QLogic Everest

eDiag OEM User's Guide

Version 3.26

TABLE OF CONTENTS

Contents

Introduction

This document provides the information on how to use eDiag utility for manufacturing and engineering level diagnostics for the 10 Gbps Ethernet controllers.

This manual is intended for engineers, testers, technicians and OEM customers.

This manual describes the eDiag commands and usage, and all the diagnostic tests supported on BCM5771x, BCM578xx and its associated components. Some tests may not apply, depending upon the hardware installed in the system and the available software licenses.

1. Operating Environment

1.1 MS-DOS environment

The eDiag utility operates in an MS-DOS environment. It includes a DOS extender (PMODE/W) that is embedded into the executable and provides access to memory above 1 MB.

OS: MS-DOS 6.22

Additional driver(s): ANSI.SYS

To install this driver, simply place ANSI.SYS file in some directory (e.g. C:\DOS) and add a line (e.g. DEVICEHIGH=C:\DOS\ANSI.SYS) in the config.sys file. The ANSI driver allows the manufacturing mode tests to display the test results clearly the Red/Green colors for Fail/Pass status. Note that the eDiag utility will run correctly even without the ANSI driver installed.

Additional driver(s): HIMEM.SYS

To install this driver, place HIMEM.SYS file in a directory (e.g., C:\DOS) and add a line (e.g., DEVICE=C:\DOS\HIMEM.SYS) in the config.sys file. The HIMEM driver is required for TOE test D7 to pass.

Software: ediag.exe

To run the eDiag tool simply run: "eDiag" from your DOS prompt.

Input File List: Whenever required, the following files should be placed in the same folder where ediag.exe is present.

- **Macaddr.txt:** This text file is used to program the primary MAC and iSCSI MAC addresses of the Everest devices during OEM manufacturing. It consists of a range of available MAC addresses (for L2 and iSCSI functionality) to be used when programming the MAC address of the Everest controller. As addresses are programmed into the Everest controller, eDiag will update the macaddr.txt file and reduce the range of available addresses by the number of addresses used. This file is not required for normal eDiag operation. Please refer to section 9.1 for more details.
- **Everest.rdf, ipblocks.rdf:** These files include the Everest registers definitions: Offset, access type, data width, address width.
- **Uniblock.rdf, uniregs.rdf:** These files are required in order to perform "nictest" testing on the Everest controller.

1.2 Linux environment

Linux eDiag currently supports RH4, RH5, SLES9, and SLES10 distributions and the latest Open Source kernels (2.6.27).

It supports both 32-bit and 64-bit kernels – there are different Linux eDiag distributions for 32-bit and 64-bit versions.

Linux eDiag supports concurrent mode with the **bnx2x** driver. In this mode, only basic functionality is enabled. The following commands are disabled in this mode:

driver

lmdev

Software:

- **load.sh** – script that should be used to run the tool.
- **edrv.ko** – kernel driver.
- **lediag** – Linux eDiag application.

How to build and run Linux eDiag:

- Open a distribution tarball.
- Enter the lediag-<version number>-<platform> directory.
- Run **make** command to build the platform dependent kernel module and link the application.
- Run **./load.sh** to run the Linux eDiag.

Compatibility with DOS eDiag:

- load.sh script receives the same parameters as DOS eDiag (-b10eng, etc.).
- Linux eDiag distribution includes the same complementary files as DOS version (macaddr.txt, RDF files, etc.).
- Linux eDiag supports most of commands supported by DOS eDiag except for a few like **serial**. See release notes for more details.

1.3 UEFI environment

The UEFI eDiag utility operates in UEFI shell environment

OS: UEFI shell version 2.0 or above

Software: ediag_x64.efi for x64 architecture

ediag_ia64.efi for ipf architecture

To run the UEFI eDiag tool use: "**ediag_x64** / **ediag_ia64**" from UEFI shell prompt.

Input File List:

UEFI eDiag requires the same input files (macaddr.txt, RDF files, etc.) as required by ediag (described in section 1.1).

2. Modes of Operation

The eDiag utility has two modes of operation: manufacturing mode and engineering mode.

2.1 Manufacturing Mode

Manufacturing mode is optimized for manufacturing environments and is characterized by its extensive use of command-line arguments to perform configuration tasks and component testing. The following commands are examples of manufacturing mode testing:

- C:\>ediag
- C:\>ediag –t a1c2
- C:\>ediag –mac 11:22:33:44:55:66 –wol 0

2.2 Engineering Mode

Engineering mode provides an interactive test environment that supports all of the functionality of manufacturing mode plus additional commands for specific tasks such as reading/writing individual chip registers, generating specific traffic patterns for wire testing, and developing complex scripts for more advanced testing. The following command starts eDiag in engineering mode:

- C:\>ediag –b10eng

The below variation is also supported for backwards compatibility:

- C:\>ediag –b06eng

2.2.1 Command Prompt

When eDiag is executed in engineering mode, it will display a command prompt and wait for a user to enter commands (documented in section 7). The command prompt has the following format:

- <Everest Device#>

For example, a command prompt of "1:>" indicates that eDiag is currently operating on the first Everest device (as found on the PCI bus). A command prompt of "2.:>" indicates that the second Everest device is selected.

3. Using eDiag over UART

To use eDiag over UART, the user should connect a serial cable between the 10 Gbps Ethernet controller board serial I/O port to the machine serial I/O port on which eDiag utility is executed.

The purpose of the use of eDiag over UART is mainly to debug failures in which the host machine fails to respond via PCI interface.

To use eDiag over UART start eDiag with the following option:

- C:\>ediag –no_pci

The command prompt for eDiag over UART has the following format:

NO_PCI:>

Upon the appearance of this command prompt the user should open serial connection using:

NO_PCI:> set_duart open 1 19200

Then user can run commands or diagnostics scripts (such as idle_chk.tcl) to debug the failure. For example to execute IDLE check using eDiag over UART do the following steps:

1. Connect a serial cable between the 10 Gbps Ethernet controller board serial I/O port to the machine serial I/O port on which eDiag utility is executed.

2. run "ediag –no_pci"

3. upon NO_PCI:> prompt run "set_duart open 1 19200"

4. run "idle_chk"

(*) Note: some of the eDiag commands are not available in this mode such as "device", "nictest".

4. eDiag Test List

The eDiag utility provides a wide range of tests to verify specific blocks of functionality within the Everest controller. The tests are divided into 6 groups (labeled A through F), with a variable number of specific tests within each group.

The table below describes what each testing group includes:

Group Name	Group Description
Group A	**Basic tests: registers tests for each block in the chip, PCI test, MSI test, MSI-X test, Interrupt test.**
Group B	**Memory tests for each block in the chip. The memory tests are based on "marching ones" and "marching zeros" algorithms for testing memory.**
Group C	**Functionality test of DMAE, CAM, Timers, PHY.**
Group D	**Traffic tests via MAC loopback, PHY loopback. Tests include sending packets of various types, receiving them and performing data integrity. Also includes LSO, RPC and statistics tests.**
Group E	**PHY External loopback tests and L2 traffic test with external loopback. This group is disabled by default.**
Group F	**L2 traffic tests with external loopback, with different parameters, mainly for firmware testing. This group is disabled by default.**

To run the default test suite in **manufacturing mode** type, the following command at the DOS prompt:

C:\> ediag

To perform the same suite of tests in engineering mode, enter the following command at the engineering mode command prompt:

1.NONE:> nictest

(*) User can run "nictest" on a desired device e.g. 2:> nictest

Some of the diagnostic tests described below will require external configuration for the test to pass (such as an external loopback adapter). These tests must be explicitly enabled to run; by default these tests are disabled.

In order to run a specific test or a specific test group, in **manufacturing mode** type the following command at the DOS prompt :

C:\> ediag [-none] [-t <groups>] [-T <groups>] [-I <num>] [-cof] [-dev <device#>]

-none Exclude all tests
-t Exclude specified tests
-T Include specified tests
-I Specify the number of iterations to run
-cof Enables to continue after test failure
-dev Execute tests on specified device or device list (e.g. –dev 1 –dev 1-2)
-debug Print debug printouts

In order to run a specific test or a specific test group, in **engineering mode** type the following command at the DOS prompt on the desired device:

1:> nictest [-none] [-t <groups>] [-T <groups>] [-I <num>] [-cof]

Example 1:

Manufacturing Mode Command:

C:> ediag –none –T A -cof

Engineering Mode Command:

1:> nictest –none –T A -cof

Will perform only test group A and will continue over failure.

Please note that in all the examples the manufacturing mode will run on all Everest devices, while the engineering example will run only on the device specified in the port (or the matching port if the –port option was used).

Example 2:

Manufacturing Mode Command:

C:> ediag –none –T B1

Engineering Mode Command:

1:> nictest –none –T B1

Will perform only test B1.

4.1 Group A – Basic Functional Tests

4.1.1 Register Test (A1 – A29)

The Register Tests verify that registers access through the PCI/PCI-E interface implement the expected read-only or read/write attributes by attempting to modify those registers.

The test uses the Everest.rdf (or evrst15.rdf for DID > 57710) register's definition file.

The test first resets the chip, and then verifies that the reset value of each register is as expected in the "everest.rdf" file. Afterwards writes to each register an all zeros value and verifies that the read-write registers have changed accordingly and that the read-only registers did not change.

Test also verifies that the read-clear registers have changed their values to zero upon the second read. The test repeats the pervious steps with writing an all ones value.

Manufacturing Mode Command	C:\>ediag –none –T A1
Engineering Mode Command	1:>regtest –b <block name>
Return Codes	0 – Test passed
Possible Failure Causes	Any failure in this test usually indicates a defective controller.
Enabled by Default	Yes. Except for test A11 Misc block register which is not enabled by default, since it fails if not run after a hard-reboot.
Special Requirements	Some critical registers are not tested as the system and/or the chip may become unstable when the register value is changed. The MISC and NIG block registers are also not tested.

4.1.2 EMAC Register Test (A30)

The EMAC Register Tests verify that EMAC registers access through the PCI/PCI-E interface implement the expected read-only or read/write attributes by attempting to modify EMAC registers.

Unlike the previous blocks, the EMAC block registers can have different read/write attributes per its bits, thus require a different test apart from the "regtest" described above.

Manufacturing Mode Command	C:\>ediag –none –T A30
Engineering Mode Command	1:>emactest
Return Codes	0 – Test passed
Possible Failure Causes	Any failure in this test usually indicates a defective controller.

Enabled by Default	No.
Special Requirements	Some critical registers are not tested as the system and/or the chip may become unstable when their value is changed.

4.1.3 MCP Register Test (A31)

The MCP Register Tests verify that MCP registers access through the PCI/PCI-E interface implement the expected read-only or read/write attributes by attempting to modify MCP registers.

Unlike the previous blocks, the MCP block registers can have different read/write attributes per its bits, thus require a different test apart from the "regtest" described above.

Manufacturing Mode Command	C:\> ediag –none –T A31
Engineering Mode Command	1:> mcptest
Return Codes	0 – Test passed
Possible Failure Causes	Any failure in this test usually indicates a defective controller.
Enabled by Default	No
Special Requirements	Some critical registers are not tested as the system and/or the chip may become unstable when their value is changed.

4.1.4 PCI Configuration Test (A32)

The PCI Configuration Test checks the functionality of the PCI BAR (Base Address Register) by varying the amount of memory requested by the BAR and verifying that the BAR actually requests the correct amount of memory (by checking the PCICFG_BAR).

Manufacturing Mode Command	C:\>ediag –none –T A32
Engineering Mode Command	1:>pcitest
Return Codes	0 – Test passed
Possible Failure Causes	Any failure in this test usually indicates a defective controller.
Enabled by Default	Yes

Special Requirements	None

4.1.5 **MSI_X Test (A33)**

The MSI_X Test checks the functionality of the MSIX (Message Signaled Interrupt). The test generates an interrupt command to the STORM, and then waits up to 10 ms for MSI-X interrupt to be generated. Verifies there are no INTA interrupts during this time and that only the correct MSI-X vector is written at this time.

A negative test is also performed to verify that when the MSI-X is masked it does not generate an MSI-X interrupt upon an interrupt command to the STORM.

The test also checks that the functionality of disabling MSI-X.

Manufacturing Mode Command	C:\>ediag –none –T A33
Engineering Mode Command	1:>int_msix
Return Codes	0 – Test passed
Possible Failure Causes	Any failure could indicate a chip defect.
Enabled by Default	No.
Special Requirements	None

4.1.6 **MSI Test (A34)**

The MSI Test checks the functionality of the MSI (Message Signaled Interrupt). The test generates an interrupt command to the STORM, and then waits up to 10 ms for MSI interrupt to be generated. Verifies there are no INTA interrupts during this time and that only the correct MSI vector is written at this time.

A negative test is also performed to verify that when the MSI is masked it does not generate an MSI interrupt upon an interrupt command to the STORM.

The test also checks that the functionality of the disable MSI.

