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.

new for C2

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 [**-cfmv**]

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 board combinations to ensure compatibility. Warning or fatal error messages will be printed if an illegal combination of boards 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.

-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
mem	All memory slots
meme	All even memory slots
memo	All odd memory slots
mem0	All memory 0 slots
mem1	All memory 1 slots
mem2	All memory 2 slots
mem3	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

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.

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

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.

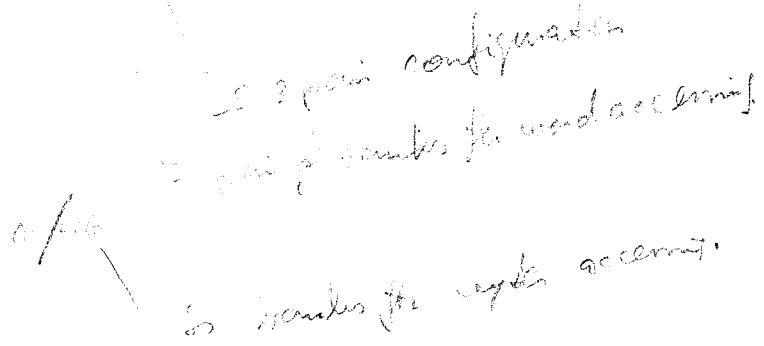
"**Invalid process exception code XXXX**" - The CPU took a process exception.

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

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:

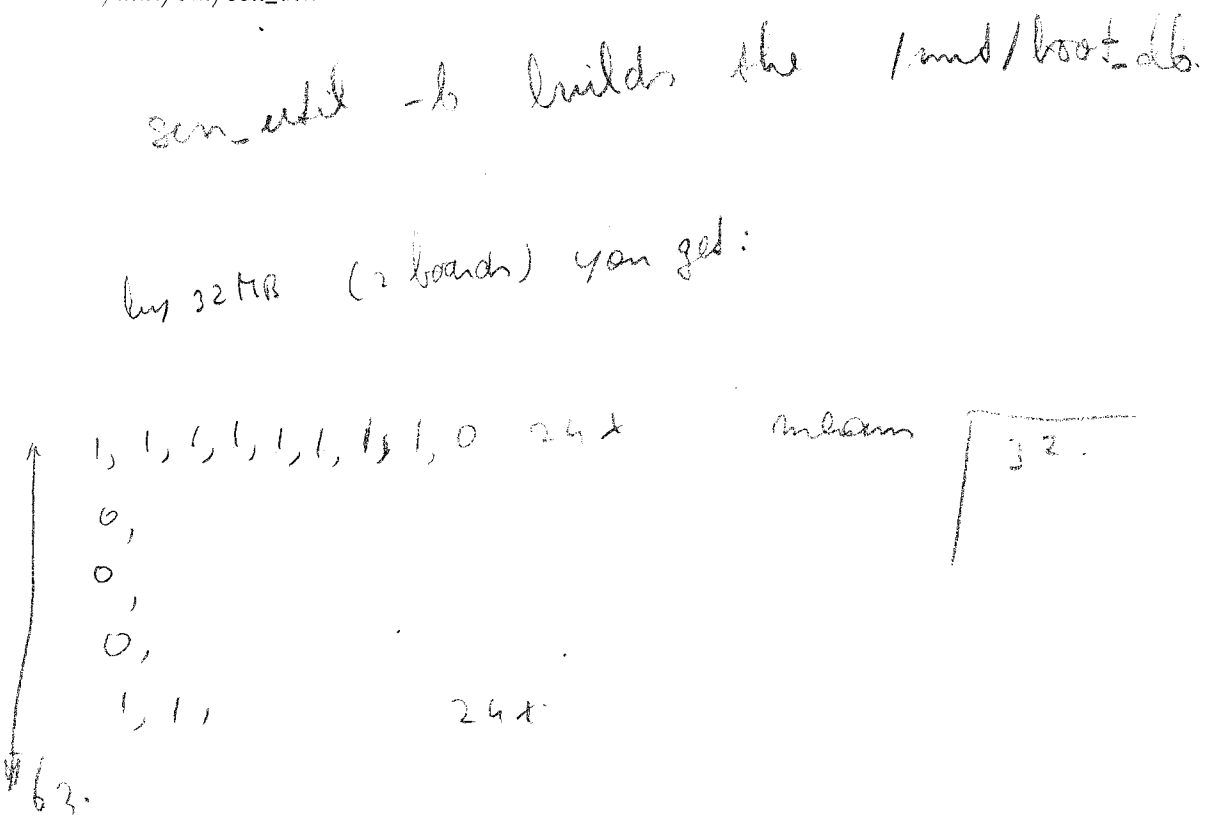
CPUTYPE	CPU instruction architecture: 1=C1, 2=C210a, 3=C2xx
MEMSTART	Starting physical address of memory
IOPS	Location of IOPs in CCU slots
VIOPS	Location of VIOPs in CCU slots
IOP_ACCEL	Location of IOPs with accelerate enable option
HSPS	Location of HSPs in CCU slots
IEEE	Does machine support IEEE mode 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 ffffffffffffffff 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

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>.

