

SPP1000/1200 Trouble Shooting Hints

Revision 5.0 Date: 10 March 1995

Compiled by TAC Hardware Support

Thanks to the folks in Product Engineering,
Manufacturing and the Field.

Additional Input Requested
e-mail tachwsup

CONVEX PRIVATE

Table of Contents

An * Indicates a New or Updated Article

Chapter	Section	Page	
Chapter 1	Test Station		
1.1	Installing/Updating Test Station Software	1-1	*
1.2	No Communication with Node	1-1	*
1.3	Selecting a Node	1-1	
1.4	Test Station dump/fbackup/frestore	1-2	*
1.5	Remote control of the SPP-UX Console	1-3	*
1.6	Adding a Printer to the Test Station	1-3	
Chapter 2	Power		
2.1	pce_util	2-1	
2.2	Power Brick Torque	2-1	
2.3	Power Brick Compatibility	2-1	
2.4	ASB/CSB Power Test Points	2-1	*
2.5	Voltages and Bricks	2-2	*
2.6	48 Volt Power Supply Adjustment	2-3	
2.7	Turning Off 48 Volts While Trouble-Shooting	2-3	*
Chapter 3	Configuring Hardware		
3.1	CCMU	3-1	
3.2	Configuration Parameters	3-4	*
3.3	xconfig	3-9	
3.4	I/O Chassis Address Assignments	3-13	*
3.5	System Complex Serial Number Assignments	3-13	*
3.6	Saving MU NVRAM Before Trouble-Shooting.....	3-14	*
3.7	Diff/SE SCSI Converter 220-000045-200 (Ancot)	3-14	
3.8	Modem Parameters (USA)	3-15	*
3.9	De-configuring Memory with "ccmu"	3-16	
3.10	SPP1000/CD Disk Limits	3-17	
3.11	CCMC2 and PITA Card	3-17	
3.12	Disk Drive Jumpers DEC DSP3210W 2 GB SCSI 204-000028-200	3-18	*
3.13	Disk Drive Jumpers Seagate ST15150W 4GB 204-000033-200	3-20	*
3.14	Conner DDS-2 DAT Drive 207-000031-200	3-21	*
3.15	Ergo Ultra VGA Display	3-21	*
3.16	MTV SIMM Locations and Sizes	3-21	*
Chapter 4	Diagnostics		
4.1	MU Firmware Power-Up Tests	4-1	*
4.2	Preparing a SPP Node for Diagnostics	4-3	*
4.3	CST	4-4	
4.4	Known CST Failures	4-5	
4.5	sppring	4-5	*
4.6	sppring quick pass	4-12	*
4.7	cputest	4-12	
4.8	Margins	4-12	
4.9	iod I/O Debugger	4-13	*
4.10	iod test scripts	4-14	*
4.11	iod p3_disk_* Alternate Unit Selection	4-15	*

Table of Contents

Chapter 4	Diagnostics (continued)	
4.12	spring failures kill config	4-16 *
4.13	CST Rev 0.15f and CPBs Error Message	4-16
4.14	Determining Utility/Diag/Firmware Versions	4-16 *
4.15	verify_cables and verify_rings	4-17 *
Chapter 5	Trouble Shooting	
5.1	hard_logger	5-1
5.2	MU Registers	5-1 *
5.3	MAUI Registers	5-16
5.4	MU Status Numbers	5-22 *
5.5	Disabling Nodes and Slices when Trouble-Shooting	5-27 *
5.6	Single-Node False "SCI incoming parity err"	5-27 *
5.7	Identifying Gate Array Locations	5-28 *
5.8	Node Temperature Monitoring	5-28 *
Chapter 6	OBP (Open Boot Prom)	
6.1	OBP (Open Boot Prom) What is it?	6-1
6.2	OBP Configuration Parameters	6-3 *
6.3	Configuring Non-Standard or CD Hypernodes	6-10 *

1.1 **Installing/Updating Test Station Software

1. Shutdown SPP-UX on all nodes.
/etc/shutdown now
2. As root bring the test station to single user mode.
/etc/shutdown now
3. Insert the Exemplar Software DAT cartridge into the DAT drive on the test station.
4. cd /
5. If /etc/installsw IS on the test station go to step 6.

If /etc/installsw IS NOT on the test station then;
tar -xv INSTALLSW (this will take about 20 minutes)
mv INSTALLSW/INSTALLSW/etc/installsw /etc
rm -rf INSTALLSW
6. Use the installsw program to install the test station software products.
/etc/installsw -n

installsw displays a list of products on the DAT tape.
Select Cst, CXTS, Diag, SPP-UX_ts and OBP (as available).
Press <cr>.
This takes approximately 45 minutes to complete.
7. Reboot the test station.
/etc/reboot
8. Login to the test station as sppuser.
9. Use node_info to determine the nodes present.
10. For each node present:
load_pl <node#>
load_mufw <node#>
11. If an OBP was extracted from the tape, for each node present,
load_obp <node#>. Additional OBP configuration may be required.
Refer to the OBP release notes.
12. Reset the system complex.
pce_util -p off
do_reset 1
13. Installation is complete. The SPP system complex may be rebooted at this time.

1.2 **No Communication with Node

If you loose communication between the test station and the node verify the following:

1. Verify the node is powered and the DaRT bus is connected.
(continued on next page)

1.2 No Communication with Node(continued)

2. ping mu_000X. Where X = 0-f = node you want to ping.
If ping is not successful go to step 3.
If ping is successful go to step 5.
3. On the test station do a netstat -i.
Does the output show lan0 up and sending/receiving packets?
If yes go to step 5.
If not, verify the entries in /etc/netlinkrc and /etc/hosts
are correct. Correct any wrong entries and go to step 4.

Example netstat -i

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Coll
ni0*	0	none	none	0	0	0	0	0
ni1*	0	none	none	0	0	0	0	0
lo0	4608	loopback	localhost	3487568	0	3487568	0	0
lan0	1500	130.169	whoosh_d	1178622	0	1432914	2645	42324
lan1	1500	130.168.77	whoosh_t	2485821	659	490530	698	33273

4. su root
landiag
lan
reset
quit
/etc/ifconfig lan0 inet 'hostname' netmask 255.255.0.0 up
ping mu_000X
If ping is successful go to step 5
If ping is not successful contact TAC for further assistance.
5. su root (If not already root).
cd /spp/bin
./fix_hung_inetd
\004
6. Communication with the node should be established at this time.
If you still have no communication with the node contact TAC.

1.3 **Selecting a Node

When you login as sppuser or start a "sppdsh", Node 0 is selected automatically. Type node # to select a different Node.
Where # = node you want.

1.4 **Test Station dump/fbackup/frestore

- A. Level 0 Dump
 1. Become root.
 2. cd /
 3. /etc/dump 0df 32000 /dev/rmt/0m /
- B. Bootable Recovery Tape
 1. Become root.
 2. /etc/mkrs -q -v -f /dev/rmt/3m -r /dev/rdisk/c201d6s0 -s -m 700
- C. fbackup
 1. Become root.
 2. /etc/fbackup -f /dev/rmt/3m -0 -i /

(continued on next page)

1.4 **Test Station dump/fbackup/frestore (continued)

D. frecover

1. Become root.
2. /etc/frecover -f /dev/rmt/3m -i /

1.5 **Remote control of the SPP-UX Console

1. rlogin to the test station.
2. sn_cnsl -F 1
This will take control of the SPP-UX Console window and keyboard input to that window.

NOTE: If the test station 'hostname' is not equal to the DaRT 'hostname', the "sn_cnsl" command will fail.

3. To disconnect type "^ec." <cntl>ec.

1.6 **Adding a Printer to the Test Station

There is not currently a way to automagically print the SPP-UX console.

Follow these steps to enable a local printer attached to the test station:

On the test station "su" to "root".

Run "SAM".

Select "Peripheral Devices"

Select "Printers and Plotters".

Under "Actions" select "Add Local Printer/Plotter-Add Parallel Printer".

Under "Available Parallel Cards", "2.0.6" should be highlighted.

Select the "OK" button.

In the next screen that appears:

Enter a printer name. <printername>

In the "Printer Model/Interface" select "dumb".

In the "Printer Device File Name" enter /dev/<printername>.

Select the "Make This System Default" button.

Select "OK"

SAM will go do some stuff and then tell you it is complete.

Exit SAM.

Remove the "setenv PRINTER swip" in sppuser's .profile and .cshrc.

To print enter "lp <filename>"

2.1 **pce_util

pce_util is the SPP Teststation Power, Clock & Environment Utility.

pce_util can be used to:

- Power the bricks a node on or off
- Margin the node clocks
- Monitor environmental conditions (Temperature and Fan Failure)

pce_util can not be used to:

- Power off the MU
- Remove 48 volts from the node
- Remove 48 volts from the I/O chassis (power off disks)

Please read the pce_util man page for a complete explanation.

2.2 **Power Brick Torque

Power bricks should be torqued to 5 in/lbs. Over torquing can cause damage to the bricks.

2.3 **Power Brick Compatibility

There are 2 different sets of 1.2 volt bricks.
 The 200-001047-200 and 200-001047-201 are used together.
 The 200-001049-200 and 200-001049-201 are used together.
 Do not mix these. The 200-001047-20X bricks are no longer in stock.
 If you have a failure on a 200-001047-20X brick, you will have to replace all 4 1.2 volt bricks with 200-001049-20X bricks.
 There is 1 driver and 3 boosters on each csb.
 The -200 bricks are drivers and the -201 bricks are boosters.

2.4 **ASB/CSB Power Test Points

 POWER TEST POINTS / ASB (SPP1200)

Z212E4					Z013L2								
VDDSCI	--	1	0	0 6	--	VPS	VEE	--	1	0	0 6	--	VTT
VTTSCIIO	--	2	0	0 7	--	VCCMU	VDDGA	--	2	0	0 7	--	VCCPR
VTTSCI	--	3	0	0 8	--	VPR	GND	--	3	0	0 8	--	VDL
GNDSCI	--	4	0	0 9	--	GND	VHC	--	4	0	0 9	--	GND
GND	--	5	0	0 10	--	GND	VDH	--	5	0	0 10	--	GND

(continued next page)

2.4 ASB/CSB Power Test Points (continued)

POWER TEST POINTS / CSB3 / CSB4 (SPP1000)

Z209E4					Z010L2						
VTTGA	-- 1	0	0	6	-- VEE	VPS	-- 1	0	0	6	-- VTT
VDLSCI	-- 2	0	0	7	-- VCCMU	VPR	-- 2	0	0	7	-- VCCPR
VTTSCI	-- 3	0	0	8	-- VDDGA	GND	-- 3	0	0	8	-- VDL
GNDSCI	-- 4	0	0	9	--	VHC	-- 4	0	0	9	--
VCCMB	-- 5	0	0	10	--	VHE	-- 5	0	0	10	--

2.5 **Voltages and Bricks

ASB (SPP1200 System Board)

VDD_GA	(ASB)	+1.2V	Fujitsu Gate Arrays (array power) (4 large bricks on ASB).
VTT	(ASB)	-2V	Fujitsu Gate Arrays (ECL I/O power + Terminators), (1 large brick on ASB).
VDD_SCI	(ASB)	+1.2V	SCI Gate Arrays core voltage. (2 large bricks on ASB).
VTT_SCI	(APB)	-2V	SCI incoming bus terminators (referenced to GNDSCI), (Avalon Power Board)
GNDSCI		~0	SCI incoming bus ground.
VCC_HP	(APB)	+5V	HP PA-RISC T-chip Module & SRAM power.
VDL	(APB)	+3.3V	HP PA-RISC T-chip Modules (I/O power) Also LANDMARC and Fujitsu gate arrays,
VHC	(APB)	+2V	HECL (GTL-like) VCC supply (ECLinPS logic, and PLL)
VEE_CLK	(APB)	-4.5V	ECLinPS Logic
VTT_SIO	(APB)	-2V	For SCI node termination referenced to ground
VDH_HP	(APB)	+4.4V	For T' Modules, main power. Actually implemented using +5V bricks.
V9NDRT		-9V	Ethernet, DART bus (small DC-DC on ASB fed with VCCMU, referenced to ethernet ground).

(continued next page)

2.7 Turning Off 48 Volts While Trouble-Shooting (continued)

Anytime you are working in the power supply section, of an XA system, you should power down the node with "pce_util -p off" and then turn off the main circuit breaker.

If you are working in the node or I/O chassis area, you may use the "key switch" or individual node or I/O chassis circuit breakers to remove 48 volts after a "pce_util -p off".

In multiple cabinet XA systems the "key switch" is cabled from cabinet to cabinet. When the "key switch" is opened (OFF) the 48 volts will be removed from all nodes in the system.

SPP1000/CD

In a CD system, there is no PCC or "key switch".

Whenever the circuit breaker is on the 48 volt supply is on.

If you are working on a CD system you should always power down the node with "pce_util -p off" and then turn off the circuit breaker.

3.1 **CCMU

ccmu is short for Convex (hardware) Configuration Management Utility.
 ccmd is short for Convex (hardware) Configuration Management Deamon.

The ccmu utility is an interactive program that manipulates the hardware configuration database for the user. The database is actually mostly generated and maintained by the ccmd deamon. The ccmu has a variety of commands dealing with all aspects of the the hardware configuration. Some commands require arguments, others don't. Some have optional arguments.

misc commands:

```

help          print the help overview
help <command> print more detailed information about <command>
quit         terminate execution
exit        terminate execution
regen       send a signal to the ccmd to regenerate the data base
unhang <#>  unhang the specified processors
debug <#>   set the debug flag to the indicated value (ccmu only)
  
```

serial/node number related:

```

assign <node #> <node serial #>
    assign the specified node ID to the node with
    the indicated node serial number.
assign <node #> <node serial #> <complex serial #> <key>
    assign the specified complex serial # to the
    node indicated by the node # and node serial #.
    <key> must be valid for this combination.
  
```

NOTE: the above two commands are differentiated by the number of arguments, and the node number being an argument. also, the specific cop chip is implied by the command

other cop functions:

```

copmod <node #> <cop id> [-c]
    enter variouse information for the
    specified cop. if -c is specifed, the cop will
    be zero'ed
cop    [<node #> [<<cop id>>]]
    display the current information for a specific cop,
    all cop's on a node, or all cops in the system
display <node #> <cop id>
    same as "cop", but both argument are required.

assign <node #> <cop id> a[] <assembly revision>
    put the specified asm rev in the specified cop, but
    also determines and sets the scan, ac, and dc revs,
    based on the current part number in the cop chip.
assign <node #> <cop id> c[] yes
    clears all cop fields except the 'misc' field, which
    is used for node # (MU) and complex sn (CSB)
assign <node #> <cop id> e[] <ecn #>
    add this ecn number to the ecn_mask field, updating
    the ecn_lvl field if necessary. a warning is displayed
    if this ecn is already set, or if updating the ecn_lvl
    field causes unset bits to be dropped.
  
```

(continued next page)

3.1 CCMU (continued)

assign <node #> <cop id> p[] <part #>
put the specified part # in the specified cop

assign <node #> <cop id> r[] <ecn #>
remove this ecn number to the ecn_mask field. the ecn_lvl field is not modified. a warning is displayed if this ecn is already clear or not present in the ecn_mask field.

assign <node #> <cop id> s[] <(part) serial #>
put the specified part serial # in the specified cop

assign <node #> <cop id> w[] <wire revision>
put the specified wire rev in the specified cop

NOTE: these commands do verify the part number is appropriate for the cop specified

NOTE: these commands manipulate both the actual COP chip and the global database

item data related:

dget <item data name>
dput <item data name> <type> <new_value(s)>
delete <item data name>

config parm operations:

NOTE: these commands default to all nodes in contact with the test station

pull [<node #> ...] tell the nodes to read ram parms from nvram

upload [<node #> ...] [parameter ...]
upload all/specified parameters from all/specified nodes to TS

get [<node #> ...] [parameter ...]
display all/specified parameters on all/specified nodes from TS copy

getstd
displays the value of the Test Station copy of a number of commonly examined parameters.
ignores any/all options.

put [<node #> ...] parameter [...] value
set one or more parameters to the indicated value (TS copy) for all/specified nodes.
NOTE: a parameter and value reqd.

unset [<node #> ...] parameter [...]
unset one or more parameters from all/specified nodes

download [<node #> ...] [parameter ...]
download all/specified parameters from TS copy to all/specified nodes

push [<node #> ...] tell the nodes to write ram parms to nvram

msize <#> on all nodes, set the USEMESIZE parameter to the correct value for this number of megabytes

mbanks <#> on all nodes, set the USEBANKFIELD parameter to this value

mnetcache <#> on all nodes, set the USECACHESIZE parameter to the correct value for this number of megabytes of

save <filename> save all local config parms to the specified file

restore <filename> restore config parms from the specified file

(continued next page)

3.1 CCMU (continued)

hardware selection stuff:

NOTE: these commands default to all nodes in contact with the test station

toggle	[<node #> ...] [thing]	toggle	all/specified things
clear	[<node #> ...] [thing]	unselect	all/specified things
select	[<node #> ...] [thing]	select	all/specified things
query	[<node #> ...] [thing]	display	all/specified things

"thing" is processed in order, left to right. multiple commands should be used to select the desired hardware to avoid confusion with the processing, and in case it ever changes in the future.

"thing" is specified by:

"N", "n"	operate on the node as a whole
"S#", "s#"	operate on these slices on the selected nodes
"M#", "m#"	operate on these memory boards on the selected nodes
"C#", "c#"	operate on these CPU modules on the selected nodes
"I", "i"	operate on the IO board on the selected nodes

NOTE: the memory size, memory banks, and cache size are config parms, and not manipulated by these commands. we may have special set commands to do this, but for now there aren't

automatic configuration:

NOTE: these commands default to all nodes in contact with the test station

auto [<node #> ...] use the predetermined rules to set the config parms in a valid configuration for the present hardware on all/specified nodes. if no memory size is currently specified, one will be defaulted to, otherwise the configuration will remain unchanged.

simulation and hwdump support:

write [file] write the hardware configuration database to the specified file
read [file] read a previously save database from the specified file

3.2 **Configuration Parameters

Here is a list of the configuration parameters and their bit definitions.

