CONVEX Diagnostic Utilities Manual (C200 Series)

Document No. 760-000830-001

Third Edition, Rev. 1
November 1990

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX Diagnostic Utilities Manual
(C200 Series)
Order No. DHW-082
Third Edition, Rev. 1

Printed in the United States of America

Revision Sheet

CONVEX Diagnostic Utilities Manual
(C200 Series)

Edition	Document No.	Date	Description
Third, Rev. 1	760-000830-001	November 1990	This addendum adds the new *rita_perr.1d* and *val_perr.1d* manpages
Third	760-000830-000	December 1989	This edition updates the *softlog.5d* manpage and contains two new manpages: *fs.1d* and *idcfmt.1d*.
Second, Rev. 1	760-000650-203	May 1989	This addendum revises the *config_chk.1d* manpage, adds the new *version.1d* manpage, removes the obsolete *diskfmt.1d* manpage, and adds a new Appendix B, "Reporting Problems."
Second	760-000650-202	December 1988	This edition reflects a change in the name of the document from *CONVEX Diagnostic Utilities Manual (C130, C210, C220)*. Added six new man pages (*commreg.1d*, *config_chk.1d*, *get_defects.1d*, *mkdiag_db.1d*, *security_clear.1d*, *softlog.5d*), updated eight man pages (*cpureg.1d*, *DB_diskfmt.5d*, *errintd.1d*, *mm.1d*, *mm_sniff.1d*, *scn_ring.1d*, *scn_util.1d*, *scnlink.1d*). Updated the permuted index and table of contents. Also added an updated Iscan chapter.
First	760-000650-201	April 1988	First release, Version 1.0

THIS PAGE INTENTIONALLY LEFT BLANK

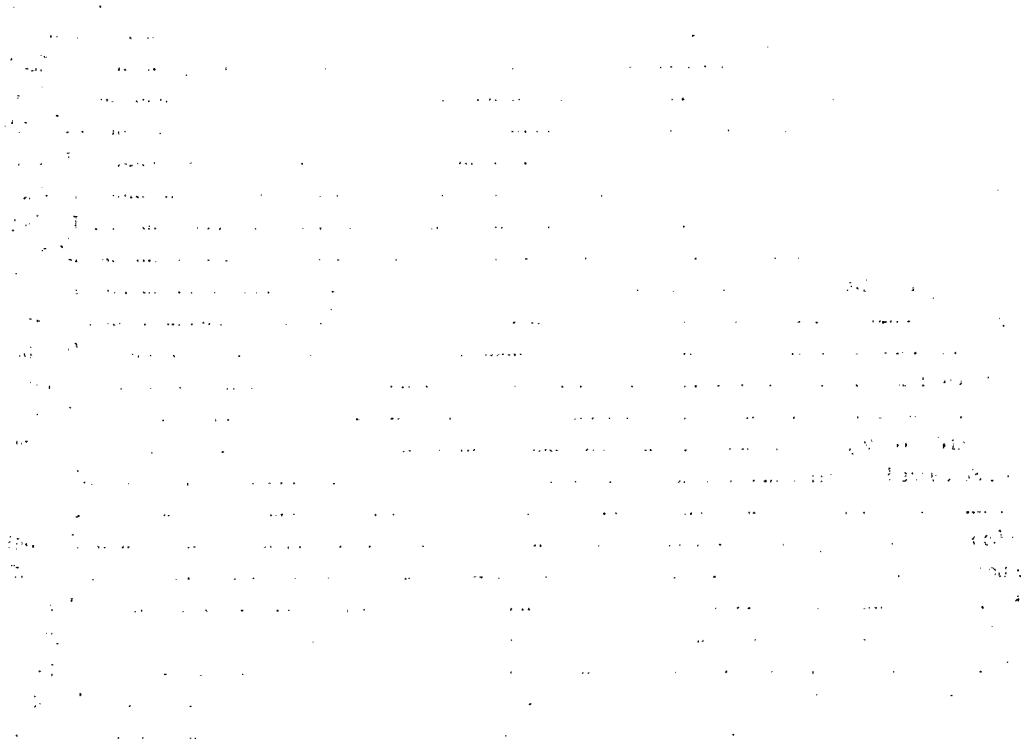


Table of Contents

Appendixes

List of Tables

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

Alphabetical Listing of Man Pages

1D. Diagnostic Commands

5D. Diagnostic File Formats

Permuted Index – Utilities

Alphabetical Listing of Man Pages

1D. Diagnostic Commands

5D. Diagnostic File Formats

Preface

Purpose and Intended Audience

The *CONVEX Diagnostic Utilities Manual* is the fourth of four volumes in the *CONVEX Diagnostics Documentation C200 Series.* The other volumes include the following:

- *CONVEX Diagnostics Master Index (C200 Series)*
- *CONVEX Processor Diagnostics Manual (C200 Series)*
- *CONVEX PBUS I/O System Diagnostics Manual (C200 Series)*

This document presents the diagnostic utilities for the CONVEX C200 Series computers. The material presented describes the features of the Service Processor operating system and contains all diagnostic utilities for the the CONVEX C200 Series computers.

The *CONVEX Diagnostic Utilities Manual (C200)* is a reference tool for CONVEX personnel who use the diagnostic utilities, CONVEX customers who perform their own maintenance, and the CONVEX diagnostics sustaining staff.

Scope

This manual applies to CONVEX C200 Series computers.

Outline

This manual contains information on the Service Processor UNIX environment, a detailed explanation of the Diagnostics Shell (Dshell) and the Interactive Scan (Iscan) utilities, diagnostic utilities, and diagnostic file formats. This manual is divided into the following chapters:

Chapter 1. Diagnostic Environment—Introduces and explains the diagnostic and Service Processor UNIX operating system environments as well as describes the directory and file structure

Chapter 2. Diagnostic Shell (Dshell)—Introduces and explains the Dshell

Chapter 3. Interactive Scan (Iscan)—Provides a detailed introduction and explanation of Iscan utility

Chapter 4. Diagnostic Utilities—Describes publicly-accessible diagnostic commands in alphabetical order

Chapter 5. Diagnostic File Formats—Contains detailed explanations of all pertinent diagnostic file formats

Appendix A. Glossary—Lists CONVEX-preferred technical nomenclature and jargon, including standard abbreviations

Appendix B. Reporting Problems—Contains information concerning how to use the *contact* facility to report problems

Notational Conventions

All entries are based on a common format, not all of which will always appear:

- The NAME subsection lists the exact names of the commands and subroutines covered under the entry and gives a short description of their purpose.
- The SYNOPSIS summarizes the use of the program being described.
- The DESCRIPTION subsection discusses in detail the subject at hand.
- The FILES subsection gives the names of files that are built into the program.
- A SEE ALSO subsection gives pointers to related information. Each cross-referenced command is boldfaced and contains the filename plus a number in parentheses. The number is parentheses is the chapter number for the cross-reference. Any cross-reference with a (1), (2), (3), (4), (5), (6), (7), or (8) will not be found in this manual. References to (1d) and (5d) are found in this manual.
- A DIAGNOSTICS subsection discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.
- The BUGS subsection gives known bugs and sometimes deficiencies. Occasionally the suggested fix is described.

A few conventions are used, particularly in the "Commands" subsection:

- **Boldface** words are considered literals, and are typed just as they appear.
- Square brackets ([ ]) around an argument indicate that the argument is optional. When an argument is given as "name," it always refers to a filename.
- Ellipses (...) are used to show that the previous argument-prototype may be repeated.
- All CONVEX illustrations have an illustration catalog number at the bottom right-hand corner that is for CONVEX use only.
- "Service Processor (SP)" is used generically to represent either the Service Processor 2 or Service Processor 4, depending on the system configuration under test.
- "CPU Utility Board(s)" is used to represent either a CPX or a CUE/CUO combination, depending on the system under test.
- "PBUS Interface Board(s)" is used to represent either a PIA, a PI2 installed in the PIY slot or two PI2s installed in the PIX and PIY slots, depending on the system configuration under test.

Warnings

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

WARNING

Warnings highlight procedures or information necessary to avoid injury to personnel. A 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

Cautions highlight procedures or information necessary to avoid damage to equipment, loss of data, or invalid test results. A 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 useful information that is supplemental in nature. A note may immediately precede or follow the information that is being highlighted.

Associated Documents

The following is a partial list of other manuals or books that may provide more detailed information on the topics presented in this manual:

- *CONVEX Processor Diagnostics Manual (C200 Series)*, Order No. DHW-081
- *CONVEX PBUS I/O System Diagnostics Manual*, Order No. DHW-008
- *CONVEX Processor Operation Guide (C100 Series, C200 Series)*, Order No. DHW-015

Ordering Documentation

To order the most current version of this or any other CONVEX document, use the product number. If the product number is not known, order by the exact title. In some situations, the most current version may not be desired. To receive a specific version of a manual, order the manual by its document number, or part number, which can be obtained by contacting the local CONVEX office or by calling the Technical Assistance Center.

The product number for this manual is DHW-082.
The document number for this manual is 760-000830-000.

CONVEX documents can be ordered by mail by sending a request to:

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

Technical Assistance

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

- From all locations in the continental United States, call 1(800)952-0379.
- From locations in Alaska, Hawaii, and Canada, call 1(214)497-4379.
- From all other locations, contact the nearest CONVEX office.

Electronic Mail

The Hardware Documentation Group has an email address for documentation comments. Use this service to give us a quick response mechanism for special documentation questions that need to be addressed immediately. For technical questions, contact the Technical Assistance Center, as described previously. To use email response service, just send mail addressed to:

```
cnvxhwdoc@convex.COM
```

We will read your comments and give you a personal reply.

What to Include in an Email Message

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 name of manual that is being critiqued
- The chapter and page number in question
- The comment

Reader's Forum

If you wish to mail your comments to us, please use the form at the end of this manual and list the document page number with your questions and comments. Thank you.

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

I would like to thank the following people for their contributions to this document:

- Technical contributors: Tom Ford, Steve Gardner, Dave Rotheroe, Jeff Venters, Faron Wickey, Tony Jones
- Document review team: Art Clark, John Clark, Don Davis, Jeff Jones, David Massey, Craig Reed, Randy Stiles
- Hardware Documentation staff: George Metafas, Larry Bonura

Without the efforts of all the aforementioned, this document would not have been possible.

David Massey, Lead Writer
CONVEX Hardware Documentation

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1
Diagnostics Environment

1.1 Overview

CONVEX system diagnostics consist of a suite of test programs designed (except where noted) to execute under the Service Processor operating system, SPU UNIX. These programs use the capabilities of the Service Processor to test the operation of one or more of the functions of the system and to report any errors detected. All diagnostics in this manual are intended to be executed off-line, that is, while CONVEX UNIX is not being executed by any of the Central Processing Units (CPUs) in the system.

The Service Processor, together with SPU UNIX and the various diagnostic utilities and test programs, comprise the CONVEX diagnostic environment. For more information about the diagnostic environment, refer to the "Diagnostic Environment" chapter in the *CONVEX Processor Diagnostics Manual (C200 Series)* or the *CONVEX Processor Diagnostics Manual (C1, C120)*, depending on the architecture of the machine under test. This chapter describes the hardware and software components of this environment and is intended to provide the background necessary to fully utilize the capabilities of the CONVEX processor diagnostics.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2
Diagnostic Shell (Dshell)

2.1 Introduction

The Diagnostic Shell (Dshell) is a command interface program that runs on the Service Processor (SP2 or SP4) on the CONVEX C200 Series computers. Most diagnostics on these machines are interfaced through the Dshell. This chapter describes the Dshell procedures for executing tests in manual mode and provides the basic information needed to execute diagnostics via the Dshell. It describes the various options available for controlling data logging and test repetition.

This chapter is particularly helpful to field service and manufacturing personnel who are responsible for board or system verification, or board or system debugging.

2.2 Dshell Overview

The Dshell has two basic functions:

- Selecting diagnostics for execution
- Selecting test options
  - Pause on a failure or at the beginning or end of any specific subtest
  - Loop on a specific subtest or on a given set of subtests
  - Select subtest execution order
  - Direct test output to a file or to the screen (or both) to monitor the test as it runs or to analyze test results later
  - Select long or short error messages, or turn messages off
  - Execute user-created command scripts

2.3 Dshell Commands for Manual Mode

This section explains the commands available under the Dshell.

To access the Dshell, enter **dshell<CR>** at the `spu>` prompt. All Dshell commands can be entered at the Dshell prompt (:).

2.3.1 *access*

The *access* command (*!*) is used to access, or *fork*, a UNIX shell to execute the command that follows *!*. Any single command available under SPU UNIX can follow the *!*. Once the command

has executed, program control reverts to the Dshell. The standard form of this command is:

! [unix_command]

2.3.2 *exit*

There are four different forms of exits available in the Dshell. They exit from tests back to the Dshell command level and from the Dshell to the UNIX command level. The following table shows the standard forms of these commands:

Table 2–1, *exit* Commands

OPTION	FUNCTION
exit (or *e*) or *quit* (or *q*)	Immediate termination of the Dshell process and any test processes that may have been forked

^C	Returns user to the Dshell command level if no subtest is running. If subtest execution has started, the following test menu appears:
	`0 continue test`
	`1 abort current test`
	`2 abort current subtest`
	`3 abort and pause at end of current subtest`

^B	Immediately terminate the Dshell and any associated active processes. Core is dumped.

Test programs that interface with sensitive portions of the hardware protect certain portions of code from interrupts. This protection is recognized by the *exit*, *quit*, and *^C* commands. In addition, some test programs execute a clean-up routine before terminating, which leaves the hardware in a known state. The *exit*, *quit*, and *^C* commands allow this clean-up routine to execute.

The *^B* command is a dirty, nonmaskable exit that pays no heed to what is currently being executed. Sensitive code is not protected from *^B*. In addition, *^B* never allows the clean-up routine to execute. As a result, the hardware may, and probably will, be left in an indeterminate state if this command is used. Use *^B* only as a last resort. System initialization should immediately follow *^B* execution.

Using *^C* after a test command has been entered, but before the Service Processor has successfully forked the test process, may not kill the test process.

2.3.3 *help*

This command causes a standard *help* menu to be displayed. The menu describes the correct command syntax for each Dshell command and gives a terse description of what each command does. The standard form of this command is:

help

The *-h* option is available on all other commands to display that command's help menu. The standard form of this command is:

[*command*] *-h*

2.3.4 *status*

The *status* command generates a report on the current state of the Dshell command options. This report gives the name of each flag, its current value, and an explanation of its current effect. The standard form of the *status* command is:

status

The Dshell keeps the current flag state in an external working file, named *testflags*, that is located in the current working directory. This file is deleted by the Dshell on all exits except *^B*.

2.4 Manual Mode of Execution

In the manual mode, the sequencing and parameters for diagnostic testing can be determined either directly or indirectly via Dshell. Manual execution allows selection of the conditions under which subtests are executed. Using this mode of operation returns low-level pass or fail data.

2.4.1 Dshell Commands for Manual Mode

This section explains the commands available under the Dshell for the manual mode of execution.

2.4.1.1 *log*

Normally, Dshell diagnostics terminate after a single failure. The *log* command provides a mechanism for specifying the number of failures that will be allowed to occur before a test or subtest terminates execution.

The standard form of the *log* command is:

log [*options*]

The options available include invoking the command at both the subtest (*-s*) and test (*-t*) levels, as well as selecting the number of failures to be allowed. The default setting is *log off -s -t*.

The following table lists *log* options:

Table 2–2, *log* Options

OPTION	FUNCTION
log off	Disable all multiple failure logging
log off [*-s*] [*-t*]	Disable previous log entries at the subtest or test level and terminating test after first failure
log -s [*nn*]	Allows the tests to run until *nn* subtest failures have been logged
log -t [*nn*]	Allows subtests to run until *nn* failures have been logged

NOTE

The *-s* flag does not work for all Dshell tests. Check the appropriate chapter to determine whether the *-s* flag of a particular test is enabled.

2.4.1.2 *loop*

The *loop* command causes the Dshell to repeat the execution of a test or subtest, as indicated in the following table:

Table 2–3, *loop* Options

OPTION	FUNCTION
loop off	Disables all looping
loop off [*-s*] [*-t*]	Disables subtest or test looping
loop -s	Loops on every subtest
loop -s [*nn*]	Loops on subtest *nn* if it is executed
loop -t	Loops on entire test

Subtests or tests continue executing until the loop mode is disabled. The standard form of the command is:

loop [*options*]

As with the *log* command, options include *-s* and *-t*, which enables the command to execute at either the subtest or test level, and the subtest number, if looping on a specific subtest is desired. The default setting is *loop off -s -t*.

To continue test execution after looping on a subtest, enter a *pause* command before test execution. This allows for disabling subtest looping when desired (*loop off -s*) and continuing normal test execution.

As an added feature, if test looping is enabled, the Dshell records the pass or fail result of each test iteration. When ^*C* is executed and test looping terminates, the results are summarized and written to the selected output.

2.4.1.3 *msgs*

The *msgs* command enables or disables different levels of test, class, and subtest result messages. The following table lists the options available for all tests:

Table 2–4, *msgs* Options

OPTION	PURPOSE
msgs off [*-f*] [*-s*] [*-t*]	Disables all or specific test messages
msgs -f [*long*] or [*short*]	Enables long or short failure messages
msgs -s	Enables subtest result messages
msgs -t	Enables test result messages

The basic format of the *msgs* command is:

msgs [*options*]

The default setting is *msgs -f long -s -t.*

2.4.1.4 *pause*

The *pause* command returns program control to the Dshell:

- At the beginning of all or specific subtests
- When a failure is encountered in all or specific subtests
- At the end of all or specific subtests

The following table shows how the available options may be used singularly or in combination:

Table 2–5, *pause* Options

OPTION	FUNCTION
pause off	Disables all pauses
pause off [*-f*] [*-b*] [*-e*]	Disables specified pauses
pause -f	A pause if a failure is encountered during execution of any subtest
pause -f [*nn*]	A pause if a failure is encountered during execution of subtest *nn*
pause -b	A pause at the beginning of each subtest that is executed
pause -b [*nn*]	A pause at the beginning of subtest *nn* if it is executed
pause -e	A pause at the end of each subtest that is executed
pause -e [*nn*]	A pause at the end of subtest *nn* if it is executed

The standard form of the *pause* command is:

pause [*option*] [*nn*]

where *nn* signifies the number of a specific subtest.

The default setting is *pause off*.

Omitting the argument *nn* causes a pause to be executed for each subtest (where applicable).

Each of the variations of *pause* can be set up either for all subtests or for a particular subtest.

When a pause occurs, program control is returned to the Dshell command level. Any UNIX command or diagnostic utility can then be executed by using the *access* command (*!*). When failure analysis or UNIX command execution has been completed, entering **<CR>** causes the test sequence to be re-engaged.

> **NOTE**
>
> The test has no knowledge of access commands occurring. If the machine state is destroyed, it is gone.

If an *-e* pause is enabled and a failure occurs, it is possible that the pause will not be taken. This happens when multiple failures per test (*log -t*) are not enabled, or when the current failure is the last failure permitted by the count entered with a *log -t* command.

2.4.1.5 *test*

The *test* command executes specific tests, and displays test, class, and subtest menus, as indicated in the following table:

Table 2–6, *test* Options

OPTION	FUNCTION
test	Presents a menu listing all tests available under the Dshell
test [*testname*]	Executes all subtests for the test named
test [*testname*] *-c*	Presents the class menu for the test named; user prompted to select class(es) for execution
test [*testname*] *-s*	Presents the subtest menu for the test named; user prompted to select subtest(s) for execution
test [*testname*] *-s* [*subtest list*]	Executes subtests in the order specified
test [*testname*] *-c* [*class list*]	Executes subtests (within the classes listed) in the order specified

To access a menu only, enter

test<CR>

at the Dshell prompt (:). This command displays a menu of all test available within the current working directory, as well as the tests located in */mnt/test*. The following menu is displayed:

Figure 2–1, Dshell Working Directory Menu

```
<CR>     Display next screen of test menu
p        Display previous screen of test menu.
t        Display top page of test menu.
b        Display bottom page of test menu.
?        Display help menu.
q        Leave the menu.
```

At the menu, a specific test can be executed or additional menus displayed. For example, for a specific subtest menu, answer the prompt with

test [testname] -s<CR>

where *testname* is the name of the test to be executed (refer to Table 2–6, *test* Options). The string *-s* is a flag that causes the subtest menu to be displayed. Similarly, the string *-c* causes the class menu to be displayed. The previous menu commands function in the class and subtest menu levels as well.

The standard form of the *test* command for test execution is:

test [*testname*] [*options*]

where *testname* refers to the name of the test to be executed. The *options* (*-s* and *-c*) enable various operations with test classes and subtests to occur.

To redirect input and output, use the following notational conventions:

- < — Causes input to be taken from the file listed immediately after the symbol
- > — Causes standard output to be directed to the file (but not the screen) listed immediately after the symbol
- +> — Causes standard output to be appended to both the following file and to the screen

To use the options for specific Dshell commands (*log, loop, pause,* etc.), enter those commands before entering the specific test to execute (or enter the name of the script file to execute). For example, to run *dev4400* subtests until three failures have been logged with *pause off*, enter

log -s 3<CR>
pause off<CR>
test dev4400<CR>

This causes *dev4400* to begin execution and run until three failures occurred or until test completion.

Once the colon prompt is displayed, any new Dshell command can be initiated.

< filename, > filename, and *+> filename* may be appended to the end of any of form of the *test* command. However, *>* and *+>* are mutually exclusive.

Default test execution is defined to be execution of each subtest that is available in a test, once, in ascending order.

When entering a test command of the type shown in the last two cases in Table 2–6, *test* Options, it is possible to arrange the execution order of the subtests or classes specified. It is also possible to execute multiple iterations of specific subtests or classes, or groups of subtests or classes. This is best illustrated by the examples in the following table:

Table 2–7, Subtest Configurations

OPTION	FUNCTION
-s 1	Subtest 1
-s 1,5	Subtests 1,5
-s 1-5	Subtests 1,2,3,4,5
-s 2(1-5)	Subtests 1,2,3,4,5,1,2,3,4,5
-s 1,5,3(5,4)	Subtests 1,5,5,4,5,4,5,4

> **NOTE**
>
> Subtest and class specification syntax is identical.

2.4.2 Dshell Script Files

The following listing typifies a Dshell script file that will execute Convex CPU Instruction Set diagnostics. For the purposes of this example, assume the name of the file is *D_novec*:

```
pause off
pause -f
log off
log -t 999
msgs -f long
msgs -t
msgs -s
test cpu4000 -s 10-499,800-919 <T_ring0 +>cpu4000.0
test cpu4000 -s 10-499,800-919 <T_fault_icache +>cpu4000.0
```

Several points should be noted:

1. The *pause off* command disables all pauses that were enabled before script execution.
2. The *pause -f* command enables the "pause on fail" capability. If a failure occurs, test execution halts, and the user is prompted, enabling a service person to pursue the cause of the failure.
3. The *log off* command disables previously enabled multiple failure logging.
4. The *log -t 999* command instructs the Dshell to allow multiple subtest failures per test without terminating the test.
5. The *msgs* commands set up long-failure, test, and subtest messages, respectively.
6. Two *test* commands are executed. Both cause subtests numbered 10-499 and 800-919 to be executed. Test command input is taken from two script files: *T_ring0* and *T_fault_icache*. Test results are output to both the console and to the file *cpu4000.0*. The +> option appends results to the end of the file, leaving previous contents of the file undisturbed.

If this command file is present in */mnt/test* on the Service Processor, simply enter the Dshell from SPU UNIX, and enter

<D_novec<CR>

This entry invokes the execution of all the commands contained in *D_novec*.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3
Interactive Scan (Iscan)

3.1 Overview

Interactive Scan (Iscan) is a diagnostic tool designed to interactively display the bit values throughout the architecture of the CONVEX C200 Series computers. The naming and format of bit locations displayed by Iscan is not determined by the user, but is part of the Iscan diagnostic database software. Iscan commands allow users to group, display, or manipulate these fixed bit sequences to obtain bit status information for booting, diagnostics, fault isolation, and testing. A description of the scan ring concept begins with a definition of rings and fields, a diagram of scan ring bus architecture, and a description of scan ring field names.

3.2 Scan Ring Concept

A *field* is a combination of 1 to 32 bits. It is the smallest meaningful combination of bits recognized by Iscan. The bits in a field usually are pulled from a similar physical or functional area of a board. By inputting data into the bits of fields and monitoring the outputs, Iscan acts as a valuable diagnostic tool.

A *ring* or *scan ring* is a series of fields grouped together. A scan ring comprises fields that are always pulled from the same board. The fixed bit locations of a scan ring reside in serial shift registers, and are later shifted out of these registers when being read by Iscan.

Ring names usually correspond to board types, and boards usually correspond to backplane slots. This is why ring names are sometimes referred to as *slots*. However, certain slots may house boards other than the board type they describe, and certain boards may contain more than one ring. These situations are the exception.

Two commands are useful in becoming familiar with Iscan ring and field names. Entering **list** displays the name of all rings that are present in a particular configuration. The names of empty or nonexistent board slots are not displayed. Entering **get**, followed by a ring name, displays the value of all the fields within a ring. The last character in every ring name is always a colon.

The definition of Iscan fields, their format, and ring specifications are inherent in the variable names used in Iscan, and the associated diagnostic database. These Iscan field characteristics are not defined by Iscan statements but rather by their names in the scan database. This method of defining the scan fields and associated information is required for compatibility with the C-library calls.

All ring definitions and the associated field definitions are generated in an automated fashion by the *scan_builder* program directly from the scan ring description file. In this way, the ring and field names are fixed before Iscan is run. These names are fixed by the names found in the *scan_builder* files used by the scan library, and generated by *scan_builder*.

The screen mode format may be specified while in Iscan, or a specification file may first be created outside of Iscan which defines the desired screen mode display field. This file is referred to as a *script*, and is optional when invoking Iscan, but is usually used when invoking the screen mode. The script specifies a screen name that later will be used while in Iscan to initiate the screen mode. Any number of separate scripts may be created and used. Iscan is invoked by entering the following:

iscn [*script1 script2 script3...*]

There are two basic kinds of specifications used frequently in Iscan: synonym specifications and screen specifications. Both resemble the struct declaration in the C language. Specify the synonyms first, and then the screens, because the screen definitions depend on synonym specifications.

3.2.1 Iscan Architecture

The following figures illustrate the broad extent of the architecture used in Iscan and the buses used to carry Iscan information. They are helpful in understanding the scan rings available and their dependence or independence on each other

While different rings on the same board use the same backplane data path to communicate with the Service Processor, the following figures contain separates lines for each type of ring, in an effort to show their independency.

An example of the dependence and independence illustrated in the figure is the memory subsystem. While all normal and log memory scan rings use the same physical data path to get data from or send data to the Service Processor, the normal and log scan rings are independent of each other. A log scan ring may be scanned while the memory remains running, and a regular ring may be scanned without affecting the log functionality. All normal rings are dependent on each other, however, as are all log rings. To scan any normal ring, all other normal rings must be disabled. Likewise, to scan any log memory scan ring, all other log memory scan rings must be disabled.

Figure 3–1, Scan Ring Bus Architecture (C201, C202, C210, C220)

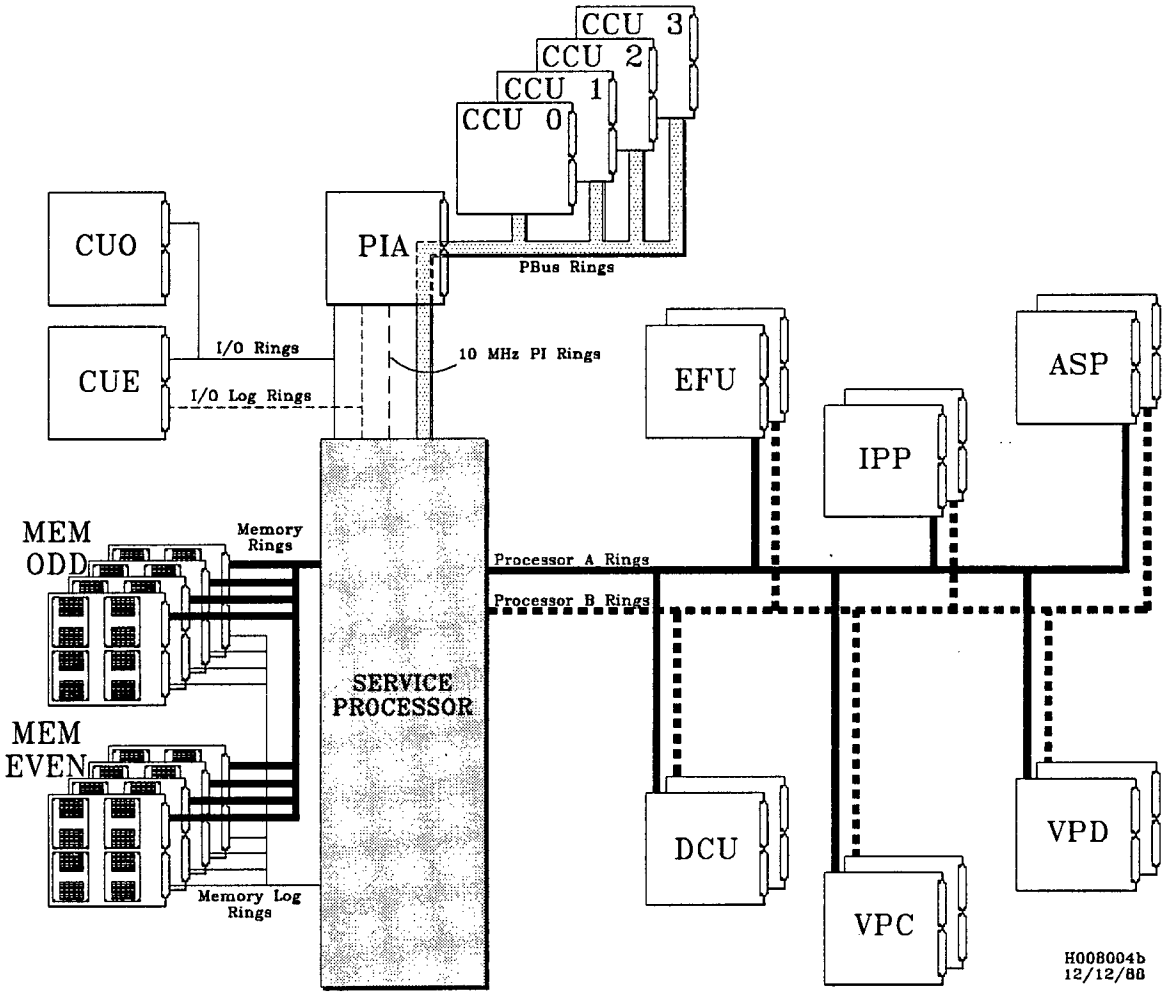
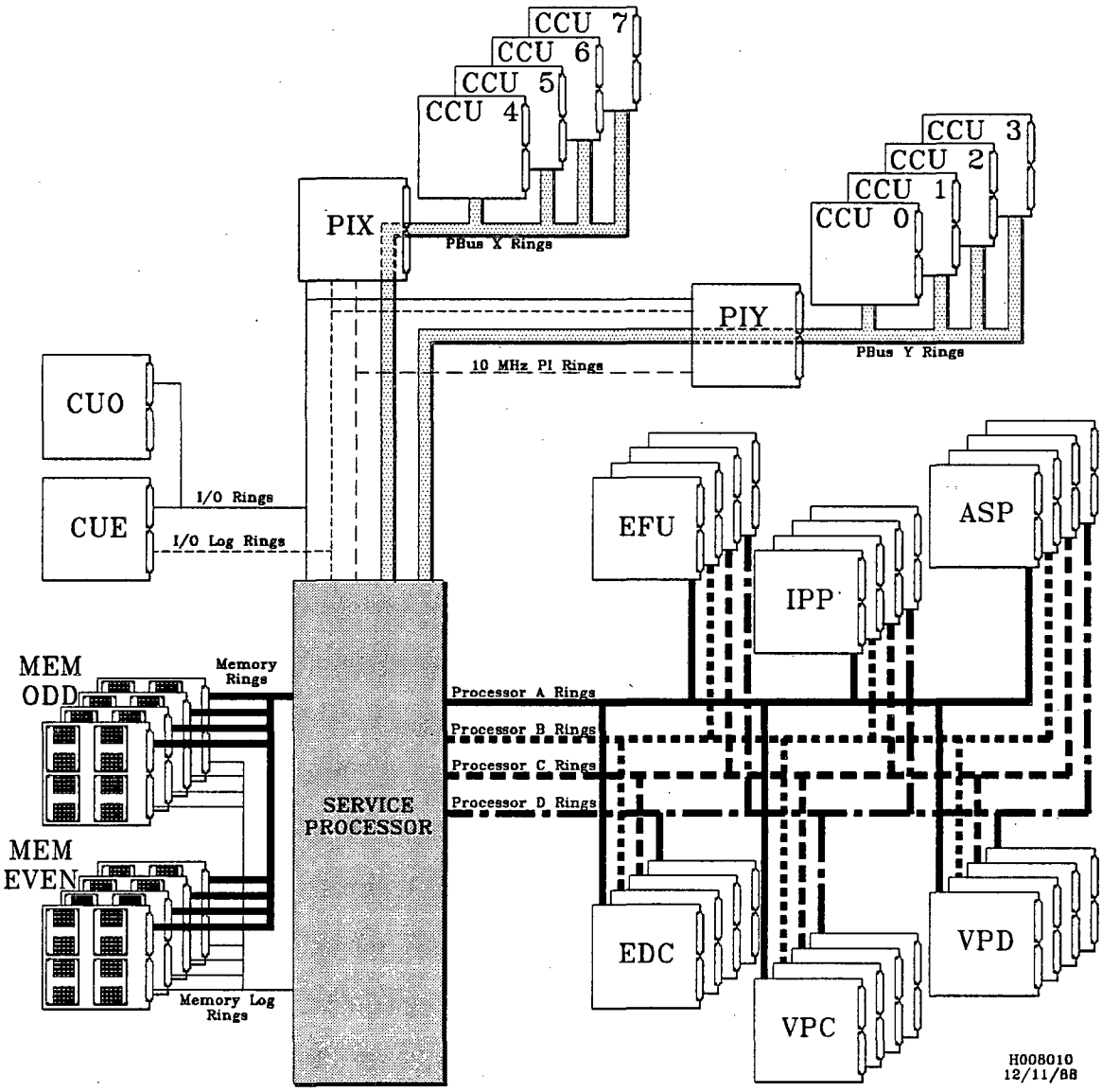


Figure 3–2, Scan Ring Bus Architecture (C230, C240)

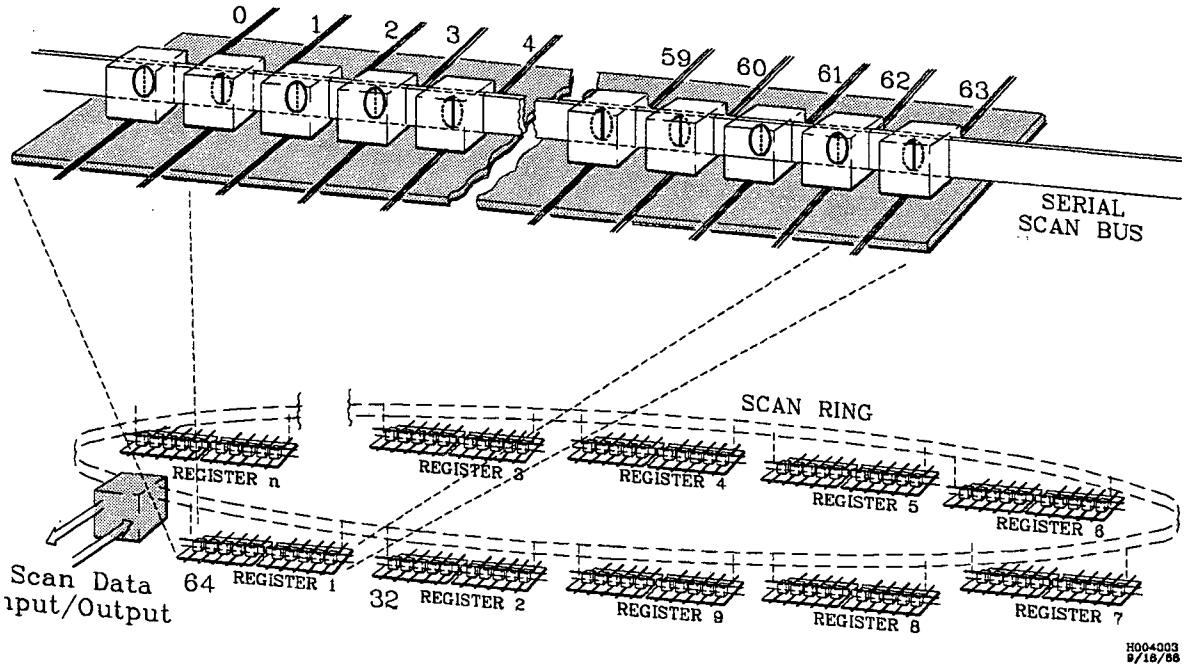


3.2.2 Scan Rings

There are registers (and register-like structures) in the machine that normally operate in a parallel fashion but include a serial input and output connections. These registers also have the ability to shift data bits right or left on command. In the CONVEX C200 Series computers, portions of some rings are unidirectional; they shift in the *right* direction only.

