HEWLETT® PACKARD

S-Class and
X-Class Servers

Exemplar Diagnostics Guide

First Edition

Hewlett-Packard Company
Convex Division
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America

Exemplar Diagnostics Guide

S-Class and X-Class Servers

A4716-90002

First Edition

January, 1997

Hewlett-Packard Company
Convex Division
Richardson, Texas
United States of America

Exemplar Diagnostic Guide
S-Class and X-Class Servers

A4716-90002

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

This entire book is recyclable.

Printed in the United States of America

Revision Information for

Exemplar Diagnostic Guide

S-Class and X-Class Servers

Edition	Document No.	Description
First	A4716-90002	Initial release January, 1997.

Contents

Preface ..... **xiii**
Purpose xiii
Using this book xiii
Notational conventions xiv
Notes and cautions xiv
Associated documents xiv
Ordering documents xv
Technical assistance xv

1 Introduction **1**
Utilities board 1
Core logic 4
Flash memory 4
Nonvolatile static RAM 4
DUART 4
RAM 4
Console ethernet 4
LEDs and LCD 5
COP interface 5
PUC 5
MUC and Power-on 5
MUC environmental monitoring 6
Environmental condition detected by power-on function 6
Environmental conditions detected by MUC 7
Environmental control 7
Power-on 7
Voltage margining 8
Clock margining 8
JTAG interface 8
Test station interface 8
AC test of a hypernode 8
JTAG fanout 9
System Displays 10
Front panel LCD 10
LED environmental display 10
ECUB 3.3-Volt error 12
ASIC installation error 12

DC OK error 12
48-volt error 12
48-volt yo-yo error 12
Clock failure 13
FPGA configuration and status 13
Board over-temperature 13
Fan sensing 13
Power failure 13
Midplane power failure 13
48-volt maintenance 13
Ambient air sensors 14

2 Configuration management..... 15
Test station 15
ccmd 15
ccmu 17
xconfig 20
Menu bar 22
Node configuration map 23
Node control panel 24

3 Power-On Self Test..... 27
Overview 27
Reset 28
Power up reset 28
Hard reset 28
Soft reset 28
TOC 28
POST modules 29
Node initialization 29
Processor initialization and selftest 29
Checksum verification of NVRAM 30
Core logic initialization 30
node configuration determination 30
node ASIC initialization 30
node main memory initialization 30
node clean up and OBP boot process 31
dcm script 33

4 Test controller..... 37
Test controller modes 37
User interface 38
Main menu 39
Test Configuration menu 45

5 cxtest 53

Overview 53
Graphics interface 54
Menus 54
File menu 54
Global Test Parameters menu 55
Command menu 56
System Configuration menu 56
Help menu 57
Test menu 57
Display area 58
Powering down the system 59
Command line interface 59
Command line options 60
Loading options 60
Tests 60
Parameters 61
Looping/pause/control 61
Class and subtest selections 62
Test output 62
Speed ups 63

6 mem3000 65

Introduction 65
Classes and subtests 65
Classes 66
Subtests 66
User parameters 68
Using mem3000 68
Configuration 68
Selecting classes and subtests 71
Starting tests 73
Viewing the results 74
Error codes 75

7 Utilities 81

sppconsole 81
hard_logger 82
pce_util 83
load_eprom 86
load_mem 89
tc_show_struct 90
RDR dump utilities 92
Version utilities 92

diag_version 92
flash_info 93
ver 93
Event processing 94
event_logger 94
event_browser 97
log_event 99
Miscellaneous tools 100
ds1620 100
ecc_calc 101
kill_by_name 101
cbus 102
fix_boot_vector 102
stop_on_hard 102

8 Excalibur scan test. 105
est utility test environment 105
Running est 105
est tests 108
est commands 108
AC Connectivity test 108
Bypass test 108
DC Connectivity test 109
Gate Array test 109
SCI test 110
JTAG Identification test 111
Margin commands 111
Miscellaneous commands 111
Flags and options 111
Script files 112

Figures

Figure 1 Hypernode Utilities board 3
Figure 2 Front panel LCD 10
Figure 3 `xconfig` window 21
Figure 4 `xconfig` window menu bar 22
Figure 5 `xconfig` window node configuration map 23
Figure 6 `xconfig` window node control panel 24
Figure 7 POST program flow 32
Figure 8 `dcm` sample output part A 34
Figure 9 `dcm` sample output part B 35
Figure 10 `dcm` sample output part C 36
Figure 11 Main menu 39
Figure 12 Main menu - Global Parameters display 41
Figure 13 Main menu - Processor Summary display 42
Figure 14 Test Configuration menu - Display processor Errors 43
Figure 15 Main menu - Test Selection menu 44
Figure 16 Main menu - Debugging menu 44
Figure 17 Test Configuration menu 45
Figure 18 Test Configuration menu - Class display 46
Figure 19 Test Configuration menu - Subtest display 46
Figure 20 Test Configuration menu - Test Parameters 47
Figure 21 Test Configuration menu - Test Parameters display 49
Figure 22 Test Configuration menu - Hardware Selection menu 49
Figure 23 cxtest menu 54
Figure 24 cxtest Global Test Parameters menu 55
Figure 25 System configuration window 57
Figure 26 Test Class Selection menu 58
Figure 27 Selecting Test Selection option from TC main menu 69
Figure 28 Selecting Memory test from Test Selection menu 69
Figure 29 Selecting Memory test from Test Selection menu 70
Figure 30 Selecting CPUs from Hardware Selection menu 70
Figure 31 `mem3000` Subtests menu 72
Figure 32 Example of mem3000 execution 74
Figure 33 Example CPU summary display 74

Figure 34 Error message format one 77
Figure 35 Error message format two 78
Figure 36 Error message format three 78
Figure 37 Error message format four 79
Figure 38 Example of `pce_util` with `-n 0` options set ... 84
Figure 39 Example of `pce_util` with `-n 0 -p n` all options set 85
Figure 40 Example of `pce_util` with `-n 0 -c s u` options set 85
Figure 41 Example of `pce_util` with `-l` option set 86
Figure 42 Example of `load_eprom` output 88
Figure 43 Example of tc_show_struct output 91
Figure 44 Example of flash_info output 93
Figure 45 32-bit event code format 95
Figure 46 Output from `stop_on_hard` with `chk` option 103
Figure 47 Example of `est` output 107

Tables

Table 1	Environmental conditions monitored by the MUC and power-on circuit	6
Table 2	Environmental LED display	11
Table 3	ccmu commands	18
Table 4	Logical and Hardware Names	19
Table 5	POST Boot Selections	40
Table 6	Processor States	42
Table 7	Parameter Defaults	48
Table 8	Command line loading options.	60
Table 9	Test Selection available in EEPROM	60
Table 10	Looping, pause, and control options	61
Table 11	Class and subtest selections	62
Table 12	mem3000 test classes	66
Table 13	`mem3000` subtests	66
Table 14	`mem3000` test parameters	68
Table 15	Test patterns for subtests 230 through 238	73
Table 16	`mem3000` error codes	75
Table 17	Extended range for error codes	77
Table 18	Error message field description for error format one	77
Table 19	Error message field description for error format two	78
Table 20	Error message field description for error format three	78
Table 21	Error message field description for error format four	79
Table 22	`pce_util` options	83
Table 23	`load_eprom` options	87
Table 24	`load_mem` utility options	89
Table 25	Values for the preamble field in the eventcode	95
Table 26	Values for type field in eventcode	96
Table 27	Values for the subsystem field in the eventcode	96
Table 28	`event_browser` options	97
Table 29	`kill_by_name` options	101
Table 30	`cbus` options	102
Table 31	`est` command line options	106
Table 32	AC Connectivity test options	108
Table 33	DC Connectivity test options	109

Table 34 Gate Array test options 109

Preface

Purpose

The *Exemplar Diagnostics Guide for S-Class and X-Class Servers* documents the diagnostics for these systems. This book is not intended to be a tutorial, but rather a reference for field service and manufacturing personnel.

Using this book

This book has the following eight chapters including:

- Chapter 1, "Introduction," provides an introduction to Exemplar S-Class and X-Class diagnostics. The node Utilities board is discussed in detail.
- Chapter 2, "Configuration management," discusses the `ccmd` and `ccmu` deamons.
- Chapter 3, "Power-On Self Test," describes how POST initializes a hypernode and handles power up errors.
- Chapter 4, "Test controller," describes an EEPROM-based utility that provides the environment for executing the off-line diagnostic tests.
- Chapter 5, "cxtest," discusses a graphical front end and a command line interpreter for the test controller that runs on the test station.
- Chapter 6, "mem3000," details the memory test module called `mem3000`.
- Chapter 7, "Utilities," describes several utilities and tools to enhance diagnostic testing.
- Chapter 8, "Excalibur scan test," details a diagnostic utility (`est`) that uses the system scan hardware making it possible to perform connectivity tests and to test gate array internals.

Notational conventions

This section discusses notational conventions used in this book.

`Monospace`

The `monospace` font identifies command names, system calls, and data structures and types.

In command and code examples, `monospace` identifies command output, including error messages

Italic

In paragraph text, *italic* identifies new and important terms and titles of documents.

Bold

The **bold** character format indicates special emphasis.

Notes and cautions

This document presents notes and cautions in the following formats.

Note

A Note highlights supplemental information.

Caution

A Caution highlights information necessary to avoid damage to the system.

Associated documents

Using Exemplar S-Class or X-Class servers may require information not specific to the functionality described in this document.

- *PA-RISC 2.0 Architecture.* This manual contains generic information about the PA-RISC architecture used as a basis for the S-Class and X-Class servers.
- *Exemplar Architecture: S-Class and X-Class Servers,* part number A4716-90001, provides details of these systems.

Ordering documents

To order the current edition of these or any other Hewlett-Packard documents, send requests to:

Hewlett-Packard Company
Convex Division
Customer Service
P.O. Box 833851
Richardson TX 75083-3851 USA

Please include the order number (DSW or DHW number) or the exact title of the document.

Technical assistance

If you have questions that are not answered in this book, contact the Hewlett-Packard Convex Technical Assistance Center (TAC) at the following locations:

Within the continental U.S., call 1 (800) 952-0379.

From Canada, call 1 (800) 345-2384.

All other locations, contact the local Convex Division office.

You can also use the `contact` utility, if you would like to report any problems you may have.

Introduction

1

This chapter presents an overview of the diagnostic mechanism for S-Class and X-Class servers.

Utilities board

Diagnostics in S-Class and X-Class servers are centered around the hypernode Utilities board which connects directly to the midplane board. Attached to the Utilities board are the core logic bus, environmental sensors, and other test points. The board also interfaces to the hypernode liquid crystal display (LCD), a test terminal (via an ethernet connection), and other external devices. The terminal is connected via these links to configure the hypernode and run diagnostics.

Figure 1 shows the Utilities board functional layout.

The Utilities board is comprised of the following hardware components:

- Core logic—Contains initialization and booting firmware.
- Monitor Utilities Chip (MUC)—Collects environmental interrupts.
- Processor Utilities Chip (PUC)—Interfaces to the core logic bus.
- Power-On circuit—Controls powering up the entire hypernode.
- JTAG (Joint Test Action Group) interface—Supports a test station and a mechanism to fanout JTAG to all the boards in a hypernode. It is used only for testing.

The MUC latches hypernode interrupts, most of which are from environmental sensors located throughout the hypernode. The MUC and the power-on circuit together control system power-up. The MUC interfaces to a light-emitting diode (LED) diagnostic display through the power-on circuit.

The PUC provides the core logic with an interface to the core logic bus. There are actually two buses, each one connects up to four Processor Agent Controllers (PACs). The PUC communicates to the PACs using data packets.

S-Class and X-Class servers use a test method called scanning to test boards and other hardware units. With the test station connected to the ethernet between hypernodes, you can scan any hypernode in a multiple hypernode system without moving connectors.

The JTAG interface contains a microprocessor to capture packets from the ethernet and apply them to the JTAG test bus controller or to take scan information from the JTAG test bus controller and send it out on the ethernet. The test station can also read and write every CSR in the hypernode.

The S-Class and X-Class systems test station is an HP-712 workstation that attaches to the system via ethernet. It is required for installing a multiple hypernode system and to service single and multiple hypernode systems. A system will boot and operate without a test station, and failure of the test station will not cause interruption of the system.

Progress/status messages are also displayed to the LCD during the boot process.

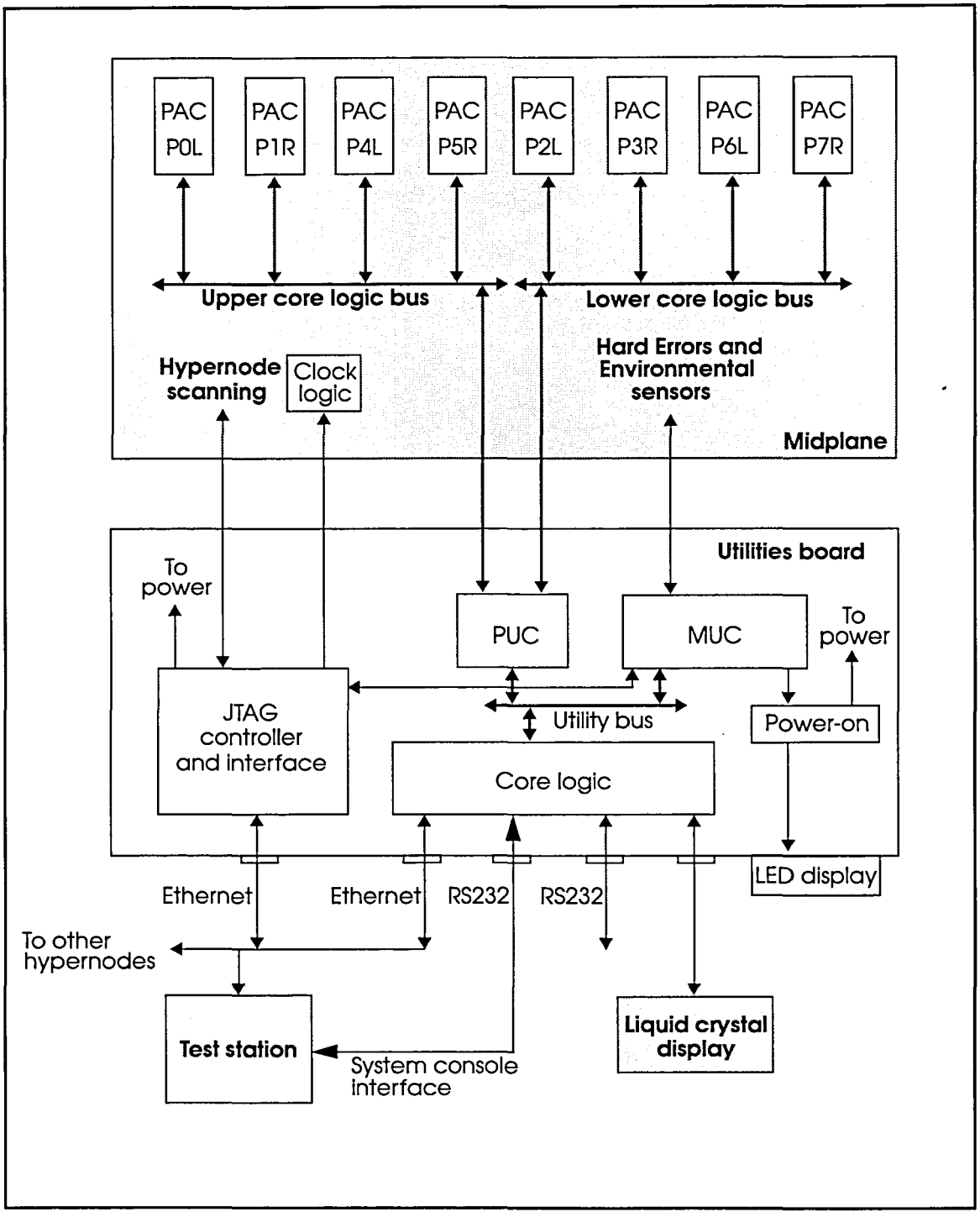

Figure 1 Hypernode Utilities board

Core logic

The core logic contains initialization and booting firmware and is described in the following sections.

Flash memory

The core logic contains nonvolatile storage for processor-dependent code. This code consists of primary loader code, the Open Boot PROM (OBP) code, and power-on self test software. This EEPROM memory is 4 MBytes, configured as 512-Kbyte addresses by 32 data bits with only 32-bit read and write accesses allowed. It is writable by the processors for field upgrades and can be written when the PUC is *scanned.*

Nonvolatile static RAM

The core logic section contains a nonvolatile battery-backed RAM (NVRAM). The NVRAM is used to write system log information and store configuration information. This RAM is byte addressable and can be accessed even after power failures occur.

DUART

The utilities logic contains a Dual Universal Asynchronous Receiver-Transmitter (DUART). One port, configured as a basic RS232 port, provides an interface to the simplest core system functions. With this interface, you can connect a terminal as a local console to analyze problems, reconfigure the system, or provide other user access.The parallel port of the DUART drives the LCD. The second RS232 port can be used for a modem for field service.

RAM

RAM is needed to support the simple core system functions. When the hypernode powers up, the processors operate out of this RAM. They run self test software to test and configure the rest of the hypernode. Once the system is fully configured, the processors execute out of main memory. The RAM is byte addressable and is 128 KBytes, configured as 32K addresses by 32 data bits (with parity).

Console ethernet

The ethernet I/O port connects to the test station over LAN1.

LEDs and LCD

Light Emitting Diodes display environmental information, such as the source of an environmental error that caused the Utilities board to power down the hypernode.

The LCD is driven by one of the processors via the Utilities board. A large amount of information can be displayed on the LCD. The core logic drives the LCD via the parallel port on the DUART.

COP interface

COP chips (serial EEPROMs) are located on the major boards with information such as serial number, error history, configuration information, and so on. The MUC connects to the COP bus selector (CBS) chip on the midplane and allows the system to read any COP in a hypernode.

PUC

The PUC provides the Utilities board a means to apply interrupts and error messages to the processors and to receive control messages from the processors. It has two 18-bit, bidirectional buses. Each interface connects up to four PACs. The PUC provides core logic bus arbitration for the sixteen hypernode processors.

Through the PUC, the PAC has an interface to the utilities bus on the Utilities board. This bus connects the PUC, the MUC, and the core logic section together.

MUC and Power-on

The MUC performs all environmental monitoring on the Utilities board. It attaches to the utilities bus so that processors can monitor the hypernode by accessing these CSRs.

The MUC works in conjunction with a hardware section on the Utilities board known as the power-on circuit. This circuit controls powering up the entire hypernode. It is able to operate when the rest of the hypernode is powered off or in some indeterminate state. It drives the environment LED display which is a basic (minimal hardware, no software) indication of what environmental error caused the Utilities board to power down the hypernode.

The test station can also read the environmental LED display using the `pce_util` utility. See the "pce_util" section on page 83.

MUC environmental monitoring

The MUC and the power-on circuit monitor the following environmental conditions:

- ASIC installation error sensing
- FPGA configuration and status
- Thermal sensing
- Fan Sensing
- Power failure sensing
- 48-V failure
- 48-V maintenance
- Ambient air temperature sensing
- Power-on

Table 1 Environmental conditions monitored by the MUC and power-on circuit

Condition	Type	Action
ASIC Not Installed OK	Environmental error	Power not turned on, LED indication
FPGA not OK	Environmental error	Power not turned on, LED indication
48-V Fail	Environmental error	Power turned off, LED indication
Midplane power fail	Environmental error	Power turned off, LED indication
Board over temp	Environmental error	Power off in one second, LED indication, Interrupt
Fan not turning	Environmental error	Power off in one second, LED indication, Interrupt
Ambient air hot	Environmental error	Power off in one second, LED indication, interrupt
Other power fail	Environmental error	Power off in one second, LED indication, interrupt
Ambient air warm	Environmental warning	LED indication, interrupt
48-Volt maintenance	Environmental warning	LED indication, interrupt
Hard error	Hard error	LED indication, interrupt

Environmental condition detected by power-on function

The power-on function detects environmental errors (such as ASIC Install or FPGA Not OK) immediately and does not turn on power to the hypernode until the conditions are corrected. It also detects environmental errors such as 48-V Fail while the system is powering up and Midplane Power Fail after the system has

powered up. If a failure is detected in these two cases, the power-on circuit turns off power to the system.

Environmental warnings such as 48-Volt Maint are also detected by the power-on circuit. It applies these to the MUC which then sends an environmental warning interrupt to the hypernode processors.

In all cases, the power-on circuit lights an environmental LED display code. The environmental LED display code is prioritized so that it only displays the highest priority error or warning. See the "LED environmental display" section on page 10 for a list of codes. For more information, use `man LEDs` on the test station.

Environmental conditions detected by MUC

The MUC detects most of the environmental conditions. It samples error conditions during a time period derived from a local 10-Hz clock that drives the power-on circuit. It registers all the environmental error conditions twice and then logically ORs them together. If the conditions persist for 200 mS, the environmental error bit is set, and an environmental error interrupt is sent to the PUC, which sends it on to the processors. The MUC then waits 1.2 seconds and commands the power-on circuit to power down the system.