Number	Name	Description
1	INITWHAT	bit mask of initialization functions
	GO	0x00000001
	PWRCLKCHECK	0x00000002
	DORESET	0x00000004
	INITCXBAR	0x00000008
	INITSCI0	0x00000010
	INITSCI1	0x00000020
	INITSCI2	0x00000040
	INITSCI3	0x00000080
	INITSLICE0	0x00000100
	INITSLICE1	0x00000200
	INITSLICE2	0x00000400
	INITSLICE3	0x00000800
	SIZEMEM	0x00001000
	CHECKMEM	0x00002000
	INITLAND	0x00004000
	MUERRENABLE	0x00008000
	CLOCKGATE	0x00010000
	STARTPL	0x40000000
	PLSELFTEST	0x20000000
	PLCPUINIT	0x10000000
	PLINITMEM	0x08000000
	PLTESTMEM	0x04000000
	PLLOADCODE	0x02000000
	PLJUMPTOCODE	0x01000000
	PLLOADOBP	0x00800000
	PLSTATEDUMP	0x00400000
	PLGETCACHE	0x00200000
	PLERRENABLE	0x00100000
	PLINITNODE	0x00080000
2	RUNWHAT	used to determine what hardware to run
	RUN_MB3	0x80000000
	RUN_MB2	0x40000000
	RUN_MB1	0x20000000
	RUN_MB0	0x10000000
	RUN_XBAR_Q	0x02000000
	RUN_XBAR_S	0x01000000
	RUN_SCI3	0x00800000
	RUN_SCI2	0x00400000
	RUN_SCI1	0x00200000
	RUN_SCI0	0x00100000
	RUN_CCMC3	0x00080000
	RUN_CCMC2	0x00040000
	RUN_CCMC1	0x00020000
	RUN_CCMC0	0x00010000
	RUN_TCHIP0	0x00008000
	RUN_TCHIP1	0x00004000
	RUN_TCHIP2	0x00002000
	RUN_TCHIP3	0x00001000
	RUN_TCHIP4	0x00000800
	RUN_TCHIP5	0x00000400

(continued next page)

3.2 Configuration Parameters (continued)

	RUN_TCHIP6	0x00000200
	RUN_TCHIP7	0x00000100
	RUN_CPA3	0x00000080
	RUN_CPA2	0x00000040
	RUN_CPA1	0x00000020
	RUN_CPA0	0x00000010
	RUN_IO1	0x00000008
	RUN_IO0	0x00000004
	RUN_MAUI	0x00000002
	RUN_MU	0x00000001
3	NODEID	this node's id, 0-15, set by MU on powerup
4	ASSERTRESET	
	XBAR_Q	0x02000000
	XBAR_S	0x01000000
	SCI3	0x00800000
	SCI2	0x00400000
	SCI1	0x00200000
	SCI0	0x00100000
	CCMC3	0x00080000
	CCMC2	0x00040000
	CCMC1	0x00020000
	CCMC0	0x00010000
	TCHIP7	0x00008000
	TCHIP6	0x00004000
	TCHIP5	0x00002000
	TCHIP4	0x00001000
	TCHIP3	0x00000800
	TCHIP2	0x00000400
	TCHIP1	0x00000200
	TCHIP0	0x00000100
	CPA3	0x00000080
	CPA2	0x00000040
	CPA1	0x00000020
	CPA0	0x00000010
	IO1	0x00000008
	IO0	0x00000004
	MAUI	0x00000002
	MU	0x00000001
5	DEASSERTRESET	same definition as ASSERTRESET
6	CPUVISVAL	CPA AG_VIS_SEL CSR value to use
7	MSGTRANSDELAY	CPA MSG_TRANS_DELAY CSR value to use
8	HOURTIME	CPA HOURINI Time-Out CSR value to use
9-12	ACTMEMSIZEy	0-5, actual CCMC mem size, y = slice number
13-16	ACTBANKPRESy	2 bits, actual CCMC present bits, y = slice number
17	USEMEMSIZE	0-5 -> memory size field (CCMC Config CSR)
18	USEBANKFIELD	0-4 -> bank field (CCMC Config CSR) The HW_CONFIG parm is used now.
19	USECACHESIZE	0-7 -> cache field (CCMC Config CSR)
20	MEMTIME0A	memory timing parameters for slice 0.
21	MEMTIME0B	three of these parameters are loaded
22	MEMTIME0C	into the CCMC during slice initialization.
23	MEMTIME0D	the fourth remains unused.
24-27	MEMTIME1w	memory timing parameters for slice 1

(continued next page)

3.2 Configuration Parameters (continued)

28-31	MEMTIME2w	memory timing parameters for slice 2
32-35	MEMTIME3w	memory timing parameters for slice 3
the location of the user code, to be copied by the MU to main memory		
36	USERSIZE	# bytes to copy
37	USERADDRMU	MU starting address (32 bit address)
38	USERADDRMAIN	system starting address (32 bit address)
39	MEMTESTCTRL	control information for the memory test
	MEM_CYC	0x000100
	TAG_NTA	0x000080
	TAG_MATS	0x000040
	TAG_UNIQUE	0x000020
	TAG_COLUMN	0x000010
	MEM_NTA	0x000008
	MEM_MATS	0x000004
	MEM_UNIQUE	0x000002
	MEM_COLUMN	0x000001
40-47	STARTADDRx	starting address for each cpu. x = cpu, (32 bit address)
48-55	STATUSx	status & error values, one per processor x = cpu, + = status, - = error
56-63	BUFSIZEx	size of the user defined buffer, in bytes x = cpu
64-71	BUFADDRx	starting address of user defined buffer x = cpu, (32 bit address)
72-79	HWDUMPCTRLx	starting address of state dump control info x = cpu, (32 bit address)
		bits for HWDUMPCTRLx parms:
	DUMP_ICACHE	0x01 dump the instruction cache
	DUMP_DCACHE	0x02 dump the data cache
	DUMP_REST	0x04 dump everything else
80-87	TRACEx	trace parms, x = cpu
89-96	MEM_START_ADDRx	starting addr for memory test, x = cpu
97-104	MEM_END_ADDRx	ending addr for memory test, x = cpu
105-112	MEM_ADDR_MASKx	addr mask for uniq, mats, and nta, x = cpu
113-120	MEM_DATA0_0_x	test data 0 lower 32 bits for mats and nta, x = cpu
121-128	MEM_DATA0_1_x	test data 0 upper 32 bits for mats and nta, x = cpu
129-136	MEM_DATA1_0_x	test data 1 lower 32 bits for mats and nta, x = cpu
137-144	MEM_DATA1_1_x	test data 1 upper 32 bits for mats and nta, x = cpu
145-152	MEM_OUTPARAM0_x	output parameter 0, x = cpu
153-160	MEM_OUTPARAM1_x	output parameter 1, x = cpu
161-168	MEM_OUTPARAM2_x	output parameter 2, x = cpu
169-176	MEM_OUTPARAM3_x	output parameter 3, x = cpu
177-184	OS_CONFIG0_x	OS boot mode - gr26, x = cpu
185-192	OS_CONFIG1_x	OS initial memory block (number bytes) - gr25
193-200	SN_BUFFERx	????, x = cpu
201-208	RD_BUFFERx	????, x = cpu
209-216	NOTIF_ADDRx	????, x = cpu

(continued next page)

3.2 Configuration Parameters (continued)

217	CHANNEL_ID	how a T-chips talks to the test station
218	TRACEMU	trace value for the MU's use
219	STATUSMU	status value for the MU's use
220	APPLY_POWER	a mask so the MU know what to turn on
	MU_PWR_IO	0x4000 (if board installed)
	MU_PWR_VEE_CLK	0x2000 (always)
	MU_PWR_VHE_HP	0x1000 (always)
	MU_PWR_VHC_HP	0x0800 (if any T-chip installed)
	MU_PWR_VCC_HP	0x0400 (if any T-chip installed)
	MU_PWR_VCC_MB3	0x0200 (if board installed)
	MU_PWR_VCC_MB2	0x0100 (if board installed)
	MU_PWR_VCC_MB1	0x0080 (if board installed)
	MU_PWR_VCC_MB0	0x0040 (if board installed)
	MU_PWR_VTT_SCI	0x0020 (never used for CSB3)
	MU_PWR_VTT_GA	0x0010 (never used for CSB3)
	MU_PWR_VDL_SCI	0x0008 (never used for CSB3)
	MU_PWR_VTT	0x0004 (always)
	MU_PWR_VDL	0x0002 (always)
	MU_PWR_VDD_GA	0x0001 (always)
221	AVAIL_HW	a mask for available hardware
	MU_BIT_AVAIL_NODE	0x80000000
	MU_BIT_AVAIL_SCI3	0x00800000
	MU_BIT_AVAIL_SCI2	0x00400000
	MU_BIT_AVAIL_SCI1	0x00200000
	MU_BIT_AVAIL_SCI0	0x00100000
	MU_BIT_AVAIL_S3	0x00080000
	MU_BIT_AVAIL_S2	0x00040000
	MU_BIT_AVAIL_S1	0x00020000
	MU_BIT_AVAIL_S0	0x00010000
	MU_BIT_AVAIL_IO	0x00001000
	MU_BIT_AVAIL_MB3	0x00000800
	MU_BIT_AVAIL_MB2	0x00000400
	MU_BIT_AVAIL_MB1	0x00000200
	MU_BIT_AVAIL_MB0	0x00000100
	MU_BIT_AVAIL_T7	0x00000080
	MU_BIT_AVAIL_T6	0x00000040
	MU_BIT_AVAIL_T5	0x00000020
	MU_BIT_AVAIL_T4	0x00000010
	MU_BIT_AVAIL_T3	0x00000004
	MU_BIT_AVAIL_T2	0x00000003
	MU_BIT_AVAIL_T1	0x00000002
	MU_BIT_AVAIL_T0	0x00000001
222	CLOCK_10MSEC	the number of clocks a T-chip sees in 10 msec (for OS timer calibration)
223	APPLY_CLOCK	a mask so the MU know what to turn on
224	INFO_MU	addition MU trace/status information
225-232	DCACHEx	# bytes in module data cache, x = cpu
233-240	ICACHEx	# bytes in module instruction cache, x = cpu
241	CPA_ERR_ENABLE	cpa error enable info
242	CPA_PMON_LATENCY	cpa error enable info
243	CPA_PMON_CONTROL	cpa error enable info

(continued next page)

3.2 Configuration Parameters (continued)

244 CMC_ERR_EN0 ccmc error enable info

CCMC DARK

RQI_PAR_ERR	0x80000000	(HARD)
RQI_SEQ_ERR	0x40000000	(HARD)
RQI_FMT_ERR	0x20000000	(HARD)
RSI_PAR_ERR	0x10000000	(HARD)
RSI_SEQ_ERR	0x08000000	(HARD)
RSI_FMT_ERR	0x04000000	(HARD)
NCI_PAR_ERR	0x02000000	(HARD)
NCI_FMT_ERR	0x01000000	(HARD)
SOCM_RS_ERR	0x00800000	(SOFT)
BAD_ADDR_ERR	0x00400000	(SOFT)
BAD_LAST_ERR	0x00200000	(SOFT)
BAD_CMD_ERR	0x00100000	(SOFT)
VIRT_INDEX_ERR	0x00080000	(SOFT)
MEM_ALIAS_ERR	0x00040000	(SOFT)
IRD_DIRTY_ERR	0x00020000	(SOFT)
NON_COH_ERR	0x00010000	(SOFT)
SICM_NON_COH_ERR	0x00008000	(SOFT)
SICM_MISS_ERR	0x00004000	(SOFT)
RQI_MISS_ERR	0x00002000	(HARD)
NOT_OWNER_ERR	0x00001000	(HARD)
NOT_LOCKED_ERR	0x00000800	(HARD)
RSQ_FULL_ERR	0x00000400	(HARD)
SRC_DIRTY_ERR	0x00000200	(HARD)
SH_LEVEL_ERR	0x00000100	(HARD)
SCI_TRAP_ERR	0x00000080	(HARD)
REFRESH_ERR	0x00000040	(SOFT)
SBIT_ECC_ERR	0x00000020	(SOFT)
MBIT_ECC_ERR	0x00000010	(SOFT)
MULT_ECC_ERR	0x00000008	(SOFT)
RSI_BAD_RESP_ERR	0x00000004	(HARD)
UNREC_TAG_ERR	0x00000002	(HARD)
FLUSH_RP_ERR	0x00000001	(HARD)

incomming

Single Bit

CCMC LITE

RQI_PAR_ERR	0x80000000	(HARD)
RQI_SEQ_ERR	0x40000000	(HARD)
RQI_FMT_ERR	0x20000000	(HARD)
RSI_PAR_ERR	0x10000000	(HARD)
RSI_SEQ_ERR	0x08000000	(HARD)
RETRY_HIT_ERR	0x04000000	(HARD)
NOT_OWNER_ERR	0x02000000	(HARD)
NOT_LOCKED_ERR	0x01000000	(HARD)
SHARED_ERR	0x00800000	(HARD)
COLLISION_ERR	0x00400000	(HARD)
SRC_DIRTY_ERR	0x00200000	(HARD)
SH_LEVEL_ERR	0x00100000	(HARD)
SCI_TRAP_ERR	0x00080000	(HARD)
MJ_MN_ERR	0x00040000	(SOFT)
MEM_ALIAS_ERR	0x00020000	(SOFT)
VIRT_INDEX_ERR	0x00010000	(SOFT)
NON_COH_ERR	0x00008000	(SOFT)
IRD_DIRTY_ERR	0x00004000	(SOFT)
INV_ADDR_ERR	0x00002000	(SOFT)

(continued next page)

3.2 Configuration Parameters (continued)

	INV_LAST_ERR	0x00001000	(SOFT)
	LICM_to_MEM_WR_PAR_ERR	0x00000800	(HARD)
	LICM_to_MEM_RD_PAR_ERR	0x00000400	(SOFT)
	LICM_frm_MEM_WR_PAR_ERR	0x00000200	(HARD)
	LICM_frm_MEM_RD_PAR_ERR	0x00000100	(SOFT)
	SBIT_ECC_ERR	0x00000080	(SOFT)
	MBIT_ECC_ERR	0x00000040	(SOFT)
	MBIT_TAG_ERR	0x00000020	(SOFT)
	NLOCK_ECC_ERR	0x00000010	(HARD)
	REFRESH_ERR	0x00000008	(SOFT)
	MULT_ECC_ERR	0x00000004	(HARD)
	LICM0_SRC_ERR	0x00000002	(SPECIAL)
	LICM1_SRC_ERR	0x00000001	(SPECIAL)
245	CMC_ERR_EN1	ccmc error enable info (Not Used)	
246	CSB_SN	the csb/node serial number, SN_CSB is identical the complex serial number	
247	COMP_SN		
248-255	EXC_CAUSE_SAVEx	HPMC exception cause csr value	
256	MU_CONFIG	enable optional MU firmware features	
	MU_SHORT_MEM_INIT	0x00000004	
	MU_CPA_PING	0x00000002	
	MU_SCRUB	0x00000001	
257	PRESENT_NODES	present nodes are one's that we should be able to talk to	
258	AVAIL_NODES	available nodes are ones that we can run things on	
259	HW_CONFIG	Enable optional hardware configurations	
	MEM_3_BANK_1	0x00008000	which memory banks are installed
	MEM_3_BANK_0	0x00004000	
	MEM_2_BANK_1	0x00002000	
	MEM_2_BANK_0	0x00001000	
	MEM_1_BANK_1	0x00000800	
	MEM_1_BANK_0	0x00000400	
	MEM_0_BANK_1	0x00000200	
	MEM_0_BANK_0	0x00000100	
	TO_CONTROL_MODE	0x00000010	disable anti-starvation?
	SCI_POLL_MODE	0x00000008	disable anti-starvation?
	CLUSTER_MODE	0x00000004	use Cluster-mode?
	CD_MODE	0x00000002	use CD-mode?
	CMC_DARK	0x00000001	at least 1 ccmc2 installed
260-267	MEM_OUTPARAM4_x	output parameter 4	
268-271	SCI_CONFIG_x	NodeChip config CSR values (1 per slice)	
272-273	IO_CONFIG_x	Landmarc config CSR values (1 per unit 0/1)	
274	SCI_SPLIT_TIMEOUT		
275	SCI_SYNC_INTERVAL		
276	SCI_SCAN_CONFIG	NodeChip scan config values	
277-278	MSIZEx	memory size on other nodes	
279	APA_CONFIG		
280	APA_TIMEOUT		
512-526	PN_*	part numbers as read from the cop chips	
528-542	SN_*	serial numbers as read from the cop chips	

(continued next page)

3.2 Configuration Parameters (continued)

544-558	REV1_*	revision info as read from the cop chips
560-574	MASK_*	ecn mask as read from the cop chips
576-590	REV2_*	more rev info as read from the cop chips

for the cop entries, the '*' is, in order from lowest number to highest number: A0 B0 A1 B1 A2 B2 A3 B3 MB0 MB1 MB2 MB3 IO CSB MU

599-613	*JTAG*	the JTAG IDs for the Convex Gate Arrays
614-767	unassigned	
768-773	BITS_RINGx	1 location for each ring 0-5
774-779	START_RINGx	1 location for each ring 0-5
780	RING_DATA_MIN	
899	RING_DATA_MAX	

3.3 **xconfig

This program was created to allow the user to manually set some of the configuration parameters in the MU based on which boards were currently in the system. It also forces the regeneration of the node configuration file on the workstation.

Synopsis

The program begins by reading the current configuration parameters from the MU. The display will show the parameters in the text window as well as graphically showing the installed boards in the picture on the left side of the window. The user can modify the installed boards and arrays by clicking on them. Arrays and boards that are installed are gold. Those items not installed are green (ie. you can see the backplane which is also green).

Some parameters such as the size of memory, bank interleave and network cache size are controlled through pull-down menus.

Finally, less well known parameters are defined in a file ("/spp/data/config/node_#.prm" where # is the node number) that may be edited. These are mostly on/off bits of the runwhat field and values that should be written to CSRs. The user may edit this file and use the Files pull-down menu to select reading those parameters.

Once all of the parameters are correct, the user may copy them back to the MU and optionally save them in the NVRAM through the config pull-down menu.

Note: Although the user can specify many options, none of these take effect until the user copies the parameters to the MU. Then the user must then go to the RESET pull-down menu and click on a Level 3 reset to get the changes to take effect.

Note: Only the parameters that are shown on the screen are affected by this program. Any other MU configuration parameters are left alone.

Files Pull-down Menu

"Reset MU Parameters File" There is a standard file containing the editable parameters for the MU configuration that can be copied to the /spp/data/config/node_#.prm file. This option allows the user (continued next page)

3.3 xconfig (continued)

to recover when (s)he accidentally corrupts the parameter file.

"Get MU Parameters from File" Once the parameters file is edited to contain CSR values the user desires, (s)he can force reading those parameter values with this button. The display is updated to show those values, but they are not copied to the MU. See the Config Pulldown Menu to get them to the MU.

The file "/spp/data/config/node_#.prm" contains a list of most of the MU's programmable options. There are descriptions of each parameter above the controlling statement. Most parameters are enabled by the argmuent "On" or "Off". Any lines begining with a # sign are comments. Those lines beginning with a number are an index into the MU parameter table, and have the value that is to be written as the second argument on the line. This file can be edited to meet the needs of the user.

"Exit"

Config Pulldown Menu

"Get MU Paramaters from MU" Forces the list of displayed parameters to be reread from the MU.

Be aware that many of the parameters that are manually controlled via the GUI interface will be overwritten to reflect the state the MU is in. In other words, if you specify a memory board size then get parameters from the MU, you must recheck your settings since the buttons will be updated to reflect MU state.

"Write MU Parameters to MU"

Once all of the local parameters have been adjusted to the proper state, they can be copied to the MU. This button does not cause the parameters to be permanently save. The user must execute a "Reset Level 3" for the MU to reread the parameters and act on them.

"Store MU Parameters in NVRAM"

This buttonb will copy all the parameters to the MU and write the parameters to the non-volatile ram on the MU so that the parameters are retained for the next time the power to the MU is cycled.

Installed Hardware

On the left of the display there is a picture representing the expected node configuration. Gate Arrays and Boards that are installed are shown in a gold color, those that are not are shown in a green (representing that you can see the backplane which is also shown as green). The user clicks on those icons that he would like to change the status of. The boards toggle between "installed" and "missing" with each click.

If parameters are copied from the MU, then this display will update to show what the MU believes it has installed.

(continued next page)

Memory Parameters

There is a pulldown menu from the main panel that allows the user to set several different memory board parameters. Since all memory boards in the system must be identical, there is no need to have separate parameters for each board. Each of the parameters that can be set have other menus where the user can click on the available features.

The parameters that the user can set are the size of the memory, the interleave scheme that is desired and the amount of memory dedicated to the node cache (whatever it is eventually called). These buttons will also change state after getting parameters from the MU to reflect the MU's opinion on current configuration.

Clocks

Another option is the ability to specify which clock you like the MU to default to when powering up. Whichever clock is selected when the "Save Configuration" is executed will be the new default.

NOTE: The clock commands can also be used to select the current clock source and can be used to change the system clock without affecting the configuration (unless a save is done).

Power

The text window on the right hand side of the main window will display the current status of the power bricks. You should NOT attempt to power up the system unless the installed configuration shown on the left is correct.

The "Power On" and "Power Off" buttons allow the user to turn off all of the supplies except the MU. 48 volts must still be turned off manually. The user can stop and change a memory board without having to power down the whole system complex. Note however that the configuration parameters (CSR values) that are written to the Agent and CCMC are lost at this point and a Level 3 reset must be issued (via the Reset menu) to rewrite those CSRs to the expected values.

The "Update Power Values" button allows the user to display/monitor the status of the node's DC voltages. Each time the button is pushed the voltages are read from the MU and displayed.