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 the number of the memory board containing the faulty device.

device
is the coordinates of the faulted device, coded as follows:

PP-SCCCRR or UKKKWW-UNN

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
Problem Reporting

B.1 Overview

The *contact* utility is the recommended way to report minor hardware deficiencies and technical documentation problems to the Technical Assistance Center (TAC). This utility is an interactive tool that prompts the user for the information to properly file a problem report.

> **NOTE**
>
> The *contact* utility is not intended for requesting customer service for hardware failures. To restore your CONVEX equipment to operational status, faster service can be obtained by directly telephoning the TAC (refer to "Technical Assistance" in the Preface).

To use the *contact* utility, there must be a phone connection to the TAC. UNIX-to-UNIX Communication Protocols (UUCP) allow communication between UNIX systems by either dial-in or hard-wired communication lines. For more information, refer to *uucp(1)* or to the *info(1)* entry in the UNIX man pages.

The name and version number of the product involved is required. Use the *vers* command to ascertain the program or utility name and version. The syntax for the command is **vers** *filename*, where *filename* is the full pathname of the program. If the full pathname of the program is not known, enter **which** *program*. For more information, refer to the *vers(1)* and *which(1)* entries in the UNIX man pages.

B.2 Information Required to Report a Problem

The *contact* utility requires the following information:

1. The customer name, title, phone number, and corporate name
2. The hardware nomenclature, part number, and revision level, or the technical manual name, document number, and version

> **NOTE**
>
> Use *vers* and *which* to identify product name and version.

3. A short (one line) summary of the problem

4. The more information provided, the more quickly the problem can be isolated and solved. At a minimum, include a detailed description of the problem (including page references, if applicable), the source code, and a stack backtrace whenever possible.

> **NOTE**
>
> See the *adb(1)* or *csd(1)* man pages for information on obtaining stack backtraces.

5. The priority of the problem, selected from a list of six levels
6. Instructions on how to reproduce the problem, including the command syntax used, any flags invoked, or anything else attempted to make the program run
7. Any other comments about the problem or files to be submitted

The *contact* user has a chance to review and edit the report prior to submitting it. If the user decides to delay submitting the report, the session can be aborted. The report is automatically saved in the user's top-level directory in a file named *dead.report*.

See the following figure for a sample *contact* session. User input is in bold lettering, and the system response is in monospace type.

Figure B–1, Sample *contact* Session

```
%contact (RETURN)
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
> Margaret Atwood, systems programmer, 814-4444, University r
> of Chicago (RETURN)
> (CTRL-D)

Enter the name of the product involved
> CONVEX UNIX Programmer's Manual, Part I (RETURN)

Enter the version number (in the form X.X or X.X.X.X) of the product
> Revision 4.0 (RETURN)

Enter a short (1 line) summary of the problem
> The finger command manual page lists nonexistent bug (RETURN)

Enter a detailed description of the problem (^D to terminate)
> The finger(1) man page says, under the BUGS section, that "Only the first
line of the .project file is printed." Happily, this is not true! (RETURN)
> (CTRL-D)

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.
> 4 (RETURN)

Enter the instructions by which the problem may be reproduced (^D to terminate)
> a) put more than one line in .project (RETURN)
> b) read the man page for finger(1) (RETURN)
> (CTRL-D)

Enter any comments that are applicable (^D to terminate) (RETURN)
> (CTRL-D)

Do you have any suggestions or comments on the documentation that you
referenced when you were trying to resolve your problem (for example,
additions, corrections organization, accessibility)? (^D to terminate)
> The man page should be updated. (RETURN)
> (CTRL-D)

Are there any files that should be included in this report (yes | no)?
> no (RETURN)

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.
> 3 (RETURN)

Problem report submitted.
%
```

THIS PAGE INTENTIONALLY LEFT BLANK

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

CONVEX Diagnostic Utilities Manual
(C200 Series)
Document No. 760-000650-202, Second Edition

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. 760-000650-202, Second Edition

Reader's Forum

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION OR DOCUMENTATION:

Location	Phone Number
In Texas	(214)952-0200
Other continental locations	1(800)952-0379
Outside continental US	Contact local 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

CUSTOMER SERVICE
CONVEX Computer Corp.
P.O. Box 833851
Richardson, TX 75083-3851

(Fold Here Second)

(Tape or Staple)