of the entire PROM to memory.
    - Verify the download checksum to ensure that the PROM data was downloaded successfully.
19. Transfer the program counter to uncached memory.

    Execution from memory is significantly faster than running out of the PROM. (The boot process still does not require the secondary cache.)
20. Move crucial data structures from the cache into uncached memory and perform a hardware inventory.
21. Test and invalidate the secondary cache.
22. Transfer the program stack to uncached memory.
    - Call UALIAS mode.
    - Discard the data (from Dex mode) in the primary data cache.
23. Transfer the program counter and stack to cached memory.

    Execution of the PROM code now runs at top speed.
24. Initialize the first 32 MB of bank 0 memory for use by the PROM.
25. Initialize permanent low-memory system data structures.
    - Initialize KLDIR (the indirection table for accessing all other low-memory structures).
    - Initialize NMI (the nonmaskable interrupt vector area).
    - Initialize KLCONFIG (the system topology and configuration information).
26. Run serial-port diagnostics on the local BaseIO.
27. Discover the CrayLink Interconnect topology.
    - Perform a depth-first search using vector routing operations.
    - Build the PROM data structure (promcfg) in low cached memory.

28. Verify that all PROMs are running the same firmware version.
    - Print the message `Barrier Sync Warning` on the console if there is a PROM mismatch.
29. Select the global master CPU. (Use a protocol that uses vector routing operations and allows one global master per partition.)
30. Have the global master CPU configure the CrayLink Interconnect.
    - Determine the Node ID for each Node (not yet assigned).
    - Program the CrayLink routing tables in all of the HUBs and Routers.
    - Tell each Node what its Node ID will be.
31. Switch all Nodes back to uncached memory, which is required to change Node IDs.
    - Flush the caches.
    - Closely synchronize all Nodes.
    - Stop bus activity with tight loops that are internal to the R10000 processor.
32. Change the Node IDs (NASIDs).
    - Update data structures that were assuming the NASID was zero.
    - Reinitialize coherence directories for the first 32 MB of memory.
33. Switch back to cached memory in the new NASID space.
34. Test and initialize the rest of local memory (above 32 MB).
35. Check partitioning information.
    - Assign a partition master CPU to each partition.
    - Reassign NASIDs, if necessary.
    - Create “partition fences” to isolate partition memory space.
36. Initialize any headless Nodes (Nodes without functional CPUs).
    - Disable the CPUs.
    - Read the NIC.
    - Probe, configure, and initialize memory.
    - Initialize the low-memory data structures.
    - Initialize the I/O.
37. If debug switch 13 is ON or if the system is booting because of a reset and not because it was just powered on, display the error state information from the error registers.
    - Decode the HUB error registers and display detailed information about their contents.

38. Transfer control to the BaseIO PROM code.

    Note: This step may be inhibited by the debug DIP switch settings. Refer to Section 1.2, "Modifying Power-On Diagnostics Execution," for more information about the debug DIP switches.

    - Retrieve a copy of the BaseIO PROM image from the master BaseIO board. If the master BaseIO board is not available, retrieve the copy of the BaseIO PROM image that is stored in the IP27 PROM.
    - Decompress the BaseIO PROM image and load it into memory.
    - Continue code execution at the entry point of the BaseIO PROM code.

2.1.2 Boot Sequence in BaseIO PROM Code

When code execution reaches the entry point of the BaseIO PROM code, the code from the BaseIO PROM performs the following actions.

1. Print the BaseIO PROM banner.
2. Initialize the global data structures.
3. Number the Nodes in the system, starting with 0.
4. Synchronize the Node clocks.
    - Print a message to indicate any midplane clock speed mismatches.
5. Initialize and test the BaseIO hardware.
    - Initialize and test the SCSI buses.
    - Initialize and test the console port.
    - Initialize and test the Ethernet port.
    - Initialize and test the graphics pipe, graphics console, and audio (if applicable).
6. Compare the current system inventory with the stored system inventory.
7. Print a summary of the system configuration and diagnostic results.
8. Fetch the environment variables and boot the IRIX operating system. When the system is ready to boot IRIX, the BaseIO PROM code displays the following menu:

```
System Maintenance Menu

1) Start System
2) Install System Software
3) Run Diagnostics
4) Recover System
5) Enter Command Monitor

Option?
```

The selection that you make from the System Maintenance Menu determines what happens next in the boot process. Table 2-1 describes each of the menu options.

Table 2-1 System Maintenance Menu Options

Option	Description
`1) Start System`	This option causes the system to boot IRIX in the default way.
`2) Install System Software`	This option is used when system software needs to be installed or upgraded. The PROM first attempts to find a tape drive on the system and if one is found, it prompts the user to insert the installation tape in it. If a tape device is not found, then installation is expected to take place through an Ethernet connection. In this case, the PROM prompts the user for the name of the system that will be used as the server.
`3) Run Diagnostics`	This option is not implemented.
`4) Recover System`	This option can be used to perform special system administration tasks such as restoring a system disk from backup tapes. It follows a sequence similar to installing system software, but instead of starting the installation program, it invokes an interactive restoration tool.
`5) Enter Command Monitor`	Additional functions can be performed from an interactive command monitor. This option puts the PROM into a manual mode of operation. It displays the BaseIO command prompt, which is the >> prompt.

2.2 Invoking the Power-On Diagnostics Manually

You can also run several of the POD tests manually from the POD prompt or BaseIO command prompt.

2.2.1 Power-On Diagnostics From the POD Prompt

Table 2-2 lists the power-on diagnostics that you can run from the POD prompt and indicates the command you can use to start the tests. For more information about each of the tests (including descriptions of the command line options), refer to the page shown in the "Page" column.

Table 2-2 Power-On Diagnostics That Run From the POD Prompt

Diagnostic Test	Command to Run the Test	Page
Backdoor directory space test	*dirtest START LEN*	3-10
Memory test	*memtest START LEN*	3-12
Local link connection test	*chklink*	3-14
HUB send data error test	*hubsde*	3-15
Router send data error test	*rtrsde*	3-17
XBOW test	*dgxbow* [mn | mh | mm] [*nNODE*]	3-20
Bridge test	*dgbrdg* [mn | mh | mm] [*nNODE*] [*sSLOT*]	3-22
PCICONFIG test	*dgconf* [mn | mh | mm] [*nNODE*] [*sSLOT*]	3-24
PCI bus test	*dgpci* [mn | mh | mm] [*nNODE*] [*sSLOT*] [p*PCINUM*]	3-25
Serial programmed I/O test	*dgspio* [mn | mh | mm | mx] [*nNODE*] [s*SLOT*] [p*PCINUM*]	3-27
Serial DMA test	*dgsdma* [mn | mh | mm | mx] [*nNODE*] [s*SLOT*] [p*PCINUM*]	3-29

2.2.2 Power-On Diagnostics From the BaseIO Command Prompt

The power-on diagnostics that reside in the BaseIO PROM do not run from the POD prompt; you must run these diagnostics from the BaseIO command (PROM monitor) prompt, which is >>. You can access the >> prompt by selecting the `5. Enter Command Monitor` option of the System Maintenance Menu.

Table 2-3 lists the power-on diagnostics that run from the BaseIO command prompt. For more information about each of the tests, refer to the page shown in the "Page" column.

Table 2-3 Power-On Diagnostics That Run From the BaseIO Command Prompt

Diagnostic Test	Command to Run the Test	Page
Comprehensive SCSI test	*diag_scsi*	n/a
SCSI SSRAM test	*diag_scsi* -t ram	4-3
SCSI SSRAM DMA test	*diag_scsi* -t dma	4-4
SCSI self-test	*diag_scsi* -t controller	4-5
Comprehensive Ethernet test	*diag_enet*	4-8
Ethernet SSRAM test	*diag_enet* -t ssram	4-11
PHY register test	*diag_enet* -t phyreg	4-14
TX clock test	*diag_enet* -t txclk	4-13
NIC test	*diag_enet* -t nic	4-16
IOC3 internal loopback test	*diag_enet* -t ioc3_loop	4-17
PHY chip internal loopback test	*diag_enet* -t phy_loop	4-20
Twister chip internal loopback test	*diag_enet* -t tw_loop	4-23
External loopback DMA test	*diag_enet* -t ext_loop	4-24
External loopback DMA test (10 Mb/s)	*diag_enet* -t ext_loop_10	4-27
External loopback DMA test (100 Mb/s)	*diag_enet* -t ext_loop_100	4-29
XTALK stress test	*diag_enet* -t xtalk	4-31

Chapter 3

IP27 PROM Tests

This Chapter describes the power-on diagnostic tests that are included in the IP27 PROM.

Each test description includes the following information:

- A short description of the test
- Any prerequisites that you should perform before you run the test (if you can manually run the test)
- Information about how to run the test (if it runs during the boot sequence and if you can run it manually)
- A description of the actions that the test performs
- Failure information for the test (including the failure indication and the failing hardware components)

All IP27 PROM tests print out pass and fail messages. The tests send the messages to the system controller. The system controller sends the messages to the console that is connected to the alternate console port (ACP). Failure messages use the following format:

`RSLT` *test_name*`FAIL  diag_rc =` *xxx* *failure_description*

Example:

```
RSLT hub_intrpt_diag    FAIL diag_rc = 157    hub interrupt test failed.
```

Refer to Appendix B, “diag_rc Values,” for descriptions of all `diag_rc` values that the power-on diagnostics use.

3.1 R10000 Processor Tests

3.1.1 Register Test

The register test checks the COP1 register portion of the R10000 processor.

3.1.1.1 How to Run This Test

This test runs automatically during the boot process. You cannot run this test from the POD prompt.

3.1.1.2 Test Description

This test checks for stuck bits in the processor registers. This test uses the following algorithm:

1. Write 0xa5a5 and 0x0000 patterns to the BSR registers.
2. Read the data back from the BSR registers.
3. Compare the data that was read back with the data that was written.

3.1.1.3 Failure Information

If this test fails, the value FLED_CP1 (0x81) flashes on the LEDs of the failing CPU (refer to Figure 3-1).

Figure 3-1 LED Pattern for a Register Test Failure (0x81)

If this test fails, the failing FRU is the Node board; the cause is most likely the R10000 chip.

3.1.2 Primary Cache Tests

The primary instruction cache test writes to and reads from the primary instruction cache. The primary data cache test writes to and reads from the primary data cache.

3.1.2.1 How to Run These Tests

These tests run automatically during the boot process. You cannot run these tests from the POD prompt.

3.1.2.2 Test Descriptions

The primary instruction cache test uses the following algorithm:

1. Write 0x5555 and 0x0000 data patterns to the primary instruction cache.
2. Read the data back from the primary instruction cache.
3. Compare the data that was read with the data that was written.

The primary data cache test uses the following algorithm:

1. Write 0x5555 and 0x0000 data patterns to the primary data cache.
2. Read the data back from the primary data cache.
3. Compare the data that was read with the data that was written.

3.1.2.3 Failure Information

If the primary instruction cache test fails, the value FLED_ICACHE (0x83) illuminates on the LEDs of the failing CPU (refer to Figure 3-2).

Figure 3-2 LED Pattern for a Primary Instruction Cache Test Failure (0x83)

If primary data cache test fails, the value FLED_DCACHE (0x84) illuminates on the LEDs of the failing CPU (refer to Figure 3-3).

Figure 3-3 LED Pattern for a Primary Data Cache Test Failure (0x84)

If either of these tests fails, the failing FRU is the Node board; the cause of the failure is most likely the R10000 chip.

3.1.3 Interrupt Test

The interrupt test verifies that the processors can correctly receive interrupts.

3.1.3.1 How to Run This Test

This test runs automatically during the boot process. You cannot run this test from the POD prompt.

3.1.3.2 Test Description

This test checks the hardware in the R10000 processors that is dedicated to handling any interrupts or exceptions. It also tests the HUB chip interrupt registers.

3.1.3.3 Failure Information

If this test fails, the value FLED_HUBINTS (0x90) illuminates on the LEDs of the failing CPU (refer to Figure 3-4), and the test prints a `RSLT hub_int_diag FAIL` message on the console that is connected to the ACP.

Figure 3-4 LED Pattern for an Interrupt Test Failure (0x90)

If this test fails for both CPUs on a Node board, the failing FRU is the Node board; the cause is most likely the HUB chip on the Node board. If this test fails for only one of the CPUs, the failing FRU is the Node board; the cause of the failure is most likely the CPU that failed; however, it is still possible that the Node board caused the failure.

3.2 Secondary Cache Test

The secondary cache test walks data patterns across the secondary data cache to detect stuck bits.

3.2.1 How to Run These Tests

These tests run automatically during the boot process. You cannot run these tests from the POD prompt.

3.2.2 Test Descriptions

These tests check for stuck bits in the secondary cache SRAMs. These tests use the following algorithm:

1. Write data patterns (0xaaaa and 0x5555 in normal mode; 0xaaaa, 0x5555, 0x0000, and 0xffff in heavy and manufacturing modes) to the secondary cache SRAMs that are located on the HIMM board.
2. Read the data back from the secondary cache SRAMs.
3. Compare the data that was read with the data that was written.

3.2.3 Failure Information

If these tests fail, the value FLED_SCACHE (0x85) illuminates on the LEDs of the failing CPU (refer to Figure 3-5), and the tests print a `RSLT cache_test_s FAIL` message on the console that is connected to the ACP.

Figure 3-5 LED Pattern for a Secondary Cache Test Failure (0x85)

If these tests fail, the failing FRU is the Node board; the failure is caused by a soft or hard failure in an SRAM chip, a bad interconnect on the HIMM board between the R10000 processor and the SRAM, or a bad R10000 processor. If a failure occurs, these tests use the syndrome information to decode the bad bit; this process enables the test to identify which of the five secondary cache chips has the bad bit.

3.3 HUB Tests

3.3.1 GClk Logic Test

The GClk logic test checks the functionality of the HUB global clock logic.

3.3.1.1 How to Run This Test

This test runs automatically during the boot process. You cannot run this test from the POD prompt.

3.3.1.2 Test Description

This test checks the GCLK logic that is internal to the HUB chip.

3.3.1.3 Failure Information

If a failure occurs, this test prints out a `RSLT rt_clock_test FAIL` failure message on the console connected to the ACP.

If this test fails, the failing FRU is the Node board.

3.3.2 BTE Test

The BTE test checks the block transfer engine (BTE) logic in the HUB.

3.3.2.1 How to Run This Test

This test runs automatically during the boot process. You cannot run this test from the POD prompt.

3.3.2.2 Test Description

This test uses the following algorithm:

1. Program both BTEs on the HUB chip.
2. Have both BTEs on the HUB chip perform local transfer operations.
3. Compare the results.

In all modes, this test performs a simple block copy transfer and then checks all basic modes of the BTE logic, including: basic copy, termination of a copy in progress, zero-fill mode, poison mode, interrupt mode, and notification mode.

3.3.2.3 Failure Information

If this test fails, the test prints a `RSLT hub_bte_diag FAIL` message on the console that is connected to the ACP. The boot process continues even if this test detects an error.

If this test fails due to memory problems, the failing FRU is the failing DIMM. If the failure is not a memory problem, the failure is caused by one of the following conditions: the HUB has an internal logic failure, there is a multibit error in memory, or some of the HUB-to-memory interconnects are broken. If the memory tests pass before you run this test, then the most likely cause of failure of this test is a bad HUB chip.

3.4 PROM Checksum Test

The PROM checksum test verifies the copy of PROM code in memory. If it fails when it is verifying the copy in memory, this test verifies the copy of the PROM code that is stored in the PROM.

3.4.1 How to Run This Test

This test runs automatically during the boot process. You cannot run this test from the POD prompt.