Manufacturing Mode Command	C:\>ediag –none –T A34
Engineering Mode Command	1:>msitest
Return Codes	0 – Test passed
Possible Failure Causes	A failure may indicate that the chip is defective.

Enabled by Default	No.
Special Requirements	None

4.1.7 IGU, PGLUE_B, ATC Register Tests (A35 – A36)

Register tests, similar to A1-A29 for BCM57712 device new blocks: IGU, PGLUE_B.

Manufacturing Mode Command	C:\>ediag –none –T A35
Engineering Mode Command	1:>regtest –b <block name>
Return Codes	0 – Test passed
Possible Failure Causes	Any failure in this test usually indicates a defective controller.
Enabled by Default	Only on the BCM57712 device.
Special Requirements	None.

4.2 Group B - Memory Tests

4.2.1 Memory Tests (B1 – B30)

The Group B tests verify all memory blocks of the Everest adapter by writing various data patterns (marching zeroes, marching ones) to each memory location in each block of the Everest controller and then reading back the data, comparing it to the value written.

The marching ones algorithm tests each memory entry with values 0x1, 0x11 until an all ones value is tested. The marching zeros algorithm tests each memory entry with all ones value until an all zeros value is tested.

Group B includes memory tests for each memory in the Everest controller blocks:

1. Block BRB1 Memory Test
2. Block CCM Memory Test
3. Block CDU Memory Test
4. Block CFC Memory Test
5. Block CSDM Memory Test
6. Block CSEM Memory Test
7. Block DBG Memory Test
8. Block DMAE Memory Test
9. Block HC Memory Test
10. Block NIG Memory Test
11. Block PBF Memory Test
12. Block PXP Memory Test
13. Block PXP2 Memory Test (not run on default setting)
14. Block QM Memory Test
15. Block SRC Memory Test
16. Block TCM Memory Test
17. Block TM Memory Test
18. Block TSDM Memory Test
19. Block TSEM Memory Test
20. Block UCM Memory Test
21. Block UPB Memory Test
22. Block USDM Memory Test
23. Block USEM Memory Test

24. Block XCM Memory Test
25. Block XPB Memory Test
26. Block XSDM Memory Test
27. Block XSEM Memory Test
28. Block IGU Memory Test
29. Block ATC Memory Test
30. Block VFC Memory Test

Manufacturing Mode Command	C:\>ediag –none –T Bx (where x is replaced by the specific test number)
Engineering Mode Command	1:> memtest –b <block name>
Return Codes	0 – Test passed
Possible Failure Causes	A failure may indicate that the chip is defective.
Enabled by Default	Yes. Tests B28-B30 are only for the BCM57712 device. Test B30 is not performed by default.
Special Requirements	None.

4.2.2 **BIST Test (B31)**

The CPU BIST test verifies all memory by operating the memory BIST HW through the GRC.

The BIST is a self-test circuit that internally generates test data patterns and expected resulting data for comparing the expected resulting data with actual resulting data.

The BIST interfaces circuitry for loading the test data patterns into memory and reading back the actual resulting data from memory.

Manufacturing Mode Command	C:\>ediag –none –T B31
Engineering Mode Command	1:> bist_test
Return Codes	0 – Test passed

Possible Failure Causes	Any failure in this test usually indicates a defective controller.
Enabled by Default	No.
Special Requirements	None.

4.3 Group C - Block Tests

4.3.1 DMA Engine Test (C1)

The DMA Engine Test verifies the functionality of the DMA Engine block by performing numerous DMA operations with random parameters (using CMD #0 of the DMAE block). All four flows are exercised: PCI to PCI, PCI to GRC, GRC to PCI and GRC to GRC.

After each operation the test waits for completion, and upon receiving a completion, the test compares the source and destination data and a mismatch is considered an error.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\>ediag –none –T C8
Engineering Mode Command	1:> dmaetest
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the "driver load" at the beginning of the test failed. A failure in this test may indicate there are issues with the system chipset, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	This test includes a patch of "driver load" and "driver unload" since the initialization of this script is incomplete. Without this patch the traffic tests after this DMAE test will fail.

4.3.2 CAM Access Test (C2)

The CAM Test Checks the functionality of the CAM (content-addressable memory) of the TSEM, CSEM, XSEM, and USEM blocks. The test performs for each block: Initialization and "marching ones" and "marching zeros" algorithms.

Manufacturing Mode Command	C:\>ediag –none –T C2
Engineering Mode Command	1:> cam_test
Return Codes	0 – Test passed
Possible Failure	A failure in this test may indicate that the chip is defective.

Causes	
Enabled by Default	Only on primary device.
Special Requirements	None

4.3.3 CAM Search Test (C3)

The CAM Search Test checks the functionality of the CAM search mechanism of all the CAMs in the Everest controller using the BIST enable flag of each CAM.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T C3
Engineering Mode Command	1:> cam_search
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	Only on primary device. On FPGA, this test is not performed.
Special Requirements	None

4.3.4 Timers Test (C4)

The Timers Test checks the functionality of the timers on the requested port. The test generates random timeouts on 512 connections of up to 8 seconds and check that a timer has expired for all 512 connections.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T C4
Engineering Mode Command	1:> tmr_test –conn 512 –timeout 8000
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.

Enabled by Default	Yes on functions 0,1 only.
Special Requirements	None.

4.3.5 GRC Test (C5)

The GRC Test checks the functionality of the GRC by attempting to access an un-mapped address and expecting a relevant attention alarm. The test attempts to access an address which is not mapped to RBC and expects an attention #123 to be asserted. The test attempts to access an address which is not mapped to RBCN and expects an attention #119 to be asserted. The test also checks the "close the gate" feature.

Note: The GRC timeout feature is not tested since it is not possible to "force" a block not to respond to a GRC transaction.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T C5
Engineering Mode Command	1:> grc_test
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	Yes
Special Requirements	Test is performed on port 0 and not available on port 1.

4.3.6 Debug Bus Test (C6)

The Debug Bus Test performs a debug bus check with output to PXP. Each STORM outputs numerous print commands to the debug bus at the same time and the Debug block transmits all the data to the PXP. The test reads from the PXP and compares the read data to the sent data.

The test is performed on port 0 only.

Manufacturing Mode Command	C:\> ediag –none –T C6
Engineering Mode Command	1:> dbg_pxp
Return Codes	0 – Test passed

Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	Only on primary device.
Special Requirements	None.

4.3.7 SERDES Register Access Test (C7)

The SERDES Register Access Test performs reset value check and read/write access check to all the SERDES registers which are part of the link training.

The test is performed on both port 0 and port 1 of the Everest controller.

This test is excluded if serdes external PHY is not connected.

Manufacturing Mode Command	C:\> ediag –none –T C7
Engineering Mode Command	1:> unicore_regs_test -phy SERDES
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	Yes. This test is excluded if SERDES external PHY is not connected.
Special Requirements	If the NVRAM configuration indicates that the SERDES is not connected for this device, the test will be excluded from testing.

4.3.8 XGXS Register Access Test (C8)

The XGXS Register Access Test performs reset value check and read/write access check to all the XGXS registers which are part of the link training.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T C8
Engineering Mode Command	1:> unicore_regs_test -phy XGXS
Return Codes	0 – Test passed

Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	Yes On FPGA/Emulation this test is not performed.
Special Requirements	None

4.3.9 SERDES PRBS Internal Loopback Test (C9)

The SERDES PRBS Internal Loopback Test performs PRBS test using internal loopback. The SERDES generates and collects patterns independently of other blocks.

The test is performed on both port 0 and port 1 of the Everest controller.

This test is excluded if serdes external PHY is not connected.

Manufacturing Mode Command	C:\> ediag –none –T C9
Engineering Mode Command	1:> prbs_test -phy Serdes –l internal
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	Yes. This test is excluded if SERDES external PHY is not connected.
Special Requirements	If the NVRAM configuration indicates that the SERDES is not connected for this device, the test will be excluded from testing

4.3.10 XGXS PRBS Internal Loopback Test (C10)

The XGXS PRBS Internal Loopback Test performs PRBS test using internal loopback. The XGXS generates and collects patterns independently of other blocks.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T C10
Engineering Mode Command	1:> prbs_test -phy xgxs –l internal

Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	Yes. On FPGA/Emulation this test is not performed.
Special Requirements	None

4.3.11 **NVM Test (C11)**

The NVM Test fills NVM last two pages with address patterns, then reads the last two pages and verifies that the data is as expected. The test then restores the original values in the last two pages of the NVM.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T C11
Engineering Mode Command	1:> nvm_test
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the NVM is defective.
Enabled by Default	Yes. On FPGA/Emulation, this test is not performed.
Special Requirements	None

4.3.12 **RMII external loopback test (C12)**

The UMP MAC Loopback test verifies the UMP block in external loopback mode. It exercises different functions in the UMP block, including frame type pre-fetch register, ingress and egress path isolation, flow control, operation in FIFO cut through mode, and traffic test with different packet size.

Manufacturing Mode Command	C:\> ediag –none –T C12
Engineering Mode Command	1:> ump_extlb_test

Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test, depending on the area of verification, may indicate that the specific function is missing, external loopback failure, or that the chip is defective.
Enabled by Default	No.
Special Requirements	This test requires an external loopback connector to be installed.

4.3.13 **VFC CAM test (C13)**

The VFC CAM test performs memory tests on VFC CAMs located at TSEM_REG_SEM_FAST_VFC_ADDR and XSEM_REG_SEM_FAST_VFC_ADDR using special marching ones and marching zeros tests.

Manufacturing Mode Command	C:\> ediag –none –T C13
Engineering Mode Command	1:> vfc_cam_test
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate chip is defective. Test is relevant to chip 57712 or above.
Enabled by Default	No.
Special Requirements	Test is relevant to chip 57712 or above.

Note:

Group C tests are not performed on ZLR (Zero-Length) machines.

4.4 Group D – Ethernet Traffic Tests

4.4.1 MAC Loopback Test (D1)

The MAC Loopback test will enable BMAC loopback mode in the Everest controller and transmit 9000 Layer-2 packets of various sizes. As the packets are received back by eDiag, they are checked for errors. Packets are returned through the MAC receive path and never reach the PHY.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T D1
Engineering Mode Command	1:> bmaclb_test -case 3 –subcase A
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the interrupt was not asserted, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, that an external LAN connection is present and receiving traffic, that there are issues with the system chipset, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	The controller should not be connected to a network.

4.4.2 PHY Loopback Test (D2)

The PHY Loopback test will enable PHY loopback mode (XGXS 10 loopback) in the Everest controller and transmit about 9000 Layer-2 packets of various sizes. As the packets are received back by eDiag they are checked for errors. Packets are returned through the PHY receive path and never reach the wire.

Manufacturing Mode Command	C:\>ediag –t abcd –T d2
Engineering Mode Command	1:> phylb_test -case 3 –subcase A
Return Codes	0 – Test passed

Possible Failure Causes	A failure in this test may indicate that the interrupt was not asserted, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, that an external LAN connection is present and receiving traffic, that there are issues with the system chipset, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	The controller should not be connected to a network.

4.4.3 External Loopback Test (D3)

The External Loopback test will transmit about 9000 Layer-2 packets of various sizes. As the packets are received back by eDiag they are checked for errors.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T D3
Engineering Mode Command	1:> extlb_test -case 3 –subcase A
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the interrupt was not asserted, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, that an external loopback adapter is not installed, that a LAN connection is present and receiving traffic, that there are issues with the system chipset, or that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.4.4 LSO Test (D4)

The LSO test verifies the functionality of the Everest Large Send Offload support by enabling BMAC loopback mode and transmitting large TCP packets with different MSS sizes, payload

sizes, IP option sizes and VLAN on/off. As the packets are received back by eDiag they are checked for proper segmentation (according to the selected MSS size) and any other errors.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T D4
Engineering Mode Command	1:> bmaclso_test –type basic
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the interrupt was not asserted, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, that an external LAN connection is present and receiving traffic, that there are issues with the system chipset, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	The controller should not be connected to a network.

4.4.5 Statistics Test (D5)

The Statistics test verifies that the basic statistics information maintained by the chip is correct by enabling BMAC loopback mode and sending Layer-2 packets of various sizes and various types (Unicast/ Multicast/ Broadcast) and verifying statistics counters are as expected.

The test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\>ediag –none –T D5
Engineering Mode Command	1:> stat_test
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the interrupt was not asserted, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, that an external LAN connection is present and receiving traffic, that there are issues with the system chipset, or that the chip is defective.

Enabled by Default	Yes.
Special Requirements	The controller should not be connected to a network.

4.4.6 **RPC Test (D6)**

The RPC test verifies the Receive Path Catch-up path by sending packets with different sizes with incremental or fixed payload and marking them to go through the RPC path. The packets will return to the receive buffers as received packets. As packets are received back by eDiag, they are checked for errors.