A number of these registers are connected together with the serial output of one register connected to the serial input of the next. Finally, the serial output of the last register in the line is connected back around to the serial input of the first register. The serial configuration of these registers is then circular, creating a scan ring.

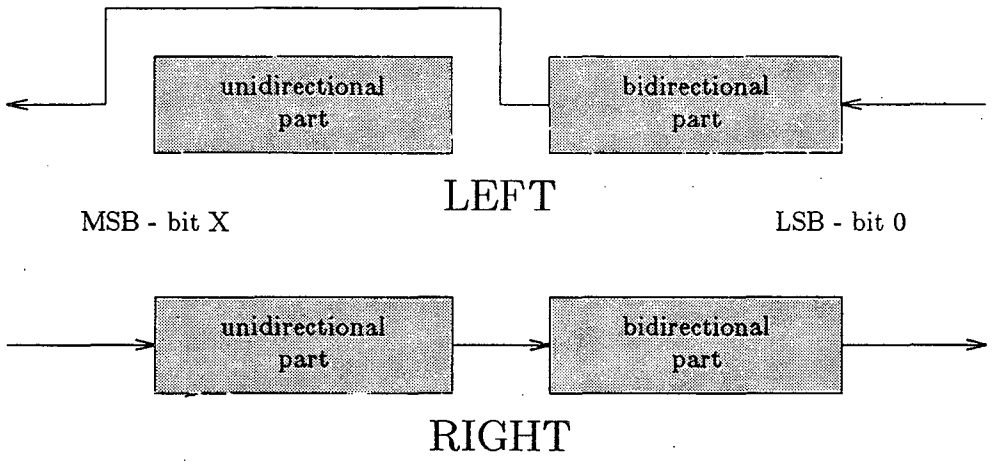
Figure 3–3, Scan Ring



All scan rings may be thought of as being made up of two parts: a *bidirectional* part and a *unidirectional* part, each of which may have any length, including zero. The terms bidirectional and unidirectional come from the ability to scan that part of a scan ring. Bidirectional parts may be scanned in both the left and right directions. Unidirectional rings may be scanned only in the right direction, and are bypassed when scanning left. A scan ring that has both unidirectional and bidirectional parts will always have the bidirectional part be at the least significant end of the scan ring.

The following figure shows a generic scan ring:

Figure 3–4, Scan Ring Directions



There are three types of scan rings:

- **Fully bidirectional** — The unidirectional part has zero length
- **Partly bidirectional** — Both parts have nonzero lengths
- **Fully unidirectional** — The bidirectional part has zero length

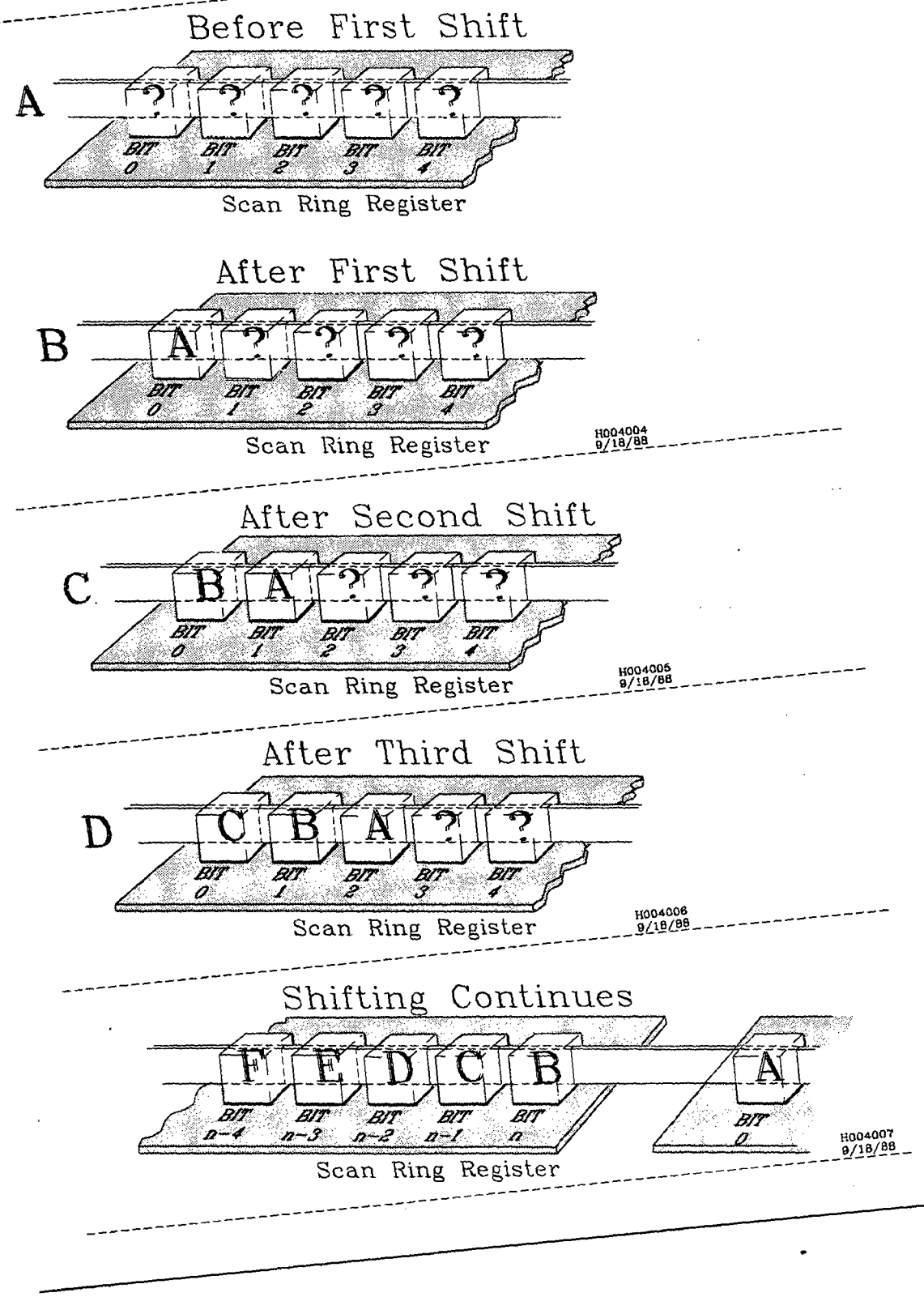
All fields of all scan rings may be read or written when scanning right. When scanning left, only the fields in the bidirectional portion of the scan rings are accessible.

The Service Processor controls the scan rings through its scan interface, which contains registers, scan control logic, and clock control logic. The other boards in the system are connected by lines to, and controlled by, the scan interface registers during scan operations.

3.2.2.1 Loading Scan Rings

Scan data is loaded into a ring sequentially, one bit at a time, as shown in the following figure:

Figure 3–5, Loading Scan Ring Registers



The first bit is applied to the serial input of the first register and then the registers are shifted. This moves the first bit into the first bit position of the first register. The next bit is then applied to the input of the first register and the registers shifted again (the same direction as before). This moves the first bit into the second position in the register and moves the second bit into the first register location.

The next shift moves everything over once again, with the third bit being moved into the first register location. When the first bit emerges from the serial output of the first register, it goes into the serial input next register in the ring on the next shift. This process is repeated until the entire scan ring is loaded with bits, with each located in its proper position.

In a similar fashion, a single bit can be placed in any location in the scan ring by applying the bit to the input and issuing the required number of shifts to move it around to the desired location in the ring. If a particular scan register is 64 bits wide, it takes 65 shifts to get a bit into the register and then shifted out the other end. A scan ring may include hundreds of registers, and, therefore, thousands of bits.

3.2.2.2 Reading Scan Rings

Scan rings are read by shifting the registers in the ring until the desired bit or bits come around to be read. Reading a scan ring is nondestructive so long as rings with unidirectional portions are not shifted in the *left* direction. The data remains in the ring after any number of right shifts during a read operation. Data can be restored to its original position in the ring by shifting the ring the appropriate number of times to move the data full circle back to its original location in the ring.

3.2.3 Scan Ring Field Names

Field names are defined for the user by the *scan_builder*. The *scan_builder* is a program that creates the structures needed by C programs to access the scan rings. The *scan_builder* creates a collection of files that form the Iscan database for field and ring definitions. Iscan constructs a field name according to the following pattern:

<ringname>[index]:<field_def>[index].<field_def>[index] . . . <field_def>[index]

where

1. *Ringname* designates the scan ring to be accessed. This is not always synonymous with the board name. A board may have more than one scan ring associated with it. There may also be more than one board of the same name in a system.
2. There is an optional index on each field definition and on the ringname.

The following are examples of how Iscan field names must be constructed:

mcm[0]:wr_dat[2]

mcm[1]:xbar[1].is[0]

The first example specifies the wr_dat field[2], in the memory array ring[0]. The second field specifies the memory array ring[1], in the xbar[1].is[0] field.

The following tables list the ringnames that are currently defined for C200 Series machines.

Table 3–1, C200 Series Processor Ringname Definitions

RINGNAME	ALIASES FOR PROCESSOR			
	A	B	C	D
Vector Processor Data Ring	vda vd[0] vpd[0]	vdb vd[1] vpd[1]	vdc vd[2] vpd[2]	vdd vd[3] vpd[3]
Vector Processor CMOS Data Ring	vdca vdc[0] vpdc[0]	vdcb vdc[1] vpdc[1]	vdcc vdc[2] vpdc[2]	vdcd vdc[3] vpdc[3]
Vector Processor Control Ring	vca vc[0] vpc[0]	vcb vc[1] vpc[1]	vcc vc[2] vpc[2]	vcd vc[3] vpc[3]
Data Cache Unit Ring	dca dc[0] dcu[0] edc[0]	dcb dc[1] dcu[1] edc[1]	dcc dc[2] dcu[2] edc[2]	dcd dc[3] dcu[3] edc[3]
Scalar Functional Unit Ring	fua fu[0] sfu[0] efu[0]	fub fu[1] sfu[1] efu[1]	fuc fu[2] sfu[2] efu[2]	fud fu[3] sfu[3] efu[3]
Instruction Preprocessor Ring	ipa ip[0] ipp[0]	ipb ip[1] ipp[1]	ipc ip[2] ipp[2]	ipd ip[3] ipp[3]
Arithmetic Scalar Processor Ring	asa as[0] asp[0]	asb as[1] asp[1]	asc as[2] asp[2]	asd as[3] asp[3]

Table 3–2, Memory Array Ringname Definitions

RINGNAME	MEMORY PAIR			
	0	1	2	3
Memory Array Run Ring Even	mcm[0] me0	mcm[2] me1	mcm[4] me2	mcm[6] me3
Memory Array Run Ring Odd	mcm[1] mo0	mcm[3] mo1	mcm[5] mo2	mcm[7] mo3
Memory Array Log Ring Even	lmcm[0] lme0	lmcm[2] lme1	lmcm[4] lme2	lmcm[6] lme3
Memory Array Log Ring Odd	lmcm[1] lmo0	lmcm[3] lmo1	lmcm[5] lmo2	lmcm[7] lmo3

Table 3–3, Service Processor Ringname Definitions

RINGNAME	ALIASES	
Service Processor Ring	sp4	sp2

Table 3–4, CPU Utility Ringname Definitions

RINGNAME	ALIASES FOR	
	C201, C202 C210, C220	C230 C240
CPU Utility Rings	cpx	cue; cuo
CPU Utility Log Rings	lcpx	lcue

Table 3–5, PBUS Interface Adapter Ringname Definitions

RINGNAME	ALIASES FOR	
	C201, C202 C210, C220	C230 C240
PBUS Interface Adapter Ring	pi[0]/pia	pi[0]/piy, pi[1]/pix
PBUS Interface Adapter Log Ring	lpi[0]/lpia	lpi[0]/lpiy, lpi[1]/lpix
PBUS Interface Adapter 10Mhz Ring	opi[0]/opia	opi[0]/opiy, opi[1]/opix

Table 3–6, Channel Control Unit Ringname Definitions

RINGNAME	ALIASES
Channel Control Unit	ccu[0], ccu[1], ... ccu[7]

Entering **List** displays the ring names in a particular configuration, and if used at system initiation may alert the user to board changes that may have been made in his absence.

Board names usually describe the slot in which they fit. For example, the same board will always go in slot ASP[0]. In other cases, the board names are quite different and there is some variety in the type of boards that can go in the slots. For example, the CCU[0] slot could contain an HSP board or an IOP board. Thus, CCU[2]:cpu_red_led might make sense if an HSP board is present in slot CCU[2], but may not make sense if an IOP board is present.

3.3 Modes of Operation

Iscan can operate in the program mode and the screen mode. Each mode offers different capabilities to the user of Iscan.

3.3.1 Program Mode

The program mode is a line-oriented interface that executes commands as they are entered. It allows interactive definitions of functions, procedures, synonym lists, and screens.

3.3.2 Screen Mode

The screen mode is a display-oriented mode that facilitates rapidly manipulating the bits within a field. This mode uses simple graphic representations of fields in horizontal columns, with numbers that indicate bit values in binary, hexadecimal, or even decimal. Interaction occurs with cursor control keys and control sequences to display and modify Iscan field bits. The program mode is used to define the format of a screen mode display.

Screen specifications allow the Iscan user to define the position of Input and Output (I/O) fields on the screen via a simple C-like screen description language. These I/O fields are used in conjunction with the *edit* command in Iscan. When entering the screen mode the user specifies a screen name. This screen name is associated with a screen specification that tells the scan facility which scan ring fields is to be displayed and where they are to be placed on the video screen.

Once a screen has been entered, the data on the screen can be modified by moving to the appropriate field and changing the numbers on that screen. The rings are not be updated until the field is written by pressing CTRL **W** or CTRL **P**. While in edit mode, the cursor keys are used to move about between fields. Any sequence of valid Iscan commands may be entered by hitting the PF2 key and entering the commands. The display of these commands appears in a command status area of the screen below the I/O fields.

Depressing PF2 again brings the cursor back up to the last field edited before escaping to command mode. This creates a speedy and simple way to interactively modify scan rings. The ring is not updated until CTRL **W** is pressed to update it. Thus after a long series of "escaped" commands, a CTRL **W** command forces updating of the ring to verify the changes that have been made. A CTRL **R** command is used to update the entire screen. These features allow the user control of when hardware reads occur to update the screen.

For a complete list of these commands, refer to the "*edit*" subsection under "Detailed Iscan Command Descriptions" elsewhere in this chapter.

3.4 Iscan Commands

Iscan commands can be seen as intrinsic functions within the CONVEX 200 Series computers' Iscan language, much in the same way that input or output functions are intrinsic in Pascal. They allow a low overhead (minimal threaded code) method to do common input or output operations or to allow access to features that are useful in an interactive environment.

3.4.1 Iscan Command Summary

NOTE

Enter **help** or **?** for an on-screen list of Iscan commands.

The following table is a comprehensive list of the Iscan commands, with the short form or alias of each command, as well as the meaning of each command:

Table 3–7, Iscan Commands

Command	Alias	Meaning
!	*!*	Execute a UNIX command
adjust	*adjust*	Check scannability of all specified rings
bdrev	*br*	Check or display the revision level of a board
bdtype	*bt*	Check the presence of a board type in a slot
clear	*clr*	Clear the screen
clock	*c*	Generate a clock pulse
dump	*du*	Dump scan ring save buffers to a named file
edit	*e*	Invoke screen mode
even_parity	*even_parity*	Generate even parity
exit	*exit*	Leave Iscan
fetch	*fetch*	Fetch a field for placement in a register
fprint	*fpr*	Print to a file
get	*g*	Get a field value
halt	*hlt*	Take one or more boards out of the run state
help	*?*	Help
include	*in*	Read a specified file as input to the interpreter
iforce	*iforce*	Force a *scn_rd* operation on all gets
iupdate	*iu*	Manipulate the *iupdate* flag or force output after puts
list	*list*	List the fields associated with the named scan ring
loadscan	*ls*	Generate a scan command and a single clock pulse
log	*l*	Create a log file (this command cannot be logged)
logl	*ll*	Create a log (this command can be logged)
odd_parity	*odd_parity*	Generate odd parity
print	*pr*	Display a set of values on the screen
put	*p*	Put a field value
reset	*re*	Reset a subsystem
restore	*rs*	Restore a scan ring from a buffer
ritchie	*ri*	Terse error messages issued when toggled on
run	*r*	Put a slot in the run state
save	*sv*	Save the state of a scan ring to a buffer
scnclear	*sclr*	Clears a scan ring to all zeros
scnout	*so*	Create identical scan rings for multiple boards
screens	*sc*	Print out all names of defined screens
undump	*undu*	Read a named file to reconstruct a scan ring save buffer
verify	*v*	Enable read or compare verification

3.4.2 Detailed Iscan Command Descriptions

The following is a detailed command description with examples of how each command might be used. The initial appearance of the command on the far left is immediately followed by the short form of the command, if one exists, in parentheses. This is followed by a description of the command, the proper form of the command, and examples of the use of the command.

! (!)

This command is the *execute unix* command. It escapes from the Iscan environment and executes a UNIX command. The user may also leave the scan environment temporarily by spawning a shell underneath the scan environment. This would be

done by entering **!sh**. To leave the shell and resume Iscan, returning to an Iscan prompt, enter CTRL **d**.

The *execute unix* command is actually an intrinsic function like *fetch*. It can be used to return the exit status of the resulting shell by making an assignment with the ! symbol on the right hand side, such as :6 = !ls -la.

FORM:

<register name> = ! [*unix command*]

EXAMPLES:

1. **:7 = ! ls**

   This lists all files in the current directory.

2. **!sh**

   This exits the scan environment and enters an underlying shell.

adjust (no short form)

The *adjust* command checks the scannability of all the specified rings. If there are any that are unscannable, it clocks them at 25 MHz with the provided flags until it has either clocked its defined maximum, or the rings are scannable. If the maximum number of clocks is exceeded, the hardware is assumed to be broken and an error condition exists.

RELEVANT FLAGS:

- **d** — The *d* flag causes the Diagnostic Mode (Dmode) to be asserted during the underlying Iscan call. Dmode is the system diagnostics mode. When Dmode is asserted, system hardware identifies incoming pulses as being diagnostic in origin, and not intended for regular processing.

FORM:

adjust [**-(d|l|r|o|h)**] [*number*] [*slot,slot_1,...slot_n*]

bdrev (br)

The *bdrev* (board revision) command allows the user to obtain the revision (rev) level of the board in the slot designated by the command. The request for rev level information must be followed by a string name. The string name must be used to recall the rev level. A string name is defined by:

[A]$[B]

Where:

- A—is a required character ranging from a...z or A...Z
- $—is required exactly as shown
- B—is an optional character ranging from a....z or A...Z

FORM:

bdrev [*slotname string*]

EXAMPLES:

1. **bdrev dcu[0]: a$**

   This obtains the revision level information for the board in dcu slot 0 and place it in the string variable named a$. At this point, the command of **print "%s" a$** would print the rev level of the board residing in dcu slot 0.

2. **br dcu[1]: d$cu_revlevel**

   This obtains the rev level for dcu slot 1 and place it in a string named d$cu_revlevel. As seen here, the short form for *bdrev* is *br*.

bdtype (bt)

The *bdtype* command checks the specified slot for the specified board name and returns a -1 if the slot is empty, a 0 if the board is present, and a -4 if another board is present.

FORM:

bdtype [*slotname "boardname"*]

EXAMPLE:

1. **:3 = bt ccu[0]: "hsp"**

   This places a 0 in register 3 if there is an hsp board in slot ccu[0]. The short form of *bdtype* is *bt*. The command **print "%d" :3** can be entered to check the contents of register 3, in order to verify that this command has been executed. A 0 is returned if the command was executed properly.

clear (clr)

The *clear* command has no arguments and is only used to clear the screen when it becomes corrupted by line noise, tty driver problems, etc.

FORM:

clear

EXAMPLES:

1. **clear**

   This clears the screen of everything but a single scan prompt at the top of the page.

2. **clr**

   The short form of the *clear* command is *clr*.

clock (c)

The *clock* command generates a specified number of clocks for a specified set of boards. If no number argument is given, one clock pulse is generated. If no ring argument is given, all boards are clocked.

RELEVANT FLAGS:

- **d** — The *d* flag causes the system Diagnostics Mode to be asserted during the underlying Iscan call. Use of the dmode flag causes the Iscan clock routine to pass an indicator to an underlying scan routine, telling it to set the dmode bit of the DCR (Diagnostic Control Register). Normally this bit is clear. The value in the dmode bit of the DCR is output to all boards in any of the subsystems affected by the command.

The following flags are scan control flags. Using any of the scan control flags causes the Iscan clock routine to pass an indicator to an underlying scan routine, telling it to set the scan control field of the DCR (Diagnostic Control Register) to the appropriate value. The values in the scan control field of the DCR are output to all boards in any of the subsystems affected by the command.

- **l** — The *l* flag causes the scan left pattern (01) to be applied to the Iscan control lines.
- **r** — The *r* flag causes the scan right pattern (10) to be applied to the Iscan control lines.
- **o** — The *o* flag causes the load pattern (00) to be applied to the Iscan control lines. This is used for parallel loads of scan rings.
- **h** — The *h* flag causes the hold pattern (11) to be applied to the Iscan control lines. This is used for writes to writable control store.

> **NOTE**
>
> Enter CTRL **C** to interrupt a lengthy clock command.
>
> A colon is always placed after the index number of the slot name.

FORM:

clock [-(d|l|r|o|h)] [*number*] [*slot,slot_1,...slot_n*]

EXAMPLES:

1. **clock**

   This generates one pulse for all boards in the system.

2. **c -d 5 ipp[0]:**

   This generates five clocks for the IPP board with the Diagnostic Mode (Dmode) asserted.

dump (du)

The *dump* command with no flags writes out to a specified file all the rings that have been previously saved with the *save* command. The file can be used to restore rings in conjunction with the *undump* command. The *undump* command allows the state of rings to be saved across Iscan sessions. If no argument is given then the dumpfile defaults to *scn_dumpfile*.

The *dump* command may also be supplied with a flag. This form of the *dump* command is not used to save rings across sessions but rather to debug Iscan scripts or sessions. When a flag is given to *dump*, the filename is ignored.

This command is used primarily for debugging purposes.

RELEVANT FLAGS:

- **-x** — The *x* flag dumps the symbol table (table of all slots, structures, and variables that Iscan builds as it uses those slots, structures, and variables)
- **-s** — The *s* flag dumps the p-code stack (Iscan interpreter's internal data structure)
- **-c** — The *c* flag dumps the p-code space (Iscan interpreter's scratch strorage space)

When a flag is used, the output is displayed in screen mode only, and cannot be written to a file.

FORM:

dump [-(**x**|**s**|**c**)] [*filename*]

EXAMPLES:

1. **du**

   This dumps the contents of the rings that have already been saved with the *save* command to the file *scn_dumpfile*.

2. **dump "cputest"**

   This dumps the contents of the rings that have been saved to a file called *cputest*.

3. **dump -x**

   This shows the contents of the symbol table.

edit (e)

The *edit* command activates the screen mode. In this mode the screen is split into two areas, and the user is allowed to interactively modify the fields that are called out in the specified screen definition. For a more detailed description of how this is done, refer to the section titled "Screen Specifications and Screen Mode." When in screen mode the user has a special set of commands available to perform various functions, including:

- CTRL R — Update the screen with a forced hardware read
- CTRL W — Force a hardware write on all displayed fields
- CTRL P — Force a hardware write on the field under the cursor
- CTRL G — Update the field under the cursor with a forced hardware read
- PF1 — Display more detailed information about the field under the cursor, including:
  - Full name and symbol type of the displayed quantity
  - Full mnemonic
  - Number of valid bits
  - Number of valid digits
  - Display base
  - Screen field width
  - Position on screen
  - Synonym defined for this field
  - Pin list
- PF2 — Drop down to command line for a series of non-editor commands, depressing PF2 again returns the user to the previous point in the display region
- CTRL N — Toggle between the synonym and value subfields
- CTRL F — Move forward in the synlist when on the synonym subfield
- CTRL B — Move backward in the synlist when on the synonym subfield
- cursor up — Move to field directly above current field
- cursor down — Move to the field directly below the current field
- cursor left — Move to the field directly to the left of the current field
- cursor right — Move to the field directly to the right of the current field
- CTRL l — Refresh the screen
- CTRL x — Exit from edit mode and return the screen to the program mode

FORM:

edit *screen_name*

EXAMPLES:

1. **edit mem_array**

   This brings up the screen defined by the *mem_array* screen specification and puts the user into screen mode.

2. **e mem_array**

   This is an illustration of the short form.

even_parity (no short form)

The *even_parity* command 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.

FORM:

<register name> = even_parity [*n*]

EXAMPLES:

1. **:0=even_parity 8**

   This causes register :0 to contain the number *7* in hexadecimal.

2. **:0=even_parity 80000000**

   This causes register :0 to contain the letter *e* in hexadecimal.

fetch (no short form)

The *fetch* command fetches the desired field in the same manner as a *get* command. However, instead of displaying the desired field, the *fetch* command returns the field for use in an Iscan script. For this reason, *fetch* can be considered an intrinsic function in Iscan.

FORM:

<register_name> = fetch [*fieldname*]

EXAMPLES:

1. **:0 = fetch mcm[0]:ecc[0]**

   This takes the field value of mcm[0].ecc[0] and places it in register :0.

2. **:0 = fetch mcm[:1]:ecc[:2]**

   This does the same as the example above, but the indices are determined by the registers :1 and :2.

fprint (fpr)

The *fprint* command writes the contents of a register or a string to a file that is specified in the command. It is very similar in use to the *fprintf()* function in C. If a flag of -a is specified then the file in question is appended, otherwise the file is truncated before the write.

> **NOTE**
>
> A maximum of 16 values is printable in a single *fprint* statement.

FORM:

fprint *"filename"* -w *"c_format_string"* [*value1,value2,...value16*]

or

fprint *"filename"* -a *"c_format_string"* [*value1,value2,...value16*]

EXAMPLES:

1. **fprint "dummy" -w "register 5 is = %d" :5**

   This prints the contents of register :5 in decimal format along with an explanatory string in the file *dummy*. Before writing, the previous contents of the file will be deleted.

2. **fpr "bugs" -a "Memory hard error"**

   This prints the string "Memory hard error" at the end of the file *bugs*. *fpr* is the short form of *fprint*.

get (g)

The *get* command is used to *fetch* and display the specified field or all fields in the specified slot. If the field specified is an element of the current screen then it is displayed in its proper place in the display region. If, however, screen mode is not active or the field specified is not on the current screen, the value and its synonym is displayed on the next line. The *get* command does a hardware read, if necessary, to fetch a field. Unless the *iforce* flag is on, a *scn_read*, or hardware read of the ring is not guaranteed.

FORM:

get [*-l*] *fieldname*

or

get *slotname*

EXAMPLES:

1. **get mcm[0]:ecc[0]**

   This displays the bit values for the field mcm[0]:ecc[0].

2. **g mcm[1]:**

   This illustrates the use of the short form of *get* and the use of an entire slot as an argument. This command displays the bit values for the ring mcm[0].

halt (hlt)

The *halt* command disables incoming clocks on a specified board or boards.

FORM:

halt [*slot,slot_1,...slot_n*]

EXAMPLES:

1. **halt sfu[0]:**

   This disables incoming clocks on the SFU board, index 0.

2. **hlt**

   This disables incoming clocks for every board in the system.
   The short form of *halt* is *hlt*.

help (?)

The *help* command with no arguments displays a summary of the available commands.

FORM:

help

EXAMPLES:

1. **help**

   This displays an on-screen summary of Iscan commands.

2. **?**

   This displays an on-screen summary of Iscan commands.

iforce (no short form)

The *iforce* command with no argument toggles the *iforce* switch. Entering **on** after the command sets the switch to on. Entering **off** after the command sets the switch to off. If the *iforce* flag is on, then all subsequent *get* commands cause a hardware read, even if the buffer is not modified or the previous get command was for the same slot.

FORM:

iforce [*on* | *off*]

EXAMPLES:

1. **iforce**

   This toggles the *iforce* flag.

2. **iforce on**

   This sets the *iforce* switch to on.

include (in)

The *include* command reads a specified file and parses it just as if it had been typed on the keyboard. This is frequently used in situations where commands are repeated, such as in defining screens or synonym lists.

FORM:

include *"filename"*

EXAMPLES:

1. **include "debug.functions"**

   This loads the functions defined in the file *debug.functions*. These functions are executed just as if they had been defined interactively in program mode.

2. **in "myscreens"**

   This defines all the screens in the file *myscreens*.

iupdate (iu)

The *iupdate* command with no argument updates all rings that have had *puts* done on them, and the *iupdate* flag is not affected. Naturally, if the *iupdate* flag is already on, the *iupdate* command with no arguments would be superfluous. This is only of use when the flag is off. With an argument, *iupdate puts* the *iupdate* flag in the specified state.

If the *iupdate* flag is on, all *puts* cause immediate hardware writes to the ring. If the *iupdate* flag is off, *puts* does not result in actual hardware writes until the *iupdate* command without arguments is issued. The default value of the *iupdate* flag upon entry to the scan environment is *iupdate* on.

FORM:

iupdate [*on* | *off*]

EXAMPLES:

1. **iupdate**

   This causes all rings that have been modified to be written to their respective boards. The state of the *iupdate* flag is unaffected.

2. **iu off**

   The short form of *iupdate* is *iu*. This command line sets the *iupdate* flag to off so that subsequent *puts* do not effect the hardware.

list (list)

The *list* command allows the user to display, in "more" format, every field in the named ring. This command is useful when the user forgets the name of a field. The *list* command without an argument displays all ring names in the system.

FORM:

list [*slot*]

EXAMPLES:

1. **list**

   This displays all rings in the existing configuration.

2. **list mcm[0]:**

   This displays the names of all fields in the ring mcm[0]:.

loadscan (ls)

The *loadscan* command generates a clock with *loadscan* asserted for the specified board or for all boards. To optionally assert the Diagnostic Mode (Dmode) line, use the *-d* flag.

FORM:

loadscan [**-d**] [*slot,slot_1,...slot_n*]

EXAMPLES:

1. **loadscan -d**

   This executes *loadscan* for all the boards in the system and asserts the Dmode line for each.

2. **ls dcu[0]:**

   This sets the Iscan control lines exclusively for the DCU[0] board without asserting the Dmode line. The short form of the *loadscan* command is *ls*.

log(l), logl (ll)

The *log* command creates a file that logs all subsequent commands executed, with the exception of the following commands:

- *log*
- *include*
- *edit*

If no argument is given to the *log* command, the default filename of *scn_logfile* will be used for the *log* file. In order to prevent the session from automatically writing over the last session's *log* file, the following scheme is used: upon issuance of the *log* command a file called *log.tmp* is created and commands are logged to it. When exiting Iscan, the *log.tmp* file will be renamed *logfile*.

If *log* was issued with the default, the user will be given another opportunity while exiting to save the *logfile*. At this time, the user will be asked to rename the *logfile*. If logging to the default filename would be a problem, write the log to a different filename.

The *logl* variant of *log* is identical to *log* except that it itself will be logged.

FORM:

log [*"filename"*]

EXAMPLES:

1. **l**

   This starts logging all commands that follow the *log* command in the default *log* file called *scn_logfile.*

2. **logl "newprog.log"**

   This creates a file called *newprog.log* that contains all subsequently logged commands.

odd_parity(no short form)

The *odd_parity* command is used to generate 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.

FORM:

<register name> = odd_parity n

EXAMPLES:

1. **:0=odd_parity 8**

   This command causes register :0 to contain the number 8 in hex.

2. **:0=odd_parity 800 (hex)**

   This command causes register :0 to contain the number 1 in hex format.

print (pr)

The *print* command displays the contents of a register or a string. It is similar in use to the *printf()* function in C.

> **NOTE**
>
> A maximum of 16 values are printable in a single *print* statement.

FORM:

print *"c_format_string"* [*value1,value2,...value16*]

EXAMPLES:

1. **print "register 5 is = %d" :5**

This prints explanatory statement "**register 5 =**" followed by the actual bit values of register 5 in decimal format.