3.4.2 Test Description

This test uses the following algorithm:

1. Add up the number of words in the PROM code.
2. Compare the result with an expected value.

3.4.3 Failure Information

If this test fails, it flashes FLED_DOWNLOAD (0x9b) on the LEDs (refer to Figure 3-6) and prints a `RSLT prom_checksum FAIL` message on the console that is connected to the ACP.

Figure 3-6 LED Pattern for PROM Checksum Test Failure (0x9b)

If this test fails, the failing FRU is either the Node board or a memory DIMM. Possible causes are a bad PROM, a bad copy of the software image in the PROM, bad bit(s) in memory, a transient error that occurred when the PROM image was copied from the PROM into memory, or a transient error in the R10000 processor that occurred when the processor computed the checksum. The most likely cause is one or more bad bits in memory.

3.5 Memory/Directory Tests

3.5.1 Memory Configuration Test

The memory configuration test verifies that the system's memory configuration is valid.

3.5.1.1 How to Run This Test

This test runs automatically during the boot process. You cannot run this test from the POD prompt.

3.5.1.2 Test Description

This test checks the memory configuration for valid memory combinations, valid memory sizes, etc.

3.5.1.3 Failure Information

If this test fails, it illuminates FLED_NOMEM (0x98) or FLED_BADMEM (0xa9) on the LEDs (refer to Figure 3-7 and Figure 3-8).

Figure 3-7 LED Pattern for Memory Configuration Test Failure (0x98)

Figure 3-8 LED Pattern for Memory Configuration Test Failure (0xa9)

The test also prints one of the following messages on the console that is connected to the ACP.

```
No Memory Found

*** Mixed standard and premium memory
*** Treating all as standard.

nasid node_number: disabling bank bank_number
```

If this test fails, the failing FRU is the DIMM that the test indicates**.** A failure of this test most likely indicates a bad memory configuration. It could also indicate a hardware failure in a HUB-to-memory interconnect. The system will not boot if this test fails.

Note: This test always prints out the following message for each Node in the system: `Total memory configured:` *number* `MB`.

3.5.2 Backdoor Directory Space Test

The backdoor directory space test checks the memory that contains the backdoor directory space.

3.5.2.1 Prerequisites to This Test

You should verify that the R10000 processor, the system interface bus, and the HUB registers are functional before you run this test.

3.5.2.2 How to Run This Test

This test runs automatically during the boot process, or you use the *dirtest* command at the POD prompt to run this test. The *dirtest* command tests a range of directory memory. This command uses the following syntax:

dirtest START LEN

The *START* variable specifies a memory address that this command uses by masking off the lower 40 bits and translating the value into a corresponding directory address. The *LEN* command specifies the size of the area to test (in bytes). *START* and *LEN* must be multiples of 4,096 (0x1000). Refer to the *IP27PROM Technical Reference Manual*, publication number 108-0170-001, for more information about the *START* and *LEN* variables.

Note: Turn off ECC checking (enter **`ecc off`**) before you test directory memory.

3.5.2.3 Test Description

The directory memory test uses the following algorithm:

1. Clear the error registers.

2. Test the store and load functions. For each doubleword address in the memory range that is being tested, perform the following actions:

   - Perform a store operation with a data pattern that contains all 0x5 values, immediately perform load operations to get the data back, and compare the loaded data with the stored data.
   - Perform a store operation with a data pattern that contains all 0xa values, immediately perform load operations to get the data back, and compare the loaded data with the stored data.

   This test may find problems with RAM cells that take too long to change states.

3. Test memory with alternating 0x5 and 0xa data patterns. (Write the data patterns, read the data back, and compare the data that was read back with the data that was written.)
4. Test memory with alternating 0xa and 0x5 data patterns. (Write the data patterns, read the data back, and compare the data that was read back with the data that was written.)
5. Test memory with alternating 0xc3c3c3c3c3c3c3c3 and 0x3c3c3c3c3c3c3c3c doubleword patterns. (Write the data patterns, read the data back, and compare the data that was read back with the data that was written.)
6. Test memory with alternating 0x3c3c3c3c3c3c3c3c and 0xc3c3c3c3c3c3c3c3 doubleword patterns. (Write the data patterns, read the data back, and compare the data that was read back with the data that was written.)
7. Test memory with random data patterns. (Write a random data pattern, read the data back, and compare the data that was read back with the data that was written.)
8. Test memory with address data patterns. (Write each doubleword with the address of the location being written, read the data back, and compare the data that was read back with the data that was written.)

Memory is left filled with a data pattern of incrementing doublewords. The *maxerr* command specifies the number of errors that the test should display for each phase before it stops the current phase and continues with the next phase of the test.

3.5.2.4 Failure Information

If this test fails, it prints one of the following messages on the console that is connected to the ACP.

```
-----MEMORY FAILURE (caught exception node slot slot_number) -----
Address address
CAUSE cause_register_value

-----MEMORY FAILURE (miscompare node slot slot_number) -----
Address address
CAUSE cause_register_value
```

If this test fails, the failing FRU is the DIMM that the test specifies. Failure of this test indicates one of the following hardware failures: a directory DIMM had a soft or hard failure (most likely), a HUB-to-directory memory interconnect failed, or the HUB failed (least likely).

3.5.3 Memory Test

The memory test checks the memory address and data lines.

3.5.3.1 Prerequisites to This Test

You should verify that the R10000 processors, the system interface bus, and the HUB registers are functional before you run this test.

3.5.3.2 How to Run This Test

This test runs automatically during the boot process, or use the *memtest* command from the POD prompt to run this test. The *memtest* command tests a range of memory. This command uses the following syntax:

memtest START LEN

The *START* variable specifies the starting address of the memory range; this address can be in cached or uncached memory. The *LEN* command specifies the size of the range of memory to test (in bytes). *START* and *LEN* must be multiples of 4,096 (0x1000). Refer to the *IP27PROM Technical Reference Manual*, publication number 108-0170-001, for more information about the *START* and *LEN* variables.

Examples:

- **memtest u:32m 1m**
  memtest 0x9600000002000000 0x100000

  These examples are equivalent. These examples test 1 MB of uncached memory space, starting at an offset of 32 MB.

- **memtest c:32m 32m**

  This example tests the second 32 MB of cached memory.

- **memtest u:b:3 0 64m**

  This example tests 64 MB of uncached memory in bank 3.

- **memtest n:7 b:3 c: 1m 63m**
  memtest (n:7 b:3 c: 1m) 63m
  memtest 0xa800000760100000 0x3f00000

  These examples are equivalent. These examples test 63 MB of cached memory on Node (NASID) 7, beginning at the 1 MB position of bank 3.

3.5.3.3 Test Description

The memory test uses the following algorithm:

1. Clear the error registers.
2. Test the store and load functions. For each doubleword address in the memory range that is being tested, perform the following actions:
   - Perform a store operation with a data pattern that contains all 0x5 values, immediately perform load operations to get the data back, and compare the loaded data with the stored data.
   - Perform a store operation with a data pattern that contains all 0xa values, immediately perform load operations to get the data back, and compare the loaded data with the stored data.

   This test may find problems with RAM cells that take too long to change states.
3. Test memory with alternating 0x5 and 0xa data patterns. (Write the data patterns, read the data back, and compare the data that was read back with the data that was written.)
4. Test memory with alternating 0xa and 0x5 data patterns. (Write the data patterns, read the data back, and compare the data that was read back with the data that was written.)
5. Test memory with alternating 0xc3c3c3c3c3c3c3c3 and 0x3c3c3c3c3c3c3c3c doubleword patterns. (Write the data patterns, read the data back, and compare the data that was read back with the data that was written.)
6. Test memory with alternating 0x3c3c3c3c3c3c3c3c and 0xc3c3c3c3c3c3c3c3 doubleword patterns. (Write the data patterns, read the data back, and compare the data that was read back with the data that was written.)
7. Test memory with random data patterns. (Write a random data pattern, read the data back, and compare the data that was read back with the data that was written.)
8. Test memory with address data patterns. (Write each doubleword with the address of the location being written, read the data back, and compare the data that was read back with the data that was written.)

Memory is left filled with a data pattern of incrementing doublewords. The *maxerr* command specifies the number of errors that the test should display for each phase before it stops the current phase and continues with the next phase of the test.

3.5.3.4 Failure Information

If this test fails, it prints one of the following messages on the console that is connected to the ACP.

```
-----MEMORY FAILURE (caught exception node slot slot_number) -----
Address address
CAUSE cause_register_value

-----MEMORY FAILURE (miscompare node slot slot_number) -----
Address address
Actual actual_value
Expected expected_value
Difference difference_value
```

If this test fails, the failing FRU is the DIMM that the test specifies. A failure in this test indicates a soft or hard failure in a memory chip (most likely), a failing HUB chip-to-memory interconnect, or a bad HUB chip (least likely).

3.6 CrayLink Tests

3.6.1 Local Link Connection Test

The local link connection test checks the connection between the local HUB and an attached HUB or Router.

3.6.1.1 Prerequisites to This Test

You should verify that the R10000 processor and the HUB are functional before you run this test.

3.6.1.2 How to Run This Test

This test runs automatically during the boot process, or you can run it from the POD prompt. The following example shows how you can run this test from the POD prompt:

```
0A 000: POD Dex> chklink
```

3.6.1.3 Test Description

This test uses the following algorithm:

1. Check the ni_port error register.
2. Perform a series of vector write and read operations; verify the results.

Note: This test runs before the boot process performs network discovery operations; this process isolates any errors to the local Node.

3.6.1.4 Failure Information

If this test fails, the test displays a `RSLT chk_local_link FAIL` message on the console that is connected to the ACP; the message includes complete link isolation information.

If this test fails, the failing FRU is the midplane (most likely), Node board, or Router (least likely). The failure is caused by a failing local HUB, a failing attached HUB or Router, a failing local HUB-to-attached HUB interconnect, or a failing local HUB-to-attached Router interconnect.

3.6.2 HUB Send-Data Error Test

The HUB send-data error test verifies that the HUB or Router attached to the local HUB can detect link errors.

3.6.2.1 Prerequisites to This Test

You should ensure that the R10000 processors and the HUB are functional before you run this test.

3.6.2.2 How to Run This Test

This test runs automatically during the boot process, or you can run it from the POD prompt. The following example shows how you can run this test from the POD prompt:

```
0A 000: POD Dex> hubsde
```

3.6.2.3 Test Description

This test uses the following algorithm:

1. Check the remote port error register.
2. Send an error in a packet across the link.
3. Verify that the hardware detected and logged the error.

This test runs the same in normal, heavy, and manufacturing modes.

Note: This test runs before the boot process performs network discovery operations; this process isolates any errors to the local Node.

3.6.2.4 Failure Information

If this test fails, the test displays a `RSLT hub_send_data_error FAIL` message on the console that is connected to the ACP; the message includes complete link isolation information.

If this test fails, the failing FRU is the Router, Node board, or midplane. A failure in this test indicates a bad local HUB, a bad attached HUB or Router, a bad local HUB-to-attached HUB interconnect, or a bad HUB-to-attached Router interconnect.

3.6.3 HUB Error Detection Test

The HUB error detection test is similar to the HUB send-data error test, but the HUB error detection test is called recursively during the discovery process.

3.6.3.1 How to Run This Test

This test runs automatically during the boot process. You cannot run this test from the POD prompt.

3.6.3.2 Test Description

This test uses the following algorithm:

1. Check the remote port error register.
2. Send an error in a packet across the link.
3. Verify that the hardware detected and logged the error.

This test runs only in heavy and manufacturing modes. The algorithm is performed recursively during the boot process.

3.6.3.3 Failure Information

If this test fails, the test displays a `RSLT hub_error_detection_error FAIL` message on the console that is connected to the ACP; the message includes complete link isolation information.

The failing FRU is the Node board (most likely), Router, or midplane (least likely). A failure in this test indicates a bad Local HUB, a bad attached HUB or Router, or a bad local HUB-to-attached HUB or Router interconnect. Because the boot tests that run before this test eliminate other causes of failure, the most likely cause of failure is a failing attached link or a failing attached board or chip.

3.6.4 Router Send-Data Error Test

The Router send-data error test verifies that the local HUB can detect link errors.

3.6.4.1 Prerequisites to This Test

You should ensure that the R10000 processors and the HUB are functional before you run this test.

3.6.4.2 How to Run This Test

This test runs automatically during the boot process, or you can run it from the POD prompt. The following example shows how you can run this test from the POD prompt:

```
0A 000: POD Dex> rtrsde
```

3.6.4.3 Test Description

This test uses the following algorithm:

1. Check the local port error register.
2. Send an error in a packet across the link.
3. Verify that the hardware detected and logged the error.

This test runs identically in normal, heavy, and manufacturing modes.

Note: This test runs before the boot process performs network discovery operations; this process isolates any errors to the local Node.

3.6.4.4 Failure Information

If this test detects an error, the test displays a `RSLT rtr_send_data_error FAIL` message on the console that is connected to the ACP; the message includes complete link isolation information.

The failing FRU is the Node board (most likely), Router, or midplane/cable (least likely). A failure in this test indicates a bad local HUB, a bad attached HUB or Router, or a bad local HUB-to-attached HUB or Router interconnect. Because the boot tests that run before this test eliminate other causes of failure, the most likely cause of failure is a failing attached link or a failing attached board or chip.

3.6.5 Router Error Detection Test

The Router error detection test is similar to the Router send-data error test except that you cannot run it from the POD prompt and that it runs recursively during the discovery process.

3.6.5.1 How to Run This Test

This test runs automatically during the boot process. You cannot manually run this test from the POD prompt.

3.6.5.2 Test Description

This test uses the following algorithm:

1. Check the local port error register.
2. Send an error in a packet across the link.
3. Verify that hardware detects and logs the error.

This test runs only in heavy and manufacturing modes.

3.6.5.3 Failure Information

If this test fails, the test displays a `RSLT test_rtr_error_detection FAIL` message on the console that is connected to the ACP; the message includes complete link isolation information. Because the boot tests that run before this test eliminate other causes of failure, the most likely cause of failure is a failing attached link or a failing attached board or chip.

The failing FRU is the Router (most likely), Node board, or midplane/cable (least likely). A failure in this test indicates a bad local HUB, a bad attached HUB or Router, or a bad local HUB-to-attached HUB or Router interconnect.

3.6.6 Get Chipid Test

The get chipid test runs during the discovery process to obtain the ID of a HUB or Router chip that is attached across an LLP link.

3.6.6.1 How to Run This Test

This test runs automatically during the boot process. You cannot manually run this test from the POD prompt.

3.6.6.2 Test Description

The get chipid test runs during the discovery process to obtain the ID of a HUB or Router chip that is attached across an LLP link.

3.6.6.3 Failure Information

If this test fails, it displays a `RSLT get_chipid FAIL` failure message on the console that is connected to the ACP; the message includes complete isolation information for the failing links.

The failing FRU is the Node board or Router. A failure in this test indicates a bad source chip, a bad destination chip, or a bad link between these chips. The most likely cause of a failure is a bad or misconnected link.

3.7 XBOW Test

The XBOW test checks for valid reset values in the XBOW registers and tests the XBOW interrupt hardware.

Note: If a HUB-link error occurs during the boot sequence, this test may not run because the link was not detected.

3.7.1 How to Run This Test

This test runs automatically during the boot process, or you can run it from the POD prompt. To run this test from the POD prompt, enter the following command:

dgxbow [mn | mh | mm] [n*NODE*]

The **m** option specifies the mode in which the test should run. Enter **mn** to run the test in normal mode, enter **mh** to run the test in heavy mode, and enter **mm** to run the test in manufacturing mode. (The default mode is normal mode.)

