System Power-on Self Test Manual

SOLBOURNE COMPUTER, Inc.
1900 Pike Road
Longmont, Colorado 80501
(303) 772-3400

Part Number: 101486-AC

August 1989

Preface

This manual details all information pertaining to hardware diagnostics on a Solbourne Computer.

The manual is divided into two sections and five appendices, as follows:

Section 1 - Introduction
This section introduces the hardware diagnostics and explains how to use the information contained in this manual.

Section 2 - Series4 Test Descriptions
All test descriptions and error codes for the Series4 Solbourne products are presented in this section.

Section 3 - Series5 Test Descriptions
All test descriptions and error codes for the Series5 Solbourne products are presented in this section.

Appendix A - Series4 Considerations
Information specific to the Series4 implementation is given in this appendix.

Appendix B - Series5 Considerations
This appendix presents information specific to the Series5 implementation.

Table of Contents

SECTION 1: INTRODUCTION

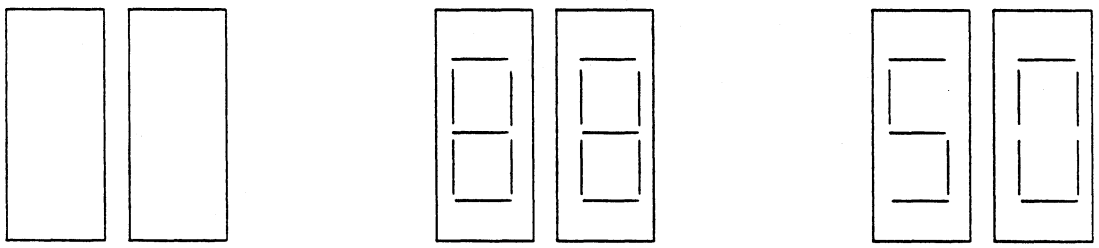
1.1 Introduction

This manual describes the boot ROM resident diagnostic code that is executed during power-up reset.

When a Series4 is connected to power, the System Board generates a reset signal that resets each board in the system and causes CPU Board(s) to begin executing Boot ROM code at the "reset" trap vector (address 0). The first code to be executed in the Boot ROM is the System Power-On Self Tests. The system self test diagnostic routines must execute to completion, without error, before the system can bootstrap any stand alone program or OS/MP.

After a system reset occurs, there is a short delay before any prompt is displayed on the console device. During this period of time the System self-test is executing. If no prompt appears on the console after several minutes, the two seven-segment light emitting diodes (LEDs) on the CPU Board(s) should be examined to determine the status of the system.

The LEDs may be viewed by removing the louvered panel on the top of the workstation, and looking down through the perforated sheet metal into the top of the chassis. Examples of three sets of the two seven-segment LED displays are shown in Example 1-1.



Example 1-1. Example of LEDs on CPU Board

In Example 1-1, the set of LEDs on the left are blank, the center set displays the number 88, and the set on the right displays the number 50.

During the power-up sequence, the LEDs change state as the self test progresses. On reset, the first value written to the LEDs is 0x00. As the self test is executing, the number of the test to be executed is written to the LEDs just before the test is invoked. If all the tests execute without error, the LEDs transition through states 00, 01, 02, etc. up to and including the last test number. When the self test is completed, there are additional LED states displayed that indicate the status of the system while it is being initialized to execute stand alone programs (see Appendix F). If the LEDs become locked on a particular state during self test execution, this means that a catastrophic failure has occurred during the test indicated by the last LED state.

When a self test program fails, error information is displayed in on the LEDs. The error information consists of the test number and a unique error code that identifies the failure. Since the LEDs cannot display both the test number and error code simultaneously, the test number and error code must be displayed in a cyclic fashion on the LEDs.

Figure 1-1 illustrates how the error information is displayed.

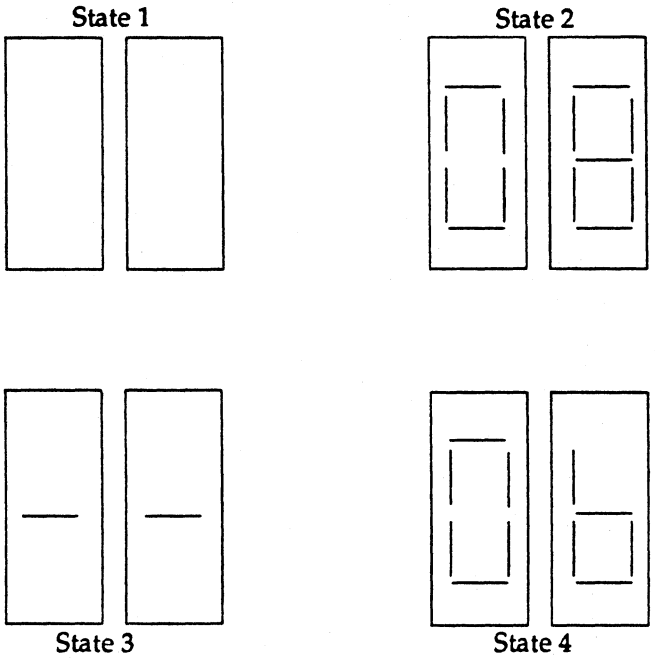


Figure 1-1. Example Display of Error Information

The states from Figure 1-1 are explained below:

- **State 1** - This marks the beginning of the cycle with both LEDs displaying blanks.
- **State 2** - The test number of the failing test is displayed in both LEDs.
- **State 3** - Both LEDs display dashes that indicate the separation of the test number from the error code.
- **State 4** - A unique error code that identifies the failure is displayed.

Figure 1-2 shows that an unexpected exception occurred. The number following the "- -" block represents the exception (trap) type a data access exception. See the *SPARC Architecture Manual* for additional information on exception trap types.

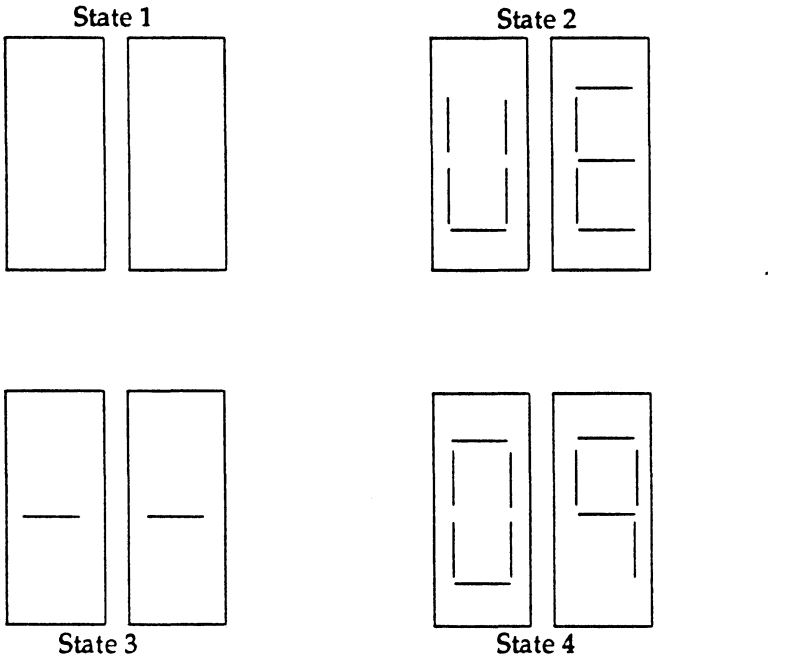


Figure 1-2. Example of Unexpected Exception

For the Series4 CPU, there are the following additional error codes:

00 -- 00	Double trap during tests 1 through 4.
00 -- 01	Double trap during tests 5 through 3X.
00 -- 02	Double trap occurred; no vector defined.

For the Series5 CPU, there are the following additional error codes:

00 -- 00	Double trap occurred; DGRAM not initialized.
00 -- 01	Watchdog trap occurred; no reset vector defined.
00 -- 02	Double trap occurred; no reset vector defined.
00 -- 03	Watchdog and double trap occurred; no reset vector defined.
00 -- 04	Cold start, cannot clear MMCR<CS> bit.

When the ROM monitor program or a stand alone program is checking for input, a dash (-) is alternately displayed between the two LEDs, as shown in Figure 1-3.

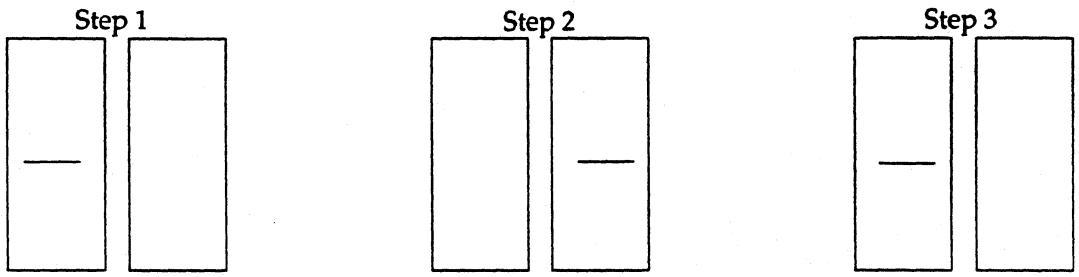


Figure 1-3. LEDs State When Checking for Input

When OS/MP is idle, a small "o" moves around from one corner to another in the LEDs, as shown in Figure 1-4.

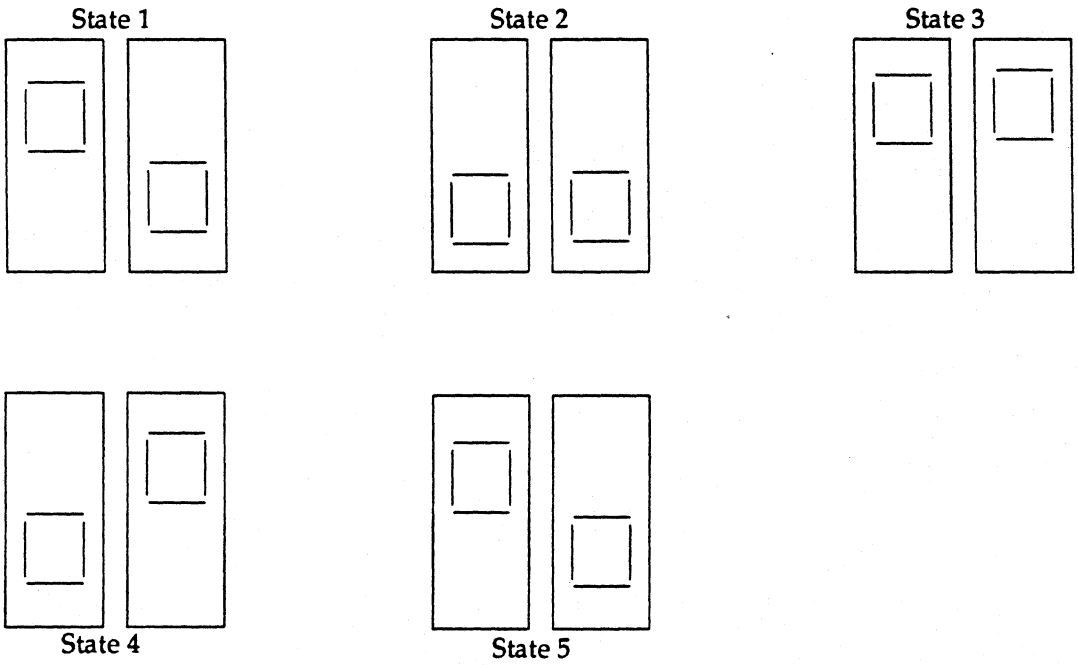


Figure 1-4. Example of LEDs State When OS/MP Is Idle

1.2 Multiprocessor Configuration Self Tests

In Solbourne's master-slave multiprocessor implementation, the power-on self-test is performed in the following sequence:

1. When power is turned on, all installed processors execute the first half of the self test concurrently.
2. The processors determine which CPU board is the master, as defined by the ROM environment variable MASTER (e.g., MASTER=1 for slot one on the Kbus). If the MASTER environment variable has not been set or points to an empty slot, the CPU in the lowest numbered slot will assume mastership.
3. Once the master is defined, the master CPU Board finishes its portion of the self test, while the slave CPU boards enter an idle loop. The master then directs each slave to finish their portion of the self test. The slaves continue to execute their self tests in descending slot order, starting with the slave CPU in the highest slot number. When the slaves complete the self test, they return to their idle loop. When the slaves are in their idle loop the LEDs display the states shown in Figure 1-3. The states shown in this figure are normal, they do not represent an error code.

   In state 4 of Figure 1-5, a 6 is displayed in the LED on the right. This number represented the number of the Kbus slot occupied by the slave CPU Board.

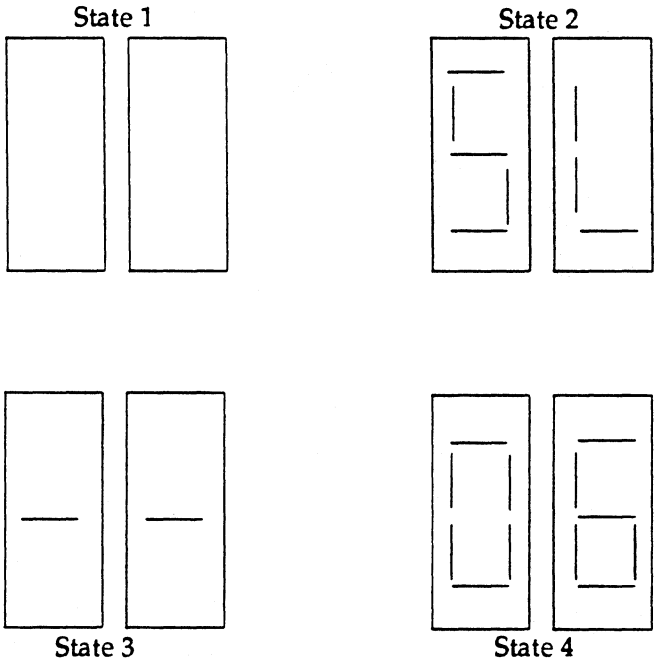


Figure 1-5. Normal Multiprocessor Slave States

4. The master CPU continues to the ROM prompt after all the slave CPU Boards have been directed to complete the self test and have reported their status back to the master.

Section 2: Series4 Test Descriptions

2.1 Introduction

This section describes the system power-on self-tests that are used on Solbourne Series4 systems. In the these test descriptions, all test numbers and error codes are represented in two digit hex values.

2.2 Test 01 - Bootrom Checksum Test

This test computes the checksum on the contents of the four, 27C512 EPROMS on the CPU Board that are used to contain the boot code and data. The expected checksum is burned into the EPROM when the roms are programed during manufacturing.

There are eight, one byte checksums that are computed. Checksum 0 is generated by summing every eighth byte starting at location 0 in the bootroms. Checksum 1 is generated by summing every eighth byte starting at location 1 in the bootrom, and so on.

Legal error codes for this test are:

```
00 - Checksum 0 incorrect
01 - Checksum 1 incorrect
02 - Checksum 2 incorrect
03 - Checksum 3 incorrect
04 - Checksum 4 incorrect
05 - Checksum 5 incorrect
06 - Checksum 6 incorrect
07 - Checksum 7 incorrect
```

2.3 Test 02 - Diagnostic RAM Addressing and Data Test

This is an addressing and data test for the diagnostic RAM on the CPU Board. The diagnostic RAM a two Kbyte static RAM which is accessed through alternate space (ASI) 0x38 (see H/W description), and responds to every eighth address in the range 0 through 0x3ff8 inclusive.

The test program performs a moving inverse test algorithm (see Appendix A for information on the moving inverse test algorithm).

Legal error codes for this test are:

```
01 - Data error on first forward pass read
02 - Data error on second forward pass read
03 - Data error on first reverse pass read
04 - Data error on second reverse pass read
```

2.4 Test 03 - Interrupt Registers Test

This is a write/read test of the interrupt registers. There are four test cases, one for each register tested. The registers tested are: Device ID Register (DIR), Interrupt Priority Register (IPR),

Interrupt Transmit Register (IXR), and the Interrupt Pending Vector Register (IPV). For each register tested, data patterns consisting of all zeroes, all ones, walking 1 and walking 0 are written to the register under test and read back and verified.

Legal error codes for this test are:

```
01 - DIR register write read error
02 - IPR register write read error
03 - IXR register write read error
04 - IPV register write read error
```

2.5 Test 04 - Directed Interrupt Test

This test verifies that the CPU Board interrupt logic can send a directed interrupt to itself. There are two test cases:

1. Case 1 verifies that directed interrupts can be transmitted and received
2. Case 2 verifies that the interrupt receiver priority level (set in the IPR register) effectively inhibits interrupts from being received.

The test uses the CPU's own board address (read from the BID register) as the target device to which the interrupt is directed. During the transmission and receiving of the directed interrupts, the basic operation of the ITXC and IRXC register is verified. (see the Kbus Interrupt Specification for more information on the interrupt transmit/receive protocol).

Case 1 sets the IPR to 0 then walks a 1 bit across the vector field of the transmitted interrupt and verifies that all interrupts are received.

Legal error codes for test 04, case 1 are:

```
01 - Interrupt was never acknowledged (ITXC<gone> bit not set).
02 - Interrupt was acknowledged but no incorrect interrupt
     type was generated.
03 - Interrupt occurred, but the IRXC<P> bit was not set.
04 - Interrupt occurred, but the IPV register did not
     contain the transmitted vector.
```

Case 2 sets the IPR to 0xff then walks a 0 bit across the vector field of the transmitted interrupt and verifies that no interrupts are received.

Legal error code for test 04, case 2 is:

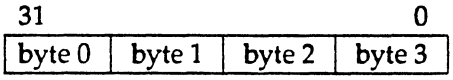
```
05 - Interrupt was acknowledged (ITXC<gone> bit set).
06 - Unexpected interrupt was generated
```

2.6 Test 05 - Control-Data Bus Test

This test reads the quiescent (undriven) state of the CPU's 32-bit, control data bus 64 Kbyte times and verifies that the bus is floats high (all ones).

A common cause of failure for this test is that one of the bootrom outputs is sinking too much current and pulling the control data bus to the low state. The test examines each eight-bit field of the 32-bit bus and reports errors using the following convention:

Control data bus format



Error encoding convention:

```
01 - byte 3 corrupted
02 - byte 2 corrupted
03 - bytes 2 and 3 corrupted
04 - byte 1 corrupted
05 - bytes 1 and 3 corrupted
06 - bytes 1 and 2 corrupted
07 - bytes 1, 2, and 3 corrupted
08 - byte 0 corrupted
09 - bytes 0 and 3 corrupted
0a - bytes 0 and 2 corrupted
0b - bytes 0, 2, and 3 corrupted
0c - bytes 0 and 1 corrupted
0d - bytes 0, 1 and 3 corrupted
0e - bytes 0, 1 and 2 corrupted
0f - bytes 0, 1, 2 and 3 corrupted
```

2.7 Test 06 - Control Registers Test

This test verifies that the MMCR and PDEP registers can be written and read. Aside from the interrrupt registers (see test 3) these are the only other two registers that are write-readable.

The MMCR<IA>, MMCR<VIN>, and MMCR<PIN> bits are read only and are not written during the test. All 32 bits of the PDEP register are tested.

Legal error codes for this test case are:

```
01 - MMCR write/read error
02 - PDEP register write/read error
```

2.8 Test 07 - TLB Instruction/Data Uniqueness Test

This test verifies that the instruction portion of the TLB is unique from the data portion of the TLB. SPARC ASI bit 1 determines which portion of the TLB is being accessed: If ASI<1> is zero, then the instruction portion is accessed, if ASI<1> is one, then the data portion is accessed. To access the instruction portion of the TLB, ASI 0x1C is used and to access the data portion of the TLB, ASI 0x1e is used. Note that when the TLB is written, not only does the page entry data get written, the TLB tags and status are written as well.

The test writes a different data pattern to the first location of each portion of the TLB, then reads reads each location to verify that the two portions are truly unique. The TLB tag and status information is obtained by reading the TIR register (see Appendix A for TIR information).

The first part of the test writes the instruction TLB followed by the data TLB, then reads the instruction TLB followed by the data TLB.

The legal error codes for test 07, case 1 are:

```
01 - The instruction TLB physical address field does not contain
     the data that was written.
02 - The instruction TLB tag and status field does not contain
     the data that was written.
03 - The data TLB physical address field does not contain the
     data that was written.
04 - The data TLB tag and status field does not contain the data
     that was written.
```

The second part of the test writes the data TLB followed by the instruction TLB, then reads the data TLB followed by the instruction TLB.

The legal error codes for this test 07, case 2 are:

```
05 - The data TLB physical address field does not contain the
     data that was written.
06 - The data TLB tag and status field does not contain the data
     that was written.
07 - The instruction TLB physical address field does not contain
     the data that was written.
08 - The instruction TLB tag and status field does not contain
     the data that was written.
```

2.9 Test 08 - Instruction TLB RAM Addressing and Data Test

This is a test of the physical address, TAG, and status fields of the instruction TLB rams. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for information on the moving inverse test algorithm).

The legal error codes for test 08 are:

```
01 - Data error on first forward pass read
02 - Tag or status error on first forward pass read
03 - Data error on second forward pass read
04 - Tag or status error on second forward pass read
05 - Data error on first reverse pass read
06 - Tag or status error on first reverse pass read
07 - Data error on second reverse pass read
08 - Tag or status error on second  reverse pass read
```

2.10 Test 09 - Data TLB RAM Addressing and Data Test

This is a test of the physical address, TAG, and status fields of the data TLB rams. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for information on the moving inverse test algorithm).

The legal error codes for test 09 are:

```
01 - Data error on first forward pass read
02 - Tag or status error on first forward pass read
03 - Data error on second forward pass read
04 - Tag or status error on second forward pass read
05 - Data error on first reverse pass read
06 - Tag or status error on first reverse pass read
07 - Data error on second reverse pass read
08 - Tag or status error on second  reverse pass read
```

2.11 Test 0a - TLB Tag Comparitors Test

This is a test of the TLB match detection logic. The test program loads a series of patterns into the tag portion of the TLB then performs a series of TIR reads to verify that the TLB match comparitor works correctly (see Appendix A for more information on TIR reads).

There are four cases for test 0a, as follow:

Case 1 - TLB tag set to walking 1 pattern with logical address set to zero.

The legal error codes for test 0a, case 1 are:

```
01 - TLB comparitor match error using instruction TLB.
02 - TLB comparitor match error using data TLB.
```

Case 2 - TLB tag set to walking 0 pattern with logical address set to all ones.

The legal error codes for test 0a, case 2 are:

```
03 - TLB comparitor match error using instruction TLB.
04 - TLB comparitor match error using data TLB.
```

Case 3 - TLB tag set to zero with logical address set to walking 1 pattern.

The legal error codes for test 0a, case 3 are:

```
05 - TLB comparitor match error using instruction TLB.
06 - TLB comparitor match error using data TLB.
```

Case 4 - TLB tag set to all ones with logical address set to walking zero pattern.

The legal error codes for test 0a, case 4 are:

```
07 - TLB comparitor match error using instruction TLB.
08 - TLB comparitor match error using data TLB.
```

2.12 Test 0b - Cache RAM Bank Uniqueness Test

This test verifies the the cache RAM bank selection mechanism works. The cache RAM bank is selected on the basis of logical address bit 2. The test also verifies that byte, half-word, word and double-word loads and stores to the cache can be performed. It is verified for each access type, that the data is placed in the correct byte/half-word/word/double-word position in the cache.

This test uses a special test feature of the CPU Board which allows caches accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to

memory). This test performs all cache accesses with the MMCR<VDCI> bit set (see Appendix A for more information on the function of the MMCR<VDCI> bit).

This test writes patterns of various sizes into the first eight bytes (addresses 0-7) of the cache. The seqeunces of pattern writes and reads and associated error codes are shown below:

Initial state: Address 0 written with 0x55555555, address 4 written with 0xaaaaaaaa.

Byte:	0	1	2	3	4	5	6	7
Data:	55	55	55	55	aa	aa	aa	aa

```
Error code 01 - word read at address 0 is not 0x55555555
Error code 02 - word read at address 4 is not 0xaaaaaaaa
```

Second state: Address 0 written with 0xaaaa, address 4 written with 0x5555.

Byte:	0	1	2	3	4	5	6	7
Data:	aa	aa	55	55	55	55	aa	aa

```
Error code 03 - word read at address 0 not 0xaaaa5555
Error code 04 - word read at address 4 not 0x5555aaaa
Error code 05 - double byte read at address 2 not 0x5555
Error code 06 - double byte read at address 6 not 0xaaaa
```

Third state: Address 0 and 7 written with 0x55, address 3 and 4 written with 0xaa.

Byte:	0	1	2	3	4	5	6	7
Data:	55	aa	55	aa	aa	55	aa	55

```
Error code 07 - word read at address 0 not 0x55aa55aa
Error code 08 - word read at address 4 not 0xaa55aa55
Error code 09 - double byte read at address 0 not 0x55aa
Error code 0a - double byte read at address 4 not 0xaa55
Error code 0b - byte read at address 0 not 0x55
Error code 0c - byte read at address 4 not 0xaa
Error code 0d - byte read at address 1 not 0xaa
Error code 0e - byte read at address 5 not 0x55
Error code 0f - byte read at address 2 not 0x55
Error code 10 - byte read at address 6 not 0xaa
Error code 11 - byte read at address 3 not 0xaa
Error code 12 - byte read at address 7 not 0x55
```

Fourth state: Address 0 written with 0xaaaaaaaa55555555 (double word write).

Byte:	0	1	2	3	4	5	6	7
Data:	aa	aa	aa	aa	55	55	55	55

```
Error code 13 - double word read at address 0, first word not 0xaaaaaaaa
Error code 14 - double word read at address 0, second word not 0x55555555
```

2.13 Test 0c - Atomic Load/Store Cache Test

This test verifies that the SPARC atomic load/store instruction "ldstub" works correctly. This instruction performs the following atomic actions: Moves a byte from memory into a register (load portion) then re-writes the same byte in memory to all ones. This test pre-loads the byte at address 0 with 0x55, then performs the ldstub instruction. Following the ldstub instruction, the byte at address 0 is re-read and verfied to contain 0xff.

This test uses a special test feature of the CPU Board which allows caches accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to memory). This test performs all cache accesses with the MMCR<VDCI> bit set (see Appendix A for more information on the function of the MMCR<VDCI> bit).

The legal error codes for test 0c are:

```
01 - load portion of ldstub instruction did not read 0x55
02 - store portion of ldstub instruction did not write 0xff
```

2.14 Test 0d - Cache RAM Addressing and Data Test

This is an addressing and data test for the Cache Data RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for information on the moving inverse test algorithm).

This test uses a special test feature of the CPU Board which allows caches accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to memory). This test performs all cache accesses with the MMCR<VDCI> bit set (see Appendix A for more information on the function of the MMCR<VDCI> bit).

The legal error codes for test 0d are:

```
01 - Data error on first forward pass read
02 - Data error on second forward pass read
03 - Data error on first reverse pass read
04 - Data error on second reverse pass read
```

2.15 Test 0e - Corrupted Block RAM Reset Test

This verifies that all the bits in the Corrupted Block RAM can be reset. To clear the Corrupted Block RAM, a write to boot space (space 0) must be performed for each block address. The Corrupted Block RAM is a two Kbit RAM and is addressed using cache block addresses (i.e., 0, 0x20, 0x40..., etc.).

This test first resets the Corrupted Block RAM as described above the performs TIR read operations to obtains the Corrupt status for each block address (see Appendix A for more information about TIR read operations).

The legal error code for test 0e is:

```
01 - Corrupted bit not zero after reset
```

2.16 Test 0f - Virtual Tag RAM Addressing and Data Test

This is an addressing and data test for the Virtual Tag RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for

information on the moving inverse test algorithm).

