sMDK-based Field Diagnostics (SGI™ 3000 Family)

SGI Confidential & Proprietary Information - For Internal Recipients Only

Document Number 108-0264-003

CONTRIBUTORS
Written by Darrin Goss
Edited by Allison Gosbin
Production by Rhonda Kunsman
Engineering contributions by Nelson Crumbaker, Joe Ivanauskas, and Lisa Steinmetz

Record of Revision

Version	Description
001	July 2000 Original release (online version only).
002	August 2000 This revision incorporates technical changes throughout the document.
003	January 2001 This revision adds descriptions of the real-time clock test (`rtc`) and the `elog_decode` utility.

Contents

Tables

Chapter 1

Introduction

This chapter introduces the scalable Micro-diagnostic Kernel (sMDK) and describes how to start, quit, and control it.

1.1 What is sMDK?

The scalable Micro-diagnostic Kernel (sMDK) is a standalone diagnostic environment that runs in kernel mode. sMDK provides test access to hardware components that cannot be accessed from a user program that is running under the operating system.

sMDK-based diagnostics require full use of the system: You must bring down (reboot) the system to run sMDK. No operating system can be running in the system when you use the sMDK-based diagnostics.

sMDK differs from the previous MDK package as follows:

- sMDK is distributed throughout an entire system
- sMDK can run multiple diagnostics at the same time
- sMDK supports an ASCII user interface that implements multiple viewable pages for each diagnostic test
- sMDK-based diagnostics display percent-completed status

Use the sMDK-based diagnostics to isolate failures for which the online diagnostics do not provide enough error information or to test hardware that the online diagnostics cannot test.

1.2 Installing sMDK

The *Internal Support Tools 2.3* CD, part number 812-0640-007, contains all of the files that you need to install sMDK and its diagnostic tests and utilities. Refer to the *Silicon Graphics Internal Support Tool 2.3 CD Installation Instructions*, part number 007-3516-007, for detailed information about installing the sMDK-related files on an SGI Origin 3000 series server.

1.3 Starting sMDK

Perform the following procedure to start sMDK:

1. Power up or reset the system.
2. Enter **`5`** to choose the `5. Enter Command Monitor` option.
3. Enter **`boot -f dksc(`*`x`*`,`*`y`*`,`*`z`*`)/stand/smdk/smdk`** at the command monitor prompt (>>) to start sMDK. (Replace *x*, *y*, and *z* with the controller, disk, and slice number of the disk that contains sMDK.)

   Note: Use the `hinv` command at the command monitor prompt (>>) to determine the controller, disk, and slice number of the disk that contains sMDK.

1.4 Quitting sMDK

To quit sMDK, enter the `reset` command at the `smdk>` prompt to reset the system:

```
smdk> reset
```

1.5 Controlling sMDK

You control sMDK by entering commands at the `smdk>` prompt. sMDK commands are grouped into the following categories:

- Configuration commands
- Control commands
- System commands
- Process commands

1.5.1 Configuration Commands

1.5.1.1 cfg

`cfg`

Displays the hardware configuration: number of nodes, number of CPUs, NASID-to-vnode mapping, and vcpu-to-vnode mapping.

`cfg n <`*vnode*`>`

Displays the configuration of the specified node.

`cfg r <`*vrouter*`>`

Displays the configuration of the specified router.

1.5.1.2 lpath

`lpath <`*load_directory_path*`>`

Specifies the location where the sMDK files are located. If you do not modify this setting, sMDK uses files from the `/stand/smdk` directory on the local system.

1.5.1.3 term

`term`

Displays the current terminal setting.

`term <`*terminal_type*`>`

Sets the terminal type. (Supported terminals: iris-ansi)

1.5.1.4 version

`version`

Displays the version of sMDK that you are using and the date and time that it was built.

1.5.2 Control Commands

1.5.2.1 load

`load <`*filename*`> <`*options*`>`

Loads a test into memory and selects it as the current test. Use the load command to load a diagnostic test so you can view the memory that the test uses or set breakpoints before you run the test.

1.5.2.2 ds

`ds`

Displays the following diagnostic status information for all diagnostics that are running: index number, name, status, pass count, life count, and fail count.

The status information updates every 5 seconds until you press the <Return> key.

1.5.2.3 drop

`drop <diag_index>`

Kills all processes that are spawned by the diagnostic specified by <*diag_index*>.

1.5.2.4 dscr

`dscr <diag_index>`

Lists the displays that are available for the diagnostic test specified by <*diag_index*>.

`dscr <diag_index> <screen> <page> <optarg0> <optarg1>`

Displays the diagnostic information page that you specify.

The <*diag_index*> argument specifies the diagnostic, the <*name*> argument specifies the name of the screen, and the <*page*> argument specifies the page to view.

The <*optarg0*> and <*optarg1*> arguments vary by diagnostic. (Refer to the individual diagnostic help pages for more information about these arguments.)

1.5.2.5 scrinfo

`scrinfo <diag_index>`

Displays information about each screen page (where it resides in memory, size, etc.) for the diagnostic that you specify with <*diag_index*>. This is a debugging feature.

1.5.2.6 reset

`reset`

Resets the system.

1.5.3 System Commands

1.5.3.1 ping

`ping`

Requests a response from all CPUs and reports the number of CPUs that respond to the request.

`ping <vcpu>`

Requests a response from the virtual CPU that you specify.

1.5.3.2 kl

`kl`

Displays a list of virtual CPUs whose kernel logs contain entries.

`kl <vcpu>`

Displays the kernel log for the virtual CPU that you specify.

1.5.3.3 klclr

`klclr`

Clears all kernel logs.

1.5.3.4 el

`el`

Lists all nodes that logged hardware-detected errors and the number of each type of error.

`el <vnode>`

Displays detailed information about the hardware-detected errors that were logged by the specified node.

1.5.3.5 elclr

`elclr`

Clears the error logs on all nodes.

1.5.3.6 ps

`ps`

Displays the status of all processes.

`ps <vcpu>`

Displays process status for the virtual CPU that you specify.

1.5.3.7 dp

`dp <paddr> [count]`

Displays [*count*] words of physical memory starting at <*paddr*> address.

If you do not specify a [*count*] value, this command displays 1 word. If you do not specify cache ALG bits, the command combines the specified address with 0xa80000000000000 using an OR function.

1.5.3.8 tlb

`tlb` <*vcpu*> <*start_index*> <*number_of_entries*>

Displays the TLB entries for the specified virtual CPU, starting at the specified index for the specified number of entries. If you do not specify a number of entries, this command displays only one entry.

1.5.3.9 th

`th`

Displays the threshold values for all error types.

`th` <*error_type*> <*threshold*>

Sets the threshold for the specified error type.

1.5.4 Process Commands

1.5.4.1 p

`p`

Displays the current process.

`p` <*vcpu*> <*pid*>

Selects the process in which the commands operate.

1.5.4.2 run

`run`

Runs the current process. Use this command to start a process that was loaded with the `load` command or to start a process that has been frozen.

`run` <*test*> <*options*>

Loads a diagnostic from the host, selects it as the current process, and then runs the diagnostic.

1.5.4.3 bp

`bp` <*vaddr*>

Sets a breakpoint at the specified virtual address.

When test execution reaches the breakpoint, execution stops and the breakpoint is removed. Then, you can examine memory, set another breakpoint, or restart the test with the `run` command. You can set only one breakpoint for a process at a time.

1.5.4.4 rb

`rb`

Removes the breakpoint that is set in the current process.

1.5.4.5 dv

`dva <`*vaddr*`> [`*count*`]`

Displays [*count*] words (in ASCII format) of virtual memory starting at address <*vaddr*>. If you do not specify a [*count*] value, this command displays 1 word.

`dvb <`*vaddr*`> [`*count*`]`

Displays [*count*] words (in byte format) of virtual memory starting at address <*vaddr*>. If you do not specify a [*count*] value, this command displays 1 word.

`dvc <`*vaddr*`> [`*count*`]`

Displays [*count*] words (in code format) of virtual memory starting at address <*vaddr*>. If you do not specify a [*count*] value, this command displays 1 word.

`dvl <`*vaddr*`> [`*count*`]`

Displays [*count*] words (in long format) of virtual memory starting at address <*vaddr*>. If you do not specify a [*count*] value, this command displays 1 word.

`dvs <`*vaddr*`> [`*count*`]`

Displays [*count*] words (in short format) of virtual memory starting at address <*vaddr*>. If you do not specify a [*count*] value, this command displays 1 word.

`dvw <`*vaddr*`> [`*count*`]`

Displays [*count*] words (in word format) of virtual memory starting at address <*vaddr*>. If you do not specify a [*count*] value, this command displays 1 word.

1.5.4.6 mmap

`mmap`

Displays the memory map of the current process.

1.5.4.7 pmap

`pmap`

Displays the physical memory reservation table of the current process.

1.5.4.8 ef

`ef`

Displays the exception frame of the current process. (The exception frame includes the general purpose registers [GPRs], cause, status, badVaddr, and EPC.)