This same procedure exists for an environmental warning, except that an environmental warning interrupt is sent and the power-on circuit does not power down the system.

The environmental error interrupt and the 1.2 second delay provide the system adequate time to read CSRs to determine the cause of the error, log the condition in NVRAM, and display the condition on the LED. Use the `pce_util` utility to read the LEDs. See the "pce_util" section on page 83.

After the system is powered down, the Utilities board is still powered up, but all outputs are disconnected from the system.

Environmental control

The Utilities board performs the following functions to control the hypernode environment.

Power-on

When the power switch is turned on, the outputs of the 48-volt power supplies become active. Several hundred milliseconds after the Utilities board 5-volt supply reaches an acceptable level, the power-on circuit starts powering up the other DC-to-DC

converters of the hypernode. It does not turn on all converters at the same time. Instead, the converters are powered-on in succession.

The power-on circuit does not power up the hypernode if an ASIC is installed incorrectly (see the "ASIC installation error" section on page 12) or if an FPGA is not configured (see the "FPGA configuration and status" section on page 13). It keeps the system powered up unless an environmental condition occurs that warrants a power-down.

Voltage margining

Voltage margin is divided into four groups to minimize control, but allows all boards that communicate with each other to be margined separately for nominal, upper, and lower voltage.

Clock margining

Parallel ports on the core logic microprocessor select the nominal, upper, or external clock that drives the hypernode.

JTAG interface

The JTAG interface supports a test station and a mechanism to fanout JTAG to all the boards in a hypernode. It is used only for testing.

The JTAG functions are described in the following sections.

Test station interface

The test station can be a PA-RISC based workstation or laptop computer. The interface to the test station is an ethernet AUI port for flexibility in connecting to many workstations and laptops. It is also easily expandable to all the hypernodes in a single wall.

AC test of a hypernode

An AC test is performed by a Test Bus Controller (TBC) scanning in data to all boards in a hypernode and loading an `AC Test` instruction into all ASICs on one board.

Once all boards have been almost loaded with the AC test instruction and paused, the TBC takes all boards out of pause mode simultaneously causing them all to exit update together and execute the AC test.

The AC test enables clocks inside the ASICs so that they test internal and external paths at the system clock rate. They all execute on the same system clock.

JTAG fanout

The test station interface is thin ethernet. This port is also used for console ethernet. There is one cable that connects to all the hypernodes and to the test station (if it exists) and to whatever device or network that will display the console.

System Displays

The Exemplar S-Class and X-Class servers provide the two displays for status and error reporting: an LCD and LEDs.

Front panel LCD

The front panel is a 20-character by 4-line liquid crystal display as shown in Figure 2.

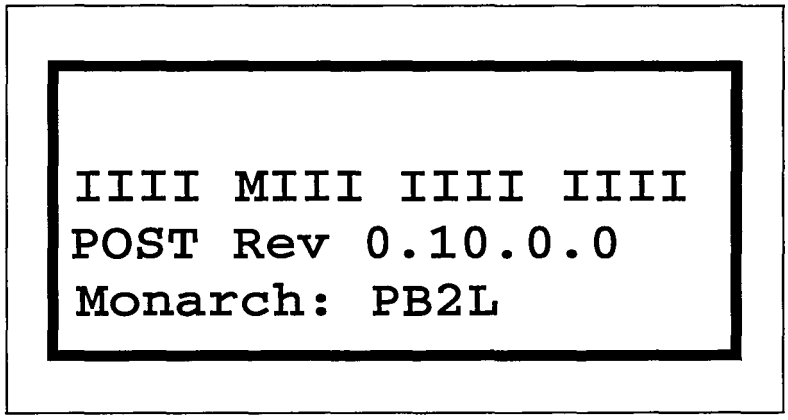

Figure 2 Front panel LCD

When the node key switch is turned on, the LCD will power up but will initially be blank.

POST takes about 20 seconds to start displaying output to the LCD. The following illustrates this output shown in Figure 2:

- First row is blank
- Second row displays the processor status
- Third row displays the revision of POST
- Last row displays which processor is in control
- POST is described in Chapter 3

LED environmental display

Second-level registers in the MUC drive the 6-bit LED display. The MUC prioritizes the environmental errors and warnings, and it passes the information to the power-on circuit. This circuit prioritizes the 6-bit field with its environmental conditions and produces a 7-bit field plus an attention bit (ATTN) that drives the LED display. ATTN is on if there is an environmental warning.

In general, the power-on-detected errors are a higher priority than MUC-detected errors, the lower the error code number, the higher

its priority. Environmental warnings are lower priority than the environmental errors. Table 2 shows the LED display error codes.

Table 2 Environmental LED display

ATTN bit	LED display	Description
1	00	ECUB 3.3-V error (highest priority)
1	01	ASIC Install 0 (ENRB)
1	02	ASIC Install 1 (MEM)
1	03	FPGA not OK
1	04-07	DC OK error (UL, UR, LL, LR)
1	08-11	48-V error, NPSUL fail, PWRUP=0-9
1	12-1B	48-V error, NPSUR failure, PWRUP=0-9
1	1C-25	48-V error, NPSLL failure, PWRUP=0-9
1	26-2F	48-V error, NPSLR failure, PWRUP=0-9
1	30-39	48-V error, no supply failure, PWRUP=0-9
1	3A	48-V yo-yo error
1	3B	ENRB power failure (ENRBPB)
1	3C	Clock failure
1	3D-3F	Not used (3)
1	40-47	MB0-MB7 power failure
1	48-4F	PB0L, PB1R, PB2L, PB3R, PB4L, PB5R, PB6L, PB7R power failure
1	50-57	PB0R, PB1L, PB2R, PB3L, PB4R, PB5L, PB6R, PB7L power failure (possibly switch R and L)
1	58-5B	IOB (LR,LF,RF,RR) power failure
1	5C-61	Fan failure (UR,UM,UL,LR,LM,LL)
1	62	Ambient hot
1	63	Overtemp ENRB
1	64-67	Overtemp quadrant (RL, RU, LL, LU)
1	68	Hard error
1	69	Ambient warm
1	6A-6F	Not used (6)
1	70-73	DC supply maintenance (UL,UR,LL,LR)

Table 2 Environmental LED display—**(continued)**

ATTN bit	LED display	Description
1	74-7F	Not used (12)
0	00-09	PWRUP state (00=System all powered up), attention LED off

The top of the table is the highest priority, the bottom the lowest. If a higher condition occurs, that one is displayed.

ECUB 3.3-Volt error

This error indicates that the ECUB 3.3-volt power supply has failed, but the 5-volt supply has not.

ASIC installation error

Each ASIC in the hypernode has ASIC Install lines to prevent power-up if an ASIC is installed incorrectly (such as a PAC installed in a RAC's position). If an ASIC is improperly installed, the Utilities board does not power up the system. This condition is not monitored after power up.

DC OK error

When this error is displayed, the power-on circuit did not power up the system, because one or more 48-volt power supplies reported an error. In systems with redundant 48-volt power supplies, this error means that two or more 48-volt supplies reported an error.

48-volt error

If the 48-volt supply has dropped below 42 volts for any reason other than normally turning off the system or an AC failure, then this error is displayed by the power-on circuit. Also, the 48-volt supply that reported the error and the power-up state of the system at the time of the error is displayed.

48-volt yo-yo error

This error indicates that a 48-volt error occurred and the ECUB lost and then later regained power without the machine being turned off. The power-on circuit will display this error and not power on the system, because the 48-volt supply is likely at fault.

Clock failure

If the system clock fails, then the MUC is unable to monitor environmental errors that could possibly damage the system. If the power-on circuit receives no response from the MUC, it powers down the system and displays this error.

FPGA configuration and status

The MUC is programmed by a serial data transfer from EEPROM upon utility board power-up. If the transfer does not complete properly, the MUC cannot configure itself and many environmental conditions cannot be monitored. The power-on circuit monitors both the MUC and PUC and does not power up the system, if they are not configured correctly.

Board over-temperature

On each board in the hypernode, there is one temperature sensor that detects board overheating. The sensors are bussed together into four hypernode quadrants plus the midplane and applied to the MUC.

Fan sensing

Sensors in the four fans determine if the fans are running properly. The MUC waits 12.8 seconds for the fans to spin up after power-up before monitoring them.

Power failure

Because a power failure on a board could cause damage to other boards, a mechanism on each board detects 3.3-Volt failures on each board. Power failures are considered environmental errors, and the system is powered down after they are detected.

Midplane power failure

If the midplane power fails, the power-on circuit powers down the entire hypernode. The Utilities board is still active, but the power-on circuit displays the power failure condition and disables all Utilities board outputs that drive the hypernode. This condition persists until power is cycled on the Utilities board.

48-volt maintenance

There are four 48-volt power supplies; three are required, one is a redundant source. Each sends a signal to the power-on circuit. If any supply fails at any time, the circuit asserts the 48-V maintenance line to the MUC, which reports the environmental

warning to the processors. The power-on circuit displays the "highest priority" 48-volt supply that failed.

Ambient air sensors

The ambient air sensors detect a too warm or too hot condition in the input air stream to the Utilities board (and therefore the entire hypernode). Ambient air too warm is an environmental warning; ambient air too hot is an environmental error that powers down the system.

The temperature set points are set using the `ds1620` utility, described in Chapter 7. The digital temperature sensor has nonvolatile storage for the temperature set points. Power-on reset starts the digital temperature sensor without the core logic microprocessor intervening.

Configuration management 2

The test station allows the user to configure the node. The test station daemon, `ccmd`, is responsible for monitoring the node and reporting back configuration information, error information and general status. Two utilities, `ccmu` and `xconfig`, are each capable of reading or writing configuration information and changing it. OBP can also be used to modify the configuration.

Test station

The test station is required in system bring-up, monitoring, testing, and error logging. It is not required for normal operation of a node.

The node JTAG interface connects to the test station.This communications port remains idle if no test station is connected to it. It receives communications packets and interpret requests and generates responses to them. The hardware on the node can read board information, system configuration, device revisions and environmental conditions. When a test station is present, all of these parameters are read or written by the configuration management tools.

The main tool to initiate communications is the complex configuration management daemon, `ccmd`.

ccmd

The communications configuration manager daemon, `ccmd`, builds configuration information database on the test station. The board names and revisions, the device names and revisions, and the start-up information generated by POST are all read and stored in memory for use by other diagnostic tools.

`ccmd` is typically run automatically from /etc/inittab on the test station. If `ccmd` dies, it is restarted if listed in /etc/inittab. It can also be started from the command line by entering:

```
ccmd [-d]
```

Very few of the diagnostic utilities can run without the information database created by ccmd. The debug option, -d, reports on its progress and problems communicating with the node.

ccmd is started by init on the test station. init monitors ccmd and respawns it if it ever dies. Once started, ccmd becomes a daemon and allocates a block of test station memory to be used as a database for all nodes, boards and devices.

Next, ccmd reads the file /spp/data/complex.info to determine what complexes, nodes, and JTAG ports are available to communicate with the test station. ccmd then broadcasts a command to all JTAG ports to report in. As each node responds with its IP address, ccmd sends a request to each node for the node identification number. ccmd continues until no new responses are detected.

Once ccmd has all valid node numbers and IP addresses, it scans each ring of each node looking for JTAG device IDs. The JTAG IDs contain device and revision information stored in the test station database. The JTAG ID is cross checked in the /spp/data/part_ids file to retrieve a complete device description of the part. The file for the part is also in /spp/data.

After ccmd loads all parts and their descriptions into the database, it reads all board information. Board and device information is cross checked in the file /spp/data/DB_RING_FILE. Complex and node configuration files for EST are written to the /spp/data directory. A complex list is written to /spp/data/config for use by Convex Error Response System (CERS) software.

Once all information is in place, ccmd monitors the node for changes in status. ccmd waits for commands from other diagnostic utilities to set or clear certain system status variables. After a 60 second pause, ccmd once again broadcasts to determine which nodes are available. As each node responds, ccmd queries it for error conditions. Also, each node is checked to see if it has just powered up. If all nodes do not respond, then ccmd reports that no nodes are responding.

If no nodes are responding, ccmd clears all node data and waits for a node to respond. If a node powers up, the entire database is rebuilt.

If a node has stopped responding, ccmd clears just that node's data in the database.

If ccmd detects a hard error, it starts the hard_logger script to extract additional information from the node through the JTAG interface.

ccmd sends output to the console. If running under X-windows as sppuser, it sends its output to the test station console message output window. The -d debug option generates a substantial amount of console output.

ccmd does respond to several signals. The sighup signal tells ccmd to rebuild the test station database. A sigint or sigabort signal terminates the ccmd process.

ccmu

ccmu is a text-based tool that can modify the reconfiguration parameters initialized by POST.

ccmu is started by the user from the test station either in command line or interactive mode. Once started, it attaches to the test station database created by ccmd. ccmu also reads the /spp/data/complex.info file to maintain a list of all possible nodes that could be reconfigured.

Note

Cross-check the test station database and the complex.info file with the `system -check` command.

ccmu command line parameters have the following format:

```
ccmu command [[: command].... [: command]]
```

Each command is the same format as those of the interactive ccmu.

In the interactive mode, ccmu is started by the command, **ccmu**, executed at a shell prompt. The following user prompt appears when ccmu is running interactively:

```
ccmu>
```

When a command is issued, ccmu communicates with the system through the JTAG interface to read or write the data and instructions required to accomplish the command.

Note

It is presumed that the user has some knowledge of how POST sets configuration parameters to enable or disable them.

Since scan is a slow operation, data retrieved from the node is kept on the test station. If the data is not present, it is retrieved. If it is present, the local copy is used.

Local data can be scanned back into the node by entering the **push** command. To acquire a fresh copy of node data, enter the **`get -f`** command to force a new copy of the data to be read.

In order to reconfigure a system, the user should use the `rebuild <node number>` defaults command. Table 3 lists `ccmu` command.

Table 3 `ccmu` commands

Command syntax	Command function
`quit`	Leave `ccmu` interactive mode.
`help`	Display the help screen.
`system [-check]`	Check the system configuration or display information on the complex.
`assign <node number> <ENRB number>`	Reassigns the node with the backplane indicated to the new node number.
`assign <node number> <ENRB number> <complex serial number> <complex key>`	Reassigns the complex serial number on the node indicated.
`list <node number> <parameter>`	Displays a list of all POST parameters that match the parameter field. The parameter field may contain wild card characters.
`get [-f|-s] <node number> <parameter>`	Display the value for the parameter specified. If the data is already present on the test station and the `-f` option is not used the local copy is used.
`push <node number>`	Sends the test station data to the node.
`put <node number> <parameter> <value>`	Assigns value to the parameter on the local copy of the node data.
`mnetcache <node number> <net cache size number>`	Assigns the network cache memory size to be the value specified.
`msize <node number> <?|memory size number>`	Deconfigures enough memory to be the memory size. If size is "?" all memory is reconfigured.
`stop_on_hard <node number> [<on>|<off>] chk`	Sets an option bit in all ASICs that stops clocks as soon as an error is detected.
`query <node number> <A | C | M | D number | I | S >`	Displays the current status of the system component specified. For a description of arguments, see text.
`toggle <node number> <A number| C number | M number | D number_number_number | I number | S number>`	Changes the local copy of the current status of the system component specified.
`nodemap <node number>`	Change multi-node configuration map.
`rebuild <all|node number> <default|new>`	Restart POST to enable a new or the default configuration.

In Table 3, the arguments in the Boot Map command are defined as follows:

- `A` or `A number`—Applies to all agents (EPACs) or individual agents 0-7.
  - Agent 0 is also known as p0l.
  - Agent 7 is also known as p7l.
- `C` or `C number`—Applies to all CPUs or individual CPUs 0 through 15.
  - CPU 0 is also known as pb0l.
  - CPU 1 is also known as pb0r.
- `M` or `M number`—Applies to all Memory boards mb0l through mb7l.
- `D number` or `D number_number_number`—Applies to the DIMM on memory board.

  `D` number_number_number is the `<memory board>_<dimm>_<row>`
- `I` or `I number`—Applies to all IO boards, such as iolf_a.
  - 0 is iolf_a
  - 1 is iolr_a
  - 2 is iorr_a
  - 3 is iorf_a
  - 4 is iolf_b
  - 5 is iolr_b
  - 6 is iorr_b
  - 7 is iorf_b
- `SCI` or `SCI number`—Applies to all IO boards, such as iolf_a

  0 refers mb0l_t; 7 refers to mb7l_t

The parameter names associated with POST configuration do not agree with the name physically on the hardware or hardware name. The name used by POST is called the logical name. Table 4 displays typical logical names and the hardware name.

Table 4 Logical and Hardware Names

Logical name	Hardware name
emb0 - emb7	mb0l, mb1l, mb2r, mb3r, mb4l, mb5,l mb6r,mb7r
epac0-epac7	p0l, p1r, p2l, p3r, p4l, p5r, p6l, p7r
cpu0-cpu15	pb0l, pb0r, pb1r, pb1l, pb2l, pb2r, pb3r, pb3l, pb4l, pb4r, pb5r, pb5l, pb6l, pb6r, pb7r, pb7l

Table 4 Logical and Hardware Names—**(continued)**

Logical name	Hardware name
emb0_etac-emb7_etac	mb0l_t, mb1l_t, mb2r_t, mb3r_t, mb4l_t, mb5l_t, mb6r_t,mb7r_t
epic0-epic7	iolf_b, iolr_b, iorr_b, iorf_b, iolf_a, iolr_a, iorr_a, iorf_a
emb0_epmb0_row0 - emb0_epmb15_row1	mb0l_B0S0_row0, mb0l_B0S0_row1, mb0l_B0S1_row0, mb0l_B0S1_row1, mb0l_B0S2_row0, mb0l_B0S2_row1, mb0l_B0S3_row0, mb0l_B0S3_row1, mb0l_B2S4_row0, mb0l_B2S4_row1, mb0l_B2S5_row0, mb0l_B2S5_row1, mb0l_B2S6_row0, mb0l_B2S6_row1, mb0l_B2S7_row0, mb0l_B2S7_row1, mb0l_B1S0_row0, mb0l_B1S0_row1, mb0l_B1S1_row0, mb0l_B1S1_row1, mb0l_B1S2_row0, mb0l_B1S2_row1, mb0l_B1S3_row0, mb0l_B1S3_row1, mb0l_B3S4_row0, mb0l_B3S4_row1, mb0l_B3S5_row0, mb0l_B3S5_row1, mb0l_B3S6_row0, mb0l_B3S6_row1, mb0l_B3S7_row0, mb0l_B3S7_row1,

xconfig

`xconfig` is the graphical tool that can also modify the parameters initialized by POST to reconfigure a node.

The graphical interface allows the user to see the configuration state. Also the names are consistent with the hardware names, since individual configuration parameters are hidden to the user. The drawback of `xconfig` is that it can not be used as a part of script-based tests, nor can it be used for remote debug.

`xconfig` is started from a shell as `ccmu` is. It reads the test station database and the possible nodes from /spp/data/complex.info.

Information on node 0 is read and interpreted to form the starting X-windows display shown in Figure 3.