Manufacturing Mode Command	C:\> ediag –none –T D6
Engineering Mode Command	1:>rpc_test
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the interrupt was not asserted, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, that an external LAN connection is present and receiving traffic, that there are issues with the system chipset, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	The controller should not be connected to a network.

4.4.7 **TOE Test (D7)**

The TOE test verifies the Layer-4 functionality by opening TX and RX connections. Setting the "rcv_next" parameter to almost 4G, transmitting data and waiting for completions. At the end of the test, the connection is terminated and an idle_chk is performed to verify that the Everest controller is back at IDLE state after this test.

Manufacturing Mode Command	C:\> ediag –none –T D7
Engineering	1:> bmacl4_test -mode pre -case 6

Mode Command	
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test can be due to lack of free memory in the tested setup. Verify himem.sys is installed according to section 1.1. From version 4.6.15 and above, a failure in this test can be a result of the license key not existing on the board or the max TOE connections allowed is less than 250 connections. A failure in this test may indicate that the interrupt was not asserted, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, that an external LAN connection is present and receiving traffic, that there are issues with the system chipset, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	The controller should not be connected to a network.

4.4.8 **External PHY Loopback Test (D8)**

The External PHY Loopback test will enable External PHY loopback mode (for now, only if external PHY type is SFX7101), then transmit about 9000 Layer-2 packets of various sizes. As the packets are received back by eDiag they are checked for errors.

Manufacturing Mode Command	C:\>ediag –t abcd –T d8
Engineering Mode Command	1:> extphy_test
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the external PHY is defective or that the chip is defective.
Enabled by Default	Only if external PHY type is SFX7101 (nvm cfg option 39)
Special Requirements	The controller should not be connected to a network.

4.5 Group E – PHY External Loopback Tests

4.5.1 SerDes PRBS External Loopback Test (E1)

The SerDes PRBS External Loopback Test performs a PRBS test using external loopback. The SerDes generates and collects patterns independently of other blocks.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

This test is excluded if SERDES external PHY is not connected.

Manufacturing Mode Command	C:\> ediag –none –T E1
Engineering Mode Command	1:> prbstest -l EXTERNAL –phy SERDES
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	Yes. This test is excluded if SerDes external PHY is not connected.
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.5.2 XGXS PRBS External Loopback Test (E2)

The XGXS PRBS External Loopback Test performs a PRBS test using external loopback. The XGXS generates and collects patterns independently of other blocks.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T E2
Engineering Mode Command	1:> prbstest -l EXTERNAL –phy XGXS
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed

	on the tested port.

4.6 Group F – Firmware Traffic Tests (via External Loopback)

4.6.1 Raw Packets with Random Payload Test (Number of BDs 1-13) (F1)

The Raw Packets with Random Payload Test includes 13 subcases, each subcase is with a different number of BDs (1-13).

Each subcase includes generation of a specification packet with random payload and the requested number of BDs, then transmission of numerous packets with packet lengths from 64-9018 bytes, then receiving the packets.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F1
Engineering Mode Command	1:> extlb_test –case 1 optional: -subcase <A..M>
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.6.2 Raw Packets with Fixed Payload Test (Number of BDs 1-13) (F2)

The Raw Packets with Fixed Payload Test includes 13 subcases, each subcase is with a different number of BDs (1-13).

Each subcase includes generation of a specification packet with fixed payload (0x55aa) and the requested number of BDs, then transmission of numerous packets with packet lengths from 64-9018 bytes. Then eDiag receives the packets and performs data-integrity check.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F2
Engineering Mode Command	1:> extlb_test –case 2

	optional: -subcase <A..M>
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.6.3 Raw Packets with Incremental Payload Test (Number of BDs 1-13) (F3)

The Raw Packets with Incremental Payload Test includes 13 subcases, each subcase is with a different number of BDs (1-13).

Each subcase includes generation of a specification packet with incremental payload and the requested number of BDs, then transmission of numerous packets with packet lengths from 64-9018 bytes. Then eDiag receives the packets and performs data-integrity check.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F3
Engineering Mode Command	1:> extlb_test –case 3 optional: -subcase <A..M>
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.6.4 Raw Packets with Four BDs, (X, 0, 0, X) Bytes in BD Test (F4)

The Raw Packets with Four BDs Test includes 13 subcases, each subcase includes generation of a specification packet with incremental payload and four BDs. The number of bytes in BD is according to the subcase (1,0,0,1) in the first case, meaning 1 byte in the first and last BD and rest of the bytes in the left BDs. (2,0,0,2) in the second case etc.

The test then transmits numerous packets with packet lengths from 64-9018 bytes. Then eDiag receives the packets and performs data-integrity check.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F4
Engineering Mode Command	1:> extlb_test –case 4 optional: -subcase <A..M>
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.6.5 Raw Packets with Four BDs, (2, 2, 0, 0) Bytes in BD Test (F5)

The Raw Packets with Four BDs (2, 2, 0, 0) Test generates a specification packet with incremental payload and four BDs (2, 2, 0, 0) – two bytes in the first two BDs and the rest of the bytes in the left BDs.

The test then transmits numerous packets with packet lengths from 64-9018 bytes. Then eDiag receives the packets and performs data-integrity check.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F5
Engineering Mode Command	1:> extlb_test –case 5
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.6.6 Raw Packets with Four BDs, (0, 3, 3, 0) Bytes in BD Test (F6)

The Raw Packets with Four BDs (0, 3, 3, 0) is similar to the previous test with the difference that the test generates a specification packet with incremental payload and four BDs (0, 3, 3, 0) - meaning 3 bytes in the second and third BDs and the rest of the bytes are in the first and last BDs.

4.6.7 Raw Packets with Four BDs, (0, 4, 4, 4) Bytes in BD Test (F7)

The Raw Packets with Four BDs (0, 4, 4, 4) is similar to the previous test with the difference that the test generates a specification packet with incremental payload and four BDs (0, 4, 4, 4) - meaning 4 bytes in the last BDs and the rest of the bytes are in the first.

4.6.8 Raw Packets with Four BDs, (5, 5, 5, 0) Bytes in BD Test (F8)

The Raw Packets with Four BDs (5, 5, 5, 0) is similar to the previous test with the difference that the test generates a specification packet with incremental payload and four BDs (5, 5, 5, 0) - meaning 5 bytes in the first BDs and the rest of the bytes are in the last BDs.

4.6.9 Raw Packets with VLAN Tag Test (F9)

The Raw Packets with VLAN Tag Test generates a specification packet with incremental payload (four BDs) and software inserted VLAN tag (0x55aa). The test then transmits numerous packets with packet lengths from 64-9018 bytes. Then eDiag receives the packets and performs data-integrity check.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F9
Engineering Mode Command	1:> extlb_test –case 9 –port <port# 0 or 1>
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.6.10 LLC Snap Packets w/o VLAN Tag Test (F10)

The LLC Snap Packets w/o VLAN Tag Test generates a specification packet with incremental payload (four BDs) and encapsulation mode 802.3llc (w/o VLAN tag). The test then transmits numerous packets with packet lengths from 64-1492 bytes. Then eDiag receives the packets and performs data-integrity check.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F10
Engineering Mode Command	1:> extlb_test –case 10
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.6.11 LLC Snap Packets with VLAN Tag Test (F11)

The LLC Snap Packets with VLAN Tag Test is similar to the previous test with the difference that the packets include VLAN tag.

4.6.12 TCP Packets Test (F12)

The TCP Packets Test transmits 100 packets with incremental payload and incremental length (starting with 64 bytes), then receives the data and performs data integrity check.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F15
Engineering Mode Command	1:> extlb_test –case 15
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.6.13 Raw packets with VLAN Offload Test (F13)

The Raw Packets with VLAN Offload Test generates a specification packet with incremental payload, 4 BDS, 1000 bytes and hardware inserted VLAN tag (0x55aa). The test transmits a packet of this type, receives it and performs a data integrity check.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F16
Engineering Mode Command	1:> extlb_test –case 16
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

4.6.14 LLC Snap Packets with VLAN Offload Test (F14)

The LLC Snap Packets with VLAN Offload Test is similar to the previous test with the use of packets with 802.3llc encapsulation mode instead of raw packets.

4.6.15 RX BD Ring Full test (F15)

The RX BD Ring Full Test transmits 500 packets of 1500 bytes and verifies that after transmission the TX ring is empty and the RX ring is full. Then the test receives the packets and verifies that the RX ring difference is similar to the difference saved before the test started.

This test is disabled by default, since it requires an external loopback cable. When enabled, this test is performed on both port 0 and port 1 of the Everest controller.

Manufacturing Mode Command	C:\> ediag –none –T F18
Engineering Mode Command	1:> extlb_test –case 18
Return Codes	0 – Test passed
Possible Failure Causes	A failure in this test may indicate that the chip is defective.
Enabled by Default	No
Special Requirements	This test requires that an external loopback connector be installed on the tested port.

5. eDiag Command Line Options

The followings are the available command line options in eDiag diagnostic tool. To be used in the command line, e.g. "eDiag –log results.log –rc regress.tcl"

OPTIONS	DESCRIPTION
-rc <script>	Specify a script to source right after startup.
-I <iteration#>	Specify how many iterations tests need to run (Default is 1).
-ver	Display information on eDiag version and exits.
-log <logfile>	Log the tests' execution results into the specified file.
-b10eng	Enter eDiag engineering mode, a prompt will be shown. The default mode is manufacture mode, in which nictest tests will start automatically. A detailed description of the commands available in engineering mode is available in section 7. This option is similar to –b06eng (which is also supported for backwards competablitity).
-no_pci	Do not use the PCI. In this mode, the user can enable the Debug UART for GRC access instead of PCI.
-noinit	Start eDiag without pre-selecting any device. (Default selected device is 1)
-dev <device#>	Select device number that the tests will be running on. In manufacture mode, tests will run on both devices. In engineering mode, enables to select a device. This option is similar to –c, which was used in Teton eDiag tool.
–pwd <password>	Specify password to update MAC address and NVM. Otherwise, user is not authorized to use –m, -mac, -iscsimac, -rdma and –nvm options.
-mac <mac>	Specify the primary MAC address (in format xx:xx:xx:xx:xx:xx)
-iscsimac <mac>	Specify the iSCSI MAC address (in format xx:xx:xx:xx:xx:xx)
-rdmamac <mac>	Specify the RDMA MAC address (in format xx:xx:xx:xx:xx:xx)
-m	Interactive add the MAC addresses (primary and iSCSI MAC addresses for both devices).
–autom	Automatically generate iSCSI and a secondary MAC address.
–num_mac	Number of MAC addresses for device <2 or 4>. Works with –autom option. Determines the gap between the MAC addresses of sequential devices.

	Default is 2, thus no gap between the MAC addresses of sequential devices.
-wol <1|0>	Enable (1) / Disable(0) magic packet Wake-on-LAN option.
-mba <1|0>	Enable (1) / Disable(0) multiple boot agent.
-mbap <n>	Specify MBA boot protocol: PXE(0), RPL(1), BOOTP(2), iSCSI_boot(3), FCoE_boot(4).
-mbav <1|0>	Enable(1) / Disable(0) MBA VLAN.
-mbavval <n>	Specify MBA VLAN value (< 65536)
-mbabr <0-7>	MBA boot retry count (< 8)
-mfw <1|0>	Enable(1) / Disable(0) management firmware agent.
-sn <number>	Specify serial number for the card.
-t <grps/tests>	Disable certain groups/tests (e.g. -t A5).
-T <grps/tests>	Enable certain tests/groups (e.g. -T A5)
-cof	Allow tests to continue tests on failure.
-none	Disable all tests.
-slave	Select device number to be in host loopback mode. The device is simply echoing all received packets, and not operable for other purposes.
-fbc <bc_image>	Specify the bin file for combined boot code.
-fbc1 <bc1_image>	Specify the bin file for boot code 1.
-fbc2 <bc2_image>	Specify the bin file for boot code 2.
-fl2b <bc2_image>	Specify the bin file for L2 firmware.
-fipmi <ipmi_image>	Specify the bin file for IPMI firmware.
-fump <ump_image>	Specify the bin file for UMP firmware (Similar to using "nvm upgrade –ump <UMP image>").
-fmba <mba_image>	Specify the bin file for MBA.
-fncsi <ncsi_image>	Specify the bin file for NCSI firmware.
-fib <ib_image>	Specify the bin file for iSCSI boot.
-fibp	Program iSCSI configuration utility, used with -fib <ib_image>
-ffeb <fcoe_file>	Specify the bin file for FCoE boot.
-ffebc <fcoe_file>	Specify the bin file for FCoE boot and config block FEB_CFG.

