CHALLENGE™ / Onyx®
Diagnostics Roadmap

Document Number 108-7045-010

Contributors

Written by Greg Morris
Illustrated by Dan Young and Greg Morris
Engineering contributions by John Kraft, Steve Whitney, Rich Altmaier, Unmesh Agarwala, Ray Mascia, Robert Thomas, Dilip Amin

Restricted Rights Legend

Use, duplication, or disclosure of the technical data contained in this document by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and/or in similar or successor clauses in the FAR, or in the DOD or NASA FAR Supplement. Unpublished rights reserved under the Copyright Laws of the United States. Contractor/manufacturer is Silicon Graphics, Inc., 2011 N. Shoreline Blvd., Mountain View, CA 94039-7311.

FCC Warning

This equipment has been tested and found compliant with the limits for a Class A digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user will be required to correct the interference at his own expense.

Attention

This product requires the use of external shielded cables in order to maintain compliance pursuant to Part 15 of the FCC Rules.

Challenge/Onyx Diagnostic Roadmap
Document Number 108-7045-010

Silicon Graphics, Inc.
Mountain View, California

Contents

Figures

Tables

Introduction

The purpose of this document is to describe the various diagnostic tools available with the Everest board set and how they relate to the Everest system components and to each other. Each of the diagnostic tools is described, the physical area being tested is identified, and the possible error messages are explained.

The information contained in this document is organized as follows:

- Chapter 1 provides the theory of operations for the Everest board set. The various buses connecting the boards are described, along with the bus errors potentially encountered by each board.
- Chapter 2 discusses the power subsystem. The inputs and outputs of all power supplies and on-board DC-to-DC converters are illustrated. The power-up sequence is described, as are the various fault LEDs and other error detection methods.
- Chapter 3 explains the functions of the system controller, what causes various error messages to be generated, and what those messages mean.
- Chapter 4 describes the PROM Monitor, as well as the Power-on tests, and the Power-on Diagnostics (POD). System board configuration and the PROM Monitor boot commands are also discussed.
- Chapter 5 supplies information on the standalone IDE and on IRIX error reporting.
- Chapter 6 analyzes several fault isolation procedures: Examining a Frozen System, Debugging Hints, and Examining the IRIX and System Controller Error Logs.

Chapter 1

Theory of Operations

1.1 Overview

This chapter is an introduction to the Everest (POWERpath-2) board set and how those boards communicate over the various system and board buses. Figure 1-1 is a high-level functional block diagram of the Everest board set, installed in a typical system. Figure 1-2 illustrates the IO4 board architecture in greater detail. As the individual diagnostics tools are discussed, you are referred to similar diagrams that highlight the areas affected by that tool (where appropriate).

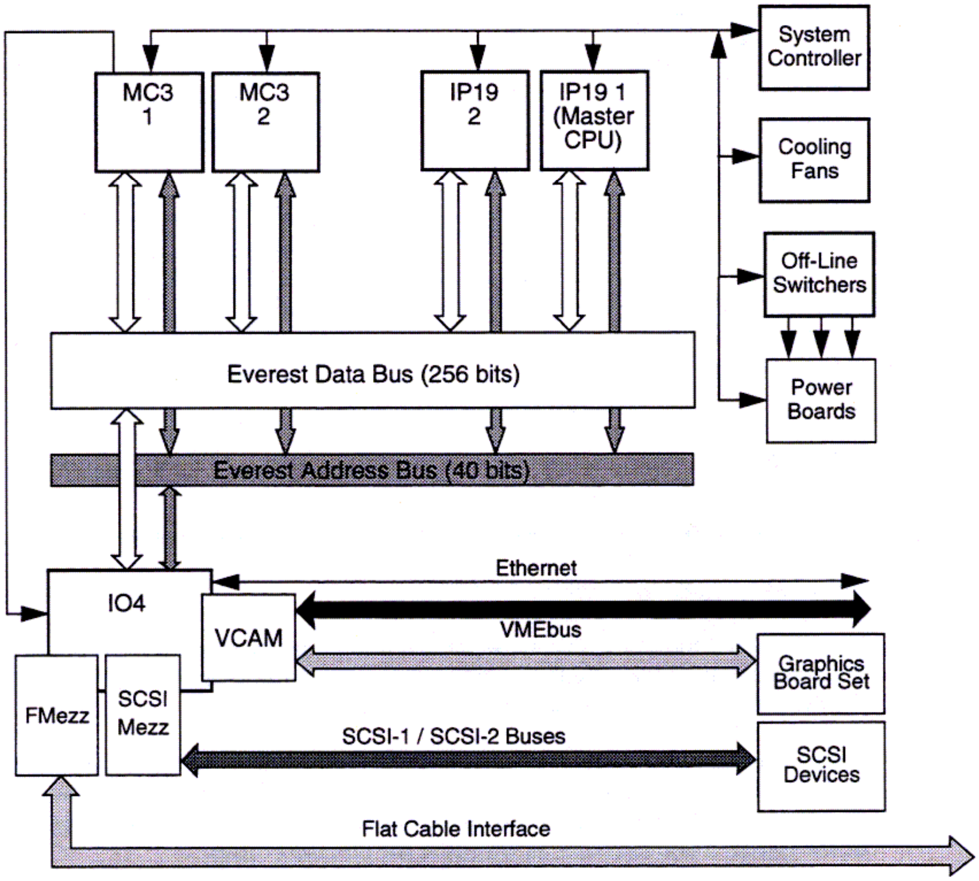


Figure 1-1 Everest Functional Block Diagram

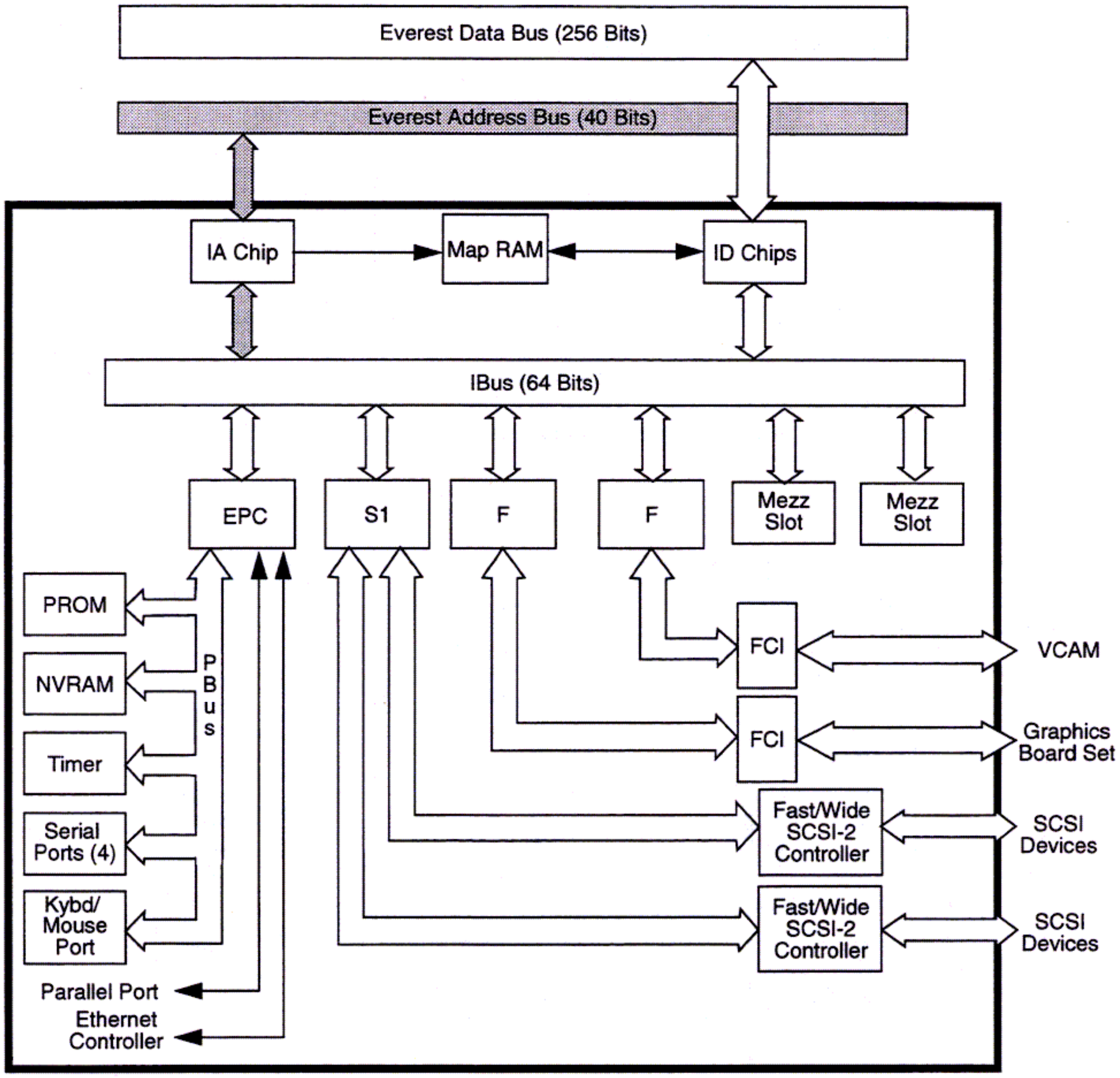


Figure 1-2 IO4 Board Functional Block Diagram

The diagnostic tools available are separated into six groups: the Parity Checkers, Power Subsystem Self-Checks, the Power-on tests, the Power-on Diagnostics (POD), the System Controller, and the Integrated Diagnostics Environment (IDE).

Parity Checkers are housed in the data and address bus ASICs on each system board, in the cache controller and CPU cache on the IP19 board, and in each of the interface ASICs on the IO4 board. The checkers are stategically placed to verify correct address and data parity at the system bus connectors, as well as at the on-board buses.

The Power Subsystem Self-Checks are automatic at power on and at reset. Voltages that are not within the specified levels, or are not present, cause the System Controller to halt the power-up sequence and display an error message. Error information is also provided by a series of LEDs located on the off-line switchers and on the system boards, but they are not visible without opening the cabinet.

The power-on tests execute whenever the system is powered up or reset. These tests verify enough of the system's basic hardware functionality to load the Standalone Diagnostics from the IDE.

The Power-on Diagnostics (POD) is a special command interpreter that is a subset of the Power-on tests. POD is automatically invoked by the PROM monitor in the event of an error during the boot process, or can be manually selected by the operator. POD provides an interface that allows the operator to inspect and modify various system parameters.

The Standalone Diagnostics are a series of functionality-oriented tests that are invoked from the PROM Monitor. They provide the highest degree of error isolation and offer the most accurate diagnosis, but require their own environment (IDE) in order to run. Completion of the Standalone Diagnostics provides sufficient confidence in the system to attempt to boot UNIX®, but is not a guarantee.

Each of these groups is described in more detail in the following chapters.

1.2 System Buses

This section lists each of the buses found in a typical Everest deskside or rackmount system (see Figure 1-3):

Everest Buses

- The Everest Data and Address buses (Ebus) interconnect the system board set.
- The Polled Serial bus specifically connects the system's CPU boards to the System Controller.

On-board Buses

- The Interface bus (Ibus) is located on the IO4 board. This internal bus connects the Ebus to the peripheral controllers on the IO4 board and its various optional interface (mezzanine) cards.
- The Peripheral bus (Pbus) is also located on the IO4 board. It connects the Everest Peripheral Controller (EPC) chip to a number of I/O ports, the NVRAM, and 1 MB of flash PROM.
- The Memory Bus is located on the MC3 and provides a path between the Ebus interface ASICs and the memory array logic.

Peripheral Buses

- The SCSI buses connect a variety of storage devices to the IO4 board. The IO4 board has two SCSI buses standard. A maximum of 32 SCSI buses can be supported by the large systems with the addition of multiple IO4 boards and SCSI mezzanine cards.

- The Flat Cable Interface (FCI) connects the graphics cards or VMEbus to the IO4 board. Like the SCSI, multiple FCI buses can be supported with the addition of FCI Mezzanine cards. In the standard configuration, 2 FCIs are on the IO4 board.
- The VMEbus is embedded in both the deskside and rackmount backplanes and is available as an optional third cardcage with the rackmount system.

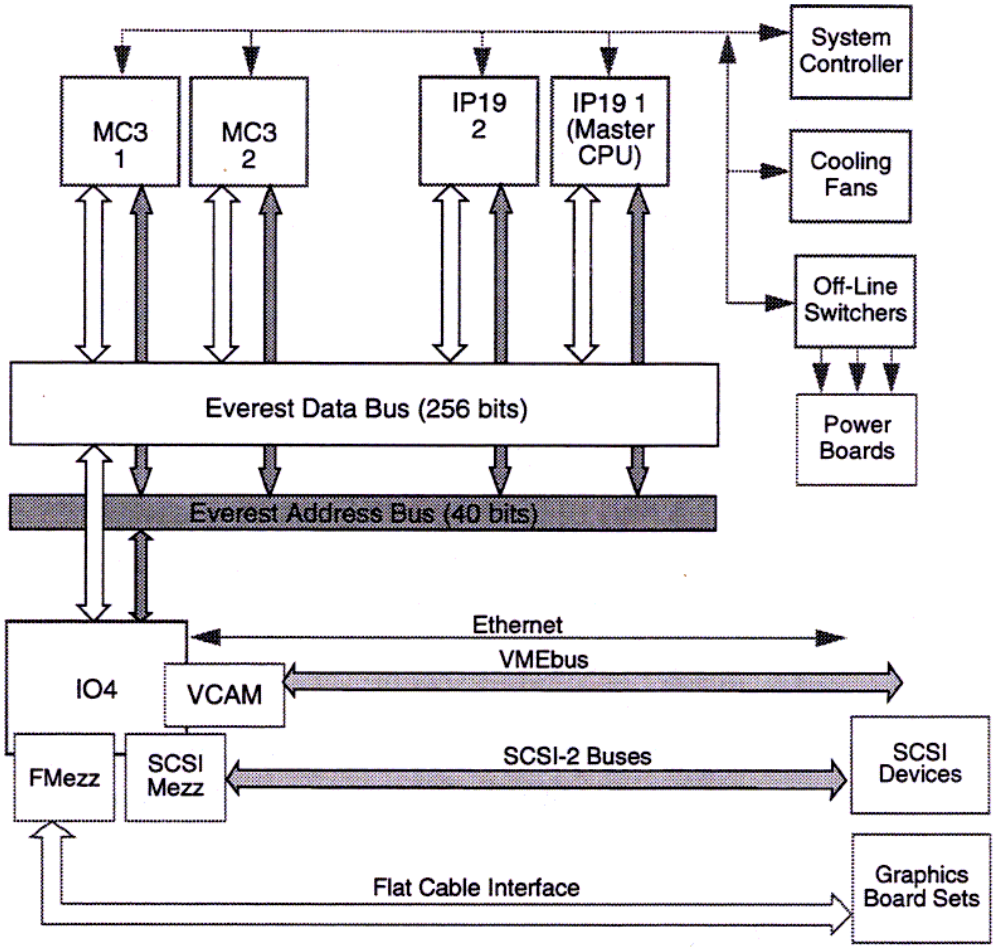


Figure 1-3 Everest System Buses

Everest Address and Data Buses

The Everest system buses consist of a 256-bit, 1200 MB/sec data bus, a 40-bit address bus, and the bus interfaces. The interfaces between the system buses and the Everest boards are supplied by a set of data and address ASICs (see Figure 1-4). There are four data ASICs (D chips) and one address ASIC (A chip) on each board. This logic is not identical from board to board but performs the same basic functions. The parity checking performed on the data, address, and control lines during every bus cycle is done by these interface chips. There are eight parity lines on the data bus and 2 parity lines on the address bus. ASIC parity error detection is explained in more detail in Section 1.3.

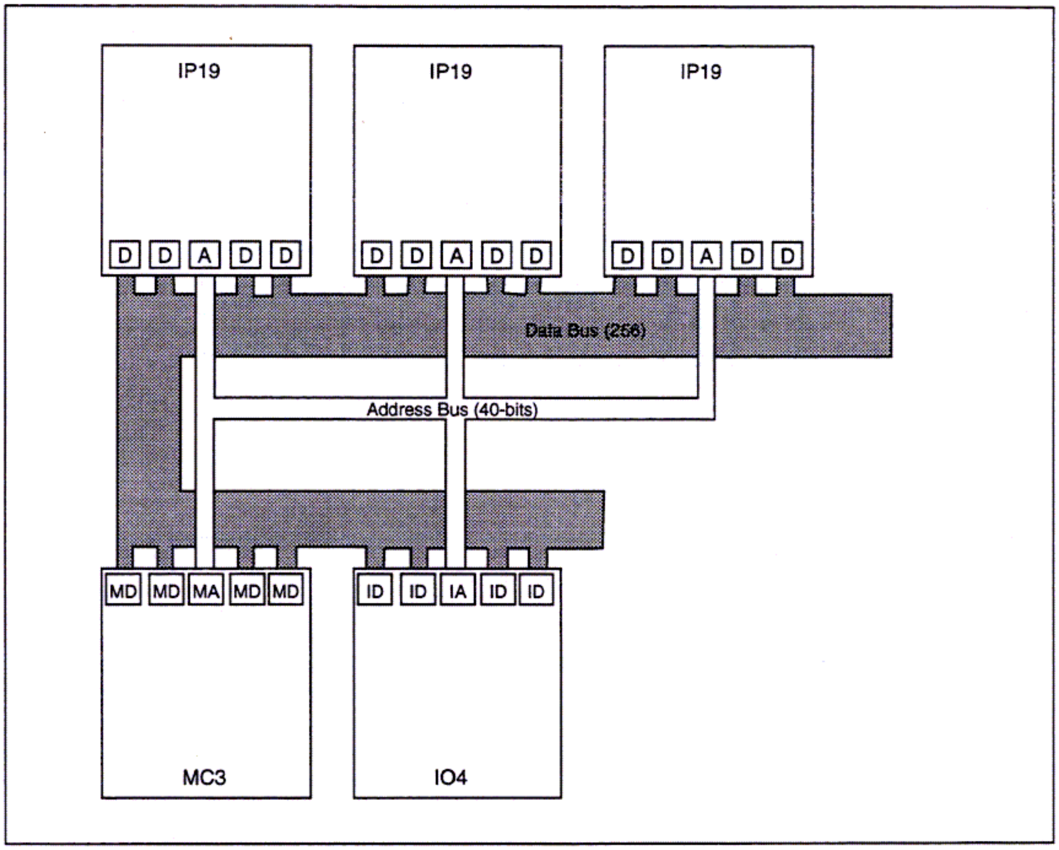


Figure 1-4 Everest Buses and Interface ASICs

Polled Serial Bus

This is a dedicated bus embedded in the system's backplane. It connects the system's CPU boards with the System Controller. During the boot process, the System Controller polls the CPU boards over this bus requesting a bootmaster. After the bootmaster is identified, all bring-up messages from the bootmaster CPU are sent to the System Controller display over the Polled Serial bus. The bootmaster CPU also outputs the bring-up messages to the system console RS-232 port (see Figure 1-5).

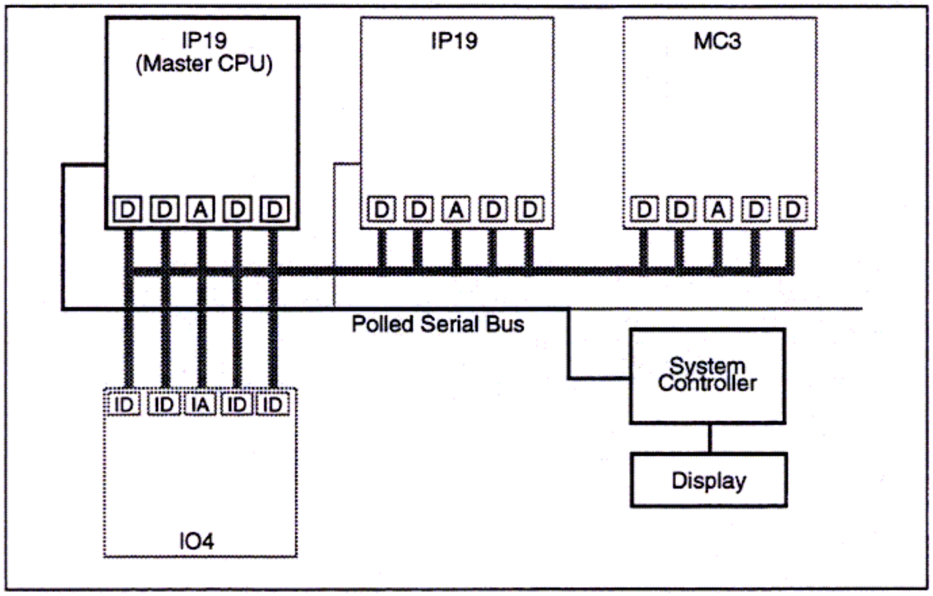


Figure 1-5 Polled Serial Bus

The Polled Serial bus provides a shortened error reporting path to the System Controller display on the front panel. The path from the bootmaster CPU to the RS-232 System Console port involves the Ebus and the IO4's Ibus. For error messages to reach the graphics monitor, they must pass through the graphics boards as well. Both of these error reporting paths require a significant percentage of the system's hardware to be functional before an error can be reported.

Interface Bus (Ibus) and Peripheral Bus (Pbus) on IO4 Board

When the PROM initializes it reads the IA configuration registers to determine what kind of devices are connected to the Ibus. Based on the configuration information, the PROM runs a series of diagnostics that check the integrity of the Ibus and Pbus, as well as the functionality of the devices themselves. These diagnostics cannot distinguish between a bus failure and an ASIC failure, but are sufficient to isolate a fault to the FRU (board) level.

SCSI Bus

The system supports 20 MB/sec SCSI buses. These buses can be configured as single-ended or differential, and as 8- or 16-bit. A five-digit drive address has been implemented to accommodate the large number of storage devices the Everest board set can manage. The first two digits represent the decimal slot number of the IO4 board. The third digit corresponds to the *nth* SCSI channel on the IO4 board (with possible SCSI mezzanine boards) 0 and 1 are assigned to the two SCSI controllers on the IO4 board. Two through seven are assigned to the controllers on the mezzanine cards that are installed on the IO4. The fourth digit represents a specific drive. The fifth digit is the partition on the selected drive. See Figure 1-6 for an example of SCSI drive addressing.

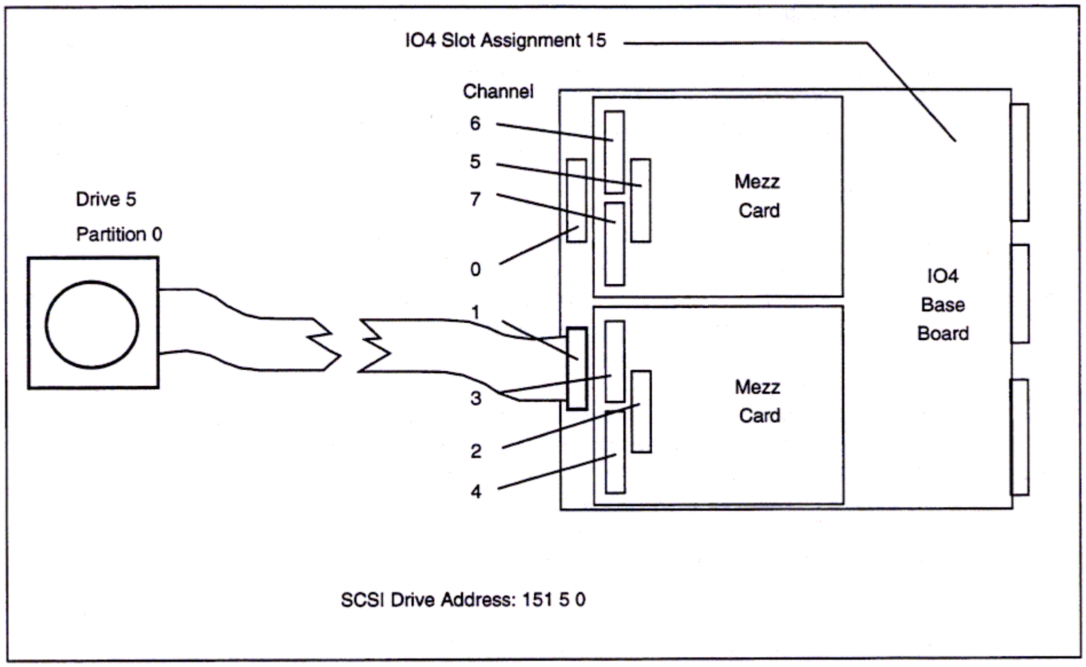


Figure 1-6 SCSI Drive Addressing

FCI Bus

The Flat Cable Interface (FCI) is a Silicon Graphics-proprietary interface used to connect a variety of local and remote peripheral resources, such as; graphics controllers, VMEbus adapters, and FDDI adapters. The FCI operates at 160 MB per second (maximum) for graphics controllers, and at a maximum of 200 MB per second with VMEbus adapters.

Note: The maximum effective VMEbus transfer rate may be in the range of 50 MB per second with DMA. The transfer rate is less with PIO-driven I/O.

1.3 ASIC Error Detection

There are various points in the Everest system where the accuracy of the information being transferred is checked. Different error checking methods are used depending upon the particular system interface. These methods include parity bits, error correction codes (ECCs), timeouts, or a combination of several methods.

Errors are generally propagated from the point of origin on throughout the system. An error is flagged at both the sending and receiving end of every interface it crosses, and an error message written to CPU-accessible registers. Eventually, the error is recognized by one or more CPUs, which then take appropriate action. How soon the error is recognized and whether or not the system can identify the origin of the error depends upon the type of operation that generated the fault.

For example, a memory read is an operation that provides a high rate of success in tracing the error back to the origin. If a CPU issues a memory or PIO read, and that read generates an error, the CPU takes a sychronous exception. Because the exception handler is invoked so soon following the error, there is a better possibility that the cause of the error can be determined.

An example of a difficult fault to trace is one that occurs during a memory write. If a CPU issues a memory or PIO write, and an error occurs, an error interrupt is sent to one of the CPUs. The CPU receiving the interrupt may not be the same CPU that issued the write operation. The difficulty is compounded when the error occurred during a transaction that originated in a DMA controller.

The Ebus is highly pipelined, and an operation, once initiated, may not be completed until some time later. Understanding these asynchronous operations requires an understanding of the ways errors propagate through the hardware. In the Everest board set, all interfaces are bridged by one or more ASICs. By associating specific error bits with a particular ASIC, and by establishing the direction of information transfer within an interface, the error can generally be traced back to its point of origin (or to a FRU level).

The figures in this section provide the locations of the error checking logic for each board, as well as the direction in which the information is flowing when checked. The accompanying tables list the various registers used to store error messages for each of these checkpoints. Figure 1-7 provides an overview of the points in the system where errors are detected. Figures 1-8 through 1-13 illustrates parity checking on each board. Tables 1-1 through 1-3 list the error registers for each board.

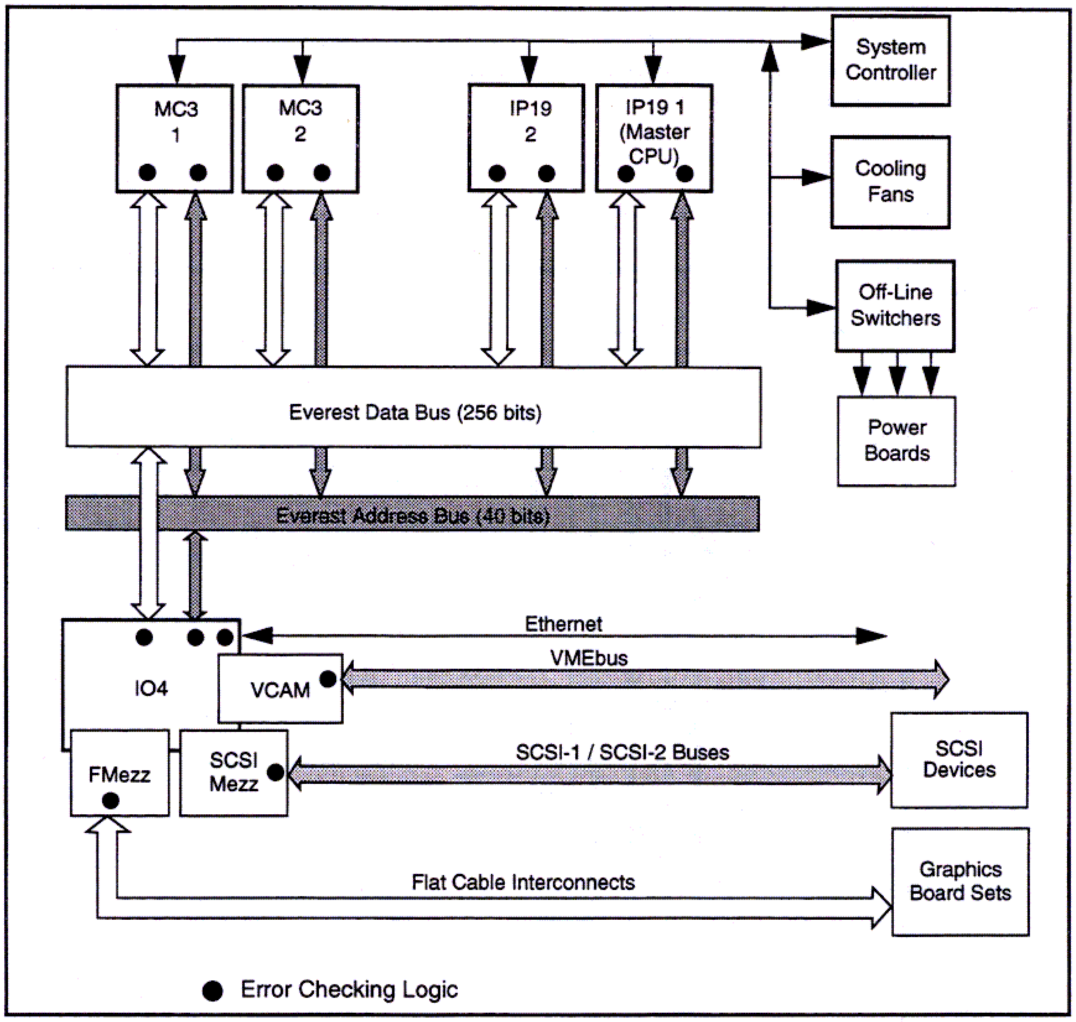


Figure 1-7 Everest Bus Parity Checkpoints

1.3.1 Error Messages

When the hardware detects an error, UNIX and the diagnostics display it in a format called the HARDWARE ERROR STATE display (see Chapter 5 for a detailed description of the error syntax). This section provides a complete list of the HARDWARE ERROR STATE messages, arranged by board type. Each message contains a number representing the location of the error checking logic (usually a bus-to-ASIC interface), as well as a description of the error bit.

Each circuit board is represented by both a functional block diagram, and by a board layout showing the physical locations of the error detection logic. Following the two figures that support each board is a table that lists the possible hardware errors, along with a number that identifies the location where the error was detected.

1.3.1.1 IP19 CPU Board

Figure 1-8 is a functional block diagram of the IP19 board with the error detection points called out. Figure 1-9 shows the physical layout of the board and the locations of the error detection logic.

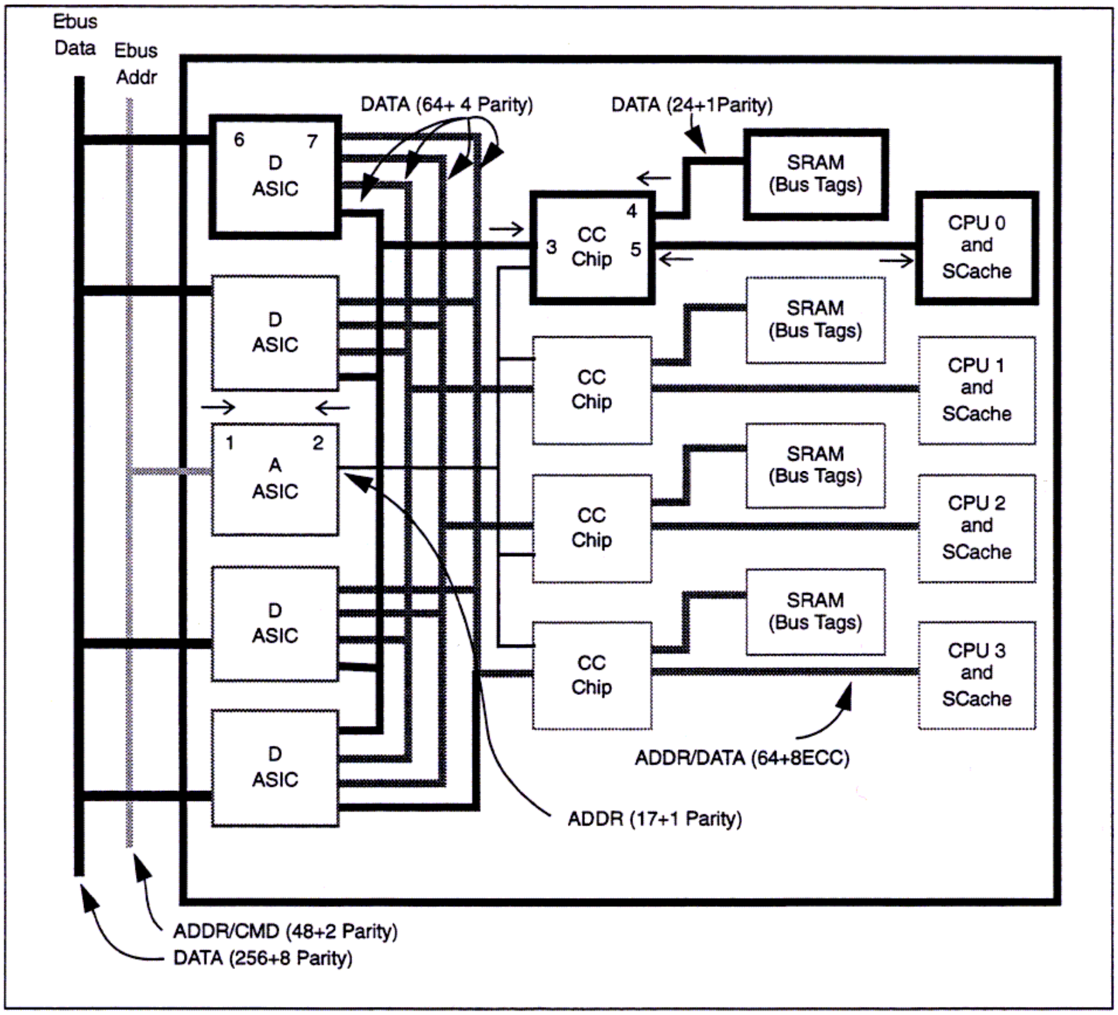


Figure 1-8 IP19 Board Error Detection Logic

Note: Because each of the four processor slices are identical, only the registers corresponding to slice 0 are described in the following section.

