CONVEX Processor Diagnostics Manual (C3800 Series)

First Edition

CONVEX COMPUTER CORPORATION

CONVEX
Processor Diagnostics Manual (C3800 Series)

Order No. DHW-303

First Edition
October 1992

J.K

CONVEX Press
Richardson, Texas
United States of America

CONVEX
Processor Diagnostics Manual (C3800 Series)

Order No. DHW-303

Printed in the United States of America

Revision Information for

CONVEX
Processor Diagnostics Manual
(C3800 Series)

Edition	Document No.	Description
First	760-006430-000	Released October 1992. A reference manual for the use of C3800 diagnostic utilities, diagnostic shell commands, diagnostic tests, and debug commands. Contains an introduction to the CONVEX Expert Test System (CXTS). Reflects diagnostics version 3.3.

FCC Notice

Note

This equipment generates, uses, and can radiate radio frequency energy. And, if not installed and used in strict accordance with the instruction manual, it may cause harmful interference to radio communications.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment.

When this equipment is operated in a residential area, it is likely to cause interference. In this case, the interference must be corrected at the operator's expense.

Do not connect external equipment to the utility outlets in CONVEX equipment cabinets. Unauthorized connection voids all agencies' emissions certification.

Contents

Figures

Tables

How to use this manual

Purpose and audience

This manual describes diagnostic tests for the CONVEX C3800 Series computers. This manual is intended for CONVEX field engineers and others with similar knowledge of CONVEX computers.

Scope

This manual applies to the CONVEX C3800 Series hardware and software. It refers to related documentation where adequate information is available in those documents.

Organization

This manual contains the following chapters:

- **Chapter 1, "SPU environment"**— Describes the CONVEX C3800 Series service processor (SPU) and sets out the working environment through which the field engineer performs diagnostic testing.
- **Chapter 2, "Diagnostic utilities"**—Describes the diagnostic utilities available for monitoring, configuring, and controlling a CONVEX C3800 Series computer.
- **Chapter 3, "Diagnostic shell (`dsh`)"**—Describes the ConvexOS shell and the powerful special commands available for troubleshooting.
- **Chapter 4, "User test interface"**—Describes the diagnostic X-Window environment available for diagnostic testing and the command set available for remote testing without the X-Window environment.
- **Chapter 5, "Service processor interface test (`spu4000`)"**—Describes the test that verifies the operation of the SPU-computer interface, computer scan rings, and circuit board connectivity.

- **Chapter 6, "Utilities subsystem test (cu4000)"**—Describes the test that verifies basic operation of the CPU utilities circuit board.
- **Chapter 7, "Memory subsystem test (mem4000)"**—Describes the test that verifies the memory subsystem and read/write paths.
- **Chapter 8, "CPU diagnostic tests"**—Describes the tests that comprehensively verify the operation of the computer central processing units (CPUs).
- **Chapter 9, "Troubleshooting with CXTS"**—Describes troubleshooting methods and introduces the CONVEX Expert Troubleshooting System (CXTS).
- **Chapter 10, "Using ddb"**—Introduces the debug utility (ddb) and describes some useful ddb commands.
- **Appendix A, "Diagnostic utilities man pages"**—Describes the man pages and contains a printout of them.
- **Appendix B, "C3800 Series boot process"**—Describes the ConvexOS boot process and gives a manual boot procedure.

Notational conventions

Notational conventions are systems of characters, symbols, terminology, or abbreviated expressions used to express technical facts or quantities as established by this manual. The following notational conventions are used in this document:

- **Bold constant-width** font indicates user-entered information for a computer program that should be entered exactly as it appears.
- Constant-width font indicates:
  - Information that appears on the monitor screen exactly as shown in text
  - Command names and options
  - Display examples and error messages returned
- *Italic* font indicates:
  - Emphasized items
  - User-supplied variables
  - Document titles
- Vertical ellipses indicate that lines have been omitted from an example.

- KEYCAP font indicates keys you press. Examples:
  - RETURN refers to the carriage return key
  - CONTROL-c indicates that you must hold down the CONTROL key and press c
- All CONVEX illustrations have an illustration file number at the bottom right-hand corner that is for CONVEX use only.

Warnings

The following is an example of a warning and its typical content and location as used in CONVEX documents:

Warning

A warning highlights procedures or information necessary to avoid injury to personnel. The warning immediately precedes the critical information and includes a description of the hazard.

Cautions

The following is an example of a caution and its typical content and location as used in CONVEX documents:

Caution

A caution highlights procedures or information necessary to avoid damage to equipment, damage to software, loss of data, or that leads to invalid test results. The caution immediately precedes the critical information and includes a description of the possible damage.

Notes

The following is an example of a note and its typical content and location as used in CONVEX documents:

Note

A note highlights information of a supplemental nature. The note immediately precedes or follows the highlighted information.

Ordering documentation

To order the current edition of this or any other CONVEX document, send requests to:

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851 USA

Include the order number with the request. The order number is on the title page of the manual and begins with the letters "DHW" or "DSW."

Technical assistance

Hardware, software, and documentation support can be obtained through the CONVEX Technical Assistance Center (TAC):

- From locations in the continental United States:
  - Customers call (800)952-0379.
  - CONVEX employees call (800)545-4839.
- From locations in Canada, call (800)345-2384.
- From all other locations, contact the nearest CONVEX office.

Electronic mail

The Processor Documentation Group has an email address for documentation comments. Use this service to give us a quick response for special documentation questions that should be addressed immediately. For technical questions, contact the Technical Assistance Center, as described in the "Technical assistance" section of this preface. To use the email response service, just send mail addressed to:

```
cnvxhwdoc@convex.com
```

When using the electronic mail service, please provide the following information:

- The reader's name and company name
- A return email address in INTERNET notation or UUCP (bang) notation
- The manual that is being critiqued
- The chapter and page number in question
- Your comment

SPU environment

1

The C3800 Series SPU is a SPARC workstation with 16 Mbytes of internal memory, a hard disk drive, and CONVEX proprietary interfaces to a C3800 Series computer. Some SPUs also have a floppy disk drive. Its operating system (SPU OS) is SunOS with added device drivers.

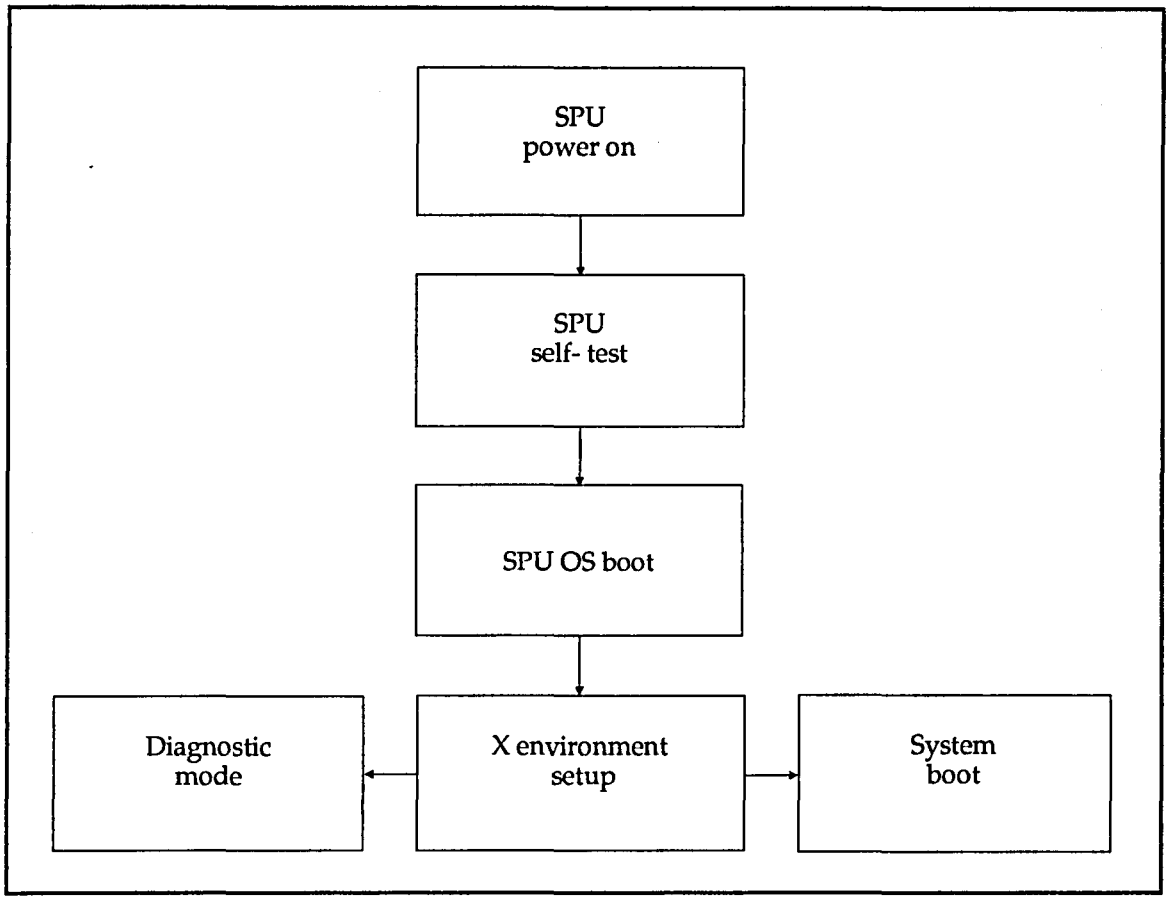
The boot process includes a power-on self-test and invoking of the X-Window environment. The X-Window environment creates several windows on the SPU monitor screen. You can control both the SPU operation and C3800 computer operation from these windows. You can also control both the SPU and the computer by dialing into the SPU from a remote terminal. The SPU directory structure holds all SPU and ConvexOS software. System administration tasks include backups, restores, software installation, system shutdown, and password control.

Refer to the *CONVEX C3800 Series SPU System Manager's Guide*, Order No. DSW-023 for more information on SPU OS.

1.1 Booting

Figure 1-1 displays the steps that occur between the time that the SPU has power applied and the system boot or diagnostics mode is invoked. The following subsections describe each step in this sequence.

Figure 1-1
SPU power-up sequence



1.1.1 SPU power on

Powering up the SPU applies a reset to the workstation's processor. The processor begins executing code from ROM located within the workstation.

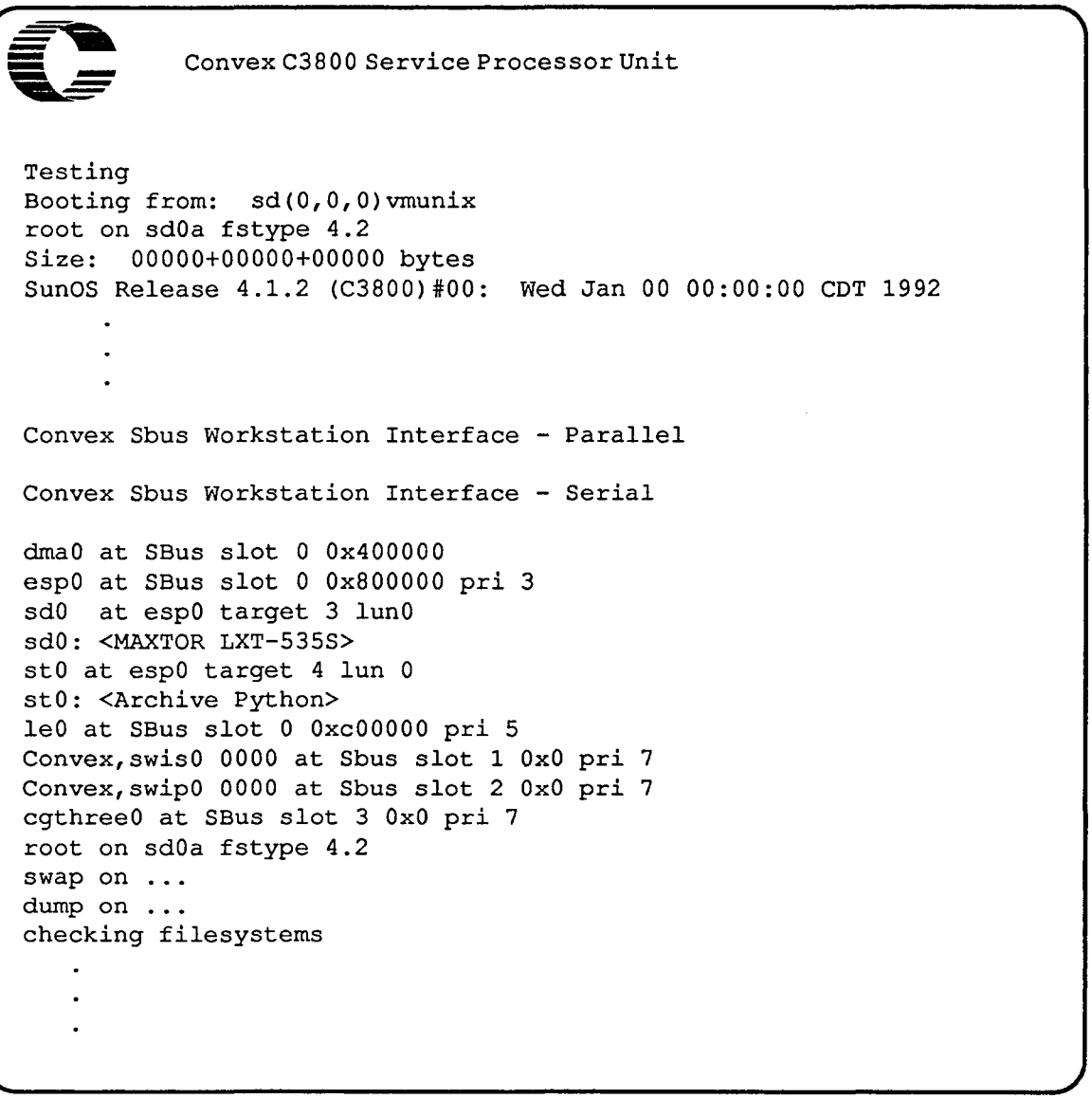
If the SPU does not respond when you operate the power switch, possible problems are:

- Power source
- Power cord
- Power switch
- Power supply fuse (inside power supply)
- Power supply

1.1.2 SPU self-test

The SPU automatically performs the boot PROM start-up sequence when you turn the main power switch on. The sequence includes the self-test sequence contained in the boot PROM. Figure 1-2 shows a typical SPU start-up sequence display of this self-test process. The display may differ between SPUs, depending on SPU configuration and start-up software.

Figure 1-2
Service processor boot display



If a component fails its test, the monitor displays an error message instead of the usual status message. Field replaceable units (FRUs) indicated by failure of these tests include the following:

- Memory boards (SIMMs)
- Motherboard
- Keyboard
- Keyboard fuse
- Ethernet fuse
- SCSI fuse

PROM-based software also identifies peripheral devices on the SPU I/O ports. These devices include:

- Hard disk
- Tape drive
- Floppy disk drive
- CONVEX proprietary interface boards (SWIS, SWIP)

1.1.3 SPU OS boot

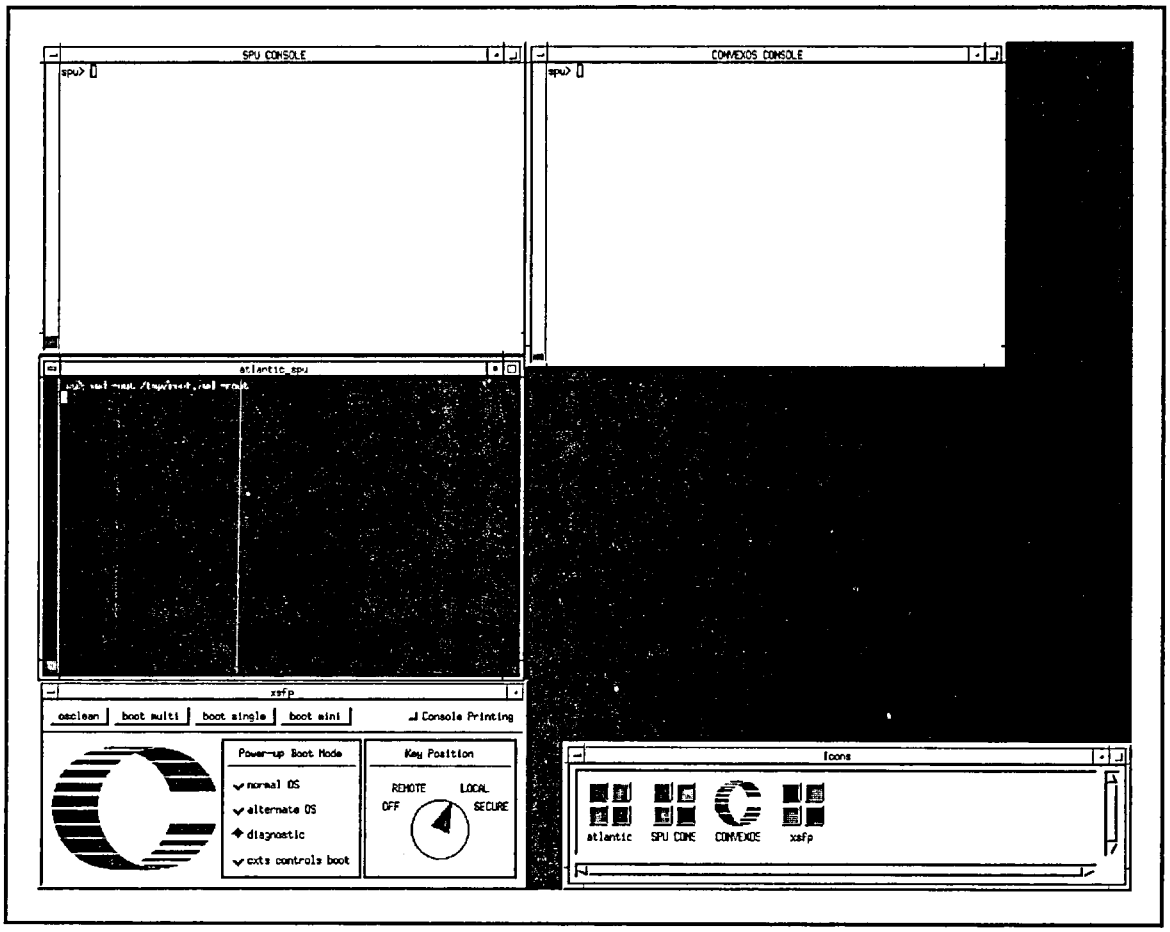
After the workstation diagnostics have been successfully completed, the ROM searches for an operating system. The SPU internal hard disk has SPU OS installed on it. The boot ROM loads SPU OS from the disk and places it in the workstation's memory. Control then passes to SPU OS.

Logic programs controlling the SWIS and SWIP circuit boards download from disk to the field programmable gate arrays on those boards.

1.1.4 X-Window environment setup

At the completion of the SPU boot process, the SPU automatically invokes the X-Window environment. Figure 1-3 shows the resulting SPU monitor screen.

Figure 1-3
Default X-Window display



1.1.5 xsfp start-up

At this point, application software automatically initiates the xsfp utility. The xsfp utility and the Motif Window Manager control the appearance and operation of the SPU window enviroment.

xsfp performs the following operation upon start-up:

1. xsfp reads an initialization file (/diag/db/xsfptab) that describes the daemons that need to be started. xsfp starts and monitors each of these daemons:
   - errlogd—Error logging daemon.
   - cdb_startup—Configuration database start-up file; it starts the following utilities:
     * rbserver—Maintains data structures for resource broker.
     * cdbserver—Maintains configuration database file.
   - bpccommd—Routes messages and commands between user processes and bay power controllers.
   - bpcwatchd—Monitors health of distributed power system.
2. If the system includes the CXTS troubleshooting utility, the xsfp window in Figure 1-3 on page 1-5 also contains a CXTS button. If this button is active, xsfp also starts the following daemons:
   - cxts_rt—Operates all the functions of CXTS.
   - cxts_server—Is the CXTS communication link with the computer and the user.
   - cxts_ui—Generates the user environment.
3. Opens the CONVEXOS CONSOLE xterm window. This window cannot be closed through menu or accelerator keys.
4. When the keyswitch is any position other than OFF, or when the keyswitch moves from OFF to any other position, xsfp checks the configuration database for the boot parameters. If the boot parameters are set to normal OS or alternate OS, it sends a SIGUSR1 signal to the CONVEXOS_CONSOLE program. When this happens, CONVEXOS_CONSOLE executes the file /mnt/os/boot.

1.1.6 Completion of boot

The normal boot sequence would now execute the ConvexOS boot of the C3800 computer. Other options would allow booting of an alternative operating system, or halting of the boot process and awaiting diagnostic commands.

1.2 X-Window appearance and functions

The .mwmrc and .Xdefaults files in `/users/diaguser` control the appearance of the windows that appear on the SPU display. Figure 1-3 on page 1-5 shows the window display.

Note

These setups should not be modified for personal preference by the casual user. Duplicate sets are in /usr/lib/X/app-defaults for use if the copies in `~diaguser` get corrupted.

Up to five areas on the screen can be visible when the X environment is set up. These are described in the following sections. Figure 1-3 shows the location and general appearance of these windows.

1.2.1 SPU CONSOLE

This window receives output destined for /dev/console on the SPU. This window should always be left open to allow workstation created error messages to be visible. SPU OS messages appear in this window. It has a scroll bar with a scroll buffer depth of 500 lines. This window cannot be closed through menu or accelerator keys.

1.2.2 CONVEXOS CONSOLE

This window is where `/mnt/os/boot` is executed. When the system has booted and the ConvexOS console driver is running, this window is the console for ConvexOS. It has a scroll bar with a scroll buffer depth of 1000 lines. This window cannot be closed through menu or accelerator keys.

Note

Never kill the CONVEXOS_CONSOLE program while ConvexOS is booted. A reboot of ConvexOS would be required to re-establish the link between the SPU and the C3800 Series computer. If ConvexOS is allowed to run without the SPU, the system eventually crashes.

1.2.3 SPU

You can use this window for issuing commands on the SPU. It can be iconified if desired since only output from processes started in this window display output to this window. This window has a scroll bar with a scroll buffer of 500 lines.

1.2.4 Icons

This box contains icons for all windows currently available for display.

1.2.5 xsfp

The `xsfp` window allows you to:

- Select and monitor the boot mode
- Control the SPU printer
- Monitor the soft panel keyswitch

Note

Never kill the `xsfp` program while ConvexOS is booted. A reboot of ConvexOS would be required to re-establish the link between the SPU and the C3800 Series computer. If ConvexOS is allowed to run without the SPU, the system eventually crashes.

1.2.5.1 Boot control buttons

The boot control buttons allow you to boot ConvexOS manually. These buttons are only effective if ConvexOS is not booted. Pushing any of these buttons invokes the script `/mnt/os/boot` with the appropriate parameters to perform the indicated boot function:

- `osclean`—Cleans up processes suspended after a halt.
- `boot multi`—Checks if already booted, if not, checks for keyswitch `OFF`. If not `OFF`, boots ConvexOS for multiple users.
- `boot single`—Checks if already booted, if not, checks for keyswitch `OFF`. If not `OFF`, boots ConvexOS for a single user.
- `boot mini`—Checks if already booted, if not, checks for keyswitch `OFF`. If not `OFF`, boots miniroot.

1.2.5.2 Console Printing button

Shows whether the SPU printer is on or off, and allows toggling between these states. When the SPU printer is on, prints all characters written to the CONVEXOS CONSOLE window.

1.2.5.3 Power-up Boot Mode

The Power-up Boot Mode panel contains four buttons with which you can select the boot mode. These buttons are active only when the SPU powers up, or the key moves from the OFF position. Only one button is active at a time.

- Normal OS—Boots ConvexOS.
- Alternate OS—Runs diaginit. You must enter the boot command.
- Diagnostic—Prepares the SPU for diagnostic testing. Does not boot ConvexOS. Does not run diaginit.
- Cxts controls boot—The CXTS utility takes control of the boot process from xsfp.

1.2.5.4 Key Position panel

The Key Position panel displays the current position of the system physical keyswitch. You cannot change the keyswitch position from this panel. The behavior of the windows for each keyswitch position is described as follows:

- OFF—With the key in the OFF position, the computer bay power controllers (BPCs) are held in reset. This mode places no restrictions on keyboard and mouse movements. Remote dial out is possible, but remote dial in as rmtdiag is not possible.
- LOCAL—With the key in the LOCAL position, mouse and keyboard inputs to the windows are not restricted. Modem dial out is possible, but modem dial in as rmtdiag is disabled through software.
- SECURE—With the key in the SECURE position, only the CONVEXOS CONSOLE window on the SPU can accept input. All logins are disabled. Modem dial out is possible, but modem dial in is disabled through software. The ConvexOS console driver is responsible for checking the position of the keyswitch and disabling the ability of CONTROL-p to allow access to the SPU.

- REMOTE—With the key in the REMOTE position, the CONVEXOS CONSOLE driver allows **CONTROL-p** to spawn a shell in the CONVEXOS CONSOLE window. The accessibility of windows on the SPU workstation is user configurable. Two choices are available:
  - REMOTE_RESTRICT—No input is allowed to any window on the SPU workstation. All commands must come over the modem.
  - REMOTE_NORESTRICT—This is the default condition. Input is allowed to occur in any SPU window. This value is set in the .xsfprc file. Refer to the `xsfp` man page.

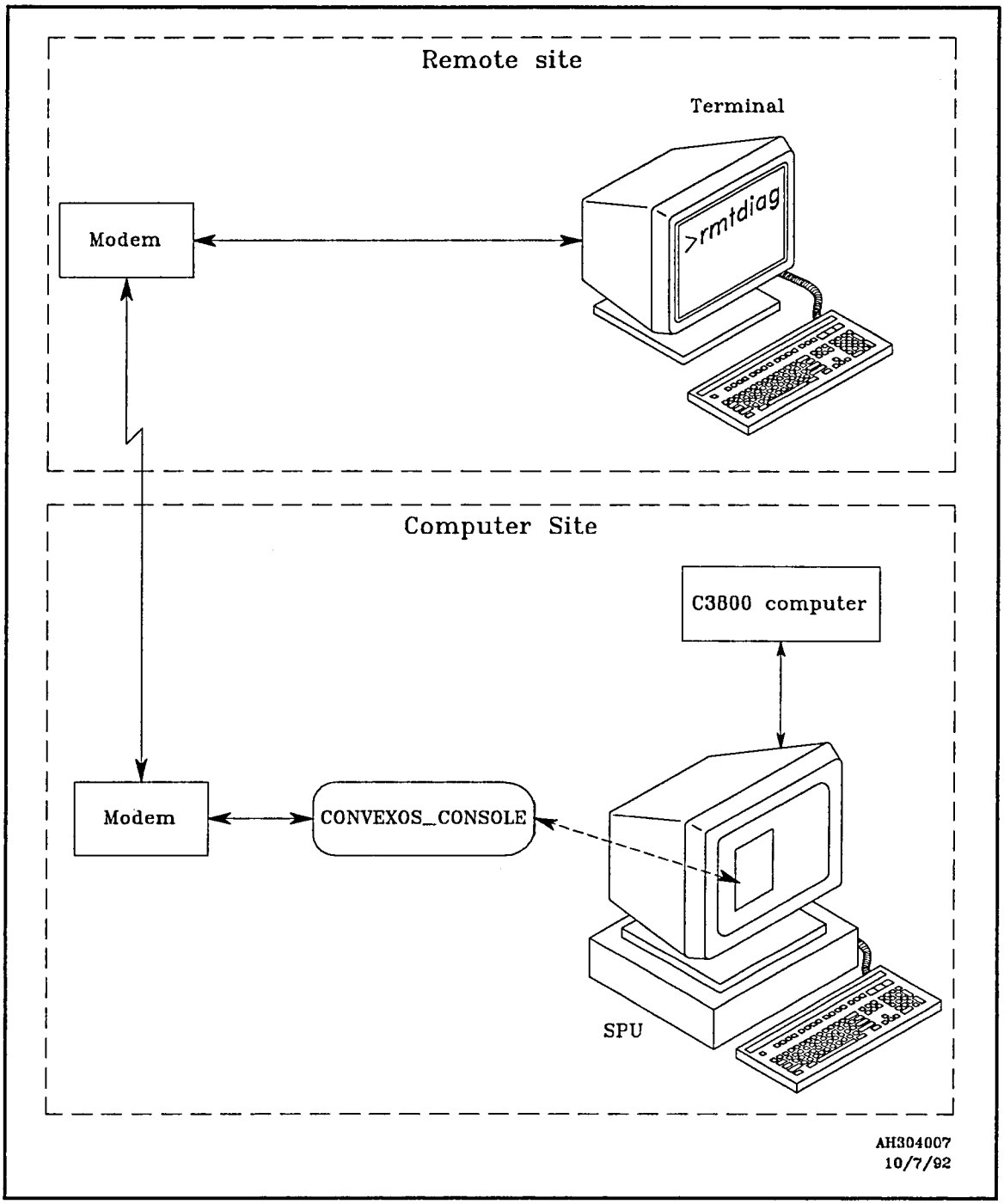
1.2.6 Modem operation

Modem dial outs are permitted with the keyswitch in any position. The modem only accepts dial-in connections as `rmtdiag` when the keyswitch is in the REMOTE position. Other logins are allowed.

When you have dialed into the SPU modem as `rmtdiag`, all actions are echoed to the CONVEXOS CONSOLE window. This occurs whether or not ConvexOS is booted. As the remote user, you may execute SPU commands, boot ConvexOS, and issue **CONTROL-p** to the CONVEXOS CONSOLE program.

When you log in as `rmtdiag`, the `console_s` process starts. This process notifies CONVEXOS CONSOLE that `rmtdiag` has logged in and identifies the login device. CONVEXOS CONSOLE attaches to the login device directly. Figure 1-4 shows the data flow through terminals and processes.

Figure 1-4
Remote dial-in execution



1.3 Remote SPU operation

Two remote SPU logins are available on C3800 Series computers with modems installed:

- You can log onto the SPU using the login name `diaguser`.
- You can take over the CONVEXOS CONSOLE remotely by using the login name `rmtdiag`.

1.3.1 Remote login to a SPU as `diaguser`

Use the following procedure to log remotely into a SPU:

Step 1 From your own system prompt (not shown here), establish the telephone connection to the SPU. A typical command for establishing the connection is:

```
tip -9600 SPU modem phone number
```

Step 2 If the connection is made, the following message appears on your terminal or workststion. Enter at the `login` prompt:

```
CONNECT baud rate
login: diaguser
Password:
```

Step 3 Enter the password. If the login is successful, several messages appear, concluding with the following messages and the system prompt:

```
starting with local interactive setup
done with local interactive setup
spu>
```

Step 4 When you complete your work on the SPU, enter at the `spu` prompt:

```
spu> exit
```

Several messages appear on your screen, including this message:

```
Connection closed.
```

1.3.2 Remote SPU operation as `rmtdiag`

When you log in as `rmtdiag`, you disable the SPU keyboard functions related to the CONVEXOS CONSOLE window on the SPU monitor and take over the CONVEXOS CONSOLE window. The CONVEXOS CONSOLE window on the SPU displays the same information that your remote terminal displays. Effectively, your remote terminal or workstation becomes the CONVEXOS CONSOLE.

You can log in as `rmtdiag` only if the function keyswitch on the C3800 Series computer input/output bay is in the `REMOTE` position.

Use the following procedure to operate the `CONVEXOS CONSOLE` remotely as `rmtdiag`:

Step 1 From your own system prompt (not shown here), establish the telephone connection to the SPU. A typical command for establishing the connection is:

```
tip -9600 SPU modem phone number
```

Step 2 If the connection is made, the following message appears on your terminal or workststion. Enter at the `login` prompt:

```
CONNECT baud rate
login: rmtdiag
Password:
```

Step 3 Enter the password. If the login is successful, several messages appear, concluding with the following message and the system prompt:

```
Attaching to CONVEXOS CONSOLE.
Use the command ''remote_disconnect'' to
exit.
spu>
```

Step 4 When you complete your work on the SPU, enter at the `spu` prompt:

```
spu> remote_disconnect
```

Caution

You must use `remote_disconnect` to end a `rmtdiag` session. Using another command may leave SPU OS in an indeterminate condition.

Several messages appear on your screen, including this message:

```
Attachment to CONVEXOS CONSOLE lost.
Connection closed.
```

1.4 Directory structure

Table 1-1 lists the SPU directories and describes their contents.

The search path is set up in the start-up files for `diaguser` and `rmtdiag`. The default search path is:

- /bin
- /usr/bin
- /diag/bin
- /mnt/os
- /diag/test/spu
- /diag/test
- /diag/local
- /sst
- /diag/hw
- /diag/db
- /usr/bin/X11
- /usr/local/bin
- /etc
- /usr/etc
- /usr/ucb
- /usr/contrib/bin
- /usr/local/etc
- . (current directory)

Table 1-1
SPU directory structure

Directory	Contents
/bin	Commands often used.
/dev	Block and character special device files to communicate with devices.
/etc	Most system administrator commands and configuration files.
/tmp	Holds files temporarily.
/users	Directories for users and the files and subdirectories they create. Diaguser and rmtdiag home directories are here.
/usr	Commands, log files, and so forth.
/usr/adm	System administration data files.
/usr/bin	Commands not required to boot, restore or repair the file system.
/usr/local/man	Online documentation for CONVEX-related man pages.
/usr/spool	Receives spooled files for various programs.
/usr/spool/lp	Control and working files for the lp spooler.
/usr/spool/uucp	Receives queued work files and contains lock files, log files, status files, and other files for UUCP.
/usr/tmp	An alternative directory for placing temporary files, typically very large files.
/usr/man/cat1...cat9	Online documentation that has already been processed to speed up access.
/usr/man/cat1.Z..cat9.Z	Compressed version of cat directories.
/diag/db	Configuration database, RT database, scan ring definitions, CPU microcode, CXTS databases, firmware for power system.
/diag/bin	All diagnostic utilities developed by Processor Test Development.
/diag/test	All test programs developed by Processor Test Development, and all I/O diagnostics.
/diag/test/cti	CTI RC files, information files, parameter files.
/diag/test/spu	SPU-based executables (`spu4000, mcu4000, em4000, cpucti`)
/diag/test/CPU	CPU test C-series object files.

Table 1-1 (continued)
SPU directory structure

Directory	Description
/diag/scripts	Hard error logging scripts (interrogators and extractors)
/diag/data	Event log output. Other output files.
/sst	SST packed pattern sets.
/mnt/os	All ConvexOS supported utilities.
/diag/bin/lib	Miscellaneous IO test related files. For backward compatibility. Due to link at /mnt/bin, this appears as /mnt/bin/lib.
/diag/hw	`dsh` scripts for debug.
/mnt	Top level directory for installation of system software and I/O software. The `boot_db` file must be here.
/mnt/usr/lib	Data files for I/O diagnostics.
/mnt/test	A link to /diag/test for backwards compatibility.
/mnt/bin	A link to /diag/bin for backwards compatibility.

1.5 System administration

The system administration tasks are described in this section.

1.5.1 Backups

The `backup` command writes disk partitions to DAT tape. Follow this procedure to perform a tape backup:

Step 1 Place an empty, retensioned, rewound tape into the SPU tape drive.

Step 2 Enter:

```
login: root
# /etc/backup
```

The SPU disk has 5 partitions. Figure 1-5 is an example of monitor screen output for 2 partitions during a backup.

Figure 1-5
Example of SPU disk dump monitor output

```
spu# /etc/backup
backing up / to /dev/nrst0
  DUMP: Date of this level 0 dump: Thu Apr  2 09:20:11 1992
  DUMP: Date of last level 0 dump: the epoch
  DUMP: Dumping /dev/rsd0a (/) to /dev/nrst0
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 23234 blocks (11.34MB) on 0.01 tape(s).
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: level 0 dump on Thu Apr  2 09:20:11 1992
  DUMP: Tape rewinding  DUMP: 23204 blocks (11.33MB) on 1 volume
  DUMP: DUMP IS DONE
backing up /usr to /dev/nrst0
  DUMP: Date of this level 0 dump: Thu Apr  2 09:21:27 1992
  DUMP: Date of last level 0 dump: the epoch
  DUMP: Dumping /dev/rsd0g (/usr) to /dev/nrst0
  DUMP: mapping (Pass I) [regular files]
  DUMP: mapping (Pass II) [directories]
  DUMP: estimated 490042 blocks (239.28MB) on 0.23 tape(s).
  DUMP: dumping (Pass III) [directories]
  DUMP: dumping (Pass IV) [regular files]
  DUMP: 21.65% done, finished in 0:18
  DUMP: 43.43% done, finished in 0:13
  DUMP: 65.41% done, finished in 0:07
  DUMP: 86.93% done, finished in 0:03
  DUMP: level 0 dump on Thu Apr  2 09:21:27 1992
  DUMP: Tape rewinding
  DUMP: 489954 blocks (239.24MB) on 1 volume
  DUMP: DUMP IS DONE
rewinding tape
backup complete
spu#
```

A full backup usually takes about an hour. It is possible to perform a backup with the SPU in multiple-user mode, but doing so presents a slight chance of catching a file in a transient state. It is safer to bring the SPU down to single-user mode before backing up. The DAT tape is large enough to hold the entire disk. Label the tape and store it in a safe place.

1.5.2 Restores

The SPU OS `restore` utility can restore a whole disk partition or parts of a partition from back-up tape. It also has a limited capability to reconfigure the disk directory structure during a restore operation. See `restore` command on the man pages for a full list of options. Some useful examples follow.

Note

The default directory must be the root directory. See the `restore` man page for more information.

To restore an entire partition from tape (for example, the third partition), enter:

```
login: root
# cd /
# mt rew
# restore xfs /dev/nrst0 3
# mt rew
#
```

The address of the tape drive is /dev/nrst0. The volume number follows the tape drive address in the `restore` command. The volume number is the order in which the disk partitions exist on the back-up tape. The file /etc/fstab contains a list of backed-up partitions in the order they are recorded on the tape. Figure 1-6 shows the contents of a typical /etc/fstab file.

Figure 1-6
Order of partitions on tape

```
/
/usr
/mnt
/diag
/sst
```

To restore a single file from the first partition on tape, enter:

```
login: root
# cd /
# mt rew
# restore xfs /dev/nrst0 1 /path/filename
You have not read any volumes yet.
Unless you know which volume your file(s) are
on you should start with the last volume and
work towards the first.
Specify next volume #: 1
set owner/mode for '.'? [yn] y
# mt rew
#
```

A full restore may take up to an hour. A full restore should not be performed while the system is up. SPU OS can kill processes when their executable is overwritten.

1.5.3 Installing software

All software distributed by CONVEX for installation on the SPU is on DAT tapes. To install the software it is generally only necessary to run the /etc/installsw program. For more information, refer to the release notes for the product being installed.

1.5.4 System shutdown

To shut down SPU OS, use the `/etc/shutdown(1M)` command. To reboot SPU OS, use the `/etc/reboot(1)` command.

1.5.5 Passwords

The system is shipped with a password file that contains the entries shown in Figure 1-7.

Figure 1-7
Password file

```
root::0:1:Operator:/:/bin/sh
nobody:*:65534:65534::/:
daemon:*:1:1::/:
sys:*:2:2::/:/
bin/csh bin:*:3:3::/bin:
uucp:*:4:8::/var/spool/uucppublic:
news:*:6:6::/var/spool/news:/bin/csh
ingres:*:7:7::/usr/ingres:/bin/csh
audit:*:9:9::/etc/security/audit:/bin/csh
sync::1:1::/:/bin/sync
sysdiag:*:0:1:
  Old System Diagnostic:/usr/diag/sysdiag:/usr/diag/sysdiag/sysdiag
sundiag:*:0:1:
  System Diagnostic:/usr/diag/sundiag:/usr/diag/sundiag/sundiag
convexos:*:20:500:convex os suid:/users/convexos:/bin/csh
diaguser::25904:500:
  general diagnostic user id:/users/diaguser:/diag/bin/dsh
rmtdiag::25904:500:
  remote diagnostic user id:/users/diaguser:/diag/bin/console_s
```

Note

None of the accounts have passwords. Therefore, the system administrator must assign passwords to the `rmtdiag`, `diaguser`, and `root` accounts as soon as possible.

For the SPU environment to run correctly, the `rmtdiag` and `diaguser` accounts must exist. Their home directories which exist in /users should not be modified. Passwords may be assigned by the local system administrator as required.

1.5.6 Groups

All CONVEX-supported users are in the /etc/group file. Figure 1-8 shows the entries in /etc/group.

Figure 1-8
Group file

```
wheel:*:0:
nogroup:*:65534:
daemon:*:1:
kmem:*:2:
bin:*:3:
tty:*:4:
operator:*:5:
news:*:6:
uucp:*:8:
audit:*:9:
staff:*:10:
other:*:20:
diag::52:
os::67:
diaguser::500:diaguser,rmtdiag,convexos,root
```

Diagnostic utilities

2

This chapter contains descriptions of the C3800 Series computer diagnostic utilities.

Appendix A contains the man pages for these utilities. In most cases, the man pages are self-explanatory. Some diagnostics have descriptions of their intended use.

Two man pages are available for information only. These are:

- config_data—Defines configuration data base entries.
- cop_contents—Defines the parameters that describe the revision state of the systems in the C3800 Series computer.

Some of the utilities are accessible through menus in service processor windows. This chapter contains descriptions for the use of these windows.

Table 2-1 summarizes the available diagnostic utilities and gives a short description of each.

Table 2-1
Diagnostic utilities summary

Utility	Function
Power	
`altsetpts`	Modifies voltage buses and temperature alarm points.
`bpccommd`	Bay power controllers communication control daemon.
`bpcwatchd`	Bay power controllers communication monitoring daemon.
`margin`	Sets voltages and clocks to margin values.
`powerdown`	Executes an orderly power down sequence.
`powermon`, `xpowermon`	Monitors power parameters, temperature parameters, and fan status; `xpowermon` provides X-Window environment for `powermon`.
`powerup`	Executes an orderly power up sequence.
`pwr_util`	Provides manual control of power settings.
System configuration	
`cdb_browser`, `xcdb_browser`	Displays and modifies contents of configuration database; `xcdb_browser` provides X-Window environment for `cdb_browser`.
`cdb_dump`	Dumps contents of configuration database.
`cdb_get`	Retrieves data from configuration database.
`cdb_update`	Writes data into configuration database by keyword.
`cdbserver`	Maintains and modifies configuration database.
`cdb_startup`	Starts rbserver and cdbserver.
`cop`	Displays circuit board and backplane identification and revision data.
`rbcdb_init`	Generates configuration database.
`rbserver`	Maintains and modifies data structures for resource broker.
`xsys_config`	Reports CPU and memory board status. Can alter some status parameters.

Table 2-1 (continued)
Diagnostic utilities summary

Utility	Function
System initialization and reset	
`cs`	Loads and verifies control stores; initializes CPU RAM.
`diaginit`	Initializes part or all of the C3800 power system.
`initall`	Initializes hardware state, main memory, and clocks.
`mminit`	Initializes main memory after a system power up.
`osclean`	Kills existing ConvexOS processes.
`scan_shm_init`	Allocates shared memory segment for scan ring operations.
`scn_util`	Supports hardware initialization and hardware clock status check.
`sysreset`	Resets the system.
System monitoring	
`bpccommd`	Bay power controllers communication control daemon.
`bpcwatchd`	Bay power controllers communication monitoring daemon.
`dump_soft_log`	Prints memory, scalar processor, and interface adapter soft error information stored in the configuration data base.
`errintd`	Error interrupt daemon and soft error logger.
`errlogd`	Logs and forwards event messages.
`logmsg`	Passes messages to event log.
`mm_sniff`	Detects and corrects single-bit main memory errors.
`powermon`, `xpowermon`	Monitors power parameters, temperature parameters, and fan status; `xpowermon` provides X-Window environment for `powermon`.

2.1 C3800 Series hardware

The hardware hierarchy of a C3800 Series computer consists of bays, ports, board slots, and boards. Figure 2-1 shows bay and port locations within the computer.

A C3800 computer system may have 2 to 5 bays. Bays are numbered consecutively, 0 to 4. Bay 4 and one other bay are always present in a system.

Bays 0, 1, 2, and 3, when present, each may contain up to 2 CPUs and up to 2 memory boards. If configured as I/O expansion units, each of these bays may contain memory boards, I/O interface boards, and I/O channel control units. A bay has 16 board slots, numbered 0x01 through 0x10 in hexadecimal form. The boards number consecutively from the left.

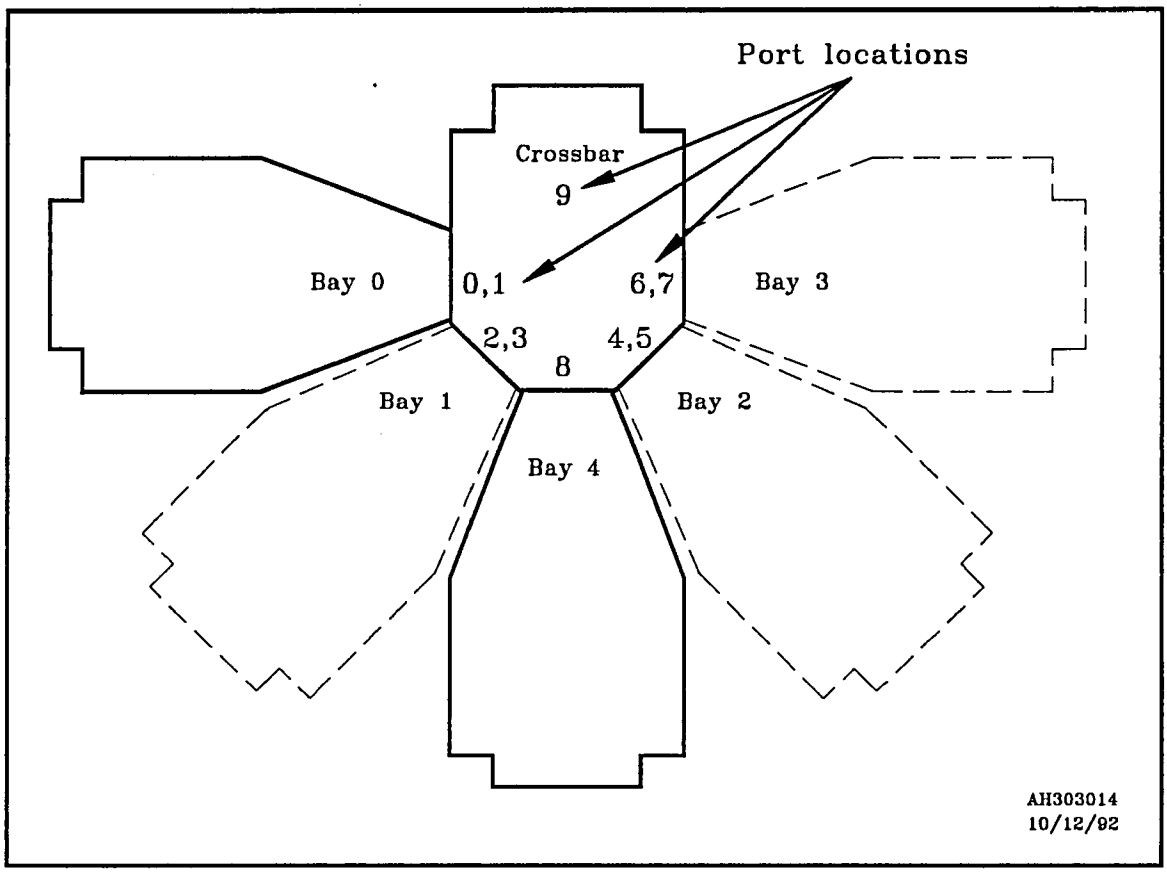
Bay 4 contains the CPU utilities (CU) board, the interface adapter (IA) board, high speed I/O channels, and I/O channel control units. Bay 4 has 13 board slots, numbered consecutively from the left, except that no slots numbered 0x01, 0x0A, and 0x0B exist.

For power monitoring purposes the central cabinet is considered part of bay 4. The central cabinet contains the crossbar boards.

Bays 0 through 3 have two memory ports each, as shown in Figure 2-1. The board slots are numbered hexadecimally. Boards 0x01 through 0x08 of each bay operate through the even numbered port of the bay. Boards 0x09 through 0x10 operate through the odd numbered port.

The I/O cabinet of bay 4 operates through port 8. The power monitoring software regards the central cabinet as port 9.

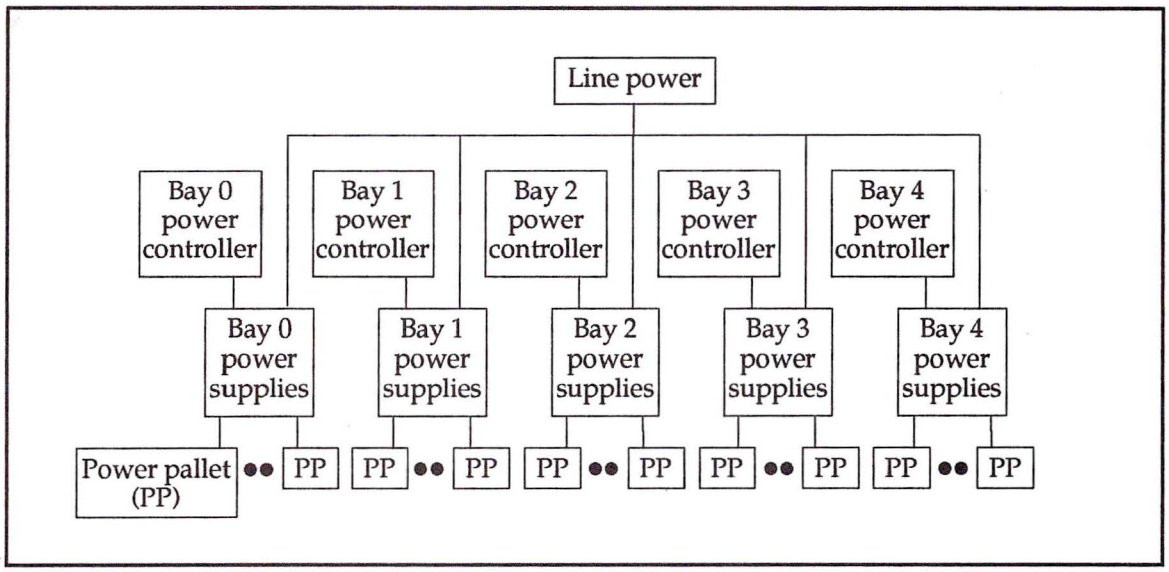
Figure 2-1
Computer bay and port locations



2.2 Power utilities

Each circuit board in a C3800 Series computer, except for channel control units (CCUs) and the crossbar system, has its own power control and distribution system. Figure 2-2 shows the power distribution hierarchy.

Figure 2-2
Power distribution hierarchy



Each bay can have up to 6 bay power supplies (BPSs). The bay power controller (BPC) controls all BPSs in 1 bay. A CPU bay normally has 4 BPCs: 1 for each of 2 memory boards, and 1 for each of 2 CPUs. Each CPU consists of a scalar processor board and a vector processor board; a single BPS supplies both boards. Each board generates its own logic voltages through its own power pallet (PP).

The line power passes through an isolation transformer and an input power harmonizer (not shown in Figure 2-2) to the BPSs. Each BPS converts the line power to 300 Vdc, 12 Vdc, and 5 Vdc. The 12 Vdc and 5 Vdc power the electronics on 1 or more PPs. The PPs convert the 300 Vdc into logic voltages that power the remainder of the board.

The BPSs and PPs operate under program control from the service processor (SPU). The BPCs control the output of the BPSs. Power pallet controllers (PPCs) control the output of the PPs. Each PP is under control of its own PPC.

2.2.1 altsetpts

The altsetpts diagnostic utility provides the capability of setting or altering the following parameters on individual buses, boards, and bays:

- System voltages
- Temperature alarm points (warm set points)
- High temperature system shutdown points (hot set points)

Any or all of the bus voltages, except for housekeeping voltages, and temperature set points can be altered.

The crossbar boards share a power pallet. Set point modifications affect all crossbar boards. Any invalid option in the command line aborts the entire command.

You can operate the altsetpts utility in either of two ways:

- Alter a specific voltage or temperature set point by directly entering a complete command.
- Display the set points in list form and alter one or more of the displayed values.

2.2.1.1 Direct command entry

Enter an altsetpts command at the service processor system prompt. Include all the switches necessary to set the desired voltage or temperature point. The altsetpts(1D) man page in Appendix A provides several examples of altsetpts commands.

2.2.1.2 Altering a list of set points

To set temperature limits in list form, enter:

altsetpts -t <*boardname*>

Figure 2-3 is an example of an `altsetpts` screen display for setting temperature limits.

Figure 2-3
`altsetpts` temperature setting display

```
mb7: warm nominal=+48.671   actual=+48.671.   New value ==>
mb7: hot nominal=+63.377   actual=+63.377.   New value ==>
```

The cursor appears after `New value ==>` on the first line of the list. Either enter a new value for the set point, or press

ENTER

without any entry. The latter action leaves the setpoint unchanged and moves the cursor to the end of the next line.

To set system voltages in list form, enter:

altsetpts *<boardname(s)>*

Figure 2-4 is the altsetpts screen display for setting voltage parameters.

Figure 2-4
`altsetpts` voltage setting screen display

```
margin: mb0 setpoints: bay=3 slot=6
        PPC actual(nominal) temp setpoints(C): warm=48.67(48.67)
hot=63.38(63.38)
        ppcm5v bus voltages:    nominal=-5.000 actual=+0.000
                killpoints:       low=-5.250    high=-4.700
        ppcvcc bus voltages:    nominal=+5.000 actual=+0.000
                killpoints:       low=+4.700    high=+5.250
        gnd bus voltages:    nominal=+0.000 actual=+0.000
                killpoints:       low=+0.000    high=+0.000
        gnd bus voltages:    nominal=+0.000 actual=+0.000
                killpoints:       low=+0.000    high=+0.000
        vttga bus voltages:    nominal=-2.000 actual=-2.000
                killpoints:       low=-2.141    high=-1.860
        vtt bus voltages:    nominal=-2.000 actual=-2.000
                killpoints:       low=-2.141    high=-1.860
        vcc bus voltages:    nominal=+5.000 actual=+5.000
                killpoints:       low=+4.561    high=+5.349
        vee bus voltages:    nominal=-4.500 actual=-4.500
                killpoints:       low=-4.814    high=-4.185
mb7: vttga nominal=-2.000  actual=-2.000.  New value ==>
mb7: vtt nominal=-2.000  actual=-2.000.  New value ==>
mb7: vcc nominal=+5.000  actual=+5.000.  New value ==>
mb7: vee nominal=-4.500  actual=-4.500.  New value ==>
```

The cursor appears after the first `New value ==>` on the list. Either enter a new value for the set point, or press

ENTER

without any entry. The latter action leaves the setpoint unchanged and moves the cursor to the end of the following line.

2.2.2 bpccommd

The BPC daemon, `bpccommd`, controls the routing of messages and commands from user processes to the requested bay power controllers and then routing the BPC's response back to the user process.

Upon receipt of a message from a user process, `bpccommd` reads the message from the queue and determines the destination of the message. If the message is intended for itself, `bpccommd` performs the requested action and responds to the requesting process. If the message is intended for one of the BPCs, `bpccommd` routes the message to that BPC and waits for a response. If the BPC does not respond within a preset time, `bpccommd` responds to the user process with an error message, otherwise it routes the BPC's response back to the user process.

Requests to the BPC daemon are always acknowledged. The BPC firmware can route an unsolicited message to the `bpccommd`. When one of these is received, `bpccommd` routes the message to another daemon process, `bpcwatchd(1d)` for interpretation and resolution.

2.2.3 bpcwatchd

The BPC daemon, `bpcwatchd`, monitors the overall health of the distributed power system. It sleeps until hearing from `bpccommd(1d)` that one or more of the BPCs in the system have been released from reset, establishing communications between the SPU and the BPC. At this point, `bpcwatchd` begins monitoring all message traffic forwarded from `bpccommd(1d)` for each active BPC.

Once communication has been established, `bpcwatchd` monitors the frequency of the messages it receives from `bpccommd`. If it has not heard from a particular BPC in a certain length of time, it declares that particular BPC dead and places it in reset, sending a message to `errlogd` in the process. In addition, `bpcwatchd` interprets each message it receives, performing any actions on the power system that it determines to be appropriate in response to the received message. In all cases, the offending message is passed to `errlogd`.

2.2.4 margin

`margin` formats the commands necessary to request nominal or marginal voltage or clock values set by the controllers. It sets voltages or clock frequencies to nominal, ±3% or ±5% for specified boards, bays, or the crossbar system.

2.2.5 powerdown

`powerdown` formats the commands necessary to instruct the appropriate BPCs to terminate power to the specified boards. This enables you to remove power from any board, any bay, logic buses only, or the whole crossbar system, without affecting power delivery to the remainder of the system. When power is removed from a board, the BPC still has power.

Warning

Issuing this command does not disable all power into a bay. The bay power controllers remain powered until their circuit breakers are thrown.

2.2.5.1 powermon, xpowermon

The powermon utility enables you to monitor and log power and temperature values throughout the computer. The powermon utility is also available through an X-Window utility via the xpowermon command. powermon formats the commands necessary to request local environment status from the appropriate BPC or PPC. xpowermon functions similarly, but it uses the X-Window system to display the data.

Note

Errors occur when monitoring a board with its bay power controller shut down.

2.2.5.2 xpowermon user interface

Power and temperature values are visible through the following window displays on the service processor:

- Bay window
- Board window
- Help window

Table 2-2 lists the xpowermon window mnemonics identifying parts of the computer.

Table 2-2
xpowermon mnemonics

Mnemonic	Definition
bay0–bay4	Bay number
xbar	Crossbar cabinet
ccu0–ccu39	I/O channel control unit
mb0–mb7	Memory board
sp0–sp7	Scalar processor
ia0–ia8	Interface adapter
vp0–vp7	Vector processor
xiop0–xiop8	Input/output processor
cu	Utilities board
kcu	Workaround utilities board

2.2.5.3 Getting started

At the service processor system prompt, enter:

```
xpowermon
```

The xpowermon window then appears on the screen. Figure 2-5 shows the xpowermon window, displaying bay information.

Many of the window facilities operate with the service processor mouse. Chapter 4 describes how to use the mouse.

2.2.5.4 xpowermon bay window

To get the xpowermon bay window display, use the mouse to select the target window and enter:

```
bay<number>
```

where <*number*> is the bay number 0 through 4.

The xpowermon software regards the central cabinet as board 0x01 of bay 4.

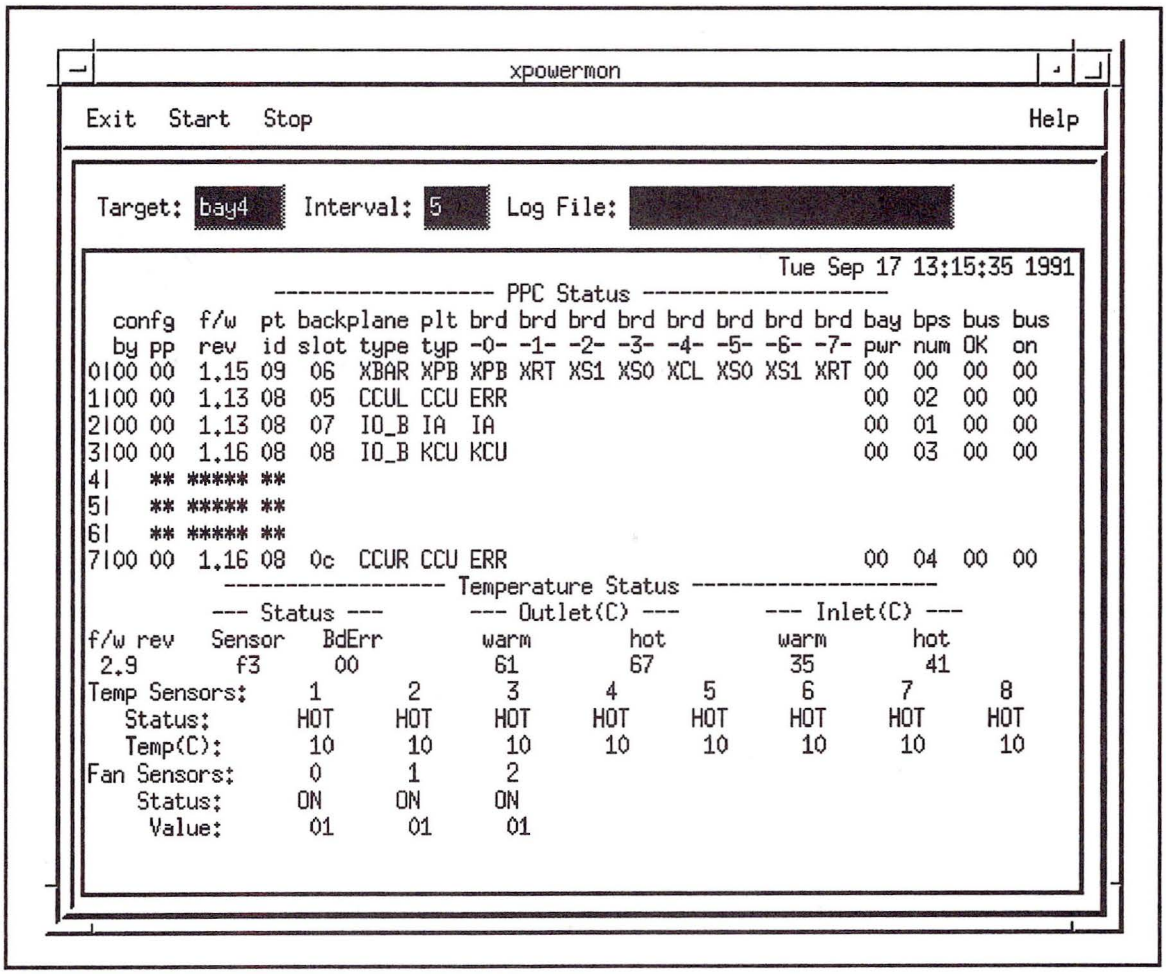
Figure 2-5 shows the xpowermon bay window.

The xpowermon bay window consists of a menu bar at the top of the window, a row of data entry boxes below the menu bar, and a window pane containing power pallet controller (PPC) status and temperature status information.

The menu bar has the following buttons:

- Exit—Exit the xpowermon program and delete the window from the screen
- Start—Begin monitoring
- Stop—Stop monitoring

Figure 2-5
xpowermon bay window



The data entry bar has the following input boxes:

- `Target:`—Enter a bay mnemonic to get the bay window display
- `Interval:`—Enter the interval between voltage and temperature readings; default is 15 seconds
- `Log File:`—Direct the logging results to a specified file

The display window pane PPC status display has the following fields:

- `confg by`—Bay configuration error code (`00`, OK; `01-ff`, errors)

- `confg pp`—Power pallet configuration error code (`00`, OK; `01-ff`, errors)
- `f/w rev`—Power pallet controller firmware revision
- `pt id`—Port identification number
- `backplane slot`—Backplane slot number
- `backplane type`—Wiring configuration of backplane slot connector
- `plt typ`—Pallet type
- `brd0–brd7`—Board type; only board 0 specified for all slots except bay 4, slot 6 (crossbar pallet); crossbar pallet serves several different types of boards
- `bay pwr`—Bay power status code (`00`, OK; `01-ff`, errors)
- `bps NUM`—Identifies BPS connected to power pallet
- `bus OK`—Bus status (`00`, OK; `01-ff`, errors)
- `bus on`—Bus on or off (`00`, OK; `01`, off by command; `02-ff`, errors)

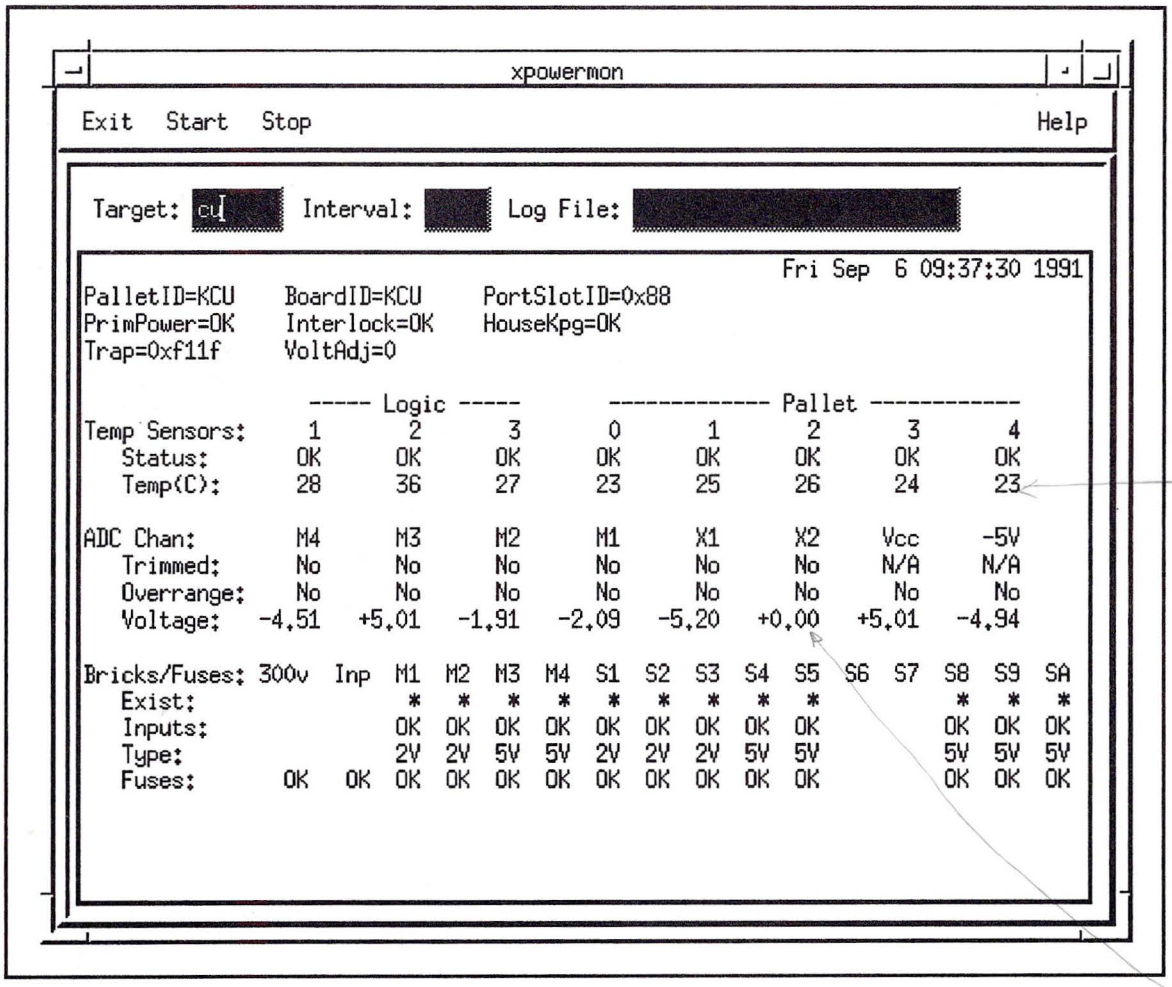
The display window pane temperature status display has the following fields:

- `f/w rev`—Bay power controller firmware revision
- `Sensor`—Status of temperature sensors (`00`, all temperatures OK; `f1`, at least one sensor hot; `f1`, at least one sensor warm)
- `BdErr`—BPC board error (`00`, OK; `01-ff`, errors)
- `Outlet (C) warm`—Outlet warning alarm temperature (Celsius)
- `Outlet (C) hot`—Outlet shutdown temperature (Celsius)
- `Inlet (C) warm`—Inlet warning alarm temperature (Celsius)
- `Inlet (C) hot`—Inlet shutdown temperature (Celsius)
- `Temp Sensors :`—ID numbers of temperature sensors
- `Status:`—`WARM, HOT, OK`
- `Temp (C) :`—Actual temperatures (Celsius)
- `Fan Sensors Status`—`ON, OFF`
- `Fan Sensors Value`—RPM/1000

2.2.5.5 xpowermon board window

Entering a board mnemonic causes the xpowermon window to display voltage and temperature parameters of the specified circuit board. Figure 2-6 shows the xpowermon board window.

Figure 2-6
xpowermon board window



The xpowermon board window consists of a top menu bar, a row of data entry boxes below the menu bar, and a window pane containing power pallet controller (PPC) status and temperature status information.

Field definitions for the menu bar and data entry boxes are identical to those of the xpowermon bay window.

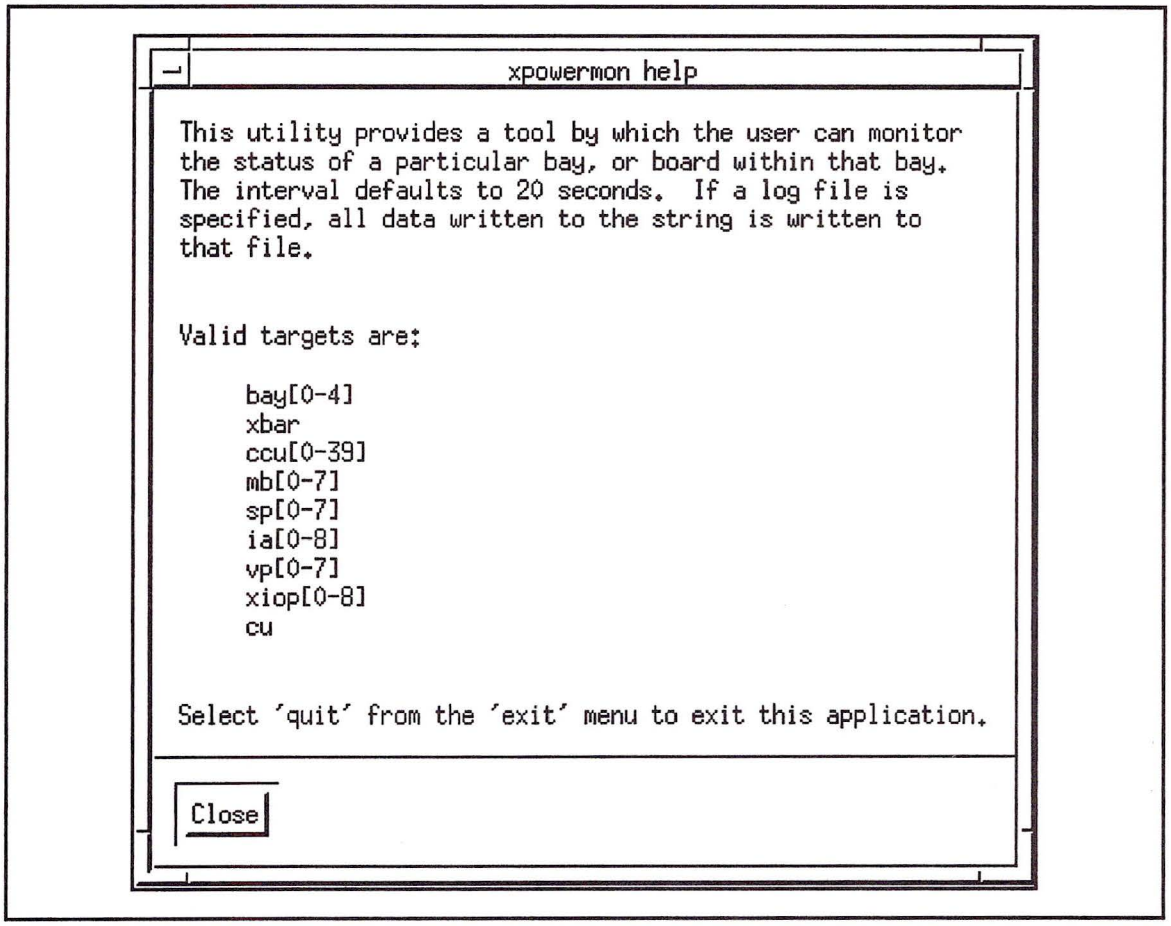
The display window pane has the following fields:

- `PalletID`—Power pallet controller identification
- `BoardID`—Target board mnemonic
- `PortSlotID`—Slot identification in hexadecimal numbering
- `PrimPower`—Primary power (`OK, ERR`)
- `Interlock`—Board to backplane connector status (`OK, ERR`)
- `HouseKpg`—Voltages to power pallet controller (`OK, ERR`)
- `Trap`—Last error code received; normal during diagnostic testing
- `VoltAdj`—Not implemented
- `Logic`—Sensors located on logic board
- `Pallet`—Sensors located on power pallet
- `Temp Sensors:`—ID numbers of temperature sensors
- `Status:`—`WARM, HOT, OK`
- `Temp (C):`—Actual temperatures (Celsius)
- `ADC Chan:`—Voltage buses on board
- `Trimmed:`—Voltage regulator corrections larger than set amount
- `Overrange:`—Voltages out of accepted range; could cause shutdown
- `Voltage:`—Bus voltage
- `Bricks/Fuses:`—Power supply and fuse designations
- `Exist:`—Displays whether the machine has the power brick or fuse installed (*, brick or fuse exists; blank, brick or fuse does not exist)
- `Inputs:`—Condition of input voltage (`OK, ERR`)
- `Type:`—Nominal brick output voltage
- `Fuses:`—`OK, ERR`

2.2.5.6 xpowermon help window

Clicking on Help in the xpowermon window causes the xpowermon help window to appear. Figure 2-7 shows the xpowermon help window.

Figure 2-7
xpowermon help window



2.2.6 powerup

`powerup` formats the commands necessary to instruct the appropriate BPCs to apply power to the specified boards.

2.2.7 pwr_util

`pwr_util` is a menu-driven utility that allows you to directly access many of the system parameters and pieces of information that are automatically initialized by `diaginit`. `pwr_util` allows manual manipulation of the bay power controllers (BPCs), the power pallet controllers (PPCs), the serial communications channels from SPU to computer, and set modify or display information on a board's firmware.

Caution

The `pwr_util` utility provides access to the C3800 power system at its lowest level. Use by untrained personnel can result in serious system damage.

`pwr_util` can change the voltage and temperature set points for a board or bay, although `altsetpts` or `margin` are recommended for these tasks.

`pwr_util` is a menu driven utility when no options are specified. When you specify the `-s` option, the same set of inputs is expected, but no messages or menu options are printed. This is primarily for a script when `pwr_util` is not being used interactively. No user prompts are necessary.

2.3 System configuration

The following utilities provide means to discover and alter the C3800 Series system configuration.

2.3.1 cdb_browser, xcdb_browser

cdb_browser is an interactive utility for displaying the configuration data base (CDB). You can display either the current data for a given keyword or attributes for the keyword. You may also modify the data for a given keyword.

Five options are available when starting cdb_browser. Refer to the cdb_browser man pages for details of these options. The default radix for displaying and/or modifying the data values is hexadecimal. Figure 2-8 is an example of this main menu.

Figure 2-8
xcdb_browser menu

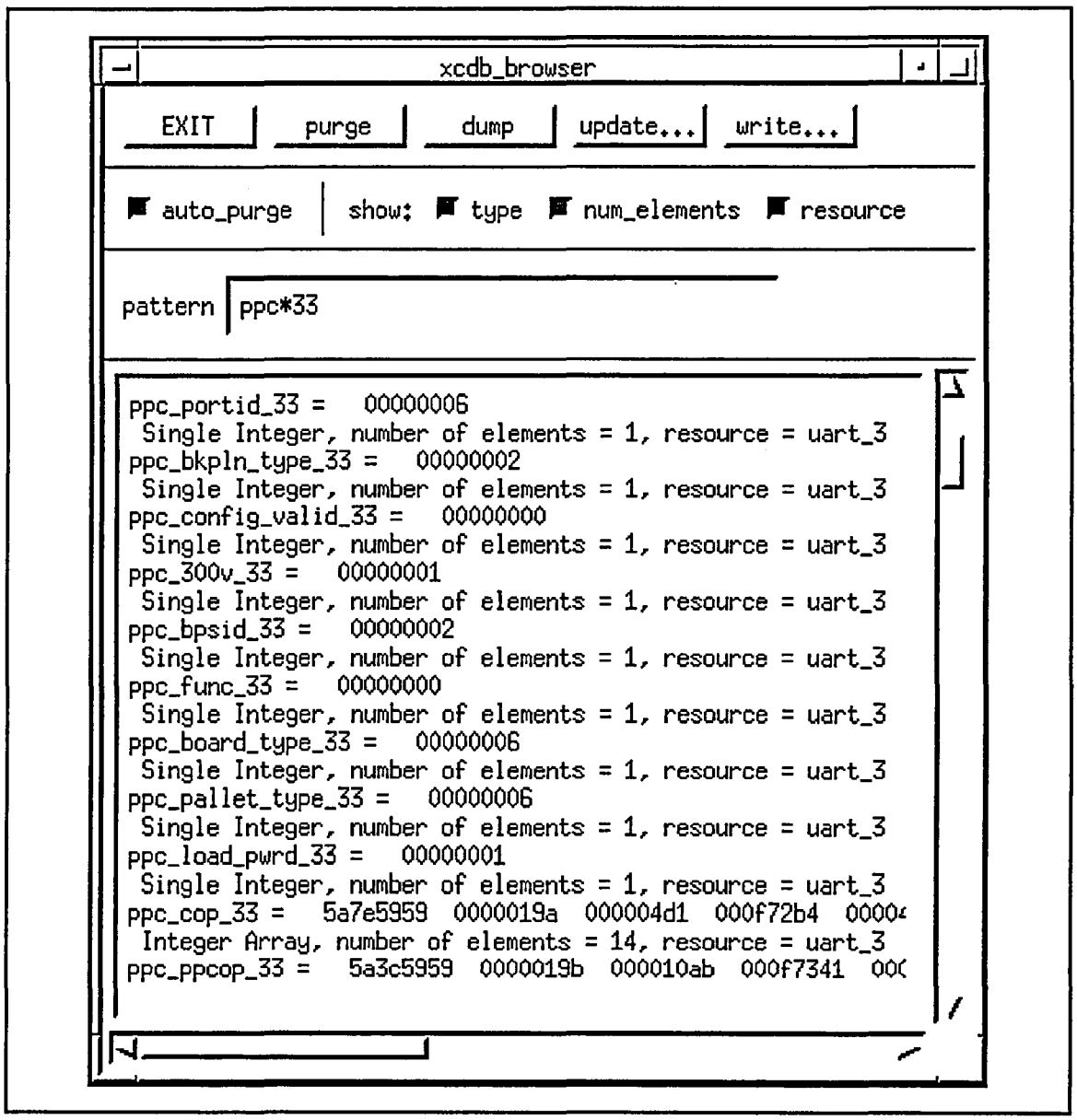
```
1. Mode (verbosity level)
2. Dump database
3. Item query
4. Update
5. Quit
```

xcdb_browser is a user-friendly X-Window version of cdb_browser. You can browse and alter the CDB by using the following X-Windows:

- The xcdb_browser main window displays selected parameters from the CDB.
- The xcdb_browser update window provides the means to alter any CDB parameter.
- The xcdb_browser write_text window enables you to write the contents of the xcdb_browser main window to a file.

Figure 2-9 shows the `xcdb_browser` main window.

Figure 2-9
`xcdb_browser` main window



The structures in the xcdb_browser main window have the following functions:

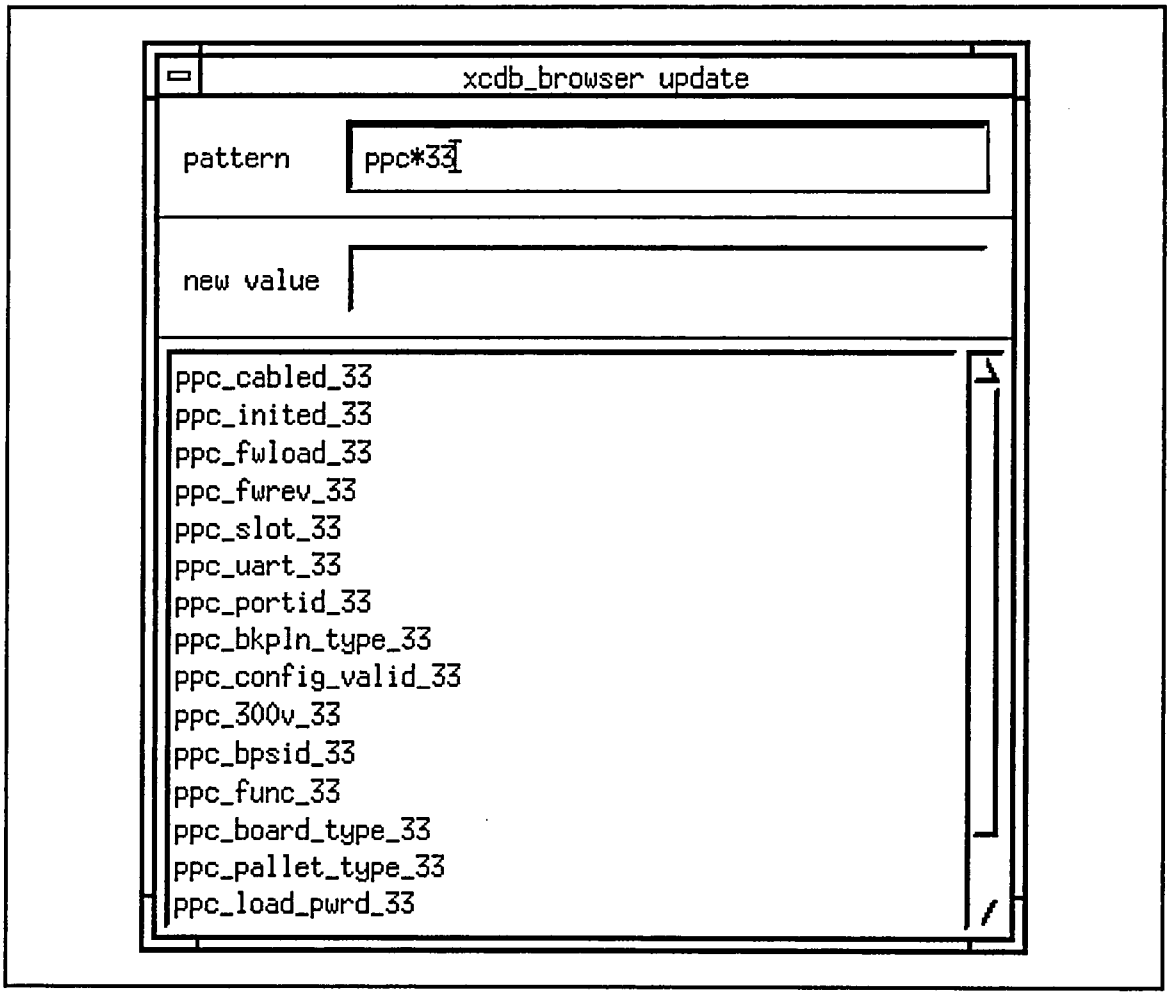
- Select the EXIT button to exit the xcdb_browser utility.
- Select the purge button to clear xcdb_browser of data.
- Select the dump button to read the entire contents of the CDB into xcdb_browser.
- Select the update... button to invoke the xcdb_browser update window.
- Select the write... button to invoke the xcdb_browser write window.
- Select the auto_purge button to cause xcdb_browser to automatically clear all data before reading new CDB data.
- The show: buttons control the display of parameter information:
  - Select the type button to cause xcdb_browser to display the parameter data type.
  - Select the num_elements button to cause xcdb_browser to display the number of array elements in the parameter.
  - Select the resource button to cause xcdb_browser to display the resource associated with the parameter.
- Select the pattern pane to selectively define the parameters to be displayed. Wild card characters are allowed. The pattern shown in the pattern pane in Figure 2-9 produced the display shown in the large pane.
- The large pane in Figure 2-9 displays the current contents of xcdb_browser. Use the scroll bars to the right and below the pane to browse the contents of xcdb_browser.

Figure 2-10 shows the xcdb_browser update window.

Caution

Improper editing of the CDB can cause a ConvexOS crash or incorrect system operation.

Figure 2-10
`xcdb_browser update` window



Perform the following steps to update the CDB:

Step 1 Display the desired parameters by entering a parameter pattern in the `pattern` window.

Step 2 Remove from the list any parameters that you do not want to change by selecting each parameter and pressing RETURN.

Step 3 Select the `new value` pane. A cursor appears in the pane.

Step 4 Enter the new value. All parameters in the window assume the new value.

Figure 2-11 shows the xcdb_browser write_text window. Use this window to write the contents of xcdb_browser to a selected file. The structures in the window have the following functions:

- The If file exists: buttons determine the type of file storage.
- The Filter pane shows the current directory path and types of files displayed.
- The Directories pane shows the parent directory link and the available child directory paths from the current directory. It may not show the full width of the directory display. You may widen the window or use the pane scroll bar to display more of the directory paths.
- The Files pane shows the files available in the current directory that match the specified filter.
- The Selection pane shows the currently selected file with its path.
- The OK button selects the file indicated in the Files pane as the file to browse.
- The Filter button selects the filter typed into the Filter pane.
- The Cancel button returns the window to its original state before you altered any of the panes.

Use the following method to write to a file:

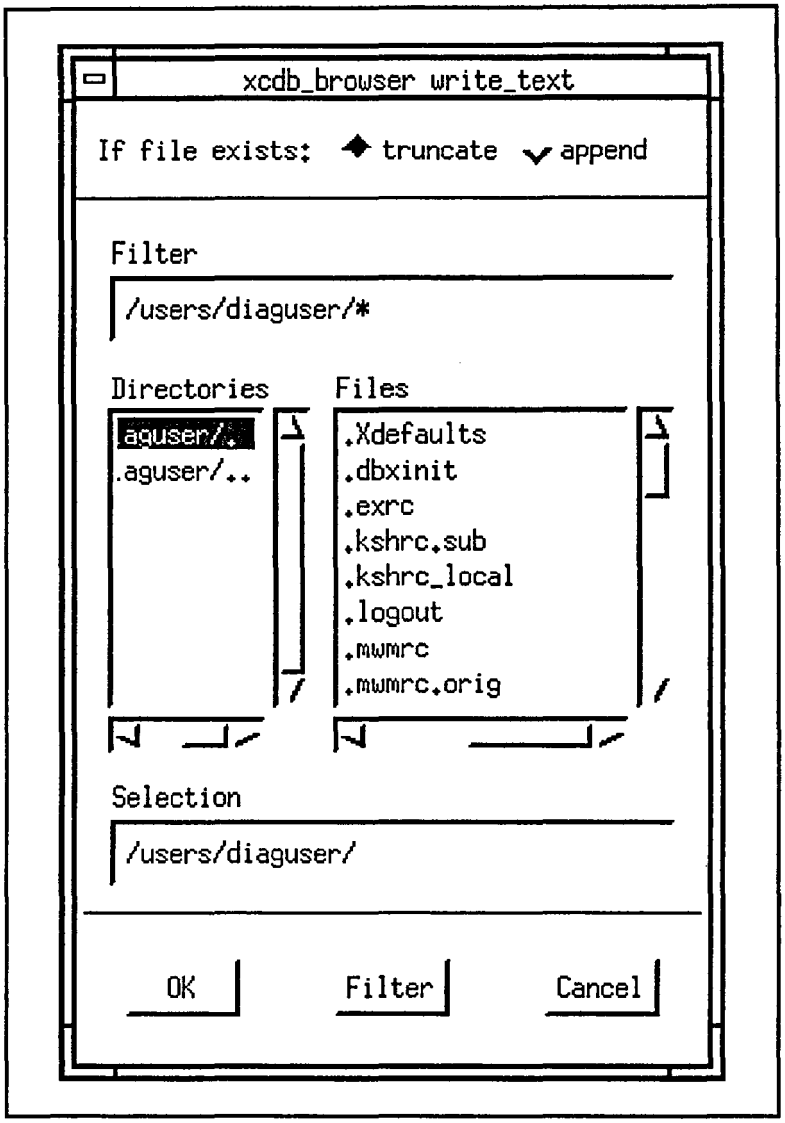
Step 1 Select truncate to purge the existing data in the file when writing new data to the file. Select append to write the new data to the end of the file.

Step 2 Move to the desired directory. Repeat either of the following two actions until the appropriate path is displayed.

- To move to a child directory of the current directory, click on a child directory name in the Directories pane.
- To move to the parent directory of the current directory, click on the parent directory link (the . . directory) in the Directories pane.

Step 3 The selected directory is highlighted in the Directories pane. The selected path appears in the Filter pane. If the move is to the parent directory, the Selection pane clears. If the move is to a child directory, the Selection pane displays the selected directory path.

Figure 2-11
`xcdb_browser write_text` window



Step 4 Click on the `Filter` button at the bottom of the window.
A highlight appears around the `Filter` button. The selected directory becomes the current directory. The parent directory link and the child directories of this new current directory appear in the `Directories` pane. A list of files in the current directory appears in the `Files` pane.

Step 5 Click on the desired file. The file becomes highlighted.

Step 6 To write the file, click on the `OK` button.
To abort the operation, click on the `Cancel` button.

2.3.2 cdb_dump

`cdb_dump` command dumps the contents of the configuration data base. Every entry is printed, along with its corresponding value(s).

2.3.3 cdb_get

`cdb_get` command prints out the data, in the appropriate type, for each of the given keywords. It retrieves only one entry at a time. It does not accept wild cards or do any pattern matching on the keywords. If pattern matching is desired, use option 3 of `cdb_browser` or use `cdb_dump | grep` *regexpr*. The default output format is `key` = *key_word_value(s)*. If the `-q` option is specified, the output is just the value of the cdb key without the preceding `key` =.

2.3.4 cdb_update

`cdb_update` command writes the specified data into the given keyword, *keyw*, in the configuration data base. The data must be appropriate for the type of keyword, for example, character, integers, floats, doubles, strings, or arrays.

2.3.5 cdbserver

The `cdbserver` daemon maintains the configuration data base file and modifies this file per requests from SPU processes. When invoked, an attempt is made to restore the data base from the current files (cdb.db and cdb.map). If these do not exist then the configuration data base is empty, that is, no entries are present.

When first started, `cdbserver` checks whether it has previously been started. If so, it generates a fatal error and the utility exits. If not, normal processing continues.

This utility maintains a local data structure (a queue) in memory. Each node contains all information concerning the configuration data base entry. A hash table is also maintained that provides quick random access to these nodes.

SPU-based processes communicate with this utility via a message protocol. This message contains the following information: process ID, function, message data, and returned status. The procedure is:

1. Process informs `cdbserver` of request.
2. `cdbserver` checks the queue for validity of request.
3. `cdbserver` sends back either an okay to continue or error response.
4. If okay to continue then the process updates the configuration data base file.
5. Process sends information back to `cdbserver` so it can be added to the queue.
6. `cdbserver` loops continuously, waiting for messages unless killed. This is done by the `killservers` utility.

2.3.6 cdb_startup

The `cdb_startup` daemon monitors the execution of the `rbserver` and `cdbserver` utilities. If either `rbserver` or `cdbserver` terminate for some reason, `cdb_startup` automatically restarts them.

`cdb_startup` should not be started manually. It is under control of `xsfp`.

The `cdb_startup` utility uses a set of configuration data base files to start both servers. These files reside in the /diag/db directory. The `cdb_startup` utility uses the following procedure to determine which set of files to use:

1. The utility uses the *checkpoint* files (cdb.db.chkpt, cdb.map.chkpt, and rb_config_file.chkpt) if they all exist.
2. If the checkpoint files do not all exist and the *normal* files (cdb.db, cdb.map, and rb_config_file) all exist, the utility uses the normal files.
3. If all checkpoint files and all normal files do not exist and all *standard* files (standard.db, standard.map, and standard.rb) exist, the utility uses the standard files.
4. If neither the checkpoint files, the normal files, nor the standard files exist in complete form, the utility calls `rbcdb_init() -i`.

If the servers are started using an existing set of data base files then a check is made if either of the servers are currently executing. If they are then they are halted and restarted. First `rbserver` is started, and then `cdbserver`.

2.3.7 config_data

`config_data` is an informational man page identifying configuration data base entries.

2.3.8 cop

The `cop` utility displays the contents of the board or backplane cop chips in a predefined format. The options you provide when you invoke `cop` determine how much of the cop information is displayed.

2.3.9 cop_contents

The `cop_contents` man page explains the contents of the cop chips to inform you of the information extracted from the cop chips. The cop chips contain:

- Part number
- Serial number
- Ring revision
- Wiring revision
- Assembly revision

The contents of the cop chip may vary, depending on whether the board or backplane is in preproduction (in initial debug) or in production (released to manufacturing).

2.3.10 rbcdb_init

The `rbcdb_init` utility tests if the `rbserver` is running. If so, it is halted and a new `rbserver` started. If the `cdbserver` is running then it also is halted and a new `cdbserver` started.

After both servers are started, calls are made to `rb_create()` to initialize the resources in the resource broker. After the resources are generated, calls are made to `cdb_create()` to create entries in the configuration data base and initialize their data values.

Use the `-i` option to copy the generated `cdbserver` files and `rbserver` files into the *standard* files.

2.3.11 rbserver

The rbserver daemon maintains the data structures for the resource broker and modifies these structures per requests from other processes. Processes communicate with this utility via a message protocol. This message contains the following information: process ID, function, message data, and returned status.

When first started, this utility checks whether the daemon has previously started. If so, it generates a fatal error and the utility exits. If not, it initializes the data structures and waits for a message.

rbserver loops continuously waiting for messages unless killed.

2.3.12 xsys_config

The xsys_config utility reports CPU and memory board status as contained in the configuration data base. It provides a way to change some status parameters.

The upper segment of the xsys_config window gives status information. Figure 2-12 shows the upper segment of the xsys_config window. The two buttons at the top of the window have the following functions:

- Select the EXIT button to exit xsys_config.
- Select the update button to cause the xsys_config window to reflect the current status of the computer.
- Selecting the bank_interleave... button displays the xsys_config Bank_Interleave window. Refer to Figure 2-14 on page 2-34 for an explanation of bank interleaving.

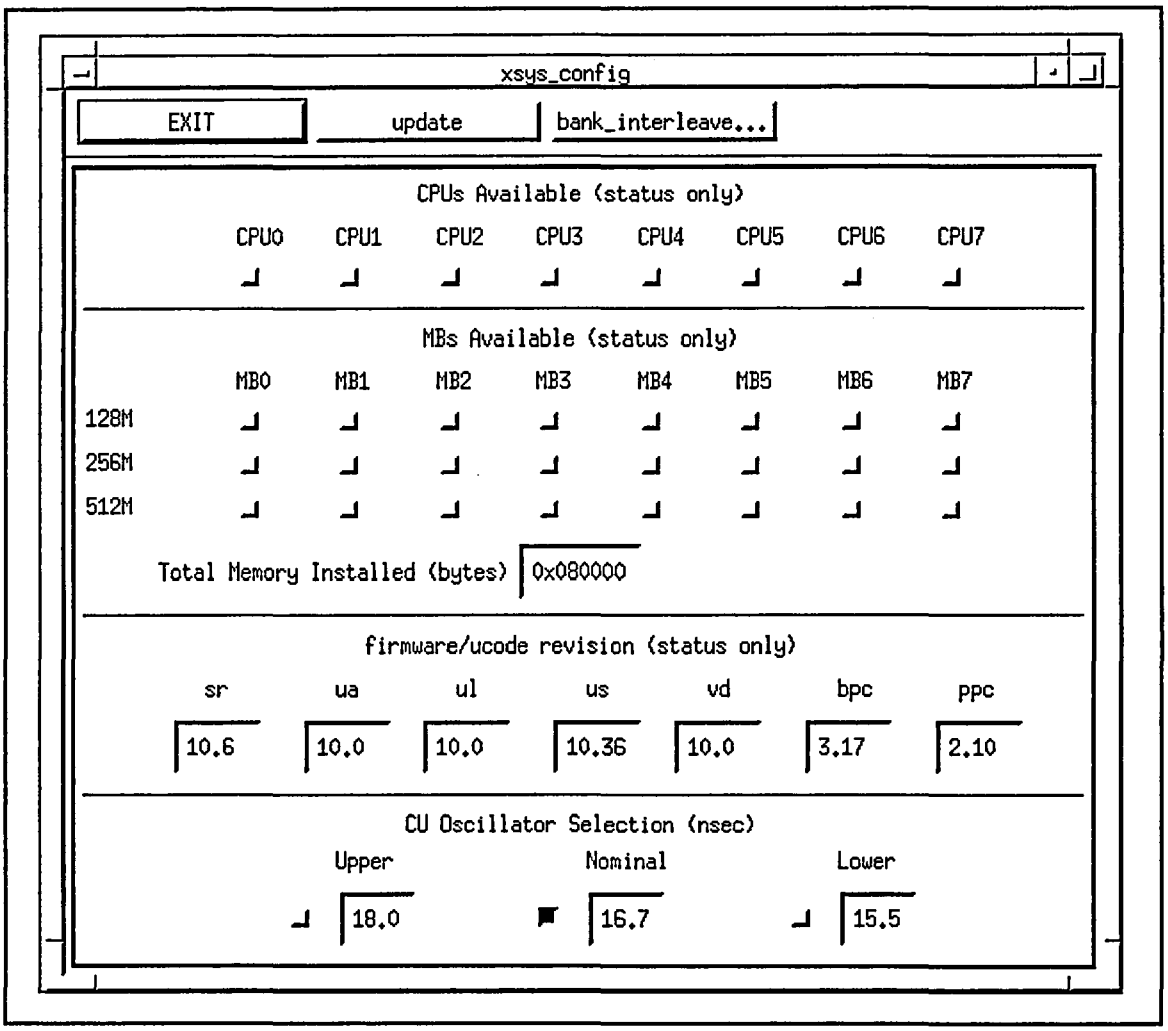
The remaining structures in the upper segment do not take input.