This test uses a special test feature of the CPU Board which allows caches accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to memory). This test performs all cache accesses with the MMCR<PDCI> bit set (see Appendix A for more information on the function of the MMCR<PDCI> bit).

The virtual tags are addressed by virtual address bits 15:5. In other words, there are 2 Kbyte VTAG locations from address 0 through 0xffe0. Each entry corresponds to a cache block address, i.e., 0, 0x20, 0x40, etc. The virtual tag data is supplied by virtual address bits 31:16.

To write the vtags, a physical hit and virtual miss must occur. A virtual miss will occur whenever the virtual valid bit is false and/or the virtual address tag at the indexed location does not match the virtual address of the access.

The virtual tags are invalidated (using MMCR<VIN>) before the test is started so that during the initial tag sequence, each access will generate a virtual miss and the tag data (logical address bits 31:16) will be written into the virtual tag rams. After the initial tag sequence, each location in the tag rams should contain valid data.

During the read-write-read sequence, a TIR read is performed to verify that the virtual tag RAM contains the correct background tag data (vmatch0 and vmatch1 true). Then, the upper 16 bits of the logical address is complemented and another TIR read is performed to verify that the virtual match status is false. Complementary tag data is then written to the same VTAG location by complementing logical address bits 31:16 and performing a cache read operation. This causes a virtual miss and the VTAG rams are written with the new pattern. Finally, a TIR read is performed to verify that the virtual tag RAM contains the correct complement data (vmatch0 and vmatch1 true).

The legal error codes for this test case are:

```
01 - Virtual Tag match error on first forward pass read
02 - Virtual Tag match error when upper 16 bits of Logical
     Address was complemented on forward pass
03 - Virtual Tag match error on second forward pass read
04 - Virtual Tag match error on first reverse pass read
05 - Virtual Tag match error when upper 16 bits of Logical
     Address was complemented on reverse pass
06 - Virtual Tag match error on second reverse pass read
```

2.17 Test 10 - Virtual Tag Comparitors Test

This is a test of the Virtual Tag match detection logic. The test program loads a series of patterns into the virtual tags then performs a series of TIR reads to verify that the Virtual Tag match comparitors works correctly (see Appendix A for more information on TIR reads).

The VMATCH0 signal (low true) represents a match between logical address bits 23:16 and Virtual Tag bits 23:16. The VMATCH1 signal (low true) represents a match between logical address bits 31:24 and Virtual Tag bits 31:24.

The four test cases are outlined below:

Case 1 - Virtual tags set to walking 1 pattern with logical address bits 31:16 set to zero.

The legal error codes for test 10, case 1 are:

```
01 - VMATCH0 status error
02 - VMATCH1 status error
```

Case 2 - Virtual tags set to walking 0 pattern with logical address bits 31:16 set to all ones.

The legal error codes for test 10, case 2 are:

```
03 - VMATCH0 status error
04 - VMATCH1 status error
```

Case 3 - Virtual tags set to zero with logical address bits 31:16 set to walking 1 pattern.

The legal error codes for test 10, case 3 are:

```
05 - VMATCH0 status error
06 - VMATCH1 status error
```

Case 4 - Virtual tags set to all ones with logical address bits set to walking zero pattern.

The legal error codes for test 10, case 4 are:

```
07 - VMATCH0 status error
08 - VMATCH1 status error
```

2.18 Test 11 - Physical Tag RAM Address and Data Test

This is an addressing and data test for the physical tag RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for information on the moving inverse test algorithm).

This test uses a special test feature of the CPU Board which allows caches accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to memory). This test performs all cache accesses with the MMCR<PDCI> bit set (see Appendix A for more information on the function of the MMCR<PDCI> bit).

The physical tags are indexed by virtual address bits 15:5. In other words, there are 2 Kbyte PTAG locations from address 0 through 0xffe0. Each entry corresponds to a cache block address, i.e., 0, 0x20, 0x40, etc. The physical tag data is supplied by physical address bits 31:13 which is read from the TLB Physical address field.

The physical tags are always loaded on a cache access regardless of whether there is a physical hit or physical miss as long as there is also a virtual miss. In order to always generate a virtual miss, this test invalidates the virtual tags (using MMCR<VIN) between each cache access.

The physical tags are invalidated before the test is started. For each cache access, the TLB is loaded with a logical to physical mapping in which the logical address is the desired physical tag RAM index (0 through 0xffe) and the physical address field is the desired data to be written to the physical tags (`0xaaaaa000` or `0x55554000`).

The physical tags are initialized to `0xaaaaa000`, the PVALID bit is set, and the POWN bit is cleared.

During the the read-write-read test sequence, a TIR read is performed to verify that the physical tag RAM contains the correct background tag (PMATCH0/1/2 are true) and that PVALID is true and POWN is false. Then, the physical address field in the TLB is complemented, and another TIR read is performed to verify that the PMATCH status bits are false and the PVALID

and POWN status is unchanged. Complementary tag, PVALID, and POWN data is then written to the target physical tag location by performing another cache read operation. Finally, a TIR read is performed to verify that the physical tag RAM contains the correct tag data (by verifying PMATCH0/1/2 are true) and that PVALID is false and POWN is true.

The legal error codes for test 11 are:

```
01 - Physical tag match or status error on first forward pass read
02 - Physical tag match or status error when TLB physical address
     field was complemented on forward pass
03 - Physical tag match or status error on second forward pass read

04 - Physical tag match or status error on first reverse pass read
05 - Physical tag match or status error when TLB physical address
     field was complemented on reverse pass
06 - Physical tag match or status error on second reverse pass read
```

2.19 Test 12 - Physical Tag Comparitors Test

This is a test of the Physical Tag match detection logic. The test program loads a series of patterns into the physical tags then performs a series of TIR reads to verify that the physical tag match comparitors works correctly (see Appendix A for more information on TIR reads).

The PMATCH0 signal (low true) represents a match between physical address bits 15:13 and bits 15:13 of the physical address field of the TLB; the PMATCH1 signal (low true) represents a match between physical address bits 23:16 and bits 23:16 of the physical address field of the TLB; and the PMATCH2 signal (low true) represents a match between physical address bits 31:24 and bits 31:24 of the physical address field of the TLB.

There are 4 test cases as outlined below:

Case 1 - Physical tags set to walking 1 pattern with physical address field of TLB set to zero.

The legal error codes for test 12, case 1 are:

```
01 - PMATCH0 status error
02 - PMATCH1 status error
03 - PMATCH2 status error
```

Case 2 - Physical tags set to walking 0 pattern with physical address field of TLB set to all ones.

The legal error codes for test 12, case 2 are:

```
04 - PMATCH0 status error
05 - PMATCH1 status error
06 - PMATCH2 status error
```

Case 3 - Physical tags set to zero with physical address field of TLB set to walking 1 pattern.

The legal error codes for test 12, case 3 are:

```
07 - PMATCH0 status error
08 - PMATCH1 status error
09 - PMATCH2 status error
```

Case 4 - Physical tags set to all ones with physical address field of TLB set to walking zero pattern.

The legal error codes for test 12, case 4 are:

```
0a - PMATCH0 status error
0b - PMATCH1 status error
0c - PMATCH2 status error
```

2.20 Test 13 - Purge RAM Addressing and Data Test

This is an addressing and data test for the Purge RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for information on the moving inverse test algorithm).

This test uses a special test feature of the CPU Board which allows caches accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to memory). This test performs all cache accesses with the MMCR<PDCI> bit set (see Appendix A for more information on the function of the MMCR<PDCI> bit).

The purge RAM is indexed using physical address bits 15:5 and contains 4 bits of information; logical address bits 15:13 and a valid bit. For each entry in the physical cache, there is a corresponding entry in the purge RAM which contains the reverse (physical to logical) address translation. The purge RAM is loaded with logical address bits 15:13 and the valid bit is set whenever the cache tags are loaded.

The physical tags are invalidated before the test is started. For each cache access, the TLB is loaded with a logical to physical mapping in which the logical address is the desired state of bits 15:13 (`0xa000` or `0x4000`) and the physical address is the desired purge RAM index (`0` through `0xffe`).

During the read-write-read sequence, a TIR read operation is performed to verify that the reverse translation in the purge RAM contains the correct bit pattern and is valid. Then, the complement pattern is written to the same purge RAM location by complementing the logical index and using the same physical address in the TLB and performing another cache read operation. This causes the purge RAM to be written with the new logical address. Finally, a TIR read operation is performed to verify that the purge RAM contains the correct complement bit pattern and is valid.

The legal error codes for this test are:

```
01 - purge address or valid bit error on first read of forward pass
02 - purge address or valid bit error on second read of forward pass
03 - purge address or valid bit error on first read of reverse pass
04 - purge address or valid bit error on second read of reverse pass
```

2.21 Test 14 - Virtual Tag Even Block Revalidation Test

Virtual tag re-validations occur when there is a physical cache hit and a virtual cache miss. This test creates valid physical cache blocks, invalidates the virtual cache, then re-accesses the target cache block. The virtual tags should be updated with VALID and OWN from the physical cache, and RO and UP from the TLB status bits.

MMCR<PDCI> operations are used to initially load the physical tags with the desired state. When the physical tags are initialized, the VRO and VUP bits in the virtual tags are written from

the TLB RO and UP bits respectively.

This test initializes the physical OWN (POWN) status for the even and odd blocks as indicated in the table below. Then performs two sequential accesses to the even block address (address 0). After each access the Physical and Virtual cache status is checked and verified to match the conditions indicated in the tables below. There are four test cases.

	EVEN STATUS						ODD STATUS					
Case 1:	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						0				
State After 1st Even Access	1		1		1		1					1
State After 2nd Even Access	1		1		1		1					1

The legal error codes for test 14, case 1 are:

```
01 - error in even block status after first access
02 - error in odd block status after first access
03 - error in even block status after second access
04 - error in odd block status after second access
```

	EVEN STATUS						ODD STATUS					
Case 2:	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After 1st Even Access	1		1		1		1	1				1
State After 2nd Even Access	1		1		1		1	1				1

The legal error codes for test 14, case 2 are:

```
05 - error in even block status after first access
06 - error in odd block status after first access
07 - error in even block status after second access
08 - error in odd block status after second access
```

Case 3:	EVEN STATUS PVALID	POWN	VVALID	VOWN	VRO	VUP	ODD STATUS PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After 1st Even Access	1	1	1	1	1		1					1
State After 2nd Even Access	1	1	1	1	1		1					1

The legal error codes for test 14, case 3 are:

```
09 - error in even block status after first access
0a - error in odd block status after first access
0b - error in even block status after second access
0c - error in odd block status after second access
```

Case 4:	EVEN STATUS PVALID	POWN	VVALID	VOWN	VRO	VUP	ODD STATUS PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						1				
State After 1st Even Access	1	1	1	1	1		1	1				1
State After 2nd Even Access	1	1	1	1	1		1	1				1

The legal error codes for test 14, case 4 are:

```
0d - error in even block status after first access
0e - error in odd block status after first access
0f - error in even block status after second access
10 - error in odd block status after second access
```

2.22 Test 15 - Virtual Tag Odd Block Revalidation Test

Virtual tag re-validations occur when there is a physical cache hit and a virtual cache miss. This test creates valid physical cache blocks, invalidates the virtual cache, then re-accesses the target cache block. The virtual tags should be updated with VALID and OWN from the physical cache, and RO and UP from the TLB status bits.

MMCR<PDCI> operations are used to initially load the physical tags with the desired state. When the physical tags are initialized, the VRO and VUP bits in the virtual tags are written from the TLB RO and UP bits respectively.

This test initializes the physical OWN (POWN) status for the even and odd blocks as indicated in the table below, then performs two sequential accesses to the odd block address (address 0x20). After each access the Physical and Virtual cache status is checked and verified to match the conditions indicated in the tables below. There are four test cases.

	EVEN STATUS						ODD STATUS					
Case 1:	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						0				
State After 1st Odd Access	1				1		1		1			1
State After 2nd Odd Access	1				1		1		1			1

The legal error codes for test 15, case 1 are:

```
01 - error in even block status after first access
02 - error in odd block status after first access
03 - error in even block status after second access
04 - error in odd block status after second access
```

	EVEN STATUS						ODD STATUS					
Case 2:	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After 1st Odd Access	1				1		1	1	1	1		1
State After 2nd Odd Access	1				1		1	1	1	1		1

The legal error codes for test 15, case 2 are:

```
05 - error in even block status after first access
06 - error in odd block status after first access
07 - error in even block status after second access
08 - error in odd block status after second access
```

Case 3:	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After 1st Odd Access	1	1			1		1		1			1
State After 2nd Odd Access	1	1			1		1		1			1

The legal error codes for test 15, case 3 are:

```
09 - error in even block status after first access
0a - error in odd block status after first access
0b - error in even block status after second access
0c - error in odd block status after second access
```

Case 4:	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						1				
State After 1st Odd Access	1	1			1		1	1	1	1		1
State After 2nd Odd Access	1	1			1		1	1	1	1		1

The legal error codes for test 15, case 4 are:

```
0d - error in even block status after first access
0e - error in odd block status after first access
0f - error in even block status after second access
10 - error in odd block status after second access
```

2.23 Test 16 - Virtual Tag Even/Odd Block Revalidation Test

Virtual tag re-validations occur when there is a physical cache hit and a virtual cache miss. This test creates valid physical cache blocks, invalidates the virtual cache, then re-accesses the target cache block. The virtual tags should be updated with VALID and OWN from the physical cache, and RO and UP from the TLB status bits.

MMCR<PDCI> operations are used to initially load the physical tags with the desired state. When the physical tags are initialized, the VRO and VUP bits in the virtual tags are written from the TLB RO and UP bits respectively.

This test initializes the physical OWN (POWN) status for the even and odd blocks as indicated in the table below, then performs two sequential accesses; the first to the even block (0) and the second to the odd block (0x20). After each access the Physical and Virtual cache status is checked and verified to match the conditions indicated in the tables below. There are four test cases.

	EVEN STATUS						ODD STATUS					
Case 1:	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						0				
State After 1st Even Access-	1		1		1		1					1
State After 2nd Odd Access-	1		1		1		1		1			1

The legal error codes for test 16, case 1 are:

```
01 - error in even block status after first access
02 - error in odd block status after first access
03 - error in even block status after second access
04 - error in odd block status after second access
```

	EVEN STATUS						ODD STATUS					
Case 2:	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After 1st Even Access-	1		1		1		1	1				1
State After 2nd Odd Access-	1		1		1		1	1	1	1		1

The legal error codes for test 16, case 2 are:

```
05 - error in even block status after first access
06 - error in odd block status after first access
07 - error in even block status after second access
08 - error in odd block status after second access
```

Case 3:	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After 1st Even Access-	1	1	1	1	1		1					1
State After 2nd Odd Access-	1	1	1	1	1		1		1			1

The legal error codes for test 16, case 3 are:

```
09 - error in even block status after first access
0a - error in odd block status after first access
0b - error in even block status after second access
0c - error in odd block status after second access
```

Case 4:	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						1				
State After 1st Even Access-	1	1	1	1	1		1	1				1
State After 2nd Odd Access-	1	1	1	1	1		1	1	1	1		1

The legal error codes for test 16, case 4 are:

```
0d - error in even block status after first access
0e - error in odd block status after first access
0f - error in even block status after second access
10 - error in odd block status after second access
```

2.24 Test 17 - Virtual Tag Odd/Even Block Revalidation Test

Virtual tag re-validations occur when there is a physical cache hit and a virtual cache miss. This test creates valid physical cache blocks, invalidates the virtual cache, then re-accesses the target cache block. The virtual tags should be updated with VALID and OWN from the physical cache, and RO and UP from the TLB status bits.

MMCR<PDCI> operations are used to initially load the physical tags with the desired state. When the physical tags are initialized, the VRO and VUP bits in the virtual tags are written from the TLB RO and UP bits respectively.

This test initializes the physical OWN (POWN) status for the even and odd blocks as indicated in the table below, then performs two sequential accesses; the first to the odd block (0x20) and the second to the even block (0). After each access the Physical and Virtual cache status is checked and verified to match the conditions indicated in the tables below. There are four test cases.

	EVEN STATUS						ODD STATUS					
Case 1:	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						0				
State After 1st Odd Access	1				1		1		1			1
State After 2nd Even Access	1		1		1		1		1			1

The legal error codes for test 17, case 1 are:

```
01 - error in even block status after first access
02 - error in odd block status after first access
03 - error in even block status after second access
04 - error in odd block status after second access
```

	EVEN STATUS						ODD STATUS					
Case 2:	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After 1st Odd Access	1				1		1	1	1	1		1
State After 2nd Even Access	1		1		1		1	1	1	1		1

The legal error codes for test 17, case 2 are:

```
05 - error in even block status after first access
06 - error in odd block status after first access
07 - error in even block status after second access
08 - error in odd block status after second access
```

Case 3:	EVEN STATUS PVALID	POWN	VVALID	VOWN	VRO	VUP	ODD STATUS PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After 1st Odd Access-	1	1			1		1		1			1
State After 2nd Even Access-	1	1	1	1	1		1		1			1

The legal error codes for test 17, case 3 are:

```
09 - error in even block status after first access
0a - error in odd block status after first access
0b - error in even block status after second access
0c - error in odd block status after second access
```

Case 4:	EVEN STATUS PVALID	POWN	VVALID	VOWN	VRO	VUP	ODD STATUS PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						1				
State After 1st Odd Access-	1	1			1		1	1	1	1		1
State After 2nd Even Access-	1	1	1	1	1		1	1	1	1		1

The legal error codes for test 17, case 4 are:

```
0d - error in even block status after first access
0e - error in odd block status after first access
0f - error in even block status after second access
10 - error in odd block status after second access
```

2.25 Test 18 - Virtual Tag Block Invalidation Test

Virtual tag re-validations occur when there is a physical cache hit and a virtual cache miss. This test creates valid physical cache blocks, invalidates the virtual cache, then generates a TLB fault when the target block is re-accessed. In this case, the virtual tags should be updated with OWN from the physical cache, RO and UP from the TLB status bits as in the virtual block re-validation tests, but in this case the virtual VALID bit should be cleared (set to invalid).

MMCR<PDCI> operations are used to initially load the physical tags with the desired state. When the physical tags are initialized, the VRO and VUP bits in the virtual tags are written from the TLB RO and UP bits, respectively.

This test initializes the physical OWN (POWN) status for the even and odd blocks as indicated in the table below, then performs several cache block accesses depending on the test case (see tables below). The last access in the sequence is designed to write protect fault. The even and odd block status is check after the last access and verified to match the conditions specified in the tables below. There are 16 test cases.

Case 1: 1 access - Even block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After WPF Access	1	1		1	1		1				1	

The legal error codes for test 18, case 1 are:

```
01 - No write protect fault was generated
02 - Write Protect Fault bit not set in Fault Cause Register
03 - Error in even block status after exception occurred
04 - Error in odd block status after exception occurred
```

Case 2: 1 access - Odd block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After WPF Access	1	1			1		1				1	

The legal error codes for test 18, case 2 are:

```
05 - No write protect fault was generated
06 - Write Protect Fault bit not set in Fault Cause Register
07 - Error in even block status after exception occurred
08 - Error in odd block status after exception occurred
```

Case 3: 2 accesses - Odd block - Even block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After WPF Access	1	1		1	1		1		1		1	

The legal error codes for test 18, case 3 are:

```
09 - No write protect fault was generated
0a - Write Protect Fault bit not set in Fault Cause Register
0b - Error in even block status after exception occurred
0c - Error in odd block status after exception occurred
```

Case 4: 2 accesses - Even block - Odd block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After WPF Access	1	1	1	1	1		1				1	

The legal error codes for test 18, case 4 are:

```
0d - No write protect fault was generated
0e - Write Protect Fault bit not set in Fault Cause Register
0f - Error in even block status after exception occurred
10 - Error in odd block status after exception occurred
```

Case 5: 2 accesses - Even block - Even block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After WPF Access	1	1		1	1		1				1	

The legal error codes for test 18, case 5 are:

```
11 - No write protect fault was generated
12 - Write Protect Fault bit not set in Fault Cause Register
13 - Error in even block status after exception occurred
14 - Error in odd block status after exception occurred
```

Case 6: 2 accesses - Odd block - Odd block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After WPF Access	1	1			1		1				1	

The legal error codes for test 18, case 6 are:

```
5 - No write protect fault was generated
16 - Write Protect Fault bit not set in Fault Cause Register
17 - Error in even block status after exception occurred
18 - Error in odd block status after exception occurred
```

Case 7: 3 accesses - Odd block - Even block - Odd block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After WPF Access	1	1	1	1	1		1				1	

The legal error codes for test 18, case 7 are:

```
19 - No write protect fault was generated
1a - Write Protect Fault bit not set in Fault Cause Register
1b - Error in even block status after exception occurred
1c - Error in odd block status after exception occurred
```

Case 8: 3 accesses - Even block - Odd block - Even block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		1						0				
State After WPF Access	1	1		1	1		1		1		1	

The legal error codes for test 18, case 8 are:

```
1d - No write protect fault was generated
1e - Write Protect Fault bit not set in Fault Cause Register
1f - Error in even block status after exception occurred
20 - Error in odd block status after exception occurred
```

Case 9: 1 access - Even block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After WPF Access	1				1		1	1			1	

The legal error codes for test 18, case 9 are:

```
21 - No write protect fault was generated
22 - Write Protect Fault bit not set in Fault Cause Register
23 - Error in even block status after exception occurred
24 - Error in odd block status after exception occurred
```

Case 10: 1 access - Odd block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After WPF Access	1				1		1	1		1	1	

The legal error codes for test 18, case 10 are:

```
25 - No write protect fault was generated
26 - Write Protect Fault bit not set in Fault Cause Register
27 - Error in even block status after exception occurred
28 - Error in odd block status after exception occurred
```

Case 11: 2 accesses - Odd block - Even block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After WPF Access	1				1		1	1	1	1	1	

The legal error codes for test 18, case 11 are:

```
29 - No write protect fault was generated
2a - Write Protect Fault bit not set in Fault Cause Register
2b - Error in even block status after exception occurred
2c - Error in odd block status after exception occurred
```

Case 12: 2 accesses - Even block - Odd block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After WPF Access	1		1		1		1	1		1	1	

The legal error codes for test 18, case 12 are:

```
2d - No write protect fault was generated
2e - Write Protect Fault bit not set in Fault Cause Register
2f - Error in even block status after exception occurred
30 - Error in odd block status after exception occurred
```

Case 13: 2 accesses - Even block - Even block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After WPF Access	1				1		1	1			1	

The legal error codes for test 18, case 13 are:

```
31 - No write protect fault was generated
32 - Write Protect Fault bit not set in Fault Cause Register
33 - Error in even block status after exception occurred
34 - Error in odd block status after exception occurred
```

Case 14: 2 accesses - Odd block - Odd block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After WPF Access	1				1		1	1		1	1	

The legal error codes for test 18, case 14 are:

```
35 - No write protect fault was generated
36 - Write Protect Fault bit not set in Fault Cause Register
37 - Error in even block status after exception occurred
38 - Error in odd block status after exception occurred
```

Case 15: 3 accesses - Even block - Odd block - Even block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After WPF Access	1				1		1	1	1	1	1	

The legal error codes for test 18, case 15 are:

```
39 - No write protect fault was generated
3a - Write Protect Fault bit not set in Fault Cause Register
3b - Error in even block status after exception occurred
3c - Error in odd block status after exception occurred
```

Case 16: 3 accesses - Odd block - Even block - Odd block

	EVEN STATUS						ODD STATUS					
	PVALID	POWN	VVALID	VOWN	VRO	VUP	PVALID	POWN	VVALID	VOWN	VRO	VUP
Initial State		0						1				
State After WPF Access	1		1		1		1	1		1	1	

The legal error codes for test 18, case 16 are:

```
3d - No write protect fault was generated
3e - Write Protect Fault bit not set in Fault Cause Register
3f - Error in even block status after exception occurred
40 - Error in odd block status after exception occurred
```

2.26 Test 19 - MMU Fault Test

This test verifies that all types of MMU exceptions can be generated. In addition, the auto-read sequence is verified to return the correct registers values.

For all MMU exceptions, the FCR is verified to be updated with the correct cause of the exception (TMISS, UP, WP, POF) and the FVAR is verified to contain the unmapped (Virtual) address used by the access.

The 13 cases for test 19 follow:

Case 1: TMISS fault (TTVALID false)

This test case maps logical address patterns to physical address zero, but reads the TLB entries to make them invalid (clears TTVALID). The logical address pattern is then used as the address in ld instruction. This causes a TLB miss to occur. The FCR, FVAR, and PDEP are read using the auto-read sequence after the exception is verified.

The legal error codes test 19, case 1 are:

```
01 - Data exception was not generated
02 - FCR<TMISS> bit was not set
03 - PDEP register did not contain correct value when read through
     auto read space.
04 - FVAR register did not contain correct value when read through
     auto read space.
05 - FCR register was not cleared when the FVAR was read.
```

Case 2: TMISS fault (TTVALID true) AND (TMATCH0 = false)

This test case maps logical address zero to physical address zero to create a valid TLB entry in TLB location zero, then performs a ld instruction at logical address 0xaa000000. This causes a TLB miss to occur due to a tag mismatch. The FCR, FVAR and PDEP are read using the auto-read sequence after the exception is verified.

The legal error codes test 19, case 2 are:

```
06 - Data exception was not generated
07 - FCR TMISS fault bit was not set
08 - PDEP register did not contain correct value when read through
     auto read space.
09 - FVAR register did not contain correct value when read through
     auto read space.
0a - FCR register was not cleared when the FVAR was read.
```

Case 3: TMISS fault (TTVALID true) AND (TMATCH0 = false)

This is the same as case 2 except that 0x55000000 is used for the logical address for the ld instruction.

The legal error codes test 19, case 3 are:

```
0b - Data exception was not generated
0c - FCR TMISS fault bit was not set
0d - PDEP register did not contain correct value when read through
     auto read space.
0e - FVAR register did not contain correct value when read through
     auto read space.
0f - FCR register was not cleared when the FVAR was read.
```

Case 4: UPF fault (TLBUP true)

This test case maps logical address 0x55555555 to physical address 0 and sets the UP bit (user protect) in the TLB, then accesses logical address 0x55555555 through user data space (ASI=10) to cause a UP fault. The FCR, FVAR and PDEP are read using the auto-read ASI after the exception is verified.

The legal error codes test 19, case 4 are:

```
10 - Data exception was not generated
11 - FCR UPF bit was not set
12 - PDEP register did not contain correct value when read through
     auto read space.
13 - FVAR register did not contain correct value when read through
     auto read space.
14 - FCR register was not cleared when the FVAR was read.
```

Case 5: UPF fault (TLBUP true)

This is the same as case 4 except logical address 0xaaaaaaaa is used.

The legal error codes test 19, case 5 are:

```
15 - Data exception was not generated
16 - FCR UPF bit was not set
17 - PDEP register did not contain correct value when read through
     auto read space.
18 - FVAR register did not contain correct value when read through
     auto read space.
19 - FCR register was not cleared when the FVAR was read.
```

Case 6: UPF fault (FE space)

This test case accesses logical address 0xff555555 through user data space (ASI=10) to cause a User Protection fault. The FCR, FVAR and PDEP are read using the auto-read ASI after the exception is verified.

The legal error codes test 19, case 6 are:

```
1a - Data exception was not generated
1b - FCR UPF bit was not set
1c - PDEP register did not contain correct value when read through
     auto read space.
1d - FVAR register did not contain correct value when read through
     auto read space.
1e - FCR register was not cleared when the FVAR was read.
```

Case 7: UPF fault (FE space)

This is the same as case 6 except logical address 0xfeaaaaaa is used.

The legal error codes test 19, case 7 are:

```
1f - Data exception was not generated
20 - FCR UPF bit was not set
21 - PDEP register did not contain correct value when read through
     auto read space.
22 - FVAR register did not contain correct value when read through
     auto read space.
23 - FCR register was not cleared when the FVAR was read.
```

Case 8: WPF fault (TLBRO)

This test case maps logical address zero to physical address zero and sets the TLB RO bit. Performs st instruction to logical address zero to cause a WPF fault to occur. The FCR, FVAR and PDEP are read using the auto-read ASI after the exception is verified.

