Exemplar Diagnostics V1.0.1 (Early Adopter) Release Notice

Document No. 760-007830-001

November 12, 1996

Richardson, Texas USA

Exemplar Diagnostics V1.0.1 (Early Adopter) Release Notice

Document No. 760-007830-001

Printed in the United States of America

Release Notes

Utility Documentation

Installing Exemplar Diagnostics V1.0.1

CPU3000 Test

Release Notes

1

1. Overview

This document is intended to enhance and clarify the existing permanent documentation for this product with information that is up-to-the minute, or was developed too late for inclusion in the permanent documentation. Always refer to this release notice before reporting problems with the Exemplar Diagnostics V1.0.1. Fixes and work-arounds are listed here that may save you time in rediscovering known problems.

The remaining sections in this document describe the contents of this release:

- Section 2 describes the contents of this distribution.
- Section 3 contains notes and warnings about the use of the software.
- Section 4 contains enhancements to the previous functionality.
- Section 5 describes fixes for previously reported problems.
- Section 6 describes known software problems.
- Section 7 describes known hardware problems.
- Section 8 describes known documentation problems.
- Section 9 contains descriptions of the new documentation.
- Section 10 in chapter 2 contains the documentation for software products that are still missing their manpages.

Appendix A contains instructions for installing this diagnostic release on the Test Station.\

Appendix B contains a description of the Cpu3000 test.

2. Contents of this Distribution

The distribution package for this release of Exemplar Diagnostics V1.0.1 consists of this document and the software. The specific contents of the software and documentation distribution are described in the following tables:

Table 1: Exemplar Diagnostics Software

Item	Quantity	Part Number	Description
1.	1	760-006315-001	Exemplar Diagnostics V1.0.1

Table 2: Exemplar Diagnostics Documentation

Item	Quantity	Part Number	Description
1.	1	760-007830-001	Exemplar Diagnostics Release Notice

3. Notes and Warnings

All products in the following /spp subdirectories are overwritten when the Exemplar Diagnostics are installed: bin, data, est, etc, firmware, man, scripts, and unsupported. If any files in this directory have been customized they must first be saved and then restored after completion of the install.

4. Enhancements

This is the field test release of the Exemplar Diagnostics V1.0.1. Enhancements to the software provided by this release are broken down into two categories: test station utilities and test software. The following subsections identify the enhancements to the two groups of software.

4.1 Test Station Utilities

4.1.1 ccmu

Added both 'q' and 'e' commands as synonyms for the "quit" command.

Added the "-c" option which clears the cop.

4.1.2 cxtest

Added buttons to the scrollbars in the parameter menu to make it easier to page through the parameters.

Added the "-t" option to the command line interface. This allows the user to specify the monarch CPU for the test_controller, in other words, the CPU on which it executes.

4.1.3 ecc_enable

This new script was added to enable or disable ECC checking across all EMACs. It's interface is as follows:

ecc_enable <node_id> [on | off | chk]

For example, "ecc_enable 0 on" would enable ECC checking on all EMACs in node 0.

4.1.4 est

Array testing optimization. The gate array tests now run in parallel.

The gate array test (command 'g') now has many new options to give the user better control.

4.1.5 EPIC2 ASIC support

This ASIC is now fully supported. Changes were made to POST, the scan tools, and the io3000 diagnostic.

4.1.6 post

Added code to unlock the EPAC read databases.

Added code to implement the EMAC queue overflow workaround.

Added code to prints '.' for valid rows, '#' for deconfigured rows, and '-' for empty rows during the memory line initialization operation.

4.1.7 Support for ECUB2/ENRB2/EIOB2 Boards

The following two files were updated to include these:

- /spp/data/DB_RING_FILE
- /spp/data/boards_list

4.2 Test Software

4.2.1 mem3000

Added code to automatically configure the test hardware. The user is no longer required to specify which CPUs to use or to deconfigure any CPUs. For example, in a 16-CPU and 8-EMB node mem3000 will automatically select an appropriate set of 4-CPUs to use in the testing. However, the user may override this automatic configuring by specifying which CPUs to use.

5. Fixes

This is the initial release of the Exemplar Diagnostics V1.0.1. Problem corrections to the software provided by this release is broken down into two categories: test station utilities and test software. The following subsections identify the problems fixed to the two groups of software.

5.1 Test Station Utility Fixes

5.1.1 ccmd

Modified to increase the stability of this utility when the node is power cycled.