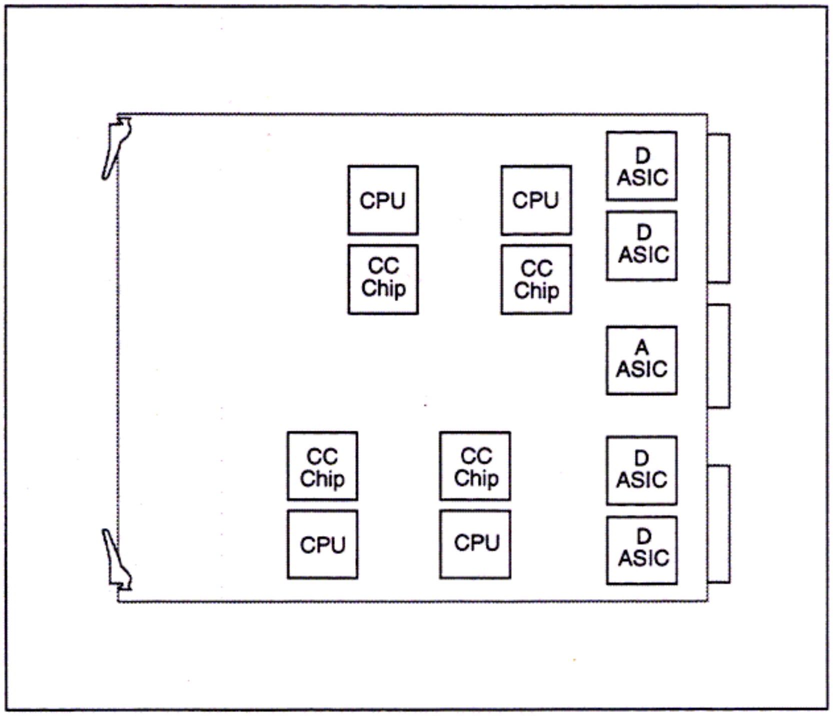


Figure 1-9 IP19 Board Component Locations

IP19 Board Error Messages

```
HARDWARE ERROR STATE:
IP19 in slot 1
+      A Chip Error Register: 0xffff
+         0:CPU 0 CC->A parity error          2
+         1:CPU 1 CC->A parity error          2
+         2:CPU 2 CC->A parity error          2
+         3:CPU 3 CC->A parity error          2
+         4:ADDR_ERROR on EBUS                1  this A detected some
                                                 board emitted an address with
                                                 bad parity.
+         5:My ADDR_ERROR on EBUS             1  some board detected a parity
                                                 error in my emitted address

+
+
+         8:CPU 0 CC->D parity error          7  detected by D, for a cache
                                                 line write or upon an EBus
                                                 intervention reading a line

+         9:CPU 1 CC->D parity error          7
+        10:CPU 2 CC->D parity error          7
+        11:CPU 3 CC->D parity error          7
+        12:CPU 0 ADDR_HERE not asserted      1  address emitted by this A
                                                 was not decoded by any board
+        13:CPU 1 ADDR_HERE not asserted      1
+        14:CPU 2 ADDR_HERE not asserted      1
+        15:CPU 3 ADDR_HERE not asserted      1
+    CC in IP19 Slot 1, cpu 0
+      CC ERTOIP  Register: 0xffff
+        0:ECC uncorrectable error in Scache    5  detected by R4000
+        1:ECC correctable error in Scache      5  detected by R4000, upon
                                                   R4000 read or upon an EBus
                                                   intervention reading a line
+        2:Parity Error on TAG RAM Data         4  detected by CC
+        3:Parity Error on Address from A-chip  3  in path from A to CC
+        4:Parity Error on Data from D-chip     3  in path from D to CC, or
                                                   when D receives data with
                                                   bad parity from EBus.
+        5:MyRequest TimeOut on EBUS            1  A was not able to get EBus
                                                   access, to emit its request
+        6:MyResponse D-Resource TimeOut in A chip
                                                1  EBus did not return a
                                                   read-response to A
+        7:MyIntervention Response D-Resource TimeOut in A chip
                                                2  this CC returned an
                                                   intervention response to
                                                   A too late
+        8:Address Error on MyRequest on EBUS   1  one or more boards detected
a                                                  parity error in A emitted
                                                   address, or ADDR_HERE not
                                                   asserted (no board decoded
                                                   A emitted address)
+        9:Data Error on MyData on EBUS         1  some board detected a parity
                                                   error in my emitted data
+        10:Internal Bus State is out of sync with A_SYNC
                                                3
+    CC in IP19 Slot 1, cpu 2
+    CC in IP19 Slot 1, cpu 3
```

Note: The numbers following each error message correspond to the interface where the error detection logic is located. These registers are duplicated for each installed processor.

1.3.1.2 IO4 Interface Board

Figure 1-10 is a functional block diagram of the IO4 board and VCAM with the error detection points called out. Figure 1-11 shows the physical layout of the IO4 board and the locations of the error detection logic. The error messages are listed in the following section.

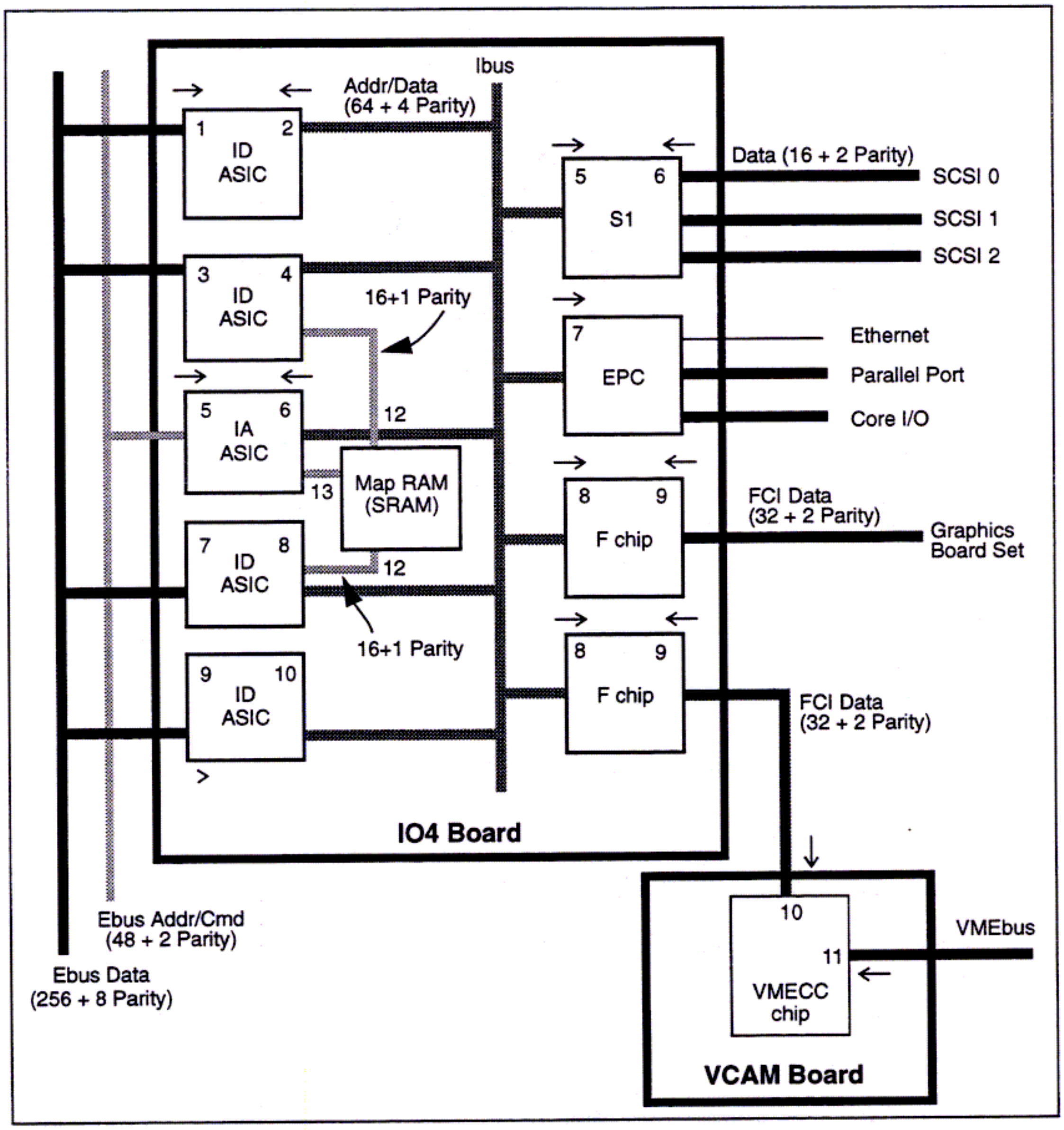


Figure 1-10 IO4/VCAM Board Error Detection Logic

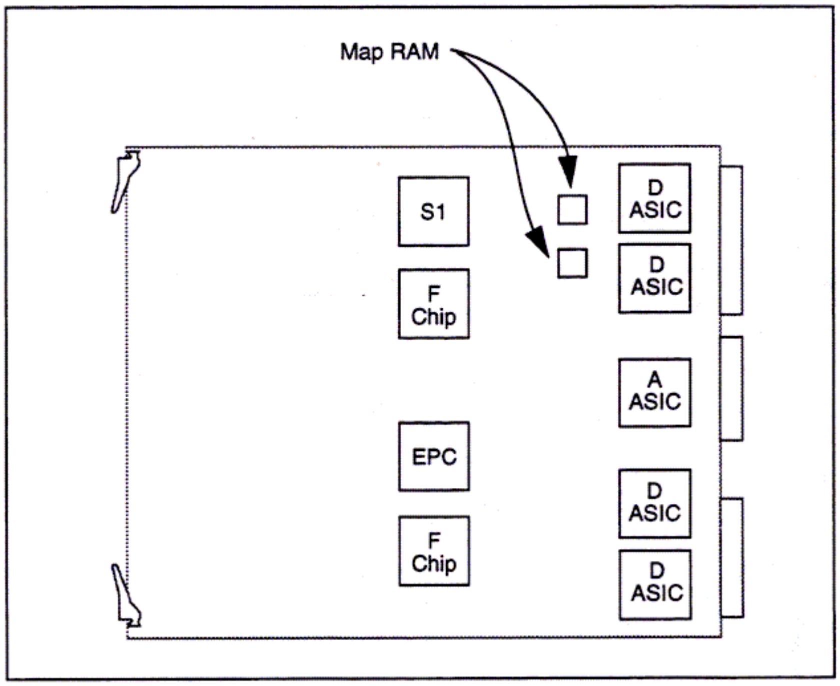


Figure 1-11 IO4/VCAM Component Locations

IO4/VCAM Error Messages

```
+  IO4 board in slot 5
+      IA IBUS Error Register: 0x7ffff
+         0: Sticky Error                    More than one occurance of
                                             one or more of the following
+         1: First Level Map Error for 2-Level Mapping
                                             12  MAPRAM data parity error
                                                 detected by ID
+         2: 2-Level Address Map Response Command Error
                                             4   F chip detected bad parity
                                                 on IBus operation from IA
+         3: 1-Level Map Data Error          12  MAPRAM data parity error
                                                 detected by ID
+         4: 1-Level Address MapResponse Command Error
                                             4   F chip detected bad parity
                                                 on IBus operation from IA
+         5: IA Response Data Bad On IBUS    4
+         6: DMA Read Response Command Error 4   F chip detected bad parity
                                                 on IBus operation from IA
+         7: GFX Write Command Error         4   F chip detected bad parity
                                                 on IBus operation from IA
+         8: PIO Read Command Error          4   target S1/EPC/F detected
                                                 bad parity on IBus operation
                                                 from IA
+         9: PIO Write Data Bad              4   target S1/EPC/F detected
                                                 bad parity on IBus data
                                                 from ID
+        10: PIO write Command Error         4  target S1/EPC/F detected
```

```
                                                   bad parity on IBus operation
                                                   from IA
+        11: PIO ReadResponse Data Error        2  ID detected bad parity
                                                   from S1/EPC/F
+        12: DMA Write Data Error (Data From IOA)
                                                2  ID detected bad parity
                                                   from S1/EPC/F
+        13: DMA Write Command Error from IOA   4  IA detected bad parity
                                                   from S1/EPC/F
+        14: PIO Read Response Command Error from IOA
                                                4  IA detected bad parity
                                                   from S1/EPC/F
+        15: Command Error on OP from IOA       4  IA detected bad parity
                                                   from S1/EPC/F
+        16: IOA number of Transaction:         7  adaptor number which caused
                                                   error, only valid for errors
                                                   detected by IA or ID.

+       IA EBUS Error Register: 0xfe0003ff
+         0: Sticky Error
+         1: My DATA_ERROR Received             3  one or more boards detected
                                                   a parity error in my emitted
                                                   data, emitted by ID
+         2: ADDR_ERROR Detected                3  this IA detected either
                                                   another board emitted an
                                                   address with bad parity, or
                                                   ADDR_HERE not asserted (no
                                                   board decoded someones
                                                   emitted address)
+         3: Non Existent IOA                   3  No F/S1/EPC configured at
                                                   specified address, probable
                                                   software error
+         4: Illegal PIO                        3  CC Write Gatherer block
                                                   write only allowed to F+FCG,
                                                   probable software error
+         5: My ADDR_ERROR Received             3  one or more boards detected
a                                                  parity error in IA emitted
                                                   address, or ADDR_HERE not
                                                   asserted (no board decoded
                                                   IA emitted address)
+         6: EBUS_TIMEOUT Received              3  IA was not able to get
                                                   EBus access, to emit its
                                                   request
+         7: Invalidate Dirty Exclusive Cache Line
                                                3  EBus cache coherence
                                                   protocol violation detected
                                                   by IA
+         8: Read Resource Time Out             3  EBus did not return a
                                                   read-response to IA
+         9: DATA_ERROR Received                3  ID detected bad parity
                                                   on data from EBus
+        25: PIO Queue full for Adapter 1       4  S1/EPC/F is not responding
+        26: PIO Queue full for Adapter 2       "
+        27: PIO Queue full for Adapter 3       "
+        28: PIO Queue full for Adapter 4       "
+        29: PIO Queue full for Adapter 5       "
+        30: PIO Queue full for Adapter 6       "
+        31: PIO Queue full for Adapter 7       "
+      IA Error Ebus Address: 0x4              Holds the EBus address of
                                               the transaction which caused
                                               a parity error on EBus.
                                               Valid if bit 5 in this
                                               register is set.
+        40: EBus Outgoing Command: 0x54       Command on EBus which caused
                                               parity error on EBus
+    EPC in IO4 slot 5 adapter 1
+      Ibus Error Register: 0x3e6311
+          3..0: EPC Detected- PIO Write Request Data Error from IA
                                               7
+          3..0: EPC Detected- DMA Read Response Data Error from IA
                                               7
+          3..0: EPC Detected- Unexpected DMA Read Response Error from IA
                                               7
+          3..0: EPC Detected- Undetermined Command Error from IA
                                               7
+         11..4: EPC Observed- PIO Write Request Data Error from IA to IOA 3
                                               7
+         11..4: EPC Observed- GFX Write Request Data Error from IA to IOA 3
                                               7
+         11..4: EPC Observed- DMA Read Response Data Error from IA to IOA 3
                                               7
+        15..12: EPC Sent- PIO (to EPC) Read Response Command Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- PIO (to EPC) Read Response Data Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- DMA (Enet) Read Request Command Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- DMA (Enet) Write Request Command Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- DMA (Enet) Write Request Data Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- Interrupt Request Command Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- PIO (thru EPC) Read Response Command Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- PIO (thru EPC) Read Response Data Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- DMA (PPort) Read Request Command Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- DMA (Pport) Write Request Command Error to IA
                                               7 Error from EPC to IA
+        15..12: EPC Sent- DMA (Pport) Write Request Data Error to IA
                                               7 Error from EPC to IA
+        16: Parity Error on IBusAD[15:0]
+        17: Parity Error on IBusAD[31:16]     7 Error from EPC to IA
+        18: Parity Error on IBusAD[47:32]     7 Error from EPC to IA
+        19: Parity Error on IBusAD[63:48]     7 Error from EPC to IA
+        20: Error Overrun
+        21: DMA Read Response 1 msec Timeout Error
+      IBus Opcode+Address: 0x2a40             7 Holds the IBus contents if
                                                 EPC detects an error in IA
                                                 initiated transaction
+    Fchip in IO4 slot 5 adapter 3, FCI master: FCG
                                                 F chip connects
                                                 to graphics
```

```
+    Fchip in IO4 slot 5 adapter 2, FCI master: VMECC
                                                   F chip connects
                                                   to VME bus
+      Error Status Register: 0x4ffffff
+         0: OverWrite
+         1: Loopback Received
+         2: Loopback Error
+         3: F to IBus Command Error - non-interruptable
                                                 8  IA detected parity error
                                                    on command from F
+         4: PIO Read Response IBus Data Error - non-interruptable
                                                 8  ID detected parity error
                                                    on data from F
+         5: DMA Read Request Timeout Error      8  IA did not return DMA
                                                    read response
+         6: Unknown IBus Command Error          8  F detected parity error
                                                    on command from IA
+         7: DMA Read Response Ibus Data Error   8  F detected parity error
                                                    on data from ID
+         8: DMA Write Data FCI Error            9  F detected parity error
                                                    on data from FCI
+         9: PIO Read Response FCI Data Error    9  F detected parity error
                                                    on data from FCI
+        10: PIO/GFX Write IBus Data Error       8  F detected parity error
                                                    on data from ID
+        11: Load Address Read FCI Error         9  F detected parity error
                                                    on address from FCI
+        12: DMA Write IBus Command Error        8  IA detected parity error
                                                    on command from F
+        13: Address Map Request IBus Command Error
                                                 8  IA detected parity error
                                                    on command from F
+        14: Interrupt IBus Command Error        8  IA detected parity error
                                                                on command from F
+        15: Load Address Write FCI Error        9  F detected parity error
                                                    on data from FCI
+        16: Unknown FCI Command Error           9  F detected parity error
                                                    on command from FCI
+        17: Address Map Request Timeout Error   8  IA did not return map
                                                    response
+        18: Address Map Response Data Error     8  F detected parity error
                                                    on data from ID
+        19: PIO F Internal Write IBus Data Error
                                                 8  F internal problem
+        20: IBus Surprise                       8  F received unexpected
                                                    bus-grant/DMA-response
                                                    from IA
+        21: DMA Write IBus Data Error           8  ID detected parity error
                                                    on data from F
+        22: System FCI Reset
+        23: Software FCI Reset
+        24: Master Reset                        9  F received a reset request
                                                    from FCI
+        25: F Error FCI Reset                   9  F reset FCI due to some error
+        26: F Chip Reset in Progress
+        27: Drop PIO Write Mode                 8  F unrecoverable error
+        28: Drop DMA Write Mode                 8  F unrecoverable error
+        29: Fake DMA Read Mode                 8  F unrecoverable error
+      IBus Opcode+Address: 0xa54b              8  captures the F operation
                                                   which received a parity
                                                   error.
+      FCI  Error Command: 0x3f                 9  captures the FCI command
                                                   which had a data parity error
+    VMECC in IO4 slot 5 adapter 2
+      Error Cause Register: 0x1fff
+        0: VME Bus Error on PIO Write (interrupts)
                                                11  error from VME
+        1: VME Bus Error on PIO Read (no interrupt), return bad parity data
                                                11  error from VME
+        2: VME Slave Got Parity Error (no interrupt)
                                                10  detected error on FCI
+        3: VME Acquisition Timeout by PIO Master (interrupt, set dropmode)
                                                11  error from VME
+        4: FCIDB Timeout (no interrupt)        10  detected error on FCI
+        5: FCI PIO Parity Error (interrupt)    10  detected error on FCI
+        6: Overrun among bit 0,1,3,4,5
+        7: in Dropmode
+      VME Address Error Register: 0xa0001248
+      Extra VME Bus signal register: 0x19e49
+        AM: 0x9 IACK Read AS DS0 DS1
+        VME-Grant-Level: 1: VME backplane levels
+    S1 in IO4 slot 5 adapter 4
+      S1 Command Status Register: 0xffff
+        6 : Error in SCSI Data DMA channel 0   6  detected error from SCSI
+        7 : Error in SCSI Data DMA channel 1   6  detected error from SCSI
+        8 : Error in SCSI Data DMA channel 2   6  detected error from SCSI
+        9 : PIO Read Error in SCSI 0           6  detected error from SCSI
+        10: PIO Read Error in SCSI 1           6  detected error from SCSI
+        11: PIO Read Error in SCSI 2           6  detected error from SCSI
+        12: PIO Write data Overrun due to PIO FIFO Full
                                                5  S1 internal error
+        13: Missing Write data during PIO write
                                                5  S1 detected error on IBus
+        14: PIO with an invalid address        5  S1 detected error on IBus
+        15: PIO Drop mode active               5  S1 detected error on IBus
+      S1 Ibus error Register: 0x1fffffff
+         1: Error on Incoming Data to S1 - multiple occurances
                                                5  S1 detected error on IBus
+         2: Error on Incoming command to S1 - multiple occurances
                                                5  S1 detected error on IBus
+         3: Error on Outgoing data from S1 - multiple occurances
                                                5  S1 detected error on IBus
+         4: Error on Outgoing command from S1 - multiple occurances
                                                5  S1 detected error on IBus
+         5: Error in DMA translation - multiple occurances
                                                5  S1 detected error on IBus
+         6: Error in Channel 0 - multiple occurances
+         7: Error in Channel 1 - multiple occurances
+         8: Error in Channel 2 - multiple occurances
+         9: Surprising DMA Read/Ibus Grant - multiple occurances
                                                5 S1 detected error on IBus
+        10: PIO Read response Error - multiple occurances
                                                5  IA detected error in data
                                                   from S1
```

```
+        11: PIO Write request Error - multiple occurances
                                                   5  S1 detected error on IBus
+        12: DMA Read response Error - multiple occurances
                                                   5  S1 detected error on IBus
+        13: Interrupt Error - multiple occurances
                                                   5  IA detected error
+      IBus Opcode+Address: 0xbfe0                 5  holds the S1 emitted IBus
                                                      operation where a Parity
                                                      error was reported by IA
```

1.3.1.3 MC3 Memory Board

Figure 1-12 is a functional block diagram of the MC3 board with the error detection points called out. Figure 1-13 shows the physical layout of the board and the locations of the error detection logic. The error messages are listed in the following section.

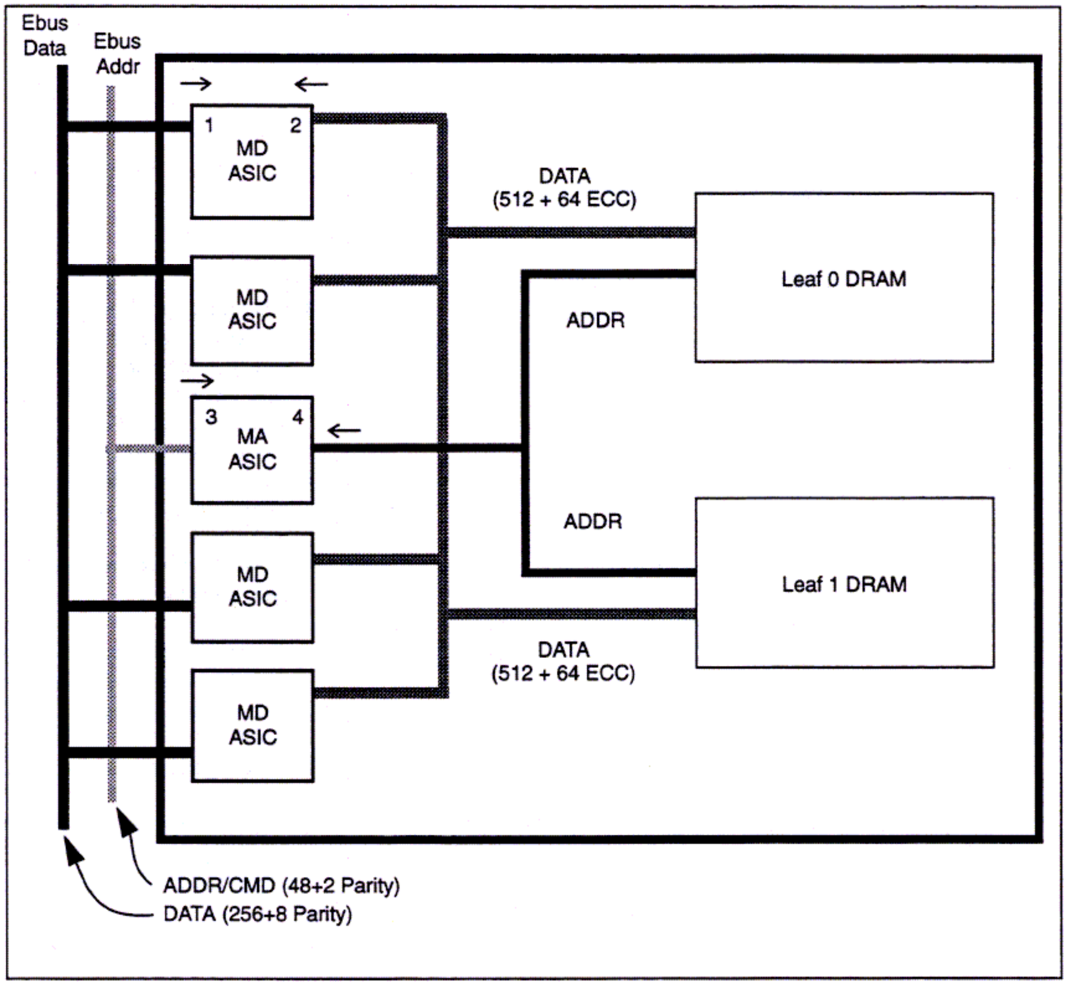


Figure 1-12 MC3 Memory Board Error Detection Logic

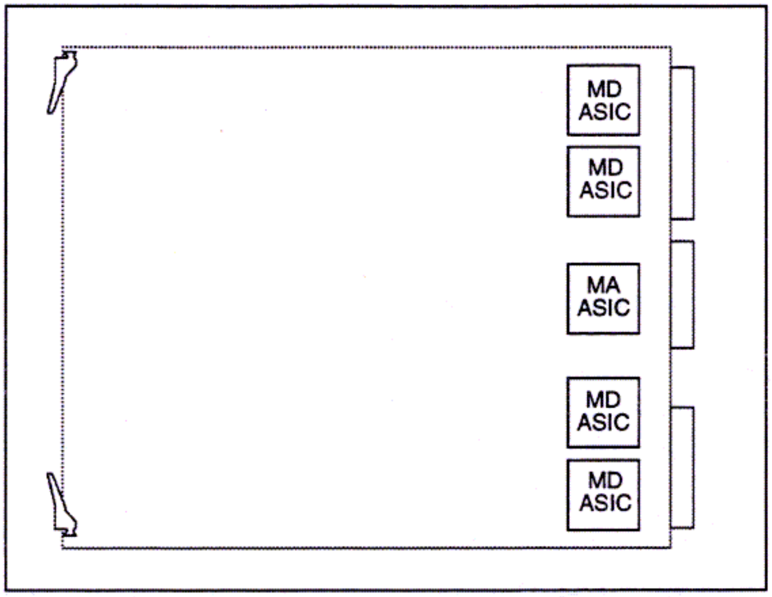


Figure 1-13 MC3 Board Component Locations

MC3 Memory Board Error Messages

```
+  MC3 in slot 3
+      MA Ebus Error register: 0xf
+        My EBus Address Error                  3 destination board detected
                                                  a parity error in MA emitted
                                                  read-response ? field
+        My EBus Data Error                     3 one or more boards detected
                                                  a parity error in MD emitted
                                                  read-response data
+        EBus Address Error                     3 this MA detected another
                                                  board emitted an address with
                                                  bad parity
+        EBus Data Error                        1 this MD received data with
                                                  bad parity

+      MA Leaf 0 Error Status Register: 0xf
+        Multiple Occurence of these errors:      Shows more than one Read
                                                  Single Bit Error occurred
+        Read Single Bit Error                  2 correctable error in this
                                                  leaf
+        Read Uncorrectable Error               2 uncorrectable error in this
                                                  leaf
+        PartialWrite Uncorrectable Error       2 uncorrectable error upon
                                                  reading out this leaf, to do
                                                  a partial write merge

+      MA Leaf 1 Error Status Register: 0xf
```

Chapter 2

Power Subsystem

2.1 Overview

The power subsystem consists of the off-line switchers (OLSs), the midplane and backplane power buses, the various power boards, and the DC-to-DC converters (power bricks) on the Everest CPU and Memory boards. See Figure 2-1 for a block diagram illustrating the power subsystem components.

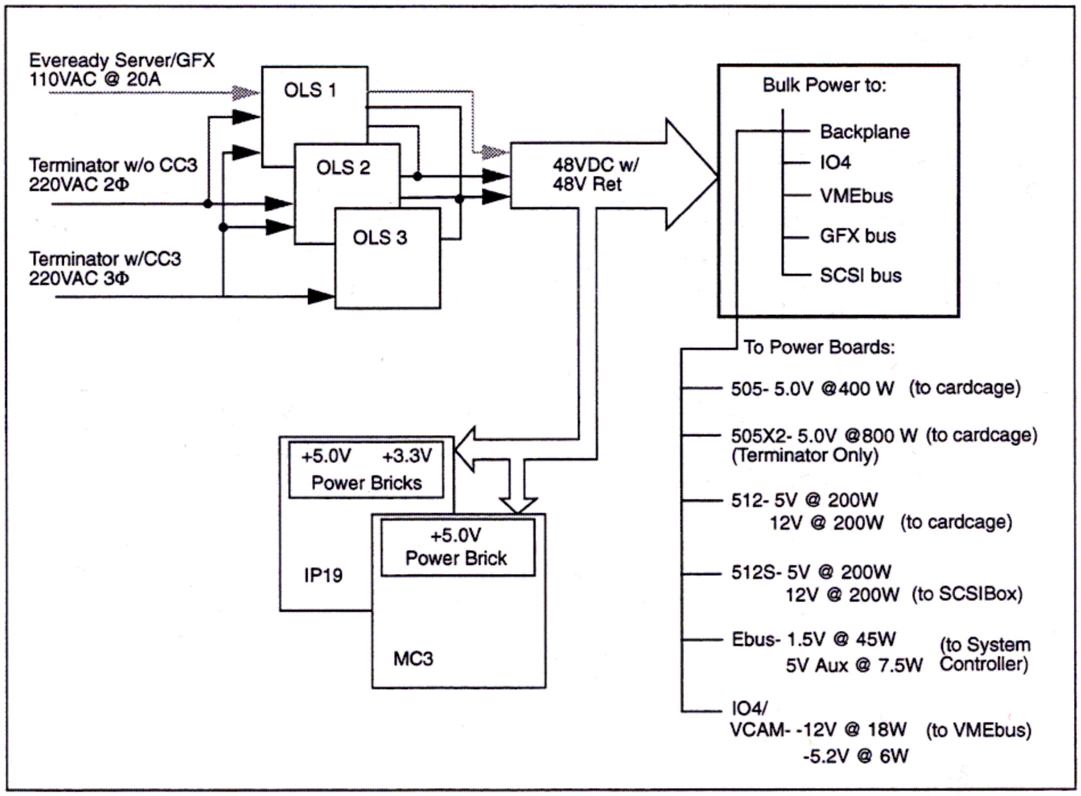


Figure 2-1 Power Subsystem Block Diagram

When the system is turned on, the power subsystem goes through a series of voltage checks before the boot process is allowed to start. Power is applied to the various system components sequentially in the following order: 5V and 12V power bricks (powers the SCSI drives in the deskside systems), 1.5V and 3.3V power bricks, 5V and 12V (powers the internal SCSIBox in the rackmount systems), 5V and 12V (powers the second internal SCSIBox in the rackmount systems), and 5V and 12V for external SCSI. This power sequencing is designed to prevent component damage due to incorrect or missing voltages, and to avoid placing a large transient demand on the voltage source.

There are only three diagnostic tools at this point in the system's start-up sequence: the red fault LEDs on each circuit board, the voltage input and output LEDs on each OLS, and the System Controller. To effectively use these indicators to troubleshoot a system fault, refer to the fault indicator descriptions and the power-up sequence described in the following sections.