The legal error codes test 19, case 8 are:

```
24 - Data exception was not generated
25 - FCR WPF bit was not set
26 - PDEP register did not contain correct value when read through
     auto read space.
27 - FVAR register did not contain correct value when read through
     auto read space.
28 - FCR register was not cleared when the FVAR was read.
```

Case 9: WPF fault (TLBRO)

This is the same as case 8 except logical address `0xfdffffff` is mapped to physical address zero.

The legal error codes test 19, case 9 are:

```
29 - Data exception was not generated
2a - FCR WPF bit was not set
2b - PDEP register did not contain correct value when read through
     auto read space.
2c - FVAR register did not contain correct value when read through
     auto read space.
2d - FCR register was not cleared when the FVAR was read.
```

Case 10: WPF fault (TLBIOB)

This test case maps logical address `0xfdffffff` to physical address 0 and sets the TLB IOB bit, then performs a st instruction to logical address `0xfdffffff` to cause a WPF fault to occur. The FCR, FVAR and PDEP are read using the auto-read ASI after the exception is verified.

The legal error codes test 19, case 10 are:

```
2e - Data exception was not generated
2f - FCR WPF bit was not set
30 - PDEP register did not contain correct value when read through
     auto read space.
31 - FVAR register did not contain correct value when read through
     auto read space.
32 - FCR register was not cleared when the FVAR was read.
```

Case 11: WPF fault (TLBIOB)

This is the same as case 10 except logical address zero is mapped to physical address 0.

The legal error codes test 19, case 11 are:

```
33 - Data exception was not generated
34 - FCR WPF bit was not set
35 - PDEP register did not contain correct value when read through
     auto read space.
36 - FVAR register did not contain correct value when read through
     auto read space.
37 - FCR register was not cleared when the FVAR was read.
```

Case 12: POF fault (TLBPVALID false)

This test case maps logical address 0x66666666 to physical address zero and clears the TLB page valid bit, then performs ld instruction to logical address 0x66666666 to cause a POF fault. The FCR, FVAR and PDEP are read using the auto-read ASI after the exception is verified.

The legal error codes test 19, case 12 are:

```
38 - Data exception was not generated
39 - FCR POF bit was not set
3a - PDEP register did not contain correct value when read through
     auto read space.
3b - FVAR register did not contain correct value when read through
     auto read space.
3c - FCR register was not cleared when the FVAR was read.
```

Case 13: POF fault (TLBPVALID false)

This is the same as case 12 except logical address 0x99999999 is mapped to physical address zero.

The legal error codes test 19, case 13 are:

```
3d - Data exception was not generated
3e - FCR POF bit was not set
3f - PDEP register did not contain correct value when read through
     auto read space.
40 - FVAR register did not contain correct value when read through
     auto read space.
41 - FCR register was not cleared when the FVAR was read.
```

2.27 Test 1a - Timeout Fault Test

This test verifies that the timeout logic on the System Board is functional, that the Kbus Address lines are good and that the timeout detection logic in the bus watcher section of the CPU Board is functional. This is the first test which generates Kbus cycles. If a KBUS transaction does not terminate within the required number of cycles, the timeout logic on the System Board terminates the transaction by asserting TERR.

This test performs an RIO access to the ID space of non-existant system slot 0. Since there is no board in slot zero to respond to the ID space read, the transaction should timeout and TERR should be asserted. The test verifies that the FCR<TOFIO> bit is set, that the FVAR register contains the correct Logical RIO address, and that FPAR contains the correct physical RIO address.

There are 50 Kbus transactions executed by this test; all of which are to ID space of non-existant slot 0. Patterns consisting of all zeroes, all ones, walking 1 and walking 0's are used for the low 24 bits of the RIO address.

The legal error codes test 1a are:

```
01 - Data fault exception was not generated
02 - FCR TOFIO bits was not set
03 - FVAR register did not contain correct logical RIO address.
04 - FCR register was not cleared when the FVAR was read.
05 - FPAR register did not contain correct physical RIO adddress.
```

2.28 Test 1b - Slot Probe and Configuration Test

This test probes each slot of the system by performing ID space reads and determines the board types which occupy each slot.

RIO accesses are performed to the ID space of each slot in the system (1 through 7) and if an RIO timeout fault does not occur, examines the ID information returned. From this information, a configuration table is built up in the on-board diagnostic ram. Once the table is built, the test checks to verifiy that the minimum valid system configuration exists.

In addition, this test checks the state of the burn-in jumper. If the burn-in jumper is present (installed) then the tests run in burn-in mode. If the burn-in jumper is not present, then the diag/normal test switch state is read, and if set to "Diag Mode" then the BOOTMODE environment variable is read from the EAROM. If the BOOTMODE variable is "burnin" then the tests are run in burn-in mode; otherwise, the tests run normally.

The minimum valid system configuration consists of at least one CPU Board, at least one Memory Board and only one System Board. This is the minimum configuration which must exist to continue testing and to boot UNIX.

In the following error codes, the *X* represents the slot number of the target board.

```
0X - Exception other than Data Fault occurred during ID SPACE
     read of slot X.
1X - Data exception occurred during initial probe, but FCR TOFIO
     was not set.
2X - FVAR contained incorrect logical RIO address.
3X - FCR not cleared after reading FVAR.
4X - Unrecognizeable board type code read from slot X.
5X - Data exception fault occurred during ID space read after valid
     board was previously located in slot X.
6X - Data exception fault occurred during RIO read of optional header
     in IDPROM on board in slot X.
7X - Data fault exception occurred during RIO read of graphics minor
     board number from IDPROM in slot X.
8X - Data fault exception occurred during RIO read of device identifier
     string from IDPROM in slot X.
9X - Data fault exception occurred during RIO read of Memory Board size
          from IDPROM in slot X.
ax - Zero size parameter read from IDPROM on Memory Board in slot X.
bx - Invalid Memory Board size read from IDPROM in slot X.
          Not an even 16 Mbyte multiple.
c0 - System Board count is not 1 (0 or more than 1).
c1 - No Memory Boards were located.
c2 - No CPU Boards were located.
d0 - Data exception occurred reading EAROM BOOTMODE variable.
```

2.29 Test 1c - IDPROM Checksum Test

This test examines the configuration information obtained from the Slot Probe and configuration test and for each board identified, performs an IDPROM checksum test. By convention, the contents of any IDPROM header can be summed and the least significant byte of the result will be zero.

The legal error codes for test 1c are:

```
1X - Data exception fault occurred while reading IDPROM size from
     the board in slot X.
2X - Zero size field for IDPROM on board in slot X.
3X - Data exception fault occurred while performing checksum on
     IDPROM on board in slot X.
4X - IDPROM checksum error for board in slot X.
5X - Data exception fault occurred while reading the optional header
     field of the IDPROM on board in slot X.f1
```

2.30 Test 1d - Master/Slave CPU Determination Test

This test determines which CPU in the system is to become the master CPU when multiple CPUs exist.

Upon reset, the slave bit on the CPUSTAT register on each of the installed CPU Boards is zero. All CPU Boards in the system execute the power-up tests in parallel until this test is executed. During this test, each CPU Board examines its own BID register to get its slot number. The slot number is then decremented and multiplied by 400 to compute the number of milliseconds to delay. Each CPU then delays based on the computed delay value.

The first CPU to finish its delay, reads its own CPUSTAT register. If the SLAVE bit is still zero and if the EAROM is unintitled or the EAROM "MASTER" variable is not valid or is valid and specifies this CPU, the first CPU looks in the configuration table generated by the Slot Probe and Configuation Test and sets the CPUSTAT SLAVE bit of all the other installed CPU Boards. The master CPU Board proceeds to finish the power-up test from this point on.

Subsequent CPU Boards which finish their delays will find their SLAVE bits set and will enter a loop which displays "SL" and the slot number on their LEDs.

The legal error codes for test 1d are:

```
10   - Data exception fault occurred while reading the
       CPUSTAT register
20   - Data exception fault occurred while writing CPUSTAT
       register of slave CPU Board.
0x10 - Data fault occurred accessing own cpustat register
0x2x - Data fault occurred accessing cpustat register of CPU in slot "X"
0x30 - Data fault occurred accessing EAROM
0x4x - Data fault occurred accessing CPUHR register of CPU in slot "X"
```

2.31 Test 1e - Bus Watcher Tag Reset Test

This test utilizes a special test feature of the P0 CPU Board. This special feature is the Diagnostic Kbus Transaction. These transactions allow the bus watcher tags to be loaded and read without actually read or writing Kbus memory. See Appendix A for more information on the Diagnostic Kbus Transactions.

This test verifies that the bus watcher tag reset function is functional. All tag RAM locations are written with unique data and the OWN and VAL bits are set. Each tag location is then read and the PM0, PM1, OWN and VAL bits are verified to be true. A bus watcher reset is then performed then each tag location is re-read. This time the PMO and PM1 bits should be true and the OWN and VAL bits should be false.

The legal error codes for test 1e are:

```
01 - Match, Own or Valid status error after initial write of tags
     and status.
02 - Match, Own or Valid status error after reset of Own and Valid
     status bits.
```

2.32 Test 1f - Bus Watcher Tag RAM Addressing Test

This test verifies the address lines for the Bus Watcher tag and status rams. Each tag RAM location corresponding to a single address bit on (0x20, 0x40, 0x80... 0x8000) is written with unique data, then tag RAM location 0 is written with zero. Each location is then probed, in the same order as written, and the physical tag match bits (PM0 and PM1) are verified to be true.

The legal error codes for test 1f, case 1 are:

```
01 - Bus watcher tag match status error on read of tag location
     other than zero.
02 - Bus watcher tag match status error on read of tag location
     zero.
```

The test is then repeated using locations corresponding to a single address bit off (0xffc0, 0xffa0..., 0x7fe0), and address 0xffe0 is written with zero.

The legal error codes for this test 1f, case 2 are:

```
03 - Bus watcher tag match status error on read of tag location
     other than 0xffe0.
04 - Bus watcher tag match status error on read of tag location
     0xffe0.
```

2.33 Test 20 - Bus Watcher Tag Comparitors Test

This is a test of the Bus Watcher Tag match detection logic. The test program loads a series of patterns into the Bus Watcher tags then performs a series of Diagnostic Kbus Transactions to read and verify that the Bus Watcher Tag match comparitors work correctly (see Appendix A for more information on Diagnostic Kbus Transactions).

The PM0 signal (low true) represents a match between physical address bits 23:16 and Bus Watcher tag bits 23:16. The PM1 signal (low true) represents a match between physical address bits 31:24 and Bus Watcher Tag bits 31:24.

There are two test cases. Both test cases use bus watcher tag location zero to contain the test patterns.

Case 1 iteratively loads bus watcher tag RAM location 0 with a walking 1 tag pattern, and performs diagnostic kbus probe transactions to tag location 0 with the upper 16 bits of the probe transaction address all zeroes. For the first eight probe transactions, PM0 will be false and PM1 will be true. For the second eight probe transactions, PM1 will be false and PM0 will be true.

The legal error codes for test 10, case 1 are:

```
01 - PM0 match status error
02 - PM1 match status error
```

Case 2 loads bus watcher tag RAM location 0 with a zero tag pattern, then performs diagnostic Kbus probe transactions to bus watcher tag RAM location 0 while walking a 1 accoss the upper 16 bits of the probe transaction address. For the first eight probe transactions, PM0 will be false and PM1 will be true. For the second eight probe transactions, PM1 will be false and PM0 will be true.

The legal error codes for test 20, case 2 are:

```
03 - PM0 match status error
04 - PM1 match status error
```

2.34 Test 21 - Bus Watcher Tag RAM Address and Data Test

This is an addressing and data test for the bus watcher tag RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for information on the moving inverse test algorithm).

This test uses a special test feature of the CPU Board which allows the bus watcher tags to be accessed without generating any Kbus memory cycles (see Appendix A for more information on the Diagnostic Kbus transactions).

During the read-write-read sequence, a Diagnostic Kbus transaction is perfomed to read and verify the bus watcher tag match, OWN and VAL status. Then a Diagnostic Kbus transaction is performed to load the same bus watcher tag RAM location with the complement tag data and toggle the OWN and VAL bits. Finally, a Diagnostic Kbus transaction is performed to read and verify the new states of the bus watcher tag match and OWN and VAL signals.

The legal error codes for test 21 are:

```
01 - PM0/PM1/VAL/OWN status error on first read of forward pass
02 - PM0/PM1/VAL/OWN status error on second read of forward pass
03 - PM0/PM1/VAL/OWN status error on first read of reverse pass
04 - PM0/PM1/VAL/OWN status error on second read of reverse pass
```

2.35 Test 22 - Memory Board Base Address and Enable Register Test

This is a test for the Base Address Register and Enable Register on each installed Memory Board.

The base address register is an 8-bit register at address `1X020000` in ID space. The Enable register is a 2-bit register at address `1X030000` in ID space. The X represents the slot number. All Memory Boards identified by the configuration table generated by the Slot Probe and Configuration Test are tested.

Part 1 is the test of the Base Address register. There are 18, one byte test patterns written to and read from the Base Address register; these consist of all zeros all ones, walking 1 and walking 0 patterns. Part 2 is the test of the Enable register. There are 4, 2 bit test patterns consisting of 0, 3, 1 and 2.

The legal error codes for test 22 are:

```
1X - Data exception fault occurred writing base address register of
     Memory Board in slot X.
2X - Data exception fault occurred reading base address register of
     Memory Board in slot X.
3X - Data miscompare error for base address register on Memory Board
     in slot X.
4X - Data exception fault occurred writing enable register of Memory
     Board in slot X.
5X - Data exception fault occurred reading enable register of Memory
     Board in slot X.
6X - Data miscompare error for enable register on Memory Board
     in slot X.
```

2.36 Test 23 - Memory Board Uniqueness Test

This test verifies that all installed Memory Boards can be accessed independently of all others.

This is the first test which attempts to write and read memory and thereby test the bus watchers ability to perform Kbus transactions other than RIO types. When a memory block is read by the CPU Board, and the data is not already in the CPU's cache, the bus watcher must perform a Kbus transaction to obtain the data from memory (see Kbus Protocol specification for detailed information on cache/memory consistency) and the data is placed in the CPU's cache. To get the block out of the CPU's cache and back to memory, a cache flush operation must be performed. To flush a cache block, the CPU generally references another block which is an exact multiple of 64 Kbytes offset from the block to be flushed.

This test writes the first byte of each installed Memory Board with the target boards slot number, then reads the first byte of each board back and verifies the slot numbers.

The legal error codes for test 23 are:

```
01-06:  Indicates an exception occurred on the initial "stb" instruction.
        The error code is the slot number of the target Memory Board.

11-16:  Indicates that a read of the data cached on the initial "stb"
        is not readable from the cache. The low nibble of the error
        code is the slot number.

21-26:  Indicates an exception occurred on the flush operation.  This
        is the instruction which causes the block flush back to memory.
        The low nibble of the error code is the slot number.

31-36:  Indicates an exception occurred on the re-read of target byte.
        The low nibble of the error code is the slot number.

41-46:  Indicates a data error on the re-read on target byte. This is
        the instruction which causes the target block to be re-cached
        and supplied to the CPU. The low nibble of the error code is
        the slot number.
```

2.37 Test 24 - Memory Board Address Uniqueness Test

This test verifies the uniqueness of the upper bits of the memory address. For 16 Mbyte Memory Boards, the upper eight bits of the memory address is used. For 32 Mbyte Memory Boards, the upper seven bits of the memory address is used. For 128 Mbyte Memory Boards, the upper five bits of the memory address is used.

The test disables all Memory Boards by writting a zero to the enable register of each board identified in the configuation table in the Diagnostic RAM (see slot probe and configuration test). Then, for each board, the Base Address register is loaded with 0x00, the board is enabled, and the CPU performs memory accesses to addresses consisting of a walking one pattern on the upper significant bits of the address. For each of these accesses, a memory timeout fault should be generated.

The legal error codes for test 24, case 1 are:

```
01-06:  Indicates that the Memory Board responded when it should not
        have.  The base address register of the target board is set to
        0x00 and it responded to some other board address.  The low
        nibble of the error code is the slot number of the target
        Memory Board.

11-16:  Indicates that the wrong exception type occurred when the
        Memory Board was read.  The expected exception vector is
        9. The low nibble of the error code is the slot number of
        the target Memory Board.

21-26:  FCR <TOFM> bit was not set when exception occurred.  The low
        nibble of the error code is the slot number of the target memory
        board.

31-36:  FVAR register did not contain the correct address when the
        exception occurred.  The low nibble of the error code is the
        slot number of the target memory board.

41-46:  FCR was not cleared when FVAR was read.  The low nibble of the
        error code is the slot number of the target memory board.
```

The test is repeated using 0xff in the Base Address register and a walking zero pattern on the upper significant bits of the address.

The legal error codes for test 24, case 2 are:

51-56: Indicates that the Memory Board responded when it should not have. The base address register of the target board is set to 0x00 and it responded to some other board address. The low nibble of the error code is the slot number of the target Memory Board.

61-66: Indicates that the wrong exception type occurred when the Memory Board was read. The expected exception vector is 9. The low nibble of the error code is the slot number of the target Memory Board.

71-76: FCR <TOFM> bit was not set when exception occurred. The low nibble of the error code is the slot number of the target memory board.

81-86: FVAR register did not contain the correct address when the exception occurred. The low nibble of the error code is the slot number of the target memory board.

91-96: FCR was not cleared when FVAR was read. The low nibble of the error code is the slot number of the target memory board.

2.38 Test 25 - Memory Board Addressing Test

This test verifies that each installed Memory Board can respond to all 256 unique Memory Board addresses (0x00 through 0xff).

The test disables all Memory Boards by writting a zero to the enable register of each board identified in the configuation table in the Diagnostic RAM (see Slot Probe and Configuration test). Then, for each board, the Base Address register is iteratively loaded with an incrementing base address value (0 through 0xff in increments of "board size"), the board is enabled, and a write/read operation is performed to the target base address. The data written is the target base address. For each of these accesses, no faults should be generated and the data written should match the data read back.

The legal error codes for test 25 are:

01-06: Indicates that an exception occurred when the board address was written. The low nibble of the error code is the slot number of the target Memory Board.

11-16: Indicates that an exception occurred when flushing the target block back to memory. The low nibble of the error code is the slot number of the target Memory Board.

21-26: Indicates that an exception occurred when the board address was read. The low nibble of the error code is the slot number of the target Memory Board.

31-36: Indicates the wrong data was returned from the target Memory Board. The low nibble of the error code is the slot number of the target Memory Board.

2.39 Test 26 - Memory Board Block Addressability Test

This test verifies the uniqueness of the address lines on each installed Memory Board. The 16 Mbyte Memory Board address is broken down as follows:

```
0x00000000 - 0x007fffe0:  Low 8 Mbyte Bank
0x00800000 - 0x00ffffe0:  High 8 Mbyte Bank
```

Note that address bit 23 is the bank select bit. In this test, low and high banks are tested independently; first the low bank (bit 23 = 0) then the high bank (bit 23 = 1).

for 32 and 128 Mbyte Memory Boards, there is no special bank select bit.

The test starts with all installed Memory Boards disabled, then for each installed board, enables it for write/read, sets the base address register to zero, then writes each word in each block corresponding to a single address bit on (walking 1 on address bits 22:5) with its word address (address tag). Then, address zero is written with zero. Each block is then read back and verified to contain the correct word address tag patterns.

The test is repeated using a walking 0 on address bits (22:5) and ends by writing address `ffffe` with -1.

The legal error codes for test 26 are:

```
0X - Data fault exception occurred on ld instruction using walking
     0/1 address from Memory Board in slot X
1X - Data miscompare occurred on ld instruction using walking 0/1
     address from Memory Board in slot X
2X - Data fault exception occurred on ld instruction using all
     zeroes/ones address from Memory Board in slot X
3X - Data miscompare occurred on ld instruction using all zeroes/ones
     address from Memory Board in slot X

4X - Data fault exception occurred on st instruction using walking
     0/1 address on Memory Board in slot X
5X - Data fault exception occurred on ld instruction after data from
     Memory Board X was already cached.
6X - Data miscompare occurred on ld instruction after data from
     Memory Board X was already cached.

7X - Data fault exception occurred on st instruction using all
     zeroes/ones address on Memory Board in slot X
8X - Data fault exception occurred on ld instruction after data from
     Memory Board X was already cached.
9X - Data miscompare occurred on ld instruction after data from
     Memory Board X was already cached.
```

2.40 Test 27 - Memory Board RAM Addressing and Data Test

This is an addressing and data test for the first 1 Mbyte of memory. Only the first 1 Mbyte of memory is tested to keep execution time during power-up selftest to a minimum.

The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for information on the moving inverse test algorithm).

During the read-write-read sequence, the target memory block is cached and checked for the correct data. The data in the cache is then complemented and the block is flushed back to memory. The target block is then re-read and verified to contain the complemented data.

If burn-in jumper is present, the entire installed address space is tested.

Legal error codes for test 27 are:

```
1X - Data fault exception occurred during write of memory
     with initial data pattern.
2X - Data fault exception occurred on first read of forward pass
3X - Data miscompare occurred on first read of forward pass
4X - Data fault exception occurred during flush of target
     memory block back to memory during forward pass.
5X - Data fault exception occurred on second read of forward pass
6X - Data miscompare occurred on second read of forward pass
7X - Data fault exception occurred on first read of reverse pass
8X - Data miscompare occurred on first read of reverse pass
9X - Data fault exception occurred during flush of target
     memory block back to memory during reverse pass.
aX - Data fault exception occurred on second read of reverse pass
bX - Data miscompare occurred on second read of reverse pass
```

2.41 Test 28 - Cache Fill-Flush Test

This test fills the entire 64 Kbytes of cache RAM with the first 64 Kbytes of the bootrom code. Next, the second 64 Kbytes of bootrom code is then written to the cache. This should displace the contents of the cache out to physical memory.

The test then reads the first 64 Kbytes (which should displace the second 64 Kbytes in cache out to memory) and verifies that the data is correct. The second 64 Kbytes of memory are then read (which should cause the first 64 Kbytes to be displaced and the second 64 Kbytes to be re-cached) and verifies that the data is correct.

The legal error codes for test 28 are:

```
01 - Data fault exception occurred on st instruction to
     memory while loading cache with first 64 Kbytes of
     bootrom code.
02 - Data fault exception occurred on st instruction to
     memory while loading cache with second 64 Kbytes of
     bootrom code.

03 - Data fault exception occurred on ld instruction from
     memory while verifying first 64 Kbytes of data.
04 - Data miscompare occurred while verifying first 64 Kbytes
     of data.
05 - Data fault exception occurred on ld instruction from
     memory while verifying second 64 Kbytes data.
06 - Data miscompare occurred while verifying second 64 Kbytes
     of data.
```

2.42 Test 29 - Virtual Fault Cache Corruption Test

This test verifies that exceptions which occur due to cache writes do not corrupt the cache data. There are eight test cases.

The legal error codes for the eight cases in test 29 are:

Case 1: Single precision misaligned store exception to FF space

```
01 - Address Alignment fault did not occur on st to misaligned
     word address.
02 - First word of cache line corrupted on st to misaligned word
     address.
03 - Second word of cache line corrupted on st to misaligned word
     address.
```

Case 2: Double precision misaligned store exception to FF space

```
04 - Address Alignment fault did not occur on std to misaligned
     double-word address.
05 - First word of cache line corrupted on std to misaligned
     double-word address.
06 - Second word of cache line corrupted on std to misaligned
     double-word address.
```

Case 3: Single precision misaligned store operation with MMU enabled

```
07 - Address Alignment fault did not occur on st to misaligned
     word address.
08 - First word of cache line corrupted on st to misaligned word
     address.
09 - Second word of cache line corrupted on st to misaligned word
     address.
```

Case 4: Double precision misaligned store operation with MMU enabled

```
0a - Address Alignment fault did not occur on std to misaligned
     double-word address.
0b - First word of cache line corrupted on std to misaligned
     double-word address.
0c - Second word of cache line corrupted on std to misaligned
     double-word address.
```

Case 5: Single precision read only store exception

```
0d - Data fault exception did not occur on st to page marked read
     only in TLB.
0e - First word of cache line corrupted on st to page marked read
     only in TLB.
0f - Second word of cache line corrupted on st to page marked read
     only in TLB.
```

Case 6: Double precision read only store exception

```
10 - Data fault exception did not occur on std to page marked read
     only in TLB.
11 - First word of cache line corrupted on std to page marked read
     only in TLB.
12 - Second word of cache line corrupted on std to page marked read
     only in TLB.
```

Case 7: Single precision TLB miss store exception

```
13 - Data fault exception did not occur on st to page marked as
     invalid in TLB.
14 - First word of cache line corrupted on st to page marked as
     invalid in TLB.
15 - Second word of cache line corrupted on st to page marked as
     invalid in TLB.
```

Case 8: Double precision TLB miss store exception

```
16 - Data fault exception did not occur on std to page marked as
     invalid in TLB.
17 - First word of cache line corrupted on std to page marked as
     invalid in TLB.
18 - Second word of cache line corrupted on std to page marked as
     invalid in TLB.
```

2.43 Test 2a - Corrupted Block RAM Addressing and Data Test

This is an addressing and data test for the Corrupted Block RAM. A modified moving inverse test algorithm is performed in which only the forward pass is executed. (see Appendix A for more information on the moving inverse test algorithm).

The Corrupted Block RAM is a two Kbit RAM and is addressed using cache block addresses (i.e., 0, 0x20, 0x40..., etc.). The RAM contains 1 bit for each cache block. To reset a bit corresponding to a particular cache block, a write to boot space (space 0) must be performed using the address of the desired cache block.

To cause a bit in the corrupted block RAM to toggle from a zero to a one, a memory access which results in a timeout exception or ECC multibit exception must occur.

In this test, the Corrupted Block RAM is reset to initialize all the bits to zero (see test 0e). A TIR read is performed to verify that a particular location in the RAM is indeed zero. A memory read (with all Memory Boards disabled) is then executed to generate a memory timeout fault and target ram bit should togggle to 1. Another TIR read is executed to verify that the bit toggled.

The legal error codes for test 2a are:

```
01 - Corrupted Bit not zero on first read of forward pass
02 - No memory timeout fault was generated to target block address.
03 - FVAR does not contain the correct FF space address after memory
     timeout fault.
04 - Corrupted Bit did not toggle to one after memory timeout fault.
```

2.44 Test 2b - Corrupted Block Flush Inhibit Test

This test verifies that cache transactions which reference a corrupted block result in a Kbus timeout.

The test starts out by disabling all Memory Boards so that a memory timeout fault may be generated. Then address 0 is read to cause a memory timeout fault and set the corresponding bit in the Corrupted Block RAM. The Memory Boards are then re-enabled and address 0 is again accessed. This should cause a Kbus timeout fault

The legal error codes for test 2b are:

```
01 - No memory timeout fault was generated when Memory Boards
     disabled.
02 - No memory timeout fault was generated on reference to corrupted
     block.
03 - FCR TOFM bit not set
04 - FVAR does not contain the correct logical address
05 - FPAR does not contain the correct physical address
```

2.45 Test 2c - Cache Purge Transaction Test

This test verifies that the cache and bus watcher logic can correctly perform a cache purge operation. A purge operation is necessary when the contents of the purge RAM location addressed by the low 16 bits of the target physical address contains a valid entry and the 3 purge ram bits do not match bits 15:13 of the target virtual address. A purge operation is then required because only 1 reverse physical to virtual mapping can be maintained in the cache.