Modified to reduce the probability of communication problems with the JTAG hardware.

Modified code to log events when cop reads are performed.

5.1.2 ccmu

Fixed a stop-on-hard bug.

Modified the nodemap structure to make it easier to write NVRAM.

Added support for both 5-bit and 7-bit node ids.

5.1.3 cxtest

Fixed a problem that resulted in hangs and/or coredumps after this utility had been executing for a period of time.

Fixed a problem in the parameter menu scrollbar usage that was causing coredumps.

Modified code to denote an error occurred in the class when a subtest running from that class selection detects an error. Makes the error reporting more consistent and complete.

5.1.4 EMAC2 alias file

This file allows the hard_logger to fully support analyzing and dumping hard error information from the EMAC2 ASIC.

5.1.5 est

The default log file is now /spp/data/est.log.

EST array tests and error output can now deal with board names in addition to using the board numbers given in the config file.

The default of settings for limiting patterns and printing pass/failsteps have been changed based on user feedback. The new defaults are:

a. P/F info per pattern = off

b. limit array patterns = on

5.1.6 event_logger

Fixed a problem compressing the archived event_log file. This problem only occurred on the second and subsequent times this operation was performed.

5.1.7 hard_logger

Fixed so the cop information is now logged both in the displayed output and in the event_log.

5.1.8 load_eprom

Changed all references to "node_name" to "ip_name".

5.1.9 post

Modified the code to set the "Tag Queue Depth" to 9 instead of 11. This resolves the problem with the LSDYNA code.

Removed the code that sets the ETAC queue size limit CSR in the EMAC.

Modified code to initialize the node to use 5-bit node ids.

Fixed the code to fully support booting to EPSDV.

Added code to set a bit in RDR 30, the FPCLKRUN2 field, to help prevent floating-pointer unit turn-on power surges.

Fixed bug in POST that would not allow an odd number of EMBs, either due to software deconfiguration or due to having an odd number of EMBs physically installed.

5.1.10 sppdsh

Modified code to speed up the get operation when using wildcards.

Fixed a bug in the get command.

5.1.11 tc_show_structs

Modified to display the master state for the test_controller.

Modified to work with the new test_controller data structures.

Fixed a problem that resulted in too many classes being displayed.

5.2 Test Software Fixes

5.2.1 intra3000

Modified code so all subtests are now working, and they have all been enabled.

Fixed the code so all subtests can be executed in parallel on all available CPUs.

5.2.2 io3000

Supports EPIC1 and EPIC2.

Multi-cpu execution is now supported. The number of available CPU's will be distributed across the EPIC's under test. Up to 8 CPU's (one per EPIC) will be used for actual testing.

The multi-cpu enable option has been obsoleted and therefore removed.

This version of io3000 expects version 2.17 of scsi firmware to be loaded in flash (sector 6).

An option has been added to allow usage of scsi firmware other than version 2.17. An enable bit must be set and the firmware length must be specified.

5.2.3 mem3000

Fixed a problem in the decoding of the soft errors that are reported.

Fixed the problems that prevented all 8-EMBs in a node from being tested by 4-CPUs. The previous limit was 2-CPUs and 4-EMBs.

5.2.4 test_controller

Fixed several problems that prevent several of the diagnostics, namely intra3000, cpu3000, and io3000, from executing on all CPUs.

6. Known Software Problems

This section describes all the software problems that are known to exist in the Exemplar Diagnostics V1.0.1.

6.1 Test Station Utilities

6.1.1 load_mem

This utility does not work at all so do not use.

6.2 Test Software

6.2.1 arch3000

This test can only be executed on a single CPU. Running it on multiple CPUs will produce incorrect and unknown results.

6.2.2 io3000

Subtest 735 will fail when 4 epics are under test. To get around this problem, only test two epics at a time.

6.2.3 mem3000

There is two known hardware test configuration requirements that must be met for this test to execute correctly. They are:

- only 1 CPU per EPAC can be used for testing
- each pair of EMBs can be tested by no more than 1 CPU; thus the largest test configuration would be 4-CPUs and 8-EMBs.

7. Known Hardware Problems

None.

8. Known Documentation Problems

There are no known documentation problems.

9. New Documentation

There is no new documentation for this product.

Utility Documentation

2

10. Overview

This chapter contains the documentation for those products that do not yet have manpage.