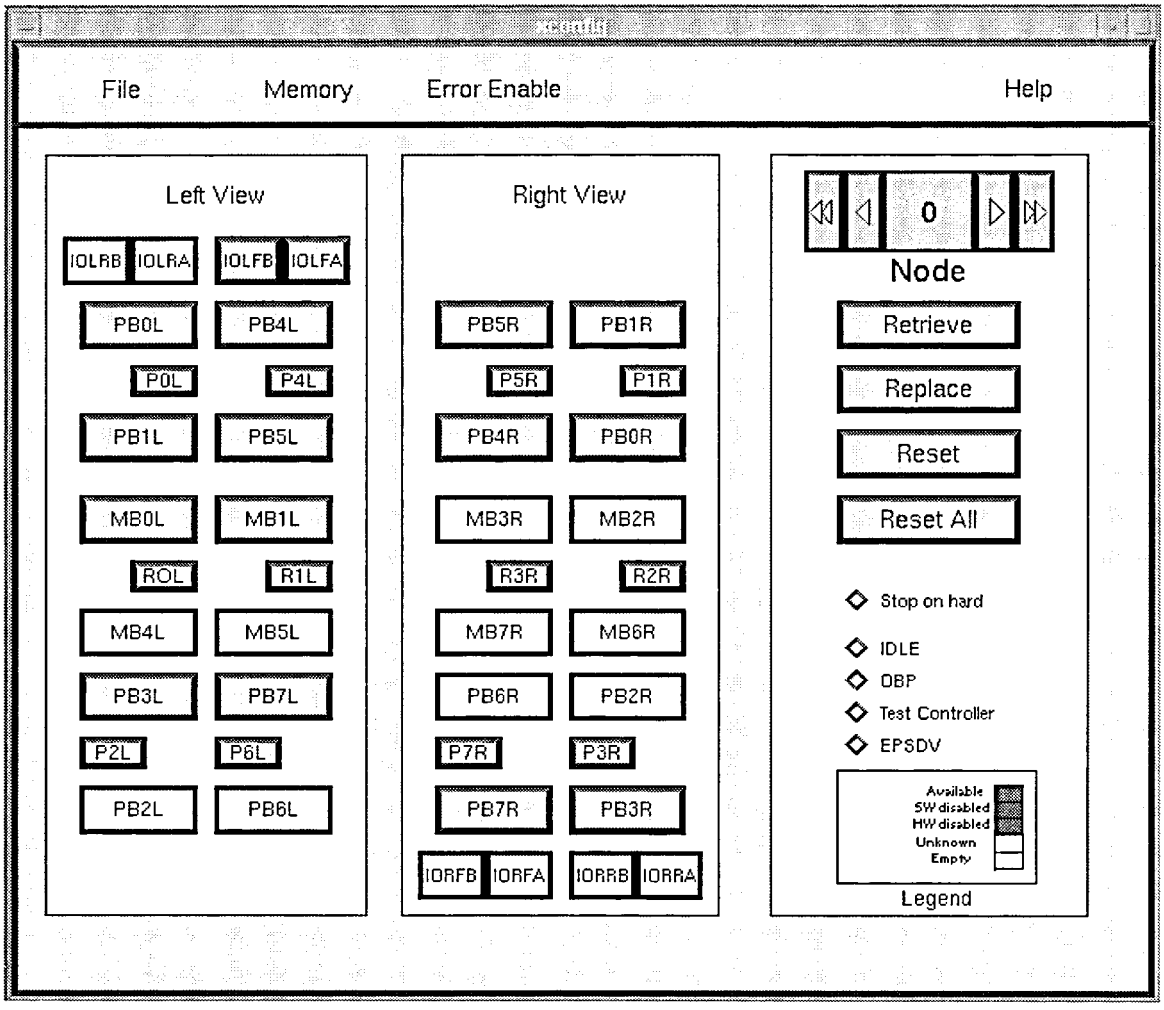

Figure 3 `xconfig` window

As buttons are clicked, the item selected changes state and color. There is a legend on the screen to explain the color and status.The change is recorded in the test stations image of the node.

When the user is satisfied with the new configuration, it should be replaced back into the node, and the node should be reset to enable the changes.

The main `xconfig` window has three sections:

- Menu bar—Provides additional capability and functions.
- Node configuration map—Provides the status of the node.

- Node control panel—Provides the capability to select a node and control the way data flows to it.

Menu bar

The menu bar appears at the top of the `xconfig` main window. It has four menus that provide additional features:

- File menu—Displays the file and exit options.
- Memory menu—Displays the main memory and CTI cache memory options.
- Error Enable menu—Displays the device menu options for error enabling and configuration.
- Help menu—Displays the help and about options.

The menu bar is shown in Figure 4.

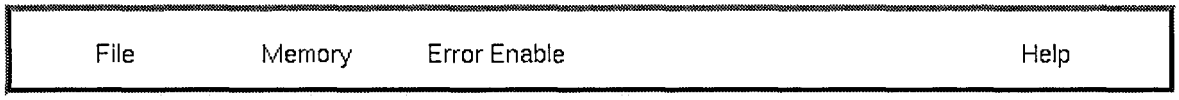

Figure 4 `xconfig` window menu bar

The File menu provides the capability to save and restore node images and to exit `xconfig`.

The Memory menu provides the capability to enable or disable memory at the memory DIMM level by the total memory size and to change the network cache size on a multinode complex.

The Error Enable menu provides the capability to change a device's response to an error condition. This is normally only used for troubleshooting.

The Help menu provides a help box that acts as on-line documentation and also contains program revision information.

Node configuration map

The node configuration map is a representation of the left and right side views of a node as shown in Figure 5.

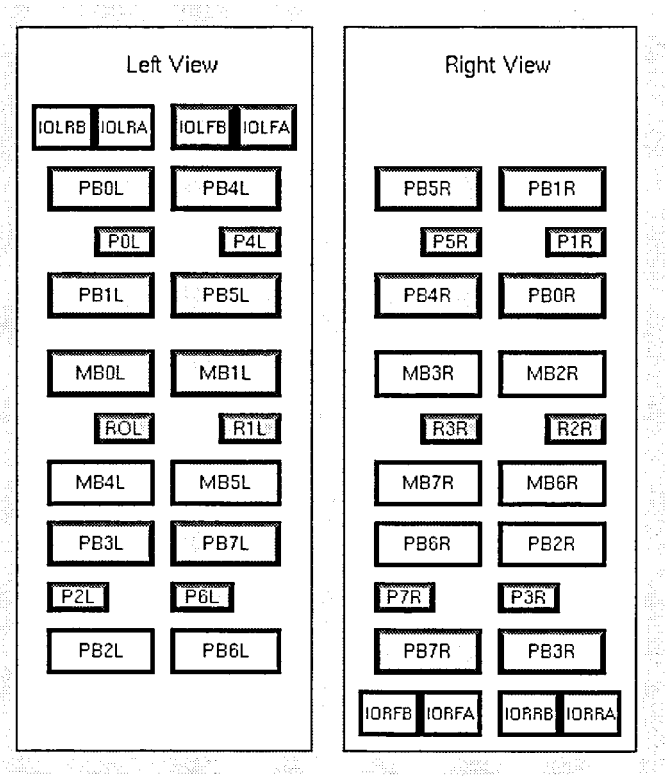

Figure 5 `xconfig` window node configuration map

The button boxes are positioned to represent the actual boards as viewed from the left and right sides. Each of the configurable components of the node is in the display.

A green button indicates that the component is present and enabled.

A red button indicates that the component is software disabled in the system.

A white button indicates that it is not possible to determine what the status of the component would be if POST were to be started.

A blue box indicates that the component is either not present or fails the power-on self tests.

A brown button indicates that POST had to hardware deconfigure this component in order to properly execute.

The colors are shown in the legend box of the node control panel.

Components can change from enabled to disabled or disabled to unknown by clicking on the appropriate button with the left mouse button.

A multinode system requires an additional component on a memory board to enable the scalable coherent memory interface. This component can be viewed by right clicking the on the memory board button. The right mouse button toggles the memory board display between the memory board and the SCI device

Node control panel

The node control panel allows the user to select a node, select the stop clocks on an error function, select the boot parameters for a node and direct data flow between the node and the `xconfig` utility. It is shown in Figure 6.

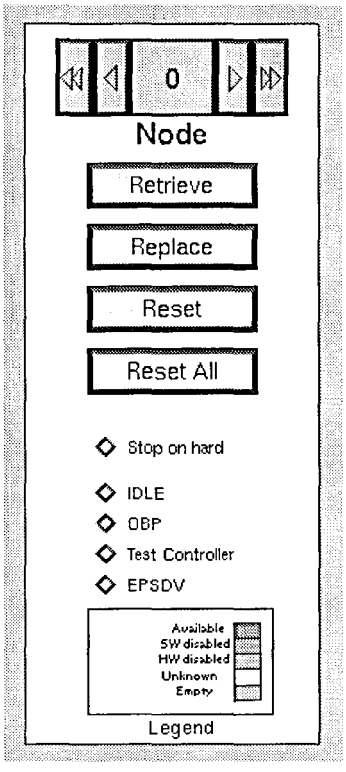

Figure 6 `xconfig` window node control panel

The node number is shown in the node box. A new number can be selected using the arrow buttons. The single arrow advances the

node counter by one. The double arrow advances the node counter by four.

When a new node is selected and available, its data is automatically read, and the node configuration map updated. The data image is kept on the test station until it is rebuilt on the node using the Replace button. This is similar to the `push` command on `ccmu`.

Even though data can be rebuilt on a node, it does not become active until POST runs again and reconfigures the system. The Reset or Reset All buttons can be used to restart POST on one or all nodes of a system. A multinode system requires a reset all to properly function.

A Retrieve button is available on the node control panel to get a fresh copy of the parameters settings in the system. Clicking this button overwrites the setting local to the test station and `xconfig`.

The Stop-on-hard button is typically used to assist in fault isolation. It stops all system clocks shortly after an error occurs. Only scan-based operations are available once system clocks have stopped.

The last group of buttons control what happens after POST completes. The node can become idle or boot OBP, the test controller or EPSDV. The test controller and EPSDV are additional diagnostic modes.

Power-On Self Test

3

The PUC contains firmware in EEPROM known as Power-On Self Test (POST) routine. This chapter describes how POST initializes a node and handles power up errors.

Overview

Upon power up, all processors and hardware must be initialized before the node proceeds with booting. POST begins executing and brings up the node from an indeterminate state and then executes OBP.

None of the POST routines can be directly controlled via a user interface. Program control is provided by a set of configuration parameters (processing flags and variable definitions) stored in EEPROM by OBP, `ccmu`, or `xconfig`.

The error reporting modules display error codes for all fatal errors that occur during the POST execution. Any errors that can be recovered from, by reconfiguring the hardware, will be reported to OBP. POST status is reflected on the LCD display.

POST performs the following tasks:

- Initializes each processor in the node
- Validates all shared data structures within the EEPROM
- Initializes the core logic required to start OBP execution
- Determines node configuration
- Initializes all ASICs
- Initializes main memory
- Sets up CTI cache
- Invokes OBP

Any fatal errors are reported to the user via the system LCD. POST passes node configuration and any options to OBP via shared data structures.

Reset

POST may be invoked by one of the following types of reset:

- Powerup reset
- Hard reset
- Soft reset
- Transfer of control (TOC)

Power up reset

Power up reset occurs when power is applied to the node. The PUC deasserts reset which is synchronized with TCLK to provide a synchronous reset to all controllers. The processors build a node configuration and memory map. They also read the midplane COP chip to determine node ID. All MACs automatically begin refresh cycles.

Hard reset

Hard reset occurs when the Hard Reset bit in the PUC reset register is set. The PUC asserts the RESET line for approximately 100 msec and then releases it, producing a synchronous reset to the node.

Hard reset can also be invoked by scanning a value of one into the Hard Reset bit. The processors perform the same sequence as in a power up reset, including memory initialization and sizing. Hard reset by scanning allows complete initialization of a node remotely by the user.

Soft reset

Soft reset occurs when the Soft Reset bit in the PUC Reset register is set. The PUC asserts the RESET line for approximately 100 msec and then releases it, producing a synchronous reset to the node.

Soft reset can be invoked by scanning a value of one into the Soft Reset bit. Each processor initializes all CSRs in each controller but does not initialize or size memory, or initialize any part of the node that has been deconfigured.

TOC

The transfer of control reset occurs when one of the following conditions exist:

- The TOC button is pushed.
- The TOC bit in the PUC reset register is set.
- The TOC bit in the PUC CSR is set by scanning.

The `do_reset` routine also asserts TOC.

All processors in the node trap into the TOC vector in processor-dependent code (PDC) space.

POST modules

POST contains the following modules:

- Core Logic SRAM Initialization—Initializes all the SRAM space in the core logic to a known state. It also initializes parity for the SRAM.
- Configuration Determination and ASIC Hardware Initialization—Determines the ASIC population and sets the ASIC CSRs to a default state to prepare for main memory initialization. It initializes core logic utilities ASICs at the beginning of this process.
- Main Memory Initialization—Configures the memory interleave hardware and initializes all of main memory to valid data, ECC, and tag values for both local memory and CTI cache. The main memory population is reported to OBP.
- Error Reporting—Sets up and maintains the boot error log and boot status word in the EEPROM and reports errors to the LCD.

Node initialization

When power is applied to each node, all controllers in the node receive a power-up reset signal. POST provides a controlled node initialization and prepares it for booting OBP. Figure 7 shows of how POST initializes a node.

Processor initialization and selftest

Upon reset, all processors are initialized and placed in selftest. The extent of the selftest is determined by a mode bit in nonvolatile static random access memory (NVRAM).

Upon successful completion of processor selftest, the existence of each possible slice is determined. This is accomplished by reading the revision register of each PAC, PCI-bus Interface Controller (PIC), Memory Array Controller (MAC), and Toroidal Access Controller (TAC). Each processor will determine its processor ID from the PAC. Also, each processor fetches the Processor Semaphore register on the PUC. Because register requests are queued, one processor will fetch this CSR before the others and becomes the booting, or *monarch*, processor. All others go into a

command wait idle loop. The booting processor begins executing POST code from the PUC EEPROM.

Checksum verification of NVRAM

The PUC EEPROM contains the POST, OBP, and node diagnostics code. In addition, these three routines use shared data structures. The shared data structures reside in sections of the NVRAM. Each data structure has an embedded checksum word. POST checks the validity of each shared data structure by reading and comparing its checksum.

Core logic initialization

The core logic contains SRAM and DUARTs that support external terminal connection for self test and the LCD panel. POST initializes the SRAM, DUARTs, and all controller CSRs in the node.

node configuration determination

POST determines which and how many ASICs reside in the system (not every system contains a full complement of support hardware). It also determines the number of memory modules and their sizes. Any ASIC that does not respond to any CSR access is considered to be not installed.

node ASIC initialization

POST sets every node controller to a known state. The state is based both on configuration parameters and the current hardware configuration.

node main memory initialization

The processor reads the node ID from the COP and uses its information to load the Local node ID register.

Next, the monarch processor determines the memory configuration for all EMACs. It determines the size, population, and installation of each SIMM on a memory board and returns this information to the POST. The results are compared with the results of each other memory mapping and the least common

denominator is determined and mapped in. Once the memory population is determined, the memory and tags are initialized.

node clean up and OBP boot process

The POST resets the PUC Processor Semaphore CSR and cleans up any residual state information from the initialization process. All processors now begin to execute the OBP routine at approximately the same time.

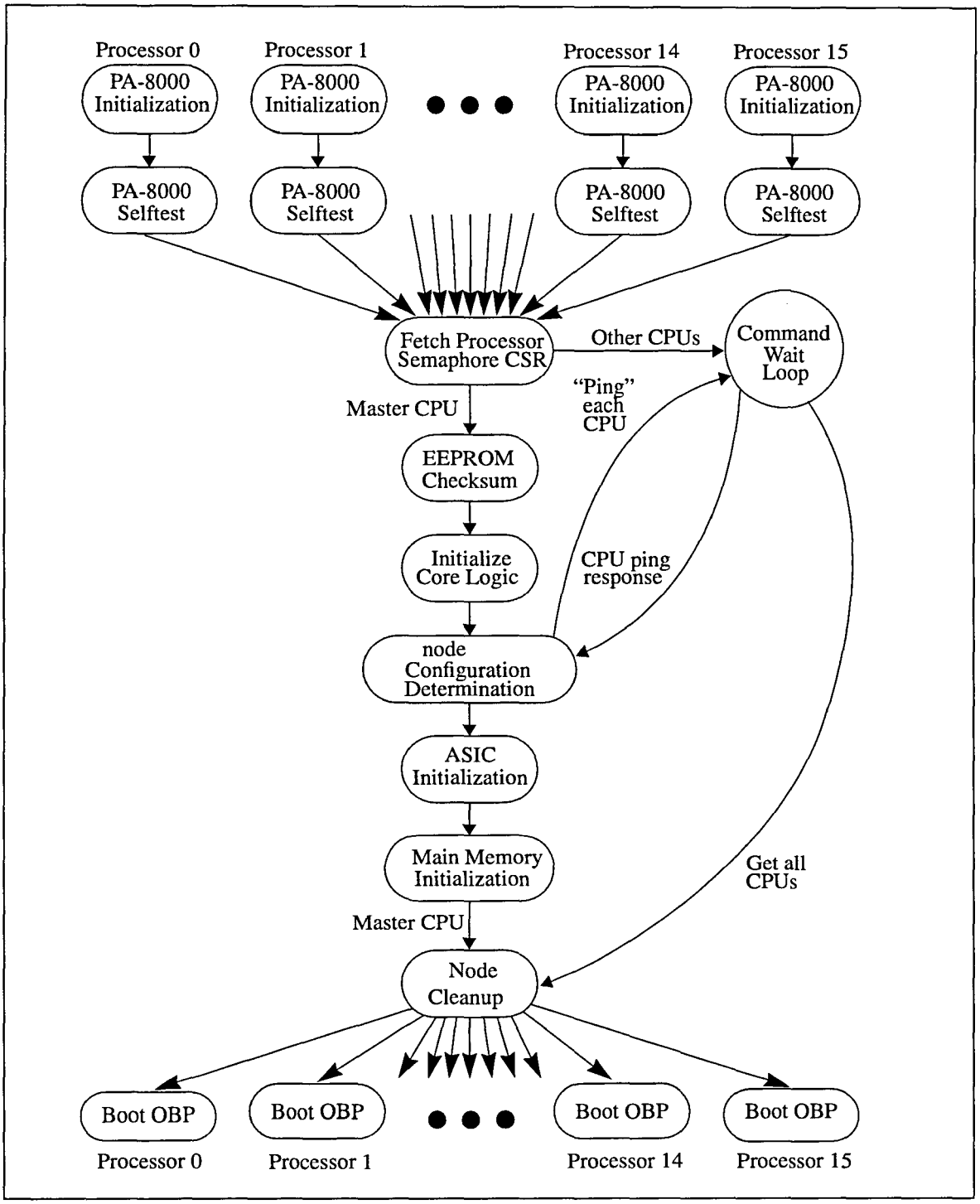

Figure 7 POST program flow

dcm script

The dcm script looks for PERL code in /spp/bin/perl and reads, parses, and prints to the screen the boot configuration map in NVRAM through calls to sppdsh. The script usage is:

```
dcm <Node number>
```

where <Node number> is a hex value and is the node on which to run dcm (in hex).

The following example runs dcm on nodes 0 and 4.:

```
% dcm 0 4
```

The following example displays the dcm help file:

```
% dcm -h
```

Running dcm provides the following information:

- Checksum for Boot configuration Map data structure
- Boot Configuration Map size
- POST Revision version used to create the Boot Configuration Map
- Processor cache (instruction cache, data cache) size in MB
- PACs (PASS, FAIL, or EMPTY)
- Status of PICs (PASS, FAIL, or EMPTY)
- Status of MACs (PASS, FAIL, or EMPTY)
- Status of TACs (PASS, FAIL, or EMPTY)
- Location of memory boards

Figure 8, Figure 9, and Figure 10 together show a single output example of entering the following on the command line:

```
% dcm 0
```

```
            Excalibur Configuration Map Dump: Node: 0
          ============================================================
VERSION: 001.000.000.000        compiled: 1996/10/10 13:23:07   by: mullins

 Acquiring the Boot Configuration Map...
         Check Sum:              0xf1a83645

         Boot Config Map Size:  113 Words
         POST Revision:         1.7.0.0

 CPUs (ICache, DCache) Size in Bytes
 ===================================
 PB0L -      PASS ( 1048576, 1048576)    PB0R -      PASS ( 1048576, 1048576)
 PB1R -      PASS ( 1048576, 1048576)    PB1L -      PASS ( 1048576, 1048576)
 PB2L -      PASS ( 1048576, 1048576)    PB2R -      PASS ( 1048576, 1048576)
 PB3R -      PASS ( 1048576, 1048576)    PB3L -      PASS ( 1048576, 1048576)
 PB4L -      PASS ( 1048576, 1048576)    PB4R -      PASS ( 1048576, 1048576)
 PB5R -      PASS ( 1048576, 1048576)    PB5L -      PASS ( 1048576, 1048576)
 PB6L -      PASS ( 1048576, 1048576)    PB6R -      PASS ( 1048576, 1048576)
 PB7R -      PASS ( 1048576, 1048576)    PB7L -      PASS ( 1048576, 1048576)

 EPACs
 =====
  P0L -          PASS      P4L -          PASS
  P1R -          PASS      P5R -          PASS
  P2L -          PASS      P6L -          PASS
  P3R -          PASS      P7R -          PASS

 EPICs
 =====
  IOLF_B -       PASS      IOLF_A -       PASS
  IOLR_B -      EMPTY      IOLR_A -      EMPTY
  IORR_B -      EMPTY      IORR_A -      EMPTY
  IORF_B -      EMPTY      IORF_A -      EMPTY

 EMACs
 =====
  MB0L_M -       PASS      MB4L_M -       PASS
  MB1L_M -       PASS      MB5L_M -       PASS
  MB2R_M -       PASS      MB6R_M -       PASS
  MB3R_M -       PASS      MB7R_M -       PASS

 ETACs
 =====
  MB0L_T -      EMPTY      MB4L_T -      EMPTY
  MB1L_T -      EMPTY      MB5L_T -      EMPTY
  MB2R_T -      EMPTY      MB6R_T -      EMPTY
  MB3R_T -      EMPTY      MB7R_T -      EMPTY
```