2. **pr "Memory hard error"**

   This prints the string, "Memory hard error".

put (p)

The *put* command places a specified value into a specified field within a scan ring. The value can be:

- An unsigned integer constant
- A register value
- An indirect register value

Select the base of the integer constant by prefixing the constant with a base selector:

- %B selects binary representation
- %D selects decimal representation
- %H selects hexadecimal representation and is the default if no base is specified.

If the *iupdate* flag is set to on, then *puts* to a field will automatically cause the entire scan ring to be written to the hardware. If the *iupdate* flag is set to off, then the *put* only causes the local copy of the ring to be modified, making it necessary to use *iupdate* to force the write to occur.

It should be noted that the first *put* on a ring causes a hardware read on the ring to supply the current values in all but the field being *put*. In this way a *put* really implies a read as well as a write if *iupdate* is on.

FORM:

put *fieldname value*

EXAMPLES:

1. **p dcu[0]:gate_array.control af**

   This example illustrates the loading of the field, *gate_array.control*, in the DCU[0] ring, with the hexadecimal constant, `af`. The short form of the *put* command is *p*.

2. **p mcm[0]:gate_array.control :12**

   This example illustrates the use of REGISTER 12 to load field *mcm[0] gate_array.control*.

reset (re)

The *reset* command resets the specified subsystem of the CONVEX C200 Series computers. The following subsystems can be *reset* (each subsystem is followed by the abbreviated form that is entered on the screen):

- Processor a (**proca**)
- Processor b (**procb**)
- Processor c (**procc**)
- Processor d (**procd**)
- The memory subsystem (**mem**)
- The I/O subsystem (**io**)
- The entire system (**sys**)
- CPU (**cpu**)

FORM:

reset [*subsystem,subsystem_1,...subsystem_n*]

EXAMPLES:

1. **reset**

   This resets all subsystems.

2. **re cpu,mem**

   This illustrates the resetting of multiple subsystems. The short form of the *reset* command is *re*.

restore (rs)

The *restore* command restores a ring that was previously saved with the *save* command. Without an argument, the *restore* command restores all the rings that have been saved with the *save* command.

FORM:

rs [*slot,slot_1,...slot_n*]

EXAMPLES:

1. **rs**

   This restores all previously saved boards. The short form of the *restore* command is *rs*.

2. **restore vcu[0]:**

   This restores the VCU[0] board.

ritchie (ri)

The *ritchie* command with no argument toggles off the *ritchie* switch. The argument following the command can specify an on or off state. When Iscan is in *ritchie* mode, terse error messages are used. The *ritchie* mode is the default

mode. When *ritchie* is off, a more detailed explanation of errors and alternative inputs will be displayed.

FORM:

ritchie [*on* | *off*]

EXAMPLES:

1. **ritchie**

   This toggles the *ritchie* flag and causes an exit from the *ritchie* mode if in *ritchie* mode, or enter *ritchie* mode if not in *ritchie* mode.

2. **ri on**

   This will switch the *ritchie* mode to on. The short form of the *ritchie* command is *ri*.

run (r)

The *run* command places a specified board or boards in the *run* state.

FORM:

run [*slot,slot_1,...slot_n*]

EXAMPLES:

1. **run sfu[0]:**

   This places the SFU[0] board in the *run* state.

2. **r**

   This places every board in the system in the *run* state.

save (sv)

The *save* command saves a scan ring into a buffer. The ring can later be restored via the *restore* command. These commands are useful when saving the state of a ring before performing operations that effect the ring. Without an argument, the *save* command *save* every ring in the system.

FORM:

save [*slot,slot_1,...slot_n*]

EXAMPLES:

1. **sv**

   This saves every ring in the system to a buffer.

2.

save vcu[0]:,vcu[1]:

This saves the rings in slots VCU[0] and VCU[1].

scnclear (sclr)

The *scnclear* command clears a scan ring to all zeros. If the *verify* flag is on, then the ring is checked after writing to make sure that zeros were actually placed in the scan ring.

FORM:

scnclear [*slot,slot_1,...slot_n*]

EXAMPLES:

1. **scnclear opia:**

   This will clear the opia ring only.

2. **sclr**

   This sets all scan rings in the system to zero. The short form of the *scnclear* command is *sclr*.

scnout (so)

The *scnout* command facilitates the duplication of scan rings to multiple slots. It copies an entire scan ring from one slot to another.

FORM:

scnout *slot_source slot_destination*

EXAMPLES:

1. **scnout dcu[0]: dcu[1]:**

   This copies the scan ring values from slot DCU[0] to slot DCU[1].

2. **so dcu[0]: dcu[2]**

   This illustrates the short form of the *scnout* command.

screens (sc)

The *screens* command displays the names of all defined screens. The output is in multicolumn format.

FORM:

screens

EXAMPLES:

1. **screens**

   This displays the names of all defined screens.

2. **sc**

   This displays the names of all defined screens. The short form of the *screens* command is *sc*.

undump (undu)

The *undump* command reads in the specified file and builds the *restore* buffer with it. The *undump* command must be created in the same format as the *dump* command. After the *undump* command is given, the *restore* command will restore all boards that were in *dumpfile*.

If no argument is given, the file called *dumpfile* will be searched. If this file does not exist, the command will fail and print a diagnostic message to this effect.

As the *undump* command is reading the *restore* buffer, it prints the names of the rings that it is restoring.

FORM:

undump [*"filename"*]

EXAMPLES:

1. **undump**

   This reads in the default dump file and rebuilds the *restore* buffers from it.

2. **undu "testcpu"**

   This reads the file *testcpu* and rebuilds the *restore* buffers from it. The short from of the *undump* command *undu*.

verify (v)

The *verify* command with no argument toggles the *verify* switch. Entering **on** immediately after the command sets the switch to on. Entering **off** following the command sets the *verify* switch to off. If the *verify* flag is on, all subsequent *puts* are checked by reading the field back again.

If the written value differs from the read-back value, three values will be printed on the screen in order to see the nature of the failure . The XOR of the read and write values is written first, the written value, second, and then the read-back value.

FORM:

verify [*on* | *off*]

EXAMPLES:

1. **verify**

   This toggles the *verify* flag.

2. **v on**

   This is the short form of the command, and sets *verify* to on.

3.5 Language Description

Iscan supports the following language constructs:

- Scan ring field names
- Structured programming control flow
- Conditional expressions
- Unconditional branching
- Registers
- Assignment between registers
- Arithmetic and logical operations on registers
- Subprogram calls
- Comments
- String variables

3.5.1 Statements

An Iscan statement is a command, assignment, or loop that performs an Iscan function. Statements are not free-formatted; rather, a statement must be on one line.

3.5.2 Compound Statements

Compound statements can exist anywhere that regular Iscan statements exist, and are separated by open or closed brackets. The most common use of compound statements is in looping and with conditional expressions.

3.5.3 Registers

There are three types of registers in Iscan:

- **Numerical registers** — These behave like the registers in a pocket calculator.
- **Hardware control status registers** — These are associated with the architecture of the SP2.
- **Software control status registers** — These are used to manipulate the underlying software of Iscan.

All Iscan register names in the CONVEX C200 Series computers begin with a colon (:), except for software control or status registers, which begin with a dollar sign

($), e.g., :99, $Scn_error, or :cgr. From the point of view of Iscan syntax, the three register types are almost identical. Any type can be read from (i.e., appearing on the right side of an assignment) or written to, (i.e., appearing on the left side of an assignment).

This orthogonality of registers allows for a great deal of power in the use of the Iscan. All numerical registers and software control registers in Iscan are unsigned 32-bit quantities. Hardware control registers are either 8-, 16- or 32-bit unsigned quantities.

If a 32-bit entity is copied into a 16- (8-) bit control register the result is the same as if only the low order 16 (8) bits had been copied. For example, :srr = 0aaff is equivalent to :srr = 0ff. Naturally, even though the syntax of Iscan is such that the registers are orthogonal, it would not be wise to treat the hardware control or status registers in the same way as a numerical register.

Constructions such as :5 = 3*:cgr are not valid and statements like :srr = :cgr are not only invalid but problematic. Because the number of numerical registers possible in an Iscan session is 1,000, valid numerical register names are :0-:999. There are as many control or status registers defined as there are such registers on an SP2.

Although commands may be used, it is recommended that software registers be used to set or clear the following flags:

- *iforce*
- *iupdate*
- *iritchie*
- *itrace*
- *iverify*

The following flags can be set or cleared only by use of the software control registers:

- *iscn_dbg*
- fail_verify
- auto_adjust

It is also important to note that the function bit width parameter can be set only with the following command:

$func_bit_width = <desired function bit width>

For example, the command **$func_bit_width = %D13** would specify 13 bits of function width.

3.5.3.1 Valid Hardware Registers

The following hardware registers have been defined on the Service Processor:

Table 3–8, Valid Hardware Registers

ACRONYM	DESCRIPTION
:MAP	UNIX local (Service Processor) memory map
:POP	Main memory population map
:WIN	EBUS window map
:PCR	Physical configuration register
:RFCNT	Refresh count
:BSR	Service Processor bus error register
:CFR	Clock frequency register
:PBUS_X	Clock mask register for PBUS X
:PBUS_Y	Clock mask register for PBUS Y
:COP	Service Processor cop chip control register
:CPR	Control panel register
:DCON	Diagnostic connect register
:DCR	Diagnostic ctrl register [dmr/smr]
:EARLY_CLK	C230, C240 early clock in bit 0
:ECR	Event count register
:EMR	Environmental monitor register
:ESR	Error status register (hard errors)
:IER	Interrupt enable register
:IO	I/O run register
:ISR	Interrupt status register
:LMCR	Service Processor local memory control register
:MEM	Memory run register
:MEM_LOG	Memory log run register
:MISC_LOG	Misc. log run register
:ODENA	Odena register
:PROC_A	Processor A run register
:PROC_B	Processor B run register

Table 3–8, Valid Hardware Registers (continued)

ACRONYM	DESCRIPTION
:PROC_C	Processor C run register
:PROC_D	Processor D run register
:RHR	Run halt register
:SDR	Scan data register
:SEL	Soft error log
:SFR	Scan feedback register
:SRR	System reset register
:SSN	Backplane serial number
:TRR	Test result register [Service Processor led's]
:ICR	Sib interrupt channel register
:IRS	Service Processor sib interrupt request pending
:EBUS_LOG	EBUS error log register
:TCSR	Timer control status register
:TDR	Timer data register
:CSR_2793	2793 control status register
:DR_2793	2793 data register
:SR_2793	2793 sector register
:TR_2793	2793 track register
:CCSR	Console uart control status reg
:CUDR	Console uart data register
:FIFO_CSR	FIFO control status register
:FIFO_data	FIFO data register
:MFCR	Misc. floppy control register
:RCSR	Remote UART control status register
:RUDR	Remote UART data register
:SCSI_CTRL	SCSI control register
:SCSI_DATA	SCSI data register

3.5.3.2 Valid Software Registers

The following software registers have been defined on the SP2:

- **$scn_debug** — This flag will turn on debug messages in the Iscan library. It should be used judiciously since it causes printfs which will confuse curses.
- **$scn_error** — This software register allows access to the Iscan package error code mechanism. Its contents are *scn_error* inside the Iscan package.

- **$_scn_mod** — This register allows the user to force a modified buffer in the scan package to be treated as unmodified.
- **$_scn_direction** — This gives access to the direction flag used internally to the scan package with either a **1** input for left, or a **2** input for right.
- **$itrace** — When this flag is set, Iscan pcode is traced.
- **$iupdate** — When this flag is set, all *put* commands are flushed immediately.
- **$iforce** — When this flag is set, all *get* command force a hardware read of the scan ring.
- **$iritchie** — When this flag is set, the terse *Dennis Ritchie* style of UNIX is preserved in Iscan. More information on errors and alternate commands is given by Iscan when this flag is set to off.
- **$iverify** — When this flag is set, all *put* is checked by reading back the data after the *put*. This is also true of the *scnclear* command.
- **$fail_verify** — This flag should be used only when debugging the Iscan code itself. It will force verify errors on CONVEX C200 Series computers.
- **$auto_adjust** — When this flag is set to on, all *put* commands from a screen will adjust, if necessary.
- **$iscn_dbg** — This flag causes diagnostic information to be printed out.
- **$silent_fail** — This flag causes functions called within screens to fail silently. It is turned on automatically when functions are called from a screen. If this flag is reset upon entry to a function, the function will not handle its own errors silently.
- **$func_bit_width** — This parameter causes a function to have a bit width just like a field. In this way, a function can behave like a field in the screen environment.

3.5.4 Looping

Iscan can expedite commands by implementing loops. The *for*, *do*, and *while* loops are easily executed in Iscan.

3.5.4.1 *for* Loop

The following is an example of a *for* loop:

```
fprint "data.dat" -w "Starting the file"
for :1 = 1,10 do {
  fprint "data.dat" -a "\n:1= %d" :1
  }
fprint "data.dat" -a "\nTest of for is over"
du -r
```

3.5.4.2 *do* Loop

The following is an example of a *do* loop:

```
:0 = 0 :10 = 1 fprint "data.dat" -w "Starting the test" do {
  :0 = :0 + 1
  fprint "data.dat" -a "\n ===> 1 :0 = %d" :0
  if( :0 > 5 ) then {
    fprint "data.dat" -a "\n\t===> 2 :0 = %d" :0
    }
  } until (:0 == 10)
fprint "data.dat" -a "\nTest of do is over"
```

3.5.4.3 *while* Loop

The following is an example of a *while* loop:

```
:0 = 10 :1 = 2 while (:0) {
  :1 = :1 + 1
  :0 = :0 - 1
  print "\nreg[0] = %d" :0
  print "\nreg[1] = %d" :1
  print "\nreg[0] = %d reg[1] = %d" :0,:1
  ::1 = :0
  print "\nreg[reg[1]] = %d" ::1
  }; du -r print "test %d %d %d %d %d" :0,:1,:2,:3,:4
```

3.5.5 Conditional Expressions

The following conditional binary operations are recognized in Iscan:

- Is equal to (==)
- Is not equal to (!=)
- Is less than (<)
- Is less than or equal to (<=)
- Is greater than (>)
- Is greater than or equal to (>=)

3.5.6 Branching

Iscan is capable of both conditional and unconditional branching to expedite programming.

3.5.6.1 Conditional Branching

Iscan supports conditional branching with the use of *if...then* statements similar to BASIC. The following is an example of the *If...then...else* syntax used in Iscan:

```
define
func test() {
  if( $1 >= 1 ) then {
    print "0irst level of nesting"
    if( $1 >= 2 ) then {
      print "0econd level of nesting"
      if( $1 >= 3 ) then {
        print "0hird level of nesting"
        }
      }
    }
  else {
    print "0 Zero argument"
    }
  }
enddefine
test(1)
test(2)
test(3)
```

3.5.6.2 Unconditional Branching

Unconditional branching is supported in Iscan for the CONVEX C200 Series computers, but it is not recommended. Functions and procedures are given more space by reclaiming storage after execution of commands. Since each line is interpreted as it is entered, *goto* statements are only usable within a function or procedure. Iscan supports a wide variety of programming language constructs for structured programming.

3.5.7 Assign Statements

Iscan supports three kinds of assignment statements:

- Direct assignment (e.g., :4 = 6 or :4 = :8) where the register in question is updated with the value of the left hand side.
- Indirect assignment (e.g., ::4 = 7 or ::5 = ::7) where the register(s) in question are pointers to the register to be used.
- String assignment (e.g., a$ = "Test" or b$ = C$) where the assignment causes the source string to be copied into the destination string.

3.5.8 Arithmetic and Logical Operations on Registers

Iscan supports a complete complement of arithmetic and logical operators. Information in registers may be easily manipulated using these operations. The operations

supported are:

- 1's complement (~)
- And (&)
- Inclusive or (|)
- Exclusive or (^)
- Shift right (>>)
- Shift left (<<)
- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)

All operators are evaluated left to right with the following precedence:

- First precedence (~)
- Second precedence (*, /, &, ^, |)
- Last precedence (+, -)

Parentheses override the precedence in Iscan just as they do in other languages.

The power of treating control or status registers on the SP2 in an analogous fashion to numeric registers is illustrated in the following code sequence:

```
if (esr&04) then {
            /* code to debug proc A fault */

.

.

.           }
else if (esr&8) then {
            /* code to debug proc B fault */

.

.

.           }
else if ... etc.
```

3.5.9 Subprogram Calls

There are two types of subprograms supported:

- Subroutine, or procedure calls, where no value is returned
- Function calls where a value is returned from the subprogram

In both function and procedure calls the following are valid arguments:

- Registers
- Constants
- Fieldnames

An empty argument list is permitted.

In the body of the function or procedure, definition arguments are referred to by their order in the argument list, much in the same way that the shell refers to arguments. Thus, $1 is the first argument passed, $2 is the second, and $n is the nth. The variable $0 is the argument count. In this fashion, it is possible to tell how many arguments were actually passed from within the function or procedure. The method of passage of arguments is by name. This means that if an argument is modified inside a function or procedure, the value of the argument in the calling program is also modified.

3.5.9.1 Procedures

Procedures are defined in the following manner:

```
proc procedure_name ()
 {
 statement_1;
 statement_2;
              .
              .
              .
 statement_n;
}
```

Procedures are invoked in the following manner:

procedure_name([arg1, . . . argn]);

Whenever the same set of actions must be performed repeatedly in the course of isolating a fault in a board, it may be wise to use a procedure in order to reduce error and increase efficiency. For example, all the *xbar.is* fields could be dumped on all seven mcm rings to a file by defining the following procedure:

```
define
proc d_xbar()
{


fprint "mcm.data" -w "State of the xbar.is fields"
for :0 = 0,3 do {
  for :1 = 0,7 do {
    for :2 = 0,3 do {
      :3 = fetch mcm[:0]:xbar[:1].is[:2]
```

```
        fprint "mcm.data" -a "0cm[%d]:xbar[%d].is[%d]= %x" :0,:1,:2,:3
        }
    }
  }
}
enddefine
```

This procedure is defined by either typing it on the keyboard or including a file containing it. Once defined, the information may be dumped to disk by entering:

```
d_xbar()
```

3.5.9.2 Functions

Functions are defined in the following manner:

```
func function_name () {
 statement_1;
 statement_2;
              .
              .
              .
 statement_n;
}
```

3.5.9.3 Example of a Function

A sample function might be:

```
func isum() {
            return( $1*($1+1)/2 );
}
```

3.5.9.4 Invocation of Functions

Functions are invoked in the following manner:

function_name([arg1, . . . argn]);

for example:

```
:5 = isum(5);
if ( isum(6)>5 ) then :4 = 5;
```

NOTE

If one of the statements in a function definition is not a return statement, the function will return when the last statement with a default return value of 0 is read.

Functions must be defined before they are invoked.

It is possible under Iscan to write a function that will behave exactly like a scan ring field in the screen field. In this fashion it is possible to place "hidden state" while in the screen mode display and deal with it in exactly the same way as if it were on a scan ring. In order for this to happen, a convention will be observed by the screen handling routines inside Iscan:

- **Write** — When a position field that is specified by a function instead of an Iscan field is being written via CTRL **P** or CTRL **W**, the function is passed a single argument and that argument is the value on the screen. The function detects this argument with the appearance of $0, and responds.
- **Reads** — When a position field is being read, the underlying function is called with no arguments. The function detects this argument with the appearance of $0, and responds. The function will be treated as if it had func_bit_width valid data bits. This corresponds to the bit width of an Iscan field.

The following is a sample function that uses the hidden state:

```
func disp_brd()
{
        :1 = $0

        if( :1 > 0 ) then {
#Do your stuff to write the hidden state here
                .
                .
                .
                }
        else {
#Do your stuff to read the hidden state here and place it in :0
                .
                .
                .
                $func_bit_width = 4 #Set the bit width to 4 bits
                return :0
                }

}
```

3.5.10 History

Iscan gives the user the ability to recall previous commands for resubmission or to edit and then submit the resulting command. The last 20 lines entered to Iscan are saved in a history buffer. When the PF1 key is hit, Iscan will display the last 20 commands entered and prompt for the number of the command to be recalled. At that point, select a command by entering a number, or return without selecting a command by depressing RETURN. It is possible to move through the history buffer by depressing the up arrow and down arrow keys. The up arrow goes back in history and the down arrow goes forward. Any line recalled in this fashion can be edited with the line editing features. The keys defined for these functions are the following:

- **Up arrow** or CTRL **P** — Go back in history
- **Down arrow** or CTRL **N** — Go forward in history
- PF1 — Display the entire history buffer and prompt for a line selection

3.5.11 Line editing

Iscan supports line editing which is similar to EMACS in UNIX. The current line, or any of the previous 20 lines, can be edited by using the familiar EMACS cursor movement and delete key strokes. The following keys are defined for line editing in Iscan:

- <- (left arrow) or CTRL **B** — Move back one character on the line
- -> (right arrow) or CTRL **F** — Move forward one character on the line
- CTRL **E** — Move to the end of the line
- CTRL **A** — Move to the beginning of the line
- CTRL **D** — Delete the character under the cursor
- CTRL **R** — Redisplay the line
- CTRL **U** — Delete the whole line
- BACKSPACE or DEL — Delete backward in the line

3.5.12 Comments

Comments may be placed in an Iscan file by prefacing them with a pound sign (#). Any input on a line after and including a # will be thrown away by the lexical analyzer. Thus, a # will cause all further input up to the next new line character to be ignored.

3.6 Screen Specifications and Screen Mode

The screen mode is invoked using a script that specifies the parameters of the screen which Iscan displays.

3.6.1 Screen Field Specification

Iscan allows displaying and editing of scan ring fields in groups with the *edit* command. The user decides which scan ring fields to display, where to displayed them, and their format. This information is communicated to Iscan with the *screen* command. All displayable quantities in Iscan have a maximum size of 32 bits, although smaller meaningful fields also exist. An output region (I/O display field) may be defined for any size from 1 to 32 characters wide. In order for Iscan to output the information it must be provided with the following:

- The name of the field being displayed as the underlying library routines would recognize it, or the name of a function that will return a 32-bit quantity field to be displayed
- A mnemonic for the output data to allow long field names to be truncated in a friendly manner. The mnemonic will be truncated to the length of the display field as specified in the base descriptor described below.
- The base in which to display the field and the field width, which is a character string that resembles a conversion control string in C. The format of the base or width descriptor is:

  %[width in characters][type]

  The width must be a number from 1 to 32 and the type can be one of the following:

  b — Binary output format

  x — Hexadecimal output format

  d — Decimal output format. (This is probably only useful in cases where the output area is updated by a function and not a field fetch.)
- The synonym field if a synonym list has been supplied
- The starting column (1–80) for the display field
- The display line upon which to place the field. Since there are 24 lines on a standard terminal and three lines of data (mnemonic, value, synonym) to a display line, there are a maximum of seven display lines plus the command or status region. This value is a number between one and seven.

3.6.1.1 Screen Specification Format

A screen specification has the following format:

screen *Screen name*
{
<Output field description>

.
.

};

Where the output field description is one of the following:

fieldname,[mnemonic],base,[synonym_name],col,line;

or

functionname,[mnemonic],base,[synonym_name],col,line;

> **NOTE**
>
> If a mnemonic is not supplied, the *screen* command defaults to the first ten characters of the field name or the function name.
>
> The mnemonic is optional; the comma holding its place is not.
>
> The synonym name is optional; the comma holding its place is not.
>
> The column and line information are numbers in Iscan and so will default to the hex format unless the numbers are prefaced with a "%D."

For example:

```
screen mem_array
{
# this is on line 1
ma[0]:ecc[0],ma0ecc0,%8x,ecc_syn,%D1,%D1;
# this is on line 2
ma[0]:ecc[1],ma0ecC120,%8x,ecc_syn,%D1,%D2;
# this is on line 3
ma[0]:ecc[2],ma0ecc2,%8x,ecc_syn,%D1,%D3;
}
```

or

```
screen Sregs
{
s0lo,"Sreg0low",%8x,,%D1,%D1;
s0hi,"Sreg0hi",%8x,,%D10,%D1;
s1lo,"Sreg1low",%8x,,%D1,%D2;
s1hi,"Sreg1hi",%8x,,%D10,%D2;
s2lo,"Sreg2low",%8x,,%D1,%D3;
s2hi,"Sreg2hi",%8x,,%D10,%D3;
                .
                .
                .
s7lo,"Sreg7low",%8x,,%D1,%D5;
s7hi,"Sreg7hi",%8x,,%D10,%D5;
}
```

3.6.2 Synonym List Specification

A synonym list is a set of mnemonic symbols that allow Iscan to attach some meaning more easily identifiable by the user to the bits in the field of a scan ring. This feature is only used in the display of fields using the screen mode. The specification of synonym lists is similar to the definition of a structure in the C language. A synonym list has the following format:

synlist *synonym_list_name*
{
"synonym" = *constant* or *list of constants*,
.
.
.
};

Where:

list of constants := *(constant,constant,...constant)*

> **NOTE**
>
> Constants that are not specified will be reported as unassigned.
>
> Synonyms will be truncated to fit the field width.

For example, the fc0-3 bits on an MC68000 would be assigned as follows:

```
synlist f_class
{
        "userdata" = %B001,
        "userprog" = %B010,
        "sup_data" = %B101,
        "sup_prog" = %B110,
        "intr_ack" = (%B000,%B000,%B000), };
```

3.6.3 Screen Display

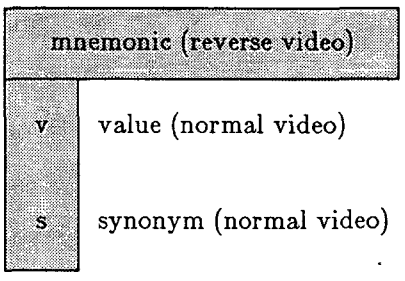
When the screen mode is entered, the screen is divided into two regions:

- Display region where the fields defined in the screen specification are placed. This region is not scrolled.
- Command or status region where nonedit Iscan commands are displayed and status error messages are displayed. This region is scrolled.

3.6.4 Detail of a Screen I/O Field

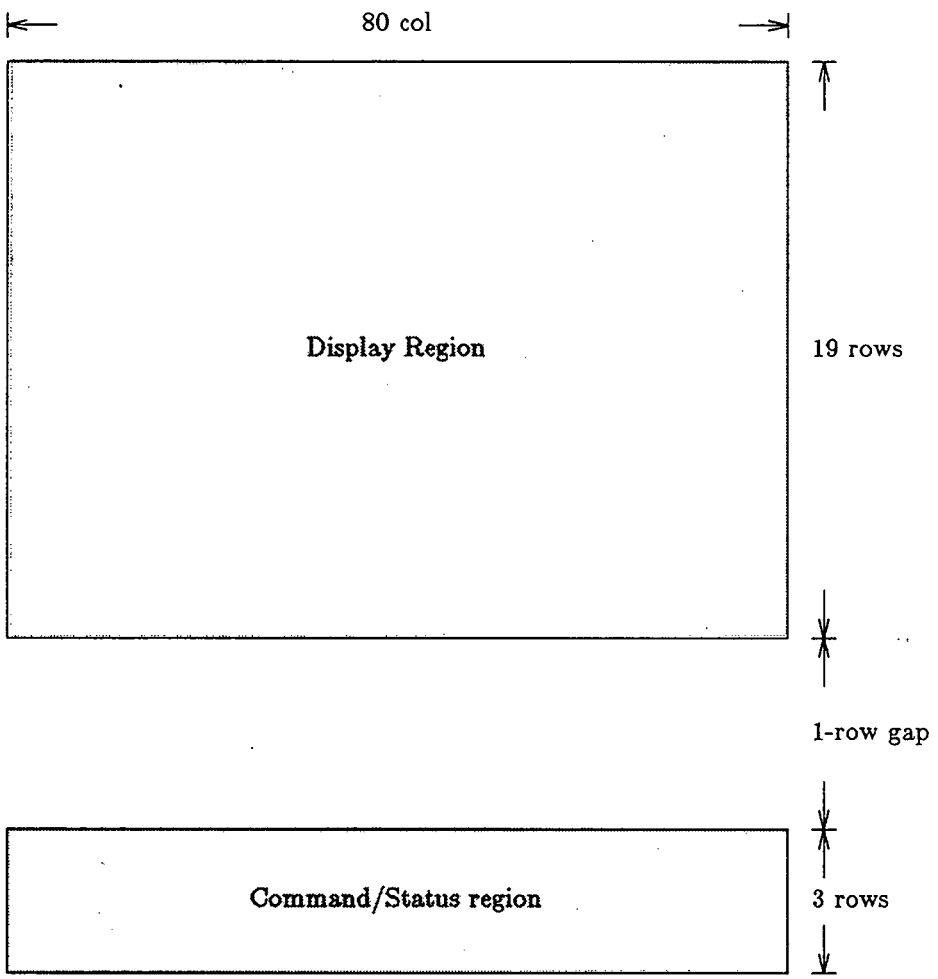
The following figure is a sample I/O screen field:

Figure 3–6, Screen I/O Field, Example



The following figure displays the regions of the video screen:

Figure 3–7, Video Screen Regions



The display region is an area of the video screen that is 80 columns wide by 19 lines long. Depending upon the width of the display fields, there can be a variable number of blocks for the display of I/O fields.

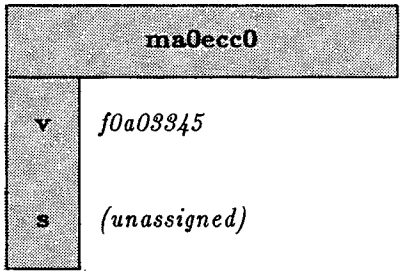
Each display field is set up as follows:

- A 1- to 32-character-wide reverse video region across the top that holds the field mnemonic
- A 1- to 32-character-wide normal video region containing the value of the displayed field
- A 1- to 32-character-wide normal video region with synonym

3.6.4.1 Sample Screen I/O Field

The following figure is an example of a screen I/O field as it would appear if the screen in the example *mem_array* above were displayed. Note that the field example only shows one of the three fields defined in the *mem_array* screen. No synonym table is defined for this field so the synonym entry is empty and therefore contains the string "unassigned."

Figure 3–8, Example of a Screen I/O Field



3.6.4.2 A Screen Definition with Associated Synonyms

The following is a sample screen definition with synonyms:

```
define
synlist d_syn{
    "one"=1,
    "two"=2,
    "three"=3,
    "four"=4,
    "five"=5
    }
synlist cycle {
    "nop"=0,
    "rd"=1,
    "wr"=2,
    "tam"=3
    }
synlist zone {
    "bad"=(0,%b1001,%B1110),
    "xoxo"=%B1010,
    "xxoo"=%B1100,
    "ooxx"=%B0011,
    "oxox"=%B0101,
    "xxxx"=%B1111,
    "ooox"=%B0001,
    "ooxo"=%B0010,
    "oxoo"=%B0100,
    "xooo"=%B1000,
    }
screen mcm {
    mcm[0]:xbar[1].is[0],"%20b",d_syn,%D23,%D1;
    mcm[0]:row_sel[4],"sp_rs","%5b",d_syn,%D1,%D3;
    mcm[0]:adr[4],"sp_adr",%6,d_syn,%D8,%D3;
    mcm[0]:wr_dat[4],"sp_dat","%8x",d_syn,%D16,%D3;
    mcm[0]:wr_par[4],"sp_par","%4b",d_syn,%D25,%D3;
    mcm[0]:cycle[4],"sp_cyc","4b",cycle,&D30,%D3;
    mcm[0]:zone[4],"sp_zon","04b",zone,%D35,%D3;
    }
endefine
```

3.6.4.3 A Sample Function with Step-by-Step Narration

The following function gives a history of the the last 16 memory accesses. It is used to verify the proper accessing of memory, as it displays the ports which requested memory, and the memory bank which was requested by these ports. It uses *fetch* commands to pull one 3-bit field, *px_win*, and one 8-bit field, *m_start*.

These two fields are stored in a single 11-bit word in RAM. The RAM can store a

maximum of 16 of these words. The *px_win* contains the number of one of the four possible ports which requested and then successfully received a memory access after arbitration. The *m_start* contains the number of one of the eight possible memory banks which was requested by the winning port.

3.6.4.4 Sample Function

The following is the sample function:

```
define
func dump_win_q()
{
   :800 = fetch mcm[:901]:win_q_adr
   :801 = fetch mcm[:901]:qpx_win
   :802 = fetch mcm[:901]:qm_start
   :803 = fetch mcm[:901]:rpx_win
   :804 = fetch mcm[:901]:rm_start
 print "— current register contents —\n"
 print "win_q_adr   qpx_win   qm_start   rpx_win   rm_start\n"
 print "   %1x         %1x         %2x" :800,:801,:802
 print "         %1x         %2x\n" :803,:804
 save mcm[:901]:
 print "\n— contents of the win queue—\n"
 print "    adr    win   start\n"
 clock -do mcm[:901]:
 for :805 = 1,%D16 do
   {
              :800 = fetch mcm[:901]:win_q_adr
              clock -do mcm[:901]:
              :801 = fetch mcm[:901]:rpx_win
              :802 = fetch mcm[:901]:rm_start
              print "     %1x      %1x      %2x\n" :800,:801,:802
   }
   restore mcm[:901]:
}

enddefine
```

3.6.4.5 Narrated Description of a Function

The following is a narrated description of the function listed above.

```
func dump_win_q() {
    :800 = fetch mcm[:901]:win_q_adr
```

This line is used to fetch the *win_q_adr* field and place in in register 800. The *win_q_adr*

field is the RAM pointer or the address in RAM where the *px_win* and *px_start* fields begin.

:801 = fetch mcm[:901]:qpx_win

This command places the *qpx_win* field in register 801. The *q* indicates that the field has not yet been put into RAM. The *px_win* field is the number of the port that requested a memory access.

:802 = mcm [:901]:qm_start

This command fetches the *qm_start* field, which holds the number of the memory bank which was requested by the port which prefixes it.

:803 = fetch mcm[:901]rpx_win

This reads the *px_win* field after it has been pulled from RAM. The *r* indicates that the field has been pulled from RAM.

```
:804 = fetch mcm[:901}:rm_start
print "— current register contents—\n"
print "win_q_adr   qpx_win   qm_start   rpx_win   rm_start\n"
```

These two lines print the title *current register contents* and then prints on the following line the name of each field that was fetched.

```
print "   %1x        %1x        %2x" :800,:801,:802
print "       %1x       %2x\n" :803,:804
```

The number after the percent sign is the field width. The *x* after each field width specifies hexadecimal as the number system in which the bits of the fields will be displayed. The registers that correspond to each field follow the field width assignments.

save mcm[:901]:

This saves the entire memory array ring to a buffer.

```
print "\n— contents of the win queue—\n"
print "   adr   win   start\n"
```

These two lines set up a print label, *contents of the win queue*, and then print a column which names the *adr*, *win*, and *start* fields. Printed directly under each label will be the bit value of the field. These values will print after the clock command on the next line has been executed.

clock -do mcm[:901]:

This command causes a single clock to the memory array ring. The *-d* flag causes Dmode, the system diagnostics mode, to be asserted. The *-o* flag causes the *load* pattern to be applied to the Iscan control lines. This is used for parallel loads of the ring.

```
for :805 = 1,%D16 do {
```

This line indicates that the commands within the brackets are a loop which is to be performed 16 times, or once for each space in RAM.

```
:800 = fetch mcm[:901]:win_q_adr
clock -do mcm[:901]:
:801 = fetch mcm[:901]:rpx_win
:802 = fetch mcm[:901]:rm_start
print "     %1x     %1x     %2x\n" :800,:801,:802
  }
```

When these commands are executed, the RAM queue address is fetched, and the numbers of the winning port and the requested memory bank are clocked out, one at a time, until all 16 are fetched and then printed in columns.

```
restore mcm[:901]:
```

This last command restores the memory array ring in a buffer.

```
endefine
```

This last line is always used to end a program.