10.1 Test Station Utilities

10.1.1 cbus

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

Usage: cbus -n <#> -p <#> [-r <addr> | -e <addr> | -w <addr> <data>}

where:

-n <#> is the node to use

-p <#> is the EPAC to use

-r <addr> reads and displays the data at the specified address

-e <addr> erases the sector at the specified address

-w <addr> <data> writes the 32-bit data value into the specified address

For example:

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

10.1.2 diag_version

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

```
$ diag_version
Exemplar Diagnostics, Version 1.0.1.0
```

10.1.3 ecc_calc

This ecc calculator will accept hex input and calculate the ecc value for that data. It is presently written to work on full 88-bit memory lines but can be used for single node (80-bit) data as well. All you have to do is enter 0s for the last byte of data and it will give you the correct single node value.

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

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

To exit this utility, type 'q'.

10.1.4 fix_boot_vector

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

10.1.5 POST

Power-On SelfTest (POST) is a firmware utility that is provides the basic selftest and hardware initialization of a hypernode. It determines the configuration of the hardware and initializes the hardware accordingly. The hardware that is initialized includes: the ECUB SRAM, the CSRs on all ASICs, and main memory.

These are the current POST LCD step definitions:

a – Monarch CPU selected

b – POST code checksum

c – Test Controller code checksum

d – OBP code checksum

e – Configuration parameter checksum

? – Rebuilding configuration parameter table

f – Starting core logic hardware initialization

g – EPUC/EMUC CSR initialization

h – Core logic SRAM pattern test, "A"s pattern

i – Core logic SRAM pattern test, "5"s pattern

j – Core logic SRAM address uniqueness test

k – Zero filling core logic SRAM

l – Starting C-runtime environment

10.1.6 rdr_dumper/rdr_formatter

This set of utilities allow the dumping, formatting, and displaying of the RDR state within a PCXU. These utilities reside in the /spp/unsupported directory and consist of rdr_dumper.fw (a firmware product), rdr_formatter, and scan_sram (an sppdsh script).

The procedure for reading, formatting, and displaying the RDRs is shown below:

Step 1: In the Test Controller, select "POST Boot Selection" option from the main menu.

Step 2: Select the "Dumper" option (#4) and hit the <Return> key.

Step 3: Now exit the Test Controller by selecting main menu option 0. This resets the machine. When POST states its "Booting ?" then wait about 10 seconds for the rdr_dumper program to complete. An alternative is to read the 4 words starting at location 0xf0804000 via sppdsh. As each CPU completes it sets its corresponding byte in this array.

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

Step 5: Now run the RDR formatter on the desired file as follows: /spp/unsupported/rdr_formatter /tmp/cpuX.dump where X is the cpu id, 0-9, A-F. This dumps output to stdout which can be redirected to a file.

Step 6: Now hit the TOC button or type the "assert_toc 0" command in an sppdsh window. This resets the machine which should return you to the OBP prompt.

10.1.7 sppconsole

This script invokes the /spp/etc/console program to provide the OS console interface. Refer to the *console(8)* manpage for more information about this program

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

This interface is used to communicate with the Test Controller, OBP, and SPP-UX. To start this console interface, type the following command in a test station window:

```
sppconsole
```

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

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

10.1.8 stop_on_hard

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

```
Usage: stop_on_hard <node #> <on | off | chk>
```

where <node #> will be 0. The output of the chk option looks like:

EPACs: p0l=0xff p1r=0x0 p2l=0x0 p3r=0x0 p4l=0x0 p5r=0x0 p6l=0x0 p7r=0x0

ERACs: r0l=0x0 r2r=0x0 r3r=0x0 r1l=0x0

EMACs: mb0l_m=0x0 mb1l_m=0x0 mb6r_m=0x0 mb7r_m=0x0

EPICs: iolf_b=0x0 iolf_a=0x0

ETACs:

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

10.2 Test Software

10.2.1 cpu3000

Refer to Appendix B for a description of the subtests within this test.

10.2.2 cpu3000_decode

This utility decodes the event messages displayed by the cpu3000 test (which is described above) and displays the type of error plus any additional information to isolate the failure.

Usage: cpu3000_decode <test controller event message>

The entire event message string must be passed. For example, the following error when passed to **cpu3000_decode:**

```
cpu3000_decode 08/21/96 15:30:32 210 866b7022 T:3c0/ffffffff:dffffa00/ffffffff:dffffe00
```

decodes to the following information:

```
Icache Tag Miscompare in bit 21, maps to U022M5
```

10.2.3 est

EST stands for Exemplar Scan Test. It's job is to allow the user to run different types of scan based test on an Exemplar node.