The upper segment of the sys_config window has these display structures:

- The CPUs Available (status only) buttons that appear active, indicate which CPUs are currently installed and active.
- The MBs Available (status only) buttons that appear active, indicate which memory boards are currently installed and active.
- The Total Memory Installed (bytes) pane gives the total amount of memory bytes currently installed and active, in hexadecimal form.
- The firmware/ucode revision (status only) gives the code revision number for each of the following storage devices:
  - sr—Scalar processor scratch RAM
  - ua—Vector processor microcontroller for arithmetic pipes
  - ul—Vector processor microcontroller for load pipe
  - us—Scalar processor control store
  - vd—Vector processor vector dispatch
  - bpc—Bay power controller EEPROMs
  - ppc—Power pallet controller EEPROMs

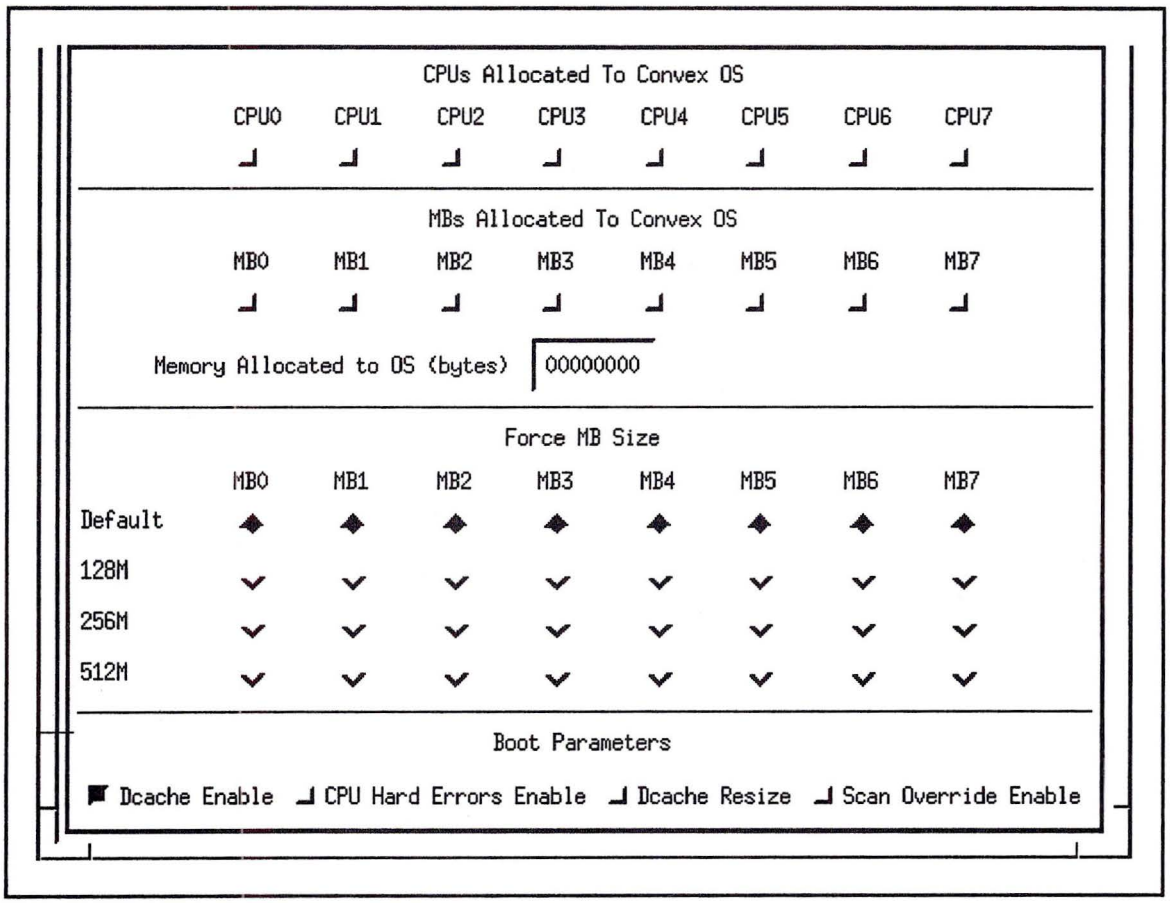
- The CU `Oscillator Selection (nsec)` panes give the periods in nanoseconds for the upper margin, the nominal, and the lower margin clocks. The active button indicates the clock period that is currently in use.

Figure 2-12
`xsys_config` window—upper segment



The lower segment of the `xsys_config` window provides toggle buttons to change configuration. Figure 2-13 shows the lower segment of the `xsys_config` window.

Figure 2-13
`xsys_config` window—lower segment



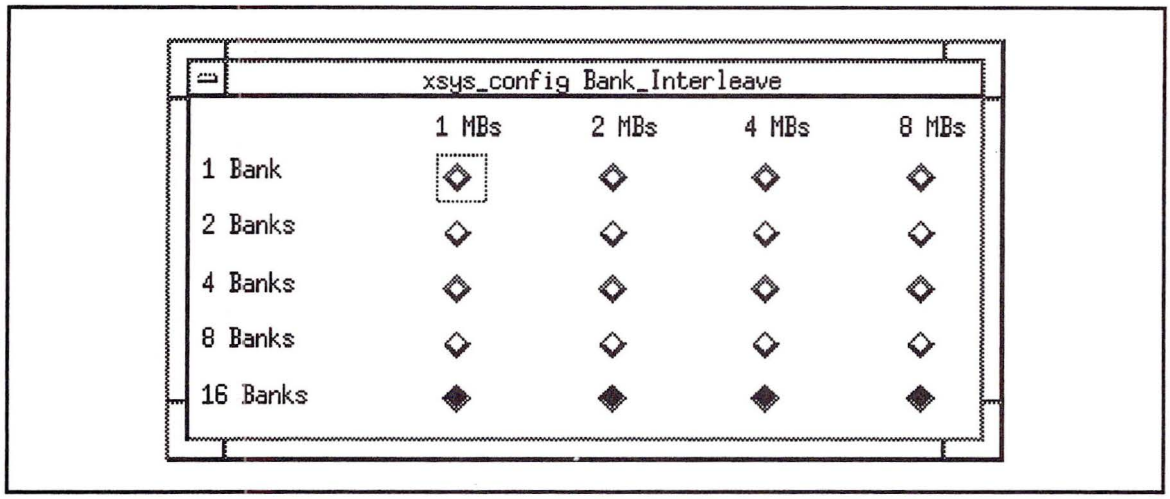
The lower segment of the `xsys_config` window has these structures:

- The `CPUs allocated to Convex OS` buttons that appear active, indicate which CPUs are currently operating under ConvexOS control.
- The `MBs Allocated to Convex OS` buttons that appear active, indicate which memory boards are currently accessible by ConvexOS.
- The `Memory Allocated to OS (bytes)` pane gives the total amount of memory bytes currently accessible, in hexadecimal form.

- The Force MB Size buttons allow you to specify the storage capacity of the memory boards. Default capacity is the full size of the memory. You can not set the memory board size parameters to sizes larger than the physical sizes of the respective memory boards.
- The Boot Parameters buttons allow you to enable or disable some CPU functions:
  - Dcache Enable, when darkened, indicates that the CPU will fetch data from the data cache if the data is there. Toggle the Dcache Enable button to force the CPU to fetch all data from main memory. The default condition is with the data cache enabled.
  - CPU Hard Errors Enable, when darkened, causes a system halt and saving of error information if a CPU hard error occurs. The converse condition causes the automatic processor recovery (APR) mode to operate, avoiding a system halt. The default condition is with CPU hard errors disabled, that is, with APR operating.
  - Dcache Resize, when darkened, causes each processor to reduce its data cache size to 1 kbyte on the next inward ring crossing from ring 4. Default condition is with data cache resizing not in effect.
  - Scan Override Enable, when darkened, invokes the scn_ovr debug script. The scn_ovr script enables you to force arbitrary data fields into scan rings. It is normally empty. This facility is for debug purposes.

The `xsys_config Bank_Interleave` window enables you to change memory interleaving. Figure 2-14 shows the `xsys_config Bank_Interleave` window.

Figure 2-14
`xsys_config Bank_Interleave` window



The memory interleaving function arranges MBs into groups of one, two, or four MBs by memory capacity. Here are some examples of interleaving operation:

- If the system has eight MBs and all have the same storage capacity, you can reduce the interleaving from the default value of 16 banks to eight-, four-, or two-bank interleaving, or disabling the interleaving entirely (one bank). To do this, select the appropriate button in the `8 MBs` column. In this case, the other columns have no effect.
- If the system has eight MBs, four with one storage capacity and four with another storage capacity (for example, four MBs have 512 Mbytes and the other four have 256 Mbytes), reduce the interleaving by selecting the appropriate button in the `4 MBs` column.
- If the system has any four MBs with the same storage capacity, any two MBs with another storage capacity, and one MB having yet a different storage capacity, reduce the interleaving of the groups by selecting the appropriate buttons in the `4 MBs`, `2MBs`, and `1MBs` columns, respectively. The columns do not interact.

2.4 System initialization and reset

The following utilities provide an orderly method for initializing and resetting machine state. Appendix B describes the automatic boot and system initialization process, and provides a procedure for a manual boot.

2.4.1 cs

Caution

If you execute `cs` while ConvexOS is running, the system will crash.

The `cs` utility loads and/or verifies the control stores of the C3 scalar processor and vector processor boards. This utility also initializes the CPU RAMs. It can process any or all control stores individually via command line options. All specified control stores of the same type load in parallel for faster processing.

The control store loader searches the current directory for the appropriate file. If the expected file is not found the utility searches for a copy in /diag/db/ctl_store. `cs` displays an error message if neither directory has the correct file.

You may load a test version of a control store by specifying the path name of the file to be loaded immediately following one of the specific control store options.

`cs` initializes the processor RAMs to zero with proper parity. All purge RAMs are also initialized.

2.4.2 diaginit

The `diaginit` utility nondestructively initializes all, or portions of the C3800 power system. `diaginit` functions whether or not ConvexOS is currently running. With `diaginit` it is possible to remove a CPU from, or add a CPU to the system, without current system users being aware of the activity.

`diaginit` normally sends a series of messages to the SPU console indicating its progress. A `-v` (verbose) switch provides additional information.

`diaginit` performs the following operations:

- Determines which bay power controllers (BPCs) are present, available, and currently not active.

- Initializes all available and inactive BPCs if possible. Sends a message to the error log file indicating which BPCs could not be initialized.
- Checks the CDB field bpc_master_fw_rev for the firmware revision levels of all BPCs initialized in the preceding step. If any BPC has the incorrect firmware revision installed, `diaginit` sends a message to the error log file and installs the correct firmware. If new firmware cannot be installed, `diaginit` sends a message to the error log file and marks the BPC(s) as unusable.
- Checks the status of all power pallet controllers (PPCs) in the bays having initialized BPCs and updates the configuration data base (CDB) accordingly.
- Checks the CDB field bpc_master_fw_rev for the revision levels of the firmware installed in the PPCs. If any PPC has the incorrect firmware revision installed, `diaginit` sends a message to the error log file and installs the correct firmware. If new firmware cannot be installed, `diaginit` sends a message to the error log file and marks the PPC(s) as unusable.
- Reads all revision and configuration information from the cop ICs on each circuit board and updates the CDB accordingly.
- Links all scan rings throughout the computer.
- Checks the power configuration of the entire computer power system to discover which boards can be powered. Updates the CDB accordingly.
- Turns on the 12 Vdc and 5 Vdc power outputs of the usable BPSs. This powers the housekeeping electronics of the pertinent PPs.
- Turns on the 300 Vdc power outputs of the useable BPSs. This supplies power to the logic circuitry power supplies of the PPs.
- Runs the spu4000 subtests that verify all scan rings and certain CPU utility functions.
- Initializes all circuit boards except for channel control units (CCUs).
- Applies power to available CCUs.
- Runs continuity tests on CCUs.
- Initializes CCUs.

2.4.3 initall

The initall utility initializes the C3800 system to support the booting of ConvexOS. initall first executes sysreset with a level 2 system reset. sysreset uses the information in the configuration data base to determine which processors, memory boards, crossbar boards, IAs, and CCUs are to be reset (see the xsys_config man page for detail on selecting configuration.) sysreset then (in a predefined order) initializes each selected board based on the individual board's requirements (see the sysreset man page for details.) Each board has its RAMs, caches, and hardware state initialized with good parity. In addition, for each processor identified to part of the C3800 complex, control stores are loaded for the scalar and vector processors.

Next, initall executes mminit. The mminit utility uses the configuration identified in the configuration data base to determine which memory boards, processors, and IA boards are to be configured.

If sysreset or mminit detects an error, initall aborts the initialization sequence and exits with a nonzero status.

2.4.4 mminit

The mminit utility initializes main memory after a system power up. The SPU, via the memory test logic (MTL), both initializes and verifies memory. If an error is detected during initialization, mminit returns a -1. Otherwise, mminit returns a 0. mminit executes the following steps:

1. Determines memory configuration of each memory board (size of DRAMs and the number of rows of DRAMs) by writing and reading various locations.
2. Generates a physical configuration map (PCM) from this information and stores both the memory configuration data from Step 1 and the PCM in the configuration data base.
3. Determines the memory interleave factors required for both the IA and the SP.
4. Default scan rings for all boards requiring PCM data: interface adapter, memory boards, crossbar, and scalar processor.
5. If specified, disables errors on all boards.
6. Fills memory with all 1s, then with all 0s.
7. If not disabled, verifies memory after each pattern test.

The type of memory configuration data obtained for each memory board includes whether 1, 2, or 4 rows of DRAMs exist on each MB and whether 1 megabit or 4 megabit DRAMs are used.

You can override the size specifications of the MBs through the `xsys_config` utility. Allowed sizes are 128 Mbytes, 256 Mbytes, and 512 Mbytes. You cannot force an MB to a size larger than the amount of installed memory. In all cases, `mminit` measures the actual size of each MB and stores the result in the CDB.

2.4.5 osclean

`osclean` kills existing SPU processes associated with ConvexOS.

2.4.6 scan_shm_init

The `scan_shm_init` utility allocates the shared memory segment used by the scan package. In doing so it initializes the rings[] array. This utility should be run only once after the SPU has been booted.

2.4.7 scn_util

`scn_util` provides a common interface between ConvexOS, I/O software, and I/O diagnostics to the C3800 Series hardware. This interface provides support for key hardware initialization and hardware clock status checking. Several options are available; some of the options should be used in combination, and others should be used independently.

2.4.8 sysreset

Resets a C3800 Series computer system.

2.5 System monitoring

The following utilities monitor the system.

2.5.1 bpccommd

The BPC daemon, `bpccommd`, performs some monitoring functions in conjunction with `bpcwatchd`. Refer to Section 2.2.2, page 2-10, for information on `bpccommd`.

2.5.2 bpcwatchd

The BPC daemon, `bpcwatchd`, monitors the overall health of the distributed power system. Refer to Section 2.2.3, page 2-11, for information on `bpcwatchd`.

2.5.3 errintd

The `errintd` daemon monitors a CONVEX C3800 Series computer system for hardware error conditions. `errintd` detects both soft and hard errors.

Soft errors include correctable single-bit memory errors, scalar processor (SP) soft errors, and and interface adapter (IA) soft errors. Single-bit memory errors are transparent to the system user. SP soft errors result from a purge RAM parity error. IA soft errors result from an error on a channel control unit (CCU). This type of error, if it occurs, is treated as a hard error if and only if the CCU failed due to a hard error.

Hard errors include all parity errors, internal references to non-existent memory, multibit memory errors, and so forth. Hard errors always result in the immediate halt of the system and are, therefore, fatal.

In addition to monitoring for errors, `errintd` starts the main memory sniffer, `mm_sniff`, if desired.

The ConvexOS boot script normally starts `errintd`. The configuration data base (CDB) time-stamps and stores all soft error output from `errintd`. The event log stores information on SP and IA soft errors.

When started, errintd initializes the hard error mask CDB entry, dynamic_cpu_harderrs_p, on the NCU. The hard error mask is active only when the enable_cpu_harderrs CDB entry is 1. The hard error mask determines which CPUs are not subject to automatic processor recovery (APR) when they generate a hard error. If enable_cpu_harderrs is 1, a CPU will not undergo APR if its dynamic_cpu_harderrs_p (p is the port number) entry is 0. The xsys_config utility enables APR and initializes the enable_cpu_harderrs CDB entry.

The CDB stores single-bit memory error data. Single-bit memory errors are isolated to the memory chip level. A count of total soft errors for each failed memory chip is maintained. By default, errintd stores a maximum of 60 memory chip entries in the CDB.

If the number of total memory chip failures reaches 75 percent capacity and a burst of errors occur (for example, at a rate of 1 every 10 seconds), the logging of new chips in error is throttled, or governed, to prevent the log from immediately reaching its capacity. Whenever throttling occurs, a message appears on the console.

In the event of a hard error (other than CCU hard errors), errintd calls the hard_logger utility. After the hard_logger analyzes the error, errintd terminates execution.

If CDB becomes full then the current contents are dumped to an archive file, CDB is reinitialized, and errintd continues with normal logging. Only one archive file is used. If CDB becomes full again, the current contents in CDB overwrite the old contents of the archive file. This archiving occurs at most once a day.

2.5.4 errlogd

errlogd provides user processes with a single point of access to the event/error log. As errlogd acts as a funnel for all messages, it also provides a single point of contact for processes that wish to be informed of certain events. errlogd permits user processes to request information about some, or all, of the events that are occurring in the system. As each event is received, errlogd scans a linked list of requests and forwards the current event to each process that has requested information about it.

2.5.5 event_browser, xevent_browser

A C3800 Series computer communicates with the SPU error_log file whenever a significant *event* occurs during its operation. event_browser enables you to examine the error_log file without displaying the whole file. xevent_browser provides a set of windows to expedite browsing the file.

Events range in severity from emergencies to normal operational changes and debugging information. The computer sends any one of hundreds of possible *messages* for each event. Each message has an *identification number* and a *description*. The computer may send the same message multiple times if multiple similar events occur.

Each message is one of 45 message *types*. Table 2-3 lists the message types.

Table 2-3
Event message types

Type number	Type code	Type name	Type description
0x0000	EC_UNKNOWN	<Unknown>	Unknown type
0x0001	EC_CONFIG_ERR	Config Error	Hardware or software configuration error detected
0x0002	EC_HARD_ERR	Hard Error	Hard error output message from extractor
0x0003	EC_SOFT_ERR	Soft Error	Soft error detected
0x0004	EC_ENV_ERR	Env Error	Environmental error detected
0x0005	EC_HW_HANG	HW Hang	SPU detected hardware hang
0x0006	EC_OS_PANIC	OS Panic	ConvexOS panic detected
0x0007	EC_OS_IO_ERR	OS I/O Error	ConvexOS device driver error
0x0008	EC_OS_REBOOT	OS Reboot	ConvexOS reboot
0x0009	EC_OS_HANG	OS Hang	ConvexOS hang detected by SPU
0x000a	EC_TEST_START	Test Start	Invocation of diagnostic test program

Table 2-3 (continued)
Event message types

Type number	Type code	Type name	Type description
0x000b	EC_TEST_END	Test End	Completion status of a diagnostic test program
0x000c	EC_SUBTEST_FAIL	Subtest Fail	Diagnostic subtest failed
0x000d	EC_SUBTEST_PASS	Subtest Pass	Diagnostic subtest passed
0x000e	EC_HW_INIT_PASS	HW Init Pass	Invocation of a hardware initialization program
0x000f	EC_HW_INIT_FAIL	HW Init Fail	Completion status of hardware initialization program
0x0010	EC_HW_CONFIG_CHG	HW Config Chg	Change of hardware configuration (board swap, and so forth.)
0x0011	EC_IO_CONFIG_CHG	IO Config Chg	Change of I/O configuration (new disk, tape, and so forth.)
0x0012	EC_SW_CONFIG_CHG	SW Config Chg	Change of software configuration (new firmware, OS, diagnostics)
0x0013	EC_UTIL_START	Util Start	Utility started—generic message
0x0014	EC_UTIL_END	Util End	Utility ended—generic message
0x0015	EC_SPU_REBOOT	SPU Reboot	SPU boot, reboot, power down
0x0016	EC_DAEMON_START	Daemon Start	Required daemon (support) software started
0x0017	EC_DAEMON_END	Daemon End	Required daemon (support) software ended
0x0018	EC_MAIL_CXTS_REPORT	Mail CXTS Report	Mail CXTS report
0x0019	EC_SPU_IF_BUSERR	SPU/IF Bus Error	Bus error occurred in SPU/CU space
0x001a	EC_SST_DATA_TEST	SST Data	SST data query results
0x001b	EC_SST_BD_FAIL	SST Board Fail	SST indicates possible board failure
0x001c	EC_SW_ERROR	SW Error	Software error detected

Table 2-3 (continued)
Event message types

Type number	Type code	Type name	Type description
0x001d	EC_SW_WARN	SW Warning	Software warning condition detected
0x001e	EC_SW_INFO	SW Info	Software information condition detected
0x001f	EC_SW_EVNSTART	SW Event Start	Software event started
0x0020	EC_SW_EVNSTOP	SW Event Stop	Software event ended
0x0021	EC_HARDLOG_START	HardLog Start	Hard logger execution started
0x0022	EC_HARDLOG_END	HardLog End	Hard logger execution ended
0x0023	EC_OS_BOOT_FAIL	Failed OS Boot	System software failed booting
0x0024	EC_OS_BOOT_START	Start of OS Boot	ConvexOS boot process started
0x0025	EC_OS_SHUTDOWN	ConvexOS shutdown	ConvexOS shutdown started
0x0026	EC_EGOS_PANIC	EGOS Panic	EGOS has panicked
0x0027	EC_IO_CONF_PASS	IO Dev Pass	A pass occurred during `autoconf`
0x0028	EC_IO_CONF_FAIL	IO Dev Fail	A failure occurred during `autoconf`
0x0029	EC_OS_BOOT_COMPLETE	OS BOOT Complete	A failure occurred during boot
0x002a	EC_NOTIFICATION_ON	Auto Notify On	Turn automatic notification on
0x002b	EC_NOTIFICATION_OFF	Auto Notify Off	Turn automatic notification off
0x002c	EC_ENV_INFO	Env Info	Environmental status information message
0x002d	EC_ENV_TRIM_INFO	Voltage Trim	Voltage trim information message

Each message is associated with one of 6 *sources*. Table 2-4 lists the event message sources.

Table 2-4
Event message sources

Source number	Source code	Source name
0	DIAG	diagnostic test or utility
1	OS	Convex OS
2	IO	Convex device driver
3	APP	application program
4	ENV	message directly from CCU
5	IODIAG	IO diagnostic or utility

The `xevent_browser` window enables you to scan selected parts of the `error_log` file.

The `xevent_browser log_select` window enables you to scan error log files that have been moved to other file names for archiving.

The `xevent_browser filter` window enables you to select specific sets of entries in an error log file for display in the `xevent_browser` window.

The `xevent_browser SaveReport` window enables you to write the contents of the `xevent_browser` window to a specified file.

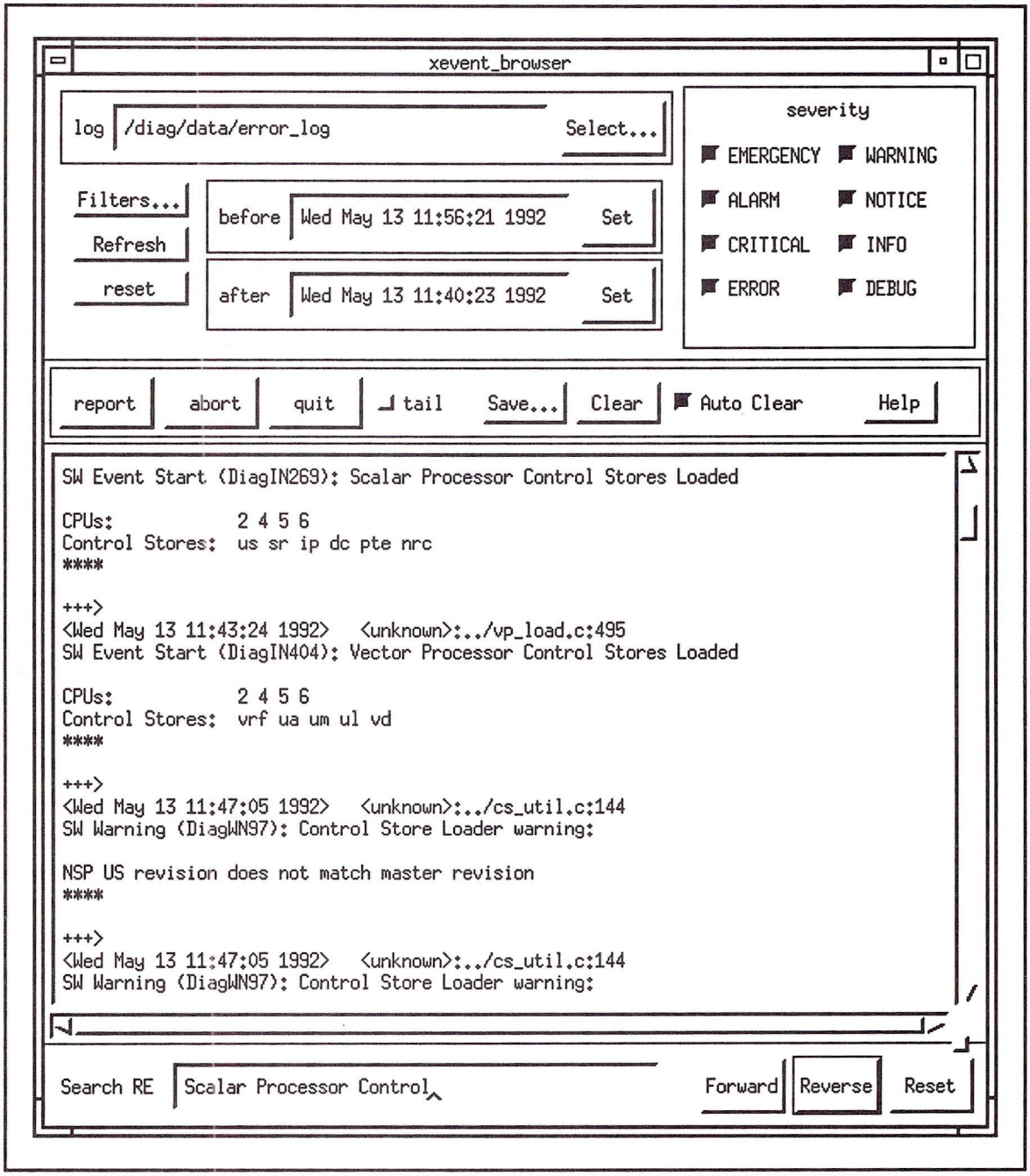
2.5.5.1 `xevent_browser` window

Figure 2-15 shows the `xevent_browser` window. The structures in the window have the following functions:

- The `log` pane displays the path and filename of the file currently being displayed.
- The `Select ...` button displays the `xevent_browser log_select` window that aids in selecting the file to be displayed. Do not use the `xevent_browser log_select` window to select a custom log file. The default error_log file should always be used to log events as they happen.
- The `severity` panel has 8 buttons for selecting the type of messages to be displayed. Select all buttons to display the whole file or to filter the file in other ways.

- The `Filters ...` button displays the `xevent_browser filter` window that offers a range of options for selecting the types of messages displayed.
- The `Refresh` button repaints the `xevent_browser window` on the monitor screen.
- The `reset` button reinitializes the `xevent_browser` program.
- The `before` and `after` windows may contain dates and times that allow displaying of data for a desired time interval. The `Set` buttons place a cursor in the corresponding window to edit the date-time group.
- The `report` button displays the desired data in the `xevent_browser` display window pane.
- The `abort` button stops the `xevent_browser` from reading a file.
- The `quit` button exits the `xevent_browser` program.
- The `tail` button is a toggle switch. If selected, it causes the xevent_browser program to read messages coming into the `error_log` file in real time.
- The `Save ...` button displays the `xevent_browser SaveReport` window.
- The `Clear` button clears the `xevent_browser` display window pane.
- The `Auto Clear` button is a toggle switch. If selected, it clears the `xevent_browser` display window pane every time you select the `report` button. If the `Auto Clear` button is deselected, the new report is appended to the existing data in the `xevent_browser` display window pane.

Figure 2-15
`xevent_browser` window



- The large pane in the `xevent_browser` window is the display pane. This pane displays report data selected from the error log file. An individual event report has the format shown in Figure 2-16.

Figure 2-16
Individual event report format

```
<Wed May 13 11:43:24 1992>   <unknown>:../vp_load.c:495.
SW Event Start (DiagIN404): Vector Processor Control Stores Loaded
```

The items in an individual event report contain the following fields:

- `Wed May 13 11:43:24 1992` is the date and time of the event.
- `<unknown>` is the name of the process that generated the event message. In the event message displayed, the process is unknown. If the process is known, the field has the form process_name(process ID). An example field is:

  ```
  /diag/bin/bpcwatchd(246)
  ```

- `vp_load.c` is the name of the file containing the process that generated the event message.
- `495` is the number of the line in the process that generated the event message.
- `SW Event Start` is the message type.
- `DiagIN` is the message source.
- `404` is the message identification number.
- `Vector Processor Control Stores Loaded` is the message description.

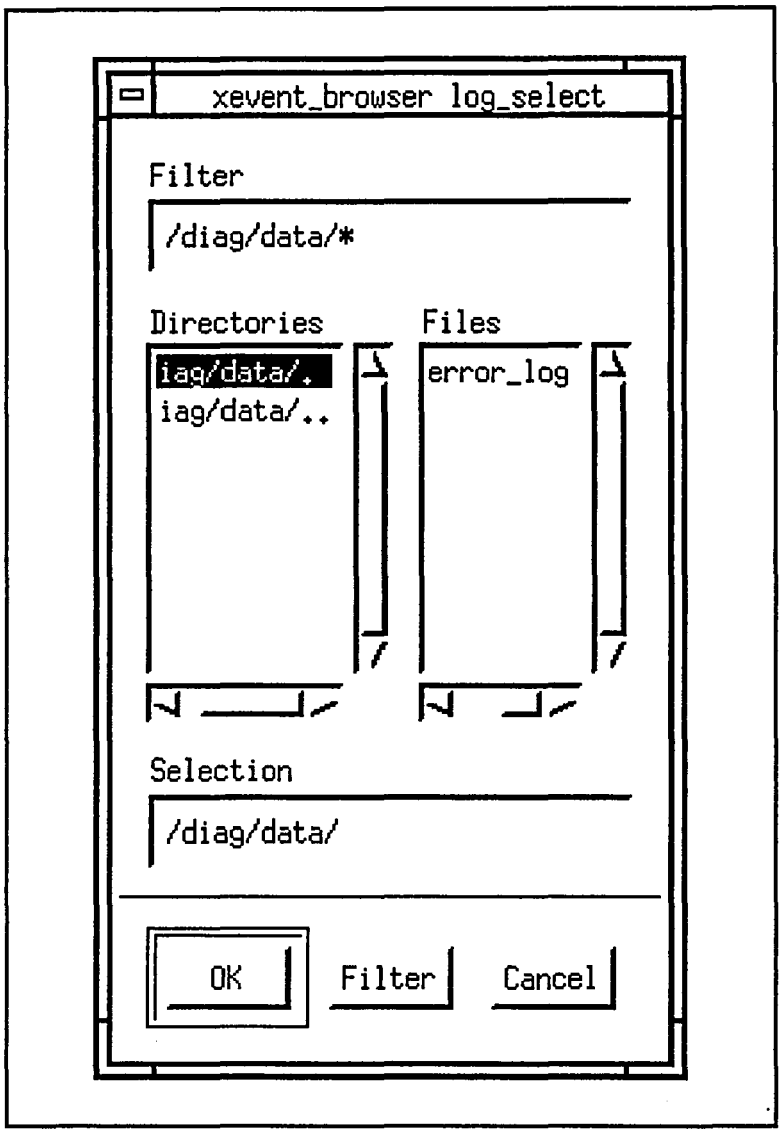
- The Search RE pane at the bottom of the xevent_browser window enables you to search the data in the display pane for the regular expression defined in the Search RE pane. Selecting this pane causes a cursor to appear, enabling you to enter a regular expression. Click on the Forward button to search for the next occurrence of the expression. Click on the Reverse button to search for the first previous occurrence of the expression. Click on Reset to search from the beginning of the report.

2.5.5.2 xevent_browser log_select window

Figure 2-17 shows the xevent_browser log_select window. Use this window only to display an archived event log. The structures in the window have the following functions:

- The Filter pane shows the current directory path and types of files displayed.
- The Directories pane shows the parent directory link and the available child directory paths from the current directory. It may not show the full width of the directory display. You may widen the window or use the pane scroll bar to display more of the directory paths.
- The Files pane shows the files available in the current directory that match the specified filter.
- The Selection pane shows the currently selected file, with its path.
- The OK button selects the file indicated in the Files pane as the file to browse.
- The Filter button selects the filter typed into the Filter pane.
- The Cancel button returns the window to its original state before you altered any of the panes.

Figure 2-17
`xevent_browser log_select` window



Use the following method to select a file for reading or writing:

Step 1 Move to the desired directory. Repeat either of the following two actions until the appropriate path is displayed.

- To move to a child directory of the current directory, click on a child directory name in the `Directories` pane.
- To move to the parent directory of the current directory, click on the parent directory link (the `..` directory) in the `Directories` pane.

The selected directory is highlighted in the `Directories` pane. The selected path appears in the `Filter` pane. If the move is to the parent directory, the `Selection` pane clears. If the move is to a child directory, the `Selection` pane displays the selected directory path.

Step 2 Click on the `Filter` button at the bottom of the window.

A highlight appears around the `Filter` button. The selected directory becomes the current directory. The parent directory link and the child directories of this new current directory appear in the `Directories` pane. A list of files in the current directory appears in the `Files` pane.

Step 3 Click on the desired file. The file becomes highlighted.

To select the file, click on the `OK` button. The contents of the file appear in the display pane of the `xevent_browser` window.

To abort the operation, click on the `Cancel` button.

2.5.5.3 `xevent_browser filter` window

Figure 2-18 shows the `xevent_browser filter` window. Use this window to select and deselect the events to display in the `xevent_browser` window display pane.

A C3800 Series computer reports an event by selecting one of several hundred pre-written messages and transmitting it to the SPU `event_log` file. Every message has a permanent message identification number assigned to it. If several similar events occur, the same message may appear in the `event_log` file several times. `xevent_browser` can display event logs from either the `event_log` file or from archived event log files having other file names. You may filter an event log file by any of the following ways:

- Select 1 or more message identification numbers and display those messages.
- Select 1 or more message identification numbers and inhibit the display of those messages.
- Select 1 or more message sources and display all messages from those sources.
- Select 1 or more message sources and inhibit the display of all messages from those sources.
- Select 1 or more message types and display all messages of those types.
- Select 1 or more message types and inhibit the display of all messages of those types.
- Enter a regular expression (grep) and display all messages containing that regular expression.
- Enter a regular expression (grep) and inhibit the display of all messages containing that regular expression.

The structures in the window have the following functions:

- The following panes display the numbers of the selected or deselected message parameters:

  `-message`

  `+message`

  `-source`

  `+source`

  `-type`

  `+type`

  You can remove a number from any of these windows by clicking on the number.

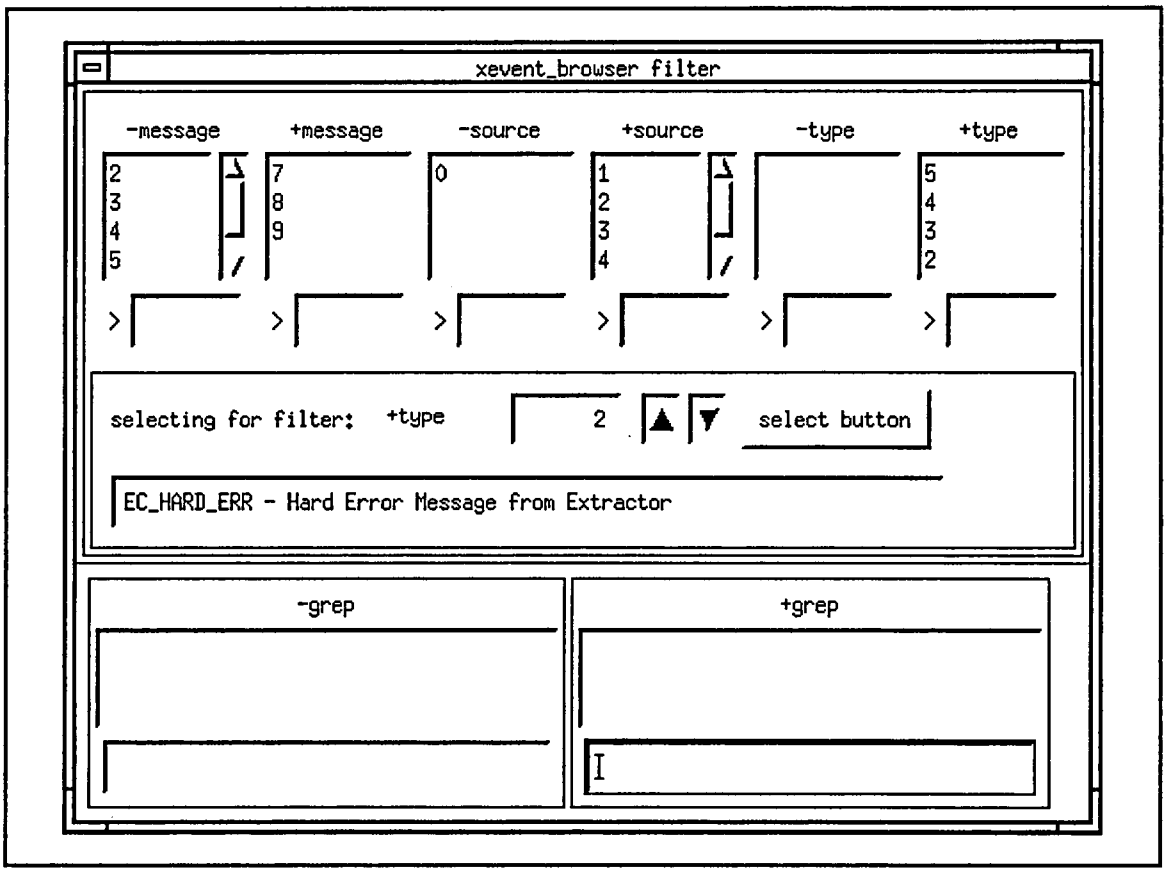
- Each of the foregoing panes has a one-line pane directly beneath it, into which you can enter the number corresponding to the parameter you wish to display or inhibit.

- The `selecting for filter:` line is a user-friendly way of entering numbers into the message parameter panes. Use the following steps to enter a parameter:

  1. Select a parameter by clicking on the one-line pane underneath the desired parameter. The parameter name appears to the right of the `selecting for filter:` text.
  2. Click on either of the triangular buttons (up or down) to the left of the `select button` text. The code and the name of the selected parameter appears in the one-line window directly beneath the `selecting for filter:` text.
  3. Press **RETURN** to enter the selected parameter. The parameter number appears in the selected parameter display pane.

- Use the `+grep` pane to enter a regular expression that causes display of messages containing the expression, or the `-grep` pane to enter a regular expression that inhibits message display. Enter the desired regular expression in the one-line pane beneath either of these panes. The expression appears in the `+grep` or `-grep` pane after pressing **ENTER**. Delete regular expressions from the `+grep` and `-grep` panes by clicking on them.

Figure 2-18
`xevent_browser filter` window



2.5.5.4 `xevent_browser` SaveReport window

Figure 2-19 shows the `xevent_browser SaveReport` window. Use this window to save a filtered or unfiltered event log in an archival file. Use the procedure in section 2.5.5.2 to select the write file.

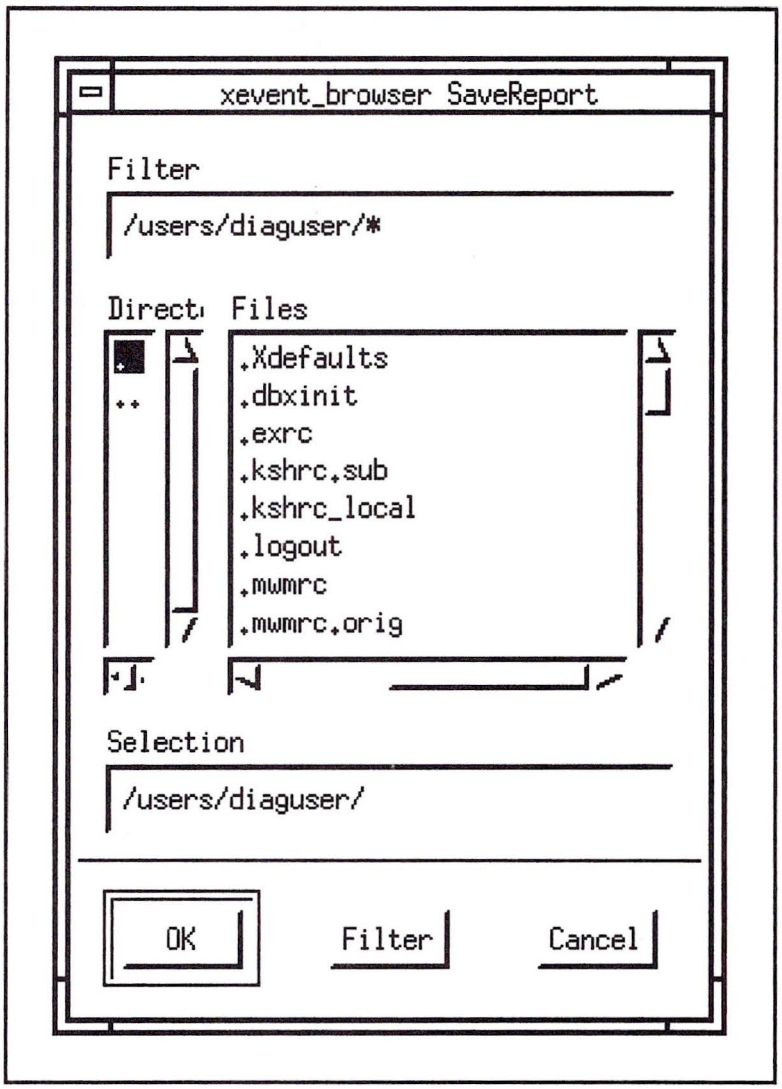
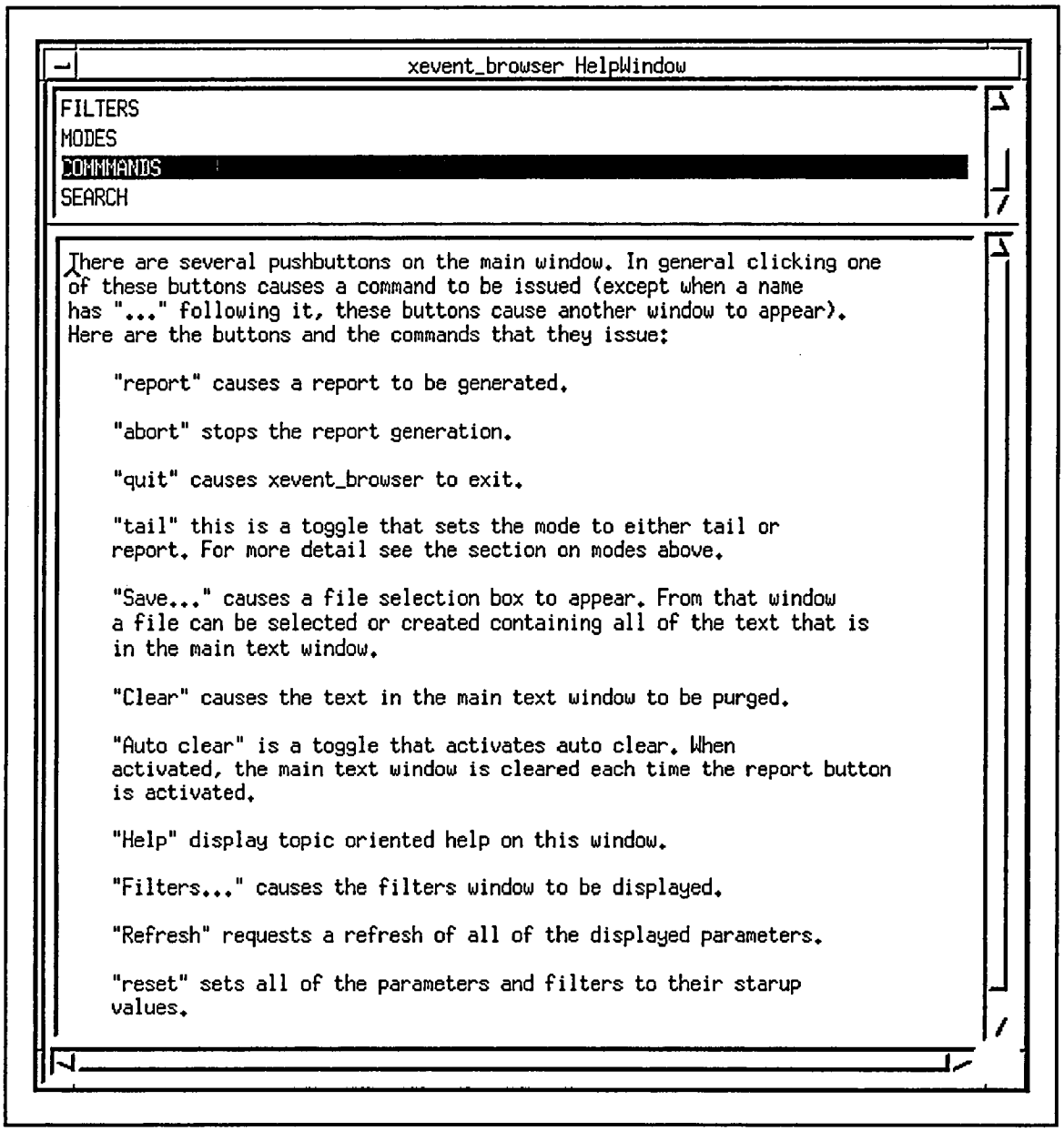


Figure 2-19
`xevent_browser SaveReport` window

2.5.5.5 xevent_browser HelpWindow

Figure 2-20 shows the xevent_browser HelpWindow. Click on the Help button in the xevent_browser window to display the xevent_browser HelpWindow.

Figure 2-20
xevent_browser HelpWindow



2.5.6 logmsg

logmsg enables a shell script to pass an ASCII message to the event log. This mechanism simplifies incorporating messages into existing tools.

2.5.7 mm_sniff

The mm_sniff program reads all main memory within a specified amount of time. The intention is to detect locations with a single-bit error. Once detected, errintd attempts to eliminate the error by performing a memory scrub operation on the memory location in error.

If a location contains a single-bit error and is not read for a long time, it could potentially drop another bit. This would result in a double-bit error that is not correctible and would halt the system.

The memory sniffer always reads main memory in four-page (16 Kbyte) groups. Upon invocation, based on the sniff rate in Mbytes per day, mm_sniff calculates the number of seconds to sleep between each read. In the event the calculated sleep time is less than 15 seconds, the value is forced to 15 seconds. This time and the time required to sniff the entire memory system are displayed on the console.

2.5.8 powermon, xpowermon

The powermon and xpowermon utilities monitor power and temperature parameters throughout a C3800 Series computer. Refer to Section 2.2.5.1, page 2-12, for information on powermon and xpowermon.

Diagnostic shell (dsh) 3

The C3800 series software includes a Korn shell based diagnostic shell (dsh) that incorporates commands previously available as a part of different executable utilities in other CONVEX environments. In particular, the utilities formerly invoked using the dshell and iscn commands are now available under dsh. All the functionality of these utilities is present in this new shell environment. Incorporating them directly into the shell has eliminated much overhead that the previous implementation required.

Caution

dsh incorporates powerful commands that operate directly on C3800 Series hardware. Incorrect use of dsh commands can cause a system crash requiring a total restart of both the computer and its workstation interface.

The dsh software resides in /diag/bin/dsh.

dsh is an enhanced version of the AT&T Korn Shell (ksh) with all of the functionality of ksh, as well as new commands that are suited to the C3800 Series diagnostic environment. For specific information on standard ksh functionality, refer to the ksh online man page, or *The Korn Shell Command and Programming Language* by Morris I. Bolsky and David G. Korn.

dsh has the following ksh features:

- Compatibility with the Bourne Shell. Existing diagnostic shell scripts written to use with the Bourne Shell are executable by dsh with few or no modifications.
- Powerful trace capability, enhanced with new single step and pause features.
- A command history similar to the history kept by the C Shell.
- Supports vi and emacs command line editing capabilities, making dsh a powerful tool from the command line as well as from shell scripts.

New dsh commands have their own wild card capabilities in addition to standard ksh command path name expansion and pattern matching. For example, the iscn commands have pattern matching capabilities to simplify access to scan ring fields:

- ? denotes any single character, or no character, in a command name. The command:

  > list mbs7:nmcf?_rerr

  lists the following scan rings:

  ```
  mbs7:nmcfe_rerr
  mbs7:mmcfo_rerr
  ```

- * denotes any string of characters, or no character, in a command name. The command:

  > list mbs7:nmcfo_data*

  lists the following scan rings:

  ```
  mbs7:nmcfo_data
  mbs7:nmcfo_data_sel
  ```

3.1 dsh commands

Table 3-1 lists the commands that have been added to the ksh environment to create dsh. dsh directly provides control flow functionality that iscn previously provided.

Note

Command aliases that were previously available by default are not available in this environment. dsh provides an alias mechanism you can use to create your own aliases.

Table 3-1
dsh commands

Command	Meaning
:	Null command
.	Read a specified file as input (same command as autoload)
alias	Define and display aliases
align[1,2]	Check scannability of all specified rings
autoload	Read a specified file as input
bdrev[2]	Check or display the revision level of a board
bdtype[2]	Check the presence of a board type in a slot
bg	Place job in background for execution
break	Exit from smallest enclosing loop or case
cd	Change the current working directory
clear	Clear the screen
clearbuf[1,2]	Clear scan buffers
clock[1,2]	Generate one or more system clock pulses
continue	Go to the top smallest enclosing loop or case
dump[2]	Dump scan ring save buffers to a named file
echo	Echo arguments to standard output
eval	Read and execute command in current environment
even_parity[2]	Generate even parity
exec	Replace ksh with command without creating a new process

[1] Command may be destructive.
[2] dsh enhancement to ksh.

Table 3-1 (continued)
dsh commands

Command	Meaning
exit	Exit from ksh now
export	Mark names for automatic export to subsequent commands
fc	Display, edit, and re-execute previous commands
fg	Bring job to foreground for execution
fprint[2]	Formatted print
get[1,2]	Get a field or register value
getopts	Check argument for legal options
halt[1,2]	Take one or more boards out of the run state
help[2]	Help
jobs	Display job related information
kill	Send a signal to the specified jobs
let	Evaluate one or more arithmetic expressions
list[2]	List the fields associated with the named scan ring
newgrp	Equivalent to exec /bin/newgrp
odd_parity[2]	Generate odd parity
print	Print to the screen
put[1,2]	Put a field or register value
pwd	Display the current working directory name
read	Read a line and split it into fields
readonly	Mark variable as read-only
restore[1,2]	Restore a scan ring from a buffer
return	Cause ksh function to return to invoking shell
save[1,2]	Save the state of a scan ring to a buffer
scnclear[1,2]	Clear a scan ring to all zeros
scnout[1,2]	Create identical scan rings for multiple boards
scnrun[1,2]	Put a board in the run state

[1] Command may be destructive.
[2] dsh enhancement to ksh.

Table 3-1 (continued)
dsh commands

Command	Meaning
set	Set or unset ksh options
shift	Shift positional parameters left
test	Check file-related information
<*testname*> [2]	Test control
time	Display how much time this shell and its children have used
tlog[2]	Allow multiple failures
tloop[2]	Enable loop mode
tmsgs[2]	Message control
tpause[2]	Test execution pause control
trap	Specify or display actions to take when conditions arise
tstatus[2]	Print test flag status
typeset	Set or unset attributes of functions and variables
ulimit	Set or display system resource limits
umask	Set or display file creation mask
unalias	Delete alias from alias list
undump[2]	Read a named file to reconstruct a scan ring save buffer
unset	unset values and attributes of functions and variables
ustep[1,2]	Generate a single clock edge for the specified boards
verify[2]	Enable read or compare verification
wait	Wait for jobs to terminate
whence	Find absolute path name or type

[1] Command may be destructive.
[2] dsh enhancement to ksh.

Table 3-1 (continued)
dsh commands

Command	Meaning
`[[ <test expression> . . .]]`	Check conditional expression primitive

`if`
`then`
`elif`
`else`
`fi`
	If, then, and else conditional construct

`case <word> in` `esac`	Case statement conditional construct

`for`
`do`
`done`
	For statement iteration construct

`select`
`do`
`done`
	Select statement iteration construct

`while`
`do`
`done`
	While statement iteration construct

`until`
`do`
`done`
	Until statement iteration construct

`( compound-list )` `{ compound-list }`	Run compound-list in subshell environment Run compound-list in current environment

`function <name>`
`{`
`compound-list`
`}`
	Define a function

`<name> ()`
`{`
`compound-list`
`}`
	Define a function (alternative form)

3.2 Numeric data

The dsh extensions to ksh recognize numeric data in the traditional sense:

- Hexadecimal data have the 0x prefix
- Octal data have the 0 prefix
- Decimal data have no prefix

dsh and ksh treat numbers that do not indicate a base as decimal. They use the base#data notation to indicate a base. Both dsh and ksh functions recognize this notation. ksh functions do not recognize the traditional representations. Arithmetic and logical operations can be performed only on base#data representations in all parts of the shell. The non-dsh enhancements treat the more traditional representations as character strings.

Example:

`0x00ff12ec` (Traditional notation)

`16#00ff12ec` (Recommended notation)

3.3 Command descriptions

The commands and their syntax as listed in this section document the enhancements that have been made to ksh, and not the functionality that is inherent to ksh.

3.3.1 dsh enhancement commands

3.3.1.1 align

This command has the following syntax:

align

Clocks may not be stopped at a point where CCUs are scannable. The align command issues the correct number of clocks to permit scanning of CCUs. Replaces previous adjust command.

3.3.1.2 autoload

This command has two forms. Either of the following syntax forms is valid:

autoload *filename*
. *filename*

Replaces the previous include command. A combination of the two achieves the same functionality with greater flexibility.

3.3.1.3 bdrev

This command has the following syntax:

bdrev *ring*

Identifies the revision level of the board in the designated slot. `bdrev` returns the revision information to the standard output. Figure 3-1 lists the C3800 Series scan rings.

Figure 3-1
Scan ring names

Channel control units—1 ring per board

```
ccu00-ccu39
```

CPU utilities board—1 ring

```
cu
```

Interface adapter board—4 rings

```
ia8
iac8
ial8
ias8
```

Memory boards—3 rings per board

```
mb0-mb7
mbl0-mbl7
mbs0-mbs7
```

Scalar processor—1 ring per board

```
sp0-sp7
```

Vector processor—1 ring per board

```
vp0-vp7
```

Crossbar control logic—1 ring

```
xcls
```

XRT—4 rings

```
xrtce
xrtco
xrte
xrto
```

XS0—4 rings

```
xs0ce
xs0co
xs0e
xs0o
```

XS1—4 rings

```
xs1ce
xs1co
xs1e
xs1o
```

The resulting display has the form:

```
001245.0B
```

The decimal digits to the left of the period is the board type number. The hex digits to the right of the period are the assembly revision.

3.3.1.4 `bdtype`

This command has the following syntax:

`bdtype` *ring* | *board*

Checks the specified slot for the specified board name and returns a 1 if the slot is empty and a 4 if the slot contains a board other than the one indicated in the command. The command returns nothing if the correct board is in the slot. Table 3-2 shows valid board names and their functions.

Table 3-2
Board names

Board name	Board function
MB0 - MB7	Memory
SP0 - SP7	Scalar processors
VP0 - VP7	Vector processors
CU	Communications and utilities
IA0 - IA8	Interface adapters
XIOP0 - XIOP8	Extended input/output processors
CCU0 - CCU39	Channel control units
XRTE, XRTO	Crossbar return data, even and odd sides
XS0E, XS1E	Crossbar send data, even side
XS0O, XS1O	Crossbar sand data, odd side
XCL	Crossbar control logic
CCUPP0 - CCUPP9	Channel control unit power pallets
XPE, XPO	Crossbar power, even and odd

The error message has the form:

```
* * * * Last command returned status 4 * * * *
```

3.3.1.5 clearbuf

This command has the following syntax:

clearbuf *buffer1* [*buffer2* . . .]

Clears the specified scan buffer. The valid buffer names are spu, out, mask, compare, and result.

3.3.1.6 clock

This command has the following syntax:

clock [*number*] [*board board* . . .]

Generates a specified number of system clocks for a specified set of boards. By system clock we mean a single 1x, two 2x, and/or three 3x clocks. If *number* is not specified, the specified board(s) are given a 1x clock. If *number* is specified, it indicates the number of clocks the system should advance in burst mode. The *board* arguments indicate the board (or boards) to which the command applies. Table 3-2 shows valid board name and their functions.

3.3.1.7 dump

This command has the following syntax:

dump [**-a**] [*filename*]

Writes out to the specified file all of the rings that have previously been saved with the `save` command. The file can then be used to restore ring state in conjunction with the `undump` command. The default file name is scn_dumpfile in the current directory.

The `-a` option permits rings to be saved at the end of the named file rather than overwriting the file if it already exists. Multiple copies of the same ring may be saved to a single file. Examples are:

save *ring*

dump -a *file*; **clock**

save *ring*

dump -a *file*

in the same session, or from multiple sessions. If a ring is saved multiple times in the same file, only the last saved version would actually be restored via the `undump` command.

3.3.1.8 `even_parity`

This command has the following syntax:

`even_parity` *number*

Generates even parity in a format where the parity of the most significant byte of a 32-bit word goes into the least significant bit of the parity nibble. The result is written to the standard output.

3.3.1.9 `halt`

This command has the following syntax:

`halt` [*board board* . . .]

Disables incoming clocks on a specified board or boards. The default is to halt all boards in the system.

3.3.1.10 `list`

This command has the following syntax:

`list` *ring*[:*field*]

Lists all fields in the named ring to the standard output. The output of this command can then be redirected to a file or to a pager such as `more`. The field portion of the ring name may be a wild card according to the following syntax:

*	Matches any string, including the null string.
?	Matches any single character.
[. . .]	Matches any one of the enclosed characters.
-	Between a pair of characters within a bracketed expression matches any character lexically between the pair, inclusive.

3.3.1.11 `odd_parity`

This command has the following syntax:

`odd_parity` *number*

Generates odd parity in a format where the parity of the most significant byte of a 32-bit word goes into the least significant bit of the parity nibble. The result is written to the standard output.

3.3.1.12 `restore`

This command has the following syntax:

`restore` {*ring1 ring2 . . . ringn*]

Restores rings that were previously saved with a `save` command.

3.3.1.13 save

This command has the following syntax:

save [*ring1 ring2 . . . ringn*]

Saves the specified scan ring(s) into a buffer. The ring(s) can later be restored via the `restore` command.

3.3.1.14 scnclear

This command has the following syntax:

scnclear [*ring1 ring2 . . . ringn*]

Clears the specified ring(s) to all zeros. The default is to clear all rings in the system.

3.3.1.15 scnout

This command has the following syntax:

scnout *ring_source ring_dest*

Copies an entire scan ring from one slot to another.

3.3.1.16 scnrun

This command has the following syntax:

scnrun [<*board1 board2 . . . boardn*>]

Places the specified board(s) in the run state. The default is to place every board in the system in the run state.

3.3.1.17 undump

This command has the following syntax:

undump [*filename*]

Reads the specified file and builds the restore buffer with it.

3.3.1.18 ustep

This command has the following syntax:

ustep [*board board . . .*]

Generate a single clock for all boards (and clocks) specified. That is, a single 1x, 2x, and/or 3x clock. The default is to `ustep` all boards in the system.

3.3.1.19 verify

This command has the following syntax:

verify [**on** | **off**]

When enabled, forces all subsequent puts to be reread and validated. If there is an error, the expected, returned, and the XOR of the two are printed to the standard output.

3.3.2 Commands for I/O tests

3.3.2.1 *testname*

This command has the following syntax:

testname [**-s** | **-c** *repetition factor*]

Replaces the previous `test` *testname* command. Displays subtest and class menus, and controls execution on a test, subtest, or class basis.

3.3.2.2 `tlog`

This command has the following syntax:

tlog [**off**] [**-s** | **-t** *num failures*]

Replaces the previous `log` command. Disables or enables test or subtest multiple fail modes.

3.3.2.3 `tloop`

This command has the following syntax:

tloop [**off**] [**-s** *subtest*] [**-t**]

Replaces the previous `loop` command. Causes test or subtest to loop.

3.3.2.4 `tmsgs`

This command has the following syntax:

tmsgs [**off**] [**-f** *long/short*] [**-s** | **-t**]

Replaces the previous `msgs` command. Disables or enables test/subtest test result messages.

3.3.2.5 `tpause`

This command has the following syntax:

tpause [**off**] [**-f** | **-b** | **-e** *subtest number*]

Replaces the previous `pause` command. Selects if and when a test or subtest should pause and await user intervention.

3.3.2.6 `tstatus`

This command has the following syntax:

tstatus

Replaces the previous `status` command. Formats and prints test flag status.

3.3.3 Other features

The following commands describe other features available with `dsh`.

3.3.3.1 `DFMT=`

This command sets the data format as follows:

`DFMT=1` Byte format

`DFMT=2` Halfword format

`DFMT=4` Word format

3.3.3.2 `get`

This command has the following syntax:

`get` {**`-q`**] *what*[,*count*] [*what*[,*count*]

The `get` command is generic and accesses all parts of the system, including SWIS registers, SWIP registers, NCU registers, SPU memory, and scan rings. It is also possible, but not recommended, to access CONVEX main memory through this command. The <*what*> argument specifies the item to be read. The `-q` (quiet) option suppresses the printing of extra data such as ring and field name when printing the results of `get`.

The <*what*> argument indicates the item to be accessed, the size of that item, and in some instances, the specific element that is to be accessed. Figure 3-2 shows some examples of `get` commands.

Figure 3-2
Sample `get` commands

```
# get the contents of a register in NWI space
get nwi_status

# get the contents of a register from NWI uart channel 4
get uart_stat[4]

# get a byte from a specific offset in NWI space
get 0x3:b

# get a value from memory using the default format
DFMT=4    # set the default format to 4 bytes (i.e. a word)
get 0xff00f340
get 0xffff:b,6  # get 6 bytes of data at address 0xffff
get 0x9630:s,6  # get 6 short words of data at address 0x9630

# get the contents of a scan ring field
get ringname:fieldname
```

Figure 3-3 provides a list of SWIS, SWIP, NCU, and NCI registers accessible through the `get` command.

Figure 3-1 on page 3-8 lists the C3800 Series scan rings. You may obtain lists of scan ring field names through use of the `list` command.

3.3.3.3 `fprint`

This command has the following syntax:

`fprint` *c_format_string*\ [*value1 value2 .. valuen*]

Formats and prints data to the standard output. This function is an enhancement to the Korn Shell. It supports single character format specifiers such as %d, %u, %x, 0x, %o and [[. It does not support more complex, multi-character format specifiers (such as those that specify field width and precision).

Figure 3-3
Registers accessible to the `get` command

burst_cnt_lsw	burst_cnt_msw	clk_cmd_shdw
clk_cmd_stat	clk_freq_ctl1	clk_freq_ctl2
clk_freq_ctl3	cmd_ena1	cmd_ena2
dis_clk1	dis_clk2	err_loc
hard_err1	hard_err2	hard_err_mask1
hard_err_mask2	hw_hung	init_burst_count_lsw
init_burst_count_msw	init_uart_count_lsw	init_uart_count_msw
io_addr	key_switch	mach_serial_number
main_mem_addr	mem_test_even	mem_test_odd
ncu_addr_loop_back	ncu_data_loop_back	ncu_err_log
ncu_int_ena	ncu_int_stat	ncu_misc_test_cntl
ncu_xfer_cnt	nmb_cntl_stat	nwi_data_loop_back
osc_freq	read_rbe_count	run_ena1
run_ena2	phase_mon	scalar_halt
scan_mem_err_log	scn_cmd_stat	scn_cnt
scn_cntl_ena1	scn_cntl_ena2	scn_comp_buf
scn_in_buf	scn_mask_buf	scn_out_buf
soft_log_ccu_cntl	swip_bsrc_lsb	swip_bsrc_msb
swip_force_int	swip_force_par_err	swip_fpga_reset
swip_int_ena	swip_int_level_ctl	swip_int_stat
swip_master_ena	swip_misc	swip_prog0
swip_prog1	swip_prog2	swip_prog3
swis_bsrc_msb	swis_force_int	swis_force_par_err
swis_fpga_reset	swis_int_level_ctl	swis_int_ena
swis_int_stat	swis_master_ena	swis_misc
swis_prog0	swis_prog1	sys_int_vec
uart_aux_cntl	uart_clk_sel	uart_cmd
uart_cnt_lsw	uart_cnt_msw	uart_rx_hld
uart_input_chng	uart_input_port	uart_int_mask
uart_int_stat	uart_mode1	uart_mode2
uart_output_cfg	uart_start_cnt	uart_stat
uart_stop_cnt	uart_tx_hld	

3.3.3.4 help

This command has the following syntax:

help [*command*]

Displays data from the diagnostic shell help file. If command is specified, displays fields in the file that match commands in much the same way as `man -k` does. For example, to get general register information for the NWI, enter:

help NWI

For information about a specific register, enter:

help *register name*

3.3.3.5 pause

This command has the following syntax:

pause

Pauses execution of the current diagnostic shell script, issues a prompt, and waits for you to press RETURN before continuing.

3.3.3.6 put

This command has the following syntax:

put *what*[,*count*] *value* [*what*[,*count*] *value* . . .]

The put command is generic and accesses all parts of the system to which data may be written. This includes SWIP registers, SWIS registers, NCU registers, SPU memory, and scan rings. It is also possible, but not recommended, to access CONVEX main memory through this command. The <*what*> argument specifies the item to be modified.

put sets the contents of the specified register, location in system memory, or scan ring to the indicated <*value*>. If <*count*> is specified, it sets consecutive registers (elements for vector registers) or memory locations (based on the size of default_format to value). Element must only be specified when setting the contents of a vector register.

The <*what*> argument indicates the item to be accessed, the size of that item, and in some instances, the specific element that is to be accessed. Figure 3-4 shows some examples of put commands.

Figure 3-4
Sample put commands

```
# write the contents of a register in NWI space
put nwi_int_ena 0x0f

# write the contents of a register from NWI uart channel 4
put uart_int_mask[4] 0xa2

# write a byte to a specific offset in NWI space
put 0x3:b 0

# write a value to memory using the default format
DFMT=4    # set the default format to 4 bytes (i.e. a word)
put 0xff00f340 0xf0f0f0f0
put 0xffff:b,3 0x01          # write 3 bytes of data at address 0xffff
put 0x9630:s 56 0x9632:s 45 # write 2 short words of data at address
                              0x9630

# write the contents of a scan ring field
put ringname:fieldname 0x10203040
```

Figure 3-3 on page 3-16 provides a list of SWIS, SWIP, NCU, and NCI registers accessible through the `put` command.

Figure 3-1 on page 3-8 lists the C3800 Series scan rings. You may use the `list` command to obtain lists of scan ring field names.

3.3.3.7 `set -o xtrpause`

This command has the following syntax:

`set -o xtrpause`

Places `ksh` into single step mode and pauses after completing execution of each line of command input in the current shell. `+o xtrpause` turns this option off.

3.3.3.8 `set -o xtrace`

This command has the following syntax:

`set -o xtrace`

Expands and prints each command or subcommand to standard error prior to executing it. `+o` turns this option off.

3.4 Examples

Figures 3-5 through 3-13 contain examples of actual test scripts developed on previous systems, and examples of those same scripts as they are implemented using `dsh`.

Figure 3-5
Old iscn `if/then` construct

```
#Check to see which icache to dump
define
proc cache_ck () {
 if (16#1 == 0) then {
  !icache -c 0 -dh -e >> xXx.4 2>&1
   } else { if (16#1 == 1) then {
    !icache -c 1 -dh -e >> xXx.4 2>&1
     } else { if (16#1 == 2) then {
      !icache -c 2 -dh -e >> xXx.4 2>&1
       } else { if (16#1 == 3) then {
        !icache -c 3 -dh -e >> xXx.4 2>&1
         } else { pr "ERROR IN SCAN SCRIPT\en" }
   }
  }
 }
}
enddefine
```

Figure 3-6
`dsh if/then` construct

```
#Check to see which icache to dump
cache_ck ()
{
   if ($CPU_NUM == 0)
      then icache -c 0 -dh -e >> xXx.4 2>&1
   elif ($CPU_NUM == 1)
      then icache -c 1 -dh -e >> xXx.4 2>&1
   elif ($CPU_NUM == 2)
      then icache -c 2 -dh -e >> xXx.4 2>&1
   elif ($CPU_NUM == 3)
      then icache -c 3 -dh -e >> xXx.4 2>&1
   else print "ERROR IN SCAN SCRIPT"
   fi}
```

Figure 3-7
dsh case construct

```
#Check to see which icache to dump
cache_ck ()
{
   case $CPU_NUM in
      [0-3]) icache -c $CPU_NUM -dh -e >> xXx.4 2>&1 ;;
      () print "ERROR IN SCAN SCRIPT" ;;
   esac
}
```

Figure 3-8
Old `iscn for` construct

```
#Dump MCM Read queues
define
proc mem_state () {
     for 16#1 = 0,7 do {
     :30 = bt mcm[:901]: "MCM"
   if (:30 == 0ffffffff) then {
      pr "No MCM %1d present\en"  16#1  }
   else { p mcm[:901]:halt_disable 1
          :0 = dump_rd_q
           xbar_ptr ()
   }
  }
}
enddefine
```

Figure 3-9
`dsh for` construct

```
#Dump MCM Read queues
mem_state ()
{
   for BOARDID in 0 1 2 3 4 5 6 7
   do
      bdtype mcm[$BOARDID]: "MCM"
      if ($? -eq 0)
         then put mcm[$BOARDID]:halt_disable 1
              DUM_RD='dump_rd_q'
              xbar_pt # this is a function call
      else print "No MCM $BOARDID present"
      fi
   done
}
```

Figure 3-10
Old `iscn while` construct

```
define
proc form_pattern ()
{
    :2 = 0
    while(:2 < 8)
    {
        # the &7 makes the indexes wrap in a ring
        # even from 22 to 23
        :22 = %D300 + (:2 & 7)
        :23 = %D300 + (:3 & 7)
        # odd from 24 to 25
        :24 = :22 + %D24
        :25 = :23 + %D24
        :2 = :2 + 1
    }
}
enddefine
```

Figure 3-11
`dsh while` construct

```
form_pattern ()
{
    VAR2=0
    while (($VAR2 < 8))
    do
        # the &7 makes the indexes wrap in a ring
        # even from 22 to 23
        (( VAR22=(300 + ($VAR2 & 16#7)) ))
        (( VAR23=(400 + ($VAR3 & 16#7)) ))
        # odd from 24 to 25
        (( VAR24=($VAR22 + 24) ))
        (( VAR24=($VAR22 + 24) ))
        (( VAR2=($VAR2 + 16#1) ))
    done
}
```

Figure 3-12
Old method of including a file

```
# Note multiple functions in one file

########## included file "names" ##############
define
proc fred ()
{
   . . .
}
proc steve ()
{
   . . .
}
proc mary ()
{
   . . .
}
enddefine
########## end of included file "names" ##############

in names        # include file "names"
```

Figure 3-13
New method of including a file

```
# Note each function in a separate file

########## included file "fred" ##############
fred ()
{
   . . .
}
########## end of included file "fred" ##############
########## included file "steve" ##############
proc steve ()
{
   . . .
}
########## end of included file "steve" ##############
########## included file "mary" ##############
proc mary ()
{
   . . .
}
enddefine
########## end of included file "mary" ##############

########## file that includes multiple functions ##############
########## included file "multifunc" ##############

autoload "steve" # include
autoload "mary"  #   individual
autoload "fred"  #     functions

########## end of included file "multifunc" ##############

  \. "multifunc" # statement that actually includes multiple functions
```

User test interface

4

The user test interface operates from the C3800 system service processor. The root window of the user test interface is the `xdiag` window. To display the `xdiag` window, enter:

```
>xdiag
```

The `xdiag` window then appears on the screen. Figure 4-1 shows the `xdiag` window.

The `xdiag` window lets you select and control diagnostic tests either by mouse or through commands entered in the command entry pane of the window. Section 4.11 of this chapter gives details of test control through keyboard command entry.

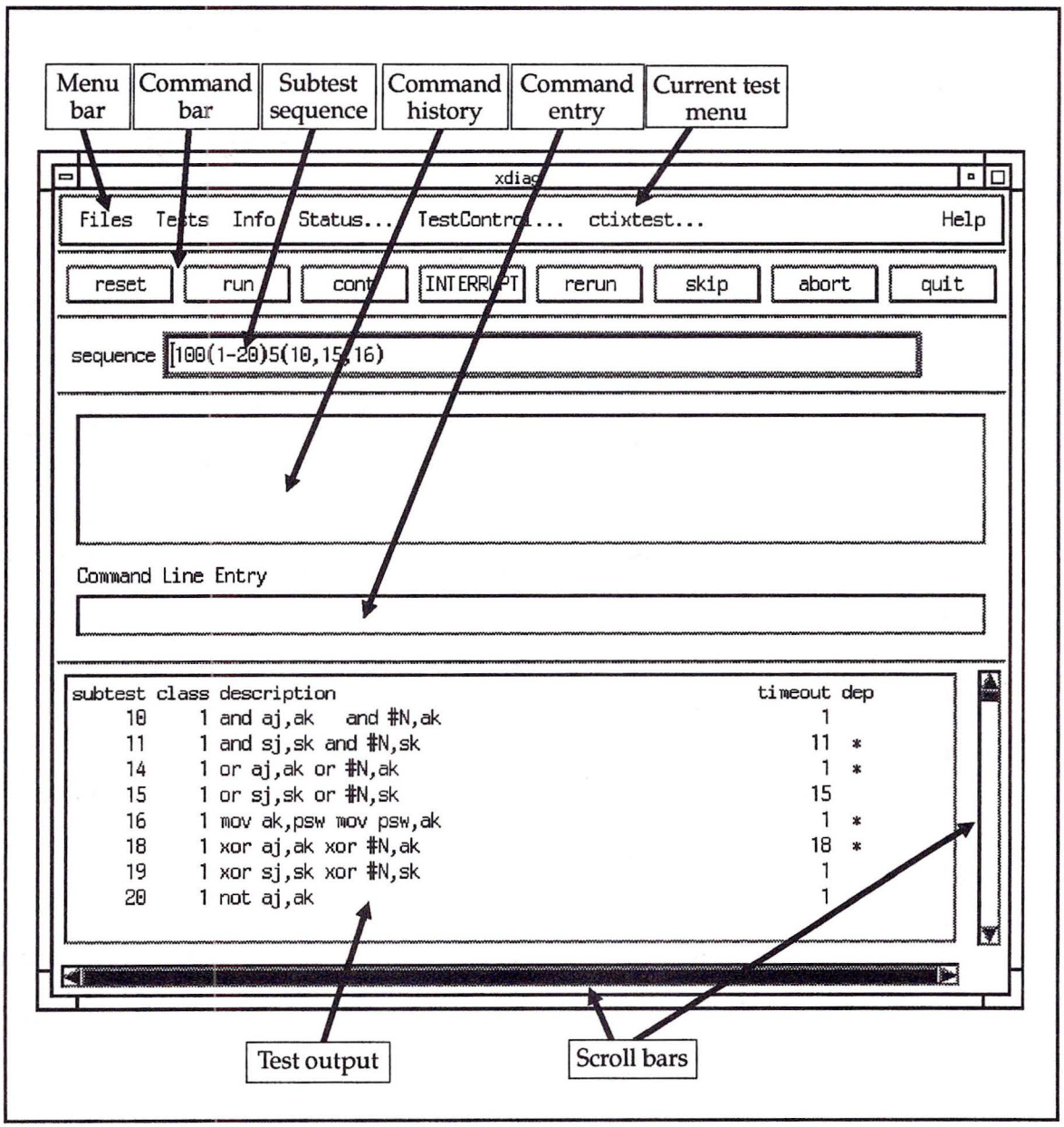
4.1 Using the mouse

This chapter uses the following terms to describe how to use the mouse:

- **Mouse cursor**—The icon that moves about the work station screen when you move the mouse. The mouse cursor may change form as you move it to different areas of the screen.
- **Select**—Use the mouse to place the mouse cursor on top of the desired item.
- **Click**—Quickly press and release the left button on the mouse.
- **Double click**—Quickly press and release the left button on the mouse twice.
- **Drag**—Place the mouse cursor on the desired item. Press and hold the left mouse key down. Move the mouse cursor to the new location. Release the mouse button.

To close an X-Window, click on the box with the minus sign in it, in the upper left corner of the window. Then click on the `CLOSE` item in the resulting pop-up box.

Figure 4-1
`xdiag` window



4.2 Using the keyboard

You can use the SPU keyboard to traverse through the xdiag window and select all window functions except the `Help` window.

- Press **TAB**, **RIGHT ARROW** or **LEFT ARROW** to travel laterally between buttons or selection fields in a window frame. Press **TAB** to move right and **SHIFT-TAB** to move left.
- Press **UP ARROW** or **DOWN ARROW** to move vertically from window frame to window frame.
- Press **F10** to move to the menu bar at the top of the window.
- If a drop-down or pop-up window appears, press **TAB** or the **ARROW** keys to move through the selection items in the window, and to move out of the window.
- Press **ENTER** to select a highlighted function.

4.3 The `xdiag` window

The `xdiag` window provides control over all diagnostic tests. Figure 4-1 shows the `xdiag` window.

The `xdiag` window consists of the following panes:

- **Menu bar**—Located at the top of the window. You can use the mouse to select a test and to set test parameters and other commands through the menu bar. Clicking on a label followed by an ellipsis (. . .) causes another window to appear.
- **Command bar**—Located beneath the menu bar. You can control the progress of the tests by clicking on these menu items.
- **Sequence box**—Located beneath the command bar. You can set the test sequence in this box. The example in the figure indicates that subtests 1 through 20 will run in sequence 100 times, and tests 10, 15, and 16 will run in sequence 5 additional times after the 100 runs of tests 1 through 20 have completed.
- **Command history display**—Located beneath the command bar. This pane displays, in the order of input, every command you made by mouse or by keyboard.
- **Command line entry**—Located beneath the command history display. You can use this pane to type commands into the `xdiag` system.

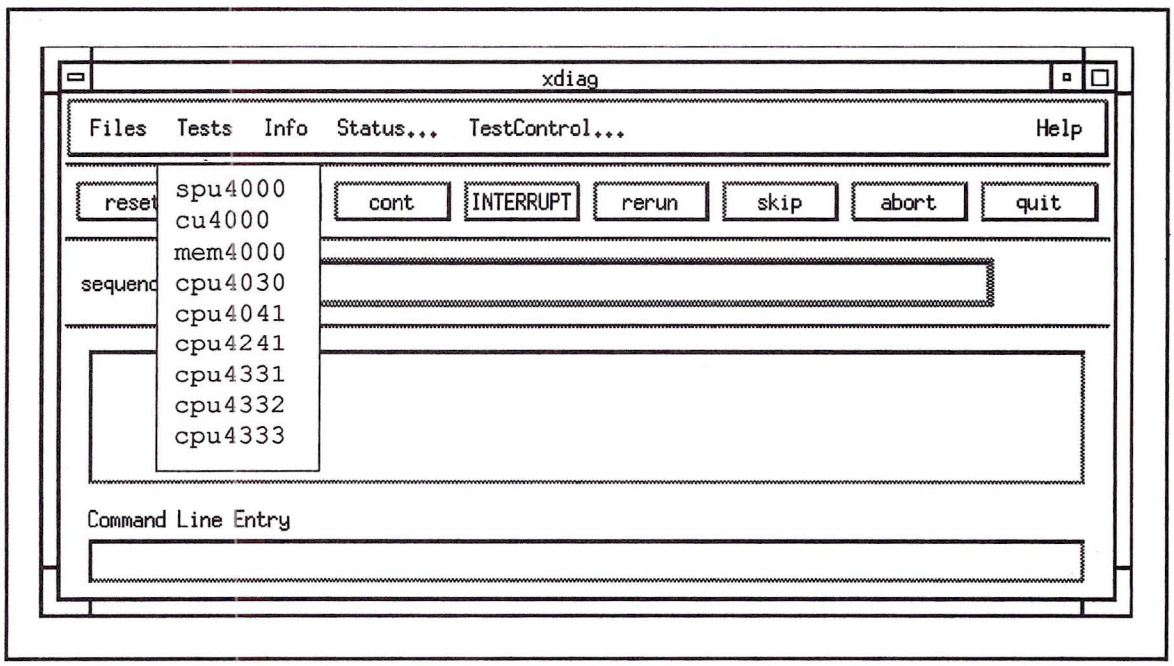
- **Test output**—Beneath the command line entry pane. Every line of test output that is not directed elsewhere appears in this window pane. You can scroll through these data by clicking on the scroll bars beneath and to the right of this pane.

4.4 Selecting a test

Click on `Tests` in the `xdiag` window menu bar. A pull-down window appears, listing the available diagnostic tests. Figure 4-2 shows the `Tests` pull-down menu.

Click on the test you wish to run. The `Tests` pull-down menu disappears and the name of the selected test appears in the `xdiag` window bar. In Figure 4-1, a development test program called ctixtest has been selected. The `ctixtest...` menu item has appeared in the menu bar for control of the test. In this case, clicking on `ctixtest...` would cause a test specific window to appear.

Figure 4-2
`Tests` pull-down menu



4.5 Running a diagnostic test with default settings

Use the following procedure if you wish to run a diagnostic test without changing the test parameters:

Step 1 Bring the `xdiag` window to the screen by entering

```
>xdiag
```

Step 2 Using the mouse, click on the `Tests` item in the menu bar shown in Figure 4-1. A pull-down menu appears listing the available tests as shown in Figure 4-2.

Step 3 Click on the test you wish to run.

Step 4 Click on the `Run` item in the command bar.

The selected test runs, using its default parameters.

4.6 Getting help

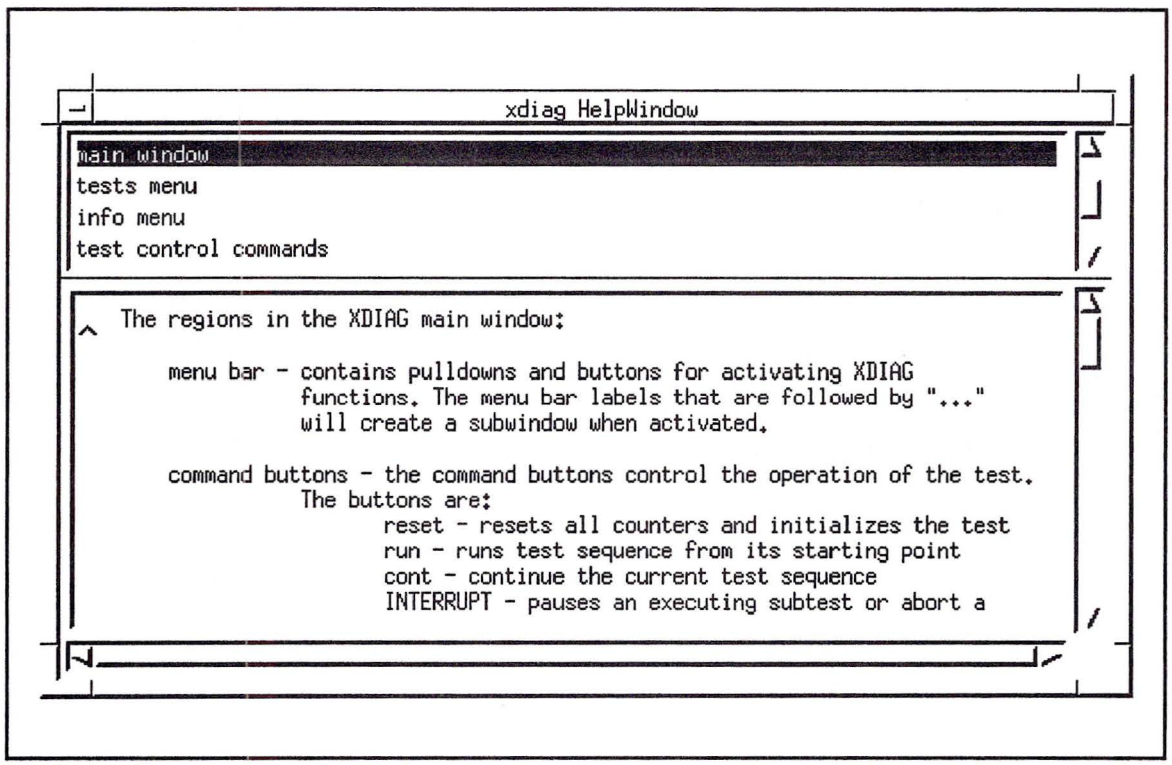
The `xdiag` environment offers the `xdiag HelpWindow` that contains information about diagnostic windows, commands, and other aspects of the diagnostic software.

To view the `xdiag HelpWindow`, click on `Help`, located at the right of the `xdiag` menu bar. Figure 4-3 shows the `Help` window.

The upper window pane in the `Help` window contains a list of subjects on which information is available. To get help, click on the subject you want in the upper window pane. The selected subject appears in reverse video and the help information appears in the lower window pane.

Click on the box with the minus in the upper left corner of the `xdiag HelpWindow` to remove the window from the display.

Figure 4-3
The `xdiag HelpWindow`

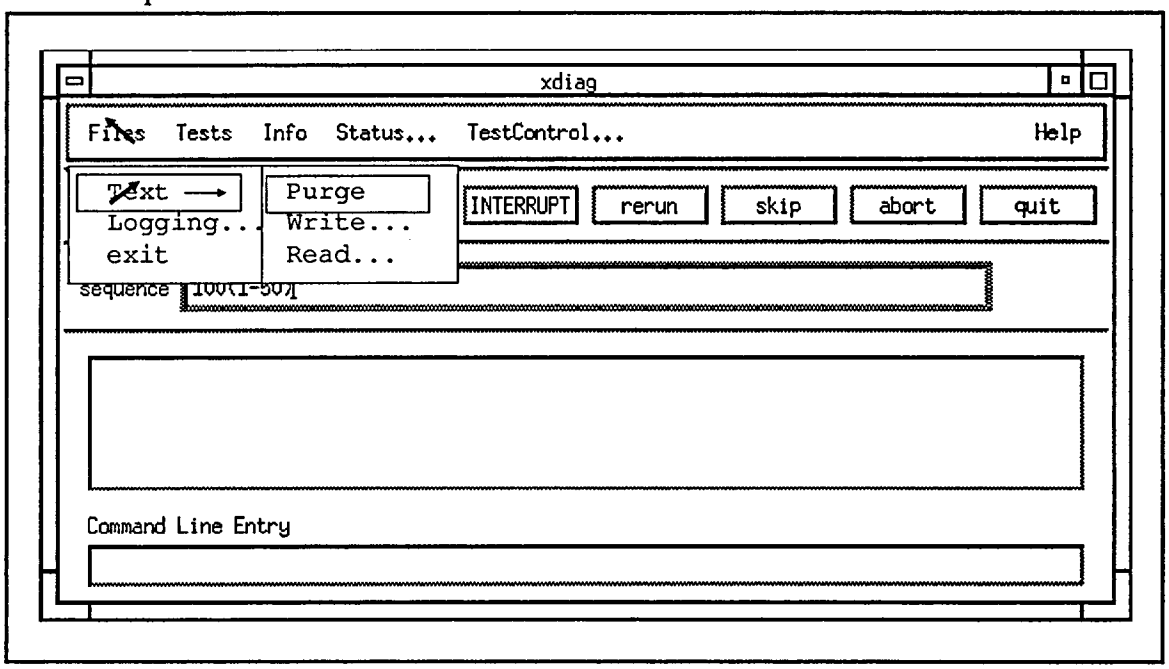


4.7 File control and exiting

To access diagnostic files, click on `Files` in the `xdiag` window menu bar. A pull-down menu appears, as shown in Figure 4-4. Clicking on `Text` in the pull-down menu causes a second pull-down menu to appear, also shown in Figure 4-4. You can use these pull-down menus to perform any of the following functions:

- Specify a file for automatic logging of all data appearing in the test output pane of the `xdiag` window.
- Purge the test output pane of all data.
- Write the contents of the test output pane to a specified file.
- Read a file into the test output pane of the `xdiag` window.
- Exit the `xdiag` environment.

Figure 4-4
The `Files` pull-down menus



4.7.1 Specifying a file

Use the method described in Section 4.7 to select the desired file operation.

The `Logging`, `Write`, and `Read` selections shown in Figure 4-4 cause other pop-up windows to appear. These windows have a similar layout. Figure 4-5 shows the `xdiag Logging` window.

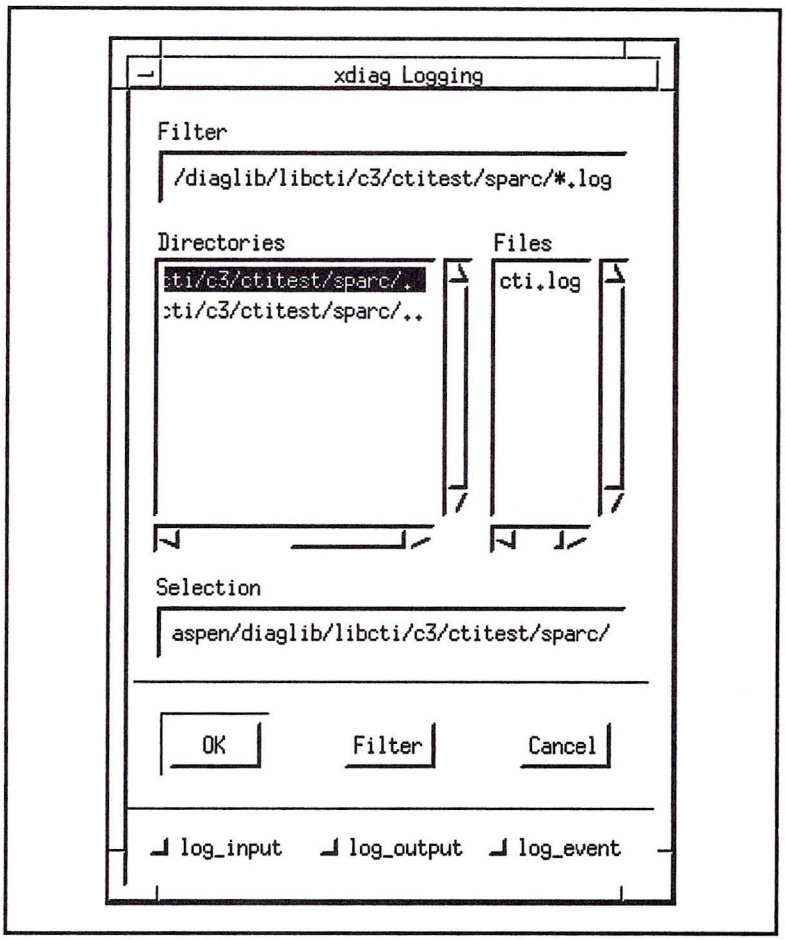


Figure 4-5
The `xdiag Logging` window

- The `Filter` pane shows the current directory path and types of files displayed.
- The `Directories` pane shows the parent directory link and the available child directory paths from the current directory. It may not show the full width of the directory display. You may widen the window or use the pane scroll bar to display more of the directory paths.

- The `Files` pane shows the files available in the current directory that match the specified filter.
- The `Selection` pane shows the currently selected file, with its path.

Use the following method to select a file for reading, writing, or logging:

Step 1 Move to the desired directory. Repeat either of the following two actions until the appropriate path is displayed:

- To move to a child directory of the current directory, click on a child directory name in the `Directories` pane.
- To move to the parent directory of the current directory, click on the parent directory link (the `..` directory) in the `Directories` pane.

The selected directory is highlighted in the `Directories` pane. The selected path appears in the `Filter` pane. If the move is to the parent directory, the `Selection` pane clears. If the move is to a child directory, the `Selection` pane displays the selected directory path.

Step 2 Click on the `Filter` button at the bottom of the window.

A highlight appears around the `Filter` button. The selected directory becomes the current directory. The parent directory link and the child directories of this new current directory appear in the `Directories` pane. A list of files in the current directory appears in the `Files` pane.

The parent directory becomes the current directory. The child directories of this new current directory appear in the `Directories` pane.

Step 3 Click on the desired file. The file becomes highlighted.

In the `xdiag Logging` and `xdiag TextWrite` windows, another step may be necessary before invoking the file. Refer to Section 4.7.2 and Section 4.7.4 of this document for more information.

To invoke the operation, click on the `OK` button.

To abort the operation, click on the `Cancel` button.

4.7.2 Logging test data into a file

Use the method described in Section 4.7 to select the `Logging` option.

Click on `Logging` in the `Files` pull-down menu. A dialog window appears, enabling you to select log files for test data. Figure 4-5, page 4-8, shows the `xdiag Logging` dialog window.

Use the method described in Section 4.7.1 to select the log file, or click on the `Selection` pane and enter the name of the file from the keyboard. The log file must have the extension `.log`.

All transactions between `xdiag` and the diagnostic test is written to the selected log file.

The three buttons at the bottom of the `Logging` dialog window enable you to select the types of data to be logged:

- Click on the `log_input` button to log all input to the selected test.
- Click on the `log_output` button to log all test output.
- The `log_event` button currently has no effect.

A button shows a distinct color when it is selected. Any or all buttons may be selected or deselected. The default condition is with all buttons selected. Deselect all three buttons to deactivate the logger.

Click on the `OK` button to activate or deactivate the logger, or click on the `Cancel` button to abort the selection. The `xdiag Logging` window disappears when you click on either of these two buttons.

4.7.3 Reading a file

Use the method described in Section 4.7 to select the `Read` option.

Click on `Read` in the second menu. The `xdiag TextRead` dialog window appears, allowing you to read a file into the test output pane of the `xdiag` window. Figure 4-6 shows the `xdiag TextRead` dialog window.

Use the method described in Section 4.7.1 to select the file to read, or click on the `Selection` pane and enter the name of the file from the keyboard.

Click on the `OK` button to copy the file into the `xdiag` window, or click on the `Cancel` button to abort the selection. The `xdiag TextRead` window disappears when you click on either of these two buttons.

4.7.4 Writing a file

Use the method described in Section 4.7 to select the `Write` option.

Click on `Write` in the second menu. The `xdiag TextWrite` dialog window appears, allowing you to write the contents of the `xdiag` window test output pane into a file. The `xdiag TextWrite` dialog window has a structure similar to the `xdiag TextRead` dialog window as shown in Figure 4-6.

Use the method described in Section 4.7.1 to select the file to write to, or click on the `Selection` pane and enter the name of the file from the keyboard.