3.4 **I/O Chassis Address Assignments

The current disk address assignments for the XA I/O chassis are:

		Left Side						
		2	6	B	9	4	Front of Cabinet	
		3	8	C	A	5		
		Right Side						
Front of Cabinet		2	6	4	9	B		
		3	8	5	A	C		

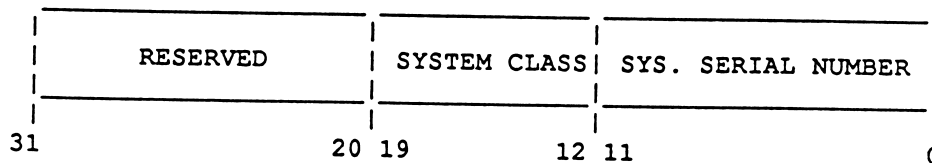
DDS (DAT) is assigned address 0

The addresses for disks in a CD I/O chassis are selected by jumpers on the disk drives. The current manufacturing defaults are:

		Left Side						
1st SCSI	6	5	4	3	2	---	DDS 0	Front of Cabinet
	SD8	SD6	SD4	SD2	SD0			
2nd SCSI	6	5	4	3	2	---	DDS 1	
	SD9	SD7	SD5	SD3	SD1			

3.5 **System Complex Serial Number Assignments

SPP/MPP COMPLEX SERIAL NUMBER DEFINITION



SPP1 STARTS AT CLASS 10.
 CLASSES GO UP FROM THERE (0x11, 0x12, 0x13...)

SERIAL NUMBERS START AT 1 AND GO UP FROM THERE

537 thru 67599	for XAs
600 thru 69631	for CDs

3.7 Diff/SE SCSI Converter 220-000045-200 (continued)

Switch Settings

When installed in the XA I/O Chassis for SPP, the last Ancot board (the one closest to the DAT drives) should provide termination of the differential chain. All of the other Ancot boards before the last one should not terminate the differential SCSI bus. In the XA I/O Chassis, none of the Ancot SED616M board should terminate the single-ended end of the bus. Termpower for the differential bus is provided by the host.

On the Ancot board closest to the DAT drives (the last one on the SCSI differential bus) the switch settings should be the following:

Switch	Position	Description
1	Down	Termpower for SE Side
2	UP	No Termpower for Diff Side
3	UP	No Terminator for SE Side
4	Down	Terminator for Diff Side

For all other Ancot boards in the SCSI bus, switch setting should be as follows:

Switch	Position	Description
1	Down	Termpower for SE Side
2	UP	No Termpower for Diff Side
3	UP	No Terminator for SE Side
4	UP	No Terminator for Diff Side

3.8 **Modem Setup (USA)

Make the nodes for the dialin and dialout lines. You must be root.

```
mknod /dev/cua00 c 1 0x204001
mknod /dev/cul00 c 1 0x204001
mknod /dev/ttyd00 c 1 0x204000
```

Add these lines at the end of the /usr/lib/uucp/Devices file.

```
Direct cul00 - 19200 direct
ACU cul00 cua00 19200 hayes96t  ## use this if tone dialing
#ACU cul00 cua00 19200 hayes96p  ## use this if pulse dialing
```

Add these lines at the end of the /usr/lib/uucp/Dialers file.

```
#
# Hayes Ultra96 Smartmodem -- This entry is set up for the configuration
# Convex Computer Corporation uses on it's SPP test station
# "hayes96t" is for tone dialing
# "hayes96p" is for pulse dialing
#
hayes96t =,-, "" \dATV1Q\d\r OK\r \dAT\r\c OK\r \d\pATDT\T\r\c CONNECT
hayes96p =,-, "" \dATV1Q\d\r OK\r \dAT\r\c OK\r \d\pATDP\T\r\c CONNECT
```

(continued next page)

3.8 Modem Setup (USA) (continued)

Add this line to inittab.

```
a0:3:respawn:/etc/getty -h ttyd00 19200
```

After installing the modem do the following:

```
cu -s 19200 -l/dev/cul00 -m dir
```

When you see the connected message, type:

```
at&fm0e0v0x0q1s37=9s0=1&c1&d3&K5&w0 <cr>
```

This will setup the modem and disconnect you.

3.9 **De-configuring Memory with "ccmu"

Here is a table of the possible configurations that can be used when de-configuring the memory system i.e telling the machine the memory board is smaller than what is physically installed.

The Table shows the value for the USEMEMSIZE field that "ccmu" requires. The table is based on a fully populated MTVs.

Original Node Mem. Size	Original Board Size	De-config. Node Mem. Size	De-config. Board Size	ccmu parm. USEMEMSIZE
256 Meg	64 Meg	N/A	N/A	N/A
512 Meg	128 Meg	256 Meg	64 Meg	1
1024 Meg	256 Meg	256 Meg	64 Meg	1
2048 Meg	512 Meg	256 Meg	64 Meg	1
2048 Meg	512 Meg	512 Meg	128 Meg	2

Note:

De-configuring memory to configurations other than those in the above table, IS NOT possible using this method and will cause system crashes.

The steps required to change the USEMEMSIZE parameter with "ccmu" would be:

```
sppdsh> ccmu<cr>
ccmu> up<cr>
ccmu> put USEMEMSIZE 0x<number><cr> Where <number> = new value.
ccmu> down<cr>
ccmu> push<cr>
ccmu> quit<cr>
sppdsh> do_reset<cr>
```

3.10 **SPP1000/CD Disk Limits

The SPP1000/CD uses single ended SCSI controllers. The CD I/O chassis uses cables instead of the I/O backplanes used in the XA I/O chassis. Cable length restrictions cause a 5 disk limit per SCSI controller.

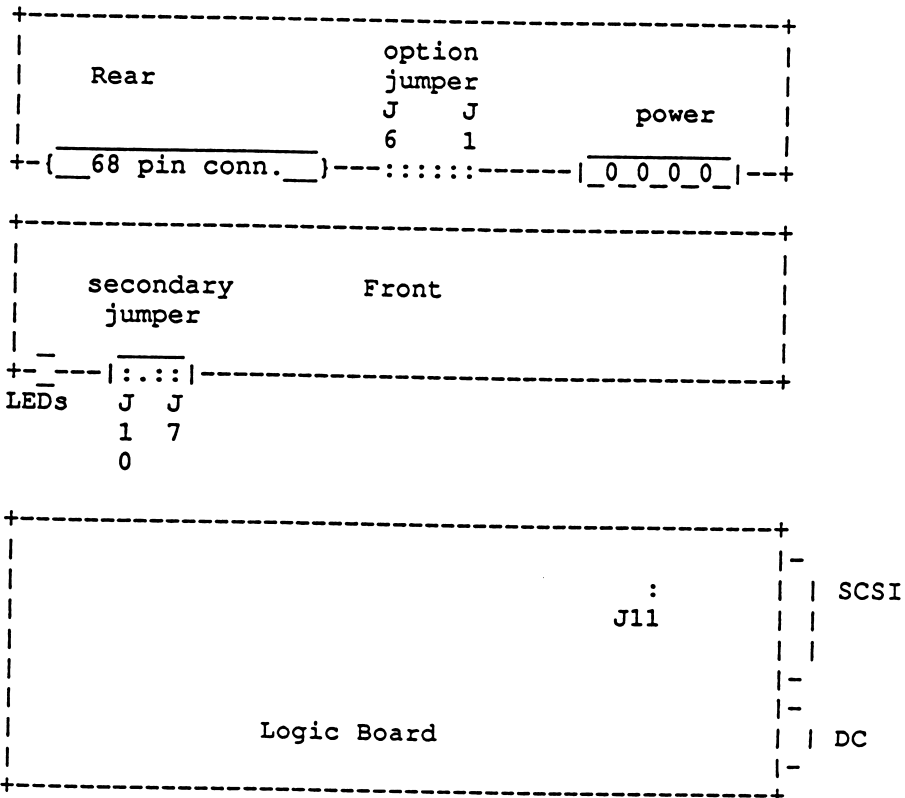
3.11 **CCMC2 and PITA Card

Because of a design error in the CCMC2 (CCMC Dark) revA, a conversion card is required to be placed between the CSB4 and the CCMC2 revA. This card is known as a PITA card. The PITA card converts ECL clocks from the CSB4 to TTL clocks for the CCMC2 revA. The CCMC2 is being re-spun (for this and other reasons), so the PITA card will eventually be dis-carded. The hardware is layered as follows:

CSB4/socket/PITA/socket/CCMC2/heatsink

3.12 **Disk Drive Jumpers DEC DSP3210W 2 GB SCSI 204-000028-200

The unit select jumpers have to be set on the disk for a CD I/O chassis. Unit selection is assigned by chassis slot in the XA I/O chassis. The following gives jumper locations, their meanings and default settings.



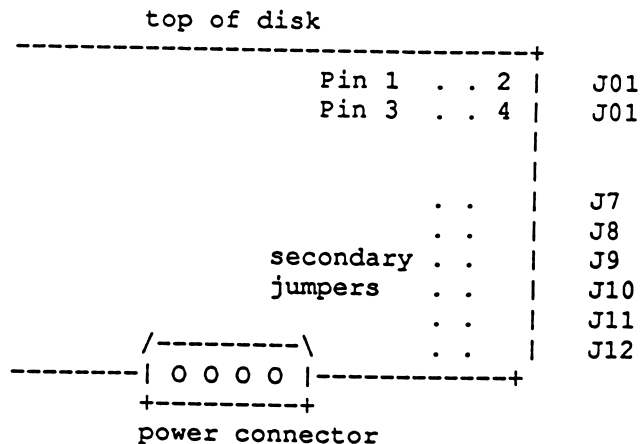
Since this device is SCSI wide (i.e. can support SCSI ID 0x0 through 0xF), four (4) jumpers are required to set the SCSI id.

3.13 Disk Drive Jumpers Seagate ST15150W (continued)

Jumper setting J6 controls spindle sync and should be left off under normal conditions. J5 controls remote LED connection and this jumper should also be left off.

For use in an SPP/XA Chassis, leave jumpers J1-J6 OUT.

Jumper J01 (the 4 pin jumper in the upper right corner of the drive) controls termination and termpower settings. In single-ended drives, we want the drive to receive and provide terminator power to the SCSI bus.



Jumper settings for J01:

- | | |
|--------------------------------------|---|
| Pin 1 to Pin 3 | Drive provides termpower to the SCSI bus.
*****This jumper should be set. |
| Pin 1 to Pin 2 | Drive provides its own termpower |
| Pin 2 to Pin 4 | Termpower for the drive from SCSI bus.
*****This jumper should be set. |
| Pin 1 to Pin 3 and
Pin 2 to Pin 4 | Terminator power to the SCSI bus and drive.
(Both should be set for default operation) |

Jumper Settings for Secondary Jumpers

J12	J11	J10	J9	J8	J7
Off	Off	Off	Off	Off	Off

Description of secondary jumper settings:

- J12 Enable Termination
 Jumper off to disable drive termination, Jumper on to enable drive termination. Note: Most of the applications have an external Fast/Wide terminator, in which case this jumper should not be installed.
- J11 Reserved
- J10 Parity Check
 Leave jumper off to enable parity check. Jumper on to disable parity checking

(continued next page)

3.15 **Ergo Ultra VGA Display

Sometimes when unpacking and starting up a new 712 workstation with the D1196A (Ergo Ultra VGA) Display, the screen will appear to have a strong green tint to it, like the green electron gun is being overdriven. Anyone who has seen one knows exactly what this looks like.

This is a result of an incorrect workstation boot prom setting (much like our MU configuration parameters or OBP NVRAM variables). The "monitor" setting is in error (most likely a 6), and needs to be changed. To do so:

Reboot the workstation. When it says to, press the escape key abort booting. It should end up at a boot prom prompt.

Enter "monitor 3" to select the best setting for the D1196A display.

Power cycle the workstation and allow it to boot.

3.16 **MTV SIMM Locations and Sizes

MTV top view

Data Simm 0DL	-----	Data Simm 1DH
Data Simm 1DL	Pwr Brick	Data Simm 0DH
Data Simm 0CL		Data Simm 1CH
Data Simm 1CL	-----	Data Simm 0CH
Data Simm 0BL	Data Simm 0AL	Data Simm 1BH
Data Simm 1BL	Data Simm 1AL	Data Simm 0BH
Tag Simm 0TL	Data Simm 1AH	Tag Simm 1TH
Tag Simm 1TL	Data Simm 0AH	Tag Simm 0TH

- There will be at least four different MTV configurations. 64 Mbyte, 128Mbyte, 256Mbyte and 512Mbyte (A 1024Mbyte MTV may be added later)
- In all but one MTV configuration two different simms sizes will be used on each MTV.
- A simm can have drams on one or two sides.
- There are four different types of simms - 4Mbyte, 8Mbyte, 16Mbyte and 32Mbyte (A 64Mbyte simm may be added later)
- Different size simms can be identified by the color code dots on each SIMM.

170-000001-001	4Mbyte	WHITE	DOT
170-000002-001	8Mbyte	YELLOW	DOT
170-000003-001	16Mbyte	GREEN	DOT
170-000004-001	32Mbyte	BLUE	DOT
170-000005-001	64Mbyte	RED	DOT(Not used at this time)

4.1 **MU Firmware Power-Up Tests

Here's the current default set of diagnostic subtests for MU firmware version 0.0.42.26.

Bus error test:

1000	source address data pattern subtest	enabled
1010	force subtest	enabled
1020	read timeout subtest	enabled
1021	write timeout subtest	enabled
1030	source address functional subtest	enabled

Access register test:

1100	data pattern subtest	enabled
------	----------------------	---------

RAM test:

1200	quadlet data pattern subtest	enabled
1201	doublet 0 data pattern subtest	enabled
1202	doublet 1 data pattern subtest	enabled
1203	byte 0 data pattern subtest	enabled
1204	byte 1 data pattern subtest	enabled
1205	byte 2 data pattern subtest	enabled
1206	byte 3 data pattern subtest	enabled
1210	quadlet address subtest	enabled
1211	doublet address subtest	enabled
1212	byte address subtest	enabled
1220	byte selection subtest	enabled
1230	parity error subtest	enabled

Interrupt test:

2000	interrupt acknowledge subtest	enabled
2010	primary int data pattern subtest	enabled
2011	hard error int data pattern subtest	enabled
2012	secondary int data pattern subtest	enabled
2020	interrupt operation subtest	enabled
2021	interrupt combination subtest	enabled
2030	source subtest	enabled

DaRT interface test:

2100	data pattern subtest	enabled
2110	register address subtest	enabled
2120	access subtest	enabled
2130	interrupt subtest	enabled
2140	loopback subtest	enabled
2150	bus error and master-mode addr subtest	enabled

BBC test:

2200	data pattern subtest	enabled
2210	address subtest	enabled
2220	battery subtest	enabled
2230	interrupt subtest	enabled
2240	NVRAM uniqueness subtest	enabled

COP interface test:

2300	data pattern subtest	enabled
2310	loopback subtest	enabled

(continued next page)

4.1 MU Firmware Power-Up Tests (continued)

EEPROM access test:		
3000	write-enable subtest	enabled
3010	protection subtest	enabled
Scan test:		
3100	TBC data pattern subtest	enabled
3110	TBC address subtest	enabled
3120	access subtest	enabled
3130	reset subtest	DISABLED
	(broken subtest)	
3140	loopback subtest	enabled
3150	interrupt subtest	enabled
????	ring integrity subtest	DISABLED
	(requires gate arrays and power)	
COP interface test:		
3200	access subtest	enabled
Power and environment test:		
3300	data pattern subtest	enabled
3310	ADC conversion control subtest	enabled
3320	ADC interrupt subtest	enabled
Refresh test:		
3400	data pattern subtest	enabled
3410	functional subtest	enabled
CXRing configuration test:		
3500	data pattern subtest	enabled
3510	loopback subtest	enabled
Clock control test:		
3600	data pattern subtest	enabled
Burst-mode access test:		
3700	burst copy subtest	enabled
3710	burst bus error subtest	enabled
LCD test:		
3800	LCD read subtest DISABLED
	(not all machines have LCDs)	
MAUI interface test:		
4000	data pattern subtest	enabled
4010	address subtest	enabled
4020	read byte selection subtest	enabled
4021	write byte selection subtest	enabled
4030	interrupt subtest	enabled
4031	hard error interrupt subtest	enabled
4040	access subtest	enabled
4050	read parity subtest	DISABLED
	(fails in MUs other than MU #8)	
4051	write parity subtest	enabled

(continued next page)

4.1 MU Firmware Power-Up Tests (continued)

CXBar interface test:

5000	request hard error interrupt subtest (problem with IO interaction)	DISABLED
5001	response hard error interrupt subtest (problem with IO interaction)	DISABLED
5010	loopback write subtest (problem with IO interaction)	DISABLED
5011	loopback read subtest (problem with IO interaction)	DISABLED

4.2 **Preparing a SPP Node for Diagnostics

Before running diagnostics on a SPP node you should save the MU configuration parameters. See article 3.6 in this guide. You should then turn off the PLLOADBP bit and turn on the INITMEM bit, if it is not on, in the INIWHAT parameter.

Here is an example:

```
elmo_d:/users/sppuser$ ccmu
Welcome to the Convex Configuration Manager Utility

ccmu: 1.3 (Fri Jun 17 09:58:08 1994)

ccmu> pull
ccmu> up
ccmu> get initwhat
node value      parm  parm
  0  0x5882df0f   1  INITWHAT
    STARTPL PLCPUINIT PLINITMEM PLLOADBP INITMEM MUERRENABLE INITLAND
    SIZEMEM INITSlice3 INITSlice2 INITSlice1 INITSlice0 INITCXBAR DORESET
    PWRLKCHECK GO
ccmu> put initwhat 0x5802df0f
ccmu> get initwhat
node value      parm  parm
  0  0x5802df0f   1  INITWHAT
    STARTPL PLCPUINIT PLINITMEM INITMEM MUERRENABLE INITLAND SIZEMEM
    INITSlice3 INITSlice2 INITSlice1 INITSlice0 INITCXBAR DORESET
    PWRLKCHECK GO
ccmu> down
ccmu> push
ccmu> quit
elmo_d:/users/sppuser$ do_reset 3 <node#>
elmo_d:/users/sppuser$
```

Monitor the test station console window or /spp/data/event_log for the following messages:

```
+++> 135
<Fri Jul 15 16:16:51 1994> info:0x47601300
<MU FW>:1.3.0.0:node_init.c:150
node: 0x0 (0)
node initialization complete
****
```

(continued next page)

4.2 Preparing a SPP Node for Diagnostics (continued)

```
+++> 159
<Fri Jul 15 15:16:51 1994> info:0x456a0401
do_reset:1.3.0.0:../do_reset.c:326
terminating normally
elapsed time: xx:xx
no errors encountered
****
```

The node and its memory have now been initialized and diagnostics, including "iod", can be executed. Remember to do a "do_reset 3 <node#>" between each diagnostic. There should be no need to initialize memory with "/spp/scripts/mem_init_tc" if the above steps are followed.

After you have completed the diagnostic session, restore the MU configuration parameters. See article 3.6 of this guide.

```
elmo_d:/users/sppuser$ do_reset 3 <node#>
elmo_d:/users/sppuser$
```

When the "do_reset 3 <node#>" completes, the OBP prompt should appear in the SPP-UX Console window and the node is ready to boot.

4.3 **CST

CST stands for Camelot Scan Test. This test utility creates and executes several different types of tests though all are related to the scan capabilities inherent in the Convex designed devices. There is a Help button in the CST window if needed,

Invoke CST, from a sppdsh, with "cst #" where # = the node to test.

There are several CST Test Categories:

Scan Ring Verification

The first test that is executed is the Scan Ring Verification. It issues a Test-Logic-Reset command to all of the JTAG devices in the design. These devices then switch to the scan device i.d. mode, or in the case of a commercial device without the optional jtag i.d., the bypass mode. The program then reads each ring to verify that what is being scanned matches the configuration file that exists on the system.

Scan Ring Tests

After guaranteeing that all of the devices are as expected, the program proceeds by putting all of the devices in all of the different scan modes and proving that the ring remains intact. This mode relies on certain clock logic being in place and therefore is useful for finding bugs in the simple environment prior to getting to the more complicated tests.

DC Connectivity Tests

These tests verify via the JTAG boundary scan mechanism that the devices on either side of testable wires exist. Testable wires are those with JTAG pins at either end. These tests are created during run time and are dependent on how fully the node is populated.

