# *SPARCserver 1000 POST User's Guide*

**Sun Microsystems Computer Corporation**
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Part No: 801-2916-10
Revision A, May 1993

Please
Recycle

# *Contents*

# *Tables*

# *Preface*

This manual, *SPARCserver™ 1000 POST User's Guide*, describes the Power-On Self-Test (POST) software that is part of the diagnostics that test the SPARCserver 1000 system. POST resides in the boot PROM (programmable read-only memory) on each SPARCserver 1000 system board.

The information in this manual is for manufacturing and test engineers, repair depot and field service personnel, and diagnostics engineers who test the SPARCserver 1000 system. The manual does not describe the system architecture; it assumes you are familiar with such hardware concepts. It provides some background information about the POST software, explains how you can use it, and contains detailed information about the tests that make up the software.

The manual is organized as follows:

**Chapter 1: Overview of POST**
The first chapter introduces you to POST and tells you how to use the software.

**Chapter 2: Test Descriptions**
The second chapter comprehensively describes the tests of the POST software. For each test, there is a test description, an LED pattern, the basic steps executed by the test, and a summary of error messages.

**Appendix A: Sample POST Output**
This appendix shows the results of a sample run of the POST software.

**Appendix B: POST Design Concepts**
This appendix describes the design principles for POST.

**Glossary**
The glossary enhances your understanding of POST by defining the SPARCserver 1000 system terminology.

## *Typographic Changes and Symbols*

The following table describes the font and symbol conventions used in this manual.

*Table P-1* Typographic Conventions

| Typeface or Symbol | Meaning | Example |
|---|---|---|
| `AaBbCc123` | The names of commands, files, and directories; on-screen computer output | Edit your `.login` file.<br>Use `ls -a` to list all files.<br>`system% You have mail.` |
| **`AaBbCc123`** | What you type, contrasted with on-screen computer output; Also, the POST test names will be shown with this typeface | `system% su`<br>`Password:` |
| *AaBbCc123* | Command-line placeholder: replace with a real name or value | To delete a file, type `rm` *filename*. |
| *AaBbCc123* | Book titles, new words or terms, or words to be emphasized | Read Chapter 6 in *User's Guide*.<br>These are called *class* options.<br>You *must* be `root` to do this. |

## *Related Manuals*

For more information on the SPARCserver 1000 system, refer to the documents listed below.

*Table P-2* Related Documentation

| **Category** | **Manual Title** | **Part Number** |
|---|---|---|
| Installation | *Memory Module Installation Guide* | 801-*2030* |
| | *SPARCserver 1000 Installation Manua*l | 800-2893 |
| | *SPARCserver 1000 System Board Manual* | 800-2900 |
| Diagnostics | *OpenBoot Command Reference* | 800-6076 |
| Service | *SPARCserver 1000 Service Manual* | 801-2895 |
| Safety | *Sun SPARCserver 1000 Cabinet and Data Center Expansion Cabinet Regulatory Compliance Manual* | 801-2892 |

# *Overview of POST* 1

The SPARCserver 1000 system's Power-On Self-Test (POST) software automatically tests the hardware resources of the system at power up or reset. POST resides in the boot PROM (programmable read-only memory) on each processor board in a SPARCserver 1000 system. It runs as a stand-alone diagnostic and multiprocessor control program.

## *1.1 Features of POST*

The SPARCserver 1000 POST has a functionality far greater than any Sun™ system POST so far. Based on test results and on customer-defined resource preferences, POST selects the optimal system configuration using as many working resources as possible. It thus tries to provide a reliable machine configuration that can be used by the OpenBoot™ firmware.

POST has two goals:

- Offer the customer a wide platform for applications even if there are hardware failures
- Facilitate field replacement and factory repair.

To these ends, it records error history information about failed or marginally functional components and also provides field replaceable unit (FRU)-level diagnostic information. A detailed log, containing information about which tests have passed and which have failed, is available after each POST run. The FRU-level information is useful for both manufacturing and field service

personnel to determine functional components. The more detailed information, which is recorded from relevant hardware error registers each time there is a failure, helps fault diagnosis in the factory.

The most important new feature of POST, which differentiates it from POST in earlier Sun machines, is *automatic reconfiguration*. When POST finds hardware failures, it tries to reconfigure the system optimally, using as many functional I/O components as possible.

## 1.2 Invoking POST

Before you begin running POST, make sure you have a SPARCserver 1000 system with a serial cable running from the system board in the lowest numbered slot to a TTY terminal or equivalent. You need this set up to see status and error information during POST execution.

You can invoke POST in one of these ways:

- Turn on the power to your SPARCserver 1000 system.
- Press the system reset switch, which is located under the front panel.

Depending on the position of the key switch (which is on the front of the system, under the front panel), POST will execute in *normal mode* (key switch in the normal or SECURE position) or diagnostic mode (key switch in the DIAG position). (Diagnostic mode is hereafter called *diag mode*.)

### *Normal Mode*

Normal mode is used for booting the operating system quickly. In normal mode, the actual operation of POST is transparent to the user. POST initializes the SPARCserver 1000 hardware state and tests all system board components out to the SBus connectors. If errors are detected, POST attempts to recover by modifying the system configuration to exclude the faulty components. When POST completes, it transfers control to the OpenBoot firmware, which then boots the operating system. In normal mode, POST should transfer control to OpenBoot firmware within one minute; it does not display any status messages, but it does display error messages as they occur.

*Diag Mode*

Diag mode is used to test and troubleshoot the SPARCserver 1000 system boards. In diag mode, POST executes a larger set of diagnostics, which provide additional coverage and better isolation of failing components on the system boards. In diag mode, you can communicate with POST. You can control POST using keyboard commands and you can use its test control features. POST is very verbose in this mode. (See Appendix A for a sample POST run.)

## 1.2.1 System and Board LEDs

The SPARCserver 1000 system has three system LEDs and ten board LEDs. Their function is described in this section.

*System LEDs*

The left system LED is green, the center LED is yellow, and the right LED is green

- The left LED (green) is the power indicator. Once the power to the system is turned on, this LED always remains lit (ON).
- When the center LED (yellow) is ON, it indicates that POST is running.
- If the center LED (yellow) remains lit for more than 1 minute in normal mode, and the right LED (green) never lights up, it shows that the machine is unable to boot. (In diag mode this LED remains lit for longer than one minute.)
- If the center and right LEDs light up simultaneously, it shows that the system has booted with failing components, which POST has disabled. (You should be able to boot UNIX or other stand-alone diagnostics.)
- If the right LED (green) is ON and the center LED (yellow) is OFF, it indicates that the system has passed POST without any failures.

*Board LEDs*

The ten board LEDs work as follows:

- Two green board LEDs (A and B) indicate the presence of functional processors on a board at end of a POST run.
- The eight yellow LEDs
  - Output test ID numbers during a POST run
  - Indicate boards with failed parts at end of POST
  - Are always lit on non-processor boards.
- The Boot Master constantly runs a Walking 1s pattern on the yellow LEDs.

## 1.2.2 Error Messages

In both normal and diag modes, error messages are sent to the TTY port and are displayed on any terminal that is connected to that port. The ID of the failing test is also displayed in the eight LEDs on the edge of the system board. (See Chapter 2, “Test Descriptions” for test LED patterns.)

The general format for a POST error message is as follows:

```
bp> TEST STATUS - test_name.subtest_name ID LED
bp> Description of Error
      Address = 0x%X
      Data = 0x%X
```

Table 1-1 explains what each field in the error message means.

*Table 1-1* Error Message Fields

| Field | Description |
|---|---|
| b | System board number. |
| p | Processor (A or B). |
| TEST_STATUS | Status of the test (pass or fail). |
| test_name | Name of the test. |
| subtest_name | Name of the subtest. |
| ID | Unique test and subtest id number. |
| LED | Value (hex) of the LED display for the test. |

Error messages also show a line explaining the failure, and display information from relevant registers.

Samples of error messages displayed by POST are shown below. The first example shows that the test `BW0 Regs` has failed because its subtest (`Timers and Interrupts`) has failed. The test ID is `38.3`, and the LED pattern for the test is `0x26`.

```
2A> TEST FAILED - BW0 Regs.Timers and Interrupts ID 38.3 LED 0x26
2A> Timer Error, expected the Limit Bit to be set
              Address = FFF02010
              Data = 00237800
```

The example below shows the failing test and subtest (`C0 SBI` and `SBI Registers`). The test ID is `56.1` and the LED pattern for this test is `0x36`.

```
0A> TEST FAILED - C0 SBI.SBI Registers ID 56.1 LED 0x38
0A> While testing Component ID register an unexpected trap occurred
              MFSR = 00000936
              MFAR = 02800000
              Trap Type = 9
              CC Error = 00000000.F01E1D58
```

## 1.3 User Interface Commands

In diag mode you can interact with POST in a limited way, using the commands shown in Table 1-2.

*Table 1-2* User Interface Key Commands

| Key | Action |
|---|---|
| a | Toggle Pause CPU A flag.<br>Press this key to stall selftest on CPU A. Press any key to resume selftest.<br>(Affects both CPUs. POST freezes on current system board; other system boards continue.) |
| b | Toggle Pause CPU B flag.<br>Press this key to stall selftest on CPU B. Press any key to resume selftest.<br>(Affects both CPUs. POST freezes on current system board; other system boards continue.) |
| c | Toggle Trace Test Case flag.<br>Set this flag to allow subtests to display trace messages on the console. This is helpful for debugging or troubleshooting the system. |
| e | Toggle Loop On Error flag.<br>Set this flag, and the current test will loop on an error till the flag is reset. If the flag is not set, the current test will try and continue execution once an error occurs. |
| h or ? | Use either key to display this command summary |
| l | Toggle Loop On Subtest flag.<br>Press this key to cause the test sequencer to loop on the current subtest. (Can be an effective scope loop.) |
| m | Go to DEMON menus.<br>Set this flag to interrupt the POST run, call a DEMON, and display the DEMON menu. |
| n | Skip to next subtest.<br>Set this flag to cause the current subtest to exit and return to the sequencer. The next subtest in the list is then dispatched.<br>(Useful for skipping long subtests.) |
| p | Toggle Print All Errors flag.<br>Set this flag to allow POST to display all the errors within each test. Reset the flag if only the first error in each test is to be displayed.<br>(Default is to print one error per subtest.) |

*Table 1-2* User Interface Key Commands *(Continued)*

| Key | Action |
|---|---|
| s | Toggle Stop POST flag.<br>Set this flag to allow POST to stop after it finishes execution and before it transfers control to the OpenBoot firmware. The DEMON menu is displayed. |
| t | Toggle Timestamp flag.<br>Set this flag to allow the sequencer to print a timestamp prior to dispatching each subtest.<br>(Uses TOD clock.) |
| v | Toggle Verbose Print Mode flag.<br>Set this flag to allow POST to display the name of each step as it goes through the system initialization sequence. Reset the flag, and POST displays only the major milestones and the spin loopbar. |
| N | Skip to next test.<br>Set this flag to terminate the current test list and allow the sequencer to fetch the next test list. |
| spacebar | Skip to next test case.<br>Set this flag while a subtest is looping on error and the loop will exit and the subtest will continue by breaking out of the current loop.<br>(Useful when looping on error.) |

## 1.4 DEMON Menu Options

The DEMON options are useful when troubleshooting the system; they are not required in a normal POST run. To use the DEMON menus, type **m** (see Table 1-2) to interrupt POST while it is running in diag mode.

The DEMON main menu is shown below.

```
1A>
DEMON
1A>Select one of the following functions
1A> '0' System Parameters
1A> '1' Read/Write device
1A> '2' Software Reset
1A> '3' NVRAM Management
1A> '4' Error Reporting
1A> '5' Analyze Error Logs
1A> '6' Power Off at Main Breaker
1A> '7' NVRAM SIMM tests
1A> 'r' Return to selftest
1A>

Command ==>
```

To go to another menu or to select a command from this menu, type the number or letter that corresponds to the option (all other keys are ignored).

### *System Parameters Option*

Type **0** at the main menu prompt, to get to the `System Parameters` sub-menu.

This sub-menu has several useful features for debugging and troubleshooting POST problems. You can view system reports, check component IDs, clear error logs, and dump system board registers.

### *Read/Write Device Option*

Type **1** at the main menu prompt, to get to the `Read/Write device` sub-menu.

This sub-menu allows you to read and write using ASIs (address space identifiers). Most of the SPARCserver 1000 ASICs can be accessed in this way. To use this menu, you must have detailed knowledge of how system physical addresses are assigned to the ASICs.

### *Software Reset Option*

The `Software Reset` DEMON option does not have a menu. When you type **2** at the main menu prompt, POST issues a software reset to the BootBus reset register. The system is reset, and POST returns to the DEMON menu.

### *NVRAM Management Option*

Type **3** at the main menu prompt, to get to the `NVRAM Management` sub-menu.

This sub-menu is used to manage the memory SIMM test results in BootBus NVRAM. It allows you to view and erase the data.

### *Error Reporting Option*

Type **4** at the main menu prompt, to get to the `Error Reporting` sub-menu.

This sub-menu is used to print out data saved on the last system watchdog reset. The sub-menu does not allow you to dump data from boards that are not present. If the menu is not used at end of POST, only data from the local board can be dumped. (The “data” is the unformatted contents of all JTAG-scannable ASIC registers.)

### *Analyze Error Logs Option*

The `Analyze Error Logs` DEMON option does not have a menu. When you type **5** at the main menu prompt, POST begins analyzing and displaying the error logs.

POST always logs the last System Watchdog error in BootBus NVRAM. The Analyze Error Logs option analyzes System Watchdog error logs. If there are any error bits set, POST formats and displays the relevant data.

This function is also be invoked by:

- All board masters upon a System Watchdog Reset. In this case, each board can only see its own error log because the BICs are in loopback. (Note that the POST System Master maintains and analyzes the error log for all non-processor boards.)
- The POST System Master after the loopback exit phase of testing (only if there was a recent System Watchdog). In this case, the POST System Master analyzes the error log from each board in the system. If no error bits are set, you only see a banner for that board.

---

**Note –** For troubleshooting purposes only, it is possible to clear the error logs using the DEMON menus. The timestamp for each error log is taken from the TOD on that board. If the operating system has not initialized the TOD, ignore the timestamp and use the `Clear Error Logs` option from the `System Parameters` menu for this task.

---

### *Power Off at Main Breaker Option*

The `Power Off at Main Breaker` DEMON option does not have a menu. When you type **`6`** at the main menu prompt, POST trips the main breaker (this is for manufacturing tests only).

### *NVRAM SIMM Tests Option*

Type **`7`** at the main menu prompt, to get to the `NVRAM SIMM tests` sub-menu.

This option is provided for users to test the NVRAM SIMMs. The operating system uses NVRAM SIMMs to store data. POST never writes to NVRAM SIMMs; it only checks the batteries.

---

**Note –** Never run these tests on a system that is operational, since vital operating system data might be erased.

---

### *Return To Selftest Option*

Type **`r`** to leave the DEMON menus. You are taken back to the point where you interrupted POST execution when you called the DEMON, and the test execution continues.