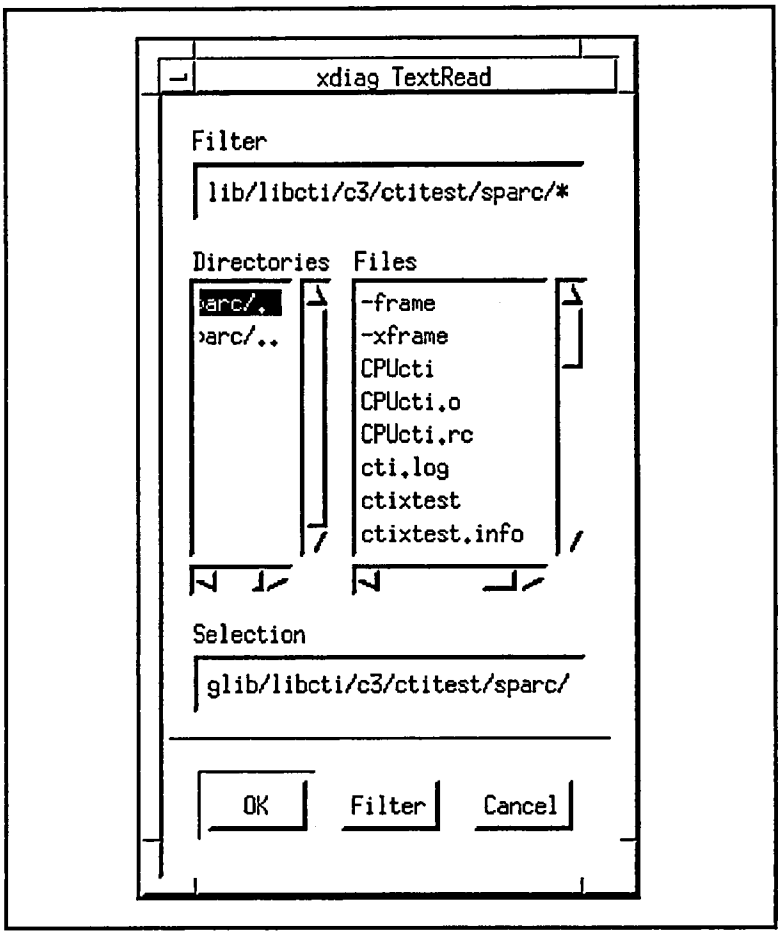
The `xdiag TextWrite` dialog window has two buttons near its top:

- `append to file`
- `overwrite file`

Click on the type of file write you desire.

Click on the `OK` button to write the contents of the test output window to the specified file, or click on the `Cancel` button to abort the action. The `xdiag TextWrite` window disappears when you click on either of these two buttons.

Figure 4-6
The `xdiag TextRead` window



4.7.5 Purging the `xdiag` window test output

Use the method described in Section 4.7 to select the `Purge` option.

Clicking on `Purge` in the second menu clears the `xdiag` window test output pane.

4.7.6 Exiting the diagnostic environment

Clicking on `exit` in the `Files` pull-down menu exits you from the diagnostic environment.

4.8 Monitoring test progress

The xdiag Status window lets you watch the progress of diagnostic tests as they run. To access the xdiag Status window, click Status on the xdiag menu bar. Figure 4-7 shows the xdiag Status window.

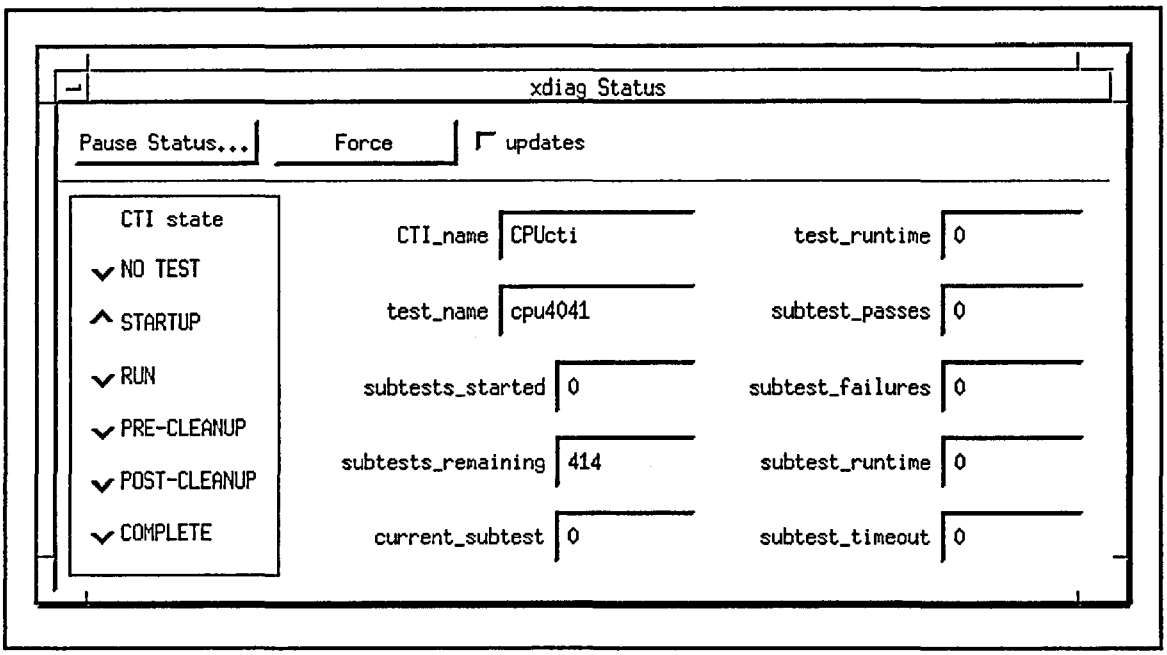
The parameters listed in the Status window are mostly self-explanatory. Some comments follow:

- You cannot operate the buttons in the CTI state pane. The dark button indicates the current status of the current subtest.
- The test_runtime, subtest_runtime, and subtest_timeout items in the right-hand column of the window are in units of seconds.
- The run-time parameters increment.

The subtest_timeout parameter indicates the total seconds that the current subtest will run if it does not complete beforehand.

Click on the box with the minus in the upper left corner of the xdiag Status window to remove the window from the display.

Figure 4-7
The `xdiag Status` window



4.9 Controlling test information output

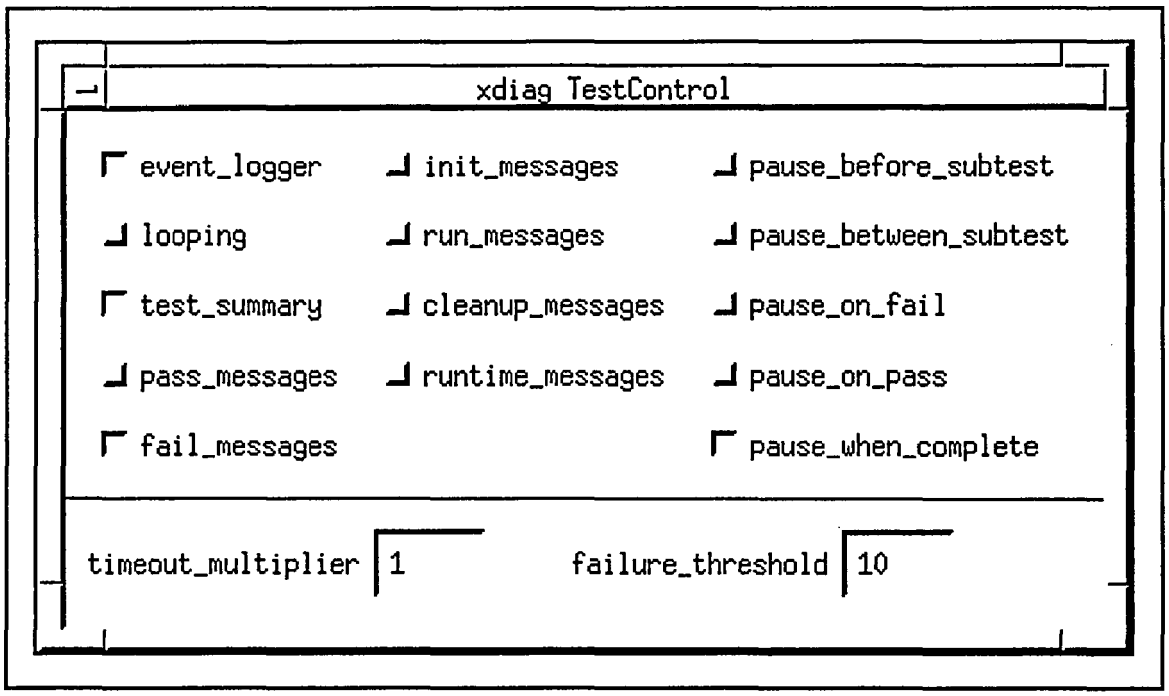
The xdiag TestControl window lets you control the information output of diagnostic tests, and to cause automatic pauses in the test sequence. To access the xdiag TestControl window, click Test Control on the xdiag menu bar. Figure 4-8 shows the xdiag TestControl window.

The xdiag TestControl window contains a list of options relating to test progress and information output. To activate these options, click on the boxes next to the desired options. An option is selected when the corresponding button is dark. For example, if the pass_messages button is dark, a message appears in the test output pane of the xdiag window every time a subtest runs with less errors than the failure_threshold parameter allows.

The parameters listed in the xdiag TestControl window are mostly self-explanatory. Some comments follow:

- The looping parameter, when active, causes the current subtest to loop continuously until you deselect the looping parameter.
- The timeout_multiplier parameter multiplies the subtest time-out default values by the value of the parameter. You can set the timeout_multiplier value by clicking the mouse cursor in the timeout_multiplier pane and entering the desired value from the keyboard.
- The failure_threshold parameter indicates the number of failures a subtest can accumulate before the test is labeled a failure. You can set the failure_threshold value by clicking the mouse cursor in the failure_threshold pane and entering the desired value from the keyboard.
- Click on the box with the minus in the upper left corner of the xdiag TestControl window to remove the window from the display.

Figure 4-8
The `xdiag TestControl` window



4.10 The xdiag Pause Status window

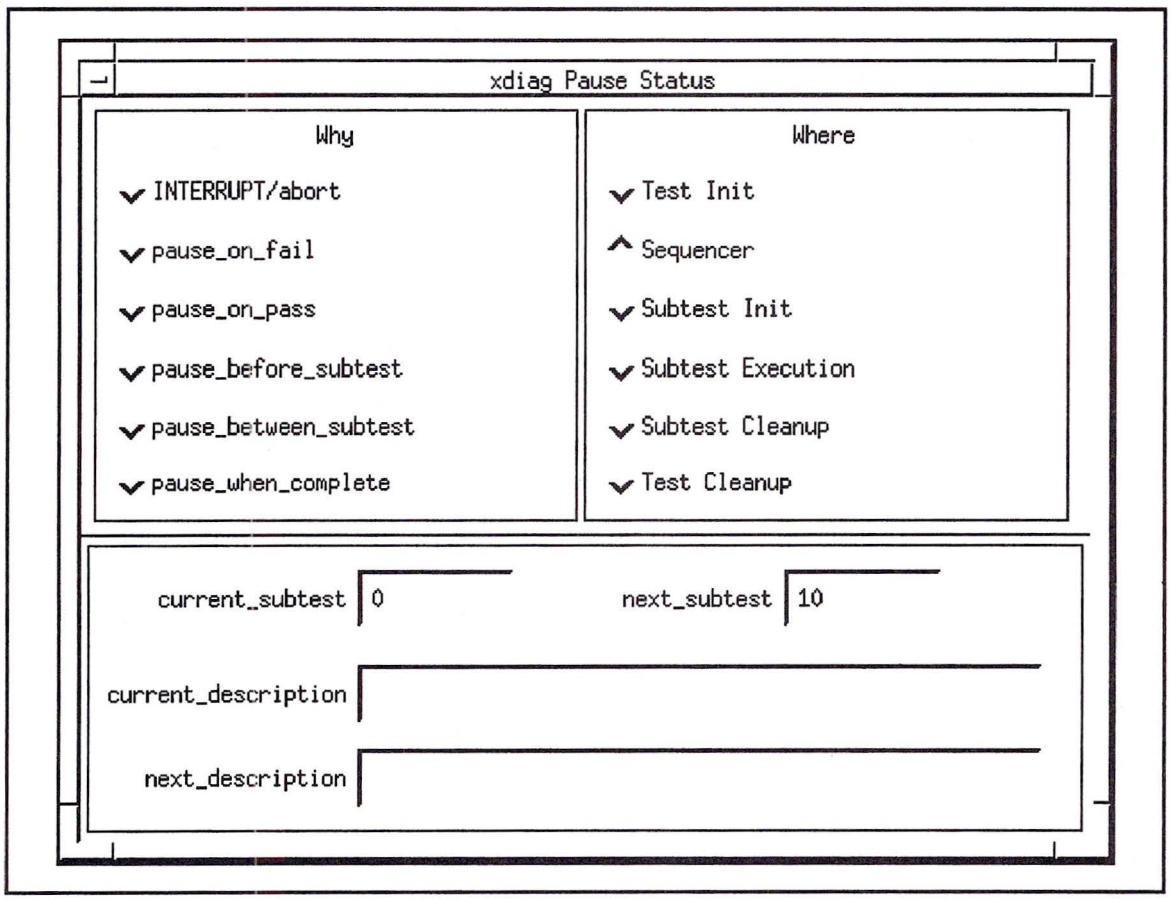
When a pause occurs in a subtest sequence, the `xdiag Pause Status` window appears on the screen. Figure 4-9 shows the `xdiag Pause Status` window.

Dark buttons appear in the `Why` and `Where` panes indicate the reason for the pause. You cannot operate the buttons in the `Why` and `Where` panes.

The `current_subtest` and `next_subtest` panes give the numbers of the corresponding subtests.

The `current_description` describes the subtest listed in the `current_subtest` pane. The `next_description` describes the subtest listed in the `next_subtest` pane.

Figure 4-9
The `xdiag Pause Status` window



4.11 Entering commands from the keyboard

You can completely control a diagnostic test sequence from the `xdiag` window by typing commands and altering test parameters in the command line entry window pane, as shown in Figure 4-10. Use the following procedure:

Step 1 Click in the command line entry window pane. A text entry cursor appears in the pane.

Step 2 Enter a command or a parameter and new value into the pane. The command or parameter appears in the pane. A typical command entry may be:

```
run
```

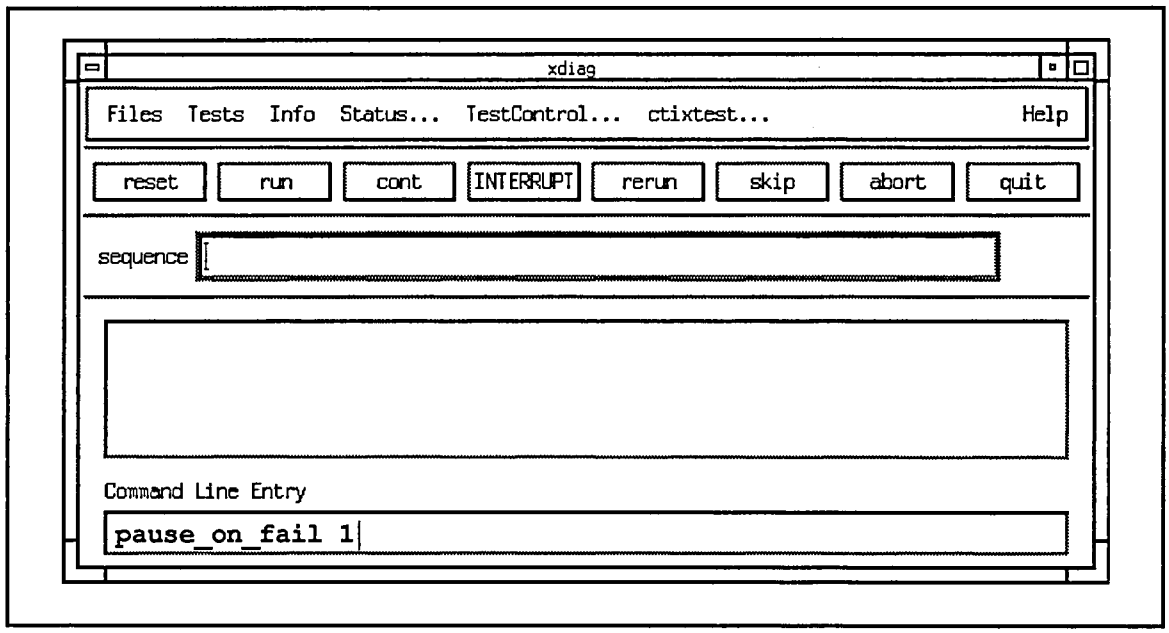
A typical parameter change may be:

```
pause_on_fail 1
```

Refer to section 4.12 for lists of commands and test parameters.

Step 3 Press RETURN. The command appears in the command history pane and executes.

Figure 4-10
Entering commands from the keyboard

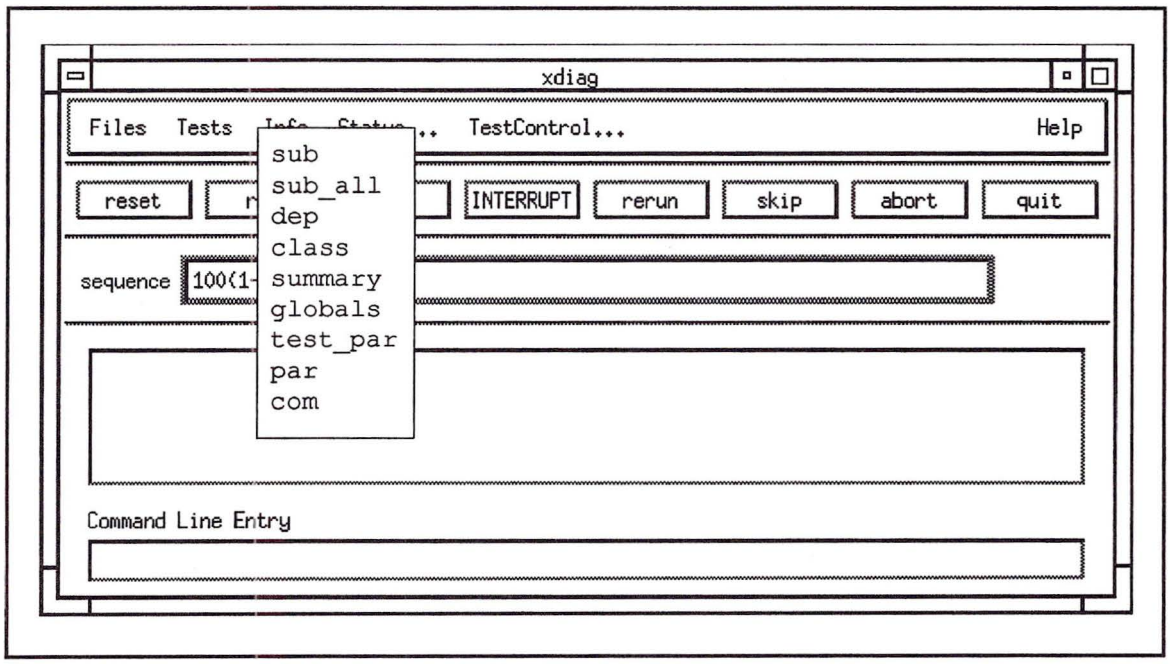


4.12 Getting test information

The most convenient way of gaining information on a test may be through the `xdiag Status`, `xdiag TestControl`, and `xdiag Pause Status` windows. An alternative way is through presentation of the information on the `xdiag` window test output pane.

Click on `Info` in the `xdiag` window menu bar. A pull-down window appears, listing the test information options. Figure 4-11 shows the `info` pull-down menu. Click on the desired information option.

Figure 4-11
The `Info` pull-down menu



The following subsections describe each of these options.

4.12.1 Subtest list (sub)

Clicking on sub causes the current list of allowed subtests to appear in the xdiag test output window, as shown in Figure 4-12.

Figure 4-12
List of current subtests

```
subtest class description                                    timeout dep
     10     1 and aj,ak   and #N,ak                              1
     11     1 and sj,sk and #N,sk                               11  *
     14     1 or aj,ak or #N,ak                                  1  *
     15     1 or sj,sk or #N,sk                                 15
     16     1 mov ak,psw mov psw,ak                              1  *
     18     1 xor aj,ak xor #N,ak                               18  *
     19     1 xor sj,sk xor #N,sk                                1
     20     1 not aj,ak                                          1  *
```

The subtest list headings are as follows:

- subtest—Number of the subtest.
- class—Class of the subtest.
- description—Function of the subtest.
- timeout—Number of seconds within which the subtest must complete, otherwise it aborts.
- dep—Subtest dependency indication; if a subtest can run only under certain conditions, an asterisk appears. If a dependency disallows a subtest, the subtest does not appear on the sub list. See Section 4.12.3 for information on subtest dependencies.

4.12.2 sub_all

Clicking on sub_all causes a list of all subtests available under the designated test, whether currently allowed or not, to appear in the xdiag test output window. The display is similar to that shown in Figure 4-12.

4.12.3 Test dependencies (dep)

Clicking on dep causes the list of dependencies pertaining to the current test to appear in the xdiag test output window, as shown in Figure 4-13.

Figure 4-13
List of current test dependencies

```
subtest dependencies
     11 -test_long 10
     14 -test_long 1
     16 -test_long 10,3,5,8
     18 -test_short 3 -test_long 10
     20 -test_long 5
```

The subtest list headings are:

- subtest—Number of the subtest.
- dependencies—Conditions that would disallow running of a subtest. The terms, -test_long and -test_short are dummy dependencies used in development. In the list of Figure 4-13, subtest 11 would not run if the -test_long dependency had a value of 10. Subtest 18 would not run if -test_short had a value of 3, and/or -test_long had a value of 10. The dep list can contain a series of dependencies, as shown for subtest 16; or it can contain a hyphenated range of dependencies.

4.12.4 class

Clicking on class causes a list of current subtests segregated by class to appear in the xdiag test output window.

4.12.5 Test progress (summary)

Clicking on summary causes a summary of current test progress to appear in the xdiag test output window, similar to that shown in Figure 4-14.

Figure 4-14
Summary of current test progress

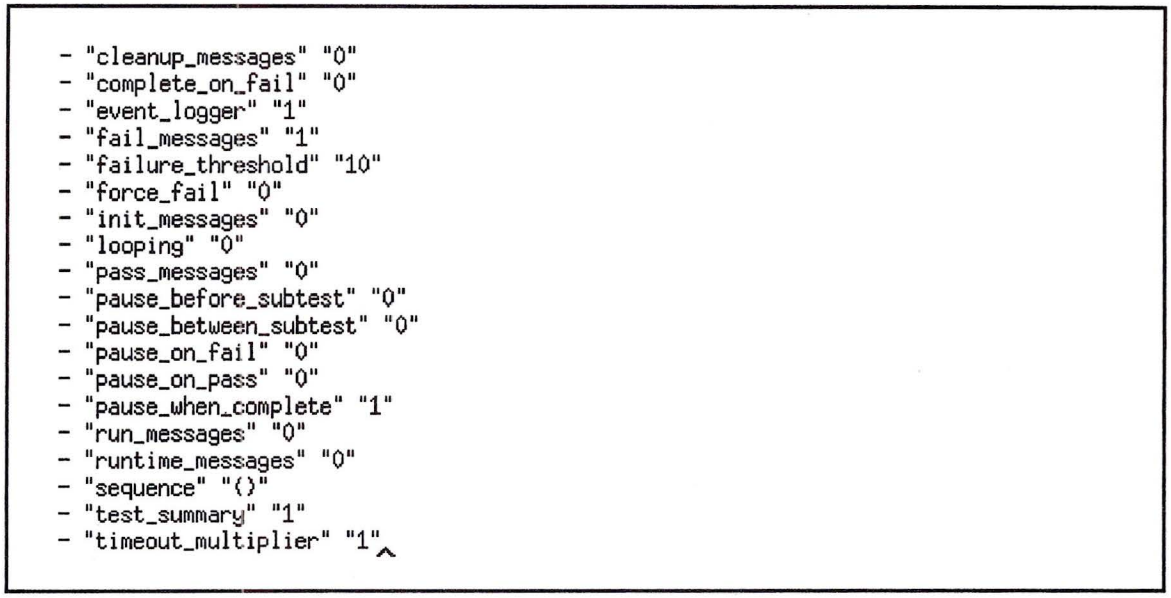
```
- "current_subtest" "0"
- "next_subtest" "0"
- "subtest_failures" "0"
- "subtest_passes" "8"
- "subtest_runtime" "0"
- "subtest_timeout" "0"
- "subtests_remaining" "0"
- "subtests_started" "8"
- "test_runtime" "1249"
- "where_paused" "2"
- "why_paused" "8"^
```

Section 4.12.8 of this document gives definitions of the list items.

4.12.6 Global parameters (`globals`)

Clicking on `globals` causes a list of test parameters to appear that apply to all tests. Figure 4-15 shows a typical list of global test parameters.

Figure 4-15
Global test parameters



Section 4.12.8 of this document gives definitions of the list items.

4.12.7 Test-specific parameters (test_par)

Clicking on test_par causes a list of test parameters to appear that are specific to the current test. Figure 4-16 shows the test parameter list for diagnostic test spu4000 as an example.

Figure 4-16
Test-specific parameters

```
- "cu_selected" "0"
- "ia_selected" "0,0,0,0,0,0,0,0,0"
- "loop_count" "1"
- "mb_selected" "0,0,0,0,0,0,0,0"
- "pattern_number" "0x00000000"
- "ring_number" "40"
- "sp_selected" "0,0,0,0,0,0,0,0"
- "vp_selected" "0,0,0,0,0,0,0,0"
- "xbar_selected" "1"
```

Each test has its own test-specific parameters. Refer to chapters on individual tests for lists of test-specific parameters.

4.12.8 Parameters and definitions (par)

Clicking on par causes a list of all test parameters to appear, as shown in Figure 4-17.

The parameter definitions follow:

- CTI_name—Name of test.
- CTI_path—Full directory path to the test file.
- CTI_pathname—Path to the test.
- CUIexists—Set automatically by xdiag. Computer user interface is active if 1; inactive if 0.
- alarm—When updates are enabled, indicates the number of seconds between display updates.
- cleanup_messages—Displays cleanup messages if 1; inactive if 0.
- complete_on_fail—If 1, causes the test sequence to terminate if the number of failures in a subtest exceeds the allowed number; inactive if 0.
- confirm_abort—Asks for confirmation of a manual abort if 1; inactive if 0.
- cu_selected—This parameter is specific to spu4000.
- current_description—In pause state, contains a string describing the function of the current subtest.
- current_subtest—Gives the ID number of the current subtest.
- event_logger—Logs failures to the system event logger if 1; inactive if 0.
- fail_messages—Displays messages describing test failures and time-outs if 1; inactive if 0.
- failure_threshold—Displays the total number of failures required to abort a subtest if 1; inactive if 0.
- force_fail—Forces a failure if 1; inactive if 0.
- ia_selected—This parameter is specific to spu4000.
- init_messages—Displays initialization messages if 1; inactive if 0.
- log_event—No current function.

Figure 4-17
Test parameters

```
- "CTI_name" "spu4000"
- "CTI_path" "./"
- "CTI_pathname" "spu4000"
- "CUILexists" "0"
- "alarm" "5"
- "cleanup_messages" "0"
- "complete_on_fail" "0"
- "confirm_abort" "0"
- "cu_selected" "0"
- "current_description" ""
- "current_subtest" "0"
- "event_logger" "1"
- "fail_messages" "1"
- "failure_threshold" "10"
- "force_fail" "0"
- "ia_selected" "0,0,0,0,0,0,0,0,0,0"
- "init_messages" "0"
- "log_event" "0"
- "log_input" "0"
- "log_output" "0"
- "loop_count" "1"
- "looping" "0"
- "mb_selected" "0,0,0,0,0,0,0,0,0"
- "next_description" ""
- "next_subtest" "410"
- "pass_messages" "0"
- "pattern_number" "0x00000000"
- "pause_before_subtest" "0"
- "pause_between_subtest" "0"
- "pause_on_fail" "0"
- "pause_on_pass" "0"
- "pause_when_complete" "1"
- "prompt" "> "
- "ring_number" "40"
- "run_messages" "1"
- "runtime_messages" "0"
- "sequence" "(410-421),(510-511),(610-614),(710-713),(810)"
- "sp_selected" "0,0,0,0,0,0,0,0,0"
- "state" "1"
- "subtest_failures" "0"
- "subtest_passes" "0"
- "subtest_runtime" "0"
- "subtest_timeout" "0"
- "subtests_remaining" "23"
- "subtests_started" "0"
- "test_runtime" "0"
- "test_summary" "1"
- "timeout_multiplier" "1"
- "updates" "0"
- "vp_selected" "0,0,0,0,0,0,0,0,0"
- "where_paused" "0"
- "why_paused" "0"
- "xbar_selected" "1"
```

- `log_input`—Writes to the log file all messages passing from the user interface to the CONVEX test interface if 1; inactive if 0.
- `log_output`—Writes to the log file all messages passing from the — to the user interface if 1; inactive if 0.
- `looping`—Causes subtest to repeat indefinitely if 1; setting to 0 ends the repeating.
- `loop_count`—This parameter is specific to `spu4000`.
- `mb_selected`—This parameter is specific to `spu4000`.
- `next_description`—Gives a description of the function of the next subtest to run.
- `next_subtest`—Gives the ID number of the next subtest.
- `pass_messages`—Causes a message to be displayed when a subtest passes if 1; no messages displayed if 0.
- `pattern_number`—This parameter is specific to `spu4000`.
- `pause_before_subtest`—Causes the test sequence to pause before the beginning of each new subtest if 1; inactive if 0.
- `pause_between_subtest`—Causes the test sequence to pause between subtests if 1; inactive if 0.
- `pause_on_fail`—If 1, causes the test sequence to pause if the number of failures in a subtest exceeds the allowed number; inactive if 0.
- `pause_on_pass`—Causes the test sequence to pause when a subtest passes if 1; inactive if 0.
- `pause_when_complete`—Causes a pause on completion of the test sequence if 1; exits on completion if 0.
- `prompt`—Specifies a prompt string that appends to the state prompt.
- `ring_number`—This parameter is specific to `spu4000`.
- `run_messages`—Causes subtest start times to be displayed if 1; inactive if 0.
- `runtime_messages`—Causes subtest duration times to be displayed if 1; inactive if 0.

- `sequence`—An alphanumeric string specifying the test sequence. The sequence executes from left to right in the string. The following syntax is supported:
  - Several entries, separated by commas.
  - Individual subtest numbers: `100,210,110,120,100`
  - Ascending and descending ranges of subtest numbers: `200-290,100-140,350-310`
  - Multiple passes, indicated by parentheses: `8(210,100,200-290)`
  - Parentheses may be nested up to 10 deep.
  - Individual subtest classes: `c3,c3,c3,c3`

  Example:

  ```
  3(100,210,110,8(c3,290-220,c3)),100-180
  ```

- `sp_selected`—This parameter is specific to `spu4000`.
- `state`—Shows the state of the test via the prompt: `STARTUP>, RUN>, PRE_CLEANUP>, POST_CLEANUP>, COMPLETE>`
- `subtest_failures`—Tells how many subtest failures have occurred in the current test.
- `subtest_passes`—Tells how many subtests have run and not failed in the current test.
- `subtest_runtime`—Gives the runtime of the current subtest in seconds.
- `subtest_timeout`—Gives the total number of seconds allowed for the subtest.
- `subtests_remaining`—Gives the number of subtests specified and not yet run.
- `subtests_started`—Gives the total number of subtests passed, failed, and current, if a pause is in effect.
- `test_runtime`—Gives the total runtime for the test, in seconds.
- `test_summary`—Displays a summary of test results if 1; inactive if 0.

- timeout_multiplier—A number that is multiplied by the default subtest time-out values to specify the actual time-out in seconds.
- updates—Status window receives continual updates if 1; inactive if 0.
- vp_selected—This parameter is specific to spu4000.
- where_paused—Gives an index number for the place the subtest was paused:
  - INTERRUPT/abort 9
  - pause_on_fail 2
  - pause_on_pass 6
  - pause_before_subtest 4
  - pause_between_subtest 5
  - pause_when_complete 8
- why_paused—Gives an index number for the reason the subtest was paused:
  - Test Init 1
  - Sequencer 2
  - Subtest Init 3
  - Subtest Execution 4
  - Subtest Cleanup 5
  - Test Cleanup 6
- xbar_selected—This parameter is specific to spu4000.

4.12.9 Commands and definitions (com)

Clicking on com causes a list of test commands and their definitions to appear. Figure 4-18 shows this list. xdiag uses these commands to communicate with the test.

All useful commands are available through buttons, boxes, and menus in the xdiag windows. You can also enter these commands manually through the xdiag window Command Line Entry pane.

The definition for the `display_scalar_registers` command is too long to fit on the display. The full definition follows:

```
"display_scalar_registers" "This prints
some lines of text that would get garbled -
portions of old lines were reprinted, and new
lines would not be in the right places."
```

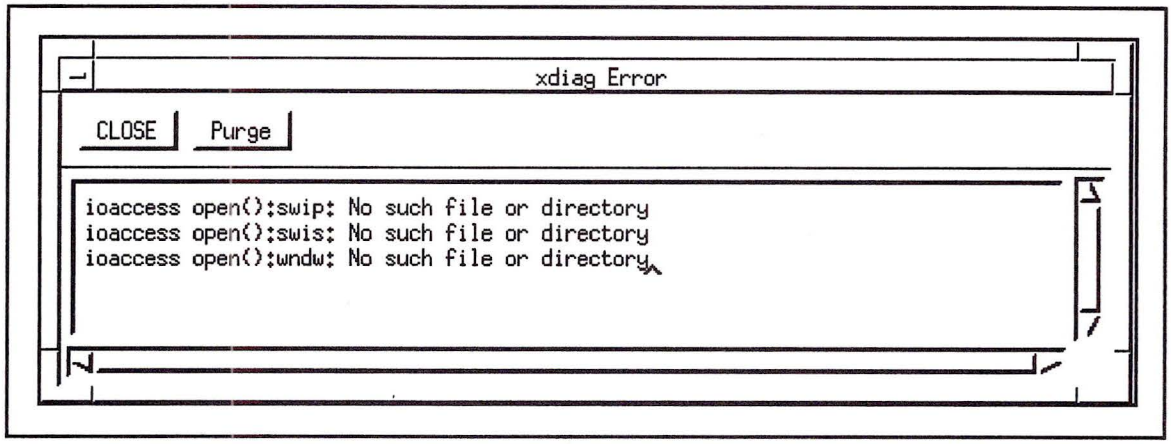
Figure 4-18
Commands and definitions

```
- "!" "passes a quoted string to the shell"
- "abort" "aborts a subtest if in "interrupt" or "run" state,
      otherwise aborts the test altogether"
- "base" "sets the default output radix - 2, 8, 10, or 16"
- "changed" "[filename] print the changed parameters and their values in order changed"
- "class" "[class]
      omit class to see all classes
      supply class to see info on that class and all of its subtests"
- "com" "[name] show the help string for a command or parameter"
- "cont" "runs the subtest sequencer"
- "dep" "[first-last] show the subtests, omit the range for all"
- "display_scalar_registers" "This prints some lines of text that would get garbled - po
- "exit" "cleanup, then exit"
- "get_par" "read in a set of parameters from the supplied filename"
- "globals" "show parameters that have the CTI_GLOBALS flag set"
- "help" "tells how to find information"
- "log_file" "specifies a file for message logging"
- "map" "map the specified window to the ctix display"
- "par" "[par_name] show the parameters and their values"
- "quit" "cleanup, then exit"
- "rerun" "runs the sequencer starting with previous subtest"
- "reset" "resets the CTI, sequencer, counters, & status "
- "run" "specify a subtest sequence and restart sequencer"
- "save" "[filename] save changed parameter values to the file."
- "save_all" "[filename] save all parameter values to the file."
- "skip" "skip n subtests in the sequence"
- "status" "request a status update"
- "sub" "[first-last] show the subtests in the run list, omit the range for all"
- "sub_all" "[first-last] show the known subtests, omit the range for all"
- "summary" "write parameters that have the CTI_SUMMARY flag set."
- "test_par" "show parameters that are specific to the test"^
```

4.12.10 The xdiag Error window

If an error occurs in testing, the xdiag Error window may appear, with a message indicating the type of error. Figure 4-19 shows the xdiag Error window.

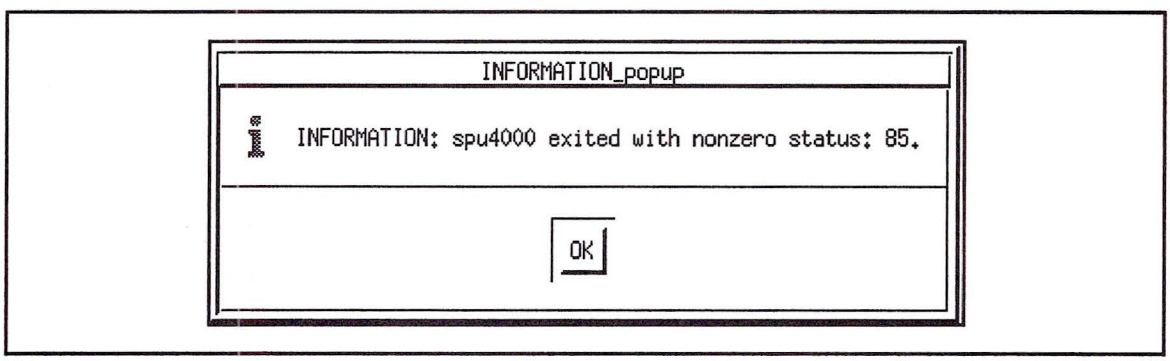
Figure 4-19
xdiag Error window



4.12.11 The INFORMATION_popup window

If a test improperly completes, the INFORMATION_popup window may appear with a message. Figure 4-20 shows the INFORMATION_popup window.

Figure 4-20
INFORMATION_popup window



4.13 When the X-Window environment is not available

The `xdiag` user interface cannot currently run remotely through a modem connection. In these circumstances you can perform CTI based tests through a command line interface that provides complete `xdiag` test control.

4.13.1 Remote login to a SPU as diaguser

Use the following procedure to log remotely into a SPU:

Step 1 From your own system prompt (not shown here), establish the telephone connection to the SPU. A typical command for establishing the connection is:

`tip -9600` *SPU modem phone number*

Step 2 If the connection is made, the following message appears on your terminal or workstation. Enter at the `login` prompt:

```
CONNECT baud rate
login: diaguser
Password:
```

Step 3 Enter the password. If the login is successful, several messages appear, concluding with the following messages and the system prompt:

```
starting with local interactive setup
done with local interactive setup
spu>
```

Step 4 Run the test. See the following sections of this manual for further instructions.

Step 5 When you complete your work on the SPU, enter at the `spu>` prompt:

```
spu> exit
```

Several messages appear on your screen, including this message:

```
Connection closed.
```

4.13.2 Selecting a test

You can invoke any diagnostic test from the `spu>` prompt. For example, enter:

```
spu> spu4000
STARTUP>
```

The command line prompt reflects the state of the test and is the same as the CTI state provided in the `xdiag` status window. These prompts appear as follows when the test is waiting for input:

- `STARTUP>` —Test has been reset and is ready to run.
- `RUN>`—Subtests are running.
- `PRE-CLEANUP>` —Subtest is paused prior to any subtest cleanup being done.
- `POST-CLEANUP>` —Test is paused after cleanup of the prior subtest.
- `COMPLETE>` —Test is paused upon completion of the subtest sequence.

Enter the command after the prompt.

4.13.3 Running a test

After entering the test name, run the test by entering:

```
STARTUP> run
```

4.13.4 Pausing test execution

Pressing **CONTROL-c** interrupts a test and allows you to enter new commands or change parameters.

4.13.5 Getting help

The `help` display provides information about test commands and parameters. At the command line, enter

```
>help
```

to view the `help` screen. Figure 4-21 shows the `help` display.

Figure 4-21
`help` display

```
There are two commands that can be used to get information about the
parameters and commands used in this program.

  com                 this command prints out the list of commands that
                      are known by the program, and a short explanation
                      of its purpose.

  par [parameter]     this command gives information on a parameter. It
                      has an optional parameter name argument. If the
                      name is omitted, a list of the parameters and
                      their current values is printed. If a specific
                      parameter is named, details about the named
                      parameter is given.

In addition, the following commands list subsets of the parameters:
summary, globals, and test_par.

To learn the value of an individual parameter, type its name.

To set the value of an individual parameter, type its name and a
value.
```

4.13.6 Logging test information

Use the following commands to log test information into a file:

```
STARTUP> log_file path/filename
STARTUP> log_input 1
STARTUP> log_output 1
```

Disable either the input or output logger by entering 0 instead of 1 in either of the latter two of the above commands.

4.13.7 Writing a file

To append all screen output to a file, use the following command when invoking a diagnostic test:

spu> *testname* | **tee -a** *filename*

4.13.8 Exiting

You can exit the diagnostics program at any time by entering:

> **exit**

4.13.9 Getting test information

You can obtain the same information by command that is available through The X Window System. To display this information, use the following commands.

4.13.9.1 Subtests

You can display a list of subtests for the current test by entering:

> **sub**

This list is limited to subtests that can actually run with the current machine configuration. To get a list of all available subtests for the current test, enter:

> **sub_all**

Figure 4-12 on page 4-21 shows a typical subtest list display.

4.13.9.2 Test dependencies

You can display a list of test dependencies by entering:

>**dep**

Figure 4-13 on page 4-22 shows the test dependencies display. Test dependencies may inhibit the operation of subtests. Refer to Section 4.12.3 for an explanation of test dependencies.

4.13.9.3 Subtests segregated by class

You can display a list of subtests for the current test, segregated by class, by entering:

```
> class
```

4.13.9.4 Summary of test parameters

You can display a summary list of test parameters by entering :

```
>summary
```

Figure 4-14 on page 4-23 shows the summary display of test parameters. Section 4.12.8 gives definitions of all parameters.

4.13.9.5 Global parameters

Parameter names are identical to those appearing in the `xdiag` windows. You can display the list of global parameters by entering:

```
>globals
```

Figure 4-15 on page 4-24 shows the global parameter display. Section 4.12.8 gives definitions of all parameters.

4.13.9.6 Test specific parameters

Test-specific parameter names are identical to the names used in their test-specific `xdiag` windows. You can display the list of test specific parameters by entering:

```
>test_par
```

Figure 4-16 on page 4-25 shows a typical display of test specific parameters. The chapters on individual diagnostic tests show the test specific parameter displays and give definitions of the parameters.

4.13.9.7 All parameters

Parameter names are identical to those appearing in the `xdiag` windows. You can display the list of all parameters by entering:

```
>par
```

Figure 4-17 on page 4-27 shows the display of all parameters. Section 4.12.8 gives definitions of all parameters.

4.13.9.8 Commands

Enter

```
>com
```

to display the list of commands. Figure 4-18 on page 4-31 shows a list of commands and definitions.

4.13.10 Altering test parameters

To alter a test parameter, enter the parameter name and new value after the prompt. For example:

```
STARTUP> pause_on_fail 1
```

Service processor interface test (spu4000)

5

The spu4000 test verifies the interface between the service processor (SPU), the SPU interface circuit boards, and parts of the scan system in C3800 Series computer field replaceable units (FRUs).

The spu4000 series 200 subtests test the SPU itself. These subtests do not require the SPU to be connected to a computer.

The rest of the spu4000 subtests test continuity between the SPU and all computer FRUs. They test the ability of the computer FRUs to respond properly to SPU operations.

5.1 Prerequisites

The following conditions must be met before running spu4000:

- A C3800 Series SPU must be powered and booted, with the SPU OS prompt displayed.
- The SPU self-test must have completed without significant error.

5.2 Hardware requirements

Table 5-1 gives the hardware requirements for the `spu4000` subtests by class.

Table 5-1
Hardware required for `spu4000` subtests by class

Class	Subtests	Minimum hardware
1	SWIP	SPU, SWIS
2	SWIS	SPU, SWIP
3	Scan engine/clock generator	SPU, SWIS, SWIP, bay power controller(s) for bay(s) containing FRU(s) under test, CPU utilities board, XCL board in crossbar subsystem
4	Scan loopback	All boards required for class 3 subtests
5	Ring integrity	All boards required for class 3 subtests, fully populated crossbar subsystem, interface adapter, at least 1 scalar processor/vector processor pair, at least one memory board
6	Hard/soft error and interrupt	All boards required for class 5 subtests, except vector processor not required
7	Connectivity	All boards required for class 5 subtests

Figure 5-1 shows the parts of the system under test and indicates the FRUs required to run the class 1 subtests of `spu4000`.

Figure 5-2 shows the parts of the system under test and indicates the FRUs required to run the class 2 subtests of `spu4000`.

Figure 5-3 shows the parts of the system under test and indicates the FRUs required to run the class 3 and class 4 subtests of `spu4000`.

Figure 5-4 shows the parts of the system under test and indicates the FRUs required to run the class 5 and class 7 subtests of `spu4000`.

Figure 5-5 shows the parts of the system under test and indicates the FRUs required to run the class 6 subtests of `spu4000`.

Figure 5-1
FRUs required and exercised by spu4000 class 1 subtests

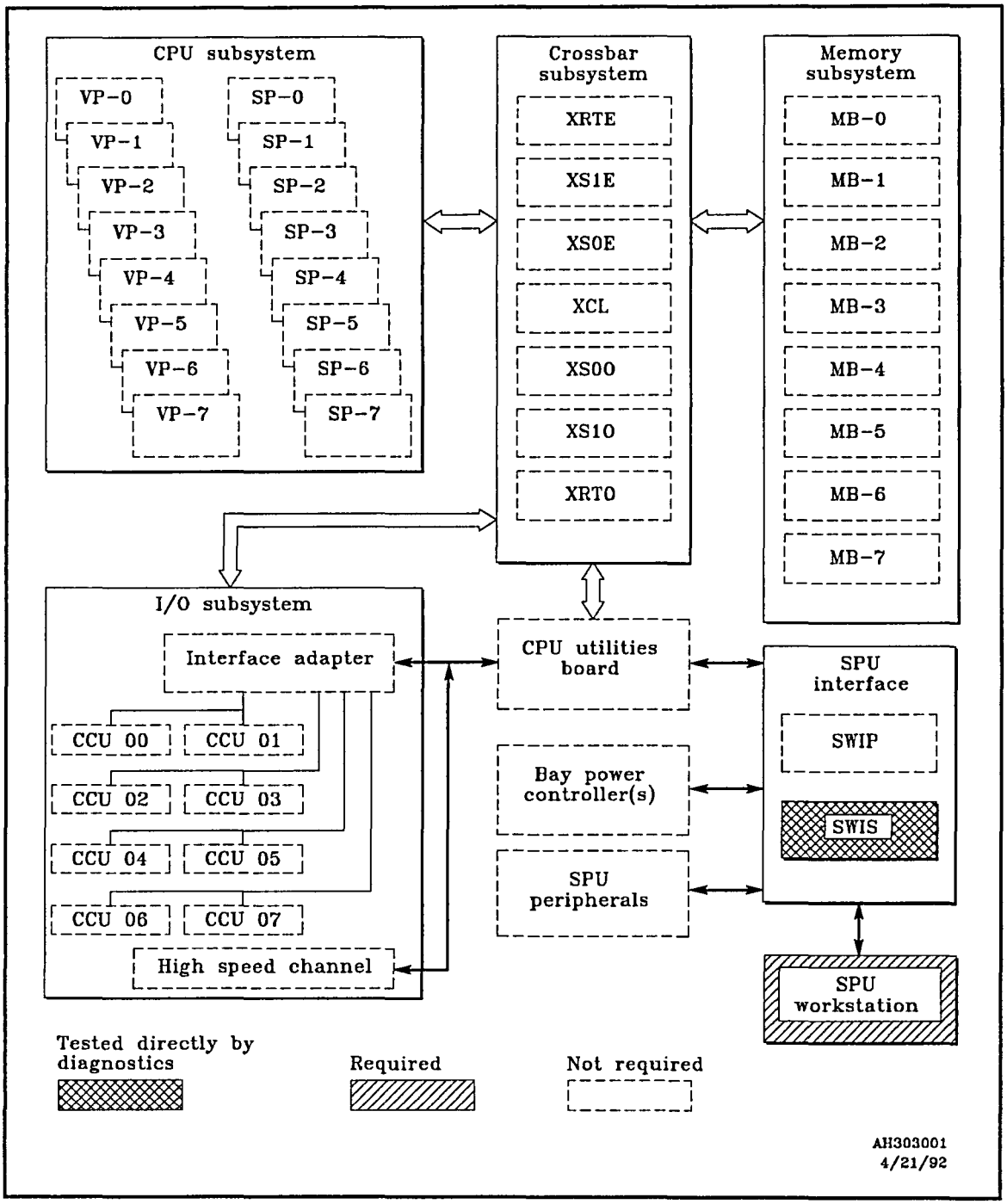


Figure 5-2
FRUs required and exercised by `spu4000` class 2 subtests

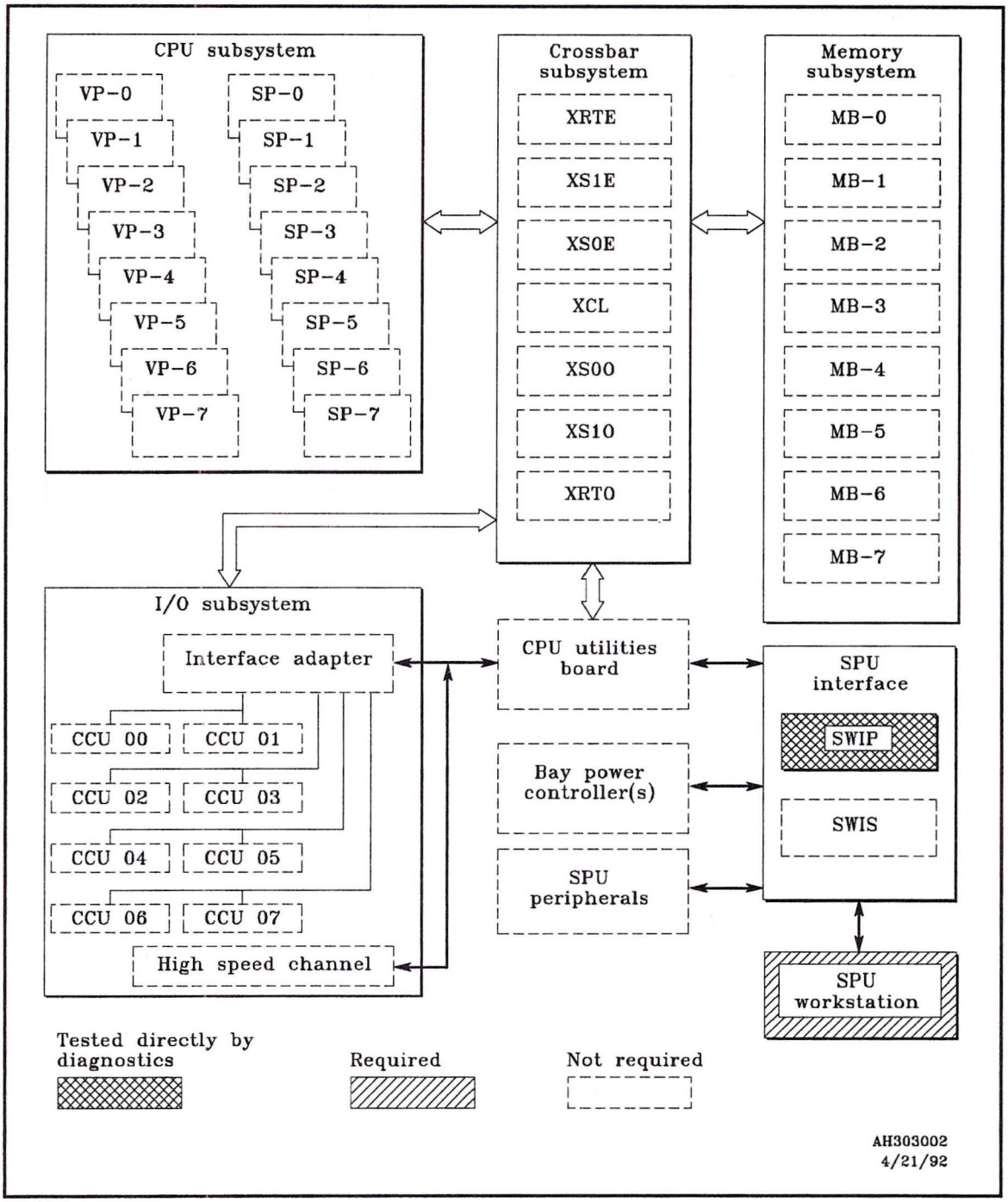


Figure 5-3
FRUs required and exercised by spu4000 class 3 and 4 subtests

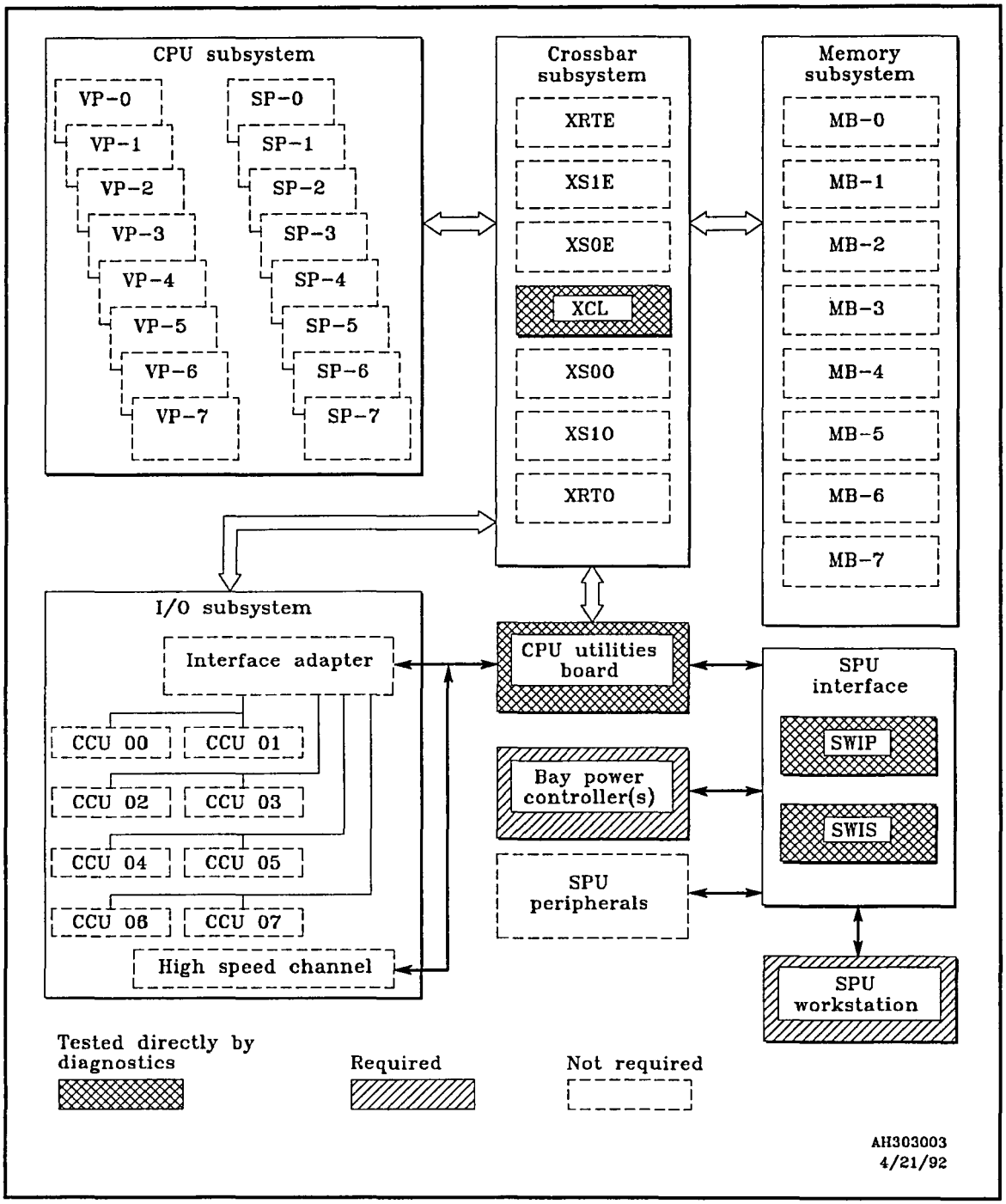


Figure 5-4
FRUs required and exercised by `spu4000` class 5 subtests

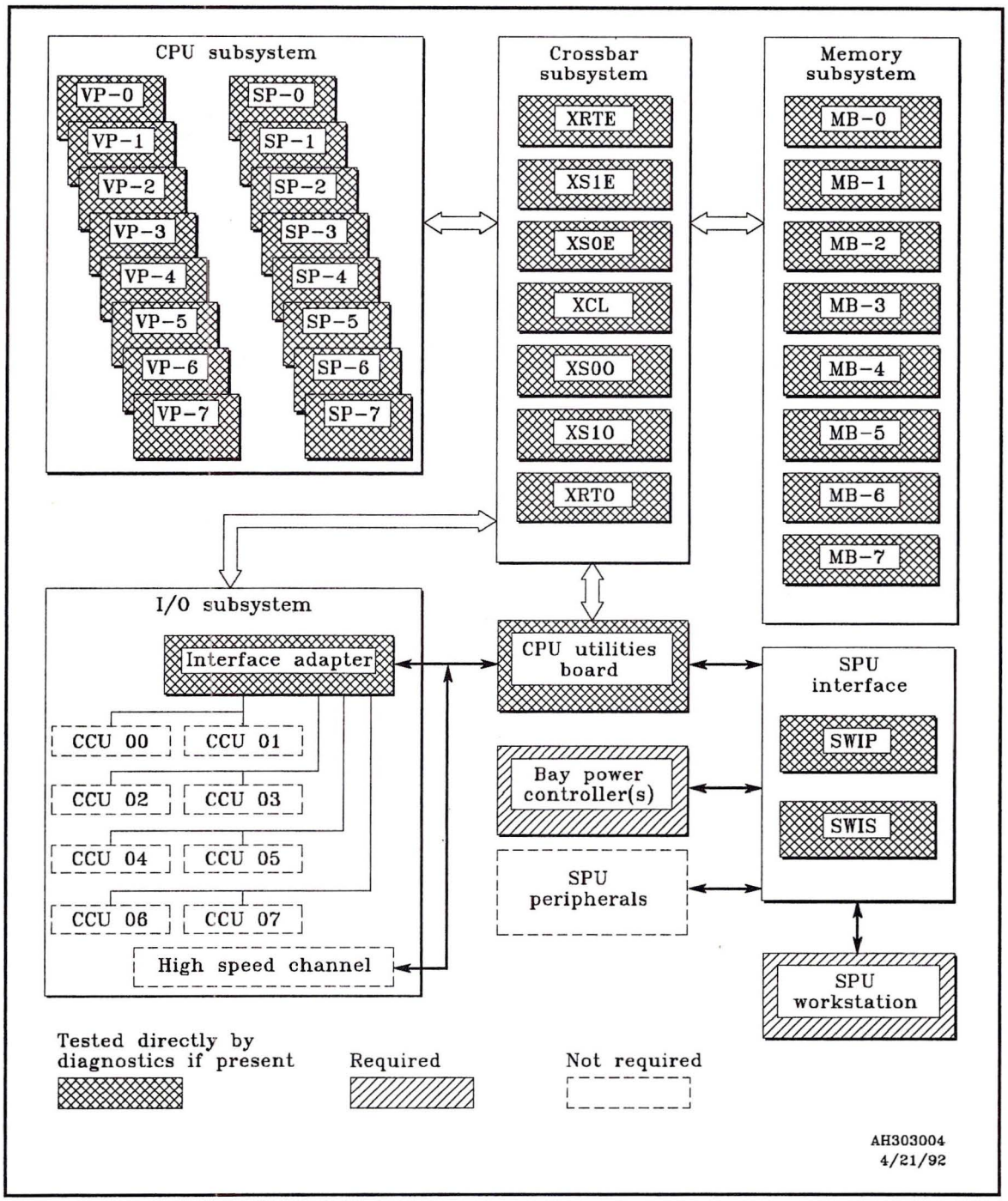


Figure 5-5
FRUs required and exercised by spu4000 class 6 subtests

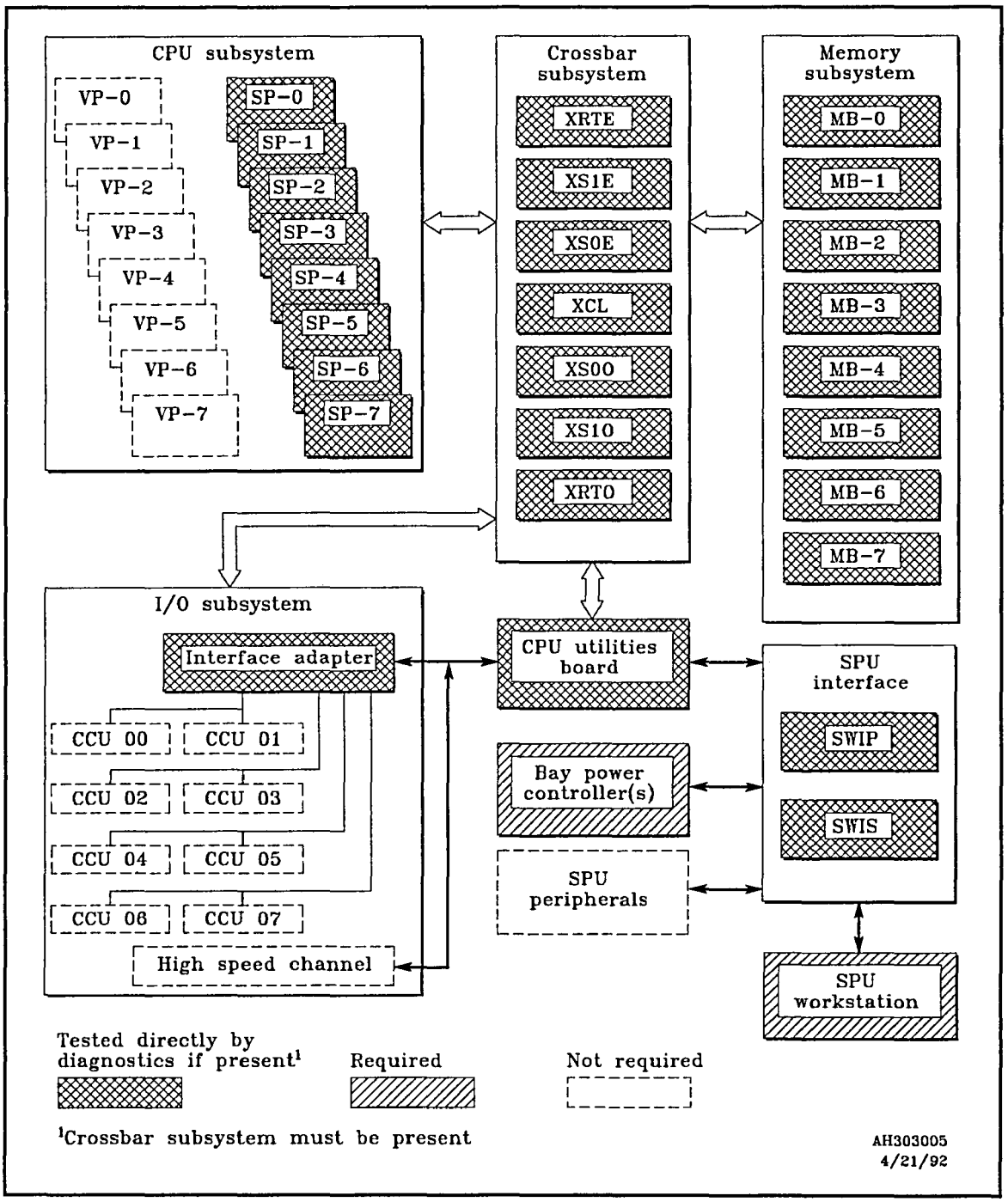
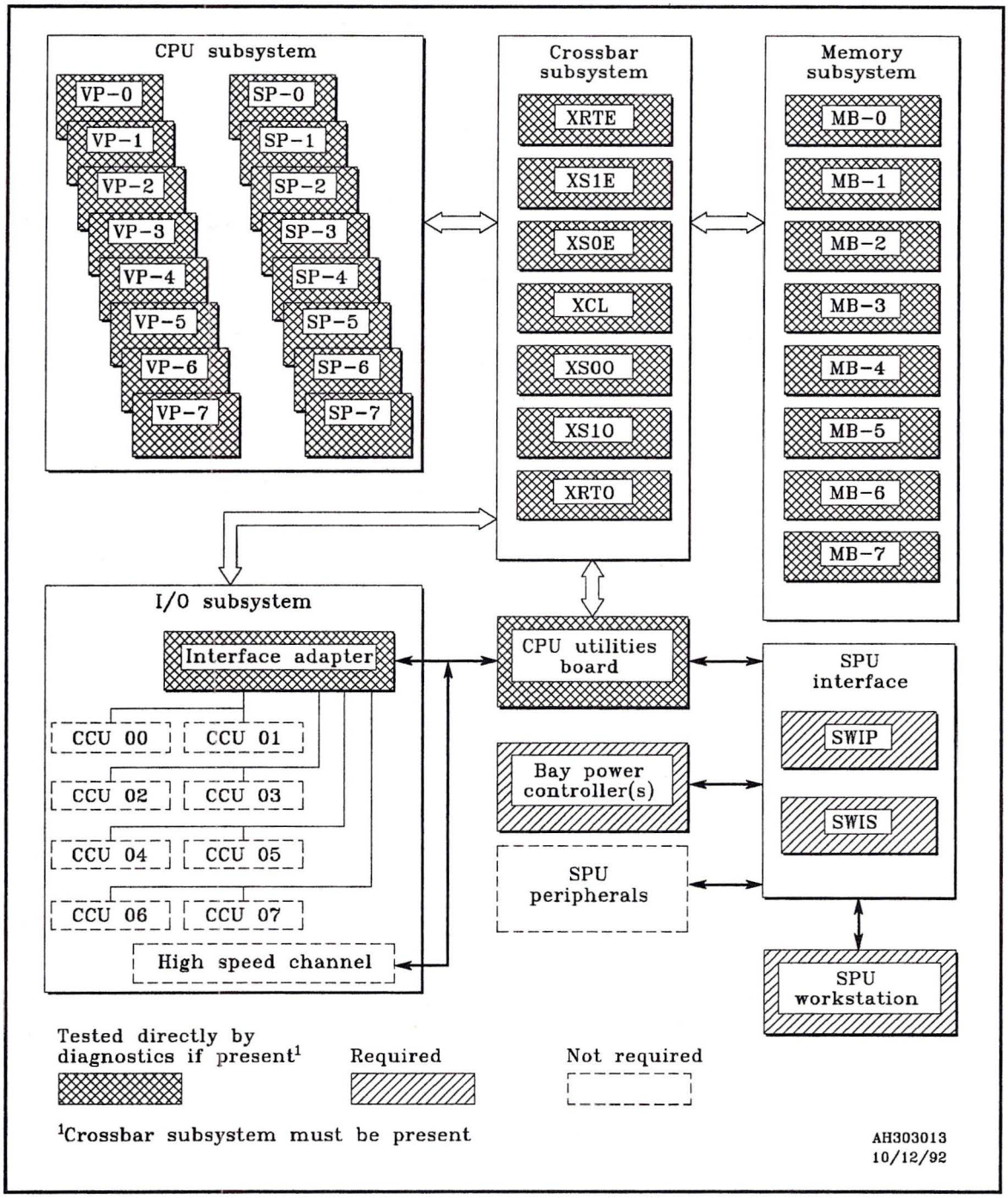


Figure 5-6
FRUs required and exercised by `spu4000` class 7 subtests



5.3 Invoking the test

Use the following command to enter the xdiag environment:

```
(spu)> xdiag
```

xdiag is the X-based CONVEX test interface (cti) used by all C3 processor diagnostics. Refer to Chapter 4 for a description of xdiag.

From the main xdiag window, select Test and click on spu4000 in the drop-down menu. This causes the cti to load the appropriate diagnostic files. The term TestParameters appears in the xdiag window menu bar.

To run the test, click on run in the xdiag window command bar. The test will run with default parameters unless you have customized the test options.

The following subsections describe the customizing process.

5.4 Test menus

The spu4000 test has several windows that are available under the xdiag test control system. These windows enable you to select and control the test-specific parameters.

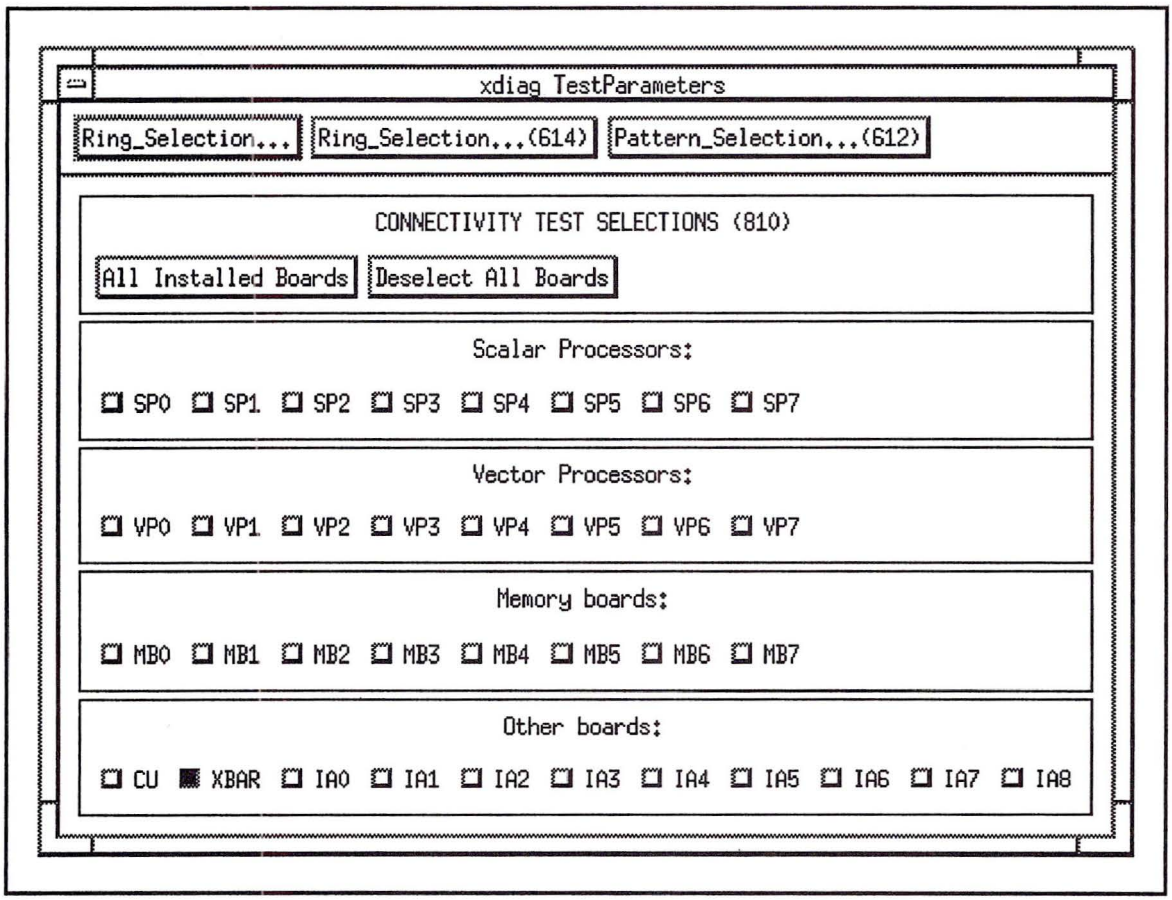
5.4.1 xdiag TestParameters window

Click on TestParameters... in the xdiag window menu bar to cause the xdiag TestParameters window to appear. The xdiag TestParameters window provides access to three other windows through selection boxes at the top of the window:

- Ring_Selection
- Ring_Selection (614)
- Pattern_Selection (612)

The xdiag TestParameters window also contains a menu of FRUs that you may select for connectivity testing (class 7). Figure 5-7 shows the xdiag TestParameters window.

Figure 5-7
xdiag TestParameters window



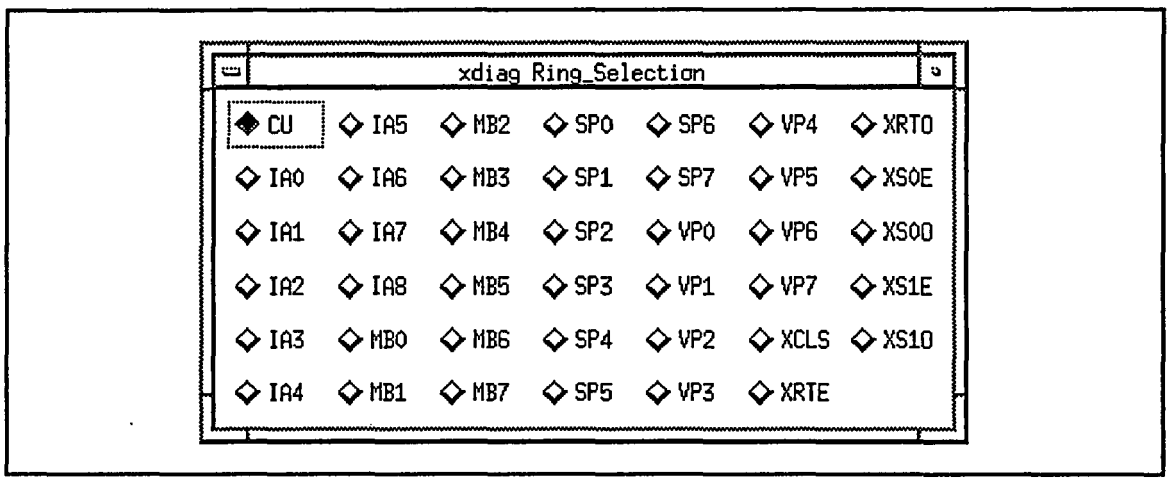
5.4.2 xdiag Ring_Selection window

The xdiag Ring_Selection window presents a list of scan rings which you may select for individual tests. The following subtests exercise one ring at a time.

- 510
- 511
- 610
- 611
- 612
- 613
- 615

Figure 5-8 shows the xdiag Ring_Selection menu window.

Figure 5-8
Test ring selection menu



Follow these steps to select a ring:

Step 1 Click on Ring_Selection in the xdiag TestParameters window to cause the xdiag Ring_Selection window to appear.

Step 2 Click on the ring to be tested.

5.4.3 xdiag Sys_Ring_Selection window

The xdiag Sys_Ring_Selection window applies to subtest 614 only. It provides a list of portions of scan rings, which you may select. Click on Ring_Selection (614) in the xdiag TestParameters window to cause the xdiag Ring_Selection window to appear. Figure 5-9 shows the xdiag Sys_Ring_Selection window.

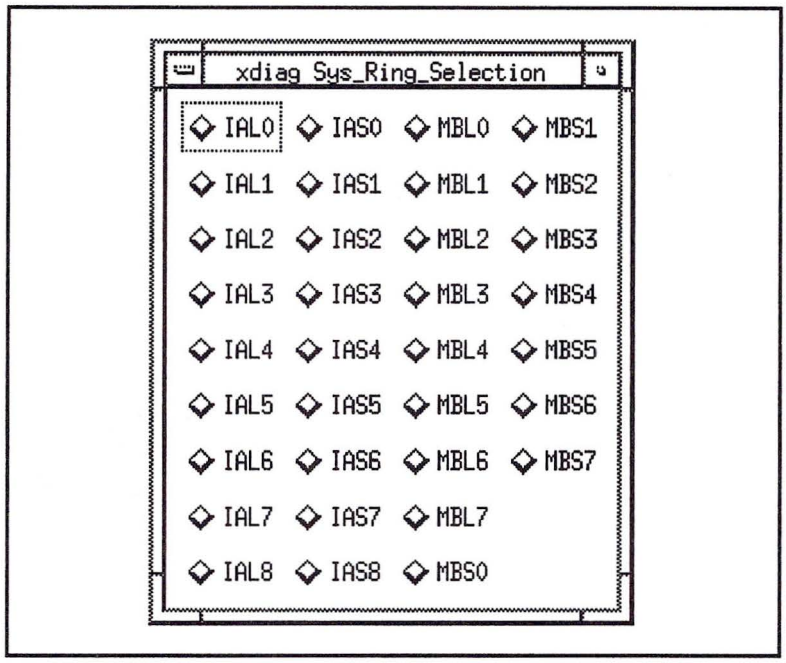


Figure 5-9
System ring selection menu for subtest 614 only

5.4.4 xdiag Patt window

The xdiag Patt window applies to subtest 612 only. It offers a range of digital test patterns which you may select. Click on Pattern_Selection (612) in the xdiag TestParameters window to cause the xdiag Patt window to appear. Figure 5-10 shows the xdiag Patt window.

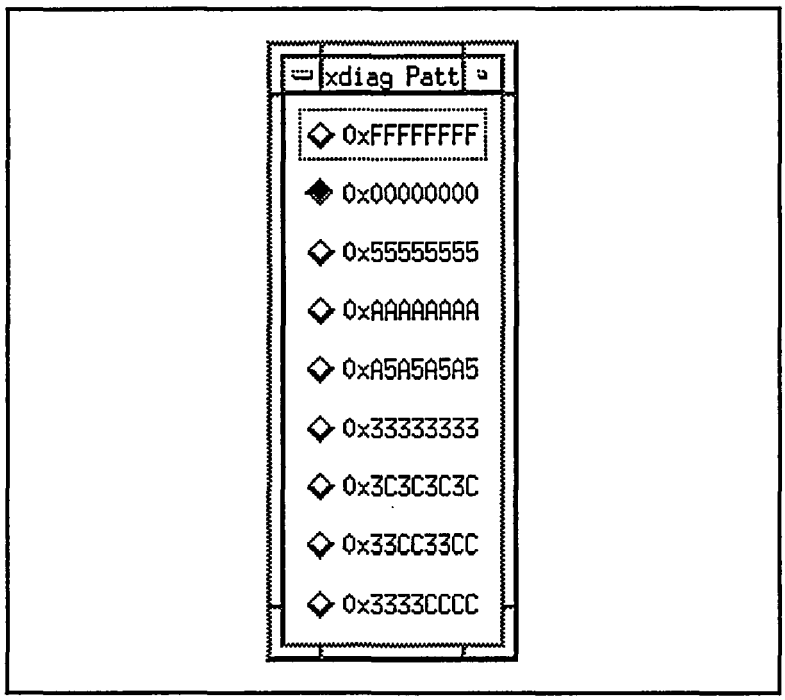


Figure 5-10
Test pattern selection menu for subtest 612 only

5.5 When the X-Window environment is not available

If you are not working in an X-Window environment, refer to Section 4.13 on page 4-33 for information on invoking the diagnostics software.

To invoke `spu4000`, enter:

```
spu> spu4000
STARTUP>
```

To display test specific parameters, enter:

```
STARTUP> test_par
```

Figure 5-11 shows the test-specific parameters for the `spu4000` test.

Figure 5-11
`spu4000` test-specific parameters

```
- "cu_selected" "0"
- "ia_selected" "0,0,0,0,0,0,0,0,0"
- "loop_count" "1"
- "mb_selected" "0,0,0,0,0,0,0,0"
- "pattern_number" "0x00000000"
- "ring_number" "40"
- "sp_selected" "0,0,0,0,0,0,0,0"
- "vp_selected" "0,0,0,0,0,0,0,0"
- "xbar_selected" "1"
```

These are the parameter definitions:

- `cu_selected`—Tests the computer utilities board if 1; inactive if 0.
- `ia_selected`—The 9 parameters in the array represent scan rings in interface adapters (IAs) 0 through 8, respectively. Tests the IAs whose parameters are set to 1; does not test the IAs whose parameters are set to 0.
- `loop_count`—This is a special parameter applying to test 612 only. Causes fast looping for the number of loops specified if greater than 1. Default is 1, allowing normal testing.
- `mb_selected`—The 8 parameters in the array represent scan rings in memory boards (MBs) 0 through 7, respectively. Tests the MBs whose parameters are set to 1; does not test the MBs whose parameters are set to 0.

- `pattern_number`—This parameter applies to test 612 only. You can set the test pattern to any hexadecimal value.
- `ring_number`—You can select any single ring for testing, using the appropriate parameter value. Table 5-2 shows parameter entries for subtests `510`, `511`, `610`, `611`, `612`, `613`, and `615`.

Table 5-2
Scan ring parameters

Ring	Select	Ring	Select	Ring	Select	Ring	Select	Ring	Select
CU	40	MB0	77	SP0	101	VP0	109	XCLS	117
IA0	41	MB1	78	SP1	102	VP1	110	XRTE	147
IA1	42	MB2	79	SP2	103	VP2	111	XRTO	148
IA2	43	MB3	80	SP3	104	VP3	112	XS0E	151
IA3	44	MB4	81	SP4	105	VP4	113	XS0O	152
IA4	45	MB5	82	SP5	106	VP5	114	XS1E	155
IA5	46	MB6	83	SP6	107	VP6	115	XS1O	156
IA6	47	MB7	84	SP7	108	VP7	116		
IA7	48								
IA8	49								

Table 5-3 shows parameter entries for subtest 614 only.

Table 5-3
System scan ring parameters, subtest 614 only

Ring	Select	Ring	Select	Ring	Select	Ring	Select
IAL0	59	IAS0	68	MBL0	85	MBS0	93
IAL1	60	IAS1	69	MBL1	86	MBS1	94
IAL2	61	IAS2	70	MBL2	87	MBS2	95
IAL3	62	IAS3	71	MBL3	88	MBS3	96
IAL4	63	IAS4	72	MBL4	89	MBS4	97
IAL5	64	IAS5	73	MBL5	90	MBS5	98
IAL6	65	IAS6	74	MBL6	91	MBS6	99
IAL7	66	IAS7	75	MBL7	92	MBS7	100
IAL8	67	IAS8	76				

- `sp_selected`—The 8 parameters in the array represent scan rings in scalar processor (SP) boards 0 through 7, respectively. Tests the SPs whose parameters are set to 1; does not test the SPs whose parameters are set to 0.
- `vp_selected`—The 8 parameters in the array represent scan rings in vector processor (VP) boards 0 through 7, respectively. Tests the VPs whose parameters are set to 1; does not test the VPs whose parameters are set to 0.
- `xbar_selected`—Tests the crossbar if 1; inactive if 0.

5.6 Class descriptions

The spu4000 test consists of 7 classes of subtests. These subtests exercise the following C3800 Series subsystems:

- **Class 1**—Exercises the interrupt and error detection circuitry and the universal asynchronous receiver-transmitter (UART) functions on the SPU workstation interface serial (SWIS) circuit board.
- **Class 2**—Exercises register access, the NCU data bus loopback, and the interrupt and error detection circuitry on the SPU workstation interface parallel (SWIP) circuit board.
- **Class 3**—Exercises the NCU cable and control lines, the memory test register interface, the clock generator and scan engine registers on the NCU and XCL boards, and the scan memory, the scan engine logic, and the XMAP memory.
- **Class 4**—Tests scan hardware and software on user-selected rings using a loopback circuit.
- **Class 5**—Passes data through scan rings and reads the data back. Checks scan engine verification logic. Checks clock cable connectivity.
- **Class 6**—Forces hard and soft errors and interrupts, and checks for proper handling of these events.
- **Class 7**—Checks that all boards and cables are properly connected to the C3800 Series complex.

5.7 Subtest descriptions

The following sections and tables give detailed descriptions of all subtests by class. All applicable subtests for the SPU are listed in Table 5-4.

Table 5-4
`spu4000` subtests

Subtest	Class	Description
210	1	SWIS register test
211	1	SWIS interrupt test
212	1	SWIS error detection test
213	1	SWIS UART register test
214	1	SWIS UART data loopback test
220	2	SWIP register test
221	2	SWIP data loopback test
222	2	SWIP interrupt test
223	2	SWIP error detection test

Table 5-4 (continued)
spu4000 subtests

Subtest	Class	Description
310	3	NCU address loopback test
311	3	NCU memory test logic registers test
410	3	NCLK register test
411	3	Scan engine register test
412	3	SECG (XCL) register test
413	3	SECG register uniqueness test
414	3	Scan memory test
415	3	Scan memory uniqueness test
416	3	Local loopback scan test
417	3	Local loopback scan with verify test
418	3	XCL loopback scan with verify test
420	3	XMAP memory test
421	3	XMAP memory uniqueness test
510	4	Individual scan rings local loopback test
511	4	Individual scan rings crossbar loopback test
610	5	XCL system scan ring verification test
611	5	Individual scan rings verification test
612	5	Individual pattern scan ring test
613	5	Individual ring signatures test
614	5	System and log ring verification test
615	5	Clock cable test
710	6	Hard errors test
711	6	Soft errors test
712	6	Interrupt enable/disable test
713	6	Forced interrupt/masking test
810	7	Connectivity test

5.7.1 Class 1 and class 2 subtests

Class 1 and class 2 subtests exercise the workstation interface boards in the SPU.

Subtests 210 through 214 exercise the SPU workstation interface serial (SWIS) circuit board in the SPU. These tests must be run with the serial interface cable connected to both the SPU and the C3800 complex, and with the C3800 complex powered down.

Caution

Subtests 210 through 214 CANNOT be run with the C3800 complex powered up. These tests destroy state of the machine.

Subtests 220 through 223 exercise the SPU workstation interface parallel (SWIP) circuit board in the SPU. These tests must be run with the serial interface cable connected to both the SPU and the C3800 complex.

Table 5-5 contains descriptions of the spu4000, 200 series class 1 subtests.

Table 5-5
spu4000 class 1 subtests

Subtest	Description
210	To verify the ability of the SPU to access the SWIS registers: force parity error register, miscellaneous register, interupt enable, and the forced interrupt register.
211	To verify the operation of the SWIS interrupt circuitry. Each interrupt is forced, masked and disabled.
212	To verify the SWIS error detection circuitry: parity and invalid address. Errors are forced and detected.
213	To verify the ability of the SPU to access the UART registers. Each UART channel is tested with different data patterns.
214	Perform a loopback test of the SWIS UARTs. Functional test of UART channels 6 and 7 which are connected with an external loopback.

Table 5-6 contains descriptions of the spu4000, 200 series class 2 subtests.

Table 5-6
spu4000 class 2 subtests

Subtest	Description
220	To verify the ability of the SPU to access the SWIP registers: force parity error register, miscellaneous register, interupt enable, and the forced interrupt register.
221	To verify the data bus out to the NCU connector. This test uses the SWIP data loop back register to perform the test.
222	To verify the operation of the SWIP interrupt circuitry. Each interrupt is forced, masked and disabled.
223	To verify the SWIP error detection circuitry: parity and invalid address. Errors are forced and detected.

5.7.2 Class 3 subtests, 300 series

The 300 series subtest tests the NWI–NCU interface as described in Table 5-7.

Table 5-7
`spu4000` class 3, 300 series subtests

Subtest	Description
310	Data patterns are written to the NCU and read back and verified. Verifies the cable to the NCU. Verifies proper control lines.
311	Verifies interface to the memory test registers on the NCU.

5.7.3 Class 3 subtests, 400 series

The 400 series subtests test the scan engine and clock generator circuitry of the NCU. Table 5-8 contains descriptions of these subtests.

Table 5-8
spu4000 class 3, 400 series subtests

Subtest	Description
410	Clock generator registers access test.
411	Scan engine registers (NCU resident) access test.
412	Scan engine registers (XCL resident) access test.
413	Scan engine/clock generator register uniqueness test. Verifies that accessing one register does not affect other registers.
414	Scan memory test. This test verifies the four data pages used by the scan engine. Multiple patterns are written/read to each location.
415	Scan memory uniqueness test. Verifies that writing or reading to one memory address does not affect other addresses.
416	Local loopback scan test. This test performs a scan in local loopback mode. Data does not actually reach a logic board. This test only involves the scan engine circuitry.
417	Local loopback with verify test. This test performs a scan with verification enabled, in local loopback mode. The goal of this test is to verify the scan engine verification circuitry.
418	Crossbar loopback scan test. Similar to subtest 416 except the loopback takes place on the XCL board. This tests out some additional scan circuitry not tested in subtest 416.
420	XMAP memory test. This test verifies the XMAP registers. Multiple patterns are written/read to each location.
421	XMAP memory uniqueness test. Verifies that writing or reading to one memory address does not affect other addresses.

5.7.4 Class 4 subtests

Class 4 subtests perform scans in loopback mode on individually selectable rings, as described in Table 5-9.

Table 5-9
spu4000 class 4 subtests

Subtest	Description
510	Similar to subtest 416 except that the user specifies which ring to select. This tests the ring-specific scan software.
511	Similar to subtest 418 except that the user specifies which ring to select. This tests the ring-specific scan software and additional XCL scan hardware.

5.7.5 Class 5 subtests

Class 5 subtests verify scan rings on logic boards, as described in Table 5-10.

Table 5-10
spu4000 class 5 subtests

Subtest	Description
610	XCL scan ring test. This test performs a scan of the xcl system ring, verifying that the data scanned into the ring matches the data scanned out of the ring. This test also forces every scan bit to fail and verifies that each bit was detected by the scan engine verification circuitry.
611	Individual scan rings verification test. This test loads a number of different patterns into the ring selected and verifies that the data read out is correct. The user selects which ring to test.
612	Individual pattern scan ring test. This test is similar to subtest 611 except that only one pattern is used and the pattern is selected by the user.
613	Individual ring signatures test. This test retrieves each of the signatures in the selected scan ring and verifies that they match the signature pattern (0x1A9).
614	System and log ring verification test for the interface adapter and memory boards. User selects which board.
615	Clock cable test. Verifies that the proper clock cable is connected to a board and that the scan control lines are functional.

5.7.5.1 Class 6 subtests

The 700 series subtests verify the ability of the system to handle hard errors, soft errors and interrupts to the SPU. These tests are described in Table 5-11.

Table 5-11
spu4000 700 series subtest descriptions

Subtest	Description
710	Forces hard errors on the logic boards and verifies that they are reported correctly.
711	Forces soft errors on the logic boards and verifies that they are reported correctly.
712	Forces interrupts and verifies that they can be enabled and disabled correctly.
713	Forces interrupts and verifies that the interrupts can be individually enabled and disabled.

5.7.5.2 Class 7 subtest

Subtest `810` performs tests to verify board connectivity. You select which boards (memory, scalar processor, vector processor, crossbar) to test. Selection is by port number. Valid range is 0 through 7. Figure 5-12 shows a typical subtest `810` error display.

The fields of Figure 5-12 have the following definitions:

- `Detected Short or Stuck-at-1`—Two signals are shorted together, or the signal is stuck at 1. Multiple error messages are probable.

  Another possible message is:

  `Detected Open or Stuck-at-0`—Signal continuity is broken, or the signal is stuck at 0. Multiple error messages are unlikely.

- `Source Signal:`—One bit of this signal was set to logic level 1, with all other bits and signals at logic level 0.
- `Source Port:`—The port of the source signal. The signal name only identifies the source board. Since several ports may have identical boards, the `Source Port:` field identifies the port.
- `Sink Signal:`—One bit of this signal should be at logic level 1, with all other bits and signals at logic level 0.
- `Sink Port:`—The port of the sink signal. The signal name only identifies the target board. Since several ports may have identical boards, the `Sink Port:` field identifies the port.
- `Sink Field:`—Used for developmental purposes only.
- `Failing Bit Position:`—The position of the bit in the sink signal that does not have the expected value.
- `Expected:` The value of the sink signal that should have appeared at the sink port.
- `Actual:` The value of the sink signal that actually appeared at the sink port.

Figure 5-12
spu4000 subtest 810 error display

```
Detected Short or Stuck-at-1
Source Signal: xso_mb.wr_data                    Source Port: 0
Sink Signal:   sp_xso.wr_data                    Sink Port:   1
Sink Field:    xslo:wr_data_is[2]
Failing Bit Position:     15
Expected:                 00000000
Actual:                   00008000
****
```

Utilities subsystem test (cu4000)

6

cu4000 is a functional test of the utilities subsystem. The major functional logic blocks residing on the CPU utilities (CU) board are:

- Scan engine and clock generator (SECG)
- Memory test logic (MTL)
- Communication registers
- Control space
- ASAP logic
- Trap processing
- Interrupt processing
- Interval timer
- Time-of-century (TOC)

All but the SECG and MTL are tested by this diagnostic. spu4000 tests the SECG. mem4000 tests the MTL.

The cu4000 diagnostic test exercises some of the communication registers, ASAP logic, I/O registers, TOC, interval timer, and trap logic. It only performs scan-based tests on the CU board. More comprehensive testing of the ASAP logic, traps, and interrupts requires CPU-based tests.

6.1 Prerequisities

The following conditions must be met before running `cu4000`:

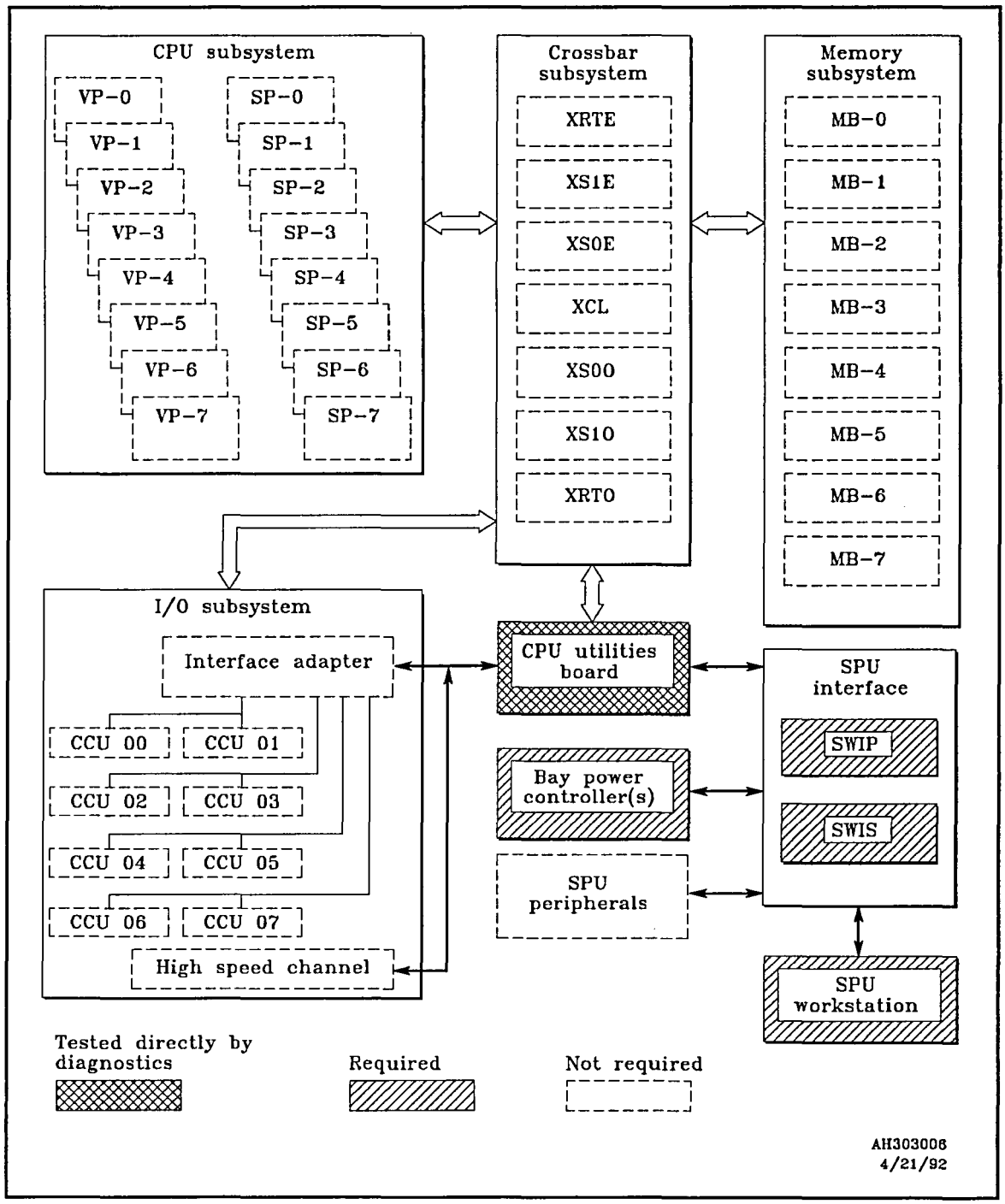
- A C3800 Series SPU with SWIS and SWIP boards must be powered and booted, with the SPU OS prompt displayed.
- The SPU self-test must have completed without significant error.
- The `spu4000` diagnostic test must have completed without error.