The **n** option specifies the Node ID (NASID value) of the Node on which the test should run. (The default Node is the Node from which you entered the command.)

For example, enter **dgxbow mh n1** to run the XBOW test in heavy mode on Node 1.

3.7.2 Test Description

This test uses the following algorithm:

1. Read the XBOW registers and verify that they contain valid reset values.
2. Create an illegal access by writing to a read-only register, which causes an XBOW interrupt.
3. Verify that the interrupt occurred on the XBOW and the HUB.

3.7.3 Failure Information

The XBOW test returns the following failure messages:

- `xbow_sanity: Bad xbow id reg value` *value*

  The XBOW test failed.

- `xbow_sanity: Bad Xbow id Exp:` *expected* `Recv:` *received*

  The XBOW test failed.

- `xbow_sanity: Reg write Read miscompare. Exp:` *expected*`:: Recv:` *received*

  This message indicates a failing XBOW or XBOW-to-HUB connection.

- `xbow_sanity: Register access violation not detected.`

  This message indicates a failing XBOW chip.

- `xbow_sanity: Status Register Clear on Read not working!!`

  The XBOW test failed.

- `xbow_sanity: Interrupt did not reach PI_INT_PEND0. exp:` *expected* `recv:` *received*

  This message indicates a failing XBOW, a failing XBOW-to-HUB connection, or a failing HUB.

- `RSLT xbow_sanity FAIL`

  This message follows any of the previous XBOW failure messages.

When the XBOW test fails, the failing FRU is either the midplane or a Node board.

3.8 Bridge Test

The Bridge test checks for valid reset values in the Bridge registers and tests the Bridge interrupt hardware.

3.8.1 How to Run This Test

This test runs automatically during the boot process, or you can run it from the POD prompt. To run this test from the POD prompt, enter the following command:

dgbrdg [mn | mh | mm] [n*NODE*] [s*SLOT*]

The **m** option specifies the mode in which the test should run. Enter **mn** to run the test in normal mode, enter **mh** to run the test in heavy mode, and enter **mm** to run the test in manufacturing mode. (The default mode is normal mode.)

The **n** option specifies the Node ID (NASID value) of the Node on which the test should run. (The default Node is the Node from which you entered the command.)

The **s** option specifies the IO slot number of the board with the Bridge that you want to test. (The default IO slot number is 1, which selects slot IO1.)

For example, enter **dgbrdg mh s4** to run this test in heavy mode on the Bridge ASIC on the I/O board in slot IO4 of the module that contains the Node from which you entered the command.

3.8.2 Test Description

This test uses the following algorithm:

1. Read the Bridge registers and verify that they contain valid reset values.
2. Force illegal accesses to the SSRAM, which causes Bridge interrupts because the SSRAM is not located on the BaseIO board.
3. Verify that the interrupts occurred on both the Bridge and the HUB.

3.8.3 Failure Information

The Bridge test returns the following failure messages:

- `bridge_sanity: Bad bridge id register value (LSB != 1)`

  This message indicates a failing Bridge ASIC. The failing FRU is the BaseIO board.

- `bridge_sanity: Bad bridge id Exp:` *expected* `Recv:` *received*

  This message indicates a failing Bridge ASIC. The failing FRU is the BaseIO board.

- `bridge_sanity: PCI bit set to 0 sn status reg.`

  The Bridge test failed. The failing FRU is the BaseIO board.

- `bridge_sanity: Timeout val reg exp:` *expected* `recv:` *received*`.`

  The Bridge test failed. The failing FRU is the BaseIO board.

- `bridge_sanity: Bridge control reg exp:` *expected* `recv:` *received*`.`

  The Bridge test failed. The failing FRU is the BaseIO board.

- `bridge_sanity: Bad reset val in dev reg exp:` *expected* `recv:` *received*`.`

  The Bridge test failed. The failing FRU is the BaseIO board.

- `bridge_sanity: Interrupt status register value non-zero:` *value*`.`

  The Bridge test failed. The failing FRU is the BaseIO board.

- `bridge_sanity: Bridge(wr): Exp:` *expected* `Recv:` *received*`.`

  The Bridge test failed. The failing FRU is the BaseIO board.

- `bridge_sanity: Interrupt status reg bit not set: exp:` *expected* `recv:` *received*`.`

  The Bridge test failed. The failing FRU is the BaseIO board.

- `bridge_sanity: Interrupt` *interrupt* `did not reach PI_INT_PEND0:  exp:` *expected* `recv:` *received*`.`

  The Bridge test failed. The failing FRU is the BaseIO board.

- `bridge_sanity: Interrupt` *interrupt* `did not occur.`

  This message indicates a failing Bridge, a failing XBOW chip, or a failing HUB.

- `RSLT bridge_sanity FAIL`

  This message follows any of the previous bridge test failure messages.

3.9 PCI Tests

3.9.1 PCICONFIG Test

The PCICONFIG test verifies the PCI configuration space on the BaseIO board.

3.9.1.1 How to Run This Test

This test runs automatically during the boot process, or you can run it from the POD prompt. To run this test from the POD prompt, enter the following command:

dgconf [mn | mh | mm] [n*NODE*] [s*SLOT*]

The **m** option specifies the mode in which the test should run. Enter **mn** to run the test in normal mode, enter **mh** to run the test in heavy mode, and enter **mm** to run the test in manufacturing mode. (The default mode is normal mode.)

The **n** option specifies the Node ID (NASID value) of the Node on which the test should run. (The default Node is the Node from which you entered the command.)

The **s** option specifies the IO slot number of the board that you want to test. (The default IO slot number is 1, which selects slot IO1.)

For example, enter **dgconf mh s7** to run the PCICONFIG test in heavy mode on the BaseIO board in slot IO7 of the module that contains the Node from which you entered the command.

3.9.1.2 Test Description

The PCICONFIG test verifies the PCI configuration space on the BaseIO board.

3.9.1.3 Failure Information

The PCICONFIG test returns the following failure messages:

- `io6config_space: IOC3 PCI_SCR reg reset val exp:` *expected* `recv:` *received*

  The PCICONFIG test failed; the failing FRU is the BaseIO board.

- `io6config_space: IOC3 PCI_ADDR reg reset val exp:` *expected* `recv:` *received*

  The PCICONFIG test failed; the failing FRU is the BaseIO board.

- `io6config_space: IOC3 PCI_ADDR reg write read error. wrote:` *value* `read:` *value*

  The PCICONFIG test failed; the failing FRU is the BaseIO board.

- `io6config_space: PCI device` *device* `QL command status reg reset val exp:` *expected* `recv:` *received*

  The PCICONFIG test failed; the failing FRU is the BaseIO board.

- `io6config_space: PCI device` *device* `QL class code and revision read as` *value*

  The PCICONFIG test failed; the failing FRU is the BaseIO board.

- `io6config_space: PCI device` *device* `QL base address reg reset val exp:` *expected* `recv:` *received*

  The PCICONFIG test failed; the failing FRU is the BaseIO board.

- `io6config_space: PCI device` *device* `QL base addr write read error. wrote:` *value* `read:` *value*

  The PCICONFIG test failed; the failing FRU is the BaseIO board.

- `RSLT io6config_space FAIL`

  This message follows any of the previous failure messages.

3.9.2 PCI Bus Test

The PCI bus test checks the PCI bus with walking data tests.

3.9.2.1 How to Run This Test

This test runs automatically during the boot process, or you can run it from the POD prompt. To run this test from the POD prompt, enter the following command:

dgpci [mn | mh | mm] [n*NODE*] [s*SLOT*] [p*PCINUM*]

The **m** option specifies the mode in which the test should run. Enter **mn** to run the test in normal mode, enter **mh** to run the test in heavy mode, or enter **mm** to run the test in manufacturing mode. (The default mode is normal mode.)

The **n** option specifies the Node ID (NASID value) of the Node on which the test should run. (The default Node is the Node from which you entered the command.)

The **s** option specifies the IO slot number of the BaseIO board that you want to test. (The default IO slot number is 1, which selects slot IO1.)

The **p** option specifies the PCI device number of the IOC3 ASIC that you want to test. (The default PCI device number is 2, which selects the IOC3 chip on the BaseIO board.)

For example, enter **dgpci mn s3** to run the PCI bus test in normal mode on the Node from which you entered the command. This example uses the IOC3 ASIC on the BaseIO board in slot number IO3 of the module that contains the Node from which you started the test.

3.9.2.2 Test Description

This test uses the following algorithm:

1. Write a walking one pattern in the 32-bit Ethernet low address register.
2. Read the data back.
3. Compare the data that was read back with the data that was written.
4. Write a walking 0's pattern in the 32-bit Ethernet low address register.
5. Read the data back.
6. Compare the data that was read back with the data that was written.

3.9.2.3 Failure Information

The PCI bus test returns the following failure messages:

- `pcibus_sanity: IOC3 enet hashed low addr reg reset val exp:` *expected* `recv:` *received*

  This message indicates a failing PCI or failing IOC3 chip. The failing FRU is the BaseIO board.

- `pcibus_sanity: PCI walk one test. exp:` *expected* `recv:` *received*

  This message indicates a failing PCI bus. The failing FRU is the BaseIO board.

- `pcibus_sanity: PCI walk zero test. exp:` *expected* `recv:` *received*

  This message indicates a failing PCI bus. The failing FRU is the BaseIO board.

- `RSLT pcibus_sanity FAIL`

  This message follows any of the previous PCI bus failure messages. The failing FRU is the BaseIO board.

3.10 Serial Port Tests

3.10.1 Serial Programmed I/O Test

The serial programmed I/O (PIO) test uses the internal loopback features of the SuperIO chip to test the serial PIO functionality of the BaseIO board.

3.10.1.1 Prerequisites to This Test

Before you can run this test in external loopback mode (by using the **mx** command line option), you must attach loopback connectors to the serial ports that you will test.

3.10.1.2 How to Run This Test

This test runs automatically during the boot process, or you can run it from the POD prompt. To run this test from the POD prompt, enter the following command:

dgspio [mn | mh | mm | mx] [n*NODE*] [s*SLOT*] [p*PCINUM*]

The **m** option specifies the mode in which the test should run. Enter **mn** to run the test in normal mode, enter **mh** to run the test in heavy mode, enter **mm** to run the test in manufacturing mode, or enter **mx** to run the test in external loopback mode. (The default mode is normal mode.)

The **n** option specifies the Node ID (NASID value) of the Node on which the test should run. (The default Node is the Node from which you entered the command.)

The **s** option specifies the IO slot number of the BaseIO board that you want to test. (The default IO slot number is 1, which selects slot IO1.)

The **p** option specifies the PCI device number of the IOC3 ASIC that you want to test. (The default PCI device number is 2, which selects the IOC3 ASIC on the BaseIO board.)

For example, enter **dgspio mh** to run the serial PIO test in heavy mode on the Node from which you entered the command. The test uses the IOC3 chip on the BaseIO board in slot IO1 of the module that contains the Node from which you started the test.

3.10.1.3 Test Description

This test uses the following algorithm:

1. Reset the SuperIO chip.
2. Fill the transmit FIFO with data.
3. Read the receive FIFO in loopback mode.
4. Compare the data in the receive FIFO with the data in the transmit FIFO.

3.10.1.4 Failure Information

The serial PIO test returns the following failure messages:

- `serial_pio: UARTA read timed out on LSR<Data Ready>`

  The serial PIO test failed; the failing FRU is the BaseIO board.

- `serial_pio: UARTB read timed out on LSR<Data Ready>`

  The serial PIO test failed; the failing FRU is the BaseIO board.

- `serial_pio: UARTA error. Chars in FIFO after reset .....`

  This message indicates a failing SuperIO chip or PCI.

- `serial_pio: UARTB error. Chars in FIFO after reset .....`

  This message indicates a failing SuperIO chip. The failing FRU is the BaseIO board.

- `serial_pio: UARTA sent:` *value* `recv:` *value*

  This message indicates that the data that was received is different from the data that was sent. This message indicates a failing IOC3-to-SuperIO interface, a failing SuperIO chip, or a failing IOC3 chip.

  This message also appears if you do not install external loopback plugs on the ports before you run the test in external loopback mode.

- `serial_pio: UARTB sent:` *value* `recv:` *value*

  This message indicates that the data that was received is different from the data that was sent. This message indicates a failing IOC3-to-SuperIO interface, a failing SuperIO chip, or a failing IOC3 chip.

  This message also appears if you do not install external loopback plugs on the ports before you run the test in external loopback mode.

- `serial_pio failed with` *number* `errors`

  The serial PIO test failed; the failing FRU is the BaseIO board.

- `RSLT serial_pio FAIL`

  This message follows any of the previous serial PIO failure messages.

3.10.2 Serial DMA Test

The serial DMA test verifies that serial DMA can be performed.

3.10.2.1 Prerequisites to This Test

The CPU from which you run this test must be in Cac mode. If you attempt to run this test from a CPU that is in Dex mode or Unc mode, the test displays a message that indicates you should enter Cac mode (type **go cac**) before you run the test.

Before you can run this test in external loopback mode (by using the **mx** command line option), you must attach loopback connectors to the serial ports that you will test.

3.10.2.2 How to Run This Test

This test runs automatically during the boot process, or you can run it from the POD prompt. To run this test from the POD prompt, enter the following command:

dgsdma [mn | mh | mm | mx] [n*NODE*] [s*SLOT*] [p*PCINUM*]

The **m** option specifies the mode in which the test should run. Enter **mn** to run the test in normal mode, enter **mh** to run the test in heavy mode, enter **mm** to run the test in manufacturing mode, or enter **mx** to run the test in external loopback mode. (The default mode is normal mode.)

The **n** option specifies the Node ID (NASID value) of the Node on which the test should run. (The default Node is the Node from which you entered the command.)

The **s** option specifies the IO slot number of the BaseIO board that you want to test. (The default IO slot number is 1, which selects slot IO1.)

The **p** option specifies the PCI device number of the IOC3 ASIC that you want to test. (The default PCI device number is 2, which selects the IOC3 ASIC on the BaseIO board.)

For example, enter **dgsdma mh** to run the serial DMA test in heavy mode on the Node from which you entered the command. The test uses the IOC3 chip on the BaseIO board in slot IO1 of the module that contains the Node from which you started the test.

3.10.2.3 Test Description

This test uses the following algorithm:

1. Set up the ring buffers.
2. Set up DMA in loopback mode.
3. Perform DMA operations simultaneously through UART channels A and B.
4. Verify the results.

3.10.2.4 Failure Information

The serial DMA test returns the following failure messages:

- `serial_dma: PCI: could not get the ioc3 base addr`

  This message indicates a failing PCI or failing IOC3 chip. The failing FRU is the BaseIO board.

- `serial_dma: Serial DMA exp:` *expected* `recv:` *received*

  This message indicates that the data that was received is different from the data that was sent. This message indicates a failing SuperIO chip, a failing IOC3 chip, a failing Bridge, or a failing HUB.

  This message also appears if you do not install external loopback plugs on the ports before you run the test in external loopback mode.

- `serial_dma: Timeout on uartA dma ....`

  The serial DMA test failed; the failing FRU is the BaseIO board.

- `serial_dma: Timeout on uartB dma ....`

  The serial DMA test failed; the failing FRU is the BaseIO board.

- `RSLT serial_dma FAIL`

  This message follows any of the previous serial DMA failure messages.

Chapter 4

BaseIO PROM Tests

This chapter describes the power-on diagnostic tests that are included in the BaseIO PROM. These diagnostics test the SCSI controller and the Ethernet port and its related hardware.