2.2 Power Fault Indicator Descriptions and Locations

This section describes the power fault indicators found on each board, on each OLS, on the SCSIBox backplane, and on the System Controller. The locations of the power fault indicators, the power bricks, the removable fuses, and the secondary regulators (where applicable) are also shown.

2.2.1 System Controller and OLSs

There are two LEDs located above the System Controller function buttons (see Figure 2-2). The green power-on LED lights to indicate that 48 volts is present at the system midplane/backplane, and remains lit as long as 48V is present. The amber fault LED lights briefly during the power-up sequence, but should go out when the Power-on tests are complete.

Each OLS also has a green and amber LED. The amber LED lights to indicate that the AC input voltage level is within acceptable levels. The green LED lights to indicate that the DC output voltage levels are within acceptable levels. Both LEDs should remain lit during normal system operation.

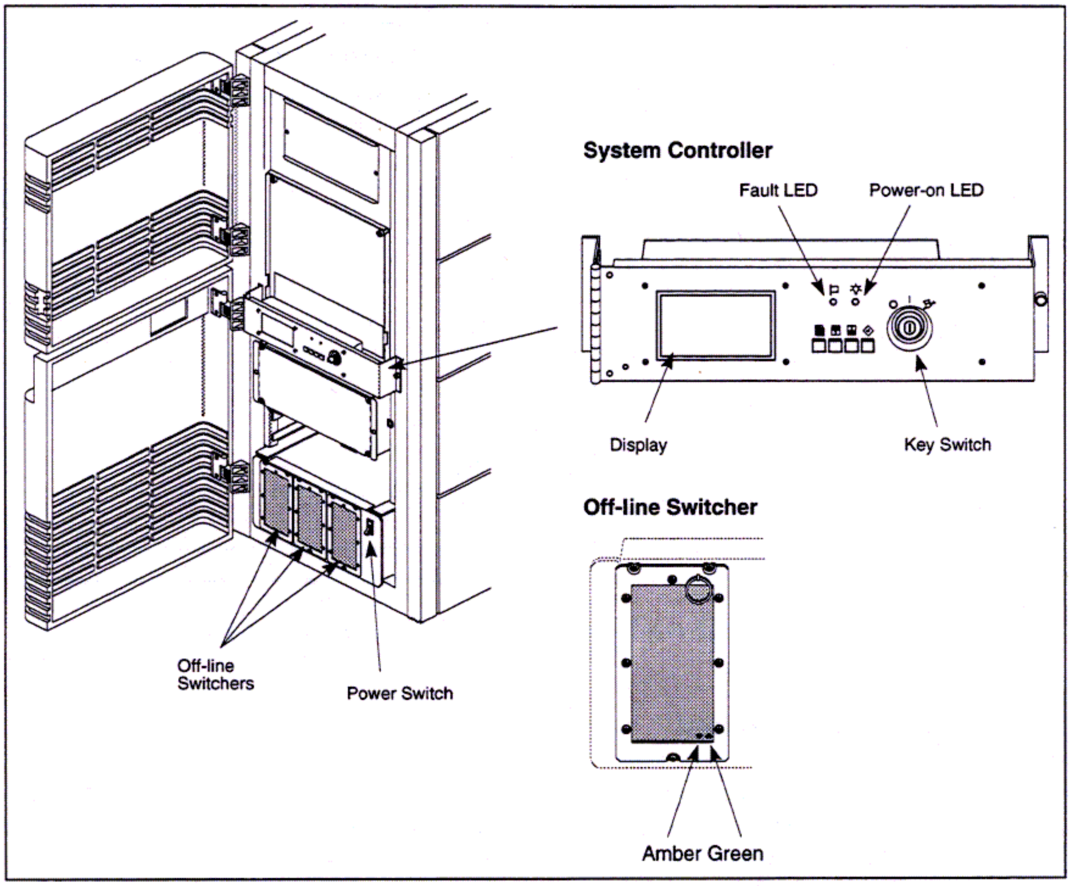


Figure 2-2 System Controller and OLS Power Indicators

2.2.2 IP19 CPU Board

The IP19 has two power bricks that step down the 48 volts from the midplane/backplane to 5.0 and 3.3 volts. Each brick has a corresponding power fault LED, as shown in Figure 2-3.

Note: The LEDs are red and indicate a fault when lit.

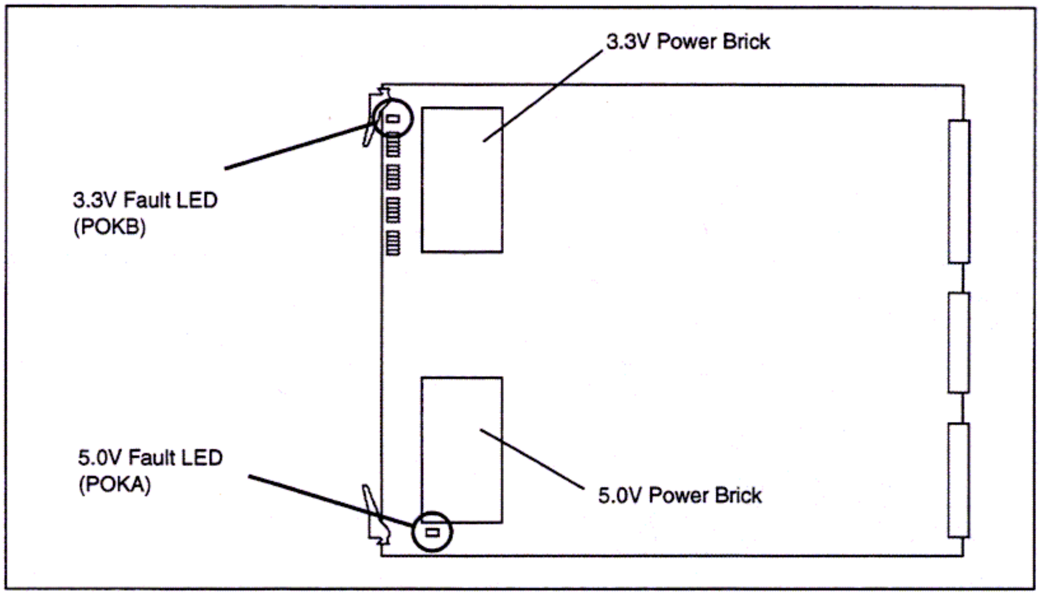


Figure 2-3 IP19 Board Power Fault Indicators and Power Brick Locations

2.2.3 MC3 Memory Board

The MC3 board currently has a single +5.0-volt power brick, and a corresponding fault indicator. Later versions of this board may have an additional +3.3V brick, as shown in Table 2-1 and Figure 2-4.

Note: The LEDs are red and indicate a fault when lit.

LED Reference Designation	Color / Meaning When Lit	Description
N8P2 (POKB)	Red - Fault	Bad 3.3V power brick
B4P2 (POKA)	Red - Fault	Bad 5.0V power brick

Table 2-1 MC3 Board Fault LEDs

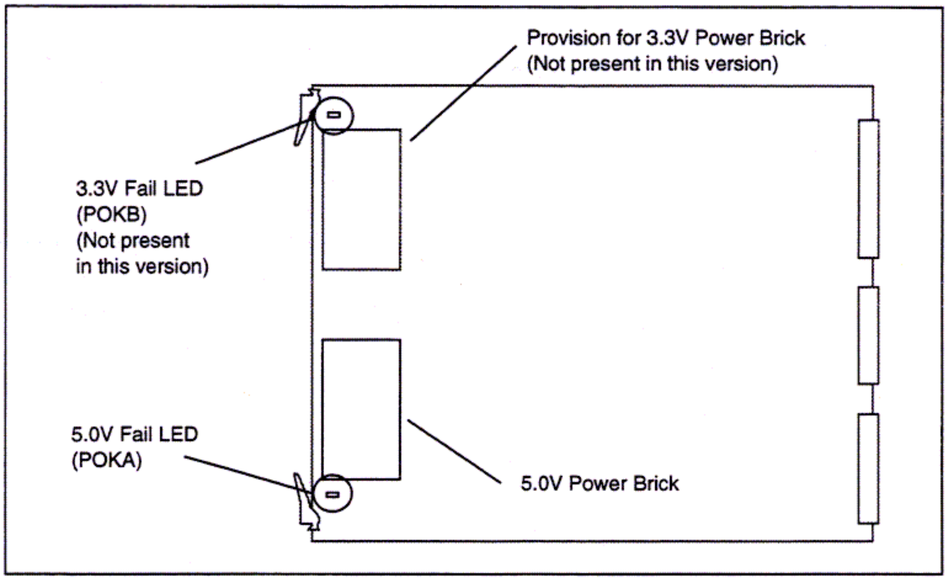


Figure 2-4 MC3 Board Fault Indicator and Power Brick Locations

2.2.4 IO4 Board

The IO4 board has a single bank of five LEDs and two secondary regulators. These regulators convert +5 volts to +1.5 volts (see Table 2-2 and Figure 2-5). The IO4 also has three replaceable fuses, as shown in Figure 2-5.

Note: All LEDs are red and indicate a fault when lit. The bottom three LEDs provide power fault indications for the attached VMEbus Channel Adapter Module (VCAM). The VCAM fault LEDs are mounted on the IO4 board because the dimensions of the VCAM would make on-board LEDs extremely difficult to read. These LEDs are unlit when no VCAM is installed.

LED Reference Designation	Color / Meaning When Lit	Description
M2P6 (POKB)	Red - Fault	Bad 1.5V regulator A (near top) on the IO4 board
M1P6 (POKB)	Red - Fault	Bad 1.5V regulator B (near bottom) on the IO4 board
M0P6 (POKB)	Red - Fault	Bad 1.5V regulator on the VCAM
L9P6 (POKA)	Red - Fault	Bad -12V to -5.2V regulator on the VCAM

Table 2-2 IO4 Board Fault LEDs

LED Reference Designation	Color / Meaning When Lit	Description
L8P6 (POKA)	Red - Fault	Bad +12.0V to -12.0V regulator on the VCAM

Table 2-2 (continued) IO4 Board Fault LEDs

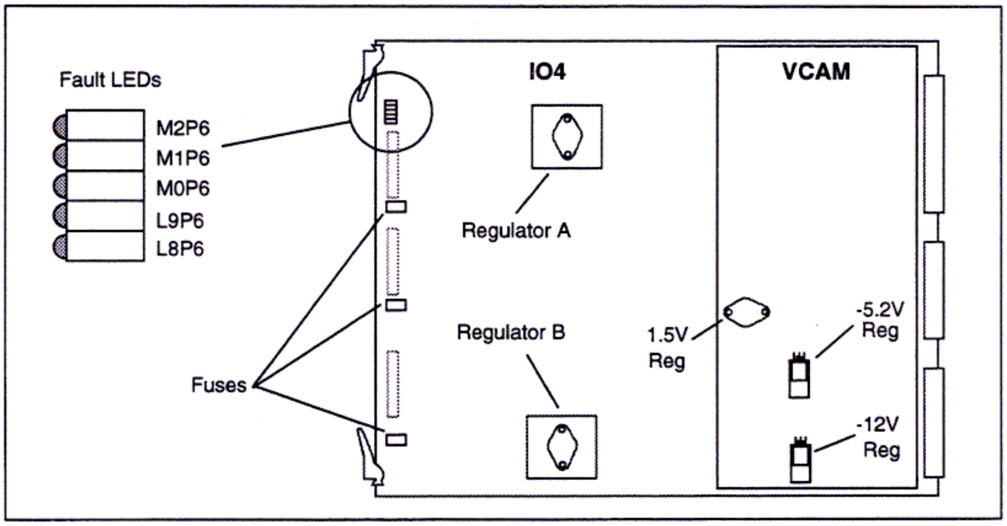


Figure 2-5 IO4 Board/VCAM Fault Indicator and Voltage Regulator Locations

2.2.5 Remote VCAM (RMT_VCAM) Board

The RMT_VCAM board has a bank of nine fault LEDs that flag power faults stemming from the Cardcage 3 backplane, the System Controller, and the three regulators on the RMT_VCAM itself. There are also six test points corresponding to the monitored voltages (see Table 2-3 and Figure 2-6).

Note: The voltage levels of the three on-board voltage regulators are monitored by two sets of LEDs: the three red LEDs and the bottom three amber LEDs. The red LEDs light when a voltage error is sensed and remain lit until the system is reset. The amber LEDs provide a "hot" measurement and light only when an error in the monitored voltage levels is currently present.

LED Reference Designation	Color / Meaning When Lit	Description
M0P6	Red - Fault	Bad +1.5V regulator on the RMT_VCAM
L9P6	Red - Fault	Bad -5.2V regulator on the RMT_VCAM

Table 2-3 Remote VCAM Fault LEDs

LED Reference Designation	Color / Meaning When Lit	Description
L8P6	Red - Fault	Bad -12V regulator on the RMT_VCAM
L7P6	Green - Good	5V input (V5_AUX) from System Controller to RMT_VCAM (should always be on)
L6P6	Amber - Fault	Bad +12V input from the backplane
L5P6	Amber - Fault	Bad +5V input (VCC) from the backplane
L4P6	Amber - Fault	Bad +1.5V regulator on the RMT_VCAM
L3P6	Amber - Fault	Bad -5.2V regulator on the RMT_VCAM
L2P6	Amber - Fault	Bad -12V regulator on the RMT_VCAM

Table 2-3 (continued) Remote VCAM Fault LEDs

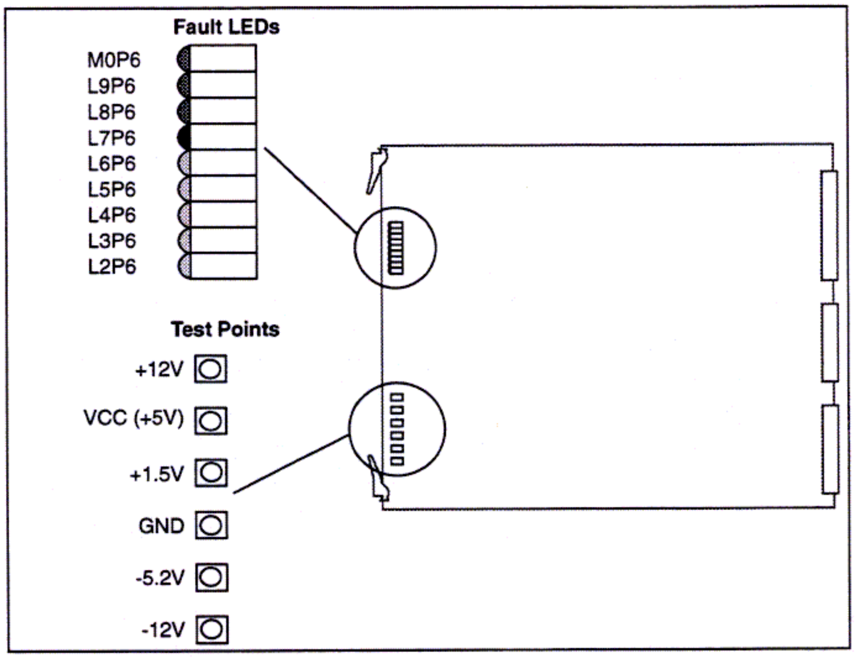


Figure 2-6 Remote VCAM Fault Indicator and Voltage Regulator Locations

2.2.6 Mezzanine (F Mezz and S Mezz) Boards

Both F mezzanine cards (F Mezz and Short F Mezz) have a 5V-to-1.5V regulator, identical to those on the IO4 and VCAM. Each F mezz board also has a single, red voltage fault LED, as shown in Figure 2-7. The LED lights when the voltage level is out of range.

The S mezzanine card (not shown) has no regulators, but has three removable fuses.

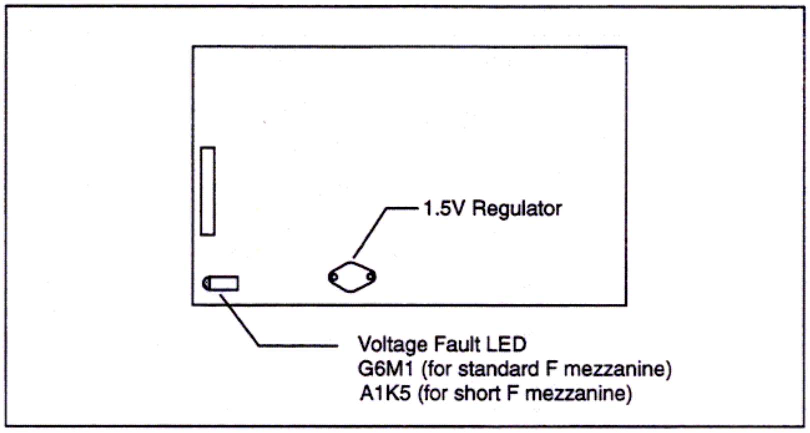


Figure 2-7 F Mezzanine Board Fault Indicator and Voltage Regulator Locations

2.2.7 SCSIBox Drive Enclosure

There is a +5V and a +12V power fault LED located behind each drive in the SCSIBox. Both LEDs are green and are lit during normal operation (see Figure 2-8).

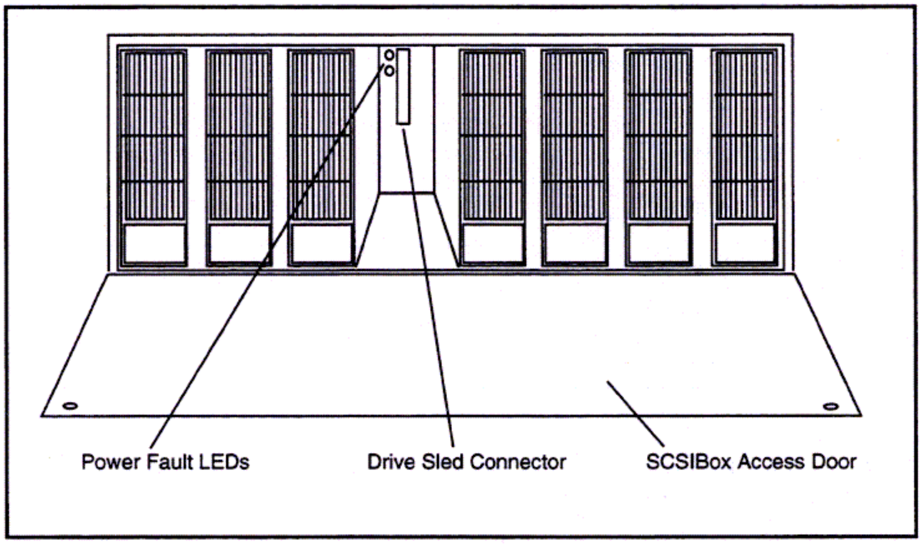


Figure 2-8 SCSIBox Fault Indicators

2.3 Power-up Sequence

Note: The Power switch (main circuit breaker), located in the lower right corner of the front of the system cabinet, must be turned on.

Turning the System Controller key switch to the ON or Manager position enables the OLSs to output 48 volts to the midplane/backplane. The green power-on LED, above the front panel display, lights to indicate that voltage is present at the midplane/backplane. The step-down regulator on the Ebus power board converts the 48 volts to 5.0 volts (V5_AUX) for use by the System Controller, power brick control circuits, and LEDs. As the System Controller powers up and begins its initialization, the amber fault LED above the display lights. When the System Controller successfully initializes and the Power-on tests are complete, the amber fault LED will go out.

Note: If the system will not power up (green front panel LED not lit), first check the LEDs on each of the off-line switchers. If the OLSs do not indicate a fault, verify that there is 48 volts at the backplane near the Power-on LED.

If the backplane voltage is correct, but the System Controller has not initiated the power sequencing, first check the display for an error message, then check the Ebus board and verify that the 5.0V_AUX line from the System Controller is supplying power to the power bricks.

The System Controller then manages the power sequencing for the rest of the system. A series of power-enable (PENx) signals are asserted:

- PENA - controls +/-5.0 and +/-12 volts
- PENB - controls 1.5 and 3.3 volts
- PENC - controls 5 and 12 volts for the SCSIBox housing the system disk (rackmount systems only)
- PEND - controls 5 and 12 volts for the optional, second SCSIBox (rackmount systems only)
- PENE - controls 5 and 12 volts for any external cabinets

Each time a signal is asserted, a corresponding power-OK (POKx) signal is tested, indicating to the System Controller that the voltage levels are correct. If any voltage-enable line does not generate an OK signal, the System Controller will stop the power-on sequence at the point of the failure. The POKx signals are continually monitored during and after power up. Any POKx signal going low indicates a power-fail condition at one or more of the system power supplies/regulators. A low POKx signal causes the System Controller to display a "Power Fault" message and begin the system power-down sequence. Figure 2-9 illustrates the relationship of the POKx signals to the various system voltages and components. The system power-on sequence is illustrated by the flowchart in Figure 2-10.

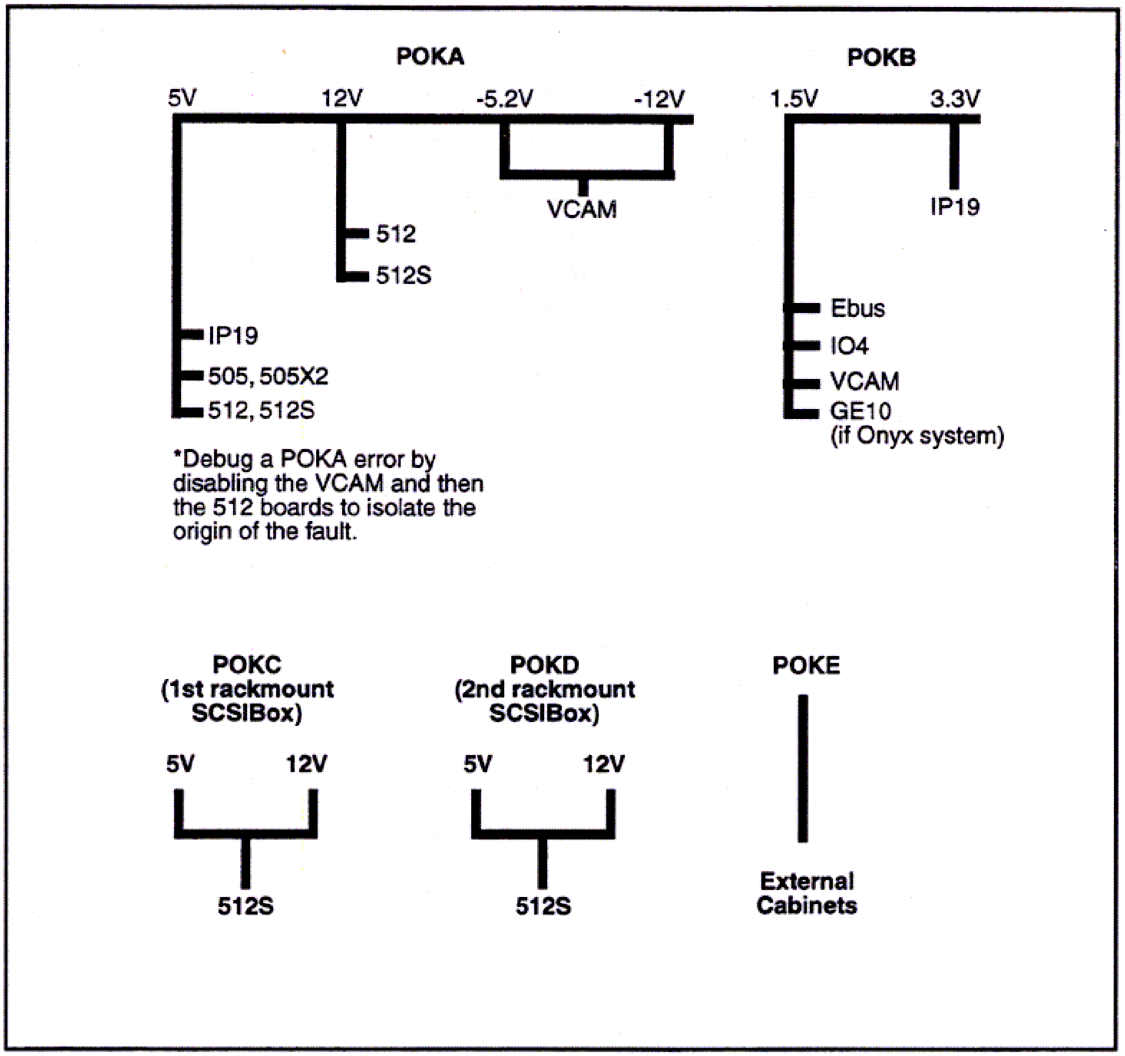


Figure 2-9 Power OK (POKx) Signals

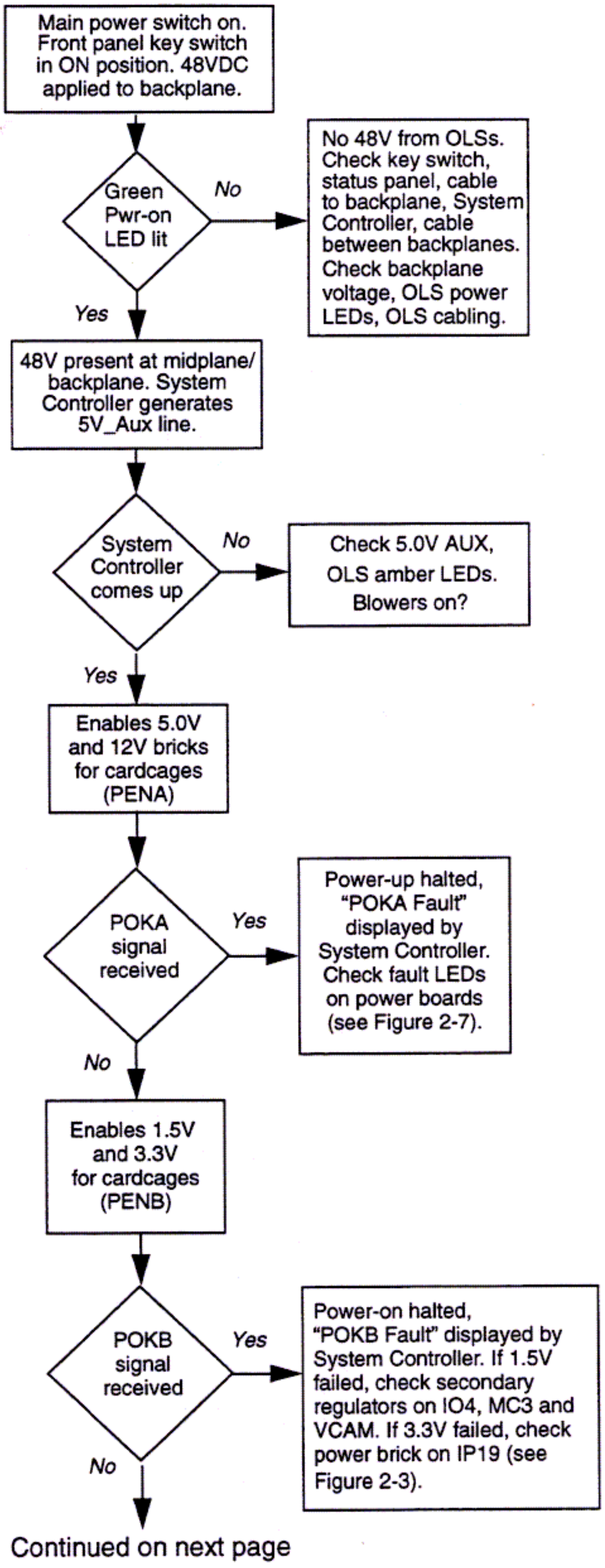

Main power switch on. Front panel key switch in ON position. 48VDC applied to backplane.
Green Pwr-on LED lit
No
No 48V from OLSs. Check key switch, status panel, cable to backplane, System Controller, cable between backplanes. Check backplane voltage, OLS power LEDs, OLS cabling.
Yes
48V present at midplane/ backplane. System Controller generates 5V_Aux line.
System Controller comes up
No
Check 5.0V AUX, OLS amber LEDs. Blowers on?
Yes
Enables 5.0V and 12V bricks for cardcages (PENA)
POKA signal received
Yes
Power-up halted, "POKA Fault" displayed by System Controller. Check fault LEDs on power boards (see Figure 2-7).
No
Enables 1.5V and 3.3V for cardcages (PENB)
POKB signal received
Yes
Power-on halted, "POKB Fault" displayed by System Controller. If 1.5V failed, check secondary regulators on IO4, MC3 and VCAM. If 3.3V failed, check power brick on IP19 (see Figure 2-3).
No
Continued on next page

Figure 2-10 Power-on Sequence

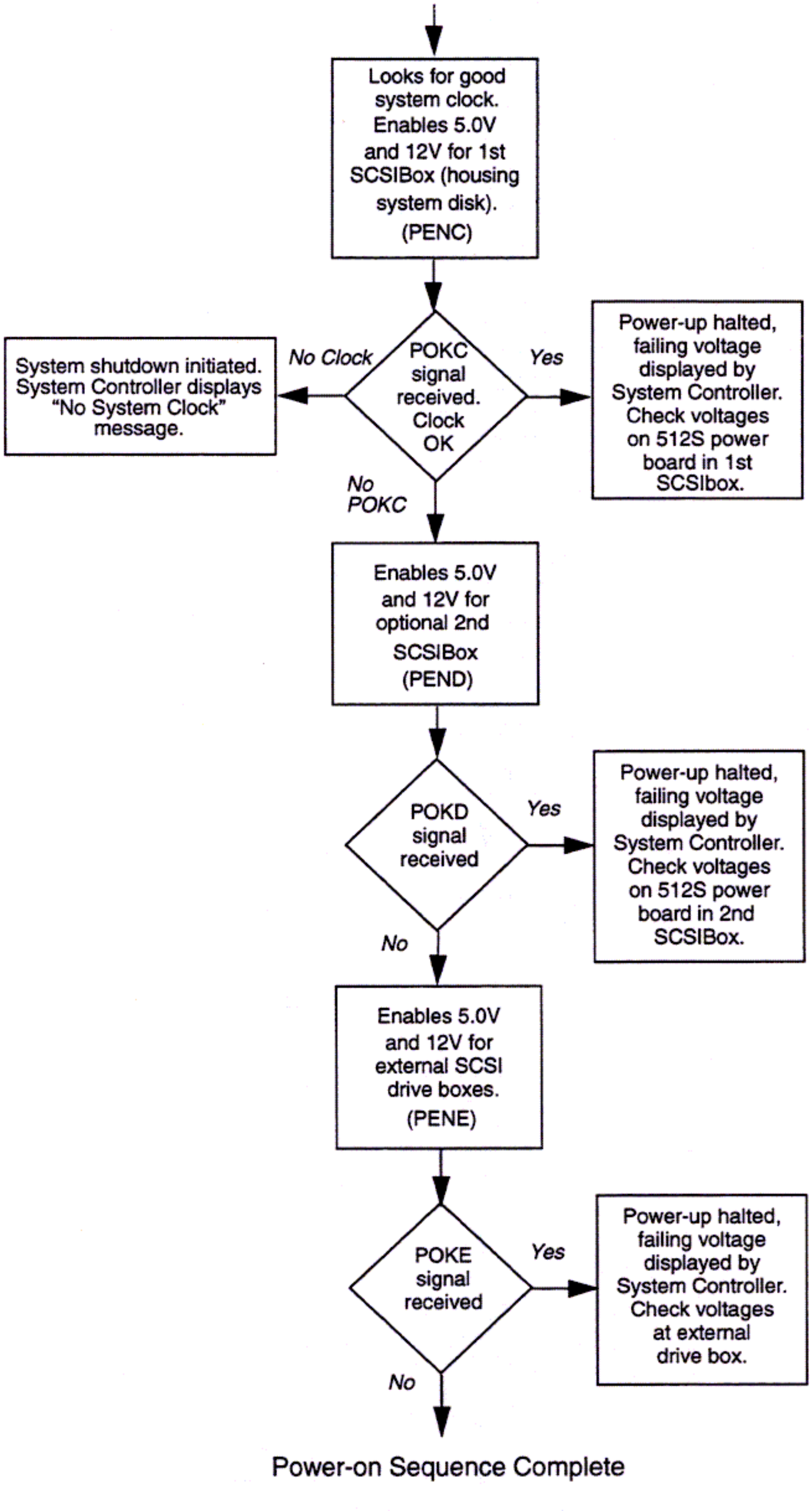

Looks for good system clock. Enables 5.0V and 12V for 1st SCSIBox (housing system disk). (PENC)
POKC signal received. Clock OK
No Clock
System shutdown initiated. System Controller displays "No System Clock" message.
Yes
Power-up halted, failing voltage displayed by System Controller. Check voltages on 512S power board in 1st SCSIbox.
No POKC
Enables 5.0V and 12V for optional 2nd SCSIBox (PEND)
POKD signal received
Yes
Power-up halted, failing voltage displayed by System Controller. Check voltages on 512S power board in 2nd SCSIBox.
No
Enables 5.0V and 12V for external SCSI drive boxes. (PENE)
POKE signal received
Yes
Power-up halted, failing voltage displayed by System Controller. Check voltages at external drive box.
No
Power-on Sequence Complete

Figure 2-10 (continued) Power-on Sequence

When the system has successfully powered up, the System Controller deasserts the power clear (PCLR) and system clear (SCLR) signals to the address ASIC on each IP19 board. The cache controllers then reset each of the system's processors and the power-on tests are started (see Chapter 4).

See Figure 2-11 for the power-enable/power-ok signal timing.