6.2 Hardware requirements

The `cu4000` diagnostic test requires an operational SPU connected to a C3800 Series computer. The computer must contain at least a CPU utilities (CU) board and the bay power controller (BPC) that supplies the CU with power. The CU board must be powered.

Figure 6-1 shows the parts of the system under test and indicates the field replaceable units (FRUs) required to run `cu4000`.

Figure 6-1
FRUs required and exercised by cu4000



6.3 Invoking the test

Invoke the xdiag environment by entering:

```
(spu)> xdiag
```

xdiag is the X-based CONVEX test interface (cti) used by all C3 processor diagnostics. Refer to Chapter 4 for a description of xdiag.

From the main xdiag window, select Test and click on cu4000 in the drop-down menu. This causes the cti to load the appropriate diagnostic files. The term TestParameters appears in the xdiag window menu bar.

To run the test, click on run in the xdiag window command bar. The test will run with default parameters unless you have customized the test options.

The following subsections describe the customizing process.

6.4 Test menu

Figure 6-2 shows the cu4000 xdiag TestParameters menu available under the xdiag test control system. This window enables you to select and control the test-specific parameters.

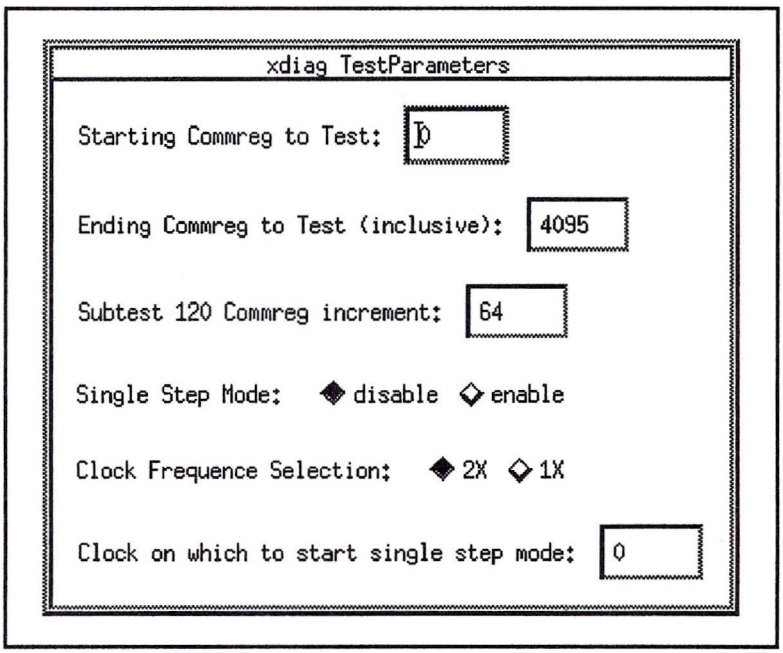


Figure 6-2
cu4000 xdiag TestParameters menu

The fields of Figure 6-2 have the following meanings:

- `Starting Commreg to Test:`—Enter the decimal number of the first communication register to test. cu4000 tests the communication registers in ascending order. The starting communication register number must not be greater than the ending communication register number. Default value is 0.
- `Ending Commreg to Test (inclusive):`—Enter the decimal number of the last communication register to test. The ending communication register number must be at least equal to the starting communication register number. Default value is 4095.
- `Subtest 120 Commreg increment:`—This parameter applies to subtest `120` only. Enter the decimal number *n* to test every *n*th register, beginning with `Starting Commreg to Test`.
- `Single Step Mode:`—Select enable to single step through the test; select disable to run the test normally.
- `Clock Frequency Selection:`—Select either the `2X` clock or the `1X` clock for single stepping.
- `Clock on which to start single step mode:`—Enter the *hexadecimal* number of clocks to run normally before beginning the single step mode.

6.5 When the X-Window environment is not available

If you are not working in an X-Window environment, refer to Section 4.13 on page 4-33 for information on invoking the diagnostics software.

To invoke `cu4000`, enter:

```
spu> cu4000
STARTUP>
```

To display test specific parameters, enter:

```
STARTUP> test_par
```

Figure 6-3 shows the test-specific parameters for the `cu4000` test.

Figure 6-3
`cu4000` test-specific parameters

```
- "Clock_frequency" "0"
- "Ending_commreg" "4095"
- "Initial_clocks" "0"
- "Single_step" "0"
- "St_120_increment" "64"
- "Starting_commreg" "0"
```

These are the parameter definitions:

- `Clock_frequency`—Single steps on 1x clock if 1; single steps on 2x clock if 0.
- `Ending_commreg`—Number of the last communication register to test. Range is 0 to 4095. Default is 4095. Must not be less than `starting_commreg`.
- `Initial_clocks`—Hexadecimal number specifies the number of clock ticks that occur before clock single stepping starts.
- `Single_step`—Applies to subtests `115`, `200`, `210`, `220`, `230`, and `300` only. The subtest pauses between each step if 1; inactive if 0.
- `St_120_increment`—Applies to subtest `120` only. Applies the test to every *n*th communication register. Range is 1 to 4095. Default value is 64. Setting to a value less than 64 results in a long execution time.
- `Starting_commreg`—Number of the first communication register to test. Range is 0 to 4095. Default is 0. Must not be greater than `ending_commreg`.

6.6 Class descriptions

This section describes the subtests in the cu4000 diagnostic. All testing is done using scan operations. The subtest scans a request into the input stage registers on the CU board, clocks the system, and checks the returned data and status.

Table 6-1 lists the 3 classes of subtests in this diagnostic.

Table 6-1
Subtest classes

Class	Description
1	Subtest 100: error verification subtest—verify error detection logic
2	Subtests 105-115: register verification subtests—verify register/lockbit space
3	Subtests 120-300: functional subtests

6.7 Subtest descriptions

The subtests are listed as follows:

- 100—Communication register parity error generation
- 105—Communication register lock bit test
- 110—Communication register pattern test
- 115—CU control space testing
- 120—Communication register functionality test
- 200—CU ASAP logic testing
- 210—CU RDCMR/WRCMR testing
- 220—CU TOC and ITC functionality testing
- 230—CU deadlock and firmware trap testing
- 300—CU interrupt testing

6.7.1 Class 1 subtests

6.7.1.1 Parity subtest (100)

Subtest 100 verifies parity checking on both data and control requests. The test verifies that the CU board correctly generates a hard error whenever bad parity is detected. In addition, it verifies that no hard error is generated whenever good parity exists. The following errors are verified:

- Data parity (both even and odd sides)
- Control parity

Subtest 100 checks parity on odd data (each byte of the 32-bit value), even data (each byte of the 32-bit value), and control parity (each 7 bits of the 28-bit value), or 12 error conditions. The order is odd data parity, even data parity, and control parity. The test uses the following procedure:

1. Scans a request with good parity into the CU board.
2. Clocks the board and verifies that no hard error was generated.
3. Scans a request with bad parity into the CU board.
4. Clocks the board.
5. Verifies that the correct hard error was generated due to a parity error.
6. Repeats for all possible error conditions.

Figure 6-4 shows an example of the subtest 100 error display.

Figure 6-4
Subtest 100 error display

```
Name of error detected

Performing longword request (r_l1_size=1)
Commreg request = PUT_L (r_l1_cmd_2_0=4)
Memory request = Write (nadr.l1_cycle=2)

Even Data = ffffffff
Even Data Parity = f  (ndat[1].l1_par)
Odd Data =  00000000
Odd Data Parity =  f  (ndat[0].l1_par & nadr.l1_dpar)
Control Parity Scan Fields
  r_l1_ctl_par_0= c
  r_l1_ctl_par_1= c
  nadr.cpar_l1= c
```

The fields in Figure 6-4 contain the following information:

- The first line displays the descriptive name of the error.
- The following 3 lines describe the execution progress of the error detection software.
- `Even Data` = displays the data from the pertinent even-numbered communication register in hex.
- `Even Data Parity` = displays odd parity of each byte from the pertinent even-numbered communication register in hex.
- `Odd Data` = displays the data from the pertinent odd-numbered communication register in hex.
- `Odd Data Parity` = displays odd parity of each byte from the pertinent odd-numbered communication register in hex.
- The last 3 lines give odd control parity values in hex. Possible values are 0, 4, 8, or C.

6.7.2 Class 2 subtests

6.7.2.1 Communication register lock bit subtest (105)

Subtest 105 verifies the ability to read and write the lock bit for each communication register. It tests the uniqueness of each lock bit and ensures that no bits are stuck. It does not verify the functionality of either the communication registers or lock bits. Figure 6-5 shows the error display for this subtest.

The test clears all communication registers and lock bits and tests each lock bit as follows:

1. Verifies lock bit is 0.
2. Sets lock bit to 1.
3. Verifies lock bit is 1 and all other lock bits are 0.
4. Clears lock bit.

Figure 6-5
Subtest 105 error display

```
Name of error detected

Commreg = 0000
Returned Lock Bit = 1
```

The fields in Figure 6-5 contain the following information:

- The first line displays the descriptive name of the error.
- Commreg = gives the hex address of the communication register whose lock bit is under test.
- Returned Lock Bit = gives the state of the communication register lock bit. A 1 means the register is locked; 0 means it is unlocked.

6.7.2.2 Communication register pattern subtest (110)

Subtest 110 verifies the communication registers' ability to hold data. It ensures that the communication registers are unique, and do not contain stuck bits. Functionality of neither the communication registers nor the lock bits is verified. Figure 6-6 shows a sample error screen for this subtest.

The test uses several different patterns to verify the communication registers. It writes all communication registers with the pattern and then reads them back and verifies them. Patterns used in testing the communication registers are 64 bits long.

Figure 6-6
Subtest 110 error display

```
Name of error detected

Expected Data for commreg = ffff
============================
  Upper 32-bits of commreg = ffffffff
  Lower 32-bits of commreg = ffffffff

Actual Data
===========
  Upper 32-bits of commreg = ffffffff
  Lower 32-bits of commreg = ffffffff
Data parity = ff
```

The fields in Figure 6-6 contain the following information:

- The first line displays the descriptive name of the error.
- `Expected Data for commreg` = gives the hex address of the communication register under test.

- Upper 32-bits of commreg = and Lower 32-bits of commreg = directly under the Expected Data field, give the bit pattern written to the upper and lower words of the communication register under test, respectively. Possible patterns include:
  - All 0s (00000000)
  - All 1s (ffffffff)
  - Alternating 1 and 0 (aaaaaaaa)
  - Alternating 0 and 1 (55555555)
  - cccccccc
  - 33333333
  - Addressing pattern
- Upper 32-bits of commreg = and Lower 32-bits of commreg = directly under the Actual Data field, give the bit pattern read from the upper and lower words of the communication register under test, respectively, after writing.
- Data Parity = gives the byte parity of the bit pattern stored in the communication register.

6.7.2.3 CU control space subtest (115)

Subtest 115 checks the integrity and accessibility of the control registers:

Step 1 Writes and reads each register using various patterns (put and get operations).

Step 2 Performs semaphored operations to various registers (snd and rcv operations).

Step 3 Checks word access to the registers.

Step 4 Performs the following sequence of operations and verifies the return status: ENI/DSI/DSI/ENI/ENI/DSI/ENI.

Step 5 Verifies separation between communication register space and control register space by making intermingled put and get requests to both spaces.

Subtest 115 uses the two error displays shown in Figure 6-7 and Figure 6-8. Figure 6-7 displays data errors. Figure 6-8 displays status errors.

6.7.3 Class 3 subtests

Class 3 subtests exercise the functionality of the communication registers and the CPU utilities associated with the communication registers.

All class 3 subtests except 120 use the two error displays shown in Figure 6-7 and Figure 6-8. Figure 6-7 displays data errors. Figure 6-8 displays status errors.

Figure 6-7
cu4000 subtest failure—return data error display

```
Name of error detected

Commreg = 0000      Operation = rcv_l          Clock = ffff
Expected Data
============
  Upper 32-bits of commreg = ffffffff
  Lower 32-bits of commreg = ffffffff

Actual Data
=========
  Upper 32-bits of commreg = ffffffff
  Lower 32-bits of commreg = ffffffff
  Upper rd_rdy  = 1
  Lower rd_rdy  = 1
```

The fields in Figure 6-7 contain the following information:

- The first line displays the descriptive name of the error.
- `Commreg` = gives the hex address of the communication register under test.
- `Operation` = gives the mnemonic of the operation being tested. Refer to Table 6-2, page 6-16, for a list of operation mnemonics and descriptions.
- `Clock` = used for development purposes only.
- `Upper 32-bits of commreg` = and `Lower 32-bits of commreg` = directly under the `Expected Data` field, give the bit patterns expected from the upper and lower words of the communication register under test, respectively.
- `Upper 32-bits of commreg` = and `Lower 32-bits of commreg` = directly under the `Actual Data` field, give the bit patterns actually read from the upper and lower words of the communication register under test, respectively.
- `Upper rd_rdy` = and `Lower rd_rdy` = are bit flags indicating whether or not the upper and lower words, respectively, of the communication register under test

are ready to be read. A 1 indicates that the word is ready; a 0 indicates that the word is not ready.

Figure 6-8
cu4000 subtest failure—incorrect status returned error display

```
Name of error detected

Commreg = 0000      Operation = snd_1          Clock = ffff
Expected Status       = 1
Actual Status         = 1
Actual Status Enable  = 1
Expected PID          = 08
Actual PID            = 08
```

The fields in Figure 6-8 contain the following information:

- The first line displays the descriptive name of the error.
- `Commreg` = gives the hex address of the communication register under test.
- `Operation` = gives the mnemonic of the operation being tested. Refer to Table 6-2, page 6-16, for a list of operation mnemonics and descriptions.
- `Clock` = used for development purposes only.
- `Expected Status` = is a bit flag giving the expected status resulting from the operation.
- `Actual Status` = is a bit flag giving the actual status resulting from the operation.
- `Actual Status Enable` = is a bit flag indicating whether the `Actual Status` flag has meaning. The `Actual Status` flag has meaning if the `Actual Status Enable` flag is 1.
- `Expected PID` and `Actual PID` give the identification number of the CPU to be exercised, and actually exercised, respectively, in the test. ID numbers 0-7 refer to CPUs 0-7. ID number 8 refers to the interface adapter (IA).

6.7.3.1 Communication register functionality subtest (120)

Subtest 120 verifies communication register functionality. It tests the ability to perform lock bit operations (lck, ulk, tst) and communication register operations (put, get, snd, rcv, inc) via scan. The test uses combinations of all 0 and all 1 data words with the lock bit state switching between set and clear. Table 6-2 lists these operations.

Table 6-2
Lock bit and communication register operations

Operation	Indices	Description
tst	0x00-0x01	Get lock bit for specified communication register
lck	0x02-0x05	Set lock bit for specified communication register
ulk	0x06-0x09	Clear lock bit for specified communication register
put_l	0x0a-0x0f	Store data into specified communication register
get_l	0x10-0x15	Read data from specified communication register
snd_l	0x16-0x1b	Send data to specified communication register (depends on lock bit state)
rcv_l	0x1c-0x21	Receive data from specified communication register (depends on lock bit state)
inc	0x22-0x28	Increment data in specified communication register (depends on lock bit state)
put_l	0x29	Special operation sequence
get_w	0x2a	

The default sequence tests all operations on every 64th communication register and lock bit. The 120 increment parameter specifies the number of communication registers under test. The first communication register tested is always 0.

Subtest 120 tests the following special cases:

- An inc operation with data parity not equal to 0xf. Subtest 120 generates a send parity error on the NDAT[0] gate array due to parity checking on the clock the data is written to RAM: the data parity is not held correctly.
- A communication register word read (get, rcv, or inc) following a communication register longword write (put or snd). Subtest 120 generates a RAM parity error due to parity checking of the even data when it should be ignored.

The following procedure tests the inc operation. The initial state of the communication register uses different combinations of all 0s and 1s with the lock bit switching between set and clear.

1. Initializes the communication register and lock bit.
2. Performs the inc operation.
3. Verifies that the returned communication register and status are correct, depending upon the initial state of the lock bit.
4. Reads and verifies the current contents of the communication register. If lock bit is 1, then the communication register should have been incremented, else it should not have changed.

Figure 6-9 and Figure 6-10 show the error displays for subtest 120.

Figure 6-9
cu4000 subtest 120 failure—return data error display

```
Name of error detected

Commreg = 0000      Operation = rcv_l          Index = 001c
Expected Data
============
  Upper 32-bits of commreg = ffffffff
  Lower 32-bits of commreg = ffffffff

Actual Data
==========
  Upper 32-bits of commreg = ffffffff
  Lower 32-bits of commreg = ffffffff
  Upper rd_rdy  = 1
  Lower rd_rdy  = 1
```

The fields in Figure 6-9 contain the same information as those of Figure 6-7 on page 6-14, except for the Index = field.

Index = gives the index number of the operation being tested. Refer to Table 6-2 on page 6-16, for a list of operation mnemonics and index numbers.

Figure 6-10
cu4000 subtest 120 failure—commreg operation error display

```
Name of error detected

Commreg Operation = put_l
Commreg = 0000
Data = ffffffff ffffffff
Lock Bit             = 1
Returned Error Code = 5
```

The fields in Figure 6-10 contain the following information:

- The first line displays the descriptive name of the error.
- `Commreg Operation` = gives the mnemonic of the operation being tested. Refer to Table 6-2, page 6-16, for a list of operation mnemonics and descriptions.
- `Commreg` = gives the hex address of the communication register whose lock bit is under test.
- `Data` = gives the data pattern used in the test.
- `Lock Bit` = gives the status of the communication register lock bit. A 1 means the communication register is unlocked.
- `Returned Error Code` = gives an indication of the type of error encountered. Table 6-3 gives a list of error codes and descriptions.

Table 6-3
`cu4000` subtest `120` error codes

Error code	Description
0	No error
1	Register number out of range
2	Utilities board not present in system
3	Illegal OP value specified
4	Status not enabled; should be
5	Send parity error detected

6.7.3.2 CU ASAP subtest (200)

Subtest 200 verifies the functionality of the ASAP logic. The test has 3 parts:

- ASAP priority rotation
- ASAP register interlock
- ASAP CPU mask usage

Subtest 200 inhibits acceptance of threads. The threads continue circulating in the ASAP logic. Normally the CPUs would claim the threads. The CPU installed mask is set to 0xff (all disabled) which causes the threads to continue circulating.

The communication registers and lock bits are cleared between the parts of the test. This avoids generation of false errors.

The first part of subtest 200 posts a few threads in CIRs 0-3, 16, 24, and 31, both `spawn` and `pfork` types. The subtest repeatedly reads the posted CIR (PCIR) without accepting the threads to allow them to circulate according to the rotating priority scheme. Different PIDs make the requests but should not affect the operation of fork posting.

The second part of subtest 200 checks that all the interlocks work for the communication registers used by the ASAP logic. The subtest uses the following algorithm:

1. Issues a `snd_c` request to the thread allocation mask communication register, offset = 0x7, CIR = 0.
2. Issues a second `snd_c` request to the thread allocation mask communication register.
3. Verifies that the first request worked and that the second did not.
4. Repeats Steps 1 through 3 for the fork_posted communication register (offset = 0xb).
5. Repeats Steps 1 through 3 for CIRs 0, 1, 2, 4, 8, 16, 17, 18, 20, 24, and 31.
6. Reads communication registers 0x7 and 0xb from the CIRs that were not written and verifies that they are 0.
7. Performs a `rcv_c` request to communication registers 0x7 and 0xb for all CIRs that were written.
8. Performs a second `rcv_c` request to communication registers 0x7 and 0xb for all CIRs that were written.
9. Verifies that the first request worked (verify data) and that the second did not.

The third part of subtest 200 checks the functioning of the CPU mask part of the thread allocation mask communication register. A request from a CPU for the PCIR should only get threads for which it is enabled. The test uses the following algorithm:

1. Posts a thread in CIR 0, masking all even CPUs.
2. Posts a thread in CIR 1, masking all odd CPUs.
3. Posts a thread in CIR 2, masking all even CPUs.
4. Posts a thread in CIR 3, masking all odd CPUs.
5. Reads the PCIR from various CPUs and verifies the correct CIR after accounting for the priority rotation.

6.7.3.3 CU RDCMR/WRCMR subtest (210)

Subtest 210 verifies the functionality of the rdcmr and wrcmr operations. It uses the following algorithm:

1. Writes the LCKB register in control space.
2. Restores the state, including the lock bit, of some ASAP registers (0x7 and 0xb in CIRs 0/3/8/16/24/31) using the wrcmr operation.
3. Reads all registers (get_c operation) and verifies the contents.
4. Tests the lock bits of all registers and verifies that they are correct.
5. Reads all registers (rcv_c operation) and verifies their contents and status. Some will fail due to the lock bit not being set.
6. Uses put_c, put_s, and lck_c operations to initialize these registers (contents and lock bits) different from above.
7. Performs the rdcmr operations on these registers and verifies the data.
8. Reads and verifies the LCKB register in control space.

6.7.3.4 TOC AND ITC subtest (220)

Subtest 220 verifies the basic functionality of the time-of-century (TOC) and interval timer (ITC) counters. It performs put and get operations to both counters and also enables the microsecond ticker to allow these counters to increment and decrement, respectively.

Subtest 220 also increments the TOC for 100 clocks and checks that the TOC has incremented the correct number of times. To do this, the subtest determines the number of microseconds that should have elapsed in 100 clocks. Figure 6-11 shows the error message that occurs if the TOC fails to increment correctly.

Figure 6-11
cu4000 subtest 220 failure—incorrect TOC error display

```
''cu4000 Subtest 220 Failure - TOC register''

Type of error detected

Expected Lower 32-bit of TOC = ffffffff
Actual Lower 32-bit of TOC   = ffffffff
```

Table 6-4 lists the clock values in each part of subtest 220, for single stepping the clock during troubleshooting.

Table 6-4
cu4000 subtest 220 TOC clock values

Function	Clocks
TOC operation	0-74
Microsecond operation	75-80
ITC operation	81-263

6.7.3.5 Deadlock and firmware trap subtest (230)

Subtest 230 verifies the detection of a trap and the generation of the appropriate signals by the CU board. All firmware traps are generated by writing the TRPCMD register in control register space. The processing of the trap is also verified via scan.

Subtest 230 tests all firmware traps by writing the TRPCMD register with the appropriate type, vector, and purge address. After clocking the CU board, the subtest verifies that the CU board dispatched the trap to all other installed CPUs with the correct type and vector or purge address. The subtest uses several CPU installed configurations to verify the CPU destination logic.

Subtest 230 tests the following firmware traps:

- ctrsg—Updates the CPU timer registers in the entire system.
- pate—Purges PTE cache entry.
- patu—Purges PTE cache.
- pmod—Purges PTE cache modified bits.
- pref—Purges PTE cache referenced bits.
- trap—Sets up a processor trap on each CPU sequentially and checks the CU for proper detection and handling.
- xmti—Transmits interrupts.

Subtest 230 causes different CPUs to use several CIRs to test deadlock. It forces a deadlock for all CPUs sharing a single CIR by writing the scan ring field nadr.dlck with the appropriate CPU mask. After a delay of 32 clocks, the subtest verifies that the CU board dispatches the deadlock trap type and vector correctly to all affected CPUs.

In addition to the 2 error displays described above, subtest 230 uses the error display of Figure 6-12 when microcode complete (MT) does not occur when expected. It uses the error display of Figure 6-13 whenever it detects one or more of the following errors in the CU generated trap:

- The trap is the wrong type.
- The vector is incorrect.
- The destination of the trap is wrong (CPUs to get the trap are incorrect).

Figure 6-12
Subtest 230—microcode trap not complete error display

```
Name of error detected

Clock = 0000

Expected mt_comp ("mt_comp")= 1
Actual mt_comp              = 1
Expected PID ("nadr.tpid")  = 08
Actual PID                  = 08
```

The fields in Figure 6-12 contain the following information:

- The first line displays the descriptive name of the error.
- `Clock` = used for development purposes only.
- `Expected mt_comp` = is a bit flag giving the expected microtrap complete status.
- `Actual mt_comp` = is a bit flag giving the actual microtrap complete status.
- `Expected PID` = and `Actual PID` = give the identification number of the CPU to be exercised, and actually exercised, respectively, in the test. ID numbers 0-7 refer to CPUs 0-7. ID number 8 refers to the interface adapter (IA).

Figure 6-13
Subtest 230—incorrect trap found error display

```
Name of error detected

Clock = 0000

Expected ready ("nadr.trp_rdy")       = 1ff
Actual ready                          = 1ff
Expected type ("cu_xc_trap_type_3_0")= 0f
Actual type                           = 0f
Expected vector ("nadr.trapvec")      = fff
Actual vector                         = fff
```

The fields in Figure 6-13 contain the following information:

- The first line displays the descriptive name of the error.
- `Clock` = used for development purposes only.
- `Expected ready` = is a 9-bit field displayed in hex, giving the expected trap ready status (CPU mask).
- `Actual ready` = is a 9-bit field displayed in hex, giving the actual trap ready status (CPU mask).
- `Expected type` = and `Actual type` = give the identification number of the expected and actual type of trap, respectively, generated in the test.
- `Expected vector` = and `Actual vector` = give the identification number of the expected and actual trap vector, respectively, resulting from the trap generated.

6.7.3.6 Interrupt processing subtest (300)

Subtest 300 verifies the interrupt processing logic on the CU board. It does all initialization and verification via scan, including setting up the ION bit as well as the interrupt registers:

- Broadcast enable (BE)
- Local enable (LE)
- Global enable (GE)

Testing is in two sections, verifying interrupt processing by the CU and the interval timer counter (ITC).

Interrupt processing by CU

The first section verifies the correct processing of interrupts by the CU board. It sets up the interrupt registers to allow interrupts on all channels to be sent to various CPUs. An xmti trap is set up and the CU board is clocked. It then verifies that the xmti trap was sent correctly (type and vector) to the correct CPUs. It tests the following functions:

1. Verifies operation of the LE register, that an interrupt sent to a CPU is correctly dependent on state of LE.
2. Verifies operation of the BE register, that an interrupt sent to a CPU is correctly dependent on state of BE.
3. Verifies operation of the GE register, that an interrupt is ignored if disabled.
4. Verifies operation of the DSI and ENI to disable/enable interrupts.

Interval timer interrupts

The second section verifies the operation of the interval timer and ITC interrupts. It tests the following functions:

- Full bit is set when timer reaches 0 and is cleared when ITSR is read.
- Overflow bit is set when timer reaches 0 and the full bit is set.
- That an xmti trap was generated correctly to the specified CPUs.

Memory subsystem test (mem4000)

7

This diagnostic test provides a functional test of the memory subsystem. The memory subsystem consists of the following hardware:

- Crossbar (XBAR)
- Memory boards (MB)
- NXP path through the interface adapter (IA) board
- Memory test logic (MTL)

7.1 Prerequisites

The following conditions must be met before running `mem4000`:

- A C3800 Series SPU with SWIS and SWIP boards must be powered and booted, with the SPU OS prompt displayed.
- The SPU self-test must have completed without significant error.
- The `spu4000` diagnostic test must have completed without error.
- System must be powered on all CPU, CU, IA, MB, and crossbar boards to be used during testing.
- SPU-based subtests require a working interface adapter (IA) board and a working CPU utilities (CU) board.
- CPU-based subtests assume proper basic operation of the CPU—diagnostic test `cpu4030` runs.

All subtests assume an unknown state for all boards and initialize the boards appropriately.

7.2 Hardware requirements

A minimum hardware configuration is required to support execution of mem4000. This minimum configuration consists of a service processor unit (SPU), a CPU utilities board (CU), and an XCL board.

Different classes of subtests require different amounts of hardware to run. These requirements are detailed in Table 7-1.

Table 7-1
Hardware required for mem4000

Class	Description	Minimum hardware
1	Scan-based crossbar only tests	One XS0, XS1, and XRT
2	Scan-based memory only tests	One MB
3	Scan-based crossbar/memory subtests	A crossbar half and 1 MB
4	Functional SPU-based subtest	Complete crossbar, IA in IA8 slot, one MB
5	Functional CPU-based subtests	Complete crossbar, IA in IA8 slot, one MB, one CPU

Figure 7-1 shows the parts of the system under test and indicates the field replaceable units (FRUs) required to run the class 1 subtests of mem4000.

Figure 7-2 shows the parts of the system under test and indicates the FRUs required to run the class 2 subtests of mem4000.

Figure 7-3 shows the parts of the system under test and indicates the FRUs required to run the class 3 subtests of mem4000.

Figure 7-4 shows the parts of the system under test and indicates the FRUs required to run the class 4 subtests of mem4000.

Figure 7-5 shows the parts of the system under test and indicates the FRUs required to run the class 5 subtests of mem4000.

Figure 7-1
FRUs required and exercised by mem4000 class 1 subtests

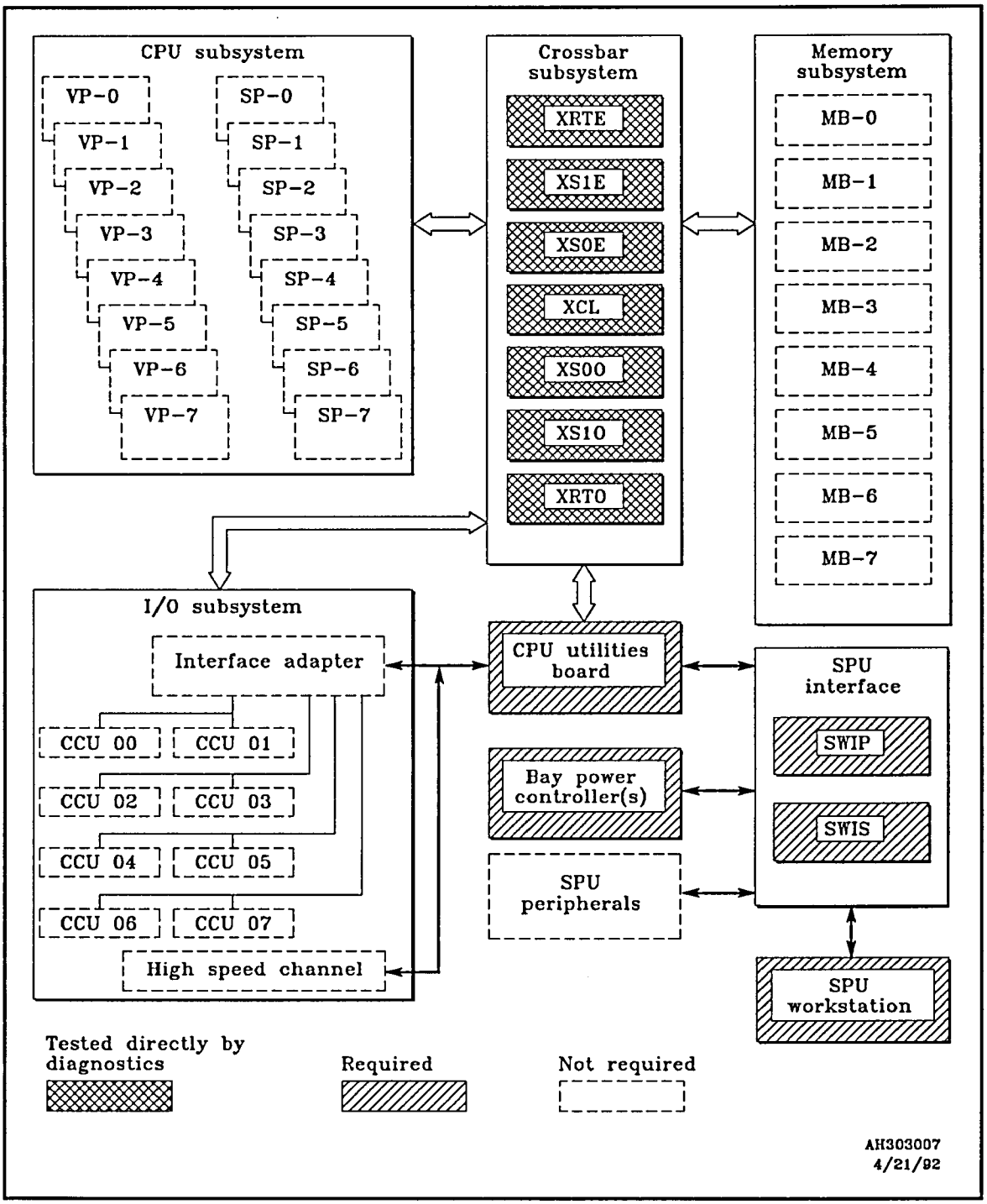


Figure 7-2
FRUs required and exercised by `mem4000` class 2 subtests

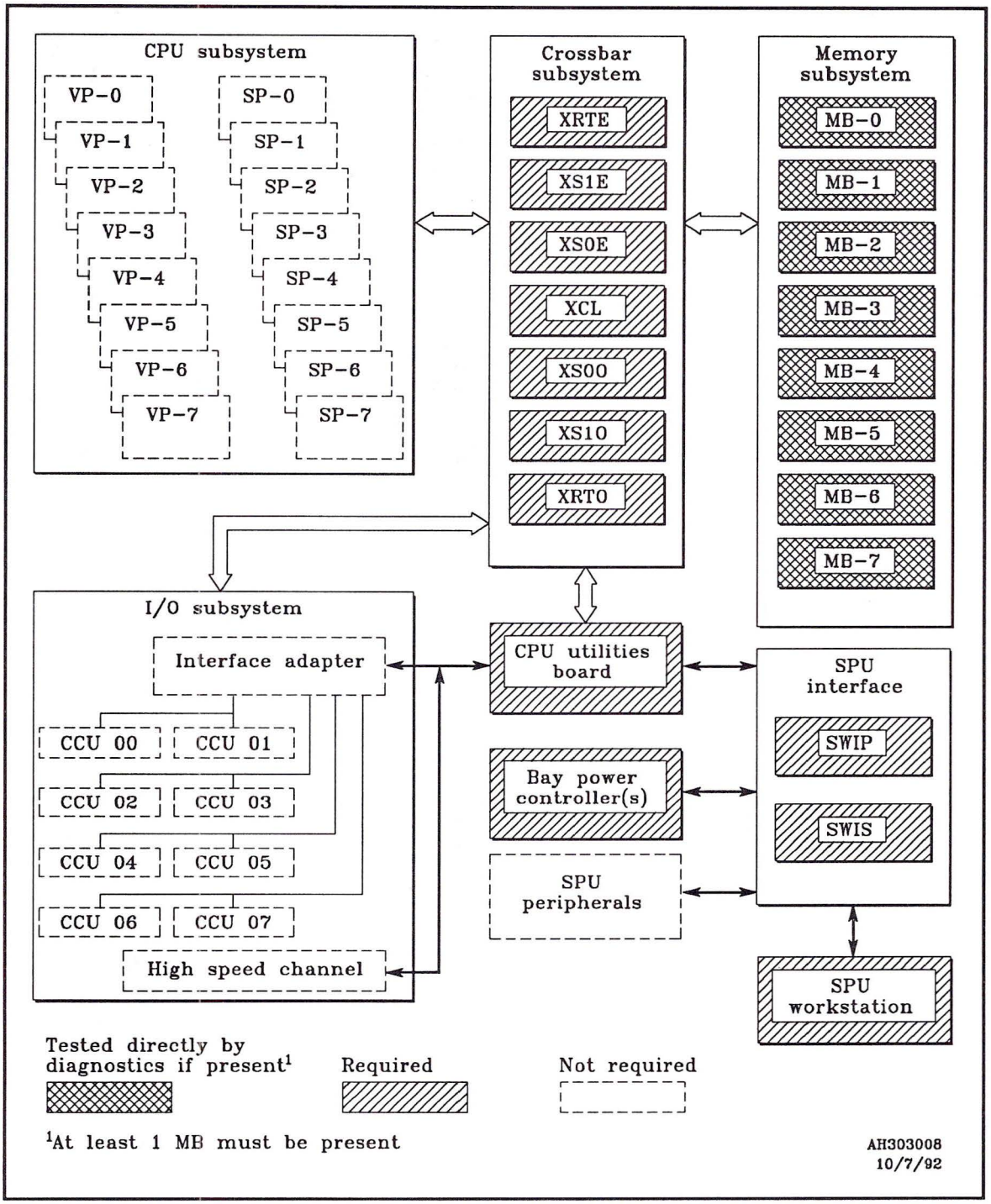


Figure 7-3
FRUs required and exercised by mem4000 class 3 subtests

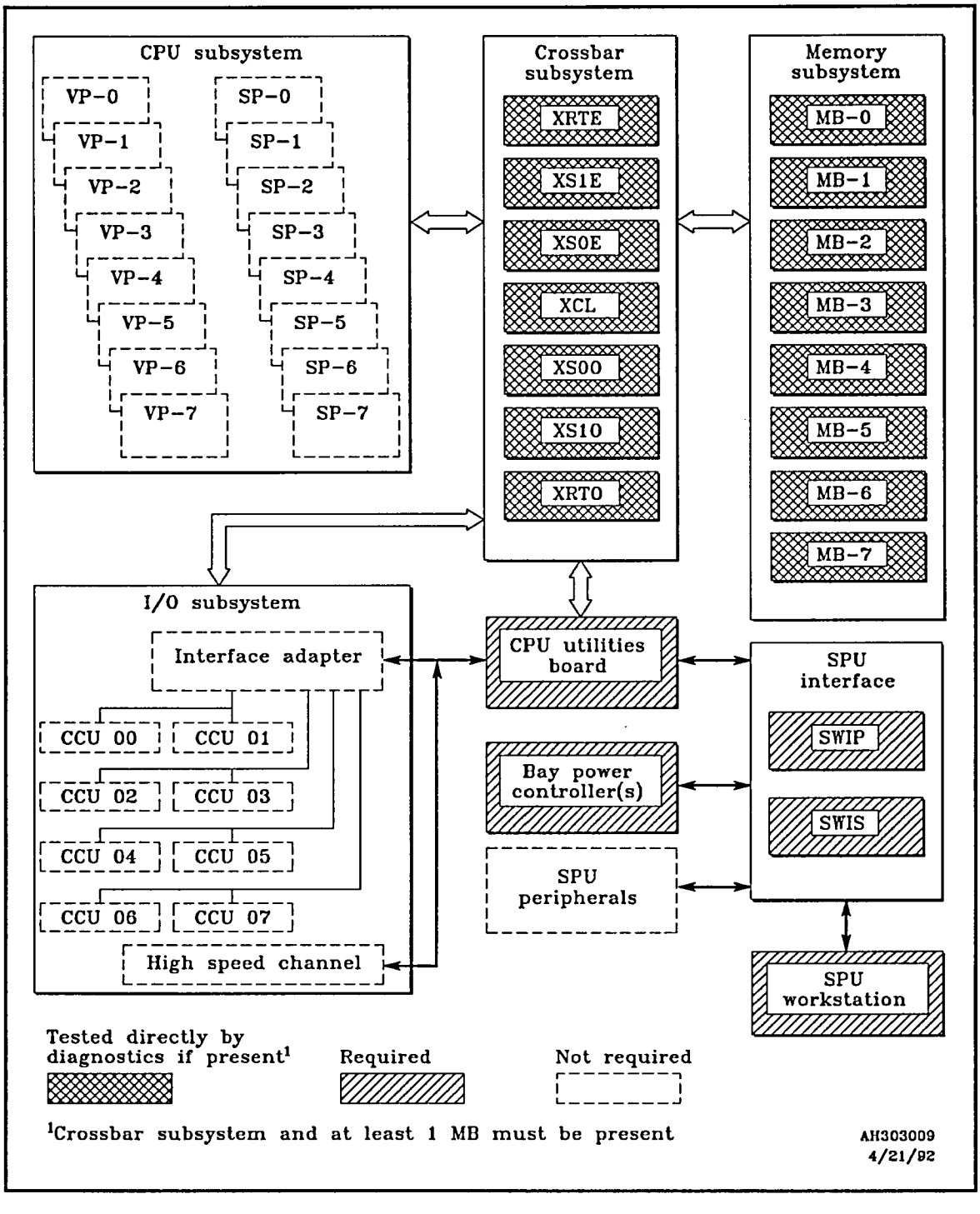


Figure 7-4
FRUs required and exercised by `mem4000` class 4 subtests

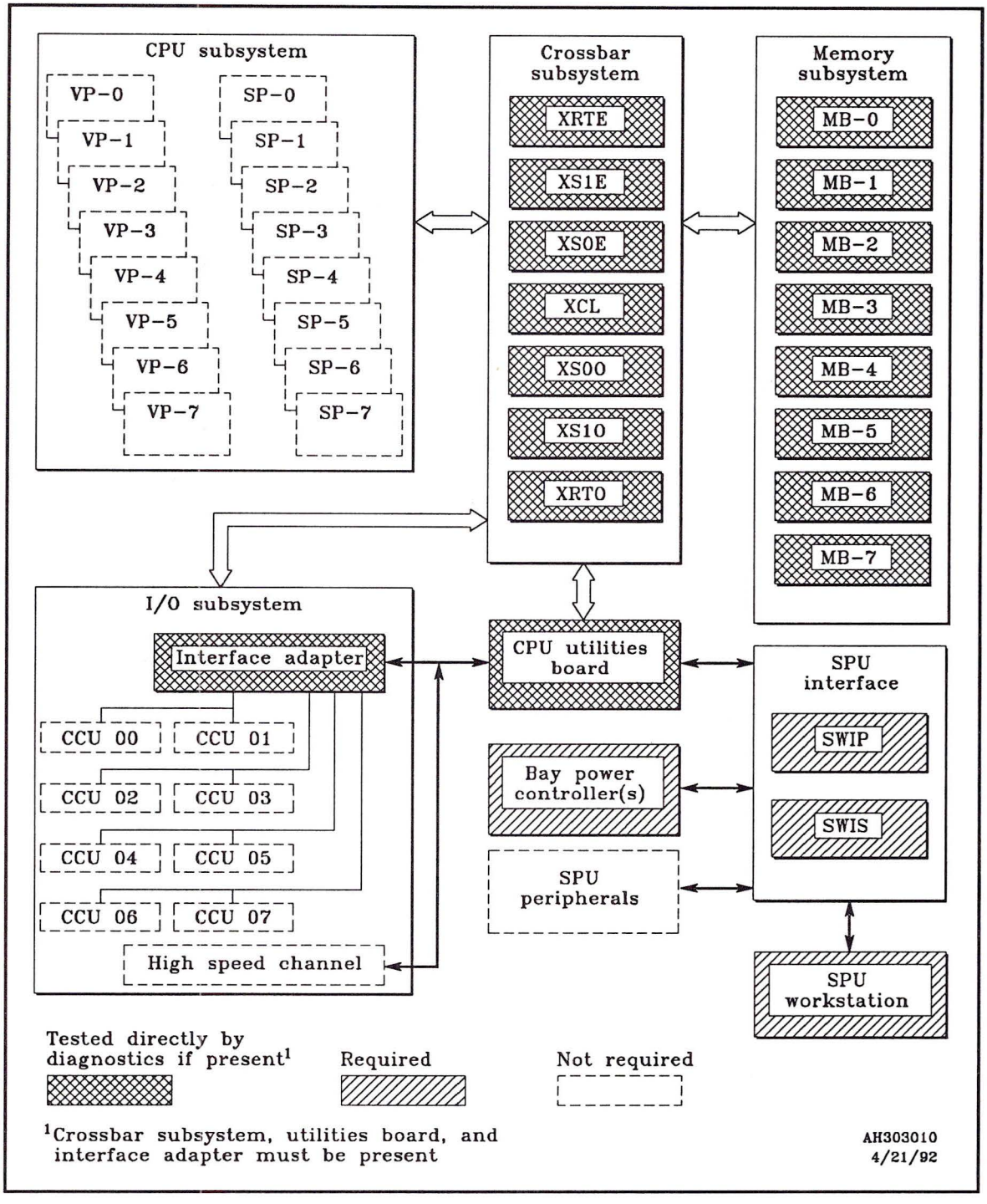
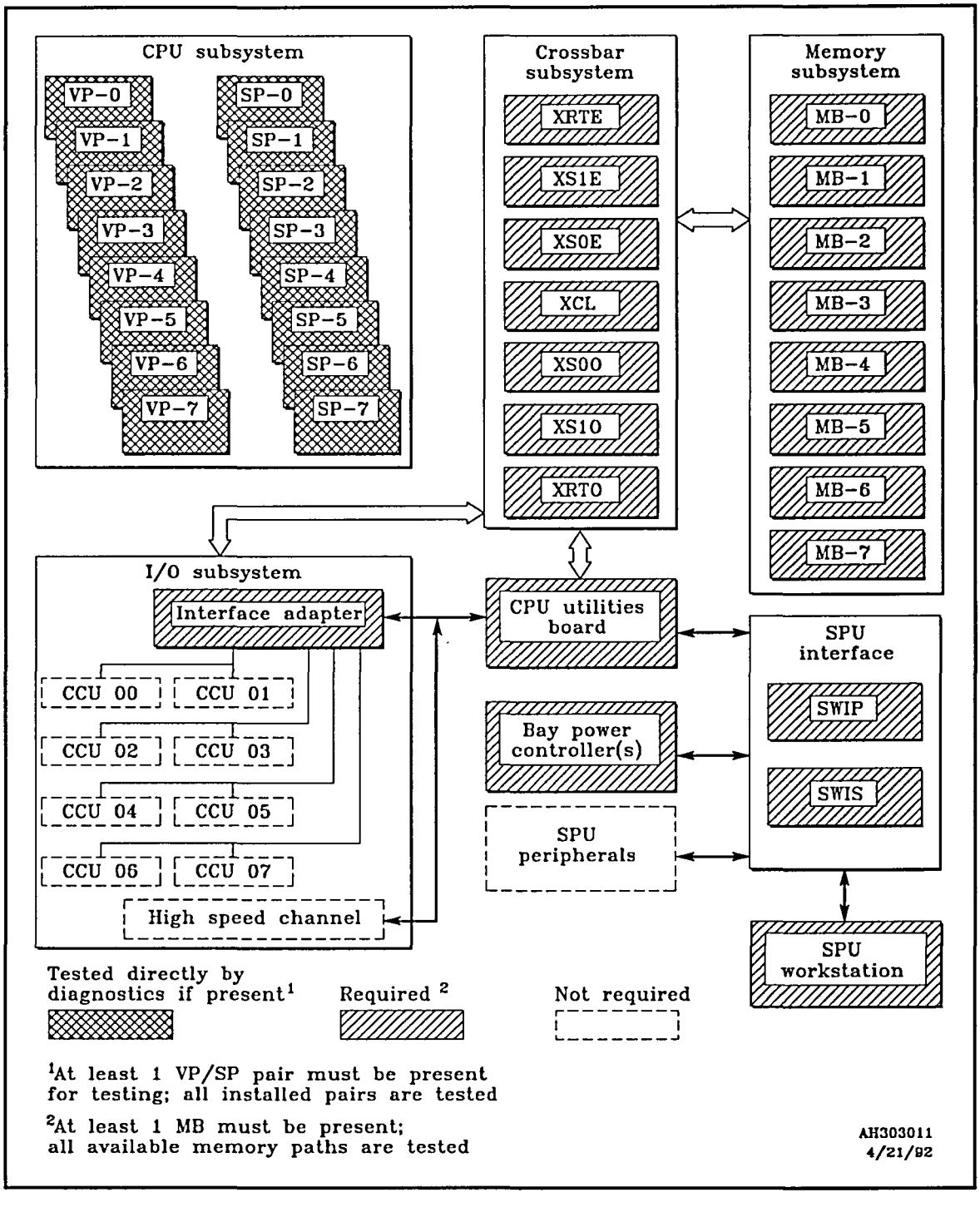


Figure 7-5
FRUs required and exercised by mem4000 class 5 subtests



7.3 Invoking the test

Use the following command to enter the xdiag environment:

```
(spu)> xdiag
```

xdiag is the X-based CONVEX test interface (CTI) used by all C3 processor diagnostics. Refer to Chapter 4 for a description of xdiag.

From the main xdiag window, select Test and click on mem4000 in the drop-down menu. This causes the CTI to load the appropriate diagnostic files. The term TestParameters appears in the xdiag window menu bar.

To run the test, click on run in the xdiag window command bar. The test will run with default parameters unless you have customized the test options.

The following subsections describe the customizing process.

7.4 Test menus

The mem4000 test has 3 available windows under the xdiag test control system. These windows enable you to select and control the test-specific parameters. Except where indicated, the darkened boxes are the default values of the menu parameters.

7.4.1 TestParameters menu window

Clicking on the TestParameters item in the xdiag window menu bar causes the xdiag TestParameters menu window to appear. Figure 7-6 shows the xdiag_TestParamters window.

The menu bar at the top of the window allows you to select two submenu windows and to force a resize of memory. Click on the Force Resize of Memory box to cause the test to resize the memory after you replace a memory board with another of a different configuration.

The following paragraphs describe the mem4000 test parameters.

7.4.1.1 CPUs to use during testing:

This parameter applies only to subtest class 5. Click on the boxes to select or deselect CPUs you wish to use, or not to use, in the current test series. The default is to use all available CPUs.

7.4.1.2 Memory boards to test:

Click on the boxes to select or deselect memory boards you wish to test, or not to test, in the current test series. The default is to test all available memory boards.

7.4.1.3 Disable Error Checking?

Click on yes to inhibit error checking by all boards during all subtests.

7.4.1.4 Disable Interleaving?

This parameter applies only to subtest classes 4 and 5. Click on yes to disable interleaving by the IA and SP boards.

7.4.1.5 XS0 Standalone Testing:

This parameter applies only to subtest class 1. Clicking on yes causes subtests 50 and 100 to run during class 1 testing.

7.4.1.6 XRT Standalone Testing:

This parameter applies only to subtest class 1. Clicking on yes causes subtests 50 and 101 to run during class 1 testing.

7.4.1.7 Direction of st_200 testing:

This parameter applies only to subtest 200. You can test either the path from the memory board to the memory card (MC), or the path in the opposite direction, or both paths.

7.4.1.8 Make PBUS Read Requests?

This parameter applies only to subtests 306 and 450. During memory restart, enables or disables checking read requests from the SPU.

7.4.1.9 Burst Count Mask:

This parameter applies only to subtest 304. Specify the burst count mask in hexadecimal form. Refer to page 7-48 for a description of the mask code.

7.4.1.10 Error Threshold:

Specify the decimal number of allowable errors before subtest failure is declared.

7.4.1.11 Wait time for refresh testing (seconds):

This parameter applies only to subtests 350, 375, and 385. In decimal form, specify the time in seconds between writing and reading memory.

7.4.1.12 Use CPUs to Test Memory?

This parameter enables or disables subtest class 5. Click on yes to test the full paths between CPUs and memory. Click on no to test only the paths between the crossbar and the memory.

7.4.1.13 Starting SPU physical address:

This parameter applies only to subtest class 4. Specify the starting address in hexadecimal form.

7.4.1.14 Ending SPU physical address:

This parameter applies only to subtest class 4. Specify the ending address in hexadecimal form.

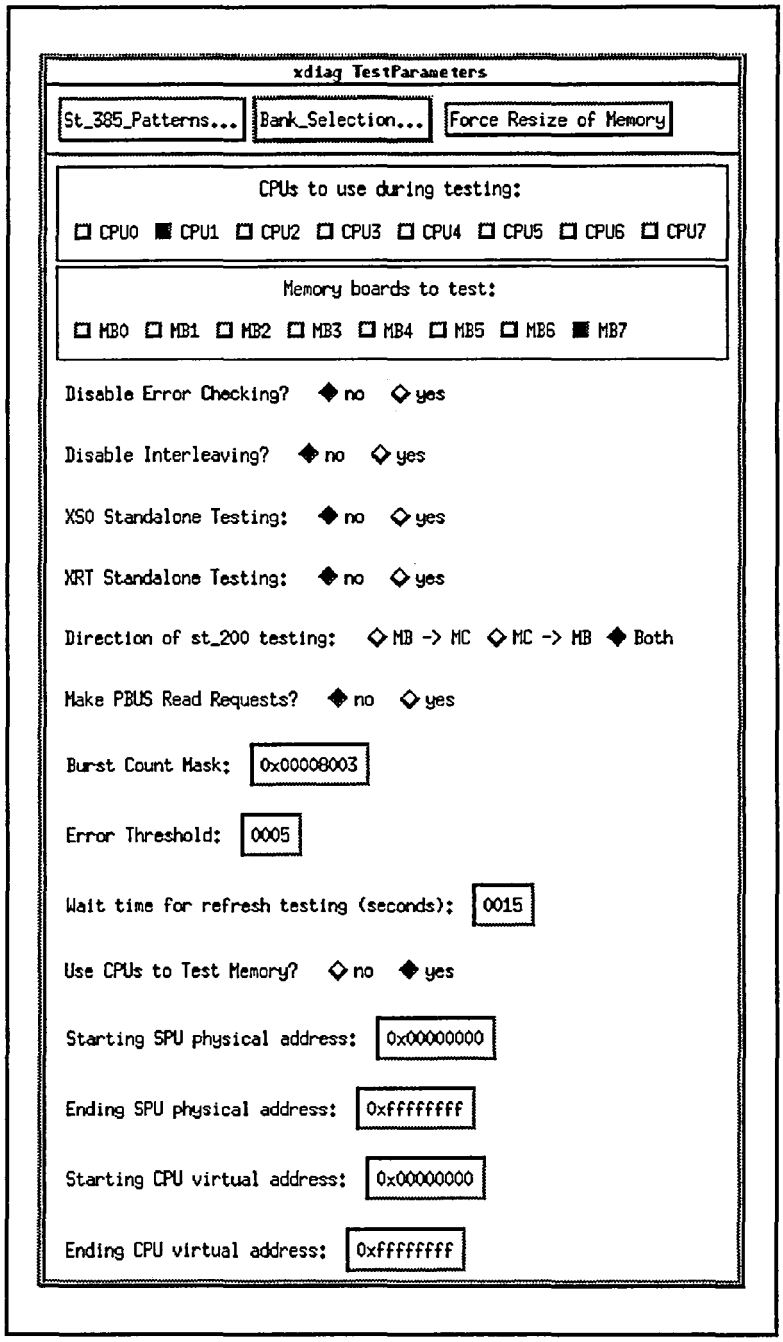
7.4.1.15 Starting CPU virtual address:

This parameter applies only to subtest class 5. Specify the starting address in hexadecimal form.

7.4.1.16 Ending CPU virtual address:

This parameter applies only to subtest class 5. Specify the ending address in hexadecimal form.

Figure 7-6
`xdiag_TestParameters` window



7.4.2 Bank_Selection window

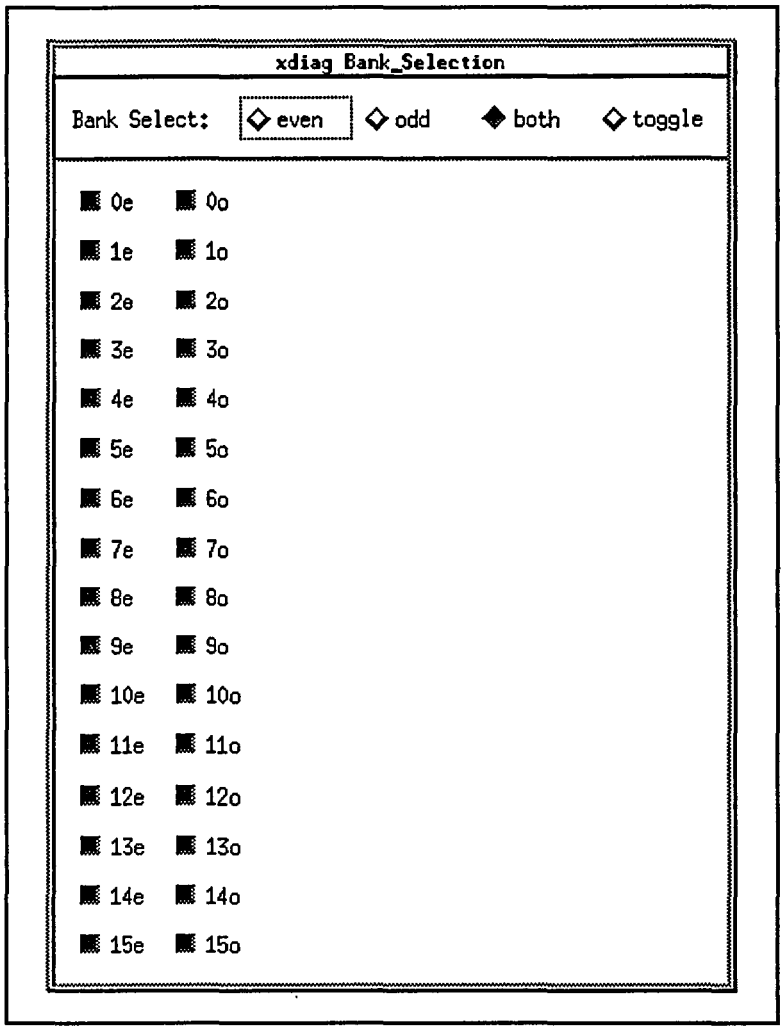
Clicking on the Bank_Selection box in the xdiag window menu bar causes the xdiag Bank_Selection window to appear. Figure 7-7 shows the xdiag Bank_Selection window.

The Bank Select: bar at the top of the window allows you to select either the entire even memory bank or the entire odd memory bank, or both. Clicking the toggle button in the menu bar disables all enabled memory banks and enables all disabled banks.

Click on individual memory bank buttons to select or deselect the corresponding memory bank.

The selection and deselection applies equally to all memory boards selected for the test.

Figure 7-7
`xdiag_Bank_Selection` window



7.4.3 St_385_Patterns menu window

Subtest 385 tests memory retention using specified test patterns. Clicking on the St_385_Patterns box in the xdiag window menu bar causes the xdiag St_385_Patterns menu window to appear. Figure 7-8 shows the St_385_Patterns window.

The window shows the available test patterns as well as a separate Special data pattern box in which you can specify any arbitrary 32-bit test pattern. The default condition is to test all patterns.

Click on a pattern button to deselect or select the corresponding standard pattern for testing. Enter any pattern in the Special data pattern box near the bottom of the window and click the Special pattern (hold) button. The subtest uses the Special data pattern, if any, last.

Use the Toggle Pattern Select button at the top of the window to select all deselected patterns and to deselect all selected patterns.

7.4.3.1 User Data Pattern Op:

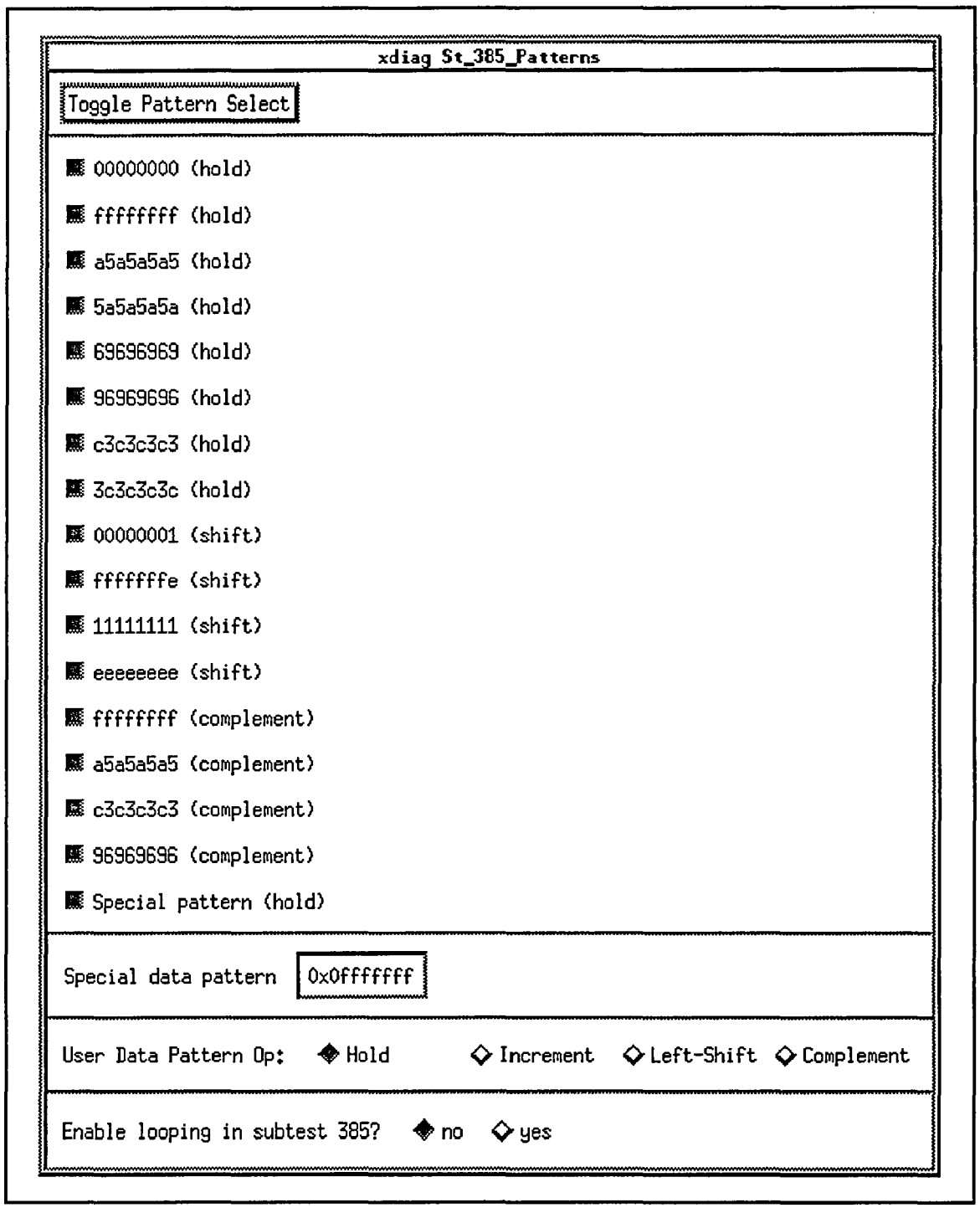
Other options are available for the special data pattern. Click on the desired option. The options have the following functions:

- **Hold**—Write the special data pattern into every word over the specified memory address range.
- **Increment**—Write or read 64 bits (two adjacent memory words, one even and one odd) and increment the pattern by 1. Repeat over the specified memory address range.
- **Left-Shift**—Write or read 64 bits (two adjacent memory words, one even and one odd) and shift the pattern one bit to the left. Repeat over the specified memory address range.
- **Complement**—Write or read 64 bits (two adjacent memory words, one even and one odd) and change the special data pattern to its complement (all 0s become 1s; all 1s become 0s). Repeat over the specified memory address range.

7.4.3.2 Enable looping in subtest 385?

Clicking yes for this item causes this subtest to continuously loop internally until you stop the test and click no.

Figure 7-8
`xdiag_st_385_Patterns` window



7.5 When the X-Window environment is not available

If you are not working in an X-Window environment, refer to Section 4.13 on page 4-33 for information on invoking the diagnostics software.

To invoke `mem4000`, enter:

```
spu> mem4000
STARTUP>
```

To display test specific parameters, enter:

```
STARTUP> test_par
```

Figure 7-9 shows the test-specific parameters for the `mem4000` test, with their default values.

Figure 7-9
`mem4000` test-specific parameters

```
- "Banks" "1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1"
- "Burst_count_mask" "0x00008003"
- "Cpu_boards" "0,0,0,0,0,0,0,0"
- "Cpu_virt_end" "0xffffffff"
- "Cpu_virt_start" "0x00000000"
- "Disable_errors" "0"
- "Disable_interleave" "0"
- "Error_threshold" "5"
- "Memory_boards" "0,0,0,0,0,0,0,0"
- "Odd_even_select" "3"
- "PBUS_read" "0"
- "Quiet_mode" "0"
- "Spu_phys_end" "0xffffffff"
- "Spu_phys_start" "0x00000000"
- "St200_testing" "0x00000003"
- "St_385_loop" "0"
- "St_385_patterns" "1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1"
- "St_385_user_op" "0"
- "St_385_user_patt" "0x0fffffff"
- "Use_cpu" "1"
- "Wait_time" "0x0000000f"
- "XRT_standalone" "0"
- "XS0_standalone" "0"
```

These are the parameter definitions:

- `Banks`—Numbering from the left, test the bank if `1`; do not test if `0`.
- `Burst_count_mask`—This parameter applies only to subtest `304`. Specify the burst count mask in hexadecimal form. Refer to page 7-48 for a description of the mask code.
- `Cpu_boards`—This parameter applies only to subtest class 5. Numbering from the left, use the CPU in testing if `1`; do not use if `0`.
- `Cpu_virt_end`—This parameter applies only to subtest class 5. Specify the virtual address of the upper bound of the memory range to be tested in hexadecimal form.
- `Cpu_virt_start`—This parameter applies only to subtest class 5. Specify the virtual address of the lower bound of the memory range to be tested in hexadecimal form.
- `Disable_errors`—Inhibit error checking by all boards during all subtests if `1`; do not inhibit if `0`.
- `Disable_interleave`—This parameter applies only to subtest classes 4 and 5. Disable interleaving by the IA and SP boards if `1`; do not disable interleaving if `0`.
- `Error_threshold`—Specify the hexadecimal number of allowable errors before subtest failure is declared.
- `Memory_boards`—Numbering from the left, test the memory board if `1`; do not test if `0`.
- `Odd_even_select`—Select all odd memory banks for testing if `1`; select all even banks if `2`; select all banks if `3`.
- `PBUS_read`—This parameter applies only to subtests `306` and `450`. During memory restart, enables checking read requests from the SPU if `1`; disables if `0`.
- `Quiet_mode`—Inhibits all test progress messages if 1. Default is 0.
- `Spu_phys_end`—This parameter applies only to subtest class 4. Specify the physical address of the upper bound of service processor memory testing in hexadecimal form.
- `Spu_phys_start`—This parameter applies only to subtest class 4. Specify the physical address of the lower bound of service processor memory testing in hexadecimal form.

- `St200_testing`—This parameter applies only to subtest `200`. Test the path from the memory board to the memory card (MC) if `1`; test the path in the opposite direction if `2`; test both paths if `3`.
- `St_385_loop`—Continuously loop through subtest `385` if `1`; make only one pass through the test if `0`.
- `St_385_patterns`—Select or deselect word patterns to write into and read from memory. Figure 7-8, page 7-15, gives the list of available patterns. The leftmost bit in this parameter corresponds to the topmost pattern in Figure 7-8. Use a pattern if the corresponding bit is `1`; do not use if `0`.
- `St_385_user_op`—Hold if `0`, increment if `1`, left-shift if `2`, complement if `3`. Refer to section 7.4.3.1 on page 7-14 for definitions of hold, increment, left-shift, and complement.
- `St_385_user_patt`—Specify any arbitrary test pattern using a hexadecimal number.
- `Use_cpu`—If `1`, enable class 5 subtests. If `0`, disable class 5 subtests.
- `Wait_time`—This parameter applies only to subtests `350`, `375`, and `385`. In hexadecimal form, specify the time in seconds between writing and reading memory.
- `XRT_standalone`—This parameter applies only to subtest class 1. Subtests `50` and `101` run during class 1 testing if `1`; these subtests do not run if `0`.
- `XS0_standalone`—This parameter applies only to subtest class 1. Subtests `50` and `100` run during class 1 testing if `1`; these subtests do not run if `0`.

7.6 Class descriptions

Class 1 through class 5 subtests perform the following verifications:

- **Class 1**—Scan-based crossbar subtests verify only the crossbar using scan operations for setup and data checking.
- **Class 2**—Scan-based MB subtests verify the MB and MC using only scan operations for setup and data checking. The crossbar is neither tested nor required.
- **Class 3**—Scan-based memory subtests verify both the crossbar and the crossbar-MB operations using only scan operations for setup and data checking.
- **Class 4**—SPU-based memory subtests verify the functionality of the memory subsystem (crossbar-MB-MC) using both the NXP and the MTL interfaces to write and read. Scan operations are used for initialization only.
- **Class 5**—CPU-based memory subtests verify the functionality of the memory subsystem (crossbar-MB-MC) from the CPU.

7.6.1 Subtest description

Table 7-2 and Table 7-3 describe the subtests. Execution time applies to a single MB (scan-based subtests) or 256 Mbytes of memory (functional tests).

Table 7-2
Scan-based subtests

Subtest	Class	Description	Execution time (seconds)[1]
50	1	Scan ring verification	2
100	1	XS0 standalone	1
101	1	XRT standalone	2
102	1	Overrun registers	5
103	1	Processor enable	7
104	1	PCM testing	58
105	1	Commreg path	16
200	2	MB—MC data path testing	22
201	2	PCM testing	58
202	2	Parity testing	37
203	2	ECC testing on RDEDC gate arrays	44
204	2	ECC testing on WREDC gate arrays	37
205	2	Single step testing	250
206	2	WREDC parity testing	14
207	2	MB to MC signal testing	4
300	3	Crossbar arbitration	130
301	3	Request processing	111
302	3	Parity testing	10
303	3	TAC/TAS operations	96
304	3	Burst mode	5
305	3	Odd/even	2
306	3	Memory restart	3

[1] Execution time is given for a single MB. Times are linear. Multiply time by number of MBs installed.

Table 7-2 (continued)
Scan-based subtests

Subtest	Class	Description	Execution time (seconds)[1]
307	3	Commreg/memory contention	5
350	4	Read, modify, and write operations via NXP transfers	4
355	4	TAC operations via NXP transfers	3
360	4	TAS operations via NXP transfers	2
365	4	SCRUB operations via NXP transfers	2
366	4	Verifies MB log ring	10
370	4	Page addressing via NXP transfers	2
375	4	Data retention via NXP transfers	24
380	4	Addressing pattern via MTL transfers	7
385	4	Data retention via MTL transfers	100

[1] Execution time is given for a single MB. Times are linear. Multiply time by number of MBs installed.

The first CPU-based subtest loads the CPU command processor into memory, adding about 12 seconds to its execution time. The command processor does not reload for further class 5 subtests. The command processor reloads if a subtest from class 1 through class 4 is executed.

Table 7-3
CPU-based subtests

Subtest	Class	Description	Execution time[1]
400	5	MATS+ testing	202 seconds
410	5	NTA testing	27 minutes
420	5	Bank uniqueness	4 seconds
430	5	Data length and alignment	76 seconds
440	5	TAC instructions	194 seconds
445	5	TAS instructions	194 seconds
450	5	Restarting memory	49 seconds
460	5	Simultaneous SPU-CPU TAC instructions	4 seconds
500	5	R&M bits testing	3 seconds

[1]Execution time is given for a single MB. Times are linear. Multiply time by number of MBs installed.

7.6.1.1 Scan-based crossbar subtests (class 1)

The crossbar subtests use only the crossbar boards in their testing. Subtest 50 can be executed on an XS0 or an XRT board only. Subtest 100 requires only an XS0 board. Subtest 101 requires an XRT board only. All other subtests in this class require at least a complete half of the crossbar (one XS0, one XS1, and one XRT).

Subtests `50, 100, 101,` and `102` use the error display shown in Figure 7-10. It displays the failing scan ring, the scan field, a short description of the scan field, and expected and actual data.

Figure 7-10
Subtests `50, 100, 101,` and `102` error display

```
Test: mem4000 R1.0   Subtest: 50 R1.0   Class: 1
Purpose: Scan Ring Verification.

Ring: xs0_e
Scan Field: addr_reg0[0]
Overrun Register

Expected Data      00000000
Actual Data        00000002
```

Scan ring verification subtest (50)

Subtest `50` verifies the continuity and integrity of all crossbar and memory board scan rings.. This is done by using an all 0 pattern and an all 1 pattern. The subtest writes the entire scan ring and then reads and verifies all scan fields it uses.

The 2 XS0, 2 XS1, 2 XRT, and all MB boards are verified. Scan fields tested on the XS0 and XS1 boards include:

- Overrun registers (all 5 stages)
- Input stage registers
- Output stage registers
- Last staged data (LSD) registers

The only scan fields verified on the XRT boards are the four stages of return data registers.

The scan fields verified on the MB boards include:

- RDEDC output data
- Input stage data
- BCGA arrays (_ref_pend,_wr_reg_hold, data_str, ecc_check, _g)

XS0 standalone scan subtest (100)

Subtest `100` verifies the basic flow of data and parity through the different ranks of registers on the XS0 board (both odd and even sides). The send section of the crossbar keeps the last request sent from a processor port to a memory port in a look-aside LSD register. Three ranks of LSD registers exist for each memory port.

Subtest `100` uses the following procedure to perform the testing on each side of the crossbar:

1. Disables errors on the XS0 board — (allows data into the last rank of LSD registers).
2. Initializes the input stage registers—(address, cycle, memory board select, memory bank, ready signal, and valid signal) for all processor ports to zero.
3. Reads and verifies the data written into the input stage registers.
4. Clocks the XS0 board twice and verifies the output stage registers (all memory ports)
5. Single steps the clock and verifies the first rank of LSD registers (all processor ports).
6. Single steps the clock and verifies the second rank of LSD registers (all processor ports).
7. Single steps the clock and verifies the third rank of LSD registers (all processor ports).
8. Repeats the previous steps using different data patterns (all 1s, all 5s, all As).

XRT standalone scan subtest (101)

Subtest 101 verifies the basic flow of data and parity through the ranks of registers on tne XRT board (both odd and even sides). The return section of the crossbar keeps the last data returned from a memory port to the sending processor port in look-aside registers. Three ranks of these error registers exist for each processor port.

Subtest 101 exercises each side of the crossbar using the following procedure:

1. Initializes the returned data registers (both data and parity) on all memory ports to zero.
2. Reads and verifies the data written into these registers.
3. Single steps the clock three times and checks the next rank of error registers.
4. Repeats the previous steps using different data patterns (all ones, all fives, all As).

Overrun register subtest (102)

Subtest `102` verifies the operation of the overrun registers. The overrun registers hold requests when a request is blocked and another request is made before the crossbar can inform the CPU to stop sending requests.

The subtest causes a bank of memory to be busy and makes multiple requests to this same bank from a CPU. The overrun registers for this CPU are read back and verified. All CPU ports are tested with `nop`, read, write, and `tam` requests.

The subtest exercises both odd and even sides of the send crossbar. It requires only the XS0 and XS1 boards. It uses the following procedure:

1. Initializes the XS0 so all banks are busy.
2. Disables all processor ports except the one under test.
3. Sets up a `nop` request to bank 0 on MB 0 from processor port 0.
4. Clocks the crossbar and verifies the contents of the overrun registers on both the XS0 and XS1.
5. Sets up a read request to bank 0 on MB 0 from processor port 0.
6. Clocks the crossbar and verifies the contents of the overrun registers on both the XS0 and XS1.
7. Sets up a write request to bank 0 on MB 0 from processor port 0.
8. Clocks the crossbar and verifies the contents of the overrun registers on both the XS0 and XS1.
9. Sets up a `tac` request to bank 0 on MB 0 from processor port 0.
10. Clocks the crossbar and verifies the contents of the overrun registers on both the XS0 and XS1.
11. Repeats steps 1 through 10 for all processor ports.

Processor enable subtest (103)

Subtest 103 verifies the processor enable logic. When a processor is disabled, the request currently in the crossbar input stage remains: the registers are never clocked, so the contents never change. If a valid request is latched, this request is continuously processed and sent to the appropriate memory port.

Subtest 103 uses only the XS0 and XS1 boards. It tests the 9 processor ports on both the even and odd sides using the following procedure:

1. Enables only the processor port under test.
2. Performs a read request and verifies that a valid request passes to the specified memory port and that the input registers change.
3. Disables the processor port under test.
4. Performs a read request and verifies that a valid request passes to the specified memory port and the input registers remain static.

An example of the error display for this subtest is shown in Figure 7-11. It displays a comment about the particular feature, the failing scan ring, the processor port under test, the failing field, and expected and actual return data.

Figure 7-11
Subtest 103 error display

```
Test: mem4000 R1.0   Subtest: 103 R1.0   Class:1
Purpose: Scan-Based Processor Enable Testing.

Memory ready signal not generated as expected
Ring: XS0E
Processor Port: 0

Failing Field: xarb[0].m_rdy1[0]
Expected Data:  1
Actual Data:    0
```

Crossbar PCM check subtest (104)

Subest `104` verifies that requests to a missing MB causes a hard error. It tests all types of memory requests from all processor ports using the following procedure:

1. Initializes crossbar so the memory port under test is marked as missing {xarb[*].vmb[mb]= 0}.
2. Sets up a read request to that MB and verifies that the correct hard error was generated {xarb[*].pcm_err}.
3. Sets up a write request to that MB and verifies that the correct hard eror was generated {xarb[*].pcm_err}.
4. Sets up a `tac` request to that MB and verifies that the correct hard error was generated {xarb[*].pcm_err}.
5. Initializes crossbar so the memory port under test is marked as present {xarb[*].vmb[mb]= 1}.
6. Sets up a read request to that MB and verifies that no PCM error was generated {xarb[*].pcm_err}.
7. Sets up a write request to that MB and verifies that no PCM error was generated {xarb[*].pcm_err}.
8. Sets up a `tac` request to MB and verifies that no PCM error was generated {xarb[*].pcm_err}.
9. Repeats steps 1 through 8 for all processor ports.
10. Repeats steps 1 through 9 for all memory ports.

Figure 7-12 is an example of the subtest `104` error display.

Figure 7-12
Subtest `104` error display

```
Test: mem4000 R1.0    Subtest: 104 R1.0 Class: 1
Purpose: Scan-Based XBAR PCM Check Testing.

Non-existent MB not detected
Ring: XS0E
Processor port: 0

xarb[0].pcm_err: Expected: 1       Actual: 0
xarb[1].pcm_err: Expected: 1       Actual: 0
```

Communication register path subtest (105)

Subtest 105 verifies the path from the crossbar to the CU board and back. Only commreg 0 is used by this subtest. It is not intended as a comprehensive commreg verification test. The subtest exercises all processor ports using the following procedure:

1. Initializes a commreg write request on the send crossbar input stage.
2. Single steps the system until the request is completed.
3. Initializes a commreg read request on the send crossbar input stage.
4. Single steps the system until request is completed and verifies the return data on the XRT.
5. Repeats steps 1 through 4, but instead of stepping, bursts the clocks.
6. Repeats steps 1 through 5 for all processor ports.

Figure 7-13 shows a typical subtest 105 error display.

Figure 7-13
Subtest 105 error display

```
Test: mem4000 R1.0     Subtest: 105 R1.0  Class: 1
Purpose: Scan-Based Communication Register Path Testing.

CREG data incorrect (step)
CPU Port = 0
Expected Data (commreg 0)          =00000000 00000000
Actual Data (commreg 0)            =ffffffff ffffffff
```

7.6.1.2 Scan based MB subtests (class 2)

These subtests use only the MB and MC boards in their testing.

MB - MC signal pattern subtest (200)

Subtest 200 verifies the signals from the MB to the MC and from the MC to the MB. This is a continuity check of these signals. The subtest exercises all MCs on all the MBs using the following procedure:

1. Loads pattern into the MB scan ring—ise.b?e_bank_wr_data, ise.b?e_bank_wr_par.
2. Initializes the MB to send data to the MC—*._ref_pend, *.wr_reg_hold.
3. Clocks the MB to propagate data to the MC (10 system clocks).
4. Verifies the results on the MC via the scan ring—nmc*_data, nmc*_ecc.
5. Repeats for all patterns—walking 1 through all 32 bits and walking 0 through all 32 bits.
6. Loads a pattern into the MC scan ring—nmc*_data, nmc*_ecc.
7. Initializes WREDCs to write diagnostic data—nmc*_diag_data, nmc*_rerr.
8. Initializes BCGAs to drive write data into the WREDCs: *.data_str, *.ecc_check, *._g.
9. Initializes BCGAs to clock data into the RDEDCs—ldreg*.ld_en_*.
10. For each MC, selects ldreg.rselect_e and ldreg.rselect_o. Clocks MB to propagate data to the MB: 1 system clock.
11. Verifies the results on the MC via the scan ring—rdedc.sys[*].rdata, rdedc.sys[*].rtn_ecc.
12. Repeats steps 1 through 11 for all patterns—walking 1 through all 32 bits and walking 0 through all 32 bits).
13. Repeats steps 1 through 12 for all MBs.

Figure 7-14 shows a typical subtest 200 error display.

Figure 7-14
Subtest 200 error display

```
Test: mem4000 R1.0   Subtest: 200 R1.0  Class: 2
Purpose: Scan-Based MB <=> MC Signal Pattern Testing.

Ring: mb_7
MC  : 1
Scan Field: rdedc.sys[0].rdata
MB receive data incorrect

Expected Data = 00000200
Actual Data   = 00000000
```

MB PCM check subtest (201)

Subtest `201` verifies that requests to a nonexistent address on a MB is detected and results in a hard error. It exercises all MBs using read, write, and tam requests using the following procedure:

1. Initializes an MB with PCM data containing nonexistent locations: sets bc[*].all.four_meg_drams and bc[*].all.four_rows to 0.
2. Sets up a read request to a nonexistent location and verifies that a correct hard error was generated: he.hard_error, bc[0].sys._rillegal*.
3. Sets up a write request to a nonexistent location and verifies that a correct hard error was generated he.hard_error, bc[0].sys._rillegal*.
4. Sets up a `tam` request to a nonexistent location and verifies that a correct hard error was generated: he.hard_error, bc[0].sys._rillegal*.
5. Initializes an MB with PCM data so locations are marked as present: set bc[*].all.four_meg_drams and {bc[*].all.four_rows to 1.
6. Sets up a read request to the same location and verifies that no hard error was generated.
7. Sets up a write request to the same location and verifies that no hard error was generated.
8. Sets up a `tam` request to the same location and verifies that no hard error was generated.
9. Repeats steps 1 through 8 for all MBs.

Figure 7-15 shows a typical subtest 201 error display.

Figure 7-15
Subtest 201 error display

```
Test: mem4000 R1.0   Subtest: 201 R1.0  Class: 2
Purpose: Scan-Based MB PCM Check Testing

Memory Board: 7
Failure Mode: Hard error NOT generated when expected, bank 01

Scan Field                    Expected     Actual
==========                    ========     ========
he.hard_error                    1            1
bc[6].sys._rillegal22            0            0
bc[6].sys._rillegal23            1            0
bc[6].sys._rillegal25            1            1
```

MB parity subtest (202)

Subtest 202 verifies the parity checking on the input stage of the memory boards. The subtest sends request types (nop, read, write, tac, tas) to all banks on all memory boards tested, using the following procedure:

1. Sets up a nop request on the MB with a data parity error and verifies that it was detected.
2. Repeats using read, write, tac, and tas requests.
3. Sets up a tac request on the MB with a control parity error and verifies that it was detected.
4. Repeats steps 1 through 3 for all banks on the MB .
5. Repeats steps 1 through 4 for all MBs.

Figure 7-16 shows a typical subtest 202 error display. The display contains the following information:

- The top two lines of the figure identify the subtest and its purpose.
- Memory Board:—Identifies the failing memory board.
- Memory Bank:—Identifies the failing field in the scan ring.
- Physical Addr:—Gives the physical address of the failing data.
- Memory Operation:—Gives the memory operation that failed.
- Parity Operation:—Gives the type of parity being tested:
  - Data
  - Control
  - None
- Failure Mode:—Describes the type of failure.
- Field Name:—The name of the failing data field.
- Expected Data =—Displays the data written to the scan field.
- Actual Data =—Displays the data read back from the scan field.

Figure 7-16
Subtest 202 error display

```
Test: mem4000 R1.0   Subtest: 202 R1.0  Class: 2
Purpose: Scan-Based MB Parity Testing

Memory Board:      0
Memory Bank:       0

Physical Addr:     00000000
Memory Operation:  READ
Parity Operation:  Data
Failure Mode:      data parity not detected

Field Name: he.hard_error
Expected Data = 1
Actual Data   = 0
```

MB error detection/correction subtest—RDEDC arrays (203)

Subtest 203 verifies the EDC circuitry of the RDEDC gate arrays on the memory boards. The subtest tests the ability to detect single and multiple bit errors and to correct single bit errors on the read path.

The subtest makes all memory requests via scan. It puts the request into the output stage of the send crossbar (XS0/XS1). It issues a single system clock to get the request into the input stage of the MB. It then invalidates the request on the output stage of the send crossbar by setting xbar[*].m_rdy1[*] signals to 0. It uses the following procedure to test all banks on all memory boards:

1. Writes a location with a single ECC error.
2. Sets up a read request to the same location and verifies that a single-bit error was detected and the data corrected.
3. Writes a location with a double ECC error.
4. Sets up a read request to the same location and verifies that 2-bit errors were detected and that a hard error was generated.
5. Repeats steps 1 through 4 for all banks on the MB.
6. Repeats steps 1 through 5 for all MBs.

Figure 7-17 shows a typical subtest 203 error display.

MB error detection/correction subtest - WREDC arrays (204)

Subtest 204 verifies the EDC circuitry on the memory boards. It tests the ability to detect and correct single-bit errors and to detect double-bit errors on the write paths. It uses the following procedure to test all banks on all memory boards.

The subtest makes all memory requests via scan. It puts each request into the output stage of the send crossbar (XS0/XS1). It issues a single system clock to get the request into the input stage of the MB. It then invalidates the request on the output stage of the send crossbar: xbar[*].m_rdy1[*] signals are set to 0. It uses the following procedure:

1. Writes the first memory location on the bank with a single ECC error.
2. Sets up a `tac` request to the same location (byte 0) and verifies that a single bit error was detected by the WREDC and the data corrected.
3. Writes the location with a double ECC error.
4. Sets up a `tac` request to the same location (byte 0) and verifies that the WREDC detected a double error and generated a hard error.
5. Repeats steps 1 through 4 for all banks on the MB.
6. Repeats steps 1 through 5 for all MBs.

Figure 7-17 shows a typical subtest 204 error display.

Figure 7-17
Subtests 203 and 204 error display

```
Test: mem4000 R1.0   Subtest: 203 R1.0  Class: 2
Purpose: Scan-Based MB ECC Testing (RDEDC arrays).

Memory Board: 7
Memory Bank:  7

Physical Addr:          00000038
Memory Operation:       READ
ECC Operation           Normal
Failure Mode:           single bit error not corrected
```

MB single-step subtest (205)

Subtest 205 verifies the return control logic on the MB. This logic keeps a record of when data is to return from the MCs. If the system clock stops after a read request has been issued to the MCs but before the data returns, restarting the system clock should not affect the number of cycles between when the request was issued and when the data returns. The time when the data returns should always be deterministic.

The subtest does all memory requests via scan. It puts each request into the output stage of the send crossbar (XS0/XS1). It issues a single system clock to get the request into the input stage of the MB. It then invalidates the request on the output stage of the send crossbar: xbar[*].m_rdy1[*] signals are set to 0. It exercises all banks on all MBs using the following procedure:

1. Initializes the first memory location on the bank to test via scan.
2. Sets up a read request to the bank via scan.
3. Issues *n* system clocks, looping from 1 up to the bank latency.
4. Issues enough system clocks (bank_latency minus *n*) so that return data is in the output stage of the MB.
5. Verifies the return data in the scan ring of the MB.
6. Repeats steps 1 through 5 for all banks on the MB.
7. Repeats steps 1 through 6 for all MBs.

Figure 7-18 shows a typical subtest 205 error display.

Figure 7-18
Subtest 205 error display

```
Test: mem4000 R1.0   Subtest: 205 R1.0  Class: 2
Purpose: Scan-Based MB Single Step Testing.

CPU Port = 0
Physical Addr = 00000000
Memory Board: 7
Memory Bank:  8  (even)
Failure Mode: read failed
Ring: mbs_7

Stop Count       = 1
Failing Field    = rdedc.sys[0].xb_data
Expected Data    = a5a5a5a5
Actual Data      = a5a5a5a4
```

MB parity subtest—WREDC arrays (206)

Subtest 206 verifies the parity checking logic in the WREDC gate arrays on the MBs. This parity check assures that data is correctly transmitted from the MB to the bank. The subtest exercises all banks on all memory boards using a walking 1s pattern across the 32-bit data field. It uses the following procedure:

1. Loads the pattern into the MB scan ring: is?.b??_bank_wr_data.
2. Loads the correct parity into the MB scan ring: is?.b??_bank_wr_par.
3. Initializes the MB to send data to the MC: bc[*].all.bctl[*]._ref_pend, bc[*].all.bctl[*].wr_reg_hold.
4. Clocks MB to propagate data to the MC: 10 system clocks.
5. Verifies the results on the MC via the scan ring: nmc*_rerr fields should be 0.
6. Loads the pattern into the MB scan ring: is?.b??_bank_wr_data.
7. Loads incorrect parity into the MB scan ring: is?.b??_bank_wr_par.
8. Initializes the MB to send data to the MC: bc[*].all.bctl[*]._ref_pend, bc[*].all.bctl[*].wr_reg_hold.
9. Clocks the MB to propagate data to the MC: 10 system clocks.
10. Verifies the results on the MC via the scan ring: nmc*_rerr fields should be 1.
11. Repeats steps 1 through 10 for all patterns.
12. Repeats steps 1 through 11 for all MBs.

Figure 7-19 shows a typical subtest 206 error display.

Figure 7-19
Subtest 206 error display

```
Test: mem4000 R1.0   Subtest: 206 R1.0  Class: 2
Purpose: Scan-Based MB WREDC Parity Testing.

Ring: mb0
Scan Field: nmc0e_rerr
Parity error detected when not expected

Expected Data= 00000000
Actual Data  = 00000001
```

MB to MC signal subtest (207)

Subtest 207 verifies the control signal paths from the MB to the MC. This is a continuity check of these paths. The subtest exercises the following signals:

- RAM address (tests all 10 bits)
- Zone (tests all 4 bits)
- RAS and RFSH_ACT signals
- Single-bit inputs: _we, _g, _cas, data_sel, _check_le, ecc_check, and wr_sel
- Data strobe
- Correct detection of parity errors

Subtest 207 uses the following procedure to test all MCs on all MBs:

1. Loads pattern into the MB scan ring: ise.b?e_bank_wr_data, ise.b?e_bank_wr_par.
2. Initializes the MB to send data to the MC: *._ref_pend, *.wr_reg_hold.
3. Clocks the MB to propagate data to the MC (10 system clocks).
4. Verifies the results on the MC via the scan ring: nmc*_data, nmc*_ecc.
5. Repeats steps 1 through 4 for all patterns: walking 1 through all 32 bits and walking 0 through all 32 bits).
6. Repeats steps 1 through 5 for all MBs.

Figure 7-20 shows a typical subtest 207 error display

Figure 7-20
Subtest 207 error display

```
Test: mem4000 R1.0   Subtest: 207 R1.0  Class: 2
Purpose: Scan-based MB to MC Signal Testing.

Ring: mbs_7
MC   : 8
Scan Field: rdedc.sys[0].rdata
NMC data miscompare

Expected Data= a5c396f0
Actual Data  = a5c390b0
```

7.6.1.3 Scan-based crossbar/memory subtests (class 3)

Class 3 subtests utilize the MBs, MCs, and crossbar boards in their testing. All subtests use the error display shown in Figure 7-21.

Figure 7-21
Class 3 subtest error display

```
Test: mem4000  R1.0     Subtest: 303 R1.0  Class: 3
Purpose: Scan-Based TAC/TAS Testing.

CPU Port = 0
Physical Addr = 00000040
Memory Board: 7
Memory Bank:  8  (even)                DRAM Row: 0
Failure Mode: TAC did not clear data

Ring: xrte
Failing Field = data_p[0]
Expected Data = a5a5a5a5
Actual Data   = 85a5a1a4
```

Request priority generator subtest (300)

Subtest 300 verifies the priority circuitry on the crossbar. It makes read requests simultaneously from several CPU ports to the same memory bank with each CPU in turn set so it has the highest initial priority. It verifies the order that the requests are processed.

Subtest 300 specifies the current CPU on the XS0 boards as follows:

"xarb[0].send_sel[0]" /* PID for MB 0 */

"xarb[0].send_sel[1]" /* PID for MB 1 */

"xarb[0].send_sel[2]" /* PID for MB 2 */

"xarb[0].send_sel[3]" /* PID for MB 3 */

"xarb[0].send_sel[4]" /* PID for CR */

"xarb[1].send_sel[0]" /* PID for MB 4 */

"xarb[1].send_sel[1]" /* PID for MB 5 */

"xarb[1].send_sel[2]" /* PID for MB 6 */

"xarb[1].send_sel[3]" /* PID for MB 7 */

"xarb[1].send_sel[4]" /* Not used */

The subtest exercises the paths from all CPU ports to all banks on all MBs using the following procedure:

1. Sets CPU 0 to the highest priority {xarb[*].psel* fields}.
2. Sets up several read requests via scan using the following algorithm:
   - if ((port == 0) | | (port == 7)) . . requests from CPU 0 and CPU 7 . else . . requests from CPU 0, CPU <port>, and CPU 7
3. Issues 1 system clock and verifies that the read request from the highest priority port was processed first {xarb[*].send_sel[*] field}.
4. Issues enough system clocks so that the next request is processed, and verifies that the CPU 7 read request is processed next {xarb[*].send_sel[*] field}.
5. If testing CPU port 1 through 6, issues enough system clocks so that the next request is processed, and verifies that the CPU 0 read request is processed next {xarb[*].send_sel[*] field}.
6. Repeats steps 1 through 5 for CPU ports 1 through 7.
7. Repeats steps 1 through 6 for all banks on the MB.
8. Repeats steps 1 through 7 for all MBs.

Processor ID generator subtest (301)

Subtest `301` verifies the operation of the crossbar circuitry. The subtest issues different memory requests (`nop`, read, write, `tac`, `tas`) and checks the registers in the crossbar and MBs after each clock. It verifies the paths to and from all MBs using all processor ports (from the crossbar input). It uses the following procedure:

1. Sets up a write request on the input stage of the send crossbar via scan.
2. Issues 2 system clocks.
3. Verifies the scan fields on the output stage of the send crossbar; checks data, data parity, zone, address, cycle, and control parity.
4. Issues 1 system clock.
5. Verifies the scan fields on the input stage of the MB; checks ready, data, data parity, zone, address, cycle, and control parity.
6. If a read is being performed, issues enough system clocks to get data to the return side of the MB, else skips the next 3 steps.
7. Verifies the scan fields on the output stage of the MB; checks ready, data, and data parity.
8. Issues 1 system clock.
9. Verifies the scan fields on the input stage of the return crossbar ; checks data and data parity.
10. Repeats steps 1 through 9 for read, `tac`, and `tas` requests.
11. Repeats steps 1 through 10 for all CPU ports.
12. Repeats steps 1 through 11 for all BCGAs (banks 0, 4, 8, and 12 on both sides) on the MB.
13. Repeats steps 1 through 12 for all MBs.

Parity error subtest (302)

Subtest 302 verifies the error information stored in crossbar registers when an MB detects a parity error on a request. It uses the following procedure:

1. Sets up a read request with incorrect parity on the even send crossbar via scan.
2. Issues the number of system clocks to process a read request.
3. Verifies that a send parity error was generated: send_par_err.
4. Reads and verifies the crossbar look-aside registers, which should contain the failing read request.
5. Repeats steps 1 through 4 for write and `tam` requests.
6. Repeats steps 1 through 5 for all processor ports.
7. Repeats steps 1 through 6 for the odd side of the send crossbar.
8. Repeats steps 1 through 7 for all MBs.

`tac/tas` instruction subtest (303)

Subtest 303 verifies that the `tac`, `tas`, and `nop` instructions execute correctly. All 16 zone bit combinations are used during testing.

1. Initializes the first memory location of a bank.
2. Executes a `tac` instruction using a zone of 1.
3. Reads and verifies the memory location (4 bytes).
4. Executes a `tas` instruction using the same zone as above.
5. Reads and verifies the memory location (4 bytes).
6. Executes a `nop` memory request.
7. Reads the memory location (4-bytes) and verifies that it is unchanged.
8. Repeats steps 1 through 7 using all 16 different zone bit combinations.
9. Repeats steps 1 through 8 for all banks on a MB.
10. Repeats steps 1 through 9 for all MBs.

Burst mode subtest (304)

Subtest 304 verifies the "burst mode" arbitration by having a single CPU make *n* sequential write requests to consecutive banks (0 to *n*-1) while a second CPU makes a write request to the last bank (*n*-1). The write request from the second CPU is processed only after the *n*th request from the first CPU. The memory contents of banks 0 to *n*-2 is from the first CPU, and the memory contents of bank *n*-1 is from the second CPU. The subtest exercises all banks on all memory boards.

The number of sequential requests to sequential banks from a single processor is scan programmable. The user can change test values by setting up the burst_cnt_mask parameter.