This test performs the following steps:

1. Invalidates bus watcher tags, TLB, and cache tags
2. Tags physical memory block 0 with `ff000000 - ff00001c`.
3. Tags physical memory block `0x10000` with `ff010000 - ff01001c`.
4. Invalidates bus watcher tags.
5. Invalidates TLB entries.
6. Makes TLB entry `0x2000` point to physical page address 0.
7. Makes TLB entry `0x4000` point to physical page address `0x10000`.
8. Reads logical address `0x2000` in Supervisor data space to set bus watcher tags for physical block 0 to VALID, OWNED. Reverse translation (address bits 15,14,13 = 001).
9. Invalidates physical and virtual cache tags.
10. Tags logical cache block `0x2000` with 0 tag pattern: PVALID = 1, POWN = 1, and writes physical address tags with 0.
11. Executes a "ld `[0x4000],reg`" instruction. This accesses physical block `0x10000` through the TLB.
12. There should be a virtual miss (logical address does not match virtual tags), and a physical miss (physical address does not match physical address tags).
13. The cache hit logic should cause the block at virtual index `0x2000` to be purged to physical address 0.

14. Following the purge of physical block 0, the bus watcher should issue a Cacheable Read for the block at physical address 0x10000 and memory should supply the data. The block should be cached at logical index 0x4000 in the cache. It should also be marked valid and owned. The block which was at logical address 0x2000 should be marked a invalid and unowned.
15. "reg" is verified to contain 0xff01000 (the memory block pattern that was cached).
16. The TIR is read using logical address 0x4000 and verifies that the OWN and VALID bits are true and that there is a virtual and physical tag match.
17. Reads the TIR using logical address 0x2000 and verifies that the OWN and VALID bits are false and that there is a virtual and physical tag match.
18. Reads and checks FPAR. It should contain the physical address of the last transaction run on Kbus (0x10000).

The legal error codes for test 2c are:

```
01 - Data returned on ld from logical address 0x4000 is not 0xff010000.
02 - Status of logical block 0x4000 is either invalid, unowned, or
     both invalid and unowned.
03 - Data read from logical block 0x2000 is not zero.
04 - Status of logical block 0x2000 is not invalid and unowned.
05 - FPAR does not contain 0x10000.
```

2.46 Test 2d - Cache Purge/Flush Transaction Test

This test verifies that the cache and bus watcher logic can correctly perform a cache purge and flush operation. A purge operation is performed when the contents of the purge RAM location addressed by the low 16 bits of the target physical address contains a valid entry and the 3 purge ram bits do not match bits 15:13 of the target virtual address. A purge operation is required because only 1 reverse physical to virtual mapping can be maintained in the cache. A flush operation is required when the logical block slot in the cache contains valid and owned data and that block slot must be used to contain a newly referenced cache block.

This test performs the following procedure:

1. Invalidates bus watcher tags, TLB, and cache tags
2. Tags physical memory block 0x20 with ff000020 - ff00003c.
3. Tags physical memory block 0x4020 with ff004020 - ff00403c.
4. Tags physical memory block 0x10020 with ff010020 - ff01003c.
5. Invalidates bus watcher tags.
6. Invalidates TLB entries.
7. Makes TLB entry 0x2000 point to physical page address 0.
8. Makes TLB entry 0x4000 point to physical page address 0x10000.
9. Makes TLB entry 0x14000 point to physical page address 0x4000.
10. Reads logical address 0x2020 in Supervisor data space to set bus watcher tags for physical block 0x20 to VALID, OWNED.

11. Writes logical address `0x14020` in Supervisor data space to set bus watcher tags for physical block `0x4020` to VALID, OWNED, DIRTY.
12. Invalidates physical and virtual cache tags.
13. Tags logical cache block `0x2020` with `0x20` tag pattern: PVALID = 1, POWN = 1, DIRTY = 0.
14. Tags logical cache block `0x14020` with `0x4020` tag pattern: PVALID = 1, POWN = 1, DIRTY = 1.
15. Executes a `st 0xffffffdf,[0x4020]` instruction. This accesses physical block `0x10020` through the TLB.
16. There should be a virtual miss (logical address != virtual address tags), and a physical miss (physical address != physical address tags).
17. The cache hit logic should cause logical cache block `0x2020` to be purged to physical address `0x20`. This block should be marked INVALID and UNOWNED.
18. The bus watcher issues an IOB transaction to flush logical cache block 0x14020 to physical address `0x4020`.
19. The bus watcher issues a R&I transaction to physical address 0x10020 and memory should supply the data. The data is cached at logical index 0x4020 in the cache.
20. A "ld [0x4020],reg" instruction is executed and "reg" is verified to contain `0xffffffdf`.
21. A "ld [0x4024],reg" instruction is executed and "reg" is verified to contain `0xff010024`.
22. Reads the TIR using logical address 0x4020 and verifies that the VALID, OWN and DIRTY bits are true, and that there is a virtual and physical tag match.
23. Reads the TIR using logical address 0x2020 and verifies that the VALID, OWN and DIRTY bits are false, and that there is a virtual and physical tag match.
24. Reads and checks FPAR. It should contain the physical address of the last transaction run on Kbus (0x10020).
25. A "ld [0x2020],reg" instruction is executed and "reg" is verified to contain 0x20.
26. A "ld [0x14024],reg" instruction is executed and "reg" is verified to contain 0x4020.

Legal error codes for test 2d are:

```
01 - Data returned on ld from logical address 0x4020 is not 0xffffffdf.
02 - Status of logical block 0x4020 is either invalid, unowned, or
     both invalid and unowned.
03 - Data returned on ld from logical address 0x4024 is not 0xff010024.
04 - Status of logical block 0x2020 is not invalid and unowned.
05 - FPAR does not contain 0x10020.
06 - Data returned on re-read of logical address 0x2020 is not 0x20.
07 - Data returned on re-read of logical address 0x14020 is not 0x4020.
```

2.47 Test 2e - Virtual Cache Block Replacement Test

This test exercises the cache and bus watcher cache block purge and flush logic by performing writes and reads to common physical addresses through all different logical addresses including FF space.

The test method is as follows:

1. After invalidating the TLB and Virtual cache tags, eight TLB entries are created which map logical address `0x10000, 0x12000, 0x14000,... 0x1e000` to physical addresses `0, 0x2000, 0x4000,... 0xe000`.
2. Eight valid, owned, and dirty block are created in the cache by executing store instructions to each of the eight logical addresses in the TLB. The data written to each word of the blocks is the physical addresss of the word. This step creates the reverse physical to logical mappings in the purgeram.
3. Eight new TLB entries are created which map all of the logical addresses previously used to target physical address 0 and the virtual cache tags are again invalidated.
4. Eight blocks are read using all eight logical addresses and the data is verified to be the correct physical address data for the target physical block. For blocks where the reverse physical to logical mapping does not match the logical address, a purge operation must be performed before the block being read can be cached.
5. The eight blocks are read again using FF space addresses and each block is verified to contain the correct physical address data.
6. Steps 1 through 5 are repeated for target physical addresses `0x2000, 0x4000, ... 0xe000`.
7. Steps 1 through 6 are repeated using a walking 1 pattern on logical address bits 31:16.

Legal error codes for test 2e are:

```
01 - Data fault exception occurred during creation of the valid
     owned and dirty cache blocks.
02 - Data fault exception occurred during read of target physical
     cache block.
03 - Physical data read through logical address does not match
     expected physical address data.
04 - Data fault exception occurred during read of target physical
     cache blocks using FF space addresses.
05 - Physical data read through FF space address does not match
     expected physical address data.
```

2.48 Test 2f - ECC Write/Read Test

This test verifies the ECC data path to and from each installed Memory Board. For each board, the test writes test patterns to each cache line in the target memory block then reads the block back with ECC enabled and verifies that no ECC exception (single bit or multi-bit) are generated. The test patterns are chosen such that all 256 ECC patterns (0 - 0xff) will be generated.

The legal error codes for test 2f are:

```
01 - ECCS or data fault exception occurred on store of data
     pattern with ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of test
     pattern back to memory.
03 - Single Bit ECC exception generated on re-read of test
     pattern from memory with ECC checking enabled.
04 - Multi-bit ECC exception generated on re-read of test
     pattern from memory with ECC checking enabled.
05 - Exception other than ECCS or ECCM generated on re-read
     of test pattern from memory with ECC checking enabled.
06 - Data error in first word of cache line 0
07 - Data error in second word of cache line 0
08 - Data error in first word of cache line 1
09 - Data error in second word of cache line 1
0a - Data error in first word of cache line 2
0b - Data error in second word of cache line 2
0c - Data error in first word of cache line 3
0d - Data error in second word of cache line 3
```

2.49 Test 30 - ECC Single Bit Correction to 1 Test

This test verifies that the ECC data correction logic can correct a bit from a zero to a one for all 64 data bit positions. The test is performed independly for each all four cache lines.

The test load the KCB register with the ECC pattern which corresponds to a data pattern with a single bit on (such as `0x0000000000000001`) then loads each line of the cache block except for the line under test with the corresponding data pattern. The cache line under test is loaded with an all zeroes data pattern. When the cache block is flushed to memory and re-read, a single bit error should be generated and the cache line under test should contain the corrected data (target bit should toggle from a 0 to a 1). The test verifies that the ECCS exception is generated, that, that the correct address and cache line is indicated in the FPAR register, that the correct syndrome byte is generated and latched in the FES register, and that all cache lines contain the correct data. The data for the target cache should match the data in the other cache lines.

The legal error codes for test 30 are:

```
01 - ECCS or data fault exception occurred on initial load
     of data patterns when ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of target
     cache block to memory when ECC checking disabled.
03 - No exception was generated on re-read of target cache block.
04 - Exception other than ECCS exception was generated on re-read
     of target cache block.
05 - FPAR register contains incorrect address.
06 - FES register contains incorrect syndrome value.
07 - Data error in cache line 0
08 - Data error in cache line 1
09 - Data error in cache line 2
0a - Data error in cache line 3
```

2.50 Test 31 - ECC Single Bit Correction to 0 Test

This test verifies that the ECC data correction logic can correct a bit from a one to a zero for all 64 data bit positions. The test is performed independly for each all 4 cache lines.

The test load the KCB register with the ECC pattern which corresponds to a data pattern with a single bit off (such as `0xfffffffffffffffe`) then loads each line of the cache block except for the line under test with the corresponding data pattern. The cache line under test is loaded with an all ones data pattern. When the cache block is flushed to memory and re-read, a single bit error should be generated and the cache line under test should contain the corrected data (target bit should toggle from a 1 to a 0). The test verifies that the ECCS exception is generated, that, that the correct address and cache line is indicated in the FPAR register, that the correct syndrome byte is generated and latched in the FES register, and that all cache lines contain the correct data. The data for the target cache should match the data in the other cache lines.

The legal error codes for test 31 are:

```
01 - ECCS or data fault exception occurred on initial load
     of data patterns when ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of target
     cache block to memory when ECC checking disabled.
03 - No exception was generated on re-read of target cache block.
04 - Exception other than ECCS exception was generated on re-read
     of target cache block.
05 - FPAR register contains incorrect address.
06 - FES register contains incorrect syndrome value.
07 - Data error in cache line 0
08 - Data error in cache line 1
09 - Data error in cache line 2
0a - Data error in cache line 3
```

2.51 Test 32 - ECC Single Bit Checkbyte Error Test

This test verifies that single bit errors in the checkbyte are detectable and causes no cache line data corruption.

Syndrome values which contain a single set bit correspond to single bit errors in the check byte. This test writes memory with a data pattern which will generate and all zeroes checkbyte but forces (via the KCB register) a checkbyte corresponding to a walking 1 pattern to be written along with the data. When the data is read back, the ECC checking logic should regenerate an all zeroes check byte and XOR it with the walking 1 checkbyte pattern. This should result in a single bit ECC error. However, the ECC correction logic should recognize that the error is in the checkbyte and not in the data word and the cache line should not be modified.

The legal error codes for test 32 are:

```
01 - ECCS or data fault exception occurred on initial load
     of data patterns when ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of target
     cache block to memory when ECC checking disabled.
03 - No exception was generated on re-read of target cache block.
04 - Exception other than ECCS exception was generated on re-read
     of target cache block.
05 - FPAR register contains incorrect address.
06 - FES register contains incorrect syndrome value.
07 - Data error in cache line 0
08 - Data error in cache line 1
09 - Data error in cache line 2
0a - Data error in cache line 3
```

2.52 Test 33 - ECC Multibit Error Detection Test

This test verifies that all syndrome values which map to a two bit or more than two bit error results in the generation of a multibit ECC exception.

The test contains a look-up table which specifies if a particular syndrom value maps to a single or multibit error. For each syndrome value which does not map to a single bit error, a test case is performed which uses the target syndrome byte as the ECC pattern forced to memory via the KCB register mechanism. The data pattern that is used during the memory write is chosen such that the ECC pattern which would normally be generated is 0x00. Under these conditions, the syndrome byte which will be generated when the memory block is re-read (causing a multi-bit error) will be the target syndrome byte (0x00 xor'ed with target syndrome results in generation of the same target syndrome). The test verifies that the ECCM exception is generated, that the FCR register ECCM bit is set, and that the FPAR register contains the correct faulted address.

The legal error codes for test 33 are:

```
01 - ECCS or data fault exception occurred on initial load
     of data patterns when ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of target
     cache block to memory when ECC checking disabled.
03 - No exception was generated on re-read of target cache block.
04 - Exception other than Data Fault exception was generated on
     re-read of target cache block.
05 - FCR ECCM bit not set.
06 - FVAR register contains incorrect address.
07 - FCR not cleared after read of FVAR.
```

2.53 Test 34 - ECC RAM Addressing and Data Test

This is an addressing and data test for the first megabyte of ECC memory. Only the first megabyte of memory are tested to keep execution time during power-up selftest to a minimum.

The test program performs a moving inverse test algorithm to verify the addressing and data paths to the ECC RAMs (see Appendix A for information on the moving inverse test algorithm).

The ECC RAMs are tested indirectly using the occurrance of an ECCS or ECCM exception as the error indication. 64 bit data patterns which correspond to checkbyte patterns of 0xaa and 0x55 are written to memory to indirectly load the ECC RAMs with the desired complementary data

patterns.

During the read-write-read sequence, the target memory block is cached and checked for the correct data. The data in the cache is then changed (so that the checkbyte will be complemented) and the block is flushed back to memory. The target block is then re-read and verified to contain the alternalte data pattern. The test is executed with ECC checking enabled so that any ECC RAM errors will result in an ECCS or ECCM exception.

If burn-in jumper is installed, the entire installed address space is tested.

Legal error codes for test 34 are:

```
1X - Data fault exception occurred during write of memory
     with initial data pattern.
2X - ECCS or data fault exception occurred on first read of
     forward pass
3X - Data miscompare occurred in upper 32 bits of cache line
     during forward pass
4X - Data miscompare occurred in lower 32 bits of cache line
     during forward pass
5X - Data fault exception occurred during flush of target
     memory block back to memory during forward pass.
6X - ECCS or data fault exception occurred on second read of
     forward pass
7X - Data miscompare occurred in upper 32 bits of cache line
     during forward pass
8X - Data miscompare occurred in lower 32 bits of cache line
     during forward pass

9X - ECCS or data fault exception occurred on first read of
     reverse pass
aX - Data miscompare occurred in upper 32 bits of cache line
     during reverse pass
bX - Data miscompare occurred in lower 32 bits of cache line
     during reverse pass
cX - Data fault exception occurred during flush of target
     memory block back to memory during reverse pass.
dX - ECCS or data fault exception occurred on second read of
     reverse pass
eX - Data miscompare occurred in upper 32 bits of cache line
     during reverse pass
fX - Data miscompare occurred in lower 32 bits of cache line
     during reverse pass
```

2.54 Test 35 - FPU Register Load/Store Test

This test verifies the primary interaction between the floating point unit and the memory system by performing a write/read test on one of the floating point register pairs. There are two test cases, one for single precision values and one for double-precision values. Data patterns consisting of all zeroes, all ones, walking 1 and walking 0 are written to the floating point register and read back and verified.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for test 35 are:

```
01 - After attempting to clear the QNE bit on the FPU state
     register, the queue (FQ) is still not empty.
02 - Write read error for single precision load/store.
03 - Write read error for double precision load/store (even register).
04 - Write read error for double precision load/store (odd register).
```

2.55 Test 36 - FPU State Register Test

The FPU state register (FSR) contains FPU mode and status information. This test performs a write/read test on this register using data patterns consisting of all zeroes, all ones, walking 1 and walking 0. The write-only bits of this register are masked-out for all patterns.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for test 36 are:

```
01 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
02 - FSR write read error.
```

2.56 Test 37 - FPU Add/Multiply/Divide Test

This test verifies the path between the FPC and the floating point arithmetic units on the FPC/FALU and the FPC/FMULT interfaces. The correct operation of both the FALU and the FMULT are henced tested by performing arithmetic operations and comparing the results with the correct expected results.

Resource and operand dependencies are forced in order to verify the that the FPU correctly handles them.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for test 37 are:

```
01 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
02 - Incorrect single precision addition result.
03 - Incorrect single precision multiplication result.
04 - Incorrect double precision addition result (even register).
05 - Incorrect double precision addition result (odd register).
06 - Incorrect double precision multiplication result (even register).
07 - Incorrect double precision multiplication result (odd register).
08 - Incorrect single precision division result.
09 - Incorrect double precision division result (even register).
0a - Incorrect double precision division result (odd register).
0b - FPU did not handled operand dependency correctly.
```

2.57 Test 38 - FPU Queue Test

The FPU queue (FQ) keeps tracks of floating point operations that are pending by the FPU when a floating point fp_exception trap occurs.

As per convention, the FPU queue should become empty after the execution of two successive STDFQ (Store Double Floating-point Queue) instructions. In order to test this function of the FPU, a floating point exception is forced by the test. When in exception mode, the queue should not be empty and the execution of two successive STDFQ instructions should cause it to become empty.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for this test are:

```
01 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
02 - FSR write read error while setting TEM (NV) bit.
03 - FPU fp_exception trap did not occur when expected.
04 - FSR QNE bit is clear when it should be set.
05 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
```

2.58 Test 39 - FPU Exceptions Test

There are two floating point trap types that are generated by the FPU hardware. These are: fp_disabled and fp_exception.

The FPU generates four types of exception traps:

1. FPC sequence error exception
2. Unimplemented floating point instruction exception [1]
3. Unfinished floating point instruction exception
4. IEEE exception

IEEE exceptions are classified as follows:

1. Invalid
2. Overflow
3. Underflow
4. Division by zero
5. Inexact

This test verifies that the FPU generates these traps and exceptions properly by performing test cases for each type. Each test case verifies that the exception logic behaves as expected for both cases: when enabled and disabled. The IEEE exception mechanism on the Weitek

[1] Not checked by this test. All instructions are implemented.

ALU/FPMULT/FPDIV chips is also exercised by this test.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for the 13 cases in test 39 are:

Test Case 1: fp_disabled trap

```
01 - FPU fp_disabled trap did not occur when expected.
```

Test Case 2: fp_exception IEEE-Invalid while enabled

```
02 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
03 - FPU fp_exception trap did not occur when expected (IEEE-Invalid).
04 - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Invalid.
```

Test Case 3: fp_exception IEEE-Invalid while disabled

```
05 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
06 - FPU fp_exception trap occured while traps were disabled (IEEE Invalid)
07 - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Invalid.
```

Test Case 4: fp_exception IEEE-Overflow while enabled

```
08 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
09 - FPU fp_exception trap did not occur when expected (IEEE-Overflow).
0a - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Overflow.
```

Test Case 5: fp_exception IEEE-Overflow while disabled

```
0b - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
0c - FPU fp_exception trap occured while traps were disabled (IEEE Overflow
0d - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Overflow.
```

Test Case 6: fp_exception IEEE-Underflow while enabled

```
0e - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
0f - FPU fp_exception trap did not occur when expected (IEEE-Underflow).
10 - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Underflow.
```

Test Case 7: fp_exception IEEE-Underflow while disabled

```
11 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
12 - FPU fp_exception trap occured while traps were disabled (IEEE Underflow).
13 - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Underflow.
```

Test Case 8: fp_exception IEEE-Inexact while enabled

```
14 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
15 - FPU fp_exception trap did not occur when expected (IEEE-Inexact).
16 - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Inexact.
```

Test Case 9: fp_exception IEEE-Inexact while disabled

```
17 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
18 - FPU fp_exception trap occured while traps were disabled (IEEE Inexact).
19 - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Inexact.
```

Test Case 10: fp_exception IEEE-Divide-By-Zero while enabled

```
1a - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
1b - FPU fp_exception trap did not occur when expected (IEEE-Divide-by-Zero).
1c - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Divide-by-Zero
```

Test Case 11: fp_exception IEEE-Divide-By-Zero while disabled

```
1d - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
1e - FPU fp_exception trap occured while traps were disabled.
1f - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Divide-By-Zero.
```

Test Case 12: fp_exception Sequence-Error

```
20 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
21 - FPU fp_exception trap did not occur when expected (IEEE-Divide-by-Zero).
22 - FPU fp_exception trap did not occur when expected (SEQUENCE).
23 - FPU fp_exception trap ocurred, but FSR FTT bits are not set
     for SEQUENCE.
```

Test Case 13: fp_exception Unfinished-Floating-Point-Instruction

```
24 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
25 - FPU fp_exception trap did not occur when expected (UNFINISHED_FPOP).
26 - FPU fp_exception trap ocurred, but FSR FTT bits are not set
     for UNFINISHED_FPOP
```

2.59 Test 3a - FPU Condition Codes Test

Floating point compares (FCMPS) and floating point condition (FBfcc) instructions interlock on the floating point condition codes. This condition codes are maintained by the FPU in the FSR. This test checks the proper updating of this bits in the FSR and the correct behavior of the FCMPS and FBfcc instructions.

The condition codes supported by the FPU are:

1. Equal Relation
2. Greater-Than Relation
3. Less-Than Relation
4. Unordered Relation

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for test 3a are:

```
01 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
```

Test Case 1: Equal Relation when A == B

```
02 - CC should reflect an equal relation, causing FBE
     instruction to fail.
03 - FSR FCC bits not reflecting an equal relation when expected.
```

Test Case 2: Equal Relation when A != B

```
04 - CC should not reflect an equal relation, causing FBE
     instruction to fail.
05 - FSR FCC bits reflecting an equal relation when not expected.
```

Test Case 3: Greater-Than Relation when A > B

```
06 - CC should reflect a greater-than relation, causing FBG
     instruction to fail.
07 - FSR FCC bits not reflecting a greater-than relation when expected.
```

Test Case 4: Greater-Than Relation when A < B

```
08 - CC should not reflect a greater_than relation, causing FBG
     instruction to fail.
09 - FSR FCC bits reflecting a greater-than relation when not expected.
```

Test Case 5: Less-Than Relation when A < B

```
0a - CC should reflect a less-than relation, causing FBL
     instruction to fail.
0b - FSR FCC bits not reflecting a less-than relation when expected.
```

Test Case 6: Less-Than Relation when A > B

```
0c - CC should not reflect a less_than relation, causing FBL
     instruction to fail.
0d - FSR FCC bits reflecting a less-than relation when not expected.
```

Test Case 7: Unordered Relation when A unordered, B ordered

```
0e - CC should reflect an unordered relation, causing FBU
     instruction to fail.
0f - FSR FCC bits not reflecting an unordered relation when expected.
```

Test Case 8: Unordered Relation when A & B ordered

```
10 - CC should not reflect an unordered relation, causing FBU
     instruction to fail.
11 - FSR FCC bits reflecting an unordered relation when not expected.
```

2.60 Test 3b - FPU Fast-Mode Enable Bit Test

When the FPU is in fast mode, the operations on denormalized numbers should not generate a fp_exception trap. While this is the case in fast mode, the case for IEEE mode is that it will result in an exception. This test enables the FPU to operate in fast mode by setting the corresponding bit in the FSR. A floating point operation is then performed on a denormalized number in order to verify that a trap does not occur.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for test 3b are:

```
01 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
02 - FAST mode was enable, but got an exception while operating
     on a denormalized number.
```

2.61 Test 3c - Frame Buffer Test

This is an addressing and data test for the frame buffer on the Graphics Boards configured into the system. This test supports the following Graphic Boards:

1. Monochrome Graphics Board (board-minor type = 0)
   Frame buffer size : 256k bytes
2. Color Graphics Board (board-minor type = 0x01)
   Overlay Plane 1 size : 128k bytes
   Overlay Plane 2 size : 128k bytes
   Image Plane size : 2M bytes

If an unknown Graphics Board is found in the system the test will not be performed.

The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for information on the moving inverse test algorithm).

This test is only executed in burn-in mode.

Legal error codes for test 3c are:

```
0X - Data fault exception occurred during writing to board
     control register for board in slot X
1X - Data fault exception occured during writing to board space
     register for board in slot X
2X - Accumulated miscompares during forward pass exceeded error limit
     on overlay planes/frame-buffer for board in slot X
3X - Accumulated miscompares during reverse pass exceeded error limit
     on overlay planes/frame-buffer for board in slot X
4X - Acccumulated miscompares during forward pass exceeded error limit
     on image plane for board in slot X
5X - Acccumulated miscompares during reverse pass exceeded error limit
     on image plane for board in slot X

6X - Data fault exception occurred during write of initial data
     pattern to frame buffer in slot X
7X - Data fault exception occurred on first read of forward pass
     for frame buffer in slot X
8X - Data fault exception occurred during complement write of forward
     pass for frame buffer in slot X
9X - Data fault exception occurred on second read of forward pass
     for frame buffer in slot X

aX - Data fault exception occurred on first read of reverse pass
     for frame buffer in slot X
bX - Data fault exception occurred during complement write of reverse
     pass for frame buffer in slot X
cX - Data fault exception occurred on second read of reverse pass
     for frame buffer in slot X

dX - Data fault exception occurred during writing to COLOR board video
     control register for board in slot X
eX - Data fault exception occurred during reading of COLOR board video
     control register for board in slot X
fX - Write/read error occurred on COLOR board video control register
     for board in slot X
```

2.62 Test 3d - System Board Interrupt Generation Test

This is a test of the interrupt register on the System Board and the ability of the System Board to generate all 16 vectors when enabled after reset.

The System Board interrupt register is an eight bit register which is written and read using byte RIO write/read operations.

The System Board interrupt logic queues up 16 interrupts whenever a system reset (power-up or software generated) occurs. The first time I/O interrupts are enabled after reset (by reading RIO address 17031000) the System Board sequentially sends 16 interrupt vectors (0x8f through 0x80) to the BID specified in the System Board interrupt register.

This test consists of two parts: Part 1, is a write/read test of the System Board interrupt register; and part 2 is a System Board directed interrupt test.

Part 1 first reads RIO address 17030000 to disable transmission of interrupts, then writes RIO address 17030000 with incrementing test patterns from 0 to 0xff. Each pattern is read back and verified.

The legal error codes for test 3d, part 1 are:

```
01 - Data fault exception occurred on initial read to disable
     System Board interrupt register.
02 - Data fault exception occurred on write of pattern to
     System Board interrupt register.
03 - Data fault exception occurred on read of System Board
     interrupt register.
04 - Data pattern written does not match data pattern read from
     System Board interrupt register.
```

Part 2 initializes the System Board interrupt register with the directed bit set and the destination ID set the BID of the CPU Board. System Board interrupts are then enabled by reading address 17031000. The test verifies that all 16 interrupt vectors are received correctly. Note that this test will fail if a system reset is not performed inbetween passes.

The legal error codes for test 3d, part 2 are:

```
05 - Timeout waiting to receive first interrupt (vector 0x8f) from
     System Board.
06 - Exeption other than Serial Interrupt Controller occurred.
07 - Higher priority interrupt vector was received 256 times without
     receiving expected vector.
8X - Lower priority interrupt vector was received. Error code is the
     vector which was expected.
```

2.63 Test 3e - Serial Port Reset Test

This test verifies the reset state of both Z8530 SCC chips on the System Board, controlling the keyboard/mouse and serial ports A/B. The SCC chip is given a command to reset for each of its ports (channels). Upon channel reset, the state of some (R0, R1, R3, R10, R15) SCC registers is then compare with an expected pattern.