3.7 Program Invocation

Iscan is invoked by entering **iscn**, followed by any screen mode script names, in the following format:

iscn [*script1,script2,script3...*]

This returns a header that contains initialization and update information on Iscan changes. The following prompt will then be displayed:

```
iscn==>
```

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4
Diagnostic Utilities

4.1 Overview

This chapter describes publicly accessible diagnostic commands in alphabetical order.

THIS PAGE INTENTIONALLY LEFT BLANK

NAME

intro – introduction to commands

DESCRIPTION

This section describes publicly accessible diagnostic commands in alphabetic order.

CROSS TOOL REFERENCES

References are made to cross tools and libraries that were used in the development of the Service Processor Unit (SPU) software. The cross tools are available only on the diagnostic development system and are not available under SPU UNIX. The cross tool information is included for those doing diagnostic software development. Cross tool references are indicated by lists of files preceded by the word **HOST:** under the **HOST LIBRARIES,** or **SEE ALSO** sections, or by the section **HOST LIBRARIES.** File lists not preceded by the word **HOST:** specify files that are available under SPU UNIX.

DIAGNOSTICS

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of 'normal' termination) one supplied by the program (see **wait(2)** and **exit(2)**). The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously 'exit code,' 'exit status,' or 'return code,' and is described only where special conventions are involved.

NAME

boot_iop – program to coldstart boot an IOP or VIOP and download an object file to it

SYNOPSIS

boot_iop -Xrtq object_file

DESCRIPTION

Boot_iop will initiate a coldstart boot of the selected IOP or VIOP.

The options to the program are:

X is the CCU number to load. The only valid CCU numbers are 3 through 7.

r will signal that the object is to be loaded at the entry point in the object file, but the CCU will simulate a reset interrupt to enter the code.

t will indicate that the IOP will be commanded to execute the EPROM selftest code before any load is started. This option is not available for VIOP's.

q will suppress any "normal" information printouts, but will not stop the error message printouts.

NAME

commreg – interactive communication register utility

SYNOPSIS

commreg [<**anything**>]

DESCRIPTION

The **commreg** interactive utility allows examination or modification of communication registers and their associated lock and modified bits. All operations are performed by scanning and clocking the CPU utilities board(s). When performing communication register operations, DMODE is normally asserted. The **wr_cmod** operation, however, requires the communication registers to be clocked normally (without DMODE), so care should be exercised when writing modified bits, otherwise state may not be preserved.

Normally, **commreg** prints only the results of the requested operation. If, however, **commreg** is invoked with any arguments on the command line, a verbose printing option is enabled. This option allow the printing of additional information which may be useful in debugging a malfunctioning set of communication registers

After invocation, **commreg** prompts for a command. Each line is a separate command, and the input varies with the command, as indicated in the list of commands below. All numbers must be entered in hex, *without* a leading "0x." Communication register addresses may be specified in one of three ways: RAM address (0 - 0x3ff), physical address (0x3c00 - 0x3fff), or logical address on the basis of a CIR. In logical mode the address is entered with a one-digit CIR number (0 - 7), immediately followed by a colon (:), then the logical address for that CIR (0x0 - 0x1f, 0x4000 - 0x401f, or 0x8000 - 0x803f).

Operation	**Description**
tst <reg>	Display lock bit status
lck <reg>	Lock or set (to 1) lock bit
ulk <reg>	Unlock or clear (to 0) lock bit
put_l <reg> <u data> <l data>	Write all 64 bits
get_l <reg>	Display all 64 bits
snd_l <reg> <u data> <l data>	Write all 64 bits
rcv_l <reg>	Display all 64 bits
put_w <reg> <l data>	Write lower 32 bits
get_w <reg>	Display lower 32 bits
snd_w <reg> <l data>	Write lower 32 bits
rcv_w <reg>	Display lower 32 bits
put_u <reg> <u data>	Write upper 32 bits
get_u <reg>	Display upper 32 bits
snd_u <reg> <u data>	Write upper 32 bits
rcv_u <reg>	Display upper 32 bits
wr_cmod <reg> <mod data>	Write associated modified bit
rd_cmod <reg>	Display associated modified bit
rest <reg> <u data> <l data> <lock data>	Restore specified register
d[ump | isplay] [<reg> [<end_reg>]]	Display one, a range, or all registers, lock bits
e[xit]	Exit or quit
q[uit]	Exit or quit

NAME

config_chk – check machine configuration and print problems and system configuration

SYNOPSIS

config_chk [**–cfmtuv**]

DESCRIPTION

The **config_chk** utility checks the boards installed in a system against desired and required configuration constraints to determine if the configuration is valid.

Normally, **config_chk** prints the current processor configuration (type, number, availability), then checks specific items, such as board combinations, to ensure compatibility. Warning or fatal error messages will be printed if an illegal condition is present.

The following options are available:

–c Don't print the configuration information.

–f Don't check for Functional Unit (FU)/Data Cache (DC) compatibility. Normally, checks to ensure that all installed FUs and DCs are either Enhanced Functional Units (EFUs) and Enhanced Data Caches (EDCs) or Scalar Functional Units (SFUs) and Data Cache Units (DCUs). Class four machines are required to have EFUs and EDCs.

–m Don't check memory timing configuration.

–t Don't check service processor clock. Insures reasonable time and date on service processor.

–u Don't check VPC compatibility. All 1205 VPCs or all 2205 VPCs are required.

–v Verbose mode; print what is being checked.

A negative number is returned if a fatal problem was encountered during execution.

NAME

cop – display board identification information

SYNOPSIS

cop [-vem] slot ...

DESCRIPTION

With no options, **cop** displays the slot name, board type, part number, serial number, and board revision of the boards in the specified slots. Proper board and slot matching is also checked, although the C130, C210, and C220 machines should not power up if any boards are installed incorrectly. If a mismatch is detected, an error message indicating the mismatch is printed, and **cop** returns a negative one. Normally, **cop** returns a zero.

The two lists below contain mnemonics that may be used to specify slots. The first list contains backplane-independent terms; the second list contains a column for each type of backplane. When invoking **cop**, any mnemonic from the first list, but only mnemonics from the appropriate column of the second list, may be used.

Term	**Description for All Backplanes**
all	All slots
mcm	All memory slots
mcme	All even memory slots
mcmo	All odd memory slots
mcm0	All memory 0 slots
mcm1	All memory 1 slots
mcm2	All memory 2 slots
mcm3	All memory 3 slots
cpu	All CPU slots
cpua	All CPU A slots
cpub	All CPU B slots
cpuc	All CPU C slots
cpud	All CPU D slots
asp	All ASP slots
dcu	All DCU slots
sfu	All SFU slots
ipp	All IPP slots
vpc	All VPC slots
vpd	All VPD slots
io	All I/O and support slots
pi	All PBUS interface slots
cu	All CPU utility card slots
ccu	All CCU slots

1	**2**	**4**	**Description for 1, 2, or 4 CPU Backplane**
	mo0	mo0	Memory 0 odd slot
	me0	me0	Memory 0 even slot
	mo1	mo1	Memory 1 odd slot
	me1	me1	Memory 1 even slot
	mo2	mo2	Memory 2 odd slot
	me2	me2	Memory 2 even slot
	mo3	mo3	Memory 3 odd slot
	me3	me3	Memory 3 even slot
	aspa	aspa	CPU A ASP slot
	dcua	dcua	CPU A DCU slot
	sfua	sfua	CPU A SFU slot

	ippa	ippa	CPU A IPP slot
	vpca	vpca	CPU A VPC slot
	vpda	vpda	CPU A VPD slot
asp	aspb	aspb	CPU B ASP slot
dcu	dcub	dcub	CPU B DCU slot
sfu	sfub	sfub	CPU B SFU slot
ipp	ippb	ippb	CPU B IPP slot
vpc	vpcb	vpcb	CPU B VPC slot
vpd	vpdb	vpdb	CPU B VPD slot
		aspc	CPU C ASP slot
		dcuc	CPU C DCU slot
		sfuc	CPU C SFU slot
		ippc	CPU C IPP slot
		vpcc	CPU C VPC slot
		vpdc	CPU C VPD slot
		aspd	CPU D ASP slot
		dcud	CPU D DCU slot
		sfud	CPU D SFU slot
		ippd	CPU D IPP slot
		vpcd	CPU D VPC slot
		vpdd	CPU D VPD slot
pia	pia	piy	PBUS Y interface slot
		pix	PBUS X interface slot
cpx	cpx		CPU utility card slot
		cue	Even CPU utility card slot
		cuo	Odd CPU utility card slot
ccu0	ccu0	ccu0	CCU 0 slot
ccu1	ccu1	ccu1	CCU 1 slot
ccu2	ccu2	ccu2	CCU 2 slot
ccu3	ccu3	ccu3	CCU 3 slot
		ccu4	CCU 4 slot
		ccu5	CCU 5 slot
		ccu6	CCU 6 slot
		ccu7	CCU 7 slot

If the **-v** option is specified, board and slot matching is verified, but board identification information is not displayed.

The **-e** option causes information about the manufacturing test level and missing assembly revisions to be added to the standard board identification information.

The **-m** option displays only the manufacturing test level code for the named slots. No other information is displyed. This option is intended for use in scripts.

FILES

/mnt/usr/lib/DB_cop

NAME

cpureg – initialize or display central processor non-vector register state

SYNOPSIS

cpureg [**-c #/all**] [**-i** [**nnnn**]]

DESCRIPTION

The **cpureg** utility displays the contents of the C130, C210, and C220 central processor non-vector register state to *stdout*. This utility also has an option to initialize the non-vector registers to a known state. The initialize option **-i** writes the following registers:

a0-a7	address registers
s0-s7	scalar registers
t0-t7	temporary registers

Selecting a constant value (*nnnn*), causes **cpureg** to initialize the address, scalar, and temporary registers to the specified value. When the initialization option is selected without an initialization value, a predefined pattern is written to the registers. Following initialization, all the non-vector registers are displayed.

The CPU option, (**-c**), allows the user to select the CPUs to initialize or display. The default is all the CPUs in the current system configuration. When a specific CPU is selected by the user, a check is made that the specified CPU is in the current system configuration.

The following registers are displayed:

pc	program counter register
psw	program status word register
upc	micro program counter register
ccr	CPU control register
a0-a7	address registers
s0-s7	scalar registers
t0-t7	temporary registers

The default operation for this utility is to display the registers for all the CPUs in the currect system configuration.

SUBSYSTEMS AFFECTED

The **cpureg** utility only affects those CPUs selected by the user. No other subsystems should be affected by the invocation of this utility.

SEE ALSO

reg_dump(3d)
regrd(3d)

NAME

cpuvreg – initialize or display central processor vector register state

SYNOPSIS

cpuvreg [**-c#**[,#]*][**-v#**[,#]*][**-s#**][**-n#**]

DESCRIPTION

The **cpuvreg** utility displays the contents of the C200 Series central processor vector register state to *stdout*. This utility has five options to initialize the vector registers to either a predefined value or a user-supplied value:

-c This option allows the specification of which CPU vector processor to manipulate. If this option is omitted, then the default is to process each of the installed and available CPU vector processors in the system.

-v This option allows the identification of which vectors of the vector processor to read or write. If this option is omitted, the default is to process all vector registers for the specified CPU vector processors.

-s This option identifies the starting element within a vector register to process. The default is to begin with element zero.

-n This option defines how many vector elements to process. The default is to process all 128 elements of the vector.

-i This option indicates the type of operation. If this option is not present, **cpuvreg** reads the vector registers. If this option is present, **cpuvreg** initializes the vector registers. The user may include a value that will be installed into each specified vector element, or the user can omit the value and the vector elements will be initialized to default values.

EXAMPLE USAGES

cpuvreg -c1 -v0 -s3 -n2: The invocation of this command displays elements 3 and 4 of vector register 0 in CPU 1.

cpuvreg -c0,1 -v0,3,4 -n2: The invocation of this command displays elements 0 and 1 of vector registers 0, 3, and 4 in CPU 0 and 1.

cpuvreg -c0 -v0 -i10: The invocation of this command initializes vector register 0 in CPU 0 to the 64-bit value 10. The absence of a leading 0x on the initializing value causes **cpuvreg** to interpret the number as a decimal.

cpuvreg -i0x123456789abcdef: This command causes all vector registers in each installed and available CPU to be initialize to the hex value 0x123456789abcdef.

SUBSYSTEMS AFFECTED

The **cpuvreg** utility only affects those CPUs selected by the user. No other subsystems should be affected by the invocation of this utility.

SEE ALSO

vregrw(3d)

NAME

cs – load the C2 writable control store(s)

SYNOPSIS

cs [-l] [-v] [-c [#] [all]] [-e nnnn]
[-us/sr/ua/ul/um/vd [file]] [-sc] [-vp] [-d [n1 [n2]]]

DESCRIPTION

The **cs** utility is used to load, verify, and display the contents of the C2 writable control stores. This command defaults to loading and verifying the control store images for all CPUs using the default file path names for each of the control store types. The microcontrol store types include:

us - Scalar, located on AS.

sr - Scratch RAM, located on AS.

ua - Addition or subtraction, located on VC.

ul - Load or store, located on VC.

um - Multiplication or division, located on VC.

vd - Vector dispatch, located on VC.

The control store image files must be in *b.out* format with the image located in the text segment and a 32-bit checksum located in the data segment. The control store image files are named xxxx.yyy, where xxxx is the control store identifier (i.e., **us**, **sr**, **ua**, **ul**, **ul**, **vd**), and yyy is the file extension.

The file extensions are specified in the file */mnt/boot_db*. An entry in this database file specifies the filename extension of each of the control store image files. The entries in the database are set up automatically at power-up by *.diaginit* and *scn_util*. If the file name extensions cannot be found in the database, the extension *.wcs* will be used as a default extension.

The **cs** utility first looks in the current directory for the control store image files to be used. If the control store image files are not found in the current directory, **cs** searches in the directory */mnt/usr/ucode* before giving up. Once the desired control store image files are found, **cs** relates this information to the user.

When executed, **cs** halts the central processors specified, loads the required files into the Service Processor Unit (SP2) memory, performs a checksum verification on the file contents, and then loads the file images from SP2 memory to the required control stores.

After loading is complete, **cs** reads the content of the specified control stores and compares them with the file images in SP2 memory. If the control store contents fail to compare, the default is to log a maximum of five errors per control store type before **cs** aborts the comparison. If an error is detected, **cs** returns a -1. Otherwise, **cs** returns 0.

The **cs** utility also verifies that the ring revisions of the CPUs are compatible and checks that all the CPUs selected by the user have the same ring revision, since **cs** uses features of the *scn (3d)* package to perform identical operations on all the user-selected CPUs.

The options recognized by **cs** allow the user to modify the default options used by the program. If no options are specified, the default is **-lv -c all -e 5** for all the control stores (**us**, **sr**, **ua**, **ul**, **um**, and **vd**) using the default filenames for each control store.

-us Allows the user to specifically load, verify, and dump the scalar microcontrol store located on the ASP.

-sr Allows the user to specifically load, verify, and dump the scalar scratch RAM located on the ASP.

-ua Allows the user to specifically load, verify, and dump the vector add/subtract microcontrol store located on the VPC.

-ul Allows the user to specifically load, verify, and dump the vector load/store microcontrol store located on the VPC.

-um Allows the user to specifically load, verify, and dump the vector multiply/divide microcontrol store located on the VPC.

-vd Allows the user to specifically load, verify, and dump the vector vector dispatch microcontrol store located on the VPC.

-sc Allows the user to specifically load, verify, and dump all the scalar microcontrol stores located on the ASP. This option is an alias for **-us -sr**.

-vp Allows the user to specifically load, verify, and dump all the vector microcontrol stores located on the VPC. This option is an alias for **-ua -ul -um -vd**.

File Allows the user to specify an alternative file path name for a single microcontrol store. This option only applies to a single control store. Only the root name of the file should be specified on the command line. The filename suffix is determined by **cs**.

-d This display option is only valid for single control store accesses. The default is to display the whole control store. Options are provided to limit the display to an area of interest or a single address. An error is displayed and an error code of -2 is returned when the **-d** option is used invalidly.

-l Load only. Does not verify the contents of the control stores after loading.

-v Verify only. Compares the current contents of the control stores to the contents of the appropriate control store image files.

-c Specifies the CPU on which operations are performed. The default for this option is to perform the designated operations on all CPUs in the current configuration. The user may specify that the designated operations are to be performed on a single specific CPU, which *must* be in the current configuration.

-e Specifies the number of errors to report during verification. The default for this option is to report five errors prior to aborting verification of the faulting control store. This option allows the user to raise or lower that maximum.

Some combinations of the options are not allowed. To indicate an alternative control store image path name, specify the control store desired. To display the contents of a control store, specify the control store desired. Additionally, the display option cannot be used in conjunction with the **load** and **verify** options.

SEE ALSO

diaginit(1d)
scn_util(1d)
scn(3d)
b.out(5)

NAME

dcache – dump the data cache

SYNOPSIS

dcache [**-c #/all**] [**-m/r**] **-dh** [**n1** [**n2**]]

DESCRIPTION

The **dcache** utility permits the contents of the data cache to be dumped to *stdout* as either RAM addresses or as logical addresses.

The default operation of **dcache** is to dump the entire cache of all the CPUs in the current configuration in logical address mode.

The **dcache** utility allows the user to specify, via the **-c** option, the CPUs on which to perform the cache operations. The default is to perform the user-specified operations on all CPUs in the current configuration. When a specific CPU is requested, **dcache** verifies that the CPU is in the current configuration before attempting any dump operations.

The specified central processors are halted (see description of the **-c** option described above for details), and the contents of the data cache are dumped to *stdout* in hex format (an option of **-dh**). The default addressing mode for this utility is memory address mode (**-m**). In memory address mode, **dcache** addresses are interpreted as being logical addresses, and bits <3..11> are significant to the utility. The three LSBs are discarded, as are bits <12..31>.

The **dcache** utility also supports the display of addresses in RAM address mode (**-r**). In RAM address mode, the user selects the addresses to display as physical RAM addresses of the cache. In this mode, bits <0..8> are the significant bits of the address. In this address mode, an error is reported if an address is larger than is supported in the hardware.

Selecting **n1** and **n2** allows the user to specify the range of **dcache** addresses to dump. If no addresses are specified, then the entire **dcache** is dumped. If an error is detected, **dcache** returns a -1. Otherwise, **dcache** returns a 0.

NAME

diaginit – diagnostic initialization script

SYNOPSIS

diaginit [**-f**]

DESCRIPTION

The **diaginit** shell script is invoked at SP2 UNIX boot time from *.profile* to initialize the scan ring description files. It also initializes the system configuration file, */mnt/boot_db*. This invocation is performed at boot time while its operation is dependent upon three conditions: power-on bit, changes in system configuration since last power cycle, and the user-supplied forced-initialization parameter.

If the *-f* switch is specified, then initialization is forced; otherwise, initialization only occurs if both the power has been cycled off and the configuration of boards within the system has changed. The **pup** utility determines the state of cycling power and **cop** determines the current set of board revisions.

If the conditions are present for scan ring initialization, then **scnlink** builds a new composite scan definition file, */mnt/usr/scn/scn_rings*. The utility **mminit** is executed to determine the memory configuration so the **scn_util** utility can initialize the system configuration file, */mnt/boot_db*. The **pup** utility the power off bit to false. The file */mnt/usr/lib/Initall_cpu* is removed if initialization is performed.

REFERENCED FUNCTIONS

pup(1d)
cop(1d)
mminit(1d)
ringrev_chk(1d)
scn_util(1d)
scnlink(1d)

FILES

/mnt/usr/scn/scn_rings
/mnt/usr/scn/scn_rings.old
/mnt/usr/lib/Initall_cpu
/mnt/errlog
/mnt/bin/config_chk
/mnt/usr/scn/cop.new
/mnt/usr/scn/cop.old
/mnt/boot_db

SEE ALSO

initall(1d)

NAME

dshell – C1 test executive (diagnostic shell)

SYNOPSIS

dshell

DESCRIPTION

The CONVEX Diagnostic Shell (**Dshell**) runs on SP2 version 7, UNIX when the CONVEX central processor UNIX is not running. All CONVEX diagnostics, except standalone and online diagnostics, run under this shell. This allows the user to become familiar with one command interface and set of command options. The **dshell** supports a manual mode of testing that provides low-level pass/fail information.

COMMAND SYNTAX (Manual Mode)

test	display test menu
test testname	execute test 'testname'
test testname -s	display subtest menu, prompt user for subtests
test testname -s n(i,j-k)	execute subtests i,j-k n times
test testname -c	display class menu, prompt user for classes
test testname -c n(i),j-k	execute classes i n times & j-k
pause off (-f) (-b) (-e)	disables all or specific pauses
pause -f	pause on all failures
pause -f nnn	pause on failures in subtest nnn
pause -b	pause at beginning of all subtests
pause -b nnn	pause at beginning of subtest nnn
pause -e	pause at end of all subtests
pause -e nnn	pause at end of subtest nnn
continue (or carriage return)	continue test from pause
msgs off (-f) (-s) (-t)	disables all or specific messages
msgs -f (long/short)	enable long/short fail messages
msgs -s	enable subtest result messages
msgs -t	enable test result messages
log off (-s) (-t)	disables multiple fail modes
log -s nn	allow nn failures per subtest
log -t nn	allow nn subtest failures per test
loop off (-s) (-t)	disables loop modes
loop -s nnn	loop on subtest nnn
loop -t	loop on test
status	print test flag status
exit	exit from Dshell
quit	exit from Dshell after all executing and/or queued tests have finished execution
^C	command level is restored to the user if no tests are executing. If any tests are running, a menu of abort procedures will be displayed.

^B	aborts Dshell and all executing or enqueued tests
any command -h	print help information on command
help	print help information for all Dshell commands
!	executes any UNIX command that immediately follows

COMMAND EXPLANATIONS (Manual Mode)

The **test** command causes a test to be executed. If no testname is given, the user is shown a menu of available tests and is asked to choose one. This menu is automatically generated the first time it is requested. If no option is specified, then all subtests are performed once in ascending numerical order. The subtest option **-s** and class option **-c** allow for selective, multiple and/or sequenced execution of available subtests and classes. If alone in the command line, they cause a menu of all subtests or classes in the chosen test to be displayed. The user is then prompted for his choices. Correct syntax for executing subtests or classes (via **-s** or **-c**) is the same whether entered in the command line, or as a response to a menu. Examples follow:

-s	display subtest menu
-c	display class menu
-s 10,15,5(20)	executes subtests 10,15,20,20,20,20,20
-s 3(10,15)	executes subtests 10,15,10,15,10,15
-c 2(13-10)	executes classes 13,12,11,10,13,12,11,10

The **-c** and **-s** options may not be mixed. Subtests or classes which are listed but do not exist will be ignored. When responding to a menu, the appropriate option (**-s** or **-c**) is assumed, and should not be included in the input stream. Test input and output may be redirected via the command line as follows:

<filename	test input from file "filename"
>filename	test output to file "filename"
+>filename	test output to both file "filename" and terminal

The **pause** command enables pauses at specific points in a test. The **-f** option causes a pause on every fail if alone, or at every fail in a specified subtest. The **-b** option causes a pause at the beginning of every subtest if alone, or at the beginning of a specified subtest. The **-e** option causes a pause at the end of every subtest if alone, or at the end of a specified subtest. When an executing test reaches a pause condition, it halts execution and prompts the user with the **dshell** prompt ": ". The user may then enter any SP2 UNIX command via the "!" directive or any **dshell** command. To continue test execution, the continue command (carriage return) should be entered in response to a **dshell** prompt. The default pause status is all pauses off.

The **continue** command is a *nop* if not executed at a test pause.

The **msgs** command allows the user to adjust the pass/fail messages output by test programs. The **-f** (long/short) option enables either long or short error messages. The **-s** option enables subtest/class result messages. The **-t** option enables test result messages. The default for these flags is long fail, subtest, class, and test messages turned on.

The **log** command allows the user to enable/disable multiple failures. The **-s** option enables the specified number of multiple failures per subtest. The **-t** option enables the specified number of multiple failures per test. The default for these flags is all multiple failures turned off.

The **loop** command allows consecutive execution of a test or subtest. The **-s** option enables

consecutive looping on the specified subtest. The **-t** option enables consecutive looping on a test. **^C** functions the same as in *exit*, but does not reset either of these flags. The default for these flags is all looping turned off.

The **status** command displays the current condition of all **dshell** test control flags on the screen, and describes the effect each has in its current configuration.

The **exit** command causes termination of all paused tests and an exit from the **dshell**. The **quit** command waits for any currently executing or queued tests to complete execution, and then exits the **dshell**. **^C** will restore command level to the user if no subtests are being executed. If any subtests are running, a menu of abort procedures is displayed. **^B** aborts the currently executing test and quits the **dshell**. There is a another important difference between **exit**, **quit**, **^C** and **^B**. Commands **exit**, **quit** and **^C** all allow an executing test to finish executing *protected code*. **^B** kills the test regardless of the type of code which is currently executing. Thus, a **^B** can leave the machine in an undefined state, necessitating a reboot.

The **help** command gives a brief syntax and command explanation of all **dshell** commands. The **!** directive is an escape to SP2 UNIX, and allows for execution of one command which must immediately follow the **!**. An example would be **!mm**.

SEE ALSO

libtest(3D)

DIAGNOSTICS

Error messages should be self explanatory. Most deal with errors in user commands. Some deal with real system problems.

BUGS

^C after a test command has been entered, but before the SP2 has successfully forked the test process may not kill the test process.

NAME

errintd – error interrupt daemon and logger

SYNOPSIS

errintd [–ehs] [–c nn] [–r nn] [–f FILE]

DESCRIPTION

The **errintd** utility monitors a CONVEX C200 Series computer system for hardware error conditions. Three classes of errors are detected by **errintd**: environmental, soft, and hard errors.

Environmental errors are related to the system operating environment. Typical environmental errors are temperature out of range, loss of airflow, and excessive current consumption.

Soft errors are usually correctable errors (i.e., the error is transparent to the system user). Soft errors include single-bit memory system errors, Central Processor Unit utility board (CPX) reference and modify parity errors, and a variety of Peripheral Interface Adapter (PIA) PBUS transfer errors.

Hard errors include parity errors, internal references to non-existent memory, etc. Hard errors always result in the immediate halt of the Central Processing Unit (CPU) and are, therefore, fatal.

In some cases, both environmental and soft errors can be fatal.

In addition to monitoring for errors, when started, **errintd** starts the main memory sniffer (**mm_sniff(1d)**).

In the normal operating environment, **errintd** is started by the Operating System (OS) SPU utility */mnt/os/prtlog*. All output from **errintd** is time-stamped by */mnt/os/prtlog*, and copied to both the system console and the SPU file */mnt/errlog*.

Single-bit memory error reports are not written to *stdout*. Instead, a record of these errors is maintained in the file */mnt/softlog*. 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 softlog file.

Once the softlog file has reached 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.

When an environmental error is detected, **errintd** will continue to report the error once a minute until the error is corrected. An unattended environmental error can result in a hard error.

In the event of a hard error, **errintd** will log the error and exit. A copy of the last hard error is kept in the file */mnt/hardlog*.

The following options are interpreted by **errintd**:

–e Log environmental errors.

–h Log hard errors.

–s Log soft errors.

Note:

If one of the previous three options is not specified (i.e., **–e**, **–h**, or **–s**), **errintd** will default to all three types being logged.

–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, the sniffer will not be started.

–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.

-f FILE Sends **errintd** output to *FILE*. By default, output goes to *stdout*.

SEE ALSO

hard_logger(1D)
mm_sniff(1D)
softlog(5D)

NAME

fs – field service script to execute diagnostics from a database of scripts and diagnostics

SYNOPSIS

fs

DESCRIPTION

The field service script utility **fs** resides on the SPU disk in the directory */hw/field*. It allows a group of diagnostics to be executed under a single script name.

This utility executes on the C200 series (except C210A) and is dependent on SPU UNIX Ver 5.1 or greater and Systems Diagnostics Ver 3.0 or greater. It is advisable to execute the diagnostic initialization script **.diaginit -f** prior to the field service script.

The following scripts, designed to functionally test each subsection of the system, are available by entering **fs** within the */hw/field* directory:

Script	**Subsection**
0) CPU	Scalars/Vectors
1) CPU-QUICK	Scalars/Vectors
2) INSTALL	All
3) IO	I/O
4) MEMORY	Memory (MCMs)
5) SPRINT	Scalars/Vectors/Chained Mode

The INSTALL script should be executed at installation time. At least one of the above scripts should be executed prior to returning a failed printed wiring board from the field.

When returning a printed wiring board to CONVEX Manufacturing, the **fs** generated report(s) should also be returned. The reports generated are *report.fail* for the failing case and *report.pass* for the passing case. They provide useful information to CONVEX Product Engineering about the failed printed wiring board in the field.

A hard copy of each report is obtained as follows:

1. Set the console printer to **ON** and ensure it is online
2. Enter **Ctrl-Enter** or **Ctrl-PF4** on the keyboard
3. Enter **cat report.fail** or **cat report.pass**

There may be variations in the time required to execute a script, as certain diagnostic tests within a script may be skipped if the system is not configured to execute them. The amount of memory in a system also affects the script execution time.

To abort a diagnostic within a script, enter **Ctrl-C**. Enter a second **Ctrl-C** to exit from the script. To resume an exited script, enter **fs resume**. Margining is done manually before executing any of the **fs** scripts.

For systems with Data Cache Unit (DCU) assemblies at Rev 1 and below, or systems with Eagle Data Cache (EDC) assemblies at Rev e and below, Ver 3.1 Diagnostics of *cpu4231* and *cpu4241* are executed from directory */hw/field/test* instead of the current diagnostics found in directory */mnt/test*.

EXAMPLE USAGES

Type the following to invoke **fs**:

cd/hw/field

fs

A menu appears with a numbered list of available script options. Enter the number corresponding to the desired script. Next enter the FE name in response to `Enter FE name:`; then enter the customer name and system serial number in response to `Enter Customer Name:`.

The selected script begins execution and checks for failures. Upon test completion, either a `No Failure. End of Script` message is displayed with the output written to *report.pass* or a `Failure Detected. Writing Report` message is displayed with the output written to *report.fail*.

To build a script, proceed as follows:

1. Create a directory in */hw/field* with the same name as the new script
2. Add the new script name to the */hw/field/dbase/avail* file
3. Create a *ptests* file in the script name directory to contain the diagnostics and options in mnemonic form. The available options are found in the *Options* file. To create an option not currently in the *Options* file, contact CONVEX Product Engineering.
4. List the diagnostics to execute, one diagnostic per line, using the names found in the *Options* file. To margin a specific diagnostic, enter a four letter mnemonic for the clock, psm2, psm45, and psp5. For example, **cpu4030 ulln** margins upper clock, lower psm2, lower psm45, and nominal psp5.

To create a script called SAMPLE, proceed as follows:

1. **cd /hw/field**
2. **mkdir SAMPLE**
3. Edit the */hw/field/dbase/avail* file to add *SAMPLE* to the script list
4. Edit the *SAMPLE/ptests* file to list the diagnostics and options

Note that any margin conditions selected remain at that margin until changed.

FILES

/hw/field/README

NAME

get_defects – read manufacturer's defect map from an SMD drive

SYNOPSIS

get_defects [-v] [-s serial_number] [-d ioconfig_number]
[filename]

DESCRIPTION

The **get_defects** utility reads the defect data recorded by the drive manufacturer on SMD drives and stores the data in an ASCII file that is readable by the disk formatter, *dev4110*.

The following options are supported:

-v If this option is specified, then all writes to the ASCII file are also output to *stdout*.
Default: No output of defects to the display.

-s serial_number
Use this option to enter the drive's serial number. The number can consist of numbers and dashes and must be from 5 to 31 characters inclusive.
Default: Prompted for the serial number by **get_defects**.

-d ioconfig_number
This option selects the entry number for the drive in the */ioconfig* file. Do not specify this option if the entry number is not known for certain.
Default: The */ioconfig* entries are displayed, prompting for a selection.

filename
This option specifies the ASCII file to which the defect data will be written.
Default: Prompted for the filename by **get_defects**.

FILES

/mnt/bin/get_defects
/mnt/bin/lib/get_defects.x00

The format of the ASCII file created by **get_defects** is:

```
# Serial Number: nnnnnn
n DKD-xxx
d cyl hd bcai len
d cyl hd bcai len
```

where **nnnnnn** is the serial number entered. The **#** means the line is a comment when read by *dev4110*. The second line is the name of the drive. For example, DKD-005 is the name of the NEC 2352 SMD drive. For valid drive names, read the man page on **DB_diskfmt(5D)**. Each subsequent line is one defect from the manufacturer's defect data.

SEE ALSO

DB_diskfmt(5D)

NAME

hard_logger – hard error logger

SYNOPSIS

hard_logger [–FNSv] [–f FILE]

DESCRIPTION

The **hard_logger** utility isolates the source of a hard error. In addition, soft errors can optionally be isolated by the hard logger. Once a hard or soft error is detected, **hard_logger** can be invoked to locate the specific type of error that occurred. However, the actual cause of the error is not necessarily isolated.

In any case, hard errors are always fatal and result in the immediate halt of the Central Processing Unit (CPU). Hard errors include parity errors, internal references to non-existent memory, etc. Soft errors include single-bit memory system errors, internal parity errors, Channel Control Unit (CCU) bus errors, etc.

Upon invocation of **hard_logger**, all system clocks are halted. This is necessary to scan the various scan rings throughout the system.

The following options are interpreted by **hard_logger**:

–**F** Normally the scan rings in a subsystem (i.e., CPU, MEMORY, I/O, etc.) are not scanned if the Error Source Register (**esr**) value (the Soft Error Logger Register [**sel**] value for soft errors) doesn't indicate a pending error in that subsystem. When –**F** is specified, all scan rings will be scanned regardless of the **esr** or **sel** value.