The following screens are an example of how you can use various options in the DEMON menus. The example begins with the selection of option `0`, `System Parameters` from the DEMON main menu.

```
0A>
DEMON
0A>Select one of the following functions
0A>    ‘0’        System Parameters
0A>    ‘1’        Read/Write device
0A>    ‘2’        Software Reset
0A>    ‘3’        NVRAM Management
0A>    ‘4’        Error Reporting
0A>    ‘5’        Analyze Error Logs
0A>    ‘6’        Power Off at Main Breaker
0A>    ‘7’        NVRAM SIMM tests
0A>    ‘r’        Return to selftest
0A>

Command ==> 0
0A>
System Parameters
0A>Select one of the following functions
0A>    ‘0’         Set POST Level
0A>    ‘1’         Dump Device Table
0A>    ‘2’         Display System
0A>    ‘3’         Dump Board Registers
0A>    ‘4’         Dump Component IDs
0A>    ‘5’         Clear Error Logs
0A>    ‘6’         Display Simms
0A>    ‘7’         Scrub Main Memory
0A>    ‘r’         Return

Command ==> 2
0A>   WARNING Board 2 has failed POST
0A>  (0=failed,1=passed,blank=untested/unavailable)
     (sbus 1=card present,0=card not present,x=failed)
0A>----+----+---+----+---+--+----+---+----+---+----+---+
0A>Slot|cpuA|bw0|cpuB|bw0|bb|ioc0|sbi|mqh0|mem|sbus|xd0|
0A>----+----+---+----+---+--+----+---+----+---+----+---+
0A>  0 | 1  | 1 | 1  | 1 | 1|  1 | 1 | 1  | 64|0001| 1 |
0A>  1 | 1  | 1 |    |   | 1|  1 | 1 | 1  |512|1001| 1 |
0A>  2 | 0  |   |    |   | 0|  1 | 1 | 1  |128|0001| 1 |
0A>----+----+---+----+---+--+----+---+----+---+----+---+
0A>
```

```
0A>Memory Group Status
   (0=failed,1=passed,m=simm missing,c=simm mismatch,blank=unpopulated/unused)
0A>+---+------+------+------+------+
0A>Slot|  g0  |  g1  |  g2  |  g3  |
0A>+---+------+------+------+------+
0A>  0 |   1  |   1  |      |      |
0A>  1 |   1  |   1  |   1  |   1  |
0A>  2 |   1  |   1  |   1  |   1  |
0A>+---+------+------+------+------+
0A>Hit any key to continue :
0A>
System Parameters
0A>Select one of the following functions
0A>    '0'         Set POST Level
0A>    '1'         Dump Device Table
0A>    '2'         Display System
0A>    '3'         Dump Board Registers
0A>    '4'         Dump Component IDs
0A>    '5'         Clear Error Logs
0A>    '6'         Display Simms
0A>    '7'         Scrub Main Memory
0A>    'r'         Return

Command ==> 3
0A>Which Board? 0
0A>Probing E0000000
0A>    BW Register Base E0000000
0A>    Comp Id          10D3907D
0A>    DCSR             0001A000.0000DD00
0A>    DDR              FFFFFFFF.FFFFFFFF
0A>    CTL              00002000
0A>    ITBL
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
```

```
0A>Probing E0800000
0A>    BW Register Base E0800000
0A>    Comp Id          10D3907D
0A>    DCSR             0001A000.8000DD00
0A>    DDR              00000000.00000000
0A>    CTL              00002020
0A>    ITBL
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
0A>                     0000
0A>Probing 01F00000
0A> CC Register Base01F00000
0A> StreamData
0A> Stream Data[0] 01F00000 00000000 00000000
0A> Stream Data[1] 01F00008 00000000 00000000
0A> Stream Data[2] 01F00010 00000000 00000000
0A> Stream Data[3] 01F00018 00000000 00000000
0A> Stream Data[4] 01F00020 00000000 00000000
0A> Stream Data[5] 01F00028 00000000 00000000
0A> Stream Data[6] 01F00030 00000000 00000000
0A> Stream Data[7] 01F00038 00000000 00000000
0A> StreamSrcAddr  01F00100 80000010 06000000
0A> StreamDstAddr  01F00200 80000010 0EAABFC0
0A> RefMissCnt     01F00300 00000000 00000000
0A> IntrptPend     01F00406    0
0A> IntrptMask     01F00506 7FFE
0A> BIST           01F00804 23AA97E6
0A> Control        01F00A04 0000002C
0A>  RC=0, DCB=0, WI=0, PF=1, MC=0, PE=1, CE=1, CS_HC=0
0A> Status         01F00B00 0000000F FFF00002
0A>  SXP=0, SM=0, NCSID=0, NCSPA=FFFF00 NCSPC=0, SPC=0, BC=0, WP=0, RP=1, PP=0
0A> Reset          01F00C04 00000000
0A> Error          01F00E00 00000000 00000000
0A>  ME=0, XP=0, CC=0, VP=0, AE=0, EV=0, CCOP=0, ERR=0, S=0, PA=0 00000000
0A> CompId         01F00F04 02000104
0A>  MID=2, MDEV=1, MREV=0, MVEND=4
```

```
0A>Probing 09F00000
0A> CC Register Base09F00000
0A> StreamData
0A> Stream Data[0] 09F00000 00000000 00000000
0A> Stream Data[1] 09F00008 00000000 00000000
0A> Stream Data[2] 09F00010 00000000 00000000
0A> Stream Data[3] 09F00018 00000000 00000000
0A> Stream Data[4] 09F00020 00000000 00000000
0A> Stream Data[5] 09F00028 00000000 00000000
0A> Stream Data[6] 09F00030 00000000 00000000
0A> Stream Data[7] 09F00038 00000000 00000000
0A> StreamSrcAddr  09F00100 80000000 00000000
0A> StreamDstAddr  09F00200 80000010 1D555FC0
0A> RefMissCnt     09F00300 00000000 00000000
0A> IntrptPend     09F00406    0
0A> IntrptMask     09F00506 FFFE
0A> BIST           09F00804 23AA97E6
0A> Control        09F00A04 0000002C
0A>  RC=0, DCB=0, WI=0, PF=1, MC=0, PE=1, CE=1, CS_HC=0
0A> Status         09F00B00 0000000F FFF00000
0A>  SXP=0, SM=0, NCSID=0, NCSPA=FFFF00 NCSPC=0, SPC=0, BC=0, WP=0, RP=0, PP=0
0A> Reset          09F00C04 00000000
0A> Error          09F00E00 00000000 00000000
0A>  ME=0, XP=0, CC=0, VP=0, AE=0, EV=0, CCOP=0, ERR=0, S=0, PA=0 00000000
0A> CompId         09F00F04 00000104
0A>  MID=0, MDEV=1, MREV=0, MVEND=4
0A>Probing E0100000
0A>MQH Register Base E0100000
0A>    Comp ID          10D8607D
0A>    DCSR             00048700.1000D000
0A>    DDR              FFFFFFFF.FFFFFFFF
0A>    G0ADR            02400009
0A>    G1ADR            02000009
0A>    G2ADR            00000000
0A>    G3ADR            00000000
0A>    G0TYPE           08000800.08000800
0A>    G1TYPE           08000800.08000800
0A>    G2TYPE           FFFFFFFF.FFFFFFFF
0A>    G3TYPE           FFFFFFFF.FFFFFFFF
0A>    MCSR             00000000.00024101
0A>    CEADR            294C4000.00F90800
0A>    CEDR             00000000.00000000
0A>    UEADR            2940C000.00F90800
0A>    UEDR             00000000.00000000
0A>    ECCDCR           00000000.00000000
```

```
0A>    Timing Registers
0A>                    00000000.00000141
0A>                    00000000.00000021
0A>                    00000000.0000022D
0A>                    00000000.000004AF
0A>                    00000000.00000147
0A>                    00000000.00000117
0A>                    00000000.0000021B
0A>                    00000000.0000008A
0A>                    00000000.00000002
0A>                    00000000.00000012
0A>                    00000000.00000090
0A>                    00000000.00000040
0A>                    00000000.00000000
0A>                    00000000.00000000
0A>                    00000000.00000000
0A>                    00000000.00000000
0A>                    00000000.00000000
0A>                    00000000.00000000
0A>                    00000000.00000000
0A>                    00000000.00000000
0A>Probing E0200000
0A>IOC Register Base E0200000
0A>    Comp ID             10ADD07D
0A>    DCSR                0001A000.2000DD00
0A>    DDR                 FFFFFFFF.FFFFFFFF
0A>    CTL                 0001E060
0A>    DBUS Tags SBUS Tags State Bits
0A>    00000000  00000000  00000000
0A>    00000000  00000000  00000000
0A>    00000000  00000000  00000000
0A>    00000000  00000000  00000000
```

```
0A>Probing 02800000
0A>SBI Register Base 02800000
0A>    Comp ID          20ADE07D
0A>    CTL              00020000
0A>    SR               00000000
0A>    S0CR             00000021
0A>    S1CR             00000021
0A>    S2CR             00000021
0A>    S3CR             00000021
0A>    S0SBCR           00000000
0A>    S1SBCR           00000000
0A>    S2SBCR           00000000
0A>    S3SBCR           00000000
0A>    ISR              00000000
0A>    ITIDR            00000000
0A>Hit any key to continue :
0A>
System Parameters
0A>Select one of the following functions
0A>    '0'         Set POST Level
0A>    '1'         Dump Device Table
0A>    '2'         Display System
0A>    '3'         Dump Board Registers
0A>    '4'         Dump Component IDs
0A>    '5'         Clear Error Logs
0A>    '6'         Display Simms
0A>    '7'         Scrub Main Memory
0A>    'r'         Return

Command ==> 4
```

```
0A>  Bus Ring(s)
0A>+------+----------+----------+----------+----------+----------+
0A> Ring  |   bic0   |    bic1  |    bic2  |   bic3   |   barb   |
0A>+------+----------+----------+----------+----------+----------+
0A>  0,1  | 30ADA07D | 30ADA07D | 30ADA07D | 30ADA07D | 20AD907D |
0A>  1,1  | 30ADA07D | 30ADA07D | 30ADA07D | 30ADA07D | 20AD907D |
0A>  2,1  | 30ADA07D | 30ADA07D | 30ADA07D | 30ADA07D | 20AD907D |
0A>+------+----------+----------+----------+----------+----------+
0A>  Processor A Ring(s)
0A>+------+----------+----------+----------+----------+----------+
0A> Ring  |   cpuA   |    mxccA |    bwA   |          |          |
0A>+------+----------+----------+----------+----------+----------+
0A>  0,2  | 0000402F | 0000302F | 10D3907D |
0A>  1,2  | 0000402F | 0000302F | 10D3907D |
0A>  2,2  | 0000402F | 0000302F | 10D3907D |
0A>+------+----------+----------+----------+----------+----------+
0A>  Memory Ring(s)
0A>+------+----------+----------+----------+----------+----------+
0A> Ring  |   mqh    |          |          |          |          |
0A>+------+----------+----------+----------+----------+----------+
0A>  0,3  | 10D8607D |
0A>  1,3  | 20D8607D |
0A>  2,3  | 10D8607D |
0A>+------+----------+----------+----------+----------+----------+
0A>  IO Ring(s)
0A>+------+----------+----------+----------+----------+----------+
0A> Ring  |   sbi    |    ioc   |          |          |          |
0A>+------+----------+----------+----------+----------+----------+
0A>  0,4  | 20ADE07D | 10ADD07D |
0A>  1,4  | 20ADE07D | 10ADD07D |
0A>  2,4  | 20ADE07D | 10ADD07D |
0A>+------+----------+----------+----------+----------+----------+
0A>  Processor B Ring(s)
0A>+------+----------+----------+----------+----------+----------+
0A> Ring  |   cpuB   |    mxccB |    bwB   |          |          |
0A>+------+----------+----------+----------+----------+----------+
0A>  0,5  | 0000402F | 0000302F | 10D3907D |
0A>  1,5  | FFFFFFFF | FFFFFFFF | FFFFFFFF |
0A>  2,5  | FFFFFFFF | FFFFFFFF | FFFFFFFF |
0A>+------+----------+----------+----------+----------+----------+
0A>Hit any key to continue :
0A>
```

```
System Parameters
0A>Select one of the following functions
0A>    ‘0’         Set POST Level
0A>    ‘1’         Dump Device Table
0A>    ‘2’         Display System
0A>    ‘3’         Dump Board Registers
0A>    ‘4’         Dump Component IDs
0A>    ‘5’         Clear Error Logs
0A>    ‘6’         Display Simms
0A>    ‘7’         Scrub Main Memory
0A>    ‘r’         Return

Command ==> 6
0A>Which Board (a = all boards)? 0
0A>
               DRAM       |    NVRAM      |
          Size     Speed  | Size    Speed |  Manufacturer
     ---+--------+--------+-------+-------+-------------
      0 |  4Mbit |  80ns  | 1Mbit |  70ns |   -
      1 | 16Mbit | 100ns  | 4Mbit |  85ns |   MS
      2 | 64Mbit |    -   |   -   |    -  |   TI
       {If NVSIMM, NV=1 and B=1 if battery is good}

0A>Board 0 SIMM Map
0A> ----+---+-----+---+----+---+---+-+--+ ----+---+-----+---+----+---+---+-+--+
0A> SIMM|Grp|Data |ECC|Size|Spd|Mfg|B|NV| SIMM|Grp|Data |ECC|Size|Spd|Mfg|B|NV|
0A> ----+---+-----+---+----+---+---+-+--+ ----+---+-----+---+----+---+---+-+--+
0A> 4100  3  31:16 3:2                    4300  3  63:48 7:6
0A> 3700  2  31:16 3:2                    3900  2  63:48 7:6
0A> 3300  1  31:16 3:2   0   0   2  0  0  3500  1  63:48 7:6   0   0   2  0  0
0A> 2900  0  31:16 3:2   0   0   2  0  0  3100  0  63:48 7:6   0   0   2  0  0
0A> 4000  3  15:00 1:0                    4200  3  47:32 5:4
0A> 3600  2  15:00 1:0                    3800  2  47:32 5:4
0A> 3200  1  15:00 1:0   0   0   2  0  0  3400  1  47:32 5:4   0   0   2  0  0
0A> 2800  0  15:00 1:0   0   0   2  0  0  3000  0  47:32 5:4   0   0   2  0  0
0A>Hit any key to continue :
0A>
```

```
System Parameters
0A>Select one of the following functions
0A>    ‘0’         Set POST Level
0A>    ‘1’         Dump Device Table
0A>    ‘2’         Display System
0A>    ‘3’         Dump Board Registers
0A>    ‘4’         Dump Component IDs
0A>    ‘5’         Clear Error Logs
0A>    ‘6’         Display Simms
0A>    ‘7’         Scrub Main Memory
0A>    ‘r’         Return

Command ==> 7
0A>Hit any key to continue :
0A>
System Parameters
0A>Select one of the following functions
0A>    ‘0’         Set POST Level
0A>    ‘1’         Dump Device Table
0A>    ‘2’         Display System
0A>    ‘3’         Dump Board Registers
0A>    ‘4’         Dump Component IDs
0A>    ‘5’         Clear Error Logs
0A>    ‘6’         Display Simms
0A>    ‘7’         Scrub Main Memory
0A>    ‘r’         Return

Command ==> r
0A>
DEMON
0A>Select one of the following functions
0A>    ‘0’        System Parameters
0A>    ‘1’        Read/Write device
0A>    ‘2’        Software Reset
0A>    ‘3’        NVRAM Management
0A>    ‘4’        Error Reporting
0A>    ‘5’        Analyze Error Logs
0A>    ‘6’        Power Off at Main Breaker
0A>    ‘7’        NVRAM SIMM tests
0A>    ‘r’        Return to selftest
0A>

Command ==> 1
```

```
0A>rwdev> ?
0A>General command format is:

       op_size_space asi address data count increment
       op = r or w or q (read or write or quit)
       size = b,h,w or d
       space = a or v (alternate or virtual space)
       asi = 2 - 0x4c (if alternate space)
       address = device or memory address
       data = write data (if write)
       count = optional range count
       increment = optional address increment (default is data type)

       NOTE: '.'s are ignored and can be used as seperators.
Examples:
rwdev> wba 2f f01e.0000 a5 4 /* writes 4 consecutive bytes into bootbus SRAM */
rwdev> rdv 0 10 /* reads the frist 16 doublewords from cachable space */
rwdev> rda 2 0180.0000 4 100 /* reads the first 4 MXCC tags */
rwdev> rwa 2f fff0.3010 40 0 /* reads the BW tick timer 64 times */

0A>rwdev> q
0A>
DEMON
0A>Select one of the following functions
0A>    '0'        System Parameters
0A>    '1'        Read/Write device
0A>    '2'        Software Reset
0A>    '3'        NVRAM Management
0A>    '4'        Error Reporting
0A>    '5'        Analyze Error Logs
0A>    '6'        Power Off at Main Breaker
0A>    '7'        NVRAM SIMM tests
0A>    'r'        Return to selftest
0A>

Command ==> 2
0A>Initiating Software Reset...
```

```
0A>
DEMON
0A>Select one of the following functions
0A>    ‘0’        System Parameters
0A>    ‘1’        Read/Write device
0A>    ‘2’        Software Reset
0A>    ‘3’        NVRAM Management
0A>    ‘4’        Error Reporting
0A>    ‘5’        Analyze Error Logs
0A>    ‘6’        Power Off at Main Breaker
0A>    ‘7’        NVRAM SIMM tests
0A>    ‘r’        Return to selftest
0A>

Command ==> 3
0A>
Bootbus NVRAM Management
0A>Select one of the following functions
0A>    ‘0’        Print Bad Group List
0A>    ‘1’        Clear Bad Group List
0A>    ‘2’        Print Bad Page List
0A>    ‘3’        Clear Bad Page List
0A>    ‘r’        Return to Main menu
0A>

Command ==> 0
0A>Bad Memory Groups on System

0A>No Bad groups found
0A>Hit any key to continue :
0A>
Bootbus NVRAM Management
0A>Select one of the following functions
0A>    ‘0’        Print Bad Group List
0A>    ‘1’        Clear Bad Group List
0A>    ‘2’        Print Bad Page List
0A>    ‘3’        Clear Bad Page List
0A>    ‘r’        Return to Main menu
0A>

Command ==> 2
0A>Bad Memory Pages in System

0A>No Bad pages found
0A>Hit any key to continue :
```

```
0A>
Bootbus NVRAM Management
0A>Select one of the following functions
0A>    ‘0’        Print Bad Group List
0A>    ‘1’        Clear Bad Group List
0A>    ‘2’        Print Bad Page List
0A>    ‘3’        Clear Bad Page List
0A>    ‘r’        Return to Main menu
0A>

Command ==> r
0A>
DEMON
0A>Select one of the following functions
0A>    ‘0’        System Parameters
0A>    ‘1’        Read/Write device
0A>    ‘2’        Software Reset
0A>    ‘3’        NVRAM Management
0A>    ‘4’        Error Reporting
0A>    ‘5’        Analyze Error Logs
0A>    ‘6’        Power Off at Main Breaker
0A>    ‘7’        NVRAM SIMM tests
0A>    ‘r’        Return to selftest
0A>

Command ==> 4
0A>

Dump Error Reset Status
0A>    ‘0’ - ‘3’ Select Board
0A>    ‘r’         Return

Command ==> 0
0A>Dumping local board 0
0A>Log Date:  Mar 17  0:14:53 GMT 1993
0A>A CC Error Register = 00000000.00000000
0A>B CC Error Register = 00000000.00000000
0A>Processor A
0A>BW0 DCSR = 0001A000.0800DD10 DDR = 00000000.00002000
0A>Processor B
0A>BW0 DCSR = 00FFF0FF.FFFFFFFF DDR = FFFFFFFF.FFFFFFFF
0A>MQH0 DCSR = 00048700.1800D090 DDR = 00000000.00002000
0A>IOC0 DCSR = 0001A000.2800DD90 DDR = 00000000.00002000
0A>SBI Control = 00020000 Status = 00000000
0A>Analyzing BIC data
```

```
0A>XDBus 0 on ***BOARD*** caused parity error
0A>History log bit 12 shows failed BICs : BIC 0, Byte 1;
0A>Hit any key to continue :
0A>

Dump Error Reset Status
0A>    '0' - '3' Select Board
0A>    'r'         Return

Command ==> r
0A>
DEMON
0A>Select one of the following functions
0A>    '0'        System Parameters
0A>    '1'        Read/Write device
0A>    '2'        Software Reset
0A>    '3'        NVRAM Management
0A>    '4'        Error Reporting
0A>    '5'        Analyze Error Logs
0A>    '6'        Power Off at Main Breaker
0A>    '7'        NVRAM SIMM tests
0A>    'r'        Return to selftest
0A>

Command ==> 5
0A>
-------------- Error Log Analysis for Board 0 --------------
0A>*BW0 (CPU A)
0A>  XDBus Parity Error, XDBus Data = 00000000.00002000 XDBus Parity = 00
0A>*MQH0
0A>  Multiple Errors
0A>  XDBus Parity Error, XDBus Data = 00000000.00002000 XDBus Parity = 00
0A>*IOC0
0A>  Multiple Errors
0A>  XDBus Parity Error, XDBus Data = 00000000.00002000 XDBus Parity = 00
0A>XDBus 0 on ***BOARD*** caused parity error
0A>History log bit 12 shows failed BICs : BIC 0, Byte 1;
0A>Log Date:  Mar 17  0:14:53 GMT 1993
0A>CPU A Function at time of error: System Level Software
0A>CPU B Function at time of error: System Level Software
0A>
```

```
-------------- Error Log Analysis for Board 1 --------------
0A>*BW0 (CPU A)
0A>  XDBus Parity Error, XDBus Data = 00000000.00002000 XDBus Parity = 00
0A>*MQH0
0A>  Multiple Errors
0A>  XDBus Parity Error, XDBus Data = 00000000.00002000 XDBus Parity = 00
0A>*IOC0
0A>  Multiple Errors
0A>  XDBus Parity Error, XDBus Data = 00000000.00002000 XDBus Parity = 00
0A>XDBus 0 on ***BACKPLANE*** caused parity error
0A>History log bit 13 shows failed BICs : BIC 0, Byte 1;
0A>Log Date:  Mar 17  0:16:12 GMT 1993
0A>CPU A Function at time of error: System Level Software
0A>
------- Error Log Analysis for Non-Processor Board 2 ------
0A>Parity error on XDBus 0 caused by ***BACKPLANE***
0A>Parity error detected by BIC 2 byte 1
0A>Log Date:  Mar 17  0:14:53 GMT 1993
0A>
-------------- System Memory Failure Analysis ----------------
0A> No Bad groups found
0A>Hit any key to continue :
0A>
DEMON
0A>Select one of the following functions
0A>    '0'        System Parameters
0A>    '1'        Read/Write device
0A>    '2'        Software Reset
0A>    '3'        NVRAM Management
0A>    '4'        Error Reporting
0A>    '5'        Analyze Error Logs
0A>    '6'        Power Off at Main Breaker
0A>    '7'        NVRAM SIMM tests
0A>    'r'        Return to selftest
0A>

Command ==> 7
0A>
```

```
NVRAM SIMM Tests
0A>Select one of the following functions
0A>    ‘0’        Read-Write 6N Test
0A>    ‘1’        Write Test (no verify)
0A>    ‘2’        Read Test (verify single pattern)
0A>    ‘r’        Return to Main menu
0A>

Command ==> 0
0A>NVRAM 6N Read-Write Test
0A>Couldn’t find any NVRAM
0A>Hit any key to continue :
0A>
NVRAM SIMM Tests
0A>Select one of the following functions
0A>    ‘0’        Read-Write 6N Test
0A>    ‘1’        Write Test (no verify)
0A>    ‘2’        Read Test (verify single pattern)
0A>    ‘r’        Return to Main menu
0A>

Command ==> r
0A>
DEMON
0A>Select one of the following functions
0A>    ‘0’        System Parameters
0A>    ‘1’        Read/Write device
0A>    ‘2’        Software Reset
0A>    ‘3’        NVRAM Management
0A>    ‘4’        Error Reporting
0A>    ‘5’        Analyze Error Logs
0A>    ‘6’        Power Off at Main Breaker
0A>    ‘7’        NVRAM SIMM tests
0A>    ‘r’        Return to selftest
0A>

Command ==> r
0A>
```

# *Test Descriptions* 2

This chapter contains the descriptions for the tests that make up the POST software.

---

**Note –** This chapter lists the tests *in the order in which they are executed* when POST is invoked.

---

The general format for each test description is as follows.

Each test has an LED pattern (shown as a set of eight lights) associated with it. The hexadecimal value of this LED pattern is also shown alongside the test name. A brief description of the test follows, along with the test ID number, attributes, and a diagnosis field showing the possible cause of a problem (should a test fail).

For test LED patterns in this manual, white lights ( ❍ ) indicate that the LED is OFF, and black lights ( ● ) indicate that the LED is ON.

The description of the test is followed by descriptions for each of the subtests within a test. Like tests, subtests also show IDs, attributes, diagnoses, and brief descriptions of the functions they perform. In addition, the algorithm (in the form of pseudocode) and the error messages for each subtest are also listed. Subtests do not have hexadecimal values (and their corresponding LED patterns) associated with them.

## 2.1 Early POST Tests

Shortly after power-on and before transferring control to the test sequencers (see Appendix B), POST does a few preliminary tests. These tests are basic checks to verify that the CPU and BootBus are working well enough so that POST can begin more comprehensive testing.

The following tests are very basic; if they fail, you may or may not see error messages (depending on the extent of the failure).

- The first check is to start BIST (built-in self-test) on the MXCC ASIC. The BIST takes one second to execute. If this operation hangs the CPU, you see the value 0x01 in the board LED display.
- The next check is to start BIST on the CPU module. This BIST takes one second to execute. If this operation hangs the CPU, you see the value 0x02 in the board LED display.

---

**Note –** If POST is running in diag mode, it displays the resultant BIST signatures.

---

- POST now does a basic BootBus NVRAM read/write test. POST tests 8 bytes of NVRAM at the NVRAM base address +8. If it detects a failure, POST attempts to print a message on TTYA, then falls into and remains in a write/read scope loop for as long as the failure persists. If this test fails, you see the value 0x04 in the LEDs. This is a non-destructive test; POST saves the 8 bytes prior to the test and later restores them.
- Finally, POST does a basic BootBus SRAM read/write test. POST tests 8 bytes of SRAM at the SRAM base address +8. If a failure is detected, POST attempts to print a message on TTYA, then falls into and remains in a write/read scope loop for as long as the failure persists. If this test fails, you see the value 0x05 in the LEDs. This is a non-destructive test; POST saves the 8 bytes prior to the test and later restores them.

## 2.2 Board Level Testing

The following series of tests verify all functional elements of CPU A, CPU B, the System Board components (Bootbus, BW's, IOC's, SBI, MQH's) and all memory present on this board. These tests are run while all System Boards are in XDBus loopback.

❍❍❍❍ ❍●●❍ **EPROMs** **0x06**

- ID: 6.0
- Attributes: C0 Mandatory Test
- Diagnosis: BootBus

Test the BootBus EPROM.

Subtest: `EPROM path`

- ID: 6.1
- Level: 17
- Attributes: Test Module
  Initialization Module

Fetch previously stored data from the EPROM, and verify that the correct byte, halfword, and word data gets fetched.