The burst_cnt_mask parameter is 8 hex digits long, but only uses the 4 least significant digits. These digits operate as a 16-bit mask. The rightmost bit <0>, if set, causes a CPU to make 1 request to memory bank 0 on a particular memory board while a second CPU makes a request to the same bank on the same board. The next bit <1>, if set, causes a CPU to sequentially make 1 request to memory banks 0 and 1 on a memory board, while a second CPU makes a request to bank 1. Mask bit <2> causes a CPU to similarly make 3 requests to sequential memory banks, while a second CPU makes its request. Mask bits <3..15> are similar, with mask bit <15> , if set, causing a CPU to make 16 sequential requests while another CPU makes a request. During the test, the system checks each mask bit in sequence and performs the burst test if the mask bit is set to 1, or goes on to check the next mask bit if the former mask bit is set to 0.

The subtest uses the following procedure:

1. Sets up a write request from CPU 1 to the bank *n*-1 of an available MB.
2. Sets up write requests from CPU 0 to banks 0 through *n*-1 of the same MB.
3. Clocks the system long enough for all requests to be completed.
4. Reads the data from banks 0 through *n*-1 (via scan) and verifies contents.
5. Repeats steps 1 through 4 for all selected burst counts.

Odd/even bank subtest (305)

Subtest 305 verifies that the odd side and the even side of the same memory board are accessible simultaneously by having one CPU request data from the odd side of a memory board and a second CPU request data from the even side of the same memory board. The subtest makes both requests to memory at the same time and returns the data on the same clock. It exercises all MBs using the following procedure:

1. Sets up a write request from CPU 0 to the even side of the MB.
2. Sets up a write request from CPU 1 to the odd side of the MB (different data).
3. Clocks the system long enough for the requests to be completed.
4. Sets up a read request from CPU 0 to the even side of the MB (same address as above).
5. Sets up a read request from CPU 1 to the odd side of the MB (same address as above).
6. Clocks the system long enough for the requests to be completed.
7. Verifies the returned data.
8. Repeats steps 1 through 7 for all MBs.

Stop/restart subtest (306)

Subtest `306` verifies that the SPU has the ability to read memory after the system clocks are stopped without corrupting a memory request or memory contents. It uses the following procedure:

1. Initializes memory locations 0 and 4 on all specified MBs.
2. Sets up `read` requests on the crossbar to all MBs (both odd and even sides) via scan.
3. Issues enough system clocks so the read requests are in the input stage of the MBs.
4. Stops all system clocks except the free-running clocks on the MB and IA.
5. Saves the SYS ring for all MBs.
6. Disables single stepping on each bank controller on all MBs: bc[*].sys.sst_enable.
7. Resets the crossbar and MBs.
8. If enabled by the SPU memory requests parameter, makes SPU `read` requests at the same addresses as used above, and verifies the return data.
9. Resets the crossbar and the MBs.
10. Searches the SYS rings for all read requests (in the input stages of the MBs) and reissues these requests on the crossbar output stage.
11. Issues 1 system clock so the read requests are in the input stage of the MBs.
12. Restores the SYS ring to all MBs.
13. Clocks the system for the remainder of the read cycle and verifies the return data on the crossbar.

Figure 7-22 shows a typical subtest 306 error display.

Figure 7-22
Subtest 306 error display

```
Test: mem4000 R1.0   Subtest: 306 R1.0  Class: 3
Purpose: Scan-Based Memory Restart Testing.

Memory Board: 1
Memory Bank: 01 (even)          DRAM Row: 00

Failure Mode: Return data on SPU read is incorrect
Physical Addr = 00000008

Expected Data = a5a5a5a5
Actual Data   = 00a5a5a5
```

Communication register/memory contention subtest (307)

Subtest 307 verifies the commreg/memory data read contention logic. This logic resolves the problem of a commreg request taking less time to execute than a memory request and the FIFO scheduling.

One CPU makes 2 sequential read requests, the first to memory and the second to the CU board (commreg). The commreg request should complete first, but the memory request was made first and its data should return before the commreg data.

The subtest exercises all CPU ports. It uses the following procedure:

1. Sets up a write request to physical address 0 of MB.
2. Sets up a write request to commreg 0 (different data).
3. Clocks the system long enough for requests to be completed.
4. Sets up a read request from CPU 0 to memory (same address as above).
5. Issues one system clock.
6. Sets up a read request from CPU 0 to commreg 0.
7. Clocks the system long enough to complete a request from the MB.
8. Clocks the system again and verifies that the returned data is from the communication register.

Figure 7-23 shows a typical subtest 307 error display.

Figure 7-23
Subtest 307 error display

```
Test: mem4000 R1.0   Subtest: 307 R1.0  Class: 3
Purpose: Scan-Based Comm Register/Memory Contention Testing

CREG data incorrect
CPU Port = 0
Expected Data (commreg 0)= 00000000 00000000
Actual Data (commreg 0)  = ffffffff ffffffff
```

7.6.1.4 SPU-based memory subtests (class 4)

Class 4 subtests exercise the crossbar, MBs, and MCs. These subtests all function via the SPU-to-memory window interface. They use scan operations only to initialize certain states on the MBs. You can select the memory range to be tested.

Zone request subtest (350)

Subtest 350 functionally verifies that writing with different combinations of zone bits works correctly. The subtest uses all 16 combinations. It uses the NXP interface (map registers) to access memory. It uses the following procedure:

1. Writes all ones to a memory location.
2. Writes a different pattern with the specific zone bits to the same memory location.
3. Reads the memory location and verifies that only the bytes specified by the zone bits were modified.

Figure 7-24 shows a typical error display for subtest 350.

Figure 7-24
Subtests 350, 355, 360, 365, 370, and 375 error display

```
Test: mem4000 R1.0.   Subtest: 350 R1.0  Class: 4
Purpose: Memory testing via NXP operations: read-modify-write.

Memory Board: 1
Memory Bank: 01 (even)          DRAM Row: 00

Failure Mode: incorrect data for zone mask 0xff000000
Physical Addr = 00000008

Expected Data = a5ffffff
Actual Data   = 00ffffff
```

tac instruction subtest (355)

Subtest 355 functionally verifies the operation of the tac instruction to all banks on all memory boards. It performs memory accesses using the NXP interface (map registers) using the following procedure:

1. Writes a background pattern to bank 0 of the first selected MB.
2. Sends a tac instruction to byte 0 of this bank.
3. Reads the first word (32-bits) of this bank and verifies that only byte 0 was cleared.
4. Repeats steps 1 through 3 for all 4 bytes in this word.
5. Repeats steps 1 through 4 for all 32 banks.
6. Repeats steps 1 through 5 for all selected MBs.

Figure 7-24, page 7-54, shows a typical error display for subtest 355.

tas instruction subtest (360)

Subtest 360 functionally verifies the operation of the tas instruction to all memory boards. It performs memory accesses using the NXP interface (map registers) using the following procedure:

1. Writes a background pattern to bank 0 of the first selected MB.
2. Sends a tas instruction to byte 0 of this bank.
3. Reads the first word (32-bits) of this bank and verifies that only byte 0 was set to all ones.
4. Repeats steps 1 through 3 for all 4 bytes in this word.
5. Repeats steps 1 through 4 for all 32 banks.
6. Repeats steps 1 through 5 for all selected MBs.

Figure 7-24, page 7-54, shows a typical error display for subtest 360.

scrub operation subtest (365)

Subtest 365 functionally verifies the scrub operation from the SPU to all memory boards. It performs memory accesses using the NXP interface (map registers) using the following procedure:

1. Writes the first word of bank 0 with a data pattern containing a single bit error.
2. Performs a scrub operation on this memory location.
3. Reads this memory location and verifies that it was not altered and that a single bit error was not detected (a soft error).
4. Repeats steps 1 through 3 for all 32 banks.
5. Repeats steps 1 through 4 for all selected MBs.

Figure 7-24, page 7-54, shows a typical error display for subtest 365.

MB log ring subtest (366)

Subtest 366 verifies the operation of the LOG ring on each MB. It uses the following procedure:

1. Initializes address 0 (even side) of the test MB with a single EDC error.
2. Sets up an NXP transfer to read this memory location.
3. Verifies that a soft error was generated.
4. Verifies that the LOG ring in the test MB contains the correct data.
5. Repeats steps 1 through 4 using address 4 (odd side).
6. Repeats steps 1 through 5 for all MBs.

The subtest uses the display in Figure 7-25 to report an error in the LOG ring contents. A double star (**) on the right side marks each data item that failed. See the last data item for an example.

Figure 7-25
Subtest 366 error display

```
Test: mem4000 R1.0.  Subtest: 366 R1.0  Class:4
Purpose: Memory testing via NXP operations: MB LOG Ring

Memory Board:  0
Failure Mode:  MB LOG ring data miscompare

Scan Field                          Expected      Actual
bc9_bank0_log_addr                     0            0
bc[9].bcga_log.bct10_soft_err          1            1
bc[9].bcga_log.bct10_rdedc_err         1            1
bc[9].bcga_log.bct10_cycle             2            2
rdedc.log[9].error_data             a5a5a5a5     a5a5a5a5
rdedc.log[9].merr                      0            0
rdedc.log[9].serr                      1            0        **
```

Page addressing subtest (370)

Subtest 370 verifies that each memory page is uniquely addressable. It performs memory accesses using the NXP interface (map registers) using the following procedure:

1. Writes the first word of page 0 with an address pattern.
2. Writes the last word of page 0 with an address pattern.
3. Writes the first word of the next page with an address pattern .
4. Repeats steps 1 through 3 for every 1024th page in the specified memory range.
5. Reads the first word of page 0 and verifies that it is the address pattern.
6. Reads the last word of page 0 and verifies that it is the address pattern.
7. Reads the last halfword of page 0 and the first halfword of the next page and verifies that it is the address pattern.
8. Repeats steps 5 through 7 for every 1024th page in the specified memory range.

Figure 7-24, page 7-54, shows a typical error display for subtest 370 .

NXP memory access subtest (375)

Subtest 375 verifies the ability to access memory via the NXP transfers (map registers). It initially loads all pages with 0s. It then reads each page to verify that contains all 0s and writes all 1s into the page. Finally, it reads back all pages and verifies that they contain all 1s. It uses the following procedure:

1. Checks the PCM to determine whether the current page is installed.
2. If the page is installed, goes to step 3, else increments the page address and goes to step 1.
3. Writes all 0s to the current page.
4. Increments the page address and goes to step 1.
5. When all pages are written with zeroes, goes to the first page.
6. Checks the PCM to determine whether the current page is installed.
7. If the page is installed, goes to step 8, else increments the page address and goes to step 6.
8. Reads and verifies that all 4096 bytes of the current page contain zeroes.
9. Writes all 1s to the current page.
10. When all pages are checked and written with 1s, goes to the first page.
11. Checks the PCM to determine whether the current page is installed.
12. If the page is installed, goes to step 13, else increments the page address and goes to step 11.
13. Reads and verifies that all 4096 bytes of the current page contain 1s.
14. Continues looping until all pages are checked.

Figure 7-24, page 7-54, shows a typical error display for subtest 375.

Memory address subtest (380)

Subtest 380 functionally verifies the address uniqueness of each memory location. It makes all memory accesses via the MTL interface. It stores the address of each memory location in the corresponding memory location. It then waits for the number of seconds specified by the wait time parameter. Finally, it reads and verifies the memory with an MTL compare operation.

If an error occurs, it obtains the expected data from the following MTL registers:

- mem_test_even
- mem_test_odd

The actual data is obtained from memory at the specified failing address (register main_mem_addr) using NXP transfers. Figure 7-26 shows a typical error display for subtest 380.

Figure 7-26
Subtests 380 and 385 error display

```
Test: mem4000 R1.0   Subtest: 380 R1.0  Class: 4
Purpose: Memory testing via Memory Test Logic: address pattern.

Memory Board: 7
Physical Addr: = 00000028

Failure Mode: MTL compare error detected

Expected Data = 0000000a  0000000b
Actual Data   = 0000002e  0000000b
```

Subtests 380 and 385 also use the error display of Figure 7-27. This display indicates that the MTL fill or compare operation failed for some reason. Two of the most probable causes are:

- System is not correctly margined.
- One or more boards have parity errors (CU, IA, MB, or crossbar).

Figure 7-27
Additional error display for subtests 380 and 385

```
Test: mem4000 R1.0   Subtest: 385 R1.0  Class: 4
Purpose: Memory testing via Memory Test Logic: various patterns.

Timeout during MTL fill operation

Physical Addr: 00000000
Status: 00010000
```

Memory data retention subtest (385)

Subtest `385` verifies the ability of memory to hold data. The subtest makes all memory accesses via the MTL interface. It tests the memory range specified in the `test_par` parameters. It uses the following data patterns:

- All zeroes
- All ones
- a5a5a5a5/5a5a5a5a
- 69696969/96969696
- c3c3c3c3/3c3c3c3c
- Walking 1 through 32 bits
- Walking 0 through 32 bits
- Walking 1 through each nibble
- Walking 0 through each nibble
- Complemented ffffffff
- Complemented a5a5a5a5
- Complemented c3c3c3c3
- Complemented 9696969

After writing to memory, the subtest waits for the number of seconds specified by the wait time parameter. It then reads and verifies memory contents with an MTL compare operation.

If an error occurs, the subtest obtains the expected data from the following MTL registers:

- mem_test_even
- mem_test_odd

The subtest obtains the actual data from memory at the specified failing address (register main_mem_addr).

7.6.1.5 CPU-based memory subtests (class 5)

Class 5 memory subtests are CPU-based. They execute on a processor and verify the functionality of the CPU-crossbar interface and the memory subsystem (crossbar-MB-MC) from the CPU side. They use scan operations only to initialize the system.

All subtests in this class use the error display shown in Figure 7-28. Use another utility such as DDB to obtain register contents of the failing CPU and various memory locations.

Figure 7-28
Error display for class 5 subtests

```
Test: mem4000 R1.0   Subtest: 400 R1.0  Class: 5
Purpose: CPU test: MATS+ algorithm using various patterns.

** Miscompare from pocessor 0

Logical Addr       = 0000e048          Physical Addr      = 0000e048
Actual High Word   = 01104000          Actual Low Word    = 00000000
Expected High Word = 00000000          Expected Low Word  = 00000000

[trace value is: 00000009]
```

Data retention subtests (400 and 410)

Subtests 400 and 410 verify the integrity of memory by writing several data patterns into memory and then reading and verifying these data. The subtests use the MATS+ and the NTA (Nair, Thatte, and Abraham) algorithms. These algorithms verify the ability of memory to hold data and test for shorts between the memory cells. They first write all of memory and then perform the read, verify, and write operation on one location at a time.

The MATS+ algorithm, subtest 400, requires 5 accesses per memory word. Its purpose is to detect all stuck-at faults. This algorithm always performs the read-verify-write sequence starting with word 0 or word 1 and goes to word *n*-1.

The NTA algorithm, subtest 410, requires 30 accesses (read or write) per memory word. Its purpose is to detect all coupling faults as well as all stuck-at faults. It performs the read-verify-write sequence both forwards and backwards. The forward sequence starts with word 0, goes forward to word *n*-1, and back to word 0. The backward sequence starts with word *n*-1, goes backward to word 0, and then forward to word *n*-1.

Subtests 400 and 410 use the following procedure:

1. Waits for a command from the SPU.
2. Sets the running status bit and clears the command field.
3. Initializes the registers with starting address, ending address, and the two patterns.
4. Determines whether the MATS+ or NTA test is to be executed.
5. Performs the appropriate test.
6. If errors exist, sets the error status bit and waits for either the continue (go to previous step) or the restart (go to step 1) command.
7. If the test is complete and no errors were detected, sets the test_complete status bit and goes to step 1.

The following patterns are used in these two subtests:

0x00000000/0xffffffff

0xffffffff/0x00000000

0x55555555/0x00000028

0x0bbbbbbb/0xcccccccc

0x33333333/0x0f0f0f0f

Bank uniqueness subtest (420)

Subtest `420` verifies that each bank on each MB is uniquely addressable. The subtest writes a unique pattern to each bank, then reads the pattern back and verifies it. This pattern consists of a board field and a bank field.

Subtest `420` uses the following procedure:

1. Initializes the CPU command processor to write a unique pattern to the first word of all banks on MB0. This pattern consists of a board field and a bank field.
2. Initializes the CPU command processor to read the first word of all banks on MB0 with the verify option.
3. Waits for test completion. If no errors, or done status, then continues to the next step, otherwise reads the error information and displays it.
4. Repeats steps 1 through 3 for all installed MBs.

Data length/alignment subtest (430)

Subtest 430 verifies memory by performing writes to memory using the different data lengths and boundaries (byte, halfword, word, longword). The data is then read and verified. All banks on all memory boards are tested.

Subtest 430 uses the following procedure:

1. Initializes the CPU command processor to write the first byte of all banks on MB0.
2. Initializes the CPU command processor to read the first byte of all banks on MB0 with the verify option.
3. Waits for test completion. If no errors, or done status, the subtest continues to the next step. Otherwise, the subtest reads the error information and displays it.
4. Repeats steps 1 through 3 for all byte boundaries (byte 0-7).
5. Repeats steps 1 through 4 for all four data lengths (byte, halfword, word, longword).
6. Repeats steps 1 through 5 for all installed MBs.

tac and tas instruction subtests (440 and 445)

Subtests 440 and 445 verify that the tac and tas instructions, respectively, work correctly from the SPU. The subtests execute these instructions over the specified range of memory on all bytes.

Subtest 440 and 445 use the following procedure:

1. Initializes the specified range of memory to 1s or 0s for testing of the tac or tas instruction, respectively.
2. Initializes the CPU command processor to execute tac or tas instructions on this memory range.
3. Waits for test completion. If no errors, or done status, then continues to the next step. Otherwise, reads the error information and displays it.
4. Initializes the CPU command processor to read and verify the contents of memory over the specified memory range. The tac tests should yield all 0s. The tas tests should yield all 1s.
5. Waits for test completion.
6. After completion of the procedure for the tac instruction, repeats it for the tas instruction.

Stop/start memory subtest (450)

Subtest 450 verifies that memory can be stopped then restarted without losing either memory requests or data. It continuously reads memory and verifies the read data. The length and number of the reads should be long enough so they are still being processed by the MB when the SPU stops the CPU.

Subtest 450 uses the following procedure:

1. Initializes the first word of all banks with an addressing pattern.
2. Initializes CPU command processor to read the first word of all banks with the verify option.
3. Stops all system clocks except the free-running clocks on the MB and IA.
4. Restarts all system clocks.
5. If selected, performs a SPU read of the first word of each bank and verifies the data (NXP transfers).

Figure 7-29 shows a typical subtest 450 error display.

Figure 7-29
Subtest 450 error display

```
Test: mem4000 R1.0   Subtest: 450 R1.0  Class: 5
Purpose: CPU test: Stop-Start Memory.

Failure Mode: Data miscompare after stopping clocks
Physical Addr  = 00000000

Expected data  = a5a5a5a5
Actual Data    = 00a5a5a5
```

Simultaneous SPU/CPU `tac` instructions subtest (460)

Subtest `460` verifies that the `tac` and `tas` instructions work correctly from the SPU and a CPU simultaneously. It executes these instructions over a small range of memory to assure simultaneous accesses to the same location.

Subtest `460` uses the following procedure:

1. Initializes the first longword of available memory to all 1s (to all 0s if the `tas` instruction is being tested).
2. Initializes the CPU command processor to execute `tac` instructions on byte 0 of this longword with the verify option. It executes these instructions continuously until halted.
3. Performs `tac` operations from the SPU on bytes 1-3 of this longword, then reads and verifes that the specified memory locations contain all 0s. Checks for errors on the CPU. If an error is detected then reads the error information and displays it. Also sends a halt command to the CPU command processor.
4. Performs `tas` operations from the SPU on bytes 0-3 of this longword, then reads and verifies that the memory locations contain all 1s. When byte 0 is written it should be overwritten by the CPU. Checks for errors on the CPU. If the subtest detects an error, it reads the error information and displays it. Also sends a `halt` command to the CPU command processor.
5. Sends a `halt` command to CPU command processor.

R&M bits subtest (500)

Subtest 500 verifies the operation of the referenced and modified bits (R&M bits). It reads and writes the first byte and last byte of each page to be tested. After each operation the R&M bits are verified and cleared for the next operation.

Subtest 500 uses the following procedure:

1. Computes the address of the halfword containing R&M bits for the page under test.
2. Reads the first byte of the page.
3. Verifies that the referenced bit is 1 and the modified bit is 0, and clears the halfword containing these bits after verifying.
4. Writes the first byte of the page.
5. Verifies that the referenced bit is 1 and the modified bit is 1, and clears the halfword containing these bits after verifying.
6. Reads the last byte of the page.
7. Verifies that the referenced bit is 1 and the modified bit is 0, and clears the halfword containing these bits after the check.
8. Writes the last byte of the page.
9. Verifies that the referenced bit is 1 and the modified bit is 1, and clears the halfword containing these bits after verifying.
10. Repeats steps 1 through 9 for all pages to be tested.

CPU diagnostic tests

8

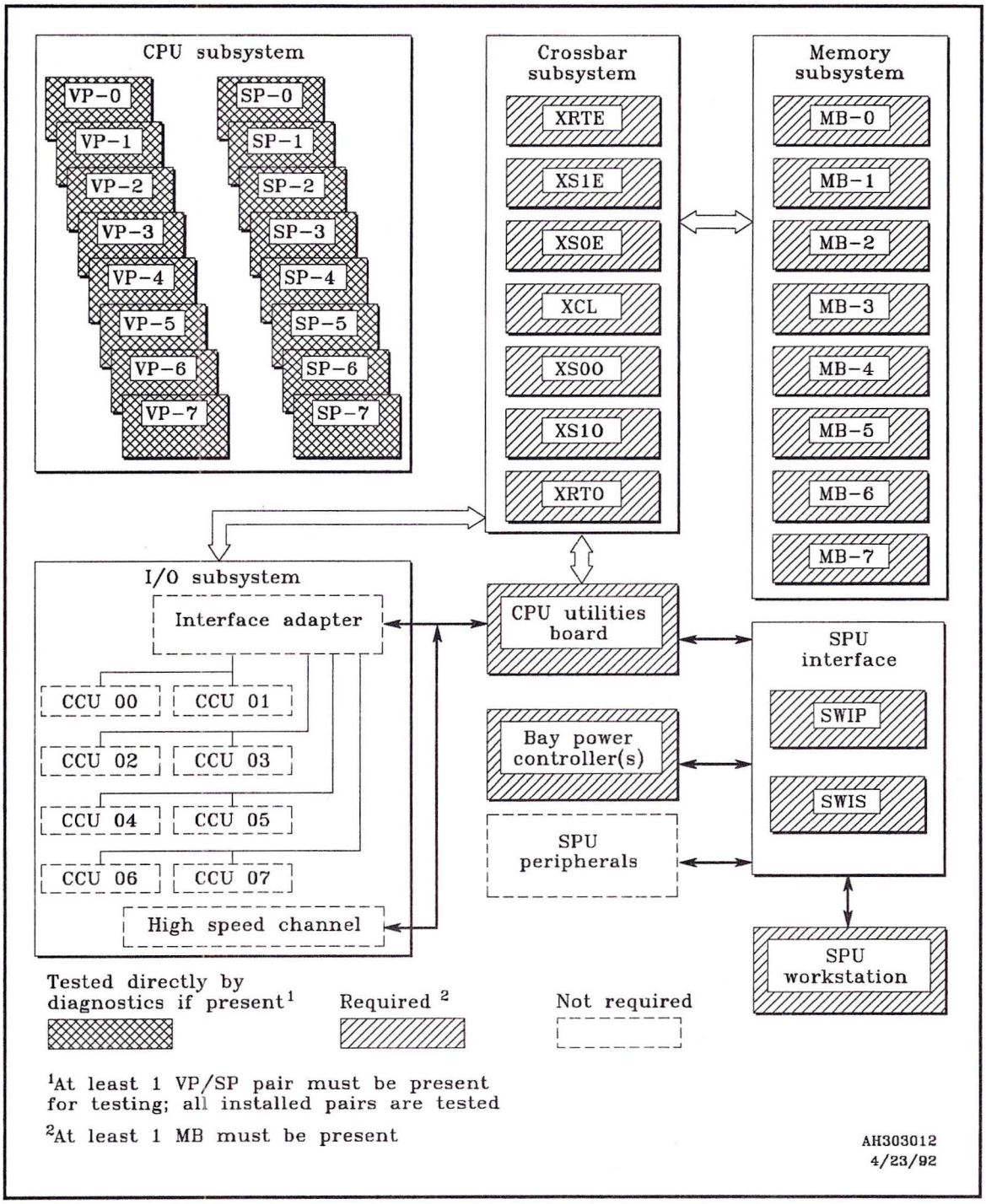
The CPU diagnostic tests exercise and verify the logic on the scalar processor (SP) and vector processor (VP) circuit boards. The CPU diagnostic tests build on one another. Each successive test requires that the VP and SP must have passed the preceding tests in the CPU diagnostic test sequence.

The recommended test sequence is as follows:

- `cpu4030`—Scalar building block test
- `cpu4041`—Vector instruction test
- `cpu4331`—Privileged instruction and architectural features
- `cpu4332`—Enhanced, non-vector, single processor instruction tests
- `cpu4241`—Enhanced vector instruction tests
- `cpu4333`—Multiprocessor diagnostics

Figure 8-1 shows the parts of the system under test and indicates the field replaceable units (FRUs) required to run the tests.

Figure 8-1
FRUs required and exercised by CPU diagnostic tests



8.1 Test invocation

Use the following command to enter the xdiag environment.

```
(spu)> xdiag
```

xdiag is the X-based CONVEX test interface (cti) used by all C3 processor diagnostics. All CPU diagnostic tests operate within a single test interface, the CPUcti. Refer to Chapter 4 for a description of xdiag.

Use this procedure to execute and control a CPU test through the xdiag windows:

Step 1 From the main xdiag window, select Test and click on the desired test in the drop-down menu. This causes the cti to load the diagnostic files for the selected test.

Step 2 Click on CPUcti in the xdiag window menu bar to invoke the CPUcti Test Options window.

Step 3 Select the desired test parameters from the CPUcti Test Options window.

8.2 CPUcti Test Options menu

The following sections describe the test parameters available through the `CPUcti Test Options` window. This window is identical for all CPU diagnostic tests, although the operation of some test parameters varies slightly between tests. Figure 8-2 shows the `CPUcti Test Options` window.

The following paragraphs describe the test parameters.

8.2.1 CPUs to Test:

Use this entry field to select the CPU(s) to use in the test. The field consists of a button for each CPU that can be selected or deselected. Select the CPUs in the order you want the software to exercise them.

The software rejects selection of a CPU that is not installed, not available, or not powered. In this case, it displays an error message in the text field of the `xdiag` window and clears the button.

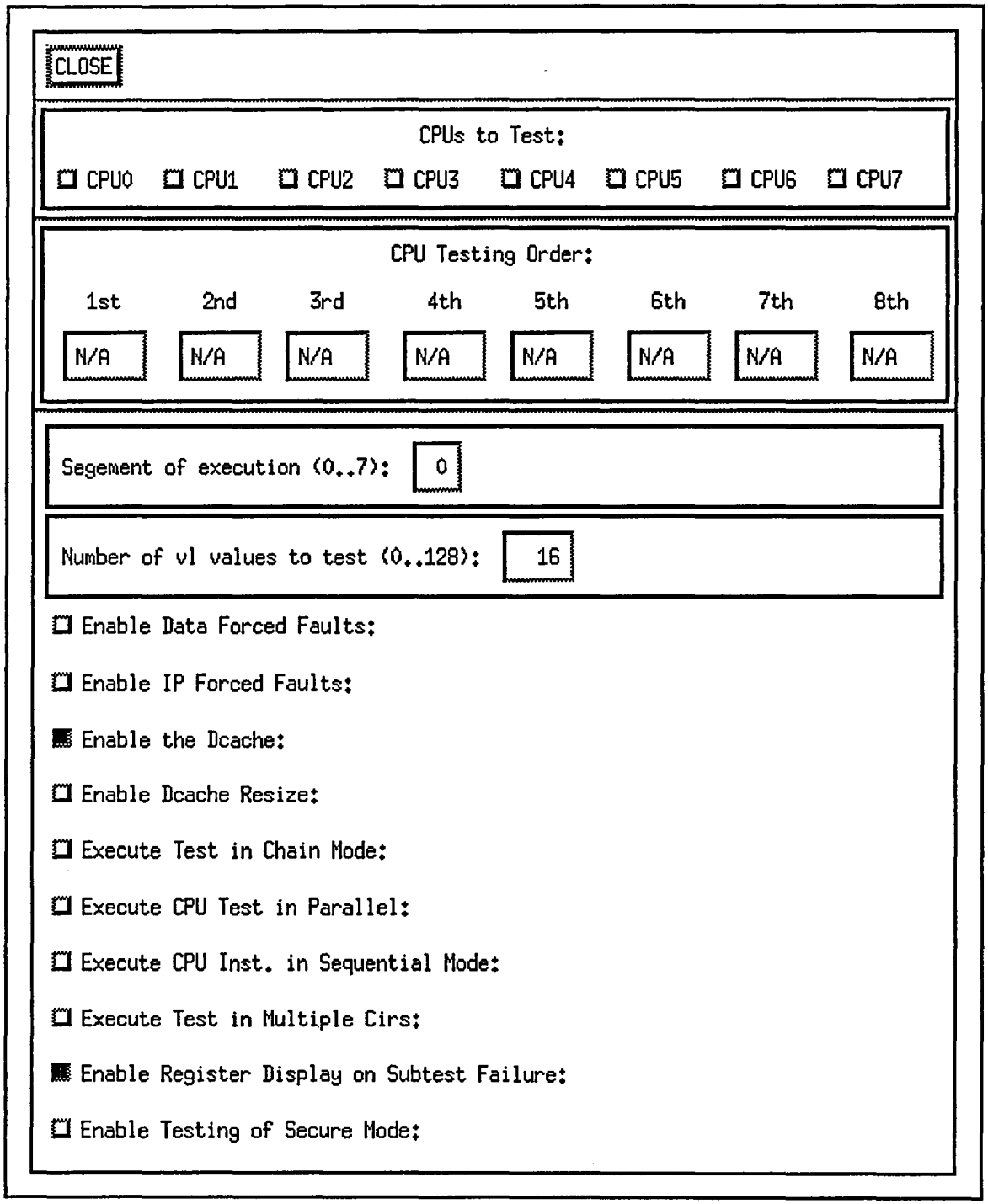
8.2.2 CPU Testing Order:

This field indicates the order in which the software exercises the CPUs. It reflects the order in which you selected the CPUs in the `CPUs to Test:` field.

8.2.3 Segment of execution [0..7]:

Bits <31..29> of the program counter (PC) contain the segment of execution parameter. If you enter 0, bits <31..29> of the PC are 000 and the test runs in ring 0. If you enter 1, bits <31..29> of the PC are 001 and the test runs in ring 1. If you enter 2, bits <31..29> of the PC are 010 and the test runs in ring 2. If you enter 3, bits <31..29> of the PC are 011 and the test is run in ring 3. If you enter 4, 5, 6, or 7, bits <31..29> of the PC are 100, 101, 110, and 111, respectively, and the test runs in ring 4. Refer to the *CONVEX Architecture Reference Manual (C Series)* for more information about the meaning of rings in the machine architecture.

Figure 8-2
CPUcti test options window



8.2.4 Number of vl values ...(0..128):

This subtest exercises vector hardware only.

Each instruction in this test executes twice, first with a vector length (`vl`) value of 128, and second with a `vl` value ranging from 0 to the `vl` count. The `vl` count is 16 if you select the default of this prompt.

8.2.5 Enable Data Forced Faults:

Select this option to include forced faults (nonresident data exceptions) on each data reference.

8.2.6 Enable IP Forced Faults:

Select this option to include forced faults on instruction fetches.

8.2.7 Enable the Dcache:

The data cache is normally enabled. Select this option to disable the data cache.

8.2.8 Enable Dcache Resize:

Select this option to test the system with the data cache resized from16 k to 4 k on inward ring crossings from ring 4.

8.2.9 Execute Test in Chain Mode:

Select this option to cause the CPU to perform subtest sequencing. Selecting this option greatly reduces test execution time. With this option enabled, the only information printed to the console upon completion or failure, regardless of the cause, is one of these messages:

- `Subtest X passed`
- `Subtest X failed`

`X` is the last subtest started.

8.2.10 Execute CPU Test in Parallel:

Select this option to enable parallel test execution.

8.2.11 Execute CPU Inst. . . . Mode:

Select this option to set the sequential bit in the processor status word (PSW) to forced sequential execution mode.

8.2.12 Execute Test in Multiple Cirs:

Select this option to use all communication register sets in the tests. This selection is not applicable for cpu4041.

8.2.13 Enable Register . . . Failure:

Select this option to display the registers for failing CPU(s).

8.2.14 Enable Secure Mode of Testing:

Select this option to test the system in secure mode.

8.3 When the X-Window environment is not available

If you are not working in an X-Window environment, refer to Section 4.13 on page 4-33 for information on invoking the diagnostics software.

To invoke cpu4030, enter:

```
spu> CPUcti -testname cpu4030
STARTUP>
```

To invoke cpu4041, enter:

```
spu> CPUcti -testname cpu4041
STARTUP>
```

To invoke cpu4331, enter:

```
spu> CPUcti -testname cpu4331
STARTUP>
```

To invoke cpu4332, enter:

```
spu> CPUcti -testname cpu4332
STARTUP>
```

To invoke cpu4241, enter:

```
spu> CPUcti -testname cpu4241
STARTUP>
```

To invoke cpu4333, enter:

```
spu> CPUcti -testname cpu4333
STARTUP>
```

To display test specific parameters, enter:

```
STARTUP> test_par
```

Figure 8-3 shows the test-specific parameters for the cu4000 test.

These are the parameter definitions:

- Chain_Mode—Execute the specified subtests in a continuous sequence if 1; execute as individual tests if 0.
- Cpu_Active_Count—Indicates the number of CPUs active in a test. You cannot change this parameter. It is informational only.
- CpuSequence#—Information only; shows the order of testing CPUs. CpuSequence0 shows the number of the first CPU to be tested, and so forth.
- CpusActive—Indicates whether or not a CPU is currently under test. To activate a CPU, enter:

  CPUsActive# on
  or
  CPUsActive# 1

Figure 8-3
`CPUcti test-specific parameters`

```
- "Chain_Mode" "0"
- "CpuSequence0" "N/A"
- "CpuSequence1" "N/A"
- "CpuSequence2" "N/A"
- "CpuSequence3" "N/A"
- "CpuSequence4" "N/A"
- "CpuSequence5" "N/A"
- "CpuSequence6" "N/A"
- "CpuSequence7" "N/A"
- "Cpu_Active_Count" "0"
- "CpusActive" "0,0,0,0,0,0,0,0"
- "Data_FFaults" "0"
- "Enable_Dcache" "1"
- "Enable_Dcache_Resize" "0"
- "Enable_State_Dumps" "1"
- "IP_FFaults" "0"
- "Multiple_Cirs" "0"
- "Parallel_Execution" "0"
- "SecureMode" "0"
- "Segment_Num" "0"
- "Sequential_Mode" "0"
- "Vector_Length" "16"
- "test_name" "cpu4030"
```

To remove a CPU from testing, enter:

`CPUsActive# off`
or
`CPUsActive# 0`

is the number of the CPU to activate or deactivate.

To set several CPUs either active or inactive, enter a 1 to test a CPU, or a 0 to not test a CPU. For example, to test only CPUs 1, 3, 5, and 8, enter:

`CPUsActive 1,0,1,0,1,0,0,1`

- `Data_FFaults`—Force data faults (take nonresident data exceptions on each data reference) if 1; do not force data faults of 0.
- `Enable_Dcache`—Execute subtests with data cache enabled if 1; execute with data cache disabled if 0. Default is 1.
- `Enable_Dcache_Resize`—In each subtest, resize the data cache from 4 k to 1 k on inward ring crossings from ring 4 if 1; do not resize data cache if 0.

- `Enable_State_Dumps`—Display register information for each failing CPU when a subtest failure occurs if 1; inactive if 0.
- `IP_FFaults`—Subtest takes a fault on each instruction prefetch if 1; fault not taken if 0.
- `Multiple_Cirs`—Applies only to tests `cpu4332` and `cpu4333`. Run each subtest in all 32 CIRs if 1; run in only one CIR if 0. Default is 0.
- `Parallel_Execution`—Run each subtest on all active CPUs in parallel if 1; run each subtest on each CPU sequentially if 0. Default is 0.
- `SecureMode`—Applies only to tests `cpu4332` and `cpu4333`. Test the system as a secure system if 1; do not test as a secure system if 0. Default is 0.
- `Segment_Num`—Load test into the specified segment number. Valid segment numbers are 0 to 7. Default segment number is 0.
- `Sequential_Mode`—Complete each subtest instruction before starting the next instruction if 1; inactive if 0.
- `Vector_Length`—Applies only to tests `cpu4041` and `cpu4241`. Indicates the maximum vector length allowed. Valid lengths are 0 to 128. Default is 16.
- `test_name`—Name of the current test. You specify this parameter when you invoke the test from the `spu>` prompt.

8.4 Scalar building block test (cpu4030)

The cpu4030 test verifies basic functionality of all nonprivileged scalar instructions in a nonexhaustive manner. It executes two types of actions:

- Fetches instructions from memory.
- Executes instructions that read and write selected memory addresses.

8.4.1 Prerequisites and required equipment

The cpu4030 test requires that the circuit boards listed in Table 8-1 be installed and verified by the tests listed. The vector processor (VP) and scalar processor (SP) boards must also be present.

Table 8-1
Hardware required for cpu4030

Board	Tests to verify
SPU workstation interface serial (SWIS)	spu4000
SPU workstation interface parallel (SWIP)	spu4000
CPU utilities (CU)	cu4000, sst, spu4000
Interface adapter (IA)	sst, spu4000
Crossbar subsystem (XRTE, XS1E, XS0E, XLL, XS00, XS10, XRT0)	sst, spu4000, mem4000
Memory subsystem (MBs)	sst, spu4000, mem4000

8.4.2 Invoking the test

Perform the following steps to invoke the `cpu4030` test:

Step 1 From the `xdiag` window, select `Test`.

Step 2 Click on `cpu4030` in the drop-down menu. The `cti` loads the cpu4030.info and cpu4030.par files. The default test parameters are initialized from the cpu4030.par file.

Step 3 Click on `CPUcti` to invoke the `CPUcti Test Options` window.

Step 4 Select the test options. See Section 8.2 of this document.

Step 5 Select the `run` button in the command bar of the `xdiag` window to start the test.

8.4.3 Class descriptions

`cpu4030` has 4 classes of subtests, as described below. All subtests terminate with a `halt` command and store a halt code in address register A1.

8.4.3.1 Class 1 subtests

Class 1 subtests are the most basic of the CPU verification tests. These subtests require no access to main memory for data. They verify the scalar functions of the CPU in a nonexhaustive manner. If on subtest completion the code stored in A1 is 0x100, then the subtest passed. Any other code in A1 indicates that the subtest failed.

8.4.3.2 Class 2 subtests

Class 2 subtests require memory accesses for data. If the code stored in A1 is 0x100, then the subtest passed. Any other code in A1 indicates that the subtest failed.

8.4.3.3 Class 3 subtests

Class 3 subtests access main memory for both data and instructions. If the code stored in A1 is 0x100, then the subtest passed. Any other code in A1 indicates that the subtest failed.

8.4.3.4 Class 4 subtests

The `cpu4030` class 4 subtests verify two basic capabilities of the machine that are not explicitly tested anywhere else. The first is the capability to execute code, branch forward and backward across all major carry addresses in the Program Counter (PC). The second is the verification of the ability to correctly wrap instructions within the current ring.

The carry testing is performed by subtests 500-515. Each of these subtests use the same object module. The module is loaded into a different logical address for each subtest, each module uses 2 pages of logical address space. The logical addresses used are 0xf000, 0xff000, 0xfff000, and 0xffff000. Each subtest executes code across all of the carries possible within its logical address space. For example, subtest 500 tests carries from 0xe \(-> 0x10, 0xfe \(-> 0x100, 0xffe \(-> 0x1000, and 0xfffe \(-> 0x10000.

The subtest executes code at the end of each segment to test ring wrapping. Code is mapped into the end of each segment (module wrapu_4030) and the end of each segment (wrapl_4030, p0r0_4030, or p0rN_4030). The module at the end of the segment then executes.

For segments 0-3, execution at the end of the segment causes the PC to wrap back to the beginning of the segment. This prevents violation of architectural ring protection constraints.

Segments 4, 5, 6, and 7 behave as 1 ring. Code executing to the end of segment 4 continues to execute in the beginning of segment 5. Similarly, code executing to the end of segments 5 and 6 continues to execute in the beginning of segments 6 and 7, respectively. Code executing at the end of segment 7 wraps to the beginning of segment 4.

8.4.4 Subtest descriptions

Table 8-2 describes the `cpu4030` class 1 subtests.

Table 8-2
`cpu4030` class 1 subtests

Subtest	Test performed		
1	`ld.h #N,Ak`	`ld.w #N,Ak`	
2	`ld.w #N,Sk`	`ld.w #N,Ak`	
5	`mov Aj,Ak`	`mov Aj,Sk`	
6	`mov Sj,Ak`	`mov.w Sj,Sk`	`mov.l Sj,Sk`
7	`mov PC,Ak`		
10	`and Aj,Ak`	`and #N,Ak`	
11	`and Sj,Sk`	`and #N,Sk`	
14	`or Aj,Ak`	`or #N,Ak`	
15	`or Sj,Sk`	`or #N,Sk`	
16	`mov Ak,psw`	`mov psw,Ak`	
18	`xor Aj,Ak`	`xor #N,Ak`	
19	`xor Sj,Sk`	`xor #N,Sk`	
20	`not Aj,Ak`		
21	`not Sj,Sk`		
24	`add.h Aj,Ak` `add.w #N,Ak`	`add.w Aj,Ak`	`add.h #N,Ak`
25	`add.b Sj,Sk`		
26	`add.h #N,Sk`	`add.h Sj,Sk`	

Table 8-2 (continued)
cpu4030 class 1 subtests

Subtest	Test performed		
27	add.w #N,Sk	add.w Sj,Sk	add.w Sj,Ak
28	add.l Sj,Sk		
30	sub.h #n,a	sub.h ai,Aj	
31	sub.w #n,Ak	sub.w Aj,Ak	
32	sub.b Sj,Sk		
33	sub.h #n,Sj	sub.w Sj,Sk	
34	sub.w #,Sk	sub.w Sj,Sk	
35	sub.l Sj,Sk		
40	neq.h Aj,Ak		
41	neq.w Aj,Ak		
42	neg.b Sj,Sk		
43	neg.h Sj,Sk		
44	neg.w Sj,Sk		
45	neg.l Sj,Sk		
50	branches		
52	cmp.h Aj,Ak		
53	cmp.w Sj,Ak		
54	cmp.h #N,Ak		
55	cmp.w #,Ak		
60	cmp.b Sj,Sk		
61	cmp.h Sj,Sk		
62	cmp.w Sj,Sk		
63	cmp.h #,Sk		
64	cmp.w #N,Sk		
65	cmp.l Sj,Sk		
66	shf Aj,Ak	shf #n,Ak	shf #N,Ak
67	shf Sj,Sk	shf #N,Sk	

Table 8-2 (continued)
cpu4030 class 1 subtests

Subtest	Test performed		
68	tzc Sj,Sk		
69	plc.t Sj,Sk		
70	mul.h #n,Ak	mul.h #N,Ak	mul.h Aj,Ak
71	mul.w #n,Ak	mul.w #N,Ak	mul.w Aj,Ak
72	mul.b Sj,Sk		
73	mul.h #N,Sk	mul.h Sj,Sk	
74	mul.w #N,Sk	mul.w Sj,Sk	
75	mul.l Sj,Sk		
80	div.h #n,Ak	div.h #N,Ak	div.h Aj,Ak
81	div.w #n,Ak	div.w #N,Ak	div.w Aj,Ak
82	div.b Sj,Sk		
83	div.h #n,Sk	div.h #N,Sk	div.h Sj,Sk
84	div.w #n,Sk	div.h #N,Sk	div.h Sj,Sk
85	div.l Sj,Sk		
90	cvt(w.b,b.w,w.h,h.w) Aj,Ak		
91	cvt(w.b,b.w,w.h,h.w,w.l,l.w) Sj,Sk		
100	eq.s #n,Sk	eq.s #N,Sk	eq.s Sj,Sk
101	le.s #n,Sk	le.s #N,Sk	eq.s Sj,Sk
102	lt.s #n,Sk	lt.s #N,Sk	eq.s Sj,Sk
103	eq.d Sj,Sk		
104	le.d Sj,Sk		
105	lt.d Sj,Sk		
110	add.s #N,Sk Sj,Sk		
111	add.d Sj,Sk		
115	sub.s #N,Sk Sj,Sk		
116	sub.d Sj,Sk		
125	neg.s Sj,Sk		
126	neg.d Sj,Sk		

Table 8-2 (continued)
cpu4030 class 1 subtests

Subtest	Test performed
130	mul.s #N,Sk mul.s Sj,Sk
131	mul.d Sj,Sk
132	div.s #N,Sk div.s Sj,Sk
133	div.d Sj,Sk
140	cvtw.s Sj,Sk
142	cvts.w Sj,Sk
144	cvtd.l Sj,Sk
146	cvtl.d Sj,Sk
148	cvtl.s Sj,Sk
150	cvts.l Sj,Sk
152	cvts.d Sj,Sk
154	cvtd.s Sj,Sk
1100	eq.s #N sk eq.s sj sk (IEEE)
1101	le.s #N sk le.s sj sk (IEEE)
1102	lt.s #N sk lt.s sj sk (IEEE)
1103	eq.d sj sk (IEEE)
1104	le.d sj sk (IEEE)
1105	lt.d sj sk (IEEE)
1110	add.s #N sk add.s sj sk (IEEE)
1111	add.d sj sk (IEEE)
1115	sub.s sj sk sub.s #N sk (IEEE)
1116	sub.d sj sk (IEEE)
1127	neg.s sj sk (IEEE)
1128	neg.d sj sk (IEEE)
1131	mul.d sj sk (IEEE)
1132	div.s #n sk div.s sj sk (IEEE)
1133	div.d sj sk (IEEE)

Table 8-2 (continued)
cpu4030 class 1 subtests

Subtest	Test performed
1135	mul.s #N sk mul.s sj sk (IEEE)
1140	cvtw.s sj sk (IEEE)
1142	cvtw.s sj sk (IEEE)
1144	cvtd.l sj sk (IEEE)
1146	cvtl.d sj sk (IEEE)
1148	cvtl.s sj sk (IEEE)
1150	cvts.l sj sk (IEEE)
1152	cvts.d sj sk (IEEE)
1154	cvtd.s sj sk (IEEE)

Table 8-3 describes the cpu4030 class 2 subtests.

Table 8-3
cpu4030 class 2 subtests

Subtest	Test performed
200	ld.b <effa>,Ak (byte boundaries)
201	ld.h <effa>,Ak (halfword boundaries)
202	ld.w <effa>,Ak (word boundaries)
205	ld.b <effa>,Sk (byte boundaries)
206	ld.h <effa>,Sk (halfword boundaries)
207	ld.w <effa>,Sk (word boundaries)
208	ld.l <effa>,Sk (word boundaries)
211	ld.h <effa>,Ak (unaligned)
212	ld.w <effa>,Ak (unaligned)
216	ld.h <effa>,Sk (unaligned)
217	ld.w <effa>,Sk (unaligned)
218	ld.l <effa>,Sk (unaligned)
220	st.b Ak,<effa> (byte boundaries)

Table 8-3 (continued)
cpu4030 class 2 subtests

Subtest	Test performed
221	st.h Ak,<effa> (halfword boundaries)
222	st.w Ak,<effa> (word boundaries)
225	st.b Sk,<effa> (byte boundaries)
226	st.h Sk,<effa> (halfword boundaries)
227	st.w Sk,<effa> (word boundaries)
228	st.l Sk,<effa> (word boundaries)
231	st.h Ak,<effa> (unaligned boundaries)
232	st.w Ak,<effa> (unaligned boundaries)
236	st.h Sk,<effa> (unaligned boundaries)
237	st.w Sk,<effa> (unaligned boundaries)
238	st.l Sk,<effa> (unaligned boundaries)
240	psh.l Sk psh.w Sk psh.w Ak pop.w Ak pop.l Sk pop.w Sk pshea
241	psh.l Sk psh.w Sk psh.w Ak pop.w Ak pop.w Sk pop.l Sk pshea(unaligned)
242	tas <effa>
245	ldea <effa>,Ak
250	callq (aligned stack) rtnq (aligned stack)
251	callq (unaligned stack) rtnq (unaligned stack)
252	call (aligned stack) rtn (aligned stack)
253	call (unaligned stack) rtn (unaligned stack)
254	calls (aligned stack) rtn (aligned stack)
255	calls (unaligned stack) rtn (unaligned stack)
260	st.b Ak,@<effa> (byte boundaries)
261	st.h Ak,@<effa> (halfword boundaries)
262	st.w Ak,@<effa> (word boundaries)

Table 8-3 (continued)
`cpu4030` class 2 subtests

Subtest	Test performed
265	`st.b Sk,@<effa>` (byte boundaries)
266	`st.h Sk,@<effa>` (halfword boundaries)
267	`st.w Sk,@<effa>` (word boundaries)
268	`st.l Sk,@<effa>` (word boundaries)
271	`st.h Ak,@<effa>` (unaligned boundaries)
272	`st.w Ak,@<effa>` (unaligned boundaries)
276	`st.h Sk,@<effa>` (unaligned boundaries)
277	`st.w Sk,@<effa>` (unaligned boundaries)
278	`st.l Sk,@<effa>` (unaligned boundaries)
280	`ld.b @<effa>,Ak` (byte boundaries)
281	`ld.h @<effa>,Ak` (halfword boundaries)
282	`ld.w @<effa>,Ak` (word boundaries)
285	`ld.b @<effa>,Sk` (byte boundaries)
286	`ld.h @<effa>,Sk` (halfword boundaries)
287	`ld.w @<effa>,Sk` (word boundaries)
288	`ld.l @<effa>,Sk` (word boundaries)
291	`ld.h @<effa>,Ak` (unaligned)
292	`ld.w @<effa>,Ak` (unaligned)
296	`ld.h @<effa>,Sk` (unaligned)
297	`ld.w @<effa>,Sk` (unaligned)
298	`ld.l @<effa>,Sk` (unaligned)

Table 8-4 describes the cpu4030 class 3 subtests.

Table 8-4
cpu4030 class 3 subtests

Subtest	Test performed
300	Load byte across page bnd
301	Load halfword across page bnd
302	Load word across page bnd
303	Load longword across page bnd
304	Store byte across page bnd
305	Store halfword across page bnd
306	Store word across page bnd
307	Store longword across page bnd
308	Load byte @, address at page bnd
309	Load halfword @, address at page bnd
310	Load word @, address at page bnd
311	Load longword @, address at page bnd
312	Load byte @, address and data at page bnd
313	Load halfword @, address and data at page bnd
314	Load word @, address and data at page bnd
315	Load longword @, address and data at page bnd
350	Execute at page bnd
351	Execute different Size op codes at page bnd
352	Execute different Size op codes at page bnd
353	Branch near page bnd

Table 8-5 describes the cpu4030 class 4 subtests.

Table 8-5
cpu4030 class 4 subtests

Subtest	Test performed
500	Pc carry test I
505	Pc carry test II
510	Pc carry test III
515	Pc carry test IV
520	Pc wraparound test I
525	Pc wraparound test II
530	Pc wraparound test III
535	Pc wraparound test IV
540	Pc wraparound test V
545	Pc wraparound test VI
550	Pc wraparound test VII
555	Pc wraparound test VIII

8.5 Vector instruction test (cpu4041)

The cpu4041 test verifies the operation of the vector unit and its interfaces to the other C3 Series subsystems. The test exercises each vector/vector and scalar/vector instruction while varying all instruction-dependent parameters.

8.5.1 Prerequisites and required equipment

The cpu4041 test exercises the vector processor (VP) and scalar processor (SP) boards. Table 8-6 lists the boards that must be installed and previously verified to run this test. No additional equipment is necessary.

Table 8-6
Hardware required for cpu4041

Board	Test to verify
SPU workstation interface serial (SWIS)	spu4000
SPU workstation interface parallel (SWIP)	spu4000
CPU utilities (CU)	cu4000, sst, spu4000
Interface adapter (IA)	sst, spu4000
Crossbar subsystem (XRTE, XS1E, XS0E, XLL, XS00, XS10, XRT0)	sst, spu4000, mem4000
Memory subsystem (MBs)	sst, spu4000, mem4000
Scalar processor (SPs)	sst, spu4000, cpu4030

8.5.2 Invoking the test

Perform the following steps to invoke the cpu4041 test:

Step 1 From the xdiag window, select Test.

Step 2 Click on cpu4030 in the drop-down menu. The cti loads the cpu4041.info and cpu4041.par files. The default test parameters are initialized from the cpu4041.par file.

Step 3 Click on CPUcti to invoke the CPUcti Test Options window.

Step 4 Select the test options. See Section 8.2 of this document.

Step 5 Select the run button in the command bar of the xdiag window to start the test.

8.5.3 Class descriptions

cpu4041 has 4 classes of subtests, as described in the following subsections.

8.5.3.1 Class 1 subtests

Class 1 subtests verify the operation of loading, storing, and modifying the vector unit control functions. Specifically, this class verifies the ability to alter and save the vector length (VL) register, vector stride (VS) register, and the vector merge (VM) register. All subtests end with a halt command and store a halt code in address register A1. If the code stored in A1 is 0x100, then the subtest passed. Any other code in A1 indicates that the subtest failed.

8.5.3.2 Class 2 subtests

Class 2 subtests verify the operation of the logical and arithmetic pipes. Specifically, this class verifies vector/vector and scalar/vector comparisons, vector/vector and scalar/vector additions and subtractions, and vector reductions.

All subtests end with a halt command and store a halt code in address register A1. If the code stored in A1 is 0x100, then the subtest passed. Any other code in A1 indicates that the subtest failed.

8.5.3.3 Class 3 subtests

Class 3 subtests verify the operation of the multiply and divide pipe. Specifically, this class verifies the vector/vector and scalar/vector multiplications and divisions.

All subtests end with a `halt` command and store a `halt` code in address register A1. If the code stored in A1 is 0x100, then the subtest passed. Any other code in A1 indicates that the subtest failed.

8.5.3.4 Class 4 subtests

Class 4 subtests verify the operation of loading and storing vector registers. Specifically, this class verifies loading and storing the vector using direct addressing and vector of indices, and storing of vectors and scalar registers using the extended operations.

All subtests end with a `halt` command and store a halt code in address register A1. If the code stored in A1 is 0x100, then the subtest passed. Any other code in A1 indicates that the subtest failed.

8.5.4 Subtest descriptions

Table 8-7 describes the `cpu4041` class 1 subtests.

Table 8-7
`cpu4041` class 1 subtests

Subtest	Test performed
10	`ld.w #N,VL`
15	`ld.w #N,VS`
20	`mov.w Sk,VL`
25	`mov.w Sk,VS`
30	`mov Ak,VL`
35	`mov VL,Ak`
40	`mov Ak,VS`
45	`mov VS,Ak`
50	`ld.l <effa>,VLS`
51	`ld.l <effa>,VLS`
55	`st.l VLS,<effa>`
56	`st.l VLS,<effa>`
60	`mov Sj,Sk,VM`
65	`mov Sj,VM,Sk`
70	`ld.x <effa>,VM`
75	`st.x VM,<effa>`
80	`plc.t VM,Sk`
85	`plc.f VM,Sk`
195	`mov Si,Sj,Vk`
198	`mov Vi,Sj,Sk`
1000	Vector valid traps
1010	Dual vector loads

Table 8-8 describes the cpu4041 class 2 subtests.

Table 8-8
cpu4041 class 2 subtests

Subtest	Test performed
105	le.b Sj,Vk
106	le.b Sj,Vk
110	le.h Sj,Vk
111	le.h Sj,Vk
115	le.w Sj,Vk
116	le.w Sj,Vk
120	le.l Sj,Vk
121	le.l Sj,Vk
125	le.s Sj,Vk (native mode)
126	le.s Sj,Vk (native mode)
130	le.d Sj,Vk (native mode)
131	le.d Sj,Vk (native mode)
135	lt.b Sj,Vk
136	lt.b Sj,Vk
140	lt.h Sj,Vk
141	lt.h Sj,Vk
145	lt.w Sj,Vk
146	lt.w Sj,Vk
150	lt.l Sj,Vk
151	lt.l Sj,Vk
155	lt.s Sj,Vk (native mode)
156	lt.s Sj,Vk (native mode)
160	lt.d Sj,Vk (native mode)
161	lt.d Sj,Vk (native mode)
165	eq.b Sj,Vk
166	eq.b Sj,Vk

Table 8-8 (continued)
cpu4041 class 2 subtests

Subtest	Test performed
170	eq.h Sj,Vk
171	eq.h Sj,Vk
175	eq.w Sj,Vk
176	eq.w Sj,Vk
180	eq.l Sj,Vk
181	eq.l Sj,Vk
185	eq.s Sj,Vk (native mode)
186	eq.s Sj,Vk (native mode)
190	eq.d Sj,Vk (native mode)
191	eq.d Sj,Vk (native mode)
250	add.b Vi,Sj,Vk
251	add.b Vi,Sj,Vk
255	add.h Vi,Sj,Vk
256	add.h Vi,Sj,Vk
260	add.w Vi,Sj,Vk
261	add.w Vi,Sj,Vk
265	add.l Vi,Sj,Vk
266	add.l Vi,Sj,Vk
270	add.s Vi,Sj,Vk (native mode)
271	add.s Vi,Sj,Vk (native mode)
275	add.d Vi,Sj,Vk (native mode)
276	add.d Vi,Sj,Vk (native mode)
280	sub.b Vi,Sj,Vk
281	sub.b Vi,Sj,Vk
285	sub.h Vi,Sj,Vk
286	sub.h Vi,Sj,Vk
290	sub.w Vi,Sj,Vk
291	sub.w Vi,Sj,Vk

Table 8-8 (continued)
cpu4041 class 2 subtests

Subtest	Test performed
295	sub.l Vi,Sj,Vk
296	sub.l Vi,Sj,Vk
300	sub.s Vi,Sj,Vk (native mode)
301	sub.s Vi,Sj,Vk (native mode)
305	sub.d Vi,Sj,Vk (native mode)
306	sub.d Vi,Sj,Vk (native mode)
310	and Vi,Sj,Vk
311	and Vi,Sj,Vk
315	or Vi,Sj,Vk
316	or Vi,Sj,Vk
320	xor Vi,Sj,Vk
321	xor Vi,Sj,Vk
325	shf Sj,Vk
326	shf Sj,Vk
330	le.b Vj,Vk
331	le.b Vj,Vk
335	le.h Vj,Vk
336	le.h Vj,Vk
340	le.w Vj,Vk
341	le.w Vj,Vk
345	le.l Vj,Vk
346	le.l Vj,Vk
350	le.s Vj,Vk (native mode)
351	le.s Vj,Vk (native mode)
355	le.d Vj,Vk (native mode)
356	le.d Vj,Vk (native mode)
360	lt.b Vj,Vk
361	lt.b Vj,Vk

Table 8-8 (continued)
cpu4041 class 2 subtests

Subtest	Test performed
365	lt.h Vj,Vk
366	lt.h Vj,Vk
370	lt.w Vj,Vk
371	lt.w Vj,Vk
375	lt.l Vj,Vk
376	lt.l Vj,Vk
380	lt.s Vj,Vk (native mode)
381	lt.s Vj,Vk (native mode)
385	lt.d Vj,Vk (native mode)
386	lt.d Vj,Vk (native mode)
390	eq.b Vj,Vk
391	eq.b Vj,Vk
395	eq.h Vj,Vk
396	eq.h Vj,Vk
400	eq.w Vj,Vk
401	eq.w Vj,Vk
405	eq.l Vj,Vk
406	eq.l Vj,Vk
410	eq.s Vj,Vk (native mode)
411	eq.s Vj,Vk (native mode)
415	eq.d Vj,Vk (native mode)
416	eq.d Vj,Vk (native mode)
430	add.b Vi,Vj,Vk
431	add.b Vi,Vj,Vk
435	add.h Vi,Vj,Vk
436	add.h Vi,Vj,Vk
440	add.w Vi,Vj,Vk
441	add.w Vi,Vj,Vk

Table 8-8 (continued)
cpu4041 class 2 subtests

Subtest	Test performed
445	add.l Vi,Vj,Vk
446	add.l Vi,Vj,Vk
450	add.s Vi,Vj,Vk (native mode)
451	add.s Vi,Vj,Vk (native mode)
456	add.d Vi,Vj,Vk (native mode)
460	sub.b Vi,Vj,Vk
461	sub.b Vi,Vj,Vk
465	sub.h Vi,Vj,Vk
466	sub.h Vi,Vj,Vk
470	sub.w Vi,Vj,Vk
471	sub.w Vi,Vj,Vk
475	sub.l Vi,Vj,Vk
476	sub.l Vi,Vj,Vk
480	sub.s Vi,Vj,Vk (native mode)
481	sub.s Vi,Vj,Vk (native mode)
485	sub.d Vi,Vj,Vk (native mode)
486	sub.d Vi,Vj,Vk (native mode)
490	and Vi,Vj,Vk
491	and Vi,Vj,Vk
495	or Vi,Vj,Vk
496	or Vi,Vj,Vk
500	xor Vi,Vj,Vk
501	xor Vi,Vj,Vk
505	not Vj,Vk
506	not Vj,Vk
510	neg.b Vj,Vk
511	neg.b Vj,Vk
515	neg.h Vj,Vk

Table 8-8 (continued)
cpu4041 class 2 subtests

Subtest	Test performed
516	neg.h Vj,Vk
520	neg.w Vj,Vk
521	neg.w Vj,Vk
525	neg.l Vj,Vk
526	neg.l Vj,Vk
530	neg.s Vj,Vk (native mode)
531	neg.s Vj,Vk (native mode)
535	neg.d Vj,Vk (native mode)
536	neg.d Vj,Vk (native mode)
770	merg.t Vi,Sj,Vk
771	merg.t Vi,Sj,Vk
775	merg.f Vi,Sj,Vk
776	merg.f Vi,Sj,Vk
780	mask.t Vi,Sj,Vk
781	mask.t Vi,Sj,Vk
785	mask.f Vi,Sj,Vk
786	mask.f Vi,Sj,Vk
790	merg.t Vi,Vj,Vk
791	merg.t Vi,Vj,Vk
800	mask.t Vi,Vj,Vk
801	mask.t Vi,Vj,Vk
805	plc.t Vj,Vk
806	plc.t Vj,Vk
810	cprs.t Vi,Vj,Vk
811	cprs.t Vi,Vj,Vk
815	cprs.t Vi,Vj,Vk
816	cprs.t Vi,Vj,Vk
820	sum.b Vk

Table 8-8 (continued)
cpu4041 class 2 subtests

Subtest	Test performed
821	sum.b Vk
825	sum.h Vk
826	sum.h Vk
830	sum.w Vk
831	sum.w Vk
835	sum.l Vk
836	sum.l Vk
840	sum.s Vk (native mode)
841	sum.s Vk (native mode)
845	sum.d Vk (native mode)
846	sum.d Vk (native mode)
880	max.b Vk
881	max.b Vk
885	max.h Vk
886	max.h Vk
890	max.w Vk
891	max.w Vk
895	max.l Vk
896	max.l Vk
900	max.s Vk (native mode)
901	max.s Vk (native mode)
905	max.d Vk (native mode)
906	max.d Vk (native mode)
910	min.b Vk
911	min.b Vk
915	min.h Vk
916	min.h Vk
920	min.w Vk

Table 8-8 (continued)
cpu4041 class 2 subtests

Subtest	Test performed
921	min.w Vk
925	min.l Vk
926	min.l Vk
930	min.s Vk (native mode)
931	min.s Vk (native mode)
935	min.d Vk (native mode)
936	min.d Vk (native mode)
940	all Vk
941	all Vk
950	any Vk
951	any Vk
960	parity Vk
961	parity Vk
1125	le.s Sj,Vk (IEEE mode)
1126	le.s Sj,Vk (IEEE mode)
1130	le.d Sj,Vk (IEEE mode)
1131	le.d Sj,Vk (IEEE mode)
1155	lt.s Sj,Vk (IEEE mode)
1156	lt.s Sj,Vk (IEEE mode)
1160	lt.d Sj,Vk (IEEE mode)
1161	lt.d Sj,Vk (IEEE mode)
1185	eq.s Sj,Vk (IEEE mode)
1186	eq.s Sj,Vk (IEEE mode)
1190	eq.d Sj,Vk (IEEE mode)
1191	eq.d Sj,Vk (IEEE mode)
1270	add.s Vi,Sj,Vk (IEEE mode)
1271	add.s Vi,Sj,Vk (IEEE mode)

Table 8-8 (continued)
cpu4041 class 2 subtests

Subtest	Test performed
1275	add.d Vi,Sj,Vk (IEEE mode)
1276	add.d Vi,Sj,Vk (IEEE mode)
1300	sub.s Vi,Sj,Vk (IEEE mode)
1301	sub.s Vi,Sj,Vk (IEEE mode)
1305	sub.d Vi,Sj,Vk (IEEE mode)
1306	sub.d Vi,Sj,Vk (IEEE mode)
1350	le.s Vj,Vk (IEEE mode)
1351	le.s Vj,Vk (IEEE mode)
1355	le.d Vj,Vk (IEEE mode)
1356	le.d Vj,Vk (IEEE mode)
1380	lt.s Vj,Vk (IEEE mode)
1381	lt.s Vj,Vk (IEEE mode)
1385	lt.d Vj,Vk (IEEE mode)
1386	lt.d Vj,Vk (IEEE mode)
1410	eq.s Vj,Vk (IEEE mode)
1411	eq.s Vj,Vk (IEEE mode)
1415	eq.d Vj,Vk (IEEE mode)
1416	eq.d Vj,Vk (IEEE mode)
1450	add.s Vi,Vj,Vk (IEEE mode)
1451	add.s Vi,Vj,Vk (IEEE mode)
1455	add.d Vi,Vj,Vk (IEEE mode)
1456	add.d Vi,Vj,Vk (IEEE mode)
1480	sub.s Vi,Vj,Vk (IEEE mode)
1481	sub.s Vi,Vj,Vk (IEEE mode)
1485	sub.d Vi,Vj,Vk (IEEE mode)
1486	sub.d Vi,Vj,Vk (IEEE mode)
1530	neg.s Vj,Vk (IEEE mode)
1531	neg.s Vj,Vk (IEEE mode)

Table 8-8 (continued)
cpu4041 class 2 subtests

Subtest	Test performed
1535	neg.d Vj,Vk (IEEE mode)
1536	neg.d Vj,Vk (IEEE mode)
1840	sum.s Vk (IEEE mode)
1841	sum.s Vk (IEEE mode)
1845	sum.d Vk (IEEE mode)
1846	sum.d Vk (IEEE mode)
1900	max.s Vk (IEEE mode)
1901	max.s Vk (IEEE mode)
1905	max.d Vk (IEEE mode)
1906	max.d Vk (IEEE mode)
1930	min.s Vk (IEEE mode)
1931	min.s Vk (IEEE mode)
1935	min.d Vk (IEEE mode)
1936	min.d Vk (IEEE mode)

Table 8-9 describes the cpu4041 class 3 subtests.

Table 8-9
cpu4041 class 3 subtests

Subtest	Test performed
630	mul.b Vi,Sj,Vk
631	mul.b Vi,Sj,Vk
635	mul.h Vi,Sj,Vk
636	mul.h Vi,Sj,Vk
640	mul.w Vi,Sj,Vk
641	mul.w Vi,Sj,Vk
645	mul.l Vi,Sj,Vk
646	mul.l Vi,Sj,Vk
650	mul.s Vi,Sj,Vk (native mode)
651	mul.s Vi,Sj,Vk (native mode)
655	mul.d Vi,Sj,Vk (native mode)
656	mul.d Vi,Sj,Vk (native mode)
660	div.b Vi,Sj,Vk
661	div.b Vi,Sj,Vk
665	div.h Vi,Sj,Vk
666	div.h Vi,Sj,Vk
670	div.w Vi,Sj,Vk
671	div.w Vi,Sj,Vk
675	div.l Vi,Sj,Vk
676	div.l Vi,Sj,Vk
680	div.s Vi,Sj,Vk (native mode)
681	div.s Vi,Sj,Vk (native mode)
685	div.d Vi,Sj,Vk (native mode)
686	div.d Vi,Sj,Vk (native mode)
700	mul.b Vi,Vj,Vk
701	mul.b Vi,Vj,Vk

Table 8-9 (continued)
cpu4041 class 3 subtests

Subtest	Test performed
705	mul.h Vi,Vj,Vk
706	mul.h Vi,Vj,Vk
710	mul.w Vi,Vj,Vk
711	mul.w Vi,Vj,Vk
715	mul.l Vi,Vj,Vk
716	mul.l Vi,Vj,Vk
720	mul.s Vi,Vj,Vk (native mode)
721	mul.s Vi,Vj,Vk (native mode)
725	mul.d Vi,Vj,Vk (native mode)
726	mul.d Vi,Vj,Vk (native mode)
740	div.b Vi,Vj,Vk
741	div.b Vi,Vj,Vk
745	div.h Vi,Vj,Vk
746	div.h Vi,Vj,Vk
750	div.w Vi,Vj,Vk
751	div.w Vi,Vj,Vk
755	div.l Vi,Vj,Vk
756	div.l Vi,Vj,Vk
760	div.s Vi,Vj,Vk (native mode)
761	div.s Vi,Vj,Vk (native mode)
765	div.d Vi,Vj,Vk (native mode)
766	div.d Vi,Vj,Vk (native mode)
850	prod.b Vk
851	prod.b Vk
855	prod.h Vk
856	prod.h Vk
860	prod.w Vk
861	prod.w Vk

Table 8-9 (continued)
cpu4041 class 3 subtests

Subtest	Test performed
865	prod.l Vk
866	prod.l Vk
870	prod.s Vk (native mode)
871	prod.s Vk (native mode)
875	prod.d Vk (native mode)
876	prod.d Vk (native mode)
1650	mul.s Vi,Sj,Vk (IEEE mode)
1651	mul.s Vi,Sj,Vk (IEEE mode)
1655	mul.d Vi,Sj,Vk (IEEE mode)
1656	mul.d Vi,Sj,Vk (IEEE mode)
1680	div.s Vi,Sj,Vk (IEEE mode)
1681	div.s Vi,Sj,Vk (IEEE mode)
1685	div.d Vi,Sj,Vk (IEEE mode)
1686	div.d Vi,Sj,Vk (IEEE mode)
1720	mul.s Vi,Vj,Vk (IEEE mode)
1721	mul.s Vi,Vj,Vk (IEEE mode)
1725	mul.d Vi,Vj,Vk (IEEE mode)
1726	mul.d Vi,Vj,Vk (IEEE mode)
1760	div.s Vi,Vj,Vk (IEEE mode)
1761	div.s Vi,Vj,Vk (IEEE mode)
1765	div.d Vi,Vj,Vk (IEEE mode)
1766	div.d Vi,Vj,Vk (IEEE mode)
1870	prod.s Vk (IEEE mode)
1871	prod.s Vk (IEEE mode)
1875	prod.d Vk (IEEE mode)
1876	prod.d Vk (IEEE mode)

Table 8-10
cpu4041 class 4 subtests

Subtest	Test performed
200	ld.b <effa>,Vk
201	ld.b <effa>,Vk
205	ld.h <effa>,Vk
206	ld.h <effa>,Vk
210	ld.w <effa>,Vk
211	ld.w <effa>,Vk
215	ld.l <effa>,Vk
216	ld.l <effa>,Vk
230	st.b Vk,<effa>
231	st.b Vk,<effa>
235	st.h Vk,<effa>
236	st.h Vk,<effa>
240	st.w Vk,<effa>
241	st.w Vk,<effa>
245	st.l Vk,<effa>
246	st.l Vk,<effa>
550	ldvi.b Vj,Vk
551	ldvi.b Vj,Vk
555	ldvi.h Vj,Vk
556	ldvi.h Vj,Vk
560	ldvi.w Vj,Vk
561	ldvi.w Vj,Vk
565	ldvi.l Vj,Vk
566	ldvi.l Vj,Vk
570	stvi.b Vi,Vj
571	stvi.b Vi,Vj
575	stvi.h Vi,Vj
576	stvi.h Vi,Vj

Table 8-10 (continued)
cpu4041 class 4 subtests

Subtest	Test performed
580	stvi.w Vi,Vj
581	stvi.w Vi,Vj
585	stvi.l Vi,Vj
586	stvi.l Vi,Vj
590	stvi.b Si,Vj
591	stvi.b Si,Vj
595	stvi.h Si,Vj
596	stvi.h Si,Vj
600	stvi.w Si,Vj
601	stvi.w Si,Vj
605	stvi.l Si,Vj
606	stvi.l Si,Vj
610	ste.b Sk,<effa>
611	ste.b Sk,<effa>
615	ste.h Sk,<effa>
616	ste.h Sk,<effa>
620	ste.w Vk,<effa>
621	ste.w Vk,<effa>
625	ste.l Vk,<effa>
626	ste.l Vk,<effa>

8.6 Privileged instruction and architectural features (cpu4331)

The cpu4331 test verifies nonvector, single processor architectural features. Included are tests of system exceptions, interrupts, privileged instructions, the various processor caches, remote invalidates, and nonresident memory pages.

8.6.1 Prerequisites and required equipment

The vector processor (VP) and scalar processor (SP) boards must be present with only the scalar processor being tested, when running the cpu4331 test. Table 8-11 lists the boards that must be installed and previously verified to run this test. No additional equipment is required to run this test.

Table 8-11
Hardware required for cpu4331

Board	Test to verify
SPU workstation interface serial (SWIS)	spu4000
SPU workstation interface parallel (SWIP)	spu4000
CPU utilities (CU)	cu4000, sst, spu4000
Interface adapter (IA)	sst, spu4000
Crossbar subsystem (XRTE, XS1E, XS0E, XLL, XS00, XS10, XRT0)	sst, spu4000, mem4000
Memory subsystem (MBs)	sst, spu4000, mem4000
Scalar processor (SPs)	sst, spu4000, cpu4030
Vector processor (VPs)	sst, spu4000, cpu4041

8.6.2 Invoking the test

Perform the following steps to invoke the `cpu4331` test:

Step 1 From the `xdiag` window, select `Test`.

Step 2 Click on `cpu4331` in the drop-down menu. The `cti` loads the cpu4331.info and cpu4331.par files. The default test parameters are initialized from the cpu4331.par file.

Step 3 Click on `CPUcti` to invoke the `CPUcti Test Options` window.

Step 4 Select the test options. See Section 8.2 of this document.

Step 5 Select the `run` button in the command bar of the `xdiag` window to start the test.

8.6.3 Class descriptions

`cpu4331` has 4 classes of subtests, as described in the following subsections.

8.6.3.1 Class 1 subtests

This class of subtests verifies various instructions and features unique to the C3 Series architecture. Included are tests of system calls, C3 Series specific nonvector instructions, thread-level addressing, exceptions, interval timers, and privileged instructions.

8.6.3.2 Class 2 subtests

Class 2 subtests verify proper operation of page faults and nonresident calls, returns, and instructions. Class 2 subtests also verify proper operation of subroutine calls and returns for cases where the code and/or the stack are in nonresident memory.

8.6.3.3 Class 3 subtests

Class 3 subtests verify proper operation of memory operations (loads and stores) for various combinations of conditions (byte, halfword, word, longword, and nonresident).

8.6.3.4 Class 4 subtests

Class 4 subtests verify the proper operation of the various caches in the C3 Series architecture (PTE cache, instruction cache, and data cache). These subtests also test remote invalidates between processors.

8.6.4 Subtest descriptions

Table 8-12 describes the `cpu4331` class 1 subtests.

Table 8-12
`cpu4331` class 1 subtests

Subtest	Test performed
10	System calls
20	`ldpa aj,ak`
21	`Patu` instruction test
22	`Pate ak` instruction test
23	Thread addressing
25	`Pich` instruction test
26	`Patu`/TER instruction test
27	`Patu`/TER instruction test
30	Traps
31	Deadlocks
32	Unimplemented op codes
35	Exceptions
36	Invalid cmr addresses
37	Trap instructions test
38	Trap/TER instruction test
43	Timer CIR switch check
44	Ring crossing faulting
45	Privileged instruction check
46	C3 interval timer test
47	Timer ring cross check
600	Test vector valid test
601	Test `mov sk,vv` test
602	Test nonvector valid trapping just after setting vv
605	`mov sk,toc`
700	Ring crossings with trap bits set

Table 8-12 (continued)
cpu4331 class 1 subtests

Subtest	Test performed
701	Ring crossings with pbkpt bits set
900	Interrupt—dsi/eni
901	Interrupt—register data retention
902	Interrupt—xmti
903	Interrupt—xmti
904	Interrupt—xmti
906	Interrupt—xmti
907	Interrupt—xmti
908	Interrupt—xmti
909	Interrupt—xmti
910	Interrupt—xmti

Table 8-13 describes the cpu4331 class 2 subtests.

Table 8-13
cpu4331 class 2 subtests

Subtest	Test performed
50	callq(abs/@) into nonresident page
51	Rtnq into nonresident page
52	callq with nonresident stack
53	rtnq with nonresident stack
60	call (abs/@) into nonresident page
61	rtn into nonresident page
62	call (abs/@) with nonresident stack
63	rtn with nonresident stack
70	calls (abs/@) into nonresident page
71	rtn into nonresident page
72	calls with nonresident stack

Table 8-13 (continued)
`cpu4331` class 2 subtests

Subtest	Test performed
73	`rtn` with nonresident stack
80	Nonresident stack, nonresident target page (`callq`)
81	Nonresident stack, nonresident target page (`calls`)
82	Nonresident stack, nonresident target page (`call`)
85	Indirect nonresident, nonresident target page (`callq`)
86	Indirect nonresident, nonresident target page (`calls`)
87	Indirect nonresident, nonresident target page (`call`)
90	Indirect nonresident, nonresident stack (`callq`)
91	Indirect nonresident, nonresident stack (`calls`)
92	Indirect nonresident, nonresident stack (`call`)
95	Nonresident stack, nonresident return page (`callq`)
96	Nonresident stack, nonresident return page (`calls`)
97	Nonresident stack, nonresident return page (`call`)
100	Nonresident stack, nonresident target page, nonresident indirect address (`callq`)
101	Nonresident stack, nonresident target page, nonresident indirect address (`calls`)
102	Nonresident stack, nonresident target page, nonresident indirect address (`call`)
200	Halfword, nonresident page execute
201	Word, nonresident page execute
202	3-halfword, nonresident page execute
322	IP lookahead faults
323	Page fault combination test I
324	Page fault combination test II
325	Page fault combination test III
326	Page fault combination test IV

Table 8-14 describes the cpu4331 class 3 subtests.

Table 8-14
cpu4331 class 3 subtests

Subtest	Test performed
210	Byte loads nonresident page
220	Load halfword nonresident page
221	Load halfword nonresident page
222	Load halfword nonresident page
223	Load halfword nonresident page
224	Load halfword nonresident page
225	Load halfword nonresident page
226	Load halfword nonresident page
227	Load halfword nonresident page
228	Load halfword nonresident page
230	Load word nonresident page
231	Load word nonresident page
232	Load word nonresident page
233	Load word nonresident page
234	Load word nonresident page
235	Load word nonresident page
236	Load word nonresident page
237	Load word nonresident page
238	Load word nonresident page
240	Load longword nonresident page
241	Load longword nonresident page
242	Load longword nonresident page
243	Load longword nonresident page
244	Load longword nonresident page
245	Load longword nonresident page
246	Load longword nonresident page

Table 8-14 (continued)
cpu4331 class 3 subtests

Subtest	Test performed
247	Load longword nonresident page
248	Load longword nonresident page
250	Store byte nonresident page
260	Store halfword nonresident page
261	Store halfword nonresident page
270	Store word nonresident page
271	Store word nonresident page
272	Store word nonresident page
273	Store word nonresident page
274	Store word nonresident page
275	Store word nonresident page
276	Store word nonresident page
277	Store word nonresident page
278	Store word nonresident page
280	Store longword nonresident page
281	Store longword nonresident page
282	Store longword nonresident page
283	Store longword nonresident page
284	Store longword nonresident page
285	Store longword nonresident page
286	Store longword nonresident page
287	Store longword nonresident page
288	Store longword nonresident page
290	Load byte indirect, address nonresident
291	Load halfword indirect, address nonresident
292	Load word indirect, address nonresident
293	Load longword indirect, address nonresident

Table 8-14 (continued)
cpu4331 class 3 subtests

Subtest	Test performed
300	Store byte indirect, address nonresident
301	Store halfword indirect, address nonresident
302	Store word indirect, address nonresident
303	Store longword indirect, address nonresident
305	Load byte indirect, address and data nonresident
306	Load byte indirect, address and data nonresident
307	Load byte indirect, address and data nonresident
308	Load byte indirect, address and data nonresident
310	Store byte indirect, address and data nonresident
311	Store byte indirect, address and data nonresident
312	Store byte indirect, address and data nonresident
313	Store byte indirect, address and data nonresident
320	Multiple fault test

Table 8-15 describes the cpu4331 class 4 subtests.

Table 8-15
cpu4331 class 4 subtests

Subtest	Test performed
400	PTE cache test
500	Icache test I
501	Icache test II
502	Icache test III
503	Icache test IV
504	Icache test V
505	Icache test VI
506	Icache test VII
507	Icache test VII

Table 8-15 (continued)
`cpu4331` class 4 subtests

Subtest	Test performed
508	Icache test VII
509	Icache test VII
510	Dcache test I
511	Dcache test II
512	Dcache test III
513	Dcache test IV
514	Dcache test V
515	Dcache test VI
516	Dcache test VII
517	Dcache test VIII
521	Dcache test XII
523	Dcache test XIII
525	Dcache test XIV
526	Dcache test XV
527	Dcache test XVI
528	Dcache test XVII
529	Dcache test XVIII
530	Dcache test XVIII
606	Load bypass instructions

8.7 Enhanced, nonvector, single processor instruction tests (cpu4332)

The cpu4332 test is an extension of the building block test cpu4030. This test provides exhaustive, single CPU testing of the scalar, address, and communication register instructions. It performs the following tests:

1. Verifies the operation of the instructions that manipulate the communications registers. The locking/unlocking, putting/getting, sending/receiving, matching/testing, and incrementing all 8 communication register sets are verified to function correctly.
2. Verifies the sending/receiving, pushing/popping, matching/incrementing, and testing-and-clearing operations on memory.
3. Verifies the scalar instructions (scalar converts, loading/storing of the communication index register (CIR), thread identifier register (TID), thread timer register (TTR); and loading of the central processor unit identifier (CPUID).
4. Verifies the multiprocessor instructions pfork, cfork, wfork, spawn, join, and idle.
5. Verifies the remaining instructions trap, pbkpt, ldcmr, stcmr, ctrsl, and ctrsg.

8.7.1 Prerequisites and required equipment

The vector processor (VP) and scalar processor (SP) boards must be present with only the scalar processor being tested, when running the cpu4332 test. Table 8-16 lists the boards that must be installed and previously verified to run this test. No additional equipment is required to run this test.

Table 8-16
Hardware required for `cpu4332`

Board	Test to verify
SPU workstation interface serial (SWIS)	`spu4000`
SPU workstation interface parallel (SWIP)	`spu4000`
CPU utilities (CU)	`cu4000, sst, spu4000`
Interface adapter (IA)	`sst, spu4000`
Crossbar subsystem (XRTE, XS1E, XS0E, XLL, XS00, XS10, XRT0)	`sst, spu4000, mem4000`
Memory subsystem (MBs)	`sst, spu4000, mem4000`
Scalar processor	`sst, spu4000, cpu4030`
Vector processor	`sst, spu4000, cpu4041`
Scalar and vector processors	`sst, spu4000, cpu4041, cpu4331`

8.7.2 Invoking the test

Perform the following steps to invoke the `cpu4332` test:

Step 1 From the `xdiag` window, select `Test`.

Step 2 Click on `cpu4332` in the drop-down menu. The `cti` loads the cpu4332.info and cpu4332.par files. The default test parameters are initialized from the cpu4332.par file.

Step 3 Click on `CPUcti` to invoke the `CPUcti Test Options` window.

Step 4 Select the test options. See Section 8.2 of this document. Select the `run` button in the command bar of the `xdiag` window to start the test.

8.7.3 Class descriptions

`cpu4332` has 5 classes of subtests, as described in the following subsections.

All subtests end with a `halt` command and store a `halt` code in address register A1. If the code stored in A1 is 0x100, then the subtest passed. Any other code in A1 indicates that the subtest failed.

8.7.3.1 Class 1 subtests

Class 1 subtests verify the ability of a single CPU to correctly manipulate each of the testable communication registers. For an explanation of valid manipulations on the communication registers, refer to the *CONVEX Architecture Reference Manual (C Series).*

8.7.3.2 Class 2 subtests

Class 2 subtests verify the correct manipulation of the C3 Series memory structures in a single CPU environment. For an explanation of memory structures and how they can be manipulated, refer to the *CONVEX Architecture Reference Manual (C Series)* and the *CONVEX Assembly Language Reference Manual (C Series).*

8.7.3.3 Class 3 subtests

Class 3 subtests verify the execution of the new C3 Series scalar instructions in a single CPU environment. For a detailed explanation of each of the new scalar instructions, refer to the *CONVEX Architecture Reference Manual (C Series)* and the *CONVEX Assembly Language Reference Manual (C Series).*

8.7.3.4 Class 4 subtests

Class 4 subtests verify the operation of the C3 Series process control instructions execution in a single CPU environment. For a detailed description of what the process control instructions are, and how they are required to function, refer to the *CONVEX Architecture Reference Manual (C Series)* and the *CONVEX Assembly Language Reference Manual (C Series).*

8.7.3.5 Class 5 subtests

Class 5 subtests verify the operation of miscellaneous instructions in a single CPU environment. Specifically, process trapping instructions, loading and storing of the communication registers, and the CPU execution timer synchronization instructions are verified to function correctly in a single CPU environment.

8.7.4 Subtest descriptions

Table 8-17 describes the cpu4332 class 1 subtests.

Table 8-17
cpu4332 class 1 subtests

Subtest	Test performed
100	lck <Ceffa>
105	ulk <Ceffa>
110	put.w Ak,<Ceffa>
115	put.l Sk,<Ceffa>
120	get.w <Ceffa>,Ak
125	get.l <Ceffa>,Sk
130	snd.w Ak,<Ceffa>
135	snd.l Sk,<Ceffa>
140	rcv.w <Ceffa>,Ak
145	rcv.l <Ceffa>,Sk
150	mat.w Ak,<Ceffa>
151	matr.w Ak,<Ceffa>
155	mat.l Sk,<Ceffa>
156	matr.l Sk,<Ceffa>
170	tst <Ceffa>
180	inc.w <Ceffa>,Ak
185	inc.l <Ceffa>,Sk

Table 8-18 describes the cpu4332 class 2 subtests.

Table 8-18
cpu4332 class 2 subtests

Subtest	Test performed
111	putr.w Ak,<Ceffa>
116	putr.l Sk,<Ceffa>
121	getr.w <Ceffa>,Ak
126	getr.l <Ceffa>,Sk
131	sndr.w Ak,<effa>
136	sndr.l Sk,<effa>
141	rcvr.w <effa>,Ak
146	rcvr.l <effa>,Sk
151	matr.w Ak,<Ceffa>
156	matr.l Sk,<Ceffa>
175	tac <effa>
190	pshr <effa>,Ak
195	popr Ak,<effa>
200	incr.w <effa>,Ak
205	incr.l <effa>,Sk

Table 8-19 describes the cpu4332 class 3 subtests.

Table 8-19
cpu4332 class 3 subtests

Subtest	Test performed
211	mov Sk,TTR
212	mov TTR,Sk
215	mov CPUID,Sk
225	mov TID,Sk
230	mov Sk,TID

Table 8-19 (continued)
cpu4332 class 3 subtests

Subtest	Test performed
290	mov Sk,Cir
295	mov Cir,Sk
305	ldea <effa>,Sk
310	shf.w Sj,Sk
320	casr
400	cvtd.w Sj,Sk (native)
401	cvtw.d Sj,Sk (native)
420	frint.s Sj,Sk (native)
425	frint.d Sj,Sk (native)
500	sqrt.s Sk (native)
510	atan.s Sk (native)
520	exp.s Sk (native)
530	ln.s Sk (native)
540	sin.s Sk (native)
550	cos.s Sk (native)
600	sqrt.d Sk (native)
610	atan.d Sk (native)
620	exp.d Sk (native)
630	ln.d Sk (native)
640	sin.d Sk (native)
650	cos.d Sk (native)
1400	cvtd.w Sj,Sk (IEEE)
1401	cvtw.d Sj,Sk (IEEE)
1420	frint.s Sj,Sk (IEEE)
1425	frint.d Sj,Sk (IEEE)
1500	sqrt.s Sk (IEEE)
1510	atan.s Sk (IEEE)
1520	exp.s Sk (IEEE)

Table 8-19 (continued)
cpu4332 class 3 subtests

Subtest	Test performed
1530	ln.s Sk (IEEE)
1540	sin.s Sk (IEEE)
1550	cos.s Sk (IEEE)
1600	sqrt.d Sk (IEEE)
1610	atan.d Sk (IEEE)
1620	exp.d Sk (IEEE)
1630	ln.d Sk (IEEE)
1640	sin.d Sk (IEEE)
1650	cos.d Sk (IEEE)

Table 8-20 describes the cpu4332 class 4 subtests.

Table 8-20
cpu4332 class 4 subtests

Subtest	Test performed
250	pfork <effa>,Ak
253	spawn <effa>,Ak
255	cfork
260	wfork
263	join
265	idle Sk

Table 8-21 describes the cpu4332 class 5 subtests.

Table 8-21
cpu4332 class 5 subtests

Subtest	Test performed
220	trap #rm, #b
225	mov TID, Sk
235	pbkpt
245	ctrsg
246	TER/ctrsg
271	ldcmr <effa>, Ak
276	stcmr <effa>, Ak

8.8 Enhanced vector instruction tests (cpu4241)

The cpu4241 test is a group of vector instruction subtests used to verify the operation of C3 Series vector and vector-under-mask instructions along with their interfaces to other C3 Series subsystems. The test exercises each vector/vector and scalar/vector instruction while varying all of the parameters on which the instructions depend.

8.8.1 Prerequisites and required equipment

The vector processor (VP) and scalar processor (SP) boards must be present with only the vector processor being tested, when running the cpu4241 test. Table 8-22 lists the boards that must be installed and previously verified to run this test. No additional equipment is required to run this test.

Table 8-22
Hardware required for cpu4241

Board	Test to verify
SPU workstation interface serial (SWIS)	spu4000
SPU workstation interface parallel (SWIP)	spu4000
CPU utilities (CU)	cu4000, sst, spu4000
Interface adapter (IA)	sst, spu4000
Crossbar subsystem (XRTE, XS1E, XS0E, XLL, XS00, XS10, XRT0)	sst, spu4000, mem4000
Memory subsystem (MBs)	sst, spu4000, mem4000
Scalar processor	sst, spu4000, cpu4030
Vector processor	sst, spu4000, cpu4041

8.8.2 Invoking the test

Perform the following steps to invoke the `cpu4241` test:

Step 1 From the `xdiag` window, select `Test`.

Step 2 Click on `cpu4241` in the drop-down menu. The `cti` loads the cpu4241.info and cpu4241.par files.

Step 3 Click on `CPUcti` to invoke the `CPUcti Test Options` window.

Step 4 Select the test options. See Section 8.2 of this document.

Step 5 Select the `run` button in the command bar of the `xdiag` window to start the test.

The default test parameters are initialized from the cpu4241.par file.

8.8.3 Class descriptions

`cpu4241` has 4 classes of subtests, as described in the following subsections.

All subtests end with a `halt` command and store a halt code in address register A1. If the code stored in A1 is 0x100, then the subtest passed. Any other code in A1 indicates that the subtest failed.

8.8.3.1 Class 1 subtests

Class 1 subtests verify the operation of loading, storing, and modifying the vector unit control functions. Specifically, this class verifies the ability to alter and save the vector length (VL) register, vector stride (VS) register, and the vector merge (VM) register.

8.8.3.2 Class 2 subtests

Class 2 subtests verify the operation of the logical and arithmetic pipes. Specifically, this class verifies vector/vector and scalar/vector comparisons, vector/vector and scalar/vector additions and subtractions, and vector reductions.

8.8.3.3 Class 3 subtests

Class 3 subtests verify the operation of the multiply and divide pipe. Specifically, this class verifies the vector/vector and scalar/vector multiplications and divisions.

8.8.3.4 Class 4 subtests

Class 4 subtests verify the operation of loading and storing vector registers. Specifically, this class verifies loading and storing the vector using direct addressing and vector of indices, and storing of vectors and scalar registers using the extended operations.

8.8.4 Subtest descriptions

Table 8-23 describes the `cpu4241` class 1 subtests.

Table 8-23
`cpu4241` class 1 subtests

Subtest	Test performed
21	`mov.w VL,Sk`
26	`mov.w VS,Sk`
61	`mov Sk,VM(U|L)`
62	`mov VM(U|L),Sk`
1005	Vector valid test

Table 8-24 describes the `cpu4241` class 2 subtests.

