Origin™ 200/2000 and Onyx2™ MDK-Based Field Diagnostics

Document Number 108-0162-004

Contributors

Revised by Jason Hatcher
Production by Carlos Miqueo
Edited by Cindi Leiser
Engineering contributions by Judy Young

Origin™ 200/2000 and Onyx2™ MDK-Based Field Diagnostics
Document Number 108-0162-004

SGI
Mountain View, California

Contents

Figures

Tables

About This Guide

This document describes the micro-diagnostic kernel (MDK) and the MDK-based diagnostics that you can use to troubleshoot Origin 200, Origin 2000, and Onyx2 computer systems. System Support Engineers (SSEs) and Field Engineers (FEs) may use this reference document during training and on-site.

The document organizes information into the following chapters:

- Chapter 1, “Introduction to MDK,” provides an overview of MDK and the diagnostic tests that you can run from MDK.
- Chapter 2, “R10000/R12000 Processor Tests,” describes the diagnostic tests that you can use to test the R10000/R12000 processors.
- Chapter 3, “Secondary Cache Test,” describes the diagnostic test that you can use to test the secondary caches on the Node boards.
- Chapter 4, “Memory Tests,” describes the diagnostic tests that you can use to test memory.
- Chapter 5, “Router SSO Test,” describes the diagnostic test that you can use to test Router and MetaRouter links.
- Appendix A, “Troubleshooting Link Failures,” provides general information to help you isolate link failures.

This document supports MDK version 1.6, which is included in the *Internal Support Tools 2.0* CD-based diagnostic release.

There are two methods to determine the version of MDK that you are using:

- Look at the MDK start-up banner:

```
**********************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
**********************************************************
```

- Use the *versions -b mdk* command at an IRIX prompt:

```
bitplane 3# versions -b mdk
I = Installed, R = Removed
Name           Date      Description
I  mdk         07/16/98  MDK Diagnostics, 1.25 for Origin and Onyx2
```

Typographical Conventions

This document uses the following typographical conventions and symbols:

- Information displayed on the screen is shown in `Courier` type.
- Commands that you should enter are shown in **`Courier bold`** type.
- Commands in text and filenames are shown in *italic* type.
- Variables are shown in *italic* type.
- The >> symbol indicates the PROM monitor (or BaseIO command) prompt.
- The `MDK>` symbol indicates the MDK prompt.

Terminology

This document uses the following terms interchangeably:

- HUB and HUB chip
- Crossbow, XBOW, and XBOW chip
- Diagnostic, diagnostic test, and test

Chapter 1

Introduction to MDK

This chapter describes the micro-diagnostic kernel (MDK) and provides an overview of the MDK-based diagnostics that you can use to troubleshoot Origin 200, Origin 2000, and Onyx2 systems.

1.1 About the Micro-Diagnostic Kernel

The micro-diagnostic kernel (MDK) is a standalone diagnostic environment that runs in kernel mode on Origin 200, Origin 2000, and Onyx2 systems. MDK lets you load and run diagnostic tests in an environment that provides access to hardware components that cannot be accessed from a user program that is running under the IRIX operating system.

MDK-based diagnostics require full use of the system: you must bring down (reboot) the system to boot MDK. Therefore, no other kernel (that is, the IRIX operating system) can be running when you use the MDK-based diagnostics. You should use the MDK-based diagnostics to isolate failures for which the IRIX based diagnostics do not provide enough error information or to test hardware that the IRIX based diagnostics cannot test.

Note: MDK does not support partitioned systems.

MDK provides features that enable MDK-based diagnostic tests to operate like normal user programs; these features include memory allocation, console output, exception handling, and multiprocessing. MDK also provides system configuration information (memory size and number of CPUs, Nodes, and Routers) for the diagnostic tests.

This document describes MDK version 1.6, which includes three test packages: *cpu.mdk*, *memory.mdk*, and *MDK_router_sso*. Each test package binary includes the micro-diagnostic kernel and a set of diagnostic tests. (Because MDK does not support a filesystem, it is not possible to load individual tests from the disk.) The following test packages are available:

- The *cpu.mdk* test package contains MDK, the *instgen* test, and the *pcache* test.
- The *memory.mdk* test package contains the micro-diagnostic kernel, the secondary cache test, and several memory tests.
- The *MDK_router_sso* test package contains MDK and the Router SSO test.

A major advantage of test packages is that you do not need to reboot the system to run another test from the same test package; you simply start the second test from the `MDK>` prompt that appears after the first test completes execution. However, you do need to reboot the system to run a test in a different test package.

Note: Earlier versions of MDK were called Nanos, so you may see references to Nanos in the output from the MDK-based tests.

MDK-based tests were developed in two modes: mapped and unmapped.

1.1.1 Mapped Mode

In mapped mode, MDK provides memory allocation and uses the translation lookaside buffer (TLB) for mapping virtual addresses to physical memory addresses. Mapped tests are limited to 1 Gbyte of user address space.

1.1.2 Unmapped Mode

MDK-based tests that run in unmapped mode do not use the TLB, so these tests are not limited to 1 Gbyte of user address space. Unmapped tests manage the physical memory themselves.

1.2 Diagnostic Tests

Most of the MDK-based diagnostic tests are directed tests, which focus on one area of the system. The areas tested include the R10000/R12000 processors, secondary cache, memory, and the Router boards.

1.2.1 R10000/R12000 Processor Tests

The R10000/R12000 processor tests are directed tests that focus on the components within each R10000/R12000 processor. These tests check the following areas:

- The functionality of the external interface to each R10000/R12000 processor
- The arithmetic logic unit (ALU) in each R10000/R12000 processor
- The floating-point unit (FPU) in each R10000/R12000 processor
- The primary cache in each R10000/R12000 processor
- The instruction pipeline in each R10000/R12000 processor
- The register-renaming functionality of each R10000/R12000 processor

If an R10000/R12000 processor test fails, the failing hardware is the processor in which the test detected the errors. The failing field replaceable unit (FRU) is the Node board that contains the failing R10000/R12000 processor.

To run the R10000/R12000 processor tests, you must load the *cpu.mdk* test package.

Refer to Chapter 2, "R10000/R12000 Processor Tests," for more information.

1.2.2 Secondary Cache Test

The secondary cache test verifies the functionality of the secondary cache located on each Node board.

If this test fails, the failing hardware component is the secondary cache or memory. The failing FRU is the Node board that contains the failing secondary cache or the failing memory dual inline memory module (DIMM).

To run the secondary cache test, you must load the *memory.mdk* test package.

Refer to Chapter 3, "Secondary Cache Test," for more information.

1.2.3 Memory Tests

The memory tests verify that memory actually contains data that was written to it. The memory tests range from testing memory with one or two CPUs to testing memory with all Nodes and CPUs in the system.

If a memory test fails, the failing hardware is one or more memory DIMMs, a directory memory DIMM, the secondary cache (HIMM), a Node board, a midplane, or a Router board. The failing FRU is either a DIMM, Node board, midplane, or Router board.

To run the memory tests, you must load the *memory.mdk* test package.

Refer to Chapter 4, “Memory Tests,” for more information.

1.2.4 Router SSO Test

The Router SSO test checks the Router links.

If this test detects a Router failure, the failing hardware is a compression (CPOP) connector, a terminator, or a Router chip. The failing FRU is a Node board, midplane, Router board, or connecting cable. If this test detects a HUB failure, the failing FRU is a Node board, Router board, or midplane.

Note: Before you replace any FRUs, reseat the suspected boards and reconnect the suspected cables, and then run the test again. If the test still fails, you will need to replace one or more of the FRUs.

To run the Router SSO test, you must load the *MDK_router_sso* test package.

Refer to Chapter 5, “Router SSO Test,” for more information.

1.3 Installing MDK

The *Internal Support Tools 2.0* CD-based diagnostic release includes version 1.6 of MDK and the MDK-based diagnostics. Refer to the instructions included in this release for information about how to install the MDK software in the */stand* directory on your system disk.

Once the MDK software is installed on your system disk, you can use the *boot* command to load MDK into the system that you want to test. Refer to Table 1-1 and the individual diagnostic descriptions later in this document for the specific commands used to load each diagnostic.

You can use the ***bootp*** command to run MDK from another system on the network. To do this, perform the following steps:

- On the bootp server, copy the *cpu.mdk, memory.mdk, and MDK_router_sso* binaries to */usr/local/boot*.
- On the client, boot MDK with the following command:

  >>bootp()server_name_or_IP_address:mdk_test_package

  (for example, *bootp()hui.csd.sgi.com:memory.mdk*)
- At the `MDK>` prompt that appears, enter the name of the MDK diagnostic that you want to run.

Any future updates to the MDK-based diagnostics will be available from the Internal Support Tools Web site (http://ist.csd.sgi.com) or on future CD releases.

1.4 Commands Available at the MDK Prompt

When you start one of the MDK packages from the PROM monitor (at the BaseIO command prompt, >>), the `MDK>` prompt appears. From this prompt, you can enter commands that control MDK or commands that run diagnostic tests.

1.4.1 Commands for Controlling MDK

1.4.1.1 help Command

The *help* command prints a description of the command that you specify or prints a list of available commands if you do not specify a command. This command uses the following syntax:

`help [`*command*`]`

1.4.1.2 kill Command

The *kill* command terminates the process that you specify. This command uses the following syntax:

`kill` *pid*

Use the *pid* variable to specify the process ID (PID) of the process that you want to terminate.

When a diagnostic process takes an unexpected exception, MDK places the process in a suspended, or frozen, state. Use the *kill* command to terminate this "frozen" process before you run any other diagnostic tests. (Use the *ps* command to determine the PID for the frozen process.)

1.4.1.3 killall Command

The *killall* command terminates all active diagnostic processes. (Using this command is equivalent to using the *kill* command with the PID from every active test.) This command uses the following syntax:

```
killall
```

1.4.1.4 ls Command

The *ls* command lists the names of all available diagnostic tests in the MDK test package that is currently loaded. This command uses the following syntax:

```
ls
```

For example, when the *memory.mdk* package is loaded, the *ls* command returns the following information:

```
MDK> ls

scache
memum.nb
memum.lo
memum.in
memum.dr
memum.qs
mem.lo
mem.qs

MDK>
```

1.4.1.5 ps Command

The *ps* command returns process status information, including the PID value, for all active diagnostic processes. This command uses the following syntax:

```
ps
```

1.4.1.6 reset Command

The *reset* command performs a soft reset of the system. (Using this command is equivalent to entering the *reset* command from the PROM monitor [at the BaseIO command prompt, >>].) This command uses the following syntax:

```
reset
```

1.4.2 Commands for Loading a Diagnostic Test

To run an MDK-based diagnostic test, enter the name of the diagnostic test at the `MDK>` prompt. When the test completes, the `MDK>` prompt reappears.

Remember, you can run only the diagnostic tests that are part of the currently loaded MDK package (*cpu.mdk*, *memory.mdk*, or *MDK_router_sso*). Table 1-1 lists the commands that are used to run each of the diagnostic tests.

1.5 Diagnostic Test Output

All MDK-based diagnostics return output to the BaseIO console (the console that connects to the BaseIO board). The diagnostics print pass and fail messages as they run; all MDK-based diagnostics print a *testname* `test has completed` message when they complete testing.

1.5.1 Output from Successful Completion

When an MDK-based diagnostic completes successfully, the diagnostic displays one or more messages that indicate that it completed without errors.

For example, the Router SSO test returns the following messages when it completes testing without detecting any hardware failures:

```
TEST RESULT: **** TEST PASSED  ****
MDK Router_sso test run is complete.
```

1.5.2 Output When a Test Detects a Failure

When an MDK-based diagnostic detects a hardware failure, it prints a failure message and any information related to the failure.

For example, the `ERROR: Test Failed` message in the following output is from a test that detected a hardware failure:

```
NODE0 CPU1 Pid 2 FPU3 Test Started
NODE0 CPU0 Pid 1 FPU3 Test Started
NODE0 CPU1 Pid 2 FPU3 Test Passed
NODE0 CPU0 Pid 1 ERROR: Test Failed
```

1.5.3 Output When an Unexpected Exception Occurs

When an unexpected exception occurs, MDK prints the contents of the Coprocessor 0 registers, the general-purpose registers, and the following information about the exception: the type of exception, the address where the exception occurred, and any other information that is relevant to the exception.

After MDK prints this information, the `MDK>` prompt appears. At this point, you can load another diagnostic test, reset the system, or issue a nonmaskable interrupt (NMI).

The following example shows MDK output from an unexpected exception:

```
ERROR: Unexpected Exception Occured.
 Nasid 1: Local CPU A: Global CPU 2: PID 3: Multiple exceptions at
 0xa800000000a0c4e8: sr = 0xb4007fe6: cause = 0x881c
 Nanos: Frozen 22A 001:
 Status: 0xffffffffb4007fe6        XCtxt: 0xffffffe000002a30
 Hi:     0x               0        Lo:    0x               0
 epc:    0xa800000000a0c4e8        cause: 0x            881c
 count:  0x        18afb669        comp:  0xffffffffffffffff
 Enhi:   0x               0        CacErr: 0xffffffffd442aa80
 R01/AT: 0x           1ff00        R02/V0: 0xffffffffb4007fe6
 R03/V1: 0x          a257b0        R04/A0: 0xa800000000a27000
 R05/A1: 0x          a1dd38        R06/A2: 0x        8bfdfdf0
 R07/A3: 0x               8        R08/A4: 0x               4
 R09/A5: 0xa800000183fdf978        R10/A6: 0x        3e17aad8
 R11/A7: 0xa800000300caaad8        R12/T0: 0x              38
 R13/T1: 0x               0        R14/T2: 0x               0
 R15/T3: 0xa800000300caaa98        R16/S0: 0xa800000183fdf978
 R17/S1: 0xa8000000005fbdc8        R18/S2: 0x               0
 R19/S3: 0x               0        R20/S4: 0x               2
 R21/S5: 0xa800000008000000        R22/S6: 0x               3
 R23/S7: 0x              e0        R24/T8: 0x         8000000
 R25/T9: 0xa800000000a07ed8        R28/GP: 0x               0
 R29/SP: 0xa800000183fdd928        R30/S8: 0xa800000000a27000
 R31/RA: 0xa800000000a08eac
 Nanos: Frozen 3
```

1.6 Debugging Tips

If a hang or unexpected exception occurs while you are running MDK-based diagnostics, use the following POD mode commands to gather more information that you can use to isolate the problem:

- *nmi*, which issues a nonmaskable interrupt (NMI)
- *why*, which displays the current exception and NMI status information
- *error*, which displays error information from the MD section of the HUB
- *error_dump*, which displays detailed information about all error bits in the HUB
- *dumpspool*, which dumps a CPU's PI error spool
- *pr*, which prints the contents of an R10000/R12000 or HUB register
- *crb*, which dumps the I/O interface (II) CRBs
- *crbx*, which dumps the II CRBs in a 133-column-wide format

Refer to the *IP27PROM Technical Reference Manual*, publication number 108-0170-001, for more information about these commands.

1.7 Quick-Reference List of MDK-Based Diagnostics

Table 1-1 provides quick-reference information about the MDK-based tests. This information includes the following items:

- the functional unit that each test checks
- the name of each test
- the commands that you use to invoke each test from the PROM monitor (at the BaseIO command prompt, >>)
- the failure message for each test
- the possible failing FRUs for each test
- the page in this document on which you can find more information about each test

Table 1-1 Quick-Reference List of MDK-Based Diagnostic Tests

Functional Unit	Test	Invocation	Failure Message	Failing FRU	Page
R10000 / R12000 processors	*instgen*	>>**boot dksc(0,1,0)/stand/cpu.mdk** MDK>**instgen**	ERROR: PASS: *number*	Node board	2-2
	pcache	>>**boot dksc(0,1,0)/stand/cpu.mdk** MDK>**pcache**	ERROR: Test Failed	Node board	2-6
Secondary cache	*scache*	>>**boot dksc(0,1,0)/stand/memory.mdk** MDK>**scache**	Scache data path test FAILED Scache cell test FAILED Scache random address test FAILED	Node board	3-1
Memory	*mem.dr*	>>**boot dksc(0,1,0)/stand/memory.mdk** MDK>**mem.dr**	NODE*x* CPU*y* PID*z* *testname* test FAILED	Directory memory DIMM	4-13
	mem.qs	>>**boot dksc(0,1,0)/stand/memory.mdk** MDK>**mem.qs**	NODE*x* CPU*y* PID*z* *testname* test FAILED	DIMM, Node board, midplane, or Router board	4-13
	memum.qs	>>**boot dksc(0,1,0)/stand/memory.mdk** MDK>**memum.qs**	NODE*x* CPU*y* PID*z* *testname* test FAILED	DIMM, Node board, midplane, or Router board	4-13
	memum.nb	>>**boot dksc(0,1,0)/stand/memory.mdk** MDK>**memum.nb**	NODE*x* CPU*y* PID*z* *testname* test FAILED	DIMM, midplane, or Node board	4-22
	memum.in	>>**boot dksc(0,1,0)/stand/memory.mdk** MDK>**memum.in**	NODE*x* CPU*y* PID*z* *testname* test FAILED	DIMM, Node board, midplane, or Router board	4-28
	mem.lo	>>**boot dksc(0,1,0)/stand/memory.mdk** MDK>**mem.lo**	NODE*x* CPU*y* PID*z* *testname* test FAILED	DIMM, Node board, midplane, or Router board	4-33
	memum.lo	>>**boot dksc(0,1,0)/stand/memory.mdk** MDK>**memum.lo**	NODE*x* CPU*y* PID*z* *testname* test FAILED	DIMM, Node board, midplane, or Router board	4-33
Router links	*router*	>>**boot dksc(0,1,0)/stand/MDK_router_sso** MDK>**router**	ERROR, TEST RESULT: **** WARNING, or TEST RESULT: ****FAIL	Router board, Node board, midplane, or cable	5-1

Chapter 2

R10000/R12000 Processor Tests

This chapter describes the diagnostics that you can use to test the R10000/R12000 processors. Two new diagnostics are available for you to test the R10000/R12000 processors. These new tests replace the old ones (*t5_1* and *t5_2*). One of the new tests is a CPU stress test, which generates a much broader set of testing conditions than the old set of R10000/R12000 processor tests. The other test is a primary data cache test, which is a directed test.

2.1 About the R10000/R12000 Processor Tests

The R10000/R12000 processor tests are tests that focus on the components within the R10000/R12000 processors.

The first R10000/R12000 processor test, *instgen*, is a random instruction generator. The second R10000/R12000 processor test, *pcache*, is a directed test that writes simultaneous switching data patterns to the primary data cache and reads back the data to try to cause ECC errors. To run these diagnostic tests, you must load the *cpu.mdk* test package. (Table 2-1 provides quick-reference descriptions of these tests. The number in the Page column indicates the page on which a detailed description of each test begins.)

Table 2-1 R10000/R12000 Processor Tests

Test	Description	Page
instgen	Tests the instruction pipeline, the arithmetic logic unit (ALU), the floating-point unit (FPU), branch conditions, the primary instruction cache, the register renaming funcionality, and the functionality of the external interface of the R10000/R12000 processor.	2-2
pcache	Tests the primary data cache in the R10000/R12000 processor.	2-6

If either of these tests fails, the failing FRU is the Node board that contains the failing R10000/R12000 processor, with the following exception: The tests could encounter an uncorrectable memory error while attempting to access data from secondary cache or memory; if this occurs, run the secondary cache test or memory tests to check for errors.

2.2 instgen Test

The *instgen* test tests the instruction pipeline, the arithmetic logic unit (ALU), the floating-point unit (FPU), branch conditions, the primary instruction cache, the register renaming functionality, and the functionality of the external interface of the R10000/R12000 processor.

2.2.1 How to Run the instgen Test

Use the following procedure to load the test from the local system disk:

- Use the *boot* command from the PROM monitor (at the BaseIO command prompt, >>) to start the cpu-mdk test package:

  ```
  >>boot dksc(0,1,0)/stand/cpu.mdk
  ```

  MDK displays boot status information and the `MDK>` prompt.

- Load the *instgen* test from the `MDK>` prompt:

  ```
  MDK> instgen
  ```

  The *instgen* test loads into the system and begins testing the R10000/R12000 processors.

2.2.2 instgen Test Description

The *instgen* test generates random sequences of instructions, random addresses to use for memory operands, and random data in the memory operands. After it executes the random instruction sequences, *instgen* compares the results that each CPU obtained with results of other CPUs to detect failures.