This test is only executed in burn-in mode.

The legal error codes for test 3e are:

```
10 - Data fault exception occurred during resetting of
     mouse port.
2X - Data fault exception occurred during resetting of port
     X
3X - Unexpected/Invalid reset state of portX
```

The ports are assigned the following port numbers:

```
Keyboard Port = 0
Mouse Port    = 1
Serial Port A = 2
Serial Port B = 3
```

2.64 Test 3f - Serial Port Internal Loopback Test

This test performs an internal loopback test of both Z8530 SCC chips on the System Board, controlling the keyboard/mouse and serial ports A/B. The SCC chip is given a command to put each of its ports into internal loopback mode. Upon programming each channel for internal loopback, a series of patterns are send to the channel, read back and compared with what was sent to determine if any of the data lines on that channel are in error.

This test is only executed in burn-in mode.

The legal error codes for test 3f are:

```
10 - Data fault exception occurred during resetting of mouse
     port
2X - Data fault exception occurred during resetting of port X
3X - Data fault exception occurred during programming of port X
4X - Receive error on port X
5X - Transmit error on port X
6X - Timeout  while waiting for Receive Character Available interrupt
     on port X
7X - Incorrect interrupt vector receive while waiting for Receive
     Character Available interrupt on port X
8X - Data miscompare (write/read) error on port X
9X - Receive Character Available interrupt pending bit is inactive
     (Z8530 RR2 Register) on port X
aX - Timeout while waiting for Transmit Buffer Empty interrupt on
     port X
bX - Incorrect interrupt vector receive while waiting for Transmit
     Buffer Empty interrupt on port X
cX - Transmit Buffer Empty interrupt pending bit is inactive
     (Z8530 RR2 register) on port X
```

The ports are assigned the follwing port numbers:

```
Keyboard Port = 0
Mouse Port    = 1
Serial Port A = 2
Serial Port B = 3
```

Section 3: Series5 Test Descriptions

3.1 Introduction

This section describes the system power-on self-tests that are used on Solbourne Series5 systems. In the these test descriptions, all test numbers and error codes are represented in two digit hex values.

3.2 Test 01 - Bootrom Checksum Test

This test computes the checksum on the contents of the four, 27C512 EPROMS on the CPU Board that are used to contain the boot code and data. The expected checksum is burned into the EPROM when the ROMs are programed during manufacturing.

There are eight, one byte checksums that are computed. Checksum 0 is generated by summing every eighth byte starting at location 0 in the bootroms. Checksum 1 is generated by summing every eighth byte starting at location 1 in the bootrom, and so on.

Legal error codes for this test are:

```
00 - Checksum 0 incorrect
01 - Checksum 1 incorrect
02 - Checksum 2 incorrect
03 - Checksum 3 incorrect
04 - Checksum 4 incorrect
05 - Checksum 5 incorrect
06 - Checksum 6 incorrect
07 - Checksum 7 incorrect
```

3.3 Test 02 - Diagnostic RAM Addressing and Data Test

This is an addressing and data test for the diagnostic RAM on the CPU Board. The diagnostic RAM a two Kbyte static RAM which is accessed through alternate space (ASI) 0xe0 (see H/W description), and responds to every fourth address in the range 0 through 0x1ffc inclusive.

The test program performs a moving inverse test algorithm (see Appendix B for information on the moving inverse test algorithm).

Legal error codes for this test are:

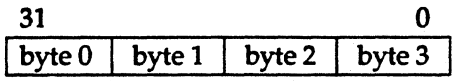
```
01 - Data error on first forward pass read
02 - Data error on second forward pass read
03 - Data error on first reverse pass read
04 - Data error on second reverse pass read
```

3.4 Test 03 - Control-Data Bus Test

This test reads the quiescent (undriven) state of the CPU's 32-bit, control data bus 64 Kbyte times and verifies that the bus is floats high (all ones).

A common cause of failure for this test is that one of the bootrom outputs is sinking too much current and pulling the control data bus to the low state. The test examines each eight-bit field of the 32-bit bus and reports errors using the following convention:

Control data bus format



Error encoding convention:

```
01 - byte 3 corrupted
02 - byte 2 corrupted
03 - bytes 2 and 3 corrupted
04 - byte 1 corrupted
05 - bytes 1 and 3 corrupted
06 - bytes 1 and 2 corrupted
07 - bytes 1, 2, and 3 corrupted
08 - byte 0 corrupted
09 - bytes 0 and 3 corrupted
0a - bytes 0 and 2 corrupted
0b - bytes 0, 2, and 3 corrupted
0c - bytes 0 and 1 corrupted
0d - bytes 0, 1 and 3 corrupted
0e - bytes 0, 1 and 2 corrupted
0f - bytes 0, 1, 2 and 3 corrupted
```

3.5 Test 04 - Control Registers Test

This test verifies that the PDBA register can be written and read. Aside from the interrupt registers (see test 08) this is the only other register that is write-readable.

All 32 bits of the PDBA register are tested.

Legal error codes for this test case are:

```
01 - PDBA register write/read error
```

3.6 Test 05 - GTLB/MTRAN Bus Data Test

This test verifies that data path to the GTLB translation data and the PTE permissions bits are unique across the MTRAN bus. The first part of the test writes ones to the GTLB at index 0 and zeroes at index 1. The data is read back and verified. This verifies the GTLB to MTRAN drivers are enabled.

Note that when the GTLB is written, not only does the page entry data get written, the GTLB tags and status are written as well. This test verifies the value of the TIR to be what is expected for every test case. The test accesses the instruction section of the GTLB through ASI 0xa8 on write and 0xb1 on read. When accesses to ASI 0xbx occur, the TIR is set with the status of the GTLB tags and permissions of the accessed address. The TIR is used to verify the permission bits of the PTE. The section of the test uses only GTLB index 0.

There are five cases for test 05, as follows:

1. Write ones to the GTLB at index 0 and zeroes at index 1. The data is read back and verified.

   The legal error codes for test 05, case 1 are:

   ```
   01 - The instruction GTLB physical address field at index 0
        does not contain the data that was written.
   02 - The instruction GTLB physical address field at index 1 does
        not contain the data that was written.
   ```

2. Walk a one across the status bits of the GTLB entry

   The legal error codes for test 05, case 2 are:

   ```
   03 - The data GTLB physical address field does not contain the
        data that was written.
   04 - The data GTLB tag and status field does not contain the data
        that was written.
   ```

3. Walk a one across the physical address field of the GTLB entry

   The legal error codes for test 05, case 3 are:

   ```
   05 - The data GTLB physical address field does not contain the
        data that was written.
   06 - The data GTLB tag and status field does not contain the data
        that was written.
   ```

4. Walk a zero across the status bits of the GTLB entry

   The legal error codes for test 05, case 4 are:

   ```
   07 - The data GTLB physical address field does not contain the
        data that was written.
   08 - The data GTLB tag and status field does not contain the data
        that was written.
   ```

5. Walk a zero across the physical address field of the GTLB entry

   The legal error codes for test 05, case 5 are:

   ```
   09 - The data GTLB physical address field does not contain the
        data that was written.
   0a - The data GTLB tag and status field does not contain the data
        that was written.
   ```

3.7 Test 06 - GTLB RAM Addressing and Data Test

This is a test of the physical address field of the GTLB RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix B for information on the moving inverse test algorithm).

This test treats the GTLB as one big RAM area, that is, that addressing is linear at the GTLB RAM itself. The test compensates for the address bits that are connected to the sparc ASI bit 1 and selection of FE/FF access areas of the RAM.

The legal error codes for test 06 are:

```
01 - Data error on first forward pass read
02 - Data error on second forward pass read
03 - Data error on first reverse pass read
04 - Data error on second reverse pass read
```

3.8 Test 07 - ROM Addressing Test

This is a read test of FE space data through the PTRAN address multiplexor. The test sets up 32 entries in the FE/FF section of the GTLB to force bits <18:13> to have unique data presented at each side of the PTRAN multiplexor. The two different words are read from ROM 0 space and two from FE space, the data is compared and an error is presented if the data does not match. One word is read from address and another is read from address + 4.

Legal error codes for this test are:

```
01 - Data mismatch on first word read
02 - Data mismatch on second word read
```

3.9 Test 08 - Interrupt Registers Test

This is a write/read test of the interrupt registers. There are four test cases, one for each register tested. The registers tested are: Device ID Register (DIR), Interrupt Priority Register (IPR), Interrupt Transmit Register (IXR), and the Interrupt Pending Vector Register (IPV). For each register tested, data patterns consisting of all zeroes, all ones, walking 1 and walking 0 are written to the register under test and read back and verified.

Legal error codes for this test are:

```
01 - DIR register write read error
02 - IPR register write read error
03 - IXR register write read error
04 - IPV register write read error
```

3.10 Test 09 - Directed Interrupt Test

This test verifies that the CPU Board interrupt logic can send a directed interrupt to itself. There are two test cases:

1. Verifies that directed interrupts can be transmitted and received
2. Verifies that the interrupt receiver priority level (set in the IPR register) effectively inhibits interrupts from being received.

The test uses the CPU's own board address (read from the BID register) as the target device to which the interrupt is directed. During the transmission and receiving of the directed interrupts, the basic operation of the ITXC and IRXC register is verified. (see the Kbus Interrupt Specification for more information on the interrupt transmit/receive protocol).

Case 1 sets the IPR to 0 then walks a 1 bit across the vector field of the transmitted interrupt and verifies that all interrupts are received.

Legal error codes for test 09, case 1 are:

```
01 - Interrupt was never acknowledged (ITXC<gone> bit not set).
02 - Interrupt was acknowledged but no incorrect interrupt
     type was generated.
03 - Interrupt occurred, but the IRXC<P> bit was not set.
04 - Interrupt occurred, but the IPV register did not
     contain the transmitted vector.
```

Case 2 sets the IPR to 0xff then walks a 0 bit across the vector field of the transmitted interrupt and verifies that no interrupts are received.

Legal error codes for test 09, case 2 is:

```
05 - Interrupt was acknowledged (ITXC<gone> bit set).
06 - Unexpected interrupt was generated
```

3.11 Test 0a - GTLB TAG Addressing and Data Test

This is a test of the TAG, and status fields of the GTLB. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix B for information on the moving inverse test algorithm).

The legal error codes for test 0a are:

```
01 - Tag or status error on first forward pass read
02 - Tag or status error on second forward pass read
03 - Tag or status error on first reverse pass read
04 - Tag or status error on second reverse pass read
```

3.12 Test 0b - GTLB Tag Match Test

This is a test of the GTLB Tag RAM chips. The test program loads a series of patterns into the tag portion of the GTLB then performs a series of TIR reads to verify that the GTLB tag RAMs work correctly (see Appendix B for more information on TIR reads).

There are six cases for test 0b, as follows:

1. GTLB tag set to walking 1 pattern with logical address set to zero.

   The legal error codes for test 0b, case 1 are:

```
01 - GTLB tag RAM match error using instruction GTLB.
02 - GTLB tag RAM match error using data GTLB.
```

2. GTLB tag set to walking 0 pattern with logical address set to all ones.

   The legal error codes for test 0b, case 2 are:

```
03 - GTLB tag RAM match error using instruction GTLB.
04 - GTLB tag RAM match error using data GTLB.
```

3. GTLB tag set to zero with logical address set to walking 1 pattern.

   The legal error codes for test 0b, case 3 are:

```
05 - GTLB tag RAM match error using instruction GTLB.
06 - GTLB tag RAM match error using data GTLB.
```

4. GTLB tag set to all ones with logical address set to walking zero pattern.

   The legal error codes for test 0b, case 4 are:

```
07 - GTLB tag RAM match error using instruction GTLB.
08 - GTLB tag RAM match error using data GTLB.
```

5. GTLB tag set to all ones, clear GTLB tags with clear GTLB tags ASI, verify with logical address set to walking one pattern.

   The legal error codes for test 0b, case 5 are:

```
09 - GTLB tag RAM match error using instruction GTLB.
0a - GTLB tag RAM match error using data GTLB.
```

6. GTLB tag set to all ones, clear GTLB tags with clear GTLB/FTLB tags ASI, verify with logical address set to walking one pattern.

   The legal error codes for test 0b, case 6 are:

```
0b - GTLB tag RAM match error using instruction GTLB.
0c - GTLB tag RAM match error using data GTLB.
```

3.13 Test 0c - FTLB/TAGADD Bus Data Test

This test verifies that data path from the GTLB/MTRAN/FTLB input to the FTLB translation data and the PTE permissions bits are unique across the TAGADD bus to the FTIR.

This test verifies the value of the FTIR and TIR to be what is expected for every test case. The test accesses the instruction section of the FTLB through ASI 0xa8 on write. The FTLB is read indirectly by causing an access to the cache but ignoring the data. The translation used is then read from the FTIR and the FTLB match status is read from the TIR. The ASI used to cause this operation is 0x70. This test uses only FTLB index 0.

There are five cases for test 0c, as follows:

1. Walk a one across the status bits of the FTLB entry

   The legal error codes for test 0c, case 1 are:

   ```
   01 - The instruction FTLB physical address field read from the
        FTIR does not contain the data that was written.
   02 - The instruction FTLB tag and status field does not contain
        the data that was written.
   ```

2. Walk a one across the physical address field of the FTLB entry

   The legal error codes for test 0c, case 2 are:

   ```
   03 - The instruction FTLB physical address field read from the
        FTIR does not contain the data that was written.
   04 - The instruction FTLB tag and status field does not contain
        the data that was written.
   ```

3. Walk a zero across the status bits of the FTLB entry

   The legal error codes for test 0c, case 3 are:

   ```
   05 - The instruction FTLB physical address field read from the
        FTIR does not contain the data that was written.
   06 - The instruction FTLB tag and status field does not contain
        the data that was written.
   ```

4. Walk a zero across the physical address field of the FTLB entry

   The legal error codes for test 0c, case 4 are:

   ```
   07 - The instruction FTLB physical address field read from the
        FTIR does not contain the data that was written.
   08 - The instruction FTLB tag and status field does not contain
        the data that was written.
   ```

3.14 Test 0d - FTLB RAM Addressing and Data Test

This is a test of the physical address, TAG and status fields of the FTLB rams. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see

Appendix B for information on the moving inverse test algorithm).

The legal error codes for test 0d are:

```
01 - Data error on first forward pass read
02 - Tag or status error on first forward pass read
03 - Data error on second forward pass read
04 - Tag or status error on second forward pass read
05 - Data error on first reverse pass read
06 - Tag or status error on first reverse pass read
07 - Data error on second reverse pass read
08 - Tag or status error on second reverse pass read
```

3.15 Test 0e - FTLB Tag Match Test

This is a test of the FTLB Tag RAM chips. The test program loads a series of patterns into the tag portion of the FTLB then performs a series of TIR reads to verify that the FTLB tag RAMs work correctly (see Appendix B for more information on TIR reads).

There are six cases for test 0e, as follows:

1. FTLB tag set to walking 1 pattern with logical address set to zero.

   The legal error codes for test 0e, case 1 are:

   ```
   01 - FTLB tag RAM match error using instruction FTLB.
   02 - FTLB tag RAM match error using data FTLB.
   ```

2. FTLB tag set to walking 0 pattern with logical address set to all ones.

   The legal error codes for test 0e, case 2 are:

   ```
   03 - FTLB tag RAM match error using instruction FTLB.
   04 - FTLB tag RAM match error using data FTLB.
   ```

3. FTLB tag set to zero with logical address set to walking 1 pattern.

   The legal error codes for test 0e, case 3 are:

   ```
   05 - FTLB tag RAM match error using instruction FTLB.
   06 - FTLB tag RAM match error using data FTLB.
   ```

4. FTLB tag set to all ones with logical address set to walking zero pattern.

   The legal error codes for test 0e, case 4 are:

```
07 - FTLB tag RAM match error using instruction FTLB.
08 - FTLB tag RAM match error using data FTLB.
```

5. FTLB tag set to all ones, clear FTLB tags with clear FTLB tags ASI, verify with logical address set to walking one pattern.

   The legal error codes for test 0e, case 5 are:

```
09 - FTLB tag RAM match error using instruction FTLB.
0a - FTLB tag RAM match error using data FTLB.
```

6. FTLB tag set to all ones, clear FTLB tags with clear GTLB/FTLB tags ASI, verify with logical address set to walking one pattern.

   The legal error codes for test 0e, case 6 are:

```
0b - FTLB tag RAM match error using instruction FTLB.
0c - FTLB tag RAM match error using data FTLB.
```

3.16 Test 0f - Corrupted Block RAM Reset Test

This test verifies that all the bits in the Corrupted Block RAM can be reset. To clear the Corrupted Block RAM, a write to special ASI 0xf8 must be performed for each block address. The Corrupted Block RAM is a four Kbit RAM and is addressed using cache block addresses (i.e., 0, 0x20, 0x40..., etc.).

This test first resets the Corrupted Block RAM as described above then performs TIR read operations to obtains the Corrupt status for each block address (see Appendix B for more information about TIR read operations).

The legal error code for test 11 is:

```
01 - Corrupted bit not zero after reset
```

3.17 Test 10 - Cache Tag RAM Address and Data Test

This is an addressing and data test for the cache tag RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix B for information on the moving inverse test algorithm).

This test uses a special test feature of the CPU Board which allows cache accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to memory). The cache tags are loaded using the cache tag load ASI's 0x90, 0x92, 0x98 and 0x9a. This test performs all cache accesses using ASI 0x6a for reads, ASI 0x9a for writes to the tags to set the OWN bit and ASI 0x98 for writes to the tags to clear the OWN bit.

The cache tags are indexed by physical address bits 16:5. In other words, there are 4 Kbyte TAG locations from address 0 through 0x1ffe0. Each entry corresponds to a cache block address, i.e., 0, 0x20, 0x40, etc. The cache tag data is supplied by physical address bits 27:17 which is read

from the TLB Physical address field.

The cache tags are invalidated before the test is started. For each cache access, the TLB is loaded with a logical to physical mapping in which the logical address is the desired physical tag RAM index (0 through 0x1ffe0) and the physical address field is the desired data to be written to the cache tags (0x0aaa0000 or 0x05540000).

The cache tags are initialized to 0x0aaa0000, the VALID bit is set, and the OWN bit is cleared.

During the the read-write-read test sequence, a TIR read is performed to verify that the cache tag RAM contains the correct background tag (CM0/1/2 are true) and that OWN is false. Then, the physical address field in the TLB is complemented, and another TIR read is performed to verify that the CM status bits are false and the OWN status is unchanged. Complementary tag, and OWN data is then written to the target cache tag location by performing another cache hit operation (ASI 0x6a). Finally, a TIR read is performed to verify that the cache tag RAM contains the correct tag data (by verifying CM0/1/2 are true) and that OWN is true.

The legal error codes for test 0f are:

```
01 - Cache tag match or status error on first forward pass read
02 - Cache tag match or status error when TLB physical address
     field was complemented on forward pass
03 - Cache tag match or status error on second forward pass read

04 - Cache tag match or status error on first reverse pass read
05 - Cache tag match or status error when TLB physical address
     field was complemented on reverse pass
06 - Cache tag match or status error on second reverse pass read
```

3.18 Test 11 - Cache Tag Match Test

This is a test of the Cache Tag match detection logic. The test program loads a series of patterns into the cache tags then performs a series of TIR reads to verify that the cache tag match comparitors works correctly (see Appendix B for more information on TIR reads).

The CM0 signal represents a match between cache address bits 20:17 and bits 20:17 of the physical address field of the TLB; the CM1 signal represents a match between cache address bits 24:21 and bits 24:21 of the physical address field of the TLB; and the CM2 signal represents a match between cache address bits 27:25 and bits 27:25 of the physical address field of the TLB. The CM2 signal also represents the Valid signal for the cache tags.

There are 5 test cases as outlined below:

1. Cache tags set to walking 1 pattern with physical address field of TLB set to zero.

   The legal error codes for test 10, case 1 are:

   ```
   01 - Cache tag status error
   ```

2. Cache tags set to walking 0 pattern with physical address field of TLB set to all ones.

   The legal error codes for test 10, case 2 are:

```
02 - Cache tag status error
```

3. Cache tags set to zero with physical address field of TLB set to walking 1 pattern.

   The legal error codes for test 10, case 3 are:

```
03 - Cache tag status error
```

4. Cache tags set to all ones with physical address field of TLB set to walking zero pattern.

   The legal error codes for test 10, case 4 are:

```
04 - Cache tag status error
```

5. Cache tags set to all ones, clear cache tags with ASI 0x94 (cache tag clear ASI), and us physical address field of TLB set to walking zero pattern.

   The legal error codes for test 10, case 5 are:

```
05 - Cache tag status error
```

3.19 Test 12 - Cache RAM Bank Uniqueness Test

This test verifies the the cache RAM bank selection mechanism works. The cache RAM bank is selected on the basis of logical address bit 2. The test also verifies that byte, half-word, word and double-word loads and stores to the cache can be performed. It is verified for each access type, that the data is placed in the correct byte/half-word/word/double-word position in the cache.

This test uses a special test feature of the CPU Board which allows caches accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to memory). This test performs all cache accesses with ASI 0x4b.

This test writes patterns of various sizes into the first eight bytes (addresses 0-7) of the cache. The seqeunces of pattern writes and reads and associated error codes are shown below:

Initial state: Address `0` written with `0x55555555`, address 4 written with `0xaaaaaaaa`.

Byte:	0	1	2	3	4	5	6	7
Data:	55	55	55	55	aa	aa	aa	aa

```
Error code 01 - word read at address 0 is not 0x55555555
Error code 02 - word read at address 4 is not 0xaaaaaaaa
```

Second state: Address 0 written with `0xaaaa`, address 4 written with `0x5555`.

Byte:	0	1	2	3	4	5	6	7
Data:	aa	aa	55	55	55	55	aa	aa

```
Error code 03 - word read at address 0 not 0xaaaa5555
Error code 04 - word read at address 4 not 0x5555aaaa
Error code 05 - double byte read at address 2 not 0x5555
Error code 06 - double byte read at address 6 not 0xaaaa
```

Third state: Address 0 and 7 written with 0x55, address 3 and 4 written with 0xaa.

Byte:	0	1	2	3	4	5	6	7
Data:	55	aa	55	aa	aa	55	aa	55

```
Error code 07 - word read at address 0 not 0x55aa55aa
Error code 08 - word read at address 4 not 0xaa55aa55
Error code 09 - double byte read at address 0 not 0x55aa
Error code 0a - double byte read at address 4 not 0xaa55
Error code 0b - byte read at address 0 not 0x55
Error code 0c - byte read at address 4 not 0xaa
Error code 0d - byte read at address 1 not 0xaa
Error code 0e - byte read at address 5 not 0x55
Error code 0f - byte read at address 2 not 0x55
Error code 10 - byte read at address 6 not 0xaa
Error code 11 - byte read at address 3 not 0xaa
Error code 12 - byte read at address 7 not 0x55
```

Fourth state: Address 0 written with `0xaaaaaaaa55555555` (double word write).

Byte:	0	1	2	3	4	5	6	7
Data:	aa	aa	aa	aa	55	55	55	55

```
Error code 13 - double word read at address 0, first word not 0xaaaaaaaa
Error code 14 - double word read at address 0, second word not 0x55555555
```

3.20 Test 13 - Cache RAM Addressing and Data Test

This is an addressing and data test for the Cache Data RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix B for information on the moving inverse test algorithm).

This test uses a special test feature of the CPU Board which allows caches accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to memory). This test performs all cache accesses with ASI 0x4b.

The legal error codes for test 13 are:

```
01 - Data error on first forward pass read
02 - Data error on second forward pass read
03 - Data error on first reverse pass read
04 - Data error on second reverse pass read
```

3.21 Test 14 - Flush RAM Addressing and Data Test

This is an addressing and data test for the Flush RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix B for information on the moving inverse test algorithm).

This test uses a special test feature of the CPU Board which allows caches accesses to be made without generating a cache miss (which would cause the bus watcher to run a Kbus cycle to memory). This test performs all cache accesses with ASI 0x90 (a set cache tag ASI).

The flush RAM is indexed using physical address bits 16:5 and contains 12 bits of information; the physical address bits 28:17. For each entry in the physical cache, there is a corresponding entry in the flush RAM which contains the physical flush address. The flush RAM is loaded with physical address bits 28:17 and is set whenever the cache tags are loaded.

The cache tags are invalidated before the test is started. For each cache access, the TLB is loaded with a logical to physical mapping in which the physical address is the desired state of bits 28:17 (`0xaaa0000` or `0x15540000`) and the physical address is the desired flush RAM index (`0` through `0x1ffe0`)

During the read-write-read sequence, a read operation is performed (using the special ASI, 0xb5) to verify that the data in the flush RAM contains the correct data.

The legal error codes for this test are:

```
01 - flush address data error on first read of forward pass
02 - flush address data error on second read of forward pass
03 - flush address data error on first read of reverse pass
04 - flush address data error on second read of reverse pass
```

3.22 Test 15 - Dirty Block RAM Addressing and Data Test

This is an addressing and data test for the Dirty Block Data RAM. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix B for information on the moving inverse test algorithm).

The test writes the Dirty Block RAM, through special ASI's 0x43 or 0x4b for each block address. The Dirty Block RAM is a four Kbit RAM and is addressed using cache block addresses (i.e., 0, 0x20, 0x40..., etc.). The test then accesses the cache using ASI 0x6b and performs a TIR read operation to obtain the Dirty status for each block address (see Appendix B for more information about TIR read operations).

The legal error codes for test 15 is:

```
01 - Data error on first forward pass read
02 - Data error on second forward pass read
03 - Data error on first reverse pass read
04 - Data error on second reverse pass read
```

3.23 Test 16 - MMU Fault Test

This test verifies that all types of MMU exceptions can be generated.

For all MMU exceptions, the FCR is verified to be updated with the correct cause of the exception (TMISS, UP, WP, POF) and the FVAR is verified to contain the unmapped (Virtual) address used by the access.

The 15 cases for test 16 follow:

Case 1: TMISS fault (TLBINV true)

This test case maps logical address patterns to physical address zero, but the TLB entries are invalid (sets TLBINV). The logical address pattern is then used as the address in ld instruction. This causes a TLB miss to occur. The FCR snd FVAR are read after the exception is verified.

The legal error codes test 16, case 1 are:

```
01 - Data exception was not generated
02 - FCR<TMISS> bit was not set
03 - FVAR register did not contain correct value when read
```

Case 2: TMISS fault (TLBINV false) AND (GM1 = false) AND (GM0 = true)

This test case maps logical address zero to physical address zero to create a valid TLB entry in TLB location zero, then performs a ld instruction at logical address `0x0a000000`. This causes a TLB miss to occur due to a tag mismatch. The FCR and FVAR are read after the exception is verified.

The legal error codes test 16, case 2 are:

```
04 - Data exception was not generated
05 - FCR TMISS fault bit was not set
06 - FVAR register did not contain correct value when read
```

Case 3: TMISS fault (TLBINV false) AND (GM1 = true) AND (GM0 = false)

This is the same as case 2 except that `0x50000000` is used for the logical address for the ld instruction (GM1 = true, GM0 = false).

The legal error codes test 16, case 3 are:

```
07 - Data exception was not generated
08 - FCR TMISS fault bit was not set
09 - FVAR register did not contain correct value when read
```

Case 4: UPF fault (UP true)

This test case maps logical address `0xaaaaaaaa` to physical address 0 and sets the UP bit (user protect) in the TLB, the FTLB is invalidated, then accesses logical address `0xaaaaaaaa` through user data space to cause a GTLB UP fault. The FCR and FVAR are read after the exception is verified.

The legal error codes test 16, case 4 are:

```
0a - Data exception was not generated
0b - FCR UPF bit was not set
0c - The FCR was found with more than just the UPF bit set
0d - FVAR register did not contain correct value when read
```

Case 5: UPF fault (UP true)

This is the same as case 4 except logical address `0x55555555` is used.