1.5.4.9 w

`w` <*vaddr*>

Sets a watchpoint for read or write operations for the current process at address <*vaddr*>.

1.5.4.10 wr

`wr` <*vaddr*>

Sets a watchpoint for read operations only for the current process at address <*vaddr*>.

1.5.4.11 ww

`ww` <*vaddr*>

Sets a watchpoint for write operations only for the current process at the virtual address that you specify with <*vaddr*>.

1.5.4.12 rw

`rw`

Removes a watchpoint for the current process.

Chapter 2

Diagnostic Tests

This chapter describes the sMDK-based diagnostic tests:

- Barrier circuitry test (`barr`)
- IO7 test (`io7bt`)
- Directory memory test (`ndir`)
- GSN test (`netkiller`)
- Node memory test (`nmem`)
- Real-time clock test (`rtc`)
- Router basic test (`rtrb`)
- Router traffic test (`rtrt`)
- Cache coherency test (`scct`)
- System-level stress test (`System_Level_Test`)
- Xbridge test (`xbg`)

2.1 Barrier Circuitry Test (barr)

The barrier circuitry test (`barr`) is a directed test that checks the router ASIC local block registers that are used for barrier configuration (VECTOR_HW_BAR) and the state machine fan-in / fan-out logic that is used for barrier operation.

2.1.1 Test Sections

Section 0 performs a basic barrier operation test.

This section verifies that the CPUs in each node can access the HW_BARRIER_CONFIG register in the local router with vector operations. It also checks the data retention capabilities of these registers and the capability of the router to detect and respond to an invalid register address.

Section 1 performs a single node test.

This section performs the simplest barrier circuit test possible to verify the basic "complete -> joined -> complete" transition of the barrier state machines.

Section 2 performs a single router test.

This section tests barrier operation with a router as *root* and all CPUs in the nodes that are locally attached to the router as active *children*.

Section 3 tests the entire system.

This section configures and tests the entire system as a single barrier tree. This section is not executed if less than two routers are present in the system.

2.1.2 Running the barr Test

Perform the following procedure to run the `barr` test:

1. Start sMDK.
2. Enter the following command to run `barr`:

   `run barr [`*`test_options`*`]`

   Refer to Table 2-1 for command line options for the `barr` test.
3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`
4. If the test detects errors, enter the following command to view the ERROR display:

   `dscr <`*`index_value`*`> error 1`
5. Enter the following command to stop the test:

   `drop <`*`index_value`*`>`

Table 2-1 barr Test Command Line Options

Option	Description
`-S` <*section_select*>	Selects the section to run (4-bit hexadecimal bitmask)
`-x` <*barrier_contexts*>	Selects the barrier contexts to test (32-bit hexadecimal bitmask)
`-e` <*ending_pass*>	Specifies an ending pass count (default: 1,000)
`-h | -H`	Displays help information

2.1.3 Test Output

The `barr` test creates an error display when it logs errors. (The error display is available only when errors have been logged.)

Example:

```
BARR - Barrier Test Error Screen Entry 1 of 4
----------------------------------------------------------------
Improper ending barrier sequence after barrier operation
VCPU 0x0, Nasid 0x0( fru# NFRU)
Virtual Router 0x0( fru# NFRU)
FRU: identification not possible
Failing Barrier Context: 0
Current Barrier Configuration: 0x3842
Expected Sequence: 0x1
Actual Sequence: 0x0
Failing Pass count: 0
Failing Section: 3, Condition 0
```

2.1.4 Troubleshooting Tips

When this test detects an error, the failing CPU halts, which may cause other CPUs to fail with synchronization errors.

2.2 IO7 Test (io7bt)

The IO7 test (`io7bt`) is a directed test that verifies all of the IO7 hardware that is embedded on the motherboard of an I brick (IOC3, USB, and IEEE 1394 [FireWire] components).

2.2.1 Test Sections

Section 0 performs a device access test.

This section verifies that each selected device can be accessed. It tests the PCI configuration space, resets, reset propagation to each device, and device-to-device reads and writes.

Section 1 verifies the IOC3 hardware.

This section tests all areas of the IOC3, including SSRAM, parity on SSRAM, PHY, and external loopback.

Section 2 verifies the USB hardware.

Section 3 verifies the IEEE 1394 hardware.

Section 4 tests concurrent data transfers.

This section simultaneously activates as many devices as possible to stress the system.

2.2.2 Running the io7bt Test

Perform the following procedure to run the `io7bt` test:

1. Start sMDK.
2. Enter the following command to run `io7bt`:

   `run io7bt [`*`test_options`*`]`

   Refer to Table 2-2 for the command line options for the `io7bt` test
3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`
4. If the test detects errors, enter the following command to view the ERROR display:

   `dscr <`*`index_value`*`> error 1`
5. Enter the following command to stop the test:

   `drop <`*`index_value`*`>`

Table 2-2 io7bt Test Command Line Options

Option	Description
`-C` [*value*]	Selects the action to perform when an error occurs: 0 = continue testing 1 = stop testing 2 = fill the buffer and stop testing
`-d` [*bitmask*]	Specifies the device to use (bitmask) xxx1 = test the IOC3 hardware (slot 0) xx1x = test the USB hardware (slot 4) x1xx = test the IEEE 1394 hardware (slot 5)
`-e` [*ending_pass*]	Specifies the ending pass count (default: 0 [run forever])
`-h | -H`	Displays help information
`-l` [*value*]	Selects the looping option 0 = loop mode off 1 = loop mode on (repeats the test based on the `-s` and `-e` values) **Note:** If you set the `-e` option to 0, this option is disabled.
`-n` [*node*]	Selects a specific node to use, which isolates testing to a specific node
`-p` [*port*]	Selects a specific port to use, which isolates testing to a specific port
`-s` [*starting_pass*]	Specifies the starting pass count (default: 0)
`-S` [*bitmask*]	Selects which sections of the diagnostic to run
`-E` [*bitmask*]	Indicates that a connector is attached to the Ethernet port Bits 3 through 0 indicate that a connector is attached to the Ethernet port (bit 1 indicates that an external loopback connector is attached to the Ethernet port)

Table 2-2 (continued) io7bt Test Command Line Options

Option	Description
`-I` [*value*]	Indicates that a connector is attached to the SuperIO port Bits 3 through 0 indicate that a connector is attached to the SuperIO port or that RTO is looped back to RTI Bits 8 through 4 indicate the baud rate for the SuperIO port x1xx = RTO is looped back to RTI (externally) 1xxx = RTO scope / measure mode (loop the code forever) xxxx1 = set SuperIO port to 9600 baud xxx1x = set SuperIO port to 19,200 baud xx1xx = set SuperIO port to 38,400 baud x1xxx = set SuperIO port to 56,000 baud 1xxxx = set SuperIO port to 115,000 baud
`-N` [*value*]	Selects NVRAM testing mode: 0 = limited testing (check 0x7fd1 - 0x7ff7, does not check 0x7ff0) 1 = maximum testing (preserves NVRAM contents, if possible) 2 - pattern all NVRAM (even locations = 0x25; odd locations = 0x52) 3 = read / check all NVRAM locations 4 = read / check limited areas of NVRAM 0x*XXX*0*YYY*f = dump NVRAM (*XXX*0 = first byte and *YYY*f = last byte) To perform NVRAM testing, you must use the `-N` option. **Caution:** Do not stop the test while it is testing the NVRAM. The NVRAM stores important system data.
`-F` [*value*]	Specifies that the test should perform FireWire specific operations 0 = no FireWire specific parameters 1 = dump the physical registers from pages 0, 1, and 7
`-U` [*value*]	Specifies that the test should perform USB-specific operations 0 = no USB-specific parameters 1 = attempt to detect an external USB device and then stop the test

2.2.3 Test Output

The io7bt test uses the RUN display to show test progress.

Example:

```
IO7BT Run Display for node 0x000 cpu 0x001                    Page 1 of 1
                                                           Pass = 00000045
                                                           Fail = 00000000

 Sec & Description                      Selected          Pass         Fail
 -----------------                      --------        --------     --------
  0 - PCI config regs and reset test         1          00000046     00000000
  1 - IOC3 specific testing                  1          00000045     00000000
  2 - USB  specific testing                  1          00000045     00000000
  3 - 1394 specific testing                  1          00000045     00000000
  4 - Concurrent device test                 0          00000000     00000000


  Running section 01  sub-section 02  condition 01

  Testing superio serial port - running internal loopback mode at 38400 baud
```

The io7bt test uses the ERROR display to report any errors.

Example:

```
IO7BT Error Display                      Error at Pass 00000000
                                      Repeat Error Cnt 00000000

Error #0x1 of 0x1 From Node 0x000 Nasid 0x000 CPU 0x001 Port 0x008
-----------------------------------------------------------------
Failing section         0x01
Failing sub-section     0x02
Failing condition       0x01

Received data is        0x00000010
Expected data is        0x00000000
Failing address         0x9200000008220178
Source code location(s) sec1_2.c  line:386


Reason for error: SuperIO internal loopback read error - character
sent out was a 0x0
```

2.2.4 Troubleshooting Tips

1. Determine the failing section, subsection, and condition.
2. Check all connectors and cables that are used for the hardware that is being tested by that section, subsection, and condition combination.
3. Power cycle the system to verify that the failure remains.
4. If the failure remains, replace the I brick.