- Test byte access.
- Test halfword access.
- Test word access.
- Test doubleword access.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X
   observed = %X
```

Subtest: EPROM checksum

- ID: 6.2
- Level: 17
- Attributes: Test Module
  Initialization Module

Compute a checksum for all addresses of the PROM except the last two bytes of each PROM. Read the last two bytes, and compare the calculated value with the observed one. If an error occurs, a message indicates the failing byte.

*Possible Error Messages*

```
EPROM %d checksum error exp=0x%X obs=0x%X
```

❍❍❍❍ ●❍❍❍ **LEDs** **0x08**

- ID: 8.0
- Attributes: C0 Useful Test
- Diagnosis: BootBus

Test the BootBus LED Register.

Subtest: WALK LED

- ID: 8.1
- Level: 8
- Attributes: Test Module

Walk 1s through the LED register.

- Clear all LEDs.
- Sequentially light up LEDs from right to left or bottom to top.

*Possible Error Messages*

This test does not report any errors.

## ○○○○ ●○○● Serial Ports 0x09

- ID: 9.0
- Attributes: C0 Useful Test
- Diagnosis: BootBus

Test the BootBus Serial Communication Control serial ports.

### Subtest: Port A Register

- ID: 9.1
- Level: 17
- Attributes: Test Module

Perform a Walking 1s test on the UART SCC (Z85C30) write/read register 12.

*Possible Error Messages*

```
Data Compare Error
  address = %X
  expected = %X
  observed = %X
```

### Subtest: Port B Register

- ID: 9.1
- Level: 17
- Attributes: Test Module

Perform a Walking 1s test on the UART SCC (Z85C30) write/read register 12.

*Possible Error Messages*

```
Data Compare Error
  address = %X
  expected = %X
  observed = %X
```

Subtest: `Serial Port A Loopback`

- ID: 9.2
- Level: 17
- Attributes: Test Module

Test Serial Port A using loopback.

- Initialize the UART and enable loopback.
- Send characters 0x20 through 0x7f.
- Check RXRDY and verify that RXDATA = TXDATA.

*Possible Error Messages*

```
pa=%x local loopback error no txready

pa=%X local loopback error no rxready

pa=%X local loopback error exp=0x%X, obs=0x%X
```

Subtest: `Serial Port B Loopback`

- ID: 9.3
- Level: 17
- Attributes: Test Module

Test Serial Port B using loopback.

- Initialize the UART and enable loopback.
- Send characters 0x20 through 0x7f.
- Check RXRDY and verify that RXDATA = TXDATA.

*Possible Error Messages*

```
pa=%x local loopback error no txready

pa=%X local loopback error no rxready

pa=%X local loopback error exp=0x%X, obs=0x%X
```

## ❍❍❍❍ ●❍●● Keybd/Mouse 0x0B

- ID: 11.0
- Attributes: C0 Useful Test
- Diagnosis: BootBus

Test the BootBus Serial Communication Control keyboard and mouse ports.

### Subtest: Keyboard Loopback

- ID: 11.1
- Level: 8
- Attributes: Test Module

Test the keyboard using loopback.

- Initialize the UART and enable loopback.
- Send characters 0x20 through 0x7f.
- Check RXRDY and verify that RXDATA = TXDATA.

*Possible Error Messages*

```
pa=%x local loopback error no txready

pa=%X local loopback error no rxready

pa=%X local loopback error exp=0x%X, obs=0x%X
```

### Subtest: Mouse Loopback

- ID: 11.2
- Level: 8
- Attributes: Test Module

Test the mouse using loopback.

- Initialize the UART and enable loopback.
- Send characters 0x20 through 0x7f.
- Check RXRDY and verify that RXDATA = TXDATA.

*Possible Error Messages*

```
pa=%x local loopback error no txready

pa=%X local loopback error no rxready

pa=%X local loopback error exp=0x%X, obs=0x%X
```

❍❍❍❍ ●●❍❍ **NVRAM/TOD** **0x0C**

- ID: 12.0
- Attributes: C0 Useful Test
- Diagnosis: BootBus

Test the BootBus NVRAM time-of-day clock function to insure that the clock is running.

❍❍❍❍ ●●❍● **Basic CPU** **0x0D**

- ID: 13.0
- Attributes: General Purpose
- Diagnosis: CPUA Module
  CPUB Module

Test the Basic CPU functions.

Subtest: `FPU Register`

- ID: 13.1
- Level: 8
- Attributes: Test Module
  Initialization Module

Test floating-point unit registers.

- Read a data pattern into an FPU register.
- Write FPU register out to memory.
- Compare data in memory to original data.
- Repeat for all FPU registers.
- Repeat for several data patterns.

*Possible Error Messages*

```
Unexpected trap occurred during FPU operation

FPU Double Reg %d, exp %X %X, obs %X %X, reg, exp, obs

Single Precision, exp = %X, obs = %X
```

Subtest: FPU Functional

- ID: 13.1
- Level: 8
- Attributes: Test Module
  Initialization Module

Test the functionality of the floating-point unit.

- Perform the following operation, using single precision:
  (((3 * 4 * 5) - 2 + 2) / 4) / 5 .
- Verify that the result is 3.0.
- Repeat, using double precision.

*Possible Error Messages*

```
Unexpected trap occurred during FPU operation

FPU Double Reg %d, exp %X %X, obs %X %X, reg, exp, obs

Single Precision, exp = %X, obs = %X
```

Subtest: MMU TLB

- ID: 13.1
- Level: 17
- Attributes: Test Module
  Subtest Disabled
  Initialization Module

Write-read-verify all TLB entries using Walking 1s pattern.

*Possible Error Messages*

```
unexptd_tlb_msg, entry, sel, exp, obs)
```

Subtest: `Instruction Cache Tags`

- ID: 13.1
- Level: 17
- Attributes: Test Module
  Initialization Module

Test that the Icache can be flash-cleared and that the tags can be addressed uniquely. Also check the tag array for data reliability.

- Write all the state bits and an incrementing pattern in the Paddr field.
- Flash clear the lock bits; check thatthey get cleared and that Paddr is not changed.
- Flash clear the valid and mru bits; check that all valid and mru bits are clear and that lock bits and Paddr field are unchanged.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X.%X
   observed = %X.%X
```

Subtest: `Instruction Cache Ram`

- ID: 13.2
- Level: 17
- Attributes: Test Module
  Initialization Module

Test the instruction cache RAM.

Address Ascending:

- Write each address with its address as the data.
- Read and verify each address.

Address Descending:

- Write each address with its address as the data.
- Read and verify each address.

Cell Disturbance:

- Write the entire cache with a checkerboard bit pattern.
- Read and verify each address.
- Reverse the checkerboard pattern and repeat.

Data Reliability:

- Write the cache with standard test patterns.
- Read and verify the data.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X.%X
   observed = %X.%X
```

## Subtest: Data Cache Tags

- ID: 13.3
- Level: 8
- Attributes: Test Module
  Initialization Module

Test the CPU's data cache tags for address uniqueness and data reliability.

Address Ascending:

- Write each tag with its address as the data.
- Read and verify each address.

Address Descending:

- Write each address with its address as the data.
- Read and verify each address.

Cell Disturbance:

- Write the entire array with a checkerboard bit pattern.
- Read and verify each address.
- Reverse the checkerboard pattern and repeat.

Data Reliability:

- Write the tag array with standard test patterns.
- Read and verify the data.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X.%X
   observed = %X.%X
```

Subtest: **Data Cache Ram**

- ID: 13.4
- Level: 8
- Attributes: Test Module
  Initialization Module

Test address uniqueness and data reliability of the CPU internal data cache RAMs.

Address Ascending:

- Write each address with its address as the data.
- Read and verify each address.

Address Descending:

- Write each address with its address as the data.
- Read and verify each address.

Cell Disturbance:

- Write the entire cache with a checkerboard bit pattern.
- Read and verify each address.
- Reverse the checkerboard pattern and repeat.

Data Reliability:

- Write the cache with standard test patterns.
- Read and verify the data.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X.%X
   observed = %X.%X
```

Subtest: `Store Buffer Tags`

- ID: 13.5
- Level: 8
- Attributes: Test Module
  Initialization Module

Verify Store Buffer tags for address uniqueness and data reliability. (The test is run with the Store Buffer off.)

Store Buffer Addressing test:

- Write address ascending.
- Read and verify.
- Write address descending.
- Read and verify.

Store Buffer RAM data reliability:

- Write all tags with test pattern.
- Read each tag and verify data.
- Loop for all patterns.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X.%X
   observed = %X.%X
```

Subtest: `Store Buffer RAM`

- ID: 13.6
- Level: 8
- Attributes: Test Module
  Initialization Module

Verify Store Buffer SRAMs for address uniqueness and data reliability. (This test is run with the Store Buffer off.)

Store Buffer Addressing test:

- Write address ascending.
- Read and verify.
- Write address descending.
- Read and verify.

Store Buffer RAM data reliability:

- Write entire RAM with test pattern.
- Read RAM and verify the data.
- Loop for all patterns.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X.%X
   observed = %X.%X
```

Subtest: `Store Buffer Functional`

- ID: 13.9
- Level: 8
- Attributes: Test Module
  Initialization Module

Test the Store Buffer functions that can be tested while in boot mode with the Ecache turned off. (Implies only non-cacheable space.)

This test currently issues stores to EPROM address space. The actual results are obtained from the Store Buffer.

Stress test using non-cacheable stores:

- Make sure Store Buffer is off.
- Zero Store Buffer control and all tags.
- Clear the tags.
  (The Dptr and Fptr are set to 0.)
- Turn on Store Buffer.
- Issue 8 stores that should use each entry in the Store Buffer (0..7).
- Turn off Store Buffer.
- Read the tags and data, and verify.

Using EPROM address space, float 1 through the address field of each tag:

- Issue the store.
- Read the tag and data.
- Float address bit, and loop for all address bits.

Use a bus parity error to force a Store Buffer error, and check the Store Buffer tags, data, and control for proper state:

- Establish the trap handler.
  (The first store will go to the Store Buffer. The load will cause the Store Buffer to flush this store with odd parity. The MXCC should complain about the bad parity. The CPU should take a data store error trap.)
- Check that the correct trap (data store error) occurs.
  (Dptr must point to the entry that incurred the data store error.)
- Check the Store Buffer control and tags.
- Zero Store Buffer control and all tags .

*Possible Error Messages*

```
Store Buffer tag error
 entry = %x
 expected = %X.%X
 observed = %X.%X

Store Buffer data error
 entry = %x
 expected = %X.%X
 observed = %X.%X

Store Buffer control error
 expected = %X
 observed = %X

Data store error trap did not occur
```

Subtest: `MXCC Registers`

- ID: 13.10
- Level: 8
- Attributes: Test Module
  Initialization Module

Test the read and write accessibility of the MXCC ASIC registers, using all access sizes allowed. The addresses of the MXCC registers are in ECSR space and Control Space (ASI 2).

To prevent XDBus transactions, this test uses Control Space Access only. If any access causes a data access exception or unexpected interrupt, the test aborts with a `FAIL` status.

The Stream Source and Destination Address registers are not tested here (because they generate XDBus transactions to the BWs), but they are tested after the rest of the system is initialized and tested. Also, the Interrupt registers must be tested later, because they generate XDBus transactions to the BW.

A write to the Status register has bad side effects (it can cause the CPU to hang). So it is not tested except to read it and insure that it does not cause a trap.

The Reset register is tested by clearing it and verifying that it clears. A Software Reset can be tested from the DEMON Menu.

The Error register is read only. The test clears all errors then checks to insure that all error bits are cleared.

*Possible Error Messages*

```
%s register value indicates XBus may be broken
   expected = %X.%X
   observed = %X.%X

While testing %s register an unexpected trap occurred
   MFSR = %X
   MFAR = %X
   Trap Type = %2x
   CC Error = %X.%X
```

```
Unexpected Component ID value
   address = %X
   expected = %X or %X
   observed = %X

%s register failed to return correct data
   address = %X
   expected = %X
   observed = %X

%s register failed to return correct data
   expected = %X.%X
   observed = %X.%X

Floating a bit through %s register failed
   expected = %X
   observed = %X

Floating a bit through %s register failed
   expected = %X.%X
   observed = %X.%X
```

Subtest: Init MXCC Regs

- ID: 13.11
- Level: 8
- Attributes: Error is Fatal

Clear the MXCC error register, clear all pending interrupts, clear the reference / miss count register, and enable Level 15 interrupts.

*Possible Error Messages*

This module does not check or report errors.

## ❍❍❍❍ ●●●❍ Ecache 0x0E

- ID: 14.0
- Attributes: C0 Useful Test
- Diagnosis: CPUA Module
  CPUB Module

Test the external cache system.

### Subtest: Setting Cache Size

- ID: 14.1
- Level: 1
- Attributes: Error is Fatal

Set or clear the bits for the selected mode, half or full cache.

*Possible Error Messages*

This module does not check or report errors.

### Subtest: Ecache Tags

- ID: 14.2
- Level: 8
- Attributes: Test Module
  Initialization Module

Test address uniqueness and data reliability of the external cache (MXCC) tags.

- Do a write pass in ascending order.
- Do a read pass in ascending order.
- Do a write pass in descending order.
- Do a read pass in descending order.
- Do the data reliability test case.
- Loop through all the patterns.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X.%X
   observed = %X.%X
```

Subtest: Ecache SRAM

- ID: 14.3
- Level: 8
- Attributes: Test Module
  Initialization Module

Test access size, addressing, and SRAM data reliability.
(This test is run with the external cache disabled.)

Test CC SRAM access:

- Write a pattern into an SRAM double word location.
- Read it back a byte at a time and verify.
- Read it back a half at a time and verify.
- Read it back a word at a time and verify.
- Write every byte in the cache line.
- Read and verify.
- Write every half in the cache line.
- Read and verify.
- Write every word in the cache line.
- Read and verify.

Test CC SRAM addressing:

- Write pass address up.
- Read pass address up.
- Write pass address down.
- Read pass address down.

Test CC SRAM data reliability:

- Only do the long test if POST LEVEL is high.
- Loop through all the patterns.
  (Checking for stuck ats.)
- Test pattern and ~pattern.
- Turn on CPU module Bus parity and watch for traps.
- Set up trap to handle data access exception (parity error).
- Loop through all patterns.
- Check for parity error.
- Check for miscompares.

Do a short test for booting (DIAG Switch OFF):

- Turn on CPU module Bus parity and watch for traps.
- Set up g5 and g6 to expect data access exception.

- Write pass; write alternate patterns.
- Loop through the cache, comparing alternate patterns.
- Check for parity error.
- Check for miscompares.
- Repeat test with mixed parity patterns.

*Possible Error Messages*

```
Viking Parity Error
   address = %X
   expected = %X.%X
   observed = %X.%X

Data XOR = %X.%X
   Part = U%d
```

Subtest: `Ecache Enable`

- ID: 14.5
- Level: 1
- Attributes: Error is Fatal

Enable the external cache.
(From this point on, the cache remains enabled.)

*Possible Error Messages*

This module does not check or report errors.

Subtest: `Clear CC SRAM`

- ID: 14.4
- Level: 1
- Attributes: Error is Fatal

Clear the external cache SRAM.
(This also insures that good parity is established for the SRAM.)

*Possible Error Messages*

This module does not check or report errors.

❍❍❍● ❍❍●❍ **BW0 Regs** **0x12**

- ID: 18.0
- Attributes: C0 Mandatory Test
- Diagnosis: BWA0
  BWB0

Test the registers and tags on Bus Watcher 0.

Subtest: C_O BW

- ID: 18.1
- Level: 1
- Attributes: Initialization Module
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: BW Registers

- ID: 18.2
- Level: 8
- Attributes: Test Module
  Initialization Module

Test the read and write accessibility of all BW ASIC registers, using all access sizes allowed. The addresses of the BW registers are in CSR space and Local space; the test uses both address spaces. If any access causes a data access exception or unexpected interrupt, the test aborts with a FAIL status.

*Possible Error Messages*

```
While testing %s register an unexpected trap occurred
   MFSR = %X
   MFAR = %X
   Trap Type = %2x
   CC Error = %X.%X
```

```
Unexpected Component ID value
   address = %X
   expected = %X or %X
   observed = %X

%s register failed to return correct data
   address = %X
   expected = %X
   observed = %X

%s register failed to return correct data
   expected = %X.%X
   observed = %X.%X

Floating a bit through %s register failed
   expected = %X
   observed = %X

Floating a bit through %s register failed
   expected = %X.%X
   observed = %X.%X
```

Subtest: Timers and Interrupts

- ID: 18.3
- Level: 8
- Attributes: Test Module
  Initialization Module

Test timer in free running mode, no interrupts.

- Make sure the prescaler is initialized for 1 microsecond.
- Configure Ptimer for non-UT mode.
- First do the Ptimer, then do the Ttimer.
- Clear all interrupt registers.
- Set timer to run free.
- Stall for a few milliseconds.
- Make sure the counter did some counting.
- Check the interrupt table (should be 0).

Test both P and T timer in limit mode with interrupts.

- First do the Ptimer, then do the Ttimer.
- Set timer to run free.
- Clear all interrupt registers.
- Set limit to 100 and see if interrupt is generated.
- Stall for a few milliseconds.
- Check the limit bits.
- Check the interrupt table.
- Check the interrupt pending.
- Setup for tick timer.
- Set timer to free running mode - turn off interrupts.

Test User Timer Mode.

- Configure the Ptimer for User Timer mode.
- Make sure it counts.

Test alarm clock interrupts.

- First do the Ptimer, then do the Ttimer.
- Set timer to run free.
- Clear all interrupt registers.
- Set ND limit to 100 and see if interrupt is generated.
- Stall for a few milliseconds.
- Check the limit bits.
- Check the interrupt table.
- Check the interupt pending.
- Setup for tick timer.
- Set timer to free running mode - turn off interrupts.

Clean up everything.

- Configure Ptimer for non-UT mode.
- Set timer to run free.
- Clean up interrupt registers.

*Possible Error Messages*

```
Timer Free Running Mode Error
  Address = %X
  start count = %X
  end count = %X
```

```
Timer Error, expected the Limit Bit to be set
  Address = %X
  Data = %X

Interrupt table has incorrect value
  expected = %4X
  observed = %4X

Interrupt Pending Register has incorrect value
  level = %d
  expected = %4X
  observed = %4X

User Timer mode not counting
  start count = %X.%X
  end count = %X.%X
```

Subtest: BW Tag RAM 6N

- ID: 18.1
- Level: 8
- Attributes: Test Module
  Initialization Module

Test the Bus Watcher Tag RAMs with a 6N algorithm.
The test is meant to be called by the POST sequencer. The code is meant to be run in diag mode only. The test is executed from the CPU's Icache.

- Determine BW tag size and mode configured.
- Execute the 6N test on the tags.
- First pass: write in ascending order.
- Second pass: read-then-write in ascending order.
- Third pass: read-compare in descending order.
- Loop for all patterns.

*Possible Error Messages*

```
Data Compare Error
  address = %X
  expected = %X
  observed = %X
```

## ❍❍❍● ●●●❍ C0 MQH 0x1E

- ID: 30.0
- Attributes: General Purpose
- Diagnosis: MQH0

Set Configuration 0 and test the MQH.

### Subtest: C_0 BW,MQH

- ID: 30.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

### Subtest: MQH Registers

- ID: 30.2
- Level: 8
- Attributes: Test Module
  Error Terminates Sequencer

Test the read and write accessibility of the MQH ASIC registers, using all access sizes allowed. The addresses of the MQH registers are in CSR space. If any access causes a data access exception or unexpected interrupt, the test aborts with a FAIL status.

The ECC Error registers are read only. Testing is limited to insuring that register access does not cause a trap and that all error bits are cleared.

The Group Type registers are read only; they are not tested.

To prevent losing refresh, the Refresh Enable bit in the MCSR is never reset.

When appropriate, the test restores the original value it found in the register.

*Possible Error Messages*

```
While testing %s register an unexpected trap occurred
   MFSR = %X
   MFAR = %X
   Trap Type = %2x
   CC Error = %X.%X

Unexpected Component ID value
   address = %X
   expected = %X or %X
   observed = %X

%s register failed to return correct data
   address = %X
   expected = %X
   observed = %X

%s register failed to return correct data
   expected = %X.%X
   observed = %X.%X

Floating a bit through %s register failed
   expected = %X
   observed = %X

Floating a bit through %s register failed
   expected = %X.%X
   observed = %X.%X
```

Subtest: MQH Initialization

- ID: 30.3
- Level: 1
- Attributes: Error is Fatal

Set up the MQH timing registers and control register. Timing values loaded depend on the types of SIMMs present. Turn on Refresh Enable, set Refresh Count, Request Delay.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `Enable ECC`

- ID: 30.4
- Level: 1
- Attributes: Error is Fatal

Enable ECC checking on the MQH.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `Memory`

- ID: 30.5
- Level: 8
- Attributes: Test Module

Test all memory on this MQH. If a group with memory is not found, return `FAIL`. The purpose of the test is to test enough memory to allow the consistency tests to run. The memory test functions are loaded into the Icache for speed.

Short memory test algorithm:

- Clear number of memory faults in current test.
- Load the CC Stream Data register with alternate patterns.
- Now loop through memory, writing 64 bytes at a time.
- Check the memory.
- Set up and load the alternate pattern.
- Loop through memory, writing 64 bytes at a time.
- Check the memory.
  (The permanent ECC handler handles memory errors.)
- If the faults exceed 2, return fail.