-ffebp <fcoe_file>	Specify the bin file for FCoE boot and config program FEB_CPRG.
-ffebcp <fcoe_file>	Specify the bin file for FCoE boot, config block (FEB_CFG) and config program (FEB_CPRG).
-fnvm <raw_image>	Program raw image into NVM.
–fnvm <img1> <img2>	Incase of -asiccnt 2, support two NVM images for ASIC1 and ASIC2.
-raw	Program complete NVM raw image, used with –fnvm
-skip_ec	Preserve EC field in VPD block, used with –fnvm
-skip_sn	Preserve SN field in VPD block, used with -fnvm
-idmatch	Enable matching of VID, DID, SVID, and SSID from image file with device's IDs. Used with "-fnvm <raw_image>" only.
-F	Force to upgrade image without checking version.
-fmac <file>	Specify a file that contains the MAC addresses to be programmed. Refer to section 8 for more information on the format of the macaddr.txt file.
-cfgchk <board type>	Check NVRAM configuration against expected default Setting of this board type.
-nvmchk	Perform NVRAM image CRC checks
-arg	Provide arbitrary arguments to be used by scripts
–pnchk	Check part number against input value from stdin
-sysop	Run a set of nictests on several cards, with hot-swap between Cards.
–mkey <file>	Program manufacture license key file
-ukey <file>	Program upgrade license key file
-chkkey <sm|su|smu>	Validate manufacture, upgrade, or both license key(s)
-fextphy <image file>	Upgrade external PHY firmware with ext_phy_image.
-dcbx <1|0>	Enable (1) / Disable(0) DCBX protocol.
–asiccnt	Indicates the number of Everest ASICs on board (for –sysop mode)
-nicp_mac	Automatically generate NICP MAC addresses (for –sysop mode)

-stride	Indicates primary LAN MAC address alignment (for –sysop mode)
-no_swap	No hot-swap between boards, automatically exit after testing (for –sysop mode)

-fmfw <mfw_image> : Specify the bin file for MFW code

-help	Prints out this help

Notes:

1. Command lines are performed by default on all devices installed on the machine. In the event the devices were selected by –dev or –c, the command line will be performed on the selected devices only.

2. For command lines: –fnvm, -fbc, -fl2b, -fipmi, -fump, -fmfw the command line will be performed on the primary devices only, since these are shared options between ports. For example on two-port board, the –fnvm option will be performed on port 0 only, since there is no need to repeat the NVM programming on port 1.

6. Return Codes

The followings are the available return codes upon exit of eDiag.

The return codes are backward compatible with xDiag return codes with additional return codes specific for Everest (starting at 101).

Unused return codes are reserved for xDiag backward compatibility.

RETURN CODE	VALUE	DESCRIPTION
ERR_NONE	00	No error
ERR_REG_TEST	01	Register test fails
ERR_PCICFG_TEST	02	PCI config test fails
ERR_INTR_TEST	03	Interrupt test fails
ERR_PCIE_LINK_TEST	04	PCIE link test fails
ERR_MSI_TEST	05	MSI test fails
ERR_NETLINK_TEST	07	Network link test fails
ERR_CAM_TEST	28	CAM test fails
ERR_TIMER_TEST	29	Timer test fails
ERR_NVRAM_TEST	32	NVRAM test fails
ERR_DMAE_TEST	34	DMA engine test fails
ERR_MACLB_TEST	38	MAC loopback test fails
ERR_PHYLB_TEST	39	PHY loopback test fails
ERR_EXTLB_TEST	40	External loopback test fails
ERR_LSO_TEST	41	LSO traffic test fails
ERR_STAT_TEST	42	Statistic test fails
ERR_RPC_TEST	43	RPC test fails
ERR_BAD_PASSWD	44	Invalid password
ERR_BAD_ARGS	45	Invalid command switch(es)
ERR_NO_DEV_FOUND	46	Operation(s) skipped, no device found
ERR_NOT_AUTH	47	Not authorized to perform the operation(s)
ERR_ONE_DEV_ONLY	48	Specify one device, no more, no less
ERR_MAC_REDUND	49	Redundant MAC address input methods
ERR_FMAC_BAD_FILE	50	Bad MAC address range file (not exist, bad format)
ERR_FMAC_RUN_OUT	51	All MAC addrress have been used up in the range file
ERR_FMAC_UPDATE	52	Fail to update the MAC address range file
ERR_PROMPT_MAC	53	Fail to input MAC address
ERR_MAC_TOO_DIFF	54	Primary and iSCSI MAC addresses are not consecutive
ERR_SAME_MAC_ADDR	55	Primary and iSCSI MAC addresses are the same
ERR_PRG_NVM_IMAGE	56	Fail to program NVRAM image
ERR_PRG_PRIM_MAC_ADDR	57	Fail to program primary MAC address
ERR_PRG_ISCSI_MAC_ADDR	58	Fail to program iSCSI MAC address
ERR_SEL_DEV	59	Fail to select the device for operation(s)
ERR_PRG_BC	60	Fail to program boot code
ERR_PRG_MFW	61	Fail to program IPMI/UMP firmware
ERR_PRG_MBA	62	Fail to program MBA image
ERR_CFG_WOL	63	Fail to configure WOL
ERR_CFG_MFW	64	Fail to configure IPMI/UMP firmware
ERR_CFG_MBA	65	Fail to configure MBA image
ERR_PN_NOT_FOUND	66	Fail to extract part number from device
ERR_PN_BAD_INPUT	67	Fail to capture part number from input
ERR_PN_MISMATCH	68	Part number does not match
ERR_CFG_MISMATCH	69	Configuration mismatch
ERR_RESTORE_PCI	70	Fail to restore pci info to the next board
ERR_USER_ABORT	71	User abort
ERR_PRG_MKEY	72	Fail to program manufacturing license key
ERR_PRG_UKEY	73	Fail to program upgrade license key
ERR_CHKKEY_SS	74	Fail to validate the SS portion of the key
ERR_CHKKEY_MK	75	Fail to validate the manufacturing license key
ERR_CHKKEY_UK	76	Fail to validate the upgrade license key

ERR_NVRAM_CRC	77	CRC check fails on one of the NVRAM blocks
ERR_CFG_MBA_SPD	78	Fail to configure MBA link speed
ERR_CFG_MBA_PROT	79	Fail to configure MBA protocol parameter
ERR_CFG_MBA_VLAN	80	Fail to configure MBA VLAN
ERR_CFG_MBA_VLAN_VAL	81	Fail to configure MBA VLAN value
ERR_MSIX_TEST	90	MSIX test fails
ERR_REG_BRB1_TEST	101	BRB1 Register test fails
ERR_REG_CCM_TEST	102	CCM Register test fails
ERR_REG_CDU_TEST	103	CDU Register test fails
ERR_REG_CFC_TEST	104	CFC Register test fails
ERR_REG_CSDM_TEST	105	CSDM Register test fails
ERR_REG_CSEM_TEST	106	CSEM Register test fails
ERR_REG_DBG_TEST	107	DBG Register test fails
ERR_REG_DMAE_TEST	108	DMAE Register test fails
ERR_REG_DORQ_TEST	109	DORQ Register test fails
ERR_REG_HC_TEST	110	HC Register test fails
ERR_REG_MISC_TEST	111	MISC Register test fails
ERR_REG_NIG_TEST	112	NIG Register test fails
ERR_REG_PBF_TEST	113	PBF Register test fails
ERR_REG_PRS_TEST	114	PRS Register test fails
ERR_REG_PXP_TEST	115	PXP Register test fails
ERR_REG_QM_TEST	116	QM Register test fails
ERR_REG_SRC_TEST	117	SRC Register test fails
ERR_REG_TCM_TEST	118	TCM Register test fails
ERR_REG_TM_TEST	119	TM Register test fails
ERR_REG_TSDM_TEST	120	TSDM Register test fails
ERR_REG_TSEM_TEST	121	TSEM Register test fails
ERR_REG_UCM_TEST	122	UCM Register test fails
ERR_REG_UPB_TEST	123	UPB Register test fails
ERR_REG_USDM_TEST	124	USDM Register test fails
ERR_REG_USEM_TEST	125	USEM Register test fails
ERR_REG_XCM_TEST	126	XCM Register test fails
ERR_REG_XPB_TEST	127	XPB Register test fails
ERR_REG_XSDM_TEST	128	XSDM Register test fails
ERR_REG_XSEM_TEST	129	XSEM Register test fails
ERR_REG_EMAC_TEST	130	EMAC Register test fails
ERR_REG_MCP_TEST	131	MCP Register test fails
ERR_MEM_BRB1_TEST	132	BRB1 Memory test fails
ERR_MEM_CCM_TEST	133	CCM Memory test fails
ERR_MEM_CDU_TEST	134	CDU Memory test fails
ERR_MEM_CFC_TEST	135	CFC Memory test fails
ERR_MEM_CSDM_TEST	136	CSDM Memory test fails
ERR_MEM_CSEM_TEST	137	CSEM Memory test fails
ERR_MEM_DBG_TEST	138	DBG Memory test fails
ERR_MEM_DMAE_TEST	139	DMAE Memory test fails
ERR_MEM_DORQ_TEST	140	DORQ Memory test fails
ERR_MEM_HC_TEST	141	HC Memory test fails
ERR_MEM_NIG_TEST	142	NIG Memory test fails
ERR_MEM_PBF_TEST	143	PBF Memory test fails
ERR_MEM_PXP_TEST	144	PXP Memory test fails
ERR_MEM_QM_TEST	145	QM Memory test fails
ERR_MEM_SRC_TEST	146	SRC Memory test fails
ERR_MEM_TCM_TEST	147	TCM Memory test fails
ERR_MEM_TM_TEST	148	TM Memory test fails
ERR_MEM_TSDM_TEST	149	TSDM Memory test fails
ERR_MEM_TSEM_TEST	150	TSEM Memory test fails
ERR_MEM_UCM_TEST	151	UCM Memory test fails
ERR_MEM_UPB_TEST	152	UPB Memory test fails
ERR_MEM_USDM_TEST	153	USDM Memory test fails
ERR_MEM_USEM_TEST	154	USEM Memory test fails
ERR_MEM_XCM_TEST	155	XCM Memory test fails
ERR_MEM_XPB_TEST	156	XPB Memory test fails

ERR_MEM_XSDM_TEST	157	XSDM Memory test fails
ERR_MEM_XSEM_TEST	158	XSEM Memory test fails
ERR_CAM_SEARCH_TEST	159	CAM test fails
ERR_GRC_TEST	160	GRC test fails
ERR_DBG_PXP_TEST	161	Debug bus w out to PXPX test fails
ERR_SERDES_REG_TEST	162	Serdes Register test fails
ERR_XGXS_REG_TEST	163	XGXS Register test fails
ERR_SERDES_INTLB_TEST	164	Serdes Internal loopback test fails
ERR_XGXS_INTLB_TEST	165	XGXS Internal loopback test fails
ERR_TOE_TEST	166	TOE test fails
ERR_EXTPHY_TEST	167	External PHY Loopback test fails
ERR_SERDES_EXTLB_TEST	168	Serdes External loopback test fails
ERR_XGXS_EXTLB_TEST	169	XGXS External loopback test fails
ERR_EXTLB_CASE1_TEST	170	External loopback case1 test fails
ERR_EXTLB_CASE2_TEST	171	External loopback case2 test fails
ERR_EXTLB_CASE3_TEST	172	External loopback case3 test fails
ERR_EXTLB_CASE4_TEST	173	External loopback case4 test fails
ERR_EXTLB_CASE5_TEST	174	External loopback case5 test fails
ERR_EXTLB_CASE6_TEST	175	External loopback case6 test fails
ERR_EXTLB_CASE7_TEST	176	External loopback case7 test fails
ERR_EXTLB_CASE8_TEST	177	External loopback case8 test fails
ERR_EXTLB_CASE9_TEST	178	External loopback case9 test fails
ERR_EXTLB_CASE10_TEST	179	External loopback case10 test fails
ERR_EXTLB_CASE11_TEST	180	External loopback case11 test fails
ERR_EXTLB_CASE12_TEST	181	External loopback case12 test fails
ERR_EXTLB_CASE13_TEST	182	External loopback case13 test fails
ERR_EXTLB_CASE14_TEST	183	External loopback case14 test fails
ERR_EXTLB_CASE15_TEST	184	External loopback case15 test fails
ERR_EXTLB_CASE16_TEST	185	External loopback case16 test fails
ERR_EXTLB_CASE17_TEST	186	External loopback case17 test fails
ERR_EXTLB_CASE18_TEST	187	External loopback case18 test fails
ERR_EXTLSO_BASIC_TEST	188	External LSO basic test fails
ERR_EXTLSO_EXTND_TEST	189	External LSO extended test fails
ERR_EXTLSO_SNAP_TEST	190	External LSO snap test fails
ERR_EXTLSO_SNAP_EXTND	191	External LSO snap extended test fails
ERR_PRG_RDMA_MAC_ADDR	192	Fail to program iSCSI MAC address
ERR_CFG_SERIAL_NO	193	Fail to configure serial number
ERR_PRG_L2B	194	Fail to program L2B image
ERR_PRG_IB	195	Fail to program iSCSI boot image

7. Engineering Mode Commands

The command line interface available in engineering mode is based on the Tool Command Language or TCL. (Refer to Appendix A – Tcl Reference for additional information on TCL.) In addition to the core TCL commands such as set, return, expr, if/else, while, etc., additional commands have been added to provide specific support for interacting with the Everest controller.