(continued next page)

4.3 CST (continued)

AC Connectivity Tests

These tests verify via a combination of JTAG boundary scan and a proprietary Convex internal scan mechanism that the high speed paths between Convex designed devices can operate at speed. These tests are created during run time and are dependent on how fully the node is populated.

Internal Gate Array Tests

These tests are design to test the internals of the Convex designs.

The tests include several subsets. There are DC internal tests which look for stuck transistors on virtually 100% of the design. There are AC internal tests that exercise certain critical paths at speed to verify they meet timing expectations. There are RAM internal tests dedicated to verifying uniqueness among addresses and that no data bit is stuck high or low.

Exiting CST

DO NOT use the KILL window selection in the CST window, to cause the program to terminate. The proper way to exit is to use the Exit CST button in the Files pull-down menu. The program establishes a hardware lock when it starts up which will not be released unless it goes through the proper exit functions. If the process is killed, this hardware lock will remain in effect and have to be manually released by the next user. All sub-windows have a Close command to clear the window from the screen.

4.4 **Known CST Failures

Sometimes CST will fail Scan Ring Verification on the first pass. Press the "System Test" button a second time and CST should pass.

4.5 **sppring

sppring is a functional test for the CTI (Convex Turoidal InterConnect) and distributed memory portion of an Exemplar System.

sppring verifies the operation of the related hardware. Specifically, the test accomplishes the following:

- Verifies that the test station can access CCMC.
- Verifies that the test station can perform read and write operations to a node's memory without assistance from the T-chip.
- Verifies that the test station can download a driver to node memory.
- Verifies that the test station can access the CTI controller through CCMC and CTI.
- Verifies that the distributed memory circuit is functioning normally.
- Verifies memory coherency logic.
- Verifies memory non-coherent logic.
- Verifies BDT, special BDT, and EIR interrupt logic.
- Verifies messaging hardware.

sppring does not exhaustively check the CTI or test the gate array internal circuit, e.g. CCMC and BPR.

(continued next page)

4.5 sppring (continued)

sppring is invoked with "sppring [-option [-option [...]]]"

Valid options are:

-c value[,value]	Execute the specified class(es) of tests. Default is to execute all classes.
-s value[,value]	Execute the specified list of subtests. Default is to execute all subtests.
-pb [on,off]	Pause at the beginning of each subtest Default is not to pause.
-pe [on,off]	Pause at the end of each subtest. Default is not to pause.
-pp [on,off]	Pause when the subtest passes. Default is not to pause.
-pf [on,off]	Pause when the subtest fails. Default is not to pause.
-mt value	Set the number of allowable total failures to "value".
-ms value	Set the number of allowable subtest failures to "value".
-lt value	Set number of loops run to "value".
-ls subtest# loop#	Set number of loops run ("loop#") for subtest#.

Valid Classes and Subtests:

Class 1 Subtests

Subtest 1000 - CCMC Access Test

verifies that the workstation can access CCMC CSRs.

Subtest 1010 - Memory Access Test

verifies that the workstation can perform read and write operations to memory through MU, MAUI, XBAR, CCMC, and BPR.

Subtest 1020 - SppRing Controller Access Test

This subtest verifies that the workstation can access SppRing controller CSRs through SppRing.

Subtest 1030 - Network Cache Access Protection Test

verifies the network cache access protection circuit. It attempts to access network cache as part of the memory and expects an error to occur.

Subtest 1040 - CCMC Error Reporting Test

verifies the CCMC error reporting circuit. It sets various CCMC error bits and checks if the corresponding error occurs.

Subtest 1050 - Variable Length Write/Read Test

writes and reads variable length packets to memory. It checks out variable length write/read to memory from MU.

Class 2 Subtests

Subtest 2000 - Tag Column Functionality Test

tests tag column functionality. It performs a walking 1s and 0s test on the first word in each bank of tag memory (walking from least significant bit to most significant bit). All subtests in this class are downloaded to T-chip cache before execution.

(continued next page)

4.5 spping (continued)

Subtest 2010 - Tag Uniqueness Test
tests tag location uniqueness. It writes an incrementing value to each location in tag memory, then verifies that all locations contain the expected value.

Subtests 2020-2080 - Tag MATS Test
performs a MATS+ memory pattern test over the tag memory. The MATS+ algorithm consists of several passes through tag memory, with intermixed reads and writes of two patterns. In this implementation of the algorithm, various 64 bit data patterns (2 per subtest) are chosen such that all data and ECC bits will be tested within the 64 bit word (over the course of all pattern subtests), and all detectable intra-longword dependencies will be detected.

Subtests 2090-2150 - Tag NTA Test
performs a NTA memory pattern test over the tag memory. The Nair, Thattle, and Abraham algorithm consists of several passes through tag memory, with intermixed reads and writes of two patterns.

Class 3 Subtests

Subtest 3000 - Memory Column Functionality Test
tests memory column functionality. It performs a walking 1s and 0s test on the first word in each bank of memory (walking from least significant bit to most significant bit). All subtests in this class are downloaded to T-chip cache before execution.

Subtest 3010 - Memory Uniqueness Test
tests memory location uniqueness. It writes an incrementing value to each location in memory, then verifies that all locations contain the expected value.

Subtests 3020-3080 - Memory MATS Test
perform a MATS+ memory pattern test over the memory. The MATS+ algorithm consists of several passes through memory, with intermixed reads and writes of two patterns. In this implementation of the algorithm, various 64 bit data patterns (2 per subtest) are chosen such that all data and ECC bits will be tested within the 64 bit word (over the course of all pattern subtests), and all detectable intra-longword dependencies will be detected.

Subtests 3090 -3150- Memory NTA Test
perform a NTA memory pattern test over the memory. The Nair, Thattle, and Abraham algorithm consists of several passes through memory, with intermixed reads and writes of two patterns. For each subtest, a pair of special patterns are used for NTA testing.

Subtest 3160 - Memory Cycle Test
performs load and store of byte/half word/full word to verify that processor can perform these basic operations correctly. It also verifies flush and purge operations. Fetch-and-inc and fetch-and-dec are also verified.

4.5 spping (continued)

Class 4 Subtests

Subtest 4000 - Local Non-Coherent Read/Write Test
tests non-coherent read/write to memory from different processors in the same node. It performs non-coherent read and write operations to memory from MU.

Subtest 4010 - SppRing Non-Coherent Read/Write Test
tests SppRing non-coherent read/write commands. It performs non-coherent read and write operations over the SppRing to another node's memory.

Class 5 Subtests

Subtest 5000 - Node Download Verification Test
The PA-RISC driver program is downloaded by spping. The HP test station communicates to the driver program to verify that it is successfully downloaded.

Subtest 5010 - External Interrupt Verification Test
tests the CPA external interrupt circuit. It writes to EIR to generate an external interrupt and checks if the correct interrupt is generated.

Subtest 5020 - MAUI CSRs Test
tests the following MAUI CSRs : MAUI_PVT_HI, MAUI_PVT_LO, MAUI_INT_MASK, MAUI_TOCMR_HI, MAUI_TOCMR_LO. It writes a walking 1, walking 0, and six different patterns to CSRs to verify that the CSR is written and read with correct result.

Subtest 5030 - MAUI Thread Count Circuit Test
tests the MAUI thread count circuit. It increments MAUI_TC by reading MAUI_TC_FINC from 0 to 0xffff and to 0. For each increment, it checks to verify that MAUI_PVT_HI and MAUI_PVT_LO are incrementing when MAUI_TC sign bit is not set, and they are not incrementing when MAUI_TC sign bit is set. It then decrements MAUI_TC by reading MAUI_TC_FDEC. MAUI_TC is checked for each FDEC read. MAUI_TC_FCLR is also verified that it resets MAUI_TC.

Subtest 5040 - MAUI TOC Circuit Test
tests the MAUI TOC circuit. It syncs up all MAUIs in different nodes. MAUI_TOC_HI and MAUI_TOC_LO are checked to ensure that they sync up correctly.

Subtest 5050 - MAUI TOC Interrupt Verification Test
verifies the MAUI TOC interrupt circuit. It sets MAUI_TOCMR to a known value and verifies that an interrupt is generated when MAUI_TOC matches MAUI_TOCMR.

Class 6 Subtests

Subtests 6000-6110 - Intranode Coherency Tag Test
verify the tag info for intranode coherency logic for both even and odd cache lines of each slice. It performs a combination of load and store operations among processors in a single node. Vector, dirty, and special BDT bits are checked. This subtest starts by flushing the processor cache. Tag vector

(continued next page)

4.5 spring (continued)

and dirty bits are checked to ensure that it equals to zero. A load operation followed by a store operation are executed by each processor. The vector and dirty bits are compared against the expected values at each stage.

Class 7 Subtests

Subtest 7160-7230 - Full Coherency Test

verify the tag info for the system coherency logic for both the even and odd cache lines of each slice. It performs combination of load and store operations among nodes and processors. These subtests perform a series of load operations among nodes and processors within a node followed by a series of store operations. The cache and memory states, and vector bits are compared with the expected values. Any discrepancy is reported.

Subtests 7240-7310 - Full Coherency Parallel Test

test the coherency logic in parallel for both even and odd cache lines of each slice. All processors in all specified nodes read from a 32-byte memory line in parallel and the vector and dirty bits, memory and cache states are checked and verified. All processors in all specified nodes then write to the 32-byte memory line and the tag info are checked.

Class 8 Subtests

Subtest 8000 - CCMC Messaging Circuit Test (per slice)

verifies the messaging circuit by writing messages to all available T-chips on the same slice of a neighboring node. It then confirms that the messages are received by the corresponding T-chip of the destination node.

Subtest 8010 - CCMC Messaging Circuit Test (all cpus)

verifies the messaging circuit by writing messages to all available T-chips of a neighboring node. It then confirms that the messages are received by the corresponding T-chip of the destination node.

Subtest 8020 - CCMC Queue Disable Test

verifies the messaging queue disable circuit by writing messages to a tchip that has its message queue disabled. It then verifies that a HPMC is received..

Subtest 8030 - CCMC Messaging Queue Full Test

verifies the messaging queue full circuit by writing messages to a tchip that has a full message queue. It then verifies that a HPMC is received..

Class 9 Subtests

Subtests 9000-9050, 9120-9170, 9240-9290, 9360-9410, 9480-9530

- SCI MATS Test

perform a MATS+ memory pattern test over the internode memory for each sci ring and all sci rings together. The MATS+ algorithm consists of several passes through memory, with intermixed reads and writes of two patterns. In this implementation of the algorithm,

(continued next page)

4.5 spping (continued)

various 64 bit data patterns (2 per subtest) are chosen such that all data and ECC bits will be tested within the 64 bit word (over the course of all pattern subtests), and all detectable intra-longword dependencies will be detected.

Subtests 9060-9110, 9180-9230, 9300-9350, 9420-9470, 9540-9590

- SCI NTA Test

perform a NTA memory pattern test over the internode memory for each sci ring and all sci rings together. The Nair, Thattle, and Abraham algorithm consists of several passes through memory, with intermixed reads and writes of two patterns.

Class 10 Subtest

Subtest 10000 - Coherency Stress Test

stresses the coherency circuit by performing random load/store/flush from each tchip.

Subtest 10010 - Messaging Stress Test

stresses the messaging hardware by sending a large amount of messages to another node.

Subtest 10020 - Messaging/Coherency Stress Test

stresses the coherency circuit by performing random load/store/flush and sending messages to another node.

Subtest 10030 - CCMC Stress Test

stresses the coherency circuit by performing random coherent and non-coherent operations.

Subtest 10040 - CCMC Messaging Stress Test

stresses the coherency circuit by performing random coherency and non-coherent operations and sending messages to another node.

Subtest 10050 - Load/Store One Line Stress Test

stresses the coherency circuit by performing random load and store on a globally shared block for all nodes.

Subtest 10060 - Mix Ops One Line Stress Test

stresses the coherency circuit by performing random coherent/non-coherent operations on a globally shared block for all nodes and sending messages to another node.

Subtest 10070 - Load/Store Two Line Stress Test

stresses the coherency circuit by performing random load and store on two globally shared blocks mapping on the same network cache for all nodes.

Subtest 10080 - Mix Ops Two Line Stress Test

stresses the coherency circuit by performing random coherent/non-coherent operations on two globally shared blocks mapping on the same network cache for all nodes and sending messages to another node.

Subtest 10090 - Load/Store Multi- Line Stress Test

stresses the coherency circuit by performing random load and store on multiple globally shared blocks mapping on the same network cache for all nodes.

(continued next page)

4.5 sprising (continued)

Subtest 10100 - Mix Ops Multi-Line Stress Test stresses the coherency circuit by performing random coherent/non-coherent operations on multiple globally shared blocks mapping on the same network cache for all nodes and sending messages to another node.

Class 11 Subtests

These subtests are mainly used by manufacturing for initial board bringup. It does not require any tchip to be present.

Subtest 11000 - TS Address Bits Init initializes tags by writing to the WRINIT csr for subtest 11010.

Subtest 11010 - TS Address Bits Uniqueness Quick Check performs a quick uniqueness check on address bits. It does an uniqueness check on a 1KByte block of each address bit starting from 0x10000 to 0x20000000.

Class 12 Subtests

These subtests are mainly used by manufacturing for initial board bringup. It does not require any tchip to be present.

Subtest 12000 - TS Tag Column Functionality Test tests tag column functionality. It performs a walking 1s and 0s test on the first word in each bank of tag (walking from least significant bit to most significant bit).

Subtest 12010 - TS Tag Uniqueness Test tests tag location uniqueness. It writes an incrementing value to each location in tag up to 1Kbytes, then verifies that the locations contain the expected value.

Subtests 12020-12070 - TS Tag MATS Test perform a MATS+ memory pattern test over the tag upto 1Kbytes. The MATS+ algorithm consists of several passes through tag, with intermixed reads and writes of two patterns. In this implementation of the algorithm, various 64 bit data patterns (2 per subtest) are chosen such that all data and ECC bits will be tested within the 64 bit word (over the course of all pattern subtests), and all detectable intra-longword dependencies will be detected. For each subtest, a pair of special patterns are used for MATS testing.

Subtests 12080-12130 - TS Tag NTA Test perform a NTA memory pattern test over the tag upto 1Kbytes. The Nair, Thattle, and Abraham algorithm consists of several passes through tag, with intermixed reads and writes of two patterns. For each subtest, a pair of special patterns are used for NTA testing.

Subtests 12140 - TS Tag Init initializes tag for class 13 subtests.

Class 13 Subtests

These Subtests are mainly used by manufacturing for initial board bringup. It does not require any tchip to be present.

(continued next page)

4.5 sppring (continued)

Subtest 13000 - TS Memory Column Functionality Test
tests memory column functionality. It performs a walking 1s and 0s test on the first word in each bank of memory (walking from least significant bit to most significant bit).

Subtest 13010 - TS Memory Uniqueness Test
tests memory location uniqueness. It writes an incrementing value to each location in memory up to 1Kbytes, then verifies that the locations contain the expected value.

Subtests 13020-13070 - TS Memory MATS Test
perform a MATS+ memory pattern test over the memory upto 1Kbytes. The MATS+ algorithm consists of several passes through memory, with intermixed reads and writes of two patterns. In this implementation of the algorithm, various 64 bit data patterns (2 per subtest) are chosen such that all data and ECC bits will be tested within the 64 bit word (over the course of all pattern subtests), and all detectable intra- longword dependencies will be detected. For each subtest, a pair of special patterns are used for MATS testing.

Subtests 13080 -13130- TS Memory NTA Test
perform a NTA memory pattern test over the memory upto 1Kbytes. The Nair, Thattle, and Abraham algorithm consists of several passes through memory, with intermixed reads and writes of two patterns. For each subtest, a pair of special patterns are used for NTA testing.

4.6 **sppring quick pass

The class 2 and 3 subtests in sppring take a long time to run. If you wish to run a "quick verify" on a node you should use the following command.

```
sppring -s 1000-1050,2000-2020,2120,3000-3020,3120,3220,4000-9590
```

This will greatly reduce the execution time. It will also reduce the amount of test coverage on the memory boards. If you are trying to reproduce a memory failure, all of class 2 and 3 may be needed.

4.7 **cputest

cputest is the HP PA7100 chip test.
It is a pass/fail test and only has 1 subtest (1000).
This test takes approximately 16 minutes per cpu.
If a failure occurs, the failing cpu should be replaced.
Remember to return the test failure output with the failing cpu.
This data has to be returned to HP along with the failing cpu.

4.8 **Margins

All diagnostics are known to pass at the following margins.

nominal clk, nominal voltages
upper clk, upper voltages
upper clk, lower voltages

4.9 **iod I/O Debugger

iod is an interactive I/O debugger.
iod should not be run on a node that is currently running SPP-UX.
iod is really a special shell that accepts test commands.

iod was written with the following attributes in mind:

- o Diag tests should be small and fast.
- o Convenient interface.
- o There should be some ability for the user to do some things manually (e.g. create and send certain data packets).

```
To start iod:
cd /spp/scripts/iod
iod <cr>
iod>
```

iod has a focus of one node at a time, but the user can change which node is being tested.

For example, consider a two node system (1 and 5) and an iod test script, foo. The following iod command sequence can be used to the landmark board in each node.

```
iod> set_node 1
iod> do foo
iod> set_node 5
iod> do foo
```

You can run multiple iod processes in different windows on different nodes without any problems. When you start iod, the target node is 0.

Various iod commands are:

@	!	?	break
buf_cmp	buf_copy	buf_del	buf_info
buf_make	buf_mod	buf_print	buf_read
chnl	chnl_reg	clear	continue
cwd	display_var	dma	do
echo	else	end	endif
file_to_mem	get_node	if	history
lm_init	loop	man	mask
mem_cmp	mem_dump	mget	mput
mem_read	mr	mem_write	mu
mu_dbg	mu_lmt	mw	path
printf	q	quit	r
rand	sbus_map	set	set_node
set_output	sleep	step	stop
unix	unset	while	scsi_cst_init
scsi_db_init	scsi_exec	scsi_format	scsi_init
scsi_inquiry	scsi_io_init	scsi_load_commands	scsi_load_init
scsi_load_msg	scsi_load_status	scsi_load_script	scsi_m2s_init
scsi_ncr_init	scsi_ncr_refresh	scsi_parm	scsi_per_map
scsi_read	scsi_read_finish	scsi_request_sense	scsi_rewind
scsi_start	scsi_stop	scsi_test_unit_ready	scsi_write

To get more information about an iod command use the "man" command.

(continued next page)

4.9 od I/O Debugger (continued)

Example:

```
iod> man do <cr>
```

USAGE

```
do file_name [param1 param2 ...]
```

DESCRIPTION

do executes the commands in a script file. Parameters listed after the file_name will be passed to the script.

EXAMPLE

```
do test1
do test2 200 $b "foo"
```

```
iod>
```

4.10 **iod test scripts

There is a set of scripts available for testing SPP1000 systems. Since the XA system uses a differential SCSI controller and a CD system uses a single ended SCSI controller, there must be separate scripts for each.

These scripts are in /spp/scripts/iod/exec and include:

all_cd	p3_cd_3490_destruct	p3_xa_3490_safe
all_xa	p3_cd_3490_safe	p3_xa_disk_destruct
dma	p3_cd_disk_destruct	p3_xa_disk_safe
dv	p3_cd_tape_safe	p3_xa_tape_destruct
fddi	p3_cd_tape_destruct	p3_xa_tape_safe
lf	p3_cd_tape_safe	safe_cd
map	p3_xa_3490_destruct	safe_xa

As always, use caution when selecting a test script. Some scripts will destroy existing data on the media.

To run the "all_xa" script from a sppdsh do the following:

CAUTION: The scripts "all_cd" and "all_xa" include data destructive tests. If it is desired to run an "all" script in a non-destructive mode, comment out all lines that include *_destruct in the script. A # = comment.