Figure 8 dcm sample output part A

```
 Memory:  B(ank#)S(lot#) - Row_0/Row_1
 =======  ============================

EMB0:
=====
B0S0 -  16MB/16MB   B0S1 -  16MB/16MB   B0S2 -  16MB/16MB   B0S3 -  16MB/16MB
B1S0 -  16MB/16MB   B1S1 -  16MB/16MB   B1S2 -  16MB/16MB   B1S3 -  16MB/16MB
B2S4 -  16MB/16MB   B2S5 -  16MB/16MB   B2S6 -  16MB/16MB   B2S7 -  16MB/16MB
B3S4 -  16MB/16MB   B3S5 -  16MB/16MB   B3S6 -  16MB/16MB   B3S7 -  16MB/16MB

EMB1:
=====
B0S0 -  16MB/16MB   B0S1 -  16MB/16MB   B0S2 -  16MB/16MB   B0S3 -  16MB/16MB
B1S0 -  16MB/16MB   B1S1 -  16MB/16MB   B1S2 -  16MB/16MB   B1S3 -  16MB/16MB
B2S4 -  16MB/16MB   B2S5 -  16MB/16MB   B2S6 -  16MB/16MB   B2S7 -  16MB/16MB
B3S4 -  16MB/16MB   B3S5 -  16MB/16MB   B3S6 -  16MB/16MB   B3S7 -  16MB/16MB

EMB2:
=====
B0S0 -  16MB/16MB   B0S1 -  16MB/16MB   B0S2 -  16MB/16MB   B0S3 -  16MB/16MB
B1S0 -  16MB/16MB   B1S1 -  16MB/16MB   B1S2 -  16MB/16MB   B1S3 -  16MB/16MB
B2S4 -  16MB/16MB   B2S5 -  16MB/16MB   B2S6 -  16MB/16MB   B2S7 -  16MB/16MB
B3S4 -  16MB/16MB   B3S5 -  16MB/16MB   B3S6 -  16MB/16MB   B3S7 -  16MB/16MB

EMB3:
=====
B0S0 -  16MB/16MB   B0S1 -  16MB/16MB   B0S2 -  16MB/16MB   B0S3 -  16MB/16MB
B1S0 -  16MB/16MB   B1S1 -  16MB/16MB   B1S2 -  16MB/16MB   B1S3 -  16MB/16MB
B2S4 -  16MB/16MB   B2S5 -  16MB/16MB   B2S6 -  16MB/16MB   B2S7 -  16MB/16MB
B3S4 -  16MB/16MB   B3S5 -  16MB/16MB   B3S6 -  16MB/16MB   B3S7 -  16MB/16MB

EMB4:
=====
B0S0 -  16MB/16MB   B0S1 -  16MB/16MB   B0S2 -  16MB/16MB   B0S3 -  16MB/16MB
B1S0 -  16MB/16MB   B1S1 -  16MB/16MB   B1S2 -  16MB/16MB   B1S3 -  16MB/16MB
B2S4 -  16MB/16MB   B2S5 -  16MB/16MB   B2S6 -  16MB/16MB   B2S7 -  16MB/16MB
B3S4 -  16MB/16MB   B3S5 -  16MB/16MB   B3S6 -  16MB/16MB   B3S7 -  16MB/16MB

EMB5:
=====
B0S0 -  16MB/16MB   B0S1 -  16MB/16MB   B0S2 -  16MB/16MB   B0S3 -  16MB/16MB
B1S0 -  16MB/16MB   B1S1 -  16MB/16MB   B1S2 -  16MB/16MB   B1S3 -  16MB/16MB
B2S4 -  16MB/16MB   B2S5 -  16MB/16MB   B2S6 -  16MB/16MB   B2S7 -  16MB/16MB
B3S4 -  16MB/16MB   B3S5 -  16MB/16MB   B3S6 -  16MB/16MB   B3S7 -  16MB/16MB
```

Figure 9 `dcm` sample output part B

```
EMB6:
=====
B0S0 -  16MB/16MB   B0S1 -  16MB/16MB   B0S2 -  16MB/16MB   B0S3 -  16MB/16MB
B1S0 -  16MB/16MB   B1S1 -  16MB/16MB   B1S2 -  16MB/16MB   B1S3 -  16MB/16MB
B2S4 -  16MB/16MB   B2S5 -  16MB/16MB   B2S6 -  16MB/16MB   B2S7 -  16MB/16MB
B3S4 -  16MB/16MB   B3S5 -  16MB/16MB   B3S6 -  16MB/16MB   B3S7 -  16MB/16MB

EMB7:
=====
B0S0 -  16MB/16MB   B0S1 -  16MB/16MB   B0S2 -  16MB/16MB   B0S3 -  16MB/16MB
B1S0 -  16MB/16MB   B1S1 -  16MB/16MB   B1S2 -  16MB/16MB   B1S3 -  16MB/16MB
B2S4 -  16MB/16MB   B2S5 -  16MB/16MB   B2S6 -  16MB/16MB   B2S7 -  16MB/16MB
B3S4 -  16MB/16MB   B3S5 -  16MB/16MB   B3S6 -  16MB/16MB   B3S7 -  16MB/16MB
```

Figure 10 dcm sample output part C

Test controller

4

The test controller is an EEPROM-based utility that provides the environment for executing the offline diagnostic tests. It is controlled through parameters stored in the NVRAM on the Utilities board. The test controller reads these parameters to determine its execution mode, the number processors to test, which MACs to include in the testing, which subtests to run, and other diagnostic test specific information.

Test controller modes

There are two basic operational modes for this utility:

- Stand-alone mode
- Interactive mode

In stand-alone mode, `cxtest` invokes the test controller. The test controller reads test parameters from NVRAM (these parameters are written into NVRAM by `cxtest` before it invokes the test controller), executes the test and subtests specified in NVRAM, and sets a completion bit in NVRAM when the test and subtests are finished. `cxtest` is described in Chapter 5.

In interactive mode, a user interface allows the user to select the processors to test, select the subtests to run, and examine error information. The user interface is a set of menus described in this chapter.

In the interactive mode, the test controller loops waiting for the `start` command. Prior to issuing the `start` command, any global and/or processor specific parameters can be modified. Other tests can be loaded into main memory. By modifying a few processor test parameters, the test controller interfaces with these tests. When all tests have completed, the test controller waits for the next `start` command. Any combination of parameter and tests may be modified and executed.

User interface

There is a common interface for subtests to access the test parameters and report events to the test controller. The test controller provides for the control of offline diagnostic test execution. It utilizes a set of parameters to control its operation. The parameters consist of the following:

- Global set that controls the overall operation of the test controller
- Test set (one per test) that controls how the tests are executed by the test controller
- CPU parameters (one per processor) that contain status information about the tests executing on each processor

All these parameters are in NVRAM.

The user interface allows the user to modify parameters that reside in NVRAM, thereby controlling the operation of the test controller. It also allows the user to select which subtests are executed on each of the processors and modify the test parameters, as well as any other test information.

The test controller user interface consists of two basic menus. The first is the main menu that gives the user the following capabilities:

- Modify the POST boot selection
- Control operation of the test controller
- Display the current global parameter selections
- Display processor summary
- Switch processors
- Go to the Test Configuration menu

The second menu is the processor Test Control menu that provides the following capabilities:

- Select classes of subtests to execute
- Select subtests to execute
- Specify pause enables
- Specify whether to loop or not
- Specify the test and/or subtest error counts
- Read and write the 20 words of test specific information
- Select the hardware to test
- Display the current parameter selections

Main menu

Figure 11 shows the test controller main menu.

```
MAIN Menu commands

       0=Quit Test Controller
       1=Begin Test Controller Execution
       2=Halt Test Controller Execution
       3=Resume Test Controller Execution
       4=Switch CPU
       5=POST Boot Selection
       6=Execution Mode Selection
       7=Global Parameter Display
       8=CPU Summary Display
       9=Display CPU Errors
       A=Test Selection Menu
       B=Test Configuration Menu
       C=Debugging Menu
       D=Display revision

Enter Command:
```

Figure 11 Main menu

Each main menu selection is defined as follows:

- 0=Quit Test Controller—Terminates the test controller utility and either reboots the system (to POST and then to the selected program) or halts the system depending on the current value of the POST Boot Selection flag.
- 1=Begin Test Controller Execution—Starts the test controller utility executing the specified subtests on the selected processors. The entire system is started from the beginning.
- 2=Halt Test Controller Execution—Suspends temporarily the operation of the test controller. This command may be entered at any time. Only the test controller is halted; subtests on other processors continue to execute.
- 3=Resume Test Controller Execution—Continues execution from the point of interruption.
- 4=Switch CPU—Allows the user to start the test controller on the specified processor. The previously used processor starts executing the command wait loop code. The user is prompted for the processor as follows:

  ```
  Enter CPU (0-f):
  ```

- 5=POST Boot Selection—Prompts the user for the new value with the following prompt:

```
POST Boot Selection (0=Spin, 1=OBP,2=Diags,
3=Standalone Diags, 4=Dumper 5=EPSDV):
```

The user must enter in value between 0 and 4 inclusive. Each of these values is further defined in Table 5.

Table 5 POST Boot Selections

Selection	Description
0	POST enters it's spin wait loop.
1	POST boots to OBP.
2	POST boots to the test controller (interactive mode).
3	POST boots to the test controller (standalone mode).
4	POST boots to RDR dumper.
5	POST boots to EPSDV.

For all values, POST boots to ESPDV, the test controller performs a hard reset to POST when the test controller terminates.

- 6=Execution Mode Selection—Allows the user to select the mode for executing the subtests. The two options are serial and parallel. The following prompt queries for the selection:

```
Execution Mode Selection (0=serial,
1=parallel):
```

- 7=Global Parameter Display—Displays the available hardware components, the current "POST Boot Selection" value, and the current "Execution Mode Selection" value. The display is shown in Figure 12. The asterisks denote the component has passed POST processing and is available for

diagnostic testing (see CPUs 0-3 and EMACs 0, 2, 4, and 6 in the display).

```
MAIN Menu - Global Parameters Display

       CPUs:                      0* 1* 2* 3* 4  5  6  7  8  9  A  B  C  D  E  F
       EPACs:                     0* 1* 2  3  4  5  6* 7
       EMACs:                     0* 1  2* 3  4* 5  6* 7
       ETACs:                     0  1  2  3  4  5  6  7
       EPICs:                     0  1  2  3  4  5  6  7
       POST Boot Selection:    Spin   OBP*   TC   Standalone   Dumper   EDPSDV
       Execution Mode Selection: Serial    Parallel*
```

Figure 12 Main menu - Global Parameters display

- 8=CPU Summary display—Displays a summary of the current processor and testing information. An example of the display is shown in Figure 13. Each available hardware component is marked with an asterisk just to the right of its number (see CPUs 0-3 and EMACs 0, 2, 4, and 6 in the display).

```
MAIN Menu - CPU Summary Display

       Total Failures = 00000000

       Configuration Map
       =============
       CPUs:          0*  1*  2*  3*  4   5   6   7   8   9   A   B   C   D   E   F
       EPACs:         0   1   2   3   4   5   6   7
       ERACs:         0   1   2   3
       EMACs:         0*  1   2*  3   4*  5   6*  7
       ETACs:         0   1   2   3   4   5   6   7
       EPICs:         0   1   2   3   4   5   6   7

                              FAIL
       CPU    STATE           COUNT          SUBTEST        TEST NAME
       ===    =====           ======         ========       ==========
       0       Idle            0             n/a            n/a
       1       Idle            0             n/a            n/a
       2       Idle            0             n/a            n/a
       3       Idle            0             n/a            n/a
       4       Idle            0             n/a            n/a
       5       Idle            0             n/a            n/a
       6       Idle            0             n/a            n/a
       7       Idle            0             n/a            n/a
       8       Idle            0             n/a            n/a
       9       Idle            0             n/a            n/a
       A       Idle            0             n/a            n/a
       B       Idle            0             n/a            n/a
       C       Idle            0             n/a            n/a
       D       Idle            0             n/a            n/a
       E       Idle            0             n/a            n/a
       F       Idle            0             n/a            n/a
```

Figure 13 Main menu - Processor Summary display

The possible states in Figure 13 are described in Table 6.

Table 6 Processor States

CPU State	Description
Not Available	Denotes processor is not available for testing.
Running	Denotes a test is currently running on this processor.
Idle	Denotes that no test is running on this processor
Ready	Denotes last subtest completed and ready for next subtest
Test Completed	Denotes test completed execution on this processor.

Table 6 Processor States—**(continued)**

CPU State	Description
Error Detected	Denotes test halted due to an error condition on this processor.
Test Timeout	Denotes a timeout detected during test execution on this processor; the test is halted.
HW Reqs Not Met	Denotes the hardware selected does not meet the minimum hardware requires for executing the test.
User Halted	Denotes user halted test.
Unexpected HPMC	Denotes running test caused an HPMC; the test is halted.
SW Deconfigured	Denotes test automatically halted testing on this processor, because of a software restriction.

- 9=Display CPU Errors—Displays the errors for the currently selected processor. When selected, the user is prompted for the processor as follows:

  ```
  Enter CPU [0-f]:
  ```

  Figure 14 shows an example of this display. There are four fields:
  - Date/Time—Date and time the error was logged.
  - Subtest—Failing subtest number
  - Event Code—32-bit event code
  - Error Message—40-character error message

```
Error Display for CPU xx
Test Name: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

       Date/Time                    Event          Error
mm/dd/yy hh:mm:ss    Subtest       Code           Message
12/34/56 77:88:99    xxxxx         xxxxxxxx       AAAAAAAAAAAAAAAAAA....
12/34/56 77:88:99    xxxxx         xxxxxxxx       AAAAAAAAAAAAAAAAAA....
12/34/56 77:88:99    xxxxx         xxxxxxxx       AAAAAAAAAAAAAAAAAA....
12/34/56 77:88:99    xxxxx         xxxxxxxx       AAAAAAAAAAAAAAAAAA....
12/34/56 77:88:99    xxxxx         xxxxxxxx       AAAAAAAAAAAAAAAAAA....
```

Figure 14 Test Configuration menu - Display processor Errors

- A=Test Selection Menu—Invokes the menu shown in Figure 15. This menu allows the user to select which tests to execute. An asterisk before the test name denotes that it has been selected. In Figure 15, only the memory test is selected. Selecting options 1 through A toggles the current state for that

particular test. Selecting option 0 returns the user to the main menu shown above in Figure 11.

```
MAIN Menu - Test Selection Display

        0=Return to Main Menu
        1=*Memory test
        2=Architecture Features test
        3=Intra-Node Coherency test
        4=Inter-Node Coherency test
        5=I/O test
        6=CPU selftests
        7=not available
        8=not available
        9=not available
        A=not available
Please enter number of test:
```

Figure 15 Main menu - Test Selection menu

- B=Test Configuration Menu—Switches the user to the Configuration menu shown in Figure 17 for the specified test.
- C=Debugging Menu—Invokes the Debugging Menu shown in Figure 16 which allows the user to read or write to any memory location on the hypernode and dump various data.

```
MAIN Menu - Debugging Menu

        0=Return to Main Menu
        1=Read 32-bit Memory Location
        2=Write 32-bit Memory Location
        3=Read 64-bit Memory Location
        4=Write 64-bit Memory Location
        5=Dump DUART Port 0 Registers
        6=Dump DUART Port 1 Registers
        7=Dump SONIC Registers
        8=Dump TC CPU Info Structure
        9=Dump TC Test Info Structure
        A=Dump PCXU General Registers
        B=Dump PCXU Control Registers
        C=Read ECC From Memory Line
        D=Read Tag From Memory Line
        E=Print Test Revision

Enter number of activity:
```

Figure 16 Main menu - Debugging menu

- Selection 1 queries for the 40-bit address to read as follows:

  ```
  Enter 40-bit address:
  ```

- Selection 2 queries for the 40-bit address and then for the 32-bits of data to write:

  ```
  Enter 32-bit data:
  ```

- Selection 3 queries for the 40-bit address to read.
- Selection 4 queries for the 40-bit address to write, and then for the 64-bits of data to write as follows:

  ```
  Enter Upper 32-bits:
  ```

  ```
  Enter Lower 32-bits:
  ```

- Selection 8 queries the user for the processor index as follows:

  ```
  Enter cpu index [0-f]:
  ```

- Selection 9 queries the user for the test index with the prompt shown. The test indices for the various tests are shown above in Figure 15.

  ```
  Enter test index [1-A]:
  ```

- Selections C and D query for the 40-bit address for which to display the ECC or tag of that memory line.
- Selection E queries the user for the test index with the following prompt:

  ```
  Enter test index [1-A]:
  ```

Test Configuration menu

The Test Configuration menu is shown in Figure 17.

```
Test Configuration Menu

     0=Return to Main Menu          9=Display Test Parameters
     1=Display Classes              A=Hardware Selection Menu
     2=Display Subtests             B=Loop Enable
     3=Select Classes               C=Loop Count
     4=Select Subtests              D=Test Error Count
     5=Read All Test Parameters     E=Pause at Test Start
     6=Read One Test Parameter      F=Pause at Test End
     7=Write Test Parameter         G=Pause at Subtest Start
     8=Reset Parameters             H=Pause at Subtest End

Enter command:
```

Figure 17 Test Configuration menu

Each Test Configuration menu selection is defined as follows:

- 0=Return to Main Menu—Returns the user to the Main menu shown in Figure 16.
- 1=Display Classes—Displays the current class definitions for this diagnostic. An example of the display is shown in Figure 18.

```
Test Configuration Menu - Class Display

       Class  Description
       0      class 0 description
       1      class 1 description
       .      .
       .      .
       n      class n description
```

Figure 18 Test Configuration menu - Class display

- 2=Display Subtests—Displays the current subtest definitions for this diagnostic. An example of the display is shown in Figure 19.

```
Test Configurationl Menu - Subtest Display

       Subtest       Description
       0             subtest 0 description
       1             subtest 1 description
       .             .
       .             .
       n*            subtest n description
```

Figure 19 Test Configuration menu - Subtest display

An asterisk following the subtest number denotes that it is selected for execution. For example, see the "n subtest n description" line.

- 3=Select Classes—Allows the user to specify which classes of subtests to execute. These selections are in addition to any subtests selected. The following prompt is displayed:

  `Enter class number:`

  The user must enter one of the following:

  - An optional operator followed by a class number, for example 2, +2 or -2
  - Multiple class numbers (class, +class, -class are allowed) separated by commas, for example 1,2,3

The class numbers are decimal.

- 4=Select Subtests—Allows the user to specify which subtests to execute. These selections are in addition to any classes selected. The following prompt is displayed:

  ```
  Enter subtest number or subtest range:
  ```

  The user may enter one of the following:

  - A single subtest number
  - A subtest range which consists of two numbers separated with a dash and is inclusive. An example of a valid range entry is 100-200.
  - An optional operator followed by a subtest selection, for example 2, +2, -2, +100-200, or -100-200.

  The subtest numbers are decimal values only.

- 5=Read All Test Parameters—Reads all 20 words that make up the test parameter set and displays this information. These test parameters reside in NVRAM and are defined by the particular test executing on this processor. An example of the display is shown in Figure 20.

```
Test Configuration Menu - Test Parameters

    Word   Value                Word   Value
    0      0xhhhhhhhh           10     0xhhhhhhhh
    1      0xhhhhhhhh           11     0xhhhhhhhh
    2      0xhhhhhhhh           12     0xhhhhhhhh
    3      0xhhhhhhhh           13     0xhhhhhhhh
    4      0xhhhhhhhh           14     0xhhhhhhhh
    5      0xhhhhhhhh           15     0xhhhhhhhh
    6      0xhhhhhhhh           16     0xhhhhhhhh
    7      0xhhhhhhhh           17     0xhhhhhhhh
    8      0xhhhhhhhh           18     0xhhhhhhhh
    9      0xhhhhhhhh           19     0xhhhhhhhh
```

Figure 20 Test Configuration menu - Test Parameters

- 6=Read One Test Parameter—Allows the user to read a single test parameter.
- 7=Write Test Parameter—Allows the user to modify any one of the test parameter words. The user is first requested to specify which word is to be modified:

  ```
  Specify Test Parameter word [0-19]:
  ```

  After specifying the word, the user is then prompted for the new value:

  ```
  New value for Test Parameter word xx:
  ```

- 8=Reset Parameters—Resets some of the parameters to their default values. Table 7 lists the affected parameters and their default values.

Table 7 Parameter Defaults

Parameter	Default value
Loop Enable	0
Loop Count	0
Test Error Count	1
Pause At Test Start	0
Pause At Test End	0
Pause At Subtest Start	0
Pause At Subtest End	0

- 9=Display Test Parameters—Displays the current values of the processor parameters. An example of the display is shown in Figure 21. An asterisk denotes the current selections. For Example, processor 0 is selected.

  This minimum hardware requirements information is enclosed in parentheses after the hardware type label and denotes the number of that type required. In the example below, 1 processor, 1 PAC (the one associated with the selected processor), and 1 MAC are needed.