All of the commands available in engineering mode use a Tcl syntax of the form "command arg1 arg2 arg3 ...". Every command has a return value. To start eDiag in engineering mode, use the following DOS command:

C:\> ediag –b10eng

Notes:

1. When a command uses a number as command line argument, the number can be specified either in decimal form or hexadecimal form (prefix with 0x).

2. Commands that require a filename (such as the load firmware command) should use the UNIX standard forward-slash (or "/") rather than the DOS style backward-slash (or "\") when using directories outside of the eDiag working directory. It is also possible to provide the path using a double DOS style backward-slash (or "\\").

This section shows the available commands in the eDiag tool, along with their usage and examples. **The commands are listed in alphabetic order**.

7.1 Add_help

Usage: **add_help <command name> <description string>**

Description: This command can be used by TCL script writers to add help on new commands written in the TCL script.

Example: `add_help parse_rdf "Parses the RDF file into global arrays"`

7.2 Assert

7.2.1 Assert last_index

Usage: **assert last_index_offset <instance>**

Description: This command returns the offset address of the last assert index. The command's input is the instance "t" (for TSEM), "x" (for XSEM), "u" (for USEM) or "c" (for CSEM).

Example: `asset last_index_offset t`

7.2.2 Assert get_offset

Usage: **assert get_offset <instance> <index>**

Description: This command returns the offset address of an assert by its index number. The command's input is the instance "t" (for TSEM), "x" (for XSEM), "u" (for USEM) or "c" (for CSEM) and the index of the assert.

Example: `asset get_offset t 1`

7.3 Bar

7.3.1 Bar read

Usage: **bar <index> read <addr> <nbytes>**

Description: This command reads from bar (index 0 or 1) from address <addr> a number of bytes <nbytes>. The number of bytes can be 1, 2 or 4.

Example:

```
1:> bar 0 read 0x0 4
0x164a14e4
```

7.3.2 Bar write

Usage: **bar <index> write <addr> <value> <nbytes>**

Description: This command writes to bar (index 0 or 1), to address <addr> the value <value>. The number of bytes can be 1, 2 or 4.

Example:

```
1:> bar 0 write 0x0 0x164a14e4 4
```

7.3.3 Bar show

Usage: **bar <index> show <addr> <nbytes>**

Description: This command shows a number of bytes <nbytes> from bar (index 0 or 1) address <addr>. The number of bytes can be any number of bytes.

Example:

```
1:> bar 0 show 0x0 10
00000000:164a14e4 02b0011e 02000002
```

7.4 Bits

7.4.1 Bits clear

Usage: **bits clear <***bit_pos [bit_pos…]***>**

Description: This command clears a bit position *bit_pos* (0-31) in a 32-bit hexadecimal number to 0 and the remaining bits to 1. It then displays the value.

Example:

```
1:> bits clear 0 31
0x7ffffffe
```

7.4.2 Bits set

Usage: **bits set** <*bit_pos [bit_pos…]*>

Description: This command sets a bit position *bit_pos* (0-31) in a 32-bit hexadecimal number to 1 and the remaining bits to 0. It then displays the value.

Example:

```
1:> bits set 2 31
0x80000004
```

7.4.3 Debug

Usage: **debug set <*path*|*level*> <*value*>**

Description: The debug command enables to change the debug message level or path.

The debug levels are:

VERBOSE – All debug messages will be shown to screen.

INFORM – Debug messages with INFORM level and above will be shown to screen.

WARN – Debug messages with WARN level and above will be shown to screen.

FATAL – Only fatal debug messages will be shown to screen.

The debug messages paths are:

INIT, NVM, L2_SP, L2_SEND, L2_RECV, L2_INTERRUPT, L2, L4_SP, L4_SEND, L4_RECV, L4, L5_SP, L5_SEND, L5_RECV, L5, DIAG, MISC , ALL

Example:

```
1:> debug set level INFORM

1:> debug set path CP_ALL
```

7.5 Delay

Usage: **delay [us | ms]** <*count*>

Description: Used by script to add delay of *count* microseconds or milliseconds.

Example: `delay us 5`

7.6 Device

Usage: **device** [<*devNum*>]

Description: Without an argument, this command displays some brief device information of all devices it detects. When *devNum* is provided, the device with the given number will be set as "**current**" device. In this case the DUT address (see **pci setdut**) will be set to the address of the selected device.

This command enables the user to move to the second device (port) using "device 2".

```
Example: 1:> device
 C:  Brd:Rv    Bus      PCI     Spd Base   IRQ     MAC     FmwVer  Config.
 1: 57710:A0    02:00:00 PCIE-4  2.5 0xFA00 11 001018112233 4.0.2   auto
   T1001 v2.0 p2                    0xFA80    S.N.:BCM957710A0
 2: 57710:A0    02:00:01 PCIE-4  2.5 0xFB00 10 00101807FBB1 4.0.2   auto
   T1001 v2.0 p2

1:> device 2
2:> exit
```

7.7 Disp64

Usage: disp64 signed/unsigned <*high_val*> <*low_val*>

Description: This command combines a 32-bit number high_val as the upper 32 bits of a 64-bit number with another 32-bit number low_val as the lower 32 bits of a 64-bit number. Then returns the 64-bit value.

Example:

```
1:> disp64 signed 9 2
38654705666

1:> disp64 unsigned 15 4
64424509444
```

Description: These commands are for internal use only.

7.8 Driver

The driver commands are used to change the driver state.

7.8.1 Driver setup

Usage: driver setup

Description: This command is used to setup the driver.

7.8.2 Driver start

Usage: driver start

Description: This command starts the driver operation.

7.8.3 Driver load

Usage: driver load

Description: This command sets up the driver and starts its operation. It replaces the use of "driver setup" and "driver start".

7.8.4 Driver intr

Usage: driver intr <enable flag>

Description: This command enables (enable flag = 1) or disables (enable flag = 0) the interrupts.

7.8.5 Driver strip_crc

Usage: **driver strip_crc <enable flag>**

Description: This command enables (enable flag = 1) or disables (enable flag = 0) the mode when driver assumes CRC stripped from L2 packet (set by default).

7.8.6 Driver intrcnt

Usage: **driver intrcnt**

Description: This command shows the number of times the ISR was called.

7.8.7 Driver link_state

Usage: **driver link_state**

Description: This command returns the last indicated link state. The state is returned in the form of list {<Media type name>, <Media speed>, <Duplex mode>, <UP|DOWN>}.

Example:

1:> driver link_state

MAC_LOOPBACK 10GBPS FULL_DUPLEX DOWN

1:>

7.8.8 Driver poll

Usage: **driver poll**

Description: This command changes the driver's mode to polling mode and triggers an immediate call to the ISR.

7.8.9 Driver unload

Usage: **driver unload**

Description: This command unloads the driver and returns the driver into state `DRIVER_STATE_SETUP`.

7.8.10 Driver disable_stat

Usage: **driver disable_stat**

Description: This command disables the periodic collection of statistics.

7.8.11 Driver enable_stat

Usage: **driver enable_stat**

Description: This command enables the periodic collection of statistics.

7.8.12 Driver stat

Usage: **driver stat <high array name> <low array name>**

Description: creates 2 tcl Associative Arrays with the given names, and inserts the statistics into them, with field name as the key value, and the statistics value as the array value. The values kept in each array are 32 bits in width. Totally, 64 bits of information will be stored for each statistics field.

7.9 Ext_phy_fw

External PHY firmware operations

7.9.1 Ext_phy_fw upgrade

Usage: **ext_phy_fw upgrade <fileName>**

<fileName> file name of the PHY firmware to upgrade

-F flag to force driver load prior to ext_phy_fw upgrade

Description: Upgrade the external PHY firmware.

7.9.2 Ext_phy_fw version

Usage: **ext_phy_fw version**

Description: Reads the external PHY firmware version.

7.10 exit

Synopsis: exit

Description: This command will exit eDiag and return to an operating system.

Example:

```
1:> exit
C:\>
```

7.11 Gpio

The "gpio" command allows the user to directly manipulate the state of the GPIO pins supported on the Everest controller. Since the use of the GPIO pins is generally system specific, users should use caution when directly manipulating GPIO pins as they may directly affect the hardware operation of the system.

7.11.1 Gpio read

Usage: **gpio read** *[optional: < pin>]*

Description: This command reads from GPIO pin 0,1,2 or 3. If no <pin> input all pins are read.

Examples:

```
1:> gpio read 1
```

```
1:> 0
1:> gpio read
1:> 0: 0 1 0 1
```

7.11.2 Gpio write

Usage: **gpio write** < *pin*> < *value*>

gpio write < *pin_n*> *{*< *value_n*> <*value_n+1*> <*value_n+2*>*}*

Description: This command writes to GPIO pin 0,1,2 or 3 the <value> 0 or 1. There is also the option to write from <pin_n> the values <value_n> <value_n+1> <value_n+2> to the next pins.

Examples:

```
1:> gpio write 0 1
1:> gpio write 0 {1 0 1}
1:> gpio read
```

7.12 Help

Usage: **help [***<command name>]*

Description: With no arguments, the help command will list all the eDiag commands in an alphabetic order and a short description on each command.

When a command name is used as an input argument, the help command will show a short description on this command and its usage.

Note: The help command will also show help on new commands written in TCL scripts, on which the script writer has used the add_help to add help on these new commands.

7.13 hex

The "hex" command allows the user to create hexadecimal numbers from decimal numbers.

7.13.1 hex16

Usage: hex16 <value>

Description: Displays a decimal number in 16-bit hexadecimal format.

Returns: The 16-bit hexadecimal value of the decimal argument.

Example:

```
1:> hex16 20
0014
```

7.13.2 hex32

Usage: hex32 <value>

Description: Displays a decimal number in 32-bit hexadecimal format.

Returns: The 16-bit hexadecimal value of the decimal argument.

Example:

```
1:> hex32 20
00000014
```

7.14 hmem

7.14.1 hmem alloc

Usage: **hmem alloc** <*bytecount*> *[<description_string>]*

Description: This command allocates a host memory region of *bytecount* bytes. If *description_string* is specified, the allocated memory will associate with the *description_string* name. *description_string* is limited to 30 characters with no space in between. It returns the virtual address of the buffer. Each double word of the allocated buffer is initialized to 0xdeadbeef.

Example:

```
1:> hmem alloc 100 MyMemory
0x15bae58
1:>
```

7.14.2 hmem fill

Usage: **hmem fill** <*low32addr*> *[<high32addr>]* <*dword_count*> <*pattern*>

Description: This command fills host memory with *dword_count* of double words at specified address. The < *pattern*> can be any 32-bit value or it can be the keyword **increment,** which will increment each byte value by 1, starting at 0.

Example:

```
1:> hmem fill 0x6000 20 0x12121212
```

7.14.3 hmem free

Usage: **hmem free** <*low32addr*> *[<high32addr>]*

Description: This command frees the allocated host memory region at specified address.

Example:

```
1:> hmem free 0x15bae58
```

7.14.4 hmem inuse

Usage: **hmem inuse**

Description: This command displays all the allocated host memory regions by **hmem** command.

Example:

```
1.MCP:> hmem inuse
(   112,015bae54/00000000) 015bae54/00000000      100 bytes(Virt) MyMemory
```

Inside () is only for debugging. The next value is the virtual address, next oen being physical (in DOS virtual == physical).

7.14.5 hmem paddr

Usage: **hmem paddr** <*low32addr*> *[<high32addr>]*

Description: This command returns the physical address of a buffer with virtual address (applicable only when memory is allocated using "**hmem palloc**" command).

Example:

```
1:> hmem palloc 200 Mybuf
0x15baf10
1:>paddr 0x15baf10
0x15baf10
```

7.14.6 hmem palloc

Usage: **hmem palloc** <*bytecount*> *[<description_string>]*

Description: This command allocates a continuous memory region of *bytecount* bytes. If *description_string* is specified, the allocated memory will associate with the *description_string* name. *description_string* is limited to 30 characters with no space in between. It returns the virtual address of the buffer.

Example:

```
1:> hmem palloc 48
0x15bb9b8
```

7.14.7 hmem read

Usage: **hmem read** <*low32addr*> *[<high32addr>]*

Description: This command reads a double word at specified address from host memory.

Example:

```
1:> hmem read 0
0xa1b0a61
```

7.14.8 hmem show

Usage: **hmem show** <*low32addr*> *[<high32addr>]* <*byte_count*>

Description: This command displays a number bytes specified by *byte_count* (multiple 0f 4) of host memory at specified host memory address.

Example:

```
1:> hmem show 0 20
0000000:0a1b3434 007006f4 1c33f416 00700601 007006f4
```

7.14.9 hmem write

Usage: **hmem write** <*low32addr*> *[*<*high32addr*>*]* <*value*>

Description: This command writes double word at specified address to host memory.