The random instruction generator generates the entire MIPS instruction set (except for kernel instructions such as translation lookaside buffer [TLB] instructions, cache instructions, etc).

Some of the features tested in this diagnostic are the ALU, the floating-point unit, various types of branches and jumps (subroutine calls and forward branches), fetches and stores on local node memory and remote node memory, CPUs sharing cache lines, placement of code and data areas contiguous in memory (a code area followed immediately in memory by a data area or the inverse).

2.2.3 instgen Test Output

The *instgen* test returns output to the BaseIO console.

2.2.3.1 Pass Output

The *instgen* test prints a message for each set of passes that it runs without detecting hardware failures. The test prints the message `MDK Random Instruction Generator is complete` when all passes have completed testing.

For example, the following message indicates that the *instgen* test ran 20,499 passes, is starting pass 20,500, and has not detected any error so far:

```
instgen: Stating PASS 20500

MDK Random Instruction Generator test run is complete.
```

2.2.3.2 Failure Output

When a section of this test detects a failure, the test prints an error message that indicates which test section failed. The types of errors that *instgen* detects are data miscompares in all R10000/R12000 registers. (Examples of hardware checked by *instgen* are the general purpose register [GPR], floating-point register [FPR], status register, HI and LO registers, or in memory [primary data cache, secondary data cache, or memory].)

For example, the following failure output indicates that GPR 16 (general purpose register 16 in the R10000/R12000 processor) miscompared between CPU 0 and CPU 4.

```
instgen: Starting PASS  1587700
ERROR: PASS: 1587738 GPR miscompare: cpu  4 gpr 16 = 0
                                     cpu  0 gpr 16 = 118

Error Analysis: PASS 1587738
    The cpu that most likely caused the error is:

      Virtual   Nasid    Module    Node    Physical
        cpu                         slot      cpu
      -------   -----    ------    ----    --------
         4        5         2        2        A

MDK Random Instruction Generator test run is complete.
```

2.2.4 instgen Test Example

In the following example, the *instgen* test runs on an 8-Node system (16 CPUs) for 1,000 passes without detecting any errors.

```
>>boot dksc(0,1,0)/stand/cpu.mdk
Booting Nanos.........

************************************************************
SGI MDK Version 1.5 (256p RtrWAR) built 07:02:55 PM Apr 4, 1999
************************************************************

CPU 0: Total no. of CPUs = 16  Total Kernel Memory = a800000000598000
Launched CPU 1
Launched CPU 2
Launched CPU 3
Launched CPU 4
Launched CPU 5
Launched CPU 6
Launched CPU 7
Launched CPU 8
Launched CPU 9
Launched CPU 10
Launched CPU 11
Launched CPU 12
Launched CPU 13
Launched CPU 14
Launched CPU 15
MDK> ls

instgen

MDK> instgen -s 1 -e 1000
.....

Starting Random Instruction Generation Test (version 2.0).

System Configuration:

   Number of nodes:      8
   Number of cpus:      16
   Number of routers:    4


Table of module, nasid, and slot numbers:

     Module      Nasid      Node      Router
                            Slot       slot
     ------      -----      ----      ------
        1          0          1          1
        1          1          2          1
        2          5          2          1
        2          4          1          1
        1          3          4          2
        1          2          3          2
        2          7          4          2
        2          6          3          2
```

```
Random Instruction Generation test parameters:

   Num_Cpus_To_Run              =  16
   Start Pass                   =  1
   End Pass                     =  1000
   Global Table Ptr             =  f3b000      a800000000619000
   Trace  Table Ptr             =  f55000      a800000000633000
   Instr Def Ptr                =  f12700      a8000000005de700
  mem_alloc_code_data_address[ 0]           =          17f6000
  mem_alloc_code_data_address physical addr = a800000000ed4000
  mem_alloc_code_data_length[ 0]            =          1000000
  mem_alloc_code_data_address[ 1]           =          27f6000
  mem_alloc_code_data_address physical addr = a800000100020000
  mem_alloc_code_data_length[ 1]            =          1000000
  mem_alloc_code_data_address[ 2]           =          37f6000
  mem_alloc_code_data_address physical addr = a800000200020000
  mem_alloc_code_data_length[ 2]            =          1000000
  mem_alloc_code_data_address[ 3]           =          47f6000
  mem_alloc_code_data_address physical addr = a800000300020000
  mem_alloc_code_data_length[ 3]            =          1000000
  mem_alloc_code_data_address[ 4]           =          57f6000
  mem_alloc_code_data_address physical addr = a800000400020000
  mem_alloc_code_data_length[ 4]            =          1000000
  mem_alloc_code_data_address[ 5]           =          67f6000
  mem_alloc_code_data_address physical addr = a800000500020000
  mem_alloc_code_data_length[ 5]            =          1000000
  mem_alloc_code_data_address[ 6]           =          77f6000
  mem_alloc_code_data_address physical addr = a800000600020000
  mem_alloc_code_data_length[ 6]            =          1000000
  mem_alloc_code_data_address[ 7]           =          87f6000
  mem_alloc_code_data_address physical addr = a800000700020000
  mem_alloc_code_data_length[ 7]            =          1000000
SysRegUsrExc: pid = 1 type = 15

instgen: Starting PASS  1

instgen: Starting PASS  100

instgen: Starting PASS  200

instgen: Starting PASS  300

instgen: Starting PASS  400

instgen: Starting PASS  500

instgen: Starting PASS  600

instgen: Starting PASS  700

instgen: Starting PASS  800

instgen: Starting PASS  900

instgen: Starting PASS  1000

MDK Random Instruction Generator test run is complete.
```

2.3 pcache Test

The *pcache* test tests the primary data cache in the R10000/R12000 processor.

2.3.1 How to Run the pcache Test

Use the following procedure to load the test from the local system disk:

- Use the *boot* command from the PROM monitor (at the BaseIO command prompt, >>) to start the *cpu.mdk* test package:

  ```
  >>boot dksc(0,1,0)/stand/cpu.mdk
  ```

  MDK displays boot status information and the `MDK>` prompt.

- Load the *pcache* test from the `MDK>` prompt:

  ```
  MDK> pcache
  ```

  The *pcache* test loads into the system and begins testing the R10000/R12000 processors.

2.3.2 pcache Test Description

The *pcache* test tests the primary data cache in the R10000/R12000 processor.

2.3.3 2.3.3 pcache Test Output

The *pcache* test returns output to the BaseIO console.

2.3.3.1 Pass Output

The *pcache* test prints the message `PCACHE Test Passed` for each CPU that passes without detecting hardware failures. The test prints the message `MDK R10000 combined test run is complete` when all CPUs have completed testing.

For example, the following messages indicate that the *pcache* test passed on all CPUs and has completed testing on all CPUs:

```
NODE0 CPU0 Pid 1 Primary Data Cache Test Started.
NODE0 CPU0 Pid 1    This test takes about 5 minutes.
NODE0 CPU1 Pid 2 Primary Data Cache Test Started.
NODE0 CPU1 Pid 2    This test takes about 5 minutes.
NODE1 CPU2 Pid 3 Primary Data Cache Test Started.
NODE1 CPU2 Pid 3    This test takes about 5 minutes.
NODE1 CPU3 Pid 4 Primary Data Cache Test Started.
NODE1 CPU3 Pid 4    This test takes about 5 minutes.
NODE0 CPU0 Pid 1 PCACHE Test Passed
NODE0 CPU1 Pid 2 PCACHE Test Passed
NODE1 CPU2 Pid 3 PCACHE Test Passed
NODE1 CPU3 Pid 4 PCACHE Test Passed
```

2.3.3.2 Failure Output

When a CPU detects a failure, the test prints an error message that indicates which CPU failed. For example, the following failure output indicates that the *pcache* test detected a failure in CPU0:

```
NODE0 CPU0 Pid 1 Primary Data Cache Test Started.
NODE0 CPU0 Pid 1    This test takes about 5 minutes.
NODE0 CPU0 Pid 1 ERROR: Test Failed
```

When a CPU detects an ECC failure, the CPU causes a cache error exception to occur and the output is in the form of an MDK exception message; for example:

```
Error, Unexpected cache error exception...
  ErrorEPC=0xa800000000022eb0, CacheErr=0x88109b80
========================================================
 Unexpected exception occured.
         Eframe Registers on Stack
========================================================
  PID     = 1
  CPUNUM  = 0
  NASID   = 0
  LOC CPU = A
  EPC     = 0x          e0216c     BADVADDR= 0x        e0007a20
  CAUSE   = 0x            4008     STATUS  = 0x        b40000e4
  XCTXT   = 0xfffffffffff00030     ENHI    = 0xc0000fffe0006001
  ENLO0   = 0x               3     ENLO1   = 0x         25b65fd
  COUNT   = 0x         7a48ab0     COMPARE = 0x         c800000
  AT/r1   = 0xa800000000109bf0
  V0/r2   = 0x          109bf0     V1/r3   = 0xa800000000108f28
  A0/r4   = 0xa800000000497808     A1/r5   = 0x               2
  A2/r6   = 0xa80000000045adc8     A3/r7   = 0x          108f28
  A4/r8   = 0x               8     A5/r9   = 0xa800000000022e88
  A6/r10  = 0xa800000000000000     A7/r11  = 0x          f400a0
  T0/r12  = 0xc0000fffe0006001     T1/r13  = 0xffffffff80000000
  T2/r14  = 0x             116     T3/r15  = 0x          f400a0
  S0/r16  = 0xa80000000045adc8     S1/r17  = 0x               2
  S2/r18  = 0xa800000000108fd8     S3/r19  = 0x               0
  S4/r20  = 0x               0     S5/r21  = 0x               2
  S6/r22  = 0x               0     S7/r23  = 0x               0
  T8/r24  = 0x          10d0f8     T9/r25  = 0x               4
  SP/r29  = 0xa80000000045acf8     GP/r28  = 0x               0
  RA/r31  = 0xa800000000022c8c


  FPSR    = 0xc0000fffe0006001
  FP0     = 0x               0     FP1     = 0x1fcf2cce3b8e599c
  FP2     = 0x 1fe396671fe7bee     FP3     = 0x   ee396671fe7be
  FP4     = 0x   771cb330ee396     FP5     = 0x 7fc72cc83b8e599
  FP6     = 0x 3f8000000cc3044     FP7     = 0x        1cddeea2
  FP8     = 0x        3dfffff3     FP9     = 0x         e6ef751
  FP10    = 0x         1eeef75     FP11    = 0x         1fdddee
  FP12    = 0x         3ffffff     FP13    = 0x b78000002701066
  FP14    = 0x 5bc2cbb8b785977     FP15    = 0x 2de165d87fe3eff
  FP16    = 0x  8ff859741ff8fb     FP17    = 0x   5bc2cb80b7859
  FP18    = 0x  1c7bc2cb80b785     FP19    = 0x        1fc04803
  FP20    = 0x         4e86d4b     FP21    = 0x         6fe3ff7
```

```
  FP22    = 0x         17a1b52     FP23    = 0x           de86d
  FP24    = 0x        347fe3ff     FP25    = 0x              0
  FP26    = 0x               0     FP27    = 0x              0
  FP28    = 0x               0     FP29    = 0x              0
  FP30    = 0x               0     FP31    = 0x              0
=========================================================
MDK>
```

When this test fails, the failing field replaceable unit (FRU) is normally the Node board that contains the failing R10000/R12000 processor. However, the test could encounter an uncorrectable memory error while it attempts to access data from secondary cache or memory; if this occurs, the secondary cache test or memory tests should also detect the error.

2.3.4 pcache Test Example

In the following example, the *pcache* test runs on an 8-Node system (16 CPUs) without detecting any errors.

```
>>boot dksc(0,1,0)/stand/cpu.mdk
Booting Nanos.........

.....
MDK built with PROM version 6.12
PROM Version 6.25 installed

************************************************************
SGI MDK Version 1.5 (256p RtrWAR) built 06:59:00 PM Apr 5, 1999
************************************************************

CPU 0: Total no. of CPUs = 16  Total Kernel Memory = a800000000598000
Launched CPU 1
Launched CPU 2
Launched CPU 3
Launched CPU 4
Launched CPU 5
Launched CPU 6
Launched CPU 7
Launched CPU 8
Launched CPU 9
Launched CPU 10
Launched CPU 11
Launched CPU 12
Launched CPU 13
Launched CPU 14
Launched CPU 15
MDK> ls

pcache

MDK> pcache
.....
```

```
Starting Primary Data Cache test (version 1.1.2).

System Configuration:

   Number of nodes:      8
   Number of cpus:      16
   Number of routers:    4
NODE0 CPU1 Pid 2 Primary Data Cache Test Started.
NODE0 CPU1 Pid 2    This test takes about 5 minutes.
NODE1 CPU3 Pid 4 Primary Data Cache Test Started.
NODE1 CPU3 Pid 4    This test takes about 5 minutes.
NODE0 CPU0 Pid 1 Primary Data Cache Test Started.
NODE0 CPU0 Pid 1    This test takes about 5 minutes.
NODE1 CPU2 Pid 3 Primary Data Cache Test Started.
NODE1 CPU2 Pid 3    This test takes about 5 minutes.
NODE5 CPU10 Pid 11 Primary Data Cache Test Started.
NODE5 CPU10 Pid 11    This test takes about 5 minutes.
NODE5 CPU11 Pid 12 Primary Data Cache Test Started.
NODE5 CPU11 Pid 12    This test takes about 5 minutes.
NODE4 CPU8 Pid 9 Primary Data Cache Test Started.
NODE4 CPU8 Pid 9    This test takes about 5 minutes.
NODE4 CPU9 Pid 10 Primary Data Cache Test Started.
NODE4 CPU9 Pid 10    This test takes about 5 minutes.
NODE3 CPU6 Pid 7 Primary Data Cache Test Started.
NODE3 CPU6 Pid 7    This test takes about 5 minutes.
NODE3 CPU7 Pid 8 Primary Data Cache Test Started.
NODE3 CPU7 Pid 8    This test takes about 5 minutes.
NODE2 CPU4 Pid 5 Primary Data Cache Test Started.
NODE2 CPU4 Pid 5    This test takes about 5 minutes.
NODE2 CPU5 Pid 6 Primary Data Cache Test Started.
NODE2 CPU5 Pid 6    This test takes about 5 minutes.
NODE7 CPU15 Pid 16 Primary Data Cache Test Started.
NODE7 CPU15 Pid 16    This test takes about 5 minutes.
NODE6 CPU12 Pid 13 Primary Data Cache Test Started.
NODE6 CPU12 Pid 13    This test takes about 5 minutes.
NODE7 CPU14 Pid 15 Primary Data Cache Test Started.
NODE7 CPU14 Pid 15    This test takes about 5 minutes.
NODE6 CPU13 Pid 14 Primary Data Cache Test Started.
NODE6 CPU13 Pid 14    This test takes about 5 minutes.
NODE0 CPU1 Pid 2 PCACHE Test Passed
NODE1 CPU3 Pid 4 PCACHE Test Passed
NODE0 CPU0 Pid 1 PCACHE Test Passed
NODE1 CPU2 Pid 3 PCACHE Test Passed
NODE5 CPU10 Pid 11 PCACHE Test Passed
NODE5 CPU11 Pid 12 PCACHE Test Passed
NODE4 CPU8 Pid 9 PCACHE Test Passed
NODE4 CPU9 Pid 10 PCACHE Test Passed
NODE3 CPU6 Pid 7 PCACHE Test Passed
NODE3 CPU7 Pid 8 PCACHE Test Passed
NODE2 CPU4 Pid 5 PCACHE Test Passed
NODE2 CPU5 Pid 6 PCACHE Test Passed
NODE7 CPU15 Pid 16 PCACHE Test Passed
NODE6 CPU12 Pid 13 PCACHE Test Passed
NODE7 CPU14 Pid 15 PCACHE Test Passed
NODE6 CPU13 Pid 14 PCACHE Test Passed
MDK pcache test run is complete.
```

Chapter 3

Secondary Cache Test

This chapter describes the diagnostic test that you can use to test the secondary caches in the system.

3.1 About the Secondary Cache Test

The secondary cache test is a directed test that checks the secondary caches that are located on the Node boards. To run the secondary cache test, you must load the *memory.mdk* test package.

It this test fails, the failing hardware is one or more of the secondary caches (HIMMs) in the system. The failing FRU is the Node board that contains the failing secondary cache.

3.2 How to Run the Secondary Cache Test

Use the following procedure to load the test from the local system disk:

- Use the *boot* command from the PROM monitor (at the BaseIO command prompt, >>) to start the *memory.mdk* test package:

  ```
  >>boot dksc(0,1,0)/stand/memory.mdk
  ```

  MDK displays boot status information and the `MDK>` prompt.

- Load the secondary cache test, *scache*, from the `MDK>` prompt:

  ```
  MDK>scache
  ```

  The secondary cache test loads into the system and begins testing the secondary caches.

3.3 Secondary Cache Test Description

The secondary cache test runs separately in each CPU to enable the test to check each CPU's local secondary cache (located on the HIMM).

The secondary cache test writes data to the secondary cache, ensuring that data is written from the primary cache into the secondary cache. The test then reads the data back and compares the data that was read back with expected data.

The secondary cache test includes three subtests that use different data patterns to test the secondary caches: a data path test, a cell test, and a random address and data test.

3.3.1 Data Path Test

The data path test uses data patterns that cause simultaneous switching on the data lines. This test uses the following algorithm:

- Fill the secondary cache with data vectors that cause simultaneous switching on the secondary cache lines. This test writes data in the following sequence:

```
0x0000 0000 0000 0000 0000 0000 0000 0001
0xffff ffff ffff ffff ffff ffff ffff fffe
0x0000 0000 0000 0000 0000 0000 0000 0002
0xffff ffff ffff ffff ffff ffff ffff fffd
. . .
. . .
0x8000 0000 0000 0000 0000 0000 0000 0000
0x7fff ffff ffff ffff ffff ffff ffff ffff
```

- Read the data back and compare the data that was read back with the data that was written.

3.3.2 Cell Test

There are two parts to the cell test: a data vectors cell test and a random data cell test.

3.3.2.1 Data Vectors Cell Test

The data vectors cell test writes specific data vectors to each memory cell in the secondary cache to test the cache as an array. This test uses the following array of data vectors:

```
0xffffffff, 0xaaaaaaaa, 0xcccccccc, 0xffff0000, 0x00000000, 0x55555555,
0x33333333, 0x0000ffff
```

The test fills the secondary cache with this data, reads the data back, and compares the data that was read back with the data that was written. The test repeats this process for several passes; each pass starts at a different value in the data array.

3.3.2.2 Random Data Cell Test

The random data cell test moves lines of data between the primary cache and secondary cache and between the secondary cache and memory. This test uses random data. The number of tag field bits that the test can manipulate is limited by the size of the user address space that MDK allows or by the amount of physical memory in the system.

3.3.3 Random Address and Data Test

The random address test selects a random address and a random range of address bits to change. The test creates new addresses to use by randomly modifying the address bits in the selected range.

This process creates addresses that may cause store operations to a stride of cache lines within the cache. This process also creates addresses that cause data lines in the cache to be written back to memory.

3.4 Secondary Cache Test Output

The secondary cache test returns output to the BaseIO console.

3.4.1 Pass Output

The secondary cache test prints the following messages if it completes testing without detecting any errors:

```
NODEx CPUy PIDz Scache data path test PASSED
NODEx CPUy PIDz Scache cell test PASSED
NODEx CPUy PIDz Scache random address test PASSED
Secondary cache test has completed
```

3.4.2 Failure Output

The secondary cache test prints the following messages if it detects errors:

```
NODEx CPUy PIDz Scache data path test FAILED
NODEx CPUy PIDz Scache cell test FAILED
NODEx CPUy PIDz Scache random address test FAILED
```

Virtual addr *virtual_address* Physical addr *physical_address*
Expected data *expected_data* Actual data *actual_data*
Syndrom *syndrome_value*

If this test fails, the failing hardware is the secondary cache (HIMM), memory, or CPU. The failing FRU is the Node board that contains the failing secondary cache.