2.3 Directory Memory Test (ndir)

The directory memory test (`ndir`) is a directed test that verifies that the directory memory on each node is operating properly.The `ndir` test verifies the paths from the hub chip to directory memory on each local node, tests the integrity of directory memory on each local node, and verifies proper SECDED operation in directory memory. The `ndir` test performs these functions at the node level and does not cross node boundaries.

2.3.1 Test Sections

This test uses five test sections. (Test complexity increases as the section number increases.)

Section 0 performs a directory memory quick-screen test.

This section uses one user-selectable CPU on a node to write data patterns to directory memory and to read data back from the directory memory. (It uses directory memory backdoor access to execute a March test algorithm with address data.) It compares the data that was written with the data that was read.

Section 1 performs a directory memory integrity test.

This section uses all available CPUs on a node to verify the integrity of each memory cell in the directory memory of a node. (It uses directory memory backdoor access to execute a March test algorithm with checkerboard and random data patterns.)

Section 2 performs a directory memory SECDED verification test.

This section verifies that the SECDED hardware corrects single-bit errors and detects double-bit errors in directory memory. It forces errors and tests interrupts.

Section 2 uses directory memory backdoor access to access directory memory. It checks SECDED for the directory entries only and not the region protection nor the page counter that are both in the same physical memory area. (The region protection is protected by a parity for each bit, and the page counters have no protection scheme.) This section also checks the parity error detection for the region protection data.

Section 3 performs a directory memory random address / data test.

This section uses all available CPUs on a node to test randomly selected blocks in directory memory on that node. For each iteration, section 3 selects a random block of contiguous directory memory, a random stride, and an initial random data pattern. It assigns each CPU a different starting backdoor directory address based on a CPU index number. It uses addresses that wrap to the starting address (plus an offset to stay within the currently selected block).

Section 4 performs a directory state verification test.

This section uses multiple CPUs on a node to verify directory memory operation. The CPUs share cache lines in memory. This section verifies that directory memory is updated with the correct state information during memory accesses.

Section 4 runs in short or long mode. In short mode, this section verifies only one directory state entry in each page (4,096 bytes). In long mode, this section verifies the directory state entry for each cache line (128 bytes).

2.3.2 Running the ndir Test

Perform the following procedure to run the `ndir` test:

1. Start sMDK.
2. Enter the following command to run `ndir`:

   `run ndir [`*`test_options`*`]`

   Refer to Table 2-3 for the command line options for the `ndir` test.
3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`
4. If the test detects errors, enter the following command to view page 2 of the RUN display:

   `dscr <`*`index_value`*`> run 2`
5. Enter the following command to stop the test:

   `drop <`*`index_value`*`>`

Table 2-3 ndir Test Command Line Options

Option	Description
`-n` <*nodes*>	Selects the nodes to test (default: all available nodes)
`-c` <*cpu_list*>	Selects the CPUs to use (default: all available CPUs)
`-S` <*section_select*>	Selects the test sections to run (default: 0x1F, which selects all sections)
`-s` <*starting_pass*>	Specifies the starting pass (default: 0)
`-e` <*ending_pass*>	Specifies the ending pass (default: 1)
`-l`	Turns on loop mode, which causes the test to run forever between the starting pass and the ending pass
`-C`	Selects continue-on-error mode
`-b` <*banks*>	Selects the memory banks to use
`-m` <*starting_address*>	Selects the starting directory address
`-M` <*ending_address*>	Selects the ending directory address
`-? | -h | -H`	Displays help information

2.3.3 Test Output

The second page of the RUN display reports error information.

Example:

```
NDIR Run Display - Error Information                        Page 2 of 3
_____________________________________________________________________________
                                                  Error Number: 0002 of 0002
VCpu...:  0000,   Pid........:  0003,             Error Count.: 0001
VNode..:  0000,   Nasid......:  0000,
Section: 0001,  Sub-Section: 0000,    Condition: 0012, Pass: 0x00000001
File...: ndirFuncs.c,                 Line.....: 0147, Fail: 0x00000002
_____________________________________________________________________________

   NodeSync Error:  Timed out waiting for 2 cpus to sync up.

   1 cpus waited for 120000000 us. before timeout.

   Sync Id: 0x100000a8.

   Time-out waiting for the following cpu processes to sync up:
   VCpu: 0, Pid: 3
   VCpu: 1, Pid: 3
_____________________________________________________________________________
 [1]=next pg, [2]=next vnode, [3]=next error, [shift-num] for previous
```

2.3.4 Troubleshooting Tips

Additional information might be available in the error log. Use the `el` command to view the error log.

Use one of the following methods to decode the error log:

- The form at the following URL:

  `http://wwwcf.americas.sgi.com/PUBLIC/sn1_diags/elog_decode/elog_decode.cgi`

- The `elog_decode` utility. (Refer to Appendix A, “elog_decode Utility,” for more information.)

2.4 GSN Test (netkiller)

The Gigabyte System Network (GSN) test (`netkiller`) is a stress test that verifies GSN XIO boards and X bricks. It checks the Scheduled Transfer and Admin logic in the GSN boards, and the SSRAM, firmware processor (FWP), LLP, IC_BUS, and PH6400 interface.

Note: `netkiller` also runs as part of the system-level stress test (`System_Level_Test`).

2.4.1 Test Sections

Section 0 runs the GSN test.

2.4.2 Running the netkiller Test

Perform the following procedure to run the `netkiller` test:

1. Start sMDK.
2. Enter the following command to run `netkiller`:

   `run netkiller [`*`test_options`*`]`

   Refer to Table 2-4 for the command line options for the `netkiller` test.
3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`
4. If the test detects errors, enter the following command to view the ERROR display:

   `dscr <`*`index_value`*`> error 1`
5. Enter the following *command* to stop the test:

   `drop <`*`index_value`*`>`

Table 2-4 netkiller Test Command Line Options

Option	Description
`-t <`*testname*`>`	Selects the test to run: xtwalkingbus, shacreg, shac_reset, fwpreset, ssram_addr, ssram_switching, ssram_randstress, st_loopback, or admin_check
`-e <`*ending_pass*`>`	Selects the ending pass (default: 800)
`-h <`*num*`>`	Specifies the hop count
`-i <`*num*`>`	Enables internal loopback for the GSN board (set `<num>` to a nonzero value to enable) (default: enabled)
`-p <`*num*`>`	Disables SSRAM parity (when *<num>* is set to a nonzero value)
`-v <`*num*`>`	Enables verbose mode (set *<num>* to a nonzero value to enable)

Table 2-4 (continued) netkiller Test Command Line Options

Option	Description
`-x <num>`	Selects the test case to run: 1 = 1-way striping, 16M bufx, LLP pattern, 16M fixed length, data check on, and multiple VC 2 = 1-way striping, 16M bufx, random pattern, random length generated every pass, data check on, and multiple VC 3 = 8-way striping, 2Kbufx, random pattern, 16M fixed length every pass, data check on, and mixed st and raw. 4 = 1-way striping, 16Mbufx, random pattern, and 50 X 2K random length data check on (This is default case when `System_Level_Test` runs `netkiller`.) 5 = 8-way striping, 16Mbufx, LLP pattern, and 16M fixed length data check off
`-o`	Configures the test to ignore unused ports

2.4.3 Test Output

The ERROR display reports error information

Example:

```
>>Starting ST Loopback Test<<
running st tx/rx test pass 0
ERROR: DMA completion timeout on shac number 0


Shac Detected ERROR
Xbrick [1], gsn_ioslot [1]
TX VC Full Timeout
ic error status low reg value = 0x00040000

Shac Detected ERROR
Xbrick [1], gsn_ioslot [4]
TX VC Full Timeout
ic error status low reg value = 0x00080000
Stopping on Shac ERROR, stop_on_shac_error is enabled
```

2.5 Node Memory Test (nmem)

The node memory test (`nmem`) is a directed test that verifies that the memory on each node is operating properly: It verifies the paths from the Bedrock ASIC to memory on each local node, tests the integrity of memory on each local node, and verifies proper SECDED operation. The `nmem` test performs these functions at the node level and does not cross node boundaries.

2.5.1 Test Sections

This test has five sections. (Test complexity increases with each successive section.)

Section 0 performs a quick-screen test.

This section uses one user-selectable CPU on a node to write data patterns to memory and to read data back from the memory. (It uses the March test algorithm, which repeats an address pattern in each 32-bit word of a quadword.) It compares the data that was written with the data that was read.

Section 1 performs a memory integrity test.

This section uses all available CPUs on a node to verify the integrity of each memory cell in the main memory of a node. (It uses the March test algorithm with checkerboard and random data patterns.) This section also tests memory accesses with strides across cache lines, banks (internal), and memory pages (rows).