Long memory test algorithm:

- Clear number of memory faults in current test.
- Loop through a set of long patterns.
- Load stream data register with pattern.
- Fill memory with pattern.
- Load stream data register with ~pattern.
- Read, then write ~pattern.
- Read ~pattern, write pattern, read.

*Possible Error Messages*

```
Memory Compare Failure
   Addr %X
   Expected %X.%X
   Observed %X.%X
```

Subtest: `Config Memory Available`

- ID: 30.6
- Level: 1
- Attributes: Error is Fatal

Count up the amount of memofy available in the current configuration.

*Possible Error Messages*

This module does not check or report errors.

❍❍●❍ ❍❍❍❍ **C0 IOC** **0x20**

- ID: 32.0
- Attributes: General Purpose
- Diagnosis: IOC0

Set Configuration 0 and test the IOC.

Subtest: `C_0 BW,IOC`

- ID: 32.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: IOC Registers

- ID: 32.2
- Level: 8
- Attributes: Test Module
  Error Terminates Sequencer

Test the read and write accessibility of all IOC ASIC registers, using all access sizes allowed. The addresses of the IOC registers are in CSR space. If any access causes a data access exception or unexpected interrupt, the test aborts with a FAIL status.

*Possible Error Messages*

```
While testing %s register an unexpected trap occurred
   MFSR = %X
   MFAR = %X
   Trap Type = %2x
   CC Error = %X.%X

Unexpected Component ID value
   address = %X
   expected = %X or %X
   observed = %X

%s register failed to return correct data
   address = %X
   expected = %X
   observed = %X

%s register failed to return correct data
   expected = %X.%X
   observed = %X.%X

Floating a bit through %s register failed
   expected = %X
   observed = %X

Floating a bit through %s register failed
   expected = %X.%X
   observed = %X.%X
```

Subtest: `IOC XDBus Tags`

- ID: 32.3
- Level: 8
- Attributes: Test Module

Read, write, and verify the IOC's XDBus tags.

*Possible Error Messages*

```
Data Compare Error
  address = %X
  expected = %X
  observed = %X
```

Subtest: `IOC Sbus Tags`

- ID: 32.4
- Level: 8
- Attributes: Test Module

Read, write, and verify the IOC's SBus tags.

*Possible Error Messages*

```
Data Compare Error
  address = %X
  expected = %X
  observed = %X
```

Subtest: `IOC Cache RAM`

- ID: 32.5
- Level: 8
- Attributes: Test Module

Read, write, and verify the IO Cache RAM.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X
   observed = %X
```

## ❍❍●❍ ❍❍❍● C0 SBI 0x21

- ID: 33.0
- Attributes: General Purpose
- Diagnosis: SBI

Test the SBI ASIC.

### Subtest: SBI Initialization

- ID: 33.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

This function initializes all SBI Registers to the default values.

*Possible Error Messages*

This module does not check or report errors.

### Subtest: SBI Registers

- ID: 33.2
- Level: 8
- Attributes: Test Module
  Error Terminates Sequencer

Test the read and write accessibility of all SBI ASIC registers, using all access sizes allowed. The addresses of the SBI registers are in ECSR space. If any access causes a data access exception or unexpected interrupt, the test aborts with a `FAIL` status.

The SBI Interrupt registers, with the exception of the Interrupt Target ID, are not tested here, but are tested in the SBus Interrupts test.

*Possible Error Messages*

```
While testing %s register an unexpected trap occurred
   MFSR = %X
   MFAR = %X
   Trap Type = %2x
   CC Error = %X.%X

Unexpected Component ID value
   address = %X
   expected = %X or %X
   observed = %X

%s register failed to return correct data
   address = %X
   expected = %X
   observed = %X

%s register failed to return correct data
   expected = %X.%X
   observed = %X.%X

Floating a bit through %s register failed
   expected = %X
   observed = %X

Floating a bit through %s register failed
   expected = %X.%X
   observed = %X.%X

%s fields SEGA, C, S updated when WSA = 0
   expected = %X
   observed = %X
```

Subtest: SBI Initialization

- ID: 33.3
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Initialize all SBI registers to the default values.

*Possible Error Messages*

This module does not check or report errors.

Subtest: SBus Interrupts

- ID: 33.4
- Level: 8
- Attributes: Test Module

Test all levels of SBus interrupts for all SBus slots. Verify that the correct interrupt state is recorded in the BW, MXCC and SBI ASICs. Insure that the correct SPARC interrrupt level is delivered to the CPU. This test is executed on each board by the CPU on that board. The system uses this test for SBus slots on non-processor boards.

- Mask all interrupts except Level 15.
- Clear all existing interrupt states.
- Establish this board's BW as the target for this SBus's interrupts.
- Verify the CC transaction to the SBI interrupt target register.
- Loop for all Levels (SBus has levels 1 through 7).
- Loop for all slots (each board has 4 SBus slots).
- Use diagnostic register to generate SBus interrupt.
- Issue a `TAKE` and check the state register.
- Issue a `GIVE` and check the state register.
- Check BW interrupt table.
- Check CC interrupt pending.
- Unmask the interrupt and insure the CPU gets the correct interrupt.
- Do necessary housekeeping.
- Clean up before exiting.

*Possible Error Messages*

```
Failed to establish new targer id, Board %x
  Address = %X
  expected = %2X
  observed = %2X

Incorrect Interrupt State, Board %x
  Address = %X
  expected = %2X
  observed = %2X
```

```
Incorrect CC Interrupt Pending, Board %x Slot %x
  Address = %X
  expected = %4X
  observed = %4X

Incorrect BW Interrupt Table, Board %x Slot %x
  Address = %X
  expected = %4X
  observed = %4X

SBus Interrupt not delivered to CPU, Board %x
  Slot = %x
  Level = %x
  Trap Type = %2x
```

❍❍●❍ ❍❍●❍ **C0 SBUS Cards** **0x22**

- ID: 34.0
- Attributes: General Purpose
- Diagnosis: None

Probe all sbus slots to see if a sbus card responds.

Subtest: SBI Initialization

- ID: 34.3
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

This function initializes all SBI Registers to the default values.

*Possible Error Messages*

This module does not check or report errors.

Subtest: Checking for SBUS cards

- ID: 34.5
- Level: 1
- Attributes: Error is Fatal

Check each slot to see if a card responds.

*Possible Error Messages*

This module does not check or report errors.

## ❍❍●❍ ❍❍●● C0 XDBus Timing 0x23

- ID: 35.0
- Attributes: General Purpose
- Diagnosis: BootBus

Using the TOD, compute the system crystal frequency.

### Subtest: C_0 BW

- ID: 35.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

### Subtest: Compute XDBus Frequency

- ID: 35.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Using the TOD, compute the System crystal frequency to the nearest Mhz.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `TOD Delay`

- ID: 35.1
- Level: 17
- Attributes: Error is Fatal

Use the TOD for a timed delay allowing sbus devices to perform self initialization.

*Possible Error Messages*

This module does not check or report errors.

❍❍●❍ ❍●❍❍ **C0 XPT** **0x24**

- ID: 36.0
- Attributes: General Purpose
- Diagnosis: SBI

Set Configuration 0 and test the IO external page tables.

Subtest: `C_0 BW,IOC`

- ID: 36.1
- Level: 17
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `XPT Read Write`

- ID: 36.2
- Level: 17
- Attributes: Test Module

Test the functionality of the external page tables.

- For the first part of the test, force parity.
- Set up the SBI control register.
- Do a 6N test.
- Load a new pattern.
- Test with the next pattern.
- First pass: write in ascending order.
- Second pass: read-then-write in ascending order.
- Third pass: read-compare in descending order.
- Move the pattern pointer along.
- Loop for all patterns.
- Restore original SBI control register.
- Test with some even and odd parity patterns.
- Clear the XPT so that there is good parity.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X
   observed = %X
```

❍❍●❍ ❍●❍● **C0 BW-MQH Consistency** **0x25**

- ID: 37.0
- Attributes: General Purpose
- Diagnosis: MQH0

Set Configuration 0 and test the consistency between BW and main memory.

Subtest: C_0 BW,MQH

- ID: 37.1
- Level: 17
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: BW MQH Cache Consistency

- ID: 37.1
- Level: 17
- Attributes: Test Module

Test consistency between the external cache subsystem and the memory (CC, BW, MEM).

The IOC chip is frozen for this test.

- Assign a test address for each board.
  (Each board has a cache line in virtual page 0.)
- Set the line to valid and owned by this cache.
- Fill the line with a test pattern.
  (Each subblock will have a different pattern.)
- Read and verify the first doubleword of each subblock.
- Read hit (external cache will return the data to CPU).
- Victimize this line, check the new tag.
- Check main memory to verify that the Ecache flush occurred.
- Verify that this new line will not have owner set.
- Read miss (fill from main memory).
- Check data and tags.

*Possible Error Messages*

**Note –** This is a common group of error messages used by all the POST Consistency tests.

```
Block compare failed
   load address = %X data = %X.%X
   store address = %X data = %X.%X

Block check error
   address = %X
   expected = %X
   observed = %X

Stream ready bit timed out

Check tags failed
   cctag=%X.%X exp. state = %x
   bwtag=%X exp. state %x

Check Dcache tags failed
   address = %X
   expected valid bit = %x
   observed valid = %x

DCache tag has inconsistent state
   ptag = %X.%X

Read hit Ecache data error
   addr = %X
   expected = %X.%X
   observed = %X.%X

Read miss Ecache data error
   address = %X
   expected = %X.%X
   observed = %X.%X

Victimize error for address = %X

Write invalidate failed for address = %X
```

```
IO loopback read data error
  address = %X
  expected = %X.%X
  observed = %X.%X

IO loopback read miss failed, address = %X

IO loopback read hit failed, address = %X

IO loopback write miss failed, address = %X

IO loopback write hit failed, address = %X

IO cache shared owner failed, address = %X

IOC check tags line = %dex = %X %Xob = %X %X

IOC check data ex = %X ob = %X word = %d

IO cache flush data error
  address = %X
  expected = %X.%X
  observed = %X.%X

CPU read data
  address = %X
  expected = %X.%X
  observed = %X.%X

Check RefMiss count
  expected CRC=%X CMC=%X
  observed CRC=%X CMC=%X
```

## ❍❍●❍ ❍●●❍ C0 IOC-MQH Consistency 0x26

- ID: 38.0
- Attributes: General Purpose
- Diagnosis: IOC0
  MQH0

Set Configuration 0 and test consistency between IOC and main memory.

Subtest: C_0 BW,IOC,MQH

- ID: 38.1
- Level: 17
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: SBus Loopback

- ID: 38.2
- Level: 17
- Attributes: Test Module

Use the SBus loopback feature of the SBI ASIC to test and exercise the paths between the CPU and SBus and the SBus and main memory. IO cache and CPU cache transactions are also verified. This test is not run unless main memory is accessible.

Stream Mode and XPT Bypass:

- Insure tags are cleared.
- Set slot configuration registers.
- Clear all error bits.
- SEGA =0, Slot Reset, Stream Mode, 64 Byte Burst, Bypass XPT.
- Clear some memory.
- SBus loopback: write data; data = offset.
  (Data should have looped to memory.)
- See if any SBI errors were set.
- Verify the memory data.
- Negate the data in memory.
- SBus loopback: read the memory a word at a time, and check for errors.
  (Data should have looped from memory.)
- Verify the read data.

Consistent Mode and XPT Bypass use the same procedure as above, except Consistent Mode is selected and the IOC tags tags are verified for the correct state.

- Insure tags are cleared.
- Set slot configuration registers.
- Clear all error bits.
- SEGA =0, Slot Reset, Stream Mode, 64 Byte Burst, Bypass XPT.
- Clear some memory.
- SBus loopback: write data; data = offset.
- Check IOC tags for correct state.
  (Data should have looped to memory.)
- See if any SBI errors were set.
- Verify the memory data.
- Negate the data in memory.
- SBus loopback: read the memory a word at a time, and check for errors.
- Check IOC tags for correct state.
  (Data should have looped from memory.)

*Possible Error Messages*

```
SBI Detected Errors, Board %x Slot %x
  Status = %X, Slot Config = %X

Bypass Stream Write Failed, Slot %x
  expected = %X
  observed = %X

Bypass Stream Read Failed, Slot %x
  expected = %X
  observed = %X

Bypass Consistent Write Failed, Slot %x
  expected = %X
  observed = %X

Bypass Consistent Read Failed, Slot %x
  expected = %X
  observed = %X

SBI Flush Write Buffers timed out, Slot %x
  Stream Buffer Control Register = %X
```

```
WARNING Test skipped, no memory in configuration [%X]
```

Subtest: IOC MQH Consistency

- ID: 38.3
- Level: 17
- Attributes: Test Module

Test consistency between the IO Cache and main memory.

- Assign a test address for each board.
  (Each board has a cache line in virtual page 0.)
- Put test data in memory.
- Initialize the IO chips for loopback operation.
- Verify that a read will miss and cause an IOC line fill.
- Verify that a read to the same address will hit in the IOC.
- Verify that a write miss will cause an IOC line fill.
- Verify that a write will hit in the IOC and update the IOC data.
- Verify that the IOC is the owner and will reply with a CPU read.
- Verify that the data became shared.

Verify that the IOC will flush owned data to memory.

- (A write miss must cause an IOC line fill.
  This write will hit in the IOC and update the IOC data.)
- Issue a read for address alias (flush).
  (The IOC must flush the dirty line to memory.)
- Check IOC flushed data.

*Possible Error Messages*

```
Block compare failed
   load address = %X data = %X.%X
   store address = %X data = %X.%X

Block check error
   address = %X
   expected = %X
   observed = %X

Stream ready bit timed out
```

```
Check tags failed
  cctag=%X.%X exp. state = %x
  bwtag=%X exp. state %x

Check Dcache tags failed
  address = %X
  expected valid bit = %x
  observed valid = %x

DCache tag has inconsistent state
  ptag = %X.%X

Read hit Ecache data error
  addr = %X
  expected = %X.%X
  observed = %X.%X

Read miss Ecache data error
  address = %X
  expected = %X.%X
  observed = %X.%X

Victimize error for address = %X

Write invalidate failed for address = %X

IO loopback read data error
  address = %X
  expected = %X.%X
  observed = %X.%X

IO loopback read miss failed, address = %X

IO loopback read hit failed, address = %X

IO loopback write miss failed, address = %X

IO loopback write hit failed, address = %X

IO cache shared owner failed, address = %X

IOC check tags line = %dex = %X %Xob = %X %X

IOC check data ex = %X ob = %X word = %d
```

```
IO cache flush data error
  address = %X
  expected = %X.%X
  observed = %X.%X

CPU read data
  address = %X
  expected = %X.%X
  observed = %X.%X

Check RefMiss count
  expected CRC=%X CMC=%X
  observed CRC=%X CMC=%X
```

❍❍●❍ ❍●●● **C0 BW-IOC Consistency** **0x27**

- ID: 39.0
- Attributes: General Purpose
- Diagnosis: IOC0

Set Configuration 0 and test consistency between BW and IOC.

Subtest: `C_0 BW,IOC,MQH`

- ID: 39.1
- Level: 17
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `Cache States`

- ID: 39.1
- Level: 17
- Attributes: Test Module
  Initialization Module

Walk the MXCC state table through all state transistions except the ones that require Write Invalidate Mode and another CPU.

The IOC is used in loopback mode to effect foreign reads and writes.

The following transistions are tested:

- Invalid to Valid, not Shared, Owner.
- Invalid to Valid, Shared, not Owner.
- Invalid to Valid, not Shared, not Owner.
- Valid to Valid via CPU read.
- Valid to Valid, not Shared, owner.
- Valid to Valid, Shared, not Owner via foreign read.
- Valid to Valid, Shared, not Owner via foreign write miss.
- Owner to Owner via CPU write.
- Owner to Owner via CPU read.
- Owner to Shared, Owner.
- Shared to Shared, Owner.
- Shared to Shared via CPU read.
- Shared to Shared via foreign read.
- Shared to Shared via foreign write.
- Shared to Shared via foreign write miss.
- Shared and Owner to Owner.
- Shared and Owner to Shared.
- Shared and Owner to Shared, Owner via CPU write.
- Shared and Owner to Shared, Owner via CPU read .
- Shared and Owner to Shared, Owner via foreign read.
- Shared and Owner to Shared via foreign write miss.

*Possible Error Messages*

```
Block compare failed
   load address = %X data = %X.%X
   store address = %X data = %X.%X
```

```
Block check error
  address = %X
  expected = %X
  observed = %X

Stream ready bit timed out

Check tags failed
  cctag=%X.%X exp. state = %x
  bwtag=%X exp. state %x

Check Dcache tags failed
  address = %X
  expected valid bit = %x
  observed valid = %x

DCache tag has inconsistent state
  ptag = %X.%X

Read hit Ecache data error
  addr = %X
  expected = %X.%X
  observed = %X.%X

Read miss Ecache data error
  address = %X
  expected = %X.%X
  observed = %X.%X

Victimize error for address = %X

Write invalidate failed for address = %X

IO loopback read data error
  address = %X
  expected = %X.%X
  observed = %X.%X

IO loopback read miss failed, address = %X

IO loopback read hit failed, address = %X

IO loopback write miss failed, address = %X

IO loopback write hit failed, address = %X
```

```
IO cache shared owner failed, address = %X

IOC check tags line = %dex = %X %Xob = %X %X

IOC check data ex = %X ob = %X word = %d

IO cache flush data error
  address = %X
  expected = %X.%X
  observed = %X.%X

CPU read data
  address = %X
  expected = %X.%X
  observed = %X.%X

Check RefMiss count
  expected CRC=%X CMC=%X
  observed CRC=%X CMC=%X
```

Subtest: BW IOC Consistency

- ID: 39.2
- Level: 17
- Attributes: Test Module

Test Consistency between the BW, CC, IO Cache, and main memory.

- Assign a test address for each board.
  (Each board has a cache line in virtual page 0.)
- Put test data in memory.
- Initialize the IO chips for loopback operation.
- Verify that a read will miss and cause an IOC line fill.
- Check BW, CC, and IOC tags for correct state.
- Verify that a read to the same address will hit in the IOC.
- Check BW, CC, and IOC tags for correct state.
- Verify that a write miss will cause an IOC line fill.
- Check BW, CC, and IOC tags for correct state.
- Verify that a write will hit in the IOC and update the IOC data.
- Check BW, CC, and IOC tags for correct state.
- Verify that the IOC is the owner and will reply with a CPU read.
- Verify that the data became shared.
- Check BW, CC, and IOC tags for correct state.

Verify that the IOC will flush owned data to memory.

- (A write miss must cause an IOC line fill.)
- Check BW, CC, and IOC tags for correct state.
  (This write will hit in the IOC and update the IOC data.)
- Check BW, CC, and IOC tags for correct state.
- Issue a read for address alias (flush).
  (The IOC must flush the dirty line to memory.)
- Check BW, CC, and IOC tags for correct state.
- Check IOC flushed data.

*Possible Error Messages*

```
Block compare failed
   load address = %X data = %X.%X
   store address = %X data = %X.%X

Block check error
   address = %X
   expected = %X
   observed = %X

Stream ready bit timed out

Check tags failed
   cctag=%X.%X exp. state = %x
   bwtag=%X exp. state %x

Check Dcache tags failed
   address = %X
   expected valid bit = %x
   observed valid = %x

DCache tag has inconsistent state
   ptag = %X.%X

Read hit Ecache data error
   addr = %X
   expected = %X.%X
   observed = %X.%X
```

```
Read miss Ecache data error
  address = %X
  expected = %X.%X
  observed = %X.%X

Victimize error for address = %X

Write invalidate failed for address = %X

IO loopback read data error
  address = %X
  expected = %X.%X
  observed = %X.%X

IO loopback read miss failed, address = %X

IO loopback read hit failed, address = %X

IO loopback write miss failed, address = %X

IO loopback write hit failed, address = %X

IO cache shared owner failed, address = %X

IOC check tags line = %dex = %X %Xob = %X %X

IOC check data ex = %X ob = %X word = %d

IO cache flush data error
  address = %X
  expected = %X.%X
  observed = %X.%X

CPU read data
  address = %X
  expected = %X.%X
  observed = %X.%X

Check RefMiss count
  expected CRC=%X CMC=%X
  observed CRC=%X CMC=%X
```

## ❍❍●❍ ●❍❍● SPARC Module Board Master 0x29

- ID: 41.0
- Attributes: General Purpose
- Diagnosis: CPUA Module
  CPUB Module

Test all functional elements of the SPARC Module

### Subtest: C_0 BW,MQH

- ID: 41.1
- Level: 8
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

### Subtest: CPU and Cache

- ID: 41.2
- Level: 8
- Attributes: Test Module
  Error Terminates Sequencer

Test the functionality of the CPU and Ecache on the SPARC module.

- Data Prefetcher. Data Cache must be enabled. Issue a series of LDDs/STDs with Data Prefetcher off, then with Data Prefetcher on, and compare non-prefetched buffers with prefetched buffers. (The data should be identical).
- SB Stress. Issue a series of stores with Store Buffer off, then with Store Buffer on, and compare the memory data. (The data should be identical.)
- Store Buffer Cacheable. Issue 8 cacheable store doubles and check the Store Buffer tags as well as the data. Read and check the memory data.
- Store Buffer (store/load) Stall CPU(snooping). Issue consecutive store/load pairs to insure that the loads are stalled until the stores complete.