Table 8-24
`cpu4241` class 2 subtests

Subtest	Test performed
2105	`le.b.t Sj,Vk`
2106	`le.b.t Sj,Vk`
4105	`le.b.f Sj,Vk`
4106	`le.b.f Sj,Vk`
2110	`le.h.t Sj,Vk`
2111	`le.h.t Sj,Vk`
4110	`le.h.f Sj,Vk`
4111	`le.h.f Sj,Vk`
2115	`le.w.t Sj,Vk`
2116	`le.w.t Sj,Vk`
4115	`le.w.f Sj,Vk`
4116	`le.w.f Sj,Vk`
2120	`le.l.t Sj,Vk`
2121	`le.l.t Sj,Vk`
4120	`le.l.f Sj,Vk`
4121	`le.l.f Sj,Vk`
2125	`le.s.t Sj,Vk` (native)
2126	`le.s.t Sj,Vk` (native)
3125	`le.s.t Sj,Vk` (IEEE)
3126	`le.s.t Sj,Vk` (IEEE)
4125	`le.s.f Sj,Vk` (native)
4126	`le.s.f Sj,Vk` (native)
5125	`le.s.f Sj,Vk` (IEEE)
5126	`le.s.f Sj,Vk` (IEEE)
2130	`le.d.t Sj,Vk` (native)
2131	`le.d.t Sj,Vk` (native)
3130	`le.d.t Sj,Vk` (IEEE)
3131	`le.d.t Sj,Vk` (IEEE)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
4130	le.d.f Sj,Vk (native)
4131	le.d.f Sj,Vk (native)
5130	le.d.f Sj,Vk (IEEE)
5131	le.d.f Sj,Vk (IEEE)
2135	lt.b.t Sj,Vk
2136	lt.b.t Sj,Vk
4135	lt.b.f Sj,Vk
4136	lt.b.f Sj,Vk
2140	lt.h.t Sj,Vk
2141	lt.h.t Sj,Vk
4140	lt.h.f Sj,Vk
4141	lt.h.f Sj,Vk
2145	lt.w.t Sj,Vk
2146	lt.w.t Sj,Vk
4145	lt.w.f Sj,Vk
4146	lt.w.f Sj,Vk
2150	lt.l.t Sj,Vk
2151	lt.l.t Sj,Vk
4150	lt.l.f Sj,Vk
4151	lt.l.f Sj,Vk
2155	lt.s.t Sj,Vk (native)
2156	lt.s.t Sj,Vk (native)
3155	lt.s.t Sj,Vk (IEEE)
3156	lt.s.t Sj,Vk (IEEE)
4155	lt.s.f Sj,Vk (native)
4156	lt.s.f Sj,Vk (native)
5155	lt.s.f Sj,Vk (IEEE)
5156	lt.s.f Sj,Vk (IEEE)
2160	lt.d.t Sj,Vk (native)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
2161	lt.d.t Sj,Vk (native)
3160	lt.d.t Sj,Vk (IEEE)
3161	lt.d.t Sj,Vk (IEEE)
4160	lt.d.f Sj,Vk (native)
4161	lt.d.f Sj,Vk (native)
5160	lt.d.f Sj,Vk (IEEE)
5161	lt.d.f Sj,Vk (IEEE)
2165	eq.b.t Sj,Vk
2166	eq.b.t Sj,Vk
4165	eq.b.f Sj,Vk
4166	eq.b.f Sj,Vk
2170	eq.h.t Sj,Vk
2171	eq.h.t Sj,Vk
4170	eq.h.f Sj,Vk
4171	eq.h.f Sj,Vk
2175	eq.w.t Sj,Vk
2176	eq.w.t Sj,Vk
4175	eq.w.f Sj,Vk
4176	eq.w.f Sj,Vk
2180	eq.l.t Sj,Vk
2181	eq.l.t Sj,Vk
4180	eq.l.f Sj,Vk
4181	eq.l.f Sj,Vk
2185	eq.s.t Sj,Vk (native)
2186	eq.s.t Sj,Vk (native)
3185	eq.s.t Sj,Vk (IEEE)
3186	eq.s.t Sj,Vk (IEEE)
4185	eq.s.f Sj,Vk (native)
4186	eq.s.f Sj,Vk (native)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
5185	eq.s.f Sj,Vk (IEEE)
5186	eq.s.f Sj,Vk (IEEE)
2190	eq.d.t Sj,Vk (native)
2191	eq.d.t Sj,Vk (native)
3190	eq.d.t Sj,Vk (IEEE)
3191	eq.d.t Sj,Vk (IEEE)
4190	eq.d.f Sj,Vk (native)
4191	eq.d.f Sj,Vk (native)
5190	eq.d.f Sj,Vk (IEEE)
5191	eq.d.f Sj,Vk (IEEE)
196	shf Vi,Sj,Vk
197	shf Vi,Sj,Vk
2196	shf.t Vi,Sj,Vk
2197	shf.t Vi,Sj,Vk
4196	shf.f Vi,Sj,Vk
4197	shf.f Vi,Sj,Vk
220	tzc Vj,Vk
221	tzc Vj,Vk
2220	tzc.t Vj,Vk
2221	tzc.t Vj,Vk
4220	tzc.f Vj,Vk
4221	tzc.f Vj,Vk
225	tzc Vj,Vk
226	tzc Vj,Vk
2225	tzc.t Vj,Vk
2226	tzc.t Vj,Vk
4225	tzc.f Vj,Vk
4226	tzc.f Vj,Vk
2250	add.b.t Vi,Sj,Vk

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
2251	add.b.t Vi,Sj,Vk
4250	add.b.f Vi,Sj,Vk
4251	add.b.f Vi,Sj,Vk
2255	add.h.t Vi,Sj,Vk
2256	add.h.t Vi,Sj,Vk
4255	add.h.f Vi,Sj,Vk
4256	add.h.f Vi,Sj,Vk
2260	add.w.t Vi,Sj,Vk
2261	add.w.t Vi,Sj,Vk
4260	add.w.f Vi,Sj,Vk
4261	add.w.f Vi,Sj,Vk
2265	add.l.t Vi,Sj,Vk
2266	add.l.t Vi,Sj,Vk
4265	add.l.f Vi,Sj,Vk
4266	add.l.f Vi,Sj,Vk
2270	add.s.t Vi,Sj,Vk (native)
2271	add.s.t Vi,Sj,Vk (native)
3270	add.s.t Vi,Sj,Vk (IEEE)
3271	add.s.t Vi,Sj,Vk (IEEE)
4270	add.s.f Vi,Sj,Vk (native)
4271	add.s.f Vi,Sj,Vk (native)
5270	add.s.f Vi,Sj,Vk (IEEE)
5271	add.s.f Vi,Sj,Vk (IEEE)
2275	add.d.t Vi,Sj,Vk (native)
2276	add.d.t Vi,Sj,Vk (native)
3275	add.d.t Vi,Sj,Vk (IEEE)
3276	add.d.t Vi,Sj,Vk (IEEE)
4275	add.d.f Vi,Sj,Vk (native)
4276	add.d.f Vi,Sj,Vk (native)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
5275	add.d.f Vi,Sj,Vk (IEEE)
5276	add.d.f Vi,Sj,Vk (IEEE)
2280	sub.b.t Vi,Sj,Vk
2281	sub.b.t Vi,Sj,Vk
4280	sub.b.f Vi,Sj,Vk
4281	sub.b.f Vi,Sj,Vk
2285	sub.h.t Vi,Sj,Vk
2286	sub.h.t Vi,Sj,Vk
4285	sub.h.f Vi,Sj,Vk
4286	sub.h.f Vi,Sj,Vk
2290	sub.w.t Vi,Sj,Vk
2291	sub.w.t Vi,Sj,Vk
4290	sub.w.f Vi,Sj,Vk
4291	sub.w.f Vi,Sj,Vk
2295	sub.l.t Vi,Sj,Vk
2296	sub.l.t Vi,Sj,Vk
4295	sub.l.f Vi,Sj,Vk
4296	sub.l.f Vi,Sj,Vk
2300	sub.s.t Vi,Sj,Vk (native)
2301	sub.s.t Vi,Sj,Vk (native)
3300	sub.s.t Vi,Sj,Vk (IEEE)
3301	sub.s.t Vi,Sj,Vk (IEEE)
4300	sub.s.f Vi,Sj,Vk (native)
4301	sub.s.f Vi,Sj,Vk (native)
5300	sub.s.f Vi,Sj,Vk (IEEE)
5301	sub.s.f Vi,Sj,Vk (IEEE)
2305	sub.d.t Vi,Sj,Vk (native)
2306	sub.d.t Vi,Sj,Vk (native)
3305	sub.d.t Vi,Sj,Vk (IEEE)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
3306	sub.d.t Vi,Sj,Vk (IEEE)
4305	sub.d.f Vi,Sj,Vk (native)
4306	sub.d.f Vi,Sj,Vk (native)
5305	sub.d.f Vi,Sj,Vk (IEEE)
5306	sub.d.f Vi,Sj,Vk (IEEE)
2310	and.t Vi,Sj,Vk
2311	and.t Vi,Sj,Vk
4310	and.f Vi,Sj,Vk
4311	and.f Vi,Sj,Vk
2315	or.t Vi,Sj,Vk
2316	or.t Vi,Sj,Vk
4315	or.f Vi,Sj,Vk
4316	or.f Vi,Sj,Vk
2320	xor.t Vi,Sj,Vk
2321	xor.t Vi,Sj,Vk
4320	xor.f Vi,Sj,Vk
4321	xor.f Vi,Sj,Vk
2325	shf.t Sj,Vk
2326	shf.t Sj,Vk
4325	shf.f Sj,Vk
4326	shf.f Sj,Vk
2330	le.b.t Vj,Vk
2331	le.b.t Vj,Vk
4330	le.b.f Vj,Vk
4331	le.b.f Vj,Vk
2335	le.h.t Vj,Vk
2336	le.h.t Vj,Vk
4335	le.h.f Vj,Vk
4336	le.h.f Vj,Vk

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
2340	le.w.t Vj,Vk
2341	le.w.t Vj,Vk
4340	le.w.f Vj,Vk
4341	le.w.f Vj,Vk
2345	le.l.t Vj,Vk
2346	le.l.t Vj,Vk
4345	le.l.f Vj,Vk
4346	le.l.f Vj,Vk
2350	le.s.t Vj,Vk (native)
2351	le.s.t Vj,Vk (native)
3350	le.s.t Vj,Vk (IEEE)
3351	le.s.t Vj,Vk (IEEE)
4350	le.s.f Vj,Vk (native)
4351	le.s.f Vj,Vk (native)
5350	le.s.f Vj,Vk (IEEE)
5351	le.s.f Vj,Vk (IEEE)
2355	le.d.t Vj,Vk (native)
2356	le.d.t Vj,Vk (native)
3355	le.d.t Vj,Vk (IEEE)
3356	le.d.t Vj,Vk (IEEE)
4355	le.d.f Vj,Vk (native)
4356	le.d.f Vj,Vk (native)
5355	le.d.f Vj,Vk (IEEE)
5356	le.d.f Vj,Vk (IEEE)
2360	lt.b.t Vj,Vk
2361	lt.b.t Vj,Vk
4360	lt.b.f Vj,Vk
4361	lt.b.f Vj,Vk
2365	lt.h.t Vj,Vk

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
2366	lt.h.t Vj,Vk
4365	lt.h.f Vj,Vk
4366	lt.h.f Vj,Vk
2370	lt.w.t Vj,Vk
2371	lt.w.t Vj,Vk
4370	lt.w.f Vj,Vk
4371	lt.w.f Vj,Vk
2375	lt.l.t Vj,Vk
2376	lt.l.t Vj,Vk
4375	lt.l.f Vj,Vk
4376	lt.l.f Vj,Vk
2380	lt.s.t Vj,Vk (native)
2381	lt.s.t Vj,Vk (native)
3380	lt.s.t Vj,Vk (IEEE)
3381	lt.s.t Vj,Vk (IEEE)
4380	lt.s.f Vj,Vk (native)
4381	lt.s.f Vj,Vk (native)
5380	lt.s.f Vj,Vk (IEEE)
5381	lt.s.f Vj,Vk (IEEE)
2385	lt.d.t Vj,Vk (native)
2386	lt.d.t Vj,Vk (native)
3385	lt.d.t Vj,Vk (IEEE)
3386	lt.d.t Vj,Vk (IEEE)
4385	lt.d.f Vj,Vk (native)
4386	lt.d.f Vj,Vk (native)
5385	lt.d.f Vj,Vk (IEEE)
5386	lt.d.f Vj,Vk (IEEE)
2390	eq.b.t Vj,Vk
2391	eq.b.t Vj,Vk

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
4390	eq.b.f Vj,Vk
4391	eq.b.f Vj,Vk
2395	eq.h.t Vj,Vk
2396	eq.h.t Vj,Vk
4395	eq.h.f Vj,Vk
4396	eq.h.f Vj,Vk
2400	eq.w.t Vj,Vk
2401	eq.w.t Vj,Vk
4400	eq.w.f Vj,Vk
4401	eq.w.f Vj,Vk
2405	eq.l.t Vj,Vk
2406	eq.l.t Vj,Vk
4405	eq.l.f Vj,Vk
4406	eq.l.f Vj,Vk
2410	eq.s.t Vj,Vk (native)
2411	eq.s.t Vj,Vk (native)
3410	eq.s.t Vj,Vk (IEEE)
3411	eq.s.t Vj,Vk (IEEE)
4410	eq.s.f Vj,Vk (native)
4411	eq.s.f Vj,Vk (native)
5410	eq.s.f Vj,Vk (IEEE)
5411	eq.s.f Vj,Vk (IEEE)
2415	eq.d.t Vj,Vk (native)
2416	eq.d.t Vj,Vk (native)
3415	eq.d.t Vj,Vk (IEEE)
3416	eq.d.t Vj,Vk (IEEE)
4415	eq.d.f Vj,Vk (native)
4416	eq.d.f Vj,Vk (native)
5415	eq.d.f Vj,Vk (IEEE)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
5416	eq.d.f Vj,Vk (IEEE)
420	frint.s Vj,Vk (native)
421	frint.s Vj,Vk (native)
1420	frint.s Vj,Vk (IEEE)
1421	frint.s Vj,Vk (IEEE)
2420	frint.s.t Vj,Vk (native)
2421	frint.s.t Vj,Vk (native)
3420	frint.s.t Vj,Vk (IEEE)
3421	frint.s.t Vj,Vk (IEEE)
4420	frint.s.f Vj,Vk (native)
4421	frint.s.f Vj,Vk (native)
5420	frint.s.f Vj,Vk (IEEE)
5421	frint.s.f Vj,Vk (IEEE)
425	frint.d Vj,Vk (native)
426	frint.d Vj,Vk (native)
1425	frint.d Vj,Vk (IEEE)
1426	frint.d Vj,Vk (IEEE)
2425	frint.d.t Vj,Vk (native)
2426	frint.d.t Vj,Vk (native)
3425	frint.d.t Vj,Vk (IEEE)
3426	frint.d.t Vj,Vk (IEEE)
4425	frint.d.f Vj,Vk (native)
4426	frint.d.f Vj,Vk (native)
5425	frint.d.f Vj,Vk (IEEE)
5426	frint.d.f Vj,Vk (IEEE)
2430	add.b.t Vi,Vj,Vk
2431	add.b.t Vi,Vj,Vk
4430	add.b.f Vi,Vj,Vk
4431	add.b.f Vi,Vj,Vk

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
2435	add.h.t Vi,Vj,Vk
2436	add.h.t Vi,Vj,Vk
4435	add.h.f Vi,Vj,Vk
4436	add.h.f Vi,Vj,Vk
2440	add.w.t Vi,Vj,Vk
2441	add.w.t Vi,Vj,Vk
4440	add.w.f Vi,Vj,Vk
4441	add.w.f Vi,Vj,Vk
2445	add.l.t Vi,Vj,Vk
2446	add.l.t Vi,Vj,Vk
4445	add.l.f Vi,Vj,Vk
4446	add.l.f Vi,Vj,Vk
2450	add.s.t Vi,Vj,Vk (native)
2451	add.s.t Vi,Vj,Vk (native)
3450	add.s.t Vi,Vj,Vk (IEEE)
3451	add.s.t Vi,Vj,Vk (IEEE)
4450	add.s.f Vi,Vj,Vk (native)
4451	add.s.f Vi,Vj,Vk (native)
5450	add.s.f Vi,Vj,Vk (IEEE)
5451	add.s.f Vi,Vj,Vk (IEEE)
2455	add.d.t Vi,Vj,Vk (native)
2456	add.d.t Vi,Vj,Vk (native)
3455	add.d.t Vi,Vj,Vk (IEEE)
3456	add.d.t Vi,Vj,Vk (IEEE)
4455	add.d.f Vi,Vj,Vk (native)
4456	add.d.f Vi,Vj,Vk (native)
5455	add.d.f Vi,Vj,Vk (IEEE)
5456	add.d.f Vi,Vj,Vk (IEEE)
2460	sub.b.t Vi,Vj,Vk

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
2461	sub.b.t Vi,Vj,Vk
4460	sub.b.f Vi,Vj,Vk
4461	sub.b.f Vi,Vj,Vk
2465	sub.h.t Vi,Vj,Vk
2466	sub.h.t Vi,Vj,Vk
4465	sub.h.f Vi,Vj,Vk
4466	sub.h.f Vi,Vj,Vk
2470	sub.w.t Vi,Vj,Vk
2471	sub.w.t Vi,Vj,Vk
4470	sub.w.f Vi,Vj,Vk
4471	sub.w.f Vi,Vj,Vk
2475	sub.l.t Vi,Vj,Vk
2476	sub.l.t Vi,Vj,Vk
4475	sub.l.f Vi,Vj,Vk
4476	sub.l.f Vi,Vj,Vk
2480	sub.s.t Vi,Vj,Vk (native)
2481	sub.s.t Vi,Vj,Vk (native)
3480	sub.s.t Vi,Vj,Vk (IEEE)
3481	sub.s.t Vi,Vj,Vk (IEEE)
4480	sub.s.f Vi,Vj,Vk (native)
4481	sub.s.f Vi,Vj,Vk (native)
5480	sub.s.f Vi,Vj,Vk (IEEE)
5481	sub.s.f Vi,Vj,Vk (IEEE)
2485	sub.d.t Vi,Vj,Vk (native)
2486	sub.d.t Vi,Vj,Vk (native)
3485	sub.d.t Vi,Vj,Vk (IEEE)
3486	sub.d.t Vi,Vj,Vk (IEEE)
4485	sub.d.f Vi,Vj,Vk (native)
4486	sub.d.f Vi,Vj,Vk (native)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
5485	sub.d.f Vi,Vj,Vk (IEEE)
5486	sub.d.f Vi,Vj,Vk (IEEE)
2490	and.t Vi,Vj,Vk
2491	and.t Vi,Vj,Vk
4490	and.f Vi,Vj,Vk
4491	and.f Vi,Vj,Vk
2495	or.t Vi,Vj,Vk
2496	or.t Vi,Vj,Vk
4495	or.f Vi,Vj,Vk
4496	or.f Vi,Vj,Vk
2500	xor.t Vi,Vj,Vk
2501	xor.t Vi,Vj,Vk
4500	xor.f Vi,Vj,Vk
4501	xor.f Vi,Vj,Vk
2505	not.t Vj,Vk
2506	not.t Vj,Vk
4505	not.f Vj,Vk
4506	not.f Vj,Vk
2510	neg.b.t Vj,Vk
2511	neg.b.t Vj,Vk
4510	neg.b.f Vj,Vk
4511	neg.b.f Vj,Vk
2515	neg.h.t Vj,Vk
2516	neg.h.t Vj,Vk
4515	neg.h.f Vj,Vk
4516	neg.h.f Vj,Vk
2520	neg.w.t Vj,Vk
2521	neg.w.t Vj,Vk
4520	neg.w.f Vj,Vk

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
4521	neg.w.f Vj,Vk
2525	neg.l.t Vj,Vk
2526	neg.l.t Vj,Vk
4525	neg.l.f Vj,Vk
4526	neg.l.f Vj,Vk
2530	neg.s.t Vj,Vk (native)
2531	neg.s.t Vj,Vk (native)
3530	neg.s.t Vj,Vk (IEEE)
3531	neg.s.t Vj,Vk (IEEE)
4530	neg.s.f Vj,Vk (native)
4531	neg.s.f Vj,Vk (native)
5530	neg.s.f Vj,Vk (IEEE)
5531	neg.s.f Vj,Vk (IEEE)
2535	neg.d.t Vj,Vk (native)
2536	neg.d.t Vj,Vk (native)
3535	neg.d.t Vj,Vk (IEEE)
3536	neg.d.t Vj,Vk (IEEE)
4535	neg.d.f Vj,Vk (native)
4536	neg.d.f Vj,Vk (native)
5535	neg.d.f Vj,Vk (IEEE)
5536	neg.d.f Vj,Vk (IEEE)
540	sub.s Si,Vj,Vk (native)
541	sub.s Si,Vj,Vk (native)
1540	sub.s Si,Vj,Vk (IEEE)
1541	sub.s Si,Vj,Vk (IEEE)
2540	sub.s.t Si,Vj,Vk (native)
2541	sub.s.t Si,Vj,Vk (native)
3540	sub.s.t Si,Vj,Vk (IEEE)
3541	sub.s.t Si,Vj,Vk (IEEE)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
4540	sub.s.f Si,Vj,Vk (native)
4541	sub.s.f Si,Vj,Vk (native)
5540	sub.s.f Si,Vj,Vk (IEEE)
5541	sub.s.f Si,Vj,Vk (IEEE)
545	sub.d Si,Vj,Vk (native)
546	sub.d Si,Vj,Vk (native)
1545	sub.d Si,Vj,Vk (IEEE)
1546	sub.d Si,Vj,Vk (IEEE)
2545	sub.d.t Si,Vj,Vk (native)
2546	sub.d.t Si,Vj,Vk (native)
3545	sub.d.t Si,Vj,Vk (IEEE)
3546	sub.d.t Si,Vj,Vk (IEEE)
4545	sub.d.f Si,Vj,Vk (native)
4546	sub.d.f Si,Vj,Vk (native)
5545	sub.d.f Si,Vj,Vk (IEEE)
5546	sub.d.f Si,Vj,Vk (IEEE)
795	shf Vi,Vj,Vk
796	shf Vi,Vj,Vk
2795	shf.t Vi,Vj,Vk
2796	shf.t Vi,Vj,Vk
4795	shf.f Vi,Vj,Vk
4796	shf.f Vi,Vj,Vk
2805	plc.t.t Vj,Vk
2806	plc.t.t Vj,Vk
4805	plc.t.f Vj,Vk
4806	plc.t.f Vj,Vk
2815	xpnd.t.t Vj,Vk
2816	xpnd.t.t Vj,Vk
4815	xpnd.t.f Vj,Vk

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
4816	xpnd.t.f Vj,Vk
2820	sum.b.t Vk
2821	sum.b.t Vk
4820	sum.b.f Vk
4821	sum.b.f Vk
2825	sum.h.t Vk
2826	sum.h.t Vk
4825	sum.h.f Vk
4826	sum.h.f Vk
2830	sum.w.t Vk
2831	sum.w.t Vk
4830	sum.w.f Vk
4831	sum.w.f Vk
2835	sum.l.t Vk
2836	sum.l.t Vk
4835	sum.l.f Vk
4836	sum.l.f Vk
2840	sum.s.t Vk (native)
2841	sum.s.t Vk (native)
3840	sum.s.t Vk (IEEE)
3841	sum.s.t Vk (IEEE)
4840	sum.s.f Vk (native)
4841	sum.s.f Vk (native)
5840	sum.s.f Vk (IEEE)
5841	sum.s.f Vk (IEEE)
2845	sum.d.t Vk (native)
2846	sum.d.t Vk (native)
3845	sum.d.t Vk (IEEE)
3846	sum.d.t Vk (IEEE)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
4845	sum.d.f Vk (native)
4846	sum.d.f Vk (native)
5845	sum.d.f Vk (IEEE)
5846	sum.d.f Vk (IEEE)
2880	max.b.t Vk
2881	max.b.t Vk
4880	max.b.f Vk
4881	max.b.f Vk
2885	max.h.t Vk
2886	max.h.t Vk
4885	max.h.f Vk
4886	max.h.f Vk
2890	max.w.t Vk
2891	max.w.t Vk
4890	max.w.f Vk
4891	max.w.f Vk
2895	max.l.t Vk
2896	max.l.t Vk
4895	max.l.f Vk
4896	max.l.f Vk
2900	max.s.t Vk (native)
2901	max.s.t Vk (native)
3900	max.s.t Vk (IEEE)
3901	max.s.t Vk (IEEE)
4900	max.s.f Vk (native)
4901	max.s.f Vk (native)
5900	max.s.f Vk (IEEE)
5901	max.s.f Vk (IEEE)
2905	max.d.t Vk (native)

Table 8-24 (continued)
`cpu4241` class 2 subtests

Subtest	Test performed
2906	`max.d.t Vk` (native)
3905	`max.d.t Vk` (IEEE)
3906	`max.d.t Vk` (IEEE)
4905	`max.d.f Vk` (native)
4906	`max.d.f Vk` (native)
5905	`max.d.f Vk` (IEEE)
5906	`max.d.f Vk` (IEEE)
2910	`min.b.t Vk`
2911	`min.b.t Vk`
4910	`min.b.f Vk`
4911	`min.b.f Vk`
2915	`min.h.t Vk`
2916	`min.h.t Vk`
4915	`min.h.f Vk`
4916	`min.h.f Vk`
2920	`min.w.t Vk`
2921	`min.w.t Vk`
4920	`min.w.f Vk`
4921	`min.w.f Vk`
2925	`min.l.t Vk`
2926	`min.l.t Vk`
4925	`min.l.f Vk`
4926	`min.l.f Vk`
2930	`min.s.t Vk` (native)
2931	`min.s.t Vk` (native)
3930	`min.s.t Vk` (IEEE)
3931	`min.s.t Vk` (IEEE)
4930	`min.s.f Vk` (native)
4931	`min.s.f Vk` (native)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
5930	min.s.f Vk (IEEE)
5931	min.s.f Vk (IEEE)
2935	min.d.t Vk (native)
2936	min.d.t Vk (native)
3935	min.d.t Vk (IEEE)
3936	min.d.t Vk (IEEE)
4935	min.d.f Vk (native)
4936	min.d.f Vk (native)
5935	min.d.f Vk (IEEE)
5936	min.d.f Vk (IEEE)
2940	all.t Vk
2941	all.t Vk
4940	all.f Vk
4941	all.f Vk
2945	any.t Vk
2946	any.t Vk
4945	any.f Vk
4946	any.f Vk
2950	parity.t Vk
2951	parity.t Vk
4950	parity.f Vk
4951	parity.f Vk
955	cvtb.w Vj,Vk
956	cvtb.w Vj,Vk
2955	cvtb.w.t Vj,Vk
2956	cvtb.w.t Vj,Vk
4955	cvtb.w.f Vj,Vk
4956	cvtb.w.f Vj,Vk
957	cvth.w Vj,Vk

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
958	cvth.w Vj,Vk
2957	cvth.w.t Vj,Vk
2958	cvth.w.t Vj,Vk
4957	cvth.w.f Vj,Vk
4958	cvth.w.f Vj,Vk
960	cvtw.b Vj,Vk
961	cvtw.b Vj,Vk
2960	cvtw.b.t Vj,Vk
2961	cvtw.b.t Vj,Vk
4960	cvtw.b.f Vj,Vk
4961	cvtw.b.f Vj,Vk
962	cvtw.h Vj,Vk
963	cvtw.h Vj,Vk
2962	cvtw.h.t Vj,Vk
2963	cvtw.h.t Vj,Vk
4962	cvtw.h.f Vj,Vk
4963	cvtw.h.f Vj,Vk v 965
966	cvtw.l Vj,Vk
2965	cvtw.l.t Vj,Vk
2966	cvtw.l.t Vj,Vk
4965	cvtw.l.f Vj,Vk
4966	cvtw.l.f Vj,Vk
967	cvtw.s Vj,Vk (native)
968	cvtw.s Vj,Vk (native)
1967	cvtw.s Vj,Vk (IEEE)
1968	cvtw.s Vj,Vk (IEEE)
2967	cvtw.s.t Vj,Vk (native)
2968	cvtw.s.t Vj,Vk (native)
3967	cvtw.s.t Vj,Vk (IEEE)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
3968	cvtw.s.t Vj,Vk (IEEE)
4967	cvtw.s.f Vj,Vk (native)
4968	cvtw.s.f Vj,Vk (native)
5967	cvtw.s.f Vj,Vk (IEEE)
5968	cvtw.s.f Vj,Vk (IEEE)
970	cvtw.d Vj,Vk (native)
971	cvtw.d Vj,Vk (native)
1970	cvtw.d Vj,Vk (IEEE)
1971	cvtw.d Vj,Vk (IEEE)
2970	cvtw.d.t Vj,Vk (native)
2971	cvtw.d.t Vj,Vk (native)
3970	cvtw.d.t Vj,Vk (IEEE)
3971	cvtw.d.t Vj,Vk (IEEE)
4970	cvtw.d.f Vj,Vk (native)
4971	cvtw.d.f Vj,Vk (native)
5970	cvtw.d.f Vj,Vk (IEEE)
5971	cvtw.d.f Vj,Vk (IEEE)
975	cvtl.w Vj,Vk
976	cvtl.w Vj,Vk
2975	cvtl.w.t Vj,Vk
2976	cvtl.w.t Vj,Vk
4975	cvtl.w.f Vj,Vk
4976	cvtl.w.f Vj,Vk
977	cvtl.s Vj,Vk (native)
978	cvtl.s Vj,Vk (native)
1977	cvtl.s Vj,Vk (IEEE)
1978	cvtl.s Vj,Vk (IEEE)
2977	cvtl.s.t Vj,Vk (native)
2978	cvtl.s.t Vj,Vk (native)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
3977	cvtl.s.t Vj,Vk (IEEE)
3978	cvtl.s.t Vj,Vk (IEEE)
4977	cvtl.s.f Vj,Vk (native)
4978	cvtl.s.f Vj,Vk (native)
5977	cvtl.s.f Vj,Vk (IEEE)
5978	cvtl.s.f Vj,Vk (IEEE)
980	cvtl.d Vj,Vk (native)
981	cvtl.d Vj,Vk (native)
1980	cvtl.d Vj,Vk (IEEE)
1981	cvtl.d Vj,Vk (IEEE)
2980	cvtl.d.t Vj,Vk (native)
2981	cvtl.d.t Vj,Vk (native)
3980	cvtl.d.t Vj,Vk (IEEE)
3981	cvtl.d.t Vj,Vk (IEEE)
4980	cvtl.d.f Vj,Vk (native)
4981	cvtl.d.f Vj,Vk (native)
5980	cvtl.d.f Vj,Vk (IEEE)
5981	cvtl.d.f Vj,Vk (IEEE)
985	cvts.w Vj,Vk (native)
986	cvts.w Vj,Vk (native)
1985	cvts.w Vj,Vk (IEEE)
1986	cvts.w Vj,Vk (IEEE)
2985	cvts.w.t Vj,Vk (native)
2986	cvts.w.t Vj,Vk (native)
3985	cvts.w.t Vj,Vk (IEEE)
3986	cvts.w.t Vj,Vk (IEEE)
4985	cvts.w.f Vj,Vk (native)
4986	cvts.w.f Vj,Vk (native)
5985	cvts.w.f Vj,Vk (IEEE)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
5986	cvts.w.f Vj,Vk (IEEE)
987	cvts.l Vj,Vk (native)
988	cvts.l Vj,Vk (native)
1987	cvts.l Vj,Vk (IEEE)
1988	cvts.l Vj,Vk (IEEE)
2987	cvts.l.t Vj,Vk (native)
2988	cvts.l.t Vj,Vk (native)
3987	cvts.l.t Vj,Vk (IEEE)
3988	cvts.l.t Vj,Vk (IEEE)
4987	cvts.l.f Vj,Vk (native)
4988	cvts.l.f Vj,Vk (native)
5987	cvts.l.f Vj,Vk (IEEE)
5988	cvts.l.f Vj,Vk (IEEE)
990	cvts.d Vj,Vk (native)
991	cvts.d Vj,Vk (native)
1990	cvts.d Vj,Vk (IEEE)
1991	cvts.d Vj,Vk (IEEE)
2990	cvts.d.t Vj,Vk (native)
2991	cvts.d.t Vj,Vk (native)
3990	cvts.d.t Vj,Vk (IEEE)
3991	cvts.d.t Vj,Vk (IEEE)
4990	cvts.d.f Vj,Vk (native)
4991	cvts.d.f Vj,Vk (native)
5990	cvts.d.f Vj,Vk (IEEE)
5991	cvts.d.f Vj,Vk (IEEE)
992	cvtd.w Vj,Vk (native)
993	cvtd.w Vj,Vk (native)
1992	cvtd.w Vj,Vk (IEEE)
1993	cvtd.w Vj,Vk (IEEE)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
2992	cvtd.w.t Vj,Vk (native)
2993	cvtd.w.t Vj,Vk (native)
3992	cvtd.w.t Vj,Vk (IEEE)
3993	cvtd.w.t Vj,Vk (IEEE)
4992	cvtd.w.f Vj,Vk (native)
4993	cvtd.w.f Vj,Vk (native)
5992	cvtd.w.f Vj,Vk (IEEE)
5993	cvtd.w.f Vj,Vk (IEEE)
995	cvtd.l Vj,Vk (native)
996	cvtd.l Vj,Vk (native)
1995	cvtd.l Vj,Vk (IEEE)
1996	cvtd.l Vj,Vk (IEEE)
2995	cvtd.l.t Vj,Vk (native)
2996	cvtd.l.t Vj,Vk (native)
3995	cvtd.l.t Vj,Vk (IEEE)
3996	cvtd.l.t Vj,Vk (IEEE)
4995	cvtd.l.f Vj,Vk (native)
4996	cvtd.l.f Vj,Vk (native)
5995	cvtd.l.f Vj,Vk (IEEE)
5996	cvtd.l.f Vj,Vk (IEEE)
997	cvtd.s Vj,Vk (native)
998	cvtd.s Vj,Vk (native)
1997	cvtd.s Vj,Vk (IEEE)
1998	cvtd.s Vj,Vk (IEEE)
2997	cvtd.s.t Vj,Vk (native)
2998	cvtd.s.t Vj,Vk (native)
3997	cvtd.s.t Vj,Vk (IEEE)
3998	cvtd.s.t Vj,Vk (IEEE)
4997	cvtd.s.f Vj,Vk (native)

Table 8-24 (continued)
cpu4241 class 2 subtests

Subtest	Test performed
4998	cvtd.s.f Vj,Vk (native)
5997	cvtd.s.f Vj,Vk (IEEE)
5998	cvtd.s.f Vj,Vk (IEEE)

Table 8-25 describes the cpu4241 class 3 subtests.

Table 8-25
cpu4241 class 3 subtests

Subtest	Test performed
2630	mul.b.t Vi,Sj,Vk
2631	mul.b.t Vi,Sj,Vk
4630	mul.b.f Vi,Sj,Vk
4631	mul.b.f Vi,Sj,Vk
2635	mul.h.t Vi,Sj,Vk
2636	mul.h.t Vi,Sj,Vk
4635	mul.h.f Vi,Sj,Vk
4636	mul.h.f Vi,Sj,Vk
2640	mul.w.t Vi,Sj,Vk
2641	mul.w.t Vi,Sj,Vk
4640	mul.w.f Vi,Sj,Vk
4641	mul.w.f Vi,Sj,Vk
2645	mul.l.t Vi,Sj,Vk
2646	mul.l.t Vi,Sj,Vk
4645	mul.l.f Vi,Sj,Vk
4646	mul.l.f Vi,Sj,Vk
2650	mul.s.t Vi,Sj,Vk (native)
2651	mul.s.t Vi,Sj,Vk (native)
3650	mul.s.t Vi,Sj,Vk (IEEE)

Table 8-25 (continued)
cpu4241 class 3 subtests

Subtest	Test performed
3651	mul.s.t Vi,Sj,Vk (IEEE)
4650	mul.s.f Vi,Sj,Vk (native)
4651	mul.s.f Vi,Sj,Vk (native)
5650	mul.s.f Vi,Sj,Vk (IEEE)
5651	mul.s.f Vi,Sj,Vk (IEEE)
2655	mul.d.t Vi,Sj,Vk (native)
2656	mul.d.t Vi,Sj,Vk (native)
3655	mul.d.t Vi,Sj,Vk (IEEE)
3656	mul.d.t Vi,Sj,Vk (IEEE)
4655	mul.d.f Vi,Sj,Vk (native)
4656	mul.d.f Vi,Sj,Vk (native)
5655	mul.d.f Vi,Sj,Vk (IEEE)
5656	mul.d.f Vi,Sj,Vk (IEEE)
2660	div.b.t Vi,Sj,Vk
2661	div.b.t Vi,Sj,Vk
4660	div.b.f Vi,Sj,Vk
4661	div.b.f Vi,Sj,Vk
2665	div.h.t Vi,Sj,Vk
2666	div.h.t Vi,Sj,Vk
4665	div.h.f Vi,Sj,Vk
4666	div.h.f Vi,Sj,Vk
2670	div.w.t Vi,Sj,Vk
2671	div.w.t Vi,Sj,Vk
4670	div.w.f Vi,Sj,Vk
4671	div.w.f Vi,Sj,Vk
2675	div.l.t Vi,Sj,Vk
2676	div.l.t Vi,Sj,Vk
4675	div.l.f Vi,Sj,Vk

Table 8-25 (continued)
cpu4241 class 3 subtests

Subtest	Test performed
4676	div.l.f Vi,Sj,Vk
2680	div.s.t Vi,Sj,Vk (native)
2681	div.s.t Vi,Sj,Vk (native)
3680	div.s.t Vi,Sj,Vk (IEEE)
3681	div.s.t Vi,Sj,Vk (IEEE)
4680	div.s.f Vi,Sj,Vk (native)
4681	div.s.f Vi,Sj,Vk (native)
5680	div.s.f Vi,Sj,Vk (IEEE)
5681	div.s.f Vi,Sj,Vk (IEEE)
2685	div.d.t Vi,Sj,Vk (native)
2686	div.d.t Vi,Sj,Vk (native)
3685	div.d.t Vi,Sj,Vk (IEEE)
3686	div.d.t Vi,Sj,Vk (IEEE)
4685	div.d.f Vi,Sj,Vk (native)
4686	div.d.f Vi,Sj,Vk (native)
5685	div.d.f Vi,Sj,Vk (IEEE)
5686	div.d.f Vi,Sj,Vk (IEEE)
690	div.s Si,Vj,Vk (native)
691	div.s Si,Vj,Vk (native)
1690	div.s Si,Vj,Vk (IEEE)
1691	div.s Si,Vj,Vk (IEEE)
2690	div.s.t Si,Vj,Vk (native)
2691	div.s.t Si,Vj,Vk (native)
3690	div.s.t Si,Vj,Vk (IEEE)
3691	div.s.t Si,Vj,Vk (IEEE)
4690	div.s.f Si,Vj,Vk (native)
4691	div.s.f Si,Vj,Vk (native)
5690	div.s.f Si,Vj,Vk (IEEE)

Table 8-25 (continued)
cpu4241 class 3 subtests

Subtest	Test performed
5691	div.s.f Si,Vj,Vk (IEEE)
695	div.d Si,Vj,Vk (native)
696	div.d Si,Vj,Vk (native)
1695	div.d Si,Vj,Vk (IEEE)
1696	div.d Si,Vj,Vk (IEEE)
2695	div.d.t Si,Vj,Vk (native)
2696	div.d.t Si,Vj,Vk (native)
3695	div.d.t Si,Vj,Vk (IEEE)
3696	div.d.t Si,Vj,Vk (IEEE)
4695	div.d.f Si,Vj,Vk (native)
4696	div.d.f Si,Vj,Vk (native)
5695	div.d.f Si,Vj,Vk (IEEE)
5696	div.d.f Si,Vj,Vk (IEEE)
2700	mul.b.t Vi,Vj,Vk
2701	mul.b.t Vi,Vj,Vk
4700	mul.b.f Vi,Vj,Vk
4701	mul.b.f Vi,Vj,Vk
2705	mul.h.t Vi,Vj,Vk
2706	mul.h.t Vi,Vj,Vk
4705	mul.h.f Vi,Vj,Vk
4706	mul.h.f Vi,Vj,Vk
2710	mul.w.t Vi,Vj,Vk
2711	mul.w.t Vi,Vj,Vk
4710	mul.w.f Vi,Vj,Vk
4711	mul.w.f Vi,Vj,Vk
2715	mul.l.t Vi,Vj,Vk
2716	mul.l.t Vi,Vj,Vk
4715	mul.l.f Vi,Vj,Vk

Table 8-25 (continued)
cpu4241 class 3 subtests

Subtest	Test performed
4716	mul.l.f Vi,Vj,Vk
2720	mul.s.t Vi,Vj,Vk (native)
2721	mul.s.t Vi,Vj,Vk (native)
3720	mul.s.t Vi,Vj,Vk (IEEE)
3721	mul.s.t Vi,Vj,Vk (IEEE)
4720	mul.s.f Vi,Vj,Vk (native)
4721	mul.s.f Vi,Vj,Vk (native)
5720	mul.s.f Vi,Vj,Vk (IEEE)
5721	mul.s.f Vi,Vj,Vk (IEEE)
2725	mul.d.t Vi,Vj,Vk (native)
2726	mul.d.t Vi,Vj,Vk (native)
3725	mul.d.t Vi,Vj,Vk (IEEE)
3726	mul.d.t Vi,Vj,Vk (IEEE)
4725	mul.d.f Vi,Vj,Vk (native)
4726	mul.d.f Vi,Vj,Vk (native)
5725	mul.d.f Vi,Vj,Vk (IEEE)
5726	mul.d.f Vi,Vj,Vk (IEEE)
730	sqrt.s Vj,Vk (native)
731	sqrt.s Vj,Vk (native)
1730	sqrt.s Vj,Vk (IEEE)
1731	sqrt.s Vj,Vk (IEEE)
2730	sqrt.s.t Vj,Vk (native)
2731	sqrt.s.t Vj,Vk (native)
3730	sqrt.s.t Vj,Vk (IEEE)
3731	sqrt.s.t Vj,Vk (IEEE)
4730	sqrt.s.f Vj,Vk (native)
4731	sqrt.s.f Vj,Vk (native)
5730	sqrt.s.f Vj,Vk (IEEE)

Table 8-25 (continued)
cpu4241 class 3 subtests

Subtest	Test performed
5731	sqrt.s.f Vj,Vk (IEEE)
735	sqrt.d Vj,Vk (native)
736	sqrt.d Vj,Vk (native)
1735	sqrt.d Vj,Vk (IEEE)
1736	sqrt.d Vj,Vk (IEEE)
2735	sqrt.d.t Vj,Vk (native)
2736	sqrt.d.t Vj,Vk (native)
3735	sqrt.d.t Vj,Vk (IEEE)
3736	sqrt.d.t Vj,Vk (IEEE)
4735	sqrt.d.f Vj,Vk (native)
4736	sqrt.d.f Vj,Vk (native)
5735	sqrt.d.f Vj,Vk (IEEE)
5736	sqrt.d.f Vj,Vk (IEEE)
2740	div.b.t Vi,Vj,Vk
2741	div.b.t Vi,Vj,Vk
4740	div.b.f Vi,Vj,Vk
4741	div.b.f Vi,Vj,Vk
2745	div.h.t Vi,Vj,Vk
2746	div.h.t Vi,Vj,Vk
4745	div.h.f Vi,Vj,Vk
4746	div.h.f Vi,Vj,Vk
2750	div.w.t Vi,Vj,Vk
2751	div.w.t Vi,Vj,Vk
4750	div.w.f Vi,Vj,Vk
4751	div.w.f Vi,Vj,Vk
2755	div.l.t Vi,Vj,Vk
2756	div.l.t Vi,Vj,Vk
4755	div.l.f Vi,Vj,Vk

Table 8-25 (continued)
cpu4241 class 3 subtests

Subtest	Test performed
4756	div.l.f Vi,Vj,Vk
2760	div.s.t Vi,Vj,Vk (native)
2761	div.s.t Vi,Vj,Vk (native)
3760	div.s.t Vi,Vj,Vk (IEEE)
3761	div.s.t Vi,Vj,Vk (IEEE)
4760	div.s.f Vi,Vj,Vk (native)
4761	div.s.f Vi,Vj,Vk (native)
5760	div.s.f Vi,Vj,Vk (IEEE)
5761	div.s.f Vi,Vj,Vk (IEEE)
2765	div.d.t Vi,Vj,Vk (native)
2766	div.d.t Vi,Vj,Vk (native)
3765	div.d.t Vi,Vj,Vk (IEEE)
3766	div.d.t Vi,Vj,Vk (IEEE)
4765	div.d.f Vi,Vj,Vk (native)
4766	div.d.f Vi,Vj,Vk (native)
5765	div.d.f Vi,Vj,Vk (IEEE)
5766	div.d.f Vi,Vj,Vk (IEEE)
2850	prod.b.t Vk
2851	prod.b.t Vk
4850	prod.b.f Vk
4851	prod.b.f Vk
2855	prod.h.t Vk
2856	prod.h.t Vk
4855	prod.h.f Vk
4856	prod.h.f Vk
2860	prod.w.t Vk
2861	prod.w.t Vk
4860	prod.w.f Vk

Table 8-25 (continued)
`cpu4241` class 3 subtests

Subtest	Test performed
4861	`prod.w.f Vk`
2865	`prod.l.t Vk`
2866	`prod.l.t Vk`
4865	`prod.l.f Vk`
4866	`prod.l.f Vk`
2870	`prod.s.t Vk` (native)
2871	`prod.s.t Vk` (native)
3870	`prod.s.t Vk` (IEEE)
3871	`prod.s.t Vk` (IEEE)
4870	`prod.s.f Vk` (native)
4871	`prod.s.f Vk` (native)
5870	`prod.s.f Vk` (IEEE)
5871	`prod.s.f Vk` (IEEE)
2875	`prod.d.t Vk` (native)
2876	`prod.d.t Vk` (native)
3875	`prod.d.t Vk` (IEEE)
3876	`prod.d.t Vk` (IEEE)
4875	`prod.d.f Vk` (native)
4876	`prod.d.f Vk` (native)
5875	`prod.d.f Vk` (IEEE)
5876	`prod.d.f Vk` (IEEE)

Table 8-26 describes the cpu4241 class 4 subtests.

Table 8-26
cpu4241 class 4 subtests

Subtest	Test performed
2200	ld.b.t <effa>,Vk
2201	ld.b.t <effa>,Vk
4200	ld.b.f <effa>,Vk
4201	ld.b.f <effa>,Vk
2205	ld.h.t <effa>,Vk
2206	ld.h.t <effa>,Vk
4205	ld.h.f <effa>,Vk
4206	ld.h.f<effa>,Vk
2210	ld.w.t <effa>,Vk
2211	ld.w.t <effa>,Vk
4210	ld.w.f <effa>,Vk
4211	ld.w.f<effa>,Vk
2215	ld.l.t <effa>,Vk
2216	ld.l.t <effa>,Vk
4215	ld.l.f <effa>,Vk
4216	ld.l.f <effa>,Vk
2230	st.b.t Vk,<effa>
2231	st.b.t Vk,<effa>
4230	st.b.f Vk,<effa>
4231	st.b.f Vk,<effa>
2235	st.h.t Vk,<effa>
2236	st.h.t Vk,<effa>
4235	st.h.f Vk,<effa>
4236	st.h.f Vk,<effa>
2240	st.w.t Vk,<effa>
2241	st.w.t Vk,<effa>

Table 8-26 (continued)
cpu4241 class 4 subtests

Subtest	Test performed
4240	st.w.f Vk,<effa>
4241	st.w.f Vk,<effa>
2245	st.l.t Vk,<effa>
2246	st.l.t Vk,<effa>
4245	st.l.f Vk,<effa>
4246	st.l.f Vk,<effa>
2550	ldvi.b.t Vj,Vk
2551	ldvi.b.t Vj,Vk
4550	ldvi.b.f Vj,Vk
4551	ldvi.b.f Vj,Vk
2555	ldvi.h.t Vj,Vk
2556	ldvi.h.t Vj,Vk
4555	ldvi.h.f Vj,Vk
4556	ldvi.h.f Vj,Vk
2560	ldvi.w.t Vj,Vk
2561	ldvi.w.t Vj,Vk
4560	ldvi.w.f Vj,Vk
4561	ldvi.w.f Vj,Vk
2565	ldvi.l.t Vj,Vk
2566	ldvi.l.t Vj,Vk
4565	ldvi.l.f Vj,Vk
4566	ldvi.l.f Vj,Vk
2570	stvi.b.t Vi,Vj
2571	stvi.b.t Vi,Vj
4570	stvi.b.f Vi,Vj
4571	stvi.b.f Vi,Vj
2575	stvi.h.t Vi,Vj
2576	stvi.h.t Vi,Vj

Table 8-26 (continued)
cpu4241 class 4 subtests

Subtest	Test performed
4575	stvi.h.f Vi,Vj
4576	stvi.h.f Vi,Vj
2580	stvi.w.t Vi,Vj
2581	stvi.w.t Vi,Vj
4580	stvi.w.f Vi,Vj
4581	stvi.w.f Vi,Vj
2585	stvi.l.t Vi,Vj
2586	stvi.l.t Vi,Vj
4585	stvi.l.f Vi,Vj
4586	stvi.l.f Vi,Vj
2590	stvi.b.t Si,Vj
2591	stvi.b.t Si,Vj
4590	stvi.b.f Si,Vj
4591	stvi.b.f Si,Vj
2595	stvi.h.t Si,Vj
2596	stvi.h.t Si,Vj
4595	stvi.h.f Si,Vj
4596	stvi.h.f Si,Vj
2600	stvi.w.t Si,Vj
2601	stvi.w.t Si,Vj
4600	stvi.w.f Si,Vj
4601	stvi.w.f Si,Vj
2605	stvi.l.t Si,Vj
2606	stvi.l.t Si,Vj
4605	stvi.l.f Si,Vj
4606	stvi.l.f Si,Vj
2610	ste.b.t Sk,<effa>
2611	ste.b.t Sk,<effa>

Table 8-26 (continued)
cpu4241 class 4 subtests

Subtest	Test performed
4610	ste.b.f Sk,<effa>
4611	ste.b.f Sk,<effa>
2615	ste.h.t Sk,<effa>
2616	ste.h.t Sk,<effa>
4615	ste.h.f Sk,<effa>
4616	ste.h.f Sk,<effa>
2620	ste.w.t Vk,<effa>
2621	ste.w.t Vk,<effa>
4620	ste.w.f Vk,<effa>
4621	ste.w.f Vk,<effa>
2625	ste.l.t Vk,<effa>
2626	ste.l.t Vk,<effa>
4625	ste.l.f Vk,<effa>
4626	ste.l.f Vk,<effa>

8.9 Multiprocessor diagnostics (cpu4333)

The cpu4333 test verifies the operation of the CPU complex in a multiheaded environment. Included are tests for concurrent access and use of communication registers, memory, thread creation and termination instructions, interrupts, CPU execution timers, privileged instructions, and exceptions. Correct execution for processors executing in both the same and in different communication index registers (CIRs) is also verified.

8.9.1 Prerequisites and required equipment

Multiple vector processor (VP) and scalar processor (SP) boards must be present with only the scalar processor being tested, when running the cpu4333 test. Table 8-27 lists the boards that must be installed and previously verified to run this test. No additional equipment is required to run this test.

Table 8-27
Hardware required for cpu4333

Board	Tests to verify
SPU workstation interface serial (SWIS)	spu4000
SPU workstation interface parallel (SWIP)	spu4000
CPU utilities (CU)	cu4000, sst, spu4000
Interface adapter (IA)	sst, spu4000
Crossbar subsystem (XRTE, XS1E, XS0E, XLL, XS00, XS10, XRT0)	sst, spu4000, mem4000
Memory subsystem (MBs)	sst, spu4000, mem4000
Scalar processors	sst, spu4000, cpu4030
Vector processors	sst, spu4000, cpu4041
Scalar and vector processors	cpu4241, cpu4331, cpu4332

8.9.2 Invoking the test

Perform the following steps to invoke the `cpu4333` test:

Step 1 From the `xdiag` window, select `Test`.

Step 2 Click on `cpu4333` in the drop-down menu. The `cti` loads the cpu4333.info and cpu4333.par files.

Step 3 Click on `CPUcti` to invoke the `CPUcti Test Options` window.

Step 4 Select the test options. See Section 8.2 of this document.

Step 5 Select the `run` button in the command bar of the `xdiag` window to start the test.

The default test parameters are initialized from the cpu4333.par file.

8.9.3 Class 1 through 4 descriptions

Test `cpu4333` classes 1, 2, 3, and 4 exercise the following machine instructions. The test sequence exercises each instruction through all 4 classes of tests sequentially, then exercises the next instruction through all 4 classes, and so forth.

The machine instructions have the following definitions:

- `lck`—Lock a communication register.
- `ulk`—Unlock a communication register.
- `tst`—Read the value of a communication register lock bit.
- `get.w`—Copy the contents of a communication register into an address register.
- `get.l`—Copy the contents of a communication register into a scalar register.
- `put.w`—Copy the contents of an address register into a communication register.
- `put.l`—Copy the contents of a scalar register into a communication register.
- `rcv.w`—Copy the contents of a communication register into an address register if the communication register lock bit is set. Clear the lock bit.
- `rcv.l`—Copy the contents of a communication register into a scalar register if the communication register lock bit is set. Clear the lock bit.
- `snd.w`—Copy the contents of an address register into a communication register if the communication register lock bit is clear. Set the lock bit.
- `snd.l`—Copy the contents of a scalar register into a communication register if the communication register lock bit is clear. Set the lock bit.
- `inc.w`—If the communication register lock bit is set, increment the contents of an address register by the contents of a communication register and copy the result into the address register.
- `inc.l`—If the communication register lock bit is set, increment the contents of a scalar register by the contents of a communication register and copy the result into the scalar register.

- `mat.w`—Compare the contents of a communication register and an address register.
- `mat.l`—Compare the contents of a communication register and a scalar register.

Classes 1, 2, 3, and 4 exercise the above instructions as follows:

- Class 1 tests exercise each instruction in a single processor.
- Class 2 tests exercise each instruction on multiple processors. The test sets all available processors to running identical processes at the same time. All processors attempt to execute the instruction on the same communication register. The processors then halt. The test checks the results.
- Class 3 tests exercise each instruction as in class 2 tests, but all processors attempt to execute the instruction many times. In each execution, all processors contend for the same communication register, as in the class 2 tests. Successive executions contend for access to successive communication register locations, until all communication register locations have been checked.
- Class 4 tests exercise each instruction on multiple processors, setting all available processors to executing the instruction on different communication registers. The test checks that all instructions were successful, without contention.

8.9.4 Class 5 through 8 descriptions

Test `cpu4333` classes 5, 6, 7, and 8 exercise the following machine instructions. The test sequence exercises each instruction through all 4 classes of tests sequentially, then exercises the next instruction through all 4 classes, and so forth.

The machine instructions have the following definitions:

- `sndr.w`—If a target resource is unlocked and its contents are not valid, copy the contents of an address register into the resource structure.
- `sndr.l`—If a target resource is unlocked and its contents are not valid, copy the contents of a scalar register into the resource structure.
- `putr.w`—Copy the contents of an address register into a resource structure if the target structure lock bit is not set.

- `putr.l`—Copy the contents of a scalar register into a resource structure if the target structure lock bit is not set.
- `getr.w`—Copy the contents of a resource structure into an address register if the resource structure lock bit is not set.
- `getr.l`—Copy the contents of a resource structure into a scalar register if the resource structure lock bit is not set.
- `matr.w`—Compare the contents of a resource structure and an address register.
- `matr.l`—Compare the contents of a resource structure and a scalar register.
- `casr`—Compare and swap a word between a resource structure and a memory location.
- `rcvr.w`—Copy the contents of a resource into an address register if the resource is unlocked and its contents are valid. Mark the resource contents as invalid.
- `rcvr.l`—Copy the contents of a resource into a scalar register if the resource is unlocked and its contents are valid. Mark the resource contents as invalid.
- `pshr`—Push the contents of an address register into a resource structure.
- `popr`—Pop a word from a resource structure into an address register.
- `incr.w`—If a target resource is unlocked and its contents are valid, increment the contents of the resource structure by the contents of an address register.
- `incr.l`—If a target resource is unlocked and its contents are valid, increment the contents of the resource structure by the contents of a scalar register.

Test `cpu4333` executes test class 9 at this point, before executing class 5, 6, 7, and 8 tests on the following instructions:

- `tas`—Set a cleared lock byte.
- `tac`—Clear a set lock byte.

Classes 5, 6, 7, and 8 exercise the previous instructions as follows:

- Class 1 tests exercise each instruction in a single processor.
- Class 2 tests exercise each instruction on multiple processors. The test sets all available processors to running identical processes at the same time. All processors attempt to execute the instruction on the same resource. The processors then halt. The test checks the results.
- Class 3 tests exercise each instruction as in class 2 tests, but all processors attempt to execute the instruction many times. In each execution, all processors contend for the same resource, as in the class 2 tests. Successive executions contend for access to different resources, until all resources have been checked.
- Class 4 tests exercise each instruction on multiple processors, setting all available processors to executing the instruction on different resources. The test checks that all instructions were successful, without contention.

In the case of the `tas` and `tac` instructions, the resource accessed is system memory.

8.9.5 Class 9 descriptions

Class 9 tests exercise the `pfork` and `cfork` instructions with and without an existing fork posted. They also verify that processors can uniquely pick up posted forks. Further they exercise the `wfork` instruction and verify that a processor continues to attempt a wfork if the target thread register is unlocked.

8.9.6 Class 10 descriptions

Class 10 tests exercise the `ldcmr` and `stcmr` instructions on a single processor and on several processors simultaneously. They verify the instructions on a single processor and verify that several processors can attempt to load one communication register without conflict. They also verify that several processors can store the contents of several communication registers into memory simultaneously.

8.9.7 Class 11 descriptions

Class 11 tests exercise the `trap` and `pbkpt` instructions on a single CPU with several CPUs operating. They verify that the `trap` instruction can cause a trap to operate on a single CPU even though other CPUs may be executing the same process in a ring of lower number. They also verify that the trap instruction can operate on all CPUs operating in the same ring, or in a higher number ring, than the target CPU.

Note

Subtests class 12 and 13 do not apply to the C3800 Series computers.

8.9.8 Class 14 descriptions

Class 14 tests exercise the following instructions with several processors operating:

- `interrupt`—Hardware interrupt.
- `deadlock`—Hardware trap to release a processor from an infinite loop due to multiprocessing conflict.
- `patu`—Purges all address translation unit entries.
- `pate`—Purges one address translation unit entry.
- `ctrsg`—Updates all CPU timers to the current time.

Class 14 tests also exercise the following actions:

- **Timer ring crossing**—Timers maintain records of the cumulative amount of time that each CPU operates in each ring. This test verifies that a timer counts when its assigned CPU is operating in its ring, that it stops counting when its assigned CPU moves to another ring, and that it retains the accumulated elapsed time value when it stops.
- **CPU execution timer**—Verifies that timer values are stored, and new timer values are fetched, when a CPU changes processes.

8.9.9 Subtest descriptions

Table 8-28 describes the `cpu4333` subtests.

Table 8-28
`cpu4333` subtests

Subtest	Class	Test performed
101	1	`lck <Ceffa>` (single head)
102	2	`lck <Ceffa>` (multiple heads, synchronized, stepped on same comm rgstrs)
103	3	`lck <Ceffa>` (multiple heads, synchronized on same comm rgstr locations)
104	4	`lck <Ceffa>` (multiple heads, synchronized on different comm rgstrs)
111	1	`ulk <Ceffa>` (single head)
112	2	`ulk <Ceffa>` (multiple heads, synchronized, stepped on same comm rgstrs)
113	3	`ulk <Ceffa>` (multiple heads, synchronized on same comm rgstr locations)
114	4	`ulk <Ceffa>` (multiple heads, synchronized on different comm rgstrs)
121	1	`tst <Ceffa>` (single head)
122	2	`tst <Ceffa>` (multiple heads, synchronized, stepped on same comm rgstrs)
123	3	`tst <Ceffa>` (multiple heads, synchronized on same comm rgstr locations)
124	4	`tst <Ceffa>` (multiple heads, synchronized on different comm rgstrs)
131	1	`get.w <Ceffa>,Ak` (single head)
132	2	`get.w <Ceffa>,Ak` (multiple heads, synchronized, stepped on same comm rgstrs)
133	3	`get.w <Ceffa>,Ak` (multiple heads, synchronized on same comm rgstr locations)
134	4	`get.w <Ceffa>,Ak` (multiple heads, synchronized on different comm rgstrs)
141	1	`get.l <Ceffa>,Sk` (single head)
142	2	`get.l <Ceffa>,Sk` (multiple heads, synchronized, stepped on same comm rgstrs)
143	3	`get.l <Ceffa>,Sk` (multiple heads, synchronized on same comm rgstr locations)
144	4	`get.l <Ceffa>,Sk` (multiple heads, synchronized on different comm rgstrs)
151	1	`put.w Ak,<Ceffa>` (single head)
152	2	`put.w Ak,<Ceffa>` (multiple heads, synchronized, stepped on same comm rgstrs)

Table 8-28 (continued)
cpu4333 subtests

Subtest	Class	Test performed
153	3	put.w Ak,<Ceffa> (multiple heads, synchronized on same comm rgstr locations)
154	4	put.w Ak,<Ceffa> (multiple heads, synchronized on different comm rgstrs)
161	1	put.l Sk,<Ceffa> (single head)
162	2	put.l Sk,<Ceffa> (multiple heads, synchronized, stepped on same comm rgstrs)
163	3	put.l Sk,<Ceffa> (multiple heads, synchronized on same comm rgstr locations)
164	4	put.l Sk,<Ceffa> (multiple heads, synchronized on different comm rgstrs)
171	1	rcv.w <Ceffa>,Ak (single head)
172	2	rcv.w <Ceffa>,Ak (multiple heads, synchronized, stepped on same comm rgstrs)
173	3	rcv.w <Ceffa>,Ak (multiple heads, synchronized on same comm rgstr locations)
174	4	rcv.w <Ceffa>,Ak (multiple heads, synchronized on different comm rgstrs)
181	1	rcv.l <Ceffa>,Sk (single head)
182	2	rcv.l <Ceffa>,Sk (multiple heads, synchronized, stepped on same comm rgstrs)
183	3	rcv.l <Ceffa>,Sk (multiple heads, synchronized on same comm rgstr locations)
184	4	rcv.l <Ceffa>,Sk (multiple heads, synchronized on different comm rgstrs)
191	1	snd.w Ak,<Ceffa> (single head)
192	2	snd.w Ak,<Ceffa> (multiple heads, synchronized, stepped on same comm rgstrs)
193	3	snd.w Ak,<Ceffa> (multiple heads, synchronized on same comm rgstr locations)
194	4	snd.w Ak,<Ceffa> (multiple heads, synchronized on different comm rgstrs)
201	1	snd.l Sk,<Ceffa> (single head)
202	2	snd.l Sk,<Ceffa> (multiple heads, synchronized, stepped on same comm rgstrs)
203	3	snd.l Sk,<Ceffa> (multiple heads, synchronized on same comm rgstr locations)

Table 8-28 (continued)
cpu4333 subtests

Subtest	Class	Test performed
204	4	snd.l Sk,<Ceffa> (multiple heads, synchronized on different comm rgstrs)
211	1	inc.w <Ceffa>,Ak (single head)
212	2	inc.w <Ceffa>,,Ak (multiple heads, synchronized, stepped on same comm rgstrs)
213	3	inc.w <Ceffa>,Ak (multiple heads, synchronized on same comm rgstr locations)
214	4	inc.w <Ceffa>,Ak (multiple heads, synchronized on different comm rgstrs)
221	1	inc.l <Ceffa>,Sk (single head)
222	2	inc.l <Ceffa>,Sk (multiple heads, synchronized, stepped on same comm rgstrs)
223	3	inc.l <Ceffa>,Sk (multiple heads, synchronized on same comm rgstr locations)
224	4	inc.l <Ceffa>,Sk (multiple heads, synchronized on different comm rgstrs)
231	1	mat.w Ak,<Ceffa> (single head)
232	2	mat.w Ak,<Ceffa> (multiple heads, synchronized, stepped on same comm rgstrs)
233	3	mat.w Ak,<Ceffa> (multiple heads, synchronized on same comm rgstr locations)
234	4	mat.w Ak,<Ceffa> (multiple heads, synchronized on different comm rgstrs)
241	1	mat.l Sk,<Ceffa> (single head)
242	2	mat.l Sk,<Ceffa> (multiple heads, synchronized, stepped on same comm rgstrs)
243	3	mat.l Sk,<Ceffa> (multiple heads, synchronized on same comm rgstr locations)
244	4	mat.l Sk,<Ceffa> (multiple heads, synchronized on different comm rgstrs)
251	5	sndr.w Ak,<effa> (single head)
252	6	sndr.w Ak,<effa> (multiple heads, synchronized, stepped on same resource)
253	7	sndr.w Ak,<effa> (multiple heads, synchronized on same resources)
254	8	sndr.w Ak,<effa> (multiple heads, synchronized on different resources)
261	5	sndr.l Sk,<effa> (single head)
262	6	sndr.l Sk,<effa> (multiple heads

Table 8-28 (continued)
cpu4333 subtests

Subtest	Class	Test performed
263	7	sndr.l Sk,<effa> (multiple heads, synchronized on same resources)
264	8	sndr.l Sk,<effa> (multiple heads, synchronized on different resources)
611	5	putr.w Ak,<effa> (single head)
612	6	putr.w Ak,<effa> (multiple heads, synchronized, stepped on same resource)
613	7	putr.w Ak,<effa> (multiple heads, synchronized on same resources)
614	8	putr.w Ak,<effa> (multiple heads, synchronized on different resources)
621	5	putr.l Sk,<effa> (single head)
622	6	putr.l Sk,<effa> (multiple heads, synchronized, stepped on same resource)
623	7	putr.l Sk,<effa> (multiple heads, synchronized on same resources)
624	8	putr.l Sk,<effa> (multiple heads, synchronized on different resources)
631	5	getr.w Ak,<effa> (single head)
632	6	getr.w Ak,<effa> (multiple heads, synchronized, stepped on same resource)
633	7	getr.w Ak,<effa> (multiple heads, synchronized on same resources)
634	8	getr.w Ak,<effa> (multiple heads, synchronized on different resources)
641	5	getr.l Sk,<effa> (single head)
642	6	getr.l Sk,<effa> (multiple heads, synchronized, stepped on same resource)
643	7	getr.l Sk,<effa> (multiple heads, synchronized on same resources)
644	8	getr.l Sk,<effa> (multiple heads, synchronized on different resources)
651	5	matr.w Ak,<effa> (single head)
652	6	matr.w Ak,<effa> (multiple heads, synchronized, stepped on same resource)
653	7	matr.w Ak,<effa> (multiple heads, synchronized on same resources)
654	8	matr.w Ak,<effa> (multiple heads, synchronized on different resources)
661	5	matr.l Sk,<effa> (single head)
662	6	matr.l Sk,<effa> (multiple heads, synchronized, stepped on same resource)
663	7	matr.l Sk,<effa> (multiple heads, synchronized on same resources)
664	8	matr.l Sk,<effa> (multiple heads, synchronized on different resources)
671	5	casr (single head)
271	5	rcvr.w <effa>,Ak (single head)

Table 8-28 (continued)
cpu4333 subtests

Subtest	Class	Test performed
272	6	rcvr.w <effa>, Ak (multiple heads, synchronized, stepped on same resource)
273	7	rcvr.w <effa>, Ak (multiple heads, synchronized on same resources)
274	8	rcvr.w <effa>, Ak (multiple heads, synchronized on different resources)
281	5	rcvr.l <effa>, Sk (single head)
282	6	rcvr.l <effa>, Sk (multiple heads, synchronized, stepped on same resource)
283	7	rcvr.l <effa>, Sk (multiple heads, synchronized on same resources)
284	8	rcvr.l <effa>, Sk (multiple heads, synchronized on different resources)
311	5	pshr Ak, <effa> (single head)
312	6	pshr Ak, <effa> (multiple heads, synchronized, stepped on same resource)
313	7	pshr Ak, <effa> (multiple heads, synchronized on same resources)
314	8	pshr Ak, <effa> (multiple heads, synchronized on different resources)
321	5	popr <effa>, Sk (single head)
322	6	popr <effa>, Sk (multiple heads, synchronized, stepped on same resource)
323	7	popr <effa>, Sk (multiple heads, synchronized on same resources)
324	8	popr <effa>, Sk (multiple heads, synchronized on different resources)
331	5	incr.w <effa>, Ak (single head)
332	6	incr.w <effa>, Ak (multiple heads, synchronized, stepped on same resource)
333	7	incr.w <effa>, Ak (multiple heads, synchronized on same resources)
334	8	incr.w <effa>, Ak (multiple heads, synchronized on different resources)
341	5	incr.l <effa>, Sk (single head)
342	6	incr.l <effa>, Sk (multiple heads, synchronized, stepped on same resource)
343	7	incr.l <effa>, Sk (multiple heads, synchronized on same resources)
344	8	incr.l <effa>, Sk (multiple heads, synchronized on different resources)
351	9	pfork <effa>, Ak (multiple headed pfork with an outstanding fork request)
352	9	pfork <effa>, Ak (multiple headed pfork with no outstanding fork request)
353	9	pfork <effa>, Ak (multiple headed pfork with no outstanding fork request)
361	9	cfork (multiple headed cfork with no outstanding fork request)
362	9	cfork (multiple headed cfork with an outstanding fork request)

Table 8-28 (continued)
cpu4333 subtests

Subtest	Class	Test performed
371	9	wfork (multiple headed wfork)
372	9	wfork (verify a wfork spins if thread rgstr is unlocked)
381	9	spawn <effa>,Ak (multiple headed spawn with an outstanding fork request)
382	9	spawn <effa>,Ak (multiple headed spawn with no outstanding fork request)
384	9	spawn <effa>,Ak (verify all threads can be uniquely picked up)
391	9	join (multiple headed wfork)
392	9	join (verify a join spins if thread rgstr is unlocked)
401	9	idle (multiple headed idle)
405	9	Secure pfork, spawn operation
431	5	tas <effa> (single head)
432	6	tas <effa> (multiple heads, synchronized, stepped on same memory)
433	7	tas <effa> (multiple heads, synchronized on same memory locations)
434	8	tas <effa> (multiple heads, synchronized on different memory)
441	5	tac <effa> (single head)
442	6	tac <effa> (multiple heads, synchronized, stepped on same memory)
443	7	tac <effa> (multiple heads, synchronized on same memory locations)
444	8	tac <effa> (multiple heads, synchronized on different memory)
456	10	ldcomm rgstr <effa>,Ak (single head)
457	10	ldcomm rgstr <effa>,Ak (multiple heads, same comm rgstr set)
458	10	ldcomm rgstr <effa>,Ak (multiple heads, different comm rgstr set)
466	10	stcomm rgstr Ak,<effa> (single head)
467	10	stcomm rgstr Ak,<effa> (multiple heads, different comm rgstr set)
471	11	trap #rm,#bit (multiple heads, single head execution, same comm index rgstr)
472	11	trap #rm,#bit (multiple heads, single head execution, diff comm index rgstr)
476	11	pbkpt (multiple heads, single head execution, same comm index rgstr)
477	11	pbkpt (multiple heads, single head execution, diff comm index rgstr)
478	11	TER/pbkpt (multiple heads, single head execution, diff comm index rgstr)

Table 8-28 (continued)
cpu4333 subtests

Subtest	Class	Test performed
1005	14	Interrupt test (multiple heads)
1010	14	Deadlock test (multiple heads)
1020	14	patu test (multiple heads)
1021	14	TER/global patu test (multiple heads)
1022	14	TER/local patu test (multiple heads)
1025	14	pate test (multiple heads)
1026	14	TER/global pate test (multiple heads)
1027	14	TER/local pate test (multiple heads)
1035	14	ctrsh test (multiple heads)
1036	14	TER/ctrsh test (multiple heads)
1040	14	Multi-headed timer update test

8.10 CPUcti diagnostic test error display

All CPUcti diagnostic test failures produce screen dumps of CPU registers. A screen dump appears for each CPU generating an error. Figure 8-4 shows a typical CPU error screen dump.

Figure 8-4
CPUcti diagnostic test error display

```
          subtest     0 [CPU1]: FAILED (00:00:01)
Register state for CPU 1
pc=00000000 upc=00000050 cir=00000001 tid=00000000 ccr=00000020
ps=00000000   (XF)
sp=fbf395ff a1=00000001 a2=2dfbe5f8 a3=eff13a25
a4=24000dde a5=26252a37 ap=196824df fp=81733845
s0=d9abbd97526f22f7 s1=59fbafbd527bb995 s2=dc695fef441f733c s3=00f7cfcd0779139b
s4=d100f3b0cd036cb7 s5=759fdfbbd3eb8fef s6=004a005d00cb2e3e s7=7f367cfc10000000
t0=00000020 t1=00000000 t2=4bdcd37e t3=ff19fb00
t4=6eecf9ab t5=10000020 t6=4b7dceff t7=00035014
v1=00000000 vs=00000000 vv=00000001
vm=00000000000000000000000000000000
```

Troubleshooting with CXTS 9

9.1 Overview

This chapter introduces the CONVEX Expert Troubleshooting System (CXTS) as it applies to the CONVEX C3800 Series computers. The chapter provides:

- Instructions on invoking CXTS and accessing CXTS online documentation.
- A top level description of CXTS operation.

Refer to CXTS documentation for specifics on using CXTS for troubleshooting.

9.2 CXTS documentation

CXTS documentation is available in both online and hard copy form. Documentation names are:

- *CONVEX CXTS Guide (C3800 Series)*, available online in the C3800 SPU.
- *CONVEX CXTS User's Guide (C3800 Series)*, Order No. DHW-310, available in hard copy.

These documents provide comprehensive details of CXTS and detailed descriptions of its use.

9.3 Accessing CXTS

The `xsfp` utility invokes CXTS as part of the boot process. The display of an operational C3800 SPU displays either the `cxts_ui` window or the `cxts_ui` icon. If the window is not present, click on the icon to display the window.

9.4 CXTS execution

The CXTS Setup window, available through the cxts_ui window, contains control buttons governing CXTS operation:

- If the AutoTroubleshoot and Autoreboot buttons are not selected, CXTS logs significant events but does not attempt any troubleshooting activity.
- If the AutoTroubleshoot and Autoreboot buttons are selected, CXTS logs significant events. If a significant event occurs and ConvexOS goes down, CXTS attempts to discover the fault and reboot ConvexOS.
- If the AutoTroubleshoot, Autoreboot, and ReconfigurationAllowed buttons are selected, CXTS logs significant events. If a significant event occurs and ConvexOS goes down, CXTS attempts to discover the fault. If CXTS isolates a faulty FRU, it disables the FRU and attempts to reconfigure and reboot ConvexOS.

Click on Document in the cxts_ui window to cause the title page of *CONVEX CXTS Guide (C3800 Series)* to appear. Click on any of the control buttons on the document pages to browse through the document, or to execute CXTS commands. The document explains how to use the control buttons. Click on any page number in the table of contents to display that page.

CXTS operations continue whether or not the cxts_ui window is displayed.

9.5 Automatic notification

You may instruct CXTS to send a message and access a pager number if it logs a significant event. A number of options are available for formatting the message and taking action as a result of the event.

9.6 Report generation

You may instruct CXTS to generate a report as a result of an event.

Using ddb

10

The ddb utility provides a way to debug the C3800 Series system, to debug the CPU diagnostics, and to access C3800 hardware state via a single utility.

This chapter summarizes some ddb commands and operations that may be useful in troubleshooting a C3800 system.

10.1 Starting ddb

To invoke ddb, enter at the SPU dsh prompt:

```
dsh> ddb
```

ddb performs the following actions at startup:

- Determines the first CPU in the system.
- Assigns that CPU ID as the default CPU and the default communication index register (CIR).
- Sets the default thread ID to zero.
- Sets the default memory accessing mode to use logical addressing.
- Removes the page_map file if one exists. The page_map file indicates to ddb the contents of main memory.

On a system with CPUs in ports 2 and 3, which have been successfully powered and initialized, and which are selected for operation from the xsys_config utility, the initial ddb prompt looks like this:

```
cpu:2,cir:2,tid:0
[DDB]->
```

ddb has a two line prompt. To exit ddb, enter:

```
[DDB]-> $q
```

10.2 Setting ddb default parameters

Many commands executed under ddb operate based on the current set of defined ddb parameters. The defined set of ddb default parameters are:

- default_cpu
- default_cir
- default_tid
- The memory accessing mode

10.2.1 Changing default_cpu

The command $dcpu #, where # is a valid CPU ID, changes the default CPU for subsequent ddb commands. For example, executing the command $dcpu 3 gives the following ddb prompt:

```
cpu:3,cir:2,tid:0
[DDB]->
```

If the CPU ID input by the user is invalid (out of range, or the CPU is not available for testing), an error message appears and the current default_cpu value remains.

10.2.2 Changing default_cir

The command $dcir #, where # is a valid communication index register (CIR), changes the default CIR for the subsequent ddb commands. For example, executing the command $dcir 6 gives the following ddb prompt:

```
cpu:3,cir:6,tid:0
[DDB]->
```

If the CIR input by the user is invalid, an error message appears and the current default_cir value remains.

10.2.3 Changing default_tid

The command $dtid #, where # is a valid thread ID, changes the default thread ID (tid) for the subsequent ddb commands. For example, executing the command $dtid 4 gives the following ddb prompt:

```
cpu:3,cir:6,tid:4
[DDB]->
```

If the tid input by the user is out of the 0 to 31 range, an error message is displayed and the current default_tid value remains.

10.2.4 Changing the memory accessing mode

When ddb starts, it sets the default memory mode to logical addressing.

- In the logical addressing memory mode, all C3800 memory addresses are translated using virtual to physical address translation.
- In the physical addressing memory mode, all memory accesses are treated as physical addresses and no address translation is done.

Use the command $mmode <mode> to change the default memory mode, where <mode> is either phys or log. Entering $mmode with no mode will cause ddb to display the current default memory accessing mode. Figure 10-1 is an example of changing the default memory accessing mode to physical, to logical, and then asking for the current default memory access mode.

Figure 10-1
Changing default memory accessing mode

```
cpu:3,cir:6,tid:4
[DDB]->$mmode phys
Using PHYSICAL addressing

1Mcpu:3,cir:6,tid:4
[DDB]->$mmode log
Using LOGICAL addressing

cpu:3,cir:6,tid:4
[DDB]->$mmode
Current memory access mode is: Logical
```

10.3 Displaying register information

The register state information of the C3800 system is in 4 categories:

- Scalar information
- Vector information
- Communication register information
- Control space information

10.3.1 Displaying scalar information

Two commands display scalar information:

- `$r` displays the scalar register information for the default CPU.
- `$R` displays the scalar register information for all CPUs installed in the system and available for testing.

With the default CPU set to 3, executing `$r` at the `ddb` prompt displays the scalar register state as shown in Figure 10-2. The display is similar for the `$R` command.

Figure 10-2
Scalar register state display

```
Register state for CPU 3
pc=00000000 upc=00000000 cir=00000000 tid=00000000 ccr=00000000
ps=00000000  (XF)
sp=00000000 a1=00000000 a2=00000000 a3=00000000
a4=00000000 a5=00000000 ap=00000000 fp=00000000
s0=0000000000000000 s1=0000000000000000 s2=0000000000000000
s3=0000000000000000
s4=0000000000000000 s5=0000000000000000 s6=0000000000000000
s7=0000000000000000
t0=00000000 t1=00000000 t2=00000000 t3=00000000
t4=00000000 t5=00000000 t6=00000000 t7=00000000
vl=00000000 vs=00000000 vv=00000000
vm=00000000000000000000000000000000
```

10.3.2 Displaying vector information

Two commands display vector information:

- `$v?1` displays all vector register information for the default CPU. With the default CPU set to 3, executing the `$v?1` at the `ddb` prompt displays the vector register state as shown in Figure 10-3.

Figure 10-3
Vector register state display

```
vl=00000000
vs=00000000
vm=00000000000000000000000000000000
v0[000]=0000000000000000
v0[001]=0000000000000000
v0[003]=0000000000000000
    .
    .
    .
v0[128]=0000000000000000
v1[000]=0000000000000000
v1[001]=0000000000000000
v1[003]=0000000000000000
    .
    .
    .
v1[128]=0000000000000000
    .
    .
    .
v7[000]=0000000000000000
v7[001]=0000000000000000
v7[003]=0000000000000000
    .
    .
    .
v7[128]=0000000000000000
```

- `$V?1` displays all vector register information for all CPUs installed in the system and available for testing. A banner before each display indicates the relevant CPU.

10.4 Displaying communication register information

The communication registers are accessible via the `$h?1` command. Executing this command displays both the register content and the lock value associated with each of the communication registers as shown in Figure 10-4.

Figure 10-4
Communication register display

```
cir<0>[0x0000]=0000000000000000 (0)
cir<0>[0x0001]=0000000000000000 (0)
    .
    .
cir<0>[0x001f]=0000000000000000 (0)
cir<0>[0x4000]=0000000000000000 (0)
cir<0>[0x4001]=0000000000000000 (0)
    .
    .
cir<0>[0x401f]=0000000000000000 (0)
cir<0>[0x8000]=0000000000000000 (0)
cir<0>[0x8001]=0000000000000000 (0)
    .
    .
cir<0>[0x803f]=0000000000000000 (0)
cir<1>[0x0000]=0000000000000000 (0)
cir<1>[0x0001]=0000000000000000 (0)
    .
    .
cir<1>[0x001f]=0000000000000000 (0)
cir<1>[0x4000]=0000000000000000 (0)
cir<1>[0x4001]=0000000000000000 (0)
    .
    .
cir<1>[0x401f]=0000000000000000 (0)
cir<1>[0x8000]=0000000000000000 (0)
cir<1>[0x8001]=0000000000000000 (0)
    .
    .
cir<1>[0x803f]=0000000000000000 (0)
    .
    .
cir<31>[0x803f]=0000000000000000 (0)
```

Each line of the display information has the following format:

- The value <0> is the CIR value. On a C3800, this value can range from 0 to 31.
- The value [0x0000] indicates the virtual communication register associated with the CIR. On a C3800, the valid virtual communication register addresses are 0x0000 to 0x001f (hardware registers), 0x4000 to 0x401f (ring 0 software registers), and 0x8000 to 0x803f (ring 4 software registers.)
- The 64-bit value following the virtual address is the data stored in the communication register.
- The final value enclosed in parentheses is the state of the lock for the virtual address.

10.5 Displaying control space information

The `$xspace` command displays control space information as shown in Figure 10-5.

Figure 10-5
Control space display

```
        CONTROL SPACE DATA
SYSTEM INTERRUPTS ARE DISABLED
SYSTEM GLOBAL INTERRUPT ENABLES = 0x00
SYSTEM GLOBAL PENDING INTERRUPTS = 0x00
CPU LOCAL ENABLES:
        CPU0 = 0x00     CPU1 = 0x00     CPU2 = 0x00     CPU3 = 0x00
        CPU4 = 0x00     CPU5 = 0x00     CPU6 = 0x00     CPU7 = 0x00
INTERRUPT BROADCAST ENABLES:
        CHAN0 = 0x00    CHAN1 = 0x00    CHAN2 = 0x00    CHAN3 = 0x00
        CHAN4 = 0x00    CHAN5 = 0x00    CHAN6 = 0x00    CHAN7 = 0x00
CPU CIR:
        CPU0 CIR=0      CPU1 CIR=0      CPU2 CIR=0      CPU3 CIR=0
        CPU4 CIR=0      CPU5 CIR=0      CPU6 CIR=0      CPU7 CIR=0
CPU IDLE STATUS:
        CPU0 NOT IDLE   CPU1 NOT IDLE   CPU2 NOT IDLE   CPU3 NOT IDLE
        CPU4 NOT IDLE   CPU5 NOT IDLE   CPU6 NOT IDLE   CPU7 NOT IDLE
```

10.6 Displaying cache information

The C3800 machine has a data cache, a page table entry (PTE) cache, an instruction cache, and a return control queue available for inspection as part of machine debug. Because of the detailed information contained in each of the caches, only the commands to display the data are described here.

10.6.1 Displaying the data cache

Two ddb commands display data cache information: `$dcache` and `$Dcache`. The `$dcache` command displays the entire contents of the data cache on the CPU identified by the default CPU. The `$Dcache` command displays the data caches for all CPUs in the system.

10.6.2 Displaying the instruction cache

Two ddb commands display instruction cache information: `$icache` and `$Icache`. The `$icache` command displays the entire contents of the instruction cache on the CPU identified by the default CPU. The `$Icache` command displays the instruction caches for all CPUs in the system.

10.6.3 Displaying the PTE cache

Two ddb commands display PTE cache information: `$ptecache` and `$Ptecache`. The `$ptecache` command displays the entire contents of the PTE cache on the CPU identified by the default CPU. The `$Ptecache` command displays the PTE caches for all CPUs in the system.

10.6.4 Displaying the return control queue

Two ddb commands display return control queue information: `$rcque` and `$Rcque`. The `$rcque` command displays the entire contents of the return control queue on the CPU identified by the default CPU. The `$Rcque` command displays the the return control queue data for all CPUs in the system.

10.7 Displaying C3800 main memory

The ddb command `address` displays data stored in memory. It has the following syntax:

`address[,`*<number>*`]/b|h|w|l|i[a]`

The parameters have the following definitions:

- `address` is either a physical or logical address, based on the memory accessing mode.
- *<number>* is an optional decimal integer indicating the number of sequential memory locations to be displayed, beginning at `address`. Default is one address location.
- `/` is a required specifier identifying the access as a memory access.
- `b` indicates that *<number>* bytes are displayed.
- `h` indicates that *<number>* halfwords are displayed.
- `w` indicates that *<number>* words are displayed.
- `l` indicates that *<number>* longwords are displayed.
- `i` indicates that *<number>* instructions are displayed.
- `a` is optional. It causes the address of each memory location to appear with the data from that location.

10.8 Loading CPU tests

The command `$test` loads a CPU test into main memory. The syntax for this command is `$test testname segment` where test name is `cpu4030`, `cpu4041`, `cpu4241`, `cpu4331`, `cpu4332`, or `cpu4333`, and `segment` is a value 0 through 7. The command performs the following tasks:

- Loads all modules for the test.
- Links the modules together.
- Initializes all CPU global variables.
- Initializes the communication registers.

The command `$clear` removes a previously loaded CPU test from the internal `ddb` structures. For example, if you execute `$test cpu4030 0`, `ddb` has `cpu4030` in residence. If you execute another `$test cpu4030 0`, a logical address collision occurs. Executing `$clear` before invoking the second `$test` removes all knowledge of the previously loaded test.

The `$dmem` command dumps the following information about the CPU test:

- How the test was loaded into main memory.
- What logical addresses and physical addresses the code segment were loaded into.
- Physical addresses of the PTE tables.

10.9 Displaying CPU status

The command `$status` gives a complete view of the ddb and CPU environment configuration. Executing `$status` produces a display similar to that of Figure 10-6.

Figure 10-6
CPU status display

MEMORY MODE	STOP MODE	BKPT MODE
LOGICAL	NORMAL	DIAG

CPUID	CPUSTATE	CPUMODE
CPU0	UNINST	PARKED
CPU1	UNINST	PARKED
CPU2	HALTED	IDLE
CPU3	HALTED	IDLE
CPU4	UNINST	PARKED
CPU5	UNINST	PARKED
CPU6	UNINST	PARKED
CPU7	UNINST	PARKED

```
Test name = 'NO TEST SPECIFIED' is loaded into segment '0'
data_ffaults = 'OFF'            ip_ffaults = 'OFF'
enable_dcache = 'ON'            vl_count = '16'
seq_mode = 'OFF'                chain_mode = 'OFF'
parallel_mode = 'OFF'           multi_cirs = 'ON'
scn_ovr = 'ON'                  secure_mode = 'OFF'
```

This status display gives the following information:

- The memory access mode is logical.
- CPUs 0, 1, and 4 through 7 are uninstalled (hence they are PARKED).
- CPUs 2 and 3 are installed with clocks HALTED.
- No CPU test has been loaded.

The fields below `Test name` display the CPU test parameters. The values of each field can be altered by preceding the parameter name by a $ followed by a value. For all parameters except `vl_count`, the value is 0 or 1. For `vl_count`, the value is 0x0 to 0x80.

10.10 Configuring CPUs for testing

The `$status` display has fields for every CPU in a system:

- The `CPUSTATE` field indicates whether or not a CPU is installed. If a CPU is installed, then the actual state of CPU clock are displayed (`HALTED` or `CLKS_ON`.) Hence, you always see either `UNINST`, `HALTED`, or `CLKS_ON` for the `CPUSTATE` on each CPU.
- The field to the right of the individual `CPUSTATE` is the `CPUMODE`. If a CPU is uninstalled, this value is `PARKED`. If a CPU is installed, the CPU is under your control.

To run a CPU test from `ddb`, first determine which CPUs are to be tested. For example, to test CPU2 and not CPU3, execute `$run 2` and `$park 3`. When the test starts, only CPU2 receives clocks.

For CPU tests `cpu4030`, `cpu4041`, `cpu4241`, `cpu4331`, and `cpu4332`, always set the CPUs to either `PARKED` or `RUN`; *never* `IDLE`.

For test `cpu4333`, select one CPU to be in the `RUN` mode; select all other CPUs to be in `IDLE` mode. For example, to run CPU3 as the controlling CPU, execute the `ddb` commands `$run 3` and `$idle 2`.

10.10.1 Parking a CPU

The `ddb` command `$park #` places an installed CPU into a mode where it is not tested. This result of executing this command places the identified CPU in the `PARKED` mode when the CPU test starts.

10.10.2 Idling a CPU

The `ddb` command `$idle #` places the CPU number into the idle loop when a CPU test starts.

10.10.3 Running a CPU

The `ddb` command `$run #` places the CPU number in the `RUN` mode when a CPU test starts.

10.10.4 Halting CPU clocks

The ddb command `$halt #` places the CPU number in the HALT mode if the CPU is in RUN mode.

10.11 CPU test execution

To execute a subtest of a loaded CPU test, enter the ddb command:

```
[DDB]-> address:r
```

where `address` is the starting address of the test. This address should be consistent with the memory accessing mode. If the address is a logical address, you can enter a symbolic name for the address. When you enter `address:r`, the ddb utility blocks user input during test execution, allowing input to ddb to come from a file. If you do not want ddb to block user input during subtest execution, press **CONTROL-c**.

10.12 CPU test termination

Any subtest in a CPU test exits in one of 3 ways:

- It can pass, indicated by 0x100 in a1.
- It can fail, indicated by anything other than 0x100 in a1.
- It can cause a hard error.

If a subtest passes or fails, the control of ddb returns to you and the monitor displays the contents of a1 and the instruction to which the program currently points. If a hard error is generated, `hard_logger` dumps the cause of the hard error and control returns to you.

If the subtest hangs, press **CONTROL-c** to unblock ddb command input and enter `$status` to determine the state of the CPUs. If a CPU is has its clocks on (`CLKS_ON`), executing `$halt #` turns off clocks for CPU#. This is necessary before you can inspect any register state on the CPU.

10.13 CPU test examples

This set of examples clarifies the operation of ddb.

10.13.1 Running a test

Enter the ddb commands shown in Figure 10-7 to run cpu4041 in segment 0, data forced faults, IP forced faults, vl of 0x10, and chain mode on CPUs 0 and 1.

Figure 10-7
Running a diagnostic test

```
[DDB]-> $clear
[DDB]-> $test cpu4041 0
[DDB]-> $data_ffaults 1
[DDB]-> $ip_ffaults 1
[DDB]-> $vl_count 10
[DDB]-> $run 0 1
[DDB]-> $chain_mode 1
[DDB]-> Chain_entrypoint:r
```

10.13.2 Reading CPUs

Use the sequence shown in Figure 10-8 to read CPUs 0 and 1 in the current subtest.

Figure 10-8
Reading CPU data from the current subtest

```
[DDB]-> $dcpu 0
[DDB]-> $dcir 0
[DDB]-> Current_subtest/w
[DDB]-> $dcpu 1
[DDB]-> $dcir 1
[DDB]-> Current_subtest/w
```

10.13.3 Running selected subtests

Figure 10-9 shows how to run selected subtests.

Figure 10-9
Running selected subtests

```
spu> ddb
cpu:0,cir:  0, tid:  0
[DDB]-> $park 1
$park: CPU1 placed into parked mode
cpu:0,cir:  0,tid:  0
[DDB]-> $run 0
$run:  CPU0 placed into run mode
cpu:0,cir:  0,tid:  0
[DDB]-> st_100:r
cpu0 halt:  a1=00000100
        cputest_complete+0xe: halt      #0x100,a1
cpu:0,cir: 0,tid:0
[DDB]-> st_271:r
cpu0 halt:  a1=00000100
        cputest_complete+0xe: halt      #0x100,a1
cpu:0,cir: 0, tid: 0
[DDB]-> $q
spu>
```

The commands have the following effects:

- `$park 1`—Prevents CPU 1 from being tested
- `$run 0`—CPU 0 is tested
- `st_100:r`—Start subtest 100 of this test
- `st_271:r`—Start subtest 271 of this test
- `$q`—Exit test

If `a1=00000100`, the test passes. If it is equal to any other value, the test fails. At this point you can look at the CPU registers by using the command:

```
[DDB]-> $r
```

10.13.4 Running a test in chained mode

Figure 10-10 shows how to initiate a complete test for CPU 1, in chained mode.

Figure 10-10
Commanding CPU tests in chained mode

```
spu> ddb
cpu:0, cir: 0,tid: 0
[DDB]-> $test cpu4332 0
loading p0r0_4332.ldr at logical offset 0x00000000
loading cpu4332.ldr at logical offset 0x00022000
Test loading completed, initializing Comm Regs
Comm Reg initialization completed, setting up SDRs
cpu:0,cir: 0,tid: 0
[DDB]-> $run 1
$run; CPU1 placed into run mode
cpu:0,cir: 0,tid: 0
[DDB]-> $park 0
$park: CPU0 placed into parked mode
cpu:0,cir: 0,tid 0
[DDB]-> $chain_mode 1
chain_mode = 'ON'
cpu:0,cir: 0,tid: 0
[DDB]-> Chain_entrypoint:r
```

The following comments apply to the commands:

- `$run 1`—CPU 1 is tested
- `$park 0`—CPU 0 is not tested
- `$chain_mode 1`—Activate chain mode
- `Chain_entrypoint:r`—No $ sign; C is uppercase

This test in the chained mode should complete within 5 minutes.

10.13.5 Test failure

Figure 10-11 is an example of a `cpu4030` failure. The test was run on one CPU with normal defaults.

Figure 10-11
Test failure example

```
spu> ddb
cpu:6,cir: 6,tid: 0
[DDB]-> $test cpu4030 0
loading p0r0_4030.ldr at logical offset 0x00000000
    .
    .
    .
loading wrapl_4030.ldr at logical offset 0xe0000000
Test loading completed, initializing Comm Regs
Comm Reg initialization completed, setting up SDRs
cpu:6,cir: 6,tid: 0
[DDB]-> $park 7
$park: CPU7 placed into parked mode
cpu:6,cir: 6,tid: 0
[DDB]-> $run 6
$run: CPU6 placed into run mode
cpu:6,cir: 6,tid: 0
[DDB]-> st_10:r
****
CPU6 halt:  a1=00005008  FAILED
        l_st_305+0x2: halt       #0x5008,a1
cpu:6,cir: 6,tid: 0
[DDB]-> $r
Register state for CPU 6
pc=000000e0 upc=00000bfd cir=00000006 tid=00000000 ccr=80000020
ps=00000000  (XF)
sp=00002f90 a1=00005008 a2=0000aabb a3=0000ccdd
a4=0000eeff a5=00000000 ap=00008899 fp=00002f90
s0=000000a000000020 s1=0000000043800000 s2=000014484a742404
s3=000000004a742403
s4=33333333ca742404 s5=0000000055555555 s6=00000000aaaaaaaa
s7=000000000000ffff
t0=00000000 t1=00000000 t2=80000020 t3=00000000
t4=00000000 t5=00000001 t6=00000008 t7=00000002
vl=00000000 vs=00000000 vv=00000001
vm=00000000000000000000000000000000
cpu:6,cir: 6,tid: 0
[DDB]-> $q
```

10.13.6 Diagnostic test output

Figure 10-12 is an example of a `cpu4041` test in the chained mode.

Figure 10-12
Typical diagnostic test output

```
spu> ddb
cpu:6,cir: 6,tid: 0
[DDB]-> $test cpu4041 0
loading p0r0_4041.ldr at logical offset 0x00000000
loading cpu4041.ldr at logical offset 0x00008000
Test loading completed, initializing Comm Regs
Comm Reg initialization completed, setting up SDRs
cpu:6,cir: 6,tid: 0
[DDB]-> $run 6
$run: CPU6 placed into run mode
cpu:6,cir: 6,tid: 0
[DDB]-> $park 7
$park: CPU7 placed into parked mode
cpu:6,cir: 6,tid: 0
[DDB]-> $chain_mode 1
chain_mode = 'ON'
cpu:6,cir: 6,tid: 0
[DDB]-> Chain_entrypoint:r
CPU6 halt:  a1=00000100  PASSED
        Chain_entrypoint_end: halt       #0x100,a1
cpu:6,cir: 6,tid: 0
[DDB]-> $q
* * * * * Last command returned status 1 * * * * *
spu>
```

Diagnostic utilities man pages

A

This appendix introduces man pages and includes the hard copy for the CONVEX C3800 Series computer diagnostic utilities.

CONVEX diagnostic utilities man pages are available in two formats:

- Online man pages accessible by using the ConvexOS `man` command
- Hard copy man pages

A.1 Using man pages

Man pages describe the commands, system calls, utilities, and file formats available for CONVEX software.

Man pages are organized into eight sections:

- **Section 1**—Commands and application programs
- **Section 2**—System calls
- **Section 3**—Subroutine libraries
- **Section 4**—Special files; each file described in this section refers to an I/O device
- **Section 5**—Structures of each type of system file
- **Section 6**—Reserved for games in BSD 4.2-based documentation (CONVEX software does not include games. There are no CONVEX man pages for this section.)
- **Section 7**—Commands used for accessing document-formatting and code-generating macros
- **Section 8**—Commands used for system management and maintenance

When a reference is made to a man page, the section number follows it in parentheses.

Example: altsetpts(1D).

A.1.1 Man page contents

The contents of each man page is divided into subsections (not all are relevant to each entry):

- **NAME**—Lists exact name of command or subroutine and describes its purpose.
- **SYNOPSIS**—Summarizes use of command or subroutine being described.
- **DESCRIPTION**—Discusses use of command or subroutine.
- **FILES**—Names files built into the program.
- **SEE ALSO/REFERENCES**—Points to related information.
- **DIAGNOSTICS/ERRORS**—Discusses diagnostic messages the system produces.
- **BUGS/NOTES**—Explains known bugs or deficiencies and any known solutions.

A.1.2 Online man pages

The `man` utility formats and displays information from the hard copy man pages. When invoked in the simplest way, without any options and without a topic, it displays the corresponding man page formatted with nroff. Access online man pages by entering:

```
% man entryname
```

where *entryname* is the name of the man page to be displayed. You can print hard copies of online man pages by entering:

```
% man -t entryname
```

For more information on man options, refer to the man(1) man page.