3.5 Secondary Cache Test Example

The following example shows the secondary cache test running on a 2-Node system (4 CPUs). In this example, the test does not detect any hardware failures.

```
>>boot dksc(0,1,0)/stand/memory.mdk
Booting Nanos.........
************************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
************************************************************

CPU 0: Total no. of CPUs = 4
Launched CPU 1
Launched CPU 2
Launched CPU 3
MDK>scache
Created successfully pid = 1
CPU 0: User mem starts from 0xa800000000641000
NODE0 CPU0 PID1 In1B 000:  main
NODE0 CPU1 PID2 In child
NODE0 CPU0 PID1 Scache data pat1B 000: h test
NODE0 CPU1 PID2 Scache data path test
CPU0: sCDataBusTest: Loop 00000000
CDataBusTest: Loop 00000000
NODE1 CPU3 PID4 2A 001: In child
NODE1 CPU2 PID3 In child
NODE1 CPU3 PID4 Scache data path test
NODE1 CPU2 PID3 Scache data path test
CPU3: sCDataBusTest: Loop 00000000
CDataBusTest: Loop 00000000
CPU0: sCDataBusTest: Loop 00000500
CPU2: sCDataBusTest: Loop 00000500
CPU3: sCDataBusTest: Loop 00000500
CPU1: sCDataBusTest: Loop 00000500
NODE0 CPU0 PID1 Scache data path test PASSED
NODE0 CPU0 PID1 Scache cell test
NODE0 CPU0 PID1 sCCellRandTest Loop 00000000
NODE1 CPU2 PID3 Scache data path test PASSED
NODE1 CPU2 PID3 Scache cell test
NODE1 CPU2 PID3 sCCellRandTest Loop 00000000
NODE1 CPU3 PID4 Scache data path test PASSED
NODE1 CPU3 PID4 Scache cell test
NODE1 CPU3 PID4 sCCellRandTest Loop 00000000
NODE0 CPU1 PID2 Scache data path test PASSED
NODE0 CPU1 PID2 Scache cell test
NODE0 CPU1 PID2 sCCellRandTest Loop 00000000
NODE0 CPU0 PID1 sCCellRandTest Loop 00000500
NODE1 CPU2 PID3 sCCellRandTest Loop 00000500
NODE1 CPU3 PID4 sCCellRandTest Loop 00000500
NODE0 CPU1 PID2 sCCellRandTest Loop 00000500
NODE0 CPU0 PID1 Scache cell test PASSED
NODE0 CPU0 PID1 Scache random address test
CPU0: sCRandAddrTest: Loop 00000000
CPU0: sCRandAddrTest: Loop 00000500
NODE0 CPU0 PID1 Scache random address test PASSED
NODE0 CPU0 PID1 PASSED
NODE1 CPU2 PID3 Scache cell test PASSED
```

```
NODE1 CPU2 PID3 Scache random address test
CPU2: sCRandAddrTest: Loop 00000000
CPU2: sCRandAddrTest: Loop 00000500
NODE1 CPU2 PID3 Scache random address test PASSED
NODE1 CPU2 PID3 child PASSED
NODE1 CPU3 PID4 Scache cell test PASSED
NODE1 CPU3 PID4 Scache random address test
CPU3: sCRandAddrTest: Loop 00000000
CPU3: sCRandAddrTest: Loop 00000500
NODE1 CPU3 PID4 Scache random address test PASSED
NODE1 CPU3 PID4 child PASSED
NODE0 CPU1 PID2 Scache cell test PASSED
NODE0 CPU1 PID2 Scache random address test
CPU1: sCRandAddrTest: Loop 00000000
CPU1: sCRandAddrTest: Loop 00000500
NODE0 CPU1 PID2 Scache random address test PASSED
NODE0 CPU1 PID2 child PASSED
**** Secondary cache test status summary ****
CNODE0 NASID0 CPU0 PID1 PASSED
CNODE0 NASID0 CPU1 PID2 PASSED
CNODE1 NASID1 CPU2 PID3 PASSED
CNODE1 NASID1 CPU3 PID4 PASSED
Secondary cache test has completed
```

Chapter 4

Memory Tests

This chapter describes the diagnostics that you can use to test memory.

4.1 About the Memory Tests

The memory tests are directed tests that verify that memory properly stores data that is written to it. Five different types of memory tests are available: directory memory tests, quick screen memory tests, Node board memory tests, internode memory tests, and long memory tests. There are mapped and unmapped versions of most of these tests.

To run the memory tests, you must load the *memory.mdk* test package.

Table 4-1 provides quick-reference descriptions of the memory tests. The number in the Page column indicates the page on which a detailed description of each test begins.

Table 4-1 Memory Tests

Type	Name	Description	Page
Directory memory test	*memum.dr*	Accesses memory while changing the state of each cache line from unowned to exclusive to shared (unmapped version)	4-9
Quick screen memory tests	*mem.qs*	Writes data patterns to memory, reads the data back, and verifies the results (mapped version)	4-13
	memum.qs	Writes data patterns to memory, reads the data back, and verifies the results (unmapped version)	4-13
Node board memory test	*memum.nb*	Uses one or two CPUs to write two different sets of data to the cache lines, reads the data back, and verifies the results (unmapped version)	4-22
Internode memory test	*memum.in*	Uses all Nodes and CPUs in the system to write data to the cache lines, reads the data back, and verifies the results (unmapped version)	4-28

Table 4-1 (continued) Memory Tests

Type	Name	Description	Page
Long memory tests	*mem.lo*	Runs extended versions of the *mem.q* and *mem.in* tests and also runs memory directory and Node board tests (mapped version)	4-33
	memum.lo	Runs extended versions of the *memum.dr*, *memum.qs*, *memum.nb*, and *memum.in* tests (unmapped version)	4-33

If any one of the memory tests fails, the failing hardware could be a memory DIMM, a secondary cache (a HIMM), a Node board, or a Router board. The failing FRU could be the memory DIMM, Node board, or Router board that contains the failing hardware component.

If the memory tests detect failing DIMMs, they print out the location of each DIMM (for example, MMXL0, MMXH0, or MMYH7). For the main memory DIMM slots, use the last two digits (either an L and a number or an H and a number) to determine which FRU to replace. Figure 4-1 shows the location of the DIMM memory slots on the Node board.

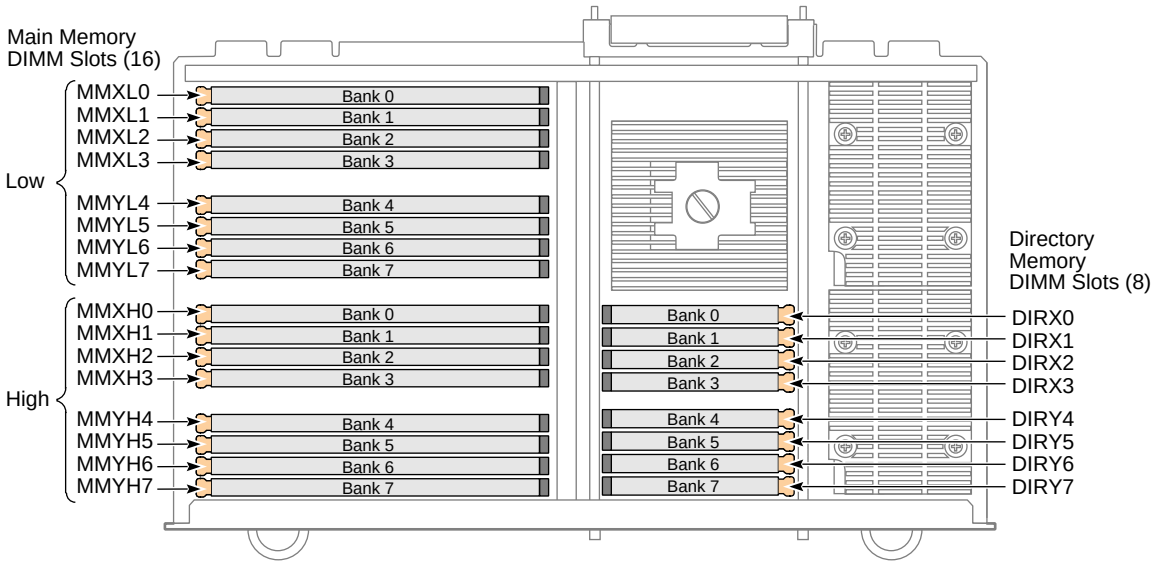

Figure 4-1 Memory DIMM Locations

4.2 Common Error Output

The memory tests print the same messages for several types of errors, including when the tests detect exceptions, when the tests find data miscompares, and when unmapped memory tests encounter single-bit ECC errors. The examples in Section 4.2.1, Section 4.2.2, and Section 4.2.3 show the output that the tests return for these errors.

4.2.1 Cache Error Exception Example

The following example shows a portion of the output from an unmapped or mapped memory test that detects an exception. In this example, the exception is a cache error exception.

```
Status: 0xffffffffb40080e5        XCtxt: 0xffffffe088888880
Hi:     0x               0        Lo:    0x              36
epc:    0xa800000000a07e28        cause: 0x            4020
count:  0x          60a3be        comp:  0x         c800000
Enhi:   0x               0        CacErr: 0xffffffffd879e100
R01/AT: 0x               1        R02/V0: 0xa8000000026bdf80
R03/V1: 0xa800000001f9e108        R04/A0: 0xa800000001f9c988
R05/A1: 0x               0        R06/A2: 0x               0
R07/A3: 0x         34a9ff8        R08/A4: 0x               0
R09/A5: 0xa800000000b14000        R10/A6: 0xa800000000b14000
R11/A7: 0x               1        R12/T0: 0xa8000000026bdf80
R13/T1: 0xa8000000026bdf80        R14/T2: 0x               1
R15/T3: 0x               0        R16/S0: 0x               0
R17/S1: 0x               0        R18/S2: 0x               0
R19/S3: 0x               0        R20/S4: 0x               0
R21/S5: 0x               0        R22/S6: 0x               0
R23/S7: 0x               0        R24/T8: 0x               0
R25/T9: 0x7777777711111111        R28/GP: 0x               0
R29/SP: 0xa800000003fde688        R30/S8: 0x               0
R31/RA: 0xa800000001f9e108
Nasid 0: Local CPU A: Global CPU 0: PID 1: Cac Err exception at
0xa800000000a05640: sr = 0xb40080e5
Nanos: Fr
MSC> nmi
        ok

*** NMI while in Kernel and no NMI vector installed on node 0
*** NMI while in Kernel and no NMI vector installed on node 0
*** Error EPC: 0xa800000000a008cc (0xa800000000a008cc)
*** Error EPC: 0xa800000000025354 (0xa800000000025354)
*** Press ENTER to continue.
*** Press ENTER to continue.
POD MSC Dex MDerr> error
Uncorrectable memory ECC error, with overrun
   Address      : 0x1f9e100
   Bad syndrome : 0xc6 (multi)
   Physical loc : MMXH0 and/or MMXL0
POD MSC Dex MDerr> pr MD_MEM_ERROR
Register: MD_MEM_ERROR (0x9200000001200070)
Value   : 0xc000007e01f20a81 (loaded from register)
   <63> R   UCE_VALID                0x1
   <62> R   CE_VALID                 0x1
```

```
<39:32> R   BAD_SYN                    0x7e
<31:03> R   ADDRESS<31:03>             0x003f3c20 << 3 = 0x01f9e100
   <01> R   UCE_OVERRUN                0x0
   <00> R   CE_OVERRUN                 0x1
POD MSC Dex MDerr> ld u:0x1f9e100
9600000001f9e100 75757777111111111111
```

In this example, the following actions were taken to obtain more information about the exception:

- A nonmaskable interrupt was issued from the `MSC>` prompt after the exception occurred to bring the system into POD mode.
- The POD *error* command was issued to obtain more information about memory or directory errors (from the MD section of the HUB) because the `Madder` text in the POD prompt indicates that there are MD errors pending.
- The POD *pr MD_MEM_ERROR* command was issued to print the contents of the memory / directory memory error register. (This register shows more detail about the ECC error.)

  Note: Refer to *IP27PROM Technical Reference Manual*, publication number 108-0170-001, for more information about the registers that you can print with the *pr* command.

- The POD *ld u:0x1f9e100* command was issued to view the contents of the memory address that the *pr* command reported. (This shows the data miscompare: The memory address contains 0x75757777111111111111 when it should contain 0x77777777111111111111.)

To summarize, the cache error exception was caused by an uncorrectable ECC memory error at address 0x1f9e100; the failing DIMM(s) are in location MMXH0 and / or MMXL0 (refer to the `Physical loc` field of the *error* command output).

4.2.2 Data Miscompare Example

The following example shows a portion of the output that MDK prints when a memory test detects a data miscompare:

```
Node 0 Bank 0 Physical addr 0xa800000002016838
Expected data 0x7777777712345678 Actual data 0x7777777711111111
Syndrom 0x0000000003254769
Address 0000000002016838 bit 0 is on NASID 0
MMXL0 DIMML line 18
Address 0000000002016838 bit 3 is on NASID 0
MMXL0 DIMML line 21
Address 0000000002016838 bit 5 is on NASID 0
MMXL0 DIMML line 23
Address 0000000002016838 bit 6 is on NASID 0
MMXL0 DIMML line 24
Address 0000000002016838 bit 8 is on NASID 0
MMXL0 DIMML line 26
Address 0000000002016838 bit 9 is on NASID 0
MMXL0 DIMML line 27
Address 0000000002016838 bit 10 is on NASID 0
MMXL0 DIMML line 28
Address 0000000002016838 bit 14 is on NASID 0
MMXL0 DIMML line 32
Address 0000000002016838 bit 16 is on NASID 0
MMXL0 DIMML line 34
Address 0000000002016838 bit 18 is on NASID 0
MMXL0 DIMML line 54
Address 0000000002016838 bit 21 is on NASID 0
MMXL0 DIMML line 57
Address 0000000002016838 bit 24 is on NASID 0
MMXL0 DIMML line 60
Address 0000000002016838 bit 25 is on NASID 0
MMXL0 DIMML line 61
**** MDK memory mapped test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test FAILED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
ERROR Memory mapped test failed
```

This output indicates that the failing DIMM is located in DIMM slot MMXL0. (Refer to Table 4-1 on page 4-2 for an illustration of the DIMM slot locations on a Node board.)

This example shows the output from a mapped memory test. For an unmapped memory test, the error output uses the following format:

```
ERROR CNODE0 CPU1 PID2 memUmRandomAddressAndDataCompare FAILED
Module 1 Slot 1 Nasid 0 Bank 0 Physical addr 0xa800000000b0f140
Expected data ----- 0x172b0e4f2c1c9821
Actual data ------- 0x172b0e4f2c1c9820
Corrupted data bits 0x0000000000000001
Address 0xa800000000b0f140 bit 8 is on NASID 0
MMXL0 == DIMM BANK 0 L - line(DQ) 8
```

Note: MDK replaces the `memUmRandomAddressAndDataCompare` label in the output with `memUmPatternTest`, `memSsoOverCacheLineTest`, `memUmRandomDataTest`, `memUmCnvrgAddrAndFlippingDataWr`, or `memUmInterNodeRandomAddressAndDataCompar` to indicate which unmapped memory test was running when the miscompare was detected.

4.2.3 ECC Error Examples (Unmapped Memory Tests Only)

The unmapped tests print summaries of any directory memory ECC errors or memory ECC occurs that occur during testing.

4.2.3.1 Output for Directory Memory ECC Errors

If directory memory ECC errors occur, the DIR SBECC SUMMARY portion of the output contains error information similar to the following example output:

```
############### DIR SBECC SUMMARY ###############

Premium directory ecc err
Module 1 Slot 2 Address 0x1511d4a328
Encountered MULTIPLE SBECC errs
MMXL2 == DIMM BANK 2 L - line(DQ) 13
syndrome 0x0d == dir_06
-----------------------------------------
Premium directory ecc err
Module 1 Slot 2 Address 0x1508108ba0
Encountered MULTIPLE SBECC errs
MMXH1 == DIMM BANK 1 H - line(DQ) 0
syndrome 0x01 == check_0
-----------------------------------------

############### MEM SBECC SUMMARY ###############

No memory ecc errs detected


**** MDK memory unmapped test status summary ****

CNODE0 NASID20 CPU0 PID1 Mem test PASSED
CNODE0 NASID20 CPU1 PID2 Mem test PASSED
CNODE1 NASID21 CPU2 PID3 Directory single bit ECC ERROR - Mem test FAILED
CNODE1 NASID21 CPU3 PID4 Directory single bit ECC ERROR - Mem test FAILED
CNODE2 NASID23 CPU4 PID5 Mem test PASSED
CNODE2 NASID23 CPU5 PID6 Mem test PASSED
CNODE3 NASID22 CPU6 PID7 Mem test PASSED
CNODE3 NASID22 CPU7 PID8 Mem test PASSED
Memory unmapped test has completed
```

4.2.3.2 Output for Memory ECC Errors

If memory ECC errors occur, the MEM SBECC SUMMARY portion of the output contains error information similar to the following example output:

```
############### DIR SBECC SUMMARY ###############

No directory ecc errs detected


############### MEM SBECC SUMMARY ###############

Main memory ecc err
Module 1 Slot 1 Address 0xa800000083887f18
Encountered MULTIPLE SBECC errs
SBECC err reproducable - YES
Address 0xa800000083887f18 bit 10 is on NASID 0
MMYL4 == DIMM BANK 4 L - line(DQ) 28
syndrome 0x43 == data_02
----------------------------------------

Main memory ecc err
Module 1 Slot 1 Address 0xa8000000c13c5a98
Encountered MULTIPLE SBECC errs
SBECC err reproducable - NO
Address 0xa8000000c13c5a98 bit 69 is on NASID 0
MMYH6 == DIMM BANK 6 H - line(DQ) 69
syndrome 0xc2 == data_61
----------------------------------------

Main memory ecc err
Module 1 Slot 2 Address 0xa800000183721408
Encountered MULTIPLE SBECC errs
SBECC err reproducable - YES
Address 0xa800000183721408 bit 32 is on NASID 1
MMYL4 == DIMM BANK 4 L - line(DQ) 68
syndrome 0x19 == data_24
----------------------------------------

Main memory ecc err
Module 1 Slot 4 Address 0xa800000341168d10
Encountered MULTIPLE SBECC errs
SBECC err reproducable - YES
Address 0xa800000341168d10 bit 5 is on NASID 3
MMXL2 == DIMM BANK 2 L - line(DQ) 5
syndrome 0x20 == check_5
----------------------------------------

**** MDK memory unmapped test status summary ****

CNODE0 NASID0 CPU0 PID1   Mem single bit ECC ERROR - Mem test FAILED
CNODE0 NASID0 CPU1 PID2   Mem single bit ECC ERROR - Mem test FAILED
CNODE1 NASID1 CPU2 PID3   Mem single bit ECC ERROR - Mem test FAILED
CNODE1 NASID1 CPU3 PID4   Mem single bit ECC ERROR - Mem test FAILED
```

```
CNODE2 NASID5 CPU4 PID5   Mem test PASSED
CNODE2 NASID5 CPU5 PID6   Mem test PASSED
CNODE3 NASID4 CPU6 PID7   Mem test PASSED
CNODE3 NASID4 CPU7 PID8   Mem test PASSED
CNODE4 NASID3 CPU8 PID9   Mem single bit ECC ERROR - Mem test FAILED
CNODE4 NASID3 CPU9 PID10  Mem single bit ECC ERROR - Mem test FAILED
CNODE5 NASID2 CPU10 PID11 Mem test PASSED
CNODE5 NASID2 CPU11 PID12 Mem test PASSED
CNODE6 NASID7 CPU12 PID13 Mem test PASSED
CNODE6 NASID7 CPU13 PID14 Mem test PASSED
CNODE7 NASID6 CPU14 PID15 Mem test PASSED
CNODE7 NASID6 CPU15 PID16 Mem test PASSED
Memory unmapped test has completed
```

4.2.3.3 Output for Directory and Memory ECC Errors

If both directory memory and memory ECC errors occur, both the DIR SBECC SUMMARY and MEM SBECC SUMMARY portions of the output contain error information.

4.3 Directory Memory Test

The directory memory test changes the state of each cache line from unknown to exclusive to shared as it accesses data in memory. There is only an unmapped version of the directory memory test (*memum.dr*).

4.3.1 How to Run the Directory Memory Test

Use the following procedure to load the test from the local system disk:

- Use the *boot* command from the PROM monitor (at the BaseIO command prompt, >>) to start the *memory.mdk* test package:

  >>**boot dksc(0,1,0)/stand/memory.mdk**

  MDK displays boot status information and the MDK> prompt.

- Load the *memum.dr* test from the MDK> prompt:

  MDK>**memum.dr**

  The directory memory test loads into the system and begins testing memory.