–N This option is used to signal the hard logger that it is being invoked non-interactively. This option changes how some errors are reported. When the hard logger is invoked interactively, it is not known if a hard error has actually occurred. When invoked non-interactively, it is always assumed that a hard error should be present. Only diagnostic tests should use this option.

–S Log soft errors also.

–v Verbose mode. This option prints more information when errors are detected. In addition, it causes the hard logger to print its version number when started.

–**f FILE** By default, the hard logger's output is to *stdout*. If the –**f** option is specified, output is sent to *FILE*. Three special file names have been provided to make **hard_logger** easier to use. These include:

#NN	Send output to file descriptor *NN*
–	Send output to *stdout* (default)
+	Send output to *stderr*

SEE ALSO

errintd(1D)

NAME

icache – load, verify, and dump the instruction cache

SYNOPSIS

icache [**-c #/all**] [**-i nnnn**] [[**-l**] [**-v**] **ifile**]

icache [**-c #/all**] [**-m/r**] **-d**[**hi**] [**n1** [**n2**]]

DESCRIPTION

The **icache** utility loads and verifies the Instruction Cache (Icache) on the Instruction Processor (IPP) of the specified CPUs. The **icache** also permits the contents of the Icache to be dumped to *stdout* as either RAM addresses or as logical addresses. The dump can be performed as either instructions or as the hex contents of the cache.

The default operation of **icache** is to dump the entire cache of all CPUs in the current configuration as instructions in logical address mode.

If the first form of the command is used, **icache** either initializes the cache to a constant value or loads the cache with the data obtained from the object file (**ifile**), which it reads into SP2 memory. Then **icache** halts the CPU of the specified CPUs, purges the Icache of the specified CPUs, and then loads the data from SP2 memory into the Icache beginning at block 0. After loading is complete, **icache** reads the contents of the cache and compares them to either the constant value or the instructions contained in the object file. If the Icache contents fail to compare, a maximum of five errors per CPU are logged before **icache** is aborted. If an error is detected, **icache** returns a -1. Otherwise, **icache** returns a 0.

The **ifile** specifies the name of the object file to be loaded and to be used for verifying the contents of the Icache. The **ifile** must be a Convex **a.out** file with an extension of **.o** (magic number of 507). However, only the root name of the file should be specified on the command line.

The following options are recognized by the first form of the **icache** command:

-c Specify the CPUs on which to perform the **icache** operations. The default is to perform the user specified operations on all CPUs in the current configuration. When a specific CPU is requested, **icache** verifies that the CPU is in the current configuration before attempting any load, verify, or dump operation.

-i Initialize cache to constant **nnnn** where **nnnn** is a 16-bit value (halfword).

-l Load only. Do not verify the contents of the Icache after loading.

-v Verify only. Compare the current contents of the Icache to the specified object file.

If the second form of the command is used, the specified central processors (see **-c** option described above for details) are halted, and the contents of the Icache are dumped to *stdout* in either hex format (an option of **-dh**) or in instruction format (an option of **-di**). The default addressing mode for this utility is memory address mode (**-m**). In memory address mode, **icache** addresses are interpreted as being logical addresses, and bits <3..12> are significant to the utility. The three LSBs are discarded, as are bits <13..31>.

The **icache** utility also supports the display of addresses in RAM address mode (**-r**). In RAM address mode, the user selects the addresses to display as physical RAM addresses of the cache. In this mode, bits <0..9> are the significant bits of the address. In this address mode, an error is reported if an address is larger than is supported in the hardware. Selecting **n1** and **n2** allows the user to specify the range of **icache** addresses to dump. If no addresses are specified, then the entire Icache is dumped. If an error is detected, **icache** returns a -1. Otherwise, **icache** returns a 0.

IDC disks contain a place for a volume name for each disk. The –v option allows the user to specify what name will be put on the disk during initial formatting. If the –v option is not used, a volume name of "NONE" is used. **Idcfmt** does not use the volume name for anything other than initializing the name filed during formatting. The maximum length of the volume name is 32 characters.

–p pattern_file

Normally, during initial formatting, **idcfmt** pattern tests the drive using the set of patterns specified in the database. The normal patterns may be overridden by putting a set of patterns into a file, and using the –p option to tell **idcfmt** where the file is located.

–y The –y option forces the answer to all yes/no prompts to be treated as if the user has responded with a "yes". The initial release of **idcfmt** has only one of these prompts. Refer to the following description of the FORMAT command. This option is useful if the user wants to format a drive in background, and the user does not wish to answer the yes/no prompt interactively. The prompt is there for the user's protection. Think carefully before using the –y option.

COMMANDS

The following commands perform various functions on IDC disks. Each command is listed along with any required arguments specific to that command. Some commands require exclusive access to the disk. The description for each command lists whether or not the command requires exclusive access. The version of **idcfmt** that runs on the Service Processor automatically assumes that it has exclusive access to the disk. The version of **idcfmt** that runs under the ConvexOS coordinates exclusive access to the disk though a diagnostic ioctl() call to the ConvexOS disk driver. Exclusive access to a disk is denied if the disk is currently in use under the ConvexOS file system, or if any process has opened any of the block or character special devices associated with the disk.

The description for some of the commands assume that the user has a basic knowledge of the layout of an IDC disk. A description of this layout follows below.

format Formats the drive if it appears that the drive has not been previously formatted. If the drive appears to be partially formatted, the format resumes at the appropriate point. This command will destroy any previous data on the disk. Exclusive access to the disk is required before this command can execute. This command tries to determine whether or not the drive is formatted by attempting to read the topology map in the topology area. If successful in reading the topology map, the command prints a message and proceeds to the format verification step. The user may override this behavior by using the FORMAT command which is the next command described. If the topology map is unreadable, the program assumes that the drive has not been completely formatted. It then proceeds to check for a partial format of the disk by looking for checkpoint data in the topology and diagnostic areas. If a valid set of checkpoint data is found, the command resumes formatting at the point specified by the checkpoint data. If no checkpoint data is found, the command assumes that the drive has never been formatted and proceeds to format the drive accordingly. The steps required to format a drive are:

o) Identify the type of drive being formatted by performing a read configuration command to the drive. The data returned by the read configuration command is used to search the data base file "DB_idc" for a match. The data base file contains

NAME

idcfmt – utility for IDC disk formatting, media repair, and data editing

SYNOPSIS

idcfmt –d ccu port unit [options]

DESCRIPTION

Idcfmt is a program used to prepare and repair an IDC disk for use under ConvexOS. There are two versions of **idcfmt**. The first version runs on the Service Processor while ConvexOS is not running. The second version runs on CONVEX C2XX series machines while ConvexOS is running. Both versions require the user invoking the program be the superuser.

Idcfmt is invoked by typing its name, a "–d", followed by the CCU slot number, IPI port number, and drive unit number of the drive to be used. A list of options may follow, then zero or more commands. If no commands are given on the command line, **idcfmt** displays a prompt and expects the commands to be entered from the standard input.

OPTIONS

With the exception of the –v option, the following options are not normally required. These options are intended for use under rare, exceptional conditions and then only by CONVEX personnel. The description for the following options make reference to various files used by **idcfmt**. A description of the layout of these files may be found at the end of this document.

–t dkd–999

The –t option overrides the automatic identification of the disk drive that **idcfmt** normally performs. If the –t option is not specified, **idcfmt** queries the target disk drive for its model number. This model number is then used as a search key in the file */mnt/usr/lib/DB_idc*. **Idcfmt** uses the matching entry in this file to determine the geometry of the disk drive. If the –t option is specified, **idcfmt** uses the name provided by the user as the search key into the database. The model number obtained from the drive is then verified against the selected entry. If the model number does not match the one in the database, a warning is printed and **idcfmt** proceeds with the database entry selected by the user. This option is useful only in the unlikely event that there are two drives that return the same model number.

–m manufacturers_defects_pathname

The –m option specifies a pathname to a file which contains the manufacturers defect data. This option is useful in the event that an otherwise usable disk drive has an unreadable defect map. The user may then manually enter the defect data from the paper copy of the defects supplied with each drive into an ascii file and use the –m option to specify the location of the file. Normally, **idcfmt** will read the defect map from the drive anytime it requires knowledge of the defect data. For information on the format of the manufacturers defect file refer to the FILES section within this manpage.

–g grown_defects_pathname

The –g option is used to specify a list of sectors that are known to be bad, but do not appear in the manufacturers defect list. It should be noted that the –g option is not the normal way to handle defects that occur after initial formatting (refer to the description of the "slip" command within this document). During initial formatting, **idcfmt** assumes there are no grown defects unless the user specifies the –g option.

–v volume_name

information about the disk drive such as number of cylinders, number of tracks, encoding schemes, etc. The user may override this automatic identification of the drive by using the -t option on the command line.

o) Read the manufacturers defect data from the drive. The user may override this step by using the -m option described previously in the OPTIONS section.

o) Assume that there are no grown defects for the drive. The user may override this step by using the -g option described previously in the OPTIONS section.

o) Format the topology area with no spare sectors. The topology area is then pattern tested with a set of patterns that are specified in the data base file "DB_idc". The user may override the normal patterns by specifying the -p option on the command line. Once pattern testing is complete, the topology area is re-formatted with the number of spare sectors specified in the data base file "DB_idc". A detailed description of pattern testing can be found in the FILES section of this manpage.

o) Format the user, diagnostic, and spare areas of the disk with no spares. The first checkpoint of the format is written after this step is complete.

o) Pattern test the user, diagnostic, and spare areas of the disk with the patterns specified in the data base file "DB_idc". The user may override the normal patterns by use of the -p option described previously in the OPTIONS section. The process of pattern testing consists of writing each of the specified patterns to the area under test, and then reading each pattern to look for errors that might be attributed to disk media problems. These include data Error Correcting Code (ECC) errors, header Cyclical Redundancy Character (CRC) errors, and missing sector sync byte errors. If any of these errors are encountered, the manufacturers defect list is searched to see if the cylinder, track and sector where the error occured is already listed. If the cylinder, track, and sector is not in the manufacturers defect list, the grown defect list is then searched. If the cylinder, track, and sector where the error occured is not found in either defect list, the location is added to the grown defect list. After each pattern is completed, a checkpoint is written to the disk.

o) Format the user, diagnostic, and spare areas of the disk, reserving a number of sectors at the end of each cylinder for use as spare sectors. Any sector which is listed in either the manufacturers or grown defect lists is re-allocated from this pool of spare sectors. If the number of defective sectors in a cylinder exceeds the number of spare sectors allowed for each cylinder, the disk is not usable under ConvexOS. A checkpoint is written to the disk after this step.

o) Initialize the topology area by writing several copies of the topology data. This includes the topology area map, the defect lists for the disk, a copy of the disks geometries, the ConvexOS label, etc. The contents of the topology area is described below.

o) Clear the diagnostic cylinder.

o) Verify that the drive received a valid format. Refer to the verify command below for a description of what steps are taken to verify that a drive is formatted correctly.

The disk in now formatted and ready for use.

FORMAT

Formats the drive regardless of its current state. This command is the same as the format command listed above, except that if the disk is determined to have a valid, complete disk format, the user is prompted as to whether or not to continue. If the user answers in the affirmative, the format operation proceeds as if the drive had never been formatted. If no valid format is found on the drive, this command does not look for checkpoint data. The format starts from the beginning. All the exclusive access requirements of the normal format command also apply to this command.

verify This command verifies that the disk is properly formatted. This automatically takes place when the drive is initially formatted. This command does not write to the drive and may be used on a drive with user data. Exclusive access to the disk is not required. The following steps are taken to verify that the drive is formatted:

o) Reads one copy of all the data on the topology cylinder. Verifies that the checksums and magic numbers are correct.

o) Reads the remaining copies of the topology data and verifies that they match the data from the first step.

o) Reads all logical blocks on the disk and verifies that there are no header CRC or data ECC errors. If any errors are found they are listed. The user may then attempt to fix the error by using the slip command described below.

o) Reads all the sector headers from the disk and verifies that there is a one to one correspondence of entries in the defect lists with sectors that are slipped.

o) Verifies that each entry in the defect lists point to a sector that has been slipped.

slip cylinder track sector

This command is used to repair header CRC or data ECC errors. If a sector in the user data area of the disk is to be slipped, it can be done without having to have exclusive access to the disk. Sectors in the topology, and diagnostic areas require exclusive access to the disk.

If a sector in the user area of the disk is to be slipped, **idcfmt** issues a slip sector command to the IDC. This command is an atomic (indivisible) operation implemented in the IDC software in a fashion that guarantees data integrity at all times, even in the event of a power failure during the operation. If the slip operation is interrupted, the operation automatically completes the next time the IDC software accesses the disk.

If the sector to be slipped is in the topology or diagnostic areas of the disk, the following steps are taken:

o) The topology data is read from the topology area, and the sector to be slipped is added to the memory copy of the grown defect list.

o) A flag is written to the topology area that keeps the idc software from allowing normal access to user data.

o) If the sector to be slipped is not in the topology area, then the cylinder that contains the sector is reformatted, causing the specified sector to be reallocated from the pool of spare sectors at the end of the cylinder.

o) If the sector to be slipped is in the topology area, then the diagnostic area is reformatted in order to ensure that the topology area data can be written to the disk.

o) If the sector to be slipped is in the topology area, a checkpoint containing the topology data is written to the diagnostic area.

o) If the sector to be slipped is in the topology area, the topology area is reformatted and the data is copied from the diagnostic area back to the topology area. The diagnostic area is then reformatted to remove the checkpoint data.

o) If the sector to be slipped is not in the topology area, the defect lists in the topology area are updated to show the new defect data.

o) The flag which keeps the IDC microcode from allowing access to user data is removed.

If a slip of a sector in the topology or diagnostic areas is interrupted, the user data on the disk is not accessible until the slip is completed. The completing of the slip is accomplished by re–issuing the slip command with the same parameters as before. If the parameters differ after an interruption, **idcfmt** prints out a message stating what sector was being slipped before the interruption.

unslip cylinder track sector

This command is the inverse of the slip command described above. It is provided in the event that user mistyped the cylinder track and sector number during a slip command, or to allow the user to slip and then unslip a sector in an attempt to fix a header CRC or data ECC error. As with the slip command, the unslip command does not require exclusive access to the disk if the sector to be unslipped is in the user data area. If the sector to be unslipped is in the topology or diagnostic areas, exclusive access to the disk is required. The algorithm used by the unslip command is identical to the one used for the slip command, except that the specified sector is removed from the defect list instead of being added to it.

WARNING!!!!

UNSLIPING A SECTOR FOR ANY REASON OTHER THAN THOSE DESCRIBED ABOVE MAY RESULT IN CATASTROPHIC AND UNRECOVERABLE LOSS OF DATA ON THE ENTIRE DISK.

EXAMPLES

idcfmt -d 2 0 1 format

THIS IS AN EXAMPLE OF HOW TO FORMAT 99% OF ALL DRIVES. Formats drive 1, on port 0, of ccu 2 if the drive has never been formatted. If a partial format is found, the format resumes at the appropriate point, with the manufacturers and grown defect lists that were in effect when the format was interrupted. If the drive has no valid format on it, then the program reads the manufacturers defect data from the drive.

The examples that follow are for drives with unreadable manufacturers defect lists or other peculiarities. If a drive develops additional defects after it has been formatted, they may be fixed with the slip command described above.

idcfmt -d 2 0 1 -m
Formats the same as the first example, but the user specified defect list in the file "foo" is used instead of reading the manufacturers supplied list on the disk drive.

idcfmt -d 2 0 1 FORMAT
Forces a format operation to occur regardless of the current status of the drive. The manufacturers defect cylinder is read and no grown defect data is assumed.

idcfmt -d 2 0 1 -m
Formats the same as the previous example, but the user specified defect list in the file "foo" is used instead of reading the manufacturers supplied list on the disk drive.

idcfmt -d 2 0 1 verify
Causes a verification of the format of the disk. The steps taken to verify the disk are described previously.

idcfmt -d 2 0 1 slip
Causes sector 6 of track 5 on cylinder 10 to be re-allocated from the pool of spare sectors at the end of the disk.

idcfmt -d 2 0 1 unslip
This command undoes the effects of the command described in the previous example. PLEASE read the description of the slip and unslip command before using the unslip command.

LAYOUT OF AN IDC DISK

An IDC disk has five distinct areas. They are:

1) The topology area
2) The user data area
3) The diagnostic area
4) A spare area
5) The manufacturers supplied defect data

Each area is immediatly adjacent to each other and they occur in the order listed above. Each area must begin and end on a cylinder boundary. The size of each area is determined by **idcfmt** during the initial format of the disk from the data contained in the database. Refer to the the section on FILES within this manpge for more information.

The topology area contains various types of information about the disk. Each section in the topology area is recorded in triplicate. The topology area is recorded in the default geometry specified by the manufacturer. The other areas are recorded in a format specified by CONVEX.

The various sections in the topology area are:

a) A map of the reset of the topology area (always located at logical block 0)

b) A copy of the manufacturers defect data. This data is used to avoid reading or writing sectors with defects in them. The defect data is also used in calculating what the sector header each sector contains. The disk is unreadable if this data is not present.

c) The grown defect data. This list of defects performs the same function as the list in "b" above, except that it contains a record of defects that were not listed in the manufacturers list.

d) A copy of the CONVEX specified disk geometry used in the other areas of the disk.

e) A copy of the manufacturers default geometry.

f) The ConvexOS disk label. This disk label contains the type of the disk, e.g. DKD-502, the volume name, e.g. Volume_123, the date that the drive was formatted, the date the the ConvexOS disk label was last changed, and a list of the logical partitions defined for the drive.

g) Two scratch areas used by the IDC software to keep track of the state of a sector slip operation in the user data area. These areas are interrogated when the IDC software is first booted. If the scratch areas indicate that a slip or unslip of a sector in the user area was in progress but was interrupted, then the operation automatically resumes at the interrupted point. If the scratch areas indicate that an exclusive access diagnostic operation was in progress, the IDC software will not allow access to the user data area until the operation is finished.

The user area is where user data is read and written under ConvexOS. Its size is the entire area of the disk, less the area allocated to the topology, diagnostic, spare, and manufacturers defect area.

The diagnostic area is a scratch area on the disk that is used for disk diagnostics, and to temporarily hold user data while a sector in the user area is being slipped or unslipped.

The spare area is an area that is not currently used, but is reserved for future use by CONVEX.

The manufacturers defect area is the area on which the disk drive manufacturer records the defects for the drive. A copy of this data is kept in the topology area. This area is never written by CONVEX, thus the original defect data is always present.

ERROR CODES

Idctfmt produces errors messages in the following format:

```
Error: (0x123) message
        : Additional information
```

where "0x123" and "message" are one of the codes and messages listed below. The "Additional information" is other useful information such as the cylinder, and track, sector where an error occured, the name of the disk command that failed, the name of a file that **idcfmt** could not open, etc. The information on the "Error:" line is intended to help the user diagnose the problem. The "Additional information" lines are elaborations on the original error message. These lines also contain information to help the CONVEX development staff assist in isolating problems in both hardware and software.

Code	Message

0x1 Error in byte count A read or write operation was initiated but not all of the data requested was transferred.

This will not normally happen without one of the other errors listed below.

0x3 Various error messages

A CMI error occured. CMI is the Common Message Interface that is being introduced by CONVEX with the IDC. The user should consult ????? for a full description of the CMI interface.

This error code will be followed by additional lines that describe the type of operation in progress when an error occured, the number of times the operation was retried, and any extended status available.

0x4 Error from fwrite

Idcfmt tried to write to a file on the SPU or the CONVEX file system and an error occured. This is most likely caused by a file system being full. This error code has nothing to do with the disk under test.

0x5 Non numeric value found when expecting a number

It is most likely that a parameter was mistyped from the command line or that one of the files described in the FILES section contains an error.

0x8 spu window ioctl error

Idcfmt received an error when trying to get SPU UNIX to allocate or remap a service processor window. This may be caused by too many processes allocating too many windows on the SPU.

0x9 mbs error

An error was received from the Message Based System (MBS). This is usually indicative of a hardware malfunction on the SPU, the IDC, the PIA/PI2, or system memory. The user may wish to consult the system error log to see if it contains more information about the error. See the discussion on FILES within this document.

0xa timeout waiting for message

Idcfmt sent a message to the IDC, but the IDC failed to respond. This can occur if the IDC encounters a situation that it is not expecting, and it panics. It is possible that this message can result when running the online version and a loaded, slow system causes a timeout to occur.

0xb main memory too small. Requires 4MB + 2MB per disk drive

For operation on the SPU, **idcfmt** requires a 4MB overhead area plus 2MB per copy of **idcfmt**. If there are many copies of **idcfmt** in execution, and the memory configuration is very small, this error may occur.

0xc must set geometry before this operation

This indicates an internal software error in **idcfmt**.

0xd error on open of file

Idcfmt attempted to open a SPU UNIX or ConvexOS file, and received and error. The file name is displayed on a subsequent line. This error can occur if the user mistyped a file name, if the specified file is not present, or if the SPU or ConvexOS denied **idcfmt** access to the file.

0xe parsedb() error

Idcfmt attempted to extract information from the file */mnt/boot_db* on the SPU and the requested information was missing. The name of the missing piece of information is also listed. Be sure that *.diaginit(1D)* has successfully completed.

attempt to position outside of defect area boundry

The defect list on the selected disk drive is unreadable.

0x12 manufacturers defect list not in CONVEX compatible format

A defect list was encountered that is not in the format needed by CONVEX.

0x13 cannot malloc() memory

Idcfmt requested memory from SPU UNIX or the ConvexOS and received an error. This is most likely caused by too many processes in execution at the same time.

0x14, 0x15
Set #1 missing from defect map

Last set in map does not have proper value in flag field These errors indicate that the defect map on the drive has been partially destroyed, and is not usable.

0x16 must read defects before this operation

This is an internal software error in **idcfmt**. Contact the CONVEX development staff to report a software error.

0x17 error from fseek()

Idcfmt tried to re-position a file and received an error.

0x19 data miscompare

Idcfmt was comparing data read from the disk with known good data and it encountered a discrepancy. The address of the data along with the expected and actual data is displayed.

0x1c topology area too small

The topology area is too small to hold the required data. Make sure that the correct diagnostic database is installed. This error is most likely caused by attempting to format a drive with the wrong paramter specified in the -t option on the command line.

0x1d magic number incorrect

0x1e topology check sum incorrect

These errors indicate corruption of the topology area.

0x2e Illegal ecc interleave

An unknown ECC interleave was specified in the database file. Make sure the correct version of system diagnostics and diagnostic database are installed.

0x2f error from fread

Idcfmt tried to read a file on the SPU UNIX or ConvexOS file system and received an error.

0x30 Bad format for manufacturers defect

The user specified the -m option and one of the entries in the file is incorrect. The line number of the entry in error is also displayed.

0x31 Manufacturers defect overlap. List not usable

The manufacturers defect data contains overlapping defect entries. This is not a normal occurrence and **idcfmt** can not handle this case.

0x37 Cmd line DKD entry not unique in DB file

The user specified -t on the command line, and the disk type specified has more than one entry in the database file. Make sure that the correct diagnostic database is installed.

0x38 Cmd line DKD entry not found in DB file

The user specified -t on the command line, and the disk type specified was not found in the database. Make sure that the correct diagnostic database is installed.

0x3b Crc incorrect on checkpoint data

An media error occured when trying to read checkpoint data off the drive. If the operation in progress was a format command, the user can retry the operation by using the FORMAT variant of the command.

0x3c, 0x3d
Version number incorrect on checkpoint data Magic number incorrect on checkpoint data

Checkpoint data was encountered that was written by an incompatible version of **idcfmt**. This error is not recoverable. Either reformat the disk using FORMAT or ensure that the disk was not previously interrupted while attempting to format with another version of **idcfmt**.

0x3e Sequence number incorrect on checkpoint data

An media error occured when trying to read checkpoint data off the drive. If the operation in progress was a format command, the user can retry the operation by using the FORMAT variant of the command.

0x3f Ran out of room for checkpoint data

The diagnostic or topology are is too small to hold the checkpoint data. This is a fatal error and in most cases the drive is defective.

0x40 Unknown checkpoint state

Checkpoint data was encountered that was written by an incompatible version of **idcfmt**. This error is not recoverable. This error usually indicates that the **idcfmt** software contains errors. Please contact the CONVEX development staff to report this error.

0x43 topology version number incorrect

A topology area was encountered with a format incompatible with this version of **idcfmt**. This error can indicate that the **idcfmt** software contains an error or that the drive was previously formatted with a different version of **idcfmt**.

0x45 Non-hex digit in pattern

The user specified a alternate pattern file that contains a non-hex digit in the field. See the section on FILES within this document.

0x46 Sector header incorrect

A sector header was found with good CRC, but incorrect data. The expected and actual header are displayed. This error message usually indicates a faulty drive.

0x47 Header crc incorrect

A sector header was encountered with bad CRC. This error message usually indicates a fixable media problem.

0x48 non-media related cmi error

An error was encountered during pattern testing of the drive that does not appear to be related to media errors. The format of the message is the same as for error code 0x3.

0x49 can't read ucode fields to resume pending diagnostic operation

The IDC software reports that a diagnostic operation was in progress, but the ucode scratch areas cannot be read in order to resume the operation.

0x4a the following operation must be completed. See idcfmt.1d

An exclusive access diagnostic operation was interrupted, and the user tried to start a different operation without completing the first one.

0x4b One or more sections in the topology area are unreadable

The user attempted an operation which requires that the topology area be read, but there is at least one topology section which is totally unreadable.

0x4c verification after write of diagnostic ucode temp field failed

The user attempted an operation which requires **idcfmt** to write a value to the topology area scratch sections, but the read after write failed to verify. **Idcfmt** will attempt to clear out the scratch areas and abort the operation.

0x4d disk address out of bounds

The user attempted to specify a disk address that is outside the allowed range for the drive.

0x4f drive model number not unique in data base file

Idcfmt encountered more than one record in the database with the same model number. The user may override **idcfmt** by specifying the -t option.

0x50 drive model number not found in data base file

Idcfmt encountered a a disk model number that it could not find in the data base. Make sure that the correct diagnostic database is installed.

0x52 can't find MEMSTART in /mnt/boot_db.

Make sure .diagint was executed

0x53 can't find PCM in /mnt/boot_db.

Make sure *.diagint* was executed The file */mnt/boot_db* appears to be incorrect. Make sure that *.diagint(1d)* completed execution.

0x54 can't acquire spu unix lock

0x55 can't get open count of spu unix lock

0x56 can't set spu unix variable

The wrong version of SPU UNIX appears to be installed. **Idcfmt** requires SPU UNIX 5.2 or greater.

0x57 initall failed to complete successfully

Idcfmt was attempting to initialize the system and *initall(1D)* reported an error. The output from *initall* can be found in */tmp/idcfmt.initall.*

0x58 can't get spu unix variable

The wrong version of SPU UNIX appears to be installed. **Idcfmt** requires SPU UNIX 5.2 or greater.

0x59 unknown main memory state. Make sure idcfmt is the only thing running

There appears to be a program running on the SPU that is using main memory in a fashion that is not compatible with **idcfmt.**

0x5a can't release spu unix lock

The wrong version of SPU UNIX appears to be installed. **Idcfmt** requires SPU UNIX 5.2 or greater.

0x5b the specified ccu is not logged as being an IDC in /mnt/boot_db

The user specified a ccu number that is not listed and being an IDC in */mnt/boot_db.* Verify that then ccu number entered is correct, and then verify that *.diaginit(1d)* finished execution.

FILES

Format of manufacturers defect file

When the user specifies the -m option on the command line, **idcfmt** expects the specified file pathname to point to an ascii file with the following data.

Cylinder track byte_count_from_index defect_length_in_bits

There should be one line in the file for each defect entry. All numbers are in decimal. Due the peculiarity of the way defect data is encoded for multiple heads per track disk drives, it is

extremely difficult to guess what sector an error will occur for a given byte count from index. The -m option is provided so that the user can recover defect data from a paper copy. If the user wishes to force a sector to be treated as defective during the initial format of the drive, he should make an entry in the grown defect table.

Format of grown defect file

When the user specifies the -g option on the command line, **idcfmt** expects the argument after the -g to point to a file in the following format:

Cylinder track sector

There should be one line in the file for each defective sector. All numbers are in decimal.

Format of pattern file

The -p option on the command line is used to change the default set of patterns used during pattern testing of a disk drive. The argument after the -p flag should point to a file that contains a set of hex digits that will be used to pattern test the drive. Thus, a file containing

1 4 c 9abdef01

will cause four patterns to be used for pattern testing. The first pattern will fill the disk with all 0x11111111, the second with all 0x44444444, the third with 0xcccccccc, and finally, 0x9abcdef01. It should be noted that entering a single hex digit of 1 is the same as entering it as 11, 1111, and 11111111. It is also possible to enter a pattern of 23456789a and then the pattern generated will be

23456789 a2345678 9a234567 89a23456 789a2345 ...

When the pattern is specified with a length of 1, 2, 4 or 8 hex digits, the initialization of the data buffer is much faster. Specifying a pattern as 111 instead of 1, 11, 1111, or 11111111 results in over 16x the amount of time needed to initialize the buffer.

Output from prtlog during idcfmt

When **idcfmt** executes, it causes error conditions that cause the IDC software to generate messages that would normally be put in */mnt/errlog*. In order keep from filling the error log with these "non errors", **idcfmt** sends all errorlog output to */tmp/idcfmt.plog*.

Format of DB_idc

Below is a sample of idcfmt's database. The numbers in this file have been carefully chosen to yield optimal performance and reliability. These numbers should not be changed by the user. Changing these numbers and formatting a drive will produce a disk drive that is not usable under the ConvexOS.

```
# DB_idc - IPI DISK DRIVE PARAMETER FILE
#
#  Drive Name      Description
#  ----------      -----------
#   DKD-501     Imprimis(CDC) 97209 Sabre 1.2 GB. IPI-2
#   DKD-502     Imprimis(CDC) 97229 Sabre-5L2 1.153 GB. IPI-2 (2 head parallel)
```

```
#
#---------------------------- IDC CONTROLLER--------------------------------
#
#   a       b       c    d    e f  g h i  j k l m n o p q r  s
# ------- ------------ ---- ------ - ---- -- - -- -- - - - - -- -- -- -  --
# DKD-501 CDC_MPI_EMD5 1635  50400 P 2048 15 1 23  9 1 1 1 1 12 10 20 1  10
#
#  t     u    v           w
# ------- ---- -------- --------------------------
# DKD-501 BYTE 3000000 /mnt/usr/lib/idc_pat_rll_2-7
#
#-------------------------------------------------------------------------
#
1 DKD-501 CDC_MPI_EMD5 1635  50400 P 2048 15 1 23  9 1 1 1 1 12 10 20 1  10
2 DKD-501 BYTE 3000000 /mnt/usr/lib/idc_pat_mfm
1 DKD-502 CDC_MPI_S5L2 1635  50400 P 2048  7 2 45 15 1 1 1 1 12 14 34 2  10
2 DKD-502 BYTE 6000000 /mnt/usr/lib/idc_pat_rll_2-7
#
#-------------------------------------------------------------------------
# LEGEND:
# a - drive name  - Must be an unique DKD-NNN number.
# b - mfg id/model number
#
#    mfg id      - 4 bytes obtained by read config of drive, eg. "CDC_"
#    model number - 8 bytes obtained by read config of drive, eg. "MPI_S5L2"
#
#    Note: See product specification manual of IPI-2 drive for these info.
#        Replace any spaces in the field with underscore characters.
#
# c - Total number of cylinders per unit
# d - Number of bytes per track (see e)
#
# e - P - ( d field is the physical bytes per track )
#     L - ( d field is the logical bytes per track)
#     The logical bytes per track is the physical bytes per track multiplied
#     by the number of heads per logical track. eg. DKD-502 entry can have an
#     alternate specification of '100800' for the d field and 'L' for the
#     e field.
#
# f - Number of data bytes per sector
# g - Number of logical tracks per cylinder
# h - Number of heads per logical track
# i - Number of sectors per logical track
# j - Number of spare sectors per user cylinder
# k - Number of cylinders reserved for diagnostic use
# l - Number of cylinders reserved for manufacturers defect map
# m - Number of cylinders reserved for describing drive topology
# n - Number of cylinders reserved for spares.
#
# o - Header size
# p - Header gap
# q - Data gap
# r - Ecc interleave (options are 0, 1, 2 or 4)
```

```
# s - Number of spare sectors per topology cylinder
# t - This should have the same DKD name as line 1 of the entry.
# u - Data Interleave, applicable only to parallel heads (BIT, BYTE, WORD)
# v - Drive data transfer rate in bytes
# w - Absolute pathname of ascii hex pattern file
```

BUGS

No known bugs as of 11/89.

NAME

initall – initialize the CPU control stores and main memory

SYNOPSIS

initall [**-c**]

DESCRIPTION

The **initall** utility initializes all control stores for the CPU and initializes main memory. This includes all control stores for each CPU installed in the current configuration (see **cs(1d)** for details on control store loading). By default, **initall** forces the initialization of all CPUs and memory.

-c Implies conditional initialization of CPUs.

In this case, if the file *mnt/usr/lib/Initall_cpu* exists, the CPUs are assumed to be initialized, and only memory initialization is performed. If **/mnt/usr/lib/Initall_cpu** does not exist, all CPUs and memory are initialized.

If an error is detected during initialization, **initall** will abort the initialization sequence. The **initall** utility returns a status of 0 for successful initialization, and a status of 1 if the initialization fails.

FILES

/mnt/usr/lib/Initall_cpu

SEE ALSO

sysreset(1D)
margin(1D)
cs(1D)
mminit(1D)

NAME

ioputil – iop register and memory utility

SYNOPSIS

ioputil [IOP number]

DESCRIPTION

The **ioputil** utility displays and modifies IOP memory locations. When the modify mode is entered, the displayed value may be changed by entering the new hex value and hitting <RETURN>. If no value is entered, the next memory location will be displayed. A <q> terminates the modify and returns to the **Cmd:** prompt.

COMMANDS

Commands are of the general form:

command parameters

All address values are in hex. A <q> may be used to terminate any command.

m a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one byte at a time.

mb a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one byte at a time.

mw a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one word at a time.

ml a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one longword at a time.

f a1 [a2] value The contents of memory locations a1 through a2 are filled, on a byte basis, with value. If a2 is not specified, only location a1 is changed.

fb a1 [a2] value
The contents of memory locations a1 through a2 are filled, on a byte basis, with value. If a2 is not specified, only location a1 is changed.

fw a1 [a2] value
The contents of memory locations a1 through a2 are filled, on a word basis, with value. If a2 is not specified, only location a1 is changed.

fl a1 [a2] value The contents of memory locations a1 through a2 are filled, on a longword basis, with value. If a2 is not specified, only location a1 is changed.