Usage: est [options] [ecub_name] node_number

where:

options:

-v = print version number, then quit.

-f script = run this script file.

-l = no log file

-o filename = redirect log info to filename

ecub_name = e.g. mu_0000

node_number = what you think it is.

Examples:

```
est 0              # will test node 0
est -v             # will print the version number and quit
est -f my_script 0 # will run my_script & give you est prompt
```

RUNNING EST:

The main est commands are (in the order they are normally run);

r = ring test

d = dc connectivity

a = ac connectivity

g = gate array test

q = quit

Usage:

r [ring [mode]]

mode = bypass, id, boundary, or internal

The default is to run all rings in all modes

d [-s -p pat]

-s = step through the test (debug mode)

-p pat = just run pat #

a [-s -p pat]

-s = step through the test (debug mode)

-p pat = just run pat #

g [options] [pattern file]

There are a number of options that control how the gate array tests are executed:

- o -r refdes = test arrays with a matching refdes value
- o -b board = test arrays on that board. board can be either a name or a number
- o -j jtag_id = test arrays with a matching jtag_id
- o -t type = test an array type (e.g. erac)
- o -s start = starting pattern number
- o -e end = ending pattern number
- o -m max = max number of patterns to run per pattern file
- o -o level = optimization level, choice of 0, 1, or 2 (2's the best).

By default, the g command will test all arrays. The optimization level controls how much parallelism is used. There are 3 level:

0 = no optimization

1 = same parts on the same ring are tested

together

2 = identical rings are tested in parallel

If when doing a parallel test and an error is encountered, the following message may appear:

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

When an error happens, parallel scan's into the parts may result in bus conflicts on TDO pins; therefore, est automatically stops using parallel scans when errors are encountered.

OTHER COMMANDS

There are additional commands. Some of these include:

i print jtag id's.

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

Margin command for clocks (-c) and power (-p).When a value is not supplied, the current states are shown. Examples:

m show all margins

m -c high go to upper clock

m -p 1 nom go to nominal on supply 1

There are 4 power supplies. Here's some info pulled from the ecub spec:

Group	Board	Voltage	Name
1	EPB,EMB,EIOB	3.3V	VDD
2	EPB	1.5V	VDDQ
3	EPB	?	Future Voltage
4	ENRB	3.3V	VDD
4	EIOB	5.0V	VCC

ms Stands for Make Safe. Puts all the parts into a safe state - no bus conflicts.

script <file>

Runs a file containing est commands.

v Print the version info.

EST FLAGS:

There are a number of flags or options that will control how the above tests operate. To set these options, one should enter 'F' at the est prompt. That will bring up the flags sub menu. To get out, just hit return at the flags prompt and you will return to the main est prompt.

Here are some of the more useful supported flags stolen right out of the flags menu (note the default settings):

l ... limit patterns is enabled

s ... stop on errors is disabled

A ... limit AC connectivity tests - no limit set

D ... limit DC connectivity tests - no limit set

E ... show libcst data is disabled

P ... pattern p/f data is disabled

<ret> return to main menu

Some explanations:

l ..... Limits the number of internal array patterns that get executed by the g command. The has the affect of decreasing coverage to approximately 90%. The

s ..... stops testing when an error is found.

A # ... Limits ac connectivity testing.

D # ... Limits dc connectivity testing.

Providing a limit <= 0 will result in all patterns being run.

E ..... shows sdp packets that go across the ethernet. Very useful when debugging est problems.

P ..... Controls whether or not the pass/fail status of individual patterns are displayed.

OUTPUT:

All output goes to the screen *and* to a log file. The default log file is /spp/data/est.log. Each time est is run, the previous log file is moved to est.log.old.

USING SCRIPT FILES:

There are two ways of running script files, either from the command line (-f file) or at the est prompt. When run at the command line, est will execute the est instructions listed and when finished, give the user the est prompt.

The script command will read a file (ascii) for est commands and run those commands. The commands should be:

1. one per line.
2. same syntax as when entered at the est prompt.
3. start with a '#' for comments

To handle the setting of options ('F'), the syntax is

F option [value]

Example file:

check the rings

r

show pattern pass/fail steps

F P

limit dc testing to 3 patterns

F D 3

do dc testing

d

10.2.4 tc_show_struct

This sppdsh script displays the current Test Controller state for the specified diagnostic test. This display includes the following information: entrypoints, status flags, selected classes and subtests, parameter values, test hardware configuration, and the state of each CPU.