Insert a write enabled tape into the DDS drive.

```
cd /spp/scripts/iod/exec
iod
iod> do all_xa(or all_cd)<cr>
```

Here is what will be done by the all_xa script:

```
cwd ../../lf
do lf_test

cwd ../../dv
do dv_test

cwd ../../p3
do p3_xa_disk_safe
do p3_xa_disk_destruct
```

(continued next page)

4.10 iod test scripts (continued)

```
do p3_xa_tape_safe
do p3_xa_tape_destruct
do fddi_test
```

```
cwd ../../dma
do dma_test
```

```
cwd ../../map
do map_test
```

```
echo "All tests passed"
```

4.11 **iod p3_disk_* Alternate Unit Selection

There has been some confusion over what the scsi id's will be for the systems going out the door. Right now, iod will try to test 3 drives at scsi id's: 2, 4, and 0xa.

Obviously, if a systems has a different configuration, iod will fail when trying to test a drive that is "missing", or not test one that is there.

To make iod do the right thing for your system, you may need to edit a couple of files:

```
/spp/scripts/iod/p3/disk/destruct/<xa or cd>/u0_s0_test
/spp/scripts/iod/p3/disk/safe/<xa or cd>/u0_s0_test
```

These files run the destructive and safe tests [you will probably just play with the safe one] inside iod.

Here is the safe one:

```
echo -n "unit:      "
scsi_parm unit 0
echo -n "sbus:      "
scsi_parm sbus_nbr 0
set unit = 0
set sbus_nbr = 0
cwd ./id_test
echo -n "scsi id:  "
scsi_parm scsi_id 0x2
do id_test
echo -n "scsi id:  "
scsi_parm scsi_id 0x4
do id_test
echo -n "scsi id:  "
scsi_parm scsi_id 0xa
do id_test
```

Note the structure:

```
echo -n "scsi id:  "
scsi_parm scsi_id 0xa
do id_test
```

Here, these three lines test scsi id 0xa. Removing them will keep the safe tests from testing this particular drive. Of course, changing the scsi_id parameter will result in a different drive being tested. You can also see where to change the unit and the sbus_number. The exact same thing applies to the destructive tests as well.

4.12 **sppring failures kill config

During startup the sppring diagnostic saves and then changes the configuration parameters in the MU. At the end of a successful pass, sppring restores the configuration parameters. If sppring detects a failure or is aborted during the test, the configuration parameters may not be restored properly. Because of the risk of this happening, it is recommended that the parameters be saved with "ccmu" before running sppring. See article 3.6 for details.

4.13 **CST Rev 0.15f and CPBs Error Message

There was a pin name change between the cpas and cpbs. You can ignore the following errors during cst startup.

```
JTAG Pin Type Error AGENT-TC1_AG_PVCH
JTAG Pin Type Error AGENT-TC1_AG_PBUSYCH
JTAG Pin Type Error AGENT-TC0_AG_PBUSYCH
JTAG Pin Type Error AGENT-TC0_AG_PVCH
ERROR: JTAG unable to match AGENT pin AG_CMCQ_RQIO_BUSY
ERROR: JTAG unable to match AGENT pin CMCS_AG_RSIO_BUSY
JTAG Pin Type Error AGENT-TC1_AG_PVCH
JTAG Pin Type Error AGENT-TC1_AG_PBUSYCH
JTAG Pin Type Error AGENT-TC0_AG_PBUSYCH
JTAG Pin Type Error AGENT-TC0_AG_PVCH
ERROR: JTAG unable to match AGENT pin AG_CMCQ_RQIO_BUSY
ERROR: JTAG unable to match AGENT pin CMCS_AG_RSIO_BUSY
JTAG Pin Type Error AGENT-TC1_AG_PVCH
JTAG Pin Type Error AGENT-TC1_AG_PBUSYCH
JTAG Pin Type Error AGENT-TC0_AG_PBUSYCH
JTAG Pin Type Error AGENT-TC0_AG_PVCH
ERROR: JTAG unable to match AGENT pin AG_CMCQ_RQIO_BUSY
ERROR: JTAG unable to match AGENT pin CMCS_AG_RSIO_BUSY
JTAG Pin Type Error AGENT-TC1_AG_PVCH
JTAG Pin Type Error AGENT-TC1_AG_PBUSYCH
JTAG Pin Type Error AGENT-TC0_AG_PBUSYCH
JTAG Pin Type Error AGENT-TC0_AG_PVCH
ERROR: JTAG unable to match AGENT pin AG_CMCQ_RQIO_BUSY
ERROR: JTAG unable to match AGENT pin CMCS_AG_RSIO_BUSY
```

4.14 **Determining Utility/Diag/Firmware Versions

The following commands can be used to determine:

Which versions of sppdsh, ccmu, do_reset, load_eprom and mu_firmware are installed on your test station.

```
diag_vers /spp/bin/sppdsh
diag_vers /spp/bin/ccmu
diag_vers /spp/bin/do_reset
diag_vers /spp/bin/load_eprom
diag_vers /spp/firmware/mu_firmware
```

Which version of cst is installed on your test station.
/spp/bin/cst -v

Which versions of mu_firmware and pl are installed on your MU.
/spp/bin/node_info

The pl version will only be displayed if the node has diags 2.3 or higher.

4.15 **verify_cables and verify_rings

The verify_cables and verify_rings scripts can be used as a quick check of the SCI cables and rings are functioning properly.

Enter "verify_cables <node cabling order>" where <node cabling order> = how the nodes are physically cabled.

If a system was cabled 0 to 3 to 2 to 1 to 0, the command would be "verify_cables 0 3 2 1". Additionally specific rings can be selected for test with the "-r" option.

To test ring 3 in the above example, the command would be "verify_cables 0 3 2 1 -r 3".

Enter "verify_rings" to run this scripts. This will test all SCI rings currently enabled.

5.1 **hard_logger

Hard errors are detected by the various gate arrays and reported by the MU. There are 3 MU registers involved in the reporting and subsequent logging of hard errors.

Name	Address
hard error interrupt status	0xf0c04050
hard error interrupt mask	0xf0c04058
first hard error	0xf0c0405c

hard_logger should be invoked automatically when a hard error occurs. hard_logger can be invoked manually: hard_logger <node_num> (f) The (f) switch forces hard_logger to run even if the hard error interrupt mask has been cleared. hard_logger uses the first hard error reg and'ed with the hard error interrupt mask to determine which interrogators to run.

Interrogators are responsible for finding out which errors are set in a particular part and then printing out the extractor name and the necessary arguments for the extractor.

Extractors are responsible for pulling out the information specific to the failure that occurred.

The interrogators are in /spp/scripts.
The extractors are in the chip specific directories.

```
/spp/scripts/cpa
/spp/scripts/cmc
/spp/scripts/maui
/spp/scripts/xbar
```

It is possible to examine the first hard error register and run the appropriate interrogators and extractors manually.

5.2 **MU Registers

MU Address address	mapping name	size	access	description
0x0000_0000 to 0x0000_1FFF	core	1/w/b	r*32	protected EEPROM, must set enable in acc_csr AND groun test point to write
0x0000_2000 to 0x000F_FFFF	eprom	1/w/b	r/w*32	must set enable in acc_csr to write
0x0010_0000 to 0x003F_FFFF	extended eprom	1/w/b	r/w*32	must set enable in acc_csr to write. may not be populated
0x0040_0000 to 0x007F_FFFF	unimplemented address space, access causes bus error			

(continued next page)

5.2 MU Registers (continued)

```

-----
0x0080_0000
  to          TBC internal registers, refer to data sheet
0x0080_007F
-----
0x0080_0080
  to          unimplemented register space, access causes bus error
0x0080_0FFF
-----
0x0080_1000
  to          SONIC internal registers, refer to data sheet
0x0080_10FF
-----
0x0080_1100
  to          unimplemented register space, access causes bus error
0x0080_1FFF
-----
0x0080_2000
  to          MAUI register space, see maui_regs.txt
0x0080_21FF
-----
0x0080_2200
  to          unimplemented register space, access causes bus error
0x0080_2FFF
-----
0x0080_3000
  to          MU local CSRs
0x0080_3027

```

address	name	size	access	description
0x0080_3000	isr_csr	word	r/w/u*16	interrupt status
0x0080_3008	ifr_csr	word	r/w/e*16	interrupt force
0x0080_3010	imr_csr	word	r/w*16	interrupt mask

the following description defines the interrupt levels and register bit positions. MU interrupts are loosely partitioned across 2 csrs - the primary interrupt csr indicates the source and level of all the interrupts, and the secondary isr contains the source of power and harderror interrupts.

As before, a 1 in any bit of the isr_csr indicates the interrupt is pending; writing 1's clears set bits and writing 0's has no effect. If a bit is cleared and the interrupt source is still asserting its interrupt line, the corresponding isr_csr bit will re-set after clearing and generate another interrupt to the processor if enabled. In the case of power and harderrori interrupts, the secondary isr should be cleared before the primary isr.

A 1 in any bit of the force register forces the corresponding interrupts, and a 0 in any bit of the mask register masks the corresponding interrupts. All interrupts may be individually masked.

(continued next page)

5.2 MU Registers (continued)

bit	level	description	
15-13	7	unused	
12	6	pwr_intr	-power interrupt occurred, read secondary interrupt csr for source
11	5	bbc_rti	-bbc square wave(periodic interrupt) *unused.
10	5	bbc_int	-bbc alarm interrupt
9	4	maui_int	-maui service interrupt
8	3	dart_int	-dart controller service interrupt
7-4	2	ccmc_serr[3-0]	-ccmc soft error occurred.
3-2	2	unused	
1	2	herrintr	-harderr interrupt(s) occurred, read secondary interrupt csr for source
0	1	tbc_int	-tbc service interrupt

address	name	size	access	description
0x0080_3004	sisr_csr	word	r/w/u*16	2ndary interrupt status
0x0080_300C	sifr_csr	word	r/w/e*16	2ndary interrupt force
0x0080_3014	simr_csr	word	r/w*16	2ndary interrupt mask

As before, a 1 in any bit of the `sisr_csr` indicates the interrupt is pending; writing 1's clears set bits and writing 0's has no effect. If a bit is cleared and the interrupt source is still asserting its interrupt line, the corresponding `isr_csr` bit will re-set after clearing and generate another interrupt to the processor if enabled.

A 1 in any bit of the force register, forces the corresponding interrupts. A 0 in any bit of the mask register masks the corresponding interrupts.

bit	level	description	
15	6	cc_int[3]	- Insufficient 48V for full node power
14	6	cc_int[2]	- A failed 48V supply exists
13	6	cc_int[1]	- io_chassis_alarm
12	6	cc_int[0]	- io_node_alarm
11	6	adc_int[2]	- a/d converter #3 conversion complete
10	6	adc_int[1]	- a/d converter #2 conversion complete
9	6	adc_int[0]	- a/d converter #1 conversion complete

expect each to occur from 8.2 to 16.4 ms depending on the voltage input. the converter comes up halted, need to start converter, wait for sip = 1, and then clr/enable interrupts.

(continued next page)

5.2 MU Registers (continued)

8	6	non_mu48vf1	- non-mu/sci 48V fail/breaker sense 1 = 48V breaker tripped
7	6	io_mu_vl2s	- io 12V undervoltage; rare, shutdown, clear/enable after turning on io power
6	6	vhcs_fail	- HECL clock voltage (+2.0V) comparator
5	6	rt_ff_int #3	- T-chip side fan #3 failed
4	6	rt_ff_int #2	- T-chip side fan #2 failed
3	6	rt_ff_int #1	- T-chip side fan #1 failed
2	6	lt_ff_int #3	- SCI-side fan #3 failed
1	6	lt_ff_int #2	- SCI-side fan #2 failed
0	6	lt_ff_int #1	- SCI-side fan #1 failed

these interrupts are rare, clr/en after
reset. fans take care of spin-up delay.

0x0080_3018 | rst_csr_lo | word | r/w*16 | reset register (low word)

All bits in this register assert gate array resets when clear; this register initializes to 0x0001, and you can force a global reset by clearing the lsb.

bit	description
-----	-------------

15	AG3.B - Tchip 7 reset; 0 = reset, 1 = not in reset
14	AG3.A - Tchip 6 reset; 0 = reset, 1 = not in reset
13	AG2.B - Tchip 5 reset; 0 = reset, 1 = not in reset
12	AG2.A - Tchip 4 reset; 0 = reset, 1 = not in reset
11	AG1.B - Tchip 3 reset; 0 = reset, 1 = not in reset
10	AG1.A - Tchip 2 reset; 0 = reset, 1 = not in reset
9	AG0.B - Tchip 1 reset; 0 = reset, 1 = not in reset
8	AG0.A - Tchip 0 reset; 0 = reset, 1 = not in reset
7-4	agent[3-0] reset; 0 = reset, 1 = not in reset
3-2	io[1-0] reset; 0 = reset, 1 = not in reset
1	MAUI reset; 0 = reset, 1 = not in reset
0	mu board s/w reset; 0 = reset, 1 = not in reset

0x0080_301C | rst_csr_hi | word | r/w*16 | reset register (high word)

All bits in this register assert array resets when clear; this register initializes to 0x0000.

bit	description
-----	-------------

15-12	read/write able, no h/w effect
11-10	read/write, no h/w effect
9	xbar_q reset; 0 = reset, 1 = not in reset
8	xbar_s reset; 0 = reset, 1 = not in reset
7-4	sci[3-0] reset; 0 = reset, 1 = not in reset
3-0	cmcc[3-0] reset; 0 = reset, 1 = not in reset

0x0080_3020 | clk_csr | long | r/w/u*32 | clk gen control register

This register resets to all 0's and is used to control the system clock generation logic on the CSB.

(continued next page)

5.2 MU Registers (continued)

Register definition for CSB3 and beyond:

bit	field	description
31-8	rsvd	reads 0's, writes ignored
7	clkimode	= 0 gate arrays not in internal scan mode; do not externally gate JTAG TCK = 1 gate arrays in internal scan mode; externally gate JTAG TCK
6	clkensel	= 0 use clken (bit 0) to enable/disable clocks = 1 use external clock enable signal
5-3	clkmode	selects clock mode: 000 = clocks running 100 = clocks running with TCK 010 = reserved 110 = reserved 001 = single clock 101 = double clocks 011 = double clocks @ 1/2 frequency 111 = double clocks @ 1/4 frequency
2-1	clkssel	selects the clock source: 00 = normal clock 10 = high-margin clock 01 = external clock via SMB connector 11 = reserved
0	clken	= 1 enables clock generation = 0 disable clock outputs to the ASICs; clocks should be disabled before changing any other control bits!

Register definition for CSB2 only:

bit	field	description
31-8	rsvd	reads 0's, writes ignored
7-6	clkcyb	extra controls
5-3	clkmode	selects clock mode: 000 = clocks stopped 001 = clocks running 010 = clocks running 011 = clocks running 100 = single clock 101 = double clocks 110 = double clocks @ 1/2 frequency 111 = double clocks @ 1/4 frequency

(continued next page)

5.2 MU Registers (continued)

2-1	clkssel	selects the clock source: 00 = normal clock 01 = high-margin clock 10 = JTAG test clock 11 = external clock via SMB connector
0	clken	= 1 enables clock generation = 0 disable clock outputs to the ASICs; clocks should be disabled before changing any other control bits!

0x0080_3024 | err_csr | word | r*6/r/w/u*10 | bus error source register

This register resets to all 0's and indicates the source of a bus error. Writes of 0's clears the error source bits and don't cause a bus error, writes of 1's set the bus error bits and force a bus error immediately - the current cycle terminates in a bus error.

bit	field	description
15-10	rsvd	reads 0's, writes ignored
9	alt_mstr	1 = bus error while SONIC was bus master 0 = bus error while 040 was bus master
8	maui	1 = maui access caused bus error
7	unexp_ta	1 = TA handshake seen when access didn't need it
6	prot_err	1 = tried to write protected memory/eprom
5	rsvd	reserved, reads 0 writes ignored
4	time_out	1 = BAT popped before saw a TA
3-0	muxd_perr	any 1 = read parity error on maui or ram access

0x0080_3028 | badl_csr | long | r/w/u*32 | berr low address register

This register resets to all 0's and stores the lower 16 bits of the address being accessed during the most recent bus error. It can be written by the 68CL040 for testing.

bit	field	description
31-16	rsvd	read only, writes ignored
16-0	badl	lower 16 bits of address during last bus error

0x0080_302C | badh_csr | long | r/w/u*32 | berr high address register

This register resets to all 0's and stores the upper 16 bits of the address being accessed during the most recent bus error. It can be written by the 68CL040 for testing.

bit	field	description
31-16	rsvd	read only, writes ignored
16-0	badh	upper 16 bits of address during last bus error

(continued next page)

5.2 MU Registers (continued)

 0x0080_3030 | cxrc_csr | long | r/w/u*32 | CxRing config control

This read/write register resets to all 0's and contains the following bits for controlling CxRing configuration:

bit	field	description
31-8	rsvd	reads 0's, writes ignored
7	busy	read only; this bit is set to 1 whenever (debug == 1) or (select == lxx); it is reset to 0 after 32 shift clock are received; the control data registers should not be written while this bit is 1
6-5	rsvd	reads 0's, writes ignored
4	debug	= 1 selects bclk for the shift clock; the data register begins shifting when this bit is set = 0 selects normal 040 read/write mode and/or normal CxRing shift mode
3	rsvd	reads 0's, writes ignored
2-0	select	selects the clock used to access the data register (aka the shift clock): 0xx = bclk 100 = CxRing ASIC #0 101 = CxRing ASIC #1 110 = CxRing ASIC #2 111 = CxRing ASIC #3

0x0080_3034 | cxrd_csr | long | r/w/u*32 | CxRing config data register

This read/write register resets to all 0's and can be written with 32 bits of data to be used in configuring the CxRing interface ASICs.

When shifting is enabled via the cxrc_csr, the register shifts right; the former lsb is inverted and becomes the new msb

0x0080_3038 | refc_csr | long | r/w/u*32 | refresh control register

This read/write register resets to all 0's and contains the following bits for controlling main memory refresh:

bit	field	description
31-15	rsvd	reads 0's, writes ignored
14	clock	when selected, alternating writes of 0 and 1 to this bit will operate the refresh logic
13	select	1 = select the clock bit to decrement the refresh counter 0 = select bclk to decrement the refresh counter
12	enable	1 = load prescale value into counter and begin refresh generation; this bit must be 1 in both test and normal mode 0 = stop refresh with all refresh signals zero; refresh must be stopped when switching between test and normal mode via the select bit

(continued next page)

5.2 MU Registers (continued)

11-8 | refresh | read only; current state of the refresh signals
 7-0 | prescale | refresh pulse width = prescale * bclk period

0x0080_303C | acc_csr | long | r/w/u*32 | access control register

This 32-bit read/write register resets to all zeros. The force bits stay set until cleared and force bad parity on writes to either MAUI or RAM, depending on the state of rlm0_frc.

bit	field	description
31-16	rsvd	read only, writes ignored
15	eeeprom4mb	read-only field indicates EEPROM size 1 = 4MB EEPROM installed 0 = 1MB EEPROM installed
14	lvasci_oe	Alexis SCI 3.3v buffers output enable 1 = enable outputs 0 = disable outputs (default)
13	lva_oe	Alexis 3.3v buffers output enable 1 = enable outputs 0 = disable outputs (default)
12	lx_oe	Alexis 5v buffers output enable 1 = enable outputs 0 = disable outputs (default)
11	Pbus_oe	P-bus output enable 1 = enable outputs 0 = disable outputs (default)
10	Cbus_oe	SCI CBUS output enable (internally inverted) this bit is no longer used in CSB's 1 = enable outputs 0 = disable outputs (default)
****ASB-10	sparetsel*	Spare Trim Select control (ASB only) 1 = disable trim pot 0 = enable trim pot (default) THIS BIT MUST BE WRITTEN TO 1 AFTER RESET
9	no_retry	1 = defeat retry logic, 0 = allow up to 4 retries
8	eeeprom_we	1 = enable writes to EEPROM 0 = disable writes to EEPROM
7	en_ints	1 = enable ALL interrupt sources 0 = disable ALL interrupt sources this bit isolates the interrupt control logic from interrupt sources. if this bit is clear, no interrupts will be allowed into the interrupt logic.
6-5	rsvd	reads 0's, writes ignored
4	rlm0_frc	1 = frc_perr bits force bad RAM write parity 0 = frc_perr bits force bad MAUI write parity
3	frc_perr0	1 = force parity error on data[31:24] (msbyte)
2	frc_perr1	1 = force parity error on data[23:16]
1	frc_perr2	1 = force parity error on data[15:8]
0	frc_perr3	1 = force parity error on data[7:0] (lsbyte)

 0x0080_3040
 to unimplemented register space, access causes bus error
 0x0080_3FFF

(continued next page)

5.2 MU Registers (continued)

0x0080_4000
to
0x0080_407F

ENVMON registers

address	name	size	access	description
0x0080_4000	cop_csr	byte	r*3,0*1,r/w/e*4	t-chip cop i/f ctl and stat
	bit	description		
7		1 = cop i/f busy, writes to "cop_dr" ignored 0 = cop i/f idle		
6		reads 0, writes ignored		
5		1 = disable automatic shift on writes to cop_dr for testing 0 = enable automatic shift on writes to cop_dr		
4		1 = cop i/f in loopback mode, data out gets inverted and clocked back into the shift register input 0 = cop i/f in normal mode		
3-0		cop chip decoder; all bits r/w/e 0x0 to 0x7 = no device selected 4'b1000: mu_hp_select = 8'h01; ta0 4'b1001: mu_hp_select = 8'h02; tb0 4'b1010: mu_hp_select = 8'h04; ta1 4'b1011: mu_hp_select = 8'h08; tb1 4'b1100: mu_hp_select = 8'h10; ta2 4'b1101: mu_hp_select = 8'h20; tb2 4'b1110: mu_hp_select = 8'h40; ta3 4'b1111: mu_hp_select = 8'h80; tb3		

0x0080_4004 | cop_dr | byte | r/w/u*8 | data to/from t-chip cops

bit	description
7-0	writing to this register stores the written data, sets the cop_csr.busy bit to a 1, and begins shifting the written data byte out to the cop chips while simultaneously sampling and shifting in data from the cop chips. Although the register may be read at any time, the result is invalid until the cop_csr.busy bit negates. Once the register is written, further writes are ignored until the busy bit negates.