  The test controller compares the selected hardware components versus these minimum requirements to

determine if the test can be executed. Unless these minimum requirements are met, the test cannot be executed.

```
Test Configuration Menu - Test Parameters Display

       CPUs:   (1)   0* 1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
       EPACs:  (1)   0* 1  2  3  4  5  6  7
       ERACs:  (4)   0* 1* 2* 3*
       EMACs:  (1)   0* 1  2  3  4  5  6* 7
       ETACs:  (0)   0  1  2  3  4  5  6  7
       EPICs:  (0)   0  1  2  3  4  5  6  7
       Nodes: (1)

       Loop Enable:          ON     OFF*
       Test Error Count:     10
       Pause Test Start:     ON     OFF*
       Pause Test End:       ON     OFF*
       Pause Subtest Start:  ON     OFF*
       Pause Subtest End:    ON     OFF*
```

Figure 21 Test Configuration menu - Test Parameters display

- A=Hardware Selection menu—Invokes Hardware Selection menu shown in Figure 22.

```
Test Configuration Menu - Hardware Selection Display

       0=Return to Test Configuration Menu
       1=CPU Selection
       2=EPAC Selection
       3=EMAC Selection
       4=ETAC Selection
       5=EPIC Selection
       6=Node Selection
Enter Command:
```

Figure 22 Test Configuration menu - Hardware Selection menu

The selection of the Hardware Selection menu are defined as follows:

- 1-5=<hardware> Selection—Selects the appropriate controller. The following prompt is displayed:

  `Select <hardware>:`

  The user must enter one of the following:

  * An optional operator followed by a hardware component number, for example 2, +2 or -2

* Multiple hardware component numbers separated by commas, for example 1,+2,-3

The format 2, or +2, denotes to use this hardware component in testing. The format -2 denotes not to use this hardware component in testing. The 1 and +1 formats are equivalent, and leaving a hardware component out of the list is equivalent to the -n format.

As an example, to select all the even processors the user would enter:

```
0,2,4,6,8,a,c,e
```

- 6=Node Selection—Allows the user to enter 7-bit node ids. The format is similar to that used for processors, i.e. an optional operator can be used and multiple entries are allowed. The following prompt is used:

```
Enter Node Ids:
```

- B=Loop Enable—Allows the user to modify the value of the loop enable flag, which causes the test controller utility to loop on all selected subtests when the last subtest is executed. The user is prompted for the new value as follows:

```
Loop Enable (0=disabled, 1=enabled):
```

- C=Loop Count—Allows the user to specify the number of times to loop through the selected subtests. It is only used if the "Loop Enable" flag is set. The user is prompted for the new value (a decimal number) as follows:

```
Loop Count:
```

- D=Test Error Count—Allows the user to modify the maximum number of test errors that can occur before the test controller utility terminates execution of subtests on this processor. The user is prompted for the new value (a decimal number) as follows:

```
Test Error Count value (1-127, N=no limit):
```

A value of N means that there is no limit to the number of errors that can occur.

- E=Pause at Test Start—Allows the user to modify the pause at test start flag. This flag results in the test controller pausing the testing on this processor just prior to starting the process of determining the first subtest to execute. The user is prompted for the new value as follows:

```
Pause at Test Start (0=disabled, 1=enabled):
```

- Selection F - Pause at Test End—Allows the user to modify the pause at test end flag. This flag results in the test controller pausing the testing on this processor after last subtest has

completed execution and all cleanup is complete. The user is prompted for the new value as follows:

```
Pause at Test End (0=disabled, 1=enabled):
```

- G=Pause at Subtest Start—Allows the user to modify the pause at subtest start flag. This flag results in the Test Controller pausing the testing on this processor just prior to starting the execution of the current subtest. The user is prompted for the new value as follows:

  ```
  Pause at Subtest Start (0=disabled,
  1=enabled):
  ```

- H=Pause at Subtest End—Allows the user to modify the pause at subtest end flag. This flag results in the test controller pausing the testing on this processor just after detecting the current subtest has completed execution. The user is prompted for the new value as follows:

  ```
  Pause at Subtest End (0=disabled, 1=enabled):
  ```

cxtest

5

The cxtest program is a graphical front end and a command line interpreter for the test controller. It is a standalone program that runs independently of any diagnostic tests loaded in the EEPROM on the Utilities board.

Overview

The cxtest program runs on the test station and communicates with the test controller via the NVRAM configuration parameters on the Utilities board. Depending on the command line, cxtest either starts the graphics display or runs as a command line interpreter.

The test controller must be in the standalone mode in order for cxtest to be able to communicate with it. To run the test controller in standalone mode, run the tc_standalone script.

When cxtest is invoked, it first retrieves system information from NVRAM and EEPROM on the Utilities board. This information includes:

- Tests loaded
- Parameters required for those tests
- Hardware configuration

The cxtest program works with the test controller to execute tests based on the options selected by the user. It performs the following functions:

- Looping
- Dispatching tests
- Configuring hardware
- Retrieving error information from the test controller

The test controller operates in the standalone mode when running in conjunction with cxtest. This is true whether one is using the command line version of cxtest or the graphics interface.

Graphics interface

To start the `cxtest` graphics interface, specify the `-d` option on the command line as follows:

```
% cxtest -d
```

This causes `cxtest` to open a window on the display. Where the window is displayed is set by the environment variable $DISPLAY. This cannot be changed on the command line.

The window has two areas of importance:

- Menu selections
- Test information display

Menus

There are six main menus in `cxtest`. Figure 23 shows the cxtest menu bar.

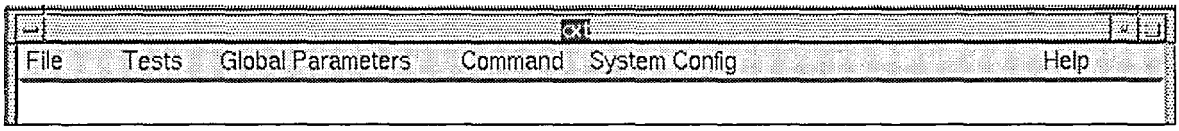

Figure 23 `cxtest` menu

File menu

The File menu has the following options:

- Save Selections
- Restore Selections
- Log to File/Close Log File
- Clear Log
- Exit

The Save Selections and Restore Selections options save and restore specific tests or configurations, respectively. With the Restore Selections option, the user can run specific tests without having to go through all the button pushing.

The Log to File option saves the information that appears in the display area to a file. When invoked, the option changes to Close Log File. Selecting this option terminates the save operation, and the menu option reverts to Log to File.

The Clear Log option clears the display area on the screen but does not truncate the log file if the Log to File option is enabled.

The Exit option closes `cxtest`.

When exiting `cxtest`, the state of the Boot option is set to what is displayed in the System Configuration menu. The default is to return to OBP, so if the user intends to return to `cxtest`, make sure the test controller standalone option is checked. See Figure 25.

Global Test Parameters menu

The Global Test Parameters menu contains the following controls for setting parameters that are not test specific:

- Looping
- Parallel execution of tests
- Pausing
- Continue on error controls

Clicking this menu option opens the window shown in Figure 24.

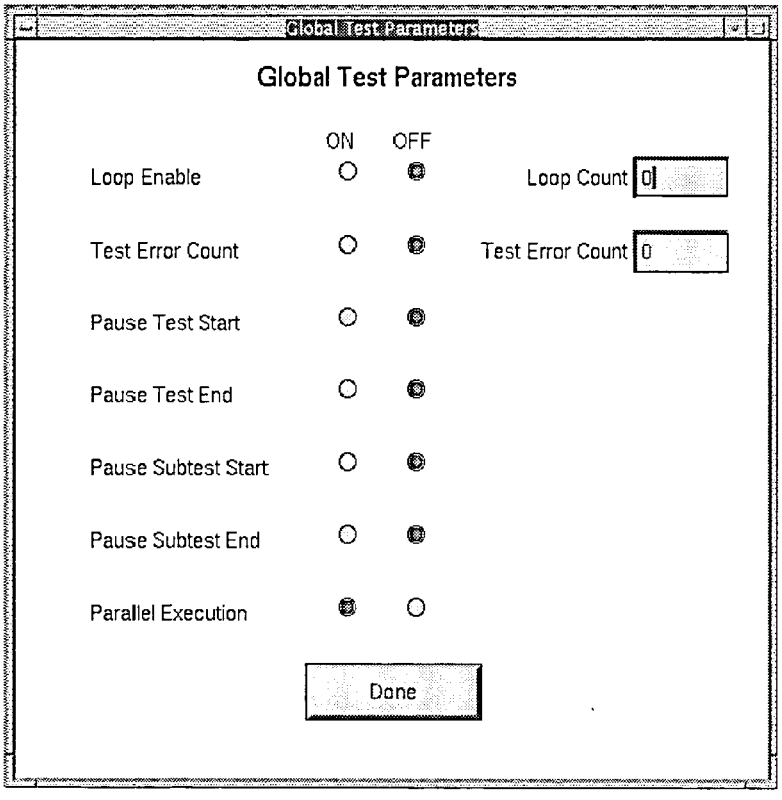

Figure 24 `cxtest` Global Test Parameters menu

Command menu

The Command menu is used to perform actions on the node or complex being tested. These actions include:

- Go
- Reset Machine
- Read Boot Config Map

The Go option starts the selected tests.

The Reset Machine option resets the system.

The Reading the Boot Config Map option is necessary if the physical location of the boards being tested changes. It lets the user keep `cxtest` running while the node is powered off and boards are moved to a different location.

System Configuration menu

The System Configuration menu displays all nodes that were online at the time `cxtest` started. Clicking one of the menu entries opens a node configuration window (Node x Configuration window) that allows the user to select the hardware to test, excluding I/O-specific devices such as disk drives or PCI adapters. See Figure 25.

Any hardware selections made on this screen apply to all tests to be run. For example, to run test A with hardware configuration A and then test B with hardware configuration B, the configuration must be changed manually after test A is completed. The information contained in the hypernode configuration window is extracted from the boot configuration map. If this map changes, it can be reread using the Command menu.

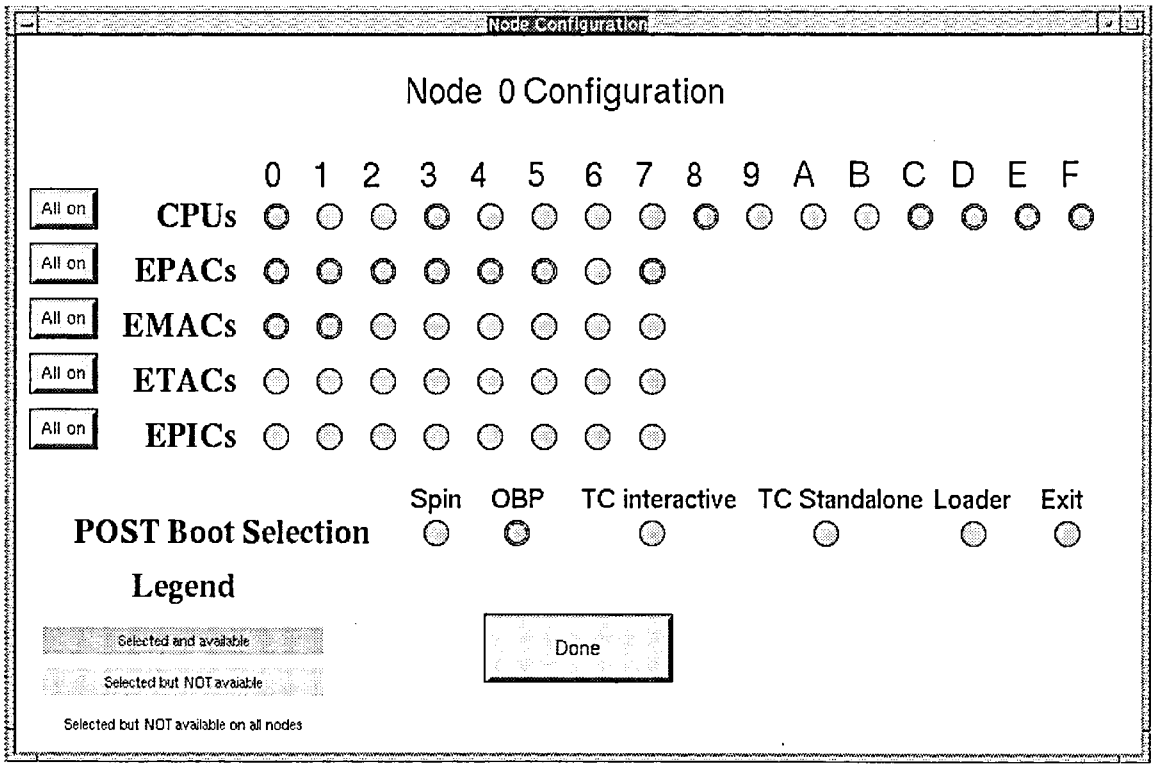

Figure 25 System configuration window

Help menu

The Help menu has two entries: About and Contents. The About selection displays the version number of `cxtest` running and the Contents selection opens a browser that can scroll through the help file.

Test menu

Selecting a test from the Test menu opens a window like the one in Figure 26. The Test menu varies depending on the tests loaded. If there is only one test loaded in EEPROM, then only one test appears in this window. The test names are presented as they appear in the EEPROM.

Selecting a test opens a window that displays all classes for the test. See Figure 26. Down the left hand side of the window are a column of round buttons, and down the right hand side of the window are two columns of buttons.

The left column of buttons allows the user to select all the subtests in the class with a single click. The button will turn yellow when selected.

The round buttons on the right hand side select subtests (opening another window for theses choices) within that class. If only some of the subtests in a class are selected, then both the round buttons for that class turn yellow.

Consider the round button on the left as in indication that tests within the class have been selected and the round button on the right as an indication that only some of the tests have been selected, as opposed to all the tests of that class.

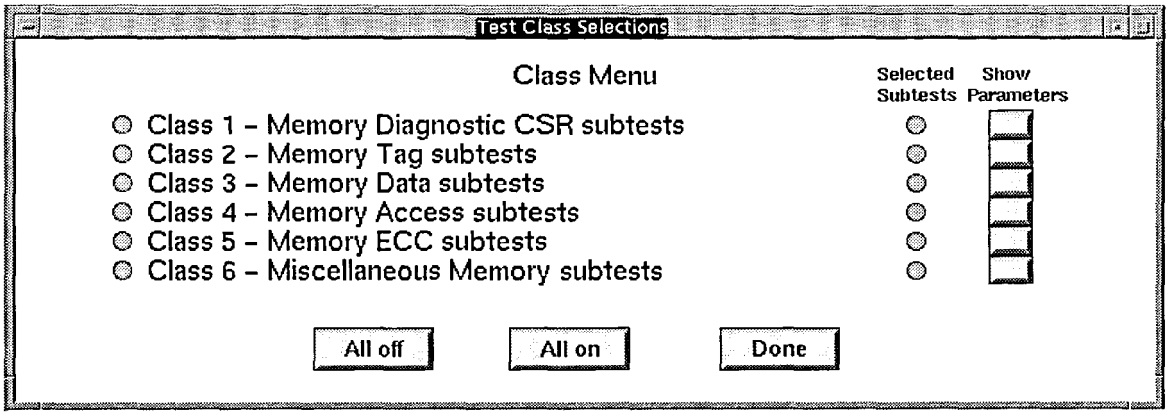

Figure 26 Test Class Selection menu

The last button on the right hand side associates parameters with the test class. Clicking this button opens another window with the parameters for that class. If there are more parameters than will fit on the screen, a scroll bar allows the user to scroll the list.

Also above and below each of the scroll bars are two buttons that allow the user to "page" up or down through the parameter list.

The Defaults button installs test default values into all the parameters.

If a class of tests has no parameters associated with it, the right most button (the square one) is not shown.

Display area

The display area shows the output of the tests. This output consists of messages that indicate when the tests start, the amount of time that the test has be running, and any error information.

The user can not cut and paste from this area. To record what transpires in test session, use the Log to File option.

When a test is started, a large ABORT button appears. Use this to escape out of a long test sequence or to stop a test not configured correctly. After using the ABORT button, reset the node from the Command menu. Once the abort command has been sent to the test controller, the button will disappear. The abort button also disappears when the test completes successfully.

Powering down the system

When using `cxtest` in a troubleshooting environment, it is not necessary to exit and enter `cxtest` each time the power is cycled.

To remove power to the system (for example, to move a board), power the system down leaving `cxtest` running. Make sure that no tests are actively running.

Once power is restored, POST returns control to the test controller in the standalone mode. The user must also wait for the `ccmd` routine to regenerate the database. A message will appear on the test station console stating the database generation is complete.

After the database is regenerated, the Boot Configuration map must be read using the Command menu. If this is not done, `cxtest` will not have the correct hardware configuration information.

Command line interface

The command line interface allows scripting of a test sequence using any preferred shell or programming language.

All the options presented here are covered in the man page for `cxtest`. The man page presents a complete list of all options. On the test station enter:

```
% man cxtest
```

The command line interface also requires information be retrieved from the Utilities board. There is, therefore, overhead for starting the tests. It is recommended that the user test as much as possible on one command line before changing the configuration or margins. It is possible to specify more than one test on the command line and change the parameters for different tests as well.

Command line options

The command line should be built around the following model:

```
cxtest [loading options] <test> [parameters] [looping/pausing/control]
[class/subtest selections] <test> [parameters] [looping/pausing/control]
[class/subtest selections]...etc.
```

This model is described in the following sections.

Loading options

Table 8 shows the command line loading options.

Table 8 Command line loading options.

Loading options	Description
-f <load_file_name>	Use the <load_file_name> to read what tests are to be loaded in to `cxtest`.
-r	Force the information need for `cxtest` to be read from the EEPROM.
default	If no load options are specified, the file `/spp/data/cxtest`.load is read to know what tests to load.

Tests

Table 9 shows the tests that can be selected.

Table 9 Test Selection available in EEPROM

Test selections	Description
-cpu	Cpu3000 - tests only cpu module
-mem	Mem3000 - tests main memory
-arch	Arch3000 - tests architectural features
-io	Io3000 - tests io devices
-intra	Intra3000 - tests intrahypernode coherency
-inter	Inter3000 - tests interhypernode coherency

These tests are dependent on the code loaded in the EEPROM. The user is expected to know the classes and subtests for a specific test. One way to see what tests are available is to peek inside the class

and subtest data files mentioned in the "Speed ups" section on page 63.

Parameters

The user may specify the parameters for the test on the command line using the `-paXX <value>` switch, where XX is the parameter number (range 0-19) and <value> is the number for that parameter. These are specified on a test-by-test basis. As an example, the command line could look like:

```
% cxtest -mem -pa1 0xffeeccdd -c 5 -io -pa1 0x1234eeff -s 300
```

This is an example of how to set parameters.

See the man pages on each of the tests to know what value to set each of the parameters.

Looping/pause/control

Table 10 describes options that are available in the looping, pause, and control of `cxtest`.

Table 10 Looping, pause, and control options

Looping/Pause/ Controls	Description
-pe <ON | OFF>	Pause at end of subtest
-pb <ON | OFF>	Pause at beginning of subtest
-ps <ON | OFF>	Pause at beginning of class
-pt <ON | OFF>	Pause at end of class
-ls <number>	Execute <number> of loops of the test that follows
-lt <number>	Execute <number> of loops of all the tests that follow
-t <number>	Switches the test controller to run on processor <number>. (range 0-15)
-mt <number>	Allows specification of error count

One additional parameter, the `-t` option, is somewhat out of place, but it effects the control of the operation of the test and is, therefore, included.

Class and subtest selections

Table 11 shows the options used to select classes and subtests from a test.

Table 11 Class and subtest selections.

Class/subtest	Description
`-c x,y-z`	Class selections can be separated by commas and/or dashes. `y-z` indicates select class `y` through `z`. Selections separated by commas form a list.
`-s yyy-zzz,xxx,vvv`	Subtest selections can be separated by commas and/or dashes. `y-z` indicates select class y through `z`. Selections separated by commas form a list.
`-x yyy-zzz,xxx`	Exclude subtest selections can be separated by commas and/or dashes. `y-z` indicates select class y through `z`. Selections separated by commas form a list.
`-y x,y-z`	Exclude class selections can be separated by commas and/or dashes. `y-z` indicates select class `y` through `z`. Selections separated by commas form a list.

The test controller remains in the standalone mode after completing a command line execution of `cxtest`. To return to the interactive mode of the test controller, press the TOC button on the system, and then exit the test controller, once the interactive menus have been displayed.

Test output

Test progress and error information is displayed just as in the graphics interface, with the exception that the information is displayed on the terminal the test was started from. There is no logging to a file with this interface, so the invocation of `cxtest` should capture standard out and standard error into a file if the log is desired.

Speed ups

One mechanism for speeding up the invocation of `cxtest` is the load file.

By default the load file /spp/data/cxtest.load is read. This file might look like the following example:

```
/spp/data/arch3000.class

/spp/data/arch3000.subtest

/spp/data/cpu3000.class

/spp/data/cpu3000.subtest

/spp/data/io3000.class

/spp/data/io3000.subtest

/spp/data/mem3000.class

/spp/data/mem3000.subtest

/spp/data/intra3000.class

/spp/data/intra3000.subtest

end
```

The load file passes information about the tests in EEPROM to `cxtest`. If the user wishes only to load information about one test, the file should contain only those files that pertain to that test. Information in the .class and .subtest files have to match what is loaded in to the ROM. The files that are provided in /spp/data match the latest release of the tests. That is not to say that `cxtest` will not run if it does not have the correct file, just that if a subtest has been eliminated or added to the ROM information for a specific test, the disk version of the files may need to be updated also.