The usage of this command is described below.

Usage: tc_show_struct <test_name>

Where testname would be one of the following

-cpu

-intra

-io

-inter

-mem

-arch

An example of this utility's output is shown below:

```
$ tc_show_struct -mem
-------------------------------------------------------------
Name : MEM3000 - EEPROM based memory tests
Entry Pt     ClTb ptr    StTb ptr      HwReq     ParmTbptr
Parm_ptr
-------------------------------------------------------------
0xf0160000  0xf016006c  0xf0160608  0xf0160064  0xf08162d0
0xf0808368
-------------------------------------------------------------
Hardware req met = ffffffff | Test inited   = ffffffff |
Selected      = ffffffff
-------------------------------------------------------------
Class[0]   = 0    Subtest[0] = 510
Class[1]   = 0    Subtest[1] = 000
Class[2]   = 0    Subtest[2] = 000
Class[3]   = 0    Subtest[3] = -001
Class[4]   = 0    Subtest[4] = -001
```

```
Class[5]    = 510     Subtest[5] = -001
------------------------------------------------------------
Current Values for Parameters
00) 0xa5a5a5a5 01) 0xa5a5a5a5 02) 0x5a5a5a5a 03) 0x5a5a5a5a
04) 0x00000003 05) 0x00000007 06) 0x00000001 07) 0x00000000
08) 0x00000000 09) 0x00000000 10) 0x00000000 11) 0x00000000
12) 0x00000000 13) 0x00000000 14) 0x00000000 15) 0x00000000
16) 0x00000000 17) 0x00000000 18) 0x00000000 19) 0x00000000
------------------------------------------------------------
CPU Mask  = 0x0005     EPAC Mask = 0xff
EMAC Mask = 0xc3       ETAC Mask = 0x00
EPIC Mask = 0x11
------------------------------------------------------------
CPU 0  - State - TC_CPU_DONE                        Subtest 510
CPU 1  - State - TC_CPU_IDLE                         Subtest 0
CPU 2  - State - TC_CPU_DONE                        Subtest 510
CPU 3  - State - TC_CPU_IDLE                         Subtest 0
CPU 4  - State - TC_CPU_IDLE                         Subtest 0
CPU 5  - State - TC_CPU_IDLE                         Subtest 0
CPU 6  - State - TC_CPU_IDLE                         Subtest 0
CPU 7  - State - TC_CPU_IDLE                         Subtest 0
CPU 8  - State - TC_CPU_IDLE                         Subtest 0
CPU 9  - State - TC_CPU_IDLE                         Subtest 0
CPU 10 - State - TC_CPU_IDLE                         Subtest 0
CPU 11 - State - TC_CPU_IDLE                         Subtest 0
CPU 12 - State - TC_CPU_IDLE                         Subtest 0
CPU 13 - State - TC_CPU_IDLE                         Subtest 0
CPU 14 - State - TC_CPU_IDLE                         Subtest 0
CPU 15 - State - TC_CPU_IDLE                         Subtest 0
```

10.2.5 tc_standalone

This sppdsh script modifies a location in NVRAM that informs the Test Controller to go to it's standalone mode of operation. The standalone mode of operation is used for running cxtest. This takes affect the next time the Test Controller is started.

10.2.6 test_controller

This firmware product provides the run-time environment for the diagnostic tests. There are two modes of operation: INTERACTIVE and STANDALONE which are further described below.

INTERACTIVE MODE

This operational mode interacts with the user via the OS console window. There are many menus that allow the user to specify which tests (and subtests) to execute, the test hardware configuration, the parameter values to use, plus several testing options. All menus should be self-explanatory.

STANDALONE MODE

This mode is used so the **cxtest** utility (which runs on the test station) can execute the diagnostic tests. This utility provides a GUI that makes it easier to execute the tests. Refer to the cxtest(1) manpage for more details.

This operational mode can be started by selecting main menu item #5 and then selecting boot option #3 (Standalone Diags). Now exiting the Test Controller will return the user to this mode. Another method to use from the OBP "OK" prompt is to specify the boot-module to be "diags", run the **tc_standalone** script on the test station, and then reset the system via the OBP "reset" command.

When this operational mode is active, the Test Controller prints the following string at start-up time:

Running in Standalone Mode.