M a1 disp Causes the memory modify mode to be entered. Memory is displayed starting at a1 with the option of modifying each byte displayed. The byte increment between displayed memory locations is "disp."

cp a1 a2 a3 Copies a block of memory starting at a1 through a2 and places it at a3.

rm regname Allows examination or modification of the contents of memory-mapped IOP-based registers. The contents of the register will be displayed. Entering a <RETURN> causes the next register to be displayed. Entering a hex value followed by a <RETURN> changes the contents of the register, if it is a writable register. Entering a <CTRL> causes the previous register to be displayed. Entering a <q> exits this mode. The valid register names are:

ier Interrupt enable register
ierm Interrupt enable register (most significant word)
ierl Interrupt enable register (least significant word)
isr Interrupt status register
isrm Interrupt status register (most significant word)
isrl Interrupt status register (least significant word)
mcr Memory control register
pis PBUS interrupt channel
pic PBUS interrupt channel number
pblgm PBUS log register (most significant word)
pblgl PBUS log register (least significant word)
clg Cache load
earm Error address register (most significant word)
earl Error address register (least significant word)
trr Test results register
mbdr Multibus diagnostic register
misc Miscellaneous diagnostic register
pstat Parity status register
slotid Slot id register

rd regname Displays the bits in the named register. Each bit position is labeled with the function of the bit. Each bit may be modified individually by moving the cursor over the bit and entering a 0 or a 1.

The cursor moves to the right (toward the LSB) when a new bit value is entered or when the <SPACE BAR> is depressed. The cursor moves to the left when a <DEL> or <BACK SPACE> key is pressed. The input is terminated when a <RETURN>, <q>, <CTRL>, or <=> is entered. A <RETURN> updates the register and then displays the next register. A <q> exits this mode. Entering a <CTRL> updates the register with the modified value then displays the previous register. Entering a <=> updates the register with the modified value then redisplays the same register.

q[uit] Exits **ioputil**.

? Prints out a list of valid commands.

NAME

iscn – C200 Series interactive scan facility

SYNOPSIS

iscn [**include_file**]

DESCRIPTION

The **iscn** utility is an interactive software tool that allows a user to control and observe the internal states of individual boards of the C130, C210, and C220 central processor and I/O processors. A file of valid **iscn** commands may be specified (*include_file*) on the command line. This file will be included prior to issuing the first **iscn** prompt.

SEE ALSO

For more information, refer the "Interactive Scan (Iscan)" chapter in the *CONVEX Diagnostic Utilities Manual.*

NAME

man – find manual information by keywords; print out the manual

SYNOPSIS

man –k *keyword ...*
man –f *filename ...*
man [–] [**–t**] [*section*] *title ...*

DESCRIPTION

The **man** program gives information from the diagnostic manuals (*CONVEX SPU UNIX Utilities Manual* or *CONVEX Diagnostic Utilities Manual*). It can list one-line descriptions of commands specified by name, or all commands whose descriptions contain the specified *keyword*. It can also provide online access to sections of the printed manual.

When given the **–k** option and one or more *keywords*, **man** prints out a one-line synopsis of each manual section whose table of contents listing contains that *keyword*.

When given the **–f** option and a list of *filenames*, **man** prints out the table of contents lines for related manual sections.

When neither **–k** nor **–f** is specified, **man** outputs the specified manual pages. If a *section* specifier is given, **man** looks in that section of the manual for the given *titles*. *Section* is an Arabic section number (3, for instance). The number may be followed by a single-letter classifier. For example, 1D for instance indicates a diagnostics program in section 1. If *section* is omitted, **man** searches all sections of the manual and prints the first section it finds, if any.

If the standard output is not a terminal, or if the – flag is given, **man** pipes its output through **cat(1)** with the option **–s** to crush out adjacent blank lines; otherwise **man** will pipe its output through **more(1)** to stop after each page on the screen. Hit a space to display the next screen of information.

FILES

*/mnt/man/cat?**

SEE ALSO

more(1)
cat(1)

BUGS

The current version of **man** does not properly underline.

NAME

map – display logical-to-physical mapping

SYNOPSIS

map [**-c nn**] [**-t nn**] **a1 a2 ... an**

DESCRIPTION

The **map** utility displays the Communication Index Register (CIR) for which the mapping is taking place, the Thread ID (TID) of the process, the logical address being mapped, the Segment Descriptor Register (SDR), the contents of the first- and second-level Page Table Entries (PTE1, PTE2), the Thread-Level Page Table Entry (PTET) for PTE2s that indicate thread-level PTEs exist, and the physical address corresponding to logical addresses **a1 a2 ... an**.

The CIR option **-c nn** is optional and allows the user to select the CIR for which the logical-to-physical address translation should occur. The default is to perform the translation for CIR 0. The thread selection option **-t nn** allows the user to specify the process thread to follow in addresses that use unshared data. The default is to use thread 0. If the valid bit is not set for the SDR or a PTE, then question marks are printed for the remaining translation values.

SUBSYSTEMS AFFECTED

The **map** utility stops the clocks to the entire system in order to read the SDRs for the addresses specified. It then enables the clocks to the memory and I/O subsystems to perform main memory read operations via the EBUS.

NAME

margin – set power supply and system clock margins

SYNOPSIS

margin [[**-{qv}**] [**{-{nul}} {ps ps{p5 m2 m45} all clk}**] [**-{xe} clk**]]
[**-d** [**-{nul}**] **clkccu {# all}**]]

DESCRIPTION

The **margin** utility provides a means for the system clock and power supply margins to be read and set. Available margins are: nominal (**-n**), upper (**-u**), lower (**-l**), one-half nominal (**-x**), and external (**-e**). They have the following effect on the power supplies and system clock:

Switch	Resultant Voltage
-n	Nominal voltage
-l	Lower margin, typically 95% of nominal
-u	Upper margin, typically 105% of nominal

Switch	Resultant Clock Rate
-n	Nominal frequency
-l	Lower margin frequency, 90% of nominal
-u	Upper margin frequency, 110% of nominal
-x	One-half nominal frequency, 50% of nominal
-e	External connector frequency

The power supplies and system clock may be margined individually, or in any combination. The following are accepted mnemonics:

Mnemonic	Meaning
psp5	+5 volt power supply
psm2	-2 volt power supplies
psm45	-4.5 volt power supplies
ps	All marginable power supplies present
clk	System clock
all	All of the above

In addition to the above mnemonics, the following mnemonics are given in the display of voltage levels using the **-v** option:

Mnemonic	Meaning
vr1	voltage reference 1
vr2	voltage reference 2
smb	system monitor board

After setting the specified margin conditions, **margin** measures the resultant clock frequency and power supply voltages. Although not all voltages can be margined, all voltages are monitored. If any clock or voltage exceeds its tolerance, **margin** displays a message indicating the clock or voltage in error, the current reading, and the acceptable tolerance limits.

If the **-q** option is specified, only those voltages or clocks out of tolerance are displayed. Normally, all voltages and clocks are displayed. If **margin** is invoked with no arguments, the current margin conditions are displayed. If **-v** is the only argument, this display of conditions will loop until <**CTRL C**> is pressed.

The **-d** option enables destructive features of **margin** and may be used to margin VIOP clocks. The clocks for the microprocessor section of a VIOP are separate from the rest of the system. The mnemonic **clkccu** allows these clocks to be margined. This margining, however, requires the microprocessor section of the VIOP to be reset, and is, therefore, a destructive operation. The **clkccu** mnemonic is followed a slot number or the word "all" to specify which Channel Control Units (CCUs) are to be affected by the command.

EXAMPLES

margin -l clk -n psp5 -u psm2 psm45
margin -d -l clkccu all

RETURNS

The **margin** utility returns a zero if all power supplies and clocks are within tolerance, and a nonzero if anything is out of tolerance.

NAME

memld – load object file into system memory

SYNOPSIS

memld file [**-x**][**-o offset**][**-a physical_address**]

DESCRIPTION

The **memld** utility loads an *a.out* or SOFF format object file into system memory. First-, second-, and thread-level Page Table Entries (PTEs) are set up for the file. The Segment Descriptor Registers (SDRs) on the CPU Utility Board (CPX) are initialized for CIR 0 to address the first-level PTEs. The default load address is the first extant physical address in the system. This can be changed with the use of the **-a** flag. With this option, the starting physical memory address may be specified. If the file is a relocatable format, the logical offset may be changed by using the **-o** flag. This flag allows an offset to be specified that will be added to the logical addresses specified in the file.

If the **-x** option is specified, then the page map file is **not cleared.** This allows **memld** to be used repetitively to load files into memory without destroying the existing page tables.

HARDWARE REQUIREMENTS

The memory system must be initialized before running this utility. In order to run, the system must contain at least an SP2, sufficient memory, a Peripheral Interface Adapter (PIA), and a CPX.

HARDWARE AFFECTED

The **memld** utility turns on clocks to the memory and I/O subsystems. These clocks are left on upon exit. No other clocks are modified. No subsystems are reset. The CPX is scanned to load the values into the SDRs.

SEE ALSO

mm_map(3d)
mem_load(3d)

MKDIAG_DB

mkdiag_db, enable_cpu, disable_cpu – maintain diagnostics configuration file

SYNOPSIS

mkdiag_db [-p][-a][-i][-210a][-201a][-e KEYWORD NUM TYPE VALUE]

enable_cpu [a, b, etc.]

disable_cpu [a, b, etc.]

DESCRIPTION

The utility **mkdiag_db** initializes the diagnostics configuration file, */mnt/diag_db*. This utility examines the system hardware and software configuration to determine the appropriate settings of the various parameters to be stored in the configuration file. The utilities **enable_cpu** and **disable_cpu** provide a means to modify the default configuration for installed CPUs. An installed CPU may be enabled or disabled by specifying the CPU as **a**, **b**, etc.

NOTE

When any change is made to the diagnostics configuration file, */mnt/diag_db*, by running **mkdiag_db**, **enable_cpu**, or **disable_cpu**, the changes must be reflected in the system boot configuration file, */mnt/boot_db*. This will be done automatically by running **.diag-init** or **boot_cpu**, or it can be done manually by running **scn_util -b > /mnt/boot_db**.

The following system parameters are determined:

PROCNUM CPU population map (which CPUs are available)
OS_MODE Set memory mode: 0=normal, 1=V6.2 mode
INTRINSICS Does machine-support microcoded intrinsic instructions
PARALLEL Does machine-support parallel CPU processing
UNDER_MASK Does machine-support the vector under mask instructions
SCALAR_ACCELERATOR Specify scalar functional unit: 0=standard
US_UCODE Specifies microcode filename for **us** control store
SR_UCODE Specifies filename for scratch RAM initialization
UA_UCODE Specifies microcode filename for **ua** control store
UL_UCODE Specifies microcode filename for **ul** control store
UM_UCODE Specifies microcode filename for **um** control store
VD_UCODE Specifies microcode filename for **vd** control store
US_UCODE Specifies microcode filename for **us** control store

If **mkdiag_db** is invoked with no options, a help message is printed. The following options are available:

–p Prints the current configuration without making any changes to the configuration.

–a Runs the automatic mode. Based on the hardware and software installed, the system configuration is determined; it is assumed that the user wants to take advantage of any available upgrades.

–i Runs interactive mode. Prompts the user for all available options in configuring the system. This option is useful to run a machine in a nonstandard configuration.

–210a or –201a

Forces the configuration to run as a CONVEX model C210a or C201a. It may be desirable to force the machine configuration to be a model C210a or C201a even if the machine is capable of running at a higher revision level.

–e Edits the database entry specified by KEYWORD. The data element number specified by NUM is replaced with the value specified by the argument VALUE. The VALUE argument may be numeric or alphanumeric. The argument TYPE specifies whether the value is a decimal number (**d**), hex number (**x**), or an ASCII string (**s**).

FILES

/mnt/diag_db

SEE ALSO

diaginit(1D)
scn_util(1D)

NAME

mm – display or modify main memory

SYNOPSIS

mm [**-s**][**sdr0**][**sdr1**][**sdr2**][**sdr3**][**sdr4**][**sdr5**][**sdr6**][**sdr7**]

DESCRIPTION

The **mm** utility displays and modifies main memory on C200 Series computers and the Population Configuration Map (PCM) on the Service Processor.

The **-s** option keeps the Service Processor from altering the clocks in the system. If this option is not present, then **mm** turns on all clocks in the memory and I/O subsystems, and turns off the clocks in all other subsystems. If Segment Descriptor Registers (SDRs) are specified on the command line, these SDRs are used for logical-to-physical address translations. If no SDRs are specified, then all SDRs are read from the utility board; this involves stopping clocks in the entire system while the SDRs are read. Specification of **any** SDR on the command line keeps **mm** from reading SDRs from the utility board. Note that **mm** initializes no subsystems.

When an **mm(CIR#,TID#):** prompt is displayed, this utility is ready for command input. The CIR is the Communication Index Register that locates SDRs used during logical-to-physical address translation. The TID is the current Thread ID number used for logical-to-physical address translation.

COMMANDS

Commands are of the general form:

command parameters

Unless noted otherwise, all numeric values are in hex. All address values represent either logical or physical addresses as specified by the **s** command. Use **<CNTRL C>** to abort an operation and return to the **mm** prompt.

b Displays the current memory usage. An example output of the command is:

```
File#  Physical Address   Pid   File Name     Logical Offset
  1    00000000-00014fff  120   p0r0_4231     00000000
  2    00015000-0006afff  120   cpu4231.rnn   00020000
  3    0006b000-0006dfff  120   support_4231  9ffff000
  4    0006e000-00077fff  120   p0rN_4231     20000000
  5    00078000-00081fff  120   p0rN_4231     40000000
  6    00082000-0008bfff  120   p0rN_4231     60000000
 ---   00ff9000-00ffefff  120   pte2          NA
 ---   00fff400-00ffffff  120   pte1          NA
```

The display lists a file number used by the **ln** command.

The physical address range shown is the locations occupied by the specified file in main memory.

The Pid field is a arbitrary number assigned by the test program during loading, all files with the same Pids share a common set of logical address to physical address mappings, i.e., they have the same SDRs.

The logical offset is the relocation offset added to the addresses in the file at the time it was loaded.

In order to use the symbolic features of **mm**, it is necessary to load the symbol table at the correct offset (refer to the **l** and **ln** commands). The reserved file names *pte1*, *pte2*, and *ptet* are the locations where the page tables are in main memory.

This information is kept in the file */mnt/test/CPU/page_map* that is used by most CPU tests. If this file does not exist, the **b** command will display this message: "/mnt/test/CPU/page_map does not exist."

d a1 [a2] Displays the contents of locations *a1* through *a2* in hex and ASCII.

d a1,count Displays the contents of locations *a1* through *a1+count* in hex and ASCII.

di a1 [a2] Displays the contents of I/O locations *a1* through *a2* in hex and ASCII. Note that only the most significant 2 bytes of a longword are meaningful in I/O space, so "xx" is printed for the low 6 bytes of each longword to avoid confusion.

di a1,count Displays the contents of I/O locations *a1* through *a1+count* in hex and ASCII.

dp Displays the contents of the Service Processor PCM.

e [on | off] Enables or disables Error Detection and Correction (EDC) for the duration of **mm**.

f a1 a2 patt The hexadecimal pattern *patt* is repetitively copied into locations *a1* through *a2*.

h Displays command summary.

i a1 a2 The memory starting at location *a1* through location *a2* is disassembled and displayed in mnemonic form.

i a1,count The memory starting at location *a1* is disassembled and displayed in mnemonic form for *count* instructions.

m [a1] The memory modification mode is entered at address *a1*. If no address is specified, the default address is zero. Memory modification is performed on a byte basis. The display has the following format:

addr: *old_data* [*new_data*]**opc**

The data at a particular location is changed by entering new data and terminating the data value with an operation code. If *new_data* is omitted, the *old_data* is unchanged. The operation codes consist of the following:

<**cr**> Increment address
- Decrement address
= Display same address
g Go to address specified by *new_data*; do not modify *old_data*
q Return to **mm** prompt

mi a1 The I/O modification mode is entered at I/O address *a1*. Modification to I/O space is performed on a halfword (16-bit) basis, and only the most significant halfword of each longword is meaningful. Therefore, only longword-aligned addresses can be modified. The commands in this mode are the same as those available to the **m** command above.

p [a1] The PCM modification mode is entered at block *a1*, where *a1* is hex. If no block address is specified, the default address is zero. The PCM modification mode uses the same operation codes as the memory modification mode.

q Exit **mm**.

l filename [-Ooffset] [id#]
Loads the symbols from the filename and adds the specified offset to each of the symbols. An identification number is optional, and is used only when the same file is going to be loaded multiple times. This identification number is a means to distinguish between two different symbol tables created from the same file.

ln filenumber Loads the symbols from the file specified by filenumber. This filenumber is the number of the file listed when the **b** command is used. The symbol table offset is obtained from the *page_map* file. The default id# associated with the symbol table is 1.

s [log][phys] Sets the addressing mode of **mm**. Logical mode assumes that logical addresses are used, and performs logical-to-physical address translations on all addresses. The SDRs used are either those specified on the command line, or those that exist for the specified **cir**.

ps Pushes (rotates) the symbol tables located on the symbol table stack. When a symbol is used, it is located by searching the tables in the order specified on the symbol table stack, where the first occurrence of a symbol is used. This command can be used to ensure that the correct instance of a symbol is located.

c num Changes the **cir** value. When the **cir** value is changed, the new SDRs are always obtained from hardware. The new SDRs will then be used for logical-to-physical translation.

t num Allows the setting of the thread id, which will then be used during all subsequent logical-to-physical address translations. Thread id is only significant in translating address located in memory that has been mapped as being threaded.

! A shell is invoked to interpret and execute the remainder of the line following the **!**. Following the shell command, system clocks are restored to the state that **mm** sets them to at startup unless **mm** was invoked with the **-s** option.

BUGS

After power up, the memory system must be reset (refer to **sysreset(1d)**) before **mm** can access main memory. Additionally, the PCM and main memory must be initialized by **mminit(1d)** before other utilities can access main memory.

HARDWARE REQUIREMENTS

Initialize the memory system before running this utility. The system must contain at least a Service Processor, two memory boards, and a peripheral interface adapater to run. If SDRs are to be read from the system, then a utility board must also be installed.

HARDWARE AFFECTED

Upon exit, **mm** will leave the memory and I/O system clocks running if the **-s** option is not specified. If the **-s** option is specified, all clocks will be as they were upon entry. The **mm** utility performs no scan operations **unless** it is necessary to read the SDRs from a utility board or the enable or disable EDC command is executed.

NAME

mminit – main memory initialization

SYNOPSIS

mminit [**-c n**] [**-m**] [**-p n**] [**-s**] [**-i 8/16/32**] [**-f**] [**-P**]

DESCRIPTION

mminit is used to initialize main memory after system power up. Normally, **mminit** sets up the interleave and mode information, sizes main memory, initializes all the Population Configuration Maps (PCMs), generates the file */mnt/usr/lib/DB_mcm*, clears all the locations in main memory, and displays a table of the memory blocks found to be present. The default initialization mode utilizes the vector processing capability of the central processor. If an error is detected during initialization, **mminit** returns a -1. Otherwise, **mminit** returns a 0.

If both */mnt/usr/lib/DB_mcm* and */mnt/boot_db* exist and **mminit** is able to correctly read the PCM entry from */mnt/boot_db*, then **mminit** uses the PCM obtained from the file. If either of the files do not exist, **mminit** sizes memory. Memory is sized by initializing portions of memory from the top of memory to the beginning of memory then reads and checks the initialized areas to determine the amount of memory in each bank of each memory slot. From this information **mminit** generates the file */mnt/usr/lib/DB_mcm*. **mminit** then initializes all system PCMs, including PCMs located on the Service Processor, utility and peripheral interface adapter boards.

mminit also initializes the configuration registers for the current interleave factor in use. This includes interleave registers located on the Service Processor, peripheral interface adapter, and the Data Cache Unit (DCU) of each CPU. **mminit** determines the maximum allowable interleave by comparing the relevant PCM entries for the slots. The general rule of thumb is for all the board pairs to have the same memory sizes in order to maximize the interleave factor. The default interleave factor is 8-way interleave. The following checks are made:

- o If slot pairs one and two or slot pairs three and four are present and have identical PCM entries, then 16-way interleave is allowed.

- o If slot pairs one through four are present, and all four have identical PCM entries, then 32-way interleave is allowed. However, if slot pairs one through four are present and either slot pairs one and two or three and four do not have identical PCM entries, then the interleave is forced back to 8-way interleave.

- o If slot pairs one through three are present and slots one and two have identical PCM entries, then 16- plus 8-way interleave is allowed.

The following options are supported:

-c Initialize main memory using CPU **n**. The default CPU is the first CPU found in the current configuration.

-m Initialize main memory only. It is assumed that main memory has already been sized and the PCM initialized. Refer to the note on the **-f** option.

-p n Set the memory initialization pattern to the longword hex pattern specified by **n**. The default longword pattern is zero.

-s Use the Service Processor Unit (SP2) to perform the initialization via the EBUS. No vector operations are performed, and the CPU is not involved in the initialization. This mode of initialization will be significantly slower than the default vectorized initialization mode.

-i 8/16/32

Force the interleave factor. This mode allows the user to bypass the normal method of initializing the interleave register to the maximum interleave. No error checking is performed

to verify that a valid interleave is being selected.

-P Perform PCM initialization only. **mminit** determines the PCM (either from the */mnt/usr/lib/DB_mcm* file or via memory poking) for all system PCMs and returns.

-f Force the PCM initialization via memory poking. This option overrides the ability to read the PCM from the */mnt/boot_db* file. It should be noted that this option should not be used in conjunction with the **-m** option. The **-m** option takes precedence over this option.

DIAGNOSTICS

mminit has the following diagnostic messages:

"**scn_init failed**" - **mminit** was unable to initialize the scan machine in order to perform its function.

"**Error trying to remove /mnt/test/CPU/page_map**" - **mminit** received an error when it tried to remove the page map file.

"**mminit: File /mnt/diag_db does not exist This file is essential to diagnostics!!! mminit assuming 6.2 OS mode**" - The file /mnt/diag_db contains information used by several utilities and tests. This file should be present in every system.

"**mminit: Unable to find OS mode in /mnt/diag_db Assuming 6.2 OS mode**" - The OS mode variable determines whether to use 6.2 mode for memory or 7.0 mode. **mminit** was unable to determine which to use and defaulted to 6.2 OS mode.

"**mminit: undefined machine class XX Using class YY as default**" - The machine class from the backplane was an unknown type. **mminit** defaults to a C130 configuration in this case.

"**No CPU available. Forcing -s option**" - **mminit** could not find a CPU to execute the initialization and is forcing an initialization from the SPU.

"**mminit: No memory boards found**" - **mminit** didn't find any memory board pairs.

"**mminit: slot: XX and slot YYx are not populated the same(mcm)**" - **mminit** did not find an MCM in both the odd and even slots.

"**mminit: continuing using internal PCM**" - **mminit** had an error initializing a PCM and is using what it considers the correct PCM.

"**Slot XX and slot YY have different memory sizes Unable to interleave across them**" - **mminit** was not able to upgrade the interleave due to memory incompatibilities.

"**Can not upgrade to 16 way interleave due to mismatch on pairs XX and YY**" - **mminit** was unable to go to 16-way interleave because one of the four slot pairs is incompatible.

The following error messages relate to the loading and execution of CPU code:

"**Error during memory load**" - **mminit** was unable to load the CPU code into main memory for execution.

"**Error during read of memory address map**" - **mminit** received an error when it

tried to read the addresses of the relevant data areas in main memory.

"**Invalid mminit.x00 request code**" - The CPU expects one of the two types of request codes. For this case, the CPU received a type from the SP2 that it did not expect.

"**Hard error occurred**" - The SP2 code detected a hard error during the CPU codes execution.

"**CPU time out**" - The CPU did not complete execution in the time allowed by the SP2 code.

"**Vector valid trap**" - A vector valid trap was detected by the CPU code.

"**Unknown CPU error type: XXXX returned in PDT**" - The CPU returned an invalid error type to the SP2 when it completed.

"**Error exit**" - The CPU took an error exit due to the execution of an all zero opcode.

"**Undefined opcode**" - The CPU detected an undefined opcode during execution.

"**Invalid process exception code XXXX**" - The CPU took a process exception.

SUBSYSTEMS AFFECTED

mminit affects the execution of code running on any of the subsystems since it initializes main memory that is used by all the subsystems.

BUGS

The force interleave option does not allow the 16+8 option to be used. This can be accomplished by using **sp2util** to modify the PCR register of the SPU and then performing a sysreset to force the interleave register to migrate to the PIA and DCU.

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** can attempt to eliminate the error by performing a memory scrub operation on the 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 Central Processing Unit (CPU).

The memory sniffer always reads main memory in four-page groups (e.g., 16 Kbytes, 1/64 the size of one Population Configuration Map (PCM) entry). 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 sec, the value is forced to 15 sec. This time and the time required to sniff the entire memory system are reported.

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

pte_cache – dump the PTE cache

SYNOPSIS

pte_cache [**-c #/all**] [**-m/r**] **-dh** [**n1** [**n2**]]

DESCRIPTION

The **pte_cache** utility allows the contents of the PTE cache to be dumped to *stdout* as either RAM addresses or as logical addresses.

The default operation of **pte_cache** is to dump the entire cache of all the CPUs in the current configuration in logical address mode.

The **pte_cache** utility allows the user to specify, via the **-c** option, the CPUs on which to perform the **pte_cache** operations. The default is to perform the user-specified operations on all CPUs in the current configuration. When a specific CPU is requested, **pte_cache** verifies that the CPU is in the current configuration before attempting any dump operations.

The specified central processors are halted (see description above for details on the **-c** command), and the contents of the PTE cache are dumped to *stdout* in hex format (an option of **-dh**). The default addressing mode for this utility is memory address mode (**-m**). In memory address mode, **pte_cache** addresses are interpreted as being logical addresses, and bits <12..21> are significant to the utility. Bits <0..11> are discarded, as are bits <22..31>.

The **pte_cache** utility also supports the display of addresses in RAM address mode (**-r**). In RAM address mode, the user selects the addresses to display as physical RAM addresses of the cache. In this mode, bits <0..9> are the significant bits of the address. In this address mode, an error is reported if an address is larger than is supported in the hardware.

Selecting **n1** and **n2** allows the user to specify the range of **pte_cache** addresses to dump. If no addresses are specified, then the entire **pte_cache** is dumped. If an error is detected, **pte_cache** returns a -1. Otherwise, **pte_cache** returns a 0.

NAME

pup – power-up bit read and write utility

SYNOPSIS

pup [**?** | [**on** | **off**]]

DESCRIPTION

The **pup** utility performs the operation of reading and writing the power-up bit on the Service Processor (SP2). **Pup** can be invoked in one of three different ways. First, **pup** can be invoked with no options. This cause the current state of the power up bit to be returned. Second, **pup** can be invoked with the **?** option. This displays the command format for invoking pup and returns a **-1** to the user. Third, **pup** can be invoked with either the option **on** (1) or **off** (0). The power up bit will be set to this value and the previous value is returned. **Pup** returns the previous value of the power-up bit if a parameter is specified, or the current value of the power-up bit (**0** or **1**). If an error occurs, **-1** is returned.

REFERENCED FUNCTIONS

ioaccess(2d)
signal(2)
cop_read(3d)
cop_write(3d)

BUGS

Due to the sensitivity of the cop functions, reading and writing the cop chip are indivisble operations to the user (ie., aborting pup during either of these operations is impossible.) What this means is that if the operation of the cop hangs, then the pup operation (along with the terminal) is hung.

NAME

rita_perr – set internal parity error handling mode RITA gate array

SYNOPSIS

rita_perr on | off | hard | soft

DESCRIPTION

The utility **rita_perr** is used to set the internal tag RAM parity error handling mode of the RITA gate array in 3220 EFU boards. Invoking **rita_perr** with the **on** or **hard** paramater will cause the CPUs to assert a parity error on the EFU and pull a hard error when a RITA RAM parity error is detected. Invoking **rita_perr** with either the **off** or **soft** parameters will cause the affected EFU to not pull a hard error or assert a parity error, but will instead cause the EFU to invalidate that address in the data cache. This command must be executed before the CPU is initialized. Once the CPU is running, this command has no affect.

IMPLEMENTATION

This command is implemented by modifying the **RITA_PE_CONTROL** entry in the */mnt/diag_db* database. Subsequent initialization routines look at the value in this database to determine how to perform initialization. The state of the **RITA_PE_CONTROL** entry governs the state of the **enb_ram_par** scan field in the EFU. If hard errors are enabled, then the value of **enb_ram_par** will be 3F; and if hard errors are disabled, then the value of **enb_ram_par** will be 00.

Running a CPU with RITA RAM parity errors not generating hard errors is desireable since any transitory RAM parity errors will only cause a data cache entry to be invalidated instead of halting the CPU and crashing ConvexOS. The worst case condition for RITA RAM would be the RAM detecting a parity error every access; this has an equivalent effect to turning off the data cache. Typically, spurious RITA RAM parity errors occur very rarely, but even one is enough to crash ConvexOS. The only danger of running with RITA RAM parity errors off is that a bad RITA might not be detected, but instead just degrade data cache performance.

FILES

/mnt/diag_db

SEE ALSO

diaginit(1D)
mkdiag_db(1D)

NAME

scn_ring – Interactive read, write, and check scan ring utility

SYNOPSIS

scn_ring

DESCRIPTION

The **scn_ring** utility allows the user to interactively read, write, or check a scan ring. Provides the user with available and default options at each prompt; also checks all input for errors.

The user is first requested to enter a ring to work with. Then **scn_ring** repeatedly prompts the user for an option, including terminating execution or the specification of a different ring.

The following options are available after ring specification:

print ring information - Prints information about scan rings in general and the specified ring in particular

read ring - Prompts for the direction and number of bits to read; then prints read data

write ring - Prompts for the direction and number of bits to read, background value, and foreground bit to set (if any); then prints data written

spu4000 class 2 subtest - Tests ring with applicable spu4000 class 2 subtest

spu4000 class 4 subtest - Tests ring with applicable spu4000 class 4 subtest

select a different ring - Goes back to the ring specification prompt

exit - Exits (quits) **scn_ring**

NAME

scn_util – hardware initialization utility

SYNOPSIS

scn_util [-i] [-I] [-b] [-j **address mode**]
[-s **sdr0 sdr1 sdr2 sdr3 sdr4 sdr5 sdr6 sdr7**]

DESCRIPTION

The **scn_util** is used by the CONVEX operating system and some I/O diagnostics to initialize hardware and check status. There are several options available, most of which should be used one at a time since some are mutually exclusive and others make little sense if used in combination.

There are several options:

–i Initializes the clocks for memory and the I/O system. After this option is used, memory can be read or written from the SPU, and CCUs have their clocks enabled. The **–i** option returns an exit code of 0.

–I Returns the state of the clocks in memory and the I/O system. If these clocks are enabled, a 0 status is returned; if the clocks are not enabled, a non-0 status is returned.

–j address mode
Enables the CPU to start execution at the desired address. If the mode is 0, execution begins immediately. If the mode is non-0, a set of prompts appears so that certain modes can be modified prior to starting the CPU. The only mode selection possible at this time is the state of the Dcache. An exit code of 0 is returned if the start is successful; a non-0 is returned if the start is unsuccessful.

–s sdr0 sdr1 ... sdr7
Allows setting the Segment Descriptor Registers (SDRs) to the specified values. There must be eight SDRs specified, otherwise an error will be returned. An exit code of 0 is returned if the operation is successful; a non-0 is returned if the operation is unsuccessful.

–b This option outputs the system configuration database to *stdout*. This ASCII file has information regarding the current system configuration. The format of *stdout* consists of multiple **IDENTIFIER**s that are terminated by a semicolon. Each **IDENTIFIER** entry is as shown below:

IDENTIFIER(entry count,entry size,format) entry0,entry1,...entryN-1;

The **IDENTIFIER** is an ASCII mnemonic describing some aspect of the system. The currently supported **IDENTIFIER**s are:

CPUTYPE	CPU instruction architecture: 1=C1, 2=C210a, 3=C2xx
MEMSTART	Starting physical address of memory
IOPS	Location of IOPs in CCU slots
VIOPS	Location of VIOPs in CCU slots
IOP_ACCEL	Location of IOPs with accelerate enable option
HSPS	Location of HSPs in CCU slots
IEEE	Does machine support IEEE mode arithmetic
PCM	Physical Configuration Map (each bit = 2-Mbyte mem block)
SERIALNUM	Serial number
MACHCLASS	Machine class: 0=C1, 1=C1XE, 2=C210,C220, 4=C230,C240, 5=C201/C202
PROCNUM	CPU population map (which CPUs are available)

OS_MODE Set memory mode: 0=normal, 1=V6.2 mode
INTRINSICS Does machine support microcoded intrinsic instructions
PARALLEL Does machine support parallel CPU processing
UNDER_MASK Does machine support the vector under mask instructions
SCALAR_ACCELERATOR
Specify scalar functional unit: 0=standard
US_UCODE Specifies microcode filename for **us** control store
SR_UCODE Specifies filename for scratch RAM initialization
UA_UCODE Specifies microcode filename for **ua** control store
UL_UCODE Specifies microcode filename for **ul** control store
UM_UCODE Specifies microcode filename for **um** control store
VD_UCODE Specifies microcode filename for **vd** control store

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.

FILES

/mnt/bin/scn_util

NAME

scnlink – intermediate scan ring definition file linker

SYNOPSIS

scnlink [-c cop file] [-o output file] [-p old file] [-d rev directory] [-f serial number (hex)]

DESCRIPTION

Even though the **scnlink** utility has changed significantly due to the changes made in 5.1 SPU UNIX, backward and host compatibility remain.

In normal use on a service processor, **scnlink** examines the file */mnt/usr/scn/cop.out* to determine the revisions of boards installed in a system, and sets up a number of data structures in shared memory. Without the initialization of these data structures, any utility or test that uses scan cannot be executed; thus **scnlink** must be executed on a service processor before most utilities or tests will work.

The following options are available:

–c *file* Use the file *file* instead of */mnt/usr/scn/cop.out*.

–o *file* Produce a specified output file in the old **scnlink** format. This option is intended for debugging or host use, as special code is required for tests and utilities to be able to make use of this file. Shared memory is not modified when this option is specified.

–p *file* Usable only in conjunction with the **–o** option; if the output file specified with the **–o** option exists it will be moved to *file* before the new file is created. Any existing file *file* will be removed.

–d *directory*
Use the directory *directory* instead of */mnt/usr/scn* as the directory to find the scan ring revision files.

–f *number*
Force the system serial number to be *number*, which must be entered in hex, without a leading "0x."