Section 2 performs a SECDED verification test.

This section uses all available CPUs on a node to verify the integrity of each memory cell in SECDED memory. This section also verifies that the SECDED hardware corrects single-bit errors and detects double-bit errors. This section forces errors and tests interrupts.

Section 3 performs a memory random address and data test.

This section uses all available CPUs on a node to verify the data integrity of memory. It writes random data to random addresses but does not verify data. It relies on SECDED to report any memory errors.

Section 4 performs a fetch-and-op test.

This section verifies proper operation of fetch and op operations by using uncached memory references in the Mspec space. It uses up to four processors to execute walk-by code, which verifies that references to the same location at the same are handled correctly. It uses the following operation types: fetch, fetch and increment, fetch and decrement, fetch and clear, initialize, increment, decrement, AND, and OR.

2.5.2 Running the nmem Test

Perform the following procedure to run the `nmem` test:

1. Start sMDK.
2. Enter the following command to run `nmem`:

   `run nmem [`*`test_options`*`]`

   Refer to Table 2-5 for the command line options for the `nmem` test.
3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`
4. If the test detects errors, enter the following command to view page 2 of the RUN display:

   `dscr <`*`index_value`*`> run 2`
5. Enter the following command to stop the test:

   `drop <`*`index_value`*`>`

Table 2-5 nmem Test Command Line Options

Option	Description
`-n` <*nodes*>	Selects the nodes to test (default: all available nodes)
`-c` <*cpu_list*>	Selects the CPUs to use (default: all available CPUs)
`-S` <*section_select*>	Selects the test sections to run (default: 0x1F, which selects all sections)
`-s` <*starting_pass*>	Specifies the starting pass (default: 0)
`-e` <*ending_pass*>	Specifies the ending pass (default: 1)
`-l`	Turns on loop mode, which causes the test to run forever between the starting pass and the ending pass
`-C`	Selects continue on error
`-b` <*banks*>	Selects the memory banks to use
`-m` <*starting_address*>	Selects the node memory starting address
`-M` <*ending_address*>	Selects the node memory ending address
`-? | -h | -H`	Displays help information

2.5.3 Test Output

The second page of the RUN display reports error information.

Example:

```
NMEM Run Display - Error Information                              Page 2 of 3
_____________________________________________________________________________

                                                    Error Number: 0001 of 0005
VCpu...:  0001,    Pid........: 0001,                Error Count.: 0001
VNode..:  0000,    Nasid......: 0000,
Section:  0001,    Sub-Section: 0001,    Condition: 0005,  Pass:  0x00000002
File...:  nmem     Sec1.c,               Line.....: 1027,  Fail:  0x00000001
_____________________________________________________________________________

    Error:  Data Miscompare Error

Phys Address: 0xa80000004a4ed5d0       Expected Data: 0xd0 0x4f3884df4cee4bdc
Bank........: 1                        Actual Data..: 0xd3 0x4f3884df4cee7bdc
                                       Difference...: 0x03 0x0000000000003000




_____________________________________________________________________________
[1]=next pg, [2]=next vnode, [3]=next error, [shift-number] for previous
```

2.5.4 Troubleshooting Tips

Additional information might be available in the error log. Use the `el` command to view the error log.

Use one of the following methods to decode the error log:

- The form at the following URL:

  `http://wwwcf.americas.sgi.com/PUBLIC/sn1_diags/elog_decode/elog_decode.cgi`

- The `elog_decode` utility. (Refer to Appendix A, “elog_decode Utility,” for more information.)

2.6 Real-time Clock Test (rtc)

The real-time clock test (`rtc`) is a directed test that checks the system real-time (or global) clock distribution hardware. (This hardware includes the router real-time clock MUX and interconnecting cables.)

This test moves the clock source from node to node and adjusts the clock MUXes in the routers. This process enables the test to select each router port as the source of the clock signal that it broadcasts to all other router ports. When this test verifies that a node receives the clock signal, it checks only if the clock is incrementing; it does not verify the frequency of the clock signal.

Note: This test checks only connected router ports. It does not check looped-back links, links to routers that are acting as repeaters, or router ports that are not connected.

2.6.1 Test Sections

Section 0 is a C brick-to-R brick link test.

This section verifies clock operation for C brick-to-R brick links. It requires at least two C bricks that are directly connected to an R brick. It uses each C brick as the clock source and then verifies the clock at each C brick.

Section 1 is a C brick-to-R brick link test.

This section verifies clock operation for C brick-to-R brick links for routers with only a single directly-connected C brick. It requires at least one R brick-to-R brick link in the clock path. It verifies the clock on each port of the router that is attached to the C brick under test, using the shortest path available to another C brick.

Section 2 is an R brick-to-R brick link test.

This section verifies clock operation for R brick-to-R brick links by checking all router ports that were not tested by sections 0 and 1.

2.6.2 Running the rtc Test

Before you run this test:

- Verify that the system configuration includes at least one router and two nodes (C bricks).
- Verify that the processors, memory, cache, and bedrock interfaces are functional.

Perform the following procedure to run the `rtc` test:

1. Start sMDK.
2. Enter the following command to run `rtc`:

   `run rtc [`*`test_options`*`]`

   Refer to Table 2-7 for the command line options for the `rtc` test.
3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`
4. If the test detects errors, enter the following command to view the ERROR display:

   `dscr <`*`index_value`*`> error 1`
5. Enter the following command to stop the test:

   `drop <`*`index_value`*`>`

Table 2-6 rtc Test Command Line Options

Option	Description
`-s` <*starting_pass*>	Specifies the starting pass count value (default: 0)
`-e` <*ending_pass*>	Specifies the ending pass count (default: 0; run forever)
`-S` <*section_select*>	Selects the sections to run (hexadecimal bitmask, default: 0x7) Example: `-s 0x3` runs sections 0 and 1
`-C`	Selects continue-on-error action: 0 = continue testing 1 = stop testing 2 = fill the buffer and stop testing (default: 1; stop testing)
`-h | -H | -?`	Displays help information

2.6.3 Test Output

The `rtc` test creates an error display when it logs errors. (The error display is available only when errors have been logged.)

Example:

```
RTC Diagnostic - Error Screen         Page 1 of 1
--------------------------------------------------
Global Clock would not run

Failing pass count: 0x00000000
Failing section: 2, condition: 1, srccode line: 274

Failing path:
  node    3 (007c27) RTC generated    -->
    port 2 router  1 (007r23) port 1 -->
    port 1 router  0 (007r25) port 2 -->
  node    0 (007c11) RTC monitored
```

2.6.4 Troubleshooting Tips

When the `rtc` test logs an error, the error display shows the clock path from the node that is the clock source to the node that is monitoring the clock signal. It is important to remember that some of the components in the path might have passed testing in a prior test section and, therefore, are unlikely sources of the failure.

For example, if a failure occurs in section 2, you can eliminate the C brick-to-R brick components in the failing path if test sections 0 and 1 passed.

2.7 Router Basic Test (rtrb)

The router basic test (`rtrb`) is a directed test that checks the router ASIC local block and lookup table register integrity and basic functionality. The `rtrb` test:

- Verifies CPU access of router local block registers
- Verifies CPU access of router lookup tables
- Verifies local block reset
- Verifies lock, test, and set functions of the router scratch registers
- Tests vector routing through all available router ports

2.7.1 Test Sections

Section 0 performs a router local block register integrity test.

This section verifies that a CPU on each node can use vector operations to write and read all applicable router local block registers and lookup tables on the local router. It also performs and verifies local block resets. It does not write data to registers and fields that cause undesirable side effects (PORT _RESETS, DIAG_ PARMS, Global Clock Selects in GLOBAL _PARM0, etc.); however, it does attempt to read these registers and fields.

Section 1 performs a scratch register functional test.

This section tests the lock as well as test and set functions of the router scratch registers. All CPUs (on up to 4 nodes) that are directly attached to a router perform the test.

Section 2 performs a vector routing test.

This section tests vector routing through all available router ports. This section uses the configuration structure of sMDK to determine the router links that are available to test. It initializes each router SCRATCH_REG0 register with a router tag that the section uses to verify that the correct target was reached. All CPUs run the test concurrently.

2.7.2 Running the rtrb Test

Perform the following procedure to run the `rtrb` test:

1. Start sMDK.
2. Enter the following command to run `rtrb`:

   `run rtrb [`*`test_options`*`]`

   Refer to Table 2-7 for the command line options for the `rtrb` test.

Note: During testing, the `rtrb` test periodically disables the sMDK kernel, which causes sluggish keyboard response. This activity is normal when you are running this test.

3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`

4. If the test detects errors, enter the following command to view the ERROR display:

   `dscr <`*`index_value`*`> error 1`

5. Enter the following command to stop the test:

   `drop <`*`index_value`*`>`

Table 2-7 rtrb Test Command Line Options

Option	Description
`-s <`*starting_pass*`>`	Specifies the starting pass count value
`-e <`*ending_pass*`>`	Specifies the ending pass count (default: run forever)
`-S <`*section_select*`>`	Selects the sections to run (hexadecimal bitmask)
`-h | -H | -?`	Displays help information

2.7.3 Test Output

The `rtrb` test creates an error display when it logs errors. (The error display is available only when errors have been logged.)