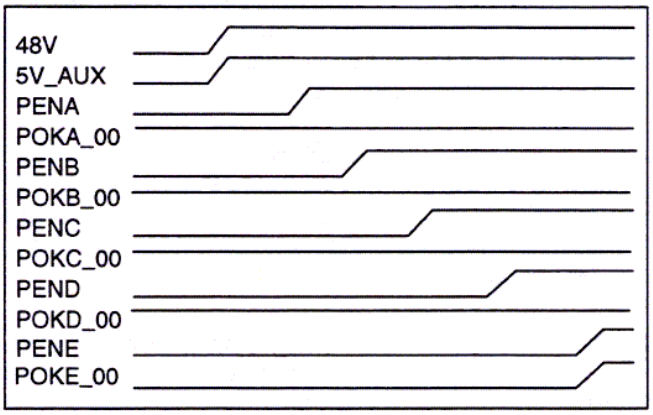


Figure 2-11 Power-on Signal Timing

Note: The POKx signals (with the exception of POKE) remain high unless there is a fault. The System Controller checks for a low to indicate a problem.

2.2.1 Isolating Errors

With the key switch in the Manager position, the voltage status menu can be called up on the System Controller display. This menu displays the actual voltages at the midplane/backplane, at the power boards, and at the VCAM. The voltage status menu is shown in Figure 2-12.

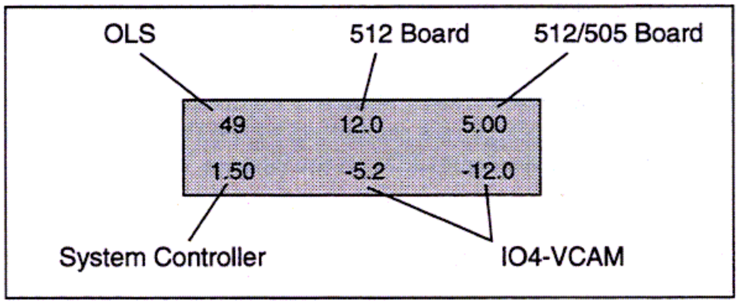


Figure 2-12 System Controller Voltage Status Menu

If an out-of-range voltage level is sensed on the 505 or 512 power boards, the System Controller will display a message indicating which voltage is out-of-range and whether it is high or low. The display will indicate which voltage level failed, but cannot isolate the failure to an individual power board because same-level voltages are ganged together.

The power-OK (POK) signals also cannot isolate a voltage fault to a specific component. The voltages at each power brick are monitored, but the power-OK lines (POKA, B, C...) are OR-tied together, so a failure sensed by any of the POK lines indicates the failing voltage but cannot isolate the cause. Also, in systems with more than one of each type of power board, identical voltages are ganged together. Finally, there are secondary regulators; one on the backplane, two on the IO4 board, one on the MC3 board, and one on each of the FMezz boards, whose output voltages are not sensed.

A voltage fault is isolated by inspecting the fault LEDs on all of the suspect boards before powering down or resetting the system (refer to the tables in the following section for the LED error codes). Check the system for a lit POKx LED when either a voltage fault message (e.g. 5V low fault) or a POK error message (e.g. POKB bad) is displayed. The points where the voltages are monitored are shown in Figure 2-13.

If a CPU, blower, voltage regulator, or power brick fails, the System Controller will disable the bricks but will leave 48 volts at the midplane/backplane and the V5_AUX on. V5_AUX allows the fault LEDs to remain lit and provides power to the front panel display and System Controller. If a shutdown has occurred, check the error message that is displayed and visually inspect the corresponding fault LEDs throughout the system to isolate the fault. See Section 2.2 for the locations of the fault LEDs and Section 3.3 for the System Controller error messages.

Note: Check the front panel display for error messages and inspect the cardcage(s) for lit fault LEDs before restarting the system. Repeated power cycling during fault diagnosis will eventually fill the System Controller event history log and overwrite the original error message. The bootmaster CPU can save the contents of the System Controller history log in */usr/adm/SYSLOG* only after the boot process is complete. If the fault prevents the system from booting, there will be no record of the fault in UNIX.

If an over-temperature fault occurs, the entire system shuts down. Isolate the fault by restarting the system with the key switch and checking the error message on the front panel display. If the temperature sensors are not given sufficient time to cool below the trip point, the system will continue to shut down. Temperature sensors are located on the IP19, MC3 and IO4 boards.

Note: The Voltage Status menu, shown in Figure 2-12, only applies to Cardcages 1 and 2. Cardcage 3 voltages are monitored by the POK lines and voltage faults are indicated by the fault LEDs on the Remote VCAM.

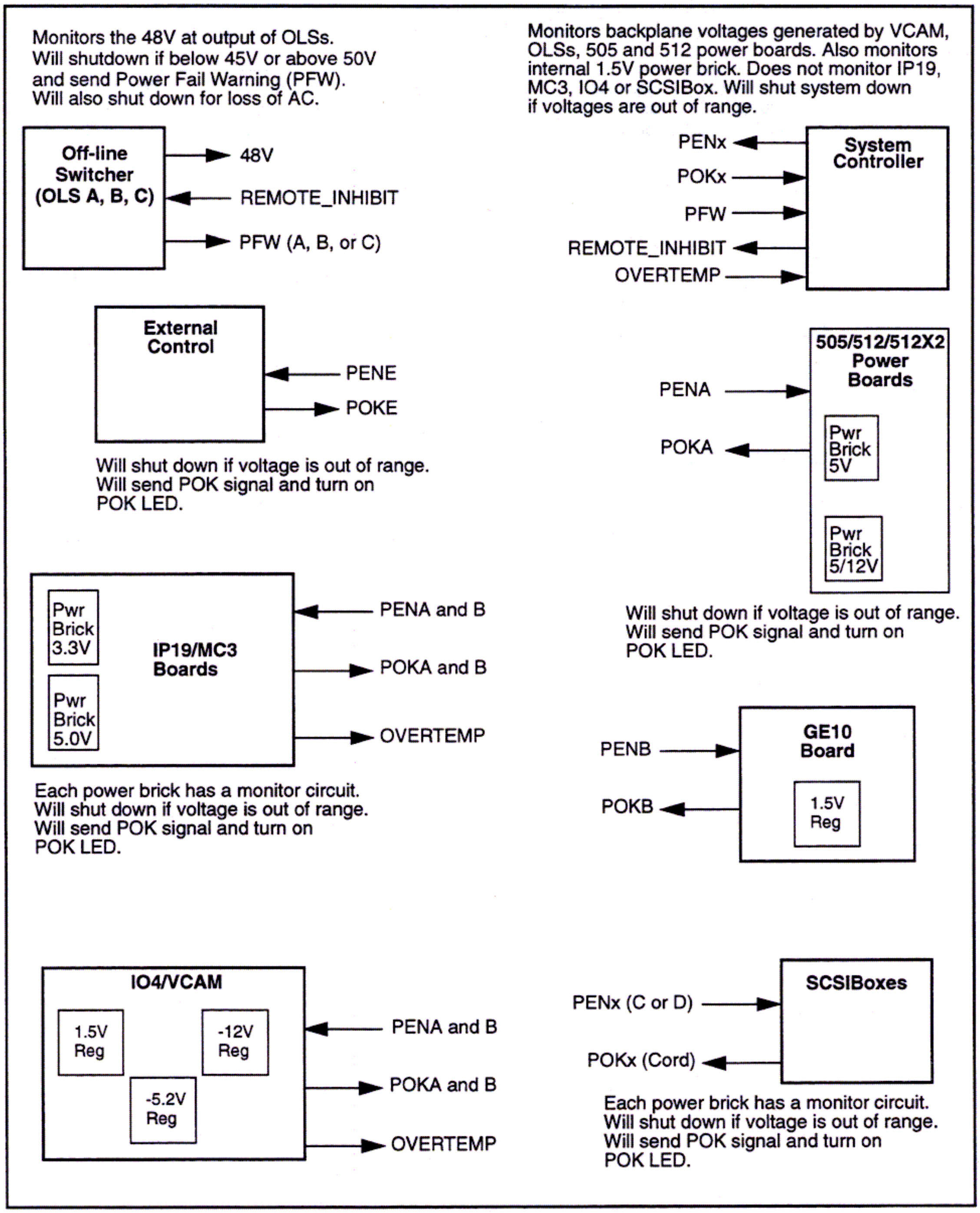

Monitors the 48V at output of OLSs.
Will shutdown if below 45V or above 50V
and send Power Fail Warning (PFW).
Will also shut down for loss of AC.
Off-line Switcher (OLS A, B, C)
48V
REMOTE_INHIBIT
PFW (A, B, or C)
Monitors backplane voltages generated by VCAM, OLSs, 505 and 512 power boards. Also monitors internal 1.5V power brick. Does not monitor IP19, MC3, IO4 or SCSIBox. Will shut system down if voltages are out of range.
System Controller
PENx
POKx
PFW
REMOTE_INHIBIT
OVERTEMP
External Control
PENE
POKE
Will shut down if voltage is out of range.
Will send POK signal and turn on
POK LED.
505/512/512X2 Power Boards
PENA
POKA
Pwr Brick 5V
Pwr Brick 5/12V
Will shut down if voltage is out of range.
Will send POK signal and turn on
POK LED.
Pwr Brick 3.3V
Pwr Brick 5.0V
IP19/MC3 Boards
PENA and B
POKA and B
OVERTEMP
Each power brick has a monitor circuit.
Will shut down if voltage is out of range.
Will send POK signal and turn on
POK LED.
GE10 Board
PENB
POKB
1.5V Reg
IO4/VCAM
1.5V Reg
-12V Reg
-5.2V Reg
PENA and B
POKA and B
OVERTEMP
SCSIBoxes
PENx (C or D)
POKx (Cord)
Each power brick has a monitor circuit.
Will shut down if voltage is out of range.
Will send POK signal and turn on
POK LED.

Figure 2-13 Power Subsystem Voltage Monitoring

Table 2-3 provides the voltage ranges monitored by the System Controller, and two sets of voltage thresholds: the upper and lower thresholds at which a voltage warning is issued, and the upper and lower thresholds at which the system is shut down.

Warning (below this voltage a warning is issued)	Maximum Undervoltage (below this voltage system shuts down)	Nominal	Maximum Overvoltage (above this voltage system shuts down)	Warning (above this voltage a warning is issued)
-----------	45V	48V	50V	-----------
10.97V	10.2V	+12V	13.8V	13.02V
4.59V	4.35V	+5V	5.65V	5.46V
1.23V	1.05V	+1.5V	1.95V	1.77V
-4.4V	-3.63V	-5.2V	-5.85V	-5.66V
-10.0V	-8.45V	-12V	-14V	-13.7V

Table 2-3 Voltage Ranges and Warning Thresholds

Note: The POK signals are asserted at the under-voltage limits. If a specific test circuit fails to assert POK, the System Controller initiates a system shutdown.

Chapter 3

System Controller

3.1 Overview

The Everest System Controller is a microprocessor with battery-backed clock and RAM. The System Controller performs three basic functions:

- The System Controller manages the system's power-up, power-down, and bootmaster arbitration processes. It also displays a running account of the status of the boot procedure and notifies the bootmaster CPU when a system event, such as power off, is initiated.

- When operating conditions are within normal limits, the System Controller is a passive monitor. The only active role the System Controller plays is in monitoring the cabinet temperature and adjusting the blower speed. Its front panel LCD offers a running CPU activity graph that shows the level of each processor's activity. Previously logged errors are not available on the front panel display, but are transferred into */usr/adm/SYSLOG*.

- The System Controller can also act independently to shut down the system when it detects a threatening condition. Or it can adjust electro mechanical parameters (such as blower speed) to compensate for external change. The Manager position, on the key switch, provides menus used to probe for system error information.

This chapter describes the operation of the System Controller during the power up, power down, and boot sequences, as well as during both normal system operation and during an emergency shutdown. Explanations of the possible error messages are presented in Section 3.3.

Figure 3-1 illustrates the system components monitored and/or controlled by the System Controller.

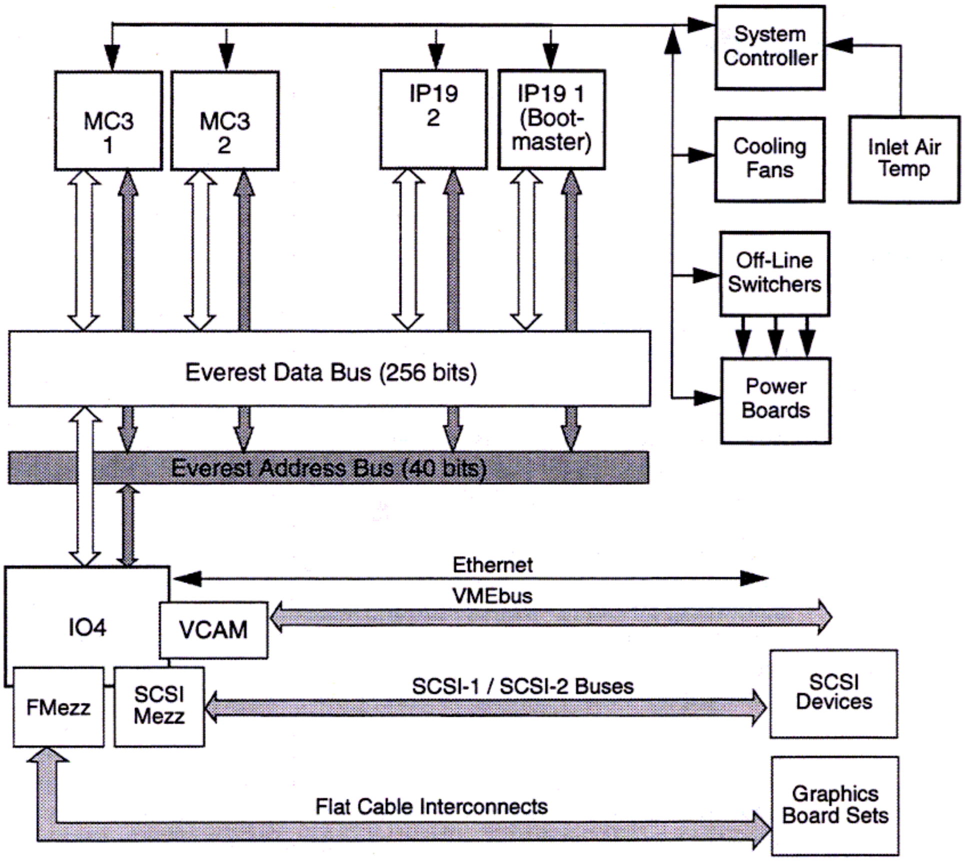


Figure 3-1 System Components Monitored/Controlled by the System Controller

3.2 Basic Functions

This section provides a step-by-step description of the System Controller operation. See Figure 3-2 for a block diagram illustrating the signals received by and transmitted from the System Controller.

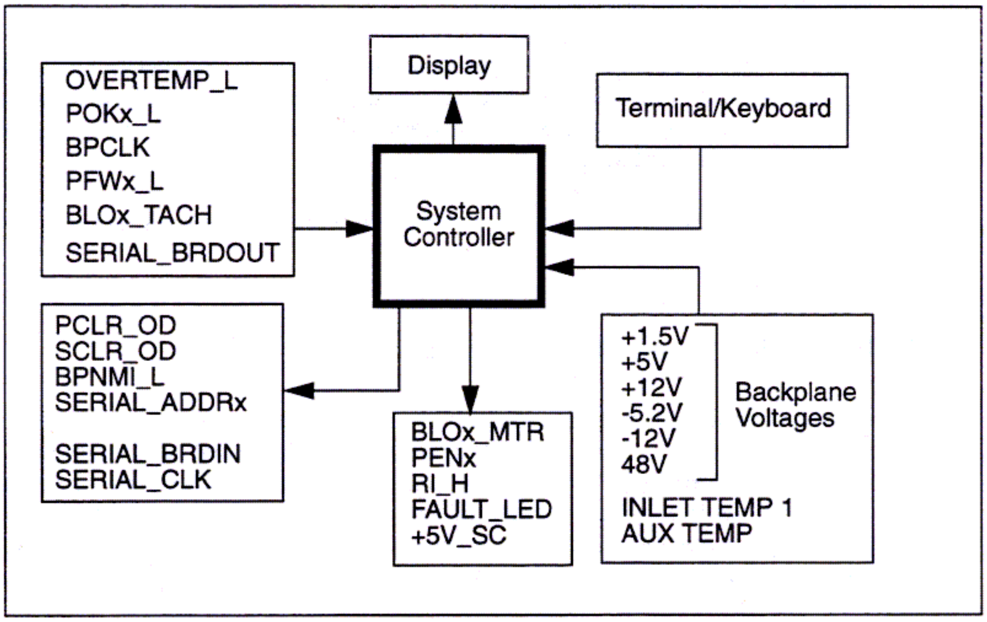


Figure 3-2 System Controller Input Signals

3.2.1 Power-up, Boot, and Reset Sequences

The System Controller plays an active role in the power-up, boot and reset processes. The power-up process begins when the System Controller enables the OLS outputs, supplying 48 volts to the midplane/backplane. Next, the blower(s) are turned on and their speed monitored. Then the System Controller sequentially turns on a series of power-enable lines (PENA - PENE). As each system component is brought up, the System Controller waits for a valid power-OK signal (POKA - POKE), which indicates that the voltages just enabled are within the specified range. If the power-OK signal remains high, the System Controller asserts the next power-enable line in the series. If a power-OK signal is bad (goes low), the System Controller will halt the power-up sequence. When the power-up sequence is complete, the System Controller sends a power-clear (PCLR) signal to the CPU boards (see Chapter 2 for additional information).

Note: If a power-OK or blower fault is sensed during the power-up, the system will shut down completely.

The PCLR/SCLR signals cause all of the system's processors to reset, beginning the first step in the bootmaster arbitration process (see Chapter 4). The System Controller then polls each of the CPU boards over the Polled Serial bus. The first CPU polled is the board with the lowest address. If that CPU has successfully passed its self-test, it notifies the System Controller that it is becoming the bootmaster, and sends interrupts to any other CPUs. If the first CPU failed its self-test, the System Controller will increment the CPU address by one and offer the bootmaster role to the next CPU. The first CPU board to successfully complete its self-test and respond to the poll becomes the bootmaster. The bootmaster CPU takes control of the boot process and uses the serial link to the System Controller to transmit status and error messages.

When the operator requests a reset using the front panel, the System Controller asserts the SCLR line, as it does following the power-up sequence. The processors are reset and the boot arbitration and power-on test processes start over, however, power is never removed from the midplane/backplane.

Note: Before requesting a system reset, terminate all processes and run an *init 0* to gracefully halt UNIX.

3.2.2 Monitoring Normal System Operation

During normal system operation, the System Controller passively monitors the system voltages, backplane clock, the air temperature in the cabinet, and the blower speed. The Power Fault Warning (PFW) signals, from the off-line switchers, are also monitored in order to allow the system to gracefully power-down in the event of an impending loss of power. Status messages from the bootmaster CPU are transmitted to the System Controller and are available at the controller's display.

The System Controller will issue and display a warning if an abnormal condition is detected but does not warrant a system shutdown. This condition can be detected by either the System Controller's sensors or by the bootmaster CPU. In these cases, the warning is issued only to inform the user.

3.2.3 Initiating a System Power-off

If a condition is detected that calls for a system shutdown, the System Controller issues an alarm. If the situation is not immediately dangerous, the System Controller will wait until it receives a "Set System Off" message or until its internal timer counts down. This delay in the shutdown sequence is designed to give UNIX ample time to perform an orderly software shutdown and to sync the system disks before power is removed.

If the reason for the shutdown requires immediate action, such as an out-of-spec voltage or a voltage failure POK condition, the System Controller will log a message and shut down immediately. In these cases, the power subsystem is shut down gracefully, but the system does not have time to sync disks or to halt UNIX.

Following the alarm, the System Controller removes power from the system boards and peripherals without turning off the 48 volts from the OLSs (all of the PEN lines go low). The System Controller will display a fault message and the fault LED next to the display will light.

Note: First, check the front panel display for error messages. Then, inspect the corresponding Fault LEDs to localize the problem. The appropriate fault LEDs will remain lit after the system has shut down. Turning off the system power or rebooting will reset the fault LEDs only if the fault has been corrected.

Restore power by turning the key switch OFF for 30 seconds, and then ON (turning the key switch OFF will clear any fault LEDs that are lit). The System Controller will begin the start-up sequence. If the fault still exists, the system will shut down again and repeat the previous fault message.

3.2.3.1 Over-Temperature Faults

If the System Controller shuts the system down because the temperature sensors on the IP19, MC3 or IO4 boards are too high, power is removed from all system components, including the System Controller itself. In order to determine the origin of the fault, cycle the key switch Off and then On and check the displayed error message. If the system immediately shuts down again, wait for several minutes to allow the mechanical temperature sensor switch to cool below its trip point.

3.3 Error Messages

Tables 3-1 through 3-4 describe each of the messages displayed by the System Controller. These messages are divided into five categories: power-up and system errors, system events, internal System Controller errors, and bootmaster arbitration errors.

Error Message	**Failure Area/Possible Solution**
EBUS TEST 2 FAILED!	This comprehensive test of the Ebus indicates probable failure of the IP19 or a fault on the system Midplane/Backplane.
PD CACHE FAILED!	The primary data cache on the bootmaster microprocessor has failed.
NO IO4 FOUND!	No IO4 is seen during the system probe. Check for bent pins, re-seat the IO4, or replace the IO4. A backplane problem is also possible.
NO IO4 UART FOUND!	IO4 probably bad.
MASTER IO4 FAILED!	IO4 probably bad.
IO4 UART FAILED!	IO4 probably bad.
INIT INV FAILED!	IO4 probably bad.
NO MC3 FOUND!	No MC3 was found during the system probe. Check for bent pins, re-seat the MC3, or replace it. A backplane problem is also possible.

Table 3-1 Power-on Test Errors and Fault Identification

Error Message	Failure Area/Possible Solution
MC3 CONFIG FAILED!	The system MC3 has failed. If there is more than one MC3 present, a potential Ebus or IP19 board problem (if system uses a single CPU). Check system voltages with the System Controller. If voltages are out of range, swap out power board or OLS.
BUS TAGS FAILED!	There is a problem with the IP19.
SCACHE FAILED!	The secondary cache on the bootmaster microprocessor failed. The SCACHE SIMM module is bad or the IP19 is faulty.
PROM DNLOAD FAILED!	The IO4 or MC3 path is blocked. A possible fault exists on the IO4's flash PROM. Check for bent pins and re-seat the IO4(s) and MC3(s).
GRAPHICS FAILED!	The graphics self tests failed. Check individual graphics boards with console/laptop.
CONSOLE FAILED!	Check the console terminal configuration and cabling. Check the IO4 to IO panel cable connection. There may be a fault in the IO4.

Table 3-1 (continued) Power-on Test Errors and Fault Identification

Note: The first 12 error messages in the table are generated by the CPU's boot PROM. The last two messages are generated by the IO4's boot PROM.

The area and possible solution for each error message in the previous table is general by default. To obtain more specific information, you can follow three possible paths:

- Swap out the suspected faulty FRU and re-power the system.
- Plug your laptop into the system console port (Port 1) and probe for more specific fault information. Note that if the fault lies in the IO4 or IO4 pathway, you may be unable to access the system console port
- Plug your laptop into the System Controller port, labeled **`External Controller Serial`**, using the cable permanently attached to the port. On rack-mounted systems, this port is located in the lower left corner of the midplane (when facing the front of the chassis). Deskside systems have the port located in the lower right corner of the backplane (when facing the rear of the chassis).

Error Message	Error Meaning
SCLR DETECTED	The System Controller detected an SCLR on the system backplane. The reset was initiated from the System Controller front panel by an operator.
SYSTEM OFF	The key switch was turned to the off position and the System controller powered down the system.
SYSTEM ON	The System Controller has successfully powered up the system.
SYSTEM RESET	The System Controller detected an SCLR and initiated a system boot arbitration process. SCLR can be generated by any processor board or by the System Controller.
NMI	The System Controller placed a non--maskable interrupt (NMI) onto the system backplane from a front panel menu option. UNIX wrote out a core dump and then restarted. The results of initiating an NMI are unpredictable and should be only used as a last resort. Note that if the system debugger (SYMMON) is installed, selecting NMI will force the operating system to return to SYMMON's "DBG" prompt (useful if the machine hangs).

Table 3-2 System Event Messages

Error Message	Error Meaning
ACIA FRAMING ERROR	The secondary System Controller serial port detected a framing error.
ACIA NOISE ERROR	The secondary System Controller serial port detected a noise error.
ACIA OVERRUN ERROR	The secondary System Controller serial port detected an overrun error.
BAD MSG: CPU PROCESS	The CPU or System Controller process has received an invalid message.
BAD MSG: DISPLAY	The display process has received an invalid message.

Table 3-3 Internal System Controller Error Messages

Error Message	Error Meaning
BAD MSG: POK CHK	The power ok check process received an invalid message.
BAD MSG: SEQUENCER	The sequencer process has received an invalid message.
BAD MSG: SYS MON	The system monitor process has received an invalid message.
COP FAILURE	The Computer Operating Properly (COP) timer has exceeded time limits. The System Controller firmware must write to a COP timer port before it times out. If the firmware exceeds the time allowed between writes to a COP port, an interrupt is generated. The System Controller firmware may have entered an endless loop.
COP MONITOR FAILURE	A Computer Operating Properly (COP) clock monitor failure was detected. The System Controller clock oscillator is operating at less than 10K Hz.
FP CONTROLLER FAULT	An error was detected in the front panel LCD display control process.
ILLEGAL OPCODE TRAP	The System Controller's microprocessor tried to execute an illegal instruction. Probably because of a stack overrun followed by a process switch.
MEMORY FAILURE	The System Controller's internal memory experienced a failure.
PULSE ACCU INPUT	An interrupt was detected on the pulse accumulator input port. The port is not used and an interrupt is considered an error.
PULSE ACCU OVERFLOW	The pulse accumulator overflow port received an interrupt. This port is unused and the interrupt is considered an error.
SCI FRAMING ERROR	The System Controller's internal microprocessor has detected a framing error on its serial port.
SCI NOISE ERROR	The System Controller's internal microprocessor has detected a noise error on its serial port.
SCI OVERRUN ERROR	The System Controller's internal microprocessor has detected an overrun error on its serial port.

Table 3-3 (continued) Internal System Controller Error Messages

Error Message	Error Meaning
SOFTWARE INTERRUPT	A software generated interrupt was detected. This function is not supported and the interrupt is considered an error.
SPI TRANSFER	An interrupt was detected on the synchronous serial peripheral Interface. This interface is not supported and the interrupt is an error.
STACK FAULT PID 0–6	One of the seven stack areas used by a System Controller process has overflowed its assigned boundaries
TIMER IN COMP 1	The timer input compare port received an interrupt. The port is not used and the interrupt is considered an error.
TIMER OUT COMP 1–5	One of the five timer output compare ports received an interrupt. The port is not supported and the interrupt is considered an error.
TIMER OVERFLOW	A timer overflow port interrupt occurred. This port is not used and the interrupt is considered an error.

Table 3-3 (continued) Internal System Controller Error Messages

Note: Internal errors will cause an error message to be displayed, but will not shut down the system.

Master CPU Selection Message	Context and Meaning of Message
BOOT ARBITRATION NOT STARTED	The system CPU board(s) have not begun the arbitration process.
BOOT ARBITRATION IN PROCESS	The System Controller is searching for the bootmaster CPU.
ARBITRATION COMPLETE BOARD OxZZ PROC OxZZ	The chosen bootmaster CPU has identified itself to the System Controller and communication is fully established.

Table 3-4 System Controller Master CPU Selection Status Messages

Master CPU Selection Message	Context and Meaning of Message
BOOT ARBITRATION INCOMPLETE NO MASTER	An error has occurred in the boot process and no boot master CPU is communicating with the System Controller. Either no CPU is running or the System Controller is faulty. If the System Controller has failed, the system will boot normally, but the histograph will not display.

Table 3-4 (continued) System Controller Master CPU Selection Status Messages

3.4 Sensor Locations

The locations of the System Controller sensors for both the deskside and rack-mounted systems are shown in Figures 3-3 and 3-4, respectively.

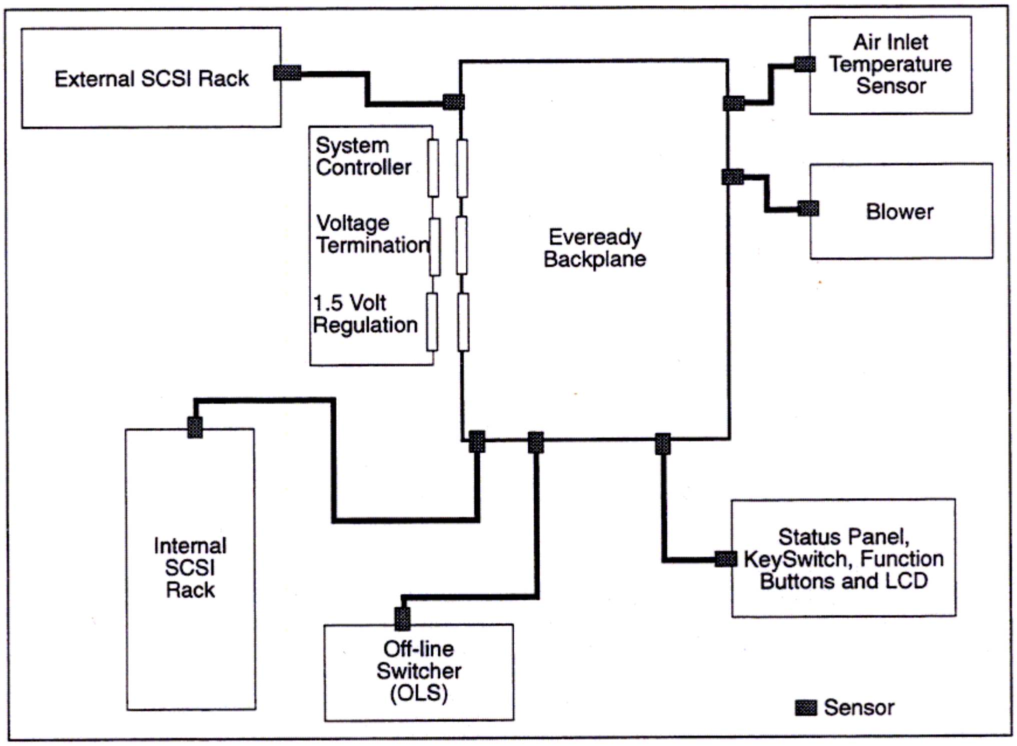


Figure 3-3 Deskside System Controller Sensors

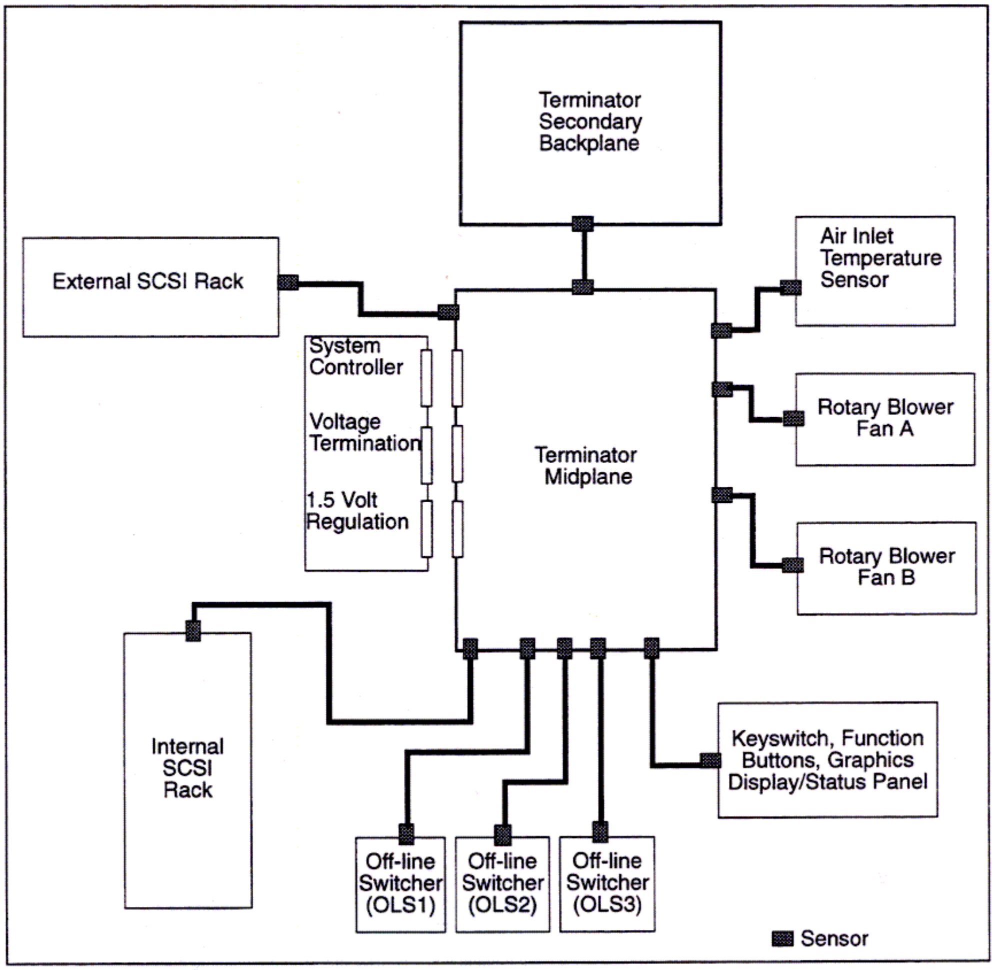


Figure 3-4 Rackmount System Controller Sensors

3.5 Menu Hierarchy

This section provides a sequential listing of the available System Controller menus, as well as descriptions of the menu functions.

The menus are accessed and their functions executed using four function buttons. Press the Scroll Up and Scroll Down buttons to locate a specific menu. Press the Menu button to view the selected menu. Press the Execute button to perform the menu function. See Figure 3-5 for an illustration of the System Controller display and function buttons.