Each test description includes the following information:

- Any prerequisites that you should perform before you run the test
- Information about how to run the test
- A description of actions that the test performs
- Failure information for the test (including the failure messages and the failing hardware components)

4.1 SCSI Tests

The BaseIO PROM's SCSI driver automatically calls the SCSI tests when the boot sequence installs the BaseIO SCSI devices. You can also manually run them from the >> prompt, which you can access by selecting option `5. Enter Command Monitor` of the System Maintenance Menu. You cannot run these tests from the `POD>` prompt.

When the tests run automatically, they use the mode that is set by the debug DIP switches. When you run the tests manually, the command line options that you specify override the mode that is set by the debug DIP switches.

The *diag_scsi* command runs the SCSI tests from the >> prompt and uses the following syntax:

diag_scsi [-N *nasid*] [-m *moduleid* -s *slotid*] [-p *pcidev*] [-h] [-v] [-t *testname*] [-r *count* | -R *count*]

where:

-N *nasid*	Specifies the NASID number of the module to test (defaults to the console BaseIO module). This option supersedes the NASID that is indicated by the module and slot values.
-m *moduleid*	Specifies the module number of the module to test (defaults to the console BaseIO module).
-s *slotid*	Specifies the name of the slot location that contains the module to test. (The default is the console BaseIO slot).
-p *pcidev*	Specifies the PCI device number of the target IOC3. (The default is 0, which is the device number of the first SCSI controller on an BaseIO board.)
-h	Runs the diagnostic in heavy mode.
-v	Runs the diagnostic in verbose mode, which returns detailed status and error messages.
-t *testname*	Runs only the specified test. Valid values for *testname* are ram, dma, and controller. (The default is to run all tests.)
-r *count*	Repeats the test *count* times. The test stops when it detects a failure.
-R *count*	Repeats the test *count* times. The test continues to execute when it detects a failure.

4.1.1 SCSI SSRAM Test

The SCSI SSRAM test verifies that data can be loaded into SCSI SSRAM memory and read back successfully. It also verifies the "mailbox" mechanism before it performs this test.

4.1.1.1 How to Run This Test

To run the SCSI SSRAM test from the >> prompt, enter the **`diag_scsi -t ram`** command and any other command line options that you want to use. For example:

>> **`diag_scsi -t ram [`***`options`***`]`**

Refer to Section 4.1 for descriptions of all available *options*.

4.1.1.2 Test Description

This test uses the following algorithm:

1. Verify the SCSI "mailbox" mechanism.
2. Write data into the SCSI SSRAM (one word at a time).
3. Read data back from the SCSI SSRAM (one word at a time).
4. Compare the data that was read back with the data that was written.

4.1.1.3 Failure Information

The SCSI SSRAM test returns the following failure messages:

- `scsi_ram: SCSI mailbox error. erp:` *expected* `recv:` *received*

  This message indicates a failing SCSI controller. The failing FRU is the BaseIO board.

- `scsi_ram: QL: wrote:` *write_value* `read` *read_value* `at addr:` *address*

  This message indicates failing SCSI SSRAM. The failing FRU is the BaseIO board.

- `RSLT scsi_dma FAIL`

  This message follows any of the previous SCSI SSRAM failure messages.

4.1.2 SCSI DMA Test

The SCSI DMA test verifies that the SCSI DMA mechanism is functional.

4.1.2.1 Prerequisites to This Test

The CPU from which you run this test must be in Cac mode. If you attempt to run this test from a CPU that is in Dex mode or Unc mode, the test displays a message that indicates you should enter Cac mode (type **`go cac`**) before you run the test.

4.1.2.2 How to Run This Test

To run the SCSI DMA test from the >> prompt, enter the **`diag_scsi -t dma`** command and any other command line options that you want to use. For example:

>> **`diag_scsi -t dma [`***options*`]`**

Refer to Section 4.1 for descriptions of all available *options*.

4.1.2.3 Test Description

The SCSI DMA test uses the following algorithm:

1. Use a SCSI DMA operation to write the SCSI SSRAM with data.
2. Use a SCSI DMA operation to read the data that is in the SCSI SSRAM.
3. Compare the data that was dumped with the data that was loaded.

4.1.2.4 Failure Information

The SCSI DMA test returns the following failure messages:

- `scsi_dma: LOAD SSRAM command failed`

  This message indicates a failing SCSI controller. The failing FRU is the BaseIO board.
- `scsi_dma: DUMP SSRAM command failed`

  This message indicates a failing SCSI controller. The failing FRU is the BaseIO board.
- `scsi_dma: DMA failure exp` *expected* `recv` *received* `at` *location*

  This message indicates a failing SCSI controller, Bridge chip, or HUB chip.
- `RSLT scsi_dma FAIL`

  This message follows any of the previous SCSI DMA failure messages.

4.1.3 SCSI Self-Test

The SCSI self-test uses the SCSI self-test firmware to test the SCSI controller.

4.1.3.1 How to Run This Test

To run the SCSI self-test from the >> prompt, enter the **`diag_scsi -t controller`** command and any other command line options that you want to use. For example:

>> **`diag_scsi -t controller [`***options*`]`**

Refer to Section 4.1 for descriptions of all available *options*.

4.1.3.2 Test Description

This test uses the following algorithm:

1. Load the SCSI self-test firmware.
2. Perform the self-test on the SCSI controller. The self-test tests the RISC unit, the SXP FIFOs, and the SCSI bus.

4.1.3.3 Failure Information

The SCSI self-test returns the following failure messages:

- `scsi_controller: EXECUTE FIRMWARE Command Failed.`

  This message indicates a failing SCSI controller. The failing FRU is the BaseIO board.

- `scsi_controller: write data failure at` *count* `word`

  This message indicates a failing SCSI controller or SCSI bus. The failing FRU is the BaseIO board.

- `scsi_controller: Bad status val ==>` *value* `....`

  This message indicates a failing SCSI controller. The failing FRU is the BaseIO board.

- `RSLT scsi_controller FAIL`

  This message follows any of the previous SCSI self-test failure messages.

4.2 Ethernet Tests

The BaseIO PROM's Ethernet driver automatically calls the Ethernet tests when the boot sequence installs the BaseIO Ethernet device. You can also manually run them from the >> prompt, which you can access by selecting option `5. Enter Command Monitor` of the System Maintenance Menu. You cannot run these tests from the `POD>` prompt.

When the tests run automatically, they use the mode that is set by the debug DIP switches. When you run the tests manually, the command line options that you specify override the mode that is set by the debug DIP switch settings.

The *diag_enet* command runs the Ethernet tests from the >> prompt and uses the following syntax:

diag_enet [-N *nasid*] [-m *moduleid* -s *slotid*] [-p *pcidev*] [-h] [-e | -x] [-v] [-t *testname*] [-r *count* | -R *count*]

where:

-N *nasid*	Specifies the NASID number of the module to test (defaults to the console BaseIO module). This option supersedes the NASID that is indicated by the module and slot values.
-m *moduleid*	Specifies the module number of the module to test (defaults to the console BaseIO module).
-s *slotid*	Specifies the name of the slot location of the module to test. (The default is the console BaseIO slot).
-p *pcidev*	Specifies the PCI device number of the target IOC3. (The default is 2, which is the device number of the IOC3 on a BaseIO board.)
-h	Runs the diagnostic in heavy mode.
-e or -x	Runs the diagnostic in external loopback mode.
-v	Runs the diagnostic in verbose mode, which returns detailed status and error messages.
-t *testname*	Runs only the specified test. Valid values for testname are ssram, txclk, phyreg, nic, ioc3_loop, phy_loop, tw_loop, ext_loop, ext_loop_10, ext_loop_100, and xtalk. (The default is to run all tests.)
-r *count*	Repeats the test *count* times. The test stops when it detects a failure.
-R *count*	Repeats the test *count* times. The test continues executing when it detects a failure.

The Ethernet diagnostics are written with the assumption that the following hardware has been tested and is functional:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address and data
- Bridge ASIC
- XTALK interface between the Bridge and XBOW chips
- XBOW, midplane, and all IP27 board components

Note: When the boot sequence runs the Ethernet diagnostics, all of these components have already been tested by other POD diagnostics. However, the actions that the Ethernet diagnostics perform stress these components harder than the other diagnostics, so it is possible that a previously undetected failure in one of these components could cause the Ethernet diagnostics to fail.

The Ethernet diagnostic tests are:

- Ethernet SSRAM test
- TX clock test
- PHY register test
- NIC test
- IOC3 internal loopback test
- PHY chip internal loopback test
- Twister chip internal loopback test
- External loopback DMA test
- External loopback DMA (10 Mb/s) test
- External loopback DMA (100 Mb/s) test
- XTALK stress test
- Comprehensive Ethernet test

4.2.1 Comprehensive Ethernet Test

The comprehensive Ethernet test runs all of the Ethernet tests in an order that optimizes fault isolation. The ef_install routine in the BaseIO PROM Ethernet driver calls this test during the boot sequence.

Note: If a failure of one of the diagnostics implies failures in subsequent diagnostics, this test does not run the diagnostics that will fail.

4.2.1.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address / data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

4.2.1.2 How to Run This Test

This test runs automatically as part of the boot sequence. To manually run this test from the >> prompt, enter **`diag_enet [`***options***`]`**.

Refer to Section 4.2 for descriptions of all available *options*.

4.2.1.3 Test Description

This test uses the following algorithm to control the execution of the other Ethernet tests:

```
Run the SSRAM test
Run the TX clock test
Run the PHY register test
Run the NIC test
If the SSRAM test passed
  {
      Run the IOC3 internal loopback DMA test
      If the IOC3 internal loopback DMA test passed AND the TX clock test passed AND
      the PHY register test passed
      {
          Run the PHY internal loopback DMA test
          If the PHY internal loopback DMA test passed
          {
              Run the Twister chip internal loopback DMA test
              If the PHY internal loopback DMA test passed AND external loopback
              mode is selected
              {
                  Run the external loopback DMA test
              }
              If the PHY internal loopback DMA test passed AND external loopback DMA
              test passed (or was not run) AND heavy mode or external loopback mode
              was selected
              {
                  Run the XTALK stress test
              }
          }
      }
  }
```

For detailed information about each of the tests, refer to the corresponding description later in this document.

4.2.1.4 Failure Information

The comprehensive Ethernet diagnostic generates the following failure messages:

- `enet_all: SSRAM test failed.`

  The SSRAM test failed. Refer to Section 4.2.2, “Ethernet SSRAM Test,” for more information about the failure messages that the SSRAM test returns.

- `enet_all: Ethernet TX clock test failed.`

  The TX clock test failed. Refer to Section 4.2.3, “TX Clock Test,” for more information about the failure messages that the TX clock test returns.

- `enet_all: Ethernet PHY chip register test failed.`

  The PHY register test failed. Refer to Section 4.2.4, “PHY Register Test,” for more information about the failure messages that the PHY register test returns.

- `enet_all: Ethernet NIC test failed.`

  The NIC test failed. Refer to Section 4.2.5, “NIC Test,” for more information about the failure messages that the NIC test returns.

- `enet_all: IOC3 internal loopback DMA test failed.`

  The IOC3 internal loopback DMA test failed. Refer to Section 4.2.6, “IOC3 Internal Loopback Test,” for more information about the failure messages that the IOC3 internal loopback test returns.

- `enet_all: Ethernet PHY chip internal loopback DMA test failed.`

  The PHY chip internal loopback DMA test failed. Refer to Section 4.2.7, “PHY Chip Internal Loopback Test,” for more information about the failure messages that the PHY chip internal loopback DMA test returns.

- `enet_all: Ethernet Twister chip internal loopback DMA test failed.`

  The Twister chip internal loopback DMA test failed. Refer to Section 4.2.8, “Twister Chip Internal Loopback Test,” for more information about the failure messages that the Twister chip internal loopback DMA test returns.

- `enet_all: Ethernet external loopback DMA test failed.`

  The external loopback DMA test failed. Refer to Section 4.2.9, “External Loopback DMA Test,” for more information about the failure messages that the external loopback DMA test returns.

- `enet_all: Xtalk stress test failed.`

  The XTALK stress test failed. Refer to Section 4.2.12, “XTALK Stress Test,” for more information about the failure messages that the XTALK stress test returns.

For fault isolation information, refer to the individual test descriptions.

4.2.2 Ethernet SSRAM Test

The Ethernet SSRAM test checks the 128-KB Ethernet packet-buffering SSRAM, the SSRAM's interface to the IOC3 ASIC, and the parity generation and checking logic in the IOC3.

4.2.2.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address / data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

4.2.2.2 How to Run This Test

To run the Ethernet SSRAM test from the >> prompt, enter the **`diag_enet -t ssram`** command and any other command line options that you want to use. For example:

>> **`diag_enet -t ssram [`***options***`]`**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.2.3 Test Description

This test uses the following algorithm:

1. Perform an address walk test.
   - Write a test pattern to addresses 0, 1, 2, 4, 8, ...
   - Read the pattern back and verify that the data that was read back equals the data that was written.
2. Perform a parity test.
   - Write a value with good parity = 0 to memory location 0.
   - Read the value back, verify that the value that was read back equals the value that was written, and verify that the error bit did not set.
   - Write a value with good parity = 1 to memory location 0.
   - Read the value back, verify that the value that was read back equals the written value, and verify that the error bit did not set.
   - Write a value with bad parity = 0 to memory location 0.
   - Read the value back, verify that the value that was read back equals the value that was written, and verify that the error bit set.
   - Write a value with bad parity = 1 to memory location 0.
   - Read the value back, verify that the value that was read back equals the value that was written, and verify that the error bit set.

3. In heavy and manufacturing modes only, perform a data pattern test.
    - Write a test pattern to all memory addresses, alternating between low and high addresses.
    - Read the data back and verify that the data that was read back equals the data that was written.

In normal mode, the address walk test and parity tests are the only tests that run.

In heavy and manufacturing modes, all tests run.

4.2.2.4 Failure Information

The Ethernet SSRAM test generates the following failure messages:

- `enet_ssram: could not get the ioc3 base addr`

  This indicates a programming error in the test; you should never see this error message in the field.

- `enet_ssram: ssram address walk error. Ex:` *expected_value* `Recv:` *received_value*

  The address walk test failed; the data that was read back (*received_value*) did not equal the data that was written (*expected_value*).

  The failing FRU is the BaseIO board. The failing components are the IOC3-to-SSRAM wires (most likely), the SSRAM, or the IOC3 ASIC (least likely).

- `enet_ssram: ssram parity test error. Exp:` *expected_value* `Recv:` *received_value*

  The parity test failed; the data that was read back (*received_value*) did not equal the data that was written (*expected_value*).

  The failing FRU is the BaseIO board. The failing component might be the IOC3 ASIC or the SSRAM.

- `enet_ssram: IOC3 ssram test failed...`

  This message will be removed from a future version of the test.

- `enet_ssram: **Data miscompare(A)** offset:` *offset_value* `exp:` *expected_value* `recv:` *received_value*

  The data pattern test failed. The failing FRU is the BaseIO board. The most likely failing component is the SSRAM. Other (less likely) failing components include the IOC3-to-SSRAM wires and the IOC3 ASIC.

4.2.3 TX Clock Test

The TX clock test verifies the presence of the TX clock that is sent from the PHY chip to the IOC3 ASIC. This test reads the IOC3 ETCSR register and checks the NO_TX_CLK bit.

4.2.3.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address / data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

4.2.3.2 How to Run This Test

To run the TX clock test from the >> prompt, enter the **`diag_enet -t txclk`** command and any other command line options that you want to use. For example:

>> **`diag_enet -t txclk [`***`options`***`]`**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.3.3 Test Description

This test uses the following algorithm:

1. Read the IOC3 ETCSR register.
2. Check the NO_TX_CLK bit.

This test runs the same in normal and heavy modes. Manufacturing mode returns more status information than normal and heavy modes.