Example:

```
RTRB Diagnostic - Error Logging Screen
------------------------------------ -
Error Logged by CPU 0 Node 0.
The target router was 0.
Failing Link is 1, 0 is the Router Local Block, 1- 8 is port a- h.
Failing Test Section 2, Condition 1.
A ROUTER or HUB REGISTER MISCOMPARE Error occured.
Actual Data ............ 0x0001000100010001.
Expected Data .......... 0xfffefffefffefffe.
Difference Data ........ 0xffffffffffffffff.
Failing register address 0x0000000000000100.
```

On the error display, note the failing section and condition, the actual and expected data, and the error type.

2.8 Router Traffic Test (rtrt)

The router traffic test (`rtrt`) is a directed test that checks the router network, router ASIC, Bedrock ASIC, and interconnecting cables.

2.8.1 Test Sections

Section 0 performs a one-to-one traffic test, during which each node targets a different node.

This section causes each node to target a prepatterned shared data buffer on a different node. (Node 0 targets node 1, node 1 targets node 2, and so on). It writes a long burst of data to the target node and then reads the data from the target node.

Section 1 performs an all-to-all traffic test, during which each node targets all other nodes.

This section causes each node to target prepatterned shared data buffers on all other nodes. The selected node writes short bursts (10 cache lines) of data to and reads short bursts of data from the target nodes in rapid succession.

Section 2 performs an all-to-one traffic test, during which all nodes target one node.

This section causes each node to target a prepatterned shared buffer in a node that is attached to the same router (or near it if a MetaRouter is involved). The selected node writes a long burst of data to and reads data from the target node.

Section 3 performs a random traffic test, during which each CPU randomly selects a node for communication.

This section causes each CPU to randomly choose a prepatterned shared data buffer on another node. Then, each CPU randomly performs write or read transfers of random length. This section does not check the data.

2.8.2 Running the rtrt Test

Perform the following procedure to run the `rtrt` test:

1. Start sMDK.
2. Enter the following command to run `rtrt`:

   `run rtrt [`*`test_options`*`]`

   Refer to Table 2-8 for the command line options for the `rtrt` test.
3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`
4. If the test detects errors, enter the following command to view the ERROR display:

   `dscr <`*`index_value`*`> error 1`
5. Enter the following command to stop the test:

   `drop <`*`index_value`*`>`

Table 2-8 rtrt Test Command Line Options

Option	Description
`-s` <*starting_pass*>	Specifies the starting pass (default: 0x0)
`-e` <*ending_pass*>	Specifies the ending pass (default: 0 [run forever])
`-S` <*section_select*>	Selects the sections to run (hexadecimal bitmask [default: 0xf])
`-c` <*virtual_cpu_select*>	Selects the CPUs to use (default: run all cpus) Enter a comma-separated list of virtual CPUs and use dashes to indicate a range of CPUs; for example: `-c 0-7,10-31` (Uses all CPUs except CPUs 8 and 9 in a 32-CPU system.)
`-n` <*virtual_node_select*>	Selects the virtual nodes to use (default: use all nodes) Enter a comma-separated list of virtual nodes and use dashes to indicate a range of nodes; for example: `-n 0-3,6-7` (Uses all nodes except nodes 4 and 5 in an 8-node system.)
`-D` <*data_check*>	Turns data checking on or off (default: on) 0 = off 1 = on
`-B` <*barrier_enable*>	Turns background barrier enable on or off (default: off) 0 = off 1 = on
`-b` <*hub_register_enable*>	Turns background hub scratch register packets enable on or off (default: off) 0 = off 1 = on

Table 2-8 (continued) rtrt Test Command Line Options

Option	Description
`-v` <*vector_register_enable*>	Turns background vector scratch register packets enable on or off (default: off) 0 = off 1= on
`-t` <*sync_type*>	Specifies the synchronization method that the test sections should use (default: fetch and increment) f = fetch and increment b = barrier
`-d` <*data_types*>	Selects the data types to run (hexadecimal bitmask; default: 0x7f) 0x1 = alternating 0x2 = sliding 0s and 1s 0x4 = checkerboard 0x8 = address 0x10 = negative address 0x20 = random 0x40 = simultaneous switching 0x80 = user data pattern
`-u` <*user_data*>	Specifies the user data pattern to use with the `-d` command line option
`-h | -H | -?`	Displays the help information

2.8.3 Test Output

The `rtrt` test creates an error display when it logs errors. (The error display is available only when errors have been logged.)

For example:

```
RTRT Diagnostic - Error Logging Screen
--------------------------------------
Error Logged by CPU 1 Node 0.
The target node was 1.
Failing Test Section 0, Condition 0.
A DATA MISCOMPARE Error occured.
Actual Data 0x00000000deadbeef.
Expected Data 0xf00010fffef00010.
Difference Data 0xf00010ff205dbeff.
Failing address 0xa80000040021d020
```

On the error display, note the failing section and condition, the actual and expected data, and the error type.

2.9 Cache Coherency Test (scct)

The cache coherency test (`scct`) is a system-level test that dynamically generates new test directives each pass. It divides the CPUs into "groups" of 1 to 8 CPUs. Test parameters enable you to select the group size and the group mode (for example, subnode, node, or random-each-pass). As the system size grows, the random group mode becomes more complex because the independent groups compete for shared memory and network resources.

During each iteration of the test, the CPUs in a group test coherency conflicts by *false sharing* a set of cache lines. Some of the CPUs false share the target cache lines with memory or PIO accesses. The other CPUs access lines outside of the test area with memory, PIO, or BTE activity, which causes memory contention without flushing or invalidating the cache lines in the test area.

Note: PIO and BTE accesses are not implemented yet.

2.9.1 Test Sections

Section 0 performs a basic addressing test.

Section 1 performs an all-to-one victims test (the target CPU generates many explicit write backs [cache victims]).

Section 2 performs an all-to-one interventions test (the target CPU generates requests for cache lines that do not generate write backs).

Section 3 performs an all-to-all test (processors use a set of false cache lines).

2.9.2 Running the scct Test

Perform the following procedure to run the `scct` test:

1. Start sMDK.
2. Enter the following command to run `scct`:

   `run scct [`*`test_options`*`]`

   Refer to Table 2-9 for the command line options for the `scct` test.
3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`
4. If the test detects errors, enter the following command to view page 2 of the RUN display:

   `dscr <`*`index_value`*`> run 2`
5. Enter the following command to stop the test:

   `drop <`*`index_value`*`>`

Table 2-9 scct Test Command Line Options

Option	Description
`-h | -H | -?`	Displays the help information
`-s` <*starting_pass*>	Specifies the starting pass
`-e` <*ending_pass*>	Specifies the ending pass
`-l`	Causes the test to loop between the selected starting pass and ending pass
`-S` <*bitmask*>	Selects the test section to run (bitmask)
`-c` <*cpu_list*>	Selects the virtual CPUs to use
`-m` <*value*>	Specifies the lowest physical byte address that can be accessed in each node (The actual range of addresses to test is determined by configuration, resource availability, and the `-M` and `-b` options. Page 3 of the RUN display lists the memory segments that are tested.)
`-M` <*value*>	Specifies the highest physical byte address that can be accessed in each node
`-b` <*select_mask*>	Selects the physical memory banks to use in each node (a bitmask where bits 0 through 7 correspond to banks 0 through 7)
`-g` <*group_mode*>	Selects the CPU group mode: 0 = CPUs share the same PI 1 = one CPU per PI within a node 2 = all available CPUs within a node 3 = random selection made before the first pass 4 = random selection made before each pass
`-G` <*group_size*>	Specifies a target group size from 1 to 8 CPUs (Use this parameter with random group modes. If you specify a value of 0, the test randomly selects a group size between 2 and 5.)
`-v` <*verbosity_level*>	Specifies the verbosity level of the test (0 through 2)

2.9.3 Test Output

scct reports error information on the second page of the RUN display.

Example:

```
SCCT Run Display - Error Information                          Page 2 of 4

VCpu:4    VNode:2    Nasid:3    Pid:1    Gid:1


error type    : data miscompare          expected data   : 0x01001b4cbc303180
test section  : 0                        actual data     : 0x0000000000000000
physical addr : 0xa800000002b03180       logical diff    : 0x01001b4cbc303180
virtual addr  : 0x40000b4cbc303180       arithmetic diff : 0x01001b4cbc303180
more info at  : 0xa8000000001ff318
```

2.10 System-level Stress Test (System_Level_Test)

The system-level stress test (`System_Level_Test`) test is a stress test that generates and simulates realistic customer use of the hardware by testing the full range of the architecture. `System_Level_Test` generates a large amount of data traffic through the Bedrock ASIC and the rest of the system. It creates many different types of data traffic through the system and generates as many different states in the system as possible.