0x0080_4008 | trim_csr | byte | r1,r/w1,r/w/u6 | trim i/f ctl and stat

bit	description
7	1 = trim i/f busy, writes to "trim_dr" ignored 0 = trim i/f idle
6	1 = stepping pot up towards Vh 0 = stepping pot down towards Vl
5-0	trim clock count-down register, range is 0 to 63 cnts

Writing to this register sets the trim_csr.busy bit, loads the
written data into a down-counter, and begins clocking the eepot.
(continued next page)

5.2 MU Registers (continued)

Each active-lo trim clock pulse steps the selected pots one time in the selected direction and decrements the register contents. Register reads reflect the number of trim clocks remaining. Register writes while busy are ignored; neither the control lines nor the select lines may be changed while busy.

 0x0080_400C | trim_selr| word | 0*6,r/w*10 | tsc804 ctl and status

bit	description
15	1 = disable eepot clocking on trim_csr writes 0 = enable eepot clocking on trim_csr writes
14-12	r/w, no h/w effect
11	1 = eepot selected; mu_io_vccsel_ (I/O board Vcc) 0 = not selected
10	1 = eepot selected; mu_pb0_vccprsel_ (HP Vcc) 0 = not selected
9-8	1 = eepot selected; mu_bd_vddscisel_[1:0] (SCI +1.2V) 0 = not selected
7	1 = eepot selected; mu_ga_vttsel_ (ASIC -2V) 0 = not selected
6-5	1 = eepot selected; mu_bd_vddgasel_[1:0] (ASIC 1.2V) 0 = not selected
4	1 = eepot selected; mu_pb1_vdlssel_ (ASIC 3.3V) 0 = not selected
3-0	1 = eepot selected; mu_mb_vccsel_[3:0] (MB3-MB0 Vcc) 0 = not selected

 Write this register to select the pots before writing the trim_csr. Resets to 0x0000 (no eepot selected, eepot clocking enabled).

0x0080_4010 | adc1_csr | byte | r*3,r/w*1,r*1,r/w3 | tsc804 ctl and stat
 0x0080_4018 | adc2_csr | byte | r*3,r/w*1,r*1,r/w3 | tsc804 ctl and stat
 0x0080_4020 | adc3_csr | byte | r*3,r/w*1,r*1,r/w3 | tsc804 ctl and stat

bit	name	description
7	over	1 = overrange on selected channel 0 = no overrange on selected channel
6	status	1 = conversion in progress 0 = a/d result valid, data will not change while status is 0
5	sip	1 = integrating input channel 0 = not integrating input channel
4	rl_h0	1 = converter in free-run mode 0 = converter will complete a started conversion and then idle in auto-zero
3	s1_d0	returns converter mode: 1 = converter set for 8 single-ended channels 0 = converter set for 4 differential channels
2-0	chan	selects input channel to be converted; may only be changed while sip is 0. If s1_d0 is set, chan[2] is ignored.

 Writing to this register sets the converter mode (free-run or halted), input mux mode (differential or single-ended), and selects the input channel to be converted. The interface h/w will not allow the channel to change while sip is 1 to
 (continued next page)

5.2 MU Registers (continued)

This read/write register drives the power brick control logic. Resets to 0x0000 (all power off).

0x0080_4030 | sf_reg | word | r*16 | supply-fail register

bit	name	description
15	pb2xist*	0 = PB2 installed (VTSCIIO, VEE, VTTSCI) 1 = no PB2 board
14	pblxist*	0 = PB1 installed (VHE, VHC, VDL) 1 = no PB1 board
13	pb0xist*	0 = PB0 installed (VCCPR) 1 = no PB0 board
12	jmpout[2]*	extra jumper header bit 1 = jumper installed 0 = jumper missing
11-10	spare[1:0]*	unused, reads 1's, may be pulled down
9-8	vddsci_sf1*	any 0 = corresponding SCI VDD +1.2V supply dead 1 = supply output has not failed
7-4	mb_sf1*	any 0 = corresponding memory board supply failed 1 = supply output has not failed
3-0	ga_1_2sf1*	any 0 = corresponding ASIC supply output failed 1 = supply output has not failed

This read-only register returns the board config and the state of the supply fails; ignore the supply fails unless there is a voltage error - the supply fail signals are informational only and may not cause a shutdown.

0x0080_4034 | cop_csr2 | byte | r*3,0*1,r/w/e*4 | non-t-chip cop i/f
| ctl/stat

bit	description
7	1 = cop i/f busy, writes to "cop_dr2" ignored 0 = cop i/f idle
6	reads 0, writes ignored
5	1 = disable automatic shift on writes to cop_dr2 for testing 0 = enable automatic shift on writes to cop_dr2 (normal mode)
4	1 = cop i/f in loopback mode, data out gets inverted and clocked back into the shift register input 0 = cop i/f in normal mode
3-0	cop chip decoder; all bits r/w/e 0x0 to 0x7 = no device selected 4'b1000: mu_bd_select = 8'h01; mb0 cop chip selected 4'b1001: mu_bd_select = 8'h02; mb1 cop chip selected 4'b1010: mu_bd_select = 8'h04; mb2 cop chip selected 4'b1011: mu_bd_select = 8'h08; mb3 cop chip selected 4'b1100: mu_bd_select = 8'h10; io cop chip selected 4'b1101: mu_bd_select = 8'h20; csb cop chip selected 4'b1110: mu_bd_select = 8'h40; mu cop chip selected 4'b1111: mu_bd_select = 8'h80; no device selected

(continued next page)

5.2 MU Registers (continued)

 0x0080_4038 | cop_dr2 | byte | r/w/u*8 | data to/from non-t-chip cops
 | bit | description

7-0 | writing to this register stores the written data,
 | sets the cop_csr2.busy bit to a 1, and begins
 | shifting the written data byte out to the cop chips
 | while simultaneously sampling and shifting in data from
 | the cop chips. Although the register may be read at
 | any time, the result is invalid until the cop_csr2.busy
 | bit negates. Once the register is written,
further writes are ignored until the busy bit negates.

0x0080_403C | led_dr | byte | r/w/u*4 | led register

This register resets to 0.

| bit | description

4 | DerBlinkenLite 1 = steady on
 | 0 = blinking ~10HZ
 3-0 | MU_LED[3:0] any bit 1 = corresponding LED on
any bit 0 = corresponding LED off

0x0080_4040 | lcd_mod_csr | byte | r1/rw*3/u*1rw*3 | lcd/MOD control
 | register

This register resets to \$00 or \$20, depending on the MOD input.

| bit | description

7 | 1 = lcd i/f busy, writes to "lcd_dr" or "lcd_csr"
 | 0 = lcd i/f idle
 6 | reserved, r/w
 |
 5 | MOD_MU_ACK, read-only, (pin name = MOD_MU_CTL)
 | 1 = MOD transfer complete
 | 0 = MOD transfer not done
 4 | MOD_MODE, Interface mode selection control
 | 1 = accessing MOD
 | 0 = accessing LCD
 3 | MU_MOD_ADDR9 (pin name = MU_MOD_CTL<3>), r/w
 | 1 = MOD addressing back boards
 | 0 = MOD addressing front boards
 2 | MU_MOD_AS (pin name = MOD_CTL<2>), r/w
 | 1 = MOD address strobe ON
 | 0 = MOD address strobe OFF
 1 | LCD_MOD_TO_MU (Bus direction, pin name = MU_FP_LCDRW)
 | 1 = read from lcd / MOD
 | 0 = write to lcd / MOD
 0 | LCD_A0_MOD_DS (pin name = MU_FP_LCDA0)
 | 1 = access lcd data reg / MOD bus data strobe ON
0 = access lcd command reg / MOD bus data strobe OFF

(continued next page)

5.2 MU Registers (continued)

 0x0080_4044 | lcd_dr | byte | r/w/u*8 | lcd data register

This register resets to 0.

| bit | description

 7-0 | Writing to this register stores data for write transfers
 | out to the LCD module or MOD display. Reads of this
 | register return either previously written data or data
 | transferred from the LCD module or MOD display.
 | This register may be read/write tested while the
 | lcd_csr.busy bit is clear; once the interface becomes
 | busy, further writes to either the lcd_dr or the lcd_cr
are ignored until the interface goes idle again.

 0x0080_4048
 to unused, writes ignored, read 0's
 0x0080_404F

address	name	size	access	description
0x0080_4050	hisr_csr	word	r/w/u*16	hard error interrupt status
0x0080_4054	hifr_csr	word	r/w/e*16	hard error interrupt force
0x0080_4058	himr_csr	word	r/w*16	hard error interrupt mask

A 1 in any bit of the hisr_csr indicates the interrupt is pending; writing 1's clears set bits and writing 0's has no effect. If a bit is cleared and the interrupt source is still asserting its interrupt line, the corresponding isr_csr bit will re-set after clearing and generate another interrupt to the processor if enabled. A 1 in any bit of the force register forces the corresponding interrupts, and a 0 in any bit of the mask register masks the corresponding interrupts.

| bit | level | description

 15-13 | 2 | unused
 12 | 2 | maui_harderr
 11-10 | 2 | io_harderror[1..0] (unit 1, unit 0)
 9 | 2 | xbar_response_harderr
 8 | 2 | xbar_request_harderr
 7-4 | 2 | agent[3..0] harderr
 3-0 | 2 | ccmc[3..0] harderr

0x0080_405C | fherr_csr | word | r/w/u*16 | first hard error

This csr attempts to indicate the first hard error received by the MU board. Since the hard errors are asynchronous, this register is implemented as resettable d-latches with the OR of all latch outputs on the latch gates. This csr is a completely combinatorial path; there are no synchronizing registers between the d-latches and the processor. The register should only be read after first determining a hard error exists by reading the hisr_csr. Once set, the fherr_csr remains set until cleared by a write to this address (the data is a don't care). The csr resets to 0.

(continued next page)

5.2 MU Registers (continued)

15-13	2	reads as 0's
12	2	maui_harderr
11-10	2	io_harderror[1..0] (unit 1, unit 0)
9	2	xbar_response_harderr
8	2	xbar_request_harderr
7-4	2	agent[3..0] harderr
3-0	2	ccmc[3..0] harderr

0x0080_4060	to	reserved for ENVMON expansion, long-word accesses don't
0x0080_407F		bus error, but reads return -1 and writes are ignored

0x0080_4080	to	reserved, access causes bus error
0x0080_4FFF		

0x0080_5000	to	BBC addresses, refer to data sheet
0x0080_51FF		

0x0080_5200	to	unimplemented address space, access causes bus error
0x00DF_FFFF		

0x00E0_0000	to	reserved for RAM expansion (to 2MB), access causes
0x00EF_FFFF		bus error

0x00F0_0000	ram	1/w/b r/w*32 general RAM
0x00FF_FFFF		

0x0100_0000	to	unimplemented address space, access causes bus error
0xFFFF_FFFF		

5.3 **MAUI Registers

MAUI register description

Note: This register space is accessible via sppedsh at base node
addr 0xf0c02XXX

address	name	size	access	description
0x00	ctl_csr	long	r/w/e*32	maui control register resets to 0x0000_0000
	bit	name	description	
31-28	reserved			
27	muri_r_pe	1 = enable muri read parity checking		
26	muri_w_pe	1 = enable muri write parity checking		
25	muro_r_pe	1 = enable muro read parity checking		
24	muro_w_pe	1 = enable muro write parity checking		

(continued next page)

5.3 MAUI Registers (continued)

bit	name	description
23-12	reserved	
11	xp0_dis	1 = disable xbar port 0 (CPU0 and CPU1)
10	xp1_dis	1 = disable xbar port 1 (CPU2 and CPU3)
9	xp2_dis	1 = disable xbar port 2 (CPU4 and CPU5)
8	xp3_dis	1 = disable xbar port 3 (CPU6 and CPU7)
7-5	reserved	r/w
4	io_psnt	1 = landmarc present, enable I/O interface 0 = disable I/O board interface
3	frc_pty[0]	1 = force bad data[31:24] (msbyte) parity on writes to MURI/MURO buffers
2	frc_pty[1]	1 = force bad data[23:16] parity on writes to MURI/MURO buffers
1	frc_pty[2]	1 = force bad data[15:08] parity on writes to MURI/MURO buffers
0	frc_pty[3]	1 = force bad data[07:00] (lsbyte) parity on writes to MURI/MURO buffers
0x04	isr_csr	long r/w u*32 maui control register
0x08	imr_csr	long r/w*32 interrupt mask register

Both csr's reset to 0x00, so interrupts reset to disabled. A 1 in the isr_csr indicates the corresponding interrupt occurred; writing a 1 to a set bit in the isr_csr clears the interrupt, and writing 0's has no effect. If a bit is cleared and the interrupt source is still asserting its interrupt line, the corresponding isr_csr bit will re-set after clearing and generate another interrupt to the processor if enabled.

A 1 in the imr_csr indicates the corresponding interrupt is enabled, a 0 indicates the interrupt is masked. Writing a 1 to a bit in imr_csr enables the interrupt, and writing a 0 masks the interrupt. Masked interrupts update the isr_csr but do not cause an MU interrupt.

bit	name	description
31-16	unimplemned	reads as 0, writes ignored
15-11	rsvd	r/w
10	sci3_errri	1 = column 3 SCI error interrupt 0 = no interrupt
9	sci2_errri	1 = column 2 SCI error interrupt 0 = no interrupt
8	sci1_errri	1 = column 1 SCI error interrupt 0 = no interrupt
7	sci0_errri	1 = column 0 SCI error interrupt 0 = no interrupt
6	io_intr[0]	1 = interrupt from Landmarc, TBD
5	io_intr[1]	1 = interrupt from Landmarc, TBD
4	toc_skip	1 = TOC missed a sync pulse 0 = no missed sync pulse
3	MURO_rsipi	1 = MURO received response_in, needs service 0 = no response_in service needed
2	MURI_rqi	1 = MURI received request_in, needs service 0 = no request_in service needed

(continued next page)

5.3 MAUI Registers (continued)

bit	name	description
1	clk_muro	1 = csr clerk needs MURO buffer 0 = csr clerk has or doesn't need it
0	clk_muri	1 = csr clerk needs MURI buffer 0 = csr clerk has or doesn't need it

0x0C | berr_csr | long | r*14,r/w*18 | Bus error source register

bit	name	description
31-24	reserved	reads 0's, writes ignored
23	muro_wpty[0]	1 = MURO write pty error on data<31..16> (msb) 0 = no parity error
22	muro_wpty[1]	1 = MURO write pty error on data<31..16> 0 = no parity error
21	muro_wpty[2]	1 = MURO write pty error on data<15..0> 0 = no parity error
20	muro_wpty[3]	1 = MURO write pty error on data<15..0> 0 = no parity error
19	muro_rpty[0]	1 = MURO read pty error on data<31..16> (msb) 0 = no parity error
18	muro_rpty[1]	1 = MURO read pty error on data<31..16> 0 = no parity error
17	muro_rpty[2]	1 = MURO read pty error on data<15..0> 0 = no parity error
16	muro_rpty[3]	1 = MURO read pty error on data<15..0> (lsb) 0 = no parity error
15	muri_wpty[0]	1 = MURI read pty error on data<31..16> (msb) 0 = no parity error
14	muri_wpty[1]	1 = MURI read pty error on data<31..16> 0 = no parity error
13	muri_wpty[2]	1 = MURI read pty error on data<15..0> 0 = no parity error
12	muri_wpty[3]	1 = MURI read pty error on data<15..0> 0 = no parity error
11	muri_rpty[0]	1 = MURI read pty error on data<31..16> (msb) 0 = no parity error
10	muri_rpty[1]	1 = MURI read pty error on data<31..16> 0 = no parity error
9	muri_rpty[2]	1 = MURI read pty error on data<15..0> 0 = no parity error
8	muri_rpty[3]	1 = MURI read pty error on data<15..0> (lsb) 0 = no parity error
7	muro_acc	1 = mu access of muro when csr clerk owned it. 0 = no error
6	muri_acc	1 = mu access of muri when csr clerk owned it. 0 = no error
5	addr_err	1 = MU attempted access to unimplemented csr 0 = no access error
4	apty_err	1 = parity error on MU address 0 = no parity error
3	dpty0_err	1 = parity error on MU data<31..24> (msb) 0 = no parity error

(continued next page)

5.3 MAUI Registers (continued)

bit	name	description
2	dpty1_err	1 = parity error on MU data<23..16> 0 = no parity error
1	dpty2_err	1 = parity error on MU data<15..8> 0 = no parity error
0	dpty3_err	1 = parity error on MU data<7..0> (lsb) 0 = no parity error

0x10 | MURI_csr | r*1,r0/wl*1,r*4,r/w*10,r*16 | MURI request_in csr

bit	name	description
31	mu_owns_buf	1 = mu owns buffer 0 = clrk owns buffer This bit set only by csr clerk when it xfers ownership to the mu; cleared only by mu when it xfers ownership back to the csr clerk by writing next bit (mu_release)
30	mu_release	reads 0 write 0 - no effect write 1 - xfers ownership to csr clerk (clearing mu_owns_buf)
29-26	rsvd	reads 0's, writes ignored
25	muri_reset	1 = mu resets muri state machines
24	mu_all_reqi	1 = mu receives all incoming requests r/w
23	mu_wants_buf	1 = mu wants buffer 0 = mu no longer wants buffer mu should clear this bit before or with bit 30
22	rspo_err_pkt	1 = outgoing response has an error pkt 0 = outgoing response does not contain error pkt
21-16	rspo_siz	size of outgoing response in 16-bit words, including error pkt if present. Range 0 to 40 (0x00 to 0x28); Writing a null response size causes the csr clerk to skip sending a response and fall thru to waiting for a request.
15-9	rsvd	reads 0, writes ignored
8	respo_sent	1 = csr clerk sent a response out set by csr clerk, cleared by writing mu_release
7	clrk_wants	1 = csr_clerk wants MURI buffer 0 = csr_clerk doesn't want MURI buffer This field is used by the csr clerk to notify mu of an incoming request while mu owns MURI.
6	rqi_err_pkt	1 = incoming response contains error packet 0 = incoming response doesn't have error packet This field gets updated when csr clerk xfers ownership to mu.
5-0	reqi_siz	size of the input request in 16-bit words, range 0 to 40 (0x00 to 0x28) This field gets updated when csr clerk xfers ownership to mu.