- Store Buffer Access. Issue stores of all sizes and on all boundaries and check the memory data.
- CC Prefetch. Verify the CC prefetch logic. When prefetch is enabled, a prefetch should occur under the following conditions: when the next sequential subblock of the one just fetched is not valid and contained within the same line, then prefetch. Burst Read Miss Subblock n Prefetch Subblock n+1. Note that prefetch is only issued for burst reads (the Dcache must be enabled).
- Issue a load for subblock 0, read the CC tag, and insure that subblock 0 and 1 are valid. Then issue a load for subblock 1 and insure that subblock 2 gets prefetched. Then issue a load for subblock 2 and insure subblock 3 gets prefetched.
- Flush Line. Verify that when a line is victimized, the valid blocks in the line get flushed to memory.
- SPARC Module Features. Test all SPARC module features. Fill a buffer with the feature off, then enable the feature and fill a second buffer. Compare the two buffers for equality. Features tested are : Store Buffer, Prefetcher, Cache, Cache, PSO, Snoop, Multi Instruction, Multi Command, Ecache Prefetch, and Write Invalidate.

*Possible Error Messages*

```
Store Buffer Memory Compare Error
   address = %X
   expected = %X.%X
   observed = %X.%X

CC Prefetch failed
   address = %X
   fetch block = %1x
   prefetch block = %1x

Flush Block did not update memory
   address = %X
   flush block = %1x

Flush Block disturbed wrong memory block
   address = %X
   flush block = %1x
```

```
Block Compare Error
   Source Address = %X
   Destination = %X
   Byte Count = %X

Cache Data and Memory Data don't agree
   doubleword = %1X
   cache data = %X.%X
   memory data = %X.%X

char sparc_enable_err_txt[]=
  Data with features on not equal to data with features off
   buffer index = %x
   feature off data = %X.%X
   feature on data = %X.%X
```

Subtest: MMU PTP Cache Invalidation

- ID: 41.3
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Verify that the root and level 2 ptp caches are invalidated properly on context switches, ctpr switches, and so on.

- Initialize root ptp and l2 ptp caches with valid entries.
- Do the following: write to ctx register; write ctpr register; flush entire, flush context; flush region; flush segment; flush page.
- In each case, verify root ptp and l2 ptp caches are invalidated or left alone according to specifications.

This test issues demap packets (writes to TLB flush ASI). BWs and MQHs need to be on, and demap must be enabled in one MQH. It does not need memory.

*Possible Error Messages*

```
Wrong root ptp cache valid state
```

```
Wrong l2 ptp cache valid state
```

Subtest: `MMU Stuff TLB Hit`

- ID: 41.4
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Test basic functionality of MMU TLB.

Context test:

- Initialize TLBs with unique contexts and ptes but same vaddr tag (= 0).
- Initialize target memory locations with unique data.
- Read from vaddr 0 using different contexts and verify that you get the correct data each time.

`vaddr` test:

- Initialize TLBs with unique vaddr and ptes but same context (= 0).
- Initialize target memory locations with unique data.
- Read from different vaddr (= Walking 1s pattern through bits 31:12) with ctx = 0, and verify that you get the correct data each time.

Assumption: Ecache is enabled.

No table walks are done in this test. Make sure ASI 0x20 accesses are cacheable so that memory packets are generated instead of I/O packets.

*Possible Error Messages*

```
Wrong data on read with tlb-hit
```

```
Wrong data on read with tlb-hit
```

Subtest: `MMU Table Walk`

- ID: 41.5
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Test MMU table walk operation.

Probe l0 test:

- Initialize memory with unique l0 ptps.
- Do l0 probe using for the following cases: ctpr = 0x80000, 0x40000, ..., 0x2000, 0 ctx = 0x8000, 0x4000, ..., 1. Verify that you get expected l0 ptp.

Probe l1 test:

- Initialize memory with unique l1 ptes.
- Clear l0 ptp register.
- Do l1 probe with all different index1, and verify that you get correct l1 pte.
- Verify l0 ptp register gets updated.

Probe l2 test:

- Initialize memory with unique l2 ptes.
- Do l1 probe with all different index2, and verify that you get correct l2 pte.

Probe l3 test:

- Initialize memory with unique l3 ptes.
- Clear l2 ptp register.
- Do l3 probe with all different index3 and verify that you get correct l3 pte.
- Verify l2 ptp register gets updated.

Probe entire test:

- Initialize memory with unique l3 ptes.
- Do entire probe with all different index3 and verify that you get correct l3 pte.
- Verify ref bit is updated.

*Possible Error Messages*

```
Wrong root pointer on l0 probe

Wrong l1 entry on l1 probe

Wrong root ptp cache

Wrong l2 entry on l2 probe

Wrong l3 entry on l3 probe

Wrong l2 ptp cache

Wrong l2 vaddr cache

Wrong l3 entry on l3 probe

Ref bit is not set
```

Subtest: `MMU Flush`

- ID: 41.6
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Test MMU TLB flush operation.

Level0 (context):

- Initialize TLBs with unique ctx tags and same vaddr (= 0).
- `ctx-flush` with different ctxs and verify correct TLBs are invalidated after each flush.

Level1 (segment):

- Initialize TLBs with unique tags and same ctx (= 0).
- seg-flush with different index1 and verify correct TLBs are invalidated after each flush.

Level2 (region):

- Initialize TLBs with unique tags and same ctx (= 0).
- reg-flush with different index2 and verify correct TLBs are invalidated after each flush.

Level3 (page):

- Initialize TLBs with unique tags and same ctx (= 0).
- pag-flush with different index3 and verify correct TLBs are invalidated after each flush.

Level4 (entire):

- Initialize TLBs with unique tags and ctx.
- entire-flush and verify all TLBs are invalidated.

This test needs to enable MMU demaps().

*Possible Error Messages*

```
Wrong tlb after l0 flush

Wrong tlb after l1 flush

Wrong tlb after l2 flush

Wrong tlb after l3 flush

Wrong tlb after flush entire
```

Subtest: `MMU TLB Lock`

- ID: 41.7
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

The the locking of TLB entries.

- Set lock bits for all entries except 1.
- Initialize all TLBs with valid ptes, otherwise they will be replaced.
- Initialize memory with unique l1 ptes.
- Do probe entire for all different index1 and verify the unlocked entry is used each time.
- Repeat for other 63 entries.

*Possible Error Messages*

```
Wrong unlocked tlb entry
```

Subtest: `MMU TLB Protection Error`

- ID: 41.8
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Test MMU protection access traps. Map EPROM, exit boot, and do the following tests.

Invalid addr error test:

- Initialize invalid pte in memory.
- Do access ld/st user/super data ld/st user/super instr.
- Verify that you got trap 0x9 and that the MMU sync error register is updated correctly in each case.
- Repeat for level1 through level3 ptes.

Protect error test:

- Initialize pte in memory with acc[2:0] = 0 through 7.
- Do access ld/st user/super data ld/st user/super instr.
- If there is an access error, verify that you got trap 0x9 and that the MMU sync error register is updated correctly. If there is no access error, verify that the MMU error register is not updated.
- Repeat for level1 through level3 ptes.

*Possible Error Messages*

```
No trap on illegal permissions access

Wrong mmu fsr on twalk protect error

Wrong mmu far on twalk protect error

Got unexpected trap

Got mfsr-fault-valid
```

Subtest: MMU Table Walk With Parity Error

- ID: 41.9
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Do table walk with parity error.

- Make pte cacheable in Ecache.
- Do table walk without parity error.
- Repeat for different access types.
- Repeat for level 1-3 ptes.
- Repeat with parity error.

*Possible Error Messages*

```
No trap on twalk parity error

Wrong mmu fsr on twalk parity error

Wrong mmu far on twalk parity error
```

Subtest: MMU Table Walk With ECC Error

- ID: 41.10
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Do table walk with ECC error.

- (ptes not cacheable in Ecache.)
- Do table walk with ce error.
- Repeat for different access types.
- Repeat for level 1-3 ptes.
- Repeat with ue error.

*Possible Error Messages*

```
No trap on table walk w/ue

Wrong mmu fsr on table walk w/ue

Wrong mmu far on table walk w/ue
```

```
Wrong mqh ue error addr reg after table-walk w/ue

Wrong mqh ue error data reg after table-walk w/ue

No trap on table walk w/ce

Wrong mqh ce error addr reg after table-walk w/ce

Wrong mqh ce error data reg ce after table-walk w/ce
```

❍❍●❍ ●❍●● **SPARC Module Board Slave** **0x2B**

- ID: 43.0
- Attributes: General Purpose
- Diagnosis: CPUA Module
  CPUB Module

Test all functional elements of the SPARC Module

Subtest: `Read MQH State`

- ID: 43.1
- Level: 1
- Attributes: Error is Fatal

Read the alternate processor's MQH state array. (This is done so that the slave processor that calls this test does not rerun JTAG INIT on the board's MQH and cause possible refresh problems.)

*Possible Error Messages*

This module does not check or report errors.

Subtest: `C_0 BW,MQH`

- ID: 43.2
- Level: 8
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: CPU and Cache

- ID: 43.3
- Level: 8
- Attributes: Test Module
  Error Terminates Sequencer

Test the functionality of the CPU and Ecache on the SPARC module.

- Data Prefetcher. Data Cache must be enabled. Issue a series of LDDs/STDs with Data Prefetcher off, then with Data Prefetcher on, and compare non-prefetched buffers with prefetched buffers. (The data should be identical).
- SB Stress. Issue a series of stores with Store Buffer off, then with Store Buffer on, and compare the memory data. (The data should be identical.)
- Store Buffer Cacheable. Issue 8 cacheable store doubles and check the Store Buffer tags as well as the data. Read and check the memory data.
- Store Buffer (store/load) Stall CPU(snooping). Issue consecutive store/load pairs to insure that the loads are stalled until the stores complete.
- Store Buffer Access. Issue stores of all sizes and on all boundaries and check the memory data.
- CC Prefetch. Verify the CC prefetch logic. When prefetch is enabled, a prefetch should occur under the following conditions: when the next sequential subblock of the one just fetched is not valid and contained within the same line, then prefetch. Burst Read Miss Subblock n Prefetch Subblock n+1. Note that prefetch is only issued for burst reads (the Dcache must be enabled).
- Issue a load for subblock 0, read the CC tag, and insure that subblock 0 and 1 are valid. Then issue a load for subblock 1 and insure that subblock 2 gets prefetched. Then issue a load for subblock 2 and insure subblock 3 gets prefetched.
- Flush Line. Verify that when a line is victimized, the valid blocks in the line get flushed to memory.

- SPARC Module Features. Test all SPARC module features. Fill a buffer with the feature off, then enable the feature and fill a second buffer. Compare the two buffers for equality. Features tested are : Store Buffer, Prefetcher, Cache, Cache, PSO, Snoop, Multi Instruction, Multi Command, Ecache Prefetch, and Write Invalidate.

*Possible Error Messages*

```
Store Buffer Memory Compare Error
   address = %X
   expected = %X.%X
   observed = %X.%X

CC Prefetch failed
   address = %X
   fetch block = %1x
   prefetch block = %1x

Flush Block did not update memory
   address = %X
   flush block = %1x

Flush Block disturbed wrong memory block
   address = %X
   flush block = %1x

Block Compare Error
   Source Address = %X
   Destination = %X
   Byte Count = %X

Cache Data and Memory Data don't agree
   doubleword = %1X
   cache data = %X.%X
   memory data = %X.%X

char sparc_enable_err_txt[]=
  Data with features on not equal to data with features off
   buffer index = %x
   feature off data = %X.%X
   feature on data = %X.%X
```

Subtest: `MMU PTP Cache Invalidation`

- ID: 43.3
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Verify that the root and level 2 ptp caches are invalidated properly on context switches, ctpr switches, and so on.

- Initialize root ptp and l2 ptp caches with valid entries.
- Do the following: write to ctx register; write ctpr register; flush entire; flush context; flush region; flush segment; flush page.
- In each case, verify that the root ptp and l2 ptp caches are invalidated or left alone, according to specifications.

This test issues demap packets (writes to TLB flush ASI). BWs and MQHs need to be on, and demap must be enabled in one MQH. It does not need memory.

*Possible Error Messages*

```
Wrong root ptp cache valid state
```

```
Wrong l2 ptp cache valid state
```

Subtest: `MMU Stuff TLB Hit`

- ID: 43.4
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Test basic functionality of MMU TLB.

Context test:

- Initialize TLBs with unique contexts and ptes but same vaddr tag (= 0).
- Initialize target memory locations with unique data.
- Read from vaddr 0 using different contexts and verify that you get the correct data each time.

`vaddr` test:

- Initialize TLBs with unique vaddr and ptes but same context (= 0).
- Initialize target memory locations with unique data.
- Read from different vaddr (= Walking 1s pattern through bits 31:12) with ctx = 0, and verify that you get the correct data each time.

Assumption: Ecache is enabled.

No table walks are done in this test. Make sure ASI 0x20 accesses are cacheable so that memory packets are generated instead of I/O packets.

*Possible Error Messages*

```
Wrong data on read with tlb-hit
```

```
Wrong data on read with tlb-hit
```

Subtest: `MMU Table Walk`

- ID: 43.5
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Test MMU table walk operation.

Probe l0 test:

- Initialize memory with unique l0 ptps.
- Do l0 probe using for the following cases: ctpr = 0x80000, 0x40000, ..., 0x2000, 0 ctx = 0x8000, 0x4000, ..., 1 and verify that you get expected l0 ptp.

Probe l1 test:

- Initialize memory with unique l1 ptes.
- Clear l0 ptp register.
- Do l1 probe with all different index1 and verify that you get correct l1 pte.
- Verify l0 ptp register gets updated.

Probe l2 test:

- Initialize memory with unique l2 ptes.
- Do l1 probe with all different index2 and verify that you get correct l2 pte.

Probe l3 test:

- Initialize memory with unique l3 ptes.
- Clear l2 ptp register.
- Do l3 probe with all different index3 and verify that you get correct l3 pte.
- Verify l2 ptp register gets updated.

Probe entire test:

- Initialize memory with unique l3 ptes.
- Do entire probe with all different index3 and verify that you get correct l3 pte.
- Verify ref bit is updated.

*Possible Error Messages*

```
Wrong root pointer on l0 probe

Wrong l1 entry on l1 probe

Wrong root ptp cache

Wrong l2 entry on l2 probe

Wrong l3 entry on l3 probe

Wrong l2 ptp cache

Wrong l2 vaddr cache

Wrong l3 entry on l3 probe

Ref bit is not set
```

Subtest: `MMU Flush`

- ID: 43.6
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Test MMU TLB flush opeartion.

Level0 (context):

- Initialize TLBs with unique ctx tags and same vaddr (= 0).
- ctx-flush with different ctxs and verify correct TLBs are invalidated after each flush.

Level1 (segment):

- Initialize TLBs with unique tags and same ctx (= 0).
- seg-flush with different index1 and verify correct TLBs are invalidated after each flush.

Level2 (region):

- Initialize TLBs with unique tags and same ctx (= 0.)
- reg-flush with different index2 and verify correct TLBs are invalidated after each flush.

Level3 (page):

- Initialize TLBs with unique tags and same ctx (= 0).
- pag-flush with different index3 and verify correct TLBs are invalidated after each flush.

Level4 (entire):

- Initialize TLBs with unique tags and ctx.
- entire-flush and verify all TLBs are invalidated.

This test needs to enable mmu demaps().

*Possible Error Messages*

```
Wrong tlb after l0 flush

Wrong tlb after l1 flush

Wrong tlb after l2 flush

Wrong tlb after l3 flush

Wrong tlb after flush entire
```

Subtest: `MMU TLB Lock`

- ID: 43.7
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Test the locking of TLB entries.

- Set lock bits for all entries except 1.
- Initialize all TLBs with valid ptes otherwise they will be replaced.
- Initialize memory with unique l1 ptes.
- Do probe entire for all different index1 and verify the unlocked entry is used each time.
- Repeat for other 63 entries.

*Possible Error Messages*

`Wrong unlocked tlb entry`

Subtest: `MMU TLB Protection Error`

- ID: 43.8
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Test MMU protection access traps. Map EPROM, exit boot, and do the following tests.

Invalid addr error test:

- Initialize invalid pte in memory.
- Do access ld/st user/super data ld/st user/super instr.
- Verify that you got trap 0x9 and MMU sync error register is updated correctly in each case.
- Repeat for level1 through level3 ptes.

Protect error test:

- Initialize pte in memory with acc[2:0] = 0 through 7.
- Do access ld/st user/super data ld/st user/super instr.
- If there is an access error, verify that you got trap 0x9 and the MMU sync error register is updated correctly. If there is no access error, verify MMU error register is not updated.
- Repeat for level1 through level3 ptes.

*Possible Error Messages*

```
No trap on illegal permissions access

Wrong mmu fsr on twalk protect error

Wrong mmu far on twalk protect error

Got unexpected trap

Got mfsr-fault-valid
```

Subtest: `MMU Table Walk With Parity Error`

- ID: 43.9
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Do table walk with parity error.

- Make pte cacheable in Ecache.
- Do table-walk without parity error.
- Repeat for different access types.
- Repeat for level 1-3 ptes.
- Repeat with parity error.

*Possible Error Messages*

```
No trap on twalk parity error

Wrong mmu fsr on twalk parity error

Wrong mmu far on twalk parity error
```

Subtest: `MMU Table Walk With ECC Error`

- ID: 43.10
- Level: 17
- Attributes: Test Module
  Error Terminates Sequencer

Do table walk with ECC error.

- (ptes not cacheable in Ecache. )
- Do table walk with ce error.
- Repeat for different access types.
- Repeat for level 1-3 ptes.
- Repeat with ue error.

*Possible Error Messages*

```
No trap on table walk w/ue

Wrong mmu fsr on table walk w/ue

Wrong mmu far on table walk w/ue

Wrong mqh ue error addr reg after table-walk w/ue

Wrong mqh ue error data reg after table-walk w/ue

No trap on table walk w/ce

Wrong mqh ce error addr reg after table-walk w/ce

Wrong mqh ce error data reg ce after table-walk w/ce
```

## ○○●○ ●●○● OnBoard IO Verification 0x2D

- ID: 45.0
- Attributes: General Purpose
- Diagnosis: None

Test the IO functions of the Lance chip and FAS 236 chip that are on the system board.

Subtest: `C_0 BW,IOC,MQH`

- ID: 45.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `Check OnBoardIO Card0`

- ID: 45.2
- Level: 1
- Attributes: Test Module
  Error Terminates Sequencer

Check that Slot 0 did not trap when probed.

*Possible Error Messages*

```
Board %x Card0 not in this configuration
```

Subtest: `Lance Memory`

- ID: 45.3
- Level: 17
- Attributes: Test Module

Test the Lance memory buffer.

- Do 1/2/4/8 byte accesses to entire 32-byte at each address bit.
- Do marching a5's (forward and backward) to entire 128Kbytes.

*Possible Error Messages*

```
Lance memory compare error
 address = %X
 expected = %X
 observed = %X
```

```
Lance memory compare error
 address = %X
 expected = %X.%X
 observed = %X.%X
```

Subtest: Lance Registers

- ID: 45.4
- Level: 17
- Attributes: Test Module

Lance RAP write / read / read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x4000, 0x8000.
- Write halfword testdata to Lance RAP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RAP register.
- Compare and print error message if miscompare.
- Testdata = 0xffff.
- Write halfword testdata to Lance RAP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RAP register.
- Compare and print error message if miscompare.

Lance CSR0 STOP bit test.

- Write 0 to RAP to point to CSR0 register.
- Testdata = CSR0_STOP.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.

Lance CSR1 register write / write / read test.

- Write 1 to RAP to point to CSR1 register.
- Testdata = 0x1, 0x2, 0x4, ..., 0x4000, 0x8000.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.
- Testdata = 0xffff.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.

- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.

Lance CSR2 register write / write / read test.

- Write 2 to RAP to point to CSR2 register.
- Testdata = 0x1, 0x2, 0x4, ..., 0x4000, 0x8000.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.
- Testdata = 0xffff.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.

Lance CSR3 register write / write / read test.

- Write 3 to RAP to point to CSR3 register.
- Testdata = 0x1, 0x2, 0x4, ..., 0x4000, 0x8000.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.
- Testdata = 0xffff.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.

*Possible Error Messages*

```
Data Compare Error
 address = %X
 expected = %4X
 observed = %4X

char lance_rap_wr_rd_txt[]=
 RAP write/write/read miscompare

char lance_csr0_not_cleared_txt[]=
 CSR0 register not cleared when STOP bit set
```

```
char lance_csr1_stop_txt[]=
 CSR1 write/write/read miscompare when CSR0 STOP bit set

char lance_csr2_stop_txt[]=
 CSR2 write/write/read miscompare when CSR0 STOP bit set

char lance_csr3_stop_txt[]=
 CSR3 write/write/read miscompare when CSR0 STOP bit set

char lance_rdp_rap_wr_rd_b2b_txt[]=
 RDP/RAP back-to-back write/write/read/read miscompare
```

Subtest: Lance Local Loopback

- ID: 45.5
- Level: 17
- Attributes: Test Module

Test the Lance transmit and receive functions, using an internal loopback.