Example: `hmem write 0 0x34343434`

7.15 ioport

7.15.1 ioport read

Usage: **ioport read byte|word|dword** <*addr*>

Description: This command either reads a byte, a word, or a double word from port address.

Example:

```
1:> ioport read byte 0xc78
0x55
1:> ioport read word 0xc78
0x5555

1:> ioport read dword 0xc78
0x55555555
```

7.15.2 ioport write

Usage: ioport write byte|word|dword <*addr*> <*value*>

Description: This command either writes a byte, a word, or a double word to port address.

Example:

```
1:> ioport write byte 0xc78 0x55
1:> ioport write word 0xc78 0x5555
1:> ioport write dword 0xc78 0x55555555
```

7.16 keyhit

The "keyhit" command is used in scripts to pause the script action until a keyboard key has been pressed.

Synopsis: keyhit <*char_str*>

Description: This command checks for keyboard activity, used in script environment. When the optional *char_str* argument is used, only a key in the argument will allow the keyhit command to return.

Returns: 0 when a key has been pressed.

Example:

```
keyhit
keyhit yn

while { 1 } {
    # Check for ESC character only
    if {[keyhit "\x1b"]!= ""} {
```

```
            break
        }
    }
```

7.17 L2spec

This section will describe the l2spec commands used to generate an L2 packet specification.

7.17.1 L2spec gen

Usage: **l2spec gen**

Description: This command generates and returns an L2 packet specification variable with the default values.

Examples:

```
1:> l2spec gen
L2spec:0x002184c
```

7.17.2 L2spec update

Usage: **l2spec update [list <parameter> <value>]**

Description: This command enables the user to change a certain parameter's value in the L2 packet specification variable.

Examples:

```
1:> set myl2spec [l2spec gen]
L2spec:0x002184c
1:> l2spec update $myl2spec [list maxBDs 5]
1:> l2spec update $myl2spec [list l3type ip iptype tcp]
1:> l2spec update $myl2spec [list ipDA "1.1.16.16" ipSA "1.1.16.17"]
```

(*) This command enables also to generate an IPv6 packet specification, and updating the IPv6 header fields - ipv6Priority, ipv6FlowLabel, ipv6NextHdr, ipv6HopLimit, ipv6DA, ipv6SA.

```
1:> set spc [l2spec gen]
L2Spec: 0x00547e9c
1:> l2spec update $spc [list encapmode ethernetII l3type ipv6 iptype tcp]
1:> l2spec update $spc [list ipv6SA
5555:6666:7777:8888:9999:AAAA:BBBB:CCCC:DDDD]
```

7.17.3 L2spec dump

Usage: **l2spec dump <l2spec variable name>**

Description: This command enables the user to dump the content of the L2 packet specification variable.

Examples:

```
1:> l2spec dump $myl2spec
1:>         pktCnt 1 maxPktLen 1518 minPktLen 64 pktLenMode fixed
            encapMode ethernetII vlanMode false vlanTag 0x0000
            vlanHwInsert hardware macDA ff:ff:ff:ff:ff:ff macSA
            00:10:18:00:00:00 l3Type ip ipType tcp ipDA 1.1.16.16
```

```
ipSA 1.1.16.17 ipOptionSize 0 ipRandOption true
ipChksumGen software ipChksumVal 0x0000 tcpDP 200
tcpSP 100 tcpOptionSize 0 tcpRandOption false
tcpChksumGen software tcpChksumVal 0x0000 tcpSeq none
tcpAck none tcpPush 0 SN none crcGen hardware crcVal
0x00000000 filtered false minIPG 12 maxIPG 12 ipgMode
fixed pauseTime 65535 payloadPttn increment maxBDs 5
randBDs false minBdSize 1 coalesceNow false
```

The dump result of an L2 IPv6 packet specification:

```
1:> l2spec dump $spc
            pktCnt 1 maxPktLen 1518 minPktLen 64 pktLenMode fixed
            encapMode ethernetII vlanMode false vlanTag 0x0000
            vlanHwInsert hardware macDA ff:ff:ff:ff:ff:ff macSA
            00:10:18:00:00:00 l3Type ipv6 ipType tcp ipDA 1.2.3.4 ipSA
            4.3.2.1 ipOptionSize 0 ipRandOption true ipChksumGen
            software ipChksumVal 0x0000 ipTTL 128 ipv6Priority 0
            ipv6FlowLabel 0x000000 ipv6NextHdr tcp ipv6HopLimit 128
            ipv6DA 1111:2222:3333:4444:5555:6666:7777:8888 ipv6SA
            5555:6666:7777:8888:9999:aaaa:bbbb:cccc tcpDP 200 tcpSP 100
            tcpOptionSize 0 tcpRandOption false tcpChksumGen software
            tcpChksumVal 0x0000 tcpSeq none tcpAck none tcpPush 0 SN
            none crcGen hardware crcVal 0x00000000 filtered false
            minIPG 12 maxIPG 12 ipgMode fixed pauseTime 65535
            payloadPttn increment maxBDs 3 randBDs false minBdSize 1
            coalesceNow false
```

7.18 L2pkt

This section will describe the l2pkt commands used to generate an L2 packet out of an L2 specification packet.

7.18.1 L2pkt gen

Usage: **l2pkt gen <l2spec variable> <pkt list name>**

Description: This command generates and returns a list of L2 packets according to the input L2 packet specification.

Examples:

```
1:> set myl2spec [l2spec gen]
L2spec:0x002184c
1:> l2pkt generate $myl2spec pktsList
1:> {L2packet: 0x00210e74 0x00210ee4}
```

7.18.2 L2pkt phys_addr

Usage: **l2pkt phys_addr <pkt variable>**

Description: This command returns the physical address of the first data byte of the packet.

The user should choose the required packet variable from the packet list by its index using the TCL lindex command.

Examples:

```
1:> set myl2spec [l2spec gen]
L2spec:0x00201c7c
1:> l2pkt gen $myl2spec pktsList
{L2packet: 0x00212050 0x002120c0}
1:> set mypkt [lindex $pktsList 0]
1:> l2pkt phys_addr $mypkt
1:> 217072
```

7.18.3 L2pkt virt_addr

Usage: **l2pkt virt_addr <pkt variable>**

Description: This command returns the virtual address of the first data byte of the packet.

The user should choose the required packet variable from the packet list by its index using the TCL lindex command.

Examples:

```
1:> set myl2spec [l2spec gen]
L2spec:0x00201c7c
1:> l2pkt gen $myl2spec pktsList
{L2packet: 0x00212050 0x002120c0}
1:> set mypkt [lindex $pktsList 0]
1:> l2pkt virt_addr $mypkt
1:> 217072

To see the actual contect of the packet data, use:
1:> hmem disp 217072 25
0x2120c0: ffffffff 1000ffff 00000018 01000000 05040302 09080706
```

7.18.4 L2pkt send <l2pkt list / l2pkt>

Usage: **l2pkt send <l2pkt list / l2pkt>**

Description: This command is used to send an L2pkt list or a single L2pkt.

Examples:

```
1:> set myl2spec [l2spec gen]
L2spec:0x00201c7c
1:> l2pkt gen $myl2spec pktsList
{L2packet: 0x00212050 0x002120c0}
1:> set mypkt [lindex $pktsList 0]
1:> l2pkt send $mypkt
1:> l2pkt send $pktsList
```

7.18.5 L2pkt receive

Usage: **l2pkt receive <number of packets>**

Description: This command is used to receive <number_of_packets> L2 packets.

If there is no input argument, only one L2 packet is received.

Example:

```
1:> set received_pkts [l2pkt receive 10]
1:> set pkt [lindex $received_pkts 0]
1:> l2pkt dump $pkt
```

7.18.6 L2pkt check

Usage: **l2pkt check <l2spec variable> <pkt variable>**

Description: This command is used to verify the L2 packet against the L2 packet specification. The command decomposes the packet into its protocol components and verifies each component against the packet specification component and returns the component which differs.

Examples: In the example below the macDA, payload and macCrc components differ.

```
1:> set mypkt [lindex $pktsList 0]
{L2Packet: 0x00212050 0x002120c0}
1:> l2pkt check $myl2spec $mypkt
macDA {ff:ff:ff:ff:ff:ff (ref) vs. aa:aa:aa:aa:aa:aa (src)}
payload {46 differences} macCrc {0x34b30d7c (ref) vs. 0x00000003
(src) CRC error (s/w);no CRC check indication}
```

7.18.7 L2pkt dump

Usage: **l2pkt dump <l2spec variable>**

Description: This is an additional command which is useful for testing and integration. The command decodes a packet into an array. The array fields are detailed in the table below.

Examples: In the example below the macDA, payload and macCrc components differ.

```
1:> set myl2spec [l2spec gen]
L2spec:0x00201c7c
1:> l2pkt generate $myl2spec pktsList
{L2packet: 0x00212050 0x002120c0}
1:> set mypkt [lindex $pktsList 0]
1:> l2pkt dump $mypkt
```

7.19 L4spec

This section will describe the l4spec commands used to generate an L4 states specification.

7.19.1 L4spec <neigh|path|tcp> new

Usage: **l4spec <neigh|path|tcp> new**

Description: This command creates a new instance of neighbour|path|tcp context with the default parameters, which may be changed with the "update"command later.

Example:
```
1:> set tcpSpec [l4spec tcp new]
```

7.19.2 L4spec <neigh|path|tcp> update

Usage: **l4spec <neigh|path|tcp> update**

Description: This command changes the values of the specified fields.

Example:
```
1:> l4spec update $tcpSpec [list destPort 50]
```

7.19.3 L4spec <neigh|path|tcp> show

Usage: **l4spec <neigh|path|tcp> show**

Description: This command prints the current values of the state specification.

Example:

```
1:> l4spec show $a

-----------------------------------------------------------------
NEIGHBOUR state parameters:
---------------------------
 1: Source MAC address                (const): 01:02:03:04:05:06
 2: VLAN tag                          (const): 1
 3: Destination MAC address         (cached): 0a:0b:0c:0d:0e:0f
```

Parameter index

Parameter class (constant, cached or delegated)

7.20 L4spath

This section will describe the "l4spath" commands used for L4 slow path operations.

7.20.1 L4spath ofld

Usage: **l4spath ofld <state specification>**

Description: This command offloads the provided state specification. On success, returns the context for the offloaded state.

Example:
```
set neighOfld [l4spath ofld $neighSpec]
set pathOfld [l4spath ofld $pathSpec $neighOfld]
set tcpOfld [l4spath ofld $tcpSpec $pathOfld]
```

7.20.2 L4spath poll

Usage: **l4spath poll <offloaded state context> [number of cqe's to poll]**

Description: This command polls for SP completion on the provided offloaded context (tcp, path or neighbor). Returns the list of triples {<state> <cqe type> <requested data>}. The number of elements in the list is less or equal to the requested number.

Note: If CQ is empty the list will be of the zero length.
In case of upload request completion, "request data" will be a TCP or PATH specification handle with updated delegated parameters (according to upload request type).

7.20.3 L4spath fin

Usage: **l4spath fin <tcp state context>**

Description: This command posts a FIN request.

7.20.4 L4spath rst

Usage: **l4spath rst <tcp state context>**

Description: This command posts an abortive disconnect request.

7.20.5 L4spath get_info

Usage: l4spath get_info [-l] <tcp context>

Description: This command returns (if –l key is given) or prints the events information for the given TCP state. The information is returned in the following list: {<Remote FIN received>, <Remote RESET received>}

7.20.6 L4spath upld

Usage: **l4spath upld <offloaded state context>**

Description: This command uploads the TCP, PATH or NEIGHBOUR state.

7.20.7 L4spath update

Usage: **l4spath update <offloaded context handle> <spec. handle>**

Description: Updates the provided offloaded context (tcp, path or neighbor) cached parameters to the given new ones (update slow path request). The request is completed on the SP ring, hence l4spath poll should be called.

7.21 L4fpath

This section will describe the "l4fpath" commands used for L4 fast path operations.

7.21.1 L4fpath send

Usage: **l4fpath send <offloaded TCP con. handle> <App. Buf. handle> [<number of repetitions>]**

Description: This command sends the given application buffer (arg2) to the given TCP connection (arg1), the given number of times (arg3 – optional, default is 1 time).

7.21.2 L4fpath poll_tx| poll_rx

Usage: **l4fpath poll_tx|poll_rx <offloaded TCP con. handle>**

Description: This command polls for a TX/RX completion on the given TCP connection. If there is no pending completion on the given TCP conn, an empty list is returned. If there is a pending buffer but is hasn't been fully completed yet, a pair {PENDING, <number of Network Buffers left>} is returned. If there is a completion, a pair {<Completion status>, <number of xfered bytes>} is returned.

7.21.3 L4fpath post_rx_buf

Usage: **l4fpath post_rx_buff <offloaded TCP con. handle> <App. Buf. handle> [<number of repetitions>]**

Description: This command posts received buffer (arg2) on the given TCP connection (arg1).