4.2.3.4 Failure Information

The TX clock test returns the following failure message:

- `enet_tx_clk: IOC3 is not seeing the PHY TX clock.`

  The TX clock test failed. The failing FRU is the BaseIO board. The most likely failing hardware component is the TX clock wire. Other (less likely) failing components are the PHY chip and the IOC3 ASIC.

4.2.4 PHY Register Test

The PHY register test checks the reset values of registers on the PHY chip (a National Semiconductor DP83840 chip). This test also verifies the capability to read and write selected bits in selected registers on the PHY chip; this test checks only the bits and registers that the Ethernet driver and the other Ethernet diagnostics use.

Note: Some documentation, including the BaseIO board schematics, refers to the PHY chip as the X-chip.

4.2.4.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address / data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

4.2.4.2 How to Run This Test

To run the PHY register test from the >> prompt, enter the **`diag_enet -t phyreg`** command and any other command line options that you want to use. For example:

>> **`diag_enet -t phyreg [`***options***`]`**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.4.3 Test Description

This test uses the following algorithm:

1. Verify the presence of the PHY chip.
2. If the PHY chip is present, perform the following actions:
   - Reset the PHY chip with a software reset.
   - Verify the reset values of all registers in the PHY chip, and generate failure messages for any miscompares that are detected.
   - Read and write data to selected bits in registers 0, 23, and 24. Compare the data that is read back to the written data, and generate failure messages for any miscompares that are detected.

4.2.4.4 Failure Information

The PHY register test generates the following failure messages:

- `enet_phy_reg: Read of PHY register 0 failed (returned 0xffff).`

  The PHY presence test failed. The failing FRU is the BaseIO board. The most likely failing component is the IOC3-to-PHY wiring (the MII interface wires or the reset wire). Other possible failing components are the PHY chip and its clocks or the IOC3 ASIC.

- `enet_phy_reg: Timed out waiting for PHY to rest.`

  The software reset of the PHY chip did not complete. The failing FRU is the BaseIO board. The most likely failing hardware component is the PHY chip and its clocks. Other less likely failing components are the IOC3-to-PHY wiring and the IOC3 ASIC.

- `enet_phy_reg:  Timed out on MICR_BUSY before read.`
  `enet_phy_reg:  Timed out on MICR_BUSY during read.`

  A read operation of a PHY register did not complete. The failing FRU is the BaseIO board. The most likely failing hardware component is the IOC3-to-PHY wiring. Other possible failing components include the PHY chip and the IOC3 ASIC.

- `enet_phy_reg: Timed out on MICR_BUSY before write.`
  `enet_phy_reg: Timed out on MICR_BUSY during write.`

  A write operation to a PHY register did not complete. The failing FRU is the BaseIO board. The most likely failing hardware component is the IOC3-to-PHY wiring. Other possible failing components include the PHY chip and the IOC3 ASIC.

- `enet_phy_reg: Warning - PHY Identifier register #1 value` *value* `is not supported by this diag!`
  `enet_phy_reg: Warning - PHY Identifier register #2 value` *value* `is not supported by this diag!`
  `enet_phy_reg: Warning - PHY revision register value` *value* `is not supported by this diag!`

  The test failed while checking a reset value in one of the registers that might change in future revisions of the PHY chip. This most likely indicates that the BaseIO board includes a newer version of the PHY chip and the diagnostic software has not been updated with this information. It could also indicate a failing PHY chip.

- `enet_phy_reg: Warning - unexpected Reg` *register* `reset value, exp:` *expected* `mask:` *mask_value* `got:` *received*

  The test failed while checking the register reset values. The failing FRU is the BaseIO board. The most likely failing hardware component is the PHY chip. Another possible failing component is the wiring from the IOC3 ASIC to the PHY chip.

- `enet_phy_reg: Reg 0 value wrong, exp:` *expected* `mask:` *mask_value* `got:` *received*
  `enet_phy_reg: Reg 23 value wrong, exp:` *expected* `mask:` *mask_value* `got:` *received*
  `enet_phy_reg: Reg 24 value wrong, exp:` *expected* `mask:` *mask_value* `got:` *received*

  A failure occurred during the write and read test to register 0, 23, or 24. The failing FRU is the BaseIO board. The most likely failing hardware component is the PHY chip. Another possible failing component is the wiring from the IOC3 ASIC to the PHY chip.

4.2.5 NIC Test

The NIC test checks the number in a can (NIC) device (a Dallas Semiconductor chip) on the BaseIO board; the NIC device stores the Ethernet MAC address. This test reads the NIC and then performs further testing if the results of the read are incorrect.

4.2.5.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address / data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

4.2.5.2 How to Run This Test

To run the NIC test from the >> prompt, enter the **`diag_enet -t nic`** command and any other command line options that you want to use. For example:

>> **`diag_enet -t nic [`***options***`]`**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.5.3 Test Description

This test uses the following algorithm:

1. Call an external function that reads the NIC.
2. Provide further testing if the NIC value is incorrect.

This test runs the same in normal and heavy modes. Manufacturing mode returns more status information than normal and heavy modes.

4.2.5.4 Failure Information

The NIC test returns the following failure messages:

- `nic_eaddr: No presence pulse from NIC`
  `nic_eaddr: (check for an empty NIC socket)`

  The NIC test failed. The failing FRU is the BaseIO board. The most likely cause is a missing NIC device. Other possible causes are a failing socket, failing NIC device, failing wiring from the IOC ASIC to the socket, or a failing IOC3 ASIC.

- `nic_eaddr: Invalid enet MAC command CRC byte` *value*`, s/b 0x8d`
  `nic_eaddr: Invalid enet crc16` *value*`, s/b 0xb001`
  `nic_eaddr: Invalid enet length byte` *value* `found in NIC, s/b 0x0a`
  `nic_eaddr: Invalid enet MAC crc16` *value*`, s/b 0xb001`
  `nic_eaddr: Failed to find readable MAC NIC, rc` *value*

  The NIC test failed. The failing FRU is the BaseIO board. The failing hardware component is the NIC device.

4.2.6 IOC3 Internal Loopback Test

The IOC3 internal loopback test verifies Ethernet DMA by using the internal loopback features of the IOC3 ASIC. (The IOC3 loopback DMA runs at 66.7 Mb/s by using its internal clock as a TX clock.)

4.2.6.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address/data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

4.2.6.2 How to Run This Test

To run the IOC3 internal loopback test from the >> prompt, enter the **`diag_enet -t ioc3_loop`** command and any other command line options that you want to use. For example:

>> **`diag_enet -t ioc3_loop [`***options***`]`**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.6.3 Test Description

This test uses the following algorithm:

1. Allocate memory for the TX descriptor ring and write the base address to the ETBR registers in the IOC3.
2. Allocate memory for the TX data buffers and write their addresses to the descriptor ring entries and to the command and buffer count fields.
3. Generate test data and write it to the TX buffers. (The test pattern is designed to stress PCI SSO.)
4. Allocate memory for the RX buffer pointer ring and write the base address to the ERBR registers on the IOC3.
5. Allocate memory for the RX buffers.
6. Enable error interrupts on the Bridge ASIC.
7. Reset the IOC3 Ethernet DMA engines, initialize the IOC3 registers for IOC3 loopback DMA, and enable the TX_EMPTY interrupt.
8. Write to the ring pointers in a way that causes the DMA to occur in one burst of 10 packets (in normal mode) or one burst of 495 packets (in heavy and manufacturing modes) and then stop.
9. Write to the EMCR register on the IOC3 to enable DMA.
10. Execute a polling loop to wait for the IOC3 to post TX_EMPTY in the EISR. During the polling loop, monitor for unexpected error conditions. Time out if this takes too long.
11. Check the Bridge ASIC ISR to verify that the interrupt went from the IOC3 to the Bridge.
12. Wait for the valid bit to set in the RX buffer for the last packet. While waiting for this to occur, the test monitors for unexpected error conditions.
13. Time out if it takes too long for the valid bit to set in the RX buffer for the last packet.
14. Check all received packets: Check the status field for each packet and also verify that the data matches the data that was transmitted.
15. Clean up: Free the allocated memory, restore interrupt enables, and reset the PHY chip.

In normal mode, this test transmits and receives 10 packets; each packet contains 301 bytes of data. In heavy and manufacturing modes, this test transmits and receives 495 packets; each packet contains 1,501 bytes of data.

4.2.6.4 Failure Information

The IOC3 internal loopback test returns the following failure messages:

- `enet_ioc3_loop: Timed out on EMCR_ARB_DIAG_IDLE trying to reset enet.`

  The IOC3 PCI arbiter was stuck with a request to the Bridge before this test could start. The failing FRU is the BaseIO board. Failing hardware component information is not available because the test could not run.

- `enet_ioc3_loop: Unexpected EISR condition while waiting for TX_EMPTY (`*string1*`: `*string2*`).`
  `enet_ioc3_loop:  EISR value =` *e_value*
  `enet_ioc3_loop:  Elapsed time =` *time* `us`

  The IOC3 ASIC detected an error during a DMA operation. The EISR value (shown in *e_value*) indicates the failing hardware. Refer to Table 4-1 for descriptions of the EISR values

Table 4-1 IOC3 Loopback Test Failure Information Based on the EISR Values

EISR Value	Failure Information
0x00000010	RX_MEMERR RX DMA access to onboard PCI bus failed The failing FRU is the BaseIO board. The cause could be a PCI parity error.
0x00000020	RX_PARERR SSRAM parity error during RX DMA The failing FRU is the BaseIO board. The most likely cause is failing SSRAM. Other possible causes are failing IOC3-to-SSRAM wiring or a failing IOC3 ASIC.
0x00200000	TX_BUF_UFLO This message is a by-product of an SSRAM parity error.
0x02000000	TX_MEMERR TX DMA access to onboard PCI bus failed. The failing FRU is the BaseIO board. The cause could be a PCI parity error or an unrecoverable error in the memory subsystem.
0x04000000	TX_PARERR SSRAM parity error during TX DMA. The failing FRU is the BaseIO board. The most likely cause is failing SSRAM. Other possible causes are failing IOC3-to-SSRAM wiring or a failing IOC3 ASIC.

- `enet_ioc3_loop: Timed out waiting for TX_EMPTY interrupt (`*string*`: `*string*`).`

  The DMA operation did not complete. The failing FRU is the BaseIO board. The failing hardware component is difficult to isolate.

- `enet_ioc3_loop: Bridge ISR bit` *value* `did not set after TX_EMPTY interrupt (`*string*`: `*string*`).`
  `enet_ioc3_loop:  ==> Bridge ISR =` *value*

  The interrupt between the IOC3 and the Bridge was not successful. The failing FRU is the BaseIO board. The most likely cause is a failing interrupt wire. Other possible causes are a failing IOC3 ASIC or a failing Bridge ASIC.

- `enet_ioc3_loop: Timed out polling RX buffer valid bit. (`*string*`: `*string*`).`
  `enet_ioc3_loop:` *value* `of` *value* `packets were received.`

  The RX buffers in main memory did not receive all of the packets that were sent. The failing FRU is the BaseIO board. The failing hardware component is difficult to isolate.

- `enet_ioc3_loop: Unexpected EISR condition while waiting for RX buffer valid bit (`*string*`: `*string*`)`
  `enet_ioc3_loop: EISR value =` *e_value*

  The IOC3 ASIC detected an error during a DMA operation. The EISR value (shown in *e_value*) indicates the failing hardware. Refer to Table 4-1 for descriptions of the EISR values.

- `enet_ioc3_loop: RX data miscompare on packet` *value* `first word.`
  `enet_ioc3_loop: ==> exp:` *expected* `recv:` *received* `(`*string*`:` *string*`).`

  The first word (which contains the valid bit, the byte count, and the IP checksum) of the RX data is corrupted. The failing FRU is the BaseIO board. The failing hardware component is difficult to isolate.

- `enet_ioc3_loop: RX data miscompare on packet` *value* `status word.`
  `enet_ioc3_loop: ==> exp:` *expected* `recv:` *received* `(`*string*`:` *string*`).`

  The status word of the RX data does not have the expected value. (This should never happen in IOC3 loopback mode.) The failing FRU is the BaseIO board.

- `enet_ioc3_loop: RX data miscompare on packet` *value* `DW` *value.*
  `enet_ioc3_loop: ==> exp:` *expected* `recv:` *received* `(`*string*`:` *string*`).`

  The specified doubleword of data is corrupted. The failing FRU is the BaseIO board. The failing hardware component is difficult to isolate.

- `enet_ioc3_loop: RX data miscompare on packet` *value* `byte` *value*`.`
  `enet_ioc3_loop: ==> exp:` *expected* `recv:` *received* `(`*string*`:` *string*`).`

  One of the last bytes of data is corrupted. The failing FRU is the BaseIO board. The failing hardware component is difficult to isolate.

4.2.7 PHY Chip Internal Loopback Test

The PHY chip loopback test checks Ethernet DMA by using the internal loopback features of the PHY chip on the BaseIO board. The PHY chip loopback DMA runs at 10 Mb/s and also at 100 Mb/s.

4.2.7.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address/data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

Also ensure that the following Ethernet tests pass before you run this test:

- IOC3 internal loopback test
- TX clock test
- PHY register test

4.2.7.2 How to Run This Test

To run the PHY chip internal loopback test from the >> prompt, enter the **diag_enet -t phy_loop** command and any other command line options that you want to use. For example:

>> **diag_enet -t phy_loop [***options***]**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.7.3 Test Description

This test uses the following algorithm twice (once for 10 Mb/s testing and then again for 100 Mb/s testing):

1. Allocate memory for the TX descriptor ring and write the base address to the ETBR registers on the IOC3.
2. Allocate memory for the TX data buffers and write their addresses to the descriptor ring entries and to the command and buffer count fields.
3. Generate test data and write it to the TX buffers. (The test pattern is designed to stress PCI SSO.)
4. Allocate memory for the RX buffer pointer ring and write the base address to the ERBR registers on the IOC3.
5. Allocate memory for the RX buffers.
6. Enable error interrupts on the Bridge ASIC.
7. Reset the IOC3 Ethernet DMA engines, initialize the IOC3 registers for PHY loopback DMA, and enable the TX_EMPTY interrupt.
8. Write to the ring pointers in a way that causes the DMA to occur in one burst of 10 packets (in normal mode) or one burst of 495 packets (in heavy and manufacturing modes) and then stop.
9. Reset the PHY chip and initialize it for internal loopback at 10 Mb/s (during first test sequence) or 100 Mb/s (during second test sequence).
10. Write to the EMCR register on the IOC3 to enable DMA.
11. Wait for the IOC3 to post TX_EMPTY in the EISR. During the polling loop, monitor for unexpected error conditions. Time out if this takes too long.
12. Check the Bridge ASIC ISR to verify that the interrupt got from the IOC3 to the Bridge.
13. Wait for the valid bit to set in the RX buffer for the last packet. While waiting for this to occur, the test monitors for unexpected error conditions.
14. Time out if it takes too long for the valid bit to set in the RX buffer for the last packet.
15. Check all received packets: Check the status field for each packet and also verify that the data matches the data that was transmitted.
16. Clean up: Free the allocated memory, restore interrupt enables, and reset the PHY chip.

In normal mode, this test transmits and receives 10 packets; each packet contains 301 bytes of data. In heavy and manufacturing modes, this test transmits and receives 495 packets; each packet contains 1,501 bytes of data.