One way to speed up the start-up time is to make a copy of the cxtest.load file and just include the test of interest. Use the `-f <filename>` option for select a specific start-up test file.

If the user wants to force the information to be read from the Utilities board, the `-r` option must be used. This is true for both the command line and the graphics interfaces. This operation takes several minutes to complete.

See Table 8 for all the loading options.

mem3000

6

Introduction

This chapter describes `mem3000`, a memory test for S-Class and X-Class systems. `mem3000` requires a node with a minimum of one processor with two memory boards.

The following test configuration parameters must be set in order for the test to properly execute:

- Memory boards (EMBs) must be installed in multiples of two.
- Only one EPB can be enabled per pair of EMBs.
- Only one EPB can be enabled per PAC.

To prevent false errors from being reported, the user must set up parameters 4, 5, and six to correspond to the actual memory configuration.

Classes and subtests

`mem3000` consists of a series of tests grouped together in classes beginning with verification of the most basic functionality and progressing toward more complex functionality. Each class is broken down into subtests which target specific functionality.

The following sections describe the classes and individual subtests.

Classes

mem3000 has six classes of tests shown in Table 12.

Table 12 mem3000 test classes

Class	Description
1	Verifies the operation of the diagnostic CSRs on each EMB.
2	Verifies the tag field.
3	Verifies the data field.
4	Verifies the various coherent and noncoherent transactions.
5	Verifies the ECC.
6	Verifies miscellaneous memory capabilities.

Classes 1, 4, 5, and 6 can be configured to test a single EMB. Classes 2 and 3 use memory interleave and thus test over the range of EMBs.

Subtests

The mem3000 subtests are listed in Table 13.

Table 13 mem3000 subtests

Subtest	Description
100	Verifies the diagnostic CSRs can be written and read.
101	Verifies the other EMAC CSRs can be written and read.
110	Verifies data can be written and read on each bank using the diagnostic CSRs.
120	Verifies ECC can written and read on each bank using the dig CSRs.
130	Verifies the tag can be written and read on each bank using the dig CSRs.
140	Verifies memory lines on each bank can be initialized using the dig CSRs.
150	Verifies the first 32 memory lines of each EMB using various data patterns.

Table 13 mem3000 subtests—**(continued)**

Subtest	Description
200	Tag Bank Test.
210	Verifies the tag portion of a memory line using an addressing pattern.
211	Verifies the tag portion of a memory line using a byte uniqueness pattern.
230-238	Verifies the tag portion of a memory line using the MarchC algorithm and different patterns.
300	Verifies the memory lines on each bank canbe written and read using coherent operations.
310	Verifies the data portion of a memory line using an addressing pattern using coherent operations.
311	Verifies the data portion of a memory line using a byte uniqueness pattern using coherent operations coherent operations.
330-338	Verifies the data portion of a memory line using the March C algorithm and different patterns.
400	Verifies loads and stores transactions to memory.
410	Verifies data flush transactions to memory.
420	Verifies noncoherent transactions to memory.
500	Verifies ECC single bit data and tag errors are detected, logged, and corrected.
510	Verifies ECC double bit data errors are detected and logged using coherent operations.
520	Verifies ECC double bit data errors are detected and logged using noncoherent operations.
530	Verifies that ECC errors are ignored when disabled.
600	Verifies the memory system detects and reports accesses to all illegal and/or invalid memory space.
610	Verifies the memory system detects and reports error conditions when the memory tag state is ERROR.
620	Verifies the various mode bits in the EMAC System Configuration register.
640	Determines whether 80-bit or 88-bit DIMMs are installed.

Subtests in classes 5 and 6 will produce HPMCs (indicated by the test controller printing the # character). These are expected.

User parameters

The test controller provides `mem3000` with up to 20 user parameters. Current parameters are defined in Table 14.

Table 14 `mem3000` test parameters

Words	Usage
0/1	64-bit user pattern 0 used in subtests 238 and 338 (defaults=0xa5a5a5a5/0xa5a5a5a5)
2/3	64-bit user pattern 1 in subtests 238 and 338 (defaults=0x5a5a5a5a/0x5a5a5a5a)
4	Last bank to test, inclusive (default=3)
5	Last row to test, inclusive (default=7)
6	Denotes 88-bit DIMMs are installed (default=1)
7	First bank to test, inclusive (default=0)
8	First row to test, inclusive (default=0)
9	Denotes if test is to run with errors disabled (default=0)

Using mem3000

The user can run `mem3000` from either the test controller command line or from the `cxtest` window. The following sections show the user how to use `mem3000`, following the example scenario:

- Configure mem3000 to run on a system with four EMBs. installed
- Set the classes and subset to be executed.
- Run the tests.
- View the results.

Configuration

To execute the scenario, perform the procedures in this and the following sections:

Step 1 From the test controller main menu shown in Figure 27, select the Test Selection Menu, option A.

```
MAIN Menu commands

       0=Quit Test Controller
       1=Begin Test Controller Execution
       2=Halt Test Controller Execution
       3=Resume Test Controller Execution
       4=Switch CPU
       5=POST Boot Selection
       6=Execution Mode Selection
       7=Global Parameter Display
       8=CPU Summary Display
       9=Display CPU Errors
       A=Test Selection Menu
       B=Test Configuration Menu
       C=Debugging Menu
```

Figure 27 Selecting Test Selection option from TC main menu

Step 2 From the Test Selection menu shown in Figure 28, select Memory test, option 1.

```
Test Selection Menu
    1=*Memory test
    2= Architecture Features test
    3= Intra-Node Coherency test
    4= Inter-Node Coherency test
    5= I/O test
    6= CPU Selftests
    7= not available
    8= not available
    9= not available
    A= not available
```

Figure 28 Selecting Memory test from Test Selection menu

This opens the Test Configuration menu.

Step 3 From the Test Configuration menu shown in Figure 29, select the Hardware Selection menu, option A.

```
Test Configuration Menu
    0=Return to Main Menu              9=Display Test Parameters
    1=Display Classes                  A=Hardware Selection Menu
    2=Display Subtests                 B=Loop Enable
    3=Select Classes                   C=Loop Count
    4=Select Subtests                  D=Test Error Count
    5=Read All Test Parameters         E=Pause at Test Start
    6=Read One Test Parameters         F=Pause at Test End
    7=Write Test Parameter             G=Pause at Subtest Start
    8=Reset Parameters                 H=Pause at Subtest End
```

Figure 29 Selecting Memory test from Test Selection menu

Step 4 From the Hardware Selection menu shown in Figure 30, select CPUs, option 1.

```
Test Configuration Menu - Hardware Selection
Display
   0=Return to Test Configuration Menu
    1=CPU Selection
    2=EPAC Selection
    3=EMAC Selection
    4=ETAC Selection
    5=EPIC Selection
     6=Node Selection
```

Figure 30 Selecting CPUs from Hardware Selection menu

Step 5 At the following prompt:

```
Select CPUs: 0 2
```

Select the number of processors (CPUs).

After the number of processors is chosen, the Hardware Selection menu reappears.

Step 6 From this menu select Return to Test Configuration Menu, option 0.

From the Test Configuration menu, the user can select option 9 to view the changes.

Selecting classes and subtests

To select test classes and subtest, perform the following procedure:

Step 1 From the Test Configuration menu, select Display Subtest, option 3.

Step 2 From the following prompt, select the test classes:

```
Enter class number:
```

Step 3 From the Test Configuration menu, select Display Subtests, option 2.

The subtest menu shown Figure 31 in lists all available subtests.

```
100*   Diagnostic CSR Read/Write Test
101*   Other EMAC CSR Read/Write Test
110*   Memory Data Read/Write Test
120*   Memory ECC Read/Write Test
130*   Memory Tag Read/Write Test
140*   Memory Initialization Test
150*   First 32 Memory Lines Test
200*   Tag Bank Test
210*   Tag Addressing Test
211*   Tag Byte Uniqueness Pattern Test
230*   Tag March-C Pattern #1 Test
231*   Tag March-C Pattern #2 Test
232*   Tag March-C Pattern #3 Test
233*   Tag March-C Pattern #4 Test
234*   Tag March-C Pattern #5 Test
235*   Tag March-C Pattern #6 Test
236*   Tag March-C Pattern #7 Test
237*   Tag March-C Pattern #8 Test
238*   Tag March-C User Data Pattern Test
300*   Memory Bank Test
310*   Memory Addressing Test
311*   Byte Uniqueness Pattern Test
330*   Memory March-C Pattern #1 Test
331*   Memory March-C Pattern #2 Test
      0xbfbfbfbfbfbfbfbf and
      0x404040404040404
332*   Memory March-C Pattern #3 Test
333*   Memory March-C Pattern #4 Test
334*   Memory March-C Pattern #5 Test
335*   Memory March-C Pattern #6 Test
336*   Memory March-C Pattern #7 Test
337*   Memory March-C Pattern #8 Test
338*   Memory March-C User Data Pattern Test
400*   Memory Load/Store Test
410*   Memory Data Flush Transaction Test
420*    Memory Semaphore Transaction Test
500*   ECC Single Error Correction Test
510*   ECC Double Error Detection (coherent
520*   ECC Double Error Detection
      (non-coherent)
530*   ECC Disable Test
600*   Memory Access Protection Test
610*   Memory Tag Test I
620*   EMAC System Configuration CSR Test
640*   80 vs. 88 Bit DIMM Test
```

Figure 31 mem3000 Subtests menu

Step 4 Select all appropriate subtests. Table 15 lists the test patterns for subtests 230 through 238.

Table 15 Test patterns for subtests 230 through 238

Subtest	Patterns
230	0x7f7f7f7f7f7f7f7f 0x8080808080808080
231	0xbfbfbfbfbfbfbfbf 0x4040404040404040
232	0xdfdfdfdfdfdfdfdf 0x2020202020202020
233	0xefefefefefefefef 0x1010101010101010
234	0xf7f7f7f7f7f7f7f7 0x0808080808080808
235	0xfbfbfbfbfbfbfbfb 0x0404040404040404
236	0xfdfdfdfdfdfdfdfd 0x0202020202020202
237	0xfefefefefefefefe 0x0101010101010101
238	0xa5a5a5a5a5a5a5a5 0x5a5a5a5a5a5a5a5a

Selecting Display Subtests, option 2, from the Test Configuration Menu reflects the changes.

Starting tests

To run the tests selected from the test controller main menu, select Begin test controller Execution, option 1. Figure 32 shows the output.

```
% Enter command: 1

  Execution Starting.
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................
  Execution Completed
```

Figure 32 Example of mem3000 execution

Viewing the results

To review the results of the test, select CPU Summary Display, option 8, from the Main menu.

Figure 33 shows the results of the example.

```
MAIN Menu - CPU Summary Display
  Total Failures = 00000000

Configuration Map
=================
 CPUs : 0* 1* 2* 3* 4* 5* 6* 7* 8  9  A* B* C  D  E  F
 EPACs: 0* 1* 2* 3* 4  5* 6  7
 ERACs: 0  1  2  3
 EMACs: 0* 1* 2  3  4  5  6* 7*
 ETACs: 0  1  2  3  4  5  6  7
 EPICs: 0  1  2  3  4* 5  6  7

                     FAIL
CPU STATE            COUNT SUBTEST TEST NAME
=== =====            ===== ======= =========
0  Test Completed      0      150  MEM3000 - EEPROM based memory tests
1  Idle               n/a    n/a   n/a
2  Test Completed      0      150  MEM3000 - EEPROM based memory tests
```

Figure 33 Example CPU summary display

Error codes

When a failure is encountered, an event code is set along with an error message. The least significant 12 bits of the event code contain the error code. Table 16 lists the mem3000 error codes.

Table 16 `mem3000` error codes

Code	Description
001	Diagnostic address CSR miscompare occurred (in upper 32-bits).
002	Diagnostic address CSR miscompare occurred (in lower 32-bits).
003	Diagnostic data CSR miscompare occurred (used only by class 1).
004	Diagnostic data CSR miscompare occurred (in upper 32-bits).
005	Diagnostic data CSR miscompare occurred (in lower 32-bits).
008	Miscompare occurred in the upper 32-bits of the CSR.
009	Miscompare occurred in the lower 32-bits of the CSR.
010	Memory data miscompare occurred.
011	Memory data miscompare occurred (upper 32-bits).
012	Memory data miscompare occurred (lower 32-bits)
013	Memory data matched when it shouldn't have (upper 32-bits).
014	Memory data matched when it shouldn't have (lower 32-bits).
020	Miscompare occurred in the upper 32-bits of the tag.
021	Miscompare occurred in the lower 32-bits of the tag.
022	The tag changed when it shouldn't have.
030	ECC data miscompare occurred.
031	An ECC error was logged when it shouldn't have been.
032	EMAC did not correct the single bit ECC failure as expected.
033	EMAC did not log the occurrence of a single bit ECC failure.
035	EMAC did not log the occurrence of a double bit ECC failure.
040	Data miscompare error occurred in sequence 1 of MarchC test (upper 32-bits).
041	Data miscompare error occurred in sequence 1 of MarchC test (lower 32-bits).
042	Data miscompare error occurred in sequence 2 of MarchC test (upper 32-bits).
043	Data miscompare error occurred in sequence 2 of MarchC test (lower 32-bits).
044	Data miscompare error occurred in sequence 3 of MarchC test (upper 32-bits).
045	Data miscompare error occurred in sequence 3 of MarchC test (lower 32-bits).
046	Data miscompare error occurred in sequence 4 of MarchC test (upper 32-bits).

Table 16 `mem3000` error codes—**(continued)**

Code	Description
047	Data miscompare error occurred in sequence 4 of MarchC test (lower 32-bits).
060	A semaphore operation did not trigger.
070	Incorrect data returned for a semaphore operation.
080*	Incorrect info in EMAC error CSRs (single bit data ECC - read).
090*	Incorrect info in EMAC error CSRs (single bit tag ECC - read).
0a0*	Incorrect info in EMAC error CSRs (double bit data ECC - read).
0b0*	Incorrect info in EMAC error CSRs (double bit data ECC -coh_inc op)
0c0*	Tag state did not equal ERROR as it should have.
0d0*	Tag state did not equal INVALID as it should have.
0e0*	An unexpected error was detected in the EMAC error CSRs
100*	Uninstalled Memory.
110*	Invalid CSR.
120*	Network Cache
130*	Unprotected Memory.
140*	Alternate Interleave.
200	Denotes the EMB contains all 80-bit DIMMs.
201	Denotes the EMB contains all 88-bit DIMMs.
202	Denotes the EMB contains a mixture of 80-bit or 88-bit DIMMs are installed.

The asterisks next to the error codes listed in Table 16 actually indicate a range of events as shown in Table 17.

Table 17 Extended range for error codes

Code	Meaning
code+1	Error cause CSR miscompare error (upper 32-bits)
code+2	Error cause CSR miscompare error (lower 32-bits)
code+3	Error info CSR miscompare error in the error type field
code+4	Error info CSR miscompare error in the ENUM field
code+5	Error info CSR miscompare error in the cc/msg field
code+6	Error address CSR miscompare error (upper 32-bits)
code+7	Error address CSR miscompare error (lower 32-bits)
code+8	Error info CSR miscompare error in the syndrome field.

An error message can have one of four formats. The first format type is shown in Figure 34.

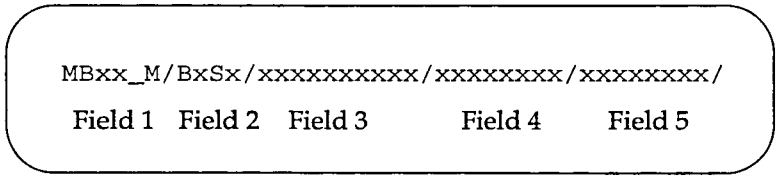

Figure 34 Error message format one

There are five fields separated by / symbols. Table 18 lists the meaning of each field.

Table 18 Error message field description for error format one

Field	Description
1	Specifies the EMB on which the failure was detected.
2	Specifies the DIMM on which the failure was detected.
3	Specifies the failing 40-bit address.
4	Specifies the actual 32-bits of data.
5	Specifies the expected 32-bits of data.

A second format type is shown Figure 35.

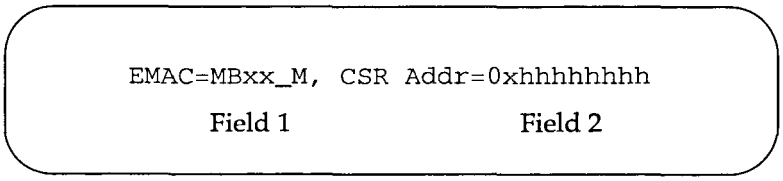

Figure 35 Error message format two

Table 18 shows the error message field description for this format.

Table 19 Error message field description for error format two

Field	Description
1	Specifies the EMB on which the failure was detected.
2	Specifies the failing 32-bit address of the CSR.

A third type of error format is shown in Figure 36. This format is used only by subtest 101 which tests the EMAC CSRs.

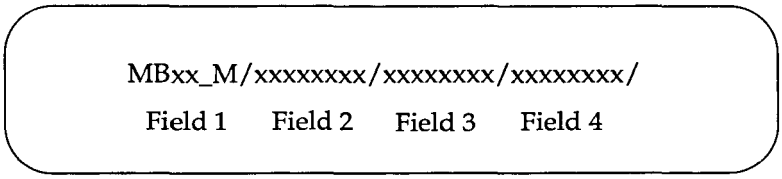

Figure 36 Error message format three

There are four fields separated by / symbols. Table 18 lists the meaning of each field.

Table 20 Error message field description for error format three

Field	Description
1	Specifies the EMB on which the failure was detected.
2	Specifies the failing 32-bit address of the CSR.
3	Specifies the actual 32-bits of data.
4	Specifies the expected 32-bits of data.

A forth type of format is used only by subtest 640 which determines what type of DIMMs installed on the first EMB specified. The format is shown Figure 37.

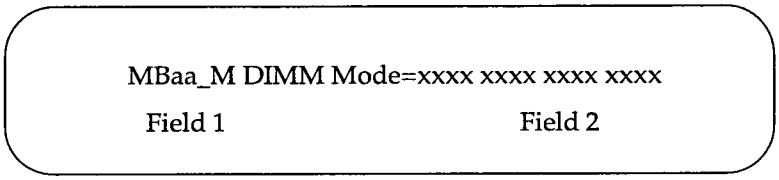

Figure 37 Error message format four

Table 18 shows the error message field description for this format.

Table 21 Error message field description for error format four

Field	Description
1	Specifies the EMB to which the information pertains.
2	Specifies the type of DIMM detected as follows: x—nonexistent DIMM 0—80-bit DIMM 1—88-bit DIMM

Utilities 7

This chapter details most of the diagnostic utilities which include:

- `sppconsole`
- `hard_logger`
- `pce_util`
- `load_eprom`
- `load_mem`
- `tc_show_struct`
- RDR dump utilities
- Version utilities
- Event processing
- Miscellaneous tools

sppconsole

The `sppconsole` script invokes the `/spp/etc/console` program to provide the operating system a console interface. Refer to the `console(8)` manpage for more information about this program.

The `console` program connects to a server running on the test station. This server is called `conserver` (console server) and more information about it can be found in the `conserver(8)` manpage.

This interface communicates with the test controller, OBP, and operating system.

To start console interface, type the following command in a test station window:

```
% sppconsole
```

After making the connection, the last 20 lines of the console output are displayed.

All information and error messages are logged into the /usr/adm/syslog system error log file.

hard_logger

`hard_logger` is a script that invokes the interrogators and extractors to log all error information on a node

The usage of the script is:

```
hard_logger [node number]
```

`[node number]` is a hex number.

As an example, entering **`hard_logger 0`** runs `hard_logger` on node 0.

Entering **`hard_logger`** on the command line without arguments, invokes the `hard_logger` man page.

Two arguments, `-h and -H`, also invoke the `hard_logger` man page.

The `hard_logger` script performs the following tasks:

- Parses the command line arguments to determine on which node it should run
- Acquires COP information for the node using `ccmu cop $node` and saves the output to /tmp/hl/COP_FILE_n$node
- Acquires PCE information using `pce_util $node` and saves the output to /tmp/hl/T_FILE_n$node
- Checks the Stop On Hard bits of each PACs to find one that is running
- Reads MUC CSRs.
- Traverses the list of hard error buses. If a bus reports a hard error, then it performs the following:
  - Interrogates each controller on that bus for hard errors.

    If the hard error group pin is set to a one value, it ignores the controller.

    If the pin is a zero value, the controller may have been the first to record the error.
  - Interrogates the controller reporting the error.

    To interrogate the controllers, `hard_logger` calls the asic-specific interrogator located in /spp/scripts/<asic>.