7.21.4 L4fpath has_gen

Usage: **l4fpath has_gen <offloaded TCP con. handle>**

Description: This command checks is there is a data from generic buffers pending on this connection. If so, returned value is 1, otherwise 0.

7.21.5 L4fpath get_gen

Usage: **l4fpath get_gen <offloaded TCP con. handle>**

Description: This command returns the following list: {<Ctx.>, {<addr1>,<size1>}, {addr2, size2},…} <ctx> is a context to be used with ret_gen command and the further list of (64_bit_address, size) pairs is the SGL for the received data.

7.21.6 L4fpath ret_gen

Usage: **l4fpath ret_gen <offloaded TCP con. handle> <Ctx>**

Description: This command returns the generic buffer to the driver.

7.21.7 L4fpath accept_all | reject_all

Usage: **l4fpath accept_all|reject_all <offloaded TCP con. handle> <Ctx>**

Description: This command configures the connection to work in generic data mode.

7.22 L4data

This section will describe the "l4data" commands used for L4 application buffer operations.

7.22.1 L4data new

Usage: **l4data new <Net. buf. list len.> <SGL len. of each NB> <SGLE size> [<no_push flag - 0 or 1>]**

Description: This command creates one application buffer with the given number of network buffers (arg 0), each of which has the same number of SG elements (arg1) of the same size (arg2). By default, NO_PUSH flag is cleared. On success, returns the application buffer handle.

7.22.2 L4data delete

Usage: **l4data delete <Application Buffer handle>**

Description: This command deletes the given application buffer and frees its memory.

7.22.3 L4data list

Usage: **l4data list <Application Buffer handle>**

Description: This command returns the following list {{addr11, size11}, {addr12, size12},...}, {{addr21, size21},{addr22, size22},...}, which contains the SGLs of each TCP buffer of the given application buffer.

7.23 License

The "license" command is used for installing and verifying Everest licensing support.

7.23.1 License secret

Usage: license secret <*filename*>

Description: This command reads the binary file specified by *filename* for the shared secret content and programs the value into NVRAM.

Example:

```
1:> license secret ss_evrst.bin
```

7.23.2 License storekey

Usage: license storekey –m|–u <*filename*>

Description: This command reads the binary file specified by *filename* for a license key and programs the value into NVRAM. The command switch "**-m**" indicates the key is stored as a manufacturing key while the switch "**-u**" indicates the key is stored as an upgrade key. The command also verifies that the key was generated by the shared secret previously programmed by the "**license secret**" command. If the key validation fails, the key is not programmed.

Example:

```
1:> license storekey –m mkey_m.bin
```

7.23.3 License rmkey

Usage: license rmkey –m|-u

Description: This command removes either the manufacturing key (**–m**) or the upgrade key (**-u**). If the key does not exist, no error will be indicated.

Example:

```
1:> license rmkey –m
```

7.23.4 License display

Usage: license display –m|–u

Description: This command displays the license content of either the manufacturing key (**-m)** or the upgrade key (**-u**).

If the key does not exist an error will be displayed:

```
Key is either corrupted or does not exist.
```

Example:

```
1:> license display -m
 1:> license display -m
 0. Max TOE conn (65535 for unlimited)                : 65535
 1. Max RDMA conn (65535 for unlimited)               : 65535
 2. Max iSCSI initiator conn (65535 for unlimited)    : 65535
 3. Max iSCSI target conn (65535 for unlimited)       : 65535
 4. Serial number                                     : 5246784
```

7.23.5 License chkkey

Usage: license chkkey <–s|–m|–u>

Description: This command checks the integrity of the license keys as well as the shared secret inside the NVRAM. If an optional switch is provided, only the shared secret (**-s**), the manufacturing key (**-m**), or the upgrade key (**-u**) will be checked. It may seem counter-intuitive, but it is possible that the check fails on shared secret while the check

passes on the license keys. The likely reason for this is that there is a mismatch between the shared secret (hence the licenses as well) used and the PCI SVID.

Returns: OK if the secret/key is valid, otherwise nothing.

Example:

```
1:> license chkkey
Shared secret       :OK
Manuf key           :OK
Upgrade key         :Invalid/Nonexistent
```

7.24 Log

The log command enables the user to save the execution results in a log file. There are two available log commands: log open | close.

7.24.1 Log open

Usage: **log open** <*logfile*>

Description: This command opens a log file and start logging all display to the log file.

Examples:

```
1:> log open results.log
```

7.24.2 Log close

Usage: **log close**

Description: This command closes and saves the log file.

Examples: 1:> log close

7.25 Led

This section will describe the "led" command which is used to control the five led in the *Everest* card.

7.25.1 Led mode

Usage: **led mode <set | get> <port> [<mode>]**

Description: This command sets or gets the led mode of the led of the requested port 0 or 1. Incase of "get" option, the command returns the led mode of the requested port.

Incase of "set" option, the command sets the led mode of the requested port according to the <mode> input argument:
"MAC1", "PHY1", "PHY2", "PHY3", "MAC2", "PHY4", "PHY5", "PHY6", "MAC3", "PHY7", "PHY8", "PHY9", "MAC4", "PHY10", "PHY11"

7.25.2 Led override

Usage: **led override <port> <led name> <value>**

Description: This command overrides the requested led value <0 or 1> by its name and port <0 or 1>. The led names are: 10M, 100M, 1000M, 2500M, 10G or TRAFFIC.

7.25.3 Led blink

Usage: **led blink <port> [optional: <blink rate>]**

Description: This command blinks the traffic led of port 0 or 1, at the requested blink rate (in msec).

7.25.4 Led status

Usage: **led status <port> <led name>**

Description: This command returns the status of the requested led by its name and port number 0 or 1. The led names are: 10M, 100M, 1000M, 2500M, 10G or TRAFFIC.

7.26 Lmdev

This section describes the "lmdev" commands used to get information from the Everest controller lm_device.

7.26.1 lmdev prod_index

Usage: **lmdev prod_index <chain type> <RSS index>**

Description: This command shows the lm_device producer index of the requested chain type (tx, rx, rcq, or sq) and the requested RSS index (0-15).

7.26.2 lmdev cons_index

Usage: **lmdev cons_index <chain type> <RSS index>**

Description: This command shows the lm_device consumer index of the requested chain type (tx, rx, rcq, or sq) and the requested RSS index (0-15).

7.26.3 lmdev ring_info

Usage: **lmdev ring_info <chain type> <RSS index>**

Description: This command shows the lm_device information, including producer index, consumer index, and number of left BDs of the requested chain type (tx, rx, rcq, or sq) and the requested RSS index (0-15).

Note: In the event of sq, then the RSS index is not relevant.

7.26.4 lmdev show

Usage: **lmdev ring_info <RSS index>**

Description: This command shows the lm_device information, including producer index, consumer index, and number of left BDs of all rings of the requested RSS index (0-15).

Note: In the event of sq, then the RSS index is not relevant.

7.26.5 lmdev vlan_remove

Usage: **lmdev vlan_remove <mode>**

Description: This command sets the lm_device "enable VLAN" flag to 1

7.26.6 lmdev set_medium

Usage: **lmdev set_medium <type>**

Description: This command sets the lm_device req_medium to either PHY_LB or BMAC_LB to enable PHY internal loopback or BMAC internal loopback.

7.26.7 lmdev set_mtu

Usage: **lmdev set_mtu <576-9216>**

Description: This command sets MTU values between 576 – 9216.

7.26.8 lmdev get_mtu

Usage: **lmdev get_mtu**

Description: This command returns the current MTU value.

7.26.9 lmdev set_rx_mode

Usage: **lmdev set_rx_mdoe <rx_mode>**

Description: This command sets the RX mode to NONE, UNICAST, MULTICAST, ALL_MULTICAST, BROADCAST, ERROR_PACKET, PROMISC.

7.26.10 lmdev set_speed

Usage: **lmdev set_speed <10M, 100M, 1000M, 2500M, 10G, 20G or AUTONEG>**

Description: This command sets the speed as requested.

7.26.11 lmdev set_mac_addrs

Usage: **lmdev set_mac_addrs [list <mac_addr 1> <mac_addr 2> <mac_addr 3> ..]**

Description: This command sets the LM's MAC addresses.

7.26.12 lmdev get_link_cnt

Usage: **lmdev get_link_cnt**

Description: Gets lmdev link changes counter result, which indicates if there was a link change.

7.26.13 lmdev enable_e2_integ

Usage: **lmdev enable_e2_integ**

Description: Sets lm_device param e2_integ_testing_enabled to 1.

7.26.14 lmdev disable_e2_integ

Usage: **lmdev disable_e2_integ**

Description: Clears lm_device param e2_integ_testing_enabled to 0.

7.27 Mcp

The "mcp" command implements a debugger for the on-chip MIPS processor.

Usage: mcp run
mcp halt
mcp gpr [reg value]
mcp ctrl [reg value]
mcp step [number of steps] [none|both|ctrl|gpr]
mcp load
mcp mem [addr [value1 [value2 ...]]]
mcp disasm [addr [count]]
mcp free_disasm [addr [count]]
mcp trace
mcp access addr [value]
mcp fill [len [value]]
mcp reset

7.28 mem_info

The "mem_info" implements memory allocation tracking API.

Usage: **mem_info < virt | phys>**

Description: Returns the currently allocated virtual / physical memory (in Bytes).

7.29 mtest

The "mtest" command implements memory test using "Marching Ones/Marching Zeroes" algorithms. This command is a C-implementation of time consuming parts of the memory test.

7.29.1 mtest init

Usage: mtest init <first offset> <last offset> <number of DWORDS in one register>

Description: Zeroes the group of registers defined by the parameters. Offsets should be 4-bytes aligned.

7.29.2 mtest [march_1s | march_0s]

Usage: mtest march_1s <first offset> <last offset> <number of bits in each register> <first offset> <last offset> <number of DWORDS in one register> <number of DWORDS in each register> <access type: R,RC,RW,ST,W,WB,WB_R,WB_W,WR>

Description: Performs "Marching Ones" or "Marching zeros" test on the given registers block. Offsets should be 4-bytes aligned.

7.30 nictest

Usage: **nictest [-t <groups>] [-T <groups>] [-I <num>] [-cof] [-debug] [-hlb] [-none]**

-t Exclude specified tests
-T Include specified tests
-I Specify the number of iterations to run
-cof Enable to continue after test failure
-debug Runs the test with debug messages
-hlb Sets host loopback on another device.
-none Disables all tests

Description: This command performs the functional testing of the Everest chip. The various test groups are described in section 3.

Example:

```
1:> nictest –t ab
Group C. Blocks and Network Tests on port 0
   C01. DMAE functionality test..................: passed
   C02. CAM R/W functionality test...............: passed
   C03. CAM Search test..........................: passed
   C04. Timers test, 512 connections.............: passed
   C05. GRC test access to un-mapped addresses....: passed
   C06. Debug bus check with output to PXP........: passed
   C08. XGXS Register access test................: passed
   C10. XGXS PRBS Internal loopback test..........: passed
Group D. Traffic Tests on port 0
   D01. BMAC Loopback Test.......................: passed
   D02. PHY Loopback Test........................: passed
   D04. LSO Test.................................: passed
   D05. Statistics Test..........................: passed
   D06. RPC Test.................................: passed
   D07. TOE Test.................................: passed

1:> nictest –none –T C1
Group C. Blocks and Network Tests on port 0
   C01. DMAE functionality test..................: passed
```

7.31 nvm

The "nvm" command provides access to the Everest NVRAM.

7.31.1 Nvm cfg

Usage:

nvm cfg

nvm cfg [<*option=choice*>]

nvm cfg [*board type*] [-dump <filename>]

nvm cfg <*option*> -

Description: This command implements both an interactive and a scripted mode of operation that allows a user to view/modify the Everest controller configuration. When invoked without any options, the user is presented with a listing of the current NVRAM configuration, and is given the opportunity to change the configuration and save the results. When invoked with options, only that NVRAM configuration setting is modified.

Each option is described with a [P] for per-port option or [S] for shared-between-ports option prefix.

In the case of **"nvm cfg <board type>"**, the default choices for this board type are set.

[-dump <filename>] dumps the NVM configuration into the file.

Notes:

1. Running "nvm cfg" without any option will show the NVRAM configurations groups.

```
1:> nvm cfg
                            NVRAM Configuration Groups:
                            ---------------------------
1 . board config
2 . board i/o
3 . dual phy
4 . features
5 . link settings
6 . management fw
7 . npar
8 . pcie
9 . phy
10 .   pre-boot
11 .   vf
```

2. To identify the configuration group of a specific option, type "nvm cfg <option>-". This command will return the configuration group of this option.
3. Use of the "nvm cfg <board type>" command replaced by XXXnvm.txt (where XXX is the board type) configuration files.
4. To insert an option that includes a string with spaces, use curly braces {} (e.g., nvm cfg {11=PCIe 2-P 10Gbe-SFP+ adapter})
5. Upgrading the MAC address field in the NVRAM via "nvm cfg" does not required system reboot.