`System_Level_Test` runs in passes. The phases of each pass are:

- Generation
  - Generates instructions
  - Generates addresses
  - Generates data
  - Generates I/O
- Execution
  - Executes all randomly generated tests
- Result checking
  - Compares results obtained by different CPUs
  - Compares the destination results with the source results
  - Checks the error bits in the hub error registers

2.10.1 Test Sections

Section 0 performs a system-level stress test.

2.10.2 Running the System_Level_Test Test

Perform the following procedure to run the `System_Level_Test`:

1. Start sMDK.
2. Enter the following command to run `System_Level_Test`:

   `run System_Level_Test [`*`test_options`*`]`

   Refer to Table 2-10 for the command line options for the `System_Level_Test` test.
3. Enter the following command to view the RUN display:

   `dscr <`*`index_value`*`> run 1`

4. If the test detects errors, enter one of the following commands to view the appropriate ERROR display:

   dscr <*index_value*> memorycc_error 1 <*vcpu*> <*error_number*>

   dscr <*index_value*> bte_error 1 <*vcpu*> <*error_number*>

   dscr <*index_value*> pio_error 1 <*vcpu*> <*error_number*>

   dscr <*index_value*> io_ii_error 1 <*vcpu*> <*error_number*>

   dscr <*index_value*> io_gsn_error 1 <*vcpu*> <*error_number*>

5. Enter the following command to stop the test:

   drop <*index_value*>

Table 2-10 System_Level_Test Command Line Options

Option	Description
`-h`	Displays help information
`-s` <*starting_pass*>	Specifies the starting pass (default: 1)
`-e` <*ending_pass*>	Specifies the ending pass (default: 100000000)
`-l`	Causes the test to loop between the starting pass and ending pass
`-u` <*value*>	Specifies how often (in passes) the test should update the user (default: 10)
`-t` <*test*>	Specifies the tests to run (default: All): All = run block transfer engine (BTE), Memory_CC, IO, and PIO tests BTE = BTE test Memory_CC = random memory / cache coherency test IO = Gigabyte System Network (GSN) test PIO = PIO interference test CPU = random instruction test Network = random network test User_Test = user-defined test
`-r` <*register*>	Specifies the hub registers to modify (default: none)
`-f` <*field*>	Specifies the hub field to modify
`-i` <*value*>	Specifies the hub register field minimum value (default: none)
`-j` <*value*>	Specifies the hub register field maximum value (default: none)
`-n` <*node*>	Specifies the nodes to use (default: all nodes)
`-c` <*cpu_list*>	Specifies the CPUs to use (default: all CPUs)

Table 2-10 (continued) System_Level_Test Command Line Options

Option	Description
`-d <option>`	Specifies the values to dump and display (default: no dump): dump_all = display addresses, registers, instructions dump_addrs = display code and data addresses dump_gpr_fpr = display initial register values dump_instrs = display randomly generated instructions dump_HUB_Reg_default = display hub default register values dump_HUB_Reg_modified = display hub modified register values dump_BTE_parms = display BTE interface table dump_BTE_addrs = display BTE saved addresses dump_BTE_init_regs = display BTE initial values dump_BTE_final_regs = display BTE final values dump_BTE_results = display BTE destination and notification results dump_BTE_initiate_msg = display BTE initiate messages
`-w`	Specifies that the test should not write the trace table entries (default: write trace table entries)
`-y <sequence>`	Selects a special code sequence to use (default: no special code sequences): hub_starvation_1 = run hub starvation bug code sequences

2.10.3 Test Output

`System_Level_Test` periodically prints out the current pass that is being executed to indicate that the test is still running. When errors occur, `System_Level_Test` reports information about the following types of errors:

- Data miscompare

  `System_Level_Test` prints the address of the data miscompare and the expected and actual data patterns.

  Example:

```
ERROR: PASS: 10 FPR miscompare: cpu  1 fpr  4 = d435c341fc7fc7
                                cpu  0 fpr  4 = 0

Error Analysis: PASS  10

    The cpus that most likely caused the error are:

      Virtual   Nasid    Module    Node    Physical
        cpu                        slot      cpu
      -------   -----    ------    ----    --------
         0        0         2        1        A
```

- Error detected in a hub error register

  `System_Level_Test` prints the name of the register that contains the error and the value in the register.

- Error detected in an I/O error register

  `System_Level_Test` prints the name of the register that contains the error and the value in the register.

- Unexpected exception

  If an exception condition occurs, `System_Level_Test` stops printing the pass number. When this happens, you need to press the <Enter> key to return to the `smdk>` prompt. At the `smdk>` prompt, enter the `kl` command to view the kernel log.

  Example:

```
smdk> kl
vcpu 0: 1271

smdk> kl 0
Unexpected XTLB Refill exception...process:System_Level pid:2
cause:  0x0000000c (TLB miss on store)
status: 0xb40080e3
epc: 0x00000000100097c4 errorEpc: 0xffffffffbfc01d8c
badVaddr: 0000000000000000
count: 0x004c4ed6 compare: 0x01000000
hi: 0x0000000000000002 lo: 0x0000000000000000
general purpose registers:
r0 (zero): 0x0000000000000000  r16 (s0): 0x0000000000000000
r1   (at): 0x0000000000000001  r17 (s1): 0x0000000000000000
r2   (v0): 0x0000000000000008  r18 (s2): 0x0000000000000000
r3   (v1): 0x0000000000000000  r19 (s3): 0x0000000000000000
r4   (a0): 0x0000000000000000  r20 (s4): 0x0000000000000000
r5   (a1): 0x00000000100abcd8  r21 (s5): 0x0000000000000000
r6   (a2): 0x0000000000000045  r22 (s6): 0x0000000000000000
r7   (a3): 0x0000000000000000  r23 (s7): 0x0000000000000000
r8   (t0): 0x0000000000000000  r24 (t8): 0x0000000000007368
r9   (t1): 0x0000000000000100  r25 (t9): 0x0000000000008370
r10  (t2): 0x00000000103c48f8  r26 (k0): 0xc0000000000271b0
r11  (t3): 0x0000000000000008  r27 (k1): 0xc0000000000298d4
r12  (t4): 0x00000000100a14f8  r28 (gp): 0x0000000000000000
r13  (t5): 0x0000000010090000  r29 (sp): 0x00000fffffffde0
r14  (t6): 0x00000fffffffdd8  r30 (s8): 0x0000000000000000
r15  (t7): 0x00000fffffffdd0  r31 (ra): 0x0000000010008bd4
```

  To interpret the kernel log for this unexpected exception, perform the following actions:

  1. Disassemble the `System_Level_Test` binary file with this command:

     `dis -Sx System_Level_Test > System_Level_Test.dis &`

  2. Look for the EPC address that is contained in the kernel log in the `System_Level_Test.dis` file.

  3. Look at the instruction that is contained at the EPC address. (It is usually a load or a store instruction.)

  4. Look at the register that is used to hold the base address. You will probably find an invalid address in that register in the kernel log register display.

- System hang

  When the system hangs, it may be possible to enter POD mode and examine the hub and I/O error registers as well as other registers to determine the state of the system.

2.10.4 Troubleshooting Tips

1. Loop on the failing pass.
2. Loop on a set of passes surrounding the failing pass.
3. Print the generated instructions, addresses, and data.
4. View the trace table.

Note: If you want to reproduce an error, you can configure `System_Level_Test` to loop on a specific pass. `System_Level_Test` reproduces the same sequence of random instructions, addresses, and data that it generated the first time for that pass.

2.11 Xbridge Test (xbg)

The Xbridge test (xbg) is a directed test that checks the functionality of the Xbridge ASIC in the P and I bricks.

Note: This test requires Alteon PCI Gigabit Ethernet cards to test the PCI slots.

2.11.1 Test Sections

Section 0 performs a basic register test.

Section 1 performs a RAM on Array (ROA) test.

Section 2 performs an interrupt test.

Section 3 verifies PCI configuration and control.

Section 4 performs a PCI DMA basic test.

Section 5 verifies PCI page-mapped addressing.

Section 6 verifies PCI direct-mapped addressing.

Section 7 verifies PCI DMA byte-swap functionality.

Section 8 verifies PCI DMA buffers.

Section 9 performs a PCI DMA random test.

Section 10 performs a PCI DMA concurrent test.

2.11.2 Running the xbg Test

Perform the following procedure to run the xbg test:

1. Start sMDK.
2. Enter the following command to run xbg:

   run xbg [*test_options*]

   Refer to Table 2-11 for the command line options for the xbg test.
3. Enter the following command to view the RUN display:

   dscr <*index_value*> run 1
4. If the test detects errors, enter the following command to view the ERROR display:

   dscr <*index_value*> error 1

5. Enter the following command to stop the test:

   `drop <index_value>`

Note: If you test the Xbridge ASIC that controls the boot device, you must reboot the system after the test completes.