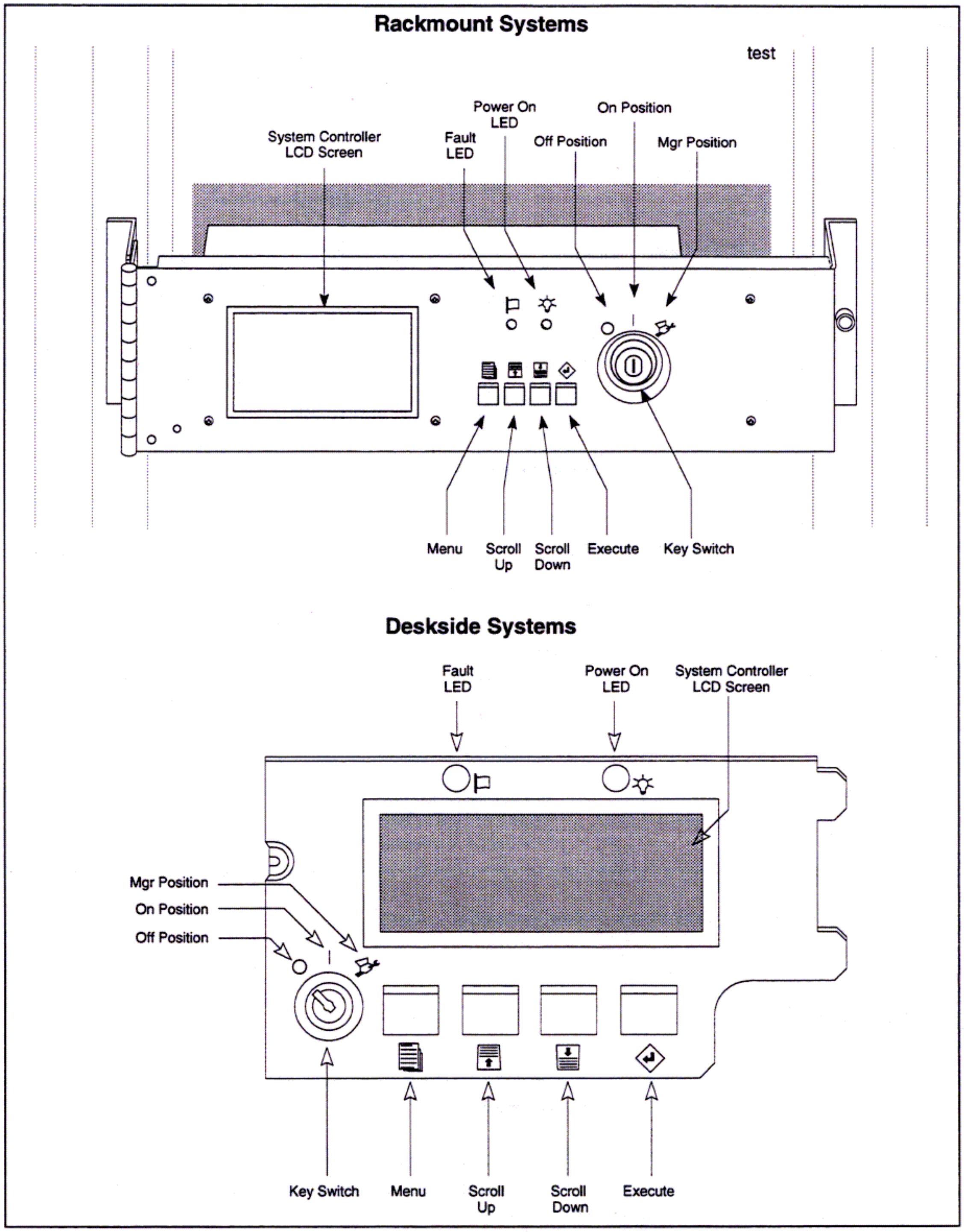
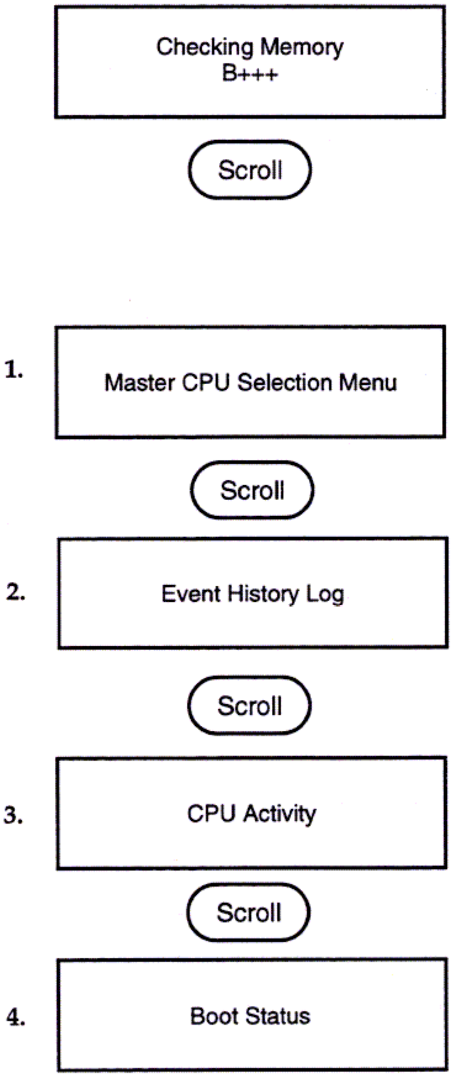


Figure 3-5 System Controller (Deskside and Rackmount Versions)

3.5.1 Key Switch in the On Position

There are four menus that are accessible when the key switch is in the On position.



This message is displayed following a successful boot. The second line indicates that the first of the four installed processors is the bootmaster (B) and that the remaining processors (+) are on-line. A blank space indicates that the processor is missing or dead, an "x" indicates the processor failed diagnostics, and a "d" means that the processor has been disabled. **Note:** This message cannot be used to identify the location of a bad processor. The symbols representing the processors do not correspond to physical locations.

Pressing the `Execute` button displays the last boot arbitration message sent out by the bootmaster CPU. A typical message is:

```
Arbitration Complete
Slot xx Proc xx
```

Pressing the `Execute` button displays the last status or error message. Repeatedly pressing either `Scroll` button will step through the contents of the history log. A maximum of 10 messages are stored. The 11th message causes the first to be discarded. The most recent message is at the end of the log.

Pressing the `Execute` button displays the processor activity as a histogram.

Pressing the `Execute` button displays the last boot arbitration message sent out by the bootmaster CPU.

Continuing to pressing the Scroll buttons will loop through the four menus. When the function buttons are not used, the display will default to the CPU activity histogram.

3.5.2 Key Switch in the Manager Position

In addition to the four menus just described, the Manager position provides access to eight more menus.

4. Boot Status

Scroll

5. Turn External Cabinet Off/On

Pressing the `Execute` button toggles between turning the external cabinet on and off. Each time the button is pressed, the menu prompt toggles between "Off" and "On."

Scroll

6. Turn Internal SCSI B Off/On

Pressing the `Execute` button toggles between turning the optional second SCSIBox on and off. Each time the button is pressed, the menu prompt toggles between "Off" and "On."

Scroll

7. Turn Internal SCSI A Off/On

Pressing the `Execute` button toggles between turning the standard SCSIBox on and off. Each time the button is pressed, the menu prompt toggles between "Off" and "On."

Scroll

8. Backplane NMI

Selecting this menu displays a second-level confirmation menu. Pressing the `Execute` button performs the interrupt. Pressing the `Menu` button returns you to the system menu.

Scroll

9. Reset System (SCLR)

Selecting this menu displays a second-level confirmation menu. Pressing the `Execute` button performs the reset. Pressing the `Menu` button returns you to the system menu.

Scroll

10. Voltage Status

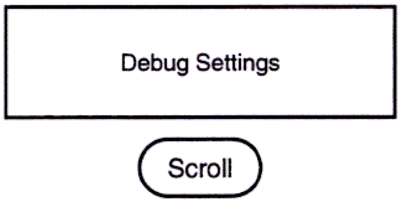
Pressing the `Execute` button displays the voltage menu (refer to Section 2.2.1).

Scroll

11. Temperature Status

Pressing the `Execute` button displays the blower speed in RPMs and the inlet temperature in degrees centigrade.

To access the Debug Settings menu: First turn the key switch to the On position. Press the Scroll Up and Scroll Down buttons simultaneously. Turn the key switch to the Manager position. Press both scroll buttons simultaneously again. Scroll through the menus until the Debug Settings menu appears. If the menu is not selected within 30 seconds, it disappears.

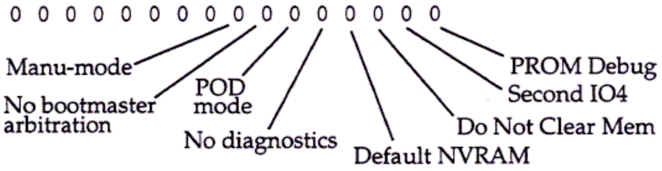


Pressing the `Execute` button displays 16 bits:

```
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The bits are numbered from right to left, starting with "0" and ending with "f."

A flashing cursor appears under the bit at the far right of the display. The `Scroll` buttons move the cursor back and forth across the screen. The `Execute` button toggles the bit. The bits are defined as follows:



The "PROM Debug Mode" switch displays additional debugging information. It also is used to alter the system's serial number (see Section 6.3.3.5).

The "Second IO4" switch causes the PROM to see the IO4 board in the second highest slot as the master IO4.

The "Do Not Clear Memory" switch instructs the PROM not to run BIST on reset. Use to debug after a system crash.

The "Default NVRAM Values" switch forces the PROM to use the default values for various environment variables, instead of reading them from the NVRAM. The default values are:

dbaud	9600	The default console baud rate
rbaud	19200	The alternate port baud rate
fastmem	0	The memory configuration algorithm

The "Manu-mode" switch sends all IP19 PROM console output to the external UART on the System Controller (External Controller Serial).

The "No bootmaster arbitration" switch prevents the System Controller from selecting a processor to communicate with. This allows the individual processors to communicate via the "CC UART" connectors located on the IP19 board edge.

The "POD mode" switch forces the IP19 PROM to stop initialization just before it would have loaded the IO4 PROM and jump to POD mode instead. Useful on a system with a bad IO4 board or PROM.

The "No diagnostics" switch prevents the system from running power-on diagnostics.

Power cycle the system to install the new settings.

Chapter 4

PROM Monitor

4.1 Overview

This Chapter describes the power-on tests, how the Everest boards are configured, and explains the Monitor boot commands.

4.2 Power-on Tests

The power-on tests are initiated when the System Controller sends the SCLR signal, resetting the processors. This series of tests begins with the IP19 logic supporting each individual processor and expands to test and configure the entire system. The power-on test sequence is illustrated in Figure 4-1.

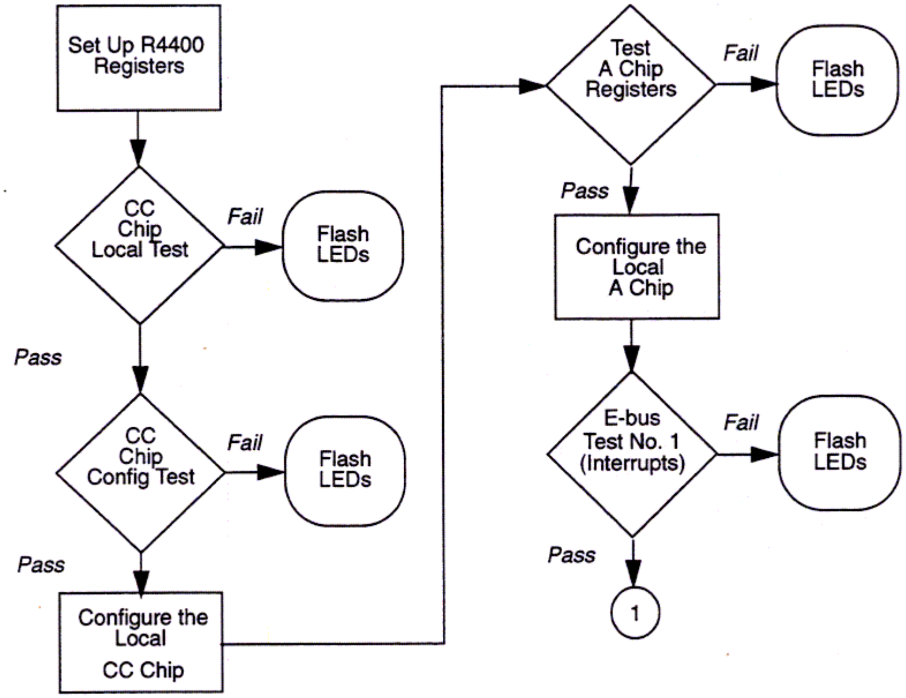


Figure 4-1 Power-on Test Sequence

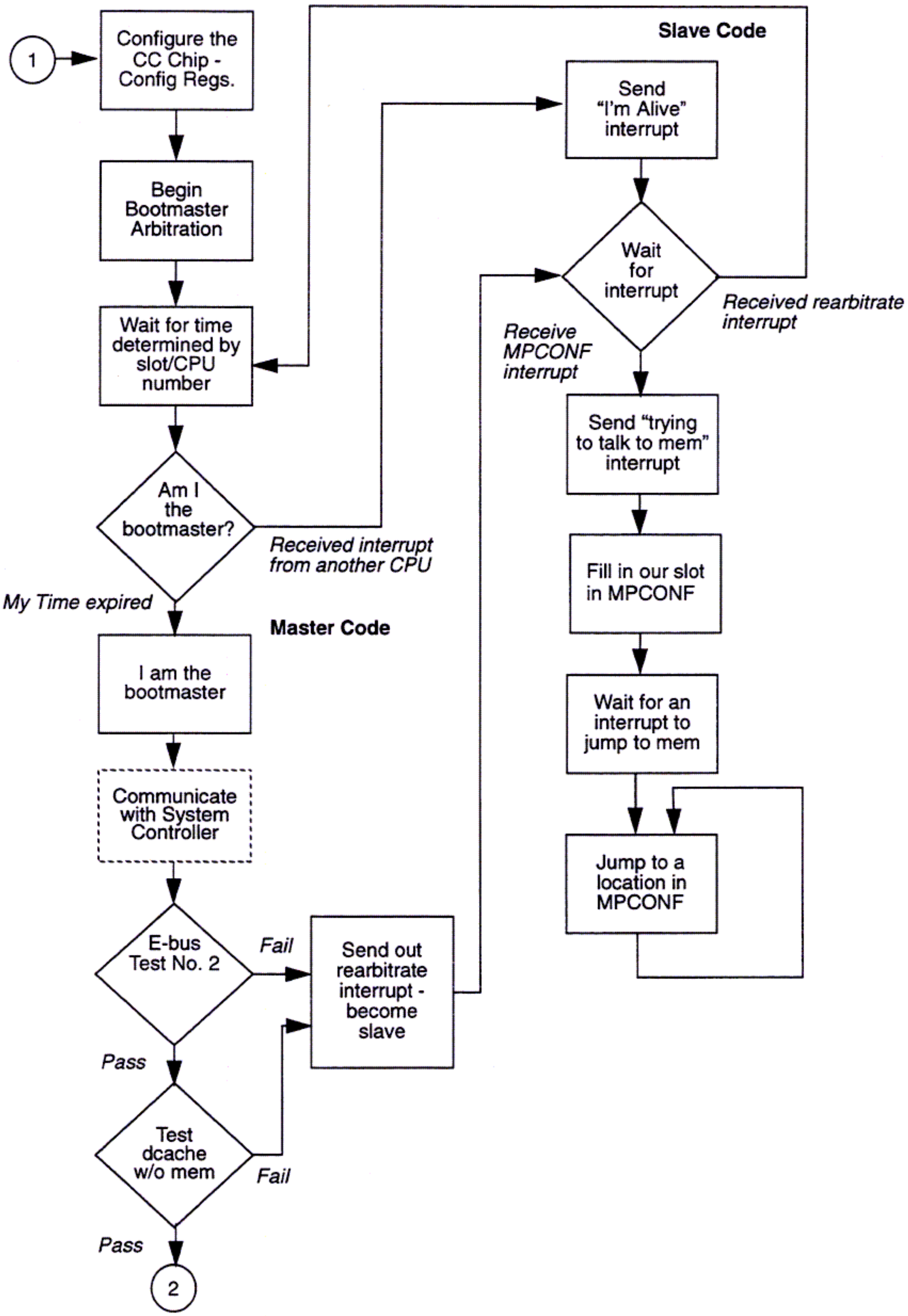
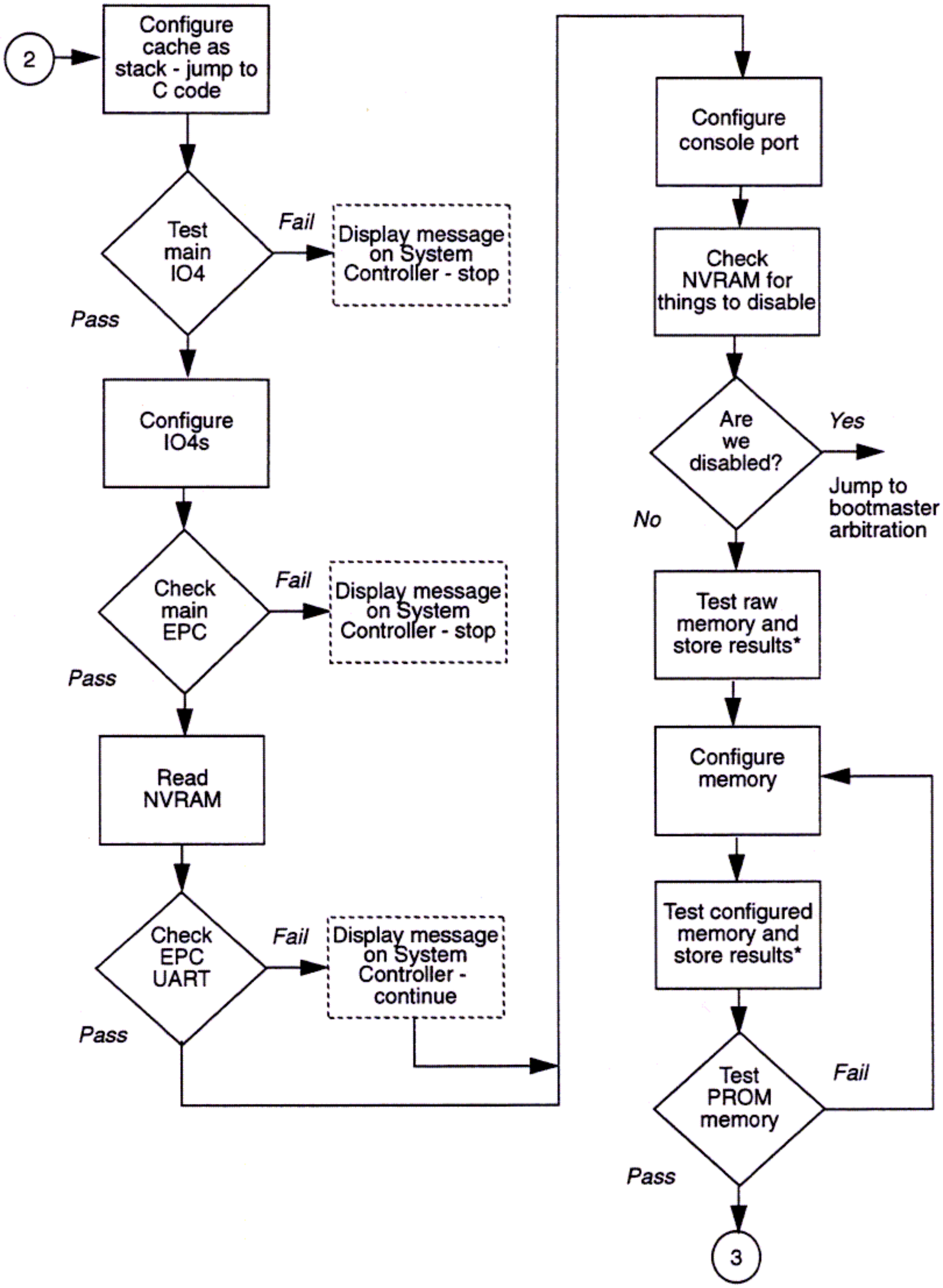

1
Configure the CC Chip - Config Regs.
Begin Bootmaster Arbitration
Wait for time determined by slot/CPU number
Am I the bootmaster?
Received interrupt from another CPU
My Time expired
Master Code
I am the bootmaster
Communicate with System Controller
E-bus Test No. 2
Fail
Pass
Test dcache w/o mem
Fail
Pass
2
Send out rearbitrate interrupt - become slave
Slave Code
Send "I'm Alive" interrupt
Wait for interrupt
Received rearbitrate interrupt
Receive MPCONF interrupt
Send "trying to talk to mem" interrupt
Fill in our slot in MPCONF
Wait for an interrupt to jump to mem
Jump to a location in MPCONF

Figure 4-1 (continued) Power-on Test Sequence


2
Configure cache as stack - jump to C code
Test main IO4
Fail
Display message on System Controller - stop
Pass
Configure IO4s
Check main EPC
Fail
Display message on System Controller - stop
Pass
Read NVRAM
Check EPC UART
Fail
Display message on System Controller - continue
Pass
Configure console port
Check NVRAM for things to disable
Are we disabled?
Yes
Jump to bootmaster arbitration
No
Test raw memory and store results*
Configure memory
Test configured memory and store results*
Test PROM memory
Fail
Pass
3

* If all memory is bad, go into POD mode.

Figure 4-1 (continued) Power-on Test Sequence

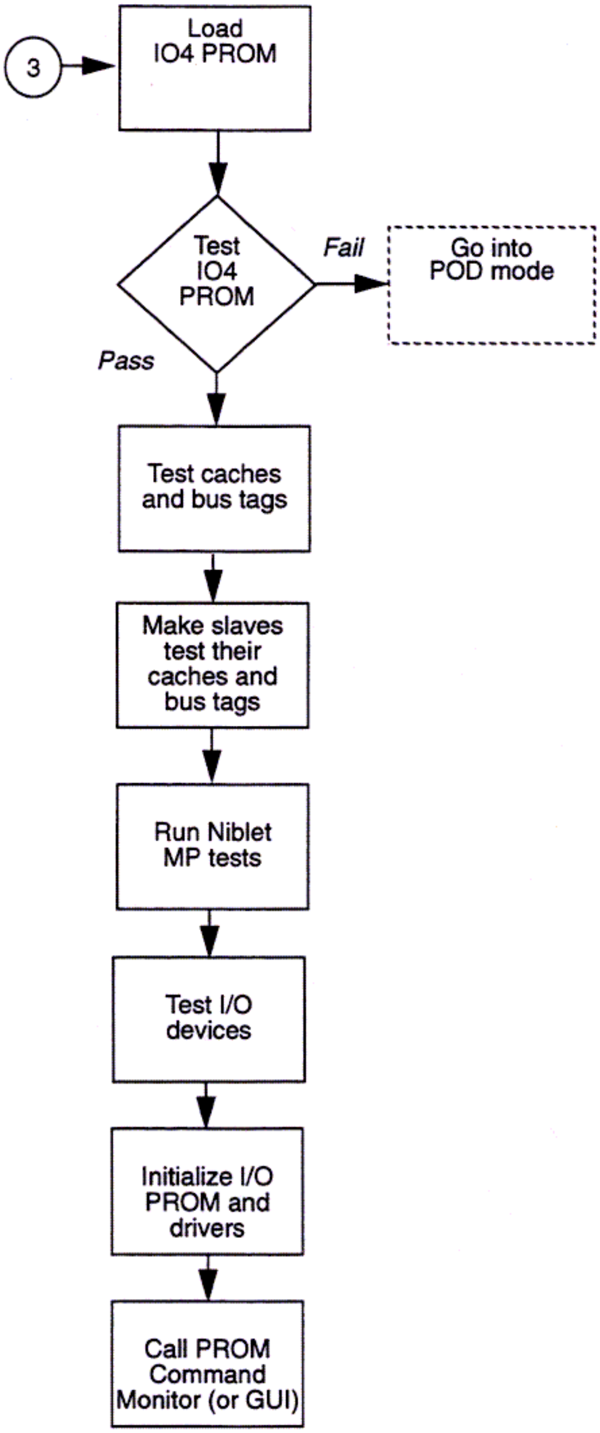


Figure 4-1 (continued) Power-on Test Sequence

4.3 Power-on Test Status Messages

This section is a sequential list of all of the status messages that are displayed by the System Controller during the normal power-on process.

1. Firmware Date
   DD-MM-YY
2. System On
   HH-MM-SS
3. Boot Arbitration in Progress
4. Boot Arbitration Complete
   All of the above messages are issued by the System Controller during initial power-on.
5. Starting System
   Displayed when bootmaster arbitration has completed. Indicates that the bootmaster CPU has started up successfully and can communicate with the System Controller.
6. EBUS Diags 2
   Displayed immediately before the secondary Ebus diagnostics are run (the secondary Ebus diagnostics test the Ebus and the interrupt logic).
7. PD Cache Test
   Displayed immediately before the primary data cache test is run.
8. Building Stack
   Displayed before the system attempts to set the cache up as a stack.

Note: If this is the last message displayed, check for a fault in the bootmaster CPU.

9. Jumping to Main
   Displayed before the system switches into the C main subroutine.
10. Initing Config Info
    Displayed before the initial hardware probing is attempted and before the Everest configuration data structure is set up.
11. Setting Timeouts
    Displayed before attempting to write to the various board timeout registers. Everest requires that all of the boards be initialized with consistent timeout values, and that the values be written before any reads or writes of the boards are attempted.
12. Initing Master IO4
    Displayed before basic initialization of the system IO4 boards is attempted (basic initialization consists of: writing the large and small window registers, setting the endianess, setting up error interrupts, clearing the Ibus and Ebus error registers, and examining the I/O adapters).
13. Initing EPC
    Displayed before the first writes to the master EPC. This routine clears the EPC error registers and takes all EPC devices out of reset.
14. Initializing EPC UART
    Displayed when first entering the UART configuration code.

15. Initing UART Chan B
    Displayed before beginning the initialization of UART Chan B's control registers.
16. Initing UART Chan A
    Displayed before beginning the initialization of UART Chan A's control registers.
17. Reading Inventory
    Displayed before attempting to read the system inventory out of the IO4 NVRAM. If the inventory is invalid or can't be read, the inventory fields are initialized with appropriate default values.
18. Running BIST
    Displayed before running the memory hardware's built-in self test.
19. Configuring Memory
    Displayed before configuring the banks into a legitimate state.
20. Testing Memory (will be added in the next release)
    Displayed before beginning the memory post-configuration tests (verifies that memory was configured correctly).
21. Testing Bus Tags
    Displayed when the CC BUS TAGS are checked and initialized (the CC chip uses the bus tags to determine whether or not it should pass a coherency transaction on to a particular processor).
22. Writing CFGINFO
    Displayed before attempting to write the Everest configuration information to main memory.
23. Initing MPCONF Blk
    Displayed before initializing the MP configuration blocks for all of the processors.
24. Testing S Cache
    Displayed before testing the secondary cache on all of the processors.
25. S Cache Passed
26. Checking Slaves
    Displayed when each slave processor is checked (check determines whether or not processor is operational, and whether or not it passed its diagnostics).
27. Loading IO4 PROM
    Displayed when the IO4 PROM is downloaded from the IO4 flash PROMs into main memory.
28. Entering IO4 PROM (will be added in the next release)
    Displayed when first jumping into the IO4 PROM.
29. Sizing Caches
    Displayed when the cache sizing code is executed.
30. Initing Environment
    Displayed when attempting to read the NVRAM variables from the IO4 PROM NVRAM.
31. Reiniting Caches
    Displayed when rechecking the caches.
32. Initing saio
    Displayed when beginning the initialization of the standalone I/O routines.

33. Initing SCSI (will be added in the next release)
    Displayed when beginning the initialization of the WD95 SCSI driver.

34. Initing UART (will be added in the next release)
    Displayed when initializing the IO4 EPC UART.

35. Initing Graphics (will be added in the next release)
    Displayed when initializing the graphics devices (if installed).

36. Starting Slaves
    Displayed when the slave processors enter the IO4 PROM slave loop.

37. Startup Complete
    Displayed when initialization is complete and the system is ready to display the main menu.

At this point, either the boot menu appears or the system autoboots.

4.4 Power-on Diagnostics Commands

The Power-on diagnostics (POD) provide an interface that allows the state of the system to be examined and modified. The PROM monitor can automatically drop into POD mode during system power-up, or in the event of an unexpected exception or diagnostic failure. POD mode can be entered manually, using the System Controller Debug Settings menu, or manually selected from the System Maintenance Menu (select **5**, then type **pod**). The POD commands that are useful as fault isolation tools are described in the following paragraphs.

Note: Set bit 5 in the Debug Settings menu to 1 in order to enter POD mode (refer to Section 3.5.2). The POD mode prompt is **POD xx/yy>**, where *xx* is the slot number of the current processor and *yy* is the CPU on the IP19 board.

Kill an individual microprocessor on the IP19 – At the POD prompt, use either the **disable** or **fdisable** commands. **disable** sets the disable bit in the stored hardware inventory. **fdisable** turns the specified processor off by writing to the CPU enable register. Both commands use the format **disable/fdisable** *x y*, where *x* is the cardcage slot number and *y* is the CPU number.

Note: The disabled processor is unable to write to memory. It remains disabled until the system is power-cycled, or the control register is rewritten with “f” and the “reset” command is typed.

List the physical locations of all boards installed in the system – Type **info** to display the board inventory.

Display the contents of all of IP19 registers – Type **dr all**.

Display the configuration of a specific board – Type **devc** *x*, where *x* is the slot number of the selected board.

Note: The **devc** command will not work if memory is not configured.

Display the configuration of a specific memory board – Type **dmc** *x*, where *x* is the slot number of the selected board.

Turn off individual banks of memory – Type **disable** *x y*, where x is the slot number of the selected memory board and y is the bank number.

Note: The system must be left with enough enabled memory to successfully boot. If you attempt to disable too much memory, the command will fail.

If memory is disabled, use the **reconf** command to reset the interleaving.

Reconfigure the enabled memory – Type **reconf** to reconfigure the memory using the currently enabled banks. The configuration will be displayed.

Display the cache error register contents – Type **ecc**. This command can only isolate the fault to either the primary or secondary caches.

Clear the Memory Error registers – Type **clear**. Use this command when POD is cycling a memory error message, to determine whether the message is old or new.

Start the PROM Monitor – Type **io**.

Display the reason why the system entered POD mode – Type **why**. Use when the original error message has scrolled off of the screen.

4.5 Niblet

Niblet is a small, symmetric multiprocessing kernel with separate virtual address spaces for its processes. Niblet was originally designed as a verification tool, but has been found useful for testing boards.

Niblet is composed of 13 separate tests. These tests are, in turn, combined in various combinations to form 13 test sets (Supertests). When Niblet is invoked, it attempts to execute all of the tests in the selected set (Niblet cannot run individual tests). Table 4-1 lists each of the basic Niblet tests, and Table 4-2 lists the available Supertests.

Note: Niblet attempts to run its tests on all of the processors that were present when the PROM set up the machine. If one or more processors are forced into POD mode (see Chapter 6), they are still included in Niblet's processor count, causing the system to hang.

Niblet may not run correctly if the system processors are running different versions of the IP19 PROM; however, if the processors launch successfully, Niblet will run as intended.

Test	Description
INVALID	Invalidates random TLB entries to cause more varied interactions.

Table 4-1 Basic Niblet Tests

Test	Description
COUNTER	Runs until a certain instruction count is reached and passed. The count is proportional to the Niblet process ID.
MPMON	Verifies that repetitive Everest reads and writes are identical.
MPINTADD	Two processors add values to a common variable, hit a barrier, and compare the final sum.
MPINTADD_4	Four-processor version of MPINTADD.
MPSLOCK	A software locking protocol test.
MPHLOCK	Tests load-link and store-conditional by grabbing a lock, storing a process ID to a protected location, waiting for a delay to expire, and then checking to see that the correct process ID is still there. Multiple processors try this; a failure should result in a processor reading the wrong PID.
MEMTEST	Tests a range of memory by writing a value, based on a process ID, to that range of memory and then verifying it. The current version's range is small enough to fit into a secondary cache.
BIGMEM	Same as above except that the range is larger than 1MB.
PRINTTEST	Tests Niblet context-switching (very fast sanity check).
BIGINTADD_4	Same as MPINTADD_4 except that it runs for a high number of iterations.
BIGHLOCK	Same as MPHLOCK except that it runs for a high number of iterations.

Table 4-1 (continued) Basic Niblet Tests

Test	Description
niblet 0	Runs one copy of the INVALID process. This test should always pass almost immediately.
niblet 1	Runs INVALID, COUNTER, COUNTER.
niblet 2	Runs MPMON, MPMON. Test takes disproportionately longer on single-processor compared to multi-processor machines.
niblet 3	Runs MPINTADD, INVALID, MPINTADD. Test takes disproportionately longer on single-processor compared to multi-processor machines.
niblet 4	Runs MPSLOCK, MPSLOCK, INVALID.

Table 4-2 Niblet Supertests

Test	Description
niblet 5	Runs MPROVE, MPSLOCK, MPROVE, MPSLOCK, INVALID.
niblet 6	Runs MPSLOCK, MPMON, INVALID, MPSLOCK, MPMON. Test takes disproportionately longer on single-processor compared to multi-processor machines.
niblet 7	Runs MPROVE, MPROVE.
niblet 8	Runs INVALID, MPMON, MPMON, MPROVE, MPROVE, MPROVE, MPINTADD, MPINTADD, MPHLOCK, MPHLOCK (total of 10 processes).
niblet 9	Runs MPINTADD_4, MPINTADD_4, MPINTADD_4, MPINTADD_4, INVALID, MPROVE, MPROVE, MPROVE, MPHLOCK, MPHLOCK, MPSLOCK, MPSLOCK (total of 12 processes).

Table 4-2 (continued) Niblet Supertests

As long as there are more processes than processors, Niblet tests will migrate. This is why there are three copies of INVALID in "niblet b." As long as INVALID is run on fewer than six processors, it will migrate eventually. Tests run on fewer processors will migrate more often.

If there are more processors than processes, one or more processors will go into a loop waiting for the Supertest to complete. Processors in this state will print "No processes left to run - twiddling."

Because Niblet is intended to run with one UART per processor, it only prints failure messages to the processor on which a test failed. The processor hosting the failure prints all pertinent information and then sends interrupts to the other processors. The cause of the failure is only available on the processor where the process actually failed; the other processors will print "Niblet failed on an interrupt." This is particularly important when Niblet fails due to a nonzero ERTOIP register, since the register can only be read by the processor on which the error occurred. The processor hosting the error prints "ERTOIP is nonzero! (ERTOIP, CAUSE, EPC)" followed by the values of ERTOIP, CAUSE, and EPC.

Whenever a Supertest completes, the bootmaster CPU prints "Supertest PASSED/FAILED" followed by "Niblet Complete." None of the 13 tests in the IP19 PROM should ever generate a "Supertest FAILED" message under normal circumstances.

Niblet can be initiated while in POD mode by typing **gm <return>** followed by **niblet** *n* **<return>**, where *n* is the set of tests that you wish to run.

4.6 IP19 PROM Error and Status Messages

During a normal system boot-up, the IP19 PROM provides a series of status messages that are displayed by the System Controller. In the event of a failure during the boot process, the IP19 PROM causes the appropriate error message to be displayed. If the system is a server, the error message is also displayed on the terminal.

Both status and error messages are displayed in the same format: A short status or error message appears near the top of the display. Immediately below it, a longer more descriptive version of the message scrolls by. This longer message is followed by a three-digit diagnostic code that corresponds to the displayed message.

Status messages scroll by as the machine comes up and will pass so quickly that they will be difficult to read. A failure causes the corresponding error message to continuously scroll across the display until the return key is pressed.

Note: If an IO4 failure has occurred, the error message will continue to scroll until the system is powered down. See Section 6.3.2 to redirect the IP19 PROM output to an external port.

The following sections list the various messages and their diagnostic codes.

IP19 PROM Messages (Short Form)