4.3.2 Directory Memory Test Description

The directory memory test uses the following algorithm:

- Acquire the system configuration:
  1. Determine the amount of memory that is populated in the system.
  2. Determine the number of CPUs and Nodes in the system.
  3. Determine the size of the data block, and divide the data block between the CPUs. (Each CPU owns one byte of the data block, and multiple CPUs can share the same byte of the data block.)
- Perform an exclusive test:
  1. Have each CPU select the next available Node (starting at Node 0 and going through the maximum Node).
  2. Have each CPU select the next available populated bank on the Node that the CPU selected (starting at bank 0 and going through bank 7).
  3. Wait until all CPUs have performed Steps 1 and 2.
  4. Have each CPU write data to its 8-byte block (which is equal to a cache line) in the data block. (Repeat this action until the CPUs have written data to all data blocks in the selected bank.)
  5. Go to Step 2 and repeat Steps 2 through 4 for the next populated bank. (Repeat this process until all populated banks on the selected Node have been used.)
  6. Go to Step 1 and repeat the entire procedure for the next available Node. (Repeat this process until all available Nodes have been used.)

- Perform a shared test:
  1. Have each CPU select the next available Node (starting at Node 0 and going through the maximum Node).
  2. Have each CPU select the next available populated bank on the Node that the CPU selected (starting at bank 0 and going through bank 7).
  3. Wait until all CPUs have performed Steps 1 and 2.
  4. Have each CPU read data from its 8-byte block (which is equal to a cache line) in the data block. (Repeat this action until the CPUs have read data from all data blocks in the selected bank.)
  5. Go to Step 2 and repeat Steps 2 through 4 for the next populated bank. (Repeat this process until all populated banks on the selected Node have been used.)
  6. Go to Step 1 and repeat the entire procedure for the next available Node. (Repeat this process until all available Nodes have been used.)

4.3.3 Output from the Directory Memory Test

The directory memory test returns output to the BaseIO console.

4.3.3.1 Pass Output

As the test runs, it prints out a `PASSED` message for each test section that completes without detecting hardware failures. The following examples show some of these messages:

```
NODE0 CPU0 PID1 unmapped directory test PASSED
NODE1 CPU3 PID4 unmapped directory test PASSED
NODE0 CPU1 PID2 unmapped directory test PASSED
NODE1 CPU2 PID3 unmapped directory test PASSED
```

When all test sections have completed successfully, the test prints out a summary for each CPU in the system, as shown in the following example messages:

```
**** MDK unmapped memory test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
Memory unmapped test has completed
```

4.3.3.2 Failure Output

If this test detects errors, it prints an `ERROR` message and related output, similar to the following example messages:

```
Premium directory ecc err
Module 1 Slot 2 Address 0x1511d4a328
Encountered MULTIPLE SBECC errs
MMXL2 == DIMM BANK 2 L - line(DQ) 13
syndrome 0x0d == dir_06
----------------------------------------
```

```
Premium directory ecc err
Module 1 Slot 2 Address 0x1508108ba0
Encountered MULTIPLE SBECC errs
MMXH1 == DIMM BANK 1 H - line(DQ) 0
syndrome 0x01 == check_0
```

If the directory memory test fails, the failing hardware could be a memory or directory memory DIMM. The failing FRU could be the specified DIMM.

4.3.4 Example of Running the Directory Memory Test

The following example output shows the directory memory test running on a 2-Node system (4 CPUs). In this example, the test completes successfully without detecting any hardware failures.

```
>>boot dksc(0,1,0)/stand/memory.mdk
Booting Nanos.........

************************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
************************************************************

CPU 0: Total no. of CPUs = 4
Launched CPU 1
Launched CPU 2
Launched CPU 3
MDK>memum.dr
CNODE 0 NASID 0 SLOT 1 MODULE 1
CNODE 1 NASID 1 SLOT 2 MODULE 1
Config_t:
  numCpus    4
  numNodes   2
  numRouters 0
  CNODE000 NASID000:
    numCpus  2
    mem size 0x0000000008000000
    Bank0    0x0000000004000000
    Bank1    0x0000000004000000
  CNODE001 NASID001:
    numCpus  2
    mem size 0x0000000008000000
    Bank0    0x0000000004000000
    Bank1    0x0000000004000000
INFO_T addr of ppid      0xa800000023fde588
INFO_T addr of sysconfig 0xa800000023fde590
INFO_T addr of procinfo  0xa800000000b0a000
INFO_T addr of nodeinfo  0xa800000000b0b000
INFO_T addr of addrinfo  0xa800000000b0c000
CNODE000 BANK0 mem       0x0000000004000000
    CNODE000 BANK0 saddr 0xa800000000b0f000
    CNODE000 BANK0 eaddr 0xa800000004000000
    CNODE000 BANK0 size  0x00000000034f1000
CNODE000 BANK1 mem       0x0000000004000000
    CNODE000 BANK1 saddr 0xa800000020000000
    CNODE000 BANK1 eaddr 0xa800000023fbdff8
```

```
    CNODE000 BANK1 size  0x0000000003fbdff8
CNODE001 BANK0 mem       0x0000000004000000
    CNODE001 BANK0 saddr 0xa800000100020000
    CNODE001 BANK0 eaddr 0xa800000104000000
    CNODE001 BANK0 size  0x0000000003fe0000
CNODE001 BANK1 mem       0x0000000004000000
    CNODE001 BANK1 saddr 0xa800000120000000
    CNODE001 BANK1 eaddr 0xa800000123fbdff8
    CNODE001 BANK1 size  0x0000000003fbdff8
PID 1 CPU 0 CNODE 0 NASID 0
PID 2 CPU 1 CNODE 0 NASID 0
PID 3 CPU 2 CNODE 1 NASID 1
PID 4 CPU 3 CNODE 1 NASID 1
CNODE0 CPU0 PID1 unmapped memory directory test
CNODE1 CPU3 PID4 unmapped memory directory test
CNODE0 CPU0 PID1 Exclusive directory test
CNODE1 CPU3 PID4 Exclusive directory test
0CNODE0 CPU1 PID2 unmapped memory directory test
CNODE1 CPU2 PID3 unmapped memory directory test
CNODE0 CPU1 PID2 Exclusive directory test
CNODE1 CPU2 PID3 Exclusive directory test
1CNODE0 CPU1 PID2 Shared directory test
CNODE0 CPU0 PID1 Shared directory test
0CNODE1 CPU3 PID4 Shared directory test
CNODE1 CPU2 PID3 Shared directory test
1CNODE0 CPU1 PID2 unmapped memory directory test PASSED
CNODE0 CPU0 PID1 unmapped memory directory test PASSED
CNODE1 CPU3 PID4 unmapped memory directory test PASSED
CNODE1 CPU2 PID3 unmapped memory directory test PASSED

################ DIR SBECC SUMMARY ################

No directory ecc errs detected


################ MEM SBECC SUMMARY ################

No memory ecc errs detected


**** MDK memory unmapped test status summary ****

CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
Memory unmapped test has completed
```

4.4 Quick Screen Memory Tests

The quick screen memory tests write data patterns to memory, read the data back, and compare the data that was read back with the data that was written. The quick screen memory tests are *mem.qs* (mapped version) and *memum.qs* (unmapped version).

4.4.1 How to Run the Quick Screen Memory Tests

Use the following procedure to load the quick screen memory tests from the local system disk:

- Use the *boot* command from the PROM monitor (at the BaseIO command prompt, >>) to start the *memory.mdk* test package:

  >>**boot dksc(0,1,0)/stand/memory.mdk**

  MDK displays boot status information and the MDK> prompt.

- Load the mapped version (*mem.qs*) or unmapped version (*memum.qs*) from the MDK> prompt:

  1. To load the mapped version, enter *mem.qs* at the MDK> prompt:

     MDK>**mem.qs**

  2. To load the unmapped version, enter *memum.qs* at the MDK> prompt:

     MDK>**memum.qs**

  The specified quick screen memory test loads into the system and begins testing memory.

4.4.2 Test Description

Both quick screen memory tests use the following algorithm:

- Acquire the system configuration:
  1. Determine the amount of memory that is populated in the system.
  2. Determine the number of CPUs and Nodes in the system.
  3. Scale the test to the system configuration.
- Select one CPU to perform only read operations: This CPU causes memory traffic while the test is running.
- Select one CPU to perform read and write operations: This CPU does the actual memory testing.

Note: If there is only one CPU in the system, it runs two processes. One of the processes performs only read operations; the other process performs read and write operations.

- Perform a memory pattern test:
  1. Repeat the following actions until all of the memory being tested is filled with the data patterns:
     - Write 0xc33cc33cc33cc33c to a cache line.
     - Write the complement data pattern (0x3cc33cc33cc33cc3) to the next cache line.
  2. Read the data back and compare it to expected values.
  3. Repeat the following actions until all of the memory being tested is filled with the data patterns:
     - Write 0xffffffffffffffff to a cache line.
     - Write the complement data pattern (0x0000000000000000) to the next cache line.
  4. Read the data back and compare it to expected values.
- Perform a simultaneous switching (SSO) over cache line test:
  1. Fill the available memory with data vectors that cause simultaneous switching on the secondary cache lines. This test writes data in the following sequence:

     0x0000 0000 0000 0000 0000 0000 0000 0001
     0xffff ffff ffff ffff ffff ffff ffff fffe
     0x0000 0000 0000 0000 0000 0000 0000 0002
     0xffff ffff ffff ffff ffff ffff ffff fffd
     . . .
     . . .
     0x8000 0000 0000 0000 0000 0000 0000 0000
     0x7fff ffff ffff ffff ffff ffff ffff ffff
  2. Read the data back and compare the data that was read back with the data that was written.
- Perform a data bus test:
  1. Write data to memory addresses that are generated using the following information:
     - Node ID
     - DIMM bank ID
     - Physical bank ID
     - Logical bank ID
  2. Read the data back and compare it to expected values.
- Perform a background test:
  1. Write all memory being tested with all 0's.
  2. Read back one doubleword at a time and compare it to the expected value of 0.
  3. Write the complement of 0 to the doubleword location that was just read.
  4. Repeat Steps 2 and 3 until all available memory has been used.
  5. Read back one doubleword of data at a time and compare it to the expected value of the complement of 0.

6. Write 0 to the doubleword location that was just read.
7. Repeat Steps 5 and 6 until all available memory has been used.

- Perform an address bus test:
  1. Write memory with the following data patterns: 0xc33cc33cc33cc33c and 0x3cc33cc33cc33cc3.
  2. Read the data back and compare it to expected values.
- Repeat the following actions several times to perform a random data test:
  1. Write all memory being tested with a random data pattern.
  2. Read the data back and compare it to expected values.

4.4.3 Output from the Quick Screen Memory Tests

The quick screen memory tests return output to the BaseIO console.

4.4.3.1 Pass Output

As the tests run, they print out `PASSED` messages for each test passing section. The following examples show some of these messages:

```
NODE0 CPU0 PID1 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
NODE1 CPU2 PID3 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
NODE0 CPU0 PID1 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE1 CPU2 PID3 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE0 CPU0 PID1 SSO over cache line test PASSED
NODE1 CPU2 PID3 SSO over cache line test PASSED
NODE1 CPU2 PID3 memory data bus test PASSED
NODE0 CPU0 PID1 memory data bus test PASSED
NODE0 CPU0 PID1 memory background test PASSED
NODE1 CPU2 PID3 memory background test PASSED
NODE0 CPU0 PID1 memory address bus test PASSED
NODE1 CPU2 PID3 memory address bus test PASSED
NODE0 CPU0 PID1 random data test PASSED
NODE1 CPU2 PID3 random data test PASSED
```

When all test sections have completed successfully, the tests print out a summary for each CPU in the system, as shown in the following example messages:

```
**** MDK memory test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
MDK memory mapped test has completed
```

Note: The last line is for the *mem.qs* test. It changes to `Memory unmapped test has completed` for the *memum.qs* test.

4.4.3.2 Failure Output

If these tests detect errors, they print an error message and related output, similar to the following example messages:

```
ERROR NODEx CPUy PIDx testname FAILED
Data type type
Virtual addr 0x000000000cdb1a07  Physical addr 0xa800000181f05a07
Expected data 0xf9  Actual data 0x79
Syndrom 0x80
Address 0000000081f05a07 bit 7 is on NASID 1
MMYH4 DIMML line 7
```

If either of the quick screen memory tests fails, the failing hardware could be a memory DIMM, a secondary cache (a HIMM), a Node board, a midplane, a cable, or a Router board. The failing FRU could be the memory DIMM, Node board, midplane, cable, or Router board that contains the failing hardware component.

4.4.4 Examples of Running the Quick Screen Memory Tests

This section contains examples of running the *mem.qs* and *memum.qs* quick screen memory tests.

4.4.4.1 Example of Running the *mem.qs* Test

The following example output shows the *mem.qs* test running on a 2-Node system (4 CPUs). In this example, the test completes successfully without detecting any hardware failures.

```
>>boot dksc(0,1,0)/stand/memory.mdk
Booting Nanos.........
************************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
************************************************************

CPU 0: Total no. of CPUs = 4
Launched CPU 1
Launched CPU 2
Launched CPU 3
MDK>mem.qs
Created successfully pid = 1
CPU 0: User mem starts from 0xa800000000641000
CPU0 PID1 Allocating 0x000000000705f000 bytes of memory on NODE0
NODE0 CPU0 PID1 Trying to malloc 0x000000000705f000 bytes
NODE0 CPU0 PID1 malloc SUCCESSFUL for 0x000000000705f000 bytes
vmem_addri 0x000000000060a000 vmem_addrf 0x0000000007669000 size
0x000000000705f000
CPU0 PID1 Allocating 0x0000000007c80000 bytes of memory on NODE1
NODE1 CPU0 PID1 Trying to malloc 0x0000000007c80000 bytes
NODE1 CPU0 PID1 malloc SUCCESSFUL for 0x0000000007c80000 bytes
vmem_addri 0x0000000007669000 vmem_addrf 0x000000000f2e9000 size
0x0000000007c80000
CNODE0 NASID0 CPU0 ppid1
```

```
SCACHE_CONFIG_T:
     sc_size     1
     sc_size_max 6
     sc_bsize    1
     qwo[sbit]   00      qwo[size]   04      qwo[mask]   0x0000000f
     qw[sbit]    04      qw[size]    02      qw[mask]    0x00000003
     index[sbit] 06      index[size] 13      index[mask] 0x00001fff
     data[sbit]  04      data[size]  15      data[mask]  0x00007fff
     tag[sbit]   19      tag[size]   21      tag[mask]   0x001fffff
     way[sbit]   07      way[size]   13      way[mask]   0x00000fff
 Config_t:
     numCpus    4
     numNodes   2
     numRouters 1
         CNODE0 NASID0:
             numCpus  2
             mem size 0x0000000008000000
         CNODE1 NASID1:
             numCpus  2
             mem size 0x0000000008000000
 NODE_MEM_PTR_T:
         CNODE0 NASID0:
         vmem_addri 0x000000000060a000
         vmem_addrf 0x0000000007669000
         pmem_addri 0xa800000000664000
         pmem_addrf 0xa8000000076c3000
         size       0x000000000705f000
         CNODE1 NASID1:
         vmem_addri 0x0000000007669000
         vmem_addrf 0x000000000f2e9000
         pmem_addri 0xa800000100020000
         pmem_addrf 0xa800000107ca0000
         size       0x0000000007c80000
 PROCESS_T:
     Cpu0:
         type   0  READ_WRITE
         status 0  IDLE
     Cpu1:
         type   1  READ_ONLY
         status 0  IDLE
     Cpu2:
         type   0  READ_WRITE
         status 0  IDLE
     Cpu3:
         type   1  READ_ONLY
         status 0  IDLE
CPU0 PID1 Sproc on CPU1
CPU0 PID1 Sproc on CPU2
NODE0 CPU1 PID2 In child
CPU0 PID1 Sproc on CPU3
NODE0 CPU1 PID2 Read only test
NODE1 CPU3 PID4 In child
NODE1 CPU2 PID3 In child
NODE1 CPU3 PID4 Read only test
NODE0 CPU0 PID1 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
NODE1 CPU2 PID3 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
```

```
NODE0 CPU0 PID1 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE1 CPU2 PID3 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE0 CPU0 PID1 SSO over cache line test PASSED
NODE1 CPU2 PID3 SSO over cache line test PASSED
NODE0 CPU1 PID2 Read only test
NODE0 CPU0 PID1 memory data bus test
NODE1 CPU2 PID3 memory data bus test
NODE1 CPU3 PID4 Read only test
NODE1 CPU2 PID3 memory data bus test PASSED
NODE0 CPU0 PID1 memory data bus test PASSED
NODE1 CPU2 PID3 memory background test
NODE0 CPU0 PID1 memory background test
NODE0 CPU0 PID1 memory background test PASSED
NODE0 CPU0 PID1 memory address bus test
NODE0 CPU0 PID1 memory address bus test “Ulong_nec” index 0
NODE1 CPU2 PID3 memory background test PASSED
NODE1 CPU2 PID3 memory address bus test
NODE1 CPU2 PID3 memory address bus test “Ulong_nec” index 0
NODE0 CPU0 PID1 memory address bus test “Ulong_nec” index 1
NODE1 CPU2 PID3 memory address bus test “Ulong_nec” index 1
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 0
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 0
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 1
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 1
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 2
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 2
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 3
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 3
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 4
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 4
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 5
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 5
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 6
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 6
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 7
NODE0 CPU0 PID1 memory address bus test PASSED
NODE0 CPU0 PID1 random data test
NODE0 CPU0 PID1 Loop 0 data type 3
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 7
NODE1 CPU2 PID3 memory address bus test PASSED
NODE1 CPU2 PID3 random data test
NODE1 CPU2 PID3 Loop 0 data type 3
NODE0 CPU0 PID1 random data test PASSED
NODE1 CPU2 PID3 random data test PASSED
**** MDK memory test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
MDK memory mapped test has completed
```

4.4.4.2 Example of Running the *memum.qs* Test

The following example output shows the *memum.qs* test running on a 2-Node system (4 CPUs). In this example, the test completes successfully without detecting any hardware failures.

```
>>boot dksc(0,1,0)/stand/memory.mdk
Booting Nanos.........
************************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
************************************************************

CPU 0: Total no. of CPUs = 4
Launched CPU 1
Launched CPU 2
Launched CPU 3
Created successfully pid = 1
MDK>memum.qs
CPU 0: User mem starts from 0xa800000000641000
CPU0 PID1 Allocating 0x000000000705f000 bytes of memory on NODE0
NODE0 CPU0 PID1 Trying to malloc 0x000000000705f000 bytes
NODE0 CPU0 PID1 malloc SUCCESSFUL for 0x000000000705f000 bytes
vmem_addri 0x000000000060a000 vmem_addrf 0x0000000007669000 size
0x000000000705f000
CPU0 PID1 Allocating 0x0000000007c80000 bytes of memory on NODE1
NODE1 CPU0 PID1 Trying to malloc 0x0000000007c80000 bytes
NODE1 CPU0 PID1 malloc SUCCESSFUL for 0x0000000007c80000 bytes
vmem_addri 0x0000000007669000 vmem_addrf 0x000000000f2e9000 size
0x0000000007c80000
CNODE0 NASID0 CPU0 ppid1
Config_t:
   numCpus    4
   numNodes   2
   numRouters 1
   CNODE000 NASID000:
     numCpus  2
     mem size 0x0000000008000000
     Bank0    0x0000000008000000
     Bank1    0x0000000000000000
     Bank2    0x0000000000000000
     Bank3    0x0000000000000000
     Bank4    0x0000000000000000
     Bank5    0x0000000000000000
     Bank6    0x0000000000000000
     Bank7    0x0000000000000000
   CNODE001 NASID001:
     numCpus  2
     mem size 0x0000000008000000
     Bank0    0x0000000008000000
     Bank1    0x0000000000000000
     Bank2    0x0000000000000000
     Bank3    0x0000000000000000
     Bank4    0x0000000000000000
     Bank5    0x0000000000000000
     Bank6    0x0000000000000000
     Bank7    0x0000000000000000
INFO_T addr of ppid      0xa800000007fde788
INFO_T addr of sysconfig 0xa800000007fde790
```

```
INFO_T addr of procinfo  0xa800000000b11000
INFO_T addr of nodeinfo  0xa800000000b12000
INFO_T addr of addrinfo  0xa800000000b13000
NODE000 BANK0 mem   0x0000000008000000
NODE000 BANK0 saddr 0xa800000000b14000
NODE000 BANK0 eaddr 0xa800000007fbdff8
NODE000 BANK0 size  0x00000000074a9ff8
```