The legal error codes test 16, case 5 are:

```
0e - Data exception was not generated
0f - FCR UPF bit was not set
10 - The FCR was found with more than just the UPF bit set
11 - FVAR register did not contain correct value when read
```

Case 6: UPF fault (UP true)

This test case maps logical address `0xaaaaaaaa` to physical address 0 and sets the UP bit (user protect) in the TLB, then accesses a half word at logical address `0xaaaaaaaa` through user data space to cause a FTLB UP fault. The FCR and FVAR are read after the exception is verified.

The legal error codes test 16, case 6 are:

```
12 - Data exception was not generated
13 - FCR UPF bit was not set
14 - The FCR was found with more than just the UPF bit set
15 - FVAR register did not contain correct value when read
```

Case 7: UPF fault (UP true)

This is the same as case 6 except an atomic ldstub through logical address 0x55555555 is used.

The legal error codes test 16, case 7 are:

```
16 - Data exception was not generated
17 - FCR UPF bit was not set
18 - The FCR was found with more than just the UPF bit set
19 - FVAR register did not contain correct value when read
```

Case 8: UPF fault (FE space)

This test case loads a byte from logical address `0xff555555` through user data space to cause a User Protection fault. The FCR and FVAR are read after the exception is verified.

The legal error codes test 16, case 8 are:

```
1a - Data exception was not generated
1b - FCR UPF bit was not set
1c - The FCR was found with more than just the UPF bit set
1d - FVAR register did not contain correct value when read
```

Case 9: UPF fault (FE space)

This is the same as case 8 except it loads a half word and logical address `0xfeaaaaaa` is used.

The legal error codes test 16, case 9 are:

```
1e - Data exception was not generated
1f - FCR UPF bit was not set
20 - The FCR was found with more than just the UPF bit set
21 - FVAR register did not contain correct value when read
```

Case 10: WPF fault (RO)

This test case maps logical address zero to physical address zero and sets the TLB RO bit. Performs st word instruction to logical address zero to cause a WPF fault to occur. This case

generates the exception from the GTLB by invalidating the FTLB after the PTE is loaded. The FCR and FVAR are read after the exception is verified.

The legal error codes test 16, case 10 are:

```
22 - Data exception was not generated
23 - FCR WPF bit was not set
24 - The FCR was found with more than just the WPF bit set
25 - FVAR register did not contain correct value when read
```

Case 11: WPF fault (RO)

This is the same as case 10 except an atomic ldstub instruction is used.

The legal error codes test 16, case 11 are:

```
26 - Data exception was not generated
27 - FCR WPF bit was not set
28 - The FCR was found with more than just the WPF bit set
29 - FVAR register did not contain correct value when read
```

Case 12: WPF fault (RO)

This test case maps logical address `0xfdffffff` to physical address zero and sets the TLB RO bit. Performs st byte instruction to logical address zero to cause a WPF fault to occur. This case generates the exception from the FTLB. The FCR and FVAR are read after the exception is verified.

The legal error codes test 16, case 12 are:

```
2a - Data exception was not generated
2b - FCR WPF bit was not set
2c - The FCR was found with more than just the WPF bit set
2d - FVAR register did not contain correct value when read
```

Case 13: WPF fault (RO)

This is the same as case 12 except logical address `0x00ffffff` is mapped to physical address zero and an atomic ldstuba instruction is used.

The legal error codes test 16, case 13 are:

```
2e - Data exception was not generated
2f - FCR WPF bit was not set
30 - The FCR was found with more than just the WPF bit set
31 - FVAR register did not contain correct value when read
```

Case 14: POF fault (PV false)

This test case maps logical address `0x66666666` to physical address zero and clears the TLB page valid bit, then performs ld halfword instruction to logical address `0x66666666` to cause a POF fault. The FCR and FVAR are read after the exception is verified.

The legal error codes test 16, case 14 are:

```
32 - Data exception was not generated
33 - FCR POF bit was not set
34 - The FCR was found with more than just the POF bit set
35 - FVAR register did not contain correct value when read
```

Case 15: POF fault (PV false)

This is the same as case 14 except logical address `0x99999999` is mapped to physical address zero and a byte load is used.

The legal error codes test 16, case 15 are:

```
36 - Data exception was not generated
37 - FCR POF bit was not set
34 - The FCR was found with more than just the POF bit set
39 - FVAR register did not contain correct value when read
```

3.24 Test 17 - Double Trap Reset Test

This test verifies that the CPU generates a reset when a double trap condition is detected. The test generates a double trap by disabling exceptions and forcing an access alignment exception. The FCR is checked to contain the correct status.

The legal error codes for test 17 are:

```
01 - A double trap did not occur
02 - No reset was flagged in the FCR
03 - FCR<DTRAP> was found low after a double trap
04 - FCR<WDOG> was found active after a double trap
```

3.25 Test 18 - Watch Dog Timer Reset Test

This test verifies that the CPU generates a reset when a watch dog timer trap is detected.

The test first verifies that the watch dog timer does not cause a reset after .75 times the time period. The MMCR is then read and the test waits an additional .75 times the time period to verify the counter was reset. The timer is then verified that a reset occurs after 1.1 times the time period has elapsed. The FCR is checked to contain the correct status.

The legal error codes for test 18 are:

```
01 - A watch dog reset occurred after .75 times the time period
02 - A watch dog reset occurred after the timer was cleared
03 - A watch dog reset did not occur after 1.1 times the time period
04 - FCR<WDOG> was found low after a watch dog reset
05 - FCR<DTRAP> was found active after a watch dog reset
```

3.26 Test 19 - Timeout Fault Test

This test verifies that the timeout logic on the System Board is functional, that the Kbus Address lines are good and that the timeout detection logic in the bus watcher section of the CPU Board is functional. This is the first test which generates Kbus cycles. If a Kbus transaction does not terminate within the required number of cycles, the timeout logic on the System Board terminates the transaction by asserting TERR.

This test performs an RIO access to the ID space of non-existant system slot 0. Since there is no board in slot zero to respond to the ID space read, the transaction should timeout and TERR should be asserted. The test verifies that the FCR<TOF> bit is set, that the FVAR register contains the correct RIO address.

In addition, the test verifies the operation of double-word RIO exception logic and fast I/O (FIO) status operation and timeout.

The 4 cases for test 18 follow:

Case 1: RIO timeout

Case 1 executes 50 Kbus transactions; all of which are to ID space of non-existant slot 0. Patterns consisting of all zeroes, all ones, walking 1 and walking 0's are used for the low 24 bits of the RIO address.

The legal error codes test 18, case 1 are:

```
01 - Data fault exception was not generated
02 - FCR TOF bits was not set
03 - FVAR register did not contain correct RIO address.
```

Case 2: Double-word RIO timeout

Case 2 verifies that double-word RIO accesses generate an exception and that the PDR register specifies that a double-word RIO transaction type was issued.

The legal error codes test 18, case 2 are:

```
04 - Data fault exception was not generated for a double-word ld
     instruction
05 - The PDR did not specify that a double-word RIO transaction was
     issued
06 - Data fault exception was not generated for a double-word st
     instruction
07 - The PDR did not specify that a double-word RIO transaction was
     issued
```

Case 3: Fast RIO (FIO) timeout

Case 3 verifies that Fast RIO accesses generate a level 8 interrupt when the cycle times out and that the MMCR<PIO> bit indicates the correct status.

The legal error codes test 18, case 3 are:

```
08 - A level 8 interrupt was not generated for an FIO timeout
09 - The pending I/O (PIO) bit was not set when the level 8 interrupt
     occurred
0a - FTOR register did not contain correct physical RIO address after
     a fast RIO timeout
0b - The pending I/O (PIO) bit did not clear after the FTOR was rearmed
0c - The pending I/O (PIO) bit did not set when the fast RIO was started
```

Case 4: Double-word fast RIO (FIO) timeout

Case 4 verifies that Fast RIO accesses generate a level 8 interrupt when a double-word fast RIO cycle is used. Space 1 through 15 is also used to verify that the FTOR latches the correct

address.

The legal error codes test 18, case 4 are:

```
0d - A level 8 interrupt was not generated for a double-word FIO
     timeout
0e - The PDR did not specify that a double-word RIO transaction was
     issued.
0f - The FTOR did not contain the correct address after a double-word
     RIO transaction was issued
```

3.27 Test 1a - Slot Probe and Configuration Test

This test probes each slot of the system by performing ID space reads and determines the board types which occupy each slot.

RIO accesses are performed to the ID space of each slot in the system (1 through 7) and if an RIO timeout fault does not occur, examines the ID information returned. From this information, a configuration table is built up in the on-board diagnostic RAM. Once the table is built, the test checks to verifiy that the minimum valid system configuration exists.

In addition, this test checks the state of the burn-in jumper. If the burn-in jumper is present (installed) then the tests run in burn-in mode. If the burn-in jumper is not present, then the diag/normal test switch state is read, and if set to "Diag Mode" then the BOOTMODE environment variable is read from the EAROM. If the BOOTMODE variable is "burnin" then the tests are run in burn-in mode; otherwise, the tests run normally.

The minimum valid system configuration consists of at least one CPU Board, at least one Memory Board and only one System Board. This is the minimum configuration which must exist to continue testing and to boot UNIX.

This test also sorts all installed memory boards on the basis of board size so that the memory will be configured with the largest boards configured first in the physical address space.

In the following error codes, the *X* represents the slot number of the target board.

```
0X - Exception other than Data Fault occurred during ID SPACE
     read of slot X
1X - Data exception occurred during initial probe, but FCR TOF
     was not set.
2X - FVAR contained incorrect logical RIO address.
4X - Unrecognizeable board type code read from slot X
5X - Data exception fault occurred during ID space read after valid
     board was previously located in slot X
6X - Data exception fault occurred during RIO read of optional header
     in IDPROM on board in slot X
7X - Data fault exception occurred during RIO read of graphics minor
     board number from IDPROM in slot X
8X - Data fault exception occurred during RIO read of device identifier
     string from IDPROM in slot X
9X - Data fault exception occurred during RIO read of Memory Board size
         from IDPROM in slot X
ax - Zero size parameter read from IDPROM on Memory Board in slot X
bx - Invalid Memory Board size read from IDPROM in slot X
         Not an even 16 Mbyte multiple.
c0 - System Board count is not 1 (0 or more than 1).
c1 - No Memory Boards were located.
c2 - No CPU Boards were located.
d0 - Data exception occurred reading EAROM BOOTMODE variable.
```

3.28 Test 1b - IDPROM Checksum Test

This test examines the configuration information obtained from the Slot Probe and configuration test and for each board identified, performs an IDPROM checksum test. By convention, the contents of any IDPROM header can be summed and the least significant byte of the result will be zero.

The legal error codes for test 1a are:

```
1X - Data exception fault occurred while reading IDPROM size from
     the board in slot X
2X - Zero size field for IDPROM on board in slot X
3X - Data exception fault occurred while performing checksum on
     IDPROM on board in slot X
4X - IDPROM checksum error for board in slot X
5X - Data exception fault occurred while reading the optional header
     field of the IDPROM on board in slot X
```

3.29 Test 1c - CPU Status Register Test

This test verifies the ability of the CPU atatus tegister to retain data. Bit 2 of the register, the NMI bit, is kept at a logical low level.

The legal error code for test 1b are:

```
01 - Data error was found in the CPUSTAT register
```

3.30 Test 1d - Master/Slave CPU Determination Test

This test determines which CPU in the system is to become the master CPU when multiple CPUs exist.

Upon reset, the slave bit on the CPUSTAT register on each of the installed CPU Boards is zero. All CPU Boards in the system execute the power-up tests in parallel until this test is executed. During this test, each CPU Board examines its own BID register to get its slot number. The slot number is then decremented and multiplied by 400 to compute the number of milliseconds to delay. Each CPU then delays based on the computed delay value.

The first CPU to finish its delay, reads its own CPUSTAT register. If the SLAVE bit is still zero and if the EAROM is unintitled or the EAROM "MASTER" variable is not valid or is valid and specifies this CPU, the first CPU looks in the configuration table generated by the Slot Probe and Configuation Test and sets the CPUSTAT SLAVE bit of all the other installed CPU Boards. The master CPU Board proceeds to finish the power-up test from this point on.

Subsequent CPU Boards which finish their delays will find their SLAVE bits set and will enter a loop which displays "SL" and the slot number on their LEDs.

The legal error codes for test 1c are:

```
10 - Data fault occurred accessing own CPUSTAT register
2x - Data fault occurred accessing CPUSTAT register of CPU in slot X
30 - Data fault occurred accessing EAROM
```

3.31 Test 1e - Bus Watcher Tag Reset Test

This test utilizes a special test feature of the P0 CPU Board. This special feature is the Diagnostic Kbus Transaction. These transactions allow the bus watcher tags to be loaded and read without actually read or writing Kbus memory (See Appendix B for more information on the Diagnostic Kbus Transactions).

This test verifies that the bus watcher tag reset function is functional. All tag RAM locations are written with unique data and the OWN and VAL bits are set. Each tag location is then read and the PM0, PM1, OWN bits are verified to be true. A bus watcher reset is then performed then each tag location is re-read. This time the PMO and PM1 bits should be false.

The legal error codes for test 1d are:

```
01 - Match or Own status error after initial write of tags
     and status.
02 - Match or Own status error after reset of bus watcher tags
     status bits.
```

3.32 Test 1f - Bus Watcher Tag RAM Addressing Test

This test verifies the address lines for the Bus Watcher tag and status rams. Each tag RAM location corresponding to a single address bit on (0x20, 0x40, 0x80... 0x8000) is written with unique data, then tag RAM location 0 is written with zero. Each location is then probed, in the same order as written, and the physical tag match bits (PM0 and PM1) are verified to be true.

The legal error codes for test 1e, case 1 are:

```
01 - Bus watcher tag match status error on read of tag location
     other than zero.
02 - Bus watcher tag match status error on read of tag location
     zero.
```

The test is then repeated using locations corresponding to a single address bit off (`0x1ffc0`, `0x1ffa0`..., `0xffe0`), and address `0x1ffe0` is written with zero.

The legal error codes for this test 1e, case 2 are:

```
03 - Bus watcher tag match status error on read of tag location
     other than 0x1ffe0.
04 - Bus watcher tag match status error on read of tag location
     0x1ffe0.
```

3.33 Test 20 - Bus Watcher Tag Comparitors Test

This is a test of the Bus Watcher Tag match detection logic. The test program loads a series of patterns into the Bus Watcher tags then performs a series of Diagnostic Kbus Transactions to read and verify that the Bus Watcher Tag match comparitors work correctly (see Appendix B for more information on Diagnostic Kbus Transactions).

The PM0 signal represents a match between physical address bits 24:17 and Bus Watcher tag bits 24:17. The PM1 signal represents a match between physical address bits 27:25 and Bus Watcher Tag bits 27:25 and the Tag Valid bit.

There are two test cases.

Case 1 iteratively loads bus watcher tag RAM locations with a walking 1 tag pattern, and performs diagnostic kbus probe transactions to all tag locations with bits 27:17 of the probe transaction address all zeroes. For the first eight probe transactions, PM0 will be false and PM1 will be true. For the second three probe transactions, PM1 will be false and PM0 will be true.

The legal error codes for test 1f, case 1 are:

```
01 - match status error
```

Case 2 loads bus watcher tag RAM locations with a zero tag pattern, then performs diagnostic Kbus probe transactions to all bus watcher tag RAM locations while walking a 1 accoss the upper 11 bits of the probe transaction address. For the first eight probe transactions, PM0 will be false and PM1 will be true. For the second three probe transactions, PM1 will be false and PM0 will be true.

The legal error codes for test 1f, case 2 are:

```
02 - match status error
```

3.34 Test 21 - Bus Watcher Tag RAM Address and Data Test

This is an addressing and data test for the bus watcher tag RAMs. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix B for

information on the moving inverse test algorithm).

This test uses a special test feature of the CPU Board which allows the bus watcher tags to be accessed without generating any Kbus memory cycles (see Appendix B for more information on the Diagnostic Kbus transactions).

During the read-write-read sequence, a Diagnostic Kbus transaction is perfomed to read and verify the bus watcher tag match and OWN status. Then a Diagnostic Kbus transaction is performed to load the same bus watcher tag RAM location with the complement tag data and toggle the OWN bit. Finally, a Diagnostic Kbus transaction is performed to read and verify the new states of the bus watcher tag match and OWN signals.

The legal error codes for test 20 are:

```
01 - PM0/PM1/OWN status error on first read of forward pass
02 - PM0/PM1/OWN status error on second read of forward pass
03 - PM0/PM1/OWN status error on first read of reverse pass
04 - PM0/PM1/OWN status error on second read of reverse pass
```

3.35 Test 22 - Kbus Transaction Type Test

This test verifies that the CPU presents the correct TTYPE to the KBus for the operations used. The transaction types used are read from bits 7:4 of the PDR. Please refer to the KBus specification for more information on transaction types.

There are two test cases for test 21:

Case 1: Generate cacheable transactions and verify proper types in the PDR.

The legal error codes for test 21, case 1 are:

```
01 - The PDR contained the wrong ttype for a read and invalidate
     bus cycle
02 - Access to KBus diagnostic transaction for write and invalidate
     did not generate an expected data exception
03 - The PDR contained the wrong ttype for a write and invalidate
     bus cycle
04 - The PDR contained the wrong ttype for a cacheable read
     bus cycle
```

Case 2: Generate RIO transactions of various sizes to unused slot 0 and verify proper types in the PDR.

The legal error codes for test 21, case 2 are:

```
05 - A data exception error was not generated for an 8 bit RIO
     read cycle
06 - The PDR contained the wrong ttype for an 8 bit RIO read cycle
07 - A data exception error was not generated for a 16 bit RIO
     read cycle
08 - The PDR contained the wrong ttype for a 16 bit RIO read cycle
09 - A data exception error was not generated for a 32 bit RIO
     read cycle
0a - The PDR contained the wrong ttype for a 32 bit RIO read cycle
0b - A data exception error was not generated for an 8 bit RIO
     read-modify-write cycle
0c - The PDR contained the wrong ttype for an 8 bit RIO
     read-modify-write cycle
0d - A data exception error was not generated for an 8 bit RIO
     write cycle
0e - The PDR contained the wrong ttype for an 8 bit RIO write cycle
0f - A data exception error was not generated for a 16 bit RIO
     write cycle
10 - The PDR contained the wrong ttype for a 16 bit RIO write cycle
11 - A data exception error was not generated for a 32 bit RIO
     write cycle
12 - The PDR contained the wrong ttype for a 32 bit RIO write cycle
```

3.36 Test 23 - Memory Board Base Address and Enable Register Test

This is a test for the Base Address Register and Enable Register on each installed Memory Board.

The base address register is an 8-bit register at address `1X020000` in ID space. The Enable register is a 2-bit register at address `1X030000` in ID space. The *X* represents the slot number. All Memory Boards identified by the configuration table generated by the Slot Probe and Configuration Test are tested.

Part 1 is the test of the Base Address register. There are 18, one byte test patterns written to and read from the Base Address register; these consist of all zeros all ones, walking 1 and walking 0 patterns. Part 2 is the test of the Enable register. There are 4, 2 bit test patterns consisting of 0, 3, 1 and 2.

The legal error codes for test 22 are:

```
1X - Data exception fault occurred writing base address register of
     Memory Board in slot X
2X - Data exception fault occurred reading base address register of
     Memory Board in slot X
3X - Data miscompare error for base address register on Memory Board
     in slot X
4X - Data exception fault occurred writing enable register of Memory
     Board in slot X
5X - Data exception fault occurred reading enable register of Memory
     Board in slot X
6X - Data miscompare error for enable register on Memory Board
     in slot X.
```

3.37 Test 24 - Memory Board Uniqueness Test

This test verifies that all installed Memory Boards can be accessed independently of all others.

This is the first test which attempts to write and read memory and thereby test the bus watchers ability to perform Kbus transactions other than RIO types. When a memory block is read by the CPU Board, and the data is not already in the CPU's cache, the bus watcher must perform a Kbus transaction to obtain the data from memory (see Kbus Protocol specification for detailed information on cache/memory consistency) and the data is placed in the CPU's cache. To get the block out of the CPU's cache and back to memory, a cache flush operation must be performed. To flush a cache block, the CPU generally references another block which is an exact multiple of 128 Kbytes offset from the block to be flushed.

This test writes the first byte of each installed Memory Board with the target boards slot number, then reads the first byte of each board back and verifies the slot numbers.

The legal error codes for test 23 are:

```
0X - Indicates an exception occurred on the initial "stb" instruction.
     The error code is the slot number of the target Memory Board.

1X - Indicates that a read of the data cached on the initial "stb"
     is not readable from the cache. The low nibble of the error
     code is the slot number.

2X - Indicates an exception occurred on the flush operation.  This
     is the instruction which causes the block flush back to memory.
     The low nibble of the error code is the slot number.

3X - Indicates an exception occurred on the re-read of target byte.
     The low nibble of the error code is the slot number.

4X - Indicates a data error on the re-read on target byte. This is
     the instruction which causes the target block to be re-cached
     and supplied to the CPU. The low nibble of the error code is
     the slot number.
```

3.38 Test 25 - Memory Board Address Uniqueness Test

This test verifies the uniqueness of the upper bits of the memory address. For 16 Mbyte Memory Boards, bits 27:24 of the memory address is used. For 32 Mbyte Memory Boards, bits 27:25 of the memory address is used. For 128 Mbyte Memory Boards, bit 27 of the memory address is used.

The test disables all Memory Boards by writting a zero to the enable register of each board identified in the configuation table in the Diagnostic RAM (see slot probe and configuration test). Then, for each board, the Base Address register is loaded with 0x00, the board is enabled, and the CPU performs memory accesses to addresses consisting of a walking one pattern on the upper significant bits of the address. For each of these accesses, a memory timeout fault should be generated.

The legal error codes for test 24, case 1 are:

0X - Indicates that the Memory Board responded when it should not have. The base address register of the target board is set to 0x00 and it responded to some other board address. The low nibble of the error code is the slot number of the target Memory Board.

1X - Indicates that the wrong exception type occurred when the Memory Board was read. The expected exception vector is 9. The low nibble of the error code is the slot number of the target Memory Board.

2X - FCR <TOF> bit was not set when exception occurred. The low nibble of the error code is the slot number of the target memory board.

3X - FVAR register did not contain the correct address when the exception occurred. The low nibble of the error code is the slot number of the target memory board.

The test is repeated using 0xff in the Base Address register and a walking zero pattern on the upper significant bits of the address.

The legal error codes for test 24, case 2 are:

5X - Indicates that the Memory Board responded when it should not have. The base address register of the target board is set to 0x00 and it responded to some other board address. The low nibble of the error code is the slot number of the target Memory Board.

6X - Indicates that the wrong exception type occurred when the Memory Board was read. The expected exception vector is 9. The low nibble of the error code is the slot number of the target Memory Board.

7X - FCR <TOF> bit was not set when exception occurred. The low nibble of the error code is the slot number of the target memory board.

8X - FVAR register did not contain the correct address when the exception occurred. The low nibble of the error code is the slot number of the target memory board.

3.39 Test 26 - Memory Board Addressing Test

This test verifies that each installed Memory Board can respond to all unique Memory Board addresses (0x00 through 0x0f).

The test disables all Memory Boards by writting a zero to the enable register of each board identified in the configuation table in the Diagnostic RAM (see Slot Probe and Configuration test). Then, for each board, the Base Address register is iteratively loaded with an incrementing

base address value (0 through 0x0f in increments of "board size"), the board is enabled, and a write/read operation is performed to the target base address. The data written is the target base address. For each of these accesses, no faults should be generated and the data written should match the data read back.

The legal error codes for test 25 are:

```
0X - Indicates that an exception occurred when the board address
     was written. The low nibble of the error code is the slot
     number of the target Memory Board.

1X - Indicates that an exception occurred when flushing the target
     block back to memory. The low nibble of the error code is the
     slot number of the target Memory Board.

2X - Indicates that an exception occurred when the board address
     was read.  The low nibble of the error code is the slot
     number of the target Memory Board.

3X - Indicates the wrong data was returned from the target Memory
     Board.  The low nibble of the error code is the slot number of
     the target Memory Board.
```

3.40 Test 27 - Memory Board Block Addressability Test

This test verifies the uniqueness of the address lines on each installed Memory Board. The 16 Mbyte Memory Board address is broken down as follows:

```
0x00000000 - 0x007fffe0:  Low 8 Mbyte Bank
0x00800000 - 0x00ffffe0:  High 8 Mbyte Bank
```

Note that address bit 23 is the bank select bit. In this test, low and high banks are tested independently; first the low bank (bit 23 = 0) then the high bank (bit 23 = 1).

for 32 and 128 Mbyte Memory Boards, there is no special bank select bit.

The test starts with all installed Memory Boards disabled, then for each installed board, enables it for write/read, sets the base address register to zero, then writes each word in each block corresponding to a single address bit on (walking 1 on address bits 22:5) with its word address (address tag). Then, address zero is written with zero. Each block is then read back and verified to contain the correct word address tag patterns.

The test is repeated using a walking 0 on address bits (22:5) and ends by writing address `ffffe` with -1.

The legal error codes for test 26 are:

```
0X - Data fault exception occurred on ld instruction using walking
     0/1 address from Memory Board in slot X.
1X - Data miscompare occurred on ld instruction using walking 0/1
     address from Memory Board in slot X.
2X - Data fault exception occurred on ld instruction using all
     zeroes/ones address from Memory Board in slot X.
3X - Data miscompare occurred on ld instruction using all zeroes/ones
     address from Memory Board in slot X.

4X - Data fault exception occurred on st instruction using walking
     0/1 address on Memory Board in slot X.
5X - Data fault exception occurred on ld instruction after data from
     Memory Board X was already cached.
6X - Data miscompare occurred on ld instruction after data from
     Memory Board X was already cached.

7X - Data fault exception occurred on st instruction using all
     zeroes/ones address on Memory Board in slot X.
8X - Data fault exception occurred on ld instruction after data from
     Memory Board X was already cached.
9X - Data miscompare occurred on ld instruction after data from
     Memory Board X was already cached.
```

3.41 Test 28 - Memory Board RAM Addressing and Data Test

This is an addressing and data test for the first 1 Mbyte of memory. Only the first 1 Mbyte of memory is tested to keep execution time during power-up selftest to a minimum.

The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix B for information on the moving inverse test algorithm).

During the read-write-read sequence, the target memory block is cached and checked for the correct data. The data in the cache is then complemented and the block is flushed back to memory. The target block is then re-read and verified to contain the complemented data.

If burn-in jumper is present, the entire installed address space is tested.

Legal error codes for test 27 are:

```
1X - Data fault exception occurred during write of memory
     with initial data pattern.
2X - Data fault exception occurred on first read of forward pass
3X - Data miscompare occurred on first read of forward pass
4X - Data fault exception occurred during flush of target
     memory block back to memory during forward pass.
5X - Data fault exception occurred on second read of forward pass
6X - Data miscompare occurred on second read of forward pass
7X - Data fault exception occurred on first read of reverse pass
8X - Data miscompare occurred on first read of reverse pass
9X - Data fault exception occurred during flush of target
     memory block back to memory during reverse pass.
aX - Data fault exception occurred on second read of reverse pass
bX - Data miscompare occurred on second read of reverse pass
```

3.42 Test 29 - Cache Fill-Flush Test

This test fills the entire 128 Kbytes of cache RAM with the first 128 Kbytes of the bootrom code. Next, the second 128 Kbytes of bootrom code is then written to the cache. This should displace the contents of the cache out to physical memory.

The test then reads the first 128 Kbytes (which should displace the second 128 Kbytes in cache out to memory) and verifies that the data is correct. The second 128 Kbytes of memory are then read (which should cause the first 128 Kbytes to be displaced and the second 128 Kbytes to be re-cached) and verifies that the data is correct.