```
003 SCACHE FAILED!
004 SCACHE FAILED!
001 DCACHE FAILED!
002 DCACHE FAILED!
005 ICACHE FAILED!
006 ICACHE FAILED!
040 MC3 CONFIG FAILED!
041 NO GOOD MEMORY FOUND
042 MC3 CONFIG FAILED!
043 MC3 CONFIG FAILED!
044 MC3 READBACK ERROR!
047 MC3 CONFIG FAILED!
048 MC3 CONFIG FAILED!
049 MC3 CONFIG FAILED!
050 INSUFFICIENT MEMORY!
051 NO MEM BOARDS FOUND!
070 NO IO BOARDS FOUND!
071 IO4PROM FAILED!
072 IO4PROM FAILED!
073 IO4PROM FAILED!
074 IO4PROM FAILED!
075 IO4PROM FAILED!
078 IO4PROM FAILED!
079 NO EPC CHIP FOUND!
080 IO4 CONFIG FAILED!
081 MASTER IO4 FAILED!
082 MASTER IO4 FAILED!
083 MASTER IO4 FAILED!
084 MASTER IO4 FAILED!
085 MASTER IO4 FAILED!
086 MASTER IO4 FAILED!
088 MASTER IO4 FAILED!
089 MASTER IO4 FAILED!
090 MASTER IO4 FAILED!
091 MASTER IO4 FAILED!
092 MASTER IO4 FAILED!
```

```
093 MASTER IO4 FAILED!
094 MASTER IO4 FAILED!
087 EPC CHIP FAILED!
095 EPC UART FAILED!
123 BUS TAGS FAILED!
123 BUS TAGS FAILED!
124 BUS TAGS FAILED!
250 Reentering POD mode
251 PROM EXCEPTION!
252 PROM NMI HANDLER
253 CPU in POD mode.
```

IP19 PROM Messages (Long Form)

```
040 Memory board configuration has failed. Cannot load IO PROM.
041 All memory banks had to be disabled due to test failures.
042 The address line self-test failed. Cannot continue.
043 Memory board configuration has failed. Cannot load IO PROM.
044 Memory board configuration has failed. Cannot load IO PROM.
047 Memory board configuration has failed. Cannot load IO PROM.
048 Memory board configuration has failed. Cannot load IO PROM.
049 The PROM was unable to disable failing memory banks.
050 You must have at least 32 megabytes of working memory to load the IO PROM.
051 The IP19 PROM did not recognize any memory boards in the system.
070 The IP19 PROM did not recognize any IO4 boards in the system.
071 Diagnostics detected a problem with your IO4 PROM.
072 Diagnostics detected a problem with your IO4 PROM.
073 Diagnostics detected a problem with your IO4 PROM.
074 Diagnostics detected a problem with your IO4 PROM.
075 Diagnostics detected a problem with your IO4 PROM.
078 An exception occurred while downloading the IO4 PROM to memory.
079 There must be an EPC chip on the IO board in the highest-numbered slot.
080 An exception occurred while configuring an IO board.
081 The IA chip on the master IO4 board has failed diagnostics.
082 The IA chip on the master IO4 board has failed diagnostics.
083 The IA chip on the master IO4 board has failed diagnostics.
084 The IA chip on the master IO4 board has failed diagnostics.
085 The IA chip on the master IO4 board has failed diagnostics.
086 The IA chip on the master IO4 board has failed diagnostics.
088 The IA chip on the master IO4 board has failed diagnostics.
089 The IA chip on the master IO4 board has failed diagnostics.
090 The IA chip on the master IO4 board has failed diagnostics.
091 The IA chip on the master IO4 board has failed diagnostics.
092 The IA chip on the master IO4 board has failed diagnostics.
093 The IA chip on the master IO4 board has failed diagnostics.
094 The IA chip on the master IO4 board has failed diagnostics.
087 The EPC chip on the master IO4 board has failed diagnostics.
251 The PROM code took an unexpected exception.
252 The PROM received a nonmaskable interrupt.
```

Diagnostic Codes and Their Meanings

```
000 Device passed diagnostics.
001 Failed dcache1 data test.
002 Failed dcache1 addr test.
003 Failed scache1 data test.
```

```
004 Failed scache1 addr test.
005 Failed icache data test.
006 Failed icache addr test.
007 Dcache test hung.
008 Scache test hung.
009 Icache test hung.
040 Memory built-in self-test failed.
041 No working memory was found.
042 Memory address line test failed.
043 Memory data line test failed.
044 Bank failed configured memory test.
045 Slave hung writing to memory.
046 Bank disabled due to downrev MA chip.
047 A bus error occurred during MC3 config.
048 A bus error occurred during MC3 testing.
049 PROM attempted to disable the same bank twice.
050 Not enough memory to load the IO4 PROM.
051 No memory boards were recognized.
052 Bank forcibly re-enabled by the PROM.
060 CPU doesn't get interrupts from CC.
061 Group interrupt test failed.
062 Lost a loopback interrupt.
063 Bit in HPIL register stuck.
070 No working IO4 is present.
071 Bad checksum on IO4 PROM.
072 Bad entry point in IO4 PROM.
073 IO4 PROM claims to be too long.
074 Bad entry point in IO4 PROM.
075 Bad magic number in IO4 PROM.
078 Bus error while downloading IO4 PROM.
079 No EPC chip found on master IO4.
080 Bus error while configuring IO4.
081 Bus error during IA register test.
082 Bus error during IA PIO test.
083 IA chip register test failed.
084 Wrong error reported for bad PIO.
085 IA error didn't generate interrupt.
086 IA error generated wrong interrupt.
087 EPC register test failed.
088 Bus error on map RAM rd/wr test.
089 Bus error on map RAM address test.
090 Bus error on map RAM walking 1 test.
091 Bus error during map RAM testing.
092 Map RAM read/write test failed.
093 Map RAM address test failed.
094 Map RAM walking 1 test failed.
095 EPC UART loopback test failed.
120 CPU can't access memory
123 CC bus tag data test failed.
124 CC bus tag addr test failed.
125 CPU forcibly re-enabled by the PROM.
240 CPU writing configuration info.
246 CPU testing dcache.
247 CPU testing icache.
248 CPU testing scache.
249 CPU initializing caches.
250 CPU returning from master's code.
```

```
251 Unexpected exception.
252 A nonmaskable interrupt occurred.
253 POD mode switch set or POD key pressed.
253 Unspecified diagnostic failure.
254 Diagnostic value unset.
255 Device not present.
```

4.7 IP19 CPU Board Fault/Status Indicators

The IP19 board has a total of 24 fault indicators (see Figure 4-2). A bank of six LEDs is assigned to each processor. Each bank displays 44 status values and 15 error values. The values are displayed by the banks as a binary number, with the most significant bit represented by the topmost LED (as viewed from the front of the cardcage). The status values are displayed as the system progresses through the power-on tests. If a constant value is displayed, convert the binary value to a decimal number and use Section 4.7.1 to identify the status message. Status messages are displayed as a constant value and have the prefix "PLED" (PROM LED) attached to their description.

If a fatal error prevents the power-on tests from completing, the LEDs will flash the error value until the system is powered down or reset. Error messages have the prefix "FLED" (Flashing LED) attached to their descriptions. Section 4.7.2 lists the error codes.

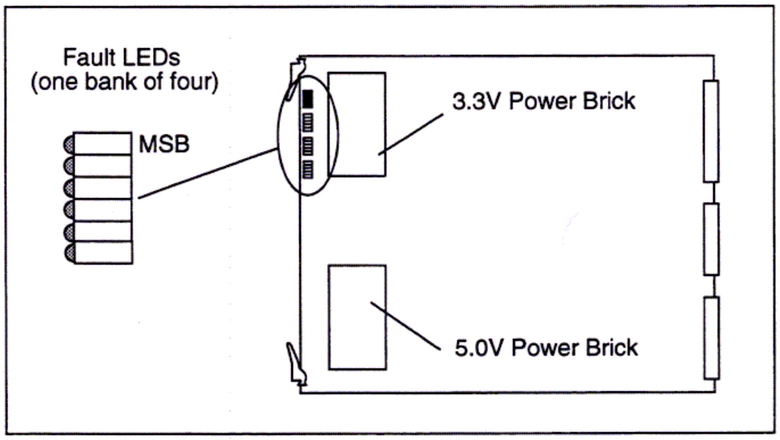


Figure 4-2 IP19 Board Fault Indicators

4.7.1 LED Status Codes

LED Pattern Displayed ● = Lit LSB MSB	Description (Constant Value Displayed)
● ○ ○ ○ ○ ○	PLED_CLEARTAGS (1) - Clearing the primary data cache tags.
○ ● ○ ○ ○ ○	PLED_CKCCLOCAL (2) - Testing CC chip local registers.
● ● ○ ○ ○ ○	PLED_CCLFAILED_INITUART (3) - Failed the local test but trying to initialize UART.
○ ○ ● ○ ○ ○	PLED_CCINIT1 (4) - Initializing the CC chip local registers.
● ○ ● ○ ○ ○	PLED_CKCCCONFIG (5) - Testing the CC chip config registers (requires usable bus to pass). If test hangs, usually means bus has failed. Check oscillator.
○ ● ● ○ ○ ○	PLED_CCCFAILED_INITUART (6) - Failed the config reg test but trying to initialize UART.
● ● ● ○ ○ ○	PLED_NOCLK_INITUART (7) - CC clock not running. Init UART anyway.
○ ○ ○ ● ○ ○	PLED_CCINIT2 (8) - Initializing the CC chip config registers.
● ○ ○ ● ○ ○	PLED_UARTINIT (9) - Initializing the CC chip UART. If test hangs, usually means bad UART clock. Check connections to System Controller.
○ ● ○ ● ○ ○	PLED_CCUARTDONE (10) - Finished initializing the CC chip UART.
● ● ○ ● ○ ○	PLED_ CKACHIP (11) - Testing the A chip registers.
○ ○ ● ● ○ ○	PLED_AINIT (12) - Initializing the A chip.
● ○ ● ● ○ ○	PLED_CKEBUS1 (13) - Checking the Ebus with interrupts.
○ ● ● ● ○ ○	PLED_SCINIT (14) - Initializing the system controller.
● ● ● ● ○ ○	PLED_BMARB (15) - Arbitrating for a bootmaster.
○ ○ ○ ○ ● ○	PLED_BMASTER (16) - This processor is the bootmaster.

Table 4-3 IP19 Board Test Status LED Codes

LED Pattern Displayed ● = Lit LSB MSB	Description (Constant Value Displayed)
● ○ ○ ○ ● ○	PLED_CKEBUS2 (17) - In second Ebus test. Run only by the bootmaster.
○ ● ○ ○ ● ○	PLED_POD (18) - Setting up this CPU slice for POD mode.
● ● ○ ○ ● ○	PLED_PODLOOP (19) - Entering POD loop.
○ ○ ● ○ ● ○	PLED_CKPDCACHE1 (20) - Checking the primary data cache.
● ○ ● ○ ● ○	PLED_MAKESTACK (21) - Creating a stack in the primary data cache.
○ ● ● ○ ● ○	PLED_MAIN (22) - Jumping into C code - calling main.
● ● ● ○ ● ○	PLED_CKIAID (23) - Check IA and ID chips on master IO4.
○ ○ ○ ● ● ○	PLED_CKEPC (24) - Check EPC chip on master IO4.
● ○ ○ ● ● ○	PLED_IO4INIT (25) - Initializing the IO4 PROM.
○ ● ○ ● ● ○	PLED_NVRAM (26) - Getting NVRAM variables.
● ● ○ ● ● ○	PLED_FINDCONS (27) - Checking the path to the EPC chip which contains the console UART.
○ ○ ● ● ● ○	PLED_CKCONS (28) - Testing the console UART.
● ○ ● ● ● ○	PLED_CONSINIT (29) - Setting up the console UART.
○ ● ● ● ● ○	PLED_CONFIGCPUS (30) - Configuring out CPUs that are disabled.
● ● ● ● ● ○	PLED_CKRAWMEM (31) - Checking raw memory (running Board Internal Self Test (BIST).

Table 4-6 (continued) IP19 Board Test Status LED Codes

LED Pattern Displayed ● = Lit LSB MSB	Description (Constant Value Displayed)
○ ○ ○ ○ ○ ●	PLED_CONFIGMEM (32) - Configuring memory.
● ○ ○ ○ ○ ●	PLED_CKMEM (33) - Checking configured memory.
○ ● ○ ○ ○ ●	PLED_WRCONFIG (34) - Writing evconfig structure: The bootmaster CPU writes the entire array. The slave CPUs only write their own entries.
● ● ○ ○ ○ ●	PLED_LOADPROM (35) - Loading IO4 PROM.
○ ○ ● ○ ○ ●	PLED_CKSCACHE1 (36) - First pass of secondary cache testing. Tests the scache like a RAM.
● ○ ● ○ ○ ●	PLED_CKPICACHE (37) - Check the primary instruction cache.
○ ● ● ○ ○ ●	PLED_CKPDCACHE2 (38) - Check the primary data cache writeback mechanism.
● ● ● ○ ○ ●	PLED_CHSCACHE2 (39) - Check the secondary data cache writeback mechanism.
○ ○ ○ ● ○ ●	PLED_CKBT (40) - Check the bus tags.
● ○ ○ ● ○ ●	PLED_BTINIT (41) - Clear the bus tags.
○ ● ○ ● ○ ●	PLED_CKPROM (42) - Checksum the I/O PROM.
● ● ○ ● ○ ●	PLED_INSLAVE (43) - This CPU is entering slave mode.
○ ○ ● ● ○ ●	PLED_PROMJUMP (44) - Jumpering to the I/O PROM.
● ○ ● ● ○ ●	PLED_SLAVEJUMP (45) - A slave is jumping to the IO4 PROM slave code.

Table 4-6 (continued) IP19 Board Test Status LED Codes

4.7.2 LED Error Codes

LED Pattern Displayed ● = Lit LSB MSB	Description (Flashing Value Displayed)
○ ● ● ● ○ ●	FLED_CANTSEEMEM (46) - Flashed by slave processors if they take an exception while trying to write their evconfig entries. Often means that processor is getting D-chip parity errors.
● ● ● ● ○ ●	FLED_NOUARTCLK (47) - The CC UART clock is not running. No system controller access possible.
○ ○ ○ ○ ● ●	FLED_IMPOSSIBLE1 (48) - System fell through an unreturning subroutine (shouldn't be possible).
● ○ ○ ○ ● ●	FLED_DEADCOP1 (49) - Coprocessor 1 is dead (no error does not mean coprocessor is good).
○ ● ○ ○ ● ●	FLED_CCCLOCK (50) - Cache controller (CC) clock is not running.
● ● ○ ○ ● ●	FLED_CCLOCAL (51) - Failed CC local register tests.
○ ○ ● ○ ● ●	FLED_CCCONFIG (52) - Failed CC config register tests.
● ○ ● ○ ● ●	FLED_ACHIP (53) - Failed A chip register tests.
○ ● ● ○ ● ●	FLED_BROKEWB (54) - By the time this CPU arrived at bootmaster arbitration barrier, the rendezvous time had passed. CPU is running too slowly, the ratio of the bus clock rate to CPU clock rate is too high, or a bit in the CC clock is stuck on.
● ● ● ○ ● ●	FLED_BADCACHE (55) - CPU's primary data cache test failed.
○ ○ ○ ● ● ●	FLED_BADIO4 (56) - IO4 board is bad (can't get to console).
● ○ ○ ● ● ●	FLED_UTLBMISS (57) - Took a TLB refill exception.
○ ● ○ ● ● ●	FLED_XTLBMISS (58) - Took an extended TLB refill exception.
● ● ○ ● ● ●	FLED_CACHE (59) - Unused.
○ ○ ● ● ● ●	FLED_GENERAL (60) - Took a general exception.
● ○ ● ● ● ●	FLED_NOTIMPL (61) - Took an unimplemented exception.
○ ● ● ● ● ●	FLED_ECC (62) - Took a cache error exception.

Table 4-7 IP19 Board Power-on Test Failure LED Codes

LED Pattern Displayed ● = Lit LSB MSB	Description (Flashing Value Displayed)
○ ● ● ○ ● ●	FLED_BROKEWB (54) - By the time this CPU arrived at bootmaster arbitration barrier, the rendezvous time had passed. CPU is running too slowly, the ratio of the bus clock rate to CPU clock rate is too high, or a bit in the CC clock is stuck on.
● ● ● ○ ● ●	FLED_BADCACHE (55) - CPU's primary data cache test failed.
○ ○ ○ ● ● ●	FLED_BADIO4 (56) - IO4 board is bad (can't get to console).
● ○ ○ ● ● ●	FLED_UTLBMISS (57) - Took a TLB refill exception.
○ ● ○ ● ● ●	FLED_XTLBMISS (58) - Took an extended TLB refill exception.
● ● ○ ● ● ●	FLED_CACHE (59) - Unused.
○ ○ ● ● ● ●	FLED_GENERAL (60) - Took a general exception.
● ○ ● ● ● ●	FLED_NOTIMPL (61) - Took an unimplemented exception.
○ ● ● ● ● ●	FLED_ECC (62) - Took a cache error exception.

Table 4-7 (continued) IP19 Board Test Failure LED Codes

Note: These binary error codes apply to all of the microprocessors resident on the board. IP19 boards are configured with all four banks of LEDs, regardless of the number of microprocessors installed.

4.7.3 LED Power-on Status Codes

When the Power-on Diagnostics (POD) are running, a pair of LEDs from each bank of processor LEDs will flash alternately. After POD runs and the system enters the PROM monitor, the LEDs on the bootmaster CPU will display a fixed value (binary 18). All other slave processors will loop on a pattern waiting for a command (see Figure 4-5).

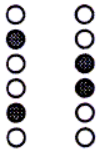


Figure 4-3 Slave Processor LED Pattern

The bootmaster CPU will loop on the pattern shown in Figure 4-6 when polling the CC UARTs.

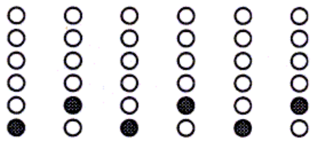


Figure 4-4 CPU LED Pattern When Polling

4.8 Board Configuration Operations

This section describes the commands used to verify and/or change the system configuration.

Check the current system configuration using either the **`hinv`** command, or two variations of it; **`hinv -b`**, and **`hinv -b -v`**:

`hinv` performs exactly as it did in previous releases.

`hinv -b` is similar to the **`info`** command in POD and provides additional information, such as: the number of processors present, the amount of memory installed, and whether or not an IO4 board is present.

`hinv -b -v` supplies additional information about each processor, memory bank, and I/O adapter.

Modify the system configuration using the **`enable`** and **`disable`** commands:

`enable` *x y*, where *x* is the board slot number and *y* is the memory bank number, turns on the selected memory bank on the MC3 board. The disable command works in the same way.

`enable` *x y*, where *x* is the cardcage slot number and *y* is the CPU number, turns on the selected processor on the IP19 board. The disable command works in the same way. Entering **`enable`** *x*, without specifying the CPU, number turns on every CPU resident in the specified cardcage slot.

Note: The system must be power-cycled following an enable command, in order for the new configuration to be activated. The disable command does not require the system to be power-cycled.

Change the system configuration and then power-cycle the system. Enter one of the **`hinv`** commands. The current configuration is compared to the stored hardware inventory, and any variations are flagged.

Revise the stored hardware inventory to reflect the new configuration by entering **`update`**. This command rewrites the stored hardware inventory locations in the IO4 board's NVRAM.

4.9 PROM Monitor Boot Commands

This section describes how to reconfigure the system to boot from a different system disk.

1. Bring up the System Maintenance menu.
2. Type **5** to enter the Command Monitor.
3. Type **printenv** and the following screen is displayed.

```
>> printenv
SystemPartition=scsi(0)disk(1)partition(8)
OSLoadPartition=scsi(0)disk(1)partition(0)
AutoLoad=No
dbgtty=multi(0)serial(0)
root=dks0d1s0
nonstop=0
rbaud=19200
TimeZone=PST8PDT
console=d
diskless=0
dbaud=9600
sgilogo=y
netaddr=192.48.150.68
ConsoleOut=multi(0)serial(0)
ConsoleIn=multi(0)serial(0)
cpufreq=50
```

Note: The lines in bold contain the values that must be changed before the system will boot from the new disk. In this example, the address of the new system disk is “4.”

4. Use the **setenv** command to enter the following information:

```
>> setenv systempartition dksc(0,4,8)
>> setenv osloadpartiton dksc(0,4,0)
>> setenv root dks0d4s0
>> single
```

The system will begin to boot in single-user mode.

Chapter 5

IDE

5.1 Overview

This chapter provides the Everest board tests that are presently supported by IDE, along with explanations of the various types of error messages.

5.2 IDE Test Structure

This section describes the IDE board tests. Because the information is still volatile, it is provided as screen captures for this revision of the manual.

```
IO4 IDE Guide
-------------
1. boot ide
2. The default report level is 2. Set the report level by typing the following:
        report=#

        level 5 : debugging messages displayed. Don't need this much detail.
        level 4 : prints out memory locations as they are written. Will
                  slow down testing time.
        level 3 : prints out 1-line functional descriptions within tests. This
                  is probably the most useful level for general use.
        level 2 : prints out only errors and titles.
        level 1 : prints out only titles and pass/fail.

        level n will print out all messages for level n and below.

3. Quickmode. For right now, you set quick mode like this:
        setenv quickmode 1

        And you unset quickmode by doing a : unsetenv quickmode
        Eventually ide will have a switch method of setting/unsetting quick
        mode.

        NOTE: All current IO4 tests run fast enough that there is no
        difference between quick and long test modes for the IO4.  If the
        total elapsed time for running all IO4 tests ever exceeds 10
        minutes, quick mode will be enabled for the IO4.
```

4. continue-on-error. For right now, you set this mode like this:

```
setenv cont_on_err 1
```

For the IO4 tests, this will continue the test even when an error has been encountered. Normally, the tests will stop after the first error.

NOTE: continue-on-error is currently the default of most of the tests. The tests will be changed to use the continue-on-error switch.

5. io_all. This command runs all working/known bugfree IO4 tests that do not require human intervention. Any mostly working but possible buggy tests, as well as any tests requiring a human to interpret the results, are not included.

6. There are currently tests for the following areas of the IO4 board: IO4 interface, VME adapter, SCSI adapter, and EPC adapter.

The detailed tests are listed below.

IO4 Interface:

---

check_iocfg - Checks IO4 config against NVRAM

This test compares the actual setup of the IO4 board to the values specified in the NVRAM. Each IO4 board in the system is checked to see that it has all the adapters specified in NVRAM, and that they are of the specified types.

In addition, if "report" is set VERBOSE, configuration information for each board is printed out even if no errors occur.

---

io4_regtest - Read/Write test of IO4 registers

```
    IO4_CONF_LW
    IO4_CONF_SW
    IO4_CONF_ADAP
    IO4_CONF_INTRVECTOR
    IO4_CONF_GFXCOMMAND
    IO4_CONF_ETIMEOUT
    IO4_CONF_RTIMEOUT
    IO4_CONF_INTRMASK
```

Although these are not the only IO4 registers, they are the only ones that may safely be Read/Write tested.

---

io4_pioerr - IO4 PIO bus error test

Attempts to generate an error interrupt by attempting a write to IO adapter 0 (nonexistant). This tests the IO4 error generation capability and the IO4 to IP error path.

---

mapram_test - Read/Write test of IO4 map ram

As the name implies, tests the IO4 mapping ram as a small memory array.

Tests memory with pattern Read/Write, address-in-address, and marching 1's test patterns.

---

check_hinv - Checks type of board in each slot

Not a test per se - merely prints out the locations and types of all boards currently installed in the system.

VME adapter:

NOTE - none of these have yet been enabled for the io_all command
More details will be added after testing is completed.

---

f_regs - Test VMECC registers

---

vme_reg - Test VMECC registers

---

vintr - Test VMEcc self interrupts

---

vberr - Test VMEcc bus errors

---

vlpbk - Test VMEcc loopback capability

---

cddata - Test cdsio interrupts

---

cdintr - Test cdsio interrupts

---

vdma - Test VMEcc DMA Engine

SCSI adapter:

---

s1_regtest - Register Read/Write test for s1 chip

This is a basic Read/Write test for the S1 chip registers. It does tests and address-in-address testing for:

```
    S1_INTF_R_SEQ_REGS 0 - 0xF
    S1_INTF_R_OP_BR_0
    S1_INTF_R_OP_BR_1
    S1_INTF_W_SEQ_REGS 0 - 0xF
    S1_INTF_W_OP_BR_0
    S1_INTF_W_OP_BR_1
```

(36 registers currently tested)

Although these are not the only S1 registers, they are the only ones that may safely be used by Read/Write tests.

---

regs_95a - Register Read/Write test for wd95a chip

Read/Write test for the wd95a chip registers. This test is still being worked on - more details later.

---

scsi_intr - SCSI interrupt test
scsi_selftest - SCSI device self test
scsi_dma - SCSI dma error generation test

These three tests are currently under development - more details later.

EPC adapter:

----

epc_regtest - Register Read/Write test for epc chip

Basic Read/Write test for the EPC chip registers, including the Parallel Port registers. Registers tested:

EPC_IIDDUART0
EPC_IIDDUART1
EPC_IIDENET
EPC_IIDPROFTIM
EPC_IIDSPARE
EPC_IIDPPORT
EPC_IIDERROR
EPC_EADDR0
EPC_EADDR1
EPC_EADDR2
EPC_EADDR3
EPC_EADDR4
EPC_EADDR5
EPC_TCMD
EPC_RCMD
EPC_TBASELO
EPC_TBASEHI
EPC_TLIMIT
EPC_TTOP

EPC_TITIMER
EPC_RBASELO
EPC_RBASEHI
EPC_RLIMIT
EPC_RTOP
EPC_RITIMER
EPC_PPBASELO
EPC_PPBASEHI
EPC_PPLEN
EPC_PPCTRL

As stated above, this is a good basic test for the Parallel Port; for more thorough testing a test fixture is required.

----

epc_nvram - NVRAM Read/Write test

Does Read/Write pattern and address-in-address testing for all the NVRAM accessable to the EPC chip. Although the NVRAM is physically on the RTC chip, it occupies a seperate address space and is accessed differently, hence the separate test.

----

epc_rtcreg - RTC register/NVRAM Read/Write test

Read/Write test for the RTC registers and the small amount of NVRAM in the RTC address space portion of the RTC chip. Registers tested:

NVR_SEC
NVR_SECALRM
NVR_MI
NVR_MINALRM
NVR_HOUR
NVR_HOURALRM
NVR_WEEKDAY
NVR_DAY
NVR_MONTH
NVR_YEAR

NVRAM tested is in the address range 0xE - 0x3F.

---

epc_rtcinc - RTC clock increment test

Tests the ability of the RTC chip to handle time-of-day transitions. Sets the RTC to a known time and date (last second of the year), waits one second, and checks to make certain that the time and date have changed correctly.

---

epc_rtcint - RTC Interrupt generation test

Tests to make certain that the RTC can correct generate Alarm, Periodic, and Update interrupts. Validates the path from the RTC chip to the IP board's master CPU.

---

duart_loopback - Duart loopback test

Attempts to configure and test all available serial ports. Does loopback testing at all baud rates for each port tested. Normally uses internal loopback, but if invoked with "duart_loopback -e" assumes that an external loopback fixture is being used.

IP19 IDE Guide

1. boot ide
2. The default report level is 2. Set the report level by typing the following:

   report=# where # is any number from 1 to 5.