[configuration information for Node 0, banks 1 through 6 (not shown)]

```
NODE000 BANK7 mem   0x0000000000000000
NODE000 BANK7 saddr 0x0000000000000000
NODE000 BANK7 eaddr 0x0000000000000000
NODE000 BANK7 size  0x0000000000000000
NODE001 BANK0 mem   0x0000000008000000
NODE001 BANK0 saddr 0xa800000100020000
NODE001 BANK0 eaddr 0xa800000107fbdff8
NODE001 BANK0 size  0x0000000007f9dff8
```

[configuration information for Node 1, banks 1 through 6 (not shown)]

```
NODE001 BANK7 mem   0x0000000000000000
NODE001 BANK7 saddr 0x0000000000000000
NODE001 BANK7 eaddr 0x0000000000000000
NODE001 BANK7 size  0x0000000000000000
CPU0 Type 0 READ_WRITE Status 0 IDLE
CPU1 Type 1 READ_ONLY Status 0 IDLE
CPU2 Type 0 READ_WRITE Status 0 IDLE
CPU3 Type 1 READ_ONLY S1B 000: tatus 0 IDLE
NODE0 CPU1 PID2 In child
NODE0 CPU0 PID1 unmapped memory phase zero test
NODE0 CPU1 PID2 unmapped memory phase zero test
NODE0 CPU0 PID1 unmapped memory pattern test
NODE0 CPU1 PID2 Read only test
NODE1 CPU2 PID3 In child
NODE1 CPU3 PID4 In child
NODE1 CPU2 PID3 unmapped memory phase zero test
NODE1 CPU3 PID4 unmapped memory phase zero test
NODE1 CPU2 PID3 unmapped memory pattern test
NODE1 CPU3 PID4 Read only test
NODE0 CPU0 PID1 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
NODE0 CPU0 PID1 unmapped memory pattern test
NODE1 CPU2 PID3 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
NODE1 CPU2 PID3 unmapped memory pattern test
NODE0 CPU0 PID1 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE0 CPU0 PID1 unmapped memory SSO over cahe line test
NODE1 CPU2 PID3 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE1 CPU2 PID3 unmapped memory SSO over cahe line test
NODE0 CPU0 PID1 SSO over cache line test PASSED
NODE0 CPU0 PID1 unmapped memory phase zero test PASSED
NODE1 CPU2 PID3 SSO over cache line test PASSED
NODE1 CPU2 PID3 unmapped memory phase zero test PASSED
NODE0 CPU1 PID2 unmapped memory phase zero test PASSED
```

```
NODE1 CPU3 PID4 unmapped memory phase zero test PASSED
NODE0 CPU1 PID2 unmapped memory phase one test
NODE0 CPU0 PID1 unmapped memory phase one test
NODE0 CPU1 PID2 Read only test
NODE0 CPU0 PID1 unmapped memory background test
NODE1 CPU3 PID4 unmapped memory phase one test
NODE1 CPU2 PID3 unmapped memory phase one test
NODE1 CPU3 PID4 Read only test
NODE1 CPU2 PID3 unmapped memory background test
NODE0 CPU0 PID1 unmapped memory background test PASSED
NODE0 CPU0 PID1 unmapped memory cell test
NODE0 CPU0 PID1 unmapped memory cell test Ulong_NEC bank 0
NODE1 CPU2 PID3 unmapped memory background test PASSED
NODE1 CPU2 PID3 unmapped memory cell test
NODE1 CPU2 PID3 unmapped memory cell test Ulong_NEC bank 0
NODE0 CPU0 PID1 unmapped memory cell test PASSED
NODE0 CPU0 PID1 unmapped memory phase one test PASSED
NODE1 CPU2 PID3 unmapped memory cell test PASSED
NODE1 CPU2 PID3 unmapped memory phase one test PASSED
NODE0 CPU1 PID2 unmapped memory phase one test PASSED
NODE1 CPU3 PID4 unmapped memory phase one test PASSED
**** MDK memory unmapped test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
Memory unmapped test has completed
```

4.5 Node Board Memory Test

The Node board memory test uses cache line sharing without internode access. This enables it to test cache coherence and to perform cache thrashing. This test does not generate internode traffic. There is only an unmapped version of the Node board memory test (*memum.nb*).

4.5.1 How to Run the Node Board Memory Test

Use the following procedure to load the test from the local system disk:

- Use the *boot* command from the PROM monitor (at the BaseIO command prompt, >>) to start the *memory.mdk* test package:

  ```
  >>boot dksc(0,1,0)/stand/memory.mdk
  ```

  MDK displays boot status information and the `MDK>` prompt.

- Load the *memum.nb* test from the `MDK>` prompt:

  ```
  MDK>memum.nb
  ```

  The Node board memory test loads into the system and begins testing memory.

4.5.2 Test Description

The Node board memory test uses the following algorithm:

- Acquire the system configuration.
  1. Determine the amount of memory that is populated in the system.
  2. Determine the number of CPUs and Nodes in the system.
  3. Scale the test to the system configuration.
- Select one CPU to perform only read operations: This CPU causes memory traffic while the test is running.
- Select one CPU to perform read and write operations: This CPU does the actual memory testing.

Note: If there is only one CPU in the system, it runs two processes. One of the processes performs only read operations; the other process performs read and write operations.

- Generate random addresses and data that the test will use.
- Split each cache line in half: CPU0 uses the lower 64 bytes of the cache line, and CPU1 uses the upper 64 bytes of the cache line.
- Have each CPU write data to its half of each cache line.
- Have each CPU read the data back from its half of the cache lines.
- Compare the data that was read back to the data that was written and verify that both sets of data are the same.

4.5.3 Output from the Node Board Memory Test

The Node board memory test returns output to the BaseIO console.

4.5.3.1 Pass Output

As the test runs, it prints out a PASSED message for each test section that completes without detecting hardware failures. The following examples show some of these messages:

```
NODE1 CPU3 PID4 unmapped memory random address and data test PASSED
NODE1 CPU2 PID3 unmapped memory random address and data test PASSED
NODE0 CPU0 PID1 unmapped memory random address and data test PASSED
NODE0 CPU1 PID2 unmapped memory random address and data test PASSED
```

When all test sections have completed successfully, the test prints out a summary for each CPU in the system, as shown in the following example messages:

```
**** MDK memory unmapped test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
Memory unmapped test has completed
```

4.5.3.2 Failure Output

If this test detects errors, it prints an error message and related output, similar to the following example messages:

```
----------------------------------------
ERROR CNODE3 CPU6 PID7 memUmCnvrgAddrAndFlippingDataWr FAILED
Module 2 Slot 1 Nasid 4 Bank 0 Physical addr 0xa8000004000205a8
Expected data ----- 0xa800000403ffffa8
Actual data ------- 0xa800000403ffffa8
Corrupted data bits 0x0000000000000000
----------------------------------------
```

If the Node board memory test fails, the failing hardware could be a memory DIMM, a secondary cache (a HIMM), or a Node board. The failing FRU could be the memory DIMM, Node board, or Router board that contains the failing hardware component.

4.5.4 Example of Running the Node Board Memory Test

The following example output shows the Node board memory test running on a 2-Node system (4 CPUs). In this example, the test completes successfully without detecting any hardware failures.

```
>>boot dksc(0,1,0)/stand/memory.mdk
Booting Nanos.........

************************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
************************************************************

CPU 0: Total no. of CPUs = 4
Launched CPU 1
Launched CPU 2
Launched CPU 3
MDK>memum.nb
CNODE 0 NASID 0 SLOT 1 MODULE 1
CNODE 1 NASID 1 SLOT 2 MODULE 1
Config_t:
  numCpus    4
  numNodes   2
  numRouters 0
  CNODE000 NASID000:
    numCpus  2
    mem size 0x0000000008000000
    Bank0    0x0000000004000000
    Bank1    0x0000000004000000
  CNODE001 NASID001:
    numCpus  2
    mem size 0x0000000008000000
    Bank0    0x0000000004000000
    Bank1    0x0000000004000000
INFO_T addr of ppid      0xa800000023fde588
INFO_T addr of sysconfig 0xa800000023fde590
INFO_T addr of procinfo  0xa800000000b0a000
INFO_T addr of nodeinfo  0xa800000000b0b000
INFO_T addr of addrinfo  0xa800000000b0c000
CNODE000 BANK0 mem       0x0000000004000000
    CNODE000 BANK0 saddr 0xa800000000b0f000
    CNODE000 BANK0 eaddr 0xa800000004000000
    CNODE000 BANK0 size  0x00000000034f1000
CNODE000 BANK1 mem       0x0000000004000000
    CNODE000 BANK1 saddr 0xa800000020000000
    CNODE000 BANK1 eaddr 0xa800000023fbdff8
    CNODE000 BANK1 size  0x0000000003fbdff8
CNODE001 BANK0 mem       0x0000000004000000
    CNODE001 BANK0 saddr 0xa800000100020000
    CNODE001 BANK0 eaddr 0xa800000104000000
    CNODE001 BANK0 size  0x0000000003fe0000
CNODE001 BANK1 mem       0x0000000004000000
    CNODE001 BANK1 saddr 0xa800000120000000
    CNODE001 BANK1 eaddr 0xa800000123fbdff8
    CNODE001 BANK1 size  0x0000000003fbdff8
PID 1 CPU 0 CNODE 0 NASID 0
PID 2 CPU 1 CNODE 0 NASID 0
```

```
PID 3 CPU 2 CNODE 1 NASID 1
PID 4 CPU 3 CNODE 1 NASID 1
CNODE0 CPU0 PID1 unmapped memory phase two test
CNODE0 CPU1 PID2 unmapped memory phase two test
CNODE1 CPU3 PID4 unmapped memory phase two test
CNODE0 CPU1 PID2 Unmapped memory random address and data test
CNODE0 CPU0 PID1 Unmapped memory random address and data test
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 0
CNODE1 CPU3 PID4 Unmapped memory random address and data test
CNODE1 CPU2 PID3 unmapped memory phase two test
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 0
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 0
CNODE1 CPU2 PID3 Unmapped memory random address and data test
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 0
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 5000
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 10000
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 15000
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 20000
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 25000
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 5000
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 30000
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 5000
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 5000
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 35000
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 40000
CNODE1 CPU3 PID4 unmapped memory random addr and data test Loop 45000
CNODE1 CPU3 PID4 unmapped memory random address and data test PASSED
CNODE1 CPU3 PID4 Unmapped memory converging address and flipping data
test
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 0
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 5000
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 10000
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 15000
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 10000
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 20000
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 25000
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 30000
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 10000
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 35000
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 10000
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 40000
CNODE1 CPU3 PID4 cnvrg addr and flip data test loop 45000
CNODE1 CPU3 PID4 unmapped memory converging address and flipping data
test PASSED
CNODE0 CPU3 PID1 Done (Node Board). [3] to go before inter node test
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 15000
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 15000
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 15000
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 20000
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 20000
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 20000
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 25000
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 25000
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 25000
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 30000
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 30000
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 35000
```

```
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 30000
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 35000
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 40000
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 40000
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 35000
CNODE1 CPU2 PID3 unmapped memory random addr and data test Loop 45000
CNODE0 CPU0 PID1 unmapped memory random addr and data test Loop 45000
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 40000
CNODE1 CPU2 PID3 unmapped memory random address and data test PASSED
CNODE1 CPU2 PID3 Unmapped memory converging address and flipping data
test
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 0
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 5000
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 10000
CNODE0 CPU0 PID1 unmapped memory random address and data test PASSED
CNODE0 CPU0 PID1 Unmapped memory converging address and flipping data
test
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 0
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 15000
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 5000
CNODE0 CPU1 PID2 unmapped memory random addr and data test Loop 45000
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 20000
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 10000
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 25000
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 15000
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 30000
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 20000
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 35000
CNODE0 CPU1 PID2 unmapped memory random address and data test PASSED
CNODE0 CPU1 PID2 Unmapped memory converging address and flipping data
test
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 0
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 25000
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 40000
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 5000
CNODE1 CPU2 PID3 cnvrg addr and flip data test loop 45000
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 30000
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 10000
CNODE1 CPU2 PID3 unmapped memory converging address and flipping data
test PASSED
CNODE0 CPU2 PID1 Done (Node Board). [2] to go before inter node test
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 35000
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 15000
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 40000
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 20000
CNODE0 CPU0 PID1 cnvrg addr and flip data test loop 45000
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 25000
CNODE0 CPU0 PID1 unmapped memory converging address and flipping data
test PASSED
CNODE0 CPU0 PID1 Done (Node Board). [1] to go before inter node test
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 30000
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 35000
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 40000
CNODE0 CPU1 PID2 cnvrg addr and flip data test loop 45000
CNODE0 CPU1 PID2 unmapped memory converging address and flipping data
test PASSED
CNODE0 CPU1 PID1 Done (Node Board). [0] to go before inter node test
```

```
################ DIR SBECC SUMMARY ################

No directory ecc errs detected


################ MEM SBECC SUMMARY ################

No memory ecc errs detected


**** MDK memory unmapped test status summary ****

CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
Memory unmapped test has completed
```

4.6 Internode Memory Test

The internode memory test uses all Nodes and CPUs in the system to achieve very aggressive interaction of cache coherence protocol. There is only an unmapped version of the internode memory test (*memum.in*).

4.6.1 How to Run the Internode Memory Test

Use the following procedure to load the test from the local system disk:

- Use the *boot* command from the PROM monitor (at the BaseIO command prompt, >>) to start the *memory.mdk* test package:

  ```
  >>boot dksc(0,1,0)/stand/memory.mdk
  ```

  MDK displays boot status information and the MDK> prompt.

- Load the *memum.in* test from the MDK> prompt:

  ```
  MDK>memum.in
  ```

  The internode memory test loads into the system and begins testing memory.

4.6.2 Internode Memory Test Description

The internode memory test uses the following algorithm:

- Acquire the system configuration.
  1. Determine the amount of memory that is populated in the system.
  2. Determine the number of CPUs and Nodes in the system.
- Determine the size of the data block and divide the data block between the CPUs.
  1. If the number of CPUs in the system is greater than 128, the size of the data block equals the number of CPUs (in bytes). Each CPU owns a byte of the data block to which the CPU can write data.
  2. If the number of CPUs in the system is less than or equal to 128, the size of the data block is 128 bytes. Each CPU owns (128 divided by the number of CPUs) bytes of the data block to which the CPU can write data.
- Have each CPU write data to its portion of the data block.
  1. Randomly select a Node.
  2. Repeat the following actions until unique addresses can no longer be generated:
     - Randomly select a variable number of bits in the address that will randomly change. The remaining bits in the address do not change.
     - Fill the selected bits with random data to generate a random address.
     - Write random data to the memory address.
- Have each CPU read data back from its portion of the data block.
- Verify that the data that was read back is the same as the data that was written.

4.6.3 Output from the Internode Memory Test

The internode memory test returns output to the BaseIO console.

4.6.3.1 Pass Output

As the test runs, it prints out a PASSED message for each test section that completes without detecting hardware failures. The following examples show some of these messages:

```
NODE0 CPU0 PID1 inter node test PASSED
NODE1 CPU3 PID4 inter node test PASSED
NODE0 CPU1 PID2 inter node test PASSED
NODE1 CPU2 PID3 inter node test PASSED
```

When all test sections have completed successfully, the test prints out a summary for each CPU in the system, as shown in the following example messages:

```
**** MDK memory test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
Memory unmapped test has completed
```

4.6.3.2 Failure Output

If this test detects errors, it prints an ERROR message and related output, similar to the following example messages:

```
----------------------------------------
ERROR CNODE0 CPU0 PID1 memUmInterNodeRandomAddressAndDataCompare FAILED
in loop 0
Module 1 Slot 1 Nasid 0 Bank 0 Physical addr 0xa800000002114887
Expected data ----- 0x00000000000000fa
Actual data ------- 0x00000000000000fa
Corrupted data bits 0x0000000000000000
----------------------------------------
```

If the internode memory test fails, the failing hardware could be a memory DIMM, a secondary cache (a HIMM), a Node board, a midplane, a cable, or a Router board. The failing FRU could be the memory DIMM, Node board, midplane, cable, or Router board that contains the failing hardware component.

4.6.4 Example of Running the Internode Memory Test

The following example output shows the internode memory test running on a 2-Node system (4 CPUs). In this example, the test completes successfully without detecting any hardware failures.

```
>>boot dksc(0,1,0)/stand/memory.mdk
Booting Nanos.........

************************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
************************************************************

CPU 0: Total no. of CPUs = 4
Launched CPU 1
Launched CPU 2
Launched CPU 3
MDK>memum.in
CNODE 0 NASID 0 SLOT 1 MODULE 1
CNODE 1 NASID 1 SLOT 2 MODULE 1
Config_t:
  numCpus    4
  numNodes   2
  numRouters 0
  CNODE000 NASID000:
    numCpus  2
    mem size 0x0000000008000000
    Bank0    0x0000000004000000
    Bank1    0x0000000004000000
  CNODE001 NASID001:
    numCpus  2
    mem size 0x0000000008000000
    Bank0    0x0000000004000000
    Bank1    0x0000000004000000
INFO_T addr of ppid      0xa800000023fde588
INFO_T addr of sysconfig 0xa800000023fde590
INFO_T addr of procinfo  0xa800000000b0a000
INFO_T addr of nodeinfo  0xa800000000b0b000
INFO_T addr of addrinfo  0xa800000000b0c000
CNODE000 BANK0 mem       0x0000000004000000
    CNODE000 BANK0 saddr 0xa800000000b0f000
    CNODE000 BANK0 eaddr 0xa800000004000000
    CNODE000 BANK0 size  0x00000000034f1000
CNODE000 BANK1 mem       0x0000000004000000
    CNODE000 BANK1 saddr 0xa800000020000000
    CNODE000 BANK1 eaddr 0xa800000023fbdff8
    CNODE000 BANK1 size  0x0000000003fbdff8
CNODE001 BANK0 mem       0x0000000004000000
    CNODE001 BANK0 saddr 0xa800000100020000
    CNODE001 BANK0 eaddr 0xa800000104000000
    CNODE001 BANK0 size  0x0000000003fe0000
CNODE001 BANK1 mem       0x0000000004000000
    CNODE001 BANK1 saddr 0xa800000120000000
    CNODE001 BANK1 eaddr 0xa800000123fbdff8
    CNODE001 BANK1 size  0x0000000003fbdff8
PID 1 CPU 0 CNODE 0 NASID 0
PID 2 CPU 1 CNODE 0 NASID 0
```

```
PID 3 CPU 2 CNODE 1 NASID 1
PID 4 CPU 3 CNODE 1 NASID 1
CNODE0 CPU1 PID2 Unmapped memory inter node test
CNODE0 CPU0 PID1 Unmapped memory inter node test
CNODE0 CPU1 PID2 Loop 0
CNODE1 CPU2 PID3 Unmapped memory inter node test
CNODE0 CPU0 PID1 Loop 0
CNODE1 CPU2 PID3 Loop 0
CNODE1 CPU3 PID4 Unmapped memory inter node test
CNODE1 CPU3 PID4 Loop 0
01
01
01
01
01
CNODE1 CPU2 PID3 Loop 50
CNODE1 CPU3 PID4 Loop 50
CNODE0 CPU1 PID2 Loop 50
CNODE0 CPU0 PID1 Loop 50
01
01
01
01
01
CNODE1 CPU3 PID4 Loop 100
CNODE1 CPU2 PID3 Loop 100
CNODE0 CPU1 PID2 Loop 100
CNODE0 CPU0 PID1 Loop 100
01
01
01
01
01
CNODE1 CPU2 PID3 Loop 150
CNODE1 CPU3 PID4 Loop 150
CNODE0 CPU1 PID2 Loop 150
CNODE0 CPU0 PID1 Loop 150
01
01
01
01
01
CNODE1 CPU2 PID3 Loop 200
CNODE1 CPU3 PID4 Loop 200
CNODE0 CPU0 PID1 Loop 200
CNODE0 CPU1 PID2 Loop 200
01
01
01
01
01
CNODE1 CPU2 PID3 Loop 250
CNODE1 CPU3 PID4 Loop 250
CNODE0 CPU0 PID1 Loop 250
CNODE0 CPU1 PID2 Loop 250
```

```
01
01
01
01
01
CNODE1 CPU2 PID3 Loop 300
CNODE1 CPU3 PID4 Loop 300
CNODE0 CPU0 PID1 Loop 300
CNODE0 CPU1 PID2 Loop 300
01
01
01
01
01
CNODE1 CPU2 PID3 Unmapped memory inter node test PASSED
CNODE1 CPU3 PID4 Unmapped memory inter node test PASSED
CNODE0 CPU1 PID2 Unmapped memory inter node test PASSED
CNODE0 CPU0 PID1 Unmapped memory inter node test PASSED

############### DIR SBECC SUMMARY ###############

No directory ecc errs detected


############### MEM SBECC SUMMARY ###############

No memory ecc errs detected


**** MDK memory unmapped test status summary ****

CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
Memory unmapped test has completed
```

Note: The lines that contain `01` are heartbeat values that the test prints while it runs.