Table 2-11 xbg Test Command Line Options

Option	Description
`-h | -H`	Displays help information
`-i` <*io_brick*>	Selects the I brick to test (default: 0xffffffffffffffff [discover hardware])
`-n` <*vnode*>	Specifies the virtual node to use (default: 0xffffffffffffffff [discover hardware])
`-p` <*port*>	Specifies the port to use: bit 8 = XIO port 8 bit 9 = XIO port 9 bit 10 = XIO port A ... bit 15 = XIO port F (default: 0x000000000000f300 [all XBOW ports])
`-d` <*device*>	Selects the devices to use: bit 1 = port 8, device 1 bit 2 = port 8, device 2 ... bit 8 = port 8, device 8 bit 9–16 = port 9, devices 1-8 bit 17–24 = port A, devices 1-8 bit 25–32 = port B, devices 1-8 bit 33–40 = port C, devices 1-8 bit 41–48 = port D, devices 1-8 bit 49–56 = port E, devices 1-8 bit 57–64 = port F, devices 1-8 (default: 0xffffffffffffffff [all devices])
`-C` <*continue_flag*>	Selects what the test should do when an error occurs (0 = continue, 1 = stop, 2 = fill the buffer and stop testing) (default: 0x0000000000000001)
`-q`	Turns on quick-look mode (default: off)
`-s` <*starting_pass*>	Specifies the starting pass count for random seed (default: 0x0000000000000000)
`-e` <*ending_pass*>	Specifies the ending pass count (0 = run forever) (default: 0x0000000000000002)
`-l` <*value*>	Specifies the number of times to loop (from the starting pass count to the ending pass count) (default: 0x0000000000000001)

Table 2-11 (continued) xbg Test Command Line Options

Option	Description
`-S` <*bitmask*>	Selects the test sections to run: bit 0 = section 0 bit 1 = section 1 ... bit 10 = section 10 (default: 0x00000000000007ff [all sections])

Note: In arguments for command line options, a leading zero indicates an octal value (using digits 0 through 7), and a leading 0x or 0X indicates a hexadecimal value (using digits 0 through 9 and letters a through f). The test is written with the assumption that any other numbers are decimal values (using digits 0 through 9).

2.11.3 Test Output

The `xbg` test reports errors on the ERROR display. The ERROR display contains information about the test when the error occurred. The upper third of the screen displays the path that is being tested. The middle third contains detailed section information. The lower third contains the error message and location. The following types of error messages are displayed:

- Data miscompare
- Status miscompare
- Register miscompare
- Unable to allocate memory
- Unable to obtain configuration information
- Unable to initialize interrupt handler
- Unexpected interrupt
- Timeout waiting for interrupt
- Timeout waiting for device interrupt to clear
- Timeout waiting for device interrupt to set
- Timeout
- Timeout DMA
- Timeout waiting for XBOW port link reset
- Timeout waiting for hub LLP link
- Timeout waiting for warm reset
- Unable to clear hub WSTAT crazy bit
- XBOW port did not recognize Warm Reset
- Bridge port did not recognize Warm Reset

- XBOW port recognized Warm Reset when fence disabled
- Bridge port recognized Warm Reset when fence disabled

The xbg test creates an error display when it logs errors. (The error display is available only when errors have been logged.)

Example:

```
XBG Error Display                                          Error   1 of 1
                                  Error at Pass = 0x0000000000000000
 Path: Node 1   Cpu 3             Error Count   = 0x0000000000000001
       testing I/O Brick 1 on Node 1
       bridge on crossbow port 0xc
       PCI device 1

 Suspect FRU: Rack #, Location #, Bus # Slot #

 -------------------
 Section     = 0x9
 Subsection  = 0x3
 Condition   = 0x6
 Subcondtion = 0x1000

 actual     = 0x00000000aaaaaaaa  address = 0x9600000400284d80  index = 0x760
 expected   = 0x0000000000000760  address = 0x9600000401284d80
 ------------------------------------------------------------------------
 difference = 0x00000000aaaaadca

 Error Message: Data miscompare

 Line 544 in source file tigonDma.c
```

The Bridge Error Interrupt Display screen (`dscr <index> bdgErr 1`) displays the bridge error interrupt counts from each bridge interrupt status register. (One page exists for each I brick.)

Example:

```
XBG - Bridge Error Interrupt Display                        Page   1 of 1

ioBrick 0: I-Brick 003i36         port 9 port 8 port F port E port C port D
-------------------------------------------------------------------------
pmuESizeFaultCnt                  0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
unexpRespCnt                      0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
badXRespPktCnt                    0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
badXReqPktCnt                     0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
respXtalkErrCnt                   0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
reqXtalkErrCnt                    0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
invalidAddrCnt                    0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
unsupportedXOPCnt                 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
xReqFifoOFlowCnt                  0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
lpRecSnErrCnt                     0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
llpRecCBErrCnt                    0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
llpXmitRetryCnt                   0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
llpTCTYCnt                        0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
pciAbortCnt                       0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
pciParityCnt                      0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
pciPERRCnt                        0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
pciMasterToutCnt                  0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
pciRetryCnt                       0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
xRdReqToutCnt                     0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
```

Chapter 3

Diagnostic Utilities

This chapter describes the sMDK-based diagnostic utilities:

- Router dump utility (`rtrdmp`)
- Router register read / write utility (`rtrreg`)
- Display topology utility (`topology`)

3.1 Router Dump Utility (rtrdmp)

The router dump utility (`rtrdmp`) dumps and displays various types of router registers from any router in a system.

You can use command line options to select the register type (port, local block, and local LUT), the router number, and the port number (for port registers) that you want to view.

3.1.1 Utility Sections

Section 0 dumps the router registers.

3.1.2 Running the rtrdmp Utility

Perform the following procedure to run the `rtrdmp` utility:

1. Start sMDK.
2. Enter the following command to run `rtrdmp`:

   `run rtrdmp [`*`test_options`***`]`**

Table 3-1 lists the command line options that are available to customize the utility.

Table 3-1 rtrdmp Utility Command Line Options

Option	Description
`-t <`*register_type*`>`	Specifies the type of register that you want to test: b = barrier L = local block p = port l = local LUT g = global LUT
`-r <`*router_number*`>`	Specifies the router number to dump and display (default: 0)
`-p <`*port_select*`>`	Selects the port to use for the following register types: port, local LUT, and global LUT (default: 1)
`-h | -H | -?`	Displays help information

3.1.3 Utility Output

The following example shows the type of output that `rtrdmp` generates:

```
Dumping the local block registers of Router 0.
STATUS_ REV_ ID data 0x2a95a013017049.
PORT_ RESET data 0x0.
PROTECTION_ CONFIG data 0xff.
GLOBAL_ PORT_ DEF data 0x0.
GLOBAL_ PARM0 data 0x1d7ff3ff01ff7e3.
GLOBAL_ PARM1 data 0xffffe000.
DIAG_ PARMS data 0x0.
DEBUG_ ADDR data 0x3fffc00.
LB_ TO_ L2 data 0x0.
L2_ TO_ LB data 0x0.
SCRATCH_ REG0 data 0xbdce3d6b320c0ef8.
SCRATCH_ REG1 data 0xaaaaaaaaaaaa0001.
SCRATCH_ REG2 data 0xaaaaaaaaaaaa0001.
SCRATCH_ REG3 data 0x1.
SCRATCH_ REG4 data 0x1.
```

3.2 Router Register Read/Write Utility (rtrreg)

The router register read / write utility (`rtrreg`) reads from or writes to all router registers in a system or any single router register that you specify.

3.2.1 Utility Sections

Section 0 reads or writes router registers.

3.2.2 Running the rtrreg Utility

Perform the following procedure to run the `rtrreg` utility:

1. Start sMDK.
2. Enter the following command to run `rtrreg`:

   `run rtrreg [`*`test_options`***`]`**

   Table 3-2 lists the command line options that are available to customize the utility.

Table 3-2 rtrreg Utility Command Line Options

Option	Description
`-f <`*function*`>`	Selects the operation that the utility should perform: r = read w = write (default: read)
`-a <`*register_address*`>`	Selects the router register address to use (default: 0)
`-d <`*register_data*`>`	Specifies the data to use for write operations (default: 0)
`-r <`*router_number*`>`	Selects the router to read or write (default: 0; 0xff = use all routers)
`-h | -H | -?`	Displays help information

3.2.3 Utility Output

The following example shows the type of output that the `rtrreg` utility generates:

```
rtrreg: read of router # 0 register 0x0. Read data 0x2a95a013017049.
```

3.3 Display Topology Utility (topology)

The display topology utility (`topology`) displays the network topology of the system.

3.3.1 Utility Sections

Section 0 displays the network topology.