The legal error codes for test 28 are:

```
01 - Data fault exception occurred on st instruction to
     memory while loading cache with first 128 Kbytes of
     bootrom code.
02 - Data fault exception occurred on st instruction to
     memory while loading cache with second 128 Kbytes of
     bootrom code.

03 - Data fault exception occurred on ld instruction from
     memory while verifying first 128 Kbytes of data.
04 - Data miscompare occurred while verifying first 128 Kbytes
     of data.
05 - Data fault exception occurred on ld instruction from
     memory while verifying second 128 Kbytes data.
06 - Data miscompare occurred while verifying second 128 Kbytes
     of data.
```

3.43 Test 2a - Virtual Fault Cache Corruption Test

This test verifies that exceptions which occur due to cache writes do not corrupt the cache data. There are eight test cases.

The legal error codes for the 13 cases in test 29 are:

Case 1: Single precision misaligned store exception to FF space

```
01 - Address Alignment fault did not occur on st to misaligned
     word address.
02 - First word of cache line corrupted on st to misaligned word
     address.
03 - Second word of cache line corrupted on st to misaligned word
     address.
```

Case 2: Double precision misaligned store exception to FF space

```
04 - Address Alignment fault did not occur on std to misaligned
     double-word address.
05 - First word of cache line corrupted on std to misaligned
     double-word address.
06 - Second word of cache line corrupted on std to misaligned
     double-word address.
```

Case 3: Single precision misaligned store operation with MMU enabled

```
07 - Address Alignment fault did not occur on st to misaligned
     word address.
08 - First word of cache line corrupted on st to misaligned word
     address.
09 - Second word of cache line corrupted on st to misaligned word
     address.
```

Case 4: Double precision misaligned store operation with MMU enabled

```
0a - Address Alignment fault did not occur on std to misaligned
     double-word address.
0b - First word of cache line corrupted on std to misaligned
     double-word address.
0c - Second word of cache line corrupted on std to misaligned
     double-word address.
```

Case 5: Single precision read only store exception

```
0d - Data fault exception did not occur on st to page marked read
     only in TLB.
0e - First word of cache line corrupted on st to page marked read
     only in TLB.
0f - Second word of cache line corrupted on st to page marked read
     only in TLB.
```

Case 6: Double precision read only store exception

```
10 - Data fault exception did not occur on std to page marked read
     only in TLB.
11 - First word of cache line corrupted on std to page marked read
     only in TLB.
12 - Second word of cache line corrupted on std to page marked read
     only in TLB.
```

Case 7: Atomic read only store exception

```
13 - Data fault exception did not occur on ldstub to page marked read
     only in TLB.
14 - First word of cache line corrupted on ldstub to page marked read
     only in TLB.
15 - Second word of cache line corrupted on ldstub to page marked read
     only in TLB.
```

Case 8: Single precision TLB miss store exception

```
16 - Data fault exception did not occur on st to page marked as
     invalid in TLB.
17 - First word of cache line corrupted on st to page marked as
     invalid in TLB.
18 - Second word of cache line corrupted on st to page marked as
     invalid in TLB.
```

Case 9: Double precision TLB miss store exception

```
19 - Data fault exception did not occur on std to page marked as
     invalid in TLB.
1a - First word of cache line corrupted on std to page marked as
     invalid in TLB.
1b - Second word of cache line corrupted on std to page marked as
     invalid in TLB.
```

Case 10: Atomic TLB miss store exception

```
1c - Data fault exception did not occur on ldstub to page marked as
     invalid in TLB.
1d - First word of cache line corrupted on ldstub to page marked as
     invalid in TLB.
1e - Second word of cache line corrupted on ldstub to page marked as
     invalid in TLB.
```

Case 11: Single precision User protect store exception

```
1f - Data fault exception did not occur on st to page marked as
     user protected in TLB.
20 - First word of cache line corrupted on st to page marked as
     user protected in TLB.
21 - Second word of cache line corrupted on st to page marked as
     user protected in TLB.
```

Case 12: Double precision User protect store exception

```
22 - Data fault exception did not occur on std to page marked as
     user protected in TLB.
23 - First word of cache line corrupted on std to page marked as
     user protected in TLB.
24 - Second word of cache line corrupted on std to page marked as
     user protected in TLB.
```

Case 13: Atomic User protect store exception

```
25 - Data fault exception did not occur on ldstub to page marked as
     user protected in TLB.
26 - First word of cache line corrupted on ldstub to page marked as
     user protected in TLB.
27 - Second word of cache line corrupted on ldstub to page marked as
     user protected in TLB.
```

3.44 Test 2b - Corrupted Block RAM Addressing and Data Test

This is an addressing and data test for the Corrupted Block RAM. A moving inverse test algorithm is performed (see Appendix B for more information on the moving inverse test algorithm).

The Corrupted Block RAM is a four Kbit RAM and is addressed using cache block addresses (i.e., `0`, `0x20`, `0x40`..., etc.). The RAM contains 1 bit for each cache block. To set/clear a bit corresponding to a particular cache block, a write to ASI `0xf9`, to set, or ASI `0xf8`, to clear, must be performed using the address of the desired cache block.

The legal error codes for test 2a are:

```
01 - Corrupt Bit not one on first read of forward pass
02 - Corrupt bit not zero on second read of forward pass
03 - Corrupt Bit not zero on first read of reverse pass
04 - Corrupt bit not one on second read of reverse pass
```

3.45 Test 2c - Corrupted Block Flush Inhibit Test

This test verifies that cache transactions which reference a corrupted block result in a Kbus timeout.

The test starts out by disabling all Memory Boards so that a memory timeout fault may be generated. Then address 0 is read to cause a memory timeout fault and set the corresponding bit in the Corrupted Block RAM. The Memory Boards are then re-enabled and address 0 is again accessed. This should cause a Kbus timeout fault

The legal error codes for test 2b are:

```
01 - No memory timeout fault was generated when Memory Boards
     disabled.
02 - No memory timeout fault was generated on reference to corrupted
     block.
03 - FCR TOFM bit not set
04 - FVAR does not contain the correct logical address
```

3.46 Test 2d - Virtual Cache Block Replacement Test

This test exercises the cache and bus watcher cache block flush logic by performing writes and reads to common physical addresses through all different logical addresses including FF space.

The test method is as follows:

1. After invalidating the TLB and Virtual cache tags, sixteen TLB entries are created which map logical address `0x20000, 0x22000, 0x24000,... 0x2e000` to physical addresses `0, 0x2000, 0x4000,... 0x1e000`.
2. Sixteen valid, owned, and dirty block are created in the cache by executing store instructions to each of the sixteen logical addresses in the TLB. The data written to each word of the blocks is the physical addresss of the word.
3. Sixteen new TLB entries are created which map all of the logical addresses previously used to target physical address 0.
4. Sixteen blocks are read using all 16 logical addresses and the data is verified to be the correct physical address data for the target physical block.
5. The sixteen blocks are read again using FF space addresses and each block is verified to contain the correct physical address data.
6. Steps 1 through 5 are repeated for target physical addresses `0x2000, 0x4000, ... 0x1e000`.
7. Steps 1 through 6 are repeated using a walking 1 pattern on logical address bits 31:17.

Legal error codes for test 2c are:

```
01 - Data fault exception occurred during creation of the valid
     owned and dirty cache blocks.
02 - Data fault exception occurred during read of target physical
     cache block.
03 - Physical data read through logical address does not match
     expected physical address data.
04 - Data fault exception occurred during read of target physical
     cache blocks using FF space addresses.
05 - Physical data read through FF space address does not match
     expected physical address data.
```

3.47 Test 2e - Atomic load/store instruction test

This test exercises the control logic for the LDSTUB instruction in conjunction with cache and TLB miss conditions.

There are 8 cases for test 2d, as follows:

1. Execute LDSTUB instruction to FF space and generate a cache hit and an FTLB hit.

   The legal error codes for test 2d, case 1 are:

```
01 - An exception occurred on the LDSTUB instruction.
02 - The data read from the cache was incorrect.
03 - The data written to the cache was not 0xff.
```

2. Execute LDSTUB instruction to user space and generate a cache hit and an FTLB hit.

   The legal error codes for test 2d, case 2 are:

```
04 - An exception occurred on the LDSTUB instruction.
05 - The data read from the cache was incorrect.
06 - The data written to the cache was not 0xff.
```

3. Execute LDSTUB instruction to FF space and generate a cache hit and an FTLB miss.

   The legal error codes for test 2d, case 3 are:

```
07 - An exception occurred on the LDSTUB instruction.
08 - The data read from the cache was incorrect.
09 - The data written to the cache was not 0xff.
```

4. Execute LDSTUB instruction to user space and generate a cache hit and an FTLB miss.

   The legal error codes for test 2d, case 4 are:

```
0a - An exception occurred on the LDSTUB instruction.
0b - The data read from the cache was incorrect.
0c - The data written to the cache was not 0xff.
```

5. Execute LDSTUB instruction to FF space and generate a cache miss and an FTLB hit.

   The legal error codes for test 2d, case 5 are:

```
0d - An exception occurred on the LDSTUB instruction.
0e - The data read from the cache was incorrect.
0f - The data written to the cache was not 0xff.
```

6. Execute LDSTUB instruction to user space and generate a cache miss and an FTLB hit.

   The legal error codes for test 2d, case 6 are:

```
10 - An exception occurred on the LDSTUB instruction.
11 - The data read from the cache was incorrect.
12 - The data written to the cache was not 0xff.
```

7. Execute LDSTUB instruction to FF space and generate a cache miss and an FTLB miss.

   The legal error codes for test 2d, case 7 are:

```
13 - An exception occurred on the LDSTUB instruction.
14 - The data read from the cache was incorrect.
15 - The data written to the cache was not 0xff.
```

8. Execute LDSTUB instruction to user space and generate a cache miss and an FTLB miss.

   The legal error codes for test 2d, case 8 are:

```
16 - An exception occurred on the LDSTUB instruction.
17 - The data read from the cache was incorrect.
18 - The data written to the cache was not 0xff.
```

3.48 Test 2f - Paged Out Test

This test verifies that simultaneous instruction and data TLB faults are handled correctly. In addition, this is the first test which actually executes instructions out of the cache by jumping from bootrom space (FE space) to cacheable space (FF space).

There are 4 cases for test 2e, as follows:

1. Perform JMP instruction to an instruction page in FF space which has the VALID bit cleared (invalid page). The instruction fault handler maps the faulted page (makes it valid) and returns to the faulted PC. The test verifies that the correct fault occurred and that the code in the invalid page gets executed correctly.

   The legal error codes for test 2e, case 1 are:

```
01 - An instruction fault did not occur.
02 - The POF bit in the FCR register did not get set.
03 - The FVAR did not contain the correct page address.
04 - The code in the invalid page did not execute correctly.
```

2. Perform JMP instruction to an instruction page in FF space which is invalid (TLB entry has the VALID bit cleared) and execute a LD instruction from an invalid data page in the delay slot of the JMP instruction. The data fault should be taken first and the data fault handler re-maps the data page to make it valid and re-executes the load instruction. An instruction fault should then be taken and the instruction fault handler re-maps the instruction page to make it valid and re-executes the faulted instruction. The test verifies that the instruction and data faults occurred in the correct order, that the load instruction completes normally, and that the code in the invalid page got executed correctly.

The legal error codes for test 2e, case 2 are:

```
05 - Instruction and data faults occurred in the wrong order or did not
     occur.
06 - The POF bit in the FCR register did not get set on the data fault.
07 - The FVAR did not contain the correct data page address.
08 - The POF bit in the FCR register did not get set on the text fault.
09 - The FVAR did not contain the correct text page address.
0a - The LD instruction did not complete correctly (wrong data returned
0b - The code in the invalid page did not execute correctly.
```

3. Perform JMP instruction to an instruction page in FF space which is invalid (TLB entry has the VALID bit cleared) and execute a ST instruction to an invalid data page in the delay slot of the JMP instruction. The data fault should be taken first and the data fault handler returns to the instruction following the store instruction (store should never complete). An instruction fault should then be taken and the instruction fault handler re-maps the instruction page to make it valid and re-executes the faulted instruction. The test verifies that the instruction and data faults occurred in the correct order, that the store instruction did not complete, and that the code in the invalid page got executed correctly.

   The legal error codes for test 2e, case 3 are:

```
0c - Instruction and data faults occurred in the wrong order or did not
     occur.
0d - The POF bit in the FCR register did not get set on the data fault.
0e - The FVAR did not contain the correct data page address.
0f - The POF bit in the FCR register did not get set on the text fault.
10 - The FVAR did not contain the correct text page address.
11 - The ST instruction completed. (store should have been prevented).
12 - The code in the invalid page did not execute correctly.
```

4. Perform LDST instruction to a page in FF space which is read only (TLB entry has the RO bit set). A data fault should be taken and the data fault handler returns to the instruction following the LDST instruction (store should never complete). The test verifies that the data fault occurred and that the store portion of the instruction did not complete.

   The legal error codes for test 2e, case 4 are:

```
13 - A data fault did not occur.
14 - The WPF bit in the FCR register did not get set.
15 - The FVAR register did not contain the correct data page address.
16 - The store part of the ldst instruction completed.
```

3.49 Test 30 - ECC Write/Read Test

This test verifies the ECC data path to and from each installed Memory Board. For each board, the test writes test patterns to each cache line in the target memory block then reads the block back with ECC enabled and verifies that no ECC exception (single bit or multi-bit) are

generated. The test patterns are chosen such that all 256 ECC patterns (0 - 0xff) will be generated.

The legal error codes for test 2f are:

```
01 - ECCS or data fault exception occurred on store of data
     pattern with ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of test
     pattern back to memory.
03 - Single Bit ECC exception generated on re-read of test
     pattern from memory with ECC checking enabled.
04 - Multi-bit ECC exception generated on re-read of test
     pattern from memory with ECC checking enabled.
05 - Exception other than ECCS or ECCM generated on re-read
     of test pattern from memory with ECC checking enabled.
06 - Data error in first word of cache line 0
07 - Data error in second word of cache line 0
08 - Data error in first word of cache line 1
09 - Data error in second word of cache line 1
0a - Data error in first word of cache line 2
0b - Data error in second word of cache line 2
0c - Data error in first word of cache line 3
0d - Data error in second word of cache line 3
```

3.50 Test 31 - ECC Single Bit Correction to 1 Test

This test verifies that the ECC data correction logic can correct a bit from a zero to a one for all 64 data bit positions. The test is performed independly for each all four cache lines.

The test load the KCB register with the ECC pattern which corresponds to a data pattern with a single bit on (such as `0x0000000000000001`) then loads each line of the cache block except for the line under test with the corresponding data pattern. The cache line under test is loaded with an all zeroes data pattern. When the cache block is flushed to memory and re-read, a single bit error should be generated and the cache line under test should contain the corrected data (target bit should toggle from a 0 to a 1). The test verifies that the ECCS exception is generated, that, that the correct address and cache line is indicated in the FPAR register, that the correct syndrome byte is generated and latched in the FES register, and that all cache lines contain the correct data. The data for the target cache should match the data in the other cache lines.

The legal error codes for test 30 are:

```
01 - ECCS or data fault exception occurred on initial load
     of data patterns when ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of target
     cache block to memory when ECC checking disabled.
03 - No exception was generated on re-read of target cache block.
04 - Exception other than ECCS exception was generated on re-read
     of target cache block.
05 - FPAR register contains incorrect address.
06 - FES register contains incorrect syndrome value.
07 - Data error in cache line 0
08 - Data error in cache line 1
09 - Data error in cache line 2
0a - Data error in cache line 3
```

3.51 Test 32 - ECC Single Bit Correction to 0 Test

This test verifies that the ECC data correction logic can correct a bit from a one to a zero for all 64 data bit positions. The test is performed independly for each all 4 cache lines.

The test load the KCB register with the ECC pattern which corresponds to a data pattern with a single bit off (such as `0xfffffffffffffffe`) then loads each line of the cache block except for the line under test with the corresponding data pattern. The cache line under test is loaded with an all ones data pattern. When the cache block is flushed to memory and re-read, a single bit error should be generated and the cache line under test should contain the corrected data (target bit should toggle from a 1 to a 0). The test verifies that the ECCS exception is generated, that, that the correct address and cache line is indicated in the FPAR register, that the correct syndrome byte is generated and latched in the FES register, and that all cache lines contain the correct data. The data for the target cache should match the data in the other cache lines.

The legal error codes for test 31 are:

```
01 - ECCS or data fault exception occurred on initial load
     of data patterns when ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of target
     cache block to memory when ECC checking disabled.
03 - No exception was generated on re-read of target cache block.
04 - Exception other than ECCS exception was generated on re-read
     of target cache block.
05 - FPAR register contains incorrect address.
06 - FES register contains incorrect syndrome value.
07 - Data error in cache line 0
08 - Data error in cache line 1
09 - Data error in cache line 2
0a - Data error in cache line 3
```

3.52 Test 33 - ECC Single Bit Checkbyte Error Test

This test verifies that single bit errors in the checkbyte are detectable and causes no cache line data corruption.

Syndrome values which contain a single set bit correspond to single bit errors in the check byte. This test writes memory with a data pattern which will generate and all zeroes checkbyte but

forces (via the KCB register) a checkbyte corresponding to a walking 1 pattern to be written along with the data. When the data is read back, the ECC checking logic should regenerate an all zeroes check byte and XOR it with the walking 1 checkbyte pattern. This should result in a single bit ECC error. However, the ECC correction logic should recognize that the error is in the checkbyte and not in the data word and the cache line should not be modified.

The legal error codes for test 32 are:

```
01 - ECCS or data fault exception occurred on initial load
     of data patterns when ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of target
     cache block to memory when ECC checking disabled.
03 - No exception was generated on re-read of target cache block.
04 - Exception other than ECCS exception was generated on re-read
     of target cache block.
05 - FPAR register contains incorrect address.
06 - FES register contains incorrect syndrome value.
07 - Data error in cache line 0
08 - Data error in cache line 1
09 - Data error in cache line 2
0a - Data error in cache line 3
```

3.53 Test 34 - ECC Multibit Error Detection Test

This test verifies that all syndrome values which map to a two bit or more than two bit error results in the generation of a multibit ECC exception.

The test contains a look-up table which specifies if a particular syndrom value maps to a single or multibit error. For each syndrome value which does not map to a single bit error, a test case is performed which uses the target syndrome byte as the ECC pattern forced to memory via the KCB register mechanism. The data pattern that is used during the memory write is chosen such that the ECC pattern which would normally be generated is `0x00`. Under these conditions, the syndrome byte which will be generated when the memory block is re-read (causing a multi-bit error) will be the target syndrome byte (`0x00` xor'ed with target syndrome results in generation of the same target syndrome). The test verifies that the ECCM exception is generated, that the FCR register ECCM bit is set, and that the FPAR register contains the correct faulted address.

The legal error codes for test 33 are:

```
01 - ECCS or data fault exception occurred on initial load
     of data patterns when ECC checking disabled.
02 - ECCS or data fault exception occurred on flush of target
     cache block to memory when ECC checking disabled.
03 - No exception was generated on re-read of target cache block.
04 - Exception other than Data Fault exception was generated on
     re-read of target cache block.
05 - FCR ECCM bit not set.
06 - FVAR register contains incorrect address.
07 - FCR not cleared after read of FVAR.
```

3.54 Test 35 - ECC RAM Addressing and Data Test

This is an addressing and data test for the first megabyte of ECC memory. Only the first megabyte of memory are tested to keep execution time during power-up selftest to a minimum.

The test program performs a moving inverse test algorithm to verify the addressing and data paths to the ECC RAMs (see Appendix B for information on the moving inverse test algorithm).

The ECC RAMs are tested indirectly using the occurrance of an ECCS or ECCM exception as the error indication. 64 bit data patterns which correspond to checkbyte patterns of `0xaa` and `0x55` are written to memory to indirectly load the ECC RAMs with the desired complementary data patterns.

During the read-write-read sequence, the target memory block is cached and checked for the correct data. The data in the cache is then changed (so that the checkbyte will be complemented) and the block is flushed back to memory. The target block is then re-read and verified to contain the alternalte data pattern. The test is executed with ECC checking enabled so that any ECC RAM errors will result in an ECCS or ECCM exception.

If burn-in jumper is installed, the entire installed address space is tested.

Legal error codes for test 34 are:

```
1X - Data fault exception occurred during write of memory
     with initial data pattern.
2X - ECCS or data fault exception occurred on first read of
     forward pass
3X - Data miscompare occurred in upper 32 bits of cache line
     during forward pass
4X - Data miscompare occurred in lower 32 bits of cache line
     during forward pass
5X - Data fault exception occurred during flush of target
     memory block back to memory during forward pass.
6X - ECCS or data fault exception occurred on second read of
     forward pass
7X - Data miscompare occurred in upper 32 bits of cache line
     during forward pass
8X - Data miscompare occurred in lower 32 bits of cache line
     during forward pass

9X - ECCS or data fault exception occurred on first read of
     reverse pass
aX - Data miscompare occurred in upper 32 bits of cache line
     during reverse pass
bX - Data miscompare occurred in lower 32 bits of cache line
     during reverse pass
cX - Data fault exception occurred during flush of target
     memory block back to memory during reverse pass.
dX - ECCS or data fault exception occurred on second read of
     reverse pass
eX - Data miscompare occurred in upper 32 bits of cache line
     during reverse pass
fX - Data miscompare occurred in lower 32 bits of cache line
     during reverse pass
```

3.55 Test 36 - FPU Register Load/Store Test

This test verifies the primary interaction between the floating point unit and the memory system by performing a write/read test on one of the floating point register pairs. There are two test cases, one for single precision values and one for double-precision values. Data patterns consisting of all zeroes, all ones, walking 1 and walking 0 are written to the floating point register and read back and verified.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for test 35 are:

```
01 - After attempting to clear the QNE bit on the FPU state
     register, the queue (FQ) is still not empty.
02 - Write read error for single precision load/store.
03 - Write read error for double precision load/store (even register).
04 - Write read error for double precision load/store (odd register).
```

3.56 Test 37 - FPU State Register Test

The FPU state register (FSR) contains FPU mode and status information. This test performs a write/read test on this register using data patterns consisting of all zeroes, all ones, walking 1 and walking 0. The write-only bits of this register are masked-out for all patterns.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for test 36 are:

```
01 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
02 - FSR write read error.
```

3.57 Test 38 - FPU Add/Multiply/Divide Test

This test verifies the path between the FPC and the floating point arithmetic units on the FPC/FALU and the FPC/FMULT interfaces. The correct operation of both the FALU and the FMULT are henced tested by performing arithmetic operations and comparing the results with the correct expected results.

Resource and operand dependencies are forced in order to verify the that the FPU correctly handles them.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for test 37 are:

```
01 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
02 - Incorrect single precision addition result.
03 - Incorrect single precision multiplication result.
04 - Incorrect double precision addition result (even register).
05 - Incorrect double precision addition result (odd register).
06 - Incorrect double precision multiplication result (even register).
07 - Incorrect double precision multiplication result (odd register).
08 - Incorrect single precision division result.
09 - Incorrect double precision division result (even register).
0a - Incorrect double precision division result (odd register).
0b - FPU did not handled operand dependency correctly.
```

3.58 Test 39 - FPU Queue Test

The FPU queue (FQ) keeps tracks of floating point operations that are pending by the FPU when a floating point fp_exception trap occurs.

As per convention, the FPU queue should become empty after the execution of two successive STDFQ (Store Double Floating-point Queue) instructions. In order to test this function of the FPU, a floating point exception is forced by the test. When in exception mode, the queue should not be empty and the execution of two successive STDFQ instructions should cause it to become empty.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for this test are:

```
01 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
02 - FSR write read error while setting TEM (NV) bit.
03 - FPU fp_exception trap did not occur when expected.
04 - FSR QNE bit is clear when it should be set.
05 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
```

3.59 Test 3a - FPU Exceptions Test

There are two floating point trap types that are generated by the FPU hardware. These are: fp_disabled and fp_exception.

The FPU generates four types of exception traps:

1. FPC sequence error exception
2. Unimplemented floating point instruction exception [2]
3. Unfinished floating point instruction exception
4. IEEE exception

IEEE exceptions are classified as follows:

1. Invalid
2. Overflow
3. Underflow
4. Division by zero
5. Inexact

This test verifies that the FPU generates these traps and exceptions properly by performing test cases for each type. Each test case verifies that the exception logic behaves as expected for both cases: when enabled and disabled. The IEEE exception mechanism on the Weitek ALU/FPMULT/FPDIV chips is also exercised by this test.

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for the 13 cases in test 39 are:

Test Case 1: fp_disabled trap

[2] Not checked by this test. All instructions are implemented.

```
01 - FPU fp_disabled trap did not occur when expected.
```

Test Case 2: fp_exception IEEE-Invalid while enabled

```
02 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
03 - FPU fp_exception trap did not occur when expected (IEEE-Invalid).
04 - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Invalid.
```

Test Case 3: fp_exception IEEE-Invalid while disabled

```
05 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
06 - FPU fp_exception trap occured while traps were disabled (IEEE Invalid)
07 - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Invalid.
```

Test Case 4: fp_exception IEEE-Overflow while enabled

```
08 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
09 - FPU fp_exception trap did not occur when expected (IEEE-Overflow).
0a - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Overflow.
```

Test Case 5: fp_exception IEEE-Overflow while disabled

```
0b - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
0c - FPU fp_exception trap occured while traps were disabled (IEEE Overflow
0d - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Overflow.
```

Test Case 6: fp_exception IEEE-Underflow while enabled

```
0e - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
0f - FPU fp_exception trap did not occur when expected (IEEE-Underflow).
10 - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Underflow.
```

Test Case 7: fp_exception IEEE-Underflow while disabled

```
11 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
12 - FPU fp_exception trap occured while traps were disabled (IEEE Underflo
13 - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Underflow.
```

Test Case 8: fp_exception IEEE-Inexact while enabled

```
14 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
15 - FPU fp_exception trap did not occur when expected (IEEE-Inexact).
16 - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Inexact.
```

Test Case 9: fp_exception IEEE-Inexact while disabled

```
17 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
18 - FPU fp_exception trap occured while traps were disabled (IEEE Inexact).
19 - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Inexact.
```

Test Case 10: fp_exception IEEE-Divide-By-Zero while enabled

```
1a - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
1b - FPU fp_exception trap did not occur when expected (IEEE-Divide-by-Zero).
1c - FPU fp_exception trap ocurred, but FSR FTT and CEXC bits are not
     set for IEEE-Divide-by-Zero
```

Test Case 11: fp_exception IEEE-Divide-By-Zero while disabled

```
1d - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
1e - FPU fp_exception trap occured while traps were disabled.
1f - FPU fp_exception trap did not occured, but FSR CEXC and AEXC bits
     are not set for Divide-By-Zero.
```

Test Case 12: fp_exception Sequence-Error

```
20 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
21 - FPU fp_exception trap did not occur when expected (IEEE-Divide-by-Zero).
22 - FPU fp_exception trap did not occur when expected (SEQUENCE).
23 - FPU fp_exception trap ocurred, but FSR FTT bits are not set
     for SEQUENCE.
```

Test Case 13: fp_exception Unfinished-Floating-Point-Instruction

```
24 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
25 - FPU fp_exception trap did not occur when expected (UNFINISHED_FPOP).
26 - FPU fp_exception trap ocurred, but FSR FTT bits are not set
     for UNFINISHED_FPOP
```

3.60 Test 3b - FPU Condition Codes Test

Floating point compares (FCMPS) and floating point condition (FBfcc) instructions interlock on the floating point condition codes. This condition codes are maintained by the FPU in the FSR.

This test checks the proper updating of this bits in the FSR and the correct behavior of the FCMPS and FBfcc instructions.

The condition codes supported by the FPU are:

1. Equal Relation
2. Greater-Than Relation
3. Less-Than Relation
4. Unordered Relation

This test as well as all other floating point unit tests are only executed if the floating point unit is available in the CPU Board.