- Initialize the init block, transmit, receive pointers.
- Issue stop to Lance chip and verify it stopped.
- Set to internal loopback.
- Initialize the init block parameters.
- Initialize the receive block parameters.
- Initialize the transmit block parameters.
- Load CSR1,2 with the base of the init structure.
- Start timeout loop waiting for Lance chip to init.
- Set up transmit and receive buffer data (32 byte incrementing pattern).
- Load buffer for transmit and clear receive buffer.
- Set up destination address in transmit block.
- Set up source address in transmit block.
- Send transmit command.
- Start timeout loop waiting for transmit confirmation.
- Start timeout loop waiting for receive confirmation.
- Verify the receive data = transmit data.

*Possible Error Messages*

```
WARNING Check Ethernet cable

Lance chip initialization failed
 CSR0 expected = %4X
 CSR0 observed = %4X

Error condition detected
 CSR0 = %4X

%s
CSR0 = %4X
 Rx msg Descriptor 1 = %4X
 Tx Msg Descriptor 1 = %4X
 Tx Msg Descriptor 3 = %4X

Rx data error
 address = %X
 expected = %2X
 observed = %2X
```

Subtest: ESC Registers

- ID: 45.6
- Level: 17
- Attributes: Test Module

SCSI DVMA control register write / write / read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x40000000, 0x80000000.
- Write word testdata to ESC DMA Control register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Control register.
- Compare and print error message if miscompare.
- Testdata = 0x0.
- Write word testdata to ESC DMA Control register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Control register.
- Compare and print error message if miscompare.

SCSI DVMA address register write/write/read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x40000000, 0x80000000.
- Write word testdata to ESC DMA Address register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Address register.
- Compare and print error message if miscompare.
- Testdata = 0xffffffff.
- Write word testdata to ESC DMA Address register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Address register.
- Compare and print error message if miscompare.

SCSI DVMA count register write/write/read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x40000000, 0x80000000.
- Write word testdata to ESC DMA Count register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Count register.
- Compare and print error message if miscompare.
- Testdata = 0xffffffff.
- Write word testdata to ESC DMA Count register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Count register.
- Compare and print error message if miscompare.

ESC register back-to-back write/write/read/read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x40000000, 0x80000000.
- Write word inverse testdata to ESC DMA Address register.
- Write word testdata to ESC DMA Count register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word inverse testdata from ESC DMA Address register.
- Read word testdata from ESC DMA Count register.
- Compare and print error message if miscompare.
- Testdata = 0xffffffff.
- Write word inverse testdata to ESC DMA Address register.
- Write word testdata to ESC DMA Count register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word inverse testdata from ESC DMA Address register.
- Read word testdata from ESC DMA Count register.
- Compare and print error message if miscompare.

*Possible Error Messages*

```
Register data compare error
 address = %X
 expected = %X
 observed = %X

char esc_d_ctl_err_txt[]=
 ESC SCSI DVMA Control

char esc_d_adr_err_txt[]=
 ESC SCSI DVMA Address

char esc_d_cnt_err_txt[]=
 ESC SCSI DVMA Count

char esc_reg_b2b_err_txt[]=
 ESC back-to-back access
```

Subtest: FAS236 Registers

- ID: 45.7
- Level: 17
- Attributes: Test Module

FAS configuration register #1 write/write/read test.

- For testdata = 0x1, 0x2, 0x4, ..., 0x40, 0x80 {
- Write byte testdata to FAS configuration register #1.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #1.
- Compare and print error message if miscompare.
- testdata = 0xff.
- Write byte testdata to FAS configuration register #1.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #1.
- Compare and print error message if miscompare.

FAS configuration register #2 write / write / read test.

- For testdata = 0x1, 0x2, 0x4, ..., 0x40, 0x80 {
- Write byte testdata to FAS configuration register #2.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #2.
- Compare and print error message if miscompare.
- testdata = 0xff.
- Write byte testdata to FAS configuration register #2.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #2.
- Compare and print error message if miscompare.

FAS configuration register #3 write / write / read test.

- For testdata = 0x1, 0x2, 0x4, ..., 0x40, 0x80 {
- Write byte testdata to FAS configuration register #3.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #3.
- Compare and print error message if miscompare.
- testdata = 0xff.
- Write byte testdata to FAS configuration register #3.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #3.
- Compare and print error message if miscompare.

FAS configuration register #1, #2 back-to-back write / write / read / read test.

- For testdata = 0x1, 0x2, 0x4, ..., 0x40, 0x80 {
- Write byte inverse testdata to FAS configuration register #1.
- Write byte testdata to FAS configuration register #2.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte inverse testdata from FAS configuration register #1.
- Read byte testdata from FAS configuration register #2.
- Compare and print error message if miscompare.
- testdata = 0xff.
- Write byte inverse testdata to FAS configuration register #1.
- Write byte testdata to FAS configuration register #2.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte inverse testdata from FAS configuration register #1.
- Read byte testdata from FAS configuration register #2.
- Compare and print error message if miscompare.

*Possible Error Messages*

```
char fas_xfr_low_txt[]=
 Transfer count low register compare error

char fas_xfr_hi_txt[]=
 Transfer count high register compare error

char fas_config1_txt[]=
 Config register #1 compare error

char fas_config2_txt[]=
 Config register #2 compare error

char fas_config3_txt[]=
 Config register #3 compare error

char fas_config2_b2b_txt[]=
 Config reg #1,#2 back-to-back compare error
```

Subtest: SCSI DVMA Read

- ID: 45.8
- Level: 17
- Attributes: Test Module

Program the FAS and ESC chips to do a DVMA read and verify.

- Reset the FAS-236.
- Send NOP command to FAS after doing reset.
- Set bit 3 (Chip Test Mode) of the Configuration Register #1.
- Set bit 0 (Target Test Mode) of the Test Register.
- Load the Transfer Count Register with the number of bytes to be transferred.
- Load the ESC SCSI DVMA Count register with the byte count.
- Store the virtual address in ESC SCSI DVMA Address register.
- Make this an INVALID virtual address.
- Store the test bytes into the FIFO register.
- Issue a 'SEND DATA W/DMA command (0xA2 into command register.
- Turn on DMA and set direction in ESC SCSI DVMA Control register.
- Wait 1uS to allow for internal command synchronization.
- Wait for the bytes to be transferred via DMA control (TCZERO).
- Check for timeout.
- Verify the transfered bytes by reading the FIFO register.
- Reset the FAS-236 to release the SCSI bus.

*Possible Error Messages*

```
Data Compare Error
 address = %X
 expected = %X
 observed = %X

char fas_dvma_xfr_timeout_err_txt[]=
 Transfer timeout waiting on SCSI DVMA Control register

char fas_dvma_read_fifo_err_txt[]=
 FIFO data compare error

char fas_dvma_write_mem_err_txt[]=
 Memory data compare error
```

## Subtest: SCSI DVMA Write

- ID: 45.9
- Level: 17
- Attributes: Test Module

Program the FAS and ESC chips to do DVMA write and verify.

- Reset the FAS-236.
- Send NOP command to FAS after doing reset.
- Set bit 3 (Chip Test Mode) of the Configuration Register #1.
- Set bit 0 (Target Test Mode) of the test register.
- Load the Transfer Count Register with the number of bytes to be transferred.
- Load the ESC SCSI DVMA Count register with the byte count.
- Store the virtual address in ESC SCSI DVMA Address register.
- Store the test data bytes directly into the FIFO register.
- Issue a 'RECEIVE DATA W/DMA command (0xAA into command register).
- Turn on DMA and set direction in ESC SCSI DVMA Control register.
- Wait 1uS to allow for internal command synchronization.
- Wait for the bytes to be transferred via DMA control (TCZERO).
- Flush the SBI Write Buffers.
- Now send the DRAIN command to the ESC.
- Check that the ACTIVE0 and ACTIVE1 bits in SCSI DVMA Control register are cleared, indicating the data transfer is complete.
- Check for timeout.

- Verify the transfered bytes by reading main memory.
- Reset the FAS-236 to release the SCSI bus.

*Possible Error Messages*

```
Data Compare Error
 address = %X
 expected = %X
 observed = %X

char fas_dvma_xfr_timeout_err_txt[]=
 Transfer timeout waiting on SCSI DVMA Control register

char fas_dvma_read_fifo_err_txt[]=
 FIFO data compare error

char fas_dvma_write_mem_err_txt[]=
 Memory data compare error
```

Subtest: SCSI DVMA Write Error

- ID: 45.10
- Level: 17
- Attributes: Test Module

Program the FAS and ESC chips to do a DVMA write to an invalid address and verify that the correct error condition occurs.

- Reset the FAS-236.
- Send NOP command to FAS after doing reset.
- Set bit 3 (Chip Test Mode) of the Configuration Register #1.
- Set bit 0 (Target Test Mode) of the Test Register.
- Load the Transfer Count Register with the number of bytes to be transferred.
- Load the ESC SCSI DVMA Count register with the byte count.
- Store the INVALID virtual address in ESC SCSI DVMA Address register.
- Store the test data bytes directly into the FIFO register.
- Issue a 'RECEIVE DATA W/DMA command (0xAA into command register).
- Turn on DMA and set direction in ESC SCSI DVMA Control register.
- Wait 1uS to allow for internal command synchronization.
- Wait for error bit (DMA_ERR) to be set in SCSI DMA Control register.
- Check for timeout.
- Reset the FAS-236 to release the SCSI bus.

*Possible Error Messages*

```
Data Compare Error
 address = %X
 expected = %X
 observed = %X

char fas_dvma_xfr_timeout_err_txt[]=
 Transfer timeout waiting on SCSI DVMA Control register

char fas_dvma_read_fifo_err_txt[]=
 FIFO data compare error

char fas_dvma_write_mem_err_txt[]=
 Memory data compare error
```

## 2.3 Loopback Exit

Using JTAG, take the BICs out of loopback. From this point on the BICs should not be in loopback.

❍●❍● ●❍❍● **Bus Ring** **0x59**

- ID: 89.0
- Attributes: General Purpose
- Diagnosis: XDBus0

Test continuity and length of the JTAG scan ring and the component IDs of all the chips on Bus Interface JTAG scan ring.

Subtest: `Verify Bus Ring`

- ID: 89.2
- Level: 1
- Attributes: Test Module
  Initialization Module

Scan in the ring containing the BIC and BARB ASICs; verify that the JTAG data is correct.

*Possible Error Messages*

```
JTAG TAP state machine not responding

Incorrect arguments passed by caller

JTAG component ID does not match

JTAG ring continuity test failed

State after initialization not expected

Ring length does not match expected
```

❍●❍● ●❍●● **C0 BP Check** **0x5B**

- ID: 91.0
- Attributes: General Purpose
- Diagnosis: XDBus0

Let each board align (using system LEDs) and run a backplane check. The boards are staggered in time so that they will not interfere with each other.

Subtest: `Wait for Alt`

- ID: 91.1
- Level: 1
- Attributes: Error is Fatal

Perform a synchronization function to insure both CPUs on this board are at a known state before continuing further.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `XDBus setup C_0`

- ID: 91.1
- Level: 1
- Attributes: Error is Fatal

Set up XDBus 0 configuration.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `C0 Backplane Check`

- ID: 91.2
- Level: 1
- Attributes: Test Module

Each board has a pre-assigned time to test its connection to the backplane. Using JTAG, take the BICs for the bus under test out of loopback. Issue a series of reads and writes to the BW's DynaData register using several test patterns. Using JTAG, put the BICs back into loopback.

*Possible Error Messages*

```
Read DDR at %X caused Data Access Exception
```

```
Data Compare Error
   address = %X
   expected = %X
   observed = %X
```

## ❍●❍● ●●❍● C0 Exit LB 0x5D

- ID: 93.0
- Attributes: General Purpose
- Diagnosis: XDBus0

Let each board exit loopback so that it can now use the backplane.

Subtest: `Loopback Exit C_0`

- ID: 93.1
- Level: 1
- Attributes: Test Module

Using JTAG, take the BICs out of loopback. From this point on the BICs should not be in loopback.

*Possible Error Messages*

This module does not check or report errors.

## 2.4 System Master Selection

POST chooses a System Master from one of the Board Masters in the system. The CPU that becomes the System Master is the CPU with a functional BootBus on the lowest-numbered board. After the System Master is selected, all other CPUs become slaves and wait for assignments from the System Master. (The System Master tests non-processor boards, then performs final system configuration, as described in the following sections.)

## 2.5 System Level Testing

The Following series of tests run after Dyanbus Loopback Exit. All System Board Components may now interact with each other. This interactivity is exercised and checked for correct results.

❍●●● ❍●●❍ **C0 NPB Loopback Exit** **0x76**

- ID: 118.0
- Attributes: NonProcessor Board Test
- Diagnosis: XDBus0

Take this non-processor board out of loopback.

Subtest: `Loopback Exit`

- ID: 118.1
- Level: 1
- Attributes: Test Module

Using JTAG, take the BICs out of loopback. From this point on the BICs should not be in loopback.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `Marking NPB`

- ID: 118.2
- Level: 1
- Attributes: Test Module

A non-processor board has been detected. Note its presence so that the test sequencer will dispatch tests to test it.

*Possible Error Messages*

This module does not check or report errors.

## ❍●●● ❍●●● C0 NPB MQH 0x77

- ID: 119.0
- Attributes: NonProcessor Board Test
- Diagnosis: MQH0

Test MQHs on non-processor boards. This test is run by the C_0 system master.

Subtest: `Check BDA`

- ID: 119.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Insure that the part to be tested next has not already failed in POST.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `C_0 NPB MQH`

- ID: 119.2
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `MQH Registers`

- ID: 119.3
- Level: 8
- Attributes: Test Module
  Error Terminates Sequencer

Test the read and write accessibility of the MQH ASIC registers, using all access sizes allowed. The addresses of the MQH registers are in CSR space. If any access causes a data access exception or unexpected interrupt, the test aborts with a `FAIL` status.

The ECC Error registers are read only. Testing is limited to insuring that register access does not cause a trap and that all error bits are cleared.

The Group Type registers are read only; they are not tested.

To prevent losing refresh, the Refresh Enable bit in the MCSR is never reset.

When it is appropriate, the test restores the original value it found in the register.

*Possible Error Messages*

```
While testing %s register an unexpected trap occurred
   MFSR = %X
   MFAR = %X
   Trap Type = %2x
   CC Error = %X.%X

Unexpected Component ID value
   address = %X
   expected = %X or %X
   observed = %X

%s register failed to return correct data
   address = %X
   expected = %X
   observed = %X
```

```
%s register failed to return correct data
  expected = %X.%X
  observed = %X.%X

Floating a bit through %s register failed
  expected = %X
  observed = %X

Floating a bit through %s register failed
  expected = %X.%X
  observed = %X.%X
```

Subtest: MQH Initialization

- ID: 119.4
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Set up the MQH timing registers and control register. Timing values loaded depend on the types of SIMMs present. Turn on Refresh Enable, set Refresh Count, Request Delay.

*Possible Error Messages*

This module does not check or report errors.

Subtest: Enable ECC

- ID: 119.5
- Level: 1
- Attributes: Error is Fatal

Enable ECC checking on the MQH.

*Possible Error Messages*

This module does not check or report errors.

Subtest: `Memory`

- ID: 119.6
- Level: 8
- Attributes: Test Module

Test all memory on this MQH. If a group with memory is not found, return `FAIL`. The purpose of the test is to test enough memory to allow the consistency tests to run. The memory test functions are loaded into the Icache for speed.

The short memory test algorithm is:

- Clear the number of memory faults in the current test.
- Load the CC Stream Data register with alternate patterns.
- Loop through memory, writing 64 bytes at a time.
- Check the memory.
- Set up and load the alternate pattern.
- Loop through memory, writing 64 bytes at a time.
- Check the memory.
- The permanent ECC handler handles memory errors. If the faults exceed 2, return `FAIL`.

The long memory test algorithm is:

- Clear the number of memory faults in the current test.
- Loop through a set of long long patterns.
- Load the stream data register with a pattern.
- Fill memory with a pattern.
- Load the stream data register with ~pattern.
- Read, then write ~pattern.
- Read ~pattern, write pattern, read.

*Possible Error Messages*

```
Memory Compare Failure
  Addr %X
  Expected %X.%X
  Observed %X.%X
```

Subtest: Config Memory Available

- ID: 119.7
- Level: 1
- Attributes: Error is Fatal

Count the amount of memory available in the current configuration.

*Possible Error Messages*

This module does not check or report errors.

## ○●●● ●○○○ C0 NPB IO Ring 0x78

- ID: 120.0
- Attributes: NonProcessor Board Test
- Diagnosis: IOC0
  SBI

Test the JTAG IO ring on non-processor boards. This test is run by the C_0 system master.

Subtest: Verify IO Ring

- ID: 120.2
- Level: 1
- Attributes: Test Module

Scan in the ring containing the IOC and SBI ASICs; verify that the JTAG data is correct.

*Possible Error Messages*

```
JTAG TAP state machine not responding

Incorrect arguments passed by caller

JTAG component ID does not match

JTAG ring continuity test failed

State after initialization not expected

Ring length does not match expected
```

## ○●●● ●○○● C0 NPB IO 0x79

- ID: 121.0
- Attributes: NonProcessor Board Test
- Diagnosis: IOC0

Test IOC0s on non-processor boards. This test is run by the C_0 system master.

### Subtest: Check BDA

- ID: 121.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Insure that the part to be tested next has not already failed eariler in POST

*Possible Error Messages*

This module does not check or report errors.

### Subtest: C_0 NPB IOC

- ID: 121.2
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: IOC Registers

- ID: 121.3
- Level: 8
- Attributes: Test Module
  Error Terminates Sequencer

Test the read and write accessibility of all IOC ASIC registers, using all access sizes allowed. The addresses of the IOC registers are in CSR space. If any access causes a data access exception or unexpected interrupt, the test aborts with a FAIL status.

*Possible Error Messages*

```
While testing %s register an unexpected trap occurred
   MFSR = %X
   MFAR = %X
   Trap Type = %2x
   CC Error = %X.%X

Unexpected Component ID value
   address = %X
   expected = %X or %X
   observed = %X

%s register failed to return correct data
   address = %X
   expected = %X
   observed = %X

%s register failed to return correct data
   expected = %X.%X
   observed = %X.%X

Floating a bit through %s register failed
   expected = %X
   observed = %X

Floating a bit through %s register failed
   expected = %X.%X
   observed = %X.%X
```

Subtest: `IOC XDBus Tags`

- ID: 121.4
- Level: 8
- Attributes: Test Module

Read, write, and verify the IOC's XDBus tags.

*Possible Error Messages*

```
Data Compare Error
  address = %X
  expected = %X
  observed = %X
```

Subtest: `IOC Sbus Tags`

- ID: 121.5
- Level: 8
- Attributes: Test Module

Read, write, and verify the IOC's SBus tags.