A.2 Hard copy man pages

The man pages in this appendix are in alphabetical order.

altsetpts(1D)
backup(1)
bpc(4D)
bpccommd(1D)
bpcwatchd(1D)
ccu(4D)
cdb_browser(1D)
cdb_dump(1D)
cdb_get(1D)
cdbserver(1D)
cdb_startup(1D)
cdb_update(1D)
config_data(5D)
CONVEXOS_CONSOLE(1D)
cop(1D)
cop_contents(5D)
cpualloc(1D)
cs(1D)
cti(5D)
diaginit(1D)
display_log (1D)
errintd(1D)
errlogd(1D)
event_browser(1D)
initall(1D)
margin(1D)
mminit(1D)
mm_sniff(1D)
powerdown(1D)
powermon(1D)
powerup(1D)
pwr_util(1D)
rbcdb_init(1D)
rbserver(1D)
remote_disconnect(1D)
scan_shm_init(1D)
scn_util(1D)
sfp(1D)
softlog(5D)
sys_shutdown(1D)
sysreset(1D)
wi(4D)
wndw(4D)
xcdb_browser(1D)
xdiag(1D)
xevent_browser(1D)

xpowermon(1D)
xsfp(1D)
xsys_config(1D)

NAME

altsetpts – modify a logic board's voltage busses

SYNOPSIS

altsetpts [-vcc value] [-vee10k value] [-vee value] [-vtt value] [-vttga value] [-xovtt value] [-xovttga value] [-xovee value] [-xevtt value] [-xevttga value] [-xevee value] {[xbar] | [boardname[#]]}

altsetpts [-t [-n] [-h <hot>] [-w <warm>]] <[bay#] ... [boardname#]...>

DESCRIPTION

Altsetpts alters the voltage bus values of a particular bus on a particular board. Any or all of the busses can be altered. However, only one logic board type may be altered at a time. The crossbar boards share a power pallet so any setpoint modifications affect all crossbar boards. Any invalid option in the command line aborts the entire command.

Altsetpts is also used to modify a board or bay's warm and hot temperature setpoints. The warm setpoint defines the temperature at which a warning will be generated. A hot setpoint defines the temperature at which the board or bay is shut down. The temperature value units are celsius degrees.

USAGE

xbar Indicates that this command applies to **xbar** boards.

boardname[#]

Indicates that this command applies to a logic board. Valid values are **mb** for the memory board, **sp** for the scalar processor, **vp** for the vector processor, **ia** for the interface adapter board, **cu** for the cpu utilities board, **ccu** for a set of ccu's, **cdb** for the cast data pod board, **cmp** for the cast memory pod board. If the port number is not specified all boards of that type will have their busses modified. If a particular bus option is not specified the user will be prompted for input on each bus of the board specified. NOTE: when altering the temperature setpoints for a board, the port number must be specified.

bay# Set the temperature setpoints for the specified bay.

-vcc value

Change the setpoint of the VCC bus of a particular board. The specified value must be within the acceptable range for the VCC bus. This option is invalid for the crossbar (xbar).

-vee10k value

Change the setpoint of the VEE10K bus of a particular board. The specified value must be within the acceptable range for the VEE10K bus. This option is invalid for the crossbar (xbar).

-vee value

Change the setpoint of the VEE bus of a particular board. The specified value must be within the acceptable range for the VEE bus. This option is invalid for the crossbar (xbar).

-vtt value

Change the setpoint of the VTT bus of a particular board. The specified value must be within the acceptable range for the VTT bus. This option is invalid for the crossbar (xbar).

-vttga value
Change the setpoint of the VTTGA bus of a particular board. The specified value must be within the acceptable range for the VTTGA bus. This option is invalid for the crossbar (xbar).

-xevtt value | -xovtt value
Change the setpoint of the even or odd side of the XVTT bus of a particular board. The specified value must be within the acceptable range for the XVTT bus. This bus is valid for the crossbar only.

-xevttga value | -xovttga value
Change the setpoint of the even or odd side of the XVTTGA bus of a particular board. The specified value must be within the acceptable range for the XVTTGA bus. This bus is valid for the crossbar only.

-xevee value | -xovee value
Change the setpoint of the even or odd side of the XVEE bus of a particular board. The specified value must be within the acceptable range for the XVEE bus. This bus is valid for the crossbar only.

-t Indicates that a board or bay's temperature setpoints are to be modified. This option will prompt the user to input a new temperature setpoint after displaying the current value.

-n Set the temperature setpoints to their nominal values.

-h Set the *hot* temperature setpoints for a board or bay.

If a bay is being modified then this is the setpoint for the inlet bay sensor. The *hot* temperature setpoint for the outlet bay sensor is calculated from this specified value.

-w Set the *warm* temperature setpoints for a board or bay.

If a bay is being modified then this is the setpoint for the inlet bay sensor. The *warm* temperature setpoint for the outlet bay sensor is calculated from this specified value.

EXAMPLES

altsetpts -xvee -4.51 -xvttga -2.07 xbar
Alter the xvee and xvttga busses of the crossbar boards.

altsetpts -vttga -2.05 sp
Alter the vttga bus of all 8 scalar processors.

altsetpts -vcc 5.05 ia8
Alter the vcc bus of the Base I/O Interface Adapter board.

FILES

All setpoint information is stored in the configuration database.

SEE ALSO

margin(1D)
powermon(1D)

BUGS

None.

NAME

backup – dump and restore files from the system disk

SYNOPSIS

/etc/backup

DESCRIPTION

Backup is a script that does a level zero dump of all partitions on the system disk (see *dump(8)*). There are no options. The order in which the partitions are backed up is the same as in /etc/fstab.

See *restore(8)* for information on how to restore a file from tape. The *backup* script may be referenced for information on the characteristics of the DAT tape unit.

FILES

None.

DIAGNOSTICS

None.

SEE ALSO

dump(8)
restore(8)

BUGS

If the order that partitions are listed in /etc/fstab changes between running *backup* and *unbackup*, files which are to be restores may not be found even though they are on the tape.

NAME

bpc - SPU octal UART bpc and priniter drivers

DESCRIPTION

The functionality of these drivers has not yet been defined.

FILES

/dev/bpc0
/dev/bpc1
/dev/bpc2
/dev/bpc3
/dev/bpc4 - SPU Bay Power controller interface drivers
/dev/prt - SPU serial printer driver

SEE ALSO

ioctl(2) perror(3)

DIAGNOSTICS

Diagnostic messages for these drivers are TBD.

NAME

bpccommd – BPC communications control daemon

SYNOPSIS

bpccommd [-t] [-d] [<baud_rate>]

DESCRIPTION

The BPC daemon, *bpccommd*, controls the routing of messages and commands from user processes to the requested bay power controllers and then routing the BPC's response back to the user process.

Upon receipt of a message from a user process, *bpccommd* reads the message from the queue and determines the destination of the message. If the message is intended for itself, *bpccommd* performs the requested action and responds to the requesting process. If the message is intended for one of the BPC's, *bpccommd* routes the message to that BPC and waits for a response. If the BPC does not respond withing a certain (TBD) length of time, *bpccommd* responds to the user process with an error message, otherwise it routes the BPC's response back to the user process.

Note that in all cases, requests to the BPC daemon are acknowledged. There are no "one way" requests from user processes. The capability does exist for the BPC firmware to route an "unsolicited" message to the *bpccommd*. When one of these is received, *bpccommd* routes the message to another daemon process, *bpcwatchd(1d)* for interpretation and resolution.

Communication between the user process and *bpccommd* is accomplished using the "System V" interprocess communication messaging mechanism. See *msgget(2)*, *msgctl(2)*, *msgop(2)*, *msgsnd(2)*, and *msgrcv(2)* for more information. *Bpccommd* is awakened by the receipt of SIGUSR1 (see *signal(5)*).

USAGE

Bpccommd is invoked simply by typing "bpccommd". It initializes itself and goes to sleep until it is instructed by a user process (typically via *bpcinit(3D)*) to open and take control of the one or more of the bays.

The **baud_rate** option permits the invoking user (typically *init(1m)*) to specify a baud rate other than 9600 baud (the default) that is to be used in communicating with the BPC's.

The **-t** option tells *bpccommd* to set its timers quite high so that messages to and from the BPC's will never timeout. This option is for debugging use only. If this option is used, *bpcwatchd(1d)* must also be started with the **-t** option.

The **-d** option tells *bpccommd* not to disassociate itself from this controlling tty, and to print any messages it generates to the screen. This option is for debugging use only.

FILES

/tmp/bpcdpid
: contains the process id if this invocation of *bpccommd*. The contents of this file tells *bpcmsg(3d)* where to send user requests for routing.

/dev/bpc[0-4]
: the device special files that correspond to each bay that is possible in the system.

DIAGNOSTICS

None.

SEE ALSO

bpcd_get_bay(3D)
bpcd_release_bay(3D)
bpcmsg(3D)
bpcinit(1D)
bpcwatchd(1D)

BUGS

None.

NAME

bpcwatchd – BPC communications monitoring daemon

SYNOPSIS

bpcwatchd [-t] [-d]

DESCRIPTION

The BPC daemon, *bpcwatchd*, monitors the overall health of the distributed power system. It sleeps until hearing from *bpccommd(1D)* that one or more of the BPC's in the system have been released from reset, establishing communications between the SPU and the BPC. At this point, *bpcwatchd* begins monitoring all message traffic forwarded from *bpccommd(1D)* for each active BPC.

Once communication has been established, *bpcwatchd* monitors the frequency of the messages it receives from *bpccommd*. If it hasn't heard from a particular BPC in a certain length of time, it declares that particular BPC dead and places it in reset, sending a message to *errlogd* in the process. In addition, *bpcwatchd* interprets each message it receives, performing any actions on the power system that it determines to be appropriate in response to the received message. In all cases, the offending message is passed to *errlogd*.

Communication between *bpccommd* and *bpcwatchd* is accomplished using the "System V" interprocess communication messaging mechanism. See *msgget(2)*, *msgctl(2)*, *msgop(2)*, *msgsnd(2)*, and *msgrcv(2)* for more information.

USAGE

Bpcwatchd is invoked simply by typing "bpcwatchd". It initializes itself and waits for *bpccommd(1D)* to initiate communications with a BPC.

The **-t** option tells *bpcwatchd* to set its timers quite high so that it will never timeout waiting for a message from *bpccommd(1d)*. This option is for debugging use only. If this option is used, *bpccommd(1d)* must also be started with the **-t** option.

The **-d** option tells *bpcwatchd* not to disassociate itself from its controlling tty, and to print any messages it generates to the screen. This option is for debugging use only.

FILES

None.

DIAGNOSTICS

None.

SEE ALSO

bpccommd(1D)
errlogd(1D)

BUGS

None.

NAME

ccu - CCU interface device driver

DESCRIPTION

The file */dev/ccu* represents a special device file that is used to send messages to CCUs. An open on /dev/ccu will cause a CCUDEV_OPEN command to be sent through the mbs message system to the CCU specified by the minor device. A close will cause a CCUDEV_CLOSE command to be send through the message system to the CCU specified by the minor device.

The open will fail if the workstation interface card is not present, the cable between the NCU and WI card is not correctly installed or if power is off in the NCU bay. Power being off in the IO bay of the target CCU is not checked.

DIAGNOSTICS

The open will return a -1 when an error occurs. **Perror(3)** can be called to print the error code found in *errno*. The following error codes are used:

[EFAULT]
Error accessing the map registers on the NCU.

[ENODEV]
Performing any operation when the NWI or NCU is not available. Trying to perform any non-existant operation on this device (read,write,select).

SEE ALSO

msg_oper(3d)
perror(3)

NAME

cdb_browser - Utility to display/modify the contents of the database

SYNOPSIS

cdb_browser

DESCRIPTION

cdb_browser is an interactive utility whereby the user can display either the current data for a given keyword or various attributes for the keyword. In addition, the user may modify the data for a given keyword.

There are five options available upon starting **cdb_browser**. Each option is described below. The default radix for displaying and/or modifying the data values is hexadecimal. An example of this main menu is shown below:

1. Mode (verbosity level)
2. Dump database
3. Item query
4. Update
5. Quit

A choice can be made by entering either the number of the item or the first letter (upper or lower case) of the choice.

Option 1 allows the user to specify what data and/or attributes are to be displayed for a given keyword. Possible selections include: name and data only, name only, decimal format, and any of the associated attributes (comments, element type, number of elements, and resource). These selections remain in effect until modified by the user or the utility is terminated.

Option 2 dumps the entire database to the screen (using the **less** pager). The user is returned to the main menu upon termination of the **less** session.

Option 3 prints the specified data and attributes for the given keywords. The user may utilize the normal wildcard characters to help specifying the keywords. The ? matches any single character and the * matches anything.

When this option is chosen, the following prompt is displayed:

Enter target string >

Upon entry of a keyword, information from the configuration database is displayed. If no match is found, the user is informed with an error message. The prompt is displayed once again for the user to input a new keyword. Entering 'q' or 'Q' kills this option and returns the user to the main menu.

Option 4 allows the user to modify the data for a given keyword. Unlike option 3, no wildcards are allowed when specifying the keyword.

When this option is chosen, the following prompt is displayed:

Enter name >

Upon entry of a keyword, the configuration database is searched for a matching keyword. If no match is found, the user is informed with an error message. If a match is found, a prompt is displayed requesting the new data. This prompt is dependent on the data type of the entry:
If the data is single character type:

Enter new character >

At this prompt the user should enter a single character.

If the data is string data type:
  Enter new character string >
At this prompt the user should enter a string (which may include white space) which will be accepted when the user hits <return>.

If the data is single integer type:
  Enter new integer >
At this prompt the user should enter a single integer.

If the data is integer array type:
  Enter number of integers you wish >
At this prompt the user should enter the number of integers that will be stored in this entry. This is the number of integers that the user will be expected to enter at the next prompt:
  Enter new integer data >
The integers should be separated by white space (end-of-lines ok).

If the data is string array type:
  Enter number of strings you wish >
At this prompt the user should enter the number of strings that will be stored in this entry. This is the number of strings that the user will be expected to enter on the lines following the series of prompts:
  Enter string [string number] :
At this prompt the user should enter a string (which may include white space) which will be accepted when the user hits <return>.

Once the new data is entered, the first prompt is displayed once again for the user to input a new keyword. Entering 'q' or 'Q' kills this option and returns the user to the main menu.

Option 5 terminates execution of the **cdb_browser** utility.

Strings are limited to 256 characters.

SEE ALSO

cdb_dump(1D)
cdb_get(1D)
cdb_update(1D)

NAME

cdb_dump - Utility to dump the contents of the database

SYNOPSIS

cdb_dump

DESCRIPTION

cdb_dump command dumps the contents of the configuration database. Every entry is printed, along with its corresponding value(s).

EXAMPLES

cdb_dump

Print all entries.

cdb_dump | more

Print all entries, viewing one page at a time.

cdb_dump | grep 300v

Print any entries with the regular expression '300v' in them.

SEE ALSO

cdb_update(1D)
cdb_browser(1D)
cdb_get(1D)

NAME

cdb_get - Utility to retrieve information for keyword(s)

SYNOPSIS

cdb_get [-q] **keyw1, ...**

DESCRIPTION

cdb_get command prints out the data, in the appropriate type, for each of the given keywords. It does not accept wildcards or do any pattern matching on the keywords. If pattern matching is desired, use option 3 of **cdb_browser** or use **cdb_dump | grep regexpr**. The default output format is 'key = key_word_value(s)'. If the **-q** option is specified, the output is just the value of the cdb key without the preceding 'key ='.

EXAMPLES

cdb_get bpc_cabled_0

Check the value of the keyword bpc_cabled for bpc 0. Output is of the form 'bpc_cabled = 0x0'.

cdb_get -q ppc_load_pwrd_00

Check the value of the keyword ppc_load_pwrd for the bpc,ppc pair 0,0. Output is of the form ' 0x1' with no preceding 'ppc_load_pwrd ='.

SEE ALSO

cdb_browser(1D)
cdb_dump(1D)
cdb_update(1D)

NAME

cdbserver – LIBRB_CDB server for the Configuration Database

SYNOPSIS

cdbserver

DESCRIPTION

The **cdbserver** utility maintains the configuration database file and modifies this file per the various LIBRB_CDB functions. When invoked, an attempt is made to restore the database from the current files (cdb.db and cdb.map). If these do not exist then the Configuration Database is empty, i.e. no entries are present.

When first started, this utility checks whether this utility has previously been started. If so then a fatal error is generated and the utility exits. If not then normal processing continues.

This utility maintains a local data structure (a queue) in memory. Each node contains all information concerning the Configuration Database entry. A hash table is also maintained that provides quick random access to these nodes.

LIBRB_CDB functions communicate with this utility via a message protocol. This message contains the following information: process id, function, message data, and returned status. The procedure is as follows:

- Process informs **cdbserver** of request
- **cdbserver** checks the queue for validity of request
- **cdbserver** sends back either an okay to continue or error response
- If okay to continue then the process updates the Configuration Database file
- Process sends information back to **cdbserver** so it can be added to the queue

cdbserver will continuously loop waiting for messages unless killed. This is done by the **killservers** utility.

FILES

/diag/db/cdb.db - contains the configuration data
/diag/db/cdb.db.chkpt - checkpoint copy of "cdb.db"
/diag/db/cdb.map - contains the local data structures
/diag/db/cdb.map.chkpt - checkpoint copy of "cdb.map"
/diag/bin/cdbserver - CDBSERVER binary

SEE ALSO

cdb_startup(1D)
rbserver(1D)
rbcdb_init(1D)

NAME

cdb_startup - start both the rbserver and cdbserver

SYNOPSIS

cdb_startup

DESCRIPTION

The **cdb_startup** daemon is started by the XSFP process. Its purpose is to monitor the **rbserver** and **cdbserver** daemons and ensure they are always running. If either should terminate then the other is terminated, **cdb_startup** terminates, and then the XSFP process restarts **cdb_startup**.

cdb_startup first determines which set of configuration database files to use when starting both servers. All of these files reside in the /diag/db directory. The following steps are used to make this determination:

(1) If all the "checkpoint" files exist then they are used.
else
(2) If all the "normal" files exist then they are used.
else
(3) If all the "standard" files exist then they are used.
else
(4) **rbcdb_init** is called with the -i option.

The "checkpoint" files consist of cdb.db.chkpt, cdb.map.chkpt, and rb_config_file.chkpt.

The "normal" files consist of cdb.db, cdb.map, and rb_config_file.

The "standard" files consist of standard.db, standard.map, and standard.rb.

If the servers are started using an existing set of database files then a check is made if either of the servers are currently executing. If they are then they are halted and restarted. First **rbserver** is started then **cdbserver** is started.

SEE ALSO

cdbserver(1D)
rbserver(1D)
rbcdb_init(1D)
xsfp(1D)

NAME

cdb_update - Utility to modify information for a keyword

SYNOPSIS

cdb_update keyw,data1,...datan

DESCRIPTION

cdb_update command writes the specified data into the given keyword, *keyw*, in the configuration database. The data must be appropriate for the the type of keyword, eg. character, integers, floats, doubles, strings, or arrays.

EXAMPLES

cdb_update bpc_cabled_0 0

Update the value of the keyword bpc_cabled for bpc 0 to a value of 0. In this case, 0 corresponds to an uncabled bay.

cdb_update ppc_load_pwrd_00 1

Update the value of the keyword ppc_load_pwrd for the bpc,ppc pair 0,0. Here a 1 means the load is powered.

SEE ALSO

cdb_dump(1D)
cdb_browser(1D)
cdb_get(1D)

NAME

config_data - Identifies some configuration database entries.

DESCRIPTION

board_volt_actval_## (## = [0..4][0..7])

This entry holds the actual values of the voltage buses for a PPC. The first # is the bay number and the second # is the slot number.

This entry returns an array of eight floats which represent the current actual value for each of the eight buses.

The resource used to reserve this entry is setpoint_data.

This entry should be updated only by margin and altsetpts. It is updated when diaginit is run, but this is accomplished by running altsetpts from diaginit. It is updated by these libpower routines: write_setpoints() (in set_points.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of board_volt_actval_##. The name of that structure array is CDB_board_volt_actval[#][#], where the first # is bay number (valid 0-4), and the second # is slot number(valid 0-7).

board_volt_killhi_#

This entry holds the actual upper limit value of the voltage buses for a PPC. The # is the boardtypeindex , which represents ccu, cdp, cmp, cu, ia, kcu, mb, sp, unused0, unused12, unused13, unused15, vp, crossbar and xiop.

This entry returns an array of eight floats which represent the highest allowable value for each of the eight buses. If voltage moves above this value, the power monitoring software shuts off the board.

The resource used to reserve this entry is setpoint_data.

This keyword's value is not updated by any libpower routine. Since it should not change (or rarely change) it is defined when the CDB is initialized, and not changed after that.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of board_volt_killhi_#. The name of that structure array is CDB_board_volt_killhi[#], where # is boardtypeindex.

board_volt_killlo_#

This entry holds the actual lower limit value of the voltage buses for a PPC. The # is the boardtypeindex, which represents ccu, cdp, cmp, cu, ia, kcu, mb, sp, unused0, unused12, unused13, unused15, vp, crossbar and xiop.

This entry returns an array of eight floats which represent the lowest allowable value for each of the eight buses. If voltage moves below this value, the power monitoring software shuts off the board.

The resource used to reserve this entry is setpoint_data.

This keyword's value is not updated by any libpower routine. Since it should not change (or rarely change) it is defined when the CDB is initialized, and not changed after that.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of board_volt_killlo_#. The name of that structure array is CDB_board_volt_killlo[#], where # is boardtypeindex.

board_volt_nomval_#

This entry holds the nominal value of the voltage buses for a PPC. The # is the

boardtypeindex, which represents ccu, cdp, cmp, cu, ia, kcu, mb, sp, unused0, unused12, unused13, unused15, vp, crossbar and xiop.

This entry returns an array of eight floats which each have valid values from approximately -6 to 6. These in dividual numbers represent the ideal nominal value for each of the eight buses. These are the values used if the buses are margined to nominal values.

The resource used to reserve this entry is setpoint_data.

This keyword's value is not updated by any libpower routine. Since it should not change (or rarely change) it is defined when the CDB is initialized, and not changed after that.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of board_volt_nomval_#. The name of that structure array is CDB_board_volt_nomval[#], where # is board typeindex.

board_volt_trimable_#

This entry holds the trimable voltage values of the voltage buses for a board. The # is the boardtypeindex , which represents ccu, cdp, cmp, cu, ia, kcu, mb, sp, unused0, unused12, unused13, unused15, vp, xbar and xiop.

Each of these entries returns an array of integers.

The resource used to reserve this entry is setpoint_data.

This keyword's value is not updated by any libpower routine. Since it should not change (or rarely change) it is defined when the CDB is initialized, and not changed after that.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of board_volt_trimable.The name of the structure array is CDB_board_volt_trimable[#] where # ranges from 0-15. Each entry returns of an array of size 8, where each element represents one of the 8 voltage buses on that respective board. If the value in the array elements equals 1 then the bus can be trimmed, if it equals 0, it cannot be trimmed.

bpc_bayid_# (# = 0..4)

This entry is used to see which bay is connected to bpc_uart number #. The bay number is returned if the bpc_uart is cabled, or a 1 is returned if no bay is connected. Valid bay numbers and valid bpc_uart numbers range from 0 to 4.

Each of these entries returns an integer.

The resource used to reserve this entry is uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update() (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of bpc_bayid_#. The name of that structure array is CDB_Bpc_bayid[#], where # identifies the bpc_uart number for the bay of interest.

bpc_cabled_# (# = 0..4)

This entry returns a value of 1 if the bpc is physically cabled to some bay or a 1 if it is not connected to any thing. Any other value (-1) indicates that the entry is uninitialized. The # is the bpc_uart number. Bpc_uart numbers are valid for 0-4.

Each of these entries returns an integer.

The resource used to reserve this entry is uart.

This entry should be updated only by diaginit and the bpcwatchd. It is updated by these libpower routines: bpcinit() (in bpcinit.c), cleanup_bpc_cdb_channel and update_bpc_cdb_channel() (in update_cdb.c), and bay_config_update (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of bpc_cabled_##. The name of that structure array is CDB_Bpc_cabled[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

bpc_fwrev_# (# = 0..4)

This entry returns an integer showing the version of firmware currently loaded in that bpc. The last byte of the integer shows the minor revision number and the second to last byte shows the major revision number. If the entry is a -1, no firmware was loaded. The # is the bpc_uart number. Bpc_uart numbers are valid for 0-4.

Each of these entries returns an integer.

The resource used to reserve this entry is uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: xpc_fw_rev() (in fw_rev.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of bpc_fwrev_#. The name of that structure array is CDB_Bpc_fwrev[#], where # is the bpc_uart number.

bpc_uartid_# (# = 0..4)

This entry is used to see which bpc_uart is connected to bay number #. The bpc_uart number is returned. Valid bay numbers and valid bpc_uart numbers range from 0 to 4.

Each of these entries returns an integer.

The resource used to reserve this entry is uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_configur_update() (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of bpc_uartid#. The name of that structure array is CDB_Bpc_uartid[#], where # identifies the bay for the bpc_uart of interest.

cpu_avail_# (# = 0..7)

This entry specifies that the CPU associated with # is available to be used by the ConvexOS or any other process. The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port in which the cpu is installed.

Each of these entries returns an integer.

The value of cpu_avail_# is an integer value which can take on the value of zero or 1. A value of zero indicates that the CPU has failed the execution of a diagnostic and the head should not be made available for the ConvexOS. A value of one indicates that the CPU has passed all diagnostics executed on the head and should be made available to ConvexOS.

Cpu_installed_# and cpu_avail_# are basically an AND operation of the sp_installed_# and the vp_installed_# entries.

Under normal circumstances, this value is initialized by diaginit to a value of one as it sees the board installed in the system. Subsequently, if either of the CPU boards fail to power up, fail scan ring integrity, or fail scan ring verification, the avail value for the CPU will be reset to zero. This value is also cleared by powerdown.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of cpu_avail_#. The name of that structure array is CDB_cpu_avail[#], where # identifies the cpuid for the cpu of interest.

cpu_installed_# (# = 0..7)

This entry specifies that the CPU associated with # is indeed installed in the machine. It does not imply any special health of the SP and VP, but only that the board pair has been seen to exist in the system. The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port in which the cpu is installed.

Each of these entries returns an integer.

The resource used to reserve this entry is sys_config.

The value of cpu_installed_# is an integer value which can take on the value of zero (not installed) or 1 (installed); all other values are invalid.

Cpu_installed_# and cpu_avail_# are basically an AND operation of the sp_installed_# and the vp_installed_# entries.

Under normal circumstances, this value is initialized by diaginit (as it sees the processor being installed) and cleared by remove_bd or powerdown (as they see the processor being removed.)

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of cpu_installed_#. The name of that structure array is CDB_cpu_installed[#], where # identifies the cpuid for the cpu of interest.

cpu_os_req_# (# = 0..7)

This entry specifies that ConvexOS has requested that the CPU associated with # be made part of the OS complex upon the next re-boot of ConvexOS. The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port in which the cpu is installed.

Each of these entries returns an integer.

The resource used to reserve this entry is sys_config.

The value of cpu_os_req_# is an integer value which can take on the value of zero or 1. A value of zero indicates that ConvexOS does not want the specified CPU to be part of the CPU complex on the next ConvexOS reboot. A value of one indicates that ConvexOS does want the specified CPU to be part of the CPU complex on the next ConvexOS reboot.

The initial configuration database defaults all processors to be part of the OS complex whether or not the CPU is even installed in the system. An X-based utility, xsys_config, is provided to allow the user to alter the values for each of the CPU's.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of cpu_os_req_#. The name of that structure array is CDB_cpu_os_req[#], where # identifies the cpuid for the cpu of interest.

cu_avail

This entry specifies that the CU is available to be used by the ConvexOS or any other process.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

The value of cu_avail is an integer value which can take on the value of zero or 1. A value of zero indicates that the CU has failed the execution of a diagnostic and the cu should not be made available for the Convex OS. A value of one indicates that the CU has passed all diagnostics executed on the cu and should be made available to ConvexOS.

Under normal circumstances, this value is initialized by diaginit to a value of one as it

sees the board installed in the system. Subsequently, if either of the CU boards fail to power up, fail scan ring integrity, or fail scan ring verification, the avail value for the CU will be reset to zero. This value is also cleared by power down.

An accessing structure has been defined and are used by the diagnostics to read and alter the value of cu_avail. The name of that structure array is CDB_cu_avail.

cu_dlock_cnt

This entry contains the deadlock count. It is the number of clock periods after a deadlock situation is detected and a harderror is generated.

This entry returns an integer.

The resource used to reserve this entry is ncu.

An accessing structure has been defined and are used by the diagnostics to read and alter the value of cu_dlock_cnt. The name of that structure is CDB_cu_dlock_cnt.

cu_installed

This entry specifies that the CU is indeed installed in the machine. It does not imply any special health of the CU, but only that the board has been seen to exist in the system.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

The value of cu_installed is an integer value which can take on the value of zero (not installed) or 1 (installed); all other values are invalid.

Under normal circumstances, this value is initialized by diaginit (as it sees the utilities board being installed) and cleared by remove_bd or powerdown (as they see the utilities board being removed.)

An accessing structure has been defined and are used by the diagnostics to read and alter the value of cu_installed. The name of that structure is CDB_cu_installed.

cu_os_req

This entry specifies that ConvexOS has requested that the CU be made part of the OS complex upon the next re-boot of ConvexOS.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

The value of cu_os_req is an integer value which can take on the value of zero or 1. A value of zero indicates that ConvexOS does not want the CU to be part of the CU complex on the next ConvexOS reboot. A value of one indicates that ConvexOS does want the specified CU to be part of the CU complex on the next ConvexOS reboot.

The initial configuration database defaults all utilities boards to be part of the OS complex whether or not the CU is even installed in the system.

A structure has been defined and are used by the diagnostics to read and alter the value of cu_os_req. The name of that structure is CDB_cu_os_req.

cu_par_en

This entry contains the CU parity enable. This boolean entry enables parity throughout the CU board. A 0 value indicates disabled, and a 1 indicates enabled.

This entry returns an integer.

The resource used to reserve this entry is ncu.

An accessing structure has been defined and are used by the diagnostics to read and alter

the value of cu_par_en. The name of that structure is CDB_cu_par_en.

cu_trap_cnt

This entry, CU trap count, indicates the number of clock periods before a trap harderror will occur. This is the number of clock cycles for the CPU to react to a trap before generating a harderror.

This entry returns an integer.

The resource used to reserve this entry is ncu.

An accessing structure has been defined and are used by the diagnostics to read and alter the value of cu_trap_cnt. The name of that structure is CDB_trap_cnt.

cu_trap_enable

This entry, CU trap enable, enables the generation of a harderror by the CU board when a trap timeout occurs.

This entry returns an integer.

The resource used to reserve this entry is ncu.

An accessing structure has been defined and are used by the diagnostics to read and alter the value of cu_trap_enable. The name of that structure is CDB_trap_enable.

dcache_enable

This boolean entry indicates whether the data cache is to operate normally or to be bypassed. A value of 1 indicates enabled, a value of 0 indicates bypass.

This entry returns an integer.

The resource used to reserve this entry is test_parms.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of dcache_enable. The name of that structure is CDB_dcache_enable.

dcache_resize

This boolean entry indicates whether the data cache is to be resized from 16K to 4K when crossing inward from ring 4. The value 0 prevents resizing, the value 1 means resize.

This entry returns an integer.

The resource used to reserve this entry is test_parms.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of dcache_resize. The name of that structure is CDB_dcache_resize.

dynamic_cpu_harderrs_# (# = 0..7)

This boolean entry indicates whether the specified CPU supports Automatic Processor Recovery (APR) or not. A value of 1 denotes the CPU/backplane pair does support APR. It is set during **diaginit** which checks the revision of the SP board and the backplane. Combinations of SPs and backplanes that support APR are: (1) all 6247 SPs at assembly revision H or greater, (2) all 8247/9247 SPs at any revision and (3) any backplane at assembly revision B or greater.

This entry returns an integer value of 0 (False) or 1 (True).

The resource used to reserve this entry is sys_config.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of dynamic_cpu_harderrs_#. The name of that structure is CDB_Dynamic_cpu_harderrs[#] where # ranges from 0 to 7.

enable_cpu_harderrs

This boolean entry indicates whether hard errors are enabled by force on all CPUs in the system. A value of 1 forces hard errors to be enabled on all CPUs when the system is booted. A value of 0 denotes that the dynamic_cpu_harderrs_# entry for each CPU is checked. A value of 1 in the dynamic_spu_harderrs_# entry disables hard errors on that CPU whereas a value of 0 enables hard errors on that CPU.

This entry returns an integer value of 0 (False) or 1 (True).

The resource used to reserve this entry is test_parms.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of enable_cpu_harderrs. The name of that structure is CDB_Enable_cpu_harderrs.

enable_cpu_softerrs

This boolean entry indicates whether soft error processing is enabled on all CPUs in the system when **sysreset** is executed. A value of 1 enables soft errors on all CPUs whereas a value of 0 disables soft errors on all CPUs.

This entry returns an integer value of 0 (False) or 1 (True).

The resource used to reserve this entry is test_parms.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of enable_cpu_softerrs. The name of that structure is CDB_Enable_cpu_softerrs.

enforce_fw_revs

This is a boolean. If set to 1, it forces the master ppc firmware revision (ppc_master_fw_rev) to be loaded by diaginit the next time it's run.

This entry returns an integer value of 0 (False) or 1 (True).

The resource used to reserve this entry is sys_config.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of enforce_fw_revs. The name of that structure is CDB_enforce_fw_revs.

ia_avail_# (# = 0..8)

This entry specifies that the IA associated with # is available to be used by the ConvexOS or any other process. The value of # for the C3800 Series can range from 0 up to, and including 8. This value specifies the port number the ia is installed into.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

The value of ia_avail_# is an integer value which can take on the value of zero or 1. A value of zero indicates that the IA has failed the execution of a diagnostic and the adapter should not be made available for the ConvexOS. A value of one indicates that the IA has passed all diagnostics executed on the adapter and should be made available to ConvexOS.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ia_avail_#. The name of that structure array is CDB_ia_avail[#], where # identifies the iaid for the ia of interest.

Under normal circumstances, this value is initialized by diaginit to a value of one as it sees the board in stalled in the system. Subsequently, if either of the IA boards fail to power up, fail scan ring integrity, or fail scan ring verification, the avail value for the IA will be reset to zero. This value is also cleared by power down.

ia_installed_# (# = 0..8)

This entry specifies that the IA associated with # is indeed installed in the machine. It does not imply any special health of the IA, but only that the board has been seen to exist in the system. The value of # for the C3800 Series can range from 0 up to, and including 8. This value specifies the port number the ia is installed into.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

The value of ia_installed_# is an integer value which can take on the value of zero (not installed) or 1 (installed); all other values are invalid.

Under normal circumstances, this value is initialized by diaginit (as it sees the board being installed) and rewritten by remove_bd or powerdown (as they see the processor being removed.)

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ia_installed_#. The name of that structure array is CDB_ia_installed[#], where # identifies the port number for the ia of interest.

ia_os_req_# (# = 0..8)

This entry specifies that ConvexOS has requested that the IA associated with # be made part of the OS complex upon the next re-boot of ConvexOS. The value of # for the C3800 Series can range from 0 through 8. This value specifies the port number the ia is installed into.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

The value of ia_os_req_# is an integer value which can take on the value of zero or 1. A value of zero indicates that ConvexOS does not want the specified IA to be part of the IA complex on the next ConvexOS re boot. A value of one indicates that ConvexOS does want the specified IA to be part of the IA complex on the next ConvexOS reboot.

The initial configuration database defaults all interface adapters to be part of the OS complex whether or not the IA is even installed in the system.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ia_os_req_#. The name of that structure array is CDB_ia_os_req[#], where # identifies the iaid for the ia of interest.

ia_par_en_p_# (#=0..8)

This boolean entry represents the parity enable on the IA boards for the ports indicated by #.

These entries return an integer.

The resource used to reserve this entry is nia_#.

A value of 0 indicates parity disabled, 1 indicates parity enabled.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ia_par_en_p_#. The name of that structure array is CDB_ia_par_en[#], where # is 0 through 8.

mb_avail_# (# = 0..7)

This entry specifies that the MB associated with # is available to be used by the ConvexOS or any other process. The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port number the mb is installed into.

Each of these entries returns an integer.

The resource used to reserve each of these entries is sys_info.

The value of mb_avail_# is an integer value which can take on the value of zero or 1. A value of zero indicates that the MB has failed the execution of a diagnostic and the board should not be made available for the ConvexOS. A value of one indicates that the MB has passed all diagnostics executed on the board and should be made available to ConvexOS.

Under normal circumstances, this value is initialized by diaginit to a value of one as it sees the board installed in the system. Subsequently, if either of the MB boards fail to power up, fail scan ring integrity, or fail scan ring verification, the avail value for the MB will be reset to zero. This value is also cleared by power down.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of mb_avail_#. The name of that structure array is CDB_mb_avail[#], where # identifies the mbid for the mb of interest.

mb_dram_rows_p_# (# = 0..7)

This entry indicates the number of rows of DRAMs on each of the NMCs used on a given memory board (0..7). It is initialized by mminit.

This entry returns an integer.

The resource used to reserve each of these entries is nmb_#.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of mb_dram_rows_p_#. The name of that structure array is CDB_mb_dram_rows_p[#] where # represents a particular memory board.

mb_dram_size_p_# (# = 0..7)

This entry indicates the size of the DRAMs on each of the NMCs used on a given memory board (0..7). It is initialized by mminit.

This entry returns an integer.

The resource used to reserve each of these entries is nmb_#.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of mb_dram_size_p_#. The name of that structure array is CDB_mb_dram_size_p[#] where # represents a particular memory board.

mb_ecc_chk_en_p_# (# = 0..7)

This boolean entry indicates whether the memory board error correction code is enabled for the memory board indicated by port #. As usual, a value of 0 indicates false, a value of 1 indicates true (enabled).

This entry returns an integer.

The resource used to reserve each of these entries is nmb_#.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of mb_ecc_chk_en_p_#. The name of that structure array is CDB_mb_ecc_en[#] where # represents a particular memory board.

mb_installed_# (# = 0..7)

This entry specifies that the MB associated with # is indeed installed in the machine. It does not imply any special health of the MB, but only that the board has been seen to exist in the system. The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port number the mb is installed into.

Each of these entries returns an integer.

The resource used to reserve each of these entries is sys_config.

The value of mb_installed_# is an integer value which can take on the value of zero (not installed) or 1 (installed); all other values are invalid.

Under normal circumstances, this value is initialized by diaginit (as it sees the memory board being installed) and cleared by remove_bd or powerdown (as they see the memory board being removed.)

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of mb_installed_#. The name of that structure array is CDB_mb_installed[#], where # identifies the mbid for the mb of interest.

mb_os_req_# (# = 0..7)

This entry specifies that ConvexOS has requested that the MB associated with # be made part of the OS complex upon the next re-boot of ConvexOS. It is also used by the mminit utility to determine which MBs to initialize. The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port number the mb is installed into.

Each of these entries returns an integer.

The resource used to reserve each of these entries is sys_config.

The value of mb_os_req_# is an integer value which can take on the value of zero or 1. A value of zero indicates that ConvexOS does not want the specified MB to be part of the MB complex on the next ConvexOS reboot. A value of one indicates that ConvexOS does want the specified MB to be part of the MB complex on the next ConvexOS reboot.

The initial configuration database defaults all memory boards to be part of the OS complex whether or not the MB is even installed in the system. An X-based utilty, xsys_config, is provided to allow the user to alter the values for each of the MB's.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of mb_os_req_#. The name of that structure array is CDB_mb_os_req[#], where # identifies the mbid for the mb of interest.

mb_par_chk_en_p_# (# = 0..7)

This boolean entry indicates whether the memory board parity checking is enabled for the memory board in dicated by port #. As usual, a value of 0 indicates false, a value of 1 indicates true (enabled).

Each of these entries returns an integer.

The resource used to reserve each of these entries is nmb_#.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of mb_par_chk_en_p_#. The name of that structure array is CDB_par_en[#] where # represents a particular memory board.

mb_size_# (# = 0..7)

This entry forces the size of each memory board (0..7). It allows mminit to cause memory boards to appear smaller than they actually are.

Each of these entries returns an integer.

The resource used to reserve each of these entries is sys_config.

This entry is used to artificially configure machines to look like they have different size memory boards. This is used for running benchmarks without physically swapping memory boards. For example, if a machine has a full complement of memory, it can be artificially "downsized" to match any given memory configuration.

A value of zero for this entry indicates to mminit that it should configure the machine

with whatever amount of physical memory is available for that particular board. A value of 128 means configure that board to 128 meg, a value of 256 means configure to 256 meg, and 512 means configure to 512 meg. All this assumes that the required memory is physically available.

The user can modify the settings for this entry using the utility xsys_config.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of mb_size#. The name of that structure array is CDB_mb_size[#] where # represents a particular memory board.

mem_installed

This entry contains the amount of memory that is installed in the machine.

This entry returns an integer.

The resource used to reserve this entry is sys_info.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of mem_installed. The name of that structure is CDB_mem_installed.

osc_freq

This entry represents the system clock speed. It is affected by margining. A value of 0 indicates nominal speed, 1 indicates lower speed, 2 indicates upper.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of os_freq. The name of that structure is CDB_osc_freq.

This entry is changed by the libpower routines when margin is used.

pcm

This entry represents the physical configuration map.

This entry returns an array of "pcm_size" characters that are used as integers. Each element in the array has a value of 0 or 1 to represent the presence of a 2 meg block of memory in the system at that location.

The resource used to reserve this entry is sys_info.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of pcm. The name of that structure is CDB_pcm.

pcm_avail

This entry is similar to the pcm entry, i.e. an array of "pcm_size" characters, each with a value of 0 or 1. In this case a value of 0 means that that 2 meg block of memory is unavailable to the user (even if it is installed), and a value of 1 means that the block is available.

The resource used to reserve this entry is sys_info.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of pcm_avail. The name of that structure is CDB_pcm_avail.

pn_count

This entry indicates how many part numbers are recognized in the system. One board may have multiple part numbers for different revisions.

This entry returns an integer.

The resource used to reserve this entry is pn_data.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of pn_count. The name of that structure is CDB_PN_count.

pn_data_# (# = 0..pn_count)

This entry describes the part number and board type.

This entry returns a string.

The resource used to reserve each of these entries is pn_data.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of pn_data_#. The name of that structure is CDB_PN_data[#] where # represents pn_count.

port_ccu_bkpln_type_# (# = 0..9)

This entry returns an integer showing the type of backplane installed in the port. The integers are defined in "backplane_types.h". If the entry is a -1, no backplane is installed, since this means an invalid backplane type. The # is the port number which is valid for values 0-9.

Each of these entries returns an integer.

The resource used to reserve each of these entries is port_copdata_#.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update().

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of port_ccu_bkpln_type_#. The name of that structure array is CDB_Port_ccu_bkpln_type[#]. The # is the port number.

port_main_bkpln_type_# (# = 0..9)

This entry returns an integer showing the type of backplane installed in the port (a port is the upper or lower half of a bay). This entry is part of the power system. Currently there are left and right CPU, base I/O, and Crossbar backplanes. The integers are defined in "backplane_types.h". If the entry is a -1, no backplane is installed, since this means an invalid backplane type. The # is the port number which is valid for values 0-9.

These entries return integers.

The resource used to reserve each of these entries is port_copdata_#.

The resource used to reserve this entry is uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update()

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of port_main_bkpln_type_#. The name of that structure array is CDB_Port_main_bkpln_type[#]. The # is the port number.

ppc_bkpln_type_## (## = [0..4][0..7])

This entry returns an integer showing the type of backplane to which the ppc is connected. The integers are defined in "backplane_types.h". If the entry is a -1, the ppc is not be connected to anything, since this means an invalid backplane type. The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update().

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_bkpln_type_##. The name of that structure array is CDB_Ppc_bkpln_type[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

ppc_board_type_## (## = [0..4][0..7])

This entry returns an integer showing the type of board to which the ppc is connected. These numbers are defined in "board_types.h". The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Note: the board type for a ccu is not the true boardtype, but rather a set of flags showing whether the individual ccus are installed and if they require ECL voltages. To see if a board is truly a ccu, you must also look at the pallet type. This should be done anytime you need to look know what type of board is installed, since the ccu board types may look like some other type of board until you look at the pallet type.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update() (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_board_type_##. The name of that structure array is CDB_Ppc_board_type[#][#], where the first # identifies the bpc_uart number and the second # the ppc_uart number.

ppc_bpsid_## (## = [0..4][0..7])

This entry returns the number of the bay power supply to which the ppc is connected, or -1 if it is not connected to any bps. Note that some boards share a bps, so there may be duplicate return values for these keys. For example, the sp and vp boards currently share a bps, so the ppc_bpsid_# key for each will return the same value. The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update() (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_bpsid_##. The name of that structure array is CDB_Ppc_bpsid[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

ppc_cabled_## (## = [0..4][0..7])

This entry returns a 0 if the ppc is physically cabled to some board or a -1 if it is not connected to anything. The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit and the bpcwatchd. It is updated by these libpower routines: cleanup_bpc_cdb_channel() (in update_cdb.c) and bay_config_update (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_cabled_##. The name of that structure array is CDB_Ppc_cabled[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

ppc_fwload_## (## = [0..4][0..7])

This entry returns an integer with value 0 or -1, showing whether the firmware for a ppc needs to be downloaded. A value of 0 indicates that downloading is not required, otherwise it is required. The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: power() (in power.c), cleanup_bpc_cdb_channel() (in update_cdb.c), bay_config_update (in config_update.c), and xpc_fw_dload() (in download.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_fwload_##. The name of that structure array is CDB_Ppc_fwload[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

ppc_fwrev_## (## = [0..4][0..7])

This entry returns an integer showing the version of firmware currently loaded in that ppc. The last byte of the integer shows the minor revision number and the second to last byte shows the major revision number. If the entry is a -1, no firmware was loaded. The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update (in config_update.c), and xpc_fw_rev() (in fw_rev.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_fwrev_##. The name of that structure array is CDB_Ppc_fwrev[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

ppc_inited_## (## = [0..4][0..7])

This entry returns an integer that indicates this PPC is alive and communicating with its BPC. A return value of 1 means it is alive and a value of 0 means it is not. The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which

uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: config_update() (in config_update.c), power() (in power.c), and cleanup_bpc_cdb_chan() (in update_cdb.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_inited_##. The name of that structure array is CDB_Ppc_inited[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

ppc_load_pwrd_## (## = [0..4][0..7])

This entry returns an integer showing whether the board the ppc is controlling has power to its voltage buses. The integer is a 1 if the buses are on and 0 if they are off. The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This routine is used by diaginit, powerup, powerdown, and the bpcwatchd. It is updated by these libpower routines: power() (in power.c) and cleanup_bpc_cdb_channel() (in update_cdb.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_load_pwrd_##. The name of that structure array is CDB_Ppc_load_pwrd[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

ppc_pallet_type_## (## = [0..4][0..7])

This entry returns an integer showing the type of pallet to which the ppc is connected. These numbers are defined in "pallet_types.h". The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update() (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_pallet_type_##. The name of that structure array is CDB_Ppc_pallet_type[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

ppc_portid_## (## = [0..4][0..7])

This entry returns the port number for a given ppc_uart. The first # is the bpc_uart number and the second is the ppc_uart number. Valid port numbers range from 0 to 9, where a port is usually half a bay. The left half of bay 0 is port0, and the right half is port 1. The left half of bay 1 is port2 and the right half is port3. Ports 4-7 are computed similarly but port8 is all of bay4, and port9 is the crossbar.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update() (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_portid_##. The name of that structure array is CDB_Ppc_portid[#][#], where the first # identifies the bpc_uart number and the second # the ppc_uart number.

ppc_slot_## (## = [0..4][0..7])

This entry returns the slot number of the given ppc_uart. The slot numbers range from 0 to 7, and correspond to the location in the bay to which the ppc is connected. For this numbering scheme, all the ccus in one port (which are also all the ccus that share a backplane) and their ppc board are counted as one slot, so the first five physical slots count as one slot. For instance, this means slot 0 in each bay actually corresponds to all the ccus in the left half of that bay. This is done because the group of four ccus share the same ppc, so these four ccus can be treated as one as far as the power system is concerned. A return value of -1 means that ppc_uart is not connected to any slot. The first # is the bpc_uart number (valid values:0-4) and the second # is the ppc_uart number (valid values:0-7).

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update() (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_slot_##. The name of that structure array is CDB_Ppc_slot[#][#], where the first # identifies the bpc_uart number and the second # the ppc_uart number.

ppc_300v_## (## = [0..4][0..7])

This entry returns an integer showing whether the ppc has 300v applied to it. The integer is a 1 if 300v is applied and 0 if it is not. This 300v entry will be off if interlock is lost, if a command has been issued through the bpc to remove the 300v from the board (or a command was never sent to apply the 300v at startup), or if the bps is physically switched off. The first # is the bpc_uart number and the second # is the ppc_uart number. Bpc_uart numbers are valid for 0-4 and ppc_uarts 0-7.

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This routine is used by diaginit, powerup, powerdown, and the bpcwatchd. It is updated by these libpower routines: power() (in power.c), bay_power_update() (in config_update.c), and cleanup_bpc_cdb_channel() (in update_cdb.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_300v_##. The name of that structure array is CDB_Ppc_300v[#][#]. The first # is the bpc_uart number and the second # is the ppc_uart number.

ppc_uart_## (## = [0..4][0..7])

This entry returns the ppc_uart number of the given slot. The slot numbers range from 0 to 7, and correspond to the location in the bay to which the ppc is connected. For this numbering scheme, all the ccus in one port (which are also all the ccus that share a

backplane) and their ppc board are counted as one slot, so the first five physical slots count as one slot. For instance, this means slot 0 in each bay actually corresponds to all the ccus in the left half of that bay. This is done because the group of four ccus share the same ppc, so these four ccus can be treated as one as far as the power system is concerned. The first # is the bpc_uart number (valid values:0-4) and the second # is the slot number (valid values:0-7).

Each of these entries returns an integer.

The resource used to reserve each of these entries is uart_#, where # indicates which uart.

This entry should be updated only by diaginit. It is updated by these libpower routines: bay_config_update() (in config_update.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of ppc_uart##. The name of that structure array is CDB_Ppc_uart[#][#], where the first # identifies the bpc_uart number and the second # the ppc_uart number.

rm_bits_base_addr

This entry contains the physical address of reference and modified bits for a 2 meg block (pcm_blksize) of memory.

This entry returns an integer.

The resource used to reserve this entry is sys_info.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of rm_bits_base_addr. The name of that structure is CDB_rm_bits_base_addr.

scn_ovr_enable

This entry is a boolean indicator of whether or not the "scan_over" script (in /diag/db) will be run when booting. 0 represents false, 1 represents true.

This entry returns an integer.

The resource used to reserve this entry is test_parms.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of scn_ovr_enable. The name of that structure is CDB_scn_ovr_enable.

secure_mode

This boolean entry indicates whether the system is running in secure mode. When in secure mode, the system will not accept any connections from the outside world.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of secure_mode. The name of that structure is CDB_secure_mode.

serial_number

This the system serial number, it is initialized to 1.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of serial_number. The name of that structure is CDB_secure_mode.

site_name

This entry is the system site name, it is initialized to "triskelion."

This entry returns a string.

The resource used to reserve this entry is sys_info.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of site_ name. The name of that structure is CDB_site_name.

sp_avail_# (# = 0..7)

This entry specifies that the SP associated with # is available to be used by the ConvexOS or any other process. The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port number the sp is installed into.

Each of these entries returns an integer.

The resource used to reserve each of these entries is sys_config.

The value of sp_avail_# is an integer value which can take on the value of zero or 1. A value of zero indicates that the SP has failed the execution of a diagnostic and the head should not be made available for the ConvexOS. A value of one indicates that the SP has passed all diagnostics executed on the head and should be made available to ConvexOS.

Under normal circumstances, this value is initialized by diaginit to a value of one as it sees the board installed in the system. Subsequently, if the SP board fails to power up, fails scan ring integrity, or fails scan ring verification, the avail value for the SP will be reset to zero.This value is also cleared by powerdown.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of sp_avail_#. The name of that structure array is CDB_sp_avail[#], where # identifies the spid for the sp of interest.

sp_installed_# (# = 0..7)

This entry specifies that the SP associated with # is indeed installed in the machine. It does not imply any special health of the SP, but only that the board has been seen to exist in the system.

The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port number the SP is installed into.

Each of these entries returns an integer.

The resource used to reserve each of these entries is sys_config.

The value of sp_installed_# is an integer value which can take on the value of zero (not installed) or 1 (installed); all other values are invalid.

Under normal circumstances, this value is initialized by diaginit (as it sees the processor being installed) and cleared by remove_bd or powerdown (as they see the processor being removed.)

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of cpu_installed_#. The name of that structure array is CDB_sp_installed[#], where # identifies the spid for the SP of interest.

sp_sr_rev_# (# = 0..7)

This entry contains the sr microcode revision for the scalar processor indicated by #.

Each of these entries returns a string.

The resource used to reserve each of these entries is nsp_#.

An accessing structure has been defined and is used by the diagnostics to read and alter

the value of sp_sr_rev_#. The name of that structure is CDB_nsp_sr_rev[#] where # represents a cpuid.

sp_us_rev_# (# = 0..7)

This entry contains the us microcode revision for the scalar processor indicated by #.

This entry returns a string.

The resource used to reserve each of these entries is nsp_#.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of sp_us_rev_#. The name of that structure is CDB_nsp_us_rev[#] where # represents a cpuid.

uart_inited_# (# = 0..4)

This entry returns an integer that indicates this uart channel is alive and communicating with the communications demon. A return value of 1 means it is alive and a value of 0 means it is not. The # is the bpc_uart number which is valid for 0-4.

Each of these entries returns an integer.

The resource used to reserve each of these these entries is uart_#.

This entry should be updated only by diaginit and the bpcwatchd. It is updated by these libpower routines: bpcoffl() (in bpcoffl.c) and bpcinit() (in bpcinit.c).

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of uart_inited_#. The name of that structure array is CDB_Uart_inited[#], where # is the bpc_uart number.

under_mask

This boolean entry contains whether the system is running "under mask" i.e. are we using masks for instructions.

This entry returns an integer.

The resource used to reserve this entry is sys_config.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of start_date. The name of that structure is CDB_under_mask.

vp_avail_# (# = 0..7)

This entry specifies that the VP associated with # is available to be used by the ConvexOS or any other process. The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port number the vp is installed into.

Each of these entries returns an integer.

The resource used to reserve this entry is sys_config.

The value of vp_avail_# is an integer value which can take on the value of zero or 1. A value of zero indicates that the VP has failed the execution of a diagnostic and the head should not be made available for the ConvexOS. A value of one indicates that the VP has passed all diagnostics executed on the head and should be made available to ConvexOS.

Under normal circumstances, this value is initialized by diaginit to a value of one as it sees the board installed in the system. Subsequently, if the VP board fails to power up, fails scan ring integrity, or fails scan ring verification, the avail value for the VP will be reset to zero. This value is also cleared by powerdown.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of vp_avail_#. The name of that structure array is CDB_vp_avail[#], where # identifies the vpid for the vp of interest.

vp_installed_# (# = 0..7)

This entry specifies that the VP associated with # is indeed installed in the machine. It does not imply any special health of the VP, but only that the board has been seen to exist in the system. The value of # for the C3800 Series can range from 0 up to, and including 7. This value specifies the port number the VP is installed into.

Each of these entries returns an integer.

The resource used to reserve this entry is sys_config.

The value of vp_installed_# is an integer value which can take on the value of zero (not installed) or 1 (installed); all other values are invalid.

Under normal circumstances, this value is initialized by diaginit (as it sees the processor being installed) and cleared by remove_bd or powerdown (as they see the processor being removed.)

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of vp_installed_#. The name of that structure array is CDB_vp_installed[#], where # identifies the vpid for the VP of interest.

vp_ua_rev_# (# = 0..7)

This entry contains the ua microcode revision for the vector processor indicated by #.

Each of these entries returns a string.

The resource used to reserve this entry is nvp_#.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of vp_ua_rev_#. The name of that structure is CDB_nvp_ua_rev[#] where # represents a cpuid.

vp_ul_rev_# (# = 0..7)

This entry contains the ul microcode revision for the vector processor indicated by #.

Each of these entries returns a string.

The resource used to reserve this entry is nvp_#.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of vp_ul_rev_#. The name of that structure is CDB_nvp_ul_rev[#] where # represents a cpuid.

vp_um_rev_# (# = 0..7)

This entry contains the um microcode revision for the vector processor indicated by #.

Each of these entries returns a string.

The resource used to reserve this entry is nvp_#.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of vp_um_rev_#. The name of that structure is CDB_nvp_um_rev[#] where # represents a cpuid.

vp_vd_rev_# (# = 0..7)

This entry contains the vd microcode revision for the vector processor indicated by #.

Each of these entries returns a string.

The resource used to reserve this entry is nvp_#.

An array of accessing structures has been defined and is used by the diagnostics to read and alter the value of vp_vd_rev_#. The name of that structure is CDB_nvp_vd_rev[#] where # represents a cpuid.

xbar_avail_# (# = 0..6)

This entry specifies that the crossbar board associated with # is available to be used by the ConvexOS or any other process. The value of # for the C3800 Series can range from 0 up to, and including 6. This value specifies the port number the crossbar is installed into.

Each of these entries returns an integer.

The resource used to reserve each of these entries is sys_config.

The value of xbar_avail_# is an integer value which can take on the value of zero or 1. A value of zero indicates that the crossbar board has failed the execution of a diagnostic and the crossbar should not be made available for the ConvexOS. A value of one indicates that the crossbar board has passed all diagnostics executed on the crossbar and should be made available to ConvexOS.

Under normal circumstances, this value is initialized by diaginit to a value of one as it sees the board installed in the system. Subsequently, if either of the crossbar board boards fail to power up, fail scan ring integrity, or fail scan ring verification, the avail value for the crossbar board will be reset to zero.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of xbar_avail_#. The name of that structure array is CDB_xbar_avail[#], where # identifies the portid for the crossbar of interest. This value is also cleared by powerdown.

xbar_burst_cnt

This entry represents the number of consecutive requests that the Crossbar allows a processor to make to a single memory board before allowing a different processor's request to access that board.

This entry returns an integer.

The resource used to reserve this entry is xs0.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of xbar_burst_cnt. The name of that structure array is CDB_xbar_burst_cnt.

xbar_cu_accel_en

This boolean entry allows the acceleration of data returning from the CU board.

This entry returns an integer.

The resource used to reserve this entry is xrt.

A value of 0 indicates that acceleration is not enabled, 1 indicates enabled.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of xbar_cr_rd_time. The name of that structure array is CDB_xbar_cr_rd_time.

xbar_cu_port_disable

This boolean entry allows the CU board to be disabled.

This entry returns an integer.

The resource used to reserve this entry is xrt.

A value of 0 indicates that the CU is not disabled, 1 indicates that it is disabled.

An accessing structure has been defined and is used by the diagnostics to read and alter the value of xbar_cu_port_disable. The name of that structure array is CDB_xbar_cu_port_disable.

xbar_installed_# (# = 0..6)

This entry specifies that the crossbar board associated with # is indeed installed in the machine. It does not imply any special health of the Crossbar, but only that the board has been seen to exist in the system. The value of # for the C3800 Series can range from 0 up to, and including 6. This value specifies the port number the crossbar is installed into.

These entries return integers.

The resource used to reserve these entries is sys_config.

The value of xbar_installed_# is an integer value which can take on the value of zero (not installed) or 1 (installed); all other values are invalid.

Under normal circumstances, this value is initialized by diaginit (as it sees the crossbar board being installed) and cleared by remove_bd or powerdown (as they see the crossbar board being removed.)

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of xbar_installed_#. The name of that structure array is CDB_xbar_installed[#], where # identifies the portid for the crossbar of interest.

xbar_os_req_# (# = 0..6)

This entry specifies that ConvexOS has requested that the crossbar board associated with # be made part of the OS complex upon the next re-boot of ConvexOS. The value of # for the C3800 Series can range from 0 up to, and including 6. This value specifies the port number the crossbar is installed into.

These entries return integers.

The resource used to reserve these entries is sys_config.

The value of xbar_os_req_# is an integer value which can take on the value of zero or 1. A value of zero indicates that ConvexOS does not want the specified crossbar board to be part of the crossbar board complex on the next ConvexOS reboot. A value of one indicates that ConvexOS does want the specified crossbar board to be part of the crossbar board complex on the next ConvexOS reboot.

The initial configuration database defaults all crossbar boards to be part of the OS complex whether or not the crossbar board is even installed in the system.

An array of accessing structures has been defined and are used by the diagnostics to read and alter the value of xbar_os_req_#. The name of that structure array is CDB_xbar_os_req[#], where # identifies the portid for the crossbar of interest.

FILES

None.

NAME

CONVEXOS_CONSOLE – the xterm window that ConvexOS is booted from.

SYNOPSIS

CONVEXOS_CONSOLE

DESCRIPTION

This program is started by **xsfp**. If it is killed, **xsfp** will restart it. Note that if the ConvexOS is booted and this window is killed, you will loose contact with the console driver (and ConvexOS) until the Convex is rebooted. The icon name and title for this process are set to **CONVEXOS CONSOLE**, and its icon is the distinctive Convex corporate logo. Beyond these superficial differences, there are some internal modifications that were added to provide functionality required for the SECURE position of the keyswitch, remote logins, and printer support.

So that there will not be confusion as to which xterm ConvexOS is booted from, typing boot in any window with a standard search path will cause the boot command to actually be executed in the CONVEXOS CONSOLE window. This is a feature that is rather easy to subvert - but if ConvexOS is booted from any other xterm, then then all of the really neat stuff for the remote logins, console printing, and SECURE mode will not work correctly.

SECURE mode

When the keyswitch is placed in the SECURE position, the access to the SPU workstation is restricted. Two things happen, logins are disabled, and mouse and keyboard input is restricted to the CONVEXOS CONSOLE window.

remote logins

If the keyswitch is in REMOTE position, the user rmtdiag is allowed to log in. When he does, the xterm or terminal that he is using becomes the CONVEXOS CONSOLE. The CONVEXOS CONSOLE window displayed on the SPU workstation screen resizes to be the same size as the remote screen. From then on all characters are output to both displays. Character typed into the CONVEXOS CONSOLE window from the workstation keyboard are ignored. Characters typed at the remote terminal are acted on by the CONVEXOS CONSOLE.

printer support

This feature provides printing of all characters as they are displayed on the CONVEXOS CONSOLE display. It is turned on by clicking the printer toggle on the 'xsfp' window, or by issuing the command 'sfp -p 1'.

SEE ALSO

xsfp(1)
xterm(1)

NAME

cop - display board/backplane identification information as extracted from the configuration database.

SYNOPSIS

cop [[all] [bay#] [xbar] [port#] [bpln name] [board name] [ccu#] [bpc#] [ppname] [ppcname]]

DESCRIPTION

The utility **cop** provides the ability to display the contents of the board/backplane cop chips in a pre-defined format to the user. The options provided by the user at the time **cop** is invoked determine exactly how much of the machine's cop information will be displayed.

NOTE The utility **cop** no longer directly accesses the cop chips for the cop information. Instead, it extracts the information from the configuration database which was updated with the cop chip information the last time the machine was powered up.

The following is an explanation of what are valid options for cop and exactly how much cop information is displayed for each option:

? This option is provided to allow the user the ability to query the utility to display the command syntax.

all When this option is supplied, the cop information for every cop chip in the system which has been powered on, is displayed to the user. This option is the default option when the user provides no other options.

bay# Valid bay options are **bay0**, bay1, bay2, bay3, and bay4. When one of the valid bay options are decoded, every cop chip in the specified bay which has been powered on, is displayed to the user.

xbar When this option is provided, the cop information for the entire cross bar is displayed.

port# When this option is provided, the cop information associated with the specified port is displayed. The valid ranges of ports are 0 upto and including 8.

bpc# Indicates to display the cop information for the bay power controller installed into bay #, where the valid range for # is 0 through 4. If the bay specifier is not supplied, then cop information for each of the bay power controllers in the system will be displayed.

lt_bpln#
Indicates to display the cop information for the left half backplane installed into bay #, where the valid range for # is 0 through 3. If the bay specifier is not supplied, then cop information for each of the left half backplanes in the system will be displayed.

rt_bpln#
Indicates to display the cop information for the right half backplane installed into bay #, where the valid range for # is 0 through 3. If the bay specifier is not supplied, then cop information for each of the right half backplanes in the system will be displayed.

fl_bpln#
Indicates to display the cop information for the full backplane installed into bay #, where the valid range for # is 0 through 3. If the bay specifier is not supplied, then cop information for each of the full backplanes in the system will be displayed. Note: there are currently no full backplanes for bays 0 through 3.

xbar_bpln
Indicates to display the cop information for the xbar backplane.

baseio_bpln
Indicates to display the cop information for the baseio bay backplane. This is the backplane installed in bay4 and it only has one cop chip.

ccu# Indicates to display the cop information for the specified CCU. Valid CCU options range from ccu0 to ccu39, where the # is a decimal number. If no CCU # is specified, cop information for each possible CCU will be displayed.

mb_assy#
Indicates to display the cop information for the Memory Board (MB) assembly installed into port #. This would include board, power pallet, and power pallet controller cop data. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all eight possible assemblies will be displayed.

mb# Indicates to display the cop information for the Memory Board (MB) installed into port #. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all eight possible MB's will be displayed.

mb_pp#
Indicates to display the cop information for the Memory Board (MB) power pallet installed into port #. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all eight of the possible MB power pallets will be displayed.

mb_ppc#
Indicates to display the cop information for the Memory Board (MB) power pallet controller installed into port #. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all nine of the possible MB power pallet controllers will be displayed.

sp_assy#
Indicates to display the cop information for the Scalar Processor (SP) board assembly installed into port #. This would include board, power pallet, and power pallet controller cop data. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all eight possible assemblies will be displayed.

sp# Indicates to display the cop information for the Scalar Processor (SP) board installed into port #. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all eight possible SP logic boards will be displayed.

sp_pp#
Indicates to display the cop information for the Scalar Processor (SP) board power pallet installed into port #. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all eight of the possible SP power pallets will be displayed.

sp_ppc#
Indicates to display the cop information for the Scalar Processor (SP) board power pallet controller installed into port #. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all nine of the possible SP power pallet controllers will be displayed.

vp_assy#
Indicates to display the cop information for the Vector Processor (VP) board assembly installed into port #. This would include board, power pallet, and power pallet controller cop data. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all eight possible assemblies will be

displayed.

vp# Indicates to display the cop information for the Vector Processor (VP) installed into port #. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all eight possible VP boards will be displayed.

vp_pp#
Indicates to display the cop information for the Vector Processor (VP) power pallet installed into port #. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all eight of the possible VP power pallets will be displayed.

vp_ppc#
Indicates to display the cop information for the Vector Processor (VP) power pallet controller installed into port #. For this option, # can take on a value from 0 through 7. If the port identifier is not specified, then the cop information for all nine of the possible VP power pallet controllers will be displayed.

ia_assy#
Indicates to display the cop information for the Interface Adapter (IA) board assembly installed into port #. This would include board, power pallet, and power pallet controller cop data. For this option, # can take on a value from 0 through 8. If the port identifier is not specified, then the cop information for all nine possible assemblies will be displayed.

ia# Indicates to display the cop information for the Interface Adapter (IA) installed into port #. For this option, # can take on a value from 0 through 8. If the port identifier is not specified, then the cop information for all nine possible IA boards will be displayed.

ia_pp#
Indicates to display the cop information for the Interface Adapter (IA) power pallet installed into port #. For this option, # can take on a value from 0 through 8. If the port identifier is not specified, then the cop information for all nine of the possible IA power pallets will be displayed.

ia_ppc#
Indicates to display the cop information for the Interface Adapter (IA) power pallet controller installed into port #. For this option, # can take on a value from 0 through 8. If the port identifier is not specified, then the cop information for all nine of the possible IA power pallet controllers will be displayed.

xp_assy#
Indicates to display the cop information for the XIOP board assembly installed into port #. This would include board, power pallet, and power pallet controller cop data. For this option, # can take on a value from 0 through 8. If the port identifier is not specified, then the cop information for all nine possible assemblies will be displayed.

xp# Indicates to display the cop information for the XIOP installed into port #. For this option, # can take on a value from 0 through 8. If the port identifier is not specified, then the cop information for all nine possible XIOP boards will be displayed.

xp_pp#
Indicates to display the cop information for the XIOP power pallet installed into port #. For this option, # can take on a value from 0 through 8. If the port identifier is not specified, then the cop information for all nine of the possible XIOP power pallets will be displayed.

xp_ppc#
Indicates to display the cop information for the XIOP power pallet controller installed into port #. For this option, # can take on a value from 0 through 8. If the port

identifier is not specified, then the cop information for all nine of the possible XIOP power pallet controllers will be displayed.

cu_assy
Indicates to display the cop information for the CPU Utilities (CU) board assembly. This would include board, power pallet, and power pallet controller cop data.

cu Indicates to display the cop information for the CPU Utilities (CU) installed in the base IO bay.

cu_pp Indicates to display the cop information for the CPU Utilities (CU) power pallet installed into the base IO bay.

cu_ppc
Indicates to display the cop information for the CPU Utilities (CU) power pallet controller installed into the base IO bay.

ccubpl#
Indicates to provide the cop information for the LEFT CCU backplane installed into bay #, where the valid bay numbers for the RIGHT CCU backplanes are 0 through 4. If the bay identifier is not provided, cop information about all 5 possible LEFT CCU backplanes will be provided.

ccubpr#
Indicates to provide the cop information for the RIGHT CCU backplane installed into bay #, where the valid bay numbers for the RIGHT CCU backplanes are 0 through 4. If the bay identifier is not provided, cop information about all 5 possible RIGHT CCU backplanes will be provided.

ccu_ppl#
Indicates to provide the cop information for the LEFT CCU backplane power pallet installed into bay #, where the valid bay numbers for the LEFT CCU backplanes are 0 through 4. If the bay identifier is not provided, cop information about all 5 possible LEFT CCU backplane power pallets will be provided.

ccu_ppr#
Indicates to provide the cop information for the RIGHT CCU backplane power pallet installed into bay #, where the valid bay numbers for the RIGHT CCU backplanes are 0 through 4. If the bay identifier is not provided, cop information about all 5 possible RIGHT CCU backplane power pallets will be provided.

ccu_ppcl#
Indicates to provide the cop information for the LEFT CCU backplane power pallet controller installed into bay #, where the valid bay numbers for the LEFT CCU backplanes are 0 through 4. If the bay identifier is not provided, cop information about all 5 possible LEFT CCU backplane power pallet controllers will be provided.

ccu_ppcr#
Indicates to provide the cop information for the RIGHT CCU backplane power pallet controller installed into bay #, where the valid bay numbers for the RIGHT CCU backplanes are 0 through 4. If the bay identifier is not provided, cop information about all 5 possible RIGHT CCU backplane power pallet controllers will be provided.

xrt Indicates to display the cop information for both the odd and the even side xrt boards installed in the crossbar.

xrte Indicates to display the cop information for the even side xrt board installed in the crossbar.

xrto Indicates to display the cop information for the odd side xrt board installed in the crossbar.

xs0 Indicates to display the cop information for both the odd and the even side xs0 boards installed in the crossbar.

xs0e Indicates to display the cop information for the even side xs0 board installed in the crossbar.

xs0o Indicates to display the cop information for the odd side xs0 board installed in the crossbar.

xs1 Indicates to display the cop information for both the odd and the even side xs1 boards installed in the crossbar.

xs1e Indicates to display the cop information for the even side xs1 board installed in the crossbar.

xs1o Indicates to display the cop information for the odd side xs1 board installed in the crossbar.

xcl Indicates to display the cop information for the XCL board installed into crossbar.

xbar_pc
Indicates to display the cop information for both the odd and the even side power pallets in the crossbar.

xbar_ppc
Indicates to display the cop information for the power pallet controller physically connected to slot 6 of the Base IO backplane which provide power control for the xbar.

The **cop** utility processes the options provided the by user and generates a minimal list of cop devices to be copped. It then processes the generated list of cop devices to determine if any of the requested devices are not installed in the system. Those devices not installed in the system will be removed for the cop list and cop information for those devices remaining in the list will then be presented to the user.

The cop information displayed to the user is: slot name, device type, part number, serial number, ring_revision, wire revision, assembly revision, SST DC pattern revision and SST AC pattern revision. The word *unit* is used to identify a board, a backplane, a power pallet, a power pallet controller, or a bay power controller. If the device being copped is a pre-production board (ie, development has not yet turned over the board to production engineering) a max xecn level is displayed along with a 128 bit field which identifies which xecn's are installed in the board. Xecn 1 is the left most bit of the XECN Mask.

FILES

None.

NAME

cop contents – defines the actual contents of the cop chips for the boards developed for the c3 machine.

DESCRIPTION

The contents of the cop chips are explained to inform the user of the information extracted from the cop chips and to inform the user of the information which must be supplied when copmod is executed to alter the contents of the cop chips. The cop chip contains seven basic pieces of information: part number, serial number, ring revision, wiring revision, assembly revision, DC revision and AC revision. Depending on if the board/backplane is in pre-production (in initial debug) or in production (released to manufacturing) the actual contents of the cop chip will vary.

part number

A Convex part number is a 16 digit number which has three parts: the family code, the sequence field, and the dash number. The family code is a three digit value which defines a first level grouping of like items. For example, a family code of 204 defines the family of disk drives. The sequence field is a 6 digit decimal value assigned by Document Control. The dash number is a 3 digit decimal value which is defines subgroupings for the device. An examples of a dash number is 201 (the board stuff listing.) Together, the family code, sequence number, and the suffix uniquely defines the part. For board identification, the family code and the sequence number are sufficient to define the board type. Thus within the cop chip the family code and sequence are all that is maintains out of the Convex part number. The definition of this field is the same for pre-production and production units.

serial number

The serial number in the cop chip is an 8 digit decimal number that defines specifically which board of the specified part number the user is dealing with. This value is read directly from the board by the user programming the cop chip. The definition of this field is the same for pre-production and production units.

ring_rev

The ring revision in the cop chip is a 2 character field which defines which revision of the unit's scan ring needs to be accessed to support scan operations. This information is NOT supplied by the user, but is derived by the copmod utility at the time the wire_rev and asm_rev are written using cop_mod. For those units which do not have a need for this field, the ring_rev will be set to the value 00 (zero zero) to identify no scan rings. The definition of this field is the same for pre-production and production units.

wire_rev

The wiring revision in the cop chip is a 2 character field which defines the wire spin level of a unit. Each unit begins with wire_rev A and increases to ZZ. The definition of this field is the save for pre-production and production units.

asm_rev

The assembly revision in the cop chip is a 2 character field which defines the current released assembly revision of the unit. For production units, this field begins with A and increases by one letter as each set of ECN's are released. The allowable range for this field is A to ZZ. It is important to note that the asm_rev is not tied to a specific wire_rev, but is a running count of the number released revisions of a unit. For a pre-production board, this field takes on the value 00 (that is zero zero), and is the method by which the user is informed that the unit being cop modified is a unit which is in development debug.

DC revision

This defines the revision of the SST data patterns to use for DC testing. The valid range for this field is 1 to 255. A value of 0 specifies that data pattern tracking is based upon the assembly and wire revisions.

AC revision

This defines the revision of the SST data patterns to use for AC testing. The valid range for this field is 1 to 255. A value of 0 specifies that data pattern tracking is based upon the assembly and wire revisions.

max_ecn_level

The max_ecn_level is a two digit field which is only in the cop of a pre-production unit. This field identifies to the user the last XECN number which was installed into the unit. Again, this field only exists for pre-production units.

installed_ecn

The installed_ecns in the cop chip is an array of bit masks which identifies which XECN's are installed into the unit and which X-ECN's are not installed in the unit. The format of this field will be a bit array which ranges from 1 to the max_ecn_level, with XECN 1 being represented by the left-most bit in the field, and XECN N being represented by the right-most bit in the field. A value of 1 in an XECN position identifies that the XECN was installed on the unit, and a 0 in an XECN position identified that the XECN is missing from the unit.

signature and checksum

In addition to the board specific information installed in the cop chip, a checksum and signature are also installed into the cop chip to aid in verifying the validity of the information.

SEE ALSO

cop(1D)
copmod(1D)

BUGS

None.

NAME

cpualloc – modifies the complex configuration by adding or removing heads

SYNOPSIS

cpualloc -d cpuid
cpualloc -e cpuid

DESCRIPTION

cpualloc is used to enable or disable heads in the system. A return of 0 indicates the operation completed successfully whereas anything else indicates an error.

The following steps are performed to disable a head. It's assumed the head is in a quiescent state, i.e. no memory requests are active from this head.

(1) Sleep for 2 seconds to allow head to complete idle transition.
(2) Check that the head is hung (scalar_hung register).
(3) Deconfigure head from the complex on the XBAR.
(4) Clear the head (sysreset level 0 initialization).

The following steps are performed to enable a head. It's assumed the head is off-line already.

(1) Initialize the head and load control stores (sysreset level 2 initialization).
(2) Initialize the head to it boots in the idle state.
(3) Configure head into the complex on the XBAR.
(4) Turn on the clocks to the head.

The following options are supported:

-d cpuid
Remove the specified head from the complex. The valid cpuid values are 0 to 7.

-e cpuid
Add the specified head into the complex. The valid cpuid values are 0 to 7.

DIAGNOSTICS

cpualloc has the following diagnostic messages:

Error	Description
1	Illegal argument passed to cpualloc
2	Unable to reserve the specified CPU
3	Specified CPU is not available for use
4	Initialization of CPU failed
5	scn_init() function failed
6	CPU is not hung and must be - cannot deconfigure
7	Cannot configure a running CPU
8	Cannot deconfigure a non-running CPU
9	Invalid data found for head 0 in the Configuration Database

Error code 1 indicates that either an unknown option (other than -d or -e) was specified, the cpuid was not specified or the cpuid was out-of-range.

Error code 2 indicates that some other SPU process has sole access to the specified CPU. All other processes are inhibited from changing the state of the CPU.

Error code 3 indicates that the Configuration Database is incorrectly setup for the specified CPU. The cpu_avail_n (n is the CPU id, 0-7) entry must be 1.

Error code 4 indicates that initialization of the CPU failed for some reason. The error returned from the init_boards() routine is displayed. An example of this error message is shown below. The number in the parenthesis is the returned error code.

```
cpualloc: error occurred when initializing CPU 0 (-1)
```

Error code 5 indicates an internal failure.

Error code 6 indicates the bit in the scalar_halt register indicating the CPU is halted is not set. If it is not set then the disconnect_cpu microcode operation did not successfully complete.

Error code 7 indicates the CPU is currently configured as part of ConvexOS and therefore cannot be configured on-line (it's already on-line.)

Error code 8 indicates the CPU is not configured as part of ConvexOS and therefore cannot be configured off-line (it' already off-line.)

Error code 9 indicates that the data contained in the Configuration Database is invalid for configuring the CPU in port 0 back on-line. Attempting to do so will result in a hard error.

SUBSYSTEMS AFFECTED

cpualloc affects the state XBAR boards (only the configuration ring) and the CPU being processed.

NAME

cs – control store loader/verifier and cpu RAM initializer

SYNOPSIS

cs [-c #] [-l] [-v] [-sp/vp] [-us/sr/ip/dc/pte/nrc/ua/ul/um/vd [filename]]

DESCRIPTION

The **cs** utility loads and/or verifies the control stores of the C3 scalar processor and vector processor boards. In addition, this utility initializes the CPU RAMs. This utility provides the ability to process any or all control stores individually via command line options. All control stores specified of the same type are loaded in parallel for speed improvement.

The control store loader will search the current directory for the corresponding .b file. If the expected file is not found the utility will search for the local copy in /diag/db/ctl_store. An error will be displayed if neither directory has the correct file.

The designer may load a test version of a control store by specifying the pathname of the file to be loaded immediately following one of the specific control store options.

The processor RAMs are initialized to zero with proper parity. All purge RAMs are also initialized.

-c # The -c option allows a specific cpu to be specified. The valid range is 0 - 7. This option can be used in conjunction with other options to process a specific control store. The default is all cpus.

-l The -l option performs a control store load/initialization without verification. This option will load the control stores much faster than if verification is enabled because each processor will be loaded in parallel.

-v The -v option performs a verification of the control store or RAM specified on the cpus specified. If any failures are detected the failing RAM or control store will be displayed along with the expected and read values. This option should not be performed on RAMs unless they had just been initialized.

-sp The -sp option specifies that scalar processors only should be loaded/verified. This option can be used in conjunction with the -c option to load or verify a specific scalar processor only. All control store and RAMs associated with the scalar processors listed will be processed.

-vp The -sp option specifies that vector processors only should be loaded/verified. This option can be used in conjunction with the -c option to load or verify a specific vector processor only. All control store and RAMs associated with the vector processors listed will be processed.

-us Load/verify the scalar processor us control stores only.

-sr Load/verify the scalar processor sr control stores only.

-ip Load/verify the scalar processor ip RAMs.

-dc Load/verify the scalar processor dc RAMs.

-pte Load/verify the scalar processor pte RAMs.

-nrc Load/verify the scalar processor nrc RAMs.

-rf Load/verify the scalar processor rf RAMs.

-ua Load/verify the vector processor ua control stores.

-ul Load/verify the vector processor ul control stores.

-um Load/verify the vector processor um control stores.

-vd Load/verify the vector processor vd control stores.

FILES

None.

SEE ALSO

sysreset(3D)

NAME

cti - special files used by the cti library: info and par.

DESCRIPTION

The cti is a library that provides parameter management and sequencing for diagnostic tests. It provides a standard control interface for interactive, script, or X11 (through xdiag) interaction.

The cti reads two types of text files to tailor its use for the particular application. One type of file is referred to as a parameter file, it sets the values of the parameters, and executes commands. Paaramеter file names MUST end in ".par". The other type of file is the information file, it contains information used by the test sequencer about the subtests in the test. Instances of information files MUST end in ".info".

The parameter and information files are searched for in the same manner. First the file is searched for relative to the current working directory. Note that the ".par" and ".info" are required, and if missing will be added to the name. If the file is not found in the current working directory, and the name provided does not contain any slashes (/), then the default path is searched. For both types of file, the default is **/diag/test/cti**. The default path may be changed by setting the appropriate global data, either **_CTI_par_path**, or **_CTI_info_path**, to a NULL terminated pathname. The last character of the pathname must be '/'.

PARAMETER FILE

The parameter file has a simple format. It is a text file, and each line specifies a parameter or command, and a value (the value is optional, depending on usage.). A parameter and a command are simply names that have been added to the cti through calls to *CTIaddParameter()*, or *CTIaddCommand()*. A parameter can have both a value and a function associated with it, while a command has only a function associated with it. Each line in the file is parsed for a name, and an optional value. For parameters the value is updated with the supplied value, then if there is an associated function, it is called. If the name is of a command, the function is called, and the value is supplied as a parameter. Note that these actions are taken as the names are encountered in the file.

INFORMATION FILE

The information file is quite a bit more complicated, it has several categories of information, and the parsing of each of these categories encompass more than one line. These categories provide information to build linked lists that are used to control the sequencing of the test. Each test has an information file named in the form: "test_name.info", i.e. "spu4000.info". The information file is read in when CTInewTest() is called.

The categories are identified on a header line, category names are; Subtest, Class, and Subtest-Function. Each of these is parsed in a slightly different manner. One common feature is that the information that makes up the category is bounded by opening and curly braces which appear alone on a line.

Subtest Each line in the subtest category contains a description of a subtest. The fields for each of the lines are:

number class description [timeout] [dependency ...]

number supplies the subtest number. This is the number used when referring to the subtest in a sequence string.

class supplies the class number that the subtest belongs to.

description is the string that is presented to the user in response to an info command. It is also displayed as part of the information displayed for a paused subtest.

timeout is the number of seconds that the subtest is allowd to run before the sequencer times it out.

dependency is a parameter name (preceded with a dash), followed by a range. When the range evaluates to true the subtest will not be run. There can be any number of dependencies. Be sure to precede the parameter name with a dash.

In addition to the subtest lines, there are several special lines that can be used. Special lines affect the default parameters for all of the lines that are parsed after the special line is encountered.

_Initialize subtest_function, _Subtest subtest_function, and **_Cleanup subtest_function** These three funciton specifications identify the functions that will be called by the sequencer. **Initialize** is called before the subtest, **Subtest** is called to run the subtest, and **Cleanup** is called after the subtest. The cti has a subtest function called CTIsubtestNULL that can be specified when no function is to be called. Before a subtest can be referred to, it must be added from within the parogram - this is done with a call to CTIaddSubtestFunction.

_Timeout provided with a number, identifies the timeout value for all subtests that are listed following this line, and that do not have a timeout value in the optional timeout field.

Class Classes are collections of subtests. The class category gives general information about the classes. The subtest membership in a class is stated in the Subtest category. The class category has lines of the form:

number description

number is the identifier that is used in the Subtest list to specify membership in the class.

description is used to give meaning to the class, explaining what binds the Subtests into a group.

SubtestFunction

This provides a means for naming the various functions for individual subtests. The usual method is to name them using the _Initialize, _Subtest, and _Cleanup lines. The SubtestFunction category contains a list of subtests whose functions are to be specified. Not that the fields are semi optional, they can only be omitted if all of the following fields are not to be specified. If no function change is needed for a preceding function, use a pair of double quotes as a placeholder. Each line has the form:

subtest subtest_function initialize_function cleanup_function

subtest identifies the number of the subtest that this line specifies functions for.

subtest_function names the function that is called to run the subtest.

initialize_function names the function that is called prior to the subtest.

cleanup_function names the function that is called after the subetest is run.

SEE ALSO

cti(3)

NAME

diaginit - power on the C3800 computer system

SYNOPSIS

diaginit [-v]

The *-v* option causes the full bay status reports to be printed.

DESCRIPTION

The **diaginit** utility powers up all boards and bays in the system. This is accomplished by interrogating the power subsystem for information concerning which bays and boards are installed in the system. This power system configuration checking includes:

(1) determining what uart channels have their bay cable interlocks satisfied,
(2) coordinating loading of firmware into the BPCs and PPCs,
(3) validating backplane, board, and power pallet combinations,
(4) updating the configuration database to reflect the system configuration,
(5) generating a relationship of uart channels and bay positions in the system,
(6) linking the scan rings for the boards the current set of linked scan rings,
(7) and applying power to the valid boards in the system.

After **diaginit** completes, every valid board in the system is powered up with its scan ring loaded with the default image. In addition, the CDB reflects what's installed in the system, what's powered, what's not powered, and cop information for all backplanes, boards, power pallets, PPCs and BPCs.

Only those bays and/or boards that have not been previously powered up are affected by **diaginit**. In other words, if a particular bay has been powered up by a previous invocation of **diaginit** then another **diaginit** will determine that fact and not attempt to power up that bay again. Each step in this process is described in more detail in the following paragraphs.

For each uart channel which indicates that a cable is connected to a BPC, initialize the uart channel and attempt to put the connected BPC into a reset state.

Diaginit waits for a message from the BPC which indicates whether good firmware is loaded or not. The firmware status is stored in the Configuration Database (CDB) for later use. Control of communications with this uart/bay is passed to **bpccommd**.

Next the state of the firmware in each BPC is checked. If either the firmware is bad (denoted by previous step) or it is the incorrect version (checked against the bpc_master_fw_rev CDB entry), the firmware is now downloaded. The current revision is stored in the CDB.

At this point, each cabled BPC should be fully functional with the exception of having received any required setpoint data.

Now each BPC is queried as to which PPCs are installed, their current firmware revision/validity and whether the configuration is valid. A determination is made as to which uart channels are connected to which bay (which determines the ports) and the bay types installed in each bay position. Power is applied to all valid PPCs. All this information and status is stored in the CDB.

For each PPC that either has invalid firmware or the incorrect revision of firmware, download the current firmware to it. Update the CDB with this data.

Now download any setpoint information into the BPCs. This includes both the temperature and voltage setpoints.

The cop chips for each valid BPC, PPC and backplane is read and stored in the CDB. The correct scan ring structure for each board, according to the cop information, is linked into the system.

For each valid PPC in the system, apply power to their associated logic board(s). At this time, setpoint information (both temperature and voltage setpoints) is downloaded to the PPCs. As each board is successfully powered up, a diagnostic test of the ring is executed to verify ring continuity and scan engine functionality. In addition, each of the board default rings are loaded in their respective rings. As each board is completed, the CDB is updated to reflect the new system state.

During the power up and initialization of the I/O bay, two other items are done. These are described below.

(1) After the CU and XBAR boards have been powered up, tested and initialized, the clock frequency oscillator on the CU board is set to run using the 16.7ns oscillator. If the 16.7ns oscillator is not found, the CU is set to run at the nominal frequency. This is done by determining the clock frequencies for all three settings (fast, nominal, slow) and setting the CU board to use the 16.7 oscillator.

(2) After all boards in the I/O bay are powered up, tested and initialized, the CCUs are powered up and initialized.

The last item **diaginit** does is print a table that shows the status of the logic boards it powered up. An example of this table is shown below.

```
*****************************************
*BAY 0  |BAY 1  |BAY 2  |BAY3   |BASE IO*
*       |       |       |       |       *
*       |       |       |sp     |ia     *
*       |       |       |vp     |cu     *
*vp @   |       |       |       |xbar   *
*sp     |       |       |       |       *
*       |       |       |mb #   |ccu-r  *
*****************************************
```

@ - Indicates failure applying power
# - Indicates failure running spu_4000 st_611

Only those boards actual manipulated are shown. Any errors applying power to a board is denoted by a '@' after the board, for example see the vp entry in bay0. Any error detected during the scan ring tests is denoted by a '#' after the board, for example see the mb entry in bay3. Any failure marks that board as unavailable for use except for diagnostic testing.

SEE ALSO

sys_shutdown(1D)
powerup(1D)
powerdown(1D)

NAME

display_log - print messages contained in the error_log file

SYNOPSIS

display_log [-t] [log_filename]

The *-t* option specifies the percentage of events to print.

The *log_filename* option specifies the log file to use.

DESCRIPTION

This utility prints messages contained in the specified log file for the specified time period. These messages are printed in sequential order, from oldest to latest, to standard output.

The *-t* option is used to specify the percentage of messages logged to print. This is a floating point number and denotes a percentage of days from the current time. For example, a value of "-1.0" specifies all messages logged within the past day, a value of "-2.0" specifies all messages logged within the past two days and a value of "-.5" specifies all messages logged within the past 12 hours. The default is all messages.

An alternate error log file may be specified. The default file is "/diag/data/error_log".

Examples:

display_log	Display all messages in /diag/data/error_log.
display_log -0.10	Display all messages in /diag/data/error_log logged within the last 2.4 hours.
display_log /tmp/errlog	Display all messages in /tmp/errlog file.
display_log -0.5 /tmp/errlog	Display all messages in /tmp/errlog file logged within the past 12 hours.

SEE ALSO

event_browser(1D)
xevent_browser(1D)

NAME

errintd – error interrupt daemon and soft error logger

SYNOPSIS

errintd [–hs] [–c nn] [-i] [–r nn] [–f FILE]

DESCRIPTION

The **errintd** utility monitors a CONVEX C3800 Series computer system for hardware error conditions. Two classes of errors are detected by **errintd**: soft and hard errors.

Soft errors include correctable single-bit memory errors, SP soft errors and IA soft errors. Single-bit memory errors are transparent to the system user. SP soft errors result from an error on a purge RAM. IA soft errors result from an error on one of the CCUs. This type of error, if it occurs, will be treated as a hard error if and only if the CCU failed due to a hard error.

Hard errors include all parity errors, internal references to non-existent memory, multi-bit memory errors, etc. Hard errors always result in the immediate halt of the system and are, therefore, fatal.

In addition to monitoring for errors, **errintd** starts the main memory sniffer, **mm_sniff**, if desired. **mm_sniff** is started whenever the *-s* option and the *-r nn* option (nn being non-zero) are specified. See **mm_sniff(1d)** for more information on the main memory sniffer.

In the normal operating environment, **errintd** is started by the ConvexOS boot script. All memory soft error output from **errintd** is time-stamped and stored in the Configuration Database (CDB). Information for IA soft errors are logged in the event log.

When started, **errintd** initializes the hard error masks on the NCU. All CPUs for which the *dynamic_cpu_harderrs_p* (p is the port number) entry in the CDB is set are not included in the mask. Hard errors on these processors are polled by reading the hard_err1 register. If one is detected then only error state from this one CPU is saved. In addition clocks on this CPU are halted and it is taken off-line. This mode of operation is called Automatic Processor Recovery (**APR**). It is enabled via the **xsys_config** utility which initializes the *enable_cpu_harderrs* CDB entry. A 0 in this entry enables the **APR** mode.

Single-bit memory error data is stored in the CDB. Single-bit memory errors are isolated to the memory chip level. A count of total soft errors for each failed memory chip is maintained. By default, **errintd** will store a maximum of 60 memory chip entries in the CDB.

Once the number of total memory chip failures reaches 75% capacity and a burst of errors occur (e.g., at a rate of 1 every 10 seconds), the logging of new chips in error is throttled, or governed, to prevent the log from immediately reaching its capacity. Whenever throttling occurs, a message is written to the console.

All SP and IA soft errors are also stored in the CDB. All that is stored is the failing board (or CCU), the dates of the first and last failures, and the total number of failures per board.

In the event of a hard error (other than CCU hard errors), **errintd** will call the **hard_logger** utility which is responsible for analyzing the error. After the **hard_logger** completes, **errintd** terminates execution.

If CDB gets full then the current contents are dumped to an archive file, CDB is reinitialized, and **errintd** continues with normal logging. Only one archive file is used so if CDB get full again, the old contents of the archive file will be overwritten the current contents in CDB. This archiving will occur at most once a day.

OPTIONS

The following options are available:

-h Log hard errors.

-s Log soft errors.

Note:

If none of the previous two options are specified (i.e. **-h** or **-s**), **errintd** will default to logging both types of errors.

-c nn The **-c** option specifies the maximum softlog size. The value **nn** is the maximum number of failed memory chips on which the softlog will retain information. The default is 60 entries.

-r nn This option specifies the memory sniff rate in Mbytes/day. This option is passed to **mm_sniff(1d)**. The default is 32 Mbytes/day. If a rate of zero is specified or soft errors are not being logged, then **mm_sniff** will not be started.

-f FILE Sends **errintd** output to *FILE*. By default, output goes to *stdout*.

BUGS

The *-c nn* option is not yet implemented.

FILES

/diag/data/softlog_archive - contains last archived soft error information

SEE ALSO

hard_logger(1D)
mm_sniff(1D)
xsys_config(1D)
softlog(5D)

NAME

errlogd - system daemon for logging and forwarding event messages

SYNOPSIS

errlogd [-d]

DESCRIPTION

Errlogd is a daemon process that provides user processes with a single point of access to the event/error log. There are two primary benefits of using a process such as this to do the logging rather than having individual processes log to disk directly.

First, the user process is not delayed by the logging of the event to disk. The only delay is in passing the message to the System V "messages" interprocess communication (IPC) mechanism (through which the event message is passed from the user process to **errlogd**). This delay is significantly less than the disk accesses required to log the message.

Second, as **errlogd** acts as a funnel for all messages, it also provides a single point of contact for processes that wish to be informed of certain events. **Errlogd** supports a mechanism that permits user processes to request to be informed about some, or all, of the events that are occurring in the system. As each event is received, **errlogd** scans a linked list of requests and forwards the current event to each process that has requested to be informed about it.

USAGE

Errlogd is typically invoked as a background process at system boot time. When it is invoked this way, it is not passed any arguments. However, there is one supported command line argument, **-d**, that may be used for debug purposes. It causes all messages received by the process to be formatted and displayed as they are logged to disk, thus providing a mechanism to monitor all event traffic through **errlogd**. If **-d** is not specified, **errlogd** will disassociate itself from its controlling terminal upon initialization.

EXAMPLES

errlogd &

Start **errlogd** in background at boot time.

errlogd -d

Start **errlogd** in "debug" mode.

FILES

/diag/data/error_log

The file where where events are actually logged to disk.

DIAGNOSTICS

None.

SEE ALSO

librpt_error(3D)
log_error(3D)
display_log(1D)
errlogd(3D)

BUGS

None.

NAME

event_browser – selectively displays messages from the event log.

SYNOPSIS

event_browser [options]

-log pathname	which log to report on
-before time	report events before time
-after time	report events after time
+/-grep reg_expression	filter events that match expression
+/-message message_number	filter by message number
+/-severity severity_number	filter on severity
+/-source source_number	filter based on source
+/-type type_number	filter on the type number

On options with '+/-'

+ includes events with that characteristic

- excludes events with that characteristic

DESCRIPTION

The program **event_browser** is used to view messages logged to the event log. Either all of the messages in the log can be displayed, or various filters may be employed to select messages of interest. Filters exist for time stamp, severity, message number, type, and source. Regular expressions may also be used to select messages, but, since the message must be formatted before this filter can be applied it is the least efficient method for selecting messages.