4.7 Long Memory Tests

The long memory tests run extended versions of the directory memory, quick screen, Node board, and internode memory tests. The long memory tests are *mem.lo* (mapped version) and *memum.lo* (unmapped version).

4.7.1 How to Run the Long Memory Tests

Use the following procedure to load the long memory tests from the local system disk:

- Use the *boot* command from the PROM monitor (at the BaseIO command prompt, >= to start the *memory.mdk* test package:

  >>**boot dksc(0,1,0)/stand/memory.mdk**

  MDK displays boot status information and the MDK> prompt.

- Load the mapped version (*mem.lo*) or unmapped version (*memum.lo*) from the MDK> prompt:

  1. To load the mapped version, enter *mem.lo* at the MDK> prompt:

     MDK>**mem.lo**

  2. To load the unmapped version, enter *memum.lo* at the MDK> prompt:

     MDK>**memum.lo**

  The specified long memory test loads into the system and begins testing memory.

4.7.2 Test Description

Both long memory tests use the following algorithm:

- Acquire the system configuration.
  1. Determine the amount of memory that is populated in the system.
  2. Determine the number of CPUs and Nodes in the system.
  3. Scale the test to the system configuration.
- Run an enhanced version of the directory memory test. (This version loops through the exclusive and shared tests and then also runs a random test. In the random test, each processor randomly selects exclusive mode or shared mode and then accesses memory.)
- Run an enhanced version of the quick screen memory test.

  The address bus test uses the following additional data patterns:

  0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3 0x6996699669966996 0x9669966996699669 0xaaaaaaaaaaaaaaaa 0x5555555555555555 0x0000000000000000 0xffffffffffffffff

- Run 50,000 loops of the Node board memory test.
- Run 350 loops of the internode memory test.

4.7.3 Output from the Long Memory Tests

The long memory tests return output to the BaseIO console.

4.7.3.1 Pass Output

As the tests run, they print out a `PASSED` message for each test section that completes without detecting hardware failures. The following examples show some of these messages:

```
NODE0 CPU0 PID1 SSO over cache line test PASSED
NODE1 CPU2 PID3 SSO over cache line test PASSED
NODE1 CPU2 PID3 memory data bus test PASSED
NODE0 CPU0 PID1 memory data bus test PASSED
NODE0 CPU0 PID1 memory background test PASSED
NODE1 CPU2 PID3 memory background test PASSED
NODE0 CPU0 PID1 memory address bus test PASSED
NODE1 CPU2 PID3 memory address bus test PASSED
NODE0 CPU0 PID1 random data test PASSED
NODE1 CPU2 PID3 random data test PASSED
NODE0 CPU0 PID1 random address and data test PASSED
NODE0 CPU1 PID2 random address and data test PASSED
NODE1 CPU3 PID4 random address and data test PASSED
NODE1 CPU2 PID3 random address and data test PASSED
NODE0 CPU0 PID1 inter node test PASSED
NODE1 CPU3 PID4 inter node test PASSED
NODE0 CPU1 PID2 inter node test PASSED
NODE1 CPU2 PID3 inter node test PASSED
```

When all test sections have completed successfully, the tests print out a summary for each CPU in the system, as shown in the following example messages:

```
**** MDK memory test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
MDK memory mapped test has completed
```

Note: The last line is for the *mem.lo* test. It changes to `Memory unmapped test has completed` for the *memum.lo* test.

4.7.3.2 Failure Output

If these tests detect errors, they print an error message and related output, similar to the following example messages:

```
-----------------------------------------
ERROR CNODE0 CPU0 PID1 memUmPatternTest FAILED
Module 1 Slot 1 Nasid 0 Bank 0 Physical addr 0xa800000002115000
Expected data ----- 0xffffffffffffffff
Actual data ------- 0xffffffffffffffff
Corrupted data bits 0x0000000000000000
-----------------------------------------
```

If either of the long memory tests fails, the failing hardware could be a memory DIMM, a secondary cache (a HIMM), a Node board, a midplane, a cable, or a Router board. The failing FRU could be the memory DIMM, Node board, midplane, cable, or Router board that contains the failing hardware component.

4.7.4 Examples of Running the Long Memory Tests

This section contains examples of running the *mem.lo* and *memum.lo* long memory tests.

4.7.4.1 Example of Running the *mem.lo* Test

The following example output shows the *mem.lo* test running on a 2-Node system (4 CPUs). In this example, the test completes successfully without detecting any hardware failures.

```
>>boot dksc(0,1,0)/stand/memory.mdk
Booting Nanos.........
************************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
************************************************************

CPU 0: Total no. of CPUs = 4
Launched CPU 1
Launched CPU 2
Launched CPU 3
MDK>mem.lo
Created successfully pid = 1
CPU 0: User mem starts from 0xa800000000641000
CPU0 PID1 Allocating 0x000000000705f000 bytes of memory on NODE0
NODE0 CPU0 PID1 Trying to malloc 0x000000000705f000 bytes
NODE0 CPU0 PID1 malloc SUCCESSFUL for 0x000000000705f000 bytes
vmem_addri 0x000000000060a000 vmem_addrf 0x0000000007669000 size
0x000000000705f000
CPU0 PID1 Allocating 0x0000000007c80000 bytes of memory on NODE1
NODE1 CPU0 PID1 Trying to malloc 0x0000000007c80000 bytes
NODE1 CPU0 PID1 malloc SUCCESSFUL for 0x0000000007c80000 bytes
vmem_addri 0x0000000007669000 vmem_addrf 0x000000000f2e9000 size
0x0000000007c80000
CNODE0 NASID0 CPU0 ppid1
SCACHE_CONFIG_T:
     sc_size     1
     sc_size_max 6
     sc_bsize    1
     qwo[sbit]   00      qwo[size]   04      qwo[mask]   0x0000000f
     qw[sbit]    04      qw[size]    02      qw[mask]    0x00000003
     index[sbit] 06      index[size] 13      index[mask] 0x00001fff
     data[sbit]  04      data[size]  15      data[mask]  0x00007fff
     tag[sbit]   19      tag[size]   21      tag[mask]   0x001fffff
     way[sbit]   07      way[size]   13      way[mask]   0x00000fff
 Config_t:
     numCpus    4
     numNodes   2
     numRouters 1
         CNODE0 NASID0:
             numCpus  2
             mem size 0x0000000008000000
```

```
        CNODE1 NASID1:
            numCpus  2
            mem size 0x0000000008000000
 NODE_MEM_PTR_T:
        CNODE0 NASID0:
        vmem_addri 0x000000000060a000
        vmem_addrf 0x0000000007669000
        pmem_addri 0xa800000000664000
        pmem_addrf 0xa8000000076c3000
        size       0x000000000705f000
        CNODE1 NASID1:
        vmem_addri 0x0000000007669000
        vmem_addrf 0x000000000f2e9000
        pmem_addri 0xa800000100020000
        pmem_addrf 0xa800000107ca0000
        size       0x0000000007c80000
 PROCESS_T:
    Cpu0:
        type   0  READ_WRITE
        status 0  IDLE
    Cpu1:
        type   1  READ_ONLY
        status 0  IDLE
    Cpu2:
        type   0  READ_WRITE
        status 0  IDLE
    Cpu3:
        type   1  READ_ONLY
        status 0  IDLE
NODE0 CPU0 PID1 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
NODE1 CPU2 PID3 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
NODE0 CPU0 PID1 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE1 CPU2 PID3 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE0 CPU0 PID1 SSO over cache line test PASSED
NODE1 CPU2 PID3 SSO over cache line test PASSED
NODE1 CPU3 PID4 Read only test
NODE0 CPU0 PID1 memory data bus test
NODE0 CPU1 PID2 Read only test
NODE1 CPU2 PID3 memory data bus test
NODE1 CPU2 PID3 memory data bus test PASSED
NODE0 CPU0 PID1 memory data bus test PASSED
NODE1 CPU2 PID3 memory background test
NODE0 CPU0 PID1 memory background test
NODE0 CPU0 PID1 memory background test PASSED
NODE0 CPU0 PID1 memory address bus test
NODE0 CPU0 PID1 memory address bus test “Ulong_nec” index 0
NODE1 CPU2 PID3 memory background test PASSED
NODE1 CPU2 PID3 memory address bus test
NODE1 CPU2 PID3 memory address bus test “Ulong_nec” index 0
NODE0 CPU0 PID1 memory address bus test “Ulong_nec” index 1
NODE1 CPU2 PID3 memory address bus test “Ulong_nec” index 1
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 0
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 0
```

```
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 1
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 1
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 2
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 2
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 3
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 3
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 4
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 4
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 5
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 5
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 6
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 6
NODE0 CPU0 PID1 memory address bus test “Ulong_NEC” index 7
NODE0 CPU0 PID1 memory address bus test “Ulong” index 0
NODE1 CPU2 PID3 memory address bus test “Ulong_NEC” index 7
NODE0 CPU0 PID1 memory address bus test “Ulong” index 1
NODE1 CPU2 PID3 memory address bus test “Ulong” index 0
NODE0 CPU0 PID1 memory address bus test “Ulong” index 2
NODE1 CPU2 PID3 memory address bus test “Ulong” index 1
NODE0 CPU0 PID1 memory address bus test “Ulong” index 3
NODE1 CPU2 PID3 memory address bus test “Ulong” index 2
NODE0 CPU0 PID1 memory address bus test “Ulong” index 4
NODE1 CPU2 PID3 memory address bus test “Ulong” index 3
NODE0 CPU0 PID1 memory address bus test “Ulong” index 5
NODE1 CPU2 PID3 memory address bus test “Ulong” index 4
NODE0 CPU0 PID1 memory address bus test “Ulong” index 6
NODE1 CPU2 PID3 memory address bus test “Ulong” index 5
NODE0 CPU0 PID1 memory address bus test PASSED
NODE0 CPU0 PID1 random data test
NODE0 CPU0 PID1 Loop 0 data type 3
NODE1 CPU2 PID3 memory address bus test “Ulong” index 6
NODE1 CPU2 PID3 memory address bus test PASSED
NODE1 CPU2 PID3 random data test
NODE1 CPU2 PID3 Loop 0 data type 3
NODE0 CPU0 PID1 Loop 10 data type 3
NODE1 CPU2 PID3 Loop 10 data type 3
NODE0 CPU0 PID1 Loop 20 data type 3
NODE1 CPU2 PID3 Loop 20 data type 3
NODE0 CPU0 PID1 Loop 30 data type 3
NODE1 CPU2 PID3 Loop 30 data type 3
NODE0 CPU0 PID1 Loop 40 data type 3
NODE1 CPU2 PID3 Loop 40 data type 3
NODE0 CPU0 PID1 random data test PASSED
NODE1 CPU2 PID3 random data test PASSED
NODE0 CPU0 PID1 random address and data test
NODE0 CPU1 PID2 random address and data test
NODE0 CPU0 PID1 Loop 0
NODE0 CPU1 PID2 Loop 0
NODE1 CPU3 PID4 random address and data test
NODE1 CPU2 PID3 random address and data test
NODE1 CPU3 PID4 Loop 0
NODE1 CPU2 PID3 Loop 0
NODE0 CPU0 PID1 Loop 5000
NODE1 CPU3 PID4 Loop 5000
NODE0 CPU1 PID2 Loop 5000
NODE1 CPU2 PID3 Loop 5000
NODE0 CPU0 PID1 Loop 10000
```

```
NODE1 CPU3 PID4 Loop 10000
NODE0 CPU1 PID2 Loop 10000
NODE1 CPU2 PID3 Loop 10000
NODE0 CPU0 PID1 Loop 15000
NODE1 CPU3 PID4 Loop 15000
NODE0 CPU1 PID2 Loop 15000
NODE0 CPU0 PID1 Loop 20000
NODE1 CPU3 PID4 Loop 20000
NODE1 CPU2 PID3 Loop 15000
NODE0 CPU1 PID2 Loop 20000
NODE0 CPU0 PID1 Loop 25000
NODE1 CPU3 PID4 Loop 25000
NODE0 CPU0 PID1 Loop 30000
NODE1 CPU2 PID3 Loop 20000
NODE0 CPU1 PID2 Loop 25000
NODE0 CPU0 PID1 Loop 35000
NODE1 CPU3 PID4 Loop 30000
NODE0 CPU1 PID2 Loop 30000
NODE1 CPU2 PID3 Loop 25000
NODE0 CPU0 PID1 Loop 40000
NODE1 CPU3 PID4 Loop 35000
NODE0 CPU1 PID2 Loop 35000
NODE0 CPU0 PID1 Loop 45000
NODE1 CPU2 PID3 Loop 30000
NODE1 CPU3 PID4 Loop 40000
NODE0 CPU1 PID2 Loop 40000
NODE0 CPU0 PID1 random address and data test PASSED
NODE1 CPU3 PID4 Loop 45000
NODE1 CPU2 PID3 Loop 35000
NODE0 CPU1 PID2 Loop 45000
NODE0 CPU1 PID2 random address and data test PASSED
NODE1 CPU3 PID4 random address and data test PASSED
NODE1 CPU2 PID3 Loop 40000
NODE1 CPU2 PID3 Loop 45000
NODE1 CPU2 PID3 random address and data test PASSED
NODE0 CPU1 PID2 inter node test
NODE0 CPU0 PID1 inter node test
NODE0 CPU1 PID2 Loop 0
NODE1 CPU3 PID4 inter node test
NODE0 CPU0 PID1 Loop 0
NODE1 CPU2 PID3 inter node test
NODE1 CPU3 PID4 Loop 0
NODE1 CPU2 PID3 Loop 0
NODE0 CPU0 PID1 Loop 5000
NODE1 CPU3 PID4 Loop 5000
NODE0 CPU1 PID2 Loop 5000
NODE1 CPU2 PID3 Loop 5000
NODE0 CPU0 PID1 Loop 10000
NODE1 CPU3 PID4 Loop 10000
NODE0 CPU1 PID2 Loop 10000
NODE1 CPU2 PID3 Loop 10000
NODE0 CPU0 PID1 Loop 15000
NODE1 CPU3 PID4 Loop 15000
NODE0 CPU0 PID1 Loop 20000
NODE1 CPU3 PID4 Loop 20000
NODE0 CPU1 PID2 Loop 15000
NODE1 CPU2 PID3 Loop 15000
```

```
NODE0 CPU1 PID2 Loop 20000
NODE1 CPU2 PID3 Loop 20000
NODE0 CPU0 PID1 Loop 25000
NODE1 CPU3 PID4 Loop 25000
NODE0 CPU0 PID1 Loop 30000
NODE1 CPU3 PID4 Loop 30000
NODE0 CPU1 PID2 Loop 25000
NODE0 CPU0 PID1 Loop 35000
NODE1 CPU2 PID3 Loop 25000
NODE1 CPU3 PID4 Loop 35000
NODE0 CPU1 PID2 Loop 30000
NODE1 CPU2 PID3 Loop 30000
NODE0 CPU0 PID1 Loop 40000
NODE0 CPU1 PID2 Loop 35000
NODE1 CPU3 PID4 Loop 40000
NODE0 CPU0 PID1 Loop 45000
NODE1 CPU2 PID3 Loop 35000
NODE1 CPU3 PID4 Loop 45000
NODE0 CPU0 PID1 inter node test PASSED
NODE0 CPU1 PID2 Loop 40000
NODE0 CPU1 PID2 Loop 45000
NODE1 CPU3 PID4 inter node test PASSED
NODE1 CPU2 PID3 Loop 40000
NODE1 CPU2 PID3 Loop 45000
NODE0 CPU1 PID2 inter node test PASSED
NODE1 CPU2 PID3 inter node test PASSED
**** MDK memory test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
MDK memory mapped test has completed
```

4.7.4.2 Example of Running the *memum.lo* Test

The following example output shows the *memum.lo* test running on a 2-Node system (4 CPUs). In this example, the test completes successfully without detecting any hardware failures.

```
>>boot dksc(0,1,0)/stand/memory.mdk
Booting Nanos.........
************************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
************************************************************

CPU 0: Total no. of CPUs = 4
Launched CPU 1
Launched CPU 2
Launched CPU 3
MDK>memum.lo
Created successfully pid = 1
CPU 0: User mem starts from 0xa800000000641000
CPU0 PID1 Allocating 0x000000000705f000 bytes of memory on NODE0
NODE0 CPU0 PID1 Trying to malloc 0x000000000705f000 bytes
NODE0 CPU0 PID1 malloc SUCCESSFUL for 0x000000000705f000 bytes
```

```
vmem_addri 0x000000000060a000 vmem_addrf 0x0000000007669000 size
0x000000000705f000
CPU0 PID1 Allocating 0x0000000007c80000 bytes of memory on NODE1
NODE1 CPU0 PID1 Trying to malloc 0x0000000007c80000 bytes
NODE1 CPU0 PID1 malloc SUCCESSFUL for 0x0000000007c80000 bytes
vmem_addri 0x0000000007669000 vmem_addrf 0x000000000f2e9000 size
0x0000000007c80000
CNODE0 NASID0 CPU0 ppid1
Config_t:
   numCpus    4
   numNodes   2
   numRouters 1
   CNODE000 NASID000:
     numCpus  2
     mem size 0x0000000008000000
     Bank0    0x0000000008000000
     Bank1    0x0000000000000000
     Bank2    0x0000000000000000
     Bank3    0x0000000000000000
     Bank4    0x0000000000000000
     Bank5    0x0000000000000000
     Bank6    0x0000000000000000
     Bank7    0x0000000000000000
   CNODE001 NASID001:
     numCpus  2
     mem size 0x0000000008000000
     Bank0    0x0000000008000000
     Bank1    0x0000000000000000
     Bank2    0x0000000000000000
     Bank3    0x0000000000000000
     Bank4    0x0000000000000000
     Bank5    0x0000000000000000
     Bank6    0x0000000000000000
     Bank7    0x0000000000000000
INFO_T addr of ppid      0xa800000007fde788
INFO_T addr of sysconfig 0xa800000007fde790
INFO_T addr of procinfo  0xa800000000b11000
INFO_T addr of nodeinfo  0xa800000000b12000
INFO_T addr of addrinfo  0xa800000000b13000
NODE000 BANK0 mem   0x0000000008000000
NODE000 BANK0 saddr 0xa800000000b14000
NODE000 BANK0 eaddr 0xa800000007fbdff8
NODE000 BANK0 size  0x00000000074a9ff8
```

[configuration information for Node 0, banks 1 through 6 (not shown)]

```
NODE000 BANK7 mem   0x0000000000000000
NODE000 BANK7 saddr 0x0000000000000000
NODE000 BANK7 eaddr 0x0000000000000000
NODE000 BANK7 size  0x0000000000000000
NODE001 BANK0 mem   0x0000000008000000
NODE001 BANK0 saddr 0xa800000100020000
NODE001 BANK0 eaddr 0xa800000107fbdff8
NODE001 BANK0 size  0x0000000007f9dff8
```