(continued next page)

5.3 MAUI Registers (continued)

 0x14 | MURO_csr | r*15,r/w1*1,r/w*8,r*8 | MURO ctl/stat reg

| bit | name | description

31	mu_owns_buf	1 = mu owns buffer 0 = csr clerk owns buffer This bit set only by csr clerk when it xfers ownership to the mu; cleared only by mu when it xfers ownership back to the csr clerk by writing next bit (mu_release)
30	mu_release	reads 0 write 0 - no effect write 1 - xfers ownership to csr clerk (clearing mu_owns_buf)
29-26	rsvd	reads 0's, writes ignored
25	muro_reset	1 = mu resets muro state machines
24	mu_all_reqi	1 = mu receives all incoming requests r/w
23	mu_wants_buf	1 = mu wants buffer 0 = mu no longer wants buffer mu should clear this bit before or with bit 16
22	rqo_err_pkt	1 = outgoing request has an error pkt 0 = outgoing request does not contain error pkt This bit set/cleared by mu when it formats rqo.
21-16	rqo_siz	size of outgoing request in 16-bit words, including error pkt if present. Range 0 to 40 (0x00 to 0x28). Writing a null request size causes the csr clerk to skip sending a request and fall thru to waiting for a response.
15-9	rsvd	reads 0's, writes ignored
8	reqo_sent	1 = csr clerk sent a request out
7	clrk_wants	1 = csr_clerk wants MURO buffer 0 = csr_clerk doesn't want MURO buffer This field is used by the csr clerk to notify mu of an incoming response while the mu owns MURO.
6	rsp_i_err_pkt	1 = incoming response has an error pkt 0 = incoming response does not contain error pkt This field gets updated when csr clerk xfers ownership to mu.
5-0	rsp_i_siz	size of incoming request in 16-bit words, including error pkt if present. Range 0 to 40 (0x00 to 0x28). A null request size indicates mu got buffer back before a response arrived. This field gets updated when csr clerk xfers ownership to mu.

 0x18 | config_csr | long | r/w/e*32 | maui configuration register

All bits reset to 0 EXCEPT virtual_ring_map[].

| bit | name | description

31-27	xb_err_config	xbar harderror enables [0:4] any 0 = corresponding hard error disabled any 1 = corresponding hard error enabled
-------	---------------	---

(continued next page)

5.3 MAUI Registers (continued)

bit	name	description
26	reserved	r/w
25-24	active_rings	active_rings[1:0]; bit 0 is really the MSB. [1:0] 00 = 4 active rings 01 = 2 active rings 10 = 1 active ring 11 = 3 active rings
23-16	virt_ring_map	virtual_ring_map[7:0] - resets to 0xD8 (virtual = physical) virtual ring # 3 2 1 0 cfg_csr 23 22 21 20 19 18 17 16 vir_ring_map [7 6] [5 4] [3 2] [1 0] default 1 1 0 1 1 0 0 0 for each 2-bit field, 00 = physical ring #0 01 = physical ring #2 10 = physical ring #1 11 = physical ring #3
15-13	reserved	r/w
12-8	lnd_ocya[4:0]	for gary stager
7	proc_avl[7]	1 = odd processor on xbar port 3 available
6	proc_avl[6]	1 = even processor on xbar port 3 available
5	proc_avl[5]	1 = odd processor on xbar port 2 available
4	proc_avl[4]	1 = even processor on xbar port 2 available
3	proc_avl[3]	1 = odd processor on xbar port 1 available
2	proc_avl[2]	1 = even processor on xbar port 1 available
1	proc_avl[1]	1 = odd processor on xbar port 0 available
0	proc_avl[0]	1 = even processor on xbar port 0 available

0x1C | hen_csr | long | r/w*32 | Hard error enable csr
0x20 | herr_csr | long | r*32 | Hard error csr

1's in the hen_csr enable holding on a particular hard error at the detection point. All bits in this register may be read or written at any time (of course, you could cause a hard-error when you do if one is pending...). Note that the hen_csr only enables HOLDING on a hard error, an active error will still be seen in the sticky herr_csr even if the enable is clear.

1's in the read-only herr_csr indicate the corresponding error occurred. If the corresponding hen_csr bit is clear, the MAUI hard error line will not assert; if the corresponding hen_csr bit is set, the MAUI hard error line should assert; and if the JTAG clock gating scan-bit is asserted, the chip will lockup and this interface is toast.

Writing a '1' to the hen_csr will both enable corresponding herr_csr errors AND clear any existing errors out of the herr_csr (sticky-bit) as long as the error is not currently active.

(continued next page)

5.3 MAUI Registers (continued)

MAUI bit	REAL bit	name	description
0	31	qi_parity	1 = xbar request-in port incoming parity error
1	30	qi_sequence	1 = xbar request-in port invalid tag sequence
2	29	qi_route	1 = xbar request-in port invalid routing word
3	28	reserved	r/w
4	27	si_parity	1 = xbar response-in port incoming parity error
5	26	si_sequence	1 = xbar response-in port invalid tag sequence
6	25	si_route	1 = xbar response-in port invalid routing word
7	24	reserved	r/w
8	23	cere_out_pty	1 = cerebus output path parity/error
9-11	22-20	reserved	r/w
12	19	ebin_w_pe	1 = elastic-buffer-in write parity enable/error
13	18	ebin_r_pe	1 = elastic-buffer-in read parity enable/error
14	17	ebot_w_pe	1 = elastic-buffer-out write parity enable/error
15	16	ebot_r_pe	1 = elastic-buffer-out read parity enable/error
16	15	muri_w_pe	1 = clerk muri write port parity enable/error
17	14	muri_r_pe	1 = clerk muri read port parity enable/error
18	13	muro_w_pe	1 = clerk muro write port parity enable/error
19	12	muro_r_pe	1 = clerk muro read port parity enable/error
20-31	11-0	reserved	r/w
0x24 to 0x7F	rsvd	long	r*32 unimplemented
0x100 0x11F	MURI message buffer	8 x r/w*32	MURI request_in/response_out message buffer
0x120 0x13F	MURO message buffer	8 x r/w*32	MURO request_out/response_in message buffer

5.4 **MU Status Numbers

There are times when the MU will detect a failure, display an error number on the LCD panel and not put an entry in the event_log. You can use the following table to translate a LCD displayed error number to a message.

* status macros	

MU_REQUEST_DONE(status)	(status & MU_STATUS_DONE_MASK)
MU_STATUS_DONE_MASK	0x0080

(continued next page)

5.4 MU Status Numbers (continued)

*-----	
* CXRing configuration status values	
*-----	
MU_STATUS_CXC_TIMEOUT_2	0x1091
MU_STATUS_CXC_TIMEOUT_1	0x1090
*-----	
* console status values	
*-----	
MU_STATUS_CONSOLE_MAX_OPEN	0x0f94
MU_STATUS_CONSOLE_WR_OVERRUN	0x0f93
MU_STATUS_CONSOLE_OPEN_ID	0x0f92
MU_STATUS_CONSOLE_RD_OVERRUN	0x0f91
MU_STATUS_CONSOLE_BAD_ID	0x0f90
*-----	
* memory allocation status values	
*-----	
MU_STATUS_MALLOC_NO_MEM	0x0e90
*-----	
* trace status values	
*-----	
MU_STATUS_TRACE_BAD_PAGE	0x0d91
MU_STATUS_TRACE_BAD_TYPE	0x0d90
*-----	
* reset status values	
*-----	
MU_STATUS_RESET_BUSY	(0x0c00 + MU_STATUS_BUSY)
MU_STATUS_RESET_BAD_LEVEL	0x0c90
*-----	
* COP status values	
*-----	
MU_STATUS_COP_BUSY	(0x0b00 + MU_STATUS_BUSY)
MU_STATUS_COP_BAD_ID	0x0b91
MU_STATUS_COP_TIMEOUT	0x0b90
*-----	
* clock control status values	
*-----	
MU_STATUS_CLOCKS_BAD_CSB_TYPE	0x0a96
MU_STATUS_CLOCKS_MODE_SET	0x0a95
MU_STATUS_CLOCKS_BAD_MODE	0x0a94
MU_STATUS_CLOCKS_SOURCE_SET	0x0a93
MU_STATUS_CLOCKS_BAD_SOURCE	0x0a92
MU_STATUS_CLOCKS_ENABLED	0x0a91
MU_STATUS_CLOCKS_DISABLED	0x0a90
(continued next page)	

5.4 MU Status Numbers (continued)

*-----
 * power and environment status values
 *-----

MU_STATUS_PNE_BUSY (0x0900 + MU_STATUS_BUSY)

MU_STATUS_PNE_PWR_BAD_CONFIG	0x09b1
MU_STATUS_PNE_PWR_VDL_AND_TCHIPS	0x09b0
MU_STATUS_PNE_WEAK_48V	0x099a
MU_STATUS_PNE_PWR_SHUTDOWN_HI	0x0999
MU_STATUS_PNE_PWR_SHUTDOWN_LO	0x0998
MU_STATUS_PNE_PWR_NO_SUPPLY	0x0997
MU_STATUS_PNE_PWR_OFF_ABORT	0x0996
MU_STATUS_PNE_PWR_ADJUST_TIMEOUT	0x0995
MU_STATUS_PNE_PWR_BAD_CALIBRATE	0x0994
MU_STATUS_PNE_PWR_MEASURE_TIMEOUT	0x0993
MU_STATUS_PNE_PWR_ON_TIMEOUT	0x0992
MU_STATUS_PNE_TRIM_TIMEOUT	0x0991
MU_STATUS_PNE_BAD_CHANNEL	0x0990

*-----
 * script status values
 *-----

MU_STATUS_SCRIPT_BUSY (0x0800 + MU_STATUS_BUSY)

MU_STATUS_SCRIPT_BAD_OPCODE	0x0892
MU_STATUS_SCRIPT_STACK_UNDERFLOW	0x0891
MU_STATUS_SCRIPT_STACK_OVERFLOW	0x0890

*-----
 * configuration status values
 *-----

MU_STATUS_CONFIG_BAD_VALUE	0x0792
MU_STATUS_NODE_ID_CONFLICT	0x0791
MU_STATUS_CONFIG_BAD_NODE_ID	0x0790

*-----
 * NVRAM status values
 *-----

MU_STATUS_NVRAM_BUSY (0x0600 + MU_STATUS_BUSY)

MU_STATUS_NVRAM_ZERO_CHECKSUM	0x0696
MU_STATUS_NVRAM_BAD_SL_CHECKSUM	0x0695
MU_STATUS_NVRAM_BAD_PL_CHECKSUM	0x0694
MU_STATUS_NVRAM_BAD_FW_CHECKSUM	0x0693
MU_STATUS_NVRAM_BAD_KEY	0x0692
MU_STATUS_NVRAM_FULL	0x0691
MU_STATUS_NVRAM_VAR_NOT_FOUND	0x0690

(continued next page)

5.4 MU Status Numbers (continued)

*-----
 * EEPROM load status values
 *-----

MU_STATUS_EEPROM_BUSY	(0x0500 + MU_STATUS_BUSY)
MU_STATUS_EEPROM_WRITE_FAILED	0x0592
MU_STATUS_EEPROM_NOT_IN_USE	0x0591
MU_STATUS_EEPROM_IN_USE	0x0590

*-----
 * scan status values
 *-----

MU_STATUS_SCAN_BUSY	(0x0400 + MU_STATUS_BUSY)
MU_STATUS_SCAN_BAD_ID	0x0496
MU_STATUS_SCAN_IN_USE	0x0495
MU_STATUS_SCAN_NOT_OWNER	0x0494
MU_STATUS_SCAN_ERROR	0x0493
MU_STATUS_SCAN_BAD_RING	0x0492
MU_STATUS_SCAN_BAD_END_STATE	0x0491
MU_STATUS_SCAN_BAD_COMMAND	0x0490
MU_STATUS_SCAN_RW_SHORT	0x0433
MU_STATUS_SCAN_RW	0x0432
MU_STATUS_SCAN_RW_START	0x0431
MU_STATUS_SCAN_WRITE	0x0421
MU_STATUS_SCAN_READ	0x0412
MU_STATUS_SCAN_READ_START	0x0411
MU_STATUS_SCAN_START	0x0401

*-----
 * MAUI status values
 *-----

MU_STATUS_MAUI_BUSY	(0x0300 + MU_STATUS_BUSY)
MU_STATUS_MAUI_REMOTE_MASK	0x1000
MU_STATUS_MAUI_UNAVAILABLE	0x03a6
MU_STATUS_MAUI_MSG_TIMEOUT	0x03a5
MU_STATUS_MAUI_MSG_TOO_LARGE	0x03a4
MU_STATUS_MAUI_NO_STATUS	0x03a3
MU_STATUS_MAUI_BAD_CHANNEL	0x03a2
MU_STATUS_MAUI_CHANNEL_IN_USE	0x03a1
MU_STATUS_MAUI_NO_CHANNELS	0x03a0
MU_STATUS_MAUI_TIMEOUT	0x0391
MU_STATUS_MAUI_CXBAR_ERROR	0x0390
MU_STATUS_MAUI_INCOMING_MSG	0x0310
MU_STATUS_MAUI_AWAIT_SCRUB_TAG_RES	0x0307
MU_STATUS_MAUI_AWAIT_SCRUB_DATA_RES	0x0306
MU_STATUS_MAUI_AWAIT_CXBAR_RES	0x0305
MU_STATUS_MAUI_AWAIT_NWR_RES	0x0304
MU_STATUS_MAUI_AWAIT_NRD_RES_64	0x0303
MU_STATUS_MAUI_AWAIT_NRD_RES_SB	0x0302
MU_STATUS_MAUI_UNUSED	0x0301

(continued next page)

5.4 MU Status Numbers (continued)

```

*-----
*  DaRT status values
*-----

    MU_STATUS_DART_BUSY           (0x0200 + MU_STATUS_BUSY)

    MU_STATUS_DART_UNAVAILABLE    0x0293
    MU_STATUS_DART_BAD_NODE       0x0292
    MU_STATUS_DART_TIMEOUT        0x0291
    MU_STATUS_DART_TRUNCATED      0x0290

*-----
*  SONIC status values
*-----

    MU_STATUS_SONIC_SEND_QUEUE    0x0103
    MU_STATUS_SONIC_RECEIVED      0x0102
    MU_STATUS_SONIC_UNUSED        0x0101

*-----
*  generic status values
*-----

    MU_STATUS_BAD_ADDR            0x00ff
    MU_STATUS_BAD_SIZE            0x00fe
    MU_STATUS_ZERO_SIZE           0x00fd
    MU_STATUS_BAD_TYPE            0x00fc
    MU_STATUS_SHORT_MSG           0x00fb
    MU_STATUS_BUSY                0x00fa
    MU_STATUS_PANIC               0x00f9
    MU_STATUS_SUBTEST_FAIL        0x00f8
    MU_STATUS_BUS_ERROR           0x00f7
    MU_STATUS_NO_BUS_ERROR        0x00f6
    MU_STATUS_LIMIT_EXCEEDED      0x00f5
    MU_STATUS_REQ_IGNORED         0x00f4

*-----
*  subtest status values
*-----

    MU_STATUS_ST_COP_ERROR        0x00d6
    MU_STATUS_ST_TIMEOUT          0x00d5
    MU_STATUS_ST_BYTE_SELECT_ERROR 0x00d4
    MU_STATUS_ST_ACCESS_ERROR     0x00d3
    MU_STATUS_ST_PARITY_ERROR     0x00d2
    MU_STATUS_ST_ADDRESS_ERROR    0x00d1
    MU_STATUS_ST_PATTERN_ERROR    0x00d0

*-----
*  miscellaneous status values
*-----

    MU_STATUS_OK                  0x0080
    MU_STATUS_WRITE                0x0004
    MU_STATUS_READ                 0x0003
    MU_STATUS_IN_USE               0x0002
    MU_STATUS_UNUSED               0x0001

```

5.5 **Disabling Nodes and Slices when Trouble-Shooting

To disable a slice, use xconfig and de-select the particular slice in each node. Do a download. Do a do_reset 3. Continue trouble-shooting. After the problem has been resolved, use ccmu to restore the original parameters.

```
ccmu> pull
ccmu> quit
do_reset 3
```

To disable a node, use ccmu.

```
ccmu> up
ccmu> select
ccmu> clear <node#> n
ccmu> auto
ccmu> down
ccmu> quit
```

Do a do_reset 3. Continue trouble-shooting.
After the problem has been resolved, use ccmu to restore the original parameters.

```
ccmu> pull
ccmu> quit
do_reset 3
```

5.6 **Single-Node False "SCI incoming parity err"

On single-node (No SCI chips) SPP systems, the "SCI incoming parity err" bit is always set. This error bit is masked in the CMC_ERR_ENO parameter and will not cause a harderror. However, if another error bit in the CCMC is set, the "SCI incoming parity err" will be decoded and reported in the event_log. This error bit should be ignored. In the example below both the "SCI incoming parity err" bit and the "single-bit ECC" error bit are set.

The actual cause of the error is the "single-bit ECC" error.

```
+++> 297
<Fri Nov 18 19:31:30 1994> warning:0x67600400
<MU FW>:2.0.1.0:herr.c:743
node: 0x0 (0)
Soft Error received from CCMC 0
CAUSE: 0x02000020
CMC type is dark
0x02000000 - SCI incoming parity err
0x00000020 - single-bit ECC
ECC_ERR: 0x80030490, ECC_ADR: 0x20fc6a40, CONFIG: 0x00030004
****
```

siehe Seite 3-8

Node
SCI
incoming
SCI_PAR-ERR

5.7 **Identifying Gate Array Locations

The cst utility will generate failures using fixed location coordinates, to report a component failure. An example of this is shown below:

```
Wire XQMC1PAR0 (ring 0 -> ring 3)
Driver:  XBAR Z073K8.AP39          ODP1_0 bit -255 Hi
Recver:  CCMC2 Z131F5.J6          XB_CM_RQI_DPAR_0 bit 415 Lo
Pattern 0008 - ac_pat.008 FAILED 1 error
```

From this information, it is difficult to know which CCMC, or Xbar is involved, from the location Z073K8. To aid in the identification, the location can be determined by a couple of simple methods.

First of all the slice is indicated by the last two digits. These will be as follow:

```
slice 0  is C0
slice 1  is F5
slice 2  is P4
slice 3  is T9
```

The XBQ, XBS and Maui are located at K8.

The first 4 digits indicate the actual component, as follow:

```
Agent  is Z058
CCMC   is Z131
SCI    is Z163
XBQ    is Z073
XBS    is Z102
Maui   is Z131
```

So from this it is possible to determine, from the info in the error, that the failing CCMC is in slice 1 and the Xbar device is the XBQ, or xbar request gate array.

5.8 **Node Temperature Monitoring

There are 6 temperature sensors in a SPP1000 node and 5 temperature sensors in a SPP1200 node. The MU monitors for both over and under temperature situations. The MU will log warnings, in the event_log, at 98F and 64F. The MU will power off the node at 107F and 53F.