There are two modes of operation for **event_browser**. If the invoking command line contains a **report** option, all of the command options are processed in order. After the last option is processed, **event_browser** exits. The other mode is interactive, it is entered if the program is invoked without a **report** option on the command line. In interactive mode, the command line options are processed, then the program processes options interactively until the **quit** command is entered.

The command set that controls **event_browser** is the same regardless of which mode the program is in. Note that command line options are acted on as they are encountered. The following additional commands are useful in interactive mode:

settings	view all of the current settings
reset [option [item]]	reset options to initial state
?,h,help [option]	print help text
q or quit	exit the program

One additional feature - sending **event_browser** a **<Cntrl> C** while a report is being printed, causes the report to terminate. The program then continues as if the report had finished normally. If **event_browser** receives a **<Cntrl> C** at any other time it exits.

In interactive mode, entering any of the options without an argument will show the current value of that option. On the command line, if an option is entered without an argument, the usage message is printed and the program exits.

Time

There are two time options, **before** and **after.** All events logged previous to the **before** time and since the **after** time pass the time filter. The initial **before** value is the present time plus one year, while the initial **after** value is "Wed Dec 31 18:00:00 1969".

The time argument format is the same basic format used in asctime(3). All of the fields are optional - the colons are used as keys for fields that occur after them. The preexisting time value is used as a seed - values provided are used to modify that time.

There are two time formats:

month_name day_of_month hour:min:sec year

day_of_week hour:min:sec

Only the first three characters of the **month_name** or **day_of_week** are used. The month format takes precedence, so if a **month_name** or a **day_of_month** is provided, the **day_of_week is ignored.**

Filters

The filters are used to select messages of interest for output. The filters can have any number of values, and can be either inclusive (+), or exclusive (-). If a filter is exclusive, messages that have that characteristic are not printed. If an inclusive filter is provided, **only** messages that have that characteristic are printed.

Filters are provided for the values in the message header; source, type, and message number. In addition regular expression filtering is applied to the entire formatted event buffer as a unit. Because regular expression filtering operates on the formatted message, this is by far the least effective method of filtering. To minimize the amount of work needed to create a report, grep should be used in combination with other filters.

Reset

Reset sets options back to their startup state. This command has optional arguments: the first identifies the option that is to be reset, the second identifies the item relating to the option that is to be reset. Items will only exist for the filters, where an option can have several values. An example of an item is an individual regular expression string.

If no option is provided, all of the options and their items are reset. The end result is that all filters are zeroed, the before time is set to the present time plus one year, the after time is set to the epoch, and the log file name is set to the default. If an option is supplied, only that option is reset. If an option and item are provided, only the item that matches both the option and item is reset.

Select

Select The **select** command puts **event_browser** into a mode where filter values and their descriptions can be viewed and selected. While in this mode, a different parser is in affect, and the following single character commands are all that is recognized:

u - up, the description of the the previous value is displayed.
d - down, the description of the next value is displayed.
carriage return - select, the value is placed in the list that you are selecting for.
q - quit, return to command mode.

In addition a to the single character commands, a number can be entered. This allows the user to jump to the region he is interested in.

SEE ALSO

xevent_browser(1)

NAME

initall – initialize system hardware state and main memory to the point where processors are defaulted, main memory has been initialized, and all associated memory clocks are running.

SYNOPSIS

initall

DESCRIPTION

The **initall** utility performs the operation of initializing the C3800 Series computer to support the booting of ConvexOS. Specifically, initall first checks if a bus error is present by reading SWIP registers. If a bus error exists then it is cleared before continuing.

Initall next executes sysreset with a level 2 system reset. Sysreset uses the information in the configuration database to determine which processors, memory boards, crossbar boards, ia, and ccu's are to be reset (see xsys_config man page for detail on selecting configuration.) Sysreset will then (in a pre-defined order) initialize each selected board based on the individual board's requirements (see man page for sysreset for details.) For the purpose of describing initall, each board has their rams caches, and hardware state initialized with good parity. In addition, for each processor identified to part of the C3800 Series complex, control stores are loaded for the scalar and vector processors.

Next **initall** executes mminit. The mminit utility uses the configuration identified in the configuration database to determine which memory boards, processors, and ia boards are to be configured. First, mminit will size the memory configuration of each memory board (128Mbyte, 256Mbyte, or 512Mbyte) by directly accessing each board to determine the size of the DRAMS and the number of DRAM rows. A physical configuration map which parallels the c2 PCM is generated and stored into the configuration database. The clocks for the memory system are then enable and main memory is initialized with zeros to clear memory and to set up good ecc (FYI: the memory test logic of the cu board is used and mminit will normally take < 10 seconds.) More specifically, memory clocks include the free running clocks for the memory and interface adapter boards; 2x and 3x clock for the crossbar, memory and interface adapter boards. When mminit terminates, clocks for the memory system are left enabled.

If an error is detected by sysreset or mminit, **initall** will abort the initialization sequence and exit with a non-zero status.

FILES

None.

SEE ALSO

sysreset(1D)
margin(1D)
cs(1D)
mminit(1D)

NAME

margin – set voltage or clock values to test margin limits.

SYNOPSIS

margin -v [-{n | u | l | U | L | a}] <[bay#] [xbar] [port#] [ccu# #] [boardname#] ...>

margin -c [-{n | u | l}]

DESCRIPTION

Margin formats the commands necessary to request nominal or marginal voltage or clock values to be set by the various controllers.

USAGE

A complete list of valid options for the margin command is outlined below.

-v Indicates that this command applies to power supplies rather than clocks. If **-v** is not specified, **-c** is the default.

-c Indicates that this command applies to clocks rather than power supplies. If **-v** is not specified, **-c** is the default.

-n Set nominal voltage (based on a predetermined value for the specified board) or clock frequency. This is the default option, unless -l or -u are specified.

-l Set lower margin voltage (based on a predetermined value for the specified board) or clock frequency to 5% lower than the nominal value.

-u Set upper margin voltage (based on a predetermined value for the specified board) or clock frequency to 5% higher than the nominal value.

-L Set upper margin voltage (based on a predetermined value for the specified board) or clock frequency to 3% lower than the nominal value.

-U Set upper margin voltage (based on a predetermined value for the specified board) or clock frequency to 3% higher than the nominal value.

-a Use the specified voltage setting (n,u,l,U, or L) to margin the voltage of all boards and bays in the system. Specifying a board name after this option is ignored, since this switch already sets all boards.

bay[0-4] xbar ccu[0-39] mb[0-7] sp[0-7]
ia[0-8] vp[0-7] xiop[0-8] cu kcu Identify a particular bay, board, etc. to which this command should apply. **(Note that all numerical values are decimal).** These arguments do not apply to margining the system clocks, but at least one must be specified when margining voltages. When a bay is specified, all boards in that bay are margined. In the other cases, new (margined) voltage setpoints are downloaded to the appropriate PPC's.

EXAMPLES

margin -v -n bay1 xbar mb0

Download nominal voltage setpoints to all boards in bay 1, the xbar and mb0.

margin -vna

Set the nominal voltage on the all system boards and bays.

margin -c

Set the nominal frequency on the system clock.

FILES

Nominal voltage setpoint data for each board in the system is stored in the configuration database.

DIAGNOSTICS

Diagnostic messages for this command are TBD.

SEE ALSO

powerup(1D)
powerdown(1D)
powermon(1D)

BUGS

None.

NAME

mminit – main memory initialization

SYNOPSIS

mminit [-] [-p n] [-n] [-d] [-x] [-t]

DESCRIPTION

mminit is used to initialize main memory after a system power up. The SPU, via the Memory Test Logic (MTL), will be used to both initialize and verify memory. The sequence of events it follows to perform this is shown below. If an error is detected during initialization, **mminit** returns a -1. Otherwise, **mminit** returns a 0.

(1) Determine memory configuration of each MB (size of DRAMs and the number of rows of DRAMs) by writing/reading various locations.
(2) Generate a Physical Configuration Map (PCM) from this information and store both the memory configuration data from step 1 and the PCM in the Configuration Database.
(3) Determine the memory interleave factors required for both the IA and SP. The interleave is maximized by grouping similar size boards together.
(4) Default scan rings for all boards requiring memory configuration data: IA, MB, XBAR, and SP.
(5) If specified then disable errors on all boards.
(6) Fill memory with specified pattern.
(7) If not disabled then verify memory.

Step 2 also examines the Configuration Database entry for each MB that is used to force that MB to a specified memory size. The four possible values are: 0 - which denotes to use the calculated size from step 1, 128 - which denotes to force the MB to be 128MB in size, 256 - which denotes to force the MB to be 256MB in size, and 512 - which denotes to force the MB to be 512MB in size. An MB cannot be forced to a larger memory size, e.g. a 128MB board cannot be made a 256MB board. Trying to do so produces an error and **mminit** terminates. These entries can be modified using the **xsys_config** utility.

The type of memory configuration data obtained for each memory board includes: whether 1, 2 or 4 rows of DRAMs exist on each DRAM memory module (DMM) and whether 1-Mbit or 4-Mbit DRAMs are used.

The following options are supported:

- Options and arguments will be read from standard input.

-p n Set the memory initialization pattern to the byte hex pattern specified by **n**. If the **-p** option is not specified then memory will be initialized to all zeroes. For example, the following command will initialize memory to all ones: **mminit -p 0xffffffffffffffff**.

-n Do not verify the memory initialization. The default is to verify.

-d Disable error checking on the CU, IA, MB, and XBAR boards. The default is to leave error checking enabled.

-x Disable all processor ports on the XBAR (except the IA port).

-t Use the memory timing information specified in the Configuration Database. This includes the DRAM timing on the NMB and the *xbar_mem_time* entry which specifies when the XBAR expects read data to be returned.

Only MBs specified to be used will be initialized. This is controlled by the user via the Configuration Database (CDB). For each MB there are three entries in the CDB. The first entry, mb_installed_*, specifies whether the board is actually plugged into the system and powered on. The second entry, mb_avail_*, specifies whether the board is scannable and is therefore available

for use. The third entry, mb_os_req_*, specifies that the MB is to be initialized. Only if all three entries for a particular MB are set will that MB be initialized. Thus, clearing the third entry (mb_os_req_*) for those MBs the user do not which to initialize will cause **mminit** to ignore them.

The memory sizing operation consists of writing to certain memory locations on all specified MBs and then reading these same locations and verifying their contents. The order that the MBs are written/read is from the highest numbered MB to the lowest, i.e from MB7 down to MB0.

The locations written on each MB (in the order shown) are listed below. In each case the data written to each location is the address of that location (this is also the expected data). These memory operations are performed using XMAP transfers.

0x04000000	Tests for existence of 4M DRAMs (if 1M DRAMs are present then this equates to address 0). In other words, if this address does not contain 0x04000000 then 1M DRAMs are present on the MB.
0x1a000000	tests for existence of DRAM row 3 (if row 3 does not exist this equates to address 0). If this address does not contain 0x1a000000 then only two rows of DRAMs are present on the MB.
0x0a000000	tests for the existence of DRAM row 1. If this address does not contain 0x0a000000 then only 1 row of DRAMs are present of the MB.
0x02000000	this is a write to row 0 (will overwrite the 0x1a000000 address if either 1M DRAMs are present or if DRAM row 3 does not exist)

This sizing operation writes data to a temporary file (/tmp/mminit.pcm) for use as a debug tool. As each memory location is read the address, expected data, and actual data are dumped into this temporary file.

DIAGNOSTICS

mminit has the following diagnostic messages:

"**scn_init failed**" - **mminit** was unable to initialize the scan machine in order to perform scan operations.

"**Fill_memory timeout at address xxxxxxxx**" - software timeout indicating MTL request did not complete (BUSY bit is set). This usually indicates a hard error somewhere in the system.

"**Fill_memory operation failed**" - MTL request appears to complete (BUSY bit is not set) but not all addresses were written.

"**Compare_memory timeout at address xxxxxxxx**" - software timeout indicating MTL request did not complete (BUSY bit is set). This usually indicates a hard error somewhere in the system.

"**Compare_memory operation failed**" - MTL request appears to complete (BUSY bit is not set) but not all addresses were written.

"**Compare_memory comparison error**" - MTL request completed but an error was detected (CMP bit is set). The failing address and expected data (obtained via XMAP transfers) are displayed.

"**initialization by SPU failed**" - initialization of memory using the SPU failed.

"**verification by SPU failed**" - verification of memory using the SPU failed.

SEE ALSO

sysreset(1D)
xsys_config(1D)

SUBSYSTEMS AFFECTED

mminit affects the state of CU, IA, MB, and XBAR boards. It also affects the execution of code running on any of the CPUs since it initializes the remainder of the system.

BUGS

The [-] option is not implemented.

NAME

mm_sniff – main memory sniffer

SYNOPSIS

mm_sniff [*-r nn* | *-t nn*]

DESCRIPTION

The **mm_sniff** program reads all main memory within a specified amount of time. The intention is to detect locations with a single-bit error. Once detected, **errintd** attempts to eliminate the error by performing a memory scrub operation on the memory location in error.

If a location contains a single-bit error and is not read for a long period of time, it could potentially drop another bit. This would result in a double-bit error that is not correctable. In addition, multiple-bit errors are hard errors, which halt the system.

The memory sniffer always reads main memory in four-page groups, i.e 16 Kbytes. Upon invocation, based on the sniff rate in Mbytes/day, **mm_sniff** calculates the number of seconds to sleep between each read. In the event the calculated sleep time is less than 15 seconds, the value is forced to 15 seconds. This time and the time required to sniff the entire memory system are displayed on the console.

The following options are interpreted by **mm_sniff**:

-r nn Set sniff rate to *nn* Mbytes/day. The default is 32 Mbytes/day.

-t nn Set sniff sleep time to *nn* sec. This option overrides the **-r** option.

SEE ALSO

errintd(1D)
softlog(5D)

NAME

powerdown – terminate power to one or more logic boards in the system

SYNOPSIS

powerdown [**-b**] [**bay#**] [**xbar**] [**ccu# #**] [**boardname#**] ...

DESCRIPTION

Powerdown formats the commands necessary to instruct the appropriate BPC's to terminate power to the specified boards. Note that when power is removed from a board, there is still power to the PPC.

WARNING: The user should be aware that issuing this command does not disable all power into a bay. The power system bay controllers will remain powered up until their circuit breakers are thrown.

USAGE

The powerdown command may be used to terminate power to any logic board in the system. For example, to terminate power to all boards in bay 1, the user would enter **powerdown bay1**. This would turn off each of the installed logic boards, leaving the BPC and PPC's powered.

If **-b** is specified, only the logic busses on the board itself are powered off, and the board may be brought back online using the *powerup* command. Note: running **powerdown -b bay4** is equivalent to **powerdown bay4**, since powering down all boards in a bay causes the bpc to powerdown the bay itself, which removes 300v from each board.

Specifying no option causes the board to be shut down and 300 volts to be taken away. In this case, *diaginit* must be run to restart the board. If a board is specified that shares a bay power supply (bps) with another board, a warning message is printed and the user is asked if they want to continue. This is done because removing 300v means shutting off the bps, and any other boards supplied from that bps will be shut down also.

A complete list of valid arguments for the powerdown command is outlined below. They identify a particular bay, board, etc. to which this command should apply. **(Note that all numerical values are decimal).**

bay[0-4]
xbar
ccu[0-39]
mb[0-7]
sp[0-7]
ia[0-8]
vp[0-7]
xiop[0-8]
cu
kcu

EXAMPLES

powerdown -b sp6

This will cause all busses on the sp6 board to be shut down, and if another board shares a bps with this sp, it will be unaffected. The board can be powered up again with 'powerup sp6'.

powerdown bay1 xbar mb0

This will cause all logic boards in bay1, the xbar, and mb0 to be turned off and 300v to be removed. Diaginit must be run to start up these boards again as 'powerup' cannot

restore the 300v supply.

FILES

None. All information is stored in the configuration database.

DIAGNOSTICS

None.

SEE ALSO

powerup(1D)
powerup(1D)
powermon(1D)
diaginit(1D)
remove_bd(1D)

BUGS

None.

NAME

powermon – monitor voltages, temperature, and air flow

SYNOPSIS

powermon [-i <interval>] [-l <file>] [-sS] <[bay#] [xbar] [ccu##] [bdname#]
xpowermon [<X11 related arguments>]

DESCRIPTION

Powermon formats the commands necessary to request local environment status from the appropriate BPC or PPC. **Xpowermon** functions similarly, but it uses the X-Window system rather than Maryland Windows to display the data. Command line arguments for **powermon** are entered on the screen for **xpowermon**.

USAGE

When executing the **powermon** command, a particular bay or board must always be specified. Due to the amount of data that potentially may be passed to the SPU, it is not be practical to monitor multiple bays and/or boards within a single instance of the command. However, if such an effect is desired, separate commands may be issued to execute in the background for each bay.

The **powermon** utility is used to monitor the environment conditions in a particular bay or voltages and temperatures on a particular logic board.

A complete list of options available for the **powermon** command is outlined below.

bay[0-4]

xbar ccu[0-39] mb[0-7] sp[0-7] ia[0-8] vp[0-7] xiop[0-8] cu Identify a particular bay or board to which this command should apply. This parameter must always be specified.

-i <interval>

Interval between requests to the various controllers for status. The default is 5 seconds.

-l <file> [-s]

Log the data to a file. Note that there is a considerable amount of data available for collection. Depending on which options are specified, the file could grow large quite rapidly. If the **-s** option is specified, the monitor function will run silently in the background, and output to the display will be suppressed. If **-s** is specified without **-l**, data will be logged to a temporary file. If the log file already exists, it is truncated before logging begins. If the **-S** option is specified, then only a single status message is requested from the given board or bay, and that status is logged to **<file>**. Silent mode (**-s**) is implicit when when **-S** is specified.

Once initiated, **powermon** will run until terminated by a SIGINT unless **-S** is specified.

EXAMPLES

powermon mb0

Initiate monitoring functions for the board "mb0".

powermon bay4

Initiate monitoring functions for the Base IO bay, "bay4".

powermon -l /tmp/cu_stat -S cu

Log a single formatted status message for the "cu" to /tmp/cu_stat.

FILES

/tmp/powermon.<pid>

Temporary file used by **-s** and **-S** if **-l** is not specified.

DIAGNOSTICS

None.

SEE ALSO

powerup(1D)
powerdown(1D)
margin(1D)

BUGS

None.

NAME

powerup – apply power to one or more logic boards in the system

SYNOPSIS

powerup [**xbar**] [**ccu##**] [**boardname#**] **...**

DESCRIPTION

Powerup formats the commands necessary to instruct the appropriate BPCs to apply power to the specified boards.

USAGE

The powerup command may be used to apply power to any logic board in the system. For example, to apply power to all boards in bay 1, the user would enter **powerup bay1**. This would turn on each of the installed logic boards, leaving the BPC and PPC's powered.

A complete list of valid arguments for the powerup command is outlined below. They identify a particular board to which this command should apply. **(Note that all numerical values are decimal).**

xbar
ccu[0-39]
mb[0-7]
sp[0-7]
ia[0-8]
vp[0-7]
xiop[0-8]
cu
kcu

Note that *powerup* is only effective if 300 volts is still applied to the board. If it is not, or the user wishes to apply power to an entire bay, *diaginit* must be run instead.

EXAMPLES

powerup xbar mb0

This will cause all logic boards in the xbar and mb0 to be turned on.

FILES

There are no files utilized by this command. All information is stored in the configuration database.

DIAGNOSTICS

None.

SEE ALSO

powerdown(1D)
powermon(1D)
diaginit(1D)
remove_bd(1D)

BUGS

None.

NAME

pwr_util – general purpose C3800 Series power system debugging tool

SYNOPSIS

pwr_util [-s]

WARNING

Pwr_util is utility designed for debugging use only. To perform basic power and margining functions, use the powertools: altsetpts, powerup, powerdown, powermon, margin, remove_bd, sys_shutdown, and diaginit. Pwr_util should only be run by an expert user to debug the power system.

DESCRIPTION

Pwr_util is a menu-driven utility that allows the user to directly access many of the elements of the power system and the Service Processor Unit (SPU) to C3800 Series communications system. Note that during normal operations these system parameters and data are automatically initialized and maintained by **diaginit** and **sys_shutdown**. **Pwr_util** allows manual manipulation of the bay power controllers (BPCs), the power pallet controllers (PPCs), the serial communications channels from SPU to C3800 Series, and allows the user to modify or display information on a board's firmware and cop memory.

Pwr_util is a menu driven utility when no options are specified. When the user specifies the -s option, the same set of inputs is expected, but no messages or menu options are printed. This is primarily for use in a script when **pwr_util** is not being used interactively, so no user prompts are required.

USAGE

a) Bpc chk interlock

Check interlock on a specified BPC. If interlock is found, the SPU has detected the existence of a BPC, so this really tests to see if a specific BPC is actually in the system.

b) Reset uart

Reset the uart channel on the SPUs communication card. This was for early, low level debugging and should not be used. If the user wishes to clean up communications to a BPC, they should do an Offline BPC (e) followed by a Init BPC (d).

c) Bpc reset

Assert the reset line on a specified BPC. This forces the set of PPCs connected to this BPC into reset also. In general, this option is no longer used. If the user wishes to clean up communications to a BPC, they should do an Offline BPC (e) followed by a Init BPC (d).

d) Init BPC

Reinitialize the BPC and force it to look for interlock with its PPCs. This operation is normally done when powering on the system as the first step in establishing communications between SPU and BPCs. To properly configure the system, an Init BPC (d) should be followed by a Bay Config Chk (h) and then a Bay Power Chk (k). If the bpcwatchd is running with messages enabled (see the -d option in the bpcwatchd man page), messages will be output as the BPC finds interlock with its PPCs.

e) Offline BPC

Take a specified BPC offline, effectively removing it from communication with its PPCs and the SPU and shutting it down. To reactivate the BPC, run an Init BPC (d).

f) F/W revision

Check the revision numbers of a BPC or PPC's firmware. The user is prompted for BPC number and PPC number. Entering a number from 0 to 7 selects the individual PPC of that number while entering an 8 selects the BPC, rather than any individual PPC. The value returned is the revision currently in the BPC or PPC. If a -1 is returned, there is no valid firmware downloaded.

g) F/W Download

Update BPC or PPC firmware. The user is prompted for BPC number and PPC number. Entering a number from 0 to 7 selects the individual PPC of that number while entering an 8 selects the BPC, rather than any individual PPC. After selecting the target, the user is prompted for the firmware revision to download. This is a pair of numbers, major and minor revision, that specify the version of firmware to download. To enter revision 4.13 enter 4.13 for this prompt. Note that these numbers are in decimal so revision 4 A is not valid. The user is also prompted of code that receives the firmware download as the SPU sends it. This loader program can be loaded even if the BPC is not initialized, while the normal download can only be done if the BPC has been initialized.

h) Bay Config chk

Force the BPC to poll its PPCs and check that the configuration in the bay associated with this BPC is valid. A block of data, called the bay status message, is reported to the user after this operation which shows the status of the bay and each PPC connected to it. The BPC updates its internal information on the status of each PPC and the BPC environment based on this information. If the BPC is to be powered up, this Bay Config Chk (h) should be followed by the Bay Power Chk (k). To inquire about the status of the BPC or one of its PPCs without forcing the BPC to update its information, use the Send Status (n) command. See the Bay Status Message section below for an explanation of the status message format.

i) Cop Read

Read the cop information for c3800 boards, backplanes, PPCs, or bps. Performs the same function as **cop** except **cop** can cop the ccu boards, while pwr_util cannot.

j) Cop Write

Write the cop information for C3800 Series boards, backplanes, PPCs, or bps. Performs the same function as **copmod** except **copmod** can cop the ccu boards, while pwr_util cannot.

k) Bay Power chk

Force the BPC to poll its PPCs and check that the power supplies in the bay associated with this BPC is valid. A block of data, called the bay status message, is reported to the user after this operation to show the status of the power system. The BPC updates its internal information on the status of each PPC and the BPC environment based on this information. To inquire about the status of the BPC or one of its PPCs without forcing the BPC to update its information, use the Send Status (n) command. See the Bay Status Message section below for an explanation of the status message format.

l) Power On

Apply power to a board's busses. This will not bring up a board that has lost its 300 volt supply. Performs the same function as **powerup**.

m) Power Down

Remove 300v supply from a board. Performs the same function as **powerdown**.

n) Send Status

Send a status message from BPC or PPC to SPU so user can view status information. The user is prompted for BPC number and PPC number. Entering a number from 0 to 7 selects the individual PPC of that number while entering an 8 selects the BPC, rather than any individual PPC. See the Bay Status Message section below for an explanation

of the status message format.

o) Set voltage
Change board voltage setpoints. Performs the same function as **altsetpts** or **margin**.

p) Set temp
Change board or bay warning (warm) and kill level (hot) temperature setpoints. Performs the same function as **altsetpts**.

r) Busses off
Turn board voltage busses off but leave the 300 volts applied to the board. The board may be reactivated with the Power On (l) option or with the powerup utility. The Busses off option performs the same function as **powerdown -b**.

t) Transparent
For expert use only.

q) Quit
Exit the program.

BAY STATUS MESSAGE FORMAT

Several commands, including Bay Config Chk (h), Bay Power Chk (k), and Send Status, return a bay status message. This message is the BPC's table of information showing the current state of the BPC and its PPCs. An example bay status message is shown below:
SW Info (DiagIN236): Bay Power Controller status message

```
PPC Status
  bay pp  fw  prt bkpln bkpln plt bd bd bd bd bd bd bd bd bay bps bus bus
  cnf cnf rev  id slot  type  typ 0  1  2  3  4  5  6  7  pwr num  OK  on
0|ff  ff  ffff ff  ff    ff    ff ff ff ff ff ff ff ff ff  f3  ff  ff  ff
1|00  00  0110 00  06    02    01 01 00 00 00 00 00 00 00  00  00  00  00
2|00  00  0110 00  07    02    02 02 00 00 00 00 00 00 00  00  01  00  00
3|00  00  010d 00  08    02    06 06 00 00 00 00 00 00 00  00  01  00  00
4|00  00  0110 01  09    03    06 06 00 00 00 00 00 00 00  00  04  00  00
5|00  00  0110 01  0a    03    02 02 00 00 00 00 00 00 00  00  04  00  00
6|00  00  0110 01  0b    03    01 01 00 00 00 00 00 00 00  00  05  00  00
7|ff  ff  ffff ff  ff    ff    ff ff ff ff ff ff ff ff ff  f3  ff  ff  ff

Temperature Status
  f/w rev temp_stat temp_0 temp_1 temp_2 temp_3 temp_4 temp_5 temp_6 temp_7
    0209      00      ff     ff     c1     ff     ff     ae     ff     ff
in_wrm_set in_hot_set out_wrm_set out_hot_set BPC_err fan0 fan1 fan2
    9c         82         47          3a          00     00   00   00
```

The first block of information gives data on each PPC attached to the BPC. Each row represents one PPC, and each column one type of data. For example, to find the pallet type of PPC 5, look in the row starting 5|, and look across under the column labeled 'plt typ' to find that PPC 5 has a pallet type of 0x02. (Note that the whole table is in hexadecimal.) The column titles from left to right are bay configuration error code, PPC configuration error code, firmware revision, port ID, backplane slot, backplane type, pallet type, board types (0-7), bay power check error code, bay power supply number, voltage bus OK code, and a voltage bus on flag. Note that zeros in the error code, bus OK, and bus on fields represent good values (no error), while 0xff's represent empty PPCs. In our example, PPCs 0 and 7 are empty (notice the 0xff's), and PPCs 1 through 6 and installed and have no errors (00's in the error code fields). By looking at the bps column, one can see that PPCs 2 and 3 share bps number 1, and PPCs 4 and 5 share bps number 4. PPCs 1,

2, 3, 5, 6 and 7 all have firmware of revision 1.10, while PPC 3 has revision 1.0d. One warning concerning the board type field -- the board type for a CCU does not show the true board type but instead is a flag showing which CCUs are installed and need ECL type voltages. The pallet type will show that the PPC is connected to a group of CCUs, and will not match the board type. Note that PPCs for all boards but the XBAR will have a value only in the first bd type field, bd 0. The XBAR is comprised of seven separate boards which are listed in bd 1 through bd 7. The number in bd 0 represents the identifier for the XBAR as a unit. An example of a BPC with the XBAR connected to PPC 6 is shown below:

```
PPC Status
  bay pp  fw  prt bkpln bkpln plt bd bd bd bd bd bd bd bd bay bps bus bus
    cnf cnf rev  id slot  type  typ 0  1  2  3  4  5  6  7  pwr num  OK  on
0|00  00  0110 08  05    0a    0d 0f 00 00 00 00 00 00 00  00  01  00  00
1|ff  ff  ffff ff  ff    ff    ff ff ff ff ff ff ff ff f3  ff  ff  ff
2|00  00  0110 08  07    08    04 04 00 00 00 00 00 00 00  00  01  00  00
3|00  00  0110 08  08    08    07 07 00 00 00 00 00 00 00  00  00  00  00
4|ff  ff  ffff ff  ff    ff    ff ff ff ff ff ff ff ff f3  ff  ff  ff
5|ff  ff  ffff ff  ff    ff    ff ff ff ff ff ff ff ff f3  ff  ff  ff
6|00  00  0110 09  06    01    09 09 0b 0a 09 08 09 0a 0b  00  00  00  00
7|00  00  0110 08  0c    0b    0d 0f 00 00 00 00 00 00 00  00  05  00  00

Temperature Status
  f/w rev temp_stat temp_0 temp_1 temp_2 temp_3 temp_4 temp_5 temp_6 temp_7
    0209     00       ff     ff     b5     b0     b1     a8     b1     ff
in_wrm_set in_hot_set out_wrm_set out_hot_set BPC_err fan0 fan1 fan2
    9c         82         47          3a          00     01   01   00
```

The previous two examples also show the temperature and fan status for the BPC itself in the two lines following the PPC status lines. The items given in these lines are the BPC firmware revision, the status of the temperature sensors in the bay, the warm and hot setpoints for the inlet sensor, the warm and hot setpoints for the outlet sensor, an overall BPC error mask, and the status of the three bay fans.

PPC STATUS MESSAGE FORMAT

More in depth information on an individual PPC can be obtained by running a Send Status (n). The format of the returned message is somewhat different than the bay status message. An example is shown below:
SW Info (DiagIN238): Power Pallet Controller status message

```
PPC Status
 fill  fw    pp  open err err  plt bd bd bd bd bd bd bd bd slot warm hot bus
 byte state fail ilck grp code id  0  1  2  3  4  5  6  7   id  flag flg trim
 00    02    00   00   f1   1f  02  02 00 00 00 00 00 00 00  07  00   00   00

 bus_on brick brick brick brick tmp0 tmp1 tmp2 tmp3 tmp4 tmp5 tmp6 tmp7 A/D
 error  exist type  fuses input snsr snsr snsr snsr snsr snsr snsr snsr over
 00     3fff  3bdb  0000  0000  52   56   55   51   4e   5a   5b   55   00

  hous A/D0 A/D1 A/D2 A/D3 A/D4 A/D5 A/D6 A/D7
  volt cnvt cnvt cnvt cnvt cnvt cnvt cnvt cnvt
  00   f7ff 081c ffff f7ad fcb8 fcef fffe f8ce
```

This messages retrieved from the individual PPC itself, and contains fields not shown in the BPC status message. The first field (top left) is a fill byte and is unused. The fw state field shows the current state of PPC, which in the above case is waiting for a command. (See 38XX Power System Firmware Description for more information on the possible values.) The next two fields show if primary power (300v) has failed and if interlock is broken. The err grp gives the code of the last solicited or unsolicited message sent to the BPC. The pallet ID, bd, and slot ID fields serve the same function as in the bay status message. The hot warm flag is a bitwise mask showing which temperature sensors are over the current warm setpoint and the hot flag serves the same function for the hot setpoint. The bus trim flag displays (bitwise) which bus voltage was trimmed. The bus_on error field is (bitwise) which busses had an error when powering on. The brick exist field shows (bitwise) which bricks are installed on the pallet. The brick type field shows which of these bricks are 5V bricks (0's) and 2V bricks (1's). The brick fuses field displays which brick fuses show open, and the brick input field shows which bricks are off. The next fields, the tmp fields are the readings of the pallet temperature sensors in A/D counts, and the A/D over field shows (bitwise) which of these sensors is over the A/D range. The house volt field is non-zero if the pallet 12V supply is down. the last fields show the A/D counts of the voltage sensors monitoring the voltage converters.

For more information on the contents of these fields consult the 38XX Power System Firmware Description.

EXAMPLES

pwr_util

Start an interactive pwr_util session.

pwr_util -s

Start pwr_util without any user prompting or other output. Primarily for use from a script.

SEE ALSO

diaginit(1D)
altsetpts(1D)
margin(1D)
powerup(1D)
powerdown(1D)
cop(1D)
copmod(1D)
sys_shutdown(1D)

BUGS

None.

NAME

rbcdb_init - initialize the Resource Broker and Configuration Database

SYNOPSIS

rbcdb_init [-i]

DESCRIPTION

The **rbcdb_init** utility first tests if the **rbserver** is running. If it is then it is halted and a new **rbserver** started. Second if the **cdbserver** is running then it too is halted and a new **cdbserver** started.

After both servers are started, calls are made to **rb_create()** to initialize the resources in the Resource Broker. After the resources are generated, calls are made to **cdb_create()** to create entries in the Configuration Data and initialize their data values.

The -i option results in the generated **cdbserver** and **rbserver** files being copied into the *standard* files. Thus, if **cdb_startup** is called and it can't find the *checkpointed* or *normal* files, the *standard* files are used.

FILES

/diag/db/standard.db - generic configuration data file (machine independent)
/diag/db/standard.map - generic "map" file (machine independent)
/diag/db/standard.rb - generic resource data file (machine independent)

SEE ALSO

cdb_startup(1D)
cdbserver(1D)
rbserver(1D)

NAME

rbserver – LIBRB_CDB server for the Resource Broker

SYNOPSIS

rbserver

DESCRIPTION

The **rbserver** utility maintains the data structures for the Resource Broker and modifies these structures per the various LIBRB_CDB functions. LIBRB_CDB functions communicate with this utility via a message protocol. This message contains the following information: process id, function, message data, and returned status.

When first started, this utility checks whether the daemon has previously been started. If so then a fatal error is generated and the utility exits. If not then the data structures are initialized and then this utility loops waiting for a message.

rbserver will continuously loop waiting for messages unless killed.

FILES

/diag/bin/rbserver - RBSERVER binary
/diag/db/rb_config_file - contains currently defined resources
/diag/db/rb_config_file.chkpt - checkpoint copy of "rb_config_file"

SEE ALSO

cdbserver(1D)
cdb_startup(1D)
rbcdb_init(1D)

NAME

remote_disconnect – terminate remote login session to ConvexOS Console

SYNOPSIS

remote_disconnect

DESCRIPTION

Remote_disconnect is used to terminate a remote login to the ConvexOS Console window. Remote login to a C3800 SPU (as rmtdiag) is allowed only when the keyswitch is in the REMOTE position.

SEE ALSO

CONVEXOS_CONSOLE(1D)
xsfp(1D)

NAME

scan_shm_init – allocates the shared memory segment used by libscan

SYNOPSIS

scan_shm_init

DESCRIPTION

The **scan_shm_init** utility allocates the shared memory segment used by the scan package. In doing so it initializes the Rings[] array. This utility should be run only once after the SPU has been booted. Diaginit cops the system and then loads the remainder of the Rings[] array.

FILES

None.

SEE ALSO

libscan(3D)
libscnlink(3D)

NAME

scn_util - hardware initialization and hardware clock status utility

SYNOPSIS

scn_util [-i] [-I] [-b] [-j address mode] [-s sdr0 sdr1 sdr2 sdr3 sdr4 sdr5 sdr6 sdr7]

DESCRIPTION

The **scn_util** utility is a Processor Test product which provides a common interface between CONVEXOS, I/O Software, and I/O Diagnostics to the C3800 Series hardware. This interface provides support for key hardware initialization and hardware clock status checking. Several options are available; however it is recommended that some of the options should be used in combination, and others should be used independently.

The following is a listing the options options supported by **scn_util** along with an explanation of each option:

–i Attempts to turn clocks on to the memory subsystem and to any installed CCU. Specifically, the function is broken into two distinct parts: memory clocks and ccu clocks. For memory, the clocks for the memory boards, the crossbar, the interface adapter boards are verified to be either be on (as they were initialized by mminit) or are in a state where they can be restored. The memory clocks are considered to be in a state where they can be restored if no harderrors are present in the system; and the free running clock for the memory boards and the interface adapter boards are still running. If either harderrors are present or the free running clocks are off, then access to main memory is impossible and **scn_util** will exit with a non-zero value. If the memory system is in a condition to have memory clocks enabled; then memory clocks are enabled, clocks for the installed CCU's are turned on, and **scn_util** exits with a zero status.

–I Returns the state of the clocks for the memory boards, crossbar, interface adapter boards and the CCU's. When the **-I** option is executed the value of the clocks for the memory boards, crossbar, interface adapter boards, and CCU's is compared to the value of the clocks saved during the execution of the **-i** option. If the clocks enabled by the **-i** options are still enabled, a 0 status is returned; otherwise, a non-zero status is returned.

–j address mode

Instructs **scn_util** to start the execution of the first available CPU in the system at the logical memory location specified by **address.**. If the **mode** parameter is zero, **scn_util** will initialize the active CPU's in the complex to a default set of control parameters and then start the clocks for those initialized CPU's. These default parameters are defined in the Diagnostic Configuration Database and can be altered by the utility **xsys_config**. If the **mode** parameter is non-zero, then the operator is prompted through a series of queries which will allow the alteration of the CPU control parameters before the active CPU's clocks are started. Currently the valid list of options and their default values are:

enable_dcache

This option identifies whether or not data writes are to be encached in the data cache. The initial default value for this option is one (enable_dcache set to TRUE.)

enable_dcache_degrading

This option identifies whether or not the size of the data cache will remain as a 16K data cache on crossing from ring four, or if the data cache will be resized to 4K data bytes. The initial default value for this option is one (enable_dcache_degrading set to TRUE.)

enable_scn_ovr

This option identifies whether or not the script file /diag/bin/scn_ovr is executed after system initialization, but before enabling processor clocks. This is a debug feature which allows in-house fine tuning of the system hardware initialization before it is placed into code. The initial default value for this option is zero (enable_scn_ovr set to FALSE.)

–s sdr0 sdr1 ... sdr7

Allows the ability to set the Segment Descriptor Registers (SDRs) located in Communication Register Set Zero to the specified values. There must be eight SDRs specified, otherwise an error will be returned. An exit code of zero is returned if the operation is successful; a non-zero is returned if the operation is unsuccessful.

–b Instructs **scn_util** to output the system configuration database to *stdout*. The data is ASCII information identifying the current system configuration. The format of this output consists of multiple **IDENTIFIERs** that are terminated by a semicolon. Each **IDENTIFIER** entry is as shown below:

IDENTIFIER(entry count,entry size,format) entry0,entry1,...entryN-1;

The entry count describes the number of entries that follow the closing parenthesis. The entry size is the number of bytes required to hold each entry. The format is the format of the entry. Formats can be either **d** for decimal or **x** for hex for numeric data. The string format specified with **s** defines a string terminated by a semicolon.

The **IDENTIFIER** is an ASCII mnemonic describing a specific aspect of the system. The currently supported **IDENTIFIERs** are:

CPUTYPE

Identifies the CPU instruction architecture. For the C3800 Series machine the defined CPUTYPE is 5.

MEMSTART

Identifies the starting physical address of memory. Under normal conditions, this value should be zero.

VIOPS

An array of 40 entries which identify with the value 1, which slots have VIOPs installed the the respective CCU slots.

HSPS

An array of 40 entries which identify with the value 1, which slots have HSPs installed the the respective CCU slots.

IDCS

An array of 40 entries which identify with the value 1, which slots have IDCs installed the the respective CCU slots.

TLIS

An array of 40 entries which identify with the value 1, which slots have TLIs installed the the respective CCU slots.

IEEE

Specifies if the machine supports IEEE mode arithmetic.

PCM

Specifies a 2048 array of entries which identifies those physical (2-Mbyte blocks) of main memory installed in the system. Note that for the C3800 Series, physical addressing are no longer tied to specific boards, but are determined by mminit to maximize the system interleave.

SERIALNUM

Specifies the 16 bit system serial number as read off the base I/O backplane. The upper four bits of the serial number represents the machine class and the 12 lower bits represents the actual serial number. Valid 16 bit serial numbers range from 41000 (decimal) to 45055 (decimal.)

REFMOD_ADDR

Identifies the base address of a 2-Mbyte block of main memory allocated to be used as the referenced and modified bits. Under normal operations this block will be the highest addressed 2M-byte block of physical memory whose associated pcm bit will be cleared.

MACHCLASS

Identifies the class of Convex machine. Currently defined machine classes include: 0=C1, 1=C1XE, 2=C210,C220, 4=C230,C240, 5=C201/C202, 10=C3800 Series.

PROCNUM

Identifies the CPU population map. This map is an eight entry array which identifies whether or not a CPU is installed in the system. A one identifies the CPU as installed and a zero identifies the CPU as not installed.

INTRINSICS

Identifies whether or not the machine supports microcoded intrinsic instructions. Under normal operations, this field will always be TRUE.

PARALLEL

Identifies whether or not the machine supports parallel CPU processing. Under normal operations, this field will always be TRUE.

UNDER_MASK

Identifies whether or not the machine supports vector under masks instructions. Under normal operations, this field will always be TRUE.

INTERLEAVE

Identifies the minimum interleave setup in the system. The value is drastically different from the C32XX and C34XX systems as the physical memory can be internally re-configured to group like boards together to improve the interleave. An example is a system which has 4 memory boards (three 512 Mbyte and one 128Mbyte) In this example, two of the 512 Mbyte boards will be in a 2 board interleave, and the remaining two boards will be in a one way board interleave. On each board there is 64 way interleaving. This value is multiplied by the minimum board interleave to get the system interleave. Hence the valid values for this entry is 64, 128, 256, and 512.

FILES

/diag/bin/scn_util

NAME

sfp – provides access to the boot mode parameter and the keyswitch setting.

SYNOPSIS

sfp [**-b mode**] | [**-k position**]

DESCRIPTION

The current keyswitch position and power-up boot mode are displayed if **sfp** is invoked with no arguments.

The power-up boot mode parameter can be set using the **-b** argument, along with one of the following values:

normal

the CONVEXOS CONSOLE window is made visible, and diaginit is executed in it. If diaginit(1) exits with a 0 status, a boot command is issued in the CONVEXOS CONSOLE window.

alternate

the CONVEXOS CONSOLE window is made visible, and diaginit(1) is executed in it. This prepares the Convex CPU for boot, but the boot command is not issued.

diagnostic

this is the lowest level of initialization. All that is done is that the CONVEXOS CONSOLE window is made visible.

The **-k** flag is used to determine if the keyswitch is in a particular position. The argument to this is one of: **OFF**, **LOCAL**, **REMOTE** (**CPU_ONLY** on HP SPU), or **SECURE**. If the keyswitch position matches the argument, sfp exits with a status of 0, if it does not match sfp exits with a status of 1.

SEE ALSO

xsfp(1)

NAME

softlog – archived memory soft error log file for *errintd*

SYNOPSIS

/diag/data/softlog_archive

DESCRIPTION

This file contains a log of corrected main memory system single-bit errors (i.e., memory read access errors corrected by the Error Correction Code (ECC)). The *errintd* utility stores this information in the Configuration Database (CDB). This information consists some header data and one entry per failed memory chips (up to the maximum limit - default is 60). The header information contains the following items:

- The date the current soft error logging was started
- A total error count
- A count of failed devices, i.e. the number of entries
- A count of errors not logged due to throttling by *errintd*

Each entry in CDB follows this format:

MB device bit stk first_fail last_fail address count

MB
is the number of the memory board containing the faulty device (0 - 7) and a letter indicating which half of board: o(dd) or e(ven).

device
is the coordinates of the faulty device (e.g. U002B1), coded as follows:

SCCCRR

where:

S	=	MC side: U is the gate array side and Z is the other side
XXX	=	failed device x-coordinate, eg. 002
YY	=	failed device y-coordinate, eg. A4

bit
is the bit that required correction. This only applies to the last error that occurred on this device.

stk
indicates the repeatability of the error. The letter "S" indicates that although the error is correctable, it could not be eliminated by 10 attempted memory scrub operations of the address under test (i.e., a bit is stuck at the address under test). This only applies to the last corrected error that occurred on this device.

first_fail
is the date the first error on this device occurred.

last_fail
is the date the most recent error on this device occurred.

address
is the address of the last single-bit error on this device. This only applies to the last corrected error that occurred on this device.

count
is the total number of single-bit errors (on this device) that have been corrected.

SEE ALSO

errintd(1d)
mm_sniff(1d)

NAME

sys_shutdown - power on the C3800 computer system

SYNOPSIS

sys_shutdown

DESCRIPTION

The **sys_shutdown** utility powers down all boards and bays in the system. It first checks the open status of each bay against the Configuration Database (CDB). If the state stored in the CDB does not match the actual bay status, the CDB is updated to match the actual bay status.

Next, all boards and bay power supplies (BPS) in all bays are powered down. The busses on all boards are turned off followed by turning off all BPS's (300V).

Sys_shutdown now pauses for 60 seconds to allow the cabinets to cool down.

Once the bays have cooled, each bay that is open is powered down. This includes closing the channel to the power daemons, pulling the BPC reset line and clearing the CDB entries. Each bay is verified that it is in a reset state. If it is not, the reset line to that BPC is pulled again to put it into the reset state.

All entries in the CDB are cleared to show that the system as being in the powered down state.

SEE ALSO

diaginit(1D)
powerdown(1D)
powerup(1D)

NAME

sysreset – reset the C3800 computer system

SYNOPSIS

sysreset [-c level] [-i level] [-m level] [-s] [-x] [-h cpu_mask [level]]
[-ccu ccu1,ccu2,...]
or
sysreset [-l level]
or
sysreset

where cpu_mask is defined as follows:

Bit	Board
==	===
0	CPU (or IA) in port 0
1	CPU (or IA) in port 1
2	CPU (or IA) in port 2
3	CPU (or IA) in port 3
4	CPU (or IA) in port 4
5	CPU (or IA) in port 5
6	CPU (or IA) in port 6
7	CPU (or IA) in port 7

-c level specifies that the CU board is to be reset at the specified level. Value for the reset level is either 0 or 2. Level 0 initializes the scan ring only. Level 2 reset includes level 0 plus initialization of the Communication Registers.

-i level specifies that the IA board (in the I/O bay) is to be reset at the specified level. Value for the reset level is either 0, 1, or 2. Level 0 pulls the backplane reset line to the IA boards, level 1 initializes the scan ring only (does not include RAMs), and level 2 initializes the scan ring plus the RAMs. The 3X (free-running) clock is enabled, i.e. it is running, following the IA reset.

-m level specifies that the MB boards are to be reset at the specified level. Value for the reset level is either 0 or 2. Level 0 initializes only the SYS ring and level 2 initializes the entire ring.

-s specifies that the scan engine is to be reset.

-x specifies that the XBAR boards is to be reset. There is only one reset level.

-h cpu_mask level specifies that the specified CPUs are to be reset to the specified level. If level is not specified then it defaults to 0. Level 0 initializes only the scan ring. Level 2 includes level 0, initialization of all caches/register files/RAMs, and loading the control stores on both the SP and VP. Level 3 is the same as level 2 except that the load of the control stores is verified.

-l level specifies that all boards (including all CCUs) are to be reset at the specified level. Values for the reset level are 0, 1, 2, or 3.

-ccu ccu1,ccu2,... specifies the CCUs to be reset. If no CCUs are specified then all CCUs are reset.

DESCRIPTION

The **sysreset** utility resets the computer based on boards and reset levels. Using **sysreset** requires the user to understand one thing: the actual reset levels are board dependent (i.e. some boards have multiple levels of resetting while others have a single default level).

Attempting to reset a board to a level higher than defined will cause that board to be reset to its maximum level. Specifying an undefined level will cause that board to be reset to the highest level less than the level specified, eg. resetting MB0 to level 1 (undefined) will reset it at level 0.

If the - option is specified then options and arguments will be read from standard input.

If the -s option is specified then the scan engine is reset regardless of what other options are specified.

If the -l option is specified then all boards are reset to the specified level. This option overrides the -c, -i, -m, -x, and -h options.

If options -c, -i, -m, -x, and/or -h are specified (without the -l option), only those boards specified are initialized to their specified levels. All other boards are untouched. For example, **sysreset -h 4** will only initialize the CPU in port 2.

The XBAR has only one level of reset, namely 0, whereas the other boards have several levels of reset. In other words, if the XBAR is reset then it will automatically be reset to level 0.

If no parameters are passed to the **sysreset** utility, all boards in the system will get reset to level 0. This is identical to **sysreset -l 0**.

If either **sysreset** or **sysreset -l level** is to be performed, the scan engine is initialized first (only if level is 2). The individual boards are then reset to the desired level by calling **init_boards()** with the correct board masks and desired reset levels. After execution of the **sysreset** utility, the clocks are stopped on all boards (except the IA 3X clock).

The **sysreset** command can be used in different ways to reset the desired boards. Several examples are shown below.

sysreset <RETURN>
: resets all boards to default level 0 (scan engine is *not* initialized).

sysreset -l 2 <RETURN>
: resets CU, IA, MB, SP, and VP boards to level 2; XBAR boards are reset to level 0.

sysreset -c 2 -i 2 -m 2<RETURN>
: resets CU, IA, and MB boards to level 2; CPUs are not affected.

sysreset -h 4 <RETURN>
: resets only CPU2 to level 0; all other boards are not affected.

sysreset -m 2 -h 3<RETURN>
: resets CPUs 0 and 1 to level 0 and the MB boards to level 2.

SEE ALSO

initall(1D)
init_boards(3D)
cu_default(3D)

BUGS

The [-] option does not yet work.

There is currently no way to reset IA boards residing in CPU-IO expansion bays.

NAME

wi – workstation interface device driver

DESCRIPTION

The file */dev/wi* is the "connection" between the user and the C3 SPU for the purpose of catching system interrupts and interfacing to the mbs system.

It is used by user programs to issue **ioctl(2)** calls that contain the specific requests to handle the interrupts or the mbs message interface.

The basic *ioctl* calls use the structure defined in */diagr/include/hp/wi.h*:

```
struct wi_cntl {
        int     signo
        int     interrupt
        union {
                struct  msgs_entry_def *msg;
                caddr_t         *rdwrt;
        } buf;
        int     size;
};
```

The *signo* field defines the signal to send for a WI_CATCHINT or a WI_MSG_INIT command. The *interrupt* field defines the interrupt that should be caught when the WI_CATCHINT command is issued. The *buf* union contain pointers to areas that serve different functions depending upon the command. The *msg* variant is used for the WI_MSG_SEND, WI_MSG_RECV_CPY, WI_MSG_RTN_CPY and WI_MSG_FREE_CPY. The *rdwrt* variant is used for the WI_MSG_DATA_WRT and WI_MSG_DATA_RD commands.

The management function is accessed via numerous *ioctl* calls of the general form:

```
#include <wi.h>

ioctl (fid, code, arg)
struct wi_cntl  *arg;
```

The applicable codes can be broken down into 3 functions, the functions associated with MBS (WI_MSG_INIT, WI_MSG_SEND, WI_MSG_RECV_CPY, WI_MSG_DATA_WRT, and WI_MSG_DATA_RD), and those associated with catching interrupts (WI_CATCHINT):

WI_CATCHINT

Will send the signal *signo* to the calling process when the interrupt specified by *interrupt* occurs. The list of valid catchable interrupts on the C3 SPU system are:

IV_HARDERR	NCU hard error (non-maskable)
IV_NIASOFT	NCU NIA soft error (non-maskable)
IV_NMBSOFT	NCU NMB soft error (non-maskable)
IV_SCANACC	NCU scan memory access error (non-maskable)
IV_SCALHALT	NCU scalar halt (non-maskable)
IV_SCANPAR	NCU scan memory parity error (non-maskable)
IV_CLKPAR	NCU clock generator parity error (non-maskable)
IV_XCLPAR	NCU XCL parity error
IV_SYSB	NCU system interrupt 11
IV_SYSA	NCU system interrupt 10
IV_SYS9	NCU system interrupt 9

IV_SYS8	NCU system interrupt 8
IV_MODEM_SWITCH	NWI modem keyswitch position change
IV_SPU_SWITCH	NWI SPU keyswitch position change

When the signal is delivered to the process, the interrupt will be disabled. It is up to the user process to reenable the cause of the interrupt. The WI_CATCHINT command will only catch the first occurance of an interrupt. So if multiple occurances need to be caught the WI_CATCHINT command needs to be reissued after each occurance.

WI_MSG_INIT

This command initializes the interface for the user process, and must be called before any of the other routines. The parameter *signo* is interpreted as a signal number used to notify the process that a message is pending.

WI_MSG_SEND and WI_MSG_RECV_CPY

These commands allow user processes to send and receive messages using the lower-level message routines. Both of these commands require the *buf.msg* pointer to point to the user's buffer that will (or does) contain the message.

The WI_MSG_SEND command copies the message from the user buffer and sends it to the desired destination, as determined from the *me_dst_prc* and *me_dst_id*, which the user must have filled in the message. The WI_MSG_RECV_CPY command copies a message received for the calling process into the user's buffer. After receiving a message using the WI_MSG_RECV_CPY command, it must eventually be returned or freed using the WI_MSG_RTN_CPY or WI_MSG_FREE_CPY commands. The WI_MSG_RTN_CPY routine returns the message to it's previous sender using *rtn_msg*. The WI_MSG_FREE_CPY routine frees the message by calling *pq_fm_free*. Both of these routines require a pointer to the user's message buffer as an argument.

The signal specified in the WI_MSG_INIT command is generated only when the first incoming message is received. Multiple messages can, however, be queued for the process. Therefore, WI_MSG_CPY must be called repeatedly after each signal until the WI_MSG_RECV_CPY command returns MBS_NO_MSG to guarantee that all outstanding messages have been received. This enables subsequent generation of another signal when a new incoming message arrives.

The WI_MSG_DATA_WRT routine writes data from the user's buffer to a system-defined single-page buffer. This buffer is shared among all processes using the msg_oper interface. The *msg_data_rd* routine reads data from the system-defined single-page buffer into the user's buffer. Both routines require the address of the user's buffer and the number of bytes to be transferred as arguments.

DIAGNOSTICS

The *ioctl* calls for management requests return -1 when errors occur and 0 if no error occurs. **Perror(3)** can be called to print the error code found in *errno*. The following error codes are returned as a result of MBS related calls:

[MBS_ERROR]

Internal error in the queue structure.

[MBS_SUCCEED]

The desired operation was successful.

[MBS_NO_MSG]

Returned by the WI_MSG_RECV_CPY command if no incoming messages are queued up for the calling process. Returned by the WI_MSG_SEND routine if the system pool of message buffers is exhausted.

[MBS_QUE_LCKD]

The queue was locked for an extended period of time.

FILES

/dev/wi

SEE ALSO

ioctl(2) perror(3)

NAME

wndw - window allocation management system

DESCRIPTION

This section describes both a particular special file, and the general nature of the system interface.

The file */dev/wndw* is the "connection" between the user and the system for the window management system. It is used by user programs to issue **ioctl(2)** calls that contain the management system request.

The basic *ioctl* calls use the structure defined in */diagr/include/hp/wndw.h*:

```
struct wndw_req {
        char    *wndw_addr;
        char    *mm_addr;
        int     mem_size;       /* in bytes */
        int     tas_value;
        int     bus_addr;
        int     bus_log;
        int     odd_pattern;
        int     even_pattern;
        int     compare_status;
        int     op_code;
};
```

The *wndw_addr* field defines the window address for the operation to be performed on or is the address returned from an allocation request. *Mm_addr* and *mem_size* fields define the Convex main memory address and size for the request. *Tas_value* is the field that contains the value of Convex main memory before a test-and-set or test-and-clear operation was issued. The *bus_addr* and *bus_log* fields are used for error reporting and contain the 68000 bus error access address and the SPU bus error log bytes, respectively. *Odd_value* and *even_value* are used as the values to place in odd and even memory locations respectively when using the fill operation. *Compare_status* returns the results of the WNDW_COMPARE command.

The management function is accessed via numerous *ioctl* functions of the general form:

```
#include <wndw.h>

ioctl (fid, code, arg)
struct wndw_req  *arg;
```

The applicable codes are:

WNDW_ALLOC

Allocate a block of windows that will correspond to the Convex main memory area defined by the fields *mm_addr* and *mem_size*. The address of the block is returned in the *wndw_addr* field.

WNDW_ALLOC_RW

Same as WNDW_ALLOC, but will set up the map registers for the area defined in *mm_addr* and *mem_size* to be read/write access via the window address in field *wndw_addr*.

WNDW_FREE

Will deallocate the map registers defined by the *wndw_addr* and *mem_size* fields. The map registers will be returned to the free pool invalid.

WNDW_RW

Will set up the map registers in read/write mode for the area defined by the *wndw_addr* and *mem_size* fields. The field *mm_addr* will define the first physical address in Convex main memory. The map registers will be mapped as a sequential block of physical memory.

WNDW_RW_OSRESERVE

Same as WNDW_RW except that the user's process is not forced to have previously allocated the window passed in *wndw_addr*. This permits a user process to access the windows that were reserved by the kernel for OS use. This command is intended for the expert user (i.e. the OS group), and should not be used by the average user.

WNDW_TAS

WNDW_TAS_PERM

Will enable only the "TAS" mode of operation for the map registers corresponding to the *wndw_addr* field. Then a single byte will be read based on the address in the field *wndw_addr*. If the operation is WNDW_TAS, the map register will then be returned to the mode it was set to prior to the *ioctl* call. If the operation is WNDW_TAS_PERM, the map register is not returned to its prior mode.

WNDW_SCRUB

WNDW_SCRUB_PERM

Will issue a memory "scrub" cycle on the address in field *wndw_addr*. Upon completion of this operation, the map register is returned to its prior mode if the operation was WNDW_SCRUB. The "scrub" bit is left set if the operation was WNDW_SCRUB_PERM.

WNDW_TAC

WNDW_TAC_PERM

Will enable only the "TAC" mode of operation for the map registers corresponding to the *wndw_addr* field. Then a single byte will be read based on the address in the field *wndw_addr*. If the operation is WNDW_TAC, The map register will then be returned to the mode it was set to prior to the *ioctl* call. If the operation is WNDW_TAC_PERM, the map register is not returned to its prior mode.

WNDW_FILL

Will allow the quick initialization of system memory via the memory test circuitry on the NCU. *Odd_pattern* is the pattern that is written into odd memory addresses while *even_pattern* is written to even memory addresses. The locations in memory are specified by *mm_addr* which specifies the start memory location and *mem_size* which specifies the number of bytes in memory to fill. The address must be even and the count must be a multiple of 8 bytes. If these parameters are not of the correct multiple, they will be rounded down to comply with these rules. The *op_code* parameter specifies the operation performed by the memory test hardware on the NCU between succesive memory writes. The possible op_codes and their actions are:

```
WNDW_OP_HOLD    0    /* hold */
WNDW_OP_INCR    1    /* increment */
WNDW_OP_LEFT    2    /* shift left */
WNDW_OP_COMP    3    /* complement */
```

WNDW_COMPARE

Will allow the comparison of a range of memory starting at *mm_addr* for *mem_size* bytes with the values contained in the *odd_pattern* and *even_pattern*. The address must be even and the count must be a multiple of 8 bytes. If these parameters are not of the correct multiple, they will be rounded down to comply with these rules. The *op_code* is the the

operation that is applied to the even and odd patterns between successive memory accesses. The *compare_status* field contains a 0 if the comparison is true throughout the specified memory range. If the comparison fails, the *compare_status* is set to 1. The failing address will be returned in the *mm_addr* and the expected values will be returned in the *odd_pattern* and *even_pattern* fields. The actual value can be obtained by reading the failing memory location. The *op_code* parameter specifies the operation performed by the memory test hardware on the NCU between succesive memory reads. The possible op_codes and their actions are described in WNDW_FILL above.

WNDW_BUS_ERR

Will return in fields *bus_addr* and *bus_log* the address of the last 68000 bus error for this process and the error log bytes for that 68000 bus error, respectively.

WNDW_MAP_GET

Will return in *mm_addr* the contents of the map register that corresponds to the window address specified in *wndw_addr*. The specified window address does not necessarily have to have been mapped by the calling process. If an invalid window address is specified in *wndw_addr*, the *ioctl* will return -1, and *errno* will be set to EINVAL. In this instance, the contents of *mm_addr* will be zero.

DIAGNOSTICS

The *ioctl* calls for management requests return -1 when errors occur and 0 if no error occurs. **Perror(3)** can be called to print the error code found in *errno*. The following error codes are used:

[EINVAL]

Invalid command or argument

[ENFILE]

Overflow of process id table. Too many processes are using */dev/wndw* maps simultaneously.

[ENOMEM]

No enough map space to satisfy request.

[EFAULT]

Error accessing the map registers on the NCU.

[ENODEV]

Performing any operation when the NWI or NCU is not available. Trying to perform any non-existant operation on this device (read,write,select).

FILES

/dev/wndw

SEE ALSO

ioctl(2) perror(3) wndw_errinit(3d)

NAME

xcdb_browser – provides access to the diagnostic configuration database.

SYNOPSIS

xcdb_browser [X-options]

DESCRIPTION

The diagnostic configuration database is the central location where all of the information on system configuration and status is stored. There is a great deal of information stored in this database - **xcdb_browser** provides easy access to this information. The first window that is presented, is used to selectively display information. Additional window can be displayed to update items in the database, and to write the text from text window to a file.

The initial window provides several buttons at the top, these are used to activate the functions described by the button name. In the cases where a button name is followed by "...", activation of that button will bring up yet another window. Just below the buttons is a row of toggle buttons, these are used to set parameters that control the output that appears in the text window. The next row contains a pattern entry field, and finally at the bottom is a scrollable text output window.

VIEWING

One way to view a parameter is to select the **dump** pushbutton. This gets every token in the database and its value, and writes them to the text window. Even though the scrollbars provide an easy means of looking through the text, the shear size of the output buffer makes it difficult to find any particular item.

If any portion of the item name is known, the preferred method of searching for information is to enter a **pattern** string. This string is a regular expression that the item names are compared to. Any items matching will be displayed.

The toggle buttons that control the text that is displayed are:

auto_purge
: when on, the text window is purged each time a new buffer is displayed. Turning this off causes additional output to be appended to the end of the current buffer. Be aware that when this is off, there is no limit to the size of the buffer - except of course the size of virtual memory. To clear the text window manually, select the **purge** button on the top row.

show **type**
: causes the report to show the data type of the value stored.

show **num_elements**
: causes the report to show the number of elements stored in the data base under that name.

show **resource**
: causes the resource associated with the item to be shown.

UPDATING

The update window is displayed when the user selects the **update** button. This window provides an easy method for updating values in the database. Caution is advised here, this software has no notion of whether an item should be changed - it does what you tell it to and expects you to know what you are doing.

The **pattern** is used to enter a regular expression that describes the items that are to be updated. This pattern generates a list if matching item names that is displayed in the bottom section of the window. Individual items can be removed from the list by selecting items that are not of interest (click mouse button one in the list - when the button is released, the highlighted item is expelled from the list).

Once the list contains ONLY the items that are to be updated, enter a new value in the **new value** test entry field. Each item named in the list will be updated to the new value, and the list will be deleted.

WRITING

The text portion of the main window can be saved to a text file. To do this, activate the **write** on the top row of the main window. This causes the write_text window to be displayed. This window provides a means to name the file that the test will be written to. This window has several regions of interest. The value of the **File Filter** text entry box controls what is shown in the **Files** list. The File Filter string can include wild cards. Selecting a file from the Files list places the name in the **Selection** box. The value of the Selection box can be edited to select an existing file or create a new file. Typing return in the Selection box, or selecting the **OK** button causes the text to be written.

X options

xcdb_browser has no command line options of its own, but it accepts all of the standard X Toolkit Intrinsics. The man page X(1) describes all of these options, but for your convenience these are the most commonly used options:

-display display_name
This option specifies the name of the X server to use (overrides DISPLAY environment variable). It has the form WORKSTATION_NAME:DISPLAY_NUMBER.

-geometry geometry_specification
geometry is of the form WIDTHxHEIGHT+XOFF+YOFF. In the case of this program, the x and y offsets will probably be used without the width and height: +XOFF+YOFF.

-iconic
start life as an icon.

-bg color, **-background** color
color for the window background.

-bd color, **-bordercolor** color
color for the window border.

-fg color, **-foreground** color
color to use for text or graphics.

SEE ALSO

cdb_browser(1)
X(1)

NAME

xdiag - is a X11 based program used to execute diagnostic tests.

SYNOPSIS

xdiag

DESCRIPTION

Xdiag provides a X11 based user interface for all diagnostics ported to use the Convex Test Interface (CTI). The set of tests that are currently available under this interface are those found listed when the **Test** pulldown menu is activated on the top menu bar in the **xdiag** main window.

In general terms, xdiag provides controls and displays for all of the parameters pertinent to the test that is selected. There are two classes of parameters, the CTI parameters and the test specific parameters. The CTI parameters are applicable to all of the tests. They relate to message levels and subtest sequencing. These parameters are displayed and controlled in the windows that become available immediately after xdiag is started. When a test is selected, it will have a set of additional parameters that are specific to the running of the test. Although there are several ways to provide access to these parameters, the usual method is that pulldown menus are added to the top menu bar in the **xdiag** main window. In these menus are selections that activate windows and functions specific to the test.

The default windows that **xdiag** always provides are described below. This manual page does not cover the diagnostic tests that run under **xdiag**. For that information, see documentation for the test.

MAIN WINDOW

The main window is the window that appears at startup. It contains a menu bar, a row of buttons, a text entry field labeled **sequence**, a command entry widget, and a scrolling text window. The menubar contains buttons and pulldowns that activate xdiag functions. The menu bar labels that are followed by "..." will create yet another window when activated.

The **Files** pulldown contains file and text control functions. It also contains the **exit** selection. The **Text** menu item is a cascade button, dragging the cursor off of the right side of the selection will cause the **Text** menu to appear. This menu provides functions that are performed on the text in the scrolling text region of the main window. That window can be purged, read and written. Activating **read...** or **write...** entries provides a window for selecting the file that the text is to be read from or written to. The remaining Files entry, **logging...**, is mostly for debugging. It provides controls for writing the messages passing between the CTI and xdiag to a text file.

The **Tests** menu contains the names of all of the tests that can be run under xdiag. If you select one of theses tests, the previous test (if there is one) is aborted, and the new test becomes active. This usually causes changes in the display - if windows were open pertinent only to the discarded test, they are closed and destroyed.

The **Info** menu contains commands that are useful for getting information from the test. The output from these requests goes to the main text window. The selections are:

sub - list of subtests *after* dependencies are applied. This is the list that tests will be selected from when a sequence list is executed. In the resulting list, Subtests with asterisks have dependencies.

sub_all - list *all* of the subtests in the test. Subtests shown with asterisks have dependencies.

dep - list the dependencies. Dependency information comes from the test information file. This list shows the subtests that have dependencies. Each dependency consists of a parameter name, and a range of values that invalidate that subtest.

class - list the classes

summary - list parameters relating to test state

global - list global parameters used to control CTI features

test_par - list the parameters specific to the test

par - list all of the parameters and their current value for the active test.

com - list the commands for the active test

The remaining buttons are **Status...** and **TestControl....** These buttons activate windows of their own. Status shows the current state of the test. Test control contains numerous toggle buttons that are used to control the actions of the sequencer before and after each subtest is run. These windows will be discussed in greater detail below.

The row of buttons immediately below the menu bar are the *test control buttons.* They control the starting and stopping of the subtest sequencer. The commands are:

reset - resets all counters and initializes the test

run - runs the specified test sequence from its starting point

cont - continues the sequence

INTERRUPT - pauses an executing subtest, or aborts a paused subtest

rerun - reruns the last started subtest

skip - skips the next subtest

abort - terminates a paused subtest

quit - stop the test, xdiag does not exit

The **sequence** entry field is a text entry field. The string entered here describes the order that the tests will be run in. The default sequence string is "()", which means to run all of the subtests. A sequence string can contain any of the following:

```
      default sequence: ()
      subtest ordering: 1,10,5
      ascending ranges: 1-100
     descending ranges: 100-1
  sequence repetition: 100(1,10,5)
              nesting: 100(1,10,5,100(1-10))
              classes: c1,c3
```

The **Command Line Entry** region is a two box affair that provides command entry, a command history, and command editing. The top box contains previously entered commands, which when selected appear in the bottom box. The contents of the bottom box can be edited. When the command is correct, hit return to activate it. Any command that the current test will understand can be entered here. The valid command names can be found by typing com. Parameters can be changed by typing the parameter name followed by the new value. Any name or value that contains whitespace must be enclosed in double quotes. The valid parameter names can be found by typing par.

Status

The **Status** window contains a row of buttons, followed by the status region. The buttons are:

Pause Status... - brings up a window showing status information that is pertinent to a paused state.

Force - requests an update of the information in the Status window. This button is only useful if **updates** are turned off.

updates - information is sent up as it changes.

The status region contains *no entry fields*. The names of the fields are mostly self-explanatory, so I won't list them. The **CTI state** could use a little explanation though, the states are:

NO TEST - no test is executing, the information from the previous test remains on the display for reference.

STARTUP - the test has been reset and is ready to run.

RUN - the sequencer is running subtests.

PRE-CLEANUP - a subtest is paused prior to any subtest cleanup being done.

POST-CLEANUP - the test is paused after cleanup of the prior subtest.

COMPLETE - test is paused upon completion of the entire subtest sequence.

Test Control

This window has numerous toggle buttons for control of CTI features. These buttons are activated if the center of the button differs from the background. The buttons are:

event_logger - log test events to the system event logger

looping - loop on the current subtest forever

test_summary - display test summary info

pass_messages - display subtest passes

fail_messages - display subtest failures

init_messages - display subtest initializations

run_messages - display subtest runs

cleanup_messages - display subtest cleanups

runtime_messages - display individual subtest runtimes

pause_before_subtest - pause after subtest initialization but before the subtest actually runs.

pause_on_fail - pause when a failure is detected but before subtest cleanup

pause_on_pass - pause once a subtest passes but before subtest cleanup

pause_between_subtest - pause after subtest cleanup

pause_when_complete - pause when the subtest sequence has completed

timeout_multiplier - used to extend the fixed timeout values

failure_threshold - aborts a subtest after this many failures within the subtest.

SEE ALSO

X(1)

NAME

xevent_browser – an X11 based program for selectively displaying messages from the event log.

SYNOPSIS

xevent_browser [X-options]

DESCRIPTION

The program *xevent_browser* is used to view messages logged to the event log. Either all of the messages in the log can be displayed, or various filters may be employed to select messages of interest. Filters exist for time, severity, message number, message source, and message type. Regular expressions may also be used to select messages, BUT, since the message must be formatted before this filter can be applied it is the least efficient, and slowest method for selecting messages.

This program provides easy access to all of the functionality available in the program *event_browser*. Input widgets are provided for each parameter that *event_browser* accepts. There are three categories of input: modes, commands, and filters.

FILTERS

The filters are useful for limiting the number of events that need to be formatted and displayed. The initial screen displays the simplest filters, clicking the Filters button causes an additional window to be displayed that gives access to the rest of the filters.

The message, source, type, and grep filters have both inclusive (+) and exclusive (-) lists. If an event matches an exclusive filter, then the event is not displayed. The inclusive filters have a small added complexity; if there are no inclusive filters in the list, then the inclusive filter is ignored. This in essence **includes everything if nothing is included.** If one or more characteristics is in the inclusive list, then only the events that match are displayed.

Below the message, source, and type lists are text entry blocks where the number or the name of an applicable value for that type of filter can be entered. For instance under source, you could enter either "2" or IO, and the number 2 would be added to the source filtering list. When the focus is in one of these filter text entry blocks, the selection window is set to that topic. The selection window provides a description of what the numeric values are. To view the description of a particular value, type the number into the text entry box to the left or the up/down arrows and hit return. The description will appear in the one line window immediately below it. The up down buttons can be used to sequence through the values, and the select button will add the value to the appropriate list.

Message

Identifies the subject of the message. An example of a message is number is 301, which translates to "NWI bus error encountered".

Source

This gives an idea where the event came from. An example is 0, which translates to DIAG, hence messages that match this come from diagnostics and diagnostic utilities.

Type Identifies the general subject of the event. An example of a type is 3, which translates to a Soft Error.

Grep Simply a regular expression. This filter is the least efficient of all of the filters, as the event must be formated before the filter is applied. As in the other lists, to remove an item click the mouse on the list entry, and it will be removed.

Time Using before and after, a segment of events can be filtered. The time can be modified by typing new values over the existing times in the before and after text entry windows. The set button provides a simple cascade menu that allows the the time to be set to one of the days of the preceding week, or the current time.

Severity

Provides toggles for each of the possible severities, when the center of the toggle box

differs from the background color, the item is selected.

MODES

There are two modes that affect the overall operation of the event_browser, tail mode, and report mode. Report mode is the default startup mode. In this mode no events are displayed until the report button is pressed. When the report button is pressed, the event_browser reads the log file, applies any filters that may be set, and displays the resulting events in the text portion of the window. The scrollbars may then be used to look through the events. Note that when a report is being generated, only the Abort and Quit buttons will have affect.

Tail mode is entered by clicking the tail toggle. All of the filter controls remain active, and can take input. In this mode, the information displayed does not come from the log. Instead it comes directly from errlogd, the program that writes the events to the log file. As events occur, xevent_browser is notified, the filters (except before and after) are applied to the events, and when appropriate, the events are displayed. In this mode, the before and after filters have no meaning, so they are not used.

COMMANDS

There are several pushbuttons on the main window. In general clicking one of these buttons causes a command to be issued (except when a name has "..." following it, these buttons cause another window to appear). Here are the buttons and the commands that they issue:

report causes a report to be generated.

abort stops the report generation.

quit causes xevent_browser to exit.

tail this is a toggle that sets the mode to either tail or report. For more detail see the section on modes above.

Save... causes a file selection box to appear. From that window a file can be selected or created containing all of the text that is in the main text window.

Clear causes the text in the main text window to be purged.

Auto clear is a toggle that activates auto clear. When activated, the main text window is cleared each time the report button is activated.

Help display topic oriented help on this window.

Filters... causes the filters window to be displayed.

Refresh requests a refresh of all of the displayed parameters.

reset sets all of the parameters and filters to their startup values.

X options

xevent_browser has no command line options of its own, but it accepts all of the standard X Toolkit Intrinsics. The man page X(1) describes all of these options, but for your convenience these are the most commonly used options:

-display display_name
This option specifies the name of the X server to

use (overrides DISPLAY environment variable). It has the form WORKSTATION_NAME:DISPLAY_NUMBER.

-geometry geometry_specification
geometry is of the form WIDTHxHEIGHT+XOFF+YOFF. In the case of this program, the x and y offsets will probably be used without the width and height: +XOFF+YOFF.

-iconic
start life as an icon.

-bg color, **-background** color
color for the window background.

-bd color, **-bordercolor** color
color for the window border.

-fg color, **-foreground** color
color to use for text or graphics.

SEE ALSO
event_browser(1)
X(1)

NAME

xpowermon - X-based power system monitor

SYNOPSIS

xpowermon

DESCRIPTION

Xpowermon is the X-based interface to the **powermon** utility. The user selectable parameters include the target, interval and the log_file.

The target is the bay or board to be monitored. The following options are available:

bay[0-4]
xbar
ccu[0-39]
mb[0-7]
sp[0-7]
ia[0-8]
vp[0-7]
xiop[0-8]
cu

The interval is the number of seconds to delay between each power system status inquiry and screen update. If none is selected the default is 20 seconds. The first update is done immediately (after *Start* is selected). Each subsequent update is delayed the specified number of seconds.

The log_file, if one is specified, contains all the temperature and voltage status information displayed by **xpowermon**. Please note that this file can grow quite rapidly depending on the interval selected. The default is no log_file.

Once a target (and optionally the interval and log_file) has been specified, select the *Start* menu item to begin querying the power system for status and displaying this information on the screen.

Querying the power system and display the status is performed until the *Stop* menu item is selected. Selecting the *Start* menu item will restart the process.

When selected, the *Help* menu item displays the available targets and a brief description of **xpowermon**.

To terminate the **xpowermon** utility, select the *Exit* menu item which produces an pulldown menu for verification of the exit. Select the *Quit* menu item to actual terminate **xpowermon**. If *Quit* is not selected then **xpowermon** continues executing.

SEE ALSO

powermon(1D)

NAME

xsfp – controls power-up reboot of ConvexOS, monitors the keyswitch position, and controls the SPU printer. Never kill this process.

SYNOPSIS

xsfp [X-options]

DESCRIPTION

This program is started by xinit. If it is killed, xinit terminates, the X server is killed, and all of the X clients are killed. The reason for this touchy behavior is that this process enforces security when the keyswitch is in the SECURE position. To restart X, login as diaguser, and type 'xinit'.

The **xsfp** monitors and displays the system keyswitch position. The Convex CPU is initialized automatically when the keyswitch is transitioned from OFF to any of the other positions. The setting of the boot mode parameter determines what level of initialization is completed. This parameter is set by clicking mouse button one while the pointer is over the desired selection. There are 3 possible boot modes:

normal OS
: the CONVEXOS CONSOLE window is made visible, and diaginit(1) is executed in it. If diaginit(1) exits with a 0 status, a boot command is issued in the CONVEXOS CONSOLE window.

alternate OS
: the CONVEXOS CONSOLE window is made visible, and diaginit(1) is executed in it. This prepares the Convex CPU for boot, but the boot command is not issued.

diagnostic
: this is the lowest level of initialization. All that is done is that the CONVEXOS CONSOLE window is made visible.

cxts controls boot
: in this mode CXTS controls the boot process.

For convenience, there is a set of buttons across the top of the window that provide control of the boot process. The buttons are for **osclean, boot multi, boot single**, and **boot mini**. Because these buttons make it very easy to accidentally kill a running system, they check to see if boot is already running. If boot is running, **xsfp** pops up one of those annoying little "do you really want to do this" boxes. This bit of checking is what causes the time lag between the button being pressed and the appearance of the command in the CONVEXOS CONSOLE window.

There is one more button on the display, **Console Printing**. This button toggles the state of the printer. It does not show the state of the printer. When the printer is on, anything that appears in the CONVEXOS CONSOLE window is printed. If text in another window is to be printed, toggle the printer on. Then, in the CONVEXOS CONSOLE window, type 'cat > /dev/null'. Cut the desired text and paste the text into the CONVEXOS CONSOLE window. Be sure to kill the cat.

Note that the lpr command can be used to print text files regardless of whether the text from CONVEXOS CONSOLE is being routed to the printer.

INNER WORKINGS

The **xsfp** starts and maintains programs, the main one being the CONVEXOS CONSOLE. In this way CONVEXOS CONSOLE can immediately restart the CONVEXOS CONSOLE if it should ever die. This arrangement is hard coded into xsfp for the CONVEXOS CONSOLE, additional programs can be started and maintained in a similar manner by adding the appropriate entries to the **xsfptab** file.

The **xsfptab** is a text file that lives in **/diag/data**. It contains the values of parameters that xsfp uses, and the list of executables that xsfp will start. The parameters that are stored here are:

_BOOT_MODE
: this is where xsfp and sfp store the value of the boot mode. Either program will correctly write this value, and using those programs is the correct way to change this line. If for some reason you must edit this by hand, the possible values are "diagnostic", "alternate", or "normal".

_REMOTE_RESTRICT
: this parameter is used to stop input to the SPU workstation when the keyswitch is in REMOTE (CPU ONLY on an HP SPU). If the value of this parameter is 1, NO input is allowed when the switch is in REMOTE. If the value of this parameter is 0, input is not restricted when the keyswitch is in the REMOTE position. This parameter is changed by editing this file, there is no programmatic interface.

_RMTDIAG_OWN
: When its value is 1, rmtdiag sessions will not be usurped. The default is 0, in which case a new rmtdiag login takes control away from a previous rmtdiag session.

As mentioned briefly above, **xsfp** can be made to start and maintain programs. This is done by adding entries to the **xsfptab** of the following form:

[_UNIQUE] [_RESTART] [KILL] full_executable_path arguments

The meanings of the optional flags are:

_UNIQUE
: the program will be the only copy. How this is accomplished depends on _KILL. If _UNIQUE is specified and _KILL is not specified, xsfp only starts and maintains the program if it does not already exist.

_KILL If _KILL is specified, and a copy already exists, it is killed before xsfp starts a new one.

_RESTART
: if the child dies, xsfp will attempt to restart it.

There are several combinations of flags and situations that will make starting and restarting of processes not work as intended. The most obvious is if _KILL is specified and someone other than diaguser started a process with the same name, in which case the startup will fail because xsfp can't kill the preexisting process. Another situation occurs if both _UNIQUE and _RESTART are specified. If for some reason a copy is already running, and _KILL is not set or if the executable can't be killed, the existing executable will remain running. The result being that xsfp does not get a signal if the process dies, and the restart will not happen. The moral of the story is that this is not bulletproof, so use it with caution.

Notice that the name of the executable must be a full pathname, no searching is provided. The process is executed with the arguments provided on the line.

SEE ALSO
: sfp(1)

NAME

xsys_config – shows the configuration of CPUs and memory boards.

SYNOPSIS

xsys_config [X-options]

DESCRIPTION

This presents a concise snapshot of the CPU and memory board status. The data used in this display is stored in the configuration database.

Selecting the *bank_interleave...* button displays a submenu that shows the current bank interleave selections. For each possible grouping of NMBs (1, 2, 4 or 8) there is a separate bank interleave value that may be specified. The initial value for all selections is 16 bank interleave. These values are scanned into the NIA and NSP boards during system initialization.

The top portion of the window contains status information, the widgets in the top region of the window do not take input.

The toggle buttons that appear in the lower portion of the window can be changed by user input. Toggling these buttons affects the system configuration - changes take affect at the next ConvexOS boot.

CPUs Available

CPUs are shown as available if they are installed and diaginit was able to initialize them.

MBs Available

memory boards are show as available if they are installed and diaginit was able to initialize them. The amount of memory on the board is shown by putting its mark in the appropriate row. There is also a field that shows the total memory installed.

firmware/ucode revision and **Oscillator Period**

are self explanatory.

CPUs Allocated

CPUs allocated to ConvexOS are marked. Activating one of these toggle buttons changes their allocation. The change takes effect at the next boot of ConvexOS.

MBs Allocated

memory boards allocated to ConvexOS are marked. Activating one of these toggle buttons changes their allocation. The change takes affect at the next boot of ConvexOS.

Force MB Size

Allows the size of each memory board to specified as a way to maximize memory interleaving. These selections are stored in the Configuration Database, and only **mminit** uses them. The *Default* selection denotes that the size as calculated by **mminit** is used. The other selections denote the size to use for that memory board - either 128MB, 256MB or 512MB. A memory board cannot be force to a larger size than physically possible, e.g. a 128MB size memory board cannot be made a 256MB size memory board.

dcache enable

This key defines whether or not data fetches are obtained from memory or from the dcache (provided the memory location has been encached.) With the value set to 'on' data will be retrieved from the data cache if it is valid, otherwise the data will be retrieved from memory. With the value set to 'off' the dcache is bypassed and all data is retrieved from memory.

CPU hard error enable

This key specifies whether CPU hard errors are enabled on the NCU or not. If the value is 'on' then CPU hard errors are enabled which results in a system halt after which all error informations saved. If the value is 'off' then CPU hard errors are disabled from halting the system. This mode of operation is called Automatic Processor Recovery.

dcache resize

This key defines whether or not the size of the data cache is reduced from 4k to 1K on inward ring crossings from ring4. The value 'on' defines the dcache to be resized and the value 'off' defines the dcache to not be resized.

scan override

This key defines whether or not a debug script named 'scn_ovr' is invoked after system initialization, but prior to turning on cpu clocks. This is a debug feature used to tune system parameters to optimize machine performance and to aid in debugging machine failures.

X options

xsys_config has no command line options of its own, but it accepts all of the standard X Toolkit Intrinsics. The man page X(1) describes all of these options, but for your convenience these are the most commonly used options:

-display display_name
This option specifies the name of the X server to use (overrides DISPLAY environment variable). It has the form WORKSTATION_NAME:DISPLAY_NUMBER.

-geometry geometry_specification
geometry is of the form WIDTHxHEIGHT+XOFF+YOFF. In the case of this program, the x and y offsets will probably be used without the width and height: +XOFF+YOFF.

-iconic
start life as an icon.

-bg color, **-background** color
color for the window background.

-bd color, **-bordercolor** color
color for the window border.

-fg color, **-foreground** color
color to use for text or graphics.

SEE ALSO

xcdb_browser(1)
cdb_browser(1)
X(1)
mminit(1)

C3800 Series boot process

B

This appendix describes the ConvexOS boot process for C3800 Series computers and gives a procedure for a manual boot.

B.1 Service processor boot process

The C3800 Series service processor unit (SPU) automatically boots itself on power up. The SPU boot process invokes the X-Window environment and places a default window display on the SPU monitor. Refer to the *CONVEX Service Processor Unit Service Guide (3800 Series)* for information on the SPU boot process.

B.2 The automatic ConvexOS boot process

The ConvexOS boot proceeds according to the following steps:

1. The `diaginit` utility checks, initializes, and activates the C3800 distributed power system.
2. The `initall` utility initializes all system circuit boards and calls the `sysreset` and `mminit` utilities.
3. The `sysreset` utility initializes the system and loads microcode into the control stores.
4. The `mminit` utility sets up the memory system.
5. Final steps in the boot process depend on the type of `boot` command used.

B.3 Booting manually

Invoke the following utilities and commands to manually boot a C3800 Series computer:

```
osclean

diaginit

boot single

/etc/preen
```

The above commands boot the computer to single user mode. Press **CONTROL-d** to go to multiuser mode.

Glossary

The following terms are defined as they are used at CONVEX. Standard acronyms are also included. Boldfaced terms within a definition are defined in separate entries.

B

b
See *byte.*

bit
A binary digit.

boot
The procedure by which a program is initiated the first time. Typically, a boot is performed when power is first applied to the processor.

BPC
Bay power controller.

BPS
Bay power supply.

byte (b)
The number of contiguous bits starting on an addressable byte boundary. In CONVEX machines, a byte is 8 bits. See *bit.*

C

channel control unit (CCU)
A circuit board that controls data flow between the PBUS and peripheral devices.

central processing unit (CPU)
The central processing unit (CPU) is that portion of a CONVEX computer that recognizes and executes the instruction set.

central cabinet
The cabinet where the crossbar logic resides. It is physically located at the center of CONVEX C3800 Series systems.

compiler
A software tool used to compile the source code of a high-level language, such as FORTRAN, into object code.

ConvexOS
The new name for the CONVEX version of the UNIX operating system.

CONVEXOS CONSOLE
The X-Window that receives `/dev/console` output and input for ConvexOS.

CPU
See *central processing unit.*

D

data type
The way in which bits are grouped and interpreted. For processor instructions, the data type identifies the size of the operand and the significance of the bits in the operand.

destination
The location where an operand will be stored at the end of an operation.

diagnostic mode
The workstation mode (or state) when ConvexOS is not booted.

E

electromagnetic interference (EMI)
Electrical signals produced by impingement of external, unwanted electromagnetic waves on an electronic circuit.

electrostatic discharge (ESD)
The release of static electricity from a charged object to a grounded object.

EMI
See *electromagnetic interference.*

EPROM
See *erasable programmable read-only memory.*

enabled
A condition or bit is said to be enabled when it is true or set to a one.

erasable programmable read-only memory (EPROM)
A random access memory that holds data written into it when power is removed from the memory, but can be erased using some special technique. Types of EPROMs include electrically erasable EPROMs (EEPROMs) and ultraviolet erasable EPROMs (UV EPROMs).

ESD
See *electrostatic discharge.*

expansion cabinet
A secondary cabinet designed to house peripheral computer equipment, for example, tape drives, disk drives, controllers, and so forth. See also *central cabinet.*

F

firmware
Computer programs that are embodied in a physical device that can form part of a machine. Also, software that resides in read-only memory.

flag
A status bit that is generally used to indicate the results of an operation. The results are in the form true or false.

FORTRAN
A high-level software language used mainly for scientific applications.

H

hard error
A noncorrectable data error.

I

IPH
Input power harmonizer.

K

kernel
The core of the operating system, which manages process creation and deletion, scheduling, and other high-level,

system-wide features. In CONVEX operating systems, the kernel resides in ring 0.

L

load
An instruction that transfers data, usually from memory to a register in the CPU.

M

megabyte (Mbyte)
1,048,576 bytes.

N

NCU
CPU utilities circuit board.

NIA
Interface adapter circuit board.

NMB
Memory circuit board.

NSP
CPU scalar processor circuit board.

NVP
CPU vector processor circuit board.

O

outer cabinet
The outer cabinet is the generic term for a cabinet that is positioned against any of five sides of the central cabinet. An outer cabinet may be built to function as any of the following: CPU bay, I/O bay, I/O expansion bay, or a CPU-I/O expansion combination bay.

P

PP
Power pallet.

PPC
Power pallet controller.

processor cabinet
A cabinet designed to hold one or more central processing units (CPUs). See *central processing unit (CPU)*.

prompt
A character or character string sent from a computer system to a terminal to indicate to the user that the system is ready to accept input. Typical CONVEX prompts are `(fp)`, `#`, `%`, `:`, and `spu`.

protection
A mechanism provided by hardware and software to ensure that one user is protected from another user or that a user does not perform an unsafe computation.

R

read
A memory operation in which the contents of a memory location are copied to another part of the machine.

reduced instruction set computer (RISC)
An architectural concept that applies to the definition of the instruction set of a processor. A RISC instruction set is an orthogonal instruction set that is easy to decode in hardware and for which a compiler can generate highly optimized code.

root directory
The base directory in ConvexOS from which all other directories stem, directly or indirectly.

root file system
The file system associated with the root directory.

runtime
A software module that is referenced as a procedure. A runtime represents a required function that is not directly supported by the hardware, but is required by the software.

S

small computer system interface (SCSI)
Industry standard interface to disk drives.

soft error
Correctable single-bit memory error. May further be defined as transient (nonreproducible) or stuck (reproducible). Recorded in the softlog.

SPU
Service processor unit. On the C3800, SPU refers to the service processor workstation.

SPU console
The X-Window on the SPU monitor screen that receives `/dev/console` output from process running on the SPU. ConvexOS output to `/dev/console` does not go to this window.

SPU tape cartridge
The magnetic tape cartridge containing the SPU programs, files, and utilities.

SPU tape drive
The tape drive containing the SPU data.

SPU OS
The CONVEX-developed, UNIX-based software that controls the SPU. It directs certain supervisory functions on CONVEX computers.

superuser
The ConvexOS and SPU OS term for the system manager.

system boot
The procedure that loads and activates ConvexOS on the C3800 System. System boot must always occur after workstation boot.

SWIP
SPU workstation interface parallel circuit board. Installed in the SPU as a parallel bus interface between the SPU and the C3800 computer.

SWIS
SPU workstation interface serial circuit board. Installed in the SPU as a serial bus interface between the SPU and the C3800 computer.

system console
The CRT or printer/terminal that serves as a communication device between the system manager and CONVEX computers. In the C3800 Series computers, the SPU is the system console.

system status display
A two-digit LED display located on the front panel of CONVEX C130, C210, and C220 computers. It is used to display hexadecimal status codes transmitted by the SCM.

U

universal asynchronous receiver-transmitter (UART)
An integrated circuit chip used in communications to an RS-232 port.

UNIX
An operating system developed by UNIX System Laboratories, Inc.

W

word
Four bytes (32 bits), the fundamental width of items in the CONVEX family of computers.

workstation boot
The procedure that loads and activates the operating system on the SPU workstation.

write
A memory operation in which a memory location is updated with new data.

X

XCL
Crossbar control logic circuit board.

XIOP
Extended input-output processor circuit board.

XPBE
Crossbar power circuit board, even side.

XPBO
Crossbar power circuit board, odd side.

XRTE
Crossbar return data circuit board, even side of memory.

XRTO
Crossbar return data circuit board, odd side of memory.

XS0E
Crossbar send data circuit board 0, even side of memory.

XS0O
Crossbar send data circuit board 0, odd side of memory.

XS1E
Crossbar send data circuit board 1, even side of memory.

XS1O
Crossbar send data circuit board 1, odd side of memory.

X Window System
A system of visualizing multiple processes developed by the Massachusetts Institute of Technology and the Digital Equipment Corporation.

Index

D

E

F

G

H

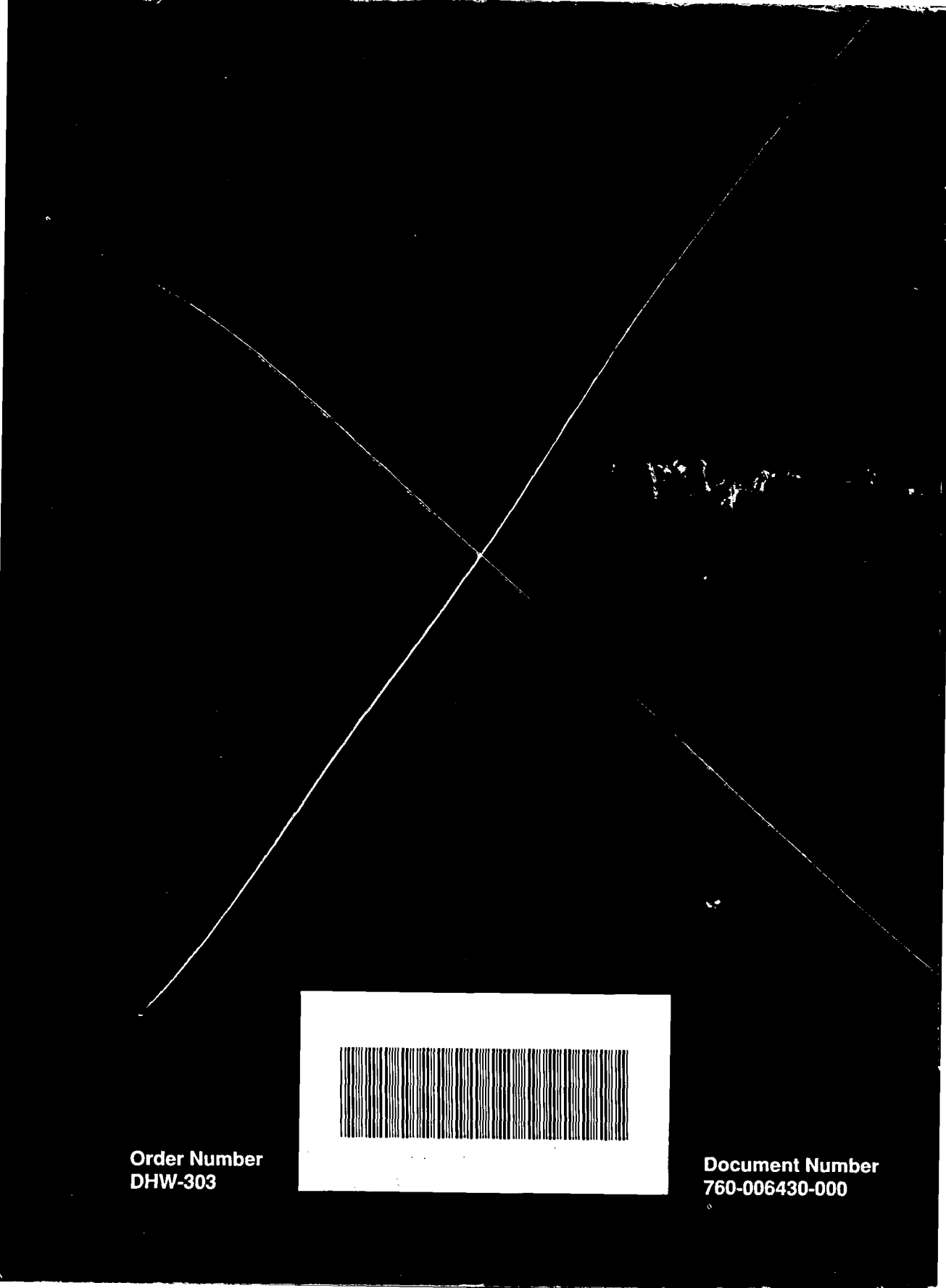
T

U

V

W

X



Order Number
DHW-303

Document Number
760-006430-000