4.2.7.4 Failure Information

The PHY chip internal loopback test returns the same messages as the IOC3 internal loopback test (refer to Section 4.2.6.4, "Failure Information,"). If the IOC3 internal loopback test passed, any failures that are listed as "difficult to isolate" are likely to be caused by a fault in the IOC3 chip to PHY chip wiring, the PHY chip, or the IOC3 ASIC.

This test also returns the following failure messages:

- `enet_phy_loop: Unexpected EISR condition while waiting for TX_EMPTY (`*`string`*`: `*`string`*`).`
  `enet_phy_loop:  EISR value =` *`e_value`*
  `enet_phy_loop:  Elapsed time =` *`value`* `us`

  The IOC3 detected an error during the DMA operation. The EISR value (shown in the output above as *e_value*) indicates the failing hardware. Refer to Table 4-2 for descriptions of the EISR values.

Table 4-2 PHY Chip Internal Loopback Test Failure Information Based on the EISR Values

EISR Value	Failure Information
0x00000010	RX_MEMERR RX DMA access to onboard PCI bus failed The failing FRU is the BaseIO board. The cause could be a PCI parity error.
0x00000020	RX_PARERR SSRAM parity error during RX DMA The failing FRU is the BaseIO board. The most likely cause is failing SSRAM. Other possible causes are failing IOC3-to-SSRAM wiring or a failing IOC3 ASIC.
0x00020000	TX_RTRY There were excessive collisions. The failing FRU is the BaseIO board. The cause is a stuck MII_COL signal from the PHY chip to the IOC3 ASIC.
0x00040000	TX_EXDEF There were excessive deferrals. The failing FRU is the BaseIO board. The cause is a stuck MII_CRS signal from the PHY chip to the IOC3 ASIC.
0x00200000	TX_BUF_UFLO This message is a by-product of an SSRAM parity error.
0x02000000	TX_MEMERR TX DMA access to onboard PCI bus failed. The failing FRU is the BaseIO board. The cause could be a PCI parity error or an unrecoverable error in the memory subsystem.
0x04000000	TX_PARERR SSRAM parity error during TX DMA. The failing FRU is the BaseIO board. The most likely cause is failing SSRAM. Other possible causes are failing IOC3-to-SSRAM wiring or a failing IOC3 ASIC.

Any failures in this test are strong evidence that the failing FRU is the BaseIO board.

4.2.8 Twister Chip Internal Loopback Test

The Twister chip internal loopback test verifies Ethernet DMA by using the internal loopback features of the Twister chip (a National Semiconductor DP83223 chip). The Twister chip loopback DMA runs at 100 Mb/s.

4.2.8.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address/data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

Also ensure that the PHY chip internal loopback test runs without errors before you run this test.

4.2.8.2 How to Run This Test

To run the Twister chip internal loopback test from the >> prompt, enter the **`diag_enet -t tw_loop`** command and any other command line options that you want to use. For example:

>> **`diag_enet -t tw_loop [`***options***`]`**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.8.3 Test Description

This test uses the following algorithm:

1. Allocate memory for the TX descriptor ring and write the base address to the ETBR registers on the IOC3.
2. Allocate memory for the TX data buffers and write their addresses to the descriptor ring entries and to the command and buffer count fields.
3. Generate test data and write it to the TX buffers. (The test pattern is designed to stress PCI SSO.)
4. Allocate memory for the RX buffer pointer ring and write the base address to the ERBR registers on the IOC3.
5. Allocate memory for the RX buffers.
6. Enable error interrupts on the Bridge ASIC.
7. Reset the IOC3 Ethernet DMA engines, initialize the IOC3 registers for Twister loopback DMA, and enable the TX_EMPTY interrupt.

8. Write to the ring pointers in a way that causes the DMA to occur in one burst of 10 packets (in normal mode) or one burst of 495 packets (in heavy and manufacturing modes) and then stop.
9. Reset the PHY chip and initialize it for Twister loopback at 100 Mb/s.
10. Write to the EMCR register on the IOC3 to enable DMA.
11. Wait for the IOC3 to post TX_EMPTY in the EISR. During the polling loop, monitor for unexpected error conditions. Time out if this takes too long.
12. Check the Bridge ASIC ISR to verify that the interrupt got from the IOC3 to the Bridge.
13. Wait for the valid bit to set in the RX buffer for the last packet. While waiting for this to occur, the test monitors for unexpected error conditions.
14. Time out if it takes too long for the valid bit to set in the RX buffer for the last packet.
15. Check all received packets: Check the status field for each packet and also verify that the data matches the data that was transmitted.
16. Clean up: Free the allocated memory, restore interrupt enables, and reset the PHY chip.

In normal mode, this test transmits and receives 10 packets; each packet contains 301 bytes of data. In heavy and manufacturing modes, this test transmits and receives 495 packets; each packet contains 1,501 bytes of data.

4.2.8.4 Failure Information

The Twister chip internal loopback test returns the same messages that the PHY chip internal loopback test returns (refer to Section 4.2.7.4, “Failure Information,”). Failures that are listed as “difficult to isolate” are likely caused by faulty wiring between the PHY chip and the Twister chip.

4.2.9 External Loopback DMA Test

The external loopback DMA test verifies Ethernet DMA by using an external loopback cable or connector. This test verifies the autonegotiation feature and verifies the DMA at 10 Mb/s and 100 Mb/s.

4.2.9.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address/data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

Also ensure that the Twister chip internal loopback test runs without errors before you run this test.

Install a loopback cable or connector before you run this test.

4.2.9.2 How to Run This Test

This test does not run during the boot sequence. It runs only from the command line. To run this test from the >> prompt, enter **diag_enet -t ext_loop [***options***]**. For example:

>> **diag_enet -t ext_loop [***options***]**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.9.3 Test Description

This test uses the following algorithm twice (once at 10 Mb / s and then again at 100 Mb / s):

1. Allocate memory for the TX descriptor ring and write the base address to the ETBR registers on the IOC3.
2. Allocate memory for the TX data buffers and write their addresses to the descriptor ring entries and to the command and buffer count fields.
3. Generate test data and write it to the TX buffers. (The test pattern is designed to stress PCI SSO.)
4. Allocate memory for the RX buffer pointer ring and write the base address to the ERBR registers on the IOC3.
5. Allocate memory for the RX buffers.
6. Enable error interrupts on the Bridge ASIC.
7. Reset the IOC3 Ethernet DMA engines, initialize the IOC3 registers for external loopback DMA, and enable the TX_EMPTY interrupt.
8. Write to the ring pointers in a way that causes the DMA to occur in one burst of 10 packets (in normal mode) or one burst of 495 packets (in heavy and manufacturing modes) and then stop.
9. Reset the PHY chip.
10. Test the autonegotiation features of the hardware.
11. Write to the EMCR register on the IOC3 to enable DMA.
12. Wait for the IOC3 to post TX_EMPTY in the EISR. During the polling loop, monitor for unexpected error conditions. Time out if this takes too long.
13. Check the Bridge ASIC ISR to verify that the interrupt went from the IOC3 to the Bridge.
14. Wait for the valid bit to set in the RX buffer for the last packet. While waiting for this to occur, the test monitors for unexpected error conditions.

15. Time out if it takes too long for the valid bit to set in the RX buffer for the last packet.
16. Check all received packets: Check the status field for each packet and also verify that the data matches the data that was transmitted.
17. Clean up: Free the allocated memory, restore interrupt enables, and reset the PHY chip.

In normal mode, this test transmits and receives 10 packets; each packet contains 301 bytes of data. In heavy mode, this test transmits and receives 495 packets; each packet contains 1,501 bytes of data. This test does not run in manufacturing mode.

4.2.9.4 Failure Information

This test returns the same failure information as the Twister loopback test (refer to Section 4.2.8.4, "Failure Information,"). If the Twister loopback test passed, then failures that were "difficult to isolate" are most likely caused by one of the following conditions:

- A faulty transformer on the BaseIO board.
- The wires between the PHY and the magnetics (for autonegotiation or 10 Mb/s DMA test).
- The wires between the Twister chip and the magnetics (for 100 Mb/s DMA test).
- The RJ45 connector.
- The loopback cable/connector.

If the loopback cable and connector are good, the failing FRU is the BaseIO board.

The following messages, which indicate that the autonegotiation test failed, are returned only with the external loopback test and indicate a failure of the autonegotiation test:

- `enet_ext_loop: Phy did not complete auto-negotiation with itself via external loopback cable.`
- `enet_ext_loop: PHY reg 6 LP_AN_ABLE bit not set after auto-negotiation.`
  `==>  Reg 0 =` *value*
  `==>  Reg 4 =` *value*
  `==>  Reg 5 =` *value*
  `==>  Reg 6 =` *value*
  `*** Are you sure there is an external loopback? ***`
- `enet_ext_loop: PHY reg 5 mode support bits (9:5) do not match reg 4 mode support bits after autonegotiation. Reg 4:` *value* `Reg 5:` *value*

4.2.10 External Loopback DMA (10 Mb/s) Test

The external loopback DMA (10 Mb / s) test is a version of the external loopback DMA test; this version has the following differences: it does not test autonegotiation and it runs only at 10 Mb / s.

4.2.10.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address / data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

Also ensure that the Twister chip internal loopback test runs without errors before you run this test.

Install a loopback cable or connector before you run this test.

4.2.10.2 How to Run This Test

This test does not run during the boot sequence. It runs only from the command line. To run this test from the >> prompt, enter **`diag_enet -t ext_loop_10 [`***options***`]`**. For example:

>> **`diag_enet -t ext_loop_10 [`***options***`]`**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.10.3 Test Description

This test uses the following algorithm:

1. Allocate memory for the TX descriptor ring and write the base address to the ETBR registers on the IOC3.
2. Allocate memory for the TX data buffers and write their addresses to the descriptor ring entries and to the command and buffer count fields.
3. Generate test data and write it to the TX buffers. (The test pattern is designed to stress PCI SSO.)
4. Allocate memory for the RX buffer pointer ring and write the base address to the ERBR registers on the IOC3.
5. Allocate memory for the RX buffers.
6. Enable error interrupts on the Bridge ASIC.

7. Reset the IOC3 Ethernet DMA engines, initialize the IOC3 registers for external loopback DMA, and enable the TX_EMPTY interrupt.
8. Write to the ring pointers in a way that causes the DMA to occur in one burst of 10 packets (in normal mode) or one burst of 495 packets (in heavy and manufacturing modes) and then stop.
9. Reset the PHY chip.
10. Write to the EMCR register on the IOC3 to enable DMA.
11. Wait for the IOC3 to post TX_EMPTY in the EISR. During the polling loop, monitor for unexpected error conditions. Time out if this takes too long.
12. Check the Bridge ASIC ISR to verify that the interrupt got from the IOC3 to the Bridge.
13. Wait for the valid bit to set in the RX buffer for the last packet. While waiting for this to occur, the test monitors for unexpected error conditions.
14. Time out if it takes too long for the valid bit to set in the RX buffer for the last packet.
15. Check all received packets: Check the status field for each packet and also verify that the data matches the data that was transmitted.
16. Clean up: Free the allocated memory, restore interrupt enables, and reset the PHY chip.

In normal mode, this test transmits and receives 10 packets; each packet contains 301 bytes of data. In heavy mode, this test transmits and receives 495 packets; each packet contains 1,501 bytes of data. This test does not run in manufacturing mode.

4.2.10.4 Failure Information

This test returns the same failure messages as the external loopback DMA test (refer to Section 4.2.9.4, “Failure Information,”).

4.2.11 External Loopback DMA (100 Mb/s) Test

The external loopback DMA (100 Mb / s) test is a version of the external loopback DMA test; the differences are that it does not test autonegotiation and it runs only at 100 Mb / s.

4.2.11.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address / data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

Also ensure that the Twister chip internal loopback test runs without errors before you run this test.

You should install a loopback cable or connector before you run this test.

4.2.11.2 How to Run This Test

This test does not run during the boot sequence. It runs only from the command line. To run this test from the >> prompt, enter **`diag_enet -t ext_loop_100 [`***options***`]`**. For example:

>> **`diag_enet -t ext_loop_100 [`***options***`]`**

Refer to Section 4.2 for descriptions of all available *options*.

4.2.11.3 Test Description

This test uses the following algorithm:

1. Allocate memory for the TX descriptor ring and write the base address to the ETBR registers on the IOC3.
2. Allocate memory for the TX data buffers and write their addresses to the descriptor ring entries and to the command and buffer count fields.
3. Generate test data and write it to the TX buffers. (The test pattern is designed to stress PCI SSO.)
4. Allocate memory for the RX buffer pointer ring and write the base address to the ERBR registers on the IOC3.
5. Allocate memory for the RX buffers.

6. Enable error interrupts on the Bridge ASIC.
7. Reset the IOC3 Ethernet DMA engines, initialize the IOC3 registers for external loopback DMA, and enable the TX_EMPTY interrupt.
8. Write to the ring pointers in a way that causes the DMA to occur in one burst of 10 packets (in normal mode) or one burst of 495 packets (in heavy and manufacturing modes) and then stop.
9. Reset the PHY chip.
10. Write to the EMCR register on the IOC3 to enable DMA.
11. Wait for the IOC3 to post TX_EMPTY in the EISR. During the polling loop, monitor for unexpected error conditions. Time out if this takes too long.
12. Check the Bridge ASIC ISR to verify that the interrupt went from the IOC3 to the Bridge.
13. Wait for the valid bit to set in the RX buffer for the last packet. While waiting for this to occur, the test monitors for unexpected error conditions.
14. Time out if it takes too long for the valid bit to set in the RX buffer for the last packet.
15. Check all received packets: Check the status field for each packet and also verify that the data matches the data that was transmitted.
16. Clean up: Free the allocated memory, restore interrupt enables, and reset the PHY chip.

In normal mode, this test transmits and receives 10 packets; each packet contains 301 bytes of data. In heavy mode, this test transmits and receives 495 packets; each packet contains 1,501 bytes of data. This test does not run in manufacturing mode.

4.2.11.4 Failure Information

This test returns the same failure messages as the external loopback DMA test (refer to Section 4.2.9.4, “Failure Information,”).

4.2.12 XTALK Stress Test

The XTALK stress test is a 100 Mb/s DMA test that uses a continuous stream of DMA for 1 to 5 seconds, instead of a single burst of 10 or 495 packets like the other DMA tests. This test does not check the received data, but it does check the IOC3 and Bridge interrupt registers for errors. This test uses a data pattern that is designed to stress SSO on the LLP links in the XTALK path.

4.2.12.1 Prerequisites to This Test

Ensure that the following hardware is functional before you run this test:

- Non-Ethernet-specific logic on the IOC3 ASIC
- PCI bus address/data
- Bridge ASIC
- XTALK interface between the Bridge ASIC and XBOW chip
- XBOW chip, midplane, and all IP27 board components

Also ensure that the Twister chip internal loopback test runs without errors before you run this test.

4.2.12.2 How to Run This Test

This test does not run as part of the boot sequence in normal mode; it does run during the boot sequence in heavy and manufacturing modes.

To run the XTALK stress test from the >> prompt, enter the **diag_enet -t xtalk** command and any other command line options that you want to use. For example:

>> **diag_enet -t xtalk [***options***]**

Refer to Section 4.2 for descriptions of all available *options*. If you use the **-e** option, this test uses the external loopback path instead of the Twister internal loopback. If you use the **-r** option, this test prints the retry counts every 50 seconds.

When you run this test manually from the command line, the command reads and clears out the LLP retry count registers on the HUB, XBOW, and Bridge before and after the test runs.