[configuration information for Node 1, banks 1 through 6 (not shown)]

```
NODE001 BANK7 mem   0x0000000000000000
NODE001 BANK7 saddr 0x0000000000000000
NODE001 BANK7 eaddr 0x0000000000000000
NODE001 BANK7 size  0x0000000000000000
NODE0 CPU0 PID1 unmapped memory phase zero test
NODE0 CPU1 PID2 unmapped memory phase zero test
NODE0 CPU0 PID1 unmapped memory pattern test
NODE1 CPU2 PID3 unmapped memory phase zero test
NODE1 CPU3 PID4 unmapped memory phase zero test
NODE1 CPU2 PID3 unmapped memory pattern test
NODE0 CPU0 PID1 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
NODE0 CPU0 PID1 unmapped memory pattern test
NODE1 CPU2 PID3 memory pattern “0xc33cc33cc33cc33c 0x3cc33cc33cc33cc3”
test PASSED
NODE1 CPU2 PID3 unmapped memory pattern test
NODE0 CPU0 PID1 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE0 CPU0 PID1 unmapped memory SSO over cahe line test
NODE1 CPU2 PID3 memory pattern “0xffffffffffffffff 0x0000000000000000”
test PASSED
NODE1 CPU2 PID3 unmapped memory SSO over cahe line test
NODE0 CPU0 PID1 SSO over cache line test PASSED
NODE0 CPU0 PID1 unmapped memory phase zero test PASSED
NODE1 CPU2 PID3 SSO over cache line test PASSED
NODE1 CPU2 PID3 unmapped memory phase zero test PASSED
NODE0 CPU1 PID2 unmapped memory phase zero test PASSED
NODE1 CPU3 PID4 unmapped memory phase zero test PASSED
NODE0 CPU1 PID2 unmapped memory phase one test
NODE0 CPU0 PID1 unmapped memory phase one test
NODE0 CPU0 PID1 unmapped memory background test
NODE1 CPU2 PID3 unmapped memory phase one test
NODE1 CPU3 PID4 unmapped memory phase one test
NODE1 CPU2 PID3 unmapped memory background test
NODE0 CPU0 PID1 unmapped memory background test PASSED
NODE0 CPU0 PID1 unmapped memory cell test
NODE0 CPU0 PID1 unmapped memory cell test Ulong_NEC bank 0
NODE1 CPU2 PID3 unmapped memory background test PASSED
NODE1 CPU2 PID3 unmapped memory cell test
NODE1 CPU2 PID3 unmapped memory cell test Ulong_NEC bank 0
NODE0 CPU0 PID1 unmapped memory cell test Ulong bank 0
NODE1 CPU2 PID3 unmapped memory cell test Ulong bank 0
NODE0 CPU0 PID1 unmapped memory cell test PASSED
NODE0 CPU0 PID1 random data test
NODE0 CPU0 PID1 random data test loop 0
NODE1 CPU2 PID3 unmapped memory cell test PASSED
NODE1 CPU2 PID3 random data test
NODE1 CPU2 PID3 random data test loop 0
```

[additional loop count status output (not shown)]

```
NODE0 CPU0 PID1 random data test loop 40
NODE1 CPU2 PID3 random data test loop 40
NODE0 CPU0 PID1 random data test PASSED
NODE0 CPU0 PID1 unmapped memory phase one test PASSED
NODE1 CPU2 PID3 random data test PASSED
NODE1 CPU2 PID3 unmapped memory phase one test PASSED
```

```
NODE0 CPU1 PID2 unmapped memory phase one test PASSED
NODE1 CPU3 PID4 unmapped memory phase one test PASSED
NODE1 CPU2 PID3 unmapped memory phase two test
NODE0 CPU1 PID2 unmapped memory phase two test
NODE0 CPU0 PID1 unmapped memory phase two test
NODE0 CPU1 PID2 Unmapped memory random address and data test
NODE0 CPU1 PID2 unmapped memory random addr and data test Loop 0
NODE0 CPU0 PID1 Unmapped memory random address and data test
NODE0 CPU0 PID1 unmapped memory random addr and data test Loop 0
NODE1 CPU3 PID4 unmapped memory phase two test
NODE1 CPU2 PID3 Unmapped memory random address and data test
NODE1 CPU2 PID3 unmapped memory random addr and data test Loop 0
NODE1 CPU3 PID4 Unmapped memory random address and data test
NODE1 CPU3 PID4 unmapped memory random addr and data test Loop 0
NODE1 CPU3 PID4 unmapped memory random addr and data test Loop 5000
NODE0 CPU1 PID2 unmapped memory random addr and data test Loop 5000
NODE0 CPU0 PID1 unmapped memory random addr and data test Loop 5000
NODE1 CPU2 PID3 unmapped memory random addr and data test Loop 5000
```

[additional loop count status output (not shown)]

```
NODE1 CPU3 PID4 unmapped memory random addr and data test Loop 45000
NODE1 CPU3 PID4 unmapped memory random address and data test PASSED
NODE1 CPU2 PID3 unmapped memory random addr and data test Loop 45000
NODE1 CPU2 PID3 unmapped memory random address and data test PASSED
NODE0 CPU1 PID2 unmapped memory random addr and data test Loop 45000
NODE0 CPU0 PID1 unmapped memory random addr and data test Loop 45000
NODE0 CPU1 PID2 unmapped memory random address and data test PASSED
NODE0 CPU0 PID1 unmapped memory random address and data test PASSED
NODE0 CPU0 PID1 Unmapped memory inter node test
NODE0 CPU1 PID2 Unmapped memory inter node test
NODE0 CPU0 PID1 Loop 0
NODE0 CPU1 PID2 Loop 0
NODE1 CPU2 PID3 Unmapped memory inter node test
NODE1 CPU3 PID4 Unmapped memory inter node test
NODE1 CPU2 PID3 Loop 0
NODE1 CPU3 PID4 Loop 0
NODE1 CPU3 PID4 Loop 5000
NODE0 CPU0 PID1 Loop 5000
NODE0 CPU1 PID2 Loop 5000
NODE1 CPU2 PID3 Loop 5000
```

[more loop count status output (not shown)]

```
NODE1 CPU3 PID4 Loop 45000
NODE1 CPU3 PID4 Unmapped memory inter node test PASSED
NODE0 CPU1 PID2 Loop 45000
NODE0 CPU0 PID1 Loop 45000
NODE1 CPU2 PID3 Loop 45000
NODE0 CPU1 PID2 Unmapped memory inter node test PASSED
NODE0 CPU0 PID1 Unmapped memory inter node test PASSED
NODE1 CPU2 PID3 Unmapped memory inter node test PASSED
**** MDK memory unmapped test status summary ****
CNODE0 NASID0 CPU0 PID1 Mem test PASSED
CNODE0 NASID0 CPU1 PID2 Mem test PASSED
CNODE1 NASID1 CPU2 PID3 Mem test PASSED
CNODE1 NASID1 CPU3 PID4 Mem test PASSED
Memory unmapped test has completed
```

Chapter 5

Router SSO Test

This chapter describes the diagnostic that you can use to test Router and MetaRouter links.

5.1 About the Router SSO Test

The Router simultaneous switching (Router SSO) test is a directed test that focuses on testing the Router links, including the MetaRouter links if the system includes a MetaRouter. To run the Router SSO test, you must load the *MDK_router_sso* test package.

This test detects the following types of failures:

- Router failures: When this test detects a Router failure, the failing hardware is a CPOP connector, a midplane, a cable, a terminator, or a Router chip. The failing FRU is a Node board, Router board, midplane, or connecting cable.
- HUB failures: When this test detects a HUB failure, the failing FRU is a Node board, Router board, or midplane.
- MetaRouter failures: When this test detects a MetaRouter failure, the failing hardware is a CPOP connector, terminator, or a Router chip on a MetaRouter board; a CrayLink cable; or a CPOP connector, terminator, or Router chip on a Router board that is connected to a MetaRouter board. The failing FRU is a MetaRouter board, CrayLink cable, or Router board.

Note: Before you replace any FRUs, reseat the suspected boards and reconnect the suspected cables, and then run the test again. If the test still fails, you will need to replace one or more of the FRUs.

5.2 How to Run the Router SSO Test

Use the following procedure to load the test from the local system disk:

- Use the *boot* command from the PROM monitor (at the BaseIO command prompt, >>) to start the *MDK_router_sso* test package:

  >>boot dksc(0,1,0)/stand/MDK_router_sso

  MDK displays boot status infor1mation and the `MDK>` prompt.

- Load the Router SSO test, *router*, from the MDK> prompt:

  ```
  MDK>router
  ```

  The Router SSO test loads into the system and begins testing the Router links.

Tip: Always restart MDK with the **`boot dksc(0,1,0)/stand/MDK_router_sso`** command from the >> prompt before you run the Router SSO test. If you run this test after completing another MDK test without restarting MDK, the Router SSO test may report unexpected results.

5.3 Router SSO Test Description

The Router SSO test performs 238 passes of the following procedure:

- Select a data pattern from a predefined set of data patterns.
- Initialize the 16-Mbyte data area in memory on each Node board to the selected data pattern.

  Note: During this step, the test performs operations that ensure that the data contained in each secondary cache is written back to memory.

- Have one CPU on each Node board execute 50 iterations of the following procedure:

  1. Select a different Node board from which the CPU will obtain data.

     The CPU selects a different Node board for each iteration to ensure that data flows through all Router links in the system.

  2. Fetch data from memory on the selected Node board.
  3. The CPU performs multiple reads until the CPU reads the entire 16 Mbytes of data from the data area of memory on the selected Node board. Each read occurs at an address that is a cache line offset of the address used in the previous read: This ensures that the test performs as many back-to-back read operations as possible.
  4. Read the Router error register on each Router. If there are more than 10 check-bit errors, print out an error message. If there are 10 check-bit errors or less, write the check-bit error count into a data structure that the test will use to generate a report at the end of the test.
  5. Read the HUB network interface (NI) error register on each HUB. If there are more than 10 check-bit errors in a HUB NI, print out an error message. If there are 10 check-bit errors or less in a HUB NI, write the check-bit error count into a data structure that the test will use to generate a report at the end of the test.

- Have CPU0 gather the data from the data structure, summarize it, and print status information for the current loop.

Note: The total amount of data moving through the system during each pass equals 16 Mbytes multiplied by 50 (the number of iterations) multiplied by the number of Node boards in the system.

5.4 Router SSO Test Output

The Router SSO test returns output to the BaseIO console. All output comes from CPU0, which summarizes the status data from all other CPUs in the system.

5.4.1 Pass Output

When the test completes all 238 passes, it prints a `TEST RESULT` message.

For a test that completes all 238 passes without detecting any failures, the `TEST RESULT` message is a `TEST PASSED` message:

```
TEST RESULT: **** TEST PASSED  ****
MDK Router_sso test run is complete.
```

5.4.2 Failure Output for Router Failures

When an individual pass detects that more than 10 check-bit errors have occurred on a Router, it prints an error message and a table that indicates which port had the errors. If a pass detects that fewer than 10 check-bit errors occurred, it updates the data in the table but does not print out the table.

The following output shows an example error message and table when there are more than 10 check-bit errors on a Router:

```
ERROR: CPU0: pid 1: Checkbit error count threshold of 50 exceeded on port 5
       Or Checkbit error count threshold of 10 exceeded on all other ports.

Checkbit_err count on ROUTER:
 ----------------------------

 Nasid  Module  Node  Router      port:
                slot   slot        1     2     3     4     5     6
 -----  ------  ----  ------      ----  ----  ----  ----  ----  ----
    0       1     1       1         0     0     0     0     0     0
    3       1     4       2         0     0     0     0     0     0
    2       1     3       2         0     0     0     0     0     0
    7       2     4       2         0     0     0     0     0     0
    6       2     3       2       190     0     0     0     0     0
    5       2     2       1         0     0     0     0     0     0
    4       2     1       1         0     0     0     0     0     0
    1       1     2       1         0     0     0     0     0     0
```

This example shows that 190 check-bit errors occurred on Router port 1 of the Router board in slot 2 of module 2.

When the test completes all 96 passes, it prints a test result message, which is either a warning message or a fail message. When one of these messages appears, check the `Checkbit_err count on router` portion of the output that follows the message to determine the ports on which the errors occurred.

5.4.2.1 Warning Message for Router Failures

A warning message indicates that the test has detected check-bit errors on one or more Router ports, but each port had fewer than the threshold of 10 check-bit errors per pass. The following example output includes a warning message because there are two ports that each have one check-bit error:

```
TEST RESULT: **** WARNING: CHECKBIT_error_count > 0,              ****
TEST RESULT: ****          BUT LESS THAN THRESHOLD OF 10 PER PASS ****

Checkbit_err count on router:
-----------------------------

Nasid  Module  Node  Router      port:
               slot   slot         1     2     3     4     5     6
-----  ------  ----  ------      ----  ----  ----  ----  ----  ----
   0      1      1      1          0     0     0     0     0     0
   3      1      4      2          0     0     0     0     1     0
   2      1      3      2          0     0     0     0     0     0
   7      2      4      2          0     0     0     0     0     0
   6      2      3      2          0     0     0     0     0     0
   5      2      2      1          0     0     0     0     0     0
   4      2      1      1          0     0     0     0     1     0
   1      1      2      1          0     0     0     0     0     0
```

If the test returns a `WARNING` message for a Router, the failing hardware is hard to isolate. The failing hardware might be the Router chip for the port that has one to nine errors. The failing FRU could be the Router board with the failing Router chip. The failing FRU could also be a CrayLink cable or midplane.

Note: Before you replace any FRUs, reseat the suspected boards and reconnect the suspected cables, and then run the test again. If the test still fails, you will need to replace one or more of the FRUs.

Refer to Appendix A, “Troubleshooting Link Failures,” for general information about troubleshooting link failures.

5.4.2.2 Fail Message for Router Failures

A fail message indicates that the test has detected more than 10 check-bit errors on one or more Routers on an individual pass. The following example output includes a fail message because Router port 1 of the Router board in slot 2 of module 2 has 190 check-bit errors:

```
TEST RESULT: **** FAIL: NUMBER OF CHECKBIT ERRORS EXCEEDED THRESHOLD ****
TEST RESULT: ****       OF 10 ON AT LEAST ONE PASS OF THE TEST ****

Checkbit_err count on router:
-----------------------------

Nasid  Module  Node  Router      port:
               slot   slot         1     2     3     4     5     6
-----  ------  ----  ------      ----  ----  ----  ----  ----  ----
   0       1     1       1         0     0     0     0     0     0
   3       1     4       2         0     0     0     0     0     0
   2       1     3       2         0     0     0     0     0     0
   7       2     4       2         0     0     0     0     0     0
   6       2     3       2       190     0     0     0     0     0
   5       2     2       1         0     0     0     0     0     0
   4       2     1       1         0     0     0     0     0     0
   1       1     2       1         0     0     0     0     0     0
```

If the test returns a fail message for a Router, check the `Checkbit_err count on router` portion of the output to determine more information about the failure:

- If the errors are on ports 1, 2, or 3; check the cables that connect the Routers on those ports.
- If the errors are on ports 4, 5, or 6; the failing hardware components are the CPOP connectors on the Router board or Node board. The failing FRU is the Router board or Node board.
- If the check-bit error count is a value that ranges from the hundreds to the thousands, the failing hardware is most likely a CPOP connector on the Router board or Node board, a midplane, or a CrayLink cable. The failing FRU is the Router board or Node board.
- If the check-bit error count is a value that ranges from the tens to hundreds, the failing hardware is most likely a terminator on the Router board. The failing FRU is the Router board.
- If the check-bit error count is less than 10, the failing hardware is hard to isolate, but it could be a Router chip. The failing FRU could be the Router board with the failing Router chip.

Note: Before you replace any FRUs, reseat the suspected boards and reconnect the suspected cables, and then run the test again. If the test still fails, you will need to replace one or more of the FRUs.

Refer to Appendix A, “Troubleshooting Link Failures,” for general information about troubleshooting link failures.

5.4.3 Failure Output for HUB Failures

When an individual pass detects that more than 10 check-bit errors have occurred on a HUB, it prints an error message and a table that indicates which HUB had the errors. If a pass detects that fewer than 10 check-bit errors occurred, it updates the data in the table but does not print out the table.

The following output shows an example error message and table when there are more than 10 check-bit errors on a HUB:

```
ERROR: CPU0: pid 1: Checkbit error count threshold of 50 exceeded on port 5
       Or Checkbit error count threshold of 10 exceeded on all other ports.

 Checkbit_err count on HUB Network Interface:
 -------------------------------------------

 Nasid  Module  Node  Router    Checkbit_err
                slot   slot         count
 -----  ------  ----  ------    ------------
    0       1     1      1             0
    3       1     4      2             0
    2       1     3      2             0
    7       2     4      2             0
    6       2     3      2            11
    5       2     2      1             0
    4       2     1      1             0
    1       1     2      1             0
```

This example shows that there were 11 check-bit errors detected on the HUB that is located on the Node board in slot 3 of module 2.

When the test completes all 238 passes, it prints a test result message, which is either a warning message or a fail message. When one of these messages appears, check the `Checkbit_err count on HUB Network Interface` portion of the output that follows the message to determine the Hubs on which the errors occurred.

5.4.3.1 Warning Message for HUB Failures

A warning message indicates that the test has detected check-bit errors on one or more HUB NI registers, but each HUB NI register had fewer than the threshold of 10 check-bit errors per pass. The following example output includes a warning message because there is one HUB NI register with one check-bit error:

```
TEST RESULT: **** WARNING: CHECKBIT_error_count > 0,              ****
TEST RESULT: ****          BUT LESS THAN THRESHOLD OF 10 PER PASS ****

 Checkbit_err count on HUB Network Interface:
 -------------------------------------------

 Nasid  Module  Node  Router    Checkbit_err
                slot   slot         count
 -----  ------  ----  ------    ------------
    0       1     1      1            0
    3       1     4      2            0
    2       1     3      2            0
    7       2     4      2            0
    6       2     3      2            1
    5       2     2      1            0
    4       2     1      1            0
    1       1     2      1            0
```

If the test returns a `WARNING` message for a HUB, the failing hardware could be a Node board, Router board, or midplane.

5.4.3.2 Fail Message for HUB Failures

A fail message indicates that the test has detected more than 10 check-bit errors in one or more HUB NI registers on an individual pass. The following example output includes a fail message because Router port 1 of the Router board in slot 2 of module 2 has 190 check-bit errors:

```
TEST RESULT: **** FAIL: NUMBER OF CHECKBIT ERRORS EXCEEDED THRESHOLD ****
TEST RESULT: ****       OF 10 ON AT LEAST ONE PASS OF THE TEST ****

 Checkbit_err count on HUB Network Interface:
 -------------------------------------------

 Nasid  Module  Node  Router    Checkbit_err
                slot   slot         count
 -----  ------  ----  ------    ------------
    0       1     1      1            0
    3       1     4      2            0
    2       1     3      2            0
    7       2     4      2            0
    6       2     3      2           11
    5       2     2      1            0
    4       2     1      1            0
    1       1     2      1            0
```

If the test returns a fail message for a HUB, check the `Checkbit_err count on HUB Network Interface` portion of the output to determine more information about the failure. The failing hardware could be a Node board, Router board, or midplane.

In the example output shown above, the check-bit errors occurred on the Node board in slot 3 of module 2. The failing hardware is either the Node board in slot 3 of module 2, the Router board in slot 2 of module 2, or the midplane.

5.4.4 Failure Output for MetaRouter Failures

When an individual pass detects that more than 10 check-bit errors have occurred on a MetaRouter, it prints an error message and a table that indicates which MetaRouter had the errors. If a pass detects that fewer than 10 check-bit errors occurred, it updates the data in the table but does not print out the table.

The following output shows an example error message and table when there are more than 10 check-bit errors on a MetaRouter:

```
ERROR: CPU0: pid 1: Checkbit error count threshold of 50 exceeded on port 5
       Or Checkbit error count threshold of 10 exceeded on all other ports.

Checkbit_err count on META_ROUTERS:
-----------------------------------

     Meta Router                  port:
Vector_route  Module  Slot         1     2     3     4     5     6
------------  ------  ----       ----  ----  ----  ----  ----  ----

         0                         0     0     0     0     0     0
         2      99      2          0     0     0     0     0     0
         6                         0     0     0     0     0     0
        22      99      4          0     0     0     0     0     0
        32      99      6          0     0     0     0     0     0
       122                        0     0     0    24     0     0
       232      99      8          0     0     0     0     0     0
       132                        0     0     0     0     0     0
      6122                        0     0     0     0    57     0
      1232                        0     0     0     0     0     0
      6132                        0     0     0     0     0     0
     61232                        0     0     0     0     0     0
```

Note: Currently this test cannot print the module and slot locations for the MetaRouter boards that it tests; instead, it prints out the vector router path from the CPU to each MetaRouter board.

This example shows that there are 24 check-bit errors on the MetaRouter board that uses vector route path 122 from the master CPU. It also shows that there are 57 check-bit errors on the MetaRouter board that uses vector route path 6122 from the master CPU.