3.3.2 Running the topology Utility

Perform the following procedure to run the `topology` utility:

1. Start sMDK.
2. Enter the following command to run `topology`:

   `run topology`

3.3.3 Utility Output

The following example shows the type of output that `topology` generates:

```
Node Topology:
------------------------------------------------------------
Node 0 Nasid 0 ---> Router 0 port 2
Node 1 Nasid 2 ---> Router 1 port 2
Router Topology:
------------------------------------------------------------
Router 0 port 1 ---> Router 1 port 1
Router 0 port 2 ---> Node 0 Nasid 0
Router 0 port 6 ---> Router 1 port 6
Router 0 port 7 ---> Router 1 port 7
Router 0 port 8 ---> Router 1 port 8
Router 1 port 1 ---> Router 0 port 1
Router 1 port 2 ---> Node 1 Nasid 2
Router 1 port 6 ---> Router 0 port 6
Router 1 port 7 ---> Router 0 port 7
Router 1 port 8 ---> Router 0 port 8
```

Appendix A

elog_decode Utility

The `elog_decode` utility is a PERL script that decodes sMDK error log information. `elog_decode` uses sMDK node configuration data and error log data to determine the failing DIMM (for correctable errors) or failing DIMM pair (for uncorrectable errors) for memory errors that the `ndir` and `nmem` tests detect.

The `elog_decode.pl` PERL script is included in the *Internal Support Tools 2.3* CD release.

A.1 Running the elog_decode Utility

Perform the following procedure to run the `elog_decode` utility from an IRIX prompt:

1. Create an input file that contains:
   - The configuration information (from the sMDK `cfg` command) for each node that that has error information in its error log.
   - The error log information (from the sMDK `el` command) for each node that has error information in its error log.

   Note: The node configuration information must come before the node error log information. Refer to Section A.1.1, “Sample Input File,” to see an example input file.
2. Enter the following command to run the `elog_decode` PERL script:

   `elog_decode.pl [-t` *types*`] [-m` *dimm_mode*`] [-d` *dimm0_sel*`]` *error_file ... error_fileN*

   Table A-1 describes the command line options that are available.
3. Interpret the output.

Table A-1 elog_decode.pl Command Line Options

Option	Description
`-t <types>`	Specifies which types of sMDK error to decode (PI, MD, NI, II, XB, LB, RT, CPU, or UNKNOWN) Default: PI, MD, NI, II, XB, LB, RT, CPU
`-m <dimm_mode>`	Specifies the default DIMM mode for all nodes if you do not provide sMDK configuration data in the input file (STANDARD or PREMIUM) Default: STANDARD
`-d <dimm0_sel>`	Specifies the dimm0 select to use if you do not provide sMDK configuration data in the input file (0, 1, 2, or 3) Default: 0 Typically, this parameter should be set to 0 and the dimm0 select should be read from the sMDK configuration information that you provide in the input file.

A.1.1 Sample Input File

```
smdk>cfg n 0
Node    0
        module  008c16
        nasid   0
        nic     0x49d1dcb0
        pi      cpu     vcpu    type    Mhz
                A       0       R12K    360
                B       1       R12K    360
                C       2       R12K    360
                D       3       R12K    360
        md      dimm    bank    size    base
                0       0       512MB   0xa800000000000000
                        1       512MB   0xa800000040000000
                -       -       -
                -       -       -
                -       -       -
                dimm0_sel:0 dir_flavor:premium
        ni      router  0       port    2
        ii      -
smdk>cfg n 1
Node    1
        module  008c21
        nasid   1
        nic     0x473d3e1e
        pi      cpu     vcpu    type    Mhz
                A       4       R12K    360
                B       5       R12K    360
                -       -       -       -
                -       -       -       -
        md      dimm    bank    size    base
                0       0       256MB   0xa800000200000000
```

```
                        1       256MB   0xa800000240000000
                -       -       -
                -       -       -
                -       -       -
        dimm0_sel:0 dir_flavor:standard
        ni      router  0       port    4
        ii      -
smdk>el 0
Hardware Error Log - vnode 0 (nasid 0x0) module 008c16: 4 entries
MDCE mem_error:0x001032001706b7e8 count:1
MDCE mem_error:0x00101c0052e7afd0 count:1
MDCE mem_error:0x0010020006428ae8 count:1
MDCE mem_error:0x001098004a88b488 count:1
smdk>el 1
Hardware Error Log - vnode 1 (nasid 0x1) module 008c21: 4 entries
MDCE mem_error:0x001085004c5e4ee8 count:1
MDCE mem_error:0x001085004c424ee8 count:1
MDCE mem_error:0x001085004c244ee8 count:1
MDCE mem_error:0x001085004c1a8ee8 count:1
```

A.2 Interpreting the Output

The output displays the node configuration information and then decodes the error log information to indicate the failing DIMM(s). The following example shows the output from the data provided in the example input file shown in Section A.1.1, “Sample Input File.”

```
------------------------------------------------------------------------
elog_decode.pl.sample.txt
------------------------------------------------------------------------
Node    0
        module  008c16
        nasid   0
        nic     0x49d1dcb0
        pi      cpu     vcpu    type    Mhz
                A       0       R12K    360
                B       1       R12K    360
                C       2       R12K    360
                D       3       R12K    360
        md      dimm    bank    size    base
                0       0       512MB   0xa800000000000000
                        1       512MB   0xa800000040000000
                                                                      I
CFG IN - Vnode:0 dimm0_sel:0 dir_flavor:premium
------------------------------------------------------------------------
Node    1
        module  008c21
        nasid   1
        nic     0x473d3e1e
        pi      cpu     vcpu    type    Mhz
                A       4       R12K    360
                B       5       R12K    360
                -       -       -       -
                -       -       -       -
        md      dimm    bank    size    base
```

```
                0       0       256MB   0xa800000200000000
                        1       256MB   0xa800000240000000
                dimm0_sel:0 dir_flavor:standard
CFG IN - Vnode:1 dimm0_sel:0 dir_flavor:standard
------------------------------------------------------------------------
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Hardware Error Log - vnode 0 (nasid 0x0) module 008c16: 4 entries
Directory Flavor is PREMIUM.
DIMM0_SEL is 0.
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

MDCE mem_error:0x001032001706b7e8 count:1
   READ_CE       = 1  (Valid) - CE on read
   Addr: 0x001706b7e8 (Bank 0), Syn: 0x32 (bit 45) - DIMM 0.

MDCE mem_error:0x00101c0052e7afd0 count:1
   READ_CE       = 1  (Valid) - CE on read
   Addr: 0x0052e7afd0 (Bank 1), Syn: 0x1c (bit 16) - DIMM 1.

MDCE mem_error:0x0010020006428ae8 count:1
   READ_CE       = 1  (Valid) - CE on read
   Addr: 0x0006428ae8 (Bank 0), Syn: 0x02 (bit 65) - DIMM 0.

MDCE mem_error:0x001098004a88b488 count:1
   READ_CE       = 1  (Valid) - CE on read
   Addr: 0x004a88b488 (Bank 1), Syn: 0x98 (bit 39) - DIMM 0.


=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Hardware Error Log - vnode 1 (nasid 0x1) module 008c21: 4 entries
Directory Flavor is STANDARD.
DIMM0_SEL is 0.
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

MDCE mem_error:0x001085004c5e4ee8 count:1
   READ_CE       = 1  (Valid) - CE on read
   Addr: 0x004c5e4ee8 (Bank 1), Syn: 0x85 (bit 23) - DIMM 1.

MDCE mem_error:0x001085004c424ee8 count:1
   READ_CE       = 1  (Valid) - CE on read
   Addr: 0x004c424ee8 (Bank 1), Syn: 0x85 (bit 23) - DIMM 1.

MDCE mem_error:0x001085004c244ee8 count:1
   READ_CE       = 1  (Valid) - CE on read
   Addr: 0x004c244ee8 (Bank 1), Syn: 0x85 (bit 23) - DIMM 1.

MDCE mem_error:0x001085004c1a8ee8 count:1
   READ_CE       = 1  (Valid) - CE on read
   Addr: 0x004c1a8ee8 (Bank 1), Syn: 0x85 (bit 23) - DIMM 1.

------------------------------------------------------------------------
```

Reader Comment Form

Title: sMDK-based Field Diagnostics (SGI™ 3000 Family) **Number: 108-0264-003**

Your feedback on this publication will help us provide better documentation in the future. Please take a moment to answer the few questions below.

For what purpose did you primarily use this document?

_____Troubleshooting
_____Tutorial or introduction
_____Reference information
_____Classroom use
_____Other - please explain

_____

Using a scale from 1 (poor) to 10 (excellent), please rate this document on the following criteria and explain your ratings:

_____Accuracy _____
_____Organization _____
_____Readability _____
_____Physical qualities (binding, printing, page layout) _____
_____Amount of diagrams and photos _____
_____Quality of diagrams and photos _____

Completeness (Check one and explain your answer)

_____Too much information _____ Too little information _____Correct amount

_____
_____
_____

You may write additional comments in the space below. Mail your comments to the address below, fax them to us at +1 715 726 4353, or e-mail them to us at *spt@sgi.com.* When possible, please give specific page and paragraph references. We will respond to your comments in writing within 48 hours.

_____
_____
_____

NAME _____
JOB TITLE _____
E-MAIL ADDRESS _____
SITE/LOCATION _____
TELEPHONE _____
DATE _____

[or attach your business card]

Attn: Service Publications and Training
890 Industrial Boulevard
P.O. Box 4000
Chippewa Falls, WI 54729-0078
USA