Installing Exemplar Diagnostics V1.0.1

A

1. Installation procedure

Step 1: Installing from tape to the Test Station:

Login as root and shut the test station down to single-user mode via the "/etc/shutdown now" command.

Now follow the procedures outlined in the Release Distribution document for installing from tape.

Step 2: After diagnostics are installed on the Test Station.

Follow these steps to setup up the diagnostic environment of the Test Station:

- cd /spp/scripts/inst
- ./ts.install
  - Follow the directions of the script and input the requested information
  - This script displays all the pertinent information and requests confirmation prior to the actual updates. Make sure the information is correct before proceeding.
- Now reboot the Test Station using the "/etc/reboot" command.

Step 3: After the Test Station is booted:

- Start the Exemplar console window by entering the "sppconsole" command in any window other than the Test Station console window.
- Download the diagnostic firmware code into the flash memory on the ECUB using the following procedure:
  - Boot the system to the OBP prompt, "[...] ok ".
  - The OBP commands used to download the diagnostic firmware are in the file /spp/scripts/dl-diags.fth file on the Test Station. Cut and paste each line one-at-a-time from this file to the OBP command line.
- At this point the Test Station is ready for use.

CPU3000 Test

B

This appendix documents the various subtests in the cpu3000 diagnostic test.

1. Overview

This test provides a basic test of the functionality of the PCXU processor. Included in the testing are most of the instruction set, the ALU, general/space/control registers, external interrupts, the RDRs, TLB ram, the instruction cache, and the data cache.

All of the testing code plus the following descriptions came were borrowed from the PA8000 selftest and initialization code generated by the Mohawk group. The changes we made include: modifying the subtest numbering scheme, adding the framework so it runs in our test environment, modifying the error reporting scheme, and adding isolation capabilities where we could.

2. Subtest Descriptions

Subtest 100 - Cpu-Basic. This routine tests the majority of registers and a basic set of instructions.

(error return code = 0x41020xxx) xxx = progress value

Subtest 101 - Cpu-Alu. This routine tests the cpu alu functionality.

(error return code = 0x41021xxx) xxx = progress value

Subtest 102 - Cpu-Br. This routine tests the branch instructions.

(error return code = 0x41022xxx) xxx = progress value

Subtest 103 Cpu-Arith-Cond. This routine tests the arithmetic conditions of the unit, extract/deposit and carry/borrow instructions.

(error return code = 0x41023xxx) xxx = progress value

Subtest 104 - Cpu-Bit-Ops. This routine tests the cpu's bit operation.

(error return code = 0x41024xxx) xxx = progress value

Subtest 105 - Cpu-Cr. This routine tests the Spaces and the Controlsregisters.

(error return code = 0x41025xxx) xxx = progress value

Subtest 111 - Cpu-Ext-Int. This routine executes sixty three external interrupts, one for each EIR_VAL position excluding Itimer.

(error return code = 0x41026xxx) xxx = progress value

Subtest 120 - Cpu-Multi-Media. This routine tests the functional operation of the multi media instructions.

(error return code = 0x41028xxx) xxx = progress value

Subtest 130 - Cpu-Shadow. This routine tests the shadow registers.

(error return code = 0x41029xxx) xxx = progress value

Subtest 140 - Cpu-Drs. This routine tests the Local Diagnose Registers.

(error return code = 0x4102axxx) xxx = progress value

Subtest 141 - Cpu-Rdrs. This routine tests the Remote Diagnose Registers.

(error return code = 0x4102bxxx) xxx = progress value

Subtest 150 - Cpu-Bypass. This routine tests the register bypass functionality of the processor. It tests three different types of bypassing that can occur between the two integer queues.

(error return code = 0x4102cxxx) xxx = progress value

Subtest 200 - Icache-Aline. This routine tests the instruction cache address lines.

(error return code = 0x42010xxx) xxx = progress value

Subtest 210 - Icache-Ram. This routine pattern tests the icache ram.

(error return code = 0x42020xxx) xxx = progress value

Subtest 310 - Dcache-Ram. This routine tests the data cache rams.

(error return code = 0x42070xxx) xxx = progress value

Subtest 320 - Dcache-tag. This routine verifies that the RPN is recognized correctly.

(error return code = 0x42080xxx) xxx = progress value

Subtest 400 - Tlb-Ram. this routine tests the TLB ram arrays with apseudo random pattern.

(error return code = 0x410b1xxx) xxx = progress value