The approximate order of execution is:

1. Insure correct version of SPU UNIX
2. Parse arguments and verify combinations
3. Set up the scan ring structures
4. Do any machine specific modification of these structures
5. Parse the *cop.out* file to see what is installed
6. Read the scan ring revision files for the installed boards
7. Get information about shared memory buffer
   a. if needed
8. Relocate and combine ring revision files
9. Write the resident buffer
   a. if enough space is available and
   b. it is to be written
10. Write the output file
    a. if it is to be written

The **scnlink** utility returns a 0 upon successful completion, 1 otherwise.

SEE ALSO

scn(3d)
cop(1d)

FILES

/mnt/usr/scn/cop.out
/mnt/usr/scn/[ringname]_rev[number]

NAME

security_clear – memory and cache purge

SYNOPSIS

security_clear [-i] [-no_hia] [-e nnn] [-E nnn] [-f pattfile] [cpu] [ccus] [spu] [all]

DESCRIPTION

The utility **security_clear** writes and verifies patterns to Central Processor Unit (CPU) caches, CPU main memory, Channel Control Unit (CCU) caches, CCU memory, High-Speed Parallel (HSP) buffers, HSP Interface Adapter (HIA) buffers, and Service Processor Unit (SPU) memory. It outputs the processor and memory range being purged as well as outputting when the write starts and ends and when the verify starts and ends. Any miscompares are displayed. The processors include the CPU, SPU, Input/Output Processor (IOP), VMEbus Input/Output Processor (VIOP), and HSP with associated caches.

The following options are supported:

-i When specified, **security_clear** stops after each verify to allow inspection of the memory. At inspection time, specify a range of addresses to dump, specify a single address that displays one line of data, enter **q** to quit inspection of the just-cleared area, or enter **cancel** to cancel any further prompts for inspection. If only one address is entered, then one line is displayed which permits choosing return to dump another line, typing a new address where the next displayed line starts, or entering a **q** to return to the **Inspect:** prompt.
Default: All memories purged with no interactive inspection.

-no_hia
This option prevents the CCU purge logic from attempting to purge HIA buffers. This may be necessary if something other than an HIA is attached to an HSP.
Default: A test is made to confirm an HIA exists and then the HIA buffers are purged.

-e nnn
If this option is specified, then if more than nnn errors occur during a single pattern, the pattern is skipped and the next pattern is started.
Default: 2,147,483,647

-E nnn
Use this option to terminate **security_clear** due to an excessive number of errors occurring during a single pattern. Termination occurs when the number of errors exceeds nnn.
Default: 2,147,483,647

-f pattfile
Defines the name of a file that contains separate patterns to be used with each processor.
Default: Searches for **purge_patterns** in the current directory first and then in */mnt/bin/lib*.

[cpu] [ccus] [spu] [all]
Specifies which processors will be purged. Implicitly specify **all** by not specifying any of the above processors.
Default: **all**

The above parameters can be specified in any order.

It is possible to specify a different pattern file than *purge_patterns* by using the **-f** option in the command parameters.

The pattern file specifies patterns for each processor. First specify a processor (**ccu**, **cpu**, or **spu**), followed by patterns for that processor. Each pattern must be 8 bytes long (16 hexadecimal digits including leading zeroes) and must be separated from other patterns for a given processor with white space (spaces, tabs, or newlines). Up to 64 patterns can be

specified per processor. If **random** is entered as one or more of the patterns, then an 8-byte random pattern is generated and used. The same 8 bytes will be written repetitively to all the area being purged and then will be verified.

A comment can be placed anywhere in the file by using a **#**. From that point to the end of the line is ignored.

Example of the **purge_patterns** file:

```
# This is a sample purge_patterns file
cpu 0000000000000000 ffffffffffff random     # three patterns for cpu
ccu aaaaaaaaaaaaaaaa                         # one pattern for ccus
spu random                                   # two patterns for spu
   2222222222222222
```

When **random** is specified, an 8-byte number is generated from current time by calling the function *srand(time(0))* to seed the random number generator and then by calling *rand()* eight times to get eight 1-byte values from which the 8-byte pattern is constructed. Each specification of **random** in the pattern file will result in eight new calls to rand() to generate a new 8-byte random number.

FILES

/mnt/bin/security_clear
/mnt/bin/lib/security_clear/purge_cpu
/mnt/bin/lib/security_clear/purge_iop
/mnt/bin/lib/security_clear/purge_viop
/mnt/bin/lib/security_clear/purge_hsp
/mnt/bin/lib/security_clear/mm_purge_ccu
/mnt/bin/lib/security_clear/mm_purge_spu
/mnt/bin/lib/security_clear/purge_patterns

SEE ALSO

rand(3)
srand(3)

NAME

sfpread – read/modify the SPU front panel switches

SYNOPSIS

sfpread [**-i**] [[**-v**] **field-name**]

DESCRIPTION

Sfpread is used to read and modify the contents of the SPU front panel. This utility operates in two different modes. If **field-name** is specified (with or without the **-v** option), a value corresponding to the current setting for that field is returned to the shell. If **-i** (interactive) is specified, **sfpread** emulates the front panel program **spu1000** and thus allows the front panel switches to be modified on a running system.

If the **-v** (verbose) option is supplied in addition to a field-name, sfpread prints the specified field name and its value in addition to returning that value to the shell.

DIAGNOSTICS

If the specified field-name does not exist or if any other error condition is found, **sfpread** exits with exit status -1. However, if invoked with the **-v** option, **sfpread** always exits with status zero.

FILES

/mnt/bin/sfpread

NAME

sp2util – SP2 register and memory utility

SYNOPSIS

sp2util

DESCRIPTION

The **sp2util** utility displays and modifies SP2 memory locations. When the modify mode is entered, the displayed value can be changed by entering the new hex value and hitting <**RETURN**>. If no value is entered, the next memory location is displayed. A **q** terminates the modify and returns to the **Cmd:** prompt.

COMMANDS

Commands are of the general form:

command parameters

All address values are in hex. A **q** terminates any command.

m a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one byte at a time.

mb a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one byte at a time.

mw a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one word at a time.

ml a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one longword at a time.

f a1 [a2] value The contents of memory locations a1 through a2 are filled, on a byte basis, with value. If a2 is not specified, only location a1 is changed.

fb a1 [a2] value
The contents of memory locations a1 through a2 are filled, on a byte basis, with value. If a2 is not specified, only location a1 is changed.

fw a1 [a2] value
The contents of memory locations a1 through a2 are filled, on a word basis, with value. If a2 is not specified, only location a1 is changed.

fl a1 [a2] value The contents of memory locations a1 through a2 are filled, on a longword basis, with value. If a2 is not specified, only location a1 is changed.

M a1 disp Causes the memory modify mode to be entered. Memory is displayed starting at a1 with the option of modifying each byte displayed. The byte increment between displayed memory locations is "disp."

cp a1 a2 a3 Copies a block of memory starting at a1 through a2 and places it at a3.

tg a1 cnt val Writes val to a1, then waits for a period of time proportional to cnt; then writes out the exclusive-or of val out to a1. This command continues until interrupted by <**CNTRL C**> or <**CNTRL ?**>.

rm regname Allows examination or modification of the contents of memory-mapped, SP2-based registers, which are displayed. Entering a <**RETURN**> displays the next register. Entering a hex value followed by a <**RETURN**> changes the contents of the register if it is a writable register. Entering <^> displays the previous register. Entering **q** exits this mode. The valid register names are:

ccsr	Console control/status register
cudr	Console data register
rcsr	Remote control/status register
rudr	Remote data register
tcsr	Timer control register
tdr	Timer data register
scsi_ctrl	SCSI status register
scsi_data	SCSI data register
ier_msw	Interrupt enable register (msw)
ier_lsw	Interrupt enable register (lsw)
isr_msw	Interrupt status register (msw)
isr_lsw	Interrupt status register (lsw)
lmcr	Local memory control register
cop	COP register
irs	SIB interrupt status register
icr	SIB interrupt channel register
ssn	System serial number
bsr	Bus error register
emr	Environmental monitor register
cpr	Control panel register
trr	Test result register
pcr	Physical configuration register
cfr	Clock frequency register
srr	System reset register
rfcnt	Refresh count register
esr	Error source register
sel	Soft error log register
ebus_log	EBUS log register
rhr	Run halt register
dcon	Diag connect register
sfr	Scan feedback register
dcr	Diag control register
ecr	Event counter register
odena	Output data enable register
mem_log	Memory log run register
mem	Memory run register
proc_a	Processor a run register
proc_b	Processor b run register
proc_c	Processor c run register
proc_d	Processor d run register
i/o	I/O run register

misc_log	Misc log run register
pbus_x	PBUS x clock mask register
pbus_y	PBUS y clock mask register

rd regname Displays the bits in the named register. Each bit position is labeled with the function of the bit. Each bit may be modified individually by moving the cursor over the bit and entering a 0 or a 1. The cursor moves to the right (toward the lsb) when a new bit value is entered, or when the **<SPACE BAR>** is depressed. The cursor moves to the left when a **<DEL>** or **<BACK SPACE>** key is pressed.

The input terminates when a **<RETURN>**, **q**, **<^>**, or **=** is entered. A **<RETURN>** updates the register, then displays the next register. A **q** exits this mode. Entering a **<^>** updates the register with the modified value, then displays the previous register. Entering a **=** updates the register with the modified value, then redisplays the same register.

!<command>

Fork a shell and execute the specified command. Use **!sh** to fork a shell.

q[uit] Exits **sp2util**.

? Prints out a list of valid commands.

NAME

syshalt – immediately halt the computer on a subsystem basis

SYNOPSIS

syshalt [*subsystem* [*subsystem*]]*

where subsystem can be:

cpus	Halt all CPUs installed
cpu0 or **cpua**	Halt CPU installed as processor A
cpu1 or **cpub**	Halt CPU installed as processor B
cpu2 or **cpuc**	Halt CPU installed as processor C
cpu3 or **cpud**	Halt CPU installed as processor D
mem	Halt memory subsystem
io	Halt CPX, PIA, and all CCUs

DESCRIPTION

The **syshalt** utility turns off the clocks for the subsystem specified as command parameters. If no parameters are included, then all subsystem clocks are turned off. Clocks are turned off by writing a zero into a subsystem run bit of the Service Processor Unit (SP2) run halt register.

SEE ALSO

sysreset(1d)
scn_halt(3d)

NAME

sysreset – reset the computer system

SYNOPSIS

sysreset [[*subsystem* [*subsystem*]*] *-l LEVEL*]+

where subsystems can be:

cpus	Reset all CPUs installed
cpu0 or **cpua**	Reset CPU installed as processor A
cpu1 or **cpub**	Reset CPU installed as processor B
cpu2 or **cpuc**	Reset CPU installed as processor C
cpu3 or **cpud**	Reset CPU installed as processor D
mem	Reset memory subsystem
io	Reset utility board, peripheral interface adapter, and all CCUs

LEVEL is an integer value defining what type of reset is being performed on the particular subsystem.

DESCRIPTION

The **sysreset** utility resets the computer based on subsystems and reset levels. Using **sysreset** requires the user to understand one thing: the actual reset levels are subsystem dependent (i.e., some subsystems have multiple levels of resetting while others have a single default level). Attempting to reset a subsystem to a level higher than defined will cause that subsystem to be reset to its maximum level.

The **sysreset** program first stops the clocks for the identified subsystems and issues a basic number of clocks to place the hardware into a stable condition. This operation is performed by referencing **reset** with the appropriate System Reset Register (SRR) mask. The individual subsystems are then reset to the desired level by referencing **sys_init** with the SSR mask for the subsystem and the desired reset level.

> **NOTE**: Stopping subsystem clocks and stabilizing the hardware is a parallel operation, while resetting subsystems is a sequential operation.

The **sysreset** command can be used in five different ways to reset the desired subsystems:

sysreset <RETURN>
resets all subsystems to default level 0.

sysreset -l3 <RETURN>
resets all subsystems to default level 3.

sysreset cpu0 -l2 <RETURN>
resets processor A (VPCA, VPDA, ASFA, IPPA, DCUA, FSUA) to level 2.

sysreset cpu0 cpu1 -i1 mem -l3 <RETURN>
resets processors A and B to level 1 and the memory subsystem to level 3.

sysreset io <RETURN>
resets the I/O subsystem to default level 0.

SEE ALSO

reset(3D)
sys_init(3D)

NAME

version – set/display version number, compile time, and date of diagnostic executables

SYNOPSIS

version [[**–V**] | [**–v** *version*]] *file* [*file* ...]

DESCRIPTION

The **version** utility sets or displays version numbers in *b.out* and *SOFF* files, with magic numbers of (octal) 405, 407, 410, 411, 425, 427, 430, 431, 601, and 603. Version numbers are of the form **X.X.X.X**, where **X** is an integer from 0 to 255. The time and date of compilation are displayed in the format returned by *ctime (3)*.

Information for each file is displayed on a separate line; the file name, file type, compile time and date, and version are displayed, as available. Displayed versions will have at least the first two numbers; the last two numbers will be displayed only if they are not zero.

The **-V** and **-v** options may be used to set the version information of the indicated file(s). The **-v** option requires the version to be specified. The **-V** option increments a version number contained in the file */diag/bin/version.data*, and provides a unique, incremented version number. When used, either option must be the first argument. In the case of the **-v** option, the version must be the second argument. Regular use of the **-V** option allows easy tracking of unreleased software. Both options are available only from the host executable; the service processor executable does not have the capability to set the version. The compilation time and date information contained in the header of *soff* format files is unaffected when setting the version, however this information is set to the current time and date for *b.out* format files.

While the host executable allows only file names to be specified, either files or directories may be specified for the service processor executable. If a directory is specified, the service processor version will print the version of all *SOFF* and *b.out* files it contains. In this case, the message *"<file name> is a directory - checking its files"* is displayed.

If a file or directory cannot be opened, the message *"unable to open <file name> - skipping"* is displayed. If a problem is encountered moving around (*fseek*) in a file, the message *"unable to fseek - file <file name>"* is displayed. If a read of a file fails, the message *"unable to read <file name> - skipping"* is displayed. If a write of version information to a file fails, a message in the format *"unable to write [version | time] to [soff | b.out] file <file name>"* is displayed. If the set version option is invoked on the service processor, the message *"version setting option available on host only"* is displayed. If a file is not *SOFF*, *b.out*, or a service processor directory, the message *"unable to process <file name>"* is displayed. If a *b.out* file contains no revision information, the phrase *"no revision information present"* is displayed. If a *b.out* file uninitialized revision information, the phrase *"revision uninitialized"* is displayed.

In *SOFF* format files (magic numbers (octal) 601 and 603), the version number and compilation date are reserved fields in the file header (refer to *a.out*(5)). In *b.out* format files (magic numbers (octal) 405, 407, 410, 411, 425, 427, 430, 431), the version number and compilation date are placed at the extreme end of the *b.out* format file (after the data relocation information).

SEE ALSO

a.out(5)
b.out(5)

NAME

val_perr – set internal parity error handling mode VAL gate array

SYNOPSIS

val_perr on | off | hard | soft

DESCRIPTION

The utility **val_perr** is used to set the internal validity RAM parity error handling mode of the VAL gate array in 2219 EDC boards. Invoking **val_perr** with the **on** or **hard** paramater will cause the CPUs to assert a parity error on the EDC and pull a hard error when a VAL RAM parity error is detected. Invoking **val_perr** with either the **off** or **soft** parameters will cause the affected EDC to not pull a hard error or assert a parity error, but will instead cause the EFU to simply ignore the parity error. This command must be executed before the CPU is initialized. Once the CPU is running, this command has no affect.

IMPLEMENTATION

This command is implemented by modifying the **VAL_PE_CONTROL** entry in the */mnt/diag_db* database. Subsequent initialization routines look at the value in this database to determine how to perform initialization. The state of the **VAL_PE_CONTROL** entry governs the state of the **soft_ctl** scan field in the EFU. If hard errors are enabled, then the value of **soft_ctl** will be 00; and if hard errors are disabled, then the value of **soft_ctl** will be FF.

Running a CPU with VAL RAM parity errors not generating hard errors is desireable since any transitory RAM parity errors will be ignored instead of halting the CPU and crashing ConvexOS. Typically, spurious VAL RAM parity errors occur very rarely, but even one is enough to crash ConvexOS. The only danger of running with VAL RAM parity errors off is that a bad VAL might not be detected, but instead just degrade data cache performance.

FILES

/mnt/diag_db

SEE ALSO

diaginit(1D)
mkdiag_db(1D)

NAME

vioputil – viop register and memory utility

SYNOPSIS

vioputil [VIOP number]

DESCRIPTION

The **vioputil** utility displays and modifies VIOP memory locations. When the modify mode is entered, the displayed value may be changed by entering the new hex value and hitting <RETURN>. If no value is entered, the next memory location is displayed. A <q> terminates the modify and returns to the `Cmd:` prompt.

COMMANDS

Commands are of the general form:

command parameters

All address values are in hex. A <q> terminates any command.

m a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one byte at a time.

mb a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one byte at a time.

mw a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one word at a time.

ml a1 [a2] Displays the contents of memory locations a1 through a2. If a2 is not entered, location a1 is displayed allowing modification of its contents. The locations are displayed and modified one longword at a time.

f a1 [a2] value The contents of memory locations a1 through a2 are filled, on a byte basis, with value. If a2 is not specified, only location a1 is changed.

fb a1 [a2] value
The contents of memory locations a1 through a2 are filled, on a byte basis, with value. If a2 is not specified, only location a1 is changed.

fw a1 [a2] value
The contents of memory locations a1 through a2 are filled, on a word basis, with value. If a2 is not specified, only location a1 is changed.

fl a1 [a2] value The contents of memory locations a1 through a2 are filled, on a longword basis, with value. If a2 is not specified, only location a1 is changed.

M a1 disp Causes the memory modify mode to be entered. Memory is displayed starting at a1 with the option of modifying each byte displayed. The byte increment between displayed memory locations is "disp."

cp a1 a2 a3 Copies a block of memory starting at a1 through a2 and places it at a3.

rm regname Allows the examination or modification of the contents of memory-mapped VIOP-based registers, which are displayed. Entering a <RETURN> displays the next register. Entering a hex value followed by a <RETURN> changes the contents of the register if it is a writable register. Entering a <CTRL> displays the previous register. Entering a <q> exits this mode. The valid register names are:

ier	Interrupt enable register
ierm	Interrupt enable register (most significant word)
ierl	Interrupt enable register (least significant word)
isr	Interrupt status register
isrm	Interrupt status register (most significant word)
isrl	Interrupt status register (least significant word)
mcr	Memory control register
pis	PBUS interrupt channel
ber	Bus error log register
pber	Parity/bus error address
pblgm	PBUS log register (most significant word)
pblgl	PBUS log register (least significant word)
pic	PBUS interrupt channel number
clg	Cache log
earm	Error address register (most significant word)
earl	Error address register (least significant word)
trr	Test results register
auxtrr	Auxiliary test results register
mbdr	Multibus diagnostic register
misc	Miscellaneous diagnostic register
pstat	Parity status register
slotid	Slot id register

rd regname Displays the bits in the named register. Each bit position is labeled with the function of the bit. Each bit may be modified individualy by moving the cursor over the bit and entering a 0 or a 1. The cursor moves to the right (toward the LSB) when a new bit value is entered, or when the <SPACE BAR> is depressed.

The cursor moves to the left when a <DEL> or <BACK SPACE> key is pressed. The input is terminated when a <RETURN>, <q>, <CTRL>, or <=> is entered. A <RETURN> updates the register and then displays the next register. A <q> exits this mode. Entering a <CTRL> updates the register with the modified value, then displays the previous register. Entering a <=> updates the register with the modified value, then redisplays the same register.

q[uit] Exits **vioputil**.

? Prints out a list of valid commands.

NAME

vp_scn – vector processor scan utility

SYNOPSIS

vp_scn

DESCRIPTION

The **vp_scn** utility is used to manipulate the vector processor scan rings for CONVEX C200 Series computers.

Commands available to the user include:

get Get the scan fields associated with the vector processor of the current head. For example, **get vpd:red_led** prints the field *red_led* on the current head. Also, **get** excepts certain regular expressions, similar to the sh expansion of regular expressions. So, typing **get vpd:*err** gets all the fields that end in *err* on the vpd ring on the current head: *vre_par_err*, *vro_par_err*, *is_par_err*, *os_par_err*, *vm_par_err*, and *hard_err*.

put Put a value into the selected field on the vector processor of the current head. For example, **put vpd:red_led 1** writes a 1 to the local copy of the scan ring vpd[<head>]:. For this value to reach the hardware, the **write** command would have to be given.

exit Leave the **vp_scn** program and return to SPU UNIX.

read This command forces the software to update its local copy of the scan rings with the hardware scan ring. It is required before the **get** and **read_vr** in order for the information to be up-to-date.

write Force the scan rings on the current head to be written out to the hardware. This command must be issued or the previous commands may be lost.

reset_rings
This command copies the rings to an internal buffer and writes them to the hardware.

clock_vpd This command clocks the vpd ring on the current head.

getrings This command copies the ring buffers to the file *rings.data*.

verify This command does a comparison of the buffers to the hardware.

read_vr This command reads the vector registers. The form of the command is as follows:

read_vr <vector register> <lower limit> <upper limit>

Thus, to read the third vector register elements 5 through 7, type:

read_vr 3 5 7

The output would be:

```
VR3[5] = 00000000 00000000 (0 0)
VR3[6] = 00000000 00000000 (0 0)
VR3[7] = 00000000 00000000 (0 0)
```

In the output above, the left 32-bit number displayed is the most significant word

and the right 32-bit number is the least significant word. The parity for each is displayed in parentheses to the right of the vector register contents.

write_vr This command writes the contents of a specified vector register and calculates and writes the correct parity with it:

write_vr <vector register> <element> <upper word> <lower word>

To write the value 12345678 87654321 into the fifth element of vector register 0, type:

write_vr 0 5 12345678 87654321

get_vm This command gets the VM register value from an internal buffer, which must be initialized by the **read** command.

get_vl This command gets the VL register value from an internal buffer, which must be initialized by the **read** command.

help Display a short description of the commands available.

head Change the current head to the value specified by the command

head <head letter>

To change the head from the default value to head B (cpu 1), type:

head b

q Same as **exit**.

FILES

rings.data File to which the **getrings** command writes the internal scan buffers.

SEE ALSO

cpuvreg(1D)

DIAGNOSTICS

The *vp_scn* utility has limited error checking. It does not recover well from mistyped commands and user mistakes.

NAME

x – hexadecimal/decimal calculator

SYNOPSIS

x [expression]

DESCRIPTION

X evaluates an arithmetic expression using 32 bit integer arithmetic. The expression may include hexadecimal numbers, decimal numbers, octal numbers, ( )'s, and the operations - (unary negation), ~ (unary inversion), *, /, % (remainder), +, and - . Unary operations are given highest precedence; multiplication and division next; addition and subtraction have lowest precedence. ( )'s may be used to override precedence. Hexadecimal numbers are specified with leading zeros (e.g. 017 = 17 hex = 23 decimal) or with a leading **x**. Octal numbers are specified with a leading **o** (letter o).

If no arguments are given, **x** prompts for expressions, one per line. To exit, respond with **q** or **<CR>**.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5
Diagnostic File Formats

5.1 Overview

This chapter contains detailed explanations of all pertinent diagnostic file formats.

THIS PAGE INTENTIONALLY LEFT BLANK

NAME

DB_cop – system board configuration database file

DESCRIPTION

The **DB_cop** utility is an ASCII file that contains information on CONVEX system boards. Board entries are organized by CONVEX part numbers. Each part number is followed by the board name (e.g., viop, sfu, mcm). In addition, one or more lines specify an assembly revision range with the corresponding diagnostic scan ring revision. Typically, **DB_cop** is used to translate the board part number and assembly revision stored in a board's COP chip to the scan ring revision used by various diagnostic tests. Lines that begins with a **#** are assumed to be comments. An example format follows:

```
#
# Cop Data Base File
#
# Entry format:
#
# PN    type
#        arr    rrn
#
# where  PN  = board part number (leading zeros not needed)
#         type= type of board (mcm, asp, etc)
#       arr = assembly revision range (e.g., a, a-b)
#         rrn = scan ring revision number [1,2,3,...]
#
001201  cpx
        a-zz    1
002201  cpx
        a-e     1
        f-zz    2
001205  vpc
        a-zz    1
002212  pia
        a-zz    2
```

SEE ALSO

cop(1d)

NAME

DB_diskfmt – disk parameters for diagnostics

DESCRIPTION

The **DB_diskfmt** utility contains all essential disk drive parameters needed by the Multibus disk and controller diagnostic (*dev4100*), the Multibus disk formatter (*dev4110*), and the combined VMEbus disk controller and drive diagnostic and formatter (*dev5130*). Each line is either a comment (starts with a #) or defines a drive's parameters needed to properly perform I/O to the drive and for format of the drive. Below is an example of the complete file as released in Diagnostics Database V2.6.

```
# DB_diskfmt - file of disk parameters
#
# >>>> WARNING - DO NOT USE 'diskfmt' TO FORMAT! It is no longer compatible
# >>>>           with the CONVEX FORMAT! Instead, format MBUS-attached drives
# >>>>           with 'dev4110' and VME-attached drives with 'dev5130'.
#
# KEY FOR DRIVE NAMES (unformatted capacity is given in parentheses):
#
#  Name        Description          Name        Description
# DKD-001      Fujitsu Eagle (452MB) DKD-008,208  NEC 2363 (1080MB)
# DKD-002      CDC 9766 (300MB)      DKD-214      Hitachi DK514-38 (356MB)
# DKD-005,206  NEC 2352 (500MB)
#
#
#------- XYLOGICS 450/451 SMD CONTROLLER (MBUS) -------
#  a   b   c  d  e  f   g    h     i j k l m  n  o p
DKD-001 2  842 20 46 45 4800 28160 1 0 1  0  0  smd mfm y
DKD-002 0  823 19 32 31 5040 20160 1 0 1  0  0  smd mfm y
DKD-005 0  760 19 60 59 4832 36288 1 0 1  0  0  smd 2-7 y
DKD-008 1 1024 27 68 67 4816 40960 1 0 1  0  0  smd 2-7 y
#
#------- INTERPHASE 4201 ESDI CONTROLLER (VME) --------
#  a   b   c  d  e  f   g    h     i j k l m  n  o p
DKD-214 0  903 14 51 50 4736 30240 5 5 1  8  8 esdi 2-7 n
#
#------- INTERPHASE 4200 SMD CONTROLLER (VME) ---------
#  a   b   c  d  e  f   g    h     i j k l m  n  o p
DKD-206 0  760 19 60 59 4832 36288 5 4 1 12 12  smd 2-7 n
DKD-208 0 1024 27 68 67 4816 40960 5 6 1 16 12  smd 2-7 n
#
# LEGEND:
# a - drive name              Must be DKD-0XX for Multibus and DKD-2XX for
#                           VMEbus
# b - disk type               For Xylogics controller
# c - # of cylinders
# d - # of heads
# e - # of physical sectors    Number of actual sectors excluding runt
# f - # of logical sectors     Number of physical sectors (e) minus number of
#                           spares
# g - bits per sector         Number of bits between sector pulses
# h - bytes per track         Total number of unformatted bytes per track
# i - skew                    Sector offset from one head to the next
```

```
#                          Must be 1 when using Xylogics 450/451 cntlr
# j - # of relocation tracks   .5% of number of cylinders (c). Raise
#                          fractional part to next higher whole number.
#                          Ignored by dev4110 (Multibus formatter)
# k - interleave             sector separation between consecutive sectors
#                          Currently must be 1 for Xylogics 450/451 cntlr
# l - gap 1 size             Number of halfwords in gap before header
#                          (2 bytes per halfword)
#                          Ignored by dev4110 (Multibus formatter)
# m - gap 2 size             Number of halfwords in gap following header
#                          (2 bytes per halfword)
#                          Ignored by dev4110 (Multibus formatter)
# n - drive interface        Used to determine how to read manufacturer's
#                          defect map.  Currently, smd or esdi
#                          Ignored by dev4110 (Multibus formatter)
# o - data encoding scheme   Way data is encoded on the media. Used to
#                          select patterns for pattern test.
#                          Currently mfm, 2-7 or 1-7
# p - Are spares interleaved  For Xylogics, 'y'. For Interphase, 'n'.
```

Note that file contains details about each of the fields.

SEE ALSO

get_defects(1D)
disktab(5)

NAME

controllers – valid controller names for device diagnostics

DESCRIPTION

The *controllers* utility contains the valid controller names for each device test as found in the */ioconfig* file. Each line is either a comment, which starts with a "#," or a device test name followed by the controller names accepted by that test. Each line may have up to 256 characters.

The following is an example of an entry in the **controllers** file:

```
# IKON 10085 Multibus Versatec Plotter controller
dev4410 VER-001 VER-002 PRC-002
```

Note that the character "#" starts a comment.

The **controllers** file resides in */mnt/bin/lib* on the SPU disk.

NAME

ioconfig – system I/O configuration description file

SYNOPSIS

The **ioconfig** file resides in the root of the SPU disk.

DESCRIPTION

The **ioconfig** file contains the description of all the Channel Control Units (CCUs), Multibus and VMEbus chassis, and peripheral controllers for a given system configuration. It resides in the SPU root partition and is used by the operating system and diagnostics for device configuration information. The **ioconfig** file consists of mixtures of three types of entries which are associated with the three types of CCU boards supported by CONVEX. These entries are illustrated below:

iop *slot_number*
  mbus *chassis_number*
    ctlr *driver_name* **csr** **0x***address* **int** *level*
      unit *unit_number* **type** *device_type*
      unit *unit_number* **type** *device_type*
      * additional units on this controller *
    ctlr *driver_name* **csr** **0x***address* **int** *level*
      unit *unit_number* **type** *device_type*
      unit *unit_number* **type** *device_type*
      * additional units on this controller *
    * additional controllers and units on this chassis *
  mbus *chassis_number*
    * controllers and units on this chassis *
  * additional Multibuses, controllers, and units on this IOP *

or

viop *slot_number*
  vme *chassis_number*
    ctlr *driver_name* **csr** **0x***address* **int** *level*
      unit *unit_number* **type** *device_type*
      unit *unit_number* **type** *device_type*
      * additional units on this controller *
    ctlr *driver_name* **csr** **0x***address* **int** *level*
      unit *unit_number* **type** *device_type*
      unit *unit_number* **type** *device_type*
      * additional units on this controller *
    * additional controllers and units on this chassis *
  vme *chassis_number*
    * controllers and units on this chassis *
  * additional VMEbuses, controllers, and units on this VIOP *

or

hsp *slot_number*
  drvr *driver_name* **csr** **0x***address*
    chnl *channel_number* **type** *device_type*

where

slot_number
: is the CCU slot number where the channel board (IOP, VIOP, or HSP) resides. For CONVEX C1 and C120 configurations, this number ranges from 1 to 7. For the CONVEX C130, C210, and C220 configurations, this number ranges from 0 to 15.

chassis_number is one of two chassis connected to the VIOP and IOP via cables. The chassis are numbered 0 and 1.

driver_name
is the name of the driver (computer program that interfaces with a controller board in the chassis). There are typically several acceptable names for each driver. The current set of names and associated controllers are listed below. However, be aware that this list changes frequently as new peripherals are added to our inventory.

CHASSIS	DRIVER_NAME	CONTROLLER BOARD
Multibus	DKC-001, DKC-002	Xylogics 450/451 SMD Disk Controller
Multibus	MTC-001, MTC-002	STC Tape Controller
Multibus	ACM-001, ACM-002	Systech Async TTY Controller
Multibus	PRC-001	Systech Line Printer Controller
Multibus	VER-001, VER-002	IKON 10085 Versatec Plotter Controller
Multibus	LAN-001, COV-002	IKON 10077 HYPERchannel Controller
Multibus	DMA-001, GPI-001	IKON DR11-W Emulator
Multibus	HEC-001	High-Speed Peripheral Interface
VMEbus	DKC-203	Interphase 4201 V/ESDI Disk Controller
VMEbus	DKC-204	Interphase 4200 V/SMD Disk Controller
VMEbus	LAN-007	EXOS 202 Ethernet Controller

address is the Control and Status Register (CSR) address within the chassis that corresponds to the controller board. Normally this address is set using jumpers or switches on the controller board itself.

level is the interrupt level used by the controller board to get the attention of the CCU.

unit_number
is the address of a device attached to a controller. For example, up to four drives can be attached to a Xylogics 451 controller and are numbered 0 through 3.

device_type
is the name of the device attached to the controller. An incomplete list of valid names is shown below:

UNIT_NAME	MULTIBUS DEVICE
DKD-001	Fujitsu Eagle disk drive
DKD-002	CDC 9766 Washtub disk drive
DKD-005	NEC 2352 500-Mbyte disk drive
DKD-008	NEC 2363 1-Gbyte disk drive
TTY,RASTER	TTY port name. RASTER doesn't handle ^s & ^q
MTD-001	STC 2921 tape drive
MTD-002	STC 196x tape drive
MTD-003	Fujitsu 200 IPS tape drive
PLT-001	Versatec Flat-bed plotter
PRT-001	Printronix Model P600 or P6080 printer

GPD-001	Graphics terminal attached to DR11-W
ex	Ethernet
COV-002	Ethernet for COVUE
HYP-001	HYPERchannel A400 device driver
HED-001	HSP-Echo Board

UNIT_NAME	VMEBUS DEVICE
DKD-206	NEC 2352 500-Mbyte disk drive
DKD-208	NEC 2363 1-Gbyte disk drive
DKD-214	Hitachi 514-38 Removable Disk
ve	Ethernet

Below is an example of an **ioconfig** file that supports 7 disks (1 on a Multibus and 6 on two VMEbuses), 1 LAN interface, 1 Versatec printer, 1 Versatec plotter, 1 HSP, and 16 terminal ports:

```
viop 3
   vme 0
      ctlr DKC-204 csr 0xc00 int 2
         unit 0 type DKD-208
      ctlr DKC-204 csr 0xe00 int 3
         unit 0 type DKD-208
   vme 1
      ctlr DKC-203 csr 0x200 int 1
         unit 0 type DKD-214
         unit 1 type DKD-214
      ctlr DKC-203 csr 0x600 int 2
         unit 0 type DKD-214
         unit 1 type DKD-214
   hsp 4
      drvr HEC-001 csr 0x0
         chnl 0 type HED-001
iop 7
   mbus 0
      ctlr DKC-002 csr 0x3f0 int 2
         unit 0 type DKD-005
   mbus 1
      ctlr LAN-001 csr 0x4c0 int 5
         unit 0 type ex
      ctlr PRC-001 csr 0x2e0 int 6
         unit 0 type PRT-001
      ctlr VER-001 csr 0x2c0 int 1
         unit 0 type PLT-001
      ctlr ACM-001 csr 0x3c0 int 7
         unit 0 type TTY
         unit 1 type TTY
         unit 2 type TTY
         unit 3 type TTY
         unit 4 type TTY
         unit 5 type TTY
         unit 6 type TTY
         unit 7 type TTY
```

```
unit  8 type TTY
unit  9 type TTY
unit 10 type TTY
unit 11 type TTY
unit 12 type TTY
unit 13 type TTY
unit 14 type TTY
unit 15 type TTY
```

SEE ALSO

ca(4)
da(4)
ex(4)
pa(4)
ta(4)

NAME

softlog – soft memory error log file for *errintd*

SYNOPSIS

/mnt/softlog

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 generates the **softlog** file, which consists of a header and a series of lines. One line per memory device (chip) is used. The header portion of the file contains the following items:

* The date the current **softlog** was created
* A field indicating whether the **softlog** is full
* A total error count
* An incremental count of failed devices (e.g., number of **softlog** entries)
* An incremental count of errors not logged due to throttling by *errintd*

Each entry in the body of the **softlog** follows this format:

MCM device S/N bit stk first_fail last_fail address count

MCM

is either the number of the memory board containing the faulty device (0 - 7) or, in later versions of *errintd*, a two character designation with a digit (0 - 3) indicating the board pair, and a letter (o or e) indicating the individual odd or even board within the pair (e.g., 2e = board pair 2, even board).

device

is the coordinates of the faulted device, coded as follows:

PP-SCCCRR for MCM1, P/N 1213 or UKKKWW-UNN for MCM2, P/N 3213

where:

PP	=	platter (e.g., UL, UR, LL, LR)
S	=	platter side (e.g., U, Z)
CCC	=	failed device column number (numeric characters only)
RR	=	failed device row number (e.g., A4, K5, etc.)

or

U	=	letter U
KKK	=	column number of the MIM containing the failed RAM
WW	=	row number of the MIM containing the failed RAM
NN	=	U-number of the failed RAM

S/N

is the serial number of the MCM containing the above device.