6. In case of an "nvm cfg" options which does require system reboot a warning message "Please reboot the system for change to take effect" will appear.

Examples:

```
1:> nvm cfg A1022G
1:> nvm cfg T1001
1:> nvm cfg T1001 -dump test.txt
1:> nvm cfg 1=00:10:18:00:11:22
1:> nvm cfg {11=PCIe 2-P 10Gbe-SFP+ adapter}
1:> nvm cfg 114-
                      Group: npar (Group 7)
------------------------------------------------------------------------
114: [S] MAC partition global cfg
        {Disabled(0), Enabled(1)}                      : Disabled
```

7.31.2 Nvm chk

Usage: nvm chk <board type> [**-fix**]

Description: Compare the NVM configuration to the default configuration setting expected for this board type.

Example:

```
1:> nvm chk T1001
The following errors were found in the configuration:
ID 40 for port 0: "Serdes external PHY address (8 bits)" mismatch:
   Expected 10
   Existing 0

1:> nvm chk T1001 -fix
```

7.31.3 Nvm crc

Usage: nvm crc [**-fix** <*entry*>]

Description: Checks the integrity of the NVRAM CRC, optionally updating the CRC to match the data.

Example:

```
1:> nvm crc
    Region       Start      Length     Content    Computed  Result
-------------------------------------------------------------------------------
 BOOTSTRAP  0x00000000  0x00000014  0xD2CC5FC1  0xD2CC5FC1  OK
 DIRECTORY  0x00000014  0x000000EC  0x9846510B  0x9846510B  OK
   HW_INFO  0x00000100  0x00000350  0x7702C28E  0x7702C28E  OK
 FEAT_INFO  0x00000450  0x000000F0  0x8C28D421  0x8C28D421  OK
 MKEY_INFO  0x00000720  0x0000007C  0x28FF4014  0xE33E6C5E  Mismatch
       BC1  0x00000818  0x000029B4  0x8A8B620B  0x8A8B620B  OK
       BC2  0x000031CC  0x000066D4  0x16FA419A  0x16FA419A  OK
```

7.31.4 Nvm dir

Usage: nvm dir [**-delete** <*entry*>]

Description: Displays a listing of the firmware programmed in NVRAM, optionally deleting a particular image.

Shows the image offset in NVM, size (in byte count), and version.

Example:

```
1:> nvm dir
     Image   SRAM Addr  NVM Offset    Byte Cnt     CPU             Version
-------------------------------------------------------------------------------
       MBA  0x00000000  0x0003DBF8  0x00040604    HOSTPCI30_CLP MBA 6.4.16
                                                          EFI x64 6.4.8
       L2T  0x001C0000  0x0001E204  0x000021A4  TSTORM           L2T 6.0.0
       L2C  0x00240000  0x000203A8  0x00000964  CSTORM           L2C 6.0.0
       L2X  0x002C0000  0x00020D0C  0x00002444  XSTORM           L2X 6.0.0
       L2U  0x00340000  0x00023150  0x00001974  USTORM           L2U 6.0.0
MODULES_PN  0x00000000  0x0003ADA4  0x0000005C    HOST             Rev 1
   ISCSI_B  0x08000010  0x00024AC4  0x0000EFF4    HOST    ipv4_ipv6 v6.4.1
   I_B_CFG  0x08000010  0x00033AB8  0x00000800    HOST              v2.0
  I_B_CFG2  0x08000010  0x000342B8  0x00000800    HOST              v2.0
  I_B_CPRG  0x08000010  0x00034AB8  0x000062EC    HOST            v6.4.0
       NIV  0x08000010  0x0003AF8C  0x00000468    HOST              Rev1
NIVPROFILES 0x08000010  0x0003B3F4  0x00002804    HOST
      MFW1  0x08000040  0x00000C28  0x000062DC     MCP         MFW1 6.4.13
      MFW2  0x08009400  0x00006F04  0x00017300     MCP         MFW2 6.4.13
     SWIM1  0x08000040  0x0007E1FC  0x00009034     MCP        SWIM1 6.4.13
     SWIM2  0x08000040  0x00087230  0x000080BC     MCP        SWIM2 6.4.13
     SWIM3  0x08000040  0x0008F2EC  0x00003D8C     MCP        SWIM3 6.4.13
     SWIM4  0x08000040  0x00093078  0x00001B30     MCP        SWIM4 6.4.13


1:> nvm dir –delete MBA
```

7.31.5 Nvm dump

Usage: **nvm dump** [<offset> <length>] <filename>

With possibility to add flags: -a –s –l -b

-l - little endian format
-b - big endian format (default)
-s - ST type flash (default)
-a - ATMEL type flash
-m_pn Qualified optic modules information

Flags can be in any order, if contradicting flags are given, the latter is considered.

Description: This command reads NVRAM starting at the offset for a length of *length* bytes (which must be a multiple of 4) and writes the contents to the file *filename*. If the *offset* or *length* arguments are omitted, the offset will default to 0 and the length will default to the entire NVRAM. If **–l** option is specified, the output will be in little endian format. If –b is specified – then the format will be big endian. If neither '-l' nor '-b' flags are given, then the default is big endian. Optionally, either –**a** or –**s** option can be provided to specify whether the resulting file is for a particular type of NVRAM (–**a** for Atmel and –**s** for ST), and the default (without specifying either option) is the native format, determined by the NVRAM used by the current device.

Note:

The '-m_pn' option was added to dump the context of the MODULES_PN image which includes the qualified optic modules information into output text file.

```
1:> nvm dump –m_pn test.txt
```

Examples:

```
1:> nvm dump file.bin
1:> nvm dump file.bin -a            ATMEL flash    big endian format
1:> nvm dump file.bin -s            ST flash       big endian format
```

```
1:> nvm dump file.bin -l            ST flash       little endian format
1:> nvm dump file.bin -s -l         ST flash       little endian format
```

7.31.6 Nvm fill

Usage: nvm fill <*addr*> <*bytecount*> <*value32* | **inc** | **addr**>

Description: This command fills NVRAM, starting at the offset *addr* for a length of *bytecount* bytes. If the argument *value32* is specified, it fills each NVRAM DWORD with that 32-bit value. If the argument **inc** is specified, it fills each NVRAM DWORD with an incrementing value starting from 0. If the argument **addr** is specified, it fills each NVRAM DWORD with its offset value.

Note: This command will not overwrite the L2 MAC address or iSCSI MAC addresses, and it will recalculate the CRC for the NVRAM manufacturing block, if it is modified as a result of this command. The "nvm write" command should be used to modify these NVRAM offsets instead.

Example:

```
1:> nvm fill 0 0x100 0xdeadbeef
Programming from: 0x00000000 to 0x00000100
Done
```

7.31.7 Nvm prg

Usage: nvm prg [options] <offset> [<filename>|<hostaddr>] [size (optional)]

<options> below are the available options:

-mem	: input from host memory
–l	: little endian file format
-force_mac	: force to update MAC addresses according to file
-force_lic	: force to update license blocks
-force_ibcfg	: force to overwrite iSCSI boot parameters (I_B_CFG, I_B_CFG2) in NVM, with parameters from the file image
-force_febcfg	: force to overwrite FCoE boot parameters (FEB_CFG, FEB_CFG2) in NVM with parameters from the ile image.
-force_lic	: force to update license blocks
-skip_lic	: skip updating license blocks
-skip_sn	: preserve serial number in VPD
-skip_ec	: preserve engineer change in VPD
-no_idmatch	: skip device ID check
-idmatch	: match device VID, DID
-raw	: program the complete raw image
-skip_options	:preserve user defined options

<offset> NVM offset address from which to start programming
[<filename>|<hostaddr>] : binary file or host memory address
<size> : Optional - the number of bytes read from host memory

Description: This command programs NVRAM starting at the offset *offset*. If –mem is specified, the data is expected to be read from host memory at the address *hostaddr* for a length of *size* bytes. Otherwise, the data is read from the file specified by *filename*.

If ***–l*** is specified, the input is read in little endian file format. If ***–idmatch*** is specified, then the vendor ID and the device ID of the loaded image will be validated. If ***–no_idmatch*** is specified, then this check is not performed.

- By default, when programming the NVRAM image from offset 0, the MAC addresses of the boards are preserved. The option ***–force_mac*** allows the user to override this default.

- The option of ***–skip_lic*** allows users to skip the license key blocks when programming the NVRAM image from offset 0. The option of ***–force_lic*** allows users to override the license key blocks when programming the NVRAM image from offset 0.

- The options of ***–skip_sn and –skip_ec*** allows users to preserve the serial number, engineer change in VPD (options 13, 14 in nvm cfg).

- By default, when programming the NVRAM image from offset 0, the iSCSI and FCoE boot parameters (I_B_CFG, I_B_CFG2, FEB_CFG, FEB_CFG2) are preserved and not over-written by the parameters in the file image to be programmed. This is in case the file image does include iSCSI and FCoE boot parameters.

- The options ***-force_ibcfg, -force_febcfg*** allow the user to force overwrite of iSCSI and FCoE boot parameters in NVM with the parameters in the file image used for programming.

- The option ***–raw*** programs the complete raw image. In this case, the options ***-force_mac, -force_ibcfg, -force_febcfg, -force_lic*** are turned on. The options ***-skip_ec, -skip_sn, -skip_lic, -skip_config*** are turned off.

- The option ***–skip_options*** preserved the user defined options. The NVM configuration options defined by the user are saved with their original value and not overwritten by the image programing.

After programming is completed, the controller needs to be reset to reflect the new NVRAM contents.

Examples:

```
1:> nvm prg 0 t1000cr.img
1:> nvm prg -skip_config 0 t1000cr.img
1:> nvm prg -skip_config –force_mac 0 t1000cr.img
1:> nvm prg –force_lic 0 t1000cr.img
1:> nvm prg -skip_lic 0 t1000cr.img
1:> nvm prg -skip_ec –skip_sn 0 t1000cr.img
1:> nvm prg –force_ibcfg –force_febcfg 0 t1000.img
1:> nvm prg -raw 0 t1000cr.img
1:> set options 1,9,55 ; nvm prg 0 –skip_options $options image.bin
```

7.31.8 Nvm read

Usage: nvm read <*addr*>

Description: This command reads a DWORD from NVRAM at the offset *addr*.

Returns: The value read.

Examples:

```
1:> nvm read 0
0x669955aa
```

7.31.9 Nvm show

Usage: nvm show <*begin_addr*> *[nbytes]*

Description: This command performs the same operation as the "**nvm read**" command except that it formats the output in a more user-friendly manner. The optional argument *nbytes* (which must be a multiple of 4) specifies the number of bytes to display.

Example:

```
1:> nvm show 0 0x40
0000000:669955aa 08006c00 0000030c 00000ec8 3a1090f5 08000000 07001f98 00002000
0000020:00000000 11010840 0000c400 00000000 00000000 00000000 00000000 00000000
```

7.31.10 Nvm write

Usage: nvm write <*addr*> <*value1 [value2…]*>

Description: This command writes NVRAM beginning at the offset addr with the 32-bit value "*value1 value 2...*".

Example:

```
1:> nvm write 0 0x01234567
```

7.31.11 Nvm upgrade

Usage: nvm upgrade [-F] –bc|-bc1|-bc2|-mfw |-mba|-ncsi |-l2t|-l2c|-l2x|-l2u|-ipmi| -m_pn|-ib|-ibpe|-ib_ipv6|-ib_ipv4n6 <*filename*>

Description: This command upgrades the firmware or bootcode for the Everest controller. The *filename* specifies the name of the file that contains the appropriate image. If the upgrade version is the same or older than the version in NVRAM, the upgrade will be aborted. The **–F** option is to force the upgrade without checking the version.

Starting from version 7.12.2, nvm upgrade checks that CRC and CRC trailer exist at the end of firmware file. Nvm upgrade will abort if CRC computed is different than CRC at the end of file. Nvm upgrade will abort if CRC or CRC trailer is missing.

The –F option allows upgrading firmware images without CRC.

OPTION	NAME OF FIRMWARE OR BOOT IMAGE
bc [1|2]	bootcode (combined or 1 or 2)
mfw [1|2]	Management Firmware (replaces bootcode)
swi[1-8]	Swapable images (as part of MFW)
l2 [c|t|x|u]	L2 firmware
ncsi	ncsi driver
mba	MBA (PXE) code
ipmi	IPMI code
ump	UMP firmware
ib	iSCSI boot driver
ibpe	iSCSI utility program
ib_ipv6	iSCSI ipv6 boot driver
ib_ipv4n6	iSCSI ipv4/ ipv6 boot driver
feb	FCOE boot
feb -c	FCOE boot and config block (FEB_CFG)
feb –p	FCOE boot and config program (FEB_CPRG)
feb –cp	FCOE boot and config block and program (FEB_