The example below shows an over temperature warning and a power off. Notice the message states it is an "over-voltage" condition. Further investigation reveals the error was detected by "power 0 temp".

```
+++> 290
<Mon Nov 14 21:03:53 1994> warning:0x676c8234
<MU FW>:2.0.1.0:pne.c:1276
node: 0x0 (0)
power supply channel 0x34 (power 0 temp) is in over-voltage WARNING
shut_up: 107  warn_up: 98  nom: 81  warn_dn: 64  shut_dn: 53
last eight readings: 100 100 100 100 100 100 100 99
****
```

(continued next page)

5.8 Node Temperature Monitoring (continued)

```
+++> 296
<Mon Nov 14 21:07:20 1994> critical:0xa76c8434
<MU FW>:2.0.1.0:pne.c:1276
node: 0x0 (0)
power supply channel 0x34 (power 0 temp) TURNED OFF due to over-voltage
shut_up: 107 warn_up: 98 nom: 81 warn_dn: 64 shut_dn: 53
last eight readings: 108 108 108 108 108 108 108 108
****
```

```
+++> 176
<Mon Nov 14 21:07:20 1994> info:0x47601200
<MU FW>:2.0.1.0:node_init.c:2343
node: 0x0 (0)
node disabled, cause 1
cause is defined as MU_NODE_DISABLE_POWER_OFF
****
```

```
+++> 880
<Mon Nov 14 21:07:20 1994> info:0x476c8600
<MU FW>:2.0.1.0:pne.c:1410
node: 0x0 (0)
power turned off in the following order:
bit mask: 0x00000002 voltage: +3.3 (gate array)
bit mask: 0x00004000 voltage: +5.0 IO
bit mask: 0x00000040 voltage: +5.0 memory board 0
bit mask: 0x00000080 voltage: +5.0 memory board 1
bit mask: 0x00000100 voltage: +5.0 memory board 2
bit mask: 0x00000200 voltage: +5.0 memory board 3
bit mask: 0x00001000 voltage: -3.1 processor module
bit mask: 0x00000800 voltage: +2.0 processor module
bit mask: 0x00000400 voltage: +5.0 processor module
bit mask: 0x00000020 voltage: -2.0 SCI unreferenced
bit mask: 0x00008000 voltage: -2.0 SCI
bit mask: 0x00000004 voltage: -2.0 Non-SCI
bit mask: 0x00002000 voltage: -4.5 clock logic
bit mask: 0x00000010 voltage: +1.2 SCI
bit mask: 0x00000001 voltage: +1.2 gate array
****
```

6.1 **OBP (Open Boot Prom) What is it?

Open Boot Prom (OBP) is a software architecture for boot firmware that Convex purchased from SunSoft.

The software Convex got from Sun was only for SPARC workstations and SPARC clones. Convex ported it to PA-RISC and the HP700 and SPP1 hardware. The architecture remains the same, however there are many changes in the specifics and some of the functionality present in the SPARC OBP 2.8 release is not implemented in R 1.0.0.0 of Exemplar OBP. (For example, Exemplar OBP has no virtual memory system to handle page mapping like the SPARC release. This is used by Sun for Standalone Diagnostics and other test programs written in C and assembly language, and Convex uses the test station for this function.)

Some OBP features were not ported, others were added specifically for Convex machines, and for multi-node, even more features will be added. The point of this disclaimer is that if you have a problem with OBP, let us know about it via Cobra. Since OBP is in EEPROM and can easily be patched, and by design is COMPLETELY separately releasable from the kernel and other system firmware, it is possible to get fixes done quickly, provided we know about them. Wishes are another matter. :-)

SECONDARY LOADER vs PRIMARY LOADER terminology

OBP is a software architecture for the firmware that controls a computer before the operating system has begun execution. It runs entirely in the CPU(s) of the computer - not in the MU or the teststation.

Though in SUN's machines OBP is used as the primary loader and therefore performs Power-On Self-Tests, we have no POST in our version of OBP.

That code is called the Primary-Loader and is released by the Diagnostics development group. OBP, our secondary loader, is released by the OS development group. (However it is installed in EEPROM on the mu just as the Primary Loader, only in a separate 508K section reserved to OBP.)

There are three main areas of functionality that OBP provides to the SPP:

1. Autoconfiguration - By far the most important feature of OBP is that it is designed to completely autoconfigure the I/O subsystem and to provide to the OS a complete and ACCURATE description of all the hardware available to the OS. It stores this information in a hierarchical data structure called the 'Device Tree' and provides C-callable interface to the OS kernel to query the database. (OBP remains resident within the first 1M of physical RAM within the OS kernel as a 'firmware subroutine library') Because the architecture of OBP is designed to be machine-independent and trivially extensible to new I/O busses and I/O bridges without ever changing its C interface, the problem of autoconfiguration of OS drivers of completely new HW is now 'SOLVED'.

(continued next page)

6.1 OBP (Open Boot Prom) What is it? (continued)

2. Boot the OS - This is by far the simplest but essential function of OBP. In the SPP, booting the OS is complicated by a couple of factors compared to SPARC OBP:
 - a. We must load 3 different images into memory before starting. (kernel, server+emulator, tunables file)
 - b. We need to be compatible with HFS, so OBP must load actual files off of the root partition, unlike SPARC OBP which simply reads the first 15 blocks off the boot device and 'executes' it as a standalone program. This program is Sun's Secondary Loader and contains the filesystem code to load their /vmunix and drivers
 - c. Multi-node systems will still only have 1 root disk (However R1.0.0.0 of OBP does not support multinode booting so you can look forward to more OBP info when it does.)
3. Configuration - A system as complex as SPP has three kinds of Parameters configuration information that must be non-volatile.

Low-Level HW configuration - this set of parameters is managed by diagnostics firmware in the MU and primary loader and utilities running on the test-station

OS Boot configuration parameters - These are managed by OBP, and are called NVRAM parameters. They cannot be accessed/changed by diagnostics, but only by OBP's user interface and the kernel's C interface to OBP. (The auto-boot? flag and boot-device are two such NVRAM parameters)

Mach and Unix configuration parameters - These are stored in the ascii 'tunables' file, which OBP loads into the OBP device tree for access by the Mach kernel. (This implementation allowed the kernel developers to completely remove the HFS filesystem code from the Mach Kernel, significantly simplifying the bootstrap of the already complicated process of booting a multi-kernel OS such as SPP/UX)

Note: Please see the article titled "OBP (Open Boot Prom) Configuration Parameters" for definitions of the individual parameters and their defaults.

6.2 **OBP Configuration Parameters

This section is going to attempt to describe the OBP configuration parameters that are present in R 1.2.3 of OBP. There are probably some parameters that will go away and possibly new ones to be defined in future versions. Furthermore, the default values of one or more parameters will likely change.

The four commands for setting/displaying OBP configuration parameters are:

`printenv [parameter-name]`

with no name, lists the entire set of parameters along with the current and default values of the parameter. If the parameter name is supplied, only the named parameter is displayed.

`setenv parameter-name parameter-value`

This is used to change the current value of a parameter. The change is 'non-volatile' and will be used next time OBP boots. The Default Value cannot be changed by the user. It is a compile-time value. When OBP is compiled, all the 'current' values are set to their defaults. Since both OBP and its NVRAM are implemented in the same EEPROM, it was simple to do this at compile time.

`set-default parameter-name`

This restores the current value of an NVRAM configuration variable to the Default Value. It is the normal way to set a string-valued property to the 'null-string'.

`set-defaults`

This will do a 'set-default' for ALL variables. It takes about 1 minute to execute, so if you do this manually, prepare to wait a bit. Normally each parameter takes only about .5 seconds to update, but it seems that doing all of them in a batch is slower - possibly because of the fact that OBP is sending a message to the MU for each parameter, and the MU is 'throttling' OBP because it is busy monitoring power, etc.

If one lists the attributes of the /options device, you will see all the current values of the NVRAM configuration parameters.

The printenv output from OBP Release 1.2.3 is:

```
[0:2] ok printenv
Parameter Name      Current Value      Default Value
stdcon-device       /console@1,0:log
output-device       /console@1,0
input-device        /console@1,0
u0sb0
u0sb1
u0sb2
u0sb3
u1sb0
u1sb0
```

(continued next page)

6.2 OBP Configuration Parameters (continued)

Parameter Name	Value	Default Value
ulsb1		
ulsb2		
ulsb3		
sbus-probe-list	U1SB3 U1SB2 U1SB1 U1SB0..	U1SB3 U1SB2 U1SB1 U1SB0..
fcode-debug?	false	false
local-mac-address?	false <i>true</i>	false <i>true</i>
tz	CST6CDT,91/0300,301/0100	CST6CDT,91/0300,301/0100
diag-device	none	none
diag-file	none	none
boot-device	sd0a	sd0a
boot-directory	/os <i>/stand/spp1</i>	/os
boot-file	mach	mach
boot-args		
pvmcti-memory-size	0	0
cluster-boot?	true <i>false</i>	true
load-tunables?	true	true
auto-boot?	false	false
watchdog-reboot?	false	false
screen-#columns	80	80
screen-#rows	24	24
internet-id	0.0.0.0	<no default>
system-mac-address		<no default>
unit-map	00 3f 00 00 00 00 00 00	<no default>
use-nvramrc?	false <i>true</i>	false
nvramrc		
security-mode	none	none
security-password		
security-#badlogins	0	<no default>
last-hardware-update		<no default>
testarea	0	0
mfg-switch?	false	false
diag-switch?	false	false

(continued next page)

6.2 OBP Configuration Parameters (continued)

Here is an explanation of the parameters in the above table, sometimes as one of a group of parameters.

log-device
output-device
input-device

These three string-value parameters contain the OBP device tree pathname for the console device used by OBP when it boots. This release of OBP can only use /console for the console device. The log-device is the device that OBP uses for log events. OBP 1.0.0.0 is ignoring the value of log-device, and always sends the log-events to the MU which forwards them to the event-log daemon on the test station.
Do not change these values.

u0sb0 u0sb1 u0sb2 u0sb3
ulsb0 ulsb1 ulsb2 ulsb3

These are parameters that allow OBP to compile a different FCode device driver that is built into OBP (called a drop-in driver) instead of the one in the controller's ROM.

The current Landmarc does not implement sb2 or sb3, but these variables are defined in anticipation of that hw being built.

The default value for each of these is the null-string. A non-null value for one of these must be the name of an FCode driver compiled into OBP. It will select the driver that OBP compiles as it probes that SBus slot (assuming a controller actually exists in the slot).

Since all new machines are supposed to be shipped with Convex ROMS and OBP will probe all implemented SBus slots automatically, there should be no reason for you to change these nvram parameters.

However, two reasons that you would need to change them are:

1. The FCode ROM in one or more SCSI controllers has NOT been upgraded or is defective.
2. You are booting a machine with this OBP but which has a Landfill I/O controller.

The three drivers built into OBP that can be used for these parameters are:

afws-fcode - differential Landmarc-only SCSI driver
se-afws-fcode - single-ended Landmarc-only SCSI driver

When all hw is up-revved, these drop-in drivers will be removed from future OBP releases.

Details about using these parameters are covered in the article: "Configuring Non-Standard or CD Hypernodes".

(continued next page)

6.2 OBP Configuration Parameters (continued)

sbus-probe-list

This string-valued property is a list of SBus controllers that OBP will probe. The default list is all four controllers:

U1SB3 U1SB2 U1SB1 U1SB0 U0SB3 U0SB2 U0SB1 U0SB0

OBP can tell when a controller is actually plugged in or not.

OBP will probe the SBus slots named in left-to-right order, but will not alternate Landmarc units. That is, all slots of the first Landmarc unit named in the list will be probed in left-to-right list order.

The default list is in decreasing order because the devices created in the device tree are listed out in most-recently-created order by the show-devs command.

fcode-debug?

This boolean property is used during development of FCode drivers. It should be left 'false'.

local-mac-address?

This boolean property is currently not used. It is used to tell the Ethernet FCode driver in an SBus ethernet controller to get the ethernet HW address from its own property list or any parent device property list, instead of using the global 'system-mac-address'. It currently has no function.

tz

This is a partially implemented idea. It was intended to allow OBP to know what timezone it is in. The value is supposed to be the same as the TZ environment variable for SPP/UX (HP/UX).

OBP has the clock/calendar chip device driver for Mach. So when the user executes a date command, it causes the server to send a message to mach and mach to call OBP to set the date and time in the date chip. Since HP/UX (and SPP/UX) keep internal time in seconds since the epoch, and corrections are applied externally by utilities, OBP has no way of really knowing what local time is. Convex had hoped to use this variable to allow OBP to have the same 'environment' for time/date as SPP/UX.

If anyone thinks this is an important or useful extension to OBP, worthy of full implementation, please log a Cobra bug. Otherwise it may languish, incomplete for a long time. It currently has no function.

(continued next page)

6.2 OBP Configuration Parameters (continued)

diag-device diag-file

These two parameters are used in place of the parameters boot-device and boot-file when the parameter diag-switch? (see below) is set to true.

Since Convex has no released diagnostic standalone programs booted by OBP at this time, the default values for both these parameters are 'none' and are therefore not actually bootable, even if you set the diag-switch? to true.

boot-device
boot-directory
boot-file
boot-args

These parameters are the ones that are most important to the typical administrator of the system. Actually only the boot-device, boot-directory and boot-args are the most important, since the default values for boot-file, server-file and tunables-file should never or rarely be changed.

boot-device names an OBP device-pathname or an alias for one that is used when the user types the short version of the 'load' or 'boot' commands. The default value is 'sd0a' which is the alias for the root device. OBP will attempt to open the named partition if the disk is correctly labeled, then it will load the file specified by 'boot-file' from the directory 'boot-directory' into memory, and setup the cpu state variables for starting the boot-file.

The boot-file is loaded into RAM according to the a.out header which should have been linked so that the exec_tmam address is 0x00100000, directly above OBP (OBP resides in address 0x1000-0x000ffff, mach resides initially in addresses 0x00100000-0x003ffff)

If the named device is a SCSI tape device ('st'), a different rule is used. the boot-directory is ignored. The named file, or file 0 by default, is loaded into ram at 0x00100000, and OBP sets up the cpu state for entry at address 0x00100000. The virgin install utility is booted from tape.

cluster-boot?

This boolean valued parameter is used to alter the behavior of the OBP system console.

If it is true, each hypernode will open a separate X-term for the system console. If it is false, they will all use the same X-term.

load-tunables?

By default, load-tunables? is true. If one wanted to boot the 'boot-file' from disk, without a tunables or server file loaded, (i.e. as with a standalone diagnostic program), you would set these to false.

(continued next page)

6.2 OBP Configuration Parameters (continued)

auto-boot?

This boolean valued variable determines whether OBP will automatically try to reboot the OS after a reset. If it is true, OBP will probe the I/O system then it will boot the hypernode as though the user simply typed 'boot' at the ok prompt. The NVRAM values of boot-device, boot-directory, boot-file, server-file, tunables-file, load-tunables?, load-server? and diag-switch? determine what program is booted.

The default values will boot SPP/UX from the root disk partition.

After OBP probes the I/O system, the user may interrupt the auto-boot process, by pressing/holding the ESCAPE key at the right moment. OBP must 'poll' the keyboard, as interrupts are not used in OBP device drivers, and the polling window is brief, only a 10 seconds. The message

To interrupt autoboot, press and hold the ESCAPE key.

is printed just after the first banner 'OBP Hard Boot' when the polling window begins. You must be quick to interrupt autoboot.

watchdog-reboot?

This is a boolean flag that is true if the user wants to have OBP do an automatic reboot after a watchdog reset. This release of OBP has no watchdog timer, so the flag is ignored and has no function. Ignore it.

screen-#columns screen-#rows

These parameters tell OBP what the width and height of its screen is. OBP assumes it is using a 'dumb-terminal' though ours is actually an X-term on the test-station. For commands that do many lines of screen output, such as show-devs, dump (memory), and dis (disassemble machine code), OBP has sort of a built-in 'more' filter, and the screen-#rows parameter is used to control when to pause output.

use-nvramrc?

This boolean variable determines whether or not OBP will attempt to execute any commands in the nvramrc script 'file' during the startup of OBP. If true, the default, it will do so. The script is executed BEFORE the Landmarc part of the device tree is built, so no commands can refer to any I/O devices that will not have yet been created, (such as creating a property in a device node) It should be left true.

(continued next page)

6.2 OBP Configuration Parameters (continued)

nvrामrc

This property is the OBP script 'file' that is executed when use-nvrामrc? is true (see above). It is NOT modified with setenv, but instead one edits the contents of nvrामrc directly with the builtin OBP line editor using the command 'nvedit'. After editing the script file (exit with ^C), it is saved by using the command 'nvstore'.

The nvrामrc script can be executed at any time (at the ok prompt, that is) with the command 'nvrun'.

The default value of nvrामrc is empty (zero-length file). The maximum size of nvrामrc depends upon how much NVRAM is used by the rest of the NVRAM parameters. 16K is reserved for everything; nvrामrc gets all the space at the end of the 16K not used by the other parameters.

security-mode	- enum value (none, command, full)
security-password	- hidden char[8] value
security-#badlogins	- integer value

These parameters are related to the security feature of OBP.

The security feature allows the system to be configured so that a password is required to access most commands. Three access protection modes are provided:

- o Non-secure mode (security-mode = none) does not require a password for any command.
- o Command secure mode (security-mode = command) requires a password to execute any command except for go and boot with no arguments (i.e., boot from the default device and file). Automatic booting (after power-on) is allowed.
- o Fully-secure mode (security-mode = full) requires a password for any command except go. Automatic booting is disabled (so the machine will not automatically reboot after a power-failure).

The security level is set with 'setenv security-mode'. The password, (a printable ASCII string of up to 8 characters) is set or changed with the password command, which prompts for the new password twice, without echoing the password. The new password takes effect immediately, i.e., the next time that a password is required.

The first attempt to execute a command requiring the password causes the command interpreter to prompt for the password, without echoing, before executing the command. The command interpreter will not ask for the password again until it (the interpreter) is re-entered, e.g., after an abort sequence, reset or exit from a standalone program). Entry of an incorrect password increments the NVRAM variable security-#badlogins.

Note: To prevent the password from being echoed or displayed in a command history, it is strongly recommended that this variable be set with the password command rather than the setenv command.

(continued next page)

6.2 OBP Configuration Parameters (continued)

last-hardware-update

This string-valued variable is a convenient place to record the last time the machine configuration was updated. Since Exemplar has a much more sophisticated system for tracking these changes, there is doubt this parameter is required. Convex might consider using it instead to save the last time OBP itself was updated, in which case the probably name will probably change.

OBP ignores this parameter - it is solely for human consumption.

mfg-switch?

This boolean variable is set to true if the NVRAM becomes corrupted, in which case the next reboot of OBP will cause all NVRAM parameters to be reset to their defaults.

diag-switch?

This boolean variable causes the internal OBP flag 'diagnostic-mode?' to be set to true, which will cause booting from 'diag-device' instead of boot-device, and diag-file instead of boot-file. It also has the sometimes useful side-effect of printing out the names of new devices as they are being compiled into the device tree at I/O system probe time.

6.3 **Configuring Non-Standard or CD Hypernodes

This information is for people using the new Release 1.0.0.0 of OBP on non-standard I/O hardware. It is believed that this release of OBP will work with ANY hardware configuration we've shipped, and I've tested it on more than 4 different configurations, but for non-standard configurations varying amounts of initial setup of NVRAM parameters is going to be needed.

Standard is defined as having the following:

1. Landmarc I/O controller (rev C or later)
2. SCSI controller(s) with Convex ROMs

Landfill systems or Landmarc and Landfill systems with old or bad SCSI ROMs need manual configuration. Here is an explanation of what is needed and examples of the setup required.

Configuring OBP on Landmarc systems that don't have the right SCSI FCode ROMs (Convex ROMs) in them, but instead have the original Antares SPARC ROMs.

For each such controller, the driver 'afws-fcode' must be set as the value of the controller NVRAM property. For example, on a node with 3 SCSI controllers and 1 FDDI, you simply 1) load and boot the new obp, 2) set the uXsbX parameters naming the 'afws-fcode' driver, 3) reboot OBP with the 'reset' command.

```
[0:2] ok setenv u0sb0 afws-fcode
[0:2] ok setenv u0sb1 afws-fcode
[0:2] ok setenv ulsb1 afws-fcode
```

(continued next page)

6.3 Configuring Non-Standard or CD Hypernodes (continued)

If you have a machine in which the Convex ROM does not properly boot as indicated by a corrupt, or partial device tree under that controller device, e.g., the sd and st children were missing. This setup can be used to fix that, too because the built-in 'afws-fcode' driver is EXACTLY the same as the one Convex burned into the ROMs.

Configuring CD hypernodes with Landmarc.

The CD machines use single-ended SCSI controllers. They must have different FCode ROMS. If the ROMS are not upgraded or are defective, the setenv command must be used to select the 'se-afws-fcode' device driver for all SCSI controllers in the hypernode. E.g.:

```
[0:2] ok setenv u0sb0 se-afws-fcode
```

The nice thing about the Landmarc controller is that no HARD ERROR is generated when OBP probes empty SBus slots. That's why setup is MUCH easier than with Landfill. Indeed, if the ROMS in the SCSI controllers are right, zero setup of OBP is required - it simply probes and configures the device tree completely automatically.