    For example, the MAC interrogator is located in /spp/scripts/emac)

    The interrogator returns a list of extractors to run on that asic in /tmp/hl/inter_n$node.
  - Runs each extractor returned by the interrogator

- Prints the results to STDOUT.
- Logs the results in /spp/data/hard_hist

  hard_hist is a permanent file which records the date, time, and results of the script.

Since STDERR updates the user on progress and syntax errors and STDOUT contains user-readable output from the program, the output of `hard_logger` can be redirected to a file and saved.

pce_util

The pce_util utility performs the following:

- Resets the Utilities board
- Display status of the Utilities Board

It processes the command line in order from left to right, sending scan messages to the appropriate registers to perform the clock, power, or other environmental changes and/or to report current status.

The following describes `pce_util` usage:

```
pce_util [-C [number]][-n N N N] [-d] [-b] [-s] [-c s (nue)] [-r X]
[-t X] [-l] [-p (lnu) 1 2 3 4 All]
```

Table 22 describes the `pce_util` options.

Table 22 `pce_util` options

Options	Description
`C [number]`	Displays the current complex name.
`-n N [N...N]`	Specifies the nodes for the pce_util commands.
`-d`	Displays the current state.
`-b`	Toggles the blinking state of the LED.
`-s`	Suppresses the output.
`-c s[ource] [nue]`	Manipulate clocks on the current nodes.
`n[ominal]`	Nominal frequency.
`u[pper]`	Upper frequency.
`e[xternal]`	External connector.
`d[isplay]`	displays current node status.
`-l`	Produce the long text description of state.
`-t X`	Set the TOC flag to X.

Table 22 pce_util options—(continued)

Options	Description
-r X	Set the Power flag to X.
-p [lnu] [1234 or All]	Manipulate power on the current nodes.

Note

Power supply 4 may not be present on all machines.

With no options, pce_util defaults to displaying the current clock, power, and environmental state for each node under test.

The power and clock setpoints are fixed in hardware, and can not be changed with this utility.

The pce_util utility normally returns a zero value, even if some voltages or temperatures are out of tolerance. If a serious error occurs, pce_util returns a nonzero value

The following are examples of how to use the pce_util utility and some of the output generated by it.

Figure 38 shows the output of the following command example.

```
% pce_util -n 0
```

```
pce_util:   new node list: 0
Complex name is Default_complex
Node Clocks          LEDS    @C U SHPT Supply 1 Supply 2 Supply 3 Supply 4
---- ------ ------------- ---- ------ -------- -------- -------- --------
  0  Normal     ATTN 0x68    28 1 0000  Nominal  Nominal  Nominal  Nominal
```

Figure 38 Example of pce_util with -n 0 options set

Figure 39 shows the output of the following command example.

```
% pce_util -n 0 -p n all
```

```
pce_util:   new node list: 0
Complex name is Default_complex
Node Clocks          LEDS   @C U SHPT Supply 1 Supply 2 Supply 3 Supply 4
---- ------ ------------ ---- ------ -------- -------- -------- --------
  0  Upper       0x00       27 1 0000    Lower    Lower    Lower    Lower
```

Figure 39 Example of `pce_util` with `-n 0 -p n` all options set

Figure 40 shows the output of the following command example.

```
% pce_util -n 0 -c s u
```

```
pce_util:   new node list: 0
data is 1 to write clock
Complex name is Default_complex
Node Clocks          LEDS   @C U SHPT Supply 1 Supply 2 Supply 3 Supply 4
---- ------ ------------ ---- ------ -------- -------- -------- --------
  0  Upper       0x00       27 1 0000    Lower    Lower    Lower    Lower
```

Figure 40 Example of `pce_util` with `-n 0 -c s u` options set

To disable the blinking LED caused by a failing power supply, use the following commands:

```
% pce_util -n0 -r1
% pde_util -n0 -r0
```

Figure 41 shows the output of the following command example.

```
% pce_util -l
```

```
pce_util:   new node list: 0
        Complex:        Default_complex
        node:     0
        clock margin:    Normal
        LED display:          0x00
        Temperature:       26
        Hardware Status:
              Utility reset state:    1
              SOFT reset state   :    0
               HARD reset state   :    0
               48V Pwr reset state:    0
               TOC reset state    :    0
        Power Supply 1:  Nominal
        Power Supply 2:  Nominal
        Power Supply 3:  Nominal
        Power Supply 4:  Nominal
```

Figure 41 Example of `pce_util` with `-l` option set

load_eprom

The `load_eprom` utility resides on the test station platform. It downloads the core firmware products into the EEPROM on the Utilites board through the scan interface. It can also update the JTAG scan interface controller firmware. If, during a download, it detects any errors, it automatically retries the download.

The `load_eprom` utility uses subroutines that perform the following functions:

- It reads a raw binary file on the test station.
- It erases the specified Flash sector and verifies that the erase was successful. It will retry if the erase fails.
- It scan downloads the contents of the binary in 4096-byte page increments, updating the screen for each page. A "w" is printed during the write operation, an "r" during the optional read operation, a "v" during the optional verify operation and a "." when the page is complete.
- It can optionally read each page back for verification.
- It can read-verify a binary in the Flash EEPROM and compare it to the binary on the test station, without performing the write operation.

The `load_eprom` utility usage is as follows:

```
load_eprom -n <node> [-QRV] [-P #] [-j|c|e|p|o|t] <file>
```

The -n <node> specifies the target node name.The options available are given in Table 23.

Table 23 `load_eprom` options

Option	Description
-Q	Quiet (no) output mode.
-R	Read and verify data only-No writing.
-P number	EPAC to use for scan operations where number is 0-7, 8 is UBUS.
-V	Verify data after a write.
-j <file>	Load binary into JTAG flash.
-c <file>	Load binary into JTAG_CORE flash.
-e <file>	Load binary into PDC Entry section.
-p <file>	Load binary into PDC POST section.
-o <file>	Load binary into PDC OBP section.
-t <file>	Load binary into PDC Test Controller section.

As an example, entering the following reads the file /spp/firmware/post.fw and updates the POST section of Flash EEPROM on the Utilities board.

```
xns3_d% load_eprom -n mu_0000 -j jtag.fw
```

Entering the following reads the file ./jtag.fw and updates the Flash EEPROM for the JTAG controller.

```
xns3_d% load_eprom -n mu_0000 -p /spp/firmware/post.fw
```

Figure 42 shows the output generated by using `load_eprom` in three ways. The first two writes to one sector in EEPROM and the last writes across several sectors.

```
% load_eprom -n mu_0000 -p entry.pdc

new_load_eprom                    000.000.000.204 1996/11/27 13:57:39 donc
Reading file "entry.pdc": 4253 (0x109d) bytes read.
Using default EPAC (P0L).
Erasing sector 0  (0xf0000000) OK
Writing sector 0  (0xf0000000) .. OK

% load_eprom -n mu_0000 -p post.fw

new_load_eprom                    000.000.000.204 1996/11/27 13:57:39 donc
Reading file "post.fw": 92820 (0x16a94) bytes read.
Using default EPAC (P0L).
Erasing sector 4  (0xf0020000) OK
Writing sector 4  (0xf0020000) ..................... OK

% load_eprom -n mu_0000 -o obp.pdc

new_load_eprom                    000.000.000.204 1996/11/27 13:57:39 donc
Reading file "obp.pdc": 499712 (0x7a000) bytes read.
Using default EPAC (P0L).
Erasing sector 7  (0xf0080000) OK
Writing sector 7  (0xf0080000) ............................... OK
Erasing sector 8  (0xf00a0000) OK
Writing sector 8  (0xf00a0000) ............................... OK
Erasing sector 9  (0xf00c0000) OK
Writing sector 9  (0xf00c0000) ............................... OK
Erasing sector 10 (0xf00e0000) OK
Writing sector 10 (0xf00e0000) ......................... OK
```

Figure 42 Example of `load_eprom` output

While `load_eprom` is writing a block of data, a "w" is printed. If the write is successful, a dot is printed. The dots continue until the whole sector is successfully written, at which time the "OK" is printed.

load_mem

The load_mem utility loads files or raw data into memory. It can be used in the absence of any other program loading device. Utility usage is as follows:

```
load_mem <filename> [-check] [-n <node number in hex>] [-r <ring
number in hex>] [-raw] [-sa <starting address>] [-ea <ending address>]
[-h] [-help]]
```

<filename> must be present for loading. The file is loaded using calls to scn_mem_wr.

Table 24 shows the options for load_mem.

Table 24 load_mem **utility options**

Option	Description
-check	Enables checking of written data via calls to scn_mem_rd.
-n <hex number>	Indicates which node to access.
-p <hex number>	Indicates which EPAC to use. If not specified, get_default_pac generates the ring number.
-raw	Indicates that <filename> is a raw binary. If this option is used, the starting and ending addresses must also be provided.
-sa <hex number>	Indicates the starting address for placement of the raw binary: <filename>.
-ea <hex number>	Indicates the ending address for terminating the raw binary: <filename>.
-s <hex number>	Specifies the size in words of the raw binary.
-help	Display the help file and prompt user for list of error codes.
-h	Any word that begins with an h displays help file.

The user can call load_mem on an a.out type file. It uses the series 700/800 implementation of a.out.h to correctly parse and load the file.

The ending address is the next address after the end of where the raw binary is loaded. The starting and ending addresses must be formatted for excalibur addressing and read in full words.

tc_show_struct

The `tc_show_struct` tool examines certain structures that the test controller uses to set up and run tests. The structures examined are:

- `tc_global_parameter_struct`
- `tc_test_info_struct`
- `tc_cpu_info_struct`

The `tc_test_info_struct` structure displays the following fields:

- The entry point
- Class pointer
- Subtest pointer
- Hardware requirements
- Test specifics
- Parameter pointer
- HW requirements met
- Test initiated
- Test selected
- Run classes
- Run subtests

The `tc_global_parameter_struct` structure displays the master state.

The `tc_cpu_info_struct` structure displays the status or state of each processor and the current subtest.

The tc_show_struct tool takes two arguments: the first is the test of interest, the second is the node of interest.

Possible selections for the test are:

- `-mem`
- `-io`
- `-inter`
- `-intra`
- `-cpu`
- `-arch`

An example of the output is shown in Figure 43.

```
exhmu4_d% tc_show_struct -io
1 Nodes found
-------------------------------------------------------------------------
Name : io3000 - EEPROM based I/O tests
Entry Pt      ClTb ptr     StTb ptr      HwReq      ParmTbptr  Parm_ptr
-------------------------------------------------------------------------
0xf01c8000  0xf01c806c  0xf01c8f60  0xf01c8064  0xf081c2e8  0xf0808438
-------------------------------------------------------------------------
Hardware req met = 1 | Test inited   = 1
Selected = 1         | TC State = TC_RUNNING
-------------------------------------------------------------------------
Class[0]    = 0    Subtest[0] = 645
Class[1]    = 0    Subtest[1] = 0
Class[2]    = 0    Subtest[2] = 0
Class[3]    = 0    Subtest[3] = 0
Class[4]    = 0    Subtest[4] = 0
-------------------------------------------------------------------------
Current Values for Parameters
00) 0x00000000 01) 0xf0000000 02) 0x00000000 03) 0x00000000
04) 0x00000000 05) 0x00000001 06) 0x00000000 07) 0x00000000
08) 0x10405090 09) 0x105050a0 10) 0x116051b0 11) 0x118051c0
12) 0x0ff00ff0 13) 0x0ff00ff0 14) 0x0ff00ff0 15) 0x0ff00ff0
16) 0x0ff00ff0 17) 0x0ff00ff0 18) 0x0ff00ff0 19) 0x0ff00ff0
-------------------------------------------------------------------------
CPU Mask  = 0x0a0a     EPAC Mask = 0xff
EMAC Mask = 0xc3       ETAC Mask = 0x00
EPIC Mask = 0x33
-------------------------------------------------------------------------
CPU 0  - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 1  - State - TC_CPU_DONE                   Subtest 645
CPU 2  - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 3  - State - TC_CPU_DONE                   Subtest 645
CPU 4  - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 5  - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 6  - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 7  - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 8  - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 9  - State - TC_CPU_DONE                   Subtest 645
CPU 10 - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 11 - State - TC_CPU_DONE                   Subtest 645
CPU 12 - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 13 - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 14 - State - TC_CPU_NOT_AVAIL              Subtest 0
CPU 15 - State - TC_CPU_NOT_AVAIL              Subtest 0
```

Figure 43 Example of `tc_show_struct` output

RDR dump utilities

This set of utilities allows the user to dump, format, and display the RDR state within a processor. They reside in the /spp/unsupported directory and consist of the following:

- `rdr_dumper.fw`
- `rdr_formatter`
- `scan_sram`

The following procedure reads, formats, and displays RDRs:

Step 1 In the test controller window, select POST Boot Selection option from the main menu.

Step 2 Select the Dumper option and press **Return**.

Step 3 Exit the test controller by selecting Main menu option 0. This resets the system. When POST displays "Booting RDR Dumper," wait approximately 10 seconds for the `rdr_dumper` utility to complete. As an alternative, read the four words starting at location 0xf0804000 via `sppdsh`. As each processor completes, it sets its corresponding byte in this array.

Step 4 Run the `/spp/unsupported/scan_sram` script. This script scans the dumped data into files kept in the /tmp directory. Instead of executing the entire script, you may execute only those lines for the processors under test by copying those particular lines from this script into a sppdsh shell.

Run the `rdr_formatter` utility on the desired file as follows:

```
/spp/unsupported/rdr_formatter /tmp/cpuX.dump
```

where X is the cpu ID 0-F.

This dumps output to `STDOUT` which can be redirected to a file.

Step 5 Press the node TOC button or type the `assert_toc 0` command in an `sppdsh` window. This resets the system and returns the system to the OBP prompt.

Version utilities

This section describes the three version utilities.

diag_version

The `diag_version` utility displays the product name and the version of the current Exemplar Diagnostic package. For example:

```
$ diag_version

Exemplar Diagnostics, Version 1.0.1.0
```

flash_info

`flash_info` is a utility that probes the contents of the EEPROM on the Utilities board and displays the product name, version number, author, and date the product was built.

It needs only one argument, the node number of the Utilities board to be examined. If `flash_info` is used without an argument, the first node found (from `ccmd`) is the node queried.

The entrypoints of the standard products are checked for the *magic header*. If no magic header is found, the product is assumed not to be loaded. The one exception is OBP.

OBP does not have the magic header. Therefore, no information can be interpreted from `flash_info` with respect to OBP.

The entrypoints are checked in the order that they appear in the ROM.

An example of the output is shown in Figure 44.

```
rough:/sppdiag/src/doc/sppa/cxtest
exhmu3_d% flash_info 0

Node : 0

  Program Name          Version          Date         Author
-------------------------------------------------------------
post                  001.000.000.007  1996/10/15   mullins
cpu3000               001.000.000.000  1996/10/10   mullins
No version string
No version string
No version string
test_controller       001.000.000.000  1996/10/10   mullins
mem3000               001.001.000.000  1996/10/14   blair
arch3000              001.000.000.000  1996/10/10   mullins
intra3000             001.000.000.000  1996/10/10   mullins
No version string
io3000                001.000.000.000  1996/10/10   mullins

rough: 24 >
```

Figure 44 Example of flash_info output

ver

`ver` is a test station version retriever utility. It is used to read and display the version information built into each diagnostic product. Its usage is:

```
ver <file>
```

`ver` searches the specified file for a version string and extracts and displays the version and date stamp, as shown in the following example:

```
% ver test_controller
PDC test_controller 001.004.000.017 1997/03/12 15:06:12 blair
```

Event processing

This section discusses three event processing utilities:

- `event_logger`
- `event_browser`
- `log_event`

event_logger

`event_logger` is an RPC server that takes requests, (events) and writes them into the event_log file(/spp/data/event_log). `Event_logger` also can become a client to other programs that want to receive events and operate on them; an example is CERS.

There are two sources for events: a compiled program on the test station and the operating system. The mechanism for receiving the events is a little different, but once they are received, the events follow the same path.

`event_logger` is a daemon process and should never terminate. If it does so, the test station will spawn a new copy. There is a 25 second wait between starting the program and becoming a daemon and being able to receive events. This delay simplifies problems when the program is started by `init` after a reboot of the test station.

Events generated by compiled programs on the test station use a library call `log_error` to establish the RPC link and send the event. In the case of the operating system events, `event_logger` accepts UDP datagrams and formats them into an event. This implies that for operating system events, it is possible to miss an event since there is no guarantee of connection with UDP.

The event log itself is a plain text file that can be examined with any paging or editing tool. When the event log reaches one MByte in size, the file is closed, copied over to event_log.old and compressed. A new event log is then opened. This preserves the history of events.

The `event_logger` has only one option, the `-d` option that prevents the `event_logger` from becoming a daemon, so the debugger can attach and start the program.

The format of the 32-bit event code is shown in Figure 45

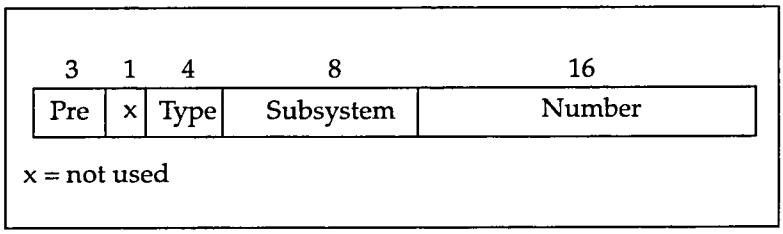

Figure 45 32-bit event code format

The fields of the event code format are defined as follows:

- ***Pre*** field—Preamble that indicates severity of the event.
- ***Type*** field— source of the event (e.g. a utility or the kernel).
- ***Subsystem*** field—Qualification of what sent the event
- ***Number*** field—Unique identifier for a particular event (for example, power failing).

Table 25 gives the values of the preamble fields in the event code.

Table 25 Values for the preamble field in the eventcode

Values	Meaning	Description
0	Normal	Request complete - no abnormalities).
1	Debug	Used for debugging.
2	Info	Informative.
3	Warning	Request successful, but no normal (example: retry).
4	Error	Request was unsuccessful, but contained.
5	Critical	Conditions with possible widespread consequences.
6	Emergency	Requires external intervention.
7	Panic	Last gasp.

Table 26 gives the values for the type field in eventcode.

Table 26 Values for type field in eventcode

Values	Meaning	Description
0	Unknown	Source is unknown.
1	Firmware	Source is OS firmware type.
2	Microkernel	Source is from the kernel.
3	Server	Source is from a server.
4	Unix utility	Source is from a utility.
5	TS diags	Source is a test station diagnostic.
6	Node diags	Source is processor-based diagnostic.
7	MU firmware	Source is the MU firmware.

Table 27 gives the values for the subsystem field in the eventcode.

Table 27 Values for the subsystem field in the eventcode

Value	Description
0x40	Miscellaneous MU firmware originated
0x41	HP T-chip SW
0x42	Other Primary load T-chip SW
0x43	Other Diag T-chip SW
0x44	TS communication SW
0x45	Diagnostic clock controller related
0x46	Diagnostic scan related
0x47	Diagnostic config related
0x48	Diagnostic memory op related
0x49	Diagnostic miscellaneous abnormal condition
0x4a	Diagnostic utility related
0x4b	Diagnostic test related
0x4c	Power/environmental related
0x4d	CXTS

Table 27 Values for the subsystem field in the eventcode

Value	Description
0x4e	CERS
0x4f	PDC firmware related

event_browser

event_browser is a text based utility that reads events from either a default or specified event log and displays the ones that meet the specified criteria to stdout. It has the following format:

```
event_browser [-D] [-a ``time"] [-b ``time"] [+/-c number [number...]]
[+/-e number [number...]] [-f <log file>] [+/-g ``phrase"] [-i] [+/-l
number [number...]] [-n] [-q] [-r] [+/- s number [number...]] [+/-t
number [number...]] [-v]
```

event_browser has a command line and an interactive mode. All options may be executed from either interface. In command line mode, the options specified on the command line are evaluated, then a report is generated, followed by event_browser terminating. In interactive mode, event_browser prompts for additional lines of commands (generating a report after each line), rather than exiting.

Interactive mode is entered by including -i on the command line. Options hold their value across reports unless changed or reset.

Table 28 shows the event_browser options.

Table 28 event_browser options

Option	Description
-D	Increment debug flag - for development use only.
-a ``time"	Display only events occurring after this time.
-b ``time"	Display only events occurring before this time.
+c number [number...]	Display only events with the specified condition codes (32 bits) [up to 256].
-c number [number...]	Display all events except those with the specified condition codes (32 bits).
-d	Display differences in event times.