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 (from this device) that have been corrected.

The **softlog** file may be examined from CONVEX UNIX by entering the following command:

/usr/convex/spu -r /mnt/softlog

SEE ALSO
errintd(1d)
mm_sniff(1d)

Appendix A
Glossary

A.1 Overview

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.

A.2 Terms

AC power-controller. The AC power-controller is the device that regulates AC power from the cabinet circuit breaker to the computer's internal electronic and electromechanical components.

access mode. Any of the five processor access modes in which software executes. On the CONVEX system, processor access modes are (in order from most to least privileged and protected): **kernel** (mode 0), **executive** (mode 1), **supervisor** (mode 2), **agent** (mode 3), and **user** (mode 4). The operating system uses access modes to define **protection** levels for software executing in the context of a **process.**

A register. *See* **address register.**

accumulator. A hardware register containing the results of arithmetic and logical operations.

address. A user-assigned number used by the operating system to identify a storage location.

address register. Abbreviated A register. A register containing the address of the instruction currently being executed.

address space. Address space, either physical or virtual, available to a process.

address translation fault (ATF). An **exception** that results from a **page table entry** violation or a nonresident page.

address translation unit (ATU). Translates logical addresses to physical addresses and stores them in a **cache**; thus, the ATU is an address cache that accelerates the generation of **physical addresses.**

addressing mode. How the effective address of an instruction **operand** is calculated using the general registers.

agent. Processor access mode 3.

ALU. *See* **arithmetic logic unit**

architecture. The physical structure of a computer's internal operations, including its registers, memory, instruction set, input/output structure, and so on.

argument pointer. An address register specifically dedicated to point to the **subroutine** argument portion of a program. This program portion can either be in the **stack** or in part of **logical memory** pre-allocated by the **compiler**.

arithmetic logic unit. Abbreviated ALU. A basic element of the **Central Processing Unit** (CPU) where arithmetic and logical operations are performed.

array. An ordered structure of operands of the same data type. The structure of an array is defined as: length, rank (or dimension), stride, and data type.

ATF. *See* **address translation fault**.

ATU. *See* **address translation unit**.

b. *See* **byte**.

backplane. The circuitry and mechanical elements used to connect the boards of a system. Also called **motherboard**.

backplane (VMEbus). A Printed Circuit (PC) board with 96-pin connectors and signal paths that bus the connector pins. Some VMEbus systems have a single PC board, called the J1 backplane. It provides the signal paths needed for basic operation. Other VMEbus systems also have an optional second PC board, called a J2 backplane. It provides the additional 96-pin connectors and signal paths needed for wider data and address transfers. Still others have a single PC board that provides the signal conductors and connectors of both the J1 and J2 backplanes.

base-level interrupt. An interrupt that occurs when the kernel stack is the process stack; thus, a base-level interrupt occurs when no other interrupts are pending or currently being processed.

bit. A binary digit.

bit complement. Exchanging 0's and 1's in the binary representation of a number. Also known as 1's complement.

block. To stop the flow of execution. Execution cannot begin until the block no longer exists. Also called a **hazard**.

boot. The procedure by which a program is initiated the first time. Typically, a bootstrap is performed when power is first applied to the processor.

branch. A class of **instructions**, specifically relative to the program counter, used to transfer control of a program.

breakpoint. An instruction that aids in the debugging of a program. In particular, a breakpoint is a specific location in a program where one would desire to determine the various values of programmer-defined variables.

byte. The number of contiguous bits starting on an addressable byte boundary. A byte is 8 **bits**. Abbreviated **b**.

C. The systems programming language of the **UNIX** operating system.

C shell. The standard shell provided with Berkeley standard versions of UNIX.

cache memory. A small, high-speed buffer memory used in modern computer systems to hold temporarily those portion of the contents of the main memory that are, or believed to be, currently in use. CONVEX computers contain many separate caches, including **logical caches**, **physical caches**, and **instruction caches**.

cache purge. The act of invalidating or removing entries in a cache memory.

central processing unit. The Central Processing Unit (CPU) is that portion of a computer that recognizes and executes the instruction set.

central processing unit bulkhead. A special panel on the CPU cabinet. Because the CONVEX C120 is **Electromagnetic Interference** (EMI) shielded, cables that connect internal components of the C120 to the components or devices that are external to the CPU cabinet must pass through EMI shielded connectors mounted in a special panel called the CPU bulkhead.

chaining. Chaining is the ability to overlap vector operations in the CPU. For instance, in the case of a **vector** load followed by a vector add, the add may be started as soon as the first operands are available, rather than waiting for the **load** to complete.

chassis. The physical box where the computer is housed.

compiler. A software tool used to compile the source code of a high-level language, such as FORTRAN, into object code.

context (processor). The entire, current state of the machine associated with the executing process.

CONVEX UNIX. The CONVEX version of the UNIX operating system.

CPU. *See* **central processing unit**.

d. *See* **double**.

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 register or memory location that receives the result of the operation.

displacement. A derived 32-bit value used to indicate the distance in bytes between the referenced data and some base value. This base value can either be 0 or the contents of an address register. Note that 16-bit displacement values are sign extended to 32 bits.

double. Abbreviated **d**. A double-precision floating-point number that is stored in 64 bits.

drawer bulkhead. The Multibus drawer for the CONVEX C120 is Electromagnetic Interference (EMI) shielded. Cables that connect the internal components of the drawer to the components or devices that are external to the drawer must pass through EMI-shielded connectors mounted in a panel in the rear of the drawer. This panel is the drawer bulkhead. *See also* **Multibus drawer**.

EBUS. There are five ports on the memory system. These are referred to as ports A, B, C, D, and E. Ports A–D feed processors A–D; port E feeds the I/O system. Thus, EBUS is the bus to port E of the memory system.

electrostatic discharge. The release of static electricity from a charged object to a grounded object.

ESD. *See* **electrostatic discharge**.

exception. A hardware-detected event that disrupts the running of a program, process, or system. *See also* **fault**.

executive mode. Processor access mode 1.

expansion cabinet. A secondary cabinet designed to house peripheral computer equipment, e.g., tape drives, disk drives, controllers, etc. *See also* **processor cabinet**.

fault. An exception that, while halting the instruction, leaves the **registers** and memory in a consistent state. The instruction can often resume its course when the cause of the fault is corrected.

FIFO. Abbreviation for first-in, first-out. *See* **queue**.

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**.

first-in, first-out. *See* queue.

flag. A 1-bit operand that is generally used to indicate the results of an operation. The results are in the form true or false.

floating point. A numerical representation. A floating point operand has a sign (positive or negative) port, an exponent port, and a fraction port. The **fraction** is a fractional representation. The exponent is the value used to produce a power of two scale factor (or portion) that is subsequently used to multiply the fractions to produce an unsigned value. *See also* **fraction**; **guard bit**.

forced faulting mode. A mode of operation where the **CPU** diagnostics cause simulated **pagefaults** to occur. In forced faults mode, a bit is set in hardware so that each time data it is accessed in main memory, the entire context of the processor is saved off and then restored. This process thoroughly exercises the buses that are used to capture and restore the context of the machine as well as the entire memory system.

FORTRAN. A high-level software language used mainly for scientific applications.

fraction. A part of a **floating point** number. The fraction is the unsigned fractional part that denotes the magnitude of the operand.

frame. *See* **page frame**

***fsck* utility**. A file systems check program used for maintenance and repair of data stored on disk.

function unit. A part of **CPU** that performs a set of operations on quantities stored in **registers**.

gate array. A structure that is used by the **ring** protection mechanism to define the entry points from a lower privileged ring to a higher privileged ring.

gather. Loading a vector register using another vector of indices instruction. See the *ldvi* instruction.

guard bit. A bit to the right (**least significant bit**) of a floating point fraction. The guard bit is used in intermediate calculations using floating point operands. *See also* **round bit**.

h. Abbreviation for halfword. *See* **halfword**.

halfword. Abbreviated **h**. Two bytes (16 bits). *See also* **longword**; **word**.

hazard. A block in the flow of execution that cannot be passed until the hazard no longer exists. Also called **block**.

Huffman's encoding. A binary encoding that results in the densest packing of information.

Icache. *See* **instruction cache**.

immediates. **Operands** that are contained within the instruction stream.

indexing. The process of adding a **displacement** to the contents of an address register.

indirection. The process of obtaining the address of an operand by first referencing a word contained within memory.

input/output processor. The Input/Output Processor (IOP) is the standard input/output device in the CONVEX C120. The IOP performs all the functions required to move data between the **Memory Control Unit** (MCU) and the **Multibus subsystems**, including logical-to-physical address translation and 64-bit-to-16-bit data path conversions. Slots 17 through 21 in the C120 **backplane** are reserved for **Channel Control Units** (CCUs). The IOP is one of a variety of CCUs. At this writing, the IOP is the only CCU using the Multibus I for a front-end. The IOP provides two ports for attaching the Multibus subsystems.

instruction. Used by the programmer to direct operations on the system's register set and memory.

instruction cache. The Instruction Cache (Icache) contains the most recently accessed instructions. The Icache accelerates the decoding of instructions to permit the simultaneous decoding on one instruction with the execution of another instruction.

interrupt. An occurrence, other than an **exception**, that changes the normal flow of instruction execution. An interruption originates from hardware, such as an I/O device. *See also* **maskable interrupt**.

interval timer. A privileged register. The interval timer is used to generate an **interrupt** based on the passage of time.

IOP. *See* **input/output processor**.

jump. Departure from normal one-step incrementing of the program counter.

kernel. A part of the UNIX operating system that resides in ring0. The kernel typically manages process creation and deletion, scheduling, and other high-level, system-wide features.

keyswitch. On CONVEX supercomputers, a four-way electrical keyswitch that controls the application of electricity to the Central Processing Unit (CPU) boards.

l. *See* **longword**.

language specific information. Abbreviated LSI. The area in the **stack** that is created as part of a **subroutine** call. It is language-dependent and may be **zero**.

last-in, first-out. *See* **stack**.

LIFO. *See* **stack**.

linker. A software tool that links separate software modules into one module.

load. An instruction that moves data from memory to a register.

locality of reference. An attribute of a memory reference pattern that refers to the likelihood of an address of a memory reference being numerically close to a recent memory reference address, or the likelihood of a subsequent memory reference being identical to a previous memory reference within a given period of time.

logical address. Logical address space is that **address space** seen by the application programmer.

logical cache. A cache that is accessed with **logical addresses** for fast retrieval of data. It resides in the **CPU**.

logical memory. That memory seen by the programmer. The logical memory of a CONVEX computer is 4 Gigabytes. Also called **virtual memory**. *See also* **page**.

longword. Abbreviated l. Eight bytes (64 bits), the largest integer data type directly supported by hardware in the CONVEX C120. *See also* **halfword**; **word**.

LSI. *See* **language specific information**.

machine exceptions. Include fatal errors in the system that cannot be handled by the operating system. *See also* **exception**.

main memory. *See* **physical memory**.

maskable interrupt. An interrupt that the operating system does not wish not to respond to at this time.

MBCU. *See* **multibus control unit**.

Mbyte. *See* **megabyte**.

megabyte. 1 million bytes, abbreviated Mbyte.

memory management. The hardware and software features that control page mapping and **protection**.

microcode. A control program that resides within the CPU. Microcode also refers to the firmware that provides the necessary control to map assembly language instructions onto processor hardware.

mode. *See* **access mode**.

mode switch. On CONVEX supercomputers, a three-way electrical switch that controls power to the System Monitor Board (SMB) (on C100 Series models) or the System Control Monitor (SCM) (on C200 Series models). The mode switch is located on the **AC power-controller**.

modified bit. A bit within the CPU that records all valid write references to **page frames**. The modified bit is used by the operating system for **memory management**.

Multibus. Refers to the **backplane** section containing terminators for the backplane wiring, receptacles for **Multibus controllers**, and the backplane wiring to connect these components. The Multibus comes as a single, 9-slot backplane with one Multibus or as a dual 10-slot backplane with two electrically-isolated Multibuses. In each Multibus, one slot is reserved for the **Multibus Control Unit** (MBCU).

Multibus control unit. The Multibus Control Unit (MBCU) is the connecting link between the **IOP** and the **Multibus**. (Typical **Multibus subsystems** have a CPU interfaced to device controllers through the Multibus, with the CPU and the controllers installed in the Multibus.) In the CONVEX I/O subsystem, the IOP is the CPU for the Multibus subsystem and is installed in the C120 **cardcage**. It is, therefore, necessary to provide an interface between the IOP and the Multibus. This interface, the MBCU, must be installed for each Multibus connected to the IOP.

Multibus controller. Any device controller designed to function within the Multibus protocol, and meeting the CONVEX conformance standards. (At this writing, CONVEX does not make use of any Multibus board other than device controllers.)

Multibus cardcage. In CONVEX terminology, a chassis that provides a mounting frame for the Multibus (backplane) and support slots for installing Multibus controllers that plug into the Multibus. The same 10-slot cardcage is used for the 10-slot dual Multibus and the 9-slot single Multibus. The standard Multibus cardcage in the C120 is a 10-slot cardcage with a dual Multibus backplane installed. The standard configuration is with Multibus 0 cabled to Multibus Port 0 of the IOP. This term is synonymous with **Multibus drawer** and **Multibus chassis**.

Multibus drawer. The Multibus drawer consists of a hardware mounting kit, a rail mount drawer, a power supply, an optional Multibus (single or dual), at least one **MBCU**, a 10-slot cardcage, a cooling fan, and cables to connect the drawer to the **IOP**. The Multibus drawer is an option that must have additional support systems. The drawer requires a connection to an IOP, mounting space in a peripheral cabinet, and an AC power source. One IOP is capable of driving two drawers when the drawers are equipped with a single Multibus. *See also* **Drawer bulkhead**.

Multibus port 0,1. The Multibus is interfaced to the IOP via the **MBCU**. Two ports, Multibus Port 0 and Multibus Port 1, are provided on the IOP for attaching Multibuses. The connection for Multibus port 0 is the desired IOP slot J2X and J2Y. The connection for Multibus port 1 is the desired IOP slot J4X and J4Y. The X connection on the IOP cables to the **J2** connection on the MBCU. The Y connection on the IOP cables to the **J1** connection on the MBCU.

Multibus subsystem. The Multibus subsystem consists of the **Multibus**, the Multibus **cardcage**, the power system, the **MBCU**, and the cables required to interface the MBCU to the **IOP**. The Multibus subsystem comes in two configurations, internal and expansion. The internal Multibus subsystem is located within the CPU's EMI-shielded cabinet and draws its power from PS2 or PS3. The standard internal Multibus subsystem uses a 10-slot Multibus cardcage with a dual Multibus. The expansion Multibus subsystem (drawer) uses the 10-slot cardcage and can be equipped with a dual or single Multibus.

multi-user mode. In **CONVEX UNIX**, the **mode** of operation where the supercomputer is being run in a general timesharing environment with multiple users. This is the normal operating mode for CONVEX UNIX. *See also* **single-user mode**.

negate. An **instruction** that performs a 2's complement on a number.

normalization. The process of left-shifting a **fraction** until the leading bit is a one.

opcode. The code or sequence of **bits** in an **instruction** that determines the operation to be performed.

operand. A register or memory location referenced by an instruction. It is the code or sequence of bits in an instruction that determines the data to be operated on.

optimize. Arranging instructions or data in storage so a minimum amount of machine time is spent for accessing the instructions or data.

orthogonality. A characteristic that pertains to the relationship of **instructions** and the **operands** they manipulate. An instruction set is orthogonal if a change in one property does not necessitate changes in other related properties.

packets. A group of related items. A packet may refer to the **subroutine** arguments or to a group of bytes that is transmitted over a network.

page. A page is the unit of logical memory controlled by the memory management algorithms. In the CONVEX C120, a page is 4 K (4,096) contiguous bytes. *See also* **logical memory**.

pagefault. A pagefault occurs when a process requests data that is not currently in main memory. The machine first saves off the state of all controllers onto a context stack in main memory. The operating system will create a free page of **physical memory** to bring the data in from the disk. The appropriate **Page Table Entries** (PTEs) are set up so that the proper logical-to-physical translation occurs. The machine reads back from memory the state of the machine from the context stack, and restores the processor to the same state that it was in when it determined that the data it needed was nonresident. The CPU can then continue with normal operation of the process.

page frame. A page frame is the unit of physical (main) memory in which pages are placed. Referenced and modified bits associated with each page frame aid in **memory management**.

page table entry. A Page Table Entry (PTE) is an entry in a page table. A PTE is a word that contains various **flags** and fields that are used in translation of logical-to-physical addresses. Address translation uses two levels of page table **indexing**. The first-level page table is referenced using bits <28> through <22> of a logical address. This is called the *Index.1* field. The second-level page table is referenced using bits <21> through <12> of a logical address. This is called the *Index.2* field.

PBUS. The PBUS is the primary internal interface between the I/O CCUs and other C120 subsystem components.

physical address. Hardware-identified address in physical (main) memory consisting of the **page frame** number and the number of a byte within the page.

physical cache. Provides rapid access to recently used **physical memory** data items.

physical memory. This is main memory.

pipeline. An overlapping operating cycle function that is used to increase the speed of computers. Pipelining provides a means by which multiple operations occur concurrently by beginning one instruction sequence before another has completed.

porting. Moving software from one type of machine to another.

priority. An ordering of events. May be applied to **protection** levels as well as to I/O interrupt levels.

privileged instruction. An **instruction** used by the operating system or privileged systems programs. It must execute in ring0, or an **exception** occurs.

process. A process is the fundamental unit of a program that is managed by the **job scheduler**.

process exception. Belongs to the currently running process and may be handled with an **exception** handler in that process. The exception handler is in the current **ring** of execution.

processor cabinet. A cabinet designed to hold a **Central Processing Unit** (CPU). On CONVEX equipment, a processor cabinet also houses AC and DC electrical devices, a **System Monitor Board** (SMB) (on C100 Series models), and a **System Control Module** (SCM) (on C200 Series models).

processor status word. A Processor Status Word is a word that contains control **flags** used to control and indicate the states of various computations and sequences within the processor.

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 that ensures that one user is protected from another user or to ensure that a user does not perform an unsafe computation.

PSW. *See* **processor status word.**

PTE. *See* **page table entry.**

push. The act of storing an **operand** on the **stack.**

queue. A data structure in which entries are made at one end and deletions at the other. Often referred to as first-in, first-out (FIFO).

quotient. The result of a division operation.

read. A memory operation in which the contents of a memory location are accessed and passed to another part of the machine.

recursion. Continued repetition of the same operation or group of operations.

reduced instruction set computer. Abbreviated RISC. This is 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.

reduction. An arithmetic operation that performs a transformation on an **array** to produce a **scalar** result.

register. A hardware entity used to contain addresses, operands, and status.

reservation. The process of managing the various function units in the CPU. A reservation table is used to record the current status and availability of the function units.

reset. The process of establishing a known state in a machine register.

reset switch. On CONVEX supercomputers, a manually operated switch used to force a hardware reset on the **Service Processor Unit** (SPU).

rings. A ring is the unit of logical memory used for **protection** purposes. There are five rings in CONVEX machines: four for system level usage and one for users. Each system ring (Ring0-Ring3) corresponds to one segment of logical memory, while the user ring (Ring4) contains four segments. User rings are Ring4–Ring7, collectively called Ring4.

ring maximization. The mechanism used to enforce protection in the logical **address space**.

RISC. *See* **reduced instruction set computer**.

root directory. The base directory in UNIX from which all other directories stem, directly or indirectly.

round bit. One of the two **guard bits** used in the intermediate representation of a **floating point** number.

rounding. The process of transforming the intermediate representation of a **floating point** number to the memory representation. **Unbiased rounding** uses the round, guard, and **sticky bits** to determine the exact nature of this transformation. Truncation (as used in converting floating point to fixed point integer) does not use the round, guard, or sticky bits.

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.

scatter. Storing a **vector** register using another vector of indices. See the *stvi* instruction.

SCM. *See* **system control module**.

SDR. *See* **segment descriptor register**.

segment. The basic partition of the logical memory space, equal to 512 Mbytes.

segment descriptor register. Abbreviated SDR. Each segment of **virtual memory** has an SDR associated with it. Each SDR contains information pertinent to the access and mapping of virtual addresses.

segmented ALU. A logic design technique that permits multiple arithmetic operations of the same type to be **pipelined**.

shift. A class of **instructions** used to shift the contents of a register right or left.

single. Abbreviated s. A single-precision **floating point** number stored in 32 bits. *See also* **double**.

single-user mode. In CONVEX UNIX, the mode of operation where the supercomputer is being controlled by a single system manager or operator. This mode is used primarily for maintenance and system administrative functions. *See also* **multi-user mode**.

SMB. *See* **system monitor board**.

soft front panel. **EPROM**-based software that controls certain **booting**, internal testing, and communications functions in CONVEX supercomputers.

software device driver. A CONVEX-supplied or user-written program that controls the operation of attached I/O peripheral devices.

source. A **register** or memory location used as an input to a CONVEX **instruction**.

spatial reference. An attribute of a memory reference pattern that pertains to the likelihood of a subsequent memory reference address being numerically close to a previous address.

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 UNIX. The CONVEX-developed, UNIX-based software used to direct certain supervisory functions on CONVEX supercomputers.

stack. A data structure in which the last item entered is the first to be removed. Also referred to as last-in, first-out (LIFO). In particular, stacks are used by the *call* and *return* instructions.

sticky bit. A **bit** used in the intermediate calculation of a **floating point** operand. The sticky bit remembers whether any binary 1's are shifted out during an alignment or partial product operation.

store. An **instruction** used to move the contents of **registers** to memory.

subroutine. A frequently used software module that is called from various places in a program.

superuser. The UNIX term for the **system manager**.

supervisor. Processor access mode 2.

system console. The CRT or printer/terminal that serves as a communication device between the **system manager** and CONVEX supercomputers.

system control module. The System Control Module (SCM) is an electronic safety mechanism that monitors hardware and environmental conditions on CONVEX C200 Series supercomputers. When an error condition is detected, the SCM transmits a hexadecimal status code to the system status display on the processor cabinet front panel.

system exceptions. Cannot be handled by the current process. They require intervention by the **kernel** executing in ring0. *See also* **exception**.

system manager. The person(s) responsible for the management and operation of a CONVEX supercomputer.

system monitor board. The System Monitor Board (SMB) is an electronic safety mechanism that monitors hardware and environmental conditions on CONVEX C120 supercomputers. When an error condition is detected, one of the 16 LED indicators on the SMB is lit.

system status display. A two-digit LED display located on the front panel of CONVEX C200 Series supercomputers. It is used to display hexadecimal status codes transmitted by the SCM.

tag. Marker or label.

trace of instruction execution. The process of tracking the execution of every **instruction** of a program.

trap. An out-of-sequence branch due to the occurrence of an abnormal condition. Typically, this condition is a result of unexpected arithmetic results. *See also* **exception**.

Trojan horse pointer. An address that is passed from one ring to another as part of a system call. In particular, this passed pointer references the more privileged ring as contrasted to the less privileged ring. This is unexpected and undesirable.

true zero. A **floating point** number with the sign bit with a value of zero, the exponent with a value of zero, and the **fraction** with a value of zero.

unbiased rounding. The process of interpreting the **round**, **guard**, and **sticky bits**. Unbiased rounding, as contrasted to biased rounding, rounds to even in the event that the intermediate floating point result is exactly midway between two floating point representations.

UNIX. An operating system developed by AT&T Laboratories.

unsigned. A value that is always positive.

user. Processor access mode 4.

valid bit. Used for the control of **caches**. The valid bit is used to determine if a cache entry contains an entry that can be used.

valid reference. A valid reference meets two requirements: first, the **PTE** must be valid (bit <31>=1), and second, the type of access being made (read, write, or execute) must be allowed by the appropriate **protection** bits (bits <3..1> of the PTE).

vector. An **array** with one dimension.

virtual address space. *See* **logical address space**.

virtual memory. *See* **logical memory**.

VMEbus. The most popular 16-/32-bit **backplane** bus.

word. Four bytes (32 bits), the fundamental width of items in the CONVEX family of computers. *See also* **halfword**; **longword**.

working set. That portion of a user program currently in **physical memory**. Typically, the working set is much smaller than the user program.

write. A memory operation in which a memory location is updated with new data.

zero. In **floating point** number representations, zero is represented by the sign bit with a value of zero and the exponent with a value of zero.

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix B
Reporting Problems

B.1 Overview

This appendix introduces the CONVEX Technical Assistance Center (TAC) and the *contact* utility. The *contact* utility is an online system for reporting problems to the TAC. To learn *contact* by using it, enter **contact** at the system prompt and then answer the questions as they appear on the screen. To find out more about using *contact*, read through this appendix. It describes prerequisites and tips for using *contact* and the step-by-step process *contact* takes you through.

B.2 Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address the diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation problem, contact the TAC. This group stands ready to solve such problems.

B.3 The *contact* Utility

The TAC recommends using the *contact* utility to report a hardware, software, or documentation problem. The *contact* utility is an interactive utility that helps the TAC track reports and route them to the the CONVEX personnel most qualified to fix them.

After invoking *contact*, it prompts for information about the problem. When you finish your report, *contact* electronically mails it to the TAC. You are notified within 48 hours that the TAC has received your report.

B.4 Prerequisites

To use *contact* requires

- a UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- the full path name of the program or utility in question
- the version number of the program or utility in question

B.4.1 UUCP Connection

Before using *contact*, check with your system administrator to be sure there is a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX system to another. The *uucp* (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

B.4.2 Finding the Program Path Name

To determine the full path name of the program or utility in question, use the *which* command. The following screen illustrates using the *which* command to find the full path name of the loader (*ld*) utility:

```
>which ld
/bin/ld
>
```

In this example, the full path name of the loader is */bin/ld*.

For more information on the *which* command, refer to the *which*(1) man page. You can also use the *info* online information system. Enter **info which** at the system prompt. If you use the C shell (*csh*), you can also use the *whence* command to find the program path name. The *whence* command works like *which*, only faster.

B.4.3 Finding the Program Version Number

To determine the version number of the program or utility in question, use the *vers* command. The following screen illustrates using the *vers* command (enter **vers**, then the path name of the program or utility) to find the version number of the loader (*ld*) utility.

```
>vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader utility version number is 7.0.

For more information on the *vers* command, refer to the *vers*(1) man page. You can also use the *info* online information system. To do so, enter **info vers** at the system prompt.

B.5 Tips on Using the *contact* Utility

The *contact* utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to

- use a *.contact* file
- abort a contact session
- resubmit an aborted report
- suspend a contact session
- move from one prompt to another
- use tilde-escape sequences in the *contact* utility

B.5.1 Using a *.contact* File

When invoked, *contact* prompts for information regarding the problem. The first prompt is for your name, title, phone number, and company name. You can, however, create a *.contact* file to skip this first prompt. Follow these steps:

1. Create a *.contact* file in your home directory.
2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke *contact*, it automatically includes the *.contact* file as input for the first prompt and proceeds to the next prompt.

B.5.2 Aborting the Report

To abort a contact report, either enter the interrupt key (usually CTRL-C) or choose the abort option when prompted by the *contact* utility. Using CTRL-C to abort does not save the contents of the report. Using the abort option saves the contents of the report in a file named *dead.report* in your home directory.

B.5.3 Submitting the *dead.report* File

When aborting a contact session, the *contact* utility saves the report in a file named *dead.report* in your home directory. Using the *contact* command with the *-r* option automatically merges the contents of the *dead.report* file into the new contact session. Enter

contact -r

and *contact* finds the *dead.report* file in your home directory and merges it into the contact report. You can then edit the report. When you end the editing session, *contact* returns to the final prompt, which asks you to review, edit, submit, or abort the report.

B.5.4 Suspending a Report

Sometimes it is necessary to stop in the middle of a contact report and return to the shell (for instance, to suspend the contact session to find the program path name or version number). To suspend the contact session, press CTRL-Z. To return to the contact session, enter **fg**. Using CTRL-Z and the *fg* (foreground) command lets you switch back and forth between the *contact* utility and the shell. You cannot, however, use CTRL-Z and *fg* to switch back and forth if you are using a Bourne shell (*sh*).

B.5.5 Ending a Response

The *contact* utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press RETURN. Other prompts require more than a one-line response; to move to the next prompt, press CTRL-D.

B.5.6 Tilde-Escape Sequences

The *contact* utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. The following tilde sequences are recognized by *contact*:

~e	Start the text editor (defined in your EDITOR environment variable).
~h	Display a list of available tilde-escape sequences.
~p	Print the contact report to the terminal screen.
~r *filename*	Read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence only works for prompts that allow more a than one-line response.
~~	Insert a single tilde as the first character in the line.

B.6 Using the *contact* Utility

The *contact* utility prompts for the following information:

- your name, title, phone number, and corporate name
- the name and version of the product involved
- a one-line summary of the problem
- a detailed description of the problem
- the priority of the problem
- instructions on how to reproduce the problem
- comments about the problem
- comments about the documentation supporting the problem
- files to include in the contact report

The following is a step-by-step discussion of these prompts:

1a. To invoke the *contact* utility, enter **contact** at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software, or documentation question. The following screen illustrates the *contact* command and the system response:

```
>contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

1b. If there is a *.contact* file in your home directory, *contact* skips the first prompt. The following screen illustrates the *contact* command and the system response when a *.contact* file is in your home directory:

```
>contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

2. The *contact* utility prompts for the version number of the product. If you do not know the version number, use CTRL-Z to suspend the session. Use the *which* (or *whence* if using *csh*) and *vers* commands to find the version number of the product. Use the *fg* command to return to the session and enter the version number in the form X.X or X.X.X.X.
3. The *contact* utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Make this summary as descriptive as possible in one line.
4. The *contact* utility prompts for a detailed description of the problem. Make this description as complete as possible. Include source code and a stack backtrace whenever possible. (Refer to the *adb*(1) or *csd*(1) man page for information on obtaining a stack backtrace.) The more information provided, the quicker the TAC can isolate and solve the problem.
5. The *contact* utility prompts for the priority of the problem. The following screen illustrates this prompt and the priority levels from which to choose; you must enter a priority number.

```
Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

6. The *contact* utility prompts for an explanation of how to reproduce the problem. Include the command syntax and options you used and anything else you did to make your program run.
7. The *contact* utility prompts for any other pertinent comments. Include any relevant information.
8. The *contact* utility prompts for suggestions regarding the documentation supporting the product. Indicate if the documentation could be revised to address the question.
9. The *contact* utility asks for the names of files necessary to reproduce the problem. The following screen illustrates the *contact* prompt and sample user response:

```
Are there any files that should be included in this report (yes | no)?
>yes
Please enter the names of the files, one to a line (^D to terminate)
>test.f
>~/subroutines/sub.f
>
```

> **NOTE**
>
> Tilde-escape sequences are not recognized in responses to this prompt. Instead, *contact* treats a tilde in this section to mean your home directory. This convention is based on use of the tilde for expanding file names in *csh*.

If the files specified are small text files, they are automatically included in the contact report. If the files are too big to be included in this report, *contact* gives further instructions on how to submit these files.

To specify a directory, combine the directory files into a single file using the *tar* command (refer to the *tar*(1) man page for further information) or enter each file name in the directory on a single line in the contact report.

10. The *contact* utility prompts you to review, edit, submit, or abort the contact report. The following screen illustrates this prompt:

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
>
```

Choose the number of the option you want to select. These options let you do the following:

Review	Review the text of your contact report. You are then prompted again to select an option.
Edit	Edit the text of the contact report. If you choose to edit the report, *contact* puts you in your default text editor.
Submit	Send the report to the CONVEX TAC. You are notified within 48 hours that the TAC has received the report. This option exits the *contact* utility and returns you to the shell environment.
Abort	Save the text of your report in a file named *dead.report* in your home directory. This option exits the *contact* utility and returns you to the shell environment.

Index

Symbol

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Electronic Mail

Using Electronic Mail

The Hardware Documentation Group has an email address for documentation comments. Use this service to give us a quick response mechanism if you have special documentation questions that you would like addressed immediately. If you have a technical question, you should still contact the Technical Assistance Center, as described previously. To use email response service, just send mail addressed to:

```
cnvxhwdoc@convex.COM
```

We will read your comments and give you a personal reply.

What to Include in an Email Message

When you use 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
- The comment

Reader's Forum

If you wish to mail your comments to us, please use the form on the next page and list the document page number with your questions and comments. Thank you.

THIS PAGE INTENTIONALLY LEFT BLANK

CONVEX Diagnostic Utilities Manual
(C200 Series)
Document No. 760-000830-000, Third Edition

Reader's Forum

Please use this form to submit comments or questions concerning the clarity and service of this manual. Constructive critical comments are most welcome and help us continue in our efforts to generate quality customer documentation. Please list the page number for questions or comments.

_____

_____

_____

_____

_____

_____

_____

_____

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION OR DOCUMENTATION:

Location	Phone Number
From all locations in continental U.S.	1(800)952-0379
From locations in Alaska, Hawaii, & Canada	1(214)497-4379
From all other locations	Contact nearest CONVEX office

Direct mail orders to:

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

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

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

(Fold Here Second)

(Tape or Staple)