```
level 5 : debugging messages displayed. Don't need this much detail.
level 4 : prints out memory locations as they are written. Will
          slow down testing time.
level 3 : prints out 1-line functional descriptions within tests. This
          is probably the most useful level for general use.
level 2 : prints out only errors, titles and pass/fail.
level 1 : prints out only titles and pass/fail.
```

   level n will print out all messages for level n and below.

3. There are currently 4 classes of IP19 tests: IP, TLB, FPU and CACHE.

   Each test in these classes can be invoked by a command from the IDE. They are listed as follows:

ip(1 - 8) Tests IP19 components not covered by TLB, FPU and CACHE.
tlb(1 - 9) Tests the TLB in R4K.
fpu(1 - 14) Tests the FPU in R4K.
cache(1 - 48) Tests the primary and secondary cache for R4K.

In addition, there are commands to invoke each class of tests in entirety, all classes of tests in entirety, and commands to facilitate the test and repair process.

ipall invokes tests ip1 through ip8.
tlball invokes tests tlb1 through tlb9.
fpuall invokes tests fpu1 through fpu14.
cacheall invokes tests cache1 through cache48.
ip19 invokes all IP, TLB, FPU and CACHE tests.

cache49 invokes a short version of cache48.
cstate(0 - 21) invokes individual cache state tests in cache48.
quickfpu invokes tests fpu1 through fpu13, skipping fpu14.
quickcache invokes tests cache1 through cache44, then cache47, skipping cache45, 46 and 48.
quickip19 invokes all IP, TLB, FPU and CACHE tests except fpu14, cache45, 46 and 48.

ipresults displays the test summaries for ipall.
tlbresults displays the test summaries for tlball.
fpuresults displays the test summaries for fpuall or quickfpu.

cacheresults displays the test summaries for cacheall or quickcache.
ip19results displays the test summaries for ip19 or quickip19. This command is automatically invoked at the end of ip19 and quickip19.

4. A brief description of each test and the possible errors are provided below for your reference. The number preceding each error message identifies each error uniquely and its format should be interpreted as follows:

01ccnnn 01 - the board id for IP19
cc - the hint for failed component(s)
01 - A chip
02 - D chip
03 - CC chip
04 - Primary cache
05 - Secondary cache
06 - R4400
07 - Primary or secondary cache
08 - TLB
09 - FRU
nnn - the error id

ip1 (local_regtest) - Check CC local registers

Basic write/read test for the local registers. The registers tested are limited to the following:

EV_WGDST	Write gatherer destination
EV_WGCNTRL	Write gatherer control
EV_IP0	Interrupts 63 - 0
EV_IP1	Interrupts 127 - 64
EV_CEL	Current execution level
EV_IGRMASK	Interrupt group mask
EV_ILE	Interrupt level enable
EV_ERTOIP	Error/timeout interrupt
EV_ECCSB_DIS	ECC single-bit error disable

The read-only registers are read and their contents are reported. These registers are:

EV_SPNUM	Slot/Processor info
EV_SYSCONFIG	System configuration
EV_HPIL	Highest pending interrupt level
EV_RO_COMPARE	RTC compare
EV_RTC	Real time clock
EV_WGCOUNT	Write gatherer count

Possible error:

0103001: Local register %s R/W error : Wrote 0x%llx Read 0x%llx

---

ip2 (cfig_regtest) - Check configuration registers

Basic write/read test for the configuration registers. The registers tested are limited to the following:

EV_PGBRDEN	Write gatherer destination
EV_PROC_DATARATE	Write gatherer control
EV_WGRETRY_TOUT	Interrupts 63 - 0
EV_CACHE_SZ	Interrupts 127 - 64
EV_CMPREG0 - 3	Timer comparator registers

Note that the timer comparator registers are checked via the read-only RTC compare register.

Possible error:

---

ip3 (bustags_reg) - Check bus tags

This test calculates the size of bus tag space based on the size of the secondary cache. Then it performs basic write/read test on the bus tags.

Possible error:

0103003: Bus tag addr 0x%x R/W error : Wrote 0x%x Read 0x%x

---

ip1 (local_regtest) - Check CC local registers

Basic write/read test for the local registers. The registers tested are limited to the following:

EV_WGDST	Write gatherer destination
EV_WGCNTRL	Write gatherer control
EV_IP0	Interrupts 63 - 0
EV_IP1	Interrupts 127 - 64
EV_CEL	Current execution level
EV_IGRMASK	Interrupt group mask
EV_ILE	Interrupt level enable
EV_ERTOIP	Error/timeout interrupt
EV_ECCSB_DIS	ECC single-bit error disable

The read-only registers are read and their contents are reported. These registers are:

EV_SPNUM	Slot/Processor info
EV_SYSCONFIG	System configuration
EV_HPIL	Highest pending interrupt level
EV_RO_COMPARE	RTC compare
EV_RTC	Real time clock
EV_WGCOUNT	Write gatherer count

Possible error:

0103001: Local register %s R/W error : Wrote 0x%llx Read 0x%llx

---

ip2 (cfig_regtest) - Check configuration registers

Basic write/read test for the configuration registers. The registers tested are limited to the following:

EV_PGBRDEN	Write gatherer destination
EV_PROC_DATARATE	Write gatherer control
EV_WGRETRY_TOUT	Interrupts 63 - 0
EV_CACHE_SZ	Interrupts 127 - 64
EV_CMPREG0 - 3	Timer comparator registers

Note that the timer comparator registers are checked via the read-only RTC compare register.

Possible error:

---

ip3 (bustags_reg) - Check bus tags

This test calculates the size of bus tag space based on the size of the secondary cache. Then it performs basic write/read test on the bus tags.

Possible error:

0103003: Bus tag addr 0x%x R/W error : Wrote 0x%x Read 0x%x

---

ip7 (intr_timer) - Check IP19 RTSC and interval timer

This test generates level 1 interrupt by writing a value into the EV_CMPREG configuration registers so that the RTSC will reach this value and interrupts the processor.

Possible errors:

```
010301b: Invalid timer interrupt occurred
010301c: Interval timer interrupt did not occur
010301d: Group interrupt pending not cleared in Cause : Cause 0x%x
```

---

ip8 (intr_group) - Check IP19 processor group interrupt

This test generated level 0 interrupts using different processor groups at different priority levels including broadcast interrupts.

Possible errors:

```
010301e: Group interrupt pending not set correctly in EV_IP0 : Expected 0x%llx
Got
0x%llx
010301f: Group highest priority interrupt level failure : HPIL 0x%llx
0103020: Group interrupt not indicated in Cause register 0x%x
0103021: Group interrupt pending not cleared : IP0 0x%llx IP1 0x%llx
0103022: Group highest priority interrupt level not cleared : HPIL 0x%llx
0103023: Group interrupt pending not cleared in Cause register : Cause 0x%x
0103024: Group interrupt did not occur : group 0x%x priority 0x%x
0103025: Group interrupt pending not cleared in Cause : Cause 0x%x
```

---

tlb2 (tlb_probe) - Check TLB functionality

Sets up all the TLB slots and then probes them with matching addresses. Checks to ensure that there is a response for each valid address.

Possible error:

```
0108018: TLB probe error : Expected entry %d Got entry %d vpnum %d addr 0x%x
```

---

tlb3 (tlb_xlate) - Check TLB address translation

Tests for correct virtual to physical translation via mapped TLB entries. Sets the virtual address to user segment and uncached.

Possible errors:

```
010801b: TLB entry %d unexpected exception for addr 0x%x
010801c: TLB entry %d translation error at addr 0x%x : Wrote %d Read %d
```

---

tlb4 (tlb_valid) - Check TLB valid exception

Tests to see if TLB invalid accesses generate exceptions. Maps the TLB entries to invalid addresses in k2seg and attempts to access them.

Possible errors:

```
0108016: TLB entry %d invalid exception VADDR error : Expected 0x%x Got 0x%x
0108017: TLB entry %d invalid exception didn't occur
```

---

tlb5 (tlb_mod) - Check TLB modification exception

This test sets up the TLB to map each page as non-writable, then attempts to write to each of the mapped pages. It verifies that an exception is generated

---

tlb1 (tlb_ram) - Test R4K TLB as RAM

Tests the TLB as a small memory array. Checks to see if all the read/write bits can be toggled and that all undefined bits read back zero.

Possible errors:

```
0108001: TLBHI      entry %d R/W error: Wrote 0x%x Read 0x%x
0108002: TLBLO even entry %d R/W error: Wrote 0x%x Read 0x%x
0108003: TLBLO odd  entry %d R/W error: Wrote 0x%x Read 0x%x
```

---

tlb2 (tlb_probe) - Check TLB functionality

Sets up all the TLB slots and then probes them with matching addresses. Checks to ensure that there is a response for each valid address.

Possible error:

```
0108018: TLB probe error : Expected entry %d Got entry %d vpnum %d addr 0x%x
```

---

tlb3 (tlb_xlate) - Check TLB address translation

Tests for correct virtual to physical translation via mapped TLB entries. Sets the virtual address to user segment and uncached.

Possible errors:

```
010801b: TLB entry %d unexpected exception for addr 0x%x
010801c: TLB entry %d translation error at addr 0x%x : Wrote %d Read %d
```

---

tlb4 (tlb_valid) - Check TLB valid exception

Tests to see if TLB invalid accesses generate exceptions. Maps the TLB entries to invalid addresses in k2seg and attempts to access them.

Possible errors:

```
0108016: TLB entry %d invalid exception VADDR error : Expected 0x%x Got 0x%x
0108017: TLB entry %d invalid exception didn't occur
```

---

tlb5 (tlb_mod) - Check TLB modification exception

This test sets up the TLB to map each page as non-writable, then attempts to write to each of the mapped pages. It verifies that an exception is generated for each write attempt.

Possible errors:

```
010800b: TLB %s entry %d mod exception VADDR error : Expected 0x%x Got 0x%x
010800c: TLB %s entry %d mod exception didn't occur
010800d: TLB %s entry %d unexpected exception during mod
010800e: TLB %s entry %d mod error : Wrote 0x%x Read 0x%x
```

----

tlb6 (tlb_pid) - Check TLB refill exception

Tests each TLB slot by attempting access with both matching and non-matching process id. It verifies that matching pid accesses are allowed and non-matching pid accesses generate exceptions.

Possible errors:

```
0108015: TLB %s entry %d unexpected exception with matching pid 0x%x
0108016: TLB %s entry %d refill exception VADDR error : Expected 0x%x Got 0x%x
0108017: TLB %s entry %d refill exception didn't occur
```

----

tlb7 (tlb_g) - Check global bit in TLB entry

Sets up all the TLB slots to allow global access, then attempts access on all slots with a variety of different pid settings. This test passes only if no invalid access exceptions occur.

Possible error:

```
010801d: Unexpected exception occurred during global access
```

----

tlb8 (tlb_c) - Check C bits in TLB entry

Attempts to access TLB-mapped memory in both cached and uncached modes. Tests all slots by writing and reading back a pattern, first in cached mode, then in uncached mode. This test checks basic functionality, and does not attempt to detect cached/uncached interactions.

Possible errors:

```
010800f: Exception during cached write to 0x%x
0108010: Cached write to 0x%x failed
0108011: TLB %s entry %d cached mode exception
0108012: TLB %s entry %d cached R/W error : Wrote 0x%x Read 0x%x
0108013: TLB %s entry %d uncached mode exception
0108014: TLB %s entry %d uncached R/W error : Wrote 0x%x Read 0x%x
```

----

tlb9 (tlb_mapuc) - Check cached/uncached TLB access

Checks that both cached and uncached mapped access work without interfering with each other. This test aims at detecting the R4000 mapped uncached writeback bug. The method used is to set up 2 TLB entries for the same page of physical memory, one using cached access and the other using uncached. A write is done via each of the TLB entries, followed by a read. If the R4000 cache is working properly, the test will be able to read back the correct (different) pattern for each access mode, since the code avoids flushing the cache to main memory. If the bug is present, the same value will be read back via both cached and uncached access. The writes are done in both cached - uncached and uncached - cached orders.

Possible errors:

```
0108004: TLB %s entry %d cached/uncached W exception
0108005: TLB %s entry %d cached/uncached W error : Wrote 0x%x Read 0x%x
0108006: TLB %s entry %d uncached/cached W execption
0108007: TLB %s en
```

---

fpu1 (fpregs) - fpu register test

This test simply writes and reads the FPU registers, reporting any readback errors.

Possible errors:

```
010901e: FP register %d data error : Expected 0x%x Got 0x%x
010901f: FP register %d inverted data error : Expected 0x%x Got 0x%x
```

---

fpu2 (fpmem) - fpu load/store mem test

This test loads FPU from memory and stores memory from FPU.

Possible errors:

```
010901c: Load/store FP reg %d data error : Expected 0x%x Got 0x%x
010901d: Load/store FP reg %d inverted data error : Expected 0x%x, Got 0x%x
```

---

fpu3 (faddsubs) - fpu add/subtract(single precision)

Tests addition and subtraction using simple single precision arithmetic.

Possible errors:

```
0109004: FP single add/sub result error : Expected 0x%x Got 0x%x
0109005: FP single add/sub status error : Expected 0 Got 0x%x
0109006: Fixed to single conversion failed : Before 0x%x After 0x%x
```

---

fpu4 (faddsubd) - fpu add/subtract(double precision)

Tests addition and subtraction using simple double precision arithmetic.

Possible errors:

```
0109001: FP double add/sub result error : Expected 0x%x Got 0x%x
0109002: FP double add/sub status error : Expected 0 Got 0x%x
```

```
0109003: Fixed to double conversion failed : Before 0x%x After 0x%x
```

----

fpu5 (fmuldivs) - fpu multiply/divide (single precision)

Tests multiplication and division using simple single precision arithmetic.

Possible errors:

```
0109011: FP single divide result error : Expected 0x%x Got 0x%x
0109012: FP single multiply result error : Expected 0x%x Got 0x%x
```

----

fpu6 (fmuldivd) - fpu multiply/divide (double precision)

Tests multiplication and division using simple double precision arithmetic.

Possible errors:

```
010900f: FP double divide result error : Expected 0x%x Got 0x%x
0109010: FP double multiply result error : Expected 0x%x Got 0x%x
```

----

fpu7 (fmulsubs) - fpu multiply/subtract (single precision)

Tests multiplication and subtraction using simple single precision arithmetic.

Possible errors:

```
0109016: FP single mul/div result error : Expected 0x%x Got 0x%x
0109017: Fixed to single conversion failed : Before 0x%x After 0x%x
0109018: FP single mul/div status error : 0x%x
```

----

fpu8 (fmulsubd) - fpu multiply/subtract (double precision)

Tests multiplication and subtraction using simple double precision arithmetic.

Possible errors:

```
0109013: FP double mul/sub result error : Expected 0x%x Got 0x%x
0109014: Fixed to double conversion failed : Before 0x%x After 0x%x
0109015: FP double mul/div status error : 0x%x
```

----

fpu9 (finvalid) - fpu invalid test

Simple test to see if an invalid operation exception can be generated. Divides

0.0 by itself to generate the exception.

Possible errors:

```
010900b: Invalid exception didn't occur
010900c: Invalid exception status error : 0x%x
010900d: Invalid exception dividend error : Expected 0x%x Got 0x%x
```

---

fpu10 (fdivzero) - fpu divided by zero test

Divides a non-zero value by 0.0. Unlike the previous test, the floating point status register is checked after the exception to make sure the divide by zero flag is set.

Possible errors:

```
0109007: Divide by Zero exception status error : 0x%x
0109008: Dividend conversion failed : Before 0x%x After 0x%x
0109009: Divisor conversion failed : Before 0x%x After 0x%x
```

---

fpu11 (foverflow) - fpu overflow test

Generates a single precision overflow by adding 2 at-the-limit large values. After the exception, the floating point status register is checked to make sure the overflow flag was set.

Possible error:

```
0109019: Overflow exception status error : 0x%x
```

---

fpu12 (funderflow) - fpu underflow test

Generates a single precision overflow by dividing an at-the-limit small value by 2. After the exception, the floating point status register is checked to make sure the underflow flag was set.

Possible errors:

```
0109020: Exception other than Underflow in FCR31 : 0x%x
0109021: Failed to generate Underflow Exception
```

---

fpu13 (finexact) - fpu inexact test

Generates a single precision inexact conversion error by attempting to convert an integer value too large for a single precision representation into a single precision value. After the error, the floating point status register is checked to make sure the inexact conversion flag was set.

Possible error:

```
010900a: Inexact exception status error : 0x%x
```

---

fpu14 (fpcmput) - fpu computation test

Given a list of "infinite" series, this test executes them a specified number of times and compares the result gotten at run-time with an expected result. Discrepancies are reported. This is a slow test.

Possible errors:

```
010900e: FP computation unexpected exception : 0x%x
010901a: Single precision %s error : Expected 0x%x Got 0x%x
010901b: Double precision %s error : Expected 0x%x 0x%x Got 0x%x 0x%x
```

---

cache1 (Taghitst) - TAGHI Register Test

This diag tests the data integrity of the taghi register. A sliding one and a sliding zero pattern are used.

Possible errors:

```
0104001: Taghi register failed walking one test
         Expected data: 0x%08x Actual data: 0x%08x
0104002: Taghi register failed walking zero test
         Expected data: 0x%08x Actual data: 0x%08x
```

---

cache2 (Taglotst) - TAGLO Register Test

This diag tests the data integrity of the taglo register. A sliding one and a sliding zero pattern are used.

Possible errors:

```
0104003: Taglo register failed walking one test
         Expected data: 0x%08x Actual data: 0x%08x
0104004: Taglo register failed walking zero test
         Expected data: 0x%08x Actual data: 0x%08x
```

---

cache3 (pdtagwlk) - Primary data TAG RAM data line Test

This diag checks the data integrity of the primary data TAG RAM path using walking ones and walking zeros patterns.

Possible error:

```
0104005: D-cache tag ram data line error
         Failed walking one (or zero) test at 0x%08x
         Expected: 0x%08x Actual 0x%08x
```

---

cache4 (pdtagadr) - Primary data TAG RAM address line Test

This diag tests the address lines to the primary data cache TAG RAM by sliding a one and then a zero on the address lines. This test assumes that the taglo register is in good working condition.

Possible error:

```
0104006: D-cache tag ram address line error
         Failed walking one (or zero) test at 0x%08x
         Expected: 0x%08x Actual 0x%08x
```

---

cache5 (PdTagKh) - Primary data TAG Knaizuk Hartmann Test

This diag tests the data integrity of the primary data cache TAG RAM with the Knaizuk Hartmann algorithm. It treats the TAG RAM array as a ordinary memory array. The parity bit is not checked in this test.

A note about the Knaizuk Hartmann Memory Test

This algorithm is used to perform a fast but non-ehaustive memory test. It will test a memory subsystem for stuck-at faults in both the address lines as well as the data locations.

The algorithm breaks up the memory to be tested into 3 partitions. Partition 0 consists of memory locations 0, 3, 6, ...; partition 1 consists of memory locations 1,4,7,...; partition 2 consists oflocations 2,5,8... The partitions are filled with either an all ones pattern or an all zeroes pattern. By varying the order in which the partitions are filled and then checked, this algorithm manages to check all combinations of possible stuck at faults.

Possible errors:

```
0104007: Partition 1 error after partition 0 set to 0xaaaaaaaa
0104008: Partition 2 error after partition 1 set to 0xaaaaaaaa
0104009: Partition 0 error after partition 1 set to 0xaaaaaaaa
010400a: Partition 1 error after partition 1 set to 0xaaaaaaaa
010400b: Partition 0 error after partition 0 set to 0x55555555
010400c: Partition 2 error after partition 2 set to 0xaaaaaaaa
```

For each of the above errors, the following additional information is also provided:

```
         Tag ram address: 0x%08x
         Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
```

---

cache6 (pitagwlk) - Primary Instruction TAG RAM data line Test

This diag checks the data integrity of the primary instruction cache TAG RAM path using a walking ones and zeros pattern.

Possible error:

```
010400d: I-cache tag ram data line error
         Failed sliding one (or zero) test at 0x%08x
         Expected: 0x%08x, Actual: 0x%08x
```

---

cache7 (pitagadr) - Primary Instruction TAG RAM address line Test

This diag tests the address lines to the primary instruction cache TAG RAM by sliding a one and then a zero one the address lines. This test assumes that the taglo register is in good working condition.

Possible error:

```
010400e: I-cache tag ram address line error
         Failed sliding one (or zero) test at 0x%08x
         Expected: 0x%08x Actual 0x%08x
```

---

cache8 (PiTagKh) - Primary Instruction TAG RAM Knaizuk Hartmann Test

This diag tests the data integrity of the primary instruction cache TAG RAM with the Knaizuk Hartmann algorithm. It treats the TAG RAM array as a ordinary memory array. The parity bit is not checked in this test.

Possible errors:

```
010400f: Partition 1 error after partition 0 set to 0xaaaaaaaa
0104010: Partition 2 error after partition 1 set to 0xaaaaaaaa
0104011: Partition 0 error after partition 1 set to 0xaaaaaaaa
0104012: Partition 1 error after partition 1 set to 0xaaaaaaaa
0104013: Partition 0 error after partition 0 set to 0x55555555
0104014: Partition 2 error after partition 2 set to 0xaaaaaaaa
```

For each of the above errors, the following additional information is provided:

```
         Tag ram index address: 0x%08x
         Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
```

---

cache9 (sd_tagwlk) - Secondary TAG data path Test

Checks the data integrity of the Secondary data TAG RAM path using a walking ones/zeros pattern.

Possible error:

```
0105015: Secondary Data TAG RAM Path Error
         on sliding one (or zero) pattern
         TAG RAM Location 0x%x
         Expected 0x%x  Actual= 0x%x  XOR= 0x%x
```

---

cache10 (sd_tagaddr) - Secondary TAG address Test

Checks the address integrity to the Primary Data TAG RAM by using a walking address.

Possible error:

```
0105016: Secondary Data TAG Address Error
         TAG RAM Location 0x%x
         Expected 0x%x  Actual= 0x%x  XOR= 0x%x
```

---

cache11 (sd_tagkh) - Secondary TAG RAM Knaizuk Hartmann Test

This diag tests the data integrity of the secondary data cache TAG RAM with the Knaizuk Hartmann algorithm. It treats the TAG RAM array as a ordinary memory array. The parity bit is not checked in this test.

Possible error:

```
0105017: Secondary Data TAG ram data Error
         Address %x, error code %d
         expected %x, actual %x, XOR %x
```

---

cache12 (d_tagparity) - Primary Data TAG RAM parity Test

This diag tests the functionality of the parity bit in the primary data cache tag. For each tag, a stream of one's and zero's are shifted into the tag to check if the parity bit change state accordingly.

Possible error:

```
0104018: D-cache tag ram parity bit error
         Tag ram address: 0x%08x expected content: 0x%08x
         Taglo: 0x%08x expected parity: 0x%x actual parity: 0x%x
```

---

cache13 (d_tagcmp) - Primary Data TAG comparitor Test

This diag tests the comparator at the D-cache tag for hit and miss detection. For each tag, set the ptag field with the values which will cause a cache hit for the Kseg0 address of 0x80002000 to 0x9fffffff. The values used are a walking one or a walking zero pattern. This will ensure only one bit location is tested at the comparator. The cache op Hit Invalidate is used to check for cache hit and miss situations.

Possible errors:

```
0104019: D-cache tag comparator did not detect a miss
0104020: D-cache tag comparator did not detect a hit
```

For each of the above errors, the following additional information are provided:

```
         Tag ram address: 0x%08x
         PTag field of tag: 0x%06x comparing with PFN: 0x%06x
```

---

cache14 (d_tagfunct) - Primary Data TAG functionality Test

This diag tests the functionality of the data cache tag. Kseg0 addresses are used to load the cache from memory. The ptag and the cache state field are checked to see if they are holding expected values. Virtual addresses 0x80000000, 0x80002000, 0x80004000, 0x80008000, ... 0x90000000 are used as the baseaddress of an 8k page which is mapped to the cache. The ptag and state of each cache line are checked against the expected value.

Possible errors:

```
0104021: D-cache tag functional error in PTAG field
         PTag field does not contain correct tag bits
         Cache line address: 0x%08x
         Expected PTag: 0x%06x
         Actual PTag: 0x%06x
         TAGLO Register %x
         Re-read DTAG %x
0104022: D-cache tag functional cache state error
         Cache line address: 0x%08x
         Expected cache state: 0x%08x
         Actual cache state: 0x%08x
         TAGLO Register %x
         Re-read DTAG %x
```

cache15 (d_slide_data) - Primary Data RAM data line Test

This diag tests the data lines to the primary data cache. A sliding one and a sliding zero data pattern is written into the first location of the D-cache to check if each data line can be toggled individually.

Possible errors:

```
0107023: D-cache data ram data lines failed walking one test
         Addr: 0x%08x Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
0107024: D-cache data ram data lines failed walking zero test
         Addr: 0x%08x Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
```

cache16 (d_slide_addr) - Primary Data RAM address line Test

This diag tests the address lines to the primary data cache. Each address line to the data cache is toggled once individually by sliding a one and then a zero across the address lines.

Possible errors:

```
0107025: D-cache data ram address lines failed walking one tes
         Addr: 0x%08x Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
0107026: D-cache data ram address lines failed walking zero test
         Addr: 0x%08x Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
```

cache17 (d_kh) - Primary Data RAM Knaizuk Hartmann Test

This diag tests the data integrity of the D-cache with the Knaizuk Hartmann algorithm. Data pattern 0x55555555 and 0xaaaaaaaa are used.

Possible errors:

```
0107027: Partition 1 error after partition 0 set to 0xaaaaaaaa
0107028: Partition 2 error after partition 1 set to 0xaaaaaaaa
0107029: Partition 0 error after partition 1 set to 0xaaaaaaaa
010702a: Partition 1 error after partition 1 set to 0xaaaaaaaa
010702b: Partition 0 error after partition 0 set to 0x55555555
010702c: Partition 2 error after partition 2 set to 0xaaaaaaaa
```

For each of the above errors, the following additional information is provided:

```
         Cache address: 0x%08x
         Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
```

---

cache18 (dsd_wlk) - Primary/Secondary Data path Test

Test the data path from memory through the secondary cache and to the Primary Data Cache.

Possible errors:

```
010702d: Data Path Error from Memory->Secondary->Primary Data
         Address %x, expected %x, actual %x, Xor %x
010702e: Data Path Error from Primary ->Secondary->Memory Data
         Address %x, Expected %x, Actual %x, Xor %x
```

---

cache19 (sd_aina) - Secondary Data RAM (address in address) Test

Performs an address in address test on the secondary data cache.

Possible errors:

```
010502f: Secondary Memory Error on pattern 1
         Address %08x
         expected %08x, actual %08x, XOR %08x
0105030: Secondary Memory Error on pattern 2
         Address %08x
         expected %08x, actual %08x, XOR %08x
```

---

cache20 (d_function) - Primary Data functionality Test

This diag tests the functionality of the entire data cache. It checks the block fill, write back on a dirty line replacement, and no write back on a clean line replacement function of the data cache lines.

Possible errors:

```
0104031: D-cache block fill error 1
         Cache contains incorrect data
         Cache Address: 0x%08x
         Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
0104032: D-cache block fill error 2
         Cache contains incorrect data
         Cache Address: 0x%08x
         Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
0104033: D-cache block write back error 1
         Memory contains incorrect data
         Cache Address: 0x%08x
         Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
0104034: D-cache block fill error 3
         Cache contains incorrect data
         Cache Address: 0x%08x
         Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
0104035: D-cache block write back error 2
         Memory content is altered
         Write back happened on a clean line
         Cache Address: 0x%08x
         Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
```

---

cache21 (d_parity) - Primary Data parity generation Test

This diag tests the parity bit generation of the D-cache data ram.

Possible error:

```
0104036: D-cache parity generation error
         error %x
         Cache byte address: 0x%08x data:0x%02x
         Parity bit position: 0x%02x
         Expected parity: 0x%02x Actual parity:0x%02x
```

---

cache22 (i_tagparity) - Primary Instruction TAG RAM parity bit Test

This diag tests the functionality of the parity bit in the primary I-cache tag. For each tag, the parity bit is tested to respond to each bit change in the tag.

Possible error:

```
0104037: I-cache tag ram parity bit error
         Tag ram address: 0x%08x expected content: 0x%08x
         Taglo: 0x%08x expected parity: 0x%x actual parity: 0x%x
```

---

cache23 (i_tagcmp) - Primary Instruction TAG RAM comparitor Test

This diag tests the comparator at the I-cache tag for hit and miss detection.

Possible errors:

```
0104038: I-cache tag comparator did not detect a miss (walking 1)
0104039: I-cache tag comparator did not detect a hit (walking 1)
010403a: I-cache tag comparator did not detect a miss (walking zero)
010403d: I-cache tag comparator did not detect a hit (walking zero)
```

For each of the above errors, the following additional information is provided:

```
         Tag ram address: 0x%08x
         PTag field of tag: 0x%06x comparing with PFN: 0x%06x
```

---

cache24 (i_tagfunct) - Primary Instruction TAG functionality Test

This diag tests the functionality of the instruction cache tag. Kseg0 addresses are used to load the cache from memory. This will test if the cache is functional on the cachable memory space. After each 8k segment of memory is loaded into the cache. The ptag and the cache state field are checked to see if they are holding expected values. Virtual addresses 0x80000000, 0x80002000, 0x80004000, 0x80008000, ..., 0x90000000 are used as the base address of each 8k page which is mapped to the cache. The ptag and cache state of each cache line are checked against the expected value.

Possible errors:

```
010403b: I-cache tag functional error in PTAG field
         PTag field does not contain correct tag bits
         Cache line address: 0x%08x
         Expected PTag: 0x%06x
         Actual PTag: 0x%06x
010403c: I-cache tag functional cache state error
         Cache state not correct
         Cache line address: 0x%08x
         Expected cache state: 0x%08x
         Actual cache state: 0x%08x
```

----

cache25 (i_slide_data) - Primary Instruction data RAM data line Test

This diag checks the data lines to the I-cache data ram by sliding a one and zero bit across the bus.

Possible errors:

```
010403f: I-cache data ram data lines failed walking one test
         Addr: 0x%08x Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
     PITAG  %x
     PDTAG  %x
     STAG  %
0104040: I-cache data ram data lines failed walking zero test
     Addr: 0x%08x Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
     PTAG  %x
     STAG  %x
```

----

cache26 (i_aina) - Primary Instruction data RAM address in address Test

Performs an address in address test on the primary instruction cache.

Possible error:

```
0107041: I-cache address in address error
     addr %x, exp %x, act %x, XOR %x
```

----

cache27 (i_function) - Primary Instruction functionality Test

This diag tests the functionality of the entire instruction cache. It checks the block fill and hit write back of the instruction cache lines.

Possible error:

```
0107042: I-cache block write back error
     Memory contains incorrect data
     Cache address: 0x%08x
     Expected: 0x%08x Actual: 0x%08x Xor: 0x%08x
     Icache TAG = %x
     Scache TAG = %x
```

----

cache28 (i_parity) - Primary Instruction parity generation Test

This diag tests the parity bit generation of the I-cache data ram.

Possible error:

```
0104043: I-cache parity generation error
     error %x
     Cache byte address: 0x%08x data:0x%02x
     Parity bit position: 0x%02
     Expected parity: 0x%02x Actual parity:0x%02x
```

---

cache29 (i_hitinv) - Primary Instruction Hit Invalidate Test

This diag tests the Hit Invalidate cache op on the Instruction cache.

Possible errors:

```
0104044: I-cache state error during initialization
     Cache state did not change to valid when filled from memory
     Cache line address: 0x%08x
     Expected cache state: 0x%08x Actual cache state: 0x%08x
0104045: I-cache state error
     Hit Invalidate changed the line to invalid on a miss
     Cache line address: 0x%08x
     Miss address: 0x%08x
     Expected cache state: 0x%08x Actual cache state: 0x%08x
0104046: I-cache state error on a Hit Invalidate Cache OP
     Hit Invalidate did not invalidate the line on a hit
     Cache line address: 0x%08x
     Expected cache state: 0x%08x Actual cache state: 0x%08x
```

---

cache30 (i_hitwb) - Primary Instruction Hit Writeback Test

This diag tests the Hit Writeback cache op on the instruction cache.

Possible errors:

```
0104047: I-cache state error during initialization
     Cache state did not change to valid when filled from memory
     Cache line address: 0x%08x
     Expected cache state: 0x%08x Actual cache state: 0x%08x
0104048: I-cache state error Hit writeback happened on a cache miss
     Cache line address: 0x%08x
     Miss address: 0x%08x
0104049: I-cache Hit writeback did not happen on a cache hit
     Cache line address: 0x%08x
     expected %x, actual %x, XOR %x
```

---

cache31 (ECC_reg_tst) - ECC register Test

This diag tests the data integrity of the ECC register. A sliding one and sliding zero pattern is used in this test.

Possible errors:

```
010404a: ECC register failed walking one test
     Expected data: 0x%08x Actual data: 0x%08x
010404b: ECC register failed walking zero test
     Expected data: 0x%08x Actual data: 0x%08x
```

----

cache32 (dd_hitinv) - Primary Data Hit Invalidate Test

This diag tests the Hit Invalidate cache op on the data cache.

Possible errors:

```
010404c: D-cache state error during initialization
     Cache state did not change to valid when filled from memory
     Cache line address: 0x%08x
     Expected cache state: 0x%08x Actual cache state: 0x%08x
010404d: D-cache state error
     Hit Invalidate changed the line to invalid on a miss
     Cache line address: 0x%08x
     Miss address: 0x%08x
     Expected cache state: 0x%08x Actual cache state: 0x%08x
010404e: D-cache state error on a  Hit Invalidate Cache OP
     Hit Invalidate did not invalidate the line on a hit
     Cache line address: 0x%08x
     Expected cache state: 0x%08x Actual cache state: 0x%08x
```

----

cache33 (d_hitwb) - Primary Data Hit Writeback Test

This diag tests the Hit Writeback cache op on the data cache.

Possible errors:

```
010404f: D-cache state error during initialization
     Cache state did not change to valid when filled from memory
     Cache line address: 0x%08x
     Expected cache state: 0x%08x Actual cache state: 0x%08x
     TAGLO Reg %x
     Re-Read dtag %x
     Re-Read stag %x
0104050: D-cache state error Hit writeback happened on a clean exclusive line
     Cache line address: 0x%08x
     PTAG %x
     Scache TAG %x
0104051: D-cache Hit writeback happened on a cache miss
     Cache line address: 0x%08x
     Miss address: 0x%08x
     PTAG %x
     Scache TAG %x
0104052: D-cache Hit writeback did not happen on a cache hit
     Cache line address: 0x%08x
     PTAG %x
     Scache TAG %x
0104053: D-cache Hit Writeback clears the write back bi
     Cache line address: 0x%08x
```

---

cache34 (d_dirtywbw) - Primary Data dirty writeback word Test

This test verifies the block (4 words) write mode in data cache. It writes to K0 (0x80020000) cached space, causing the cache dirty. Then it replace the cache line by reading 0x80022000, different cache line with same offset. This causes the data in 0x80020000 writeback to memory which now has the same data as in 0x80020000. Multiple cache lines are tested back to back.

Possible errors:

```
0104054: Unexpected Cache write through to memory
     addr %x, expected %x, actual %x, XOR %x
     Seconday TAG %x
0104055: Cache writeback did not occur on a word store to a dirty line
     addr %x
     expected %x, actual %x, XOR %x
     Seconday TAG %x
```

---

cache35 (d_refill) - Primary Data refill from Secondary Cache Test

This test verifies the block write/read mode in data cache. It writes to K0 (0x80020000) cached space, causing the cache dirty. Then it replace the cache line by reading 0x80022000, different cache line with same offset. This causes the data in primary data cache to be written back to the secondary. The address 0x80020000 is reread and compared. Should be a cache hit in the secondary.

Possible errors:

```
0104056: Unexpected Cache write through to memory
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
0104057: Secondary Cache miss, expected a cache hit
     addr = %x
     xpected = %x, actual = %x, XOR = %x
     Data in memory = 0xdeadbeef
     Seconday TAG %x
```

---

cache36 (sd_dirtywbw) - Secondary Dirty Writeback (word) Test

This test verifies the block (4 words) write mode in data cache. It writes to K0 (0x80020000) cached space, causing the cache dirty. Then it replace the cache line by reading 0x80022000, different cache line with same offset. This causes the data in 0x80020000 write back to secondary which now has the same data as in 0x80020000. A write to address 0x80060000 will replace the secondary lines, thus forcing a writeback from the Secondary Cache. Note, there is another flavor of this test d_dirtywbw.c which forces the writeback from the primary when the secondary line is replaced.

Possible errors:

```
0105058: Unexpected Cache write through to memory
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
0105059: Data read replaced a dirty line in Secondary
     Dirty line not written back to memory
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
```

---

cache37 (sd_dirtywbh) - Secondary Dirty Writeback (halfword) Test

This test verifies the block (4 words) write mode in data cache. It writes to K0 (0x80020000) cached space, causing the cache dirty. Then it replace the cache line by reading 0x80022000, different cache line with same offset. This causes the data in 0x80020000 write back to memory which now has the same data as in 0x80020000. Multiple cache lines are tested back to back. Half word transactions are tested.

Possible errors:

```
010505a: Unexpected Cache write through to memory on store halfword
     addr = %x
     expected = %4x, actual = %4x, XOR %4x
     Seconday TAG %
010505b: Halfword read replaced a dirty line in Secondary, dirty line not
written back to memory
     addr = %x
     expected = %4x, actual = %4x, XOR %4x
     Seconday TAG %x
```

---

cache38 (sd_dirtywbb) - Secondary Dirty Writeback (byte) Test

This test verifies the block (4 words) write mode in data cache. It writes to K0 (0x80020000) cached space, causing the cache dirty. Then it replace the cache line by reading 0x80022000, different cache line with same offset. This causes the data in 0x80020000 write back to memory which now has the same data as in 0x80020000. Multiple cache lines are tested back to back. Byte transactions are tested.

Possible errors:

```
010505c: Unexpected Cache write through to memory on store byte
     addr = %x
     expected = %2x, actual = %2x, XOR %2x
     Seconday TAG %x
010505d: Byte read replaced a dirty line in Secondary, dirty line not written
back to memory
     Dirty line not written back to memory
     addr = %x
     expected = %2x, actual = %2x, XOR %2x
     Seconday TAG %x
```

---

cache39 (sd_tagecc) - Secondary TAG ECC Test

Checks the data integrity of the Secondary data tag ram path using a walking ones/zeros pattern.

Possible errors:

```
010505e: Secondary Data TAG RAM ECC Path error (walking one as data)
     TAG RAM Location 0x%x
     Expected 0x%x  Actual= 0x%x  XOR= 0x%x
010505f: Secondary Data TAG RAM ECC Path error (walking zero as data)
     TAG RAM Location 0x%x
     Expected 0x%x  Actual= 0x%x  XOR= 0x%x
```

----

cache40 (sdd_hitinv) - Secondary Hit Invalidate Test

This test verifies the Hit Invalidate Cache operation.

Possible errors:

```
0105060: S-cache state error during initialization
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
0105061: S-Cache error during Primary Cache dirty line writeback to Scache
0105062: S-Cache state error on a  Hit Invalidate Cache OP
0105063: Data written back to memory after a Hit Invalidate on the Secondary
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
0105064: S-Cache state error on a  Hit Invalidate Cache OP
0105065: Primary Cache TAG not invalid after a Hit Invalidate on the Scache
     addr %x
     Seconday TAG %x
     Primary TAG %x
0105066: Data written back to memory after a Hit Invalidate on the Secondary
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
     Primary TAG %x
```

For errors 0105061, 0105062 and 0105064, the following additional information are provided:

```
     Error in Secondary Cache TAG State field
      OR Error in Secondary Cache TAG physical tag field
      OR Error in Secondary Cache TAG Virtual Address field
     Address 0x%08x\nSecondary TAG Data 0x%08x
     Expected Cache State:  0x%x = [STATE]
```

STATE is one of the decoded cache states: Invalid, Clean Exclusive, Dirty Exclusive, Shared, and Dirty Shared.

----

cache41 (sd_hitwb) - Secondary Hit Writeback Test

This test verifies the Hit Writeback Cache operation.
It verifies that the data can be written back from the Secondary

or in the case where the primary data is more current that the data is written from the Primary to memory. Also checked is the fact that the cache lines are not invalidated as with the Hit Writeback Invalidate Cache Op. Instead it checks that the lines are set to the clean exclusive state.

Possible errors:

```
0105067: Initialization error, unexpected Cache write through to memory
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
0105068: SCache error during Primary Cache dirty line writeback to Scache
0105069: Data not written back from Scache to Memory on Hit Writeback Cache OP
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
010506a: Initialization error, unexpected Cache write through to memory
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
010506b: SCache state error during Hit Writeback on S-Cache dirty line
010506c: Error in Primary Cache TAG after a Hit Writeback cache Op on the SCache
     addr %x
     Expected cache state: Dirty Exclusive
     Primary Data TAG %x
010506d: Data not written back from D-Cache to Memory on a Hit Writeback on the
S-Cache
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
     Primary Data TAG %x
```

For errors 0105068 and 010506b, the following additional information are provided:

```
     Error in Secondary Cache TAG State field
       OR Error in Secondary Cache TAG physical tag field
       OR Error in Secondary Cache TAG Virtual Address field
     Address 0x%08x\nSecondary TAG Data 0x%08x
     Expected Cache State:  0x%x = [STATE]
```

STATE is one of the decoded cache states: Invalid, Clean Exclusive, Dirty Exclusive, Shared, and Dirty Shared.

----

cache42 (sd_hitwbinv) - Secondary Hit Writeback Invalidate Test

This test verifies the Hit Writeback Invalidate Cache operation. It verifies that the data can be written back from the Secondary or in the case where the primary data is more current that the data is written from the Primary to memory. Also checked is the fact that the cache lines are invalidated.

Possible errors:

```
010506e: Initialization error, unexpected Cache write through to memory
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
010506f: S-Cache TAG error after Hit Writeback Invalidate cacheo
0105070: Data not written back from Scache to Memory after Hit Writeback
Invalidate Cacheop
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
0105071: Initialization error, unexpected Cache write through to memory
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
0105072: S-Cache TAG error after Hit Writeback Invalidate cacheop, test case 2
0105073: Error in Primary Cache TAG after a Hit Writeback Invalidate cacheop or
the SCache
     addr %x
     Expected cache state: Invalid
     Primary Data TAG %x
0105074: Data not written back from D-Cache to Memory on a Hit Writeback
Invalidate on the S-Cache
     addr = %x
     expected = %x, actual = %x, XOR %x
     Seconday TAG %x
     Primary Data TAG %x
```

For errors 010506f and 0105072, the following additional information are provided:

```
     Error in Secondary Cache TAG State field
      OR Error in Secondary Cache TAG physical tag field
      OR Error in Secondary Cache TAG Virtual Address field
     Address 0x%08x\nSecondary TAG Data 0x%08x
     Expected Cache State:  0x%x = [STATE]
```

STATE is one of the decoded cache states: Invalid, Clean Exclusive, Dirty Exclusive, Shared, and Dirty Shared.

---

cache43 (cluster) - Secondary Cluster Test

Possible errors:

```
0105075: SCache data incorrectly written to memory during a dirty writeback
operation
     1st mem block
     Mem Address 0x%08x
     Expected 0x%08x, Actual 0x%08x, XOR 0x%08x
0105076: SCache data incorrectly written to memory during a dirty writeback
operation
     2nd mem block
     Mem Address 0x%08x
     Expected 0x%08x, Actual 0x%08x, XOR 0x%08x
```

---

cache44 (clusterwb) - Secondary Cluster Writeback Test

```
Possible errors:

0105077: SCache data incorrectly written to memory during a dirty writeback
operation on 1st block
     Mem Address 0x%08x
     Expected 0x%08x, Actual 0x%08x, XOR 0x%08x
0105078: SCache data incorrectly written to memory during a dirty writeback
operation on 2nd block
     Mem Address 0x%08x
     Expected 0x%08x, Actual 0x%08x, XOR 0x%08x
0105079: SCache data incorrectly written to memory during a dirty writeback
operatio on 3rd block
     Mem Address 0x%08x
     Expected 0x%08x, Actual 0x%08x, XOR 0x%08x

----------------------------------------------------------------------------
cache45 (hammer_pdcache) - stress primary D-cache--runs icached


Possible error:

010407b: Primary cache stress error at addr : 0x%x Expected 0x%x Got 0x%x

----------------------------------------------------------------------------
cache46 (hammer_scache) - stress secondary cache--runs icached


Possible error:

010507c: Secondary cache stress error at addr : 0x%x Expected 0x%x Got 0x%x

----------------------------------------------------------------------------
cache47 (cache_stress) - cache stress test

Write/read to one word in every page through 0x80000000 space.

Possible error:

010507a: Secondary cache stress error at addr : 0x%x Expected 0x%x Got 0x%x

----------------------------------------------------------------------------
cache48 (cache_states) - complete cache-state transitions test

The abbreviation of the following cache states are to be used in the
description of each cache state transition test:

   CE  clean exclusive
   DE  dirty exclusive
   I   invalid

cstate0 (RHH_CE_CE)
-------------------
Read hit primary (CE) and 2nd (CE). Check that the value is
correct (the physmem addr) and that both tags are still CE.

cstate1 (RHH_DE_DE)
-------------------
```

Read hit primary (DE) and 2nd (DE). Check value and that both are still DE.

cstate2 (WHH_CE_CE)

Write hit primary (CE) and 2nd (CE). Check that 2nd and memory still have old value and that both cache lines are now DE.

cstate3 (WHH_DE_DE)

Write hit primary (DE) and 2nd (DE). Check that 2nd and memory still have old value and that both lines are still DE.

cstate4 (RMH_I_CE)

Read miss primary (I) and hit 2nd (CE). Check that 2nd and memory still have old value and that both lines are CE.

cstate5 (RMH_I_DE)

Read miss primary (I) and hit 2nd (DE). Check that 2nd and memory still have old value and that both lines are DE.

cstate6 (RMH_CE_CE)

Read miss primary (CE) and hit 2nd (CE). Check that 2nd and memory still have old value and that both lines are still CE.

cstate7 (RMH_DE_DE)

Read miss primary (DE) and hit 2nd (DE). Check that 2nd and memory still have old value and that both lines are still CE.

cstate8 (WMH_I_CE)

Write miss primary (I) and hit 2nd (CE). Check that 2nd and memory still have old value and that both lines are DE.

cstate9 (WMH_I_DE)

Write miss primary (I) and hit 2nd (DE). Check that 2nd and memory still have old value and that both lines are DE.

cstate10 (WMH_CE_CE)

Write miss primary (CE) and hit 2nd (CE).

cstate11 (WMH_DE_DE)

Write miss primary (DE) and hit 2nd (DE).

cstate12 (RMM_I_I)

Read miss primary (I) and 2nd (I). Check that value is correct, that 2nd and memory still have old value and that both lines are CE.

cstate13 (RMM_I_CE)

```
-------------------
Read miss primary (I) and miss 2nd (CE). Check that value is
correct, that 2nd and memory still have old value and that both lines are CE.

cstate14 (RMM_I_DE)
-------------------
Read miss primary (I) and miss 2nd (DE). Check that 2ndary line
matches memory, that both tags are CE, that the addr tags on both lines
are correct, and that the dirty altaddr secondary line was flushed to memory.

cstate15 (RMM_CE_CE)
--------------------
Read miss primary (CE) and miss 2nd (CE). Fill cache lines with
a word from physaddr+2ndcachesize; do a read, then check that the tags for
both lines are CE and have the correct phys addrs, and that the alternate
memory word hasn't changed ###.

cstate16 (RMM_DE_DE)
--------------------
Read miss primary (DE) and miss 2nd (DE). Fill cache lines with
a word from physaddr+2ndcachesize; do a read, then check that the tags for
both lines are now CE and have the correct phys addrs, and that the alternate
memory word was written when the altaddr line was flushed.

cstate17 (WMM_I_I)
------------------
Write miss primary (I) and 2nd (I). Check that 2ndary line matches
memory, that both tags are DE, and that the addr tags on both lines are
correct.

cstate18 (WMM_I_CE)
-------------------
Write miss primary (I) and miss 2nd (CE). Check that 2ndary line
matches memory, that both tags are DE, and that the addr tags on both lines
are correct.

cstate19 (WMM_I_DE)
-------------------
Write miss primary (I) and miss 2nd (DE). Check that 2ndary line
matches memory, that both tags are DE, that the addr tags on both lines are
correct, and that the dirty altaddr secondary line was flushed to memory.

cstate20 (WMM_CE_CE)
--------------------
Write miss primary (CE) and miss 2nd (CE). Fill cache lines with
a word from physaddr+2ndcachesize; do a store, then check that the tags for
both lines are DE and have the correct phys addrs, and that the alternate
memory word hasn't changed.

cstate21 (WMM_DE_DE)
--------------------
Write miss primary (DE) and miss 2nd (DE). Check that 2ndary line
matches memory, that both tags are DE, that the addr tags on both lines are
correct, and that the dirty altaddr primary and secondary lines were flushed
to memory.
```

Possible errors:

```
010707d: RHH_CE_CE : physaddr 0x%x contents incorrect (0x%x)
010707e: RHH_DE_DE : physaddr 0x%x contents incorrect (0x%x)
010707f: RMH_I_CE : physaddr 0x%x contents incorrect (0x%x)
0107080: RMH_I_DE : physaddr 0x%x contents incorrect (0x%x)
0107081: RMH_CE_CE : physaddr 0x%x contents incorrect (0x%x)
0107082: RMH_DE_DE : physaddr 0x%x contents incorrect (0x%x)
0107083: RMM_I_I : physaddr 0x%x contents incorrect (0x%x)
0107084: RMM_I_CE : physaddr 0x%x contents incorrect (0x%x)
0107085: RMM_I_DE : physaddr 0x%x contents incorrect (0x%x)
0107086: PRIMARYD cache state error at addr 0x%x : Expected 0x%x Got 0x%x
     OR  PRIMARYI cache state error at addr 0x%x : Expected 0x%x Got 0x%x
     OR  SECONDAR cache state error at addr 0x%x : Expected 0x%x Got 0x%x
0107087: PRIMARYD addr error at slot 0x%x : Expected 0x%x Got 0x%x
     OR  PRIMARYI addr error at slot 0x%x : Expected 0x%x Got 0x%x
     OR  SECONDARY addr error at slot 0x%x : Expected 0x%x Got 0x%x
0107088: Mem value error at addr 0x%x : Expected 0x%x Got 0x%
0107089: Writeback missed 2ndary level cache at addr 0x%x
010708a: 2ndary cache value error at addr 0x%x : Expected 0x%x Got 0x%x
```

MC3 IDE Guide

1. boot ide

2. The default report level is 2. Set the report level by typing the following:
   report=#

   level 5 : debugging messages displayed. Don't need this much detail.
   level 4 : prints out memory locations as they are written. Will slow down testing time.
   level 3 : prints out 1-line functional descriptions within tests. This is probably the most useful level for general use.
   level 2 : prints out only errors and titles.
   level 1 : prints out only titles and pass/fail.

   level n will print out all messages for level n and below.

3. Quickmode. For right now, you set quick mode like this:
   setenv quickmode 1

   And you unset quickmode by doing a : unsetenv quickmode
   Eventually ide will have a switch method of setting/unsetting quick mode.

   For the memory tests, quick mode will test every nth byte instead of every byte, where n varies from 96 to 7680 depending upon the test. The goal in quick mode is to test 16GB in about 10 minutes-- and this is accomplished by testing every nth byte. n varies depending upon how fast or slow a test was timed to run.

4. continue-on-error. For right now, you set this mode like this:
   setenv cont_on_err 1

And you unset it by doing a : unsetenv cont_on_err
Eventually ide will have a switch method of setting/unsetting quick mode.

For the memory tests, this will continue the test even when an error has been encountered. Normally, the tests will stop after the first error.

5. memall and memquick. These are 2 defined commands. Each can be run in quick mode or in normal mode. memall will run all commands, while memquick runs just the faster tests (mem1, mem2, mem3, mem5, mem8, mem9, mem10).

6. There are currently 15 memory commands, mem1-mem15. They are detailed below:

---

mem1 - Read the mc3 configuration registers (real fast)

The following registers are probed:

reg	test description
00	Read the BankEnable
01	Read BoardType
02	Read RevLevel
03	Read AccessControl: endianness, subBlockOrder, ebus=64bitsOrNot
04	Read MemoryErrorInterrupt
05	Read EBUSErrorInterrupt
06	Read BIST result
07	Read DRSC timeout
0a	Read LeafControlEnable
	Read leaf regs 10-24, 30-33 (leaf0), 50-64, 70-73 (leaf1)

mem1 is very similar to mem14 which is the pod-mode dmc command.

---

mem2 - Memory sockets connection test (ported from IP17, mem1) (real fast)

The memory sockets connection test writes patterns to the first 2Kbytes of each configured leaf and then reads them back. By writing 2Kbytes, all simms are ensured of being written to regardless of the interleaving factor specified.

If the pattern read back does not match, the socket is assumed to have a connection problem.

---

mem3 - Walking address test (ported from IP17's mem2) (real fast)

This is a traditional test that checks for shorts and opens on the address lines. Address lines that are greater or equal to the most significant address lines of the memory bounds are not tested. Testing is done by byte read/writes from first_address up to last_address.

---

mem4 - Write/read data patterns (ported from IP17’s mem3) (4 min / 128MB)

This test does word read/writes of all-1’s and all-0’s patterns. It shows if all addresses appear to be writable, and that all bits may be set to both 1 and 0. However, it provides no address error or adjacent-bits-shorted detection. The flow is as follows:

(w0), u(r0,w1), d(r1,w5a), u(r5a,ra5), d(ra5) -- word and byte
(read as: write 0 to all locations, read 0 and write 1 to all locations in ascending order, read 1
and write 5a to all locations in descending order, read 5a and write a5 to all locations in
ascending order, read a5 from all locations in descending order)

mem13 does byte read/writes in the same pattern. The tests were separated out since the byte read/writes take a long time.

---

mem5 - Address in address memory test (4 min / 128MB)

This is a traditional, hueristic, rule-of-thumb, “address-in-address” memory test. It also puts the complement of the address in the address, and makes passes in both ascending and descending addressing order. There are both full memory store then check passes, as well as read-after-write passes (with complementing).

---

mem6 - walking 1/0 memory test (slow: 40 minutes / 32 MB)

Another traditional test - walking 1’s and walking 0’s through memory. This is a whole-memory test that is very good at shaking out shorted data bits, but provides little protection for addressing errors.

---

mem7 - MarchX (4 min / 128MB)

Described in van de Goor’s book, “Testing Semiconductor Memories” and has the following flow:

(w0), u(r0,w1), d(r1,w0), (r0)

Will detect address decoder faults, stuck-at-faults, transition faults, coupling faults, and inversion coupling faults(see van de Goor for definitions)

---

mem8 - MarchY (4 min / 128MB)

Described in van de Goor’s book, “Testing Semiconductor Memories” and has the following flow:

(w0), u(r0,w1,r1), d(r1,w0,r0), (r0)

Will detect address decoder faults, stuck-at-faults, transition faults, coupling faults, and linked transition faults(see van de Goor for definitions)

---

mem9 - Memory with ecc test (ported from IP17's mem6) (2 min /128MB)

This test writes to memory via uncached space and reads back through cached space (ECC exceptions enabled). Although it provides a simple level of ECC checking, its main function is to verify that cached and uncached memory addresses are accessing the same area of physical memory. The test values used are address-in-address and inverted address-in-address patterns, so a certain amount of address uniqueness checking is done as well.

---

mem10 - Cache write-through memory test (ported from IP17's mem9)(2 min /128MB)

This is a traditional, hueristic, rule-of-thumb, "address-in-address" memory test. It also puts the complement of the address in the address, making passes in ascending order only. All of memory is stored and then checked. All reads and writes are made through K0 seg, so the the reads and writes are cached. However, since the size of main memory exceeds the cache sizes, all data will be written to main memory and then read back.

This is not a particularly thorough test, and it depends upon a good cache to function correctly, but it is fast, at least compared to the other full-memory tests.

---

mem11 - User-specified patter/location write/read test(ported from IP17's mem7)

type "mem11" without any arguments to see the usage.

```
Usage: mem11 [-b|h|w] [-r] [-l] [-v 0xpattern] RANGE
```

This test is allows the technician to fill a range of memory with a specified test value and read it back, done as a series of byte (-b), half-word (-h), or word (-w) writes and reads. If the -v option is not used to select the test pattern, an address-in-address pattern is used instead. (-r) will do read only and will not do any writes. (-l) will loop forever.

---

mem12 - Decode a bad address into slot, leaf, bank, simm

```
Usage: mem12 [-a 0xaddress] [-b xxxxx] [-s x]
           -b expects a hex number showing which bits are bad.
              e.g. If bits 1 and 4 are bad, enter: -b 0x5
           -s 1, 2, or 4 for byte, half-word or word
           -b defaults to 0x0 and -s defaults to 4
```

For example, to decode address 0x4000 with bad bits 1 and 4 and it's a word, type:

```
        mem12 -a 0x4000 -b 0x5 -s 4
```

---

mem13 - byte read / write (see mem4) (slow: 15 minutes/32 MBytes)

---

mem14 - Read the mc3 config register

This is the same as the dmc command from pod mode. See also mem1

```
------------------------------------------------------------------------------
mem15 - Double word MarchY pattern test (4 min / 128 MB)

Same as mem8 but does double word writes/reads instead of word writes/reads.

------------------------------------------------------------------------------
ena_bnk, dis_bnk - enable / disable one bank at a time

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

5.2.1 Error Message Syntax

Everest hardware errors are displayed following UNIX kernel panics, in the IDE standalone diagnostics, and in some of the PROM-based power-on tests. The display format of the error messages is referred to as the "HARDWARE ERROR STATE," and is defined as follows:

- Only bits indicating an error has been detected are displayed. "Normal" bits are not displayed.
- The display walks through all of the boards in the system and through every ASIC on each board.
- A HARDWARE ERROR STATE display consists of the banner line "HARDWARE ERROR STATE:" followed by indented lines prefixed by a "+" sign. Line indentation, from left to right, indicates the board, the ASIC, the register, and the bit (see the following example).

```
HARDWARE STATE:
+   IP19 in slot 1                      (CPU Board)
+     CC in IP19 slot 1, CPU 0          (ASIC on CPU Board)
+       CC ERTOIP Register: 0xffff      (Register in the CC and its hex value)
+         Parity Error on TAG RAM Data  (Bit in the register that is set)
```

- Each error register's value is shown in hexadecimal, followed by a line for each bit set.
- Each board identifies its location with its board slot number. Each ASIC identifies its location with some address information; a CC by the CPU it is associated with, the EPC, F chip, or S chip by its Ibus adapter number.

  Note: The F chip also identifies the ASIC that is at the other end of its flatcable.
- The decimal bit number precedes the name of each error bit.
- Some registers have multi-bit values and are displayed in hexadecimal, rather than as bits.

The kernel will panic in response to many possible hardware errors. The HARDWARE ERROR STATE messages allow you to trace the error back to the ASIC that originally detected the fault, thereby identifying the FRU to be replaced. Relate the error message to a block diagram of the system, and walk the propagated errors backwards to determine where the fault originated.

As an example, assume that a driver, executing on an IP19 CPU, attempts to read a control register on a VMEbus device. If the controller fails to respond, the VMEbus will timeout.

The timeout causes the VMECC to record "VME Bus Error on PIO Read." The VMECC will return an error message to the F ASIC. From the F ASIC, the error passes through the IA ASIC, the A ASIC, the CC chip, and finally reaches the CPU as a bus error. Each of the ASICs in this sequence may record the error. The kernel will panic and dump all of the set error register bits. You must understand the possible error propagation paths throughout the machine in order to distinguish the secondary, propagated errors from the origin of the fault.

5.3 IRIX Error Reporting

This section describes the various types of UNIX kernel messages displayed by the console. These messages may also appear in */usr/adm/SYSLOG*, where they are prefixed by "*<systemname> unix:*." Not all kernel messages appear in the *SYSLOG* file because a daemon must be running to transfer the error message from the kernel to the file. If the system panics, the kernel messages only appear on the console.

There are three types of kernel messages:

- Panic messages
- Warning messages
- Driver messages

5.3.1 Panic Messages

The panic message syntax is: **`PANIC CPU`** *n: xxx*, where "n" is the processor number and "xxx" is the string indicating the general area of the fault. The kernel "panics" when it cannot continue operation without the risk of corrupting user data. Common causes of kernel panics are:

- Detecting problems in the kernel data structures.
- Processor exceptions taken during kernel code, resulting from a software bug.
- Processor exceptions taken during kernel code, resulting from a bus timeout when hardware doesn't respond to a PIO operation (such as a read/write to a control register).

5.3.2 Warning Messages

The warning message syntax is: **`WARNING: CPU`** *n xxx*, where "n" is the processor number and "xxx" is the string indicating the general area of the fault. Warnings often result from the kernel nearly running out of some resource, indicating that a kernel software configuration change is needed.

5.3.3 Driver Messages

The driver message syntax is: *dddn: xxxx,* where "ddd" is a two- or three-character string indicating the driver name, "n" is a number indicating the controller, and "xxxx" is the string indicating the general area of the fault. These messages are sometimes embedded inside a warning message. Driver messages are generally hardware specific and will not directly cause a kernel panic.

An example of a message from the SCSI driver is: `dks0d1s6: invalid partition`. Where "dks" identifies the SCSI driver, "0" indicates which SCSI bus, and "d1s6" identifies which drive and partition.

Chapter 6

Diagnostic Procedures

6.1 Overview

This chapter describes the various diagnostic tools available to manufacturing and field service. The scenario of a frozen system is provided to demonstrate how the debugging tools are used.

6.2 Examining a Frozen System

This section provides a step-by-step approach to diagnosing the cause of a frozen system.

1. Examine the System Controller log.
2. Check the voltages, blower operation and other physical parameters if indicated.
3. Turn the key switch to the Manager position, select the "Backplane NMI" menu, and execute a non-maskable interrupt. The NMI should generate a core dump.
4. Reset the system and start UNIX. The operating system creates core files in */usr/adm/crash*, which can then be analyzed to determine the cause of the crash.
5. If the system comes back up, look in *usr/adm/SYSLOG* for messages from the system controller daemon. The message syntax is: **`sysctlrd: event: xxxx`**.

Note: Whenever the system is power-cycled, or UNIX is brought up, the contents of the System Controller error log are dumped to *usr/adm/SYSLOG*.

6.3 Troubleshooting

This section is a loose grouping of various troubleshooting/debugging tips. Some of the methods described require equipment that is not readily available in the field, and are better suited for use by manufacturing or a repair depot.

6.3.1 IP19 Troubleshooting Procedures

This section describes common problems with the IP19 board and their solutions.

<u>Power bricks operating within acceptable voltage levels?</u>

Note: Before starting to troubleshoot the board, verify that both the 3.3V and 5.0V bricks are working within acceptable levels. If the 5.0V brick fails while the 3.3V brick continues to work, the ASICs will overheat.

1. Check the power fault LEDs on the board (refer to Figure 2-3).
2. If the fault LEDs are off, but the bricks are still suspect, check the voltages at the red and black test points at the back of the board.

Note: If the 5.0V brick has failed, it will still show approximately 2.5V, due to the 3.3V brick pulling up through the ASICs.

<u>Power levels are within the specified ranges, but the processor LEDs are all off.</u>

1. There is a power or ground short.
2. The LED controller PAL has failed.
3. One or more of the R4KIO PALs has failed. Check the PALs on the failing slices. The PALs are at locations I5H1, J1M0, I3J9, and F2E1 for slices 0 through 3, respectively.

<u>Power levels are within the specified ranges, but the processor LEDs are all on.</u>

The processors are not booting from their EAROMs and PROMs.

1. Check for missing clocks.
2. Check for bad PROMs or PALs.
3. Check for a dead processor.

<u>All processor slices have failed.</u>

The clocks may not be running. Check the clocks as follows:

1. Look for a 50MHz ECL-level clock at pin F8.
2. Check the MC100E11 backplane clock driver at G0C8.
3. Check that the MC100E11 power supply is at 4.5V.
4. Look for a 50 or 75Mhz TTL-level clock at pin V4.
5. If no clock is present, check the local crystal at G9K7 or H5K7.
6. Check the driver at G9J8. Clock pulses should be present at pins 12, 14, 16 and 18.

<u>A single processor slice has failed.</u>

1. Look for a clock at pin V4 of the failing slice's CC chip.
2. If not clock is present, check the R4k Reset PAL. The PALs are located at F9G5, G1K7, I3J9, and I7G5 for processor slices 0 through 3, respectively.

Note: If the clock is running at 25% frequency, it is probable that the EAROM is faulty.

3. Check that the processor is securely seated in the socket. Remove the heat sink and check for snugness.

Clocks are good, but the processor LEDs are all on.

1. Check the R4kIO PAL, EAROM, and processor, as noted previously.
2. Replace the EPROM and check the board again.
3. Check the frequency of the ModeClk at N19. The frequency should be approximately 200KHz. If no ModeClk signal is present, the processor has failed.
4. Check the reset line (pin 20) to the CC chip.
5. If the reset line does not pulse when the SCLR line is enabled, check pin 17 on the A chip.
6. If the A chip line doesn't pulse either, check for a failure in the System Controller.

LEDs begin to go through boot pattern, but registers a failure code before the boot sequence completes.

This procedure assumes that the processor is basically functional. It can fetch and execute EPROM instructions, but may be having trouble reaching the bus or ASICs beyond the CC chip.

1. Record the binary LED value and refer to Section 4.5 for a description of the fault.
2. Check the UART output.

The Enable Register is incorrect.

1. From a working board in the same system, read register 0 on the board under test. This is done by using the POD command **dc** *<slot number>* **0**. Register 0 contains the ENABLE vector, which represents the number of processors populating the board (0x3=2 CPUs, 0xf=4 CPUs).
2. If the register value does not match the number of occupied slices, replace the CC_SHARED PAL at G0C0.

LEDs show a static 0xe pattern.

The serial clock is not running.

1. Look for approximately 100KHz at pin 20 on the CC chip.
2. If no clock is present, check the System Controller for serial clock generation.

LEDs show a static 0xc pattern.

System is stuck in bootmaster arbitration. Possible cause is a CC clock problem.

1. Replace the CC chip.
2. Replace the EAROM.

LEDs complete boot sequence and display master/slave patterns, but UART output is garbled or missing.

1. Check UART cable and connections.
2. Verify that the serial clock is matched to the UART speed. When connecting to the UART, match the speed with the System Controller speed.

Note: The standard System Controller produces a 9600 Baud clock.

6.3.2 IO4 Troubleshooting Procedures

If an IO4 failure occurs during the boot process, the error message will continue to scroll across the System Controller display until the system is powered off. Clear the display by first turning the key switch to the Manager position. Enter the Debug Settings menu and set the "Manu-mode" bit. Setting this bit sends the IP19 PROM error messages to the external UART on the System Controller (see Section 6.3.3.1).

6.3.3 Using the System Controller

This section contains various troubleshooting procedures utilizing the System Controller.

6.3.3.1 Systems with Dead Monitors or Terminals

This workaround provides a method for connecting an ASCII terminal to a system with a faulty IO4 board, bad monitor or bad terminal.

The workaround is based on the System Controller's connection to all of the CC chips on the lowest-numbered IP19 board, over the Polled Serial Bus. The Polled Serial Bus is composed of six address lines and two data lines, and is used primarily during the bootmaster arbitration process. The System Controller can address all of the processors through their respective CC chips, but only the bootmaster CPU is capable of responding.

There is a port (or UART) tied directly to the System Controller. A terminal can be attached to this port and used to reach the CPU board by going through the controller and over the Polled Serial Bus. On the rack-mounted systems, the System Controller UART is located in the lower left corner of the midplane (when facing the front of the chassis). Deskside systems have the UART located in the lower, right corner of the backplane (when facing the rear of the chassis). The port is labelled **`External Controller Serial`** on all systems.

Note: The System Controller UART is equipped with a permanently attached cable, with a DB-25 connector at the terminal end.

Select the Debug Settings menu and toggle bit 7 (the Manu-mode bit) to select the System Controller UART. Refer to Section 3.5.2 for more information on the Debug Settings menu.

6.3.3.2 Communicating with the System Over the System Controller Port

These are a general collection of commands that can be initiated over the System Controller port.

Get the selected processor out of slave mode – Type **`control-x s`** *u z* **`control-y`** (with no spaces), where **`control-x s`** begins the select command, *u* indicates the IP19 slot number, *z* specifies the slice number, and **`control-y`** executes the command.

Select a specific processor to communicate with – Type **`select`** *x*, where *x* is the processor slice. The new prompt will `take the form` **`POD`** *xx/yy***`>`**, where *xx* indicates the slot number and *yy* indicates the slice.

Put all of the processors in a selected slot into POD mode – Type **`<cntrl> p`**.

To exit this mode – Type **`reset`** to return to the PROM Monitor.

To cycle system power – Type **`control-x`** **`c`** **`control-y`** (with no spaces and no return).

To reset the system – Type **`control-x`** **`r control-y`** (with no spaces and no return).

6.3.3.3 Defeating the System Controller

Defeat the System Controller when a dead controller or a bad sensor is suspected.

Cycling the key switch while pressing the Execute button allows the System Controller to come up without starting the power-on sequence. This is valuable if an error such as a power fault generates a repeating message on the display. The error log can then be checked and the voltage protection turned off, using the Debug Settings menu (refer to Section 3.5.2).

Return to the default debug settings by simultaneously pressing the Menu and Scroll Down buttons while cycling the key switch.

Cycle the key switch to return to normal controller operation.

6.3.3.4 Communicating With a Disabled Processor

When a processor fails, it is disabled by the system. Because a disabled processor is unable to talk to the system bus, IDE cannot be used to diagnose the cause of the fault. Enable the processor by first turning the key switch to the Manager position. Select the debug settings menu and set the "No Diagnostics" bit. Power-cycle the system to activate the change to the debug settings menu and enable the faulty processor. See Section 3.5.2 for additional information on the debug settings menu.

6.3.3.5 Changing the System's Serial Number

If a System Controller must be replaced, the serial number of the replacement controller must be set to the value of the original. This is necessary in order to run programs that use the serial number as a unique machine identifier (such as NetLS).

Change the system's serial number as follows:

1. Turn the key switch to the Manager position and select the Debug Settings menu.
2. Set the PROM Debug Mode bit.
3. Enter the Command Monitor.
4. Type **`serial`** to display the system's serial number.
5. Type **`serial`** *xxxx*, where *xxxx* is the serial number of the original System Controller.

6. Type **serial** again and verify the new serial number.
7. Return to the Debug Settings menu and turn off the PROM Debug Mode bit.