Table 28 `event_browser` options—**(continued)**

Option	Description
`+e number [number...]`	Display only events with the specified event numbers (16 bits) [up to 256].
`-e number [number...]`	Display all events except those with the specified event numbers (16 bits).
`-f <log file>`	Read events from this log file.
`+g ``phrase"`	Display only events that contain the specified phrase.
`-g ``phrase"`	Display all events except those that contain the specified phrase.
`+i`	Interactive mode - the log is not read initially and additional commands are read from `stdin`.
`-i`	Interactive mode - additional commands are read from `stdin`.
`+l number [number...]`	Display only events with the specified severity field values.
`-l number [number...]`	Display all events except those with the specified severity field values.
`+n`	"sniff" starting at the end of the log. It ignores existing events. **Cntl-C** stops sniff.
`-n`	"sniff" the end of the log when reached. **Cntl-C** terminates sniffing.
`-q`	Terminate execution, the current line is not processed.
`-r`	Reset severity, type, subsystem, and event sniff options (to all events).
`+s number [number...]`	Display only events with the specified subsystem values [up to 256].
`-s number [number...]`	Display all events except those with the specified subsystem values.
`+t number [number...]`	Display only events with the specified type field values [up to 256].
`-t number [number...]`	Display all events except those with the specified type field values.

Several of the options allow for two ways of displaying the information: either "all except the specified" or "the specified only."

A minus sign (-) causes all events except those specified to be accepted, and a plus sign (+) causes only those events with the particular characteristic to be accepted. The options are mutually exclusive.

Only the last specification for each field applies. When a date is specified, the form should be the same as used in the event message, for example ``Wed Dec 16 14:07:30 1992." If any part of a date is missing, the missing parts will be extrapolated from the previous setting.

When in interactive mode, `event_browser` reads a line from `stdin` and processes it as if it were a command line (minus the executable name). After the options on each line are processed, a report is generated. If the `-n` option is specified, the log is "sniffed" until a **Cntl-C** is received (at which time sniffing terminates). A `-quit` causes sniffing to terminate without processing any further options.

`event_logger` returns a zero value if no error was detected, and a nonzero value if a fatal error occurred. Nonfatal errors result in a message being printed to `stdout`.

log_event

`log_event` accepts input from standard in and packages it into an event. The event number is specified in one of three ways:

- Command line
- First line of the input
- Default

`log_event` is used by scripts such as the interrogators and extractors to put information into the event log as follows:

```
log_event [-c] [number]
```

If the `-c` option displays event information output on the console as well. If the event severity is high enough, this happens automatically. `event_logger` displays any events that have a severity greater than the warning level.

The following two examples show how `log_event` can be used:

```
cat data_file | log_event 0x86340001
```

This example puts an event in the event log with the event code of 0x86340001. The data will be the information contained in the file data_file.

```
echo "This is a test event" | log_event
```

This example puts the message "This is a test event" in the event log with the default event code from `log_event`.

Miscellaneous tools

The following miscellaneous tools are described in this section:

- `ds1620`
- `ecc_calc`
- `kill_by_name`

ds1620

The `ds1620` utility reads from or writes to the temperature sensing device. This utility determines when the system should shut down due to an over-temperature condition.

The following is the usage of this utility:

```
ds1620 [-h] [-n node number | All] [-q] [-c] [-b] [-e] [-s <l T>|<h T>|<c X>]
```

The following produces a help message displaying all commands:

```
ds1620 -h
```

The following produces the node number, the temperature, and the set points with a description. The `-q` option prints the results without the header:

```
ds1620 -n [number or All] [-q]
```

The following sets all nodes lower set point to T (temperature) and displays the results:

```
ds1620 -n [number or All] [-q] -s l T
```

The following sets nodes higher set points to T and displays the results:

```
ds1620 -n [number or All] [-q] -s h T
```

The following sets nodes higher set points to X and displays the results:

```
ds1620 -n [number or All] [-q] -s c X
```

The following reads the configuration register:

```
ds1620 -n [number or All] [-q] -c
```

The following starts temperature monitor mode:

```
ds1620 -n [number or All] [-q] -b
```

The following ends the temperature monitoring mode and displays the results:

```
ds1620 -n [number or All] [-q] -e
```

ecc_calc

`ecc_calc` is a calculator that accepts hex input and calculates the error correction code (ECC) value for that data. It is presently written to work on full 88-bit memory lines but can be used for single node (80-bit) data as well. To calculate ECC on 80-bit data, enter zeros for the last byte of data.

An example of the use of this script is shown below:

```
$ecc_calc
enter data[8:87]: 0123456789
 ecc_low_out = 'h 2c
 ecc_high_out = 'h 69
 ecc_out data[0:7] = 'h 45
enter data[8:87]:
```

To exit this utility, type **q**.

kill_by_name

The `kill_by_name` script kills processes by name rather than by process identification. The following is the usage of this script:

```
kill_by_name <file name> <signal to send> <process id to not kill>
```

Table 29describes the options in kill_by_name.

Table 29 `kill_by_name` options

Option	Description
`file name`	Process name to kill.
`signal to send`	Default is kill command.
`process id to not kill`	Kills all processes the match the specified name except the one matching this ID.

If the third argument is used, the second argument must also be specified. For example:

```
kill_by_name foo 15 1234
```

cbus

The cbus utility performs read, write, and erase operations on the core logic address space. All addresses are 32-bit in hexadecimal format. The format of this command is:

```
cbus -n <number> -p <number> [-r <addr> | -e <addr> | -w <addr> <data>}
```

Table 30 lists the cbus options.

Table 30 cbus options

Option	Description
-n <number>	the node to use
-p <number>	the EPAC to use
-r <addr>	reads and displays the data at the specified address
-e <addr>	erases the sector at the specified address
-w <addr> <data>	writes the 32-bit data value into the specified address

An example is as follows:

```
$ /spp/unsupported/cbus -n 0 -p 0 -r 0xf0d00000 0xfebfffc
```

fix_boot_vector

This sppdsh script restores the four words at the beginning of NVRAM to point to POST. These four words are used by the ENTRY firmware to determine which process was executing last when an HPMC, TOC, or reset occurs.

stop_on_hard

This sppdsh script allows the user to enable, disable, and check the stop-on-hard bit in the option ring. Its usage is as follows:

```
stop_on_hard <node number> <on|off|chk>
```

where <node number> is zero.

The output of the `chk` option is shown in.

```
EPACs: p0l=0xff p1r=0x0 p2l=0x0 p3r=0x0 p4l=0x0 p5r=0x0 p6l=0x0 p7r=0x0
ERACs: r0l=0x0 r2r=0x0 r3r=0x0 r1l=0x0
EMACs: mb0l_m=0x0 mb1l_m=0x0 mb6r_m=0x0 mb7r_m=0x0
EPICs: iolf_b=0x0 iolf_a=0x0
ETACs:
```

Figure 46 Output from `stop_on_hard` with `chk` option

A value of 0x80 denotes that stop-on-hard is enabled whereas a 0 denotes that it is disabled.

Excalibur scan test

8

The Excalibur scan test (est) is a diagnostic utility that uses the system scan hardware making it possible to perform connectivity tests and to test gate array internals. The est utility runs on the optional test station and sends scan instructions to a given node via Ethernet.

est utility test environment

est is started on the teststation and is located in/spp/bin/est. The user has the option of either starting up a user interface or having the est utility run a script.

est works on one node at a time by sending scan instructions and data and receiving the results over the diagnostic ethernet connection.

Since est has to communicate closely with the Utilities board, other diagnostics can not be run at the same time. Also, while est is moving data through the scan rings, the operating system can not be running.

Running est

The following is the command line usage for est:

```
est [-options] <node_number>
```

For example, to test node 0, enter:

```
% est 0
```

est reads configuration information from files stored in /spp/data (e.g node_0.cfg). These configuration files are automatically generated by ccmd each time the system is powered up. While ccmd is running, it prints its status to the console window. When database generation is complete and no errors were reported, it writes the necessary configuration files and est may be executed.

Table 31 shows `est` command line options.

Table 31 `est` command line options

Option	Description
`-v`	Print version and exit.
`-f <filename>`	Run a given script file.
`-l`	do not generate a log file.
`-o <filename>`	Redirect the log file to the given filename.

Some examples of `est` usage are:

```
est -v
```

```
est -l -f my_script 0
```

```
est -o ./my_log_file 0
```

The `est` utility relies on certain data and vector files located in the /spp/est directory.

Unless disabled or redirected, the `est` utility will generate a log file, est.log, and store it /spp/data/est.log. Any previous log file will be renamed to est.log.old

Figure 47 shows an example of output using `est`.

```
% est mu_0000 0
 Excalibur Scan Test    Vr. 0.17.0     28-AUG-96     Steven Terry
sdp_send: node = start, fd = 4
************************************************
REQUEST:
        1100800d 10010009 089a786e 73315f64
        00
fragment 1 of 1
sequence: 0
bytes:    13
compression flag :  0
response flag: 1
Utility Command
Utility command: reserve hw
sdp_recv: node = start, fd = 3

************************************************
RESPONSE:
        11000000 8002e000
fragment 1 of 1
sequence: 0
status:    0
bytes:    2
compression flag :  0
data:
        e000
.......................
.............Memory Group: design
             11 Memory blocks manage the allocations
        2883573 Bytes alloc'ed from the system
        2751176 Bytes (95 %) used by the program

General EST Tests:
a       ... board level ac tests
d       ... board level dc tests
g [ [dev file [limit] ] ... gen_ga_patterns( )
sso dev ... sso test for a given device

Special Scan Tests:
b       ... bypass/id test
c       ... compare id's to config file
i       ... print id's found in design

EST Options:
D       ... design query menu
F       ... set option & debug flags
Q       ... quit, not so nice
q       ... quit nicely
h       ... print this help message
v       ... print EST version info
!cmd    ... send the command to Unix (ex. "!ls patterns")
```

Figure 47 Example of `est` output

est tests

`est` works on the JTAG scan rings throughout the system. Tests provided are:

- Ring (test command `r`)—Moves data through the scan rings to make sure the rings are connected and that basic scan hardware is operational.
- DC connectivity (command `d`)—Checks that wires on the boards between scan devices are intact (no shorts or opens).
- AC connectivity (command `a`)—Examines wires on the boards. AC tests look for timing problems between parts at full speed. If DC connectivity patterns passed, but AC connectivity failed, the failure is bound to be timing related.
- Gate array (command `g`)—Executes scan tests internal to selected arrays. When these tests fail, the array usually has to be replaced.

These tests are listed in the order in which they should normally be run.

est commands

The `est` commands are described in the following sections.

AC Connectivity test

The AC Connectivity test format is:

```
a [-s -p #]
```

Table 32 shows the options for the this test.

Table 32 AC Connectivity test options

Option	Description
`-s`	Step mode (for debug purposes).
`-p <number>`	Run pattern number only.

Bypass test

The Bypass test format is:

```
b
```

The Bypass test places the scan ring hardware into bypass mode.

DC Connectivity test

DC Connectivity test format is:

```
d [-s -p #]
```

Table 32 shows the options for the this test.

Table 33 DC Connectivity test options

Option	Description
`-s`	Step mode (for debug purposes).
`-p <number>`	Run pattern number only.

Gate Array test

The Gate Array test format is:

```
g [options] [pattern file]
```

Table 34 shows the options for the this test.

Table 34 Gate Array test options

Option	Description
`-r <refdes>`	Test arrays with matching a reference designator value.
`-b <board>`	Test arrays on given board. `<board>` may either be a number or a name.
`-j <jtag_id>`	Test arrays matching a jtag_id.
`-t <type>`	Test an array type (For example, RAC).
`-s <number>`	Start with a given pattern number.
`-e <number>`	End on a certain pattern number.
`-m <number>`	Run a maximum of `<number>` patterns per file.
`-o <number>`	Optimization level (0, 1, or 2) Two is the most optimized and is the default.

By default, the `g` command tests all arrays. When the `-r`, `-b`, `-j`, or `-t` options are used, only arrays that meet all criteria are tested.

Gate array tests use test vectors that have been pregenerated for the certain arrays (each array has multiple files associated with it). The `-s`, `-e`, and `-o` options can be used to limit the number of patterns that are run for each pattern file. The default is to run all

patterns in each file. While it may take more time to run all patterns, using options to limit the number of patterns may result in a significant loss of test coverage.

The `-o` option controls either run time optimization or how much parallelism takes place. There are three levels:

- 0—No optimization
- 1—Same parts on the same ring are tested together
- 2—Identical rings are tested in parallel

If an error is encountered during parallel testing, the following message may appear:

```
*** errors found - must switch to serial testing ***
```

When an error occurs, parallel scans into the scan hardware may result in bus conflicts on TDO pins. Therefore, `est` automatically stops using parallel scans when errors happen.

SCI test

`sci` is a utility that tests the Coherent Toroidal Interface (CTI) cables between nodes. The term SCI (Scalable Coherent Interface) is often used in place of the term CTI; the terms are interchangeable.

The usage of `sci` is as follows:

```
sci [driver] [receiver] ring test
```

where:

`[driver]`

Refers to the node and memory board to which the CTI cable is connected and from which the test data originates.

`[receiver]`

Refers to the node and memory board to which the other end of the CTI cable is connected and receives test data.

`ring`

Refers to the CTI ring associated with the cable, either x or y.

`test`

Refers to the specific test: dc, dc_clk, ac. With the dc test, the clock from the receiver node is used. The dc_clk test derives its clock from the cable.

The following is an example of `sci` usage:

```
% sci 0 mb01 1 mb01 y dc_clk
```

This command runs the dc_clk test on the cable connected between node 0, memory board 1 (driver) and node 1, memory board 1 receiver.

JTAG Identification test

The JTAG Identification test prints all JTAG IDs. the format is:

```
i
```

Margin commands

The Margin command for clocks and power format is:

```
m [-c | -p [supply] [value]
```

The `-c` option specifies the clock, and the `-p` option specifies power.

When a value is not supplied, the current states are displayed.

By itself, `m` show all margins.

`-c high` displays the upper clock limit.

`-p 1 nom` sets the supply 1 margin to nominal.

Miscellaneous commands

This section gives the following useful commands entered at the `est` prompt:

- **`ms`**—Puts all the scan hardware into a safe state
- **`q`**—Quit
- **`script <file>`**—Runs a file contain est commands
- **`v`**—Prints version information
- **`F`**—Opens the flags submenu

Flags and options

There are flags and options that control how the tests operate. Entering **`F`** at the `est` prompt brings up the Flags submenu. Pressing return at the flags prompt returns to the main `est` prompt.

Some of the more useful flags and options are:

- `l`—Limits the number of internal array patterns that get executed by the g command. This has the affect of decreasing coverage to approximately 90%.
- `s`—Stops testing when an error is detected
- `A <number>`—Limits AC Connectivity testing
- `D <number>`—Limits DC Connectivity testing.
- `E`—Shows `sdp` packets transferred across the ethernet
- `P`—Controls whether or not the pass/fail status of individual patterns are displayed

Script files

There are two ways of running script files: from the command line (**`-f <filename>`**) or from the `est` prompt. From the command line, `est` executes the instructions listed and when finished, displays the `est` prompt. To cause `est` to quit when the script is finished, put **`q`** at the end of the script file.

The script command reads `est` commands from an ASCII file and runs those commands. The following rules apply to the file:

- The file must have only one command per line.
- Command syntax must be the same as entered at the `est` prompt.
- Comments lines must start with a **#** sign.

An example file might contain the following lines:

```
# check the rings
r
# show pattern pass/fail steps
F P
#limit dc testing to 3 patterns
F D 3
#do dc testing
d
```

Index

A

AC Connectivity test 108
AC test of a hypernode 8
associated documents xiv

B

Boot Configuration map 59
Bypass test 108

C

C 109
cbus 102
cbus options 102
ccmd 15, 16, 17, 93, 105
ccmu 17, 20
  commands 18
CERS 16, 94
clock margining 8
console ethernet 4
controllers
  MUC 1, 5
  PUC 2, 5
COP 5
COP chip 28
core logic
  DUART 4
  flash memory 4
  initialization 30
  nonvolatile SRAM 4
CTI
  cache 22, 27
cxtest 53
  class and subtest selections 62
  command line interface 59
  Command menu 56
  File menu 54
  Global Parameters menu 55
  graphics interface 54
  Help menu 57
  loading options 60
  looping, pause, and control 61
  parameters 61
  powering down the system 59
  speeding up start 63
  System Configuration menu 56
  Test Class Selection menu 58
  Test menu 57

D

DC Connectivity 109
dcm script 33
diag_version 92
diagnostic database 16
ds1620 100
Dual Universal Asynchronous Receiver-Transmitter (DUART) 4, 5, 30

E

ecc_calc 101
ECUB 3.3-Volt error 12
EEPROM 30
environmental conditions 7
environmental control 7
environmental error interrupt 7
environmental errors 6
environmental monitoring functions 10
environmental sensors 1
environmental warning 11
errors
  control of 55
  ECUB 3.3-Volt 12
  mem3000 error codes 75
  mem3000 error message formats 77
  mem3000 extended error codes 77
  MUC-detected 10
  power-on detected 10
  reporting 29
est 105
  AC Connectivity test 108
  AC Connectivity test options 108
  Bypass test 108
  command line options 106
  commands 108
  DC Connectivity test 109
  DC Connectivity test options 109
  flags and options 111
  Gate Array test 109
  Gate Array test options 109
  JTAG Identification test 111
  margin commands 111
  miscellaneous commands 111
  running 105

script files 112
tests 108
utility test environment 105
ethernet 9
event code format 95
event_browser 94, 97
event_browser options 97
event_logger 94, 99

F

Field-Programmable Gate Array (FPGA)
MUC 3, 5, 7, 13
PUC 3, 4, 5, 7, 13, 27, 29
fix_boot_vector 102
flash memory 4
flash_info 93
Front panel LCD 10

G

Gate Array test 109

H

hard reset 28
hard_logger 16, 81, 82
headings 15
first level, H1 15
hypernode
ASIC initialization 30
clean up and boot 31
configuration 30
environmental monitoring functions 7, 10
environmental sensors 1
error conditions 5, 7
ID 28
initialization 29
main memory initialization 30
power-on function 6, 7
power-on reset 29

I

initialization 29
interrupt processing
environmental error 7

J

JTAG 1, 15, 16, 17, 108
device IDs 16
Identification test 111
port 15, 16
scan interface 17
JTAG fanout 9
JTAG interface 1, 8

K

kill_by_name 101

L

LCD 2, 5, 27
LED 5
lists 111
load_eprom 81, 86
load_eprom options 87
load_mem 81, 89
load_mem utility options 89
log_event 94, 99, 100
looping 55

M

magic header 93
margin commands 111
mem3000 65
classes 66
command line 68
configuration 68
cxtest 68
error codes 75
error message formats 77
extended error codes 77
selecting classes and subtests 71
starting 73
subtests 66
Subtests menu 72
viewing the results 74
memory boards 65
miscellaneous tools 100
cbus 102
ds1620 100
ecc_calc 101
fix_boot_sector 102
kill_by_name 101
stop_on_hard 102
MUC 1, 5

MUC environmental monitoring 6

N

NonVolatile battery-backed RAM (NVRAM) 4, 38, 53
nonvolatile static RAM 4
NVRAM 37
NVRAM verification 30

O

OBP 81, 93
Open Boot PROM (OBP) 4, 27, 30
  boot process 31

P

packet 2
parallel execution of tests 55
pausing 55
pce_util 81, 83
pce_util options 83
PCI 56
POST 10, 15, 17, 19, 20, 23, 25, 27, 28, 29, 30, 33, 38, 39, 59, 92, 102
  boot selections 40
  modules 29
  program flow 32
power up reset 28
powering down the system 59
power-on circuit 5
power-on function 6, 7
power-on reset 29
power-on-detected errors 10
processor initialization and selftest 29
processor-dependent code (PDC) 4, 29
PUC 5, 28

R

RAM 4
RDR dump utilities 92
rdr_dumper.fw 92
rdr_formatter 92
reset
  hard 28
  powerup 28
  soft 28
  TOC 28

S

scan_sram 92
scanning 2
scn_mem_wr 89
script 111
script files 112
selftest 29
soft reset 28
sppconsole 81
sppdsh 33, 92, 102
STDERR 83
STDOUT 83, 92
stop_on_hard 102
system -check command 17

T

tc_cpu_info_struct 90
tc_global_parameter_struct 90
tc_show_struct 92
tc_test_info_struct 90
test controller 37, 38, 53, 81, 90
  debugging menu 44
  interactive mode 37, 62
  main menu 39
  modes 37
  stand-alone mode 37, 59, 62
  test configuration menu 45
  Test Selection menu 68
  test selection menu 44
  user interface 38
test station 15, 17, 93
test station interface 8
TOC 28

U

User interface 38
utilities
  cbus 102
  diag_version 92
  ds1620 100
  ecc_calc 101
  event_browser 97
  event_logger 94
  fix_boot_sector 102
  flash_info 93
  hard_logger 82
  kill_by_name 101
  load_eprom 86
  load_mem 89
  log_event 99

pce_util 83
rdr_dumper.fw 92
rdr_formatter 92
scan_sram 92
sppconsole 81
stop_on_hard 102
ver 93
Utilities board 1, 4, 5, 7, 13, 53, 59, 63, 83, 93, 105
utilities bus 5

V

ver 93
version utilities 92
diag_version 92
flash_info 93
ver 93
voltage margining 8

X

xconfig 20, 21, 24, 25
menu bar 22
node configuration map 23
node control panel 24

A4716-90002