*Possible Error Messages*

```
Data Compare Error
  address = %X
  expected = %X
  observed = %X
```

Subtest: `IOC Cache RAM`

- ID: 121.6
- Level: 8
- Attributes: Test Module

Read, write, and verify the IO Cache RAM.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X
   observed = %X
```

## ❍●●● ●❍●❍ C0 NPB SBI 0x7A

- ID: 122.0
- Attributes: NonProcessor Board Test
- Diagnosis: SBI

Test the SBI on the non-processor board.

### Subtest: SBI Initialization

- ID: 122.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Initialize all SBI registers to the default values.

*Possible Error Messages*

This module does not check or report errors.

### Subtest: SBI Registers

- ID: 122.2
- Level: 8
- Attributes: Test Module
  Error Terminates Sequencer

Test the read and write accessibility of all SBI ASIC registers, using all access sizes allowed. The addresses of the SBI registers are in ECSR space. If any access causes a data access exception or unexpected interrupt, the test aborts with a `FAIL` status.

The SBI Interrupt registers, with the exception of the Interrupt Target ID, are not tested here, but are tested in the SBus Interrupts test.

*Possible Error Messages*

```
While testing %s register an unexpected trap occurred
   MFSR = %X
   MFAR = %X
   Trap Type = %2x
   CC Error = %X.%X

Unexpected Component ID value
   address = %X
   expected = %X or %X
   observed = %X

%s register failed to return correct data
   address = %X
   expected = %X
   observed = %X

%s register failed to return correct data
   expected = %X.%X
   observed = %X.%X

Floating a bit through %s register failed
   expected = %X
   observed = %X

Floating a bit through %s register failed
   expected = %X.%X
   observed = %X.%X

%s fields SEGA, C, S updated when WSA = 0
   expected = %X
   observed = %X
```

Subtest: SBI Initialization

- ID: 122.3
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Initialize all SBI registers to the default values.

*Possible Error Messages*

This module does not check or report errors.

Subtest: SBus Interrupts

- ID: 122.4
- Level: 8
- Attributes: Test Module

Test all levels of SBus interrupts for all SBus slots. Verify that the correct interrupt state is recorded in the BWs, MXCC, and SBI chips. Insure that the correct SPARC interrrupt level is delivered to the CPU. This test is executed on eack board by the CPU on that board. SBus slots on non-processor boards will be tested using this test.

- Mask all interrupts except level 15s.
- Clear all existing interrupt states.
- Establish this board‘s BW as the target for this SBus’s interrupts.
- Verify the CC transaction to the SBI interrupt target register.
- Loop for all levels; SBus has levels 1 through 7.
- Loop for all slots; each board has 4 SBus slots.
- Use diagnostic register to generate SBus interrupt.
- Issue a `TAKE` and check the state register.
- issue a `GIVE` and check the state register.
- Check the BW interrupt table.
- Check CC interrupt pending.
- Unmask the interrupt and insure the CPU gets the correct interrupt.
- Do necessary housekeeping.
- Clean up before exiting.

*Possible Error Messages*

```
Failed to establish new targer id, Board %x
  Address = %X
  expected = %2X
  observed = %2X

Incorrect Interrupt State, Board %x
  Address = %X
  expected = %2X
  observed = %2X
```

```
Incorrect CC Interrupt Pending, Board %x Slot %x
  Address = %X
  expected = %4X
  observed = %4X

Incorrect BW Interrupt Table, Board %x Slot %x
  Address = %X
  expected = %4X
  observed = %4X

SBus Interrupt not delivered to CPU, Board %x
  Slot = %x
  Level = %x
  Trap Type = %2x
```

## ❍●●● ●❍●● C0 NPB SBUS Cards 0x7B

- ID: 123.0
- Attributes: NonProcessor Board Test
- Diagnosis: None

Probe each slot on this board to see if a card responds.

### Subtest: SBI Initialization

- ID: 123.3
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Initialize all SBI registers to the default values.

*Possible Error Messages*

This module does not check or report errors.

### Subtest: Checking for SBUS cards

- ID: 123.5
- Level: 1
- Attributes: Error is Fatal

Check each slot to see if a card responds.

*Possible Error Messages*

This module does not check or report errors.

## ❍●●● ●●❍❍ C0 NPB Delay 0x7C

- ID: 124.0
- Attributes: NonProcessor Board Test
- Diagnosis: None

Allow the SBus cards time to self-initialize.

Subtest: TOD Delay

- ID: 124.1
- Level: 1
- Attributes: Error is Fatal

Use the TOD for a timed delay, allowing SBus devices to perform self initialization.

*Possible Error Messages*

This module does not check or report errors.

Subtest: TOD Delay

- ID: 124.1
- Level: 1
- Attributes: Error is Fatal

Use the TOD for a timed delay, allowing SBus devices to perform self initialization.

*Possible Error Messages*

This module does not check or report errors.

## ❍●●● ●●❍● C0 NPB XPT 0x7D

- ID: 125.0
- Attributes: NonProcessor Board Test
- Diagnosis: SBI

Test XPTs on non-processor boards. This test is run by the C_0 system master.

### Subtest: XPT Read Write

- ID: 125.1
- Level: 17
- Attributes: Test Module
  - For the first part of the test, force parity.
  - Set up the SBI control register.
  - Do a 6N test.
  - Load a new pattern.
  - Test with the next pattern.
  - First pass: write in ascending order.
  - Second pass: read-then-write in ascending order.
  - Third pass: read-compare in descending order.
  - Move the pattern pointer along.
  - Loop for all patterns.
  - Restore original SBI control register.
  - Test with some even and odd parity patterns.
  - Clear the XPT so that there is good parity.

*Possible Error Messages*

```
Data Compare Error
   address = %X
   expected = %X
   observed = %X
```

## ❍●●● ●●●❍ NPB OnBoard IO Verification 0x7E

- ID: 126.0
- Attributes: NonProcessor Board Test
- Diagnosis: None

Test the IO functions of the Lance chip and FAS 236 chip that are on the NPB System Board.

Subtest: C_0 NPB IOC,MQH

- ID: 126.1
- Level: 1
- Attributes: Error Terminates Sequencer
  Error is Fatal

Establish the board configuration for this test.

*Possible Error Messages*

This module does not check or report errors.

Subtest: Check OnBoardIO Card0

- ID: 126.2
- Level: 1
- Attributes: Test Module
  Error Terminates Sequencer

Check that Slot 0 did not trap when probed.

*Possible Error Messages*

```
Board %x Card0 not in this configuration
```

Subtest: Lance Memory

- ID: 126.3
- Level: 17
- Attributes: Test Module

Test the Lance memory buffer.

- Do 1/2/4/8 byte accesses to entire 32-byte at each address bit.
- Do marching a5's (forward and backward) to entire 128Kbytes.

*Possible Error Messages*

```
Lance memory compare error
   address = %X
   expected = %X
   observed = %X

Lance memory compare error
   address = %X
   expected = %X.%X
   observed = %X.%X
```

Subtest: `Lance Registers`

- ID: 126.4
- Level: 17
- Attributes: Test Module

Lance RAP write/read/read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x4000, 0x8000.
- Write halfword testdata to Lance RAP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RAP register.
- Compare and print error message if miscompare.
- Testdata = 0xffff.
- Write halfword testdata to Lance RAP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RAP register.
- Compare and print error message if miscompare.

Lance CSR0 STOP bit test.

- Write 0 to RAP to point to CSR0 register.
- Testdata = CSR0_STOP.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.

Lance CSR1 register write/write/read test.

- Write 1 to RAP to point to CSR1 register.
- Testdata = 0x1, 0x2, 0x4, ..., 0x4000, 0x8000.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.
- Testdata = 0xffff.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.

Lance CSR2 register write/write/read test.

- Write 2 to RAP to point to CSR2 register.
- Testdata = 0x1, 0x2, 0x4, ..., 0x4000, 0x8000.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.
- Testdata = 0xffff.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.

Lance CSR3 register write/write/read test.

- Write 3 to RAP to point to CSR3 register.
- Testdata = 0x1, 0x2, 0x4, ..., 0x4000, 0x8000.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.
- Testdata = 0xffff.
- Write halfword testdata to Lance RDP register.
- Write halfword inverse testdata to word 0 in Lance buffered memory.
- Read halfword testdata from Lance RDP register.
- Compare and print error message if miscompare.

*Possible Error Messages*

```
%s
   address = %X
   expected = %4X
   observed = %4X

char lance_rap_wr_rd_txt[]=
 RAP write/write/read miscompare

char lance_csr0_not_cleared_txt[]=
 CSR0 register not cleared when STOP bit set

char lance_csr1_stop_txt[]=
 CSR1 write/write/read miscompare when CSR0 STOP bit set

char lance_csr2_stop_txt[]=
 CSR2 write/write/read miscompare when CSR0 STOP bit set

char lance_csr3_stop_txt[]=
 CSR3 write/write/read miscompare when CSR0 STOP bit set

char lance_rdp_rap_wr_rd_b2b_txt[]=
 RDP/RAP back-to-back write/write/read/read miscompare
```

Subtest: Lance Local Loopback

- ID: 126.5
- Level: 17
- Attributes: Test Module

Test Lance transmit and receive functions using internal loopback.

- Initialize the init block, transmit, receive pointers.
- Issue stop to Lance chip and verify it stopped.
- Set to internal loopback.
- Initialize the init block parameters.
- Initialize the receive block parameters.
- Initialize the transmit block parameters.
- Load CSR1,2 with the base of the init structure.
- Start timeout loop waiting for Lance chip to init.
- Set up transmit and receive buffer data (32 byte incrementing pattern).
- Load buffer for transmit and clear receive buffer.

- Set up destination address in transmit block.
- Set up source address in transmit block.
- Send transmit command.
- Start timeout loop waiting for transmit confirmation.
- Start timeout loop waiting for receive confirmation.
- Verify the receive data = transmit data.

*Possible Error Messages*

```
WARNING Check Ethernet cable
```

```
Lance chip initialization failed
   CSR0 expected = %4X
   CSR0 observed = %4X
```

```
Error condition detected
   CSR0 = %4X
```

```
%s
 CSR0 = %4X
 Rx msg Descriptor 1 = %4X
 Tx Msg Descriptor 1 = %4X
  x Msg Descriptor 3 = %4X
```

```
Rx data error
   address = %X
   expected = %2X
   observed = %2X
```

Subtest: ESC Registers

- ID: 126.6
- Level: 17
- Attributes: Test Module

SCSI DVMA control register write/write/read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x40000000, 0x80000000.
- Write word testdata to ESC DMA Control register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Control register.
- Compare and print error message if miscompare.
- Testdata = 0x0.
- Write word testdata to ESC DMA Control register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Control register.
- Compare and print error message if miscompare.

SCSI DVMA address register write/write/read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x40000000, 0x80000000.
- Write word testdata to ESC DMA Address register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Address register.
- Compare and print error message if miscompare.
- Testdata = 0xffffffff.
- Write word testdata to ESC DMA Address register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Address register.
- Compare and print error message if miscompare.

SCSI DVMA count register write/write/read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x40000000, 0x80000000.
- Write word testdata to ESC DMA Count register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Count register.
- Compare and print error message if miscompare.
- Testdata = 0xffffffff.
- Write word testdata to ESC DMA Count register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word testdata from ESC DMA Count register.
- Compare and print error message if miscompare.

ESC register back-to-back write / write / read / read test.

- Testdata = 0x1, 0x2, 0x4, ..., 0x40000000, 0x80000000.
- Write word inverse testdata to ESC DMA Address register.
- Write word testdata to ESC DMA Count register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word inverse testdata from ESC DMA Address register.
- Read word testdata from ESC DMA Count register.
- Compare and print error message if miscompare.
- Testdata = 0xffffffff.
- Write word inverse testdata to ESC DMA Address register.
- Write word testdata to ESC DMA Count register.
- Write word inverse testdata to word 0 in Lance buffered memory.
- Read word inverse testdata from ESC DMA Address register.
- Read word testdata from ESC DMA Count register.
- Compare and print error message if miscompare.

*Possible Error Messages*

```
%s Register data compare error
   address = %X
   expected = %X
   observed = %X

char esc_d_ctl_err_txt[]=
 ESC SCSI DVMA Control

char esc_d_adr_err_txt[]=
 ESC SCSI DVMA Address

char esc_d_cnt_err_txt[]=
 ESC SCSI DVMA Count

char esc_reg_b2b_err_txt[]=
 ESC back-to-back access
```

Subtest: FAS236 Registers

- ID: 126.7
- Level: 17
- Attributes: Test Module

FAS configuration register #1 write/write/read test.

- For testdata = 0x1, 0x2, 0x4, ..., 0x40, 0x80 {
- Write byte testdata to FAS configuration register #1.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #1.
- Compare and print error message if miscompare.
- testdata = 0xff.
- Write byte testdata to FAS configuration register #1.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #1.
- Compare and print error message if miscompare.

FAS configuration register #2 write/write/read test.

- For testdata = 0x1, 0x2, 0x4, ..., 0x40, 0x80 {
- Write byte testdata to FAS configuration register #2.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #2.
- Compare and print error message if miscompare.
- testdata = 0xff.
- Write byte testdata to FAS configuration register #2.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #2.
- Compare and print error message if miscompare.

FAS configuration register #3 write/write/read test.

- For testdata = 0x1, 0x2, 0x4, ..., 0x40, 0x80 {
- Write byte testdata to FAS configuration register #3.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #3.
- Compare and print error message if miscompare.
- testdata = 0xff.
- Write byte testdata to FAS configuration register #3.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte testdata from FAS configuration register #3.
- Compare and print error message if miscompare.

FAS configuration register #1, #2 back-to-back write / write / read / read test.

- For testdata = 0x1, 0x2, 0x4, ..., 0x40, 0x80 {
- Write byte inverse testdata to FAS configuration register #1.
- Write byte testdata to FAS configuration register #2.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte inverse testdata from FAS configuration register #1.
- Read byte testdata from FAS configuration register #2.
- Compare and print error message if miscompare.
- testdata = 0xff.
- Write byte inverse testdata to FAS configuration register #1.
- Write byte testdata to FAS configuration register #2.
- Write byte inverse testdata to byte 0 in Lance buffered memory.
- Read byte inverse testdata from FAS configuration register #1.
- Read byte testdata from FAS configuration register #2.
- Compare and print error message if miscompare.

*Possible Error Messages*

```
char fas_xfr_low_txt[]=
 Transfer count low register compare error

char fas_xfr_hi_txt[]=
 Transfer count high register compare error

char fas_config1_txt[]=
 Config register #1 compare error

 char fas_config2_txt[]=
  Config register #2 compare error

char fas_config3_txt[]=
 Config register #3 compare error

char fas_config2_b2b_txt[]=
 Config reg #1,#2 back-to-back compare error
```

Subtest: SCSI DVMA Read

- ID: 126.8
- Level: 17
- Attributes: Test Module

Program the FAS and ESC chips to do a DVMA read and verify.

- Reset the FAS-236.
- Send NOP command to FAS after doing reset.
- Set bit 3 (Chip Test Mode) of the Configuration Register #1.
- Set bit 0 (Target Test Mode) of the Test Register.
- Load the Transfer Count Register with the number of bytes to be transferred.
- Load the ESC SCSI DVMA Count register with the byte count.
- Store the virtual address in ESC SCSI DVMA Address register.
- Make this an INVALID virtual address.
- Store the test bytes into the FIFO register.
- Issue a 'SEND DATA W/DMA command (0xA2 into command register.
- Turn on DMA and set direction in ESC SCSI DVMA Control register.
- Wait 1uS to allow for internal command synchronization.
- Wait for the bytes to be transferred via DMA control (TCZERO).
- Check for timeout.
- Verify the transfered bytes by reading the FIFO register.
- Reset the FAS-236 to release the SCSI bus.

*Possible Error Messages*

```
%s
   address = %X
   expected = %2X
   observed = %2X

char fas_dvma_xfr_timeout_err_txt[]=
 Transfer timeout waiting on SCSI DVMA Control register

char fas_dvma_read_fifo_err_txt[]=
 FIFO data compare error

char fas_dvma_write_mem_err_txt[]=
 Memory data compare error
```

Subtest: SCSI DVMA Write

- ID: 126.9
- Level: 17
- Attributes: Test Module

Program the FAS and ESC chips to do DVMA write and verify.

- Reset the FAS-236.
- Send NOP command to FAS after doing reset.
- Set bit 3 (Chip Test Mode) of the Configuration Register #1.
- Set bit 0 (Target Test Mode) of the test register.
- Load the Transfer Count Register with the number of bytes to be transferred.
- Load the ESC SCSI DVMA Count register with the byte count.
- Store the virtual address in ESC SCSI DVMA Address register.
- Store the test data bytes directly into the FIFO register.
- Issue a 'RECEIVE DATA W/DMA command (0xAA into command register).
- Turn on DMA and set direction in ESC SCSI DVMA Control register.
- Wait 1uS to allow for internal command synchronization.
- Wait for the bytes to be transferred via DMA control (TCZERO).
- Flush the SBI Write Buffers.
- Now send the DRAIN command to the ESC.
- Check that the ACTIVE0 and ACTIVE1 bits in SCSI DVMA Control register are cleared, indicating the data transfer is complete.
- Check for timeout.
- Verify the transfered bytes by reading main memory.
- Reset the FAS-236 to release the SCSI bus.

*Possible Error Messages*

```
%s
   address = %X
   expected = %2X
   observed = %2X

char fas_dvma_xfr_timeout_err_txt[]=
 Transfer timeout waiting on SCSI DVMA Control register

char fas_dvma_read_fifo_err_txt[]=
 FIFO data compare error
```

```
char fas_dvma_write_mem_err_txt[]=
 Memory data compare error
```

Subtest: SCSI DVMA Write Error

- ID: 126.10
- Level: 17
- Attributes: Test Module

Program the FAS and ESC chips to do a DVMA write to an invalid address and verify that the correct error condition occurs.

- Reset the FAS-236.
- Send NOP command to FAS after doing reset.
- Set bit 3 (Chip Test Mode) of the Configuration Register #1.
- Set bit 0 (Target Test Mode) of the Test Register.
- Load the Transfer Count Register with the number of bytes to be transferred.
- Load the ESC SCSI DVMA Count register with the byte count.
- Store the INVALID virtual address in ESC SCSI DVMA Address register.
- Store the test data bytes directly into the FIFO register.
- Issue a 'RECEIVE DATA W/DMA command (0xAA into command register).
- Turn on DMA and set direction in ESC SCSI DVMA Control register.
- Wait 1uS to allow for internal command synchronization.
- Wait for error bit (DMA_ERR) to be set in SCSI DMA Control register.
- Check for timeout.
- Reset the FAS-236 to release the SCSI bus.

*Possible Error Messages*

```
%s
   address = %X
   expected = %2X
   observed = %2X

char fas_dvma_xfr_timeout_err_txt[]=
 Transfer timeout waiting on SCSI DVMA Control register

char fas_dvma_read_fifo_err_txt[]=
 FIFO data compare error
```

```
char fas_dvma_write_mem_err_txt[]=
 Memory data compare error
```

## 2.6 System Reconfiguration

At this point, POST performs the following tasks to boot the system.

- Selects the System Master
- Configures and interleaves the memory
- Prints the system and memory display (in verbose mode only)
- Builds structures to pass to the OpenBoot firmware
- Copies the system IDPROM into NVRAM on each board
- Selects an MQH to do the demap replies
- Sets final LED values
- Scrubs the memory
- Initializes all CPUs and MXCCs for booting

Finally, POST completes execution by transferring control to the OpenBoot™ program.

# *Sample POST Output*

This appendix shows the output that POST typically displays when it is run in diag mode on a SPARCserver 1000 system.

```
1B>
BIST Status = 00000001 Signatures - CPU = 456E34F1 MXCC = 23AA97E6
1B>map16 test
1A>
BIST Status = 00000001 Signatures - CPU = 456E34F1 MXCC = 23AA97E6
1B>
        **** SPARCserver_1000 MP POST Rev 7 ****

1A>map16 test
1B>EPROMs Test
1B>    EPROM path Test
1A>
        **** SPARCserver_1000 MP POST Rev 7 ****

1B>    EPROM checksum Test
1A>EPROMs Test
1A>    EPROM path Test
1A>    EPROM checksum Test
1B>LEDs Test
1B>    WALK LED Test
1B>Serial Ports Test
1B>    Port A Register Testj
!"#$%&'()*+,
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
1A>Serial Ports Test
1B>    Serial Port B Loopback Testj
```

```
1A>    Port A Register Test
!"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
1B>    Mouse Loopback Test
1A>    Serial Port B Loopback TestJ
1B>NVRAM/TOD Test
1A>Keybd/Mouse Test
1B>Basic CPU Test
1A>    Keyboard Loopback Test
1B>    FPU Register Test
1A>    Mouse Loopback Test
1B>    FPU Functional Test
1A>NVRAM/TOD Test
1A>Basic CPU Test
1B>    MMU TLB Test
1A>    FPU Register Test
1B>    Instruction Cache Tags Test
1A>    FPU Functional Test
1A>    MMU TLB Test
1A>    Instruction Cache Tags Test
1B>    Instruction Cache Ram Test
1A>    Instruction Cache Ram Test
1B>    Data Cache Tags Test
1A>    Data Cache Tags Test
1B>    Data Cache Ram Test
1A>    Data Cache Ram Test
1B>    Store Buffer Tags Test
1B>    Store Buffer RAM Test
1B>    Store Buffer Functional Test
1A>    Store Buffer Tags Test
1B>    MXCC Registers Test
1A>    Store Buffer RAM Test
1A>    Store Buffer Functional Test
1A>    MXCC Registers Test
1B>    Init MXCC Regs
1B>Ecache Test
1B>    Setting Cache Size
1B>    Ecache Tags Test
1A>    Init MXCC Regs
1A>Ecache Test
1A>    Setting Cache Size
1A>    Ecache Tags Test
1B>    Ecache SRAM Test
1A>    Ecache SRAM Test
1B>    Ecache Enable
```

```
1B>    Clear CC SRAM
1A>    Ecache Enable
1A>    Clear CC SRAM
1A>BW0 Regs Test
1A>    C_O BW
1A>    BW Registers Test
1A>    Timers and Interrupts Test
1A>    BW Tag RAM 6N Test
1B>BW0 Regs Test
1B>    C_O BW
1B>    BW Registers Test
1B>    Timers and Interrupts Test
1B>    BW Tag RAM 6N Test
1A>C0 MQH Test
1A>    C_0 BW,MQH
1A>    MQH Registers Test
1A>    MQH Initialization
1A>    Enable ECC
1A>    Memory Test
1A>
 *** Skip to Next Subtest ***

1A>    Config Memory Available
1A> Config Board = 512MB, Config Total = 512MB
1A>C0 IOC Test
1A>    C_0 BW,IOC
1A>    IOC Registers Test
1A>    IOC XDBus Tags Test
1A>    IOC Sbus Tags Test
1A>    IOC Cache RAM Test
1A>C0 SBI Test
1A>    SBI Initialization
1A>    SBI Registers Test
1A>    SBI Initialization
1A>    SBus Interrupts Test
1A>C0 SBUS Cards Test
1A>    SBI Initialization
1A>    Checking for SBUS cards
1A>Board 1 Slot 0 occupied
1A>Board 1 Slot 3 occupied
1A>C0 XDBus Timing Test
1A>    C_0 BW
1A>    Compute XDBus Frequency
1A>Bus frequency = 33 MHz
1A>    TOD Delay
```