When the test completes all 238 passes, it prints a test result message, which is either a warning message or a fail message. When one of these messages appears, check the `Checkbit_err count on META_ROUTERS` portion of the output that follows the message to determine the MetaRouters on which the errors occurred.

5.4.4.1 Warning Message for MetaRouter Failures

A warning message indicates that the test has detected check-bit errors on one or more MetaRouter ports, but each port had fewer than the threshold of 10 check-bit errors per pass. The following example output includes a warning message because there are two ports that each have one check-bit error:

```
TEST RESULT: **** WARNING: CHECKBIT_error_count > 0                  ****
TEST RESULT: ****          BUT LESS THAN THRESHOLD OF 10 PER PASS    ****

Checkbit_err count on META_ROUTERS:
-----------------------------------

     Meta Router                 port:
Vector_route  Module  Slot        1     2     3     4     5     6
------------  ------  ----      ----  ----  ----  ----  ----  ----

         0                        0     0     0     0     0     0
         2      99       2        0     0     0     0     0     0
         6                        0     0     0     0     0     0
        22      99       4        0     0     0     0     0     0
        32      99       6        0     0     0     0     0     0
       122                       0     0     0     3     0     0
       232      99       8        0     0     0     0     0     0
       132                       0     0     0     0     0     0
      6122                       0     0     0     0     4     0
      1232                       0     0     0     0     0     0
      6132                       0     0     0     0     0     0
     61232                       0     0     0     0     0     0
```

If the test returns a `WARNING` message for a MetaRouter, the failing hardware is hard to isolate. The failing hardware might be the Router chip for the port that has one to nine errors. The failing FRU could be the MetaRouter board with the failing Router chip. The failing FRU could also be a CrayLink cable or midplane.

Note: Before you replace any FRUs, reseat the suspected boards and reconnect the suspected cables, and then run the test again. If the test still fails, you will need to replace one or more of the FRUs.

Refer to Appendix A, “Troubleshooting Link Failures,” for general information about troubleshooting link failures.

5.4.4.2 Fail Message for MetaRouter Failures

A fail message indicates that the test has detected more than 10 check-bit errors on one or more MetaRouters on an individual pass. The following example output includes a fail message because two MetaRouter boards have more than 10 check-bit errors:

```
TEST RESULT: **** FAIL: NUMBER OF CHECKBIT ERRORS EXCEEDED THRESHOLD ****
TEST RESULT: ****       OF 10 ON AT LEAST ONE PASS OF THE TEST       ****

Checkbit_err count on META_ROUTERS:
-----------------------------------
     Meta Router                 port:
Vector_route  Module  Slot         1     2     3     4     5     6
------------  ------  ----       ----  ----  ----  ----  ----  ----

         0                         0     0     0     0     0     0
         2      99      2          0     0     0     0     0     0
         6                         0     0     0     0     0     0
        22      99      4          0     0     0     0     0     0
        32      99      6          0     0     0     0     0     0
       122                        0     0     0    24     0     0
       232      99      8          0     0     0     0     0     0
       132                        0     0     0     0     0     0
      6122                        0     0     0     0    57     0
      1232                        0     0     0     0     0     0
      6132                        0     0     0     0     0     0
     61232                        0     0     0     0     0     0
```

If the test returns a fail message for a MetaRouter, check the `Checkbit_err count on META_ROUTERS` portion of the output to determine more information about the failure:

- If the errors are on ports 1, 2, or 3; check the cables that connect the MetaRouters on those ports.
- If the errors are on ports 4, 5, or 6; the failing hardware components are the CPOP connectors on the MetaRouter board or Router board. The failing FRU is the MetaRouter board or the Router board.
- If the check-bit error count is a value that ranges from the hundreds to the thousands, the failing hardware is most likely a CPOP connector on the Router board, a CPOP connector on the MetaRouter board, or a CrayLink cable. The failing FRU is the Router board or the MetaRouter board.
- If the check-bit error count is a value that ranges from the tens to hundreds, the failing hardware is most likely a terminator on the Router board. The failing FRU is the Router board.
- If the check-bit error count is less than 10, the failing hardware is hard to isolate, but it could be a Router chip. The failing FRU could be the Router board with the failing Router chip.

Note: Before you replace any FRUs, reseat the suspected boards and reconnect the suspected cables, and then run the test again. If the test still fails, you will need to replace one or more of the FRUs.

Refer to Appendix A, “Troubleshooting Link Failures,” for general information about troubleshooting link failures.

5.4.5 Failure Output When the Test Takes an Unexpected Exception

If an unexpected exception occurs, the MDK exception handler code takes control of the system. MDK prints `ERROR: Unexpected Exception Occurred`, followed by a register dump. The register dump contains Coprocessor 0 registers, the general-purpose registers, and the following information about the exception: the type of exception, the address where the exception occurred, and any other information that is relevant to the exception.

The following example shows the output from MDK when this test takes an unexpected exception:

```
ERROR: Unexpected Exception Occured.
Status: 0xffffffffb40084e3        XCtxt: 0xffffffe0000078b0
  Hi:     0x               0        Lo:    0x               0
  epc:    0x          e09f48        cause: 0x            401c
  count:  0x          135857        comp:  0x         c800000
  Enhi:   0x               0        CacErr: 0xffffffffdfffffff
  R01/AT: 0x          f0f000        R02/V0: 0x          f0f000
  R03/V1: 0x              7f        R04/A0: 0xffffffffffffffff
  R05/A1: 0x              7f        R06/A2: 0x9600007f00000000
  R07/A3: 0x9600007f00002048        R08/A4: 0x9600000000000000
  R09/A5: 0x               0        R10/A6: 0x               1
  R11/A7: 0x              40        R12/T0: 0x              3f
  R13/T1: 0x               8        R14/T2: 0x               4
  R15/T3: 0x              1f        R16/S0: 0x               0
  R17/S1: 0x               0        R18/S2: 0x               0
  R19/S3: 0x               0        R20/S4: 0x               0
  R21/S5: 0x               0        R22/S6: 0x               0
  R23/S7: 0x               0        R24/T8: 0x               1
  R25/T9: 0x              1f        R28/GP: 0x               0
  R29/SP: 0x        3ffffbf0        R30/S8: 0x               0
  R31/RA: 0x          f10de5
  Nasid 0: Local CPU A: Global CPU 0: PID 1: Bus error at instr 0xe09f48:
       sr = 0xb40084e3
  Nanos: Frozen 1
```

This example is a bus exception error, which indicates that a Router or MetaRouter link went down with a hard failure (there is no longer a connection).

Refer to Appendix A, “Troubleshooting Link Failures,” for general information about troubleshooting link failures.

5.4.6 Special Output When the System Includes a MetaRouter

When the system includes a MetaRouter, the Router SSO test prints out the following additional information at the beginning of the test:

- a list of vector routes from the master NASID to each Router and MetaRouter
- the version number for the test
- the system configuration information
- a table of the mappings of modules, NASIDs, Node slots, and Router slots

The following example output is from a system with a MetaRouter:

```
router vector[0] = 0
        router vector[1] = 1
   meta_router vector[0] = 2   module = 0   slot = 2
        router vector[2] = 3
        router vector[3] = 6
   meta_router vector[1] = 21   module = 0   slot = 2
        router vector[4] = 13
        router vector[5] = 16
        router vector[6] = 22
        router vector[7] = 32
        router vector[8] = 62
   meta_router vector[2] = 23   module = 0   slot = 6
        router vector[9] = 36
   meta_router vector[3] = 26   module = 0   slot = 4
        router vector[10] = 122
        router vector[11] = 132
        router vector[12] = 162
   meta_router vector[4] = 213   module = 0   slot = 6
        router vector[13] = 136
   meta_router vector[5] = 216   module = 0   slot = 4
        router vector[14] = 223
        router vector[15] = 226
        router vector[16] = 323
        router vector[17] = 326
        router vector[18] = 623
        router vector[19] = 626
   meta_router vector[6] = 236   module = 0   slot = 8
        router vector[20] = 1223
        router vector[21] = 1226
        router vector[22] = 1323
        router vector[23] = 1326
        router vector[24] = 1623
        router vector[25] = 1626
   meta_router vector[7] = 2136   module = 0   slot = 8
        router vector[26] = 2236
        router vector[27] = 3236
        router vector[28] = 6236
        router vector[29] = 12236
        router vector[30] = 13236
        router vector[31] = 16236
```

```
Starting Router SSO test (version 2.3.2).

  Checkbit Error Threshold Count                        =  10 per pass.
  Checkbit Error Threshold Port 5 Count                 =  50 per pass.
  Checkbit Error Threshold Count for star router port 3 =  10 per pass.
  Debug Switch value                                    =  0 0
  Start Pass                                            =  0
  End Pass                                              =  238
  Number of Iteration for random mode                   =  100
  Number of Iteration for single link mode              =  2
  Number of Iteration for single node source mode       =  5
  Number of initialized data patterns                   =  238
  Max Retry Value                                       =
3ff000000000000
  Mem_Areas pointer                                     =  f09000
  Mem_Areas physical pointer                            =
a800000001487000

System Configuration:

   Number of nodes:          64
   Number of cpus:           125
   Number of routers:        40


Table of module, nasid, and slot numbers:

     Module      Nasid      Node      Router
                            Slot       slot
     ------      -----      ----      ------
        1          0          1          1
        1          1          2          1
        2          5          2          1
        2          4          1          1
        3          9          2          1
        1          3          4          2
        1          2          3          2
        4         13          2          1
        4         12          1          1
        2          7          4          2
        2          6          3          2
        9         49          2          1
        9         48          1          1
       13         33          2          1
       13         32          1          1
        5         17          2          1
        5         16          1          1
        3         11          4          2
        3         10          3          2
       10         53          2          1
       10         52          1          1
       14         37          2          1
       14         36          1          1
        6         21          2          1
        6         20          1          1
        4         15          4          2
        4         14          3          2
```

```
11          57          2           1
11          56          1           1
 9          51          4           2
 9          50          3           2
15          41          2           1
15          40          1           1
13          35          4           2
13          34          3           2
 7          25          2           1
 7          24          1           1
 5          19          4           2
 5          18          3           2
12          61          2           1
12          60          1           1
10          55          4           2
10          54          3           2
16          45          2           1
16          44          1           1
14          39          4           2
14          38          3           2
 8          29          2           1
 8          28          1           1
 6          23          4           2
 6          22          3           2
11          59          4           2
11          58          3           2
15          43          4           2
15          42          3           2
 7          27          4           2
 7          26          3           2
12          63          4           2
12          62          3           2
16          47          4           2
16          46          3           2
 8          31          4           2
 8          30          3           2
```

The normal pass information follows this output, starting with pass 0.

Note: Due to a PROM function bug, the module number for a MetaRouter that is indicated in the Router vector route list and MetaRouter vector route list is incorrect. The actual MetaRouter module number should be 98 (0x62) or 99 (0x63).

5.5 Example of Running the Router SSO Test

The following example shows passes 0, 236, and 237 from running the test on a system with 16 CPUs (2 modules with a total of 8 Nodes, 16 CPUs, and 4 Routers). In this example, pass 0 does not detect any errors, pass 236 detects more than 10 check-bit errors, and pass 237 (the final pass) does not detect any errors.

In this example, the test reports 190 check-bit errors on Router port 1 of the Router board in slot 2 of module 2. The Router board is most likely causing this failure because the check-bit errors are on port 1, which is a Router board connection.

Pass 236 of the test also reports 11 check-bit errors in the HUB NI register on the Node board in slot 3 of module 2. This failure is caused by the Node board in slot 3 of module 2, the Router board in slot 2 of module 2, or the midplane.

```
>>boot dksc(0,1,0)/stand/MDK_router_sso
Booting Nanos.........

************************************************************
SGI MDK Version 1.25 SN0 built 04:20:02 PM Jul 15, 1998
************************************************************

CPU 0: Total no. of CPUs = 16
Launched CPU 1
Launched CPU 2
Launched CPU 3
Launched CPU 4
Launched CPU 5
Launched CPU 6
Launched CPU 7
Launched CPU 8                                    Initial parameters
Launched CPU 9
Launched CPU 10
Launched CPU 11
Launched CPU 12
Launched CPU 13
Launched CPU 14
Launched CPU 15
MDK>router
Created successfully pid = 1
CPU 0: User mem starts from 0xa800000000732000

Starting Router SSO test.

Checkbit Error Threshold Count = 10 per pass.

System Configuration:

   Number of nodes:      8
   Number of cpus:      16
   Number of routers:    4
```

Figure 5-1 Router SSO Test Sample Output (Part 1 of 4)

```
Table of module, nasid, and slot numbers:
     Module      Nasid      Node      Router
                            Slot       slot
     ------      -----      ----      ------
        1          0          1          1
        1          3          4          2
        1          2          3          2
        2          7          4          2
        2          6          3          2
        2          5          2          1
        2          4          1          1
        1          1          2          1

Number of initialized data patterns = 96


CPU0: pid 1: PASS = 0 Started
CPU0: pid 1: Initializing memory on each node
             to data pattern = 5aaaaaaaaaaaaaaa 5aaaaaaaaaaaaaaa
             to test router link data line = 0
CPU0: pid 1: Starting test code on all nodes.
CPU0: pid 1: PASS = 0 Completed




CPU0: pid 1: PASS = 236 Started
CPU0: pid 1: Initializing memory on each node
             to data pattern = 3333333333333373 ccccccccccccc8c
             to test router link data line = 14
CPU0: pid 1: Starting test code on all nodes.

ERROR: CPU0: pid 1: Checkbit error count threshold of 50 exceeded on port 5
       Or Checkbit error count threshold of 10 exceeded on all other ports.

Checkbit_err count on router:
-----------------------------


Nasid  Module  Node  Router      port:
               slot   slot         1     2     3     4     5     6
-----  ------  ----  ------      ----  ----  ----  ----  ----  ----
   0      1      1      1          0     0     0     0     0     0
   3      1      4      2          0     0     0     0     0     0
   2      1      3      2          0     0     0     0     0     0
   7      2      4      2          0     0     0     0     0     0
   6      2      3      2        190     0     0     0     0     0
   5      2      2      1          0     0     0     0     0     0
   4      2      1      1          0     0     0     0     0     0
   1      1      2      1          0     0     0     0     0     0
```

Initial parameters

Pass 0 information

Passes 2 through 235 are not shown in this example.

Pass 236 information

Failure for pass 236:

Router port 1 on the Router board in slot 2 of module 2 has 190 check-bit errors

Figure 5-2 Router SSO Test Sample Output (Part 2 of 4)

```
ERROR: CPU0: pid 1: Checkbit error count threshold of 10 exceeded.

 Checkbit_err count on HUB Network Interface:
 ----------------------------------------

 Nasid  Module  Node  Router    Checkbit_err
                slot   slot        count
 -----  ------  ----  ------    ------------
    0      1      1      1            0
    3      1      4      2            0
    2      1      3      2            0
    7      2      4      2            0
    6      2      3      2           11
    5      2      2      1            0
    4      2      1      1            0
    1      1      2      1            0

CPU0: pid 1: PASS = 236 Completed

CPU0: pid 1: PASS = 237 Started
CPU0: pid 1: Initializing memory on each node
             to data pattern = 3333333333333337 ccccccccccccccc8
             to test router link data line = 15
CPU0: pid 1: Starting test code on all nodes.

CPU0: pid 1: PASS = 237 Completed
TEST RESULT: **** FAIL: NUMBER OF CHECKBIT ERRORS EXCEEDED THRESHOLD ****
TEST RESULT: ****       OF 10 ON AT LEAST ONE PASS OF THE TEST ****

Checkbit_err count on router:
----------------------------

Nasid  Module  Node  Router      port:
               slot   slot        1     2     3     4     5     6
-----  ------  ----  ------      ----  ----  ----  ----  ----  ----
   0      1      1      1          0     0     0     0     0     0
   3      1      4      2          0     0     0     0     0     0
   2      1      3      2          0     0     0     0     0     0
   7      2      4      2          0     0     0     0     0     0
   6      2      3      2        190     0     0     0     0     0
   5      2      2      1          0     0     0     0     0     0
   4      2      1      1          0     0     0     0     0     0
   1      1      2      1          0     0     0     0     0     0
```

Failure for pass 236:

HUB NI register on Node board in slot 3 of module 2 has 11 check-bit errors

Pass 237 information

Test result message and check-bit error counts for Router failure

Figure 5-3 Router SSO Test Sample Output (Part 3 of 4)

```
TEST RESULT: **** FAIL: NUMBER OF CHECKBIT ERRORS EXCEEDED THRESHOLD ****
TEST RESULT: ****       OF 10 ON AT LEAST ONE PASS OF THE TEST       ****

Checkbit_err count on HUB Network Interface:
-------------------------------------------

Nasid  Module  Node  Router    Checkbit_err
               slot   slot         count
-----  ------  ----  ------    ------------
   0       1     1       1             0
   3       1     4       2             0
   2       1     3       2             0
   7       2     4       2             0
   6       2     3       2            11
   5       2     2       1             0
   4       2     1       1             0
   1       1     2       1             0

MDK Router_sso test run is complete.
```

Test result message and check-bit error counts for HUB failure

Failures for this test:

Router port 1 on the Router board in slot 2 of module 2 has 190 check-bit errors, and HUB NI register on the Node board in slot 3 of module 2 has 11 check-bit errors

Figure 5-4 Router SSO Test Sample Output (Part 4 of 4)

Appendix A

Troubleshooting Link Failures

This appendix provides some general information for troubleshooting link failures. This information may help you fix failures that are detected by the Router SSO test.

A.1 About the Green LEDs on the Router Boards

A properly working link illuminates the green LED on the Router board. Failures are indicated by the following conditions:

- The green LED will not illuminate if nothing is connected to the link.
- The green LED will not illuminate if the link is broken.
- A poor link may negotiate successfully and illuminate the green LED, but the link may fail when it is being stress tested: This causes the link to shut down and turns off the green LED for the link.
- A marginal link may never fail (shut down), but it will report excessive errors in the Node NI error register, the Router error register, or the XBOW error register. The green LED will remain illuminated, but the diagnostics will read the error registers and report an excessive number of errors.

A.2 Troubleshooting Internal (CPOP) Link Failures

Internal link failures are most often caused by poor connections through one or more CPOP connectors. For Router link failures, the failing CPOP connectors will be on the Node boards or Router boards. For I/O link (XBOW) failures, the failing CPOP connectors will be on the Node boards.

If you suspect that an internal link problem is causing a link failure, perform the following procedure:

- If the green LED on a Router board is not illuminated, perform the following actions to check the CPOP connectors on the Router board:

  1. Remove the Router board and inspect the CPOP connector for damage. Also inspect the midplane connector area for foreign material.
  2. Reinstall the Router board.

  Caution: When you reinstall the Router board, screw each of the hex rods in halfway before you screw both rods in completely. Screwing in one rod completely before starting to screw in the other rod can damage the CPOP connector, which will cause more link failures.

- If the green LED on the Router board still does not illuminate or the corresponding XBOW green LED is not illuminated, perform the following actions to check the CPOP connectors on the Node board used in the link:

  1. Remove the corresponding Node board and inspect the CPOP connector for damage. Also inspect the midplane connector area for foreign material.
  2. Reinstall the Node board.

  Caution: When you reinstall the Node board, screw each of the hex rods in halfway before you screw both rods in completely. Screwing in one rod completely before starting to screw in the other rod can damage the CPOP connector, which will cause more link failures.

- If the link problem still exists, swap the Node board into a different slot.

  If the link error follows the Node board, the failing FRU is most likely the Node board.

- If the link error does not follow the Node board, swap the Router board into a different slot.

  If the link error follows the Router board, the failing FRU is most likely the Router board.

  If the link error does not follow the Router board, the midplane is most likely the failing FRU.

A.3 Troubleshooting External (Cable) Failures

External link failures apply only to Router link failures. Poor mating between a cable and the Router is the most likely cause of external link failures.

If the Router link green LED is not illuminated and you suspect that an external failure is the problem, perform the following procedure:

- Inspect the failing cable connections for proper mating.
- Remove and inspect the cables for damage.
- Re-attach the cables.

Caution: When you reconnect the cable, support the connector with one hand and carefully tighten the top and bottom screws. Ensure that there is no play in the mated cable connection.

- If the problem persists, replace the cable.
- If the problem persists with a different cable, try connecting one end of the cable to an alternate Router port. This will narrow the failure to a single port of a single board.