4.2.12.3 Test Description

This test uses the following algorithm:

1. Allocate memory for the TX descriptor ring and write the base address to the ETBR registers on the IOC3.
2. Allocate memory for the TX data buffers and write their addresses to the descriptor ring entries and to the command and buffer count fields.
3. Generate test data and write it to the TX buffers. (The test pattern is designed to stress LLP.)
4. Allocate memory for the RX buffer pointer ring and write the base address to the ERBR registers on the IOC3.
5. Allocate memory for the RX buffers.
6. Enable error interrupts on the Bridge ASIC.
7. Reset the IOC3 Ethernet DMA engines, initialize the IOC3 registers for Twister loopback DMA, and enable the TX_EMPTY interrupt.
8. Write to the ring pointers in a way that causes the DMA to occur in one burst of 10 packets (in normal mode) or one burst of 495 packets (in heavy and manufacturing modes) and then stop.
9. Reset the PHY chip and initialize it for Twister loopback at 100 Mb/s.
10. Write to the EMCR register on the IOC3 to enable DMA.
11. Check for errors in the IOC3 EISR and the Bridge ISR and chase the consume pointers. (In normal mode, repeat this step for 1 second. In heavy and manufacturing modes, repeat this step for 5 seconds.)
12. Wait for the IOC3 to post TX_EMPTY in the EISR. During the polling loop, monitor for unexpected error conditions. Time out if this takes too long.
13. Check the Bridge ASIC ISR to verify that the interrupt got from the IOC3 to the Bridge.
14. Wait for the valid bit to set in the RX buffer for the last packet. While waiting for this to occur, the test monitors for unexpected error conditions.
15. Time out if it takes too long for the valid bit to set in the RX buffer for the last packet.
16. Check all received packets: Check the status field for each packet and also verify that the data matches the data that was transmitted.
17. Clean up: Free the allocated memory, restore interrupt enables, and reset the PHY chip.

In normal mode, this test performs 1 second of DMA operations. In heavy and manufacturing modes, this test performs 5 seconds of DMA operations.

4.2.12.4 Failure Information

The failure messages are the same as the failure messages for the Twister chip internal loopback test (refer to Section 4.2.8.4, "Failure Information,"). This test also includes the following messages:

- `enet_xtalk_stress: RX CONSUME did not increment from Xtalk stress loop` *value* `to` *value*`.`
  `enet_xtalk_stress: TX CONSUME did not increment from Xtalk stress loop` *value* `to` *value*`.`

  The DMA hung. The failing hardware component is difficult to isolate.

- `enet_xtalk_stress: Unexpected EISR condition in Xtalk stress loop` *value*`.`
  `enet_xtalk_stress:  EISR value =` *e_value*

  The IOC3 detected an error during the DMA operation. The EISR value (shown in the output above as *e_value*) indicates the failing hardware. Refer to Table 4-1 for descriptions of the EISR values.

- `enet_xtalk_stress: Unexpected Bridge ISR condition in Xtalk stress loop` *value*`.`
  `enet_xtalk_stress:  Bridge ISR value =` *value*

  An unexpected Bridge ISR condition occurred. Most cases are difficult to isolate.

Appendix A

Node Board LED Values

This Appendix decodes all Node board LED values that the boot process and power-on diagnostics use.

The tables in this Appendix use the following symbols to indicate the state of the Node board LEDs:

= illuminated LED = 0
= unilluminated LED = 1

A.1 Progress LED Values

Table A-1 decodes the progress LED (PLED) values that appear on the Node board LEDs during the boot process. The “Failing hardware” information shown in the table indicates the hardware that is failing when a CPU hangs while displaying the specified LED pattern.

Table A-1 Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x00	PLED_RESET	System reset Failing hardware = Node board (CPU)

a. 
= Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x01	PLED_INITCPU	Initializing the R10000 general-purpose registers (GPRs), floating-point registers (FPR), and COP0 registers Failing hardware = Node board (CPU)
	0x02	PLED_TESTCP1	Performing the register test Failing hardware = Node board (CPU)
	0x03	PLED_RUNTLB	Switching to mapped mode Failing hardware = Node board (CPU)
	0x04	PLED_TESTICACHE	Performing the primary instruction cache test Failing hardware = Node board (CPU)

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x05	PLED_TESTDCACHE	Performing the primary data cache test Failing hardware = Node board (CPU)
	0x06	PLED_TESTSCACHE	Performing the secondary cache test Failing hardware = Node board (CPU)
	0x07	PLED_FLUSHCACHES	Flushing all caches Failing hardware = Node board (CPU)
	0x0a	PLED_INVICACHE	Invalidating R10000 primary instruction cache Failing hardware = Node board (CPU)

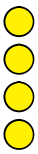
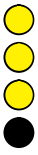
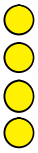
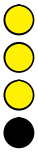
a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x0b	PLED_INVDCACHE	Invalidating R10000 primary data cache Failing hardware = Node board (CPU)
	0x0c	PLED_INVSCACHE	Invalidating secondary cache Failing hardware = Node board (Scache)
	0x0d	PLED_INMAIN	Transferred control to the PROM **main()** routine Failing hardware = Node board
	0x0e	PLED_SPEEDUP	Preparing to increase PROM access speed Failing hardware = Node board (PROM)

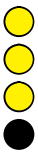
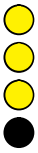
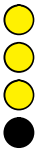
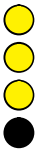
a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x0f	PLED_SPEEDUPOK	Successfully increased PROM access speed Failing hardware = Node board (PROM)
	0x10	PLED_INITDCACHE	Initializing primary data cache Failing hardware = Node board (CPU)
	0x11	PLED_INITICACHE	Initializing primary instruction cache Failing hardware = Node board (CPU)
	0x12	PLED_INITCOP0	Initializing R10000 COP0 registers Failing hardware = Node board (CPU)

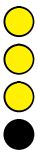
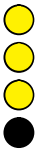
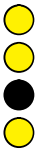
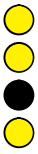
a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x13	PLED_FLUSHTLB	Flushing TLB Failing hardware = Node board (CPU)
	0x1a	PLED_ELSCPROBE	Preparing to probe for presence of MSC Failing hardware = MSC, Node board, or midplane
	0x1b	PLED_JUNKPROBE	Preparing to probe for the presence of JUNK UART Failing hardware = MSC, Node board, or midplane
	0x1c	PLED_DONEPROBE	Done probing for presence of MSC Failing hardware = MSC or UART

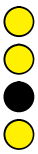
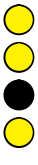
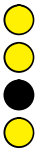
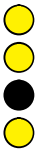
a.  = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x1d	PLED_UARTINIT	Preparing to initialize selected UART Failing hardware = MSC
	0x1e	PLED_UARTINITDONE	Done initializing selected UART Failing hardware = UART
	0x21	PLED_PODLOOP	Preparing to enter POD mode (C code portion) Failing hardware = Node board
	0x22	PLED_PODPROMPT	Preparing to enter the POD prompt loop Failing hardware = Node board

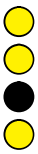
a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x23	PLED_PODMODE	Preparing to enter POD mode (assembly code portion) Failing hardware = Node board
	0x24	PLED_LOCALARB	Performing local arbitration (between CPU A and CPU B) Failing hardware = Node board (CPU)
	0x25	PLED_SCINIT	Initializing the secondary cache Failing hardware = Node board (Scache)
	0x28	PLED_BARRIER	Preparing to perform first local barrier Failing hardware = Node board (CPU or Hub)

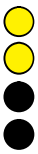
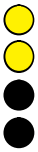
a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x2a	PLED_MAKESTACK	Preparing to configure Dex mode stack and data Failing hardware = Node board (CPU or Scache)
	0x2b	PLED_MAIN	Code execution has reached the **main()** function Failing hardware = Node board (CPU)
	0x31	PLED_NMI	NMI received Failing hardware = None
	0x35	PLED_RTCINIT	Preparing to initialize the HUB real-time counter Failing hardware = Node board

a.  = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x36	PLED_RTCINITDONE	Done initializing the HUB real-time counter Failing hardware = Node board
	0x38	PLED_BARRIEROK	Successfully completed the first local barrier operation Failing hardware = Node board
	0x3c	PLED_JUMPRAMU	Preparing to jump to UALIAS space Failing hardware = Node board (DIMM)
	0x3d	PLED_JUMPRAMUOK	Successfully jumped to UALIAS space Failing hardware = Node board (DIMM)

a.  = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x3e	PLED_JUMPRAMC	Preparing to jump to cached space Failing hardware = Node board (Scache)
	0x3f	PLED_JUMPRAMCOK	Successfully jumped to cached space Failing hardware = Node board (Scache)
	0x40	PLED_STACKRAM	Preparing to test the stack area of memory Failing hardware = Node board (DIMM)
	0x41	PLED_STACKRAMOK	Successfully tested the stack area of memory Failing hardware = Node board (Scache or DIMM)

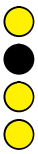
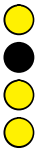
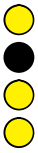
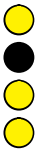
a.  = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x45	PLED_LAUNCHLOOP	Preparing to enter the slave launch loop Failing hardware = Node board (Master CPU)
	0x46	PLED_LAUNCHINTR	Received a launch interrupt Failing hardware = None
	0x47	PLED_LAUNCHCALL	Calling the **launched()** function Failing hardware = Node board (DIMM or Scache)
	0x48	PLED_LAUNCHDONE	Returned from the **launched()** function Failing hardware = Node board

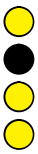
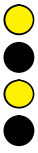
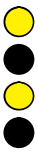
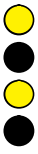
a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x4a	PLED_MDIRINIT	Preparing to initialize the HUB MD and SIMM controls Failing hardware = Node board (HUB)
	0x4b	PLED_MDIRCONFIG	Preparing to determine and configure the memory size Failing hardware = Node board (DIMM)
	0x4c	PLED_I2CINIT	Preparing to initialize the PCF8584 I2C chip Failing hardware = MSC or midplane
	0x4d	PLED_I2CDONE	Completed initializing the PCF8584 I2C chip Failing hardware = MSC or Node board

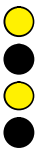
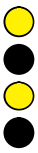
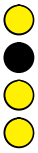
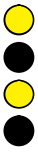
a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x4f	PLED_IODISCOVER	Preparing to discover HUB I/O Failing hardware = Node board (HUB)
	0x50	PLED_HUB_CONFIG	Updating configuration structures for HUBs and for Routers without HUBs Failing hardware = Any Node board
	0x51	PLED_ROUTER_CONFIG	Preparing to write the Router configuration data into the KLCONFIG structure Failing hardware = Any Router board
	0x52	PLED_INITIO	Preparing to initialize the I/O interface (II) on the HUB Failing hardware = Node board (HUB)

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x53	PLED_CONSOLE_GET	Preparing to probe the I/O section to locate the console Failing hardware = BaseIO board, I/O card (Bridge), or midplane (Crossbow)
	0x54	PLED_CONSOLE_GET_OK	Successfully located the console Failing hardware = BaseIO board
	0x56	PLED_INITIODONE	Successfully initialized the II on the HUB Failing hardware = BaseIO board, I/O card (Bridge), or midplane (Crossbow)
	0x57	PLED_STASH2	Preparing to clear the HUB error registers Failing hardware = Node board (HUB)

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x58	PLED_STASH3	Preparing to enable error checking Failing hardware = Node board (HUB)
	0x59	PLED_STASH4	Completed enabling error checking Failing hardware = Node board (HUB)
	0x5a	PLED_IODISCOVER_DONE	Completed discovering HUB I/O Failing Hardware = Node board (HUB)
	0x5b	PLED_NMI_INIT	Preparing to initialize the NMI handler area Failing hardware = Node board (DIMM or Scache)

a.  = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x5c	PLED_TEST_INTS	Preparing to test HUB interrupts Failing hardware = Node board (HUB)
	0x5d	PLED_IORESET	Performing early reset of HUB I/O section Failing hardware = Node board
	0x5e	PLED_RESET_OK	Completed resetting CPU Failing hardware = Node board
	0x5f	PLED_PRE_I2C_RESET	Preparing to reset I2C Failing hardware = Node board or midplane

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-1 (continued) Node Board Progress LED Values

Pattern [a]	Value	Name	Description
	0x60	PLED_POST_I2C_RESET	Completed resetting I2C Failing hardware = Node board

a.  = Illuminated LED = 0
= Unilluminated LED = 1

A.2 Failure LED Values

Table A-2 decodes the failure LED (FLED) values that appear on the Node board LEDs while the power-on diagnostics are running.The “Failing hardware” information shown in the table indicates the hardware that is failing when a CPU hangs while displaying the specified LED pattern.

Table A-2 Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0x81	FLED_CP1	Register test failed Failing hardware = Node board (CPU) Refer to Section 3.1.1, “Register Test,” for more information.
	0x82	FLED_RESTART	Restart master was unable to load the BaseIO PROM Failing hardware = Node board, BaseIO board, midplane, or XBOW
	0x83	FLED_ICACHE	Primary instruction cache test failed Failing hardware = Node board Refer to Section 3.1.2, “Primary Cache Tests,” for more information.

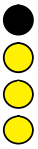
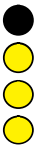
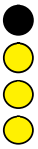
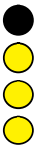
a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0x84	FLED_DCACHE	Primary data cache test failed Failing hardware = Node board Refer to Section 3.1.2, “Primary Cache Tests,” for more information.
	0x85	FLED_SCACHE	Secondary cache tests failed Failing hardware = Node board Refer to Section 3.2, “Secondary Cache Test,” for more information.
	0x86	FLED_KILLED	CPU was disabled by another CPU Failing hardware = Node board
	0x87	FLED_RTC	Real-time counter is not working correctly Failing hardware = Node board or midplane

a.  = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0x88	FLED_ECC	An ECC exception occurred Failing hardware = Node board (CPU)
	0x89	FLED_XTLBMISS	An XTLB miss exception occurred Failing hardware = Node board (CPU)
	0x8a	FLED_UTLBMISS	An UTLB miss exception occurred Failing hardware = Node board (CPU)
	0x8b	FLED_KTBLMISS	An KTBL miss exception occurred Failing hardware = Node board (CPU)

a.  = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0x8c	FLED_GENERAL	A general exception occurred Failing hardware = Node board (CPU)
	0x8d	FLED_NOIMPL	An unimplemented exception occurred Failing hardware = Node board (CPU)
	0x8e	FLED_CACHE	A cache error exception occurred Failing hardware = Node board (CPU)
	0x8f	FLED_OS	OS requested LEDs Failing hardware = None

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0x90	FLED_HUBINTS	Interrupt test failed Failing hardware = Node board (CPU) Refer to Section 3.1.3, “Interrupt Test,” for more information.
	0x91	FLED_HUBLOCAL	HUB local failed Failing hardware = Node board (HUB or CPU)
	0x92	FLED_HUBCONFIG	HUB configuration failed Failing hardware = Node board (HUB)
	0x93	FLED_PREM_DIR_REQ	Some Node does not have premium memory Failing hardware = Node board (DIMM) This error indicates one or more Node boards in a system with more than 32 processors does not have premium memory.