```
1A>C0 XPT Test
1A>    C_0 BW,IOC
1A>    XPT Read Write Test
1A>C0 BW-MQH Consistency Test
1A>    C_0 BW,MQH
1A>    BW MQH Cache Consistency Test
1A>C0 IOC-MQH Consistency Test
1A>    C_0 BW,IOC,MQH
1A>    SBus Loopback Test
1A> Testing slot 0 on board 1
1A> Testing slot 1 on board 1
1A> Testing slot 2 on board 1
1A> Testing slot 3 on board 1
1A>    IOC MQH Consistency Test
1A>C0 BW-IOC Consistency Test
1A>    C_0 BW,IOC,MQH
1A>    Cache States Test
1A>    BW IOC Consistency Test
1A>SPARC Module Board Master Test
1A>    C_0 BW,MQH
1A>    CPU and Cache Test
1A>    MMU PTP Cache Invalidation Test
1A>    MMU Stuff TLB Hit Test
1A>    MMU Table Walk Test
1A>    MMU Flush Test
1A>    MMU TLB Lock Test
1A>    MMU TLB Protection Error Test
1A>    MMU Table Walk With Parity Error Test
1A>    MMU Table Walk With ECC Error Test
1B>SPARC Module Board Slave Test
1B>    Read MQH State
1B>    C_0 BW,MQH
1B>    CPU and Cache Test
1B>    MMU PTP Cache Invalidation Test
1B>    MMU Stuff TLB Hit Test
1B>    MMU Table Walk Test
1B>    MMU Flush Test
1B>    MMU TLB Lock Test
1B>    MMU TLB Protection Error Test
1B>    MMU Table Walk With Parity Error Test
1B>    MMU Table Walk With ECC Error Test
1A>OnBoard IO Verification Test
1A>    C_0 BW,IOC,MQH
1A>    Check OnBoardIO Card0 Test
1A>    Lance Memory Test
```

```
1A>    Lance Registers Test
1A>    Lance Local Loopback Test
1A>    ESC Registers Test
1A>    FAS236 Registers Test
1A>    SCSI DVMA Read Test
1A>    SCSI DVMA Write Test
1A>    SCSI DVMA Write Error Test
1A>Bus Ring Test
1A>    Verify Bus Ring Test
1A>C0 BP Check Test
1A>    Wait for Alt
1A>    XDBus setup C_0
1A>    C0 Backplane Check Test
1A>C0 Exit LB Test
1A>    Loopback Exit C_0 Test
1A>programming MQH group addr at E1101000 to 0180000C
1A>programming MQH group addr at E1101008 to 0100000C
1A>programming MQH group addr at E1101040 to 0080000C
1A>programming MQH group addr at E1101048 to 0000000C
1A>Reading Address Decoding Registers from Hardware:
1A>b1 d0 g0 IF 0 IV 0 ssize 3
1A>b1 d0 g1 IF 0 IV 0 ssize 3
1A>b1 d0 g2 IF 0 IV 0 ssize 3
1A>b1 d0 g3 IF 0 IV 0 ssize 3
1A>total pmem 0x00020000 [pages] 0x020000000 [bytes] in 1 chunks
1A>DRAM chunk 0 base 0x00000000 size 0x00020000
1A>  (0=failed,1=passed,blank=untested/unavailable)
     (sbus 1=card present,0=card not present,x=failed)
1A>----+----+---+----+---+--+----+---+----+---+----+---+
1A>Slot|cpuA|bw0|cpuB|bw0|bb|ioc0|sbi|mqh0|mem|sbus|xd0|
1A>----+----+---+----+---+--+----+---+----+---+----+---+
1A>  1 | 1  | 1 | 1  | 1 | 1|  1 | 1 | 1  |512|1001| 1 |
1A>----+----+---+----+---+--+----+---+----+---+----+---+
1A>
1A>Memory Group Status
   (0=failed,1=passed,m=simm missing,c=simm mismatch,blank=unpopulated/unused)
1A>+---+------+------+------+------+
1A>Slot|  g0  |  g1  |  g2  |  g3  |
1A>+---+------+------+------+------+
1A>  1 |   1  |   1  |   1  |   1  |
1A>+---+------+------+------+------+
```

# POST Design Concepts

B

This appendix describes some principles on which the SPARCcenter™ 2000 and SPARCserver 1000 POST is designed. These two systems use the same POST, and the information in this appendix applies to both systems.

## B.1 Tests and Subtests

POST is a collection of diagnostics or *Tests* that examine system hardware and ensure that the system can boot successfully. Tests perform the following types of tasks: initializing and checking the hardware, acquiring resources needed during testing, and reconfiguring the hardware. Usually, each group of related components in a system has a test associated with it. Each test can contain an ordered list of *SubTests*; these subtests perform the individual tasks needed by the test. Once a test is selected to run, all the subtests associated with it execute unconditionally and in the specified order unless errors cause the test to be aborted. A subtest can be called by several tests. A test *fails* if any of its subtests fails and *passes* only if all of its subtests pass.

### B.1.1 TestIDs and SubtestIDs

A *TestID* uniquely identifies each test. Similarly, within a test, a *SubTestID* uniquely identifies a subtest. The TestID and SubTestID together uniquely identify a given subtest from all other subtests in POST.

### B.1.2 TestLists and Sequencers

Tests are grouped into ordered sequences called *TestLists*. POST executes in several phases, and each phase of POST has its own TestList. Each TestList has its own *Sequencer*, which executes each test in the TestList systematically. The Sequencer records the status of the current test in a global structure called the *TestStatesArray*. This table-driven structure allows tests to be added or removed easily while simplifying control flow.

### B.1.3 Test Levels and Error Levels

Whenever a processor executes POST in normal mode, it does so at a given *Level*, which is an 8-bit number. The higher the level, the greater the number of tests and the more exhaustive the tests. Every test is assigned a *TestLevel*, and a given test can run only if its level is at or below the current level of POST

When POST executes in error mode, it uses the *ErrorLevel* variable instead of the Level variable used in normal mode. This variable allows the selection of specific initialization tests that are executed when POST encounters unexpected errors.

### B.1.4 Test Design

Tests are designed and organized on the following principles:

- Each component should test itself as much as possible.
- Each component should be tested before it is used. A functional component must then be added to a pool of other working components.
- Two components, capable of testing themselves, must be allowed to interact only after exercising that capability, and their interaction must be controlled.
- As many tests as possible should deal with the normal case; tests for handling exceptions should be minimal because they are used infrequently and are hard to debug.

In the SPARCcenter 2000 and the SPARCserver 1000 system, the processor is the only component capable of testing itself as well as other components. Thus, testing begins at the processor. Using the “onion skin” method, the tests work their way outward from the processor, collecting a group of tested and functional components.

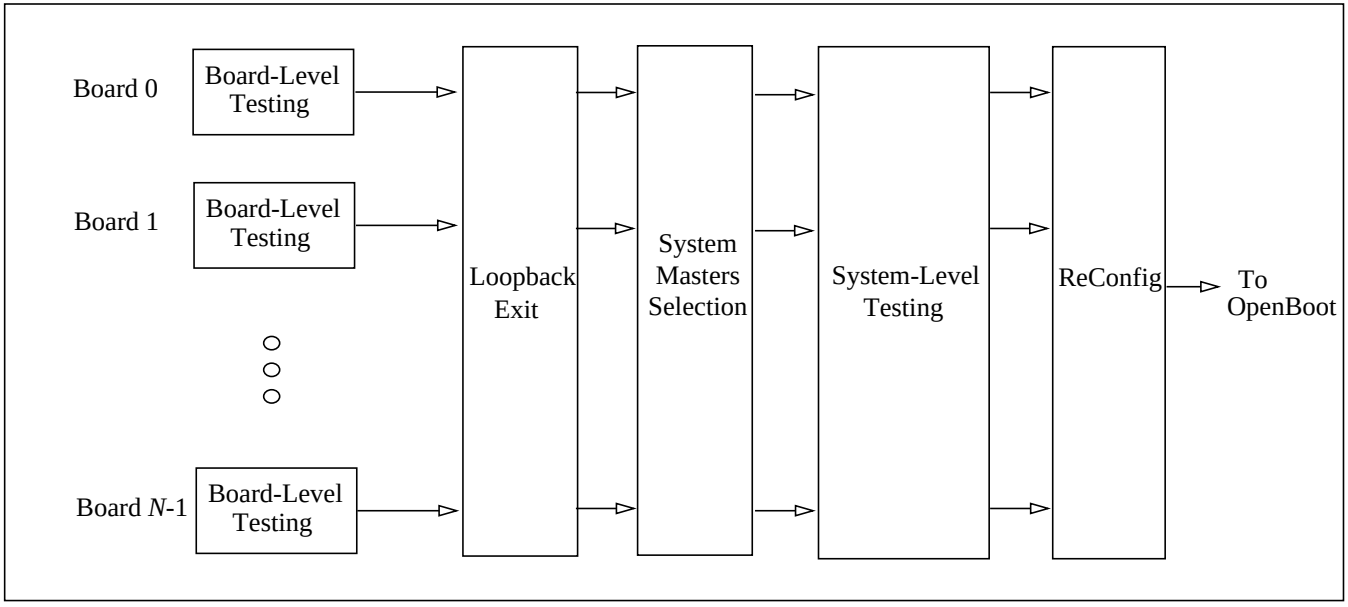
## B.2 Phases of POST

POST execution in a SPARCserver 1000 system takes place in five phases:

- Board-Level Testing
- Loopback Exit
- System Masters Selection
- System-Level Testing
- Reconfiguration.

Each phase has its own TestList and Sequencer. A Sequencer traverses its TestList, thereby determining the order in which tests are executed.

POST is a multiprocessor program; there are as many threads of control as there are processors in the system. A processor enters POST after a reset and leaves POST to hand control to the OpenBoot firmware once the machine has been tested and successfully configured. While POST is running, processors synchronize with each other at various points to coordinate testing and reconfiguration. The overall flow of control for POST is shown in the following diagram.



### *Board-Level Testing*

On reset, the hardware automatically puts each board in loopback mode, and the processors on the board coordinate to do board-level testing before exiting loopback. During this testing phase, each board in the system is logically and electrically isolated from other boards. This prevents failures on one board from corrupting other boards. A processor that fails during this phase becomes inactive.

The types of tests executed during this phase include BootBus tests, CPU module, cache controller, external RAM, and Bus Watcher tests, as well as tests that ensure the functionality of non-processor components on the board.

### *Loopback Exit*

During loopback exit, each processor verifies its connection to the backplane. Processors that pass this test synchronize and enable their boards on to the backplane; processors that fail remain inactive. The purpose of this phase is to get working boards out of loopback so that they can be tested together.

### *System Master Selection*

In this phase, the processors collectively elect system masters for each of the three configurations:

- *C0* (only XDBus0 enabled)
- *C1* (only XDBbus1 enabled)
- *C2* (both buses enabled).

These system masters are called *C0SM*, *C1SM*, and *C2SM*, respectively.

---

**Note –** The SPARCserver 1000 has only one XDBus system bus, so the only possible configuration for this server is C0. C1 and C2 do not apply.

---

### *System-Level Testing*

System-level testing is divided into three sub-phases: one for the C0 configuration, one for C1, and one for C2 (the only possible configuration for the SPARCserver 1000 is C0). Each sub-phase is run by the corresponding system master. Each system master puts all the boards into the configuration for which it is master, then tests non-processor boards, and (for the C0 and C1 configurations) the system's main memory. Next, it runs system-level tests that

include checking inter-processor interrupts, device-to-processor interrupts, and cache coherency. Finally, each system master computes the value of its configuration as the weighted sum of available resources and stores this value in specific registers of all working Bus Watcher ASICs in the system. These values are used in the reconfiguration phase of testing.

### *Reconfiguration*

In the reconfiguration phase, the configuration with the highest value is the winner. The system master of the winning configuration initializes the system to be in this configuration. Memory is configured and interleaved, and bad memory groups are excluded. POST sends the OpenBoot firmware information on functioning system resources as well as:

- The device table, which includes data on the parts that failed POST.
- The memory list.
- The failed memory SIMM group listing. These are automatically cleared by POST at power-on reset.
- A list of failed memory SIMM pages. The list is cleared during a power-on reset.
- The System Watchdog error log for each board. This error log is saved across power-on resets, and the log can even be examined if the board is returned to a repair depot.

## *B.3 Error Handling*

A processor can encounter two types of errors: *expected* and *unexpected* during any phase of POST. An expected error is defined as a test failure in which the processor's control flow is not forcibly altered. In such a case, the Sequencer marks the test as having failed, and proceeds to the next test.

An unexpected error is defined as a failure in which the processor gets reset, and hardware forcibly transfers control to location 0xFF0000000. In the case of an unexpected error, the processor runs in a special *error* mode. In this mode, the Sequencer for each phase traverses its list of tests as before, but executes only certain *initialization* subtests in a process called *replay*. When the processor reaches the test that generated the unexpected failure, it marks this test as having failed, leaves error mode, and proceeds with the normal execution sequence.

## B.4 Running POST

You can invoke POST in two ways: by using *entry points* and by using *call-back routines*. When you use an entry point to invoke POST, tests are always executed at a particular *level*. The higher the level, the greater the number of tests that are run and the more thorough the tests.

### *Entry Points*

POST has three entry points corresponding to the *power-on reset* (POR), *reboot* (RBT), and *post error* (ERR) cases.

- When entered at POR, POST runs at one of two levels determined by whether the machine's DIAG switch is set or not.
- When entered at RBT, POST can be invoked at any level under program control.
- When entered at ERR, POST tries to recover from the unexpected error it encountered during the last entry-point invocation of POST.

When POST is invoked using an entry point, it executes as a multi-thread program; the number of threads are equal to the number of processors in the system. Each processor keeps a record of where it is in its overall sequence of tests. If an error occurs, this record can tell POST or a user what a processor was attempting to do just before the error occurred.

### *Call-back Routines*

POST also implements a number of *call-back routines* used by the OpenBoot firmware and higher level programs. A processor does not keep track of where it is in the overall sequence of its tests when executing a call-back routine. As a result, unexpected (asynchronous) errors within call-back routines are *not* handled by the ERR entry-point.

# *Glossary*

**Arbitration System**

A bus arbitration system determines which processor can control the system at any instant. The arbitration system consists of circuits on the control board and the system boards.

**ASIC**

Applications specific integrated circuits (ASICs) are integrated circuits which perform specialized tasks in the system. If an ASIC (BARB, BBC, BIC, BW, BX, IOC, or SBI) fails, the entire system board must be replaced.

**Backplane Subsystem**

The backplane subsystem consists of the control board and the card cage.

**Bank**

A bank of memory consists of eight SIMMs. Each system board has space for two independent memory banks.

**BARB**

Board arbiter (BARB) ASICs are part of the bus arbitration system. BARBs are located on the system boards. See *Arbitration System*.

**BBC**

The BootBus Controller (BBC) ASIC is located on the control board. The BBC works with BBC2 ASICs on system boards to control parts of the boot process.

**BBC2**

BootBus Controller 2 (BBC2) ASICs are located on system boards. See *BBC*.

**BIC**

Every SPARCserver 1000 system board has four Bus Interface Chip (BIC) ASICs. BICs connect the board to the backplane XDBuse.

**Board**

The term board refers to the control board or to system boards. The SPARCserver 1000 system has 1 control board and up to 4 system boards.

**Board ID**

Board slot ID codes are hardwired on the card cage backplane. A system board can be moved to any card cage slot without the need for jumper changes.

**BootBus**

The BootBus connects the OpenBoot PROM set on a system board to the SPARC processor modules on that same board.

**BW**

The system board has two Bus Watcher (BW) ASICs (two for each processor) to convert XDBus signals to higher-speed processor module bus signals.

**C0, C1, and C2 Configurations**

**Note –** The SPARCserver 1000 has only one XDBus system bus, so C0 is the only possible configuration that applies. However, this version of POST was also also designed to operate on the SPARCcenter 2000, which permits the C1 and C2 configurations.

In the C0 configuration, POST configures the system to operate with XDBus0 operational and XDBus1 disabled. All devices connected to XDBus1 are disabled so that they cannot interact with or affect the operation of the enabled devices on XDBus0.

In the C1 configuration, POST configures the system to operate with XDBus1 operational and XDBus0 disabled. All devices connected to XDBus0 are disabled so that they cannot interact with or affect the operation of the enabled devices on XDBus1.

In C2 configuration POST configures the system to operate with both XDBus0 and XDBus1 enabled. All devices connected to both these buses are enabled and functional.

**Cache**

Memory caches are located near various buses on the system board.

**CARB**

One Central Arbiter (CARB) ASICs on the control board are part of the multiprocessor arbitration system.

**Card Cage**

The SPARCserver 1000 card cage has 3 system board slots.

**Card Slot**

A SPARCserver 1000 system board has four SBus card slots.

**Clock Generation**

System clocks are generated on the control board.

**Control Board**

The control board generates system clocks and is part of the multiprocessor arbitration system. The control board is located outside the card cage.

**IOC**

The I/O Cache (IOC) ASIC on the system board controls movement of data to and from the SBus card slots.

**Key Switch**

The key switch, located behind the front panel, controls the AC supplies and the modes of system operation.

**LED Indicators**

*System Cabinet*

LED indicators are on the front panel of the system cabinet. Ideally, the left and right LEDs should be on and the middle LED should be off; however, the system can still run (reliably) if all three LEDs are on.

*Table G-1* Front Panel LED System Status

| LED Position | Condition |
|---|---|
| Left (green) | On — DC power supply is receiving AC current |
| Middle (yellow) | On — (first 60 seconds of AC power) self tests are running<br>Off — (after self-tests end) no hardware failures<br>On — (after self-tests end) hardware failure was detected |
| Right (green) | Off – (first 60 seconds of AC power) self tests are running<br>On — (after self-tests end) system is running<br>Off — (after self-tests end) system cannot run; repair is needed |

*System Board*

Every system board has 10 LEDs on the back panel. Two green LEDs (top positions) indicate the presence of one or two SPARC processor modules (or none). The remaining eight yellow LEDs (lower positions) are activated by software commands to the system's status registers and no particular meanings have been assigned to these 8 LEDs. In normal operation, the eight lower LEDs on the system master board will constantly turn on and off, in a cyclical pattern, while corresponding LEDs on the remaining system boards will be off.

**MQH**

The Memory Queue Handler (MQH) ASIC on the system board provides the interface between the system board SIMMs and the backplane buses.

**MXCC**

The Module XBus Cache Controller (MXCC) ASIC is located on the SPARC module and controls the flow of data to and from the XDBus.

**NVRAM**

Non-volatile, random access memory.

**NVSIMM**

The non-volatile SIMM (NVSIMM) is a battery-backed SIMM. Battery current is shared in a group of NVSIMMs, in the event that a single battery fails.

**Power-on Position**

This is one of four positions of the front panel key switch. When the key is in this position, turning on AC power (at the main AC breaker on the back of the server cabinet) or pressing the reset switch (on the inside of the front panel) causes POST diagnostics to run, followed by booting of the system.

**Reset Switch**

This switch is located behind the front panel.

**SBI**

The SBus Interface (SBI) ASIC converts signals between the SBus and the higher-speed XDBus. There is one SBI on each system board.

**SBus Card**

SBus cards provide external interfaces and optional features to the system. There are four SBus connectors on a SPARCserver 1000 system board.

**Secure Position**

This is one of four positions on the system key switch. In this position, the reset switch is disabled and the Stop-A keyboard combination is disabled.

**SIMM**

There are several types of single in-line memory modules (SIMMs). SIMMs are socketed on the system board for easy replacement.

**TLB**

The SuperSPARC™ chip translation buffer.

**TODC (Time of Day Clock)**

TODC contains the system date and time. Every system board has a TODC, but only the TODC on the master board is used.

**XDBus**

The XDBus is the SPARCserver 1000's system bus.

# *Index*

## Reader Comments

We welcome your comments and suggestions to help improve the *SPARCserver 1000 POST User's Guide*, part number *801-2916-10*. Please take time to let us know what you think about this manual.

- The information was well documented and easy to follow.

| Strongly Agree | Agree | Disagree | Strongly Disagree | Not Applicable |
|---|---|---|---|---|
| ❑ | ❑ | ❑ | ❑ | ❑ |

Comments _____

- The information provided in the manuals was complete.

| Strongly Agree | Agree | Disagree | Strongly Disagree | Not Applicable |
|---|---|---|---|---|
| ❑ | ❑ | ❑ | ❑ | ❑ |

Comments _____

- The information I needed was easy to find.

| Strongly Agree | Agree | Disagree | Strongly Disagree | Not Applicable |
|---|---|---|---|---|
| ❑ | ❑ | ❑ | ❑ | ❑ |

Comments _____

- The manuals were useful to me.

| Strongly Agree | Agree | Disagree | Strongly Disagree | Not Applicable |
|---|---|---|---|---|
| ❑ | ❑ | ❑ | ❑ | ❑ |

Comments _____

- Do you have additional comments about the *SPARCserver 1000 POST User's Guide*?

_____

_____

_____

_____

Name: _____

Title: _____

Company: _____

Address: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

# BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 1 MOUNTAIN VIEW, CA

POSTAGE WILL BE PAID BY ADDRESSEE

SUN MICROSYSTEMS, INC.
Attn: SMCC Technical Publications
MS MTV15-42
2550 Garcia Avenue
Mt. View, CA 94043