Legal error codes for test 3a are:

```
01 - After attempting to clear the QNE bit on the FSR, the queue
     (FQ) is still not empty.
```

Test Case 1: Equal Relation when A == B

```
02 - CC should reflect an equal relation, causing FBE
     instruction to fail.
03 - FSR FCC bits not reflecting an equal relation when expected.
```

Test Case 2: Equal Relation when A != B

```
04 - CC should not reflect an equal relation, causing FBE
     instruction to fail.
05 - FSR FCC bits reflecting an equal relation when not expected.
```

Test Case 3: Greater-Than Relation when A > B

```
06 - CC should reflect a greater-than relation, causing FBG
     instruction to fail.
07 - FSR FCC bits not reflecting a greater-than relation when expected.
```

Test Case 4: Greater-Than Relation when A < B

```
08 - CC should not reflect a greater_than relation, causing FBG
     instruction to fail.
09 - FSR FCC bits reflecting a greater-than relation when not expected.
```

Test Case 5: Less-Than Relation when A < B

```
0a - CC should reflect a less-than relation, causing FBL
     instruction to fail.
0b - FSR FCC bits not reflecting a less-than relation when expected.
```

Test Case 6: Less-Than Relation when A > B

```
0c - CC should not reflect a less_than relation, causing FBL
     instruction to fail.
0d - FSR FCC bits reflecting a less-than relation when not expected.
```

Test Case 7: Unordered Relation when A unordered, B ordered

```
0e - CC should reflect an unordered relation, causing FBU
     instruction to fail.
0f - FSR FCC bits not reflecting an unordered relation when expected.
```

Test Case 8: Unordered Relation when A & B ordered

```
10 - CC should not reflect an unordered relation, causing FBU
     instruction to fail.
11 - FSR FCC bits reflecting an unordered relation when not expected.
```

3.61 Test 3c - System Board Interrupt Generation Test

This is a test of the interrupt register on the System Board and the ability of the System Board to generate all 16 vectors when enabled after reset.

The System Board interrupt register is an eight bit register which is written and read using byte RIO write/read operations.

The System Board interrupt logic queues up 16 interrupts whenever a system reset (power-up or software generated) occurs. The first time I/O interrupts are enabled after reset (by reading RIO address `17031000`) the System Board sequentially sends 16 interrupt vectors (`0x8f` through `0x80`) to the BID specified in the System Board interrupt register.

This test consists of two parts: Part 1, is a write/read test of the System Board interrupt register; and part 2 is a System Board directed interrupt test.

Part 1 first reads RIO address `17030000` to disable transmission of interrupts, then writes RIO address `17030000` with incrementing test patterns from `0` to `0xff`. Each pattern is read back and verified.

The legal error codes for test 3b, part 1 are:

```
01 - Data fault exception occurred on initial read to disable
     System Board interrupt register.
02 - Data fault exception occurred on write of pattern to
     System Board interrupt register.
03 - Data fault exception occurred on read of System Board
     interrupt register.
04 - Data pattern written does not match data pattern read from
     System Board interrupt register.
```

Part 2 initializes the System Board interrupt register with the directed bit set and the destination ID set the BID of the CPU Board. System Board interrupts are then enabled by reading address `17031000`. The test verifies that all 16 interrupt vectors are received correctly. Note that this test will fail if a system reset is not performed inbetween passes.

The legal error codes for test 3b, part 2 are:

```
05 - Timeout waiting to receive first interrupt (vector 0x8f) from
     System Board.
06 - Exeption other than Serial Interrupt Controller occurred.
07 - Higher priority interrupt vector was received 256 times without
     receiving expected vector.
8X - Lower priority interrupt vector was received. Error code is the
     vector which was expected.
```

Appendix A: Series4 Considerations

A.1 Introduction

This appendix contains information relating to the Solbourne Series4 power on self tests. The information presented in this appendix includes

- MMCR VDCI bit operation
- Physical diagnostic register
- Diagnostic Kbus transactions
- Test information register usage
- Moving inversions test algorithm
- Power-up status indicators

A.2 MMCR VDCI bit Operation

When the MMCR VDCI bit is set, virtual data cache misses are inhibited from occuring. This means that for all cache reads and writes, a virtual cache hit will occur and the data will be read from or written to the cache without generating a cache miss, which would result in the execution of a Kbus cacheable transaction. The MMCR VDCI bit overrides the effect of the MMCR PDCI bit (see below).

The rules for using the MMCR VDCI bit are as follows:

1. The MMCR ME bit can be either set or cleared depending on if TLB translations are desired.
2. If Alternalte Space Identifier bit 1 is set (data access) and ASI bits 5 and 4 are 0, then the cache is accessed as if a virtual hit occurred. Otherwise a control space operation is decoded.

A.2.1 MMCR PDCI bit Operation

When the MMCR PDCI bit is set and the MMCR VDCI bit is cleared, physical data cache misses are inhibited from occuring. This means that for all cache reads and writes a physical cache hit will occur and the data will be read from or written to the cache without generating a cache miss, which would result in the execution of a Kbus cacheable transaction. If the logical address used during a PDCI operation also results in a virtual miss (virtual tag mismatch) then a the virtual tags are re-loaded and validated.

The rules for using the MMCR PDCI bit are as follows:

1. A virtual miss must occur on each cache access. If a virtual hit occurs, the physical tags will not be re-loaded with the physical address from the TLB.
2. The MMCR ME bit should be set to enable the TLB physical address translation.
3. Each PDCI cache write operation results in the invalidation of the corresponding Virtual Valid bit.

A.2.2 MMCR TPV0 and TPV1 Bits Operation

When the MMCR PDCI bit is set and a cache write operation is performed, the Physical VALID bits are written according to the state of the TPV0 and TPV1 bits. TPV0 controls the state of the PVALID bit for even cache blocks (address bit 5 is zero). TPV1 controls the state of the PVALID bit for odd cache blocks (address bit 5 is set).

The TPV0 bit also controls whether a cache block dirty bit will be set on a cache write operation. If TPV0 is clear, the target cache block dirty bit will be set. If TPV0 is set, the target cache block will be not dirty. This function of the TPV0 bit is independent of even or odd cache block addresses.

Because of the dual functionality of the TPV0 bit, the following effects are seen when a PDCI cache write is performed:

State on cache write:

addr5	TPV1	TPV0	Created block
0	x	0	Invalid (dirty = don't care)
0	x	1	valid, not dirty
1	0	0	invalid (dirty = don't care)
1	0	1	invalid (dirty = don't care)
1	1	0	valid, dirty
1	1	1	valid, not dirty

Note from the above table that a dirty, even block cannot be created using PDCI operations.

A.2.3 MMCR TPO0 and TPO1 Bits Operation

When the MMCR PDCI bit is set and a cache write operation is performed, the Physical OWN bits are written according to the state of the TPO0 and TPO1 bits. TPO0 controls the state of the POWN bit for even cache blocks (address bit 5 is zero). TPO1 controls the state of the POWN bit for odd cache blocks (address bit 5 is set).

A.3 Physical Diagnostic Register

The Physical Diagnostic Register (PDR) is loaded with status informatin from the bus watcher on every Kbus transaction initiated by the CPU. This register can be read at any time. It is especially useful when used in conjunction with the Diagnostic Kbus Transactions (see Diagnostic Kbus Transactions below) to passively interrogate the status of the bus watcher tags. The PDR is defined as follows:

Bit	7	6	5	4	3	2	1	0
	PMO	PM1	OWN	VAL	TT0	TT1	TT2	x

```
PM0 - status of bus watcher address comparitor for bits 23:16;
      active low.
PM1 - status of bus watcher address comparitor for bits 31:24;
      active low.
OWN - bus watcher block ownership status; active low
VAL - bus watcher block valid status; active low
TT0 - encoded transaction type bit 0
TT1 - encoded transaction type bit 1
TT2 - encoded transaction type bit 2
x - unused
```

The encoded transaction type of the last Kbus transaction initiated by the CPU is shown below:

TT2	TT1	TT0	Encoded transaction type
0	0	0	Read and Invalidate
0	0	1	Cacheable Read
0	1	0	IOB
0	1	1	Write and Invalidate
1	0	0	IOR or IORMW
1	0	1	IOW
1	1	0	Flush (IOB to flush a cache block to memory
1	1	1	Other

A.4 Diagnostic Kbus Transactions

There are special Kbus transactions types that are defined only for diagnostic purposes.

Diagnostic Transactions can be initiated by the cpu by performing an Alternate Space load operation from ASI 0x74. The data returned by the load is undefined. A full 32 bit address is generated on Kbus. The Diagnostic Transaction is used only for test purposes and is ignored by all cpus and Kbus masters other than the one generating it. The cpu handshakes the transaction by asserting OKI and DS to prevent a timeout fault. Diagnostic Transactions are usefull for probing the status of the bus watcher tags when followed by a read of the Physical Diagnostic Register (PDR).

Diagnostic Transactions can be used to write the bus watcher cache block address tags, reverse translation ram, as well as set the ownership and valid status of a cache block.

There are several type of Diagnostic Transactions which can be performed. Since only the most significant 27 bits of the address is used to address a cache block, the Diagnostic Transaction function is encoded into the least significant 3 bits of the cache block address as follows:

A2	A1	A0	Encoded diagnostic transaction type
0	0	0	Nop Diagnostic Transaction Load PDR
0	0	1	Load bus watcher tags and Reverse Translation
0	1	0	Undefined
0	1	1	Undefined
1	0	0	Clear OWN, Clear VAL
1	0	1	Clear OWN, Set VAL
1	1	0	Set OWN, Clear VAL
1	1	1	Set OWN, Set VAL

To perform a Kbus Diagnostic Probe operation perform the following instructions:

```
set   "address"+"diag_ttype",%rd      ! physical block address + diag ttype
lda   [%rd]0x74,%g0                   ! access Kbus Diag Transaction space
```

Example: The following code performs a bus watcher tag write and status write followed by a PDR read to verify the contents of the bus watcher tag ram location zero.

```
#define DT_PDRLD      0       ! load PDR
#define DT_TAGLD      1       ! tag load diagnostic transaction code
#define DT_COSV       5       ! clear OWN, set VAL diag transaction code

#define ASI_KDT       0x74    ! ASI for Kbus Diagnostic Transaction

#define     BWTAG   0xaaaa0000     ! bus watcher tag pattern
#define     BWINDEX 0              ! bus watcher tag ram index
#define PDRMSK  PDR_PM0|PDR_PM1|PDR_OWN|PDR_VAL
                                ! PDR bits to check
#define PDREXP  PDR_VAL           ! PM bits and VAL true, OWN false

start:
        set     BWINDEX,%l0         ! bus watcher tag address
        or      %l0,DT_TAGLD,%l1    ! address + encoded diag ttype
        lda     [%l1]ASI_KDT,%g0    ! loads bus watcher tags

        or      %l0,DT_COSV,%l1     ! address + encoded diag ttype
        lda     [%l1]ASI_KDT,%g0    ! clears OWN sets VAL

        or      %l0,DT_PDRLD,%l1    ! address + encoded diag ttype
        lda     [%l1]ASI_KDT,%g0    ! probes tags and loads PDR

        lduba   [%g0]ASI_PDR,%l2    ! read PDR
        and     %l2,PDRMSK,%l2      ! mask off other bits
        cmp     %l2,PDREXP          ! check data
        bne     error
                .
                .
                .

end:
```

A.5 Test Information Register Usage

The Test Information Register (TIR) is used to obtain the status of the TLB and virtual and and physical cache. The TIR is a read only register which, when read, supplies the status of the addressed TLB, Virtual Cache and Physical Cache locations. In other words a "look-up" operation is performed using the supplied address on the TLB, Virtual cache and Physical Cache and the status associated with the logical address for each these is latched into the TIR and supplied to the cpu on the read.

The TIR contains the following bits:

```
VMATCH1         0x80000000      Virtual Match 1 - 31:24 (low true)
VMATCH0         0x40000000      Virtual Match 0 - 23:16 (low true)
VVALID          0x20000000      Virtual Valid (high true)
VOWN            0x10000000      Virtual OWN (high true)

VOWNHIT         0x08000000      Virtual OWN Hit (low true)
VUP             0x04000000      Virtual UP (high true)
VRO             0x02000000      Virtual RO (high true)
CB              0x01000000      Corrupted Block (high true)

PMATCH2         0x00800000      Physical Match 2 - 31:24 (low true)
PMATCH1         0x00400000      Physical Match 1 - 23:16 (low true)
PMATCH0         0x00200000      Physical Match 0 - 15:13 (low true)
PVALID          0x00100000      Physical Valid (high true)

POWN            0x00080000      Physical Own (high true)
FLSHVALID       0x00040000      Flush Valid (low true)
TMATCH0         0x00020000      TLB Match 0 - 31:23 (low true)
TTVALID         0x00010000      TLB Valid (low true)

TPVALID         0x00008000      TLB Page-valid (high true)
TUP             0x00004000      TLB UP (high true)
TRO             0x00002000      TLB RO (high true)
FLSHADD0        0x00001000      Flush Address 0 (high true)

FLSHADD1        0x00000800      Flush Address 1 (high true)
FLSHADD2        0x00000400      Flush Address 2 (high true)

IOOP            0x00000080      IO operation (TIO|TIOB) (low true)
BDRT            0x00000040      Block Dirty (high true)
TIO             0x00000020      TLB IO (high true)
TIOB            0x00000010      TLB IOB (high true)
```

A.6 Moving Inversions Test Algorithm

This appendix contains a general description of the Moving Inversions Algorithm (MOVI) commonly used to verify the addressing and data integrity of RAM devices. The strengths of this algorithm are its relatively short execution time, functional testing of memory bits, and dynamic tests of best and worst case access times.

In principle, MOVI inverts the data of each address sequentially, thus creating an access time by the jump from one address to ahother which contains different information. To measure access times, the data at each address is read before and after inversion. This requires three operations on each address: a read, a write, and another read.

The read/write/read operations are performed with both forward and backward (reverse) address sequences, and also with n orders of the address-bit significance (where n is the number of address bits).

A general stepwise description of the MOVI algorithm is presented below:

1. The target memory is filled with a background data pattern.

2. In the forward direction (from low to high addresses) a target location is read and check for the correct background pattern (this is the 1st read of the forward pass).
3. The complement pattern (1's complement of the background pattern) is written to the target location.
4. The target location is read and checked for the correct complement pattern (this is the 2nd read of the forward pass).
5. Steps 2 through 4 are repeated for all addresses until the high address of the memory has been reached. At this point the memory should be filled with the complement pattern (assuming no errors were encountered).
6. In the reverse direction (from high to low address) a target location is read and checked for the correct complement pattern (this is the 1st read of the reverse pass).
7. The background pattern (1's complement of the complement pattern) is written to the target location.
8. The target location is read and checked for the correct background pattern (this is the 2nd read of the reverse pass).
9. Steps 6 through 8 are repeated for all addresses until the low address of memory has been reached. At this point the memory should be filled with the background pattern again (assuming no errors were encountered during the reverse pass).

The steps above implement the first iteration of the MOVI test, where the basic address increment value is 1. Successive iterations use higher address-bit significance up to 2**n where n is the number of involved address bits. This is the same as using a different bit of the address each time as the least significant bit for incrementing through all possible addresses. This has the effect of incrementing through all the addresses by 2's, 4's, 8's, and so on; every address

overflow generates and end-around carry, so that all addresses are tested once in each sequence.

The table illustrates the binary address sequences generated by MOVI for and eight location memory:

Forward sequences:

Iteration	0	1	2
	lsb	lsb	lsb
	v	v	v
	000 0	000 0	000 0
	001 1	010 2	100 4
	010 2	100 4	——
	011 3	110 6	001 1
	100 4	——	101 5
	101 5	001 1	——
	110 6	011 3	010 2
	111 7	101 5	110 6
	——	111 7	——
		——	011 3
			111 7
			——

Reverse sequences:

Iteration	0	1	2
	lsb	lsb	lsb
	v	v	v
	111 7	111 7	111 7
	110 6	101 5	011 3
	101 5	011 3	——
	100 4	001 1	110 6
	011 3	——	010 2
	010 2	110 6	——
	001 1	100 4	101 5
	000 0	010 2	001 1
	——	000 0	——
		——	100 4
			000 0
			——

A.7 Power-up Status Indicators

This appendix contains the power-up status indicators for the CPU Board.

0x00	ROM started executing
0x90	Bad IDPROM checksum, range 0x91-0x97. (right-hand digit contains slot number):
0xa0	Master failed [3]
0xa1	Relinquishing mastership [3]
0xa2	Slave received mastership [4]
0xa3	Timeout while giving mastership [3]
0xa4	Awaking slave CPU [3]
0xa5	Slave CPU received slave command [4]
0xa6	Slave CPU passed power-up tests [3]
0xa7	Slave failed [4]
0xa8	Slave CPU failed power-up tests [3]
0xa9	Timeout while awaking slave CPU [3]
0xab	Burn-in jumper detected, looping [3]
0xad	ROM power-up tests completed [3]
0xae	Initializing ECC [3]

ROM initialization LED codes:

0xb0	ROM main() starting
0xb1	Initializing I/O mapping addresses
0xb2	Bad EEROM checksum (warning)
0xb3	Initializing EEROM
0xb4	Initializing the IOBs
0xb5	Initializing the devices
0xb6	Initializing stdin, stdout, stderr
0xb7	Initializing the file systems
0xb8	Initializing ARP
0xb9	Initializing mbufs
0xba	Initializing sockets
0xbb	Initializing the console
0xbc	Could not open console device
0xbd	Initializing main before cmdloop
0xbe	Waiting for command from console
0xbf	Executing a command
0xc0	Stand-alone crt0 starting
0xc1	Stand-alone crt0 calling main
0xc8	No System Board found
0xce	FCR not zero on reset
0xcf	Executing a reset halt

LED codes for devices (displayed during device initialization):

[3] Displayed by master.

[4] Displayed by slave.

0xd0	Simulated UART
0xd1	Simulated disk
0xd2	LANCE Ethernet
0xd3	Real time clock
0xd4	RAM disk
0xd5	Execelan Ethernet
0xd6	VME-to-SCSI controller
0xd7	UART driver
0xd8	Keyboard/mouse
0xd9	Frame buffer

The following three tests are executed only in burn-in mode.

- Test 3c - Frame Buffer Test
- Test 3e - Serial Port Reset Test
- Test 3f - Serial Port Internal Loopback Test

The behavior of the Memory Board RAM Addressing and Data Test and the ECC RAM Addressing and Data Test are modified when the burn-in jumper is installed. Instead of testing only 1 Mbyte of memory, if the burn-in jumper is installed test 27 and 34 tests all memory boards are tested.

Appendix B: Series5 Considerations

B.1 Introduction

This appendix contains information relating to the Solbourne Series4 power on self tests. The information presented in this appendix includes

- Physical diagnostic register
- Diagnostic Kbus transactions
- Test information register usage
- Moving inversions test algorithm

B.2 Physical Diagnostic Register

The Physical Diagnostic Register (PDR) is loaded with status informatin from the bus watcher on every Kbus transaction initiated by the CPU. This register can be read at any time. It is especially useful when used in conjunction with the Diagnostic Kbus Transactions (see Diagnostic Kbus Transactions below) to passively interrogate the status of the bus watcher tags. The PDR is defined as follows:

Bit	7	6	5	4	3	2	1	0
	TT3	TT2	TT1	TT0	res	OWN	PM1	PM0

```
PM0 - status of bus watcher address comparitor for bits 24:17;
      active high.
PM1 - status of bus watcher address comparitor for bits 27:25;
      active high.
OWN - bus watcher block ownership status; active low
res - reserved
TT0 - encoded transaction type bit 0
TT1 - encoded transaction type bit 1
TT2 - encoded transaction type bit 2
TT3 - encoded transaction type bit 3
```

The encoded transaction type of the last Kbus transaction initiated by the CPU is shown below:

TT3	TT2	TT1	TT0	Encoded transaction type
0	0	0	0	Probe: Set OWN, set VALID, store tag
0	0	0	1	Probe: Set OWN, clear VALID, store tag
0	0	1	0	Probe: Clear OWN, set VALID, store tag
0	0	1	1	Probe: Clear OWN, clear VALID, store tag
0	1	1	1	Probe: NOP (Interogate tags)
1	0	0	0	Read and Invalidate
1	0	0	1	Cacheable Read
1	0	1	0	IOB
1	0	1	1	Write and Invalidate
1	1	0	0	IOR or IORMW
1	1	0	1	IOW
1	1	1	0	Flush (IOB to flush a cache block to memory
1	1	1	1	Other

B.3 Diagnostic Kbus Transactions

There are special Kbus transactions types that are defined only for diagnostic purposes.

Diagnostic Transactions can be initiated by the cpu by performing an Alternate Space load operation from ASI 0xf4. The data returned by the load is undefined. A full 32 bit address is generated on Kbus. The Diagnostic Transaction is used only for test purposes and is ignored by all cpus and Kbus masters other than the one generating it. The cpu handshakes the transaction by asserting OKI and DS to prevent a timeout fault. Diagnostic Transactions are useful for probing the status of the bus watcher tags when followed by a read of the Physical Diagnostic Register (PDR).

Diagnostic Transactions can be used to write the bus watcher cache block address tags and set the ownership and valid status of a cache block.

There are several type of Diagnostic Transactions which can be performed. Since only the most significant 27 bits of the address is used to address a cache block, the Diagnostic Transaction function is encoded into the least significant 3 bits of the cache block address as follows:

A2	A1	A0	Encoded diagnostic transaction type
0	0	0	Nop Diagnostic Transaction Load PDR
0	0	1	Undefined
0	1	0	Undefined
0	1	1	PMODIFY: Load cache tags with same data
1	0	0	Clear OWN, Clear VALID
1	0	1	Clear OWN, Set VALID
1	1	0	Set OWN, Clear VALID
1	1	1	Set OWN, Set VALID

To perform a Kbus Diagnostic Probe operation perform the following instructions:

```
set    "address"+"diag_ttype",%rd      ! physical block address + diag ttype
lda    [%rd]0xf4,%g0                   ! access Kbus Diag Transaction space
```

Example: The following code performs a bus watcher tag write and status write followed by a PDR read to verify the contents of the bus watcher tag ram location zero.

```
#define DT_PDRLD      0         ! load PDR
#define DT_COSV       5         ! clear OWN, set VAL diag transaction code

#define ASI_KDT       0xf4      ! ASI for Kbus Diagnostic Transaction

#define      BWTAG    0x0aaa0000     ! bus watcher tag pattern
#define      BWINDEX  0              ! bus watcher tag ram index
#define PDRMSK  PDR_PM0|PDR_PM1|PDR_OWN     ! PDR bits to check
#define PDREXP  PDR_PM0|PDR_PM1        ! PM bits and VAL true, OWN false

start:
        set     BWINDEX,%l0         ! bus watcher tag address

        or      %l0,DT_COSV,%l1     ! address + encoded diag ttype
        lda     [%l1]ASI_KDT,%g0    ! clears OWN sets VAL

        or      %l0,DT_PDRLD,%l1    ! address + encoded diag ttype
        lda     [%l1]ASI_KDT,%g0    ! probes tags and loads PDR

        lduba   [%g0]ASI_PDR,%l2    ! read PDR
        and     %l2,PDRMSK,%l2      ! mask off other bits
        cmp     %l2,PDREXP          ! check data
        bne     error
                .
                .
                .

end:
```

B.4 Test Information Register Usage

The Test Information Register (TIR) is used to obtain the status of the TLB and cache. The TIR is a read only register which, when read, supplies the status of the addressed TLB and Cache locations. In other words a "look-up" operation is performed using the supplied address on the TLB and Cache and the status associated with the logical address for each these is latched into the TIR and supplied to the cpu on the read.

The TIR contains the following bits:

```
FM3             0x00040000      FTLB Match 3 - 31:28 (high true)
CORRUPT         0x00020000      Cache block corrupt bit (high true)
DIRTY           0x00010000      Cache block dirty bit (high true)

GPV             0x00008000      GTLB page valid (high true)
GWP             0x00004000      GTLB WP (write protect) (high true)
GUP             0x00002000      GTLB UP (high true)
GIO             0x00001000      GTLB IO (high true)

GM2             0x00000800      GTLB Match 2 - GTLB valid (high true)
GM1             0x00000400      GTLB Match 1 - 31:28 (high true)
GM0             0x00000200      GTLB Match 0 - 27:24 (high true)
FUP             0x00000100      FTLB UP (high true)

FWP             0x00000080      FTLB WP (write protect) (high true)
FM2             0x00000040      FTLB Match 2 - FTLB valid (high true)
FM1             0x00000020      FTLB Match 1 - 27:24 (high true)
FM0             0x00000010      FTLB Match 0 - 23:20 (high true)

CM2             0x00000008      Cache Match 2 - 27:25 and valid (high true)
CM1             0x00000004      Cache Match 1 - 24:21 (high true)
CM0             0x00000002      Cache Match 0 - 20:17 (high true)
OWN             0x00000001      Cache block ownership bit (high true)
```

B.5 Moving Inversions Test Algorithm

This appendix contains a general description of the Moving Inversions Algorithm (MOVI) commonly used to verify the addressing and data integrity of RAM devices. The strengths of this algorithm are its relatively short execution time, functional testing of memory bits, and dynamic tests of best and worst case access times.

In principle, MOVI inverts the data of each address sequentially, thus creating an access time by the jump from one address to ahother which contains different information. To measure access times, the data at each address is read before and after inversion. This requires three operations on each address: a read, a write, and another read.

The read/write/read operations are performed with both forward and backward (reverse) address sequences, and also with n orders of the address-bit significance (where n is the number of address bits).

A general stepwise description of the MOVI algorithm is presented below:

1. The target memory is filled with a background data pattern.
2. In the forward direction (from low to high addresses) a target location is read and check for the correct background pattern (this is the 1st read of the forward pass).
3. The complement pattern (1's complement of the background pattern) is written to the target location.
4. The target location is read and checked for the correct complement pattern (this is the 2nd read of the forward pass).
5. Steps 2 through 4 are repeated for all addresses until the high address of the memory has been reached. At this point the memory should be filled with the complement pattern

(assuming no errors were encountered).

6. In the reverse direction (from high to low address) a target location is read and checked for the correct complement pattern (this is the 1st read of the reverse pass).
7. The background pattern (1's complement of the complement pattern) is written to the target location.
8. The target location is read and checked for the correct background pattern (this is the 2nd read of the reverse pass).
9. Steps 6 through 8 are repeated for all addresses until the low address of memory has been reached. At this point the memory should be filled with the background pattern again (assuming no errors were encountered during the reverse pass).

The steps above implement the first iteration of the MOVI test, where the basic address increment value is 1. Successive iterations use higher address-bit significance up to 2**n where n is the number of involved address bits. This is the same as using a different bit of the address each time as the least significant bit for incrementing through all possible addresses. This has the effect of incrementing through all the addresses by 2's, 4's, 8's, and so on; every address overflow generates and end-around carry, so that all addresses are tested once in each sequence.

The table illustrates the binary address sequences generated by MOVI for and eight location memory:

Forward sequences:

Iteration	0	1	2
	lsb	lsb	lsb
	v	v	v
	000 0	000 0	000 0
	001 1	010 2	100 4
	010 2	100 4	——
	011 3	110 6	001 1
	100 4	——	101 5
	101 5	001 1	——
	110 6	011 3	010 2
	111 7	101 5	110 6
	——	111 7	——
		——	011 3
			111 7
			——

Reverse sequences:

Iteration	0	1	2
	lsb	lsb	lsb
	v	v	v
	111 7	111 7	111 7
	110 6	101 5	011 3
	101 5	011 3	——
	100 4	001 1	110 6
	011 3	——	010 2
	010 2	110 6	——
	001 1	100 4	101 5
	000 0	010 2	001 1
	——	000 0	——
		——	100 4
			000 0
			——

Index

T