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0x97	FLED_MAINRET	Returned from **main()** function Failing hardware = None (failing software)
	0x98	FLED_NOMEM	Node Board does not have local memory This failure indicates that a Node does not have memory installed in Bank 0 or that the memory has been marked “disabled.”
	0x9a	FLED_DISABLED	CPU is disabled by an environment variable This failure indicates that a Node board has been marked as “disabled.”
	0x9b	FLED_DOWNLOAD	Memory download failed Failing hardware = Node board (DIMM or PROM)

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0x9e	FLED_HUB_CONFIG	Failure occurred while writing the HUB information into the KLCONFIG structure Failing hardware = Node board (DIMM)
	0x9f	FLED_ROUTER_CONFIG	Failure occurred while writing the Router information into the KLCONFIG structure Failing hardware = Node board (DIMM)
	0xa0	FLED_HUBIO_INIT	Failure occurred while trying to initialize the HUB I/O interface Failing hardware = Node board (HUB)
	0xa1	FLED_CONFIG_INIT	Failure occurred while trying to initialize the KLCONFIG structure Failing hardware = Node board (DIMM)

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0xa4	FLED_HUBBIST	HUB LBIST or HUB ABIST test failed Failing hardware = Node board (HUB)
	0xa5	FLED_RTRBIST	Router LBIST or Router ABIST test failed Failing hardware = Router board
	0xa6	FLED_RESETWAIT	Waiting for a reset Failing hardware = Node board (HUB)
	0xa7	FLED_LLP_FAIL	LLP failed after a reset Failing hardware = Node board (HUB)

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0xa8	FLED_LLP_NORESET	LLP did not come up after a reset Failing hardware = Node board (HUB)
	0xa9	FLED_BADMEM	Local memory is bad This failure indicates that a Node does not have memory installed in Bank 0 or that the memory has been marked “disabled.”
	0xab	FLED_NET_DISCOVER	Network discovery failed (misconfigured system) Failing hardware = Node board or Router board
	0xac	FLED_NASID_CALC	Calculation of NASID failed Failing hardware = CrayLink cable or connection

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0xad	FLED_ROUTE_CALC	Calculation of route failed Failing hardware = CrayLink cable or connection
	0xae	FLED_ROUTE_DIST	Distribution of route failed Failing hardware = Router board, Node board, internal link (router to router or router to Node board), CrayLink cable, or CrayLink connection
	0xaf	FLED_NASID_DIST	Distribution of NASID failed Failing hardware = Router board, Node board, internal link (router to router or router to Node board), CrayLink cable, or CrayLink connection
	0xb0	FLED_NO_NASID	Master was not assigned a NASID Failing hardware = Router board, Node board, internal link (router to router or router to Node board), CrayLink cable, or CrayLink connection

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0xb1	FLED_NO_MODULEID	Module ID arbitration failed Failing hardware = MSC
	0xb2	FLED_MIXED_SN00	SN0 mixed with SN00 This failure indicates an illegal configuration that includes a CrayLink between an Origin200 system and an Origin2000 system, which is not supported.
	0xb3	FLED_ERRPART	Partition configuration error This failure indicates an illegal partition configuration.

a. = Illuminated LED = 0
= Unilluminated LED = 1

Table A-2 (continued) Node Board Failure LED Values

Pattern [a]	Value	Name	Description
	0xb4	FLED_MODEBIT	An error occurred while copying the mode bits Failing hardware = Node board
	0xb5	FLED_BACK_CALC	An error occurred while copying the backplane frequency Failing hardware = Node board or Router board

a.  = Illuminated LED = 0
= Unilluminated LED = 1

A.3 Exception LED Values

Table A-3 shows the LED values that *flash* on the Node board LEDs when an exception occurs before a tty port is available to receive information about the exception.

If the LEDs on a Node board display any of these patterns, disable or replace the Node board because it contains failing hardware.

Table A-3 LED Status Values for Exceptions

Pattern [a]	Value	Name	Exception Type
	0xf2	EXC_GENERAL	General exception Failing hardware = Node board (CPU)
	0xf3	EXC_ECC	ECC exception Failing hardware = Node board (CPU)
	0xf4	EXC_TLB	TLB exception Failing hardware = Node board (CPU)

a.  = Illuminated LED = 0
= Unilluminated LED = 1

Table A-3 (continued) LED Status Values for Exceptions

Pattern [a]	Value	Name	Exception Type
	0xf5	EXC_XTLB	XTLB exception Failing hardware = Node board (CPU)
	0xf6	EXC_UNIMPL	Unimplemented exception Failing hardware = Node board (CPU)
	0xf7	EXC_CACHE	Cache error exception Failing hardware = Node board (CPU)

a. = Illuminated LED = 0
= Unilluminated LED = 1

Appendix B

diag_rc Values

This Appendix decodes all of the diag_rc values that the power-on diagnostics return.

Table B-1 diag_rc Values

Value	Error Description	FRU
0	No error occurred. The test completed successfully.	None
	L1 and L2 Cache Error Codes	
1	Data cache data test failed.	Node board
2	Data cache data test failed.	Node board
3	Data cache address test failed.	Node board
4	Data cache address test failed.	Node board
5	Secondary cache data test failed.	Node board
6	Secondary cache data test failed.	Node board
7	Secondary cache address test failed.	Node board
8	Secondary cache address test failed.	Node board
9	Instruction cache data test failed.	Node board
10	Instruction cache data test failed.	Node board
11	Instruction cache address test failed.	Node board
12	Instruction cache address test failed.	Node board
13	Data cache test is hung.	Node board
14	Secondary cache test is hung.	Node board
15	Instruction cache test is hung.	Node board
16	Cache initialization failed.	Node board
17	Data cache tag RAM test failed.	Node board
18	Secondary cache tag RAM test failed.	Node board
19	Secondary cache tag RAM test failed.	Node board
20	FPU secondary cache tag RAM test failed.	Node board

Table B-1 (continued) diag_rc Values

Value	Error Description	FRU
21	Starting the data cache test.	Node board
22	Starting the instruction cache test.	Node board
23	Starting the secondary cache test.	Node board
24	Invalidating the instruction and data caches.	Node board
25	Invalidating the instruction and data caches.	Node board
26	Data cache test failed (generic failure).	Node board
27	Instruction cache test failed (generic failure).	Node board
28 - 29	Not used	None
	CPU Error Codes	
30	The other CPU on this Node failed.	Node board (CPU)
31	The CPU is disabled.	Node board (CPU)
32	The CPU failed in **earlyinit**.	Node board (CPU)
33	One of the CPUs took an unexpected exception.	Node board (CPU)
34	Coprocessor1 is dead.	Node board (CPU)
35	CPU died before or during local arbitration.	Node board (CPU)
36	The CPU and PROM mode bits do not match.	Node board (CPU)
37 - 39	Not used	None
	Memory Error Codes	
40	The checksum in the PROM is bad.	Node board (PROM)
41	The copy of the PROM in memory is bad.	Node board (PROM)
42	The copy of PROM to memory failed.	Node board (PROM)
43	The memory test (memtest) failed for one or more banks.	Node board or DIMM
44	The directory memory test (dirtest) failed for one or more banks.	Node board or DIMM
45	Memory is disabled by the user.	Node board
46	Bank 0 is unusable.	Node board
47 - 49	Not used	None

Table B-1 (continued) diag_rc Values

Value	Error Description	FRU
	IO6 Error Codes	
50	XBOW sanity on the IO6 failed.	BaseIO board (XBOW)
51	Bridge sanity on the IO6 failed.	BaseIO board
52	The IOC3 base address could not be found.	BaseIO board
53	The configuration space on the IO6 failed.	BaseIO board
54	The PCI bus test failed.	BaseIO board
55	Characters already exist in port A.	BaseIO board
56	Characters already exist in port B.	BaseIO board
57	Unable to transmit characters on port A.	BaseIO board
58	Unable to transmit characters on port B.	BaseIO board
59	Miscompares occurred on port A.	BaseIO board
60	Miscompares occurred on port B.	BaseIO board
61	Miscompares occurred on port A in DMA mode.	BaseIO board
62	Miscompares occurred on port B in DMA mode.	BaseIO board
63	There was a SCSI mailbox failure.	BaseIO board
64	Unable to load words into the SSRAM.	BaseIO board
65	Unable to read words out of the SSRAM.	BaseIO board
66	There were miscompares on the SCSI SSRAM.	BaseIO board
67	Unable to load words into the SSRAM.	BaseIO board
68	Unable to dump words from the SSRAM.	BaseIO board
69	There were DMA miscompares from the SSRAM.	BaseIO board
70	Unable to load the self-test firmware.	BaseIO board
71	Unable to execute the SCSI self-test.	BaseIO board
72	The SCSI self-test failed with miscompares.	BaseIO board
73 - 79	Not used	None
80	There was an XBOW exception.	BaseIO board
81	There was a Bridge exception.	BaseIO board
82	There was an IO6 configuration exception.	BaseIO board
83	There was a PCI bus exception.	BaseIO board
84	There was a serial programmed I/O (PIO) exception	BaseIO board

Table B-1 (continued) diag_rc Values

Value	Error Description	FRU
85	There was serial DMA exception.	BaseIO board
86	There was a SCSI RAM exception.	BaseIO board
87	There was a SCSI DMA exception.	BaseIO board
88	There was a SCSI controller exception.	BaseIO board
89 - 100	Not used	BaseIO board
101	ENET_EXCEPTION	BaseIO board
102	ENET_SSRAM_FAIL	BaseIO board
103	ENET_PHYREG_FAIL	BaseIO board
104	ENET_PHY_R_BUSY_TO	BaseIO board
105	ENET_PHY_W_BUSY_TO	BaseIO board
106	ENET_PHY_RESET_TO	BaseIO board
107	ENET_NO_TXCLK	BaseIO board
108	ENET_LOOP_ILL_PARM	BaseIO board
109	ENET_LOOP_MALLOC	BaseIO board
110	ENET_EMCR_RST_TO	BaseIO board
111	ENET_EMCR_RXEN_TO	BaseIO board
112	ENET_LOOP_AN_TO	BaseIO board
113	ENET_LOOP_AN_ABLE	BaseIO board
114	ENET_LOOP_AN_MODE	BaseIO board
115	ENET_LOOP_DMA_TO1	BaseIO board
116	ENET_LOOP_DMA_TO2	BaseIO board
117	ENET_LOOP_DMA_TO3	BaseIO board
118	ENET_LOOP_BAD_RUPT1	BaseIO board
119	ENET_LOOP_BAD_RUPT2	BaseIO board
120	ENET_LOOP_BAD_RUPT3	BaseIO board
121	ENET_LOOP_BAD_RUPT4	BaseIO board
122	ENET_LOOP_NO_RUPT	BaseIO board
123	ENET_LOOP_BAD_W1	BaseIO board
124	ENET_LOOP_BAD_STAT	BaseIO board
125	ENET_LOOP_BAD_DATA1	BaseIO board
126	ENET_LOOP_BAD_DATA2	BaseIO board

Table B-1 (continued) diag_rc Values

Value	Error Description	FRU
127	ENET_NIC_FAIL	BaseIO board
128 - 149	Not used	None
150	There were too many link errors in the HUB NI.	Node board, Router board, midplane, or CrayLink cable
151	There were too many miscellaneous errors from the Router in the HUB NI.	Node board, Router board, or midplane
152	The HUB NI did not detect forced errors.	Node board
153	The HUB chip failed the lbist test.	Node board
154	The HUB chip failed the abist test.	Node board
155	The LLP interface on the HUB chip is down.	Node board
156	The HUB could not launch discovery.	Node board
157	The interrupt test failed for the HUB chip.	Node board
158	There is an NI error in the kernel.	Node board
159	The BTE test failed for the HUB chip.	Node board
160	The BTE for the HUB chip hung while diagnostics were running.	Node board
161	The HUB has a real-time clock error.	Node board
162	There was a reset propagation error.	Node board
163 - 199	Not used	None
	Router Error Codes	
200	The HUB did not get a response from a Router.	Node board, Router board, or midplane
201	The HUB received an overrun / mismatch reply from a Router.	Node board, Router board, or midplane
202	The HUB encountered a hardware error while accessing a Router.	Node board, Router board, or midplane
203	The HUB received a protection error from a Router.	Node board, Router board, or midplane
204	The HUB received an invalid vector error from a Router.	Node board, Router board, or midplane
205	The Router did not detect forced errors.	Node board, Router board, or midplane
206	The Router detected too many link errors.	Node board, Router board, or midplane

Table B-1 (continued) diag_rc Values

Value	Error Description	FRU
207	The Router failed the lbist test.	Node board, Router board, or midplane
208	The Router failed the abist test.	Node board, Router board, or midplane
209	FRU analysis for the local Node is available.	Node board, Router board, or midplane
210	FRU analysis for all Nodes is available.	Node board, Router board, or midplane
211	RTC distribution failed.	Node board, Router board, or midplane
212 - 239	Not used	None
	Generic Error Codes	
240	No CPU is available.	Anywhere
241	Returning to POD mode.	Anywhere
242	A PROM panic occurred.	Anywhere
243	A PROM panic occurred.	Anywhere
244	A PROM panic occurred.	Anywhere
245	A PROM panic occurred.	Anywhere
246	A PROM panic occurred.	Anywhere
247	A PROM panic occurred.	Anywhere
248	An NMI occurred.	Anywhere
249	An NMI occurred.	Anywhere
250	An NMI occurred.	Anywhere
251	An NMI occurred.	Anywhere
252	An NMI occurred.	Anywhere
253	Going to POD mode at request of user.	Anywhere

Glossary

cached (Cac) mode

The mode in which the power-on diagnostics place program code, data, and stack into L2 cache memory. Cac mode is available only after memory has been probed and configured. POD execution is much quicker in Cac mode than in Dex mode or Unc mode. *See also* dirty exclusive (Dex) mode and uncached (Unc) mode.

dirty exclusive (Dex) mode

The mode in which the power-on diagnostics require the fewest system resources to run. Dex mode does not require memory; it accesses program code directly from the PROM and uses the data cache in the R10000 processors as the memory for its stack. Dex mode does not use the secondary cache. Nonmaskable interrupts and uncaught exceptions typically return you to the Dex mode prompt.

Dex mode is very slow because PROM instruction fetches are slow. Avoid performing long memory tests or flashing remote PROMs from Dex mode. *See also* cached (Cac) mode and uncached (Unc) mode.

headless Node

A Node without functional CPUs.

NASID value

A number that indicates the Numa Address Space IDentifier. A NASID value is a unique number that specifies a Node in the system.

nonmaskable interrupt (NMI)

An interrupt that forces a CPU to stop processing and jump to a fixed address in the PROM. The CPU then runs the code, called the NMI handler, which is located at that address. Nonmaskable interrupts are used to debug a system that is hung (usually because of a PROM or Kernel bug).

You can issue an NMI by pressing the NMI button on the MSC or by executing the MSC *nmi* command.

NMI handler

The code at a fixed address in the PROM that the CPU accesses when it receives a nonmaskable interrupt (NMI). This code attempts to bring the system to a PROM or kernel monitor so you can examine the state of the system and determine why it hung. This code also dumps registers and other information to disk.

number in a can (NIC)

A Dallas Semiconductor chip that contains a 48-bit number that is permanently burned in by a laser when the chip is manufactured. The 48-bit number is different for each board in the system and provides a way for software to determine which boards are in the system.

An NIC also contains several pages of nonvolatile memory that the system can read and write. This memory contains the following information: board type, board revision number, board serial number, and board-specific information.

PROM environment list

A group of PROM environment variable values that are stored under one name.

PROM environment variable

A variable stored in the PROM log that changes the way in which the PROM environment operates.

R10000 processor

The MIPS processor used on the IP27 board. This processor is also called the T5.

T5

The internal code name for the R10000 processor. Some diagnostics still refer to R10000 processors as T5 processors.

uncached (Unc) mode

The mode in which power-on diagnostics place their program code, data, and stack into main memory. This mode is similar to Cac mode, but Unc mode runs slower; however, program execution is quicker in Unc mode than in Dex mode. *See also* cached (Cac) mode and dirty exclusive (Dex) mode.

Widget

A generic name for any device that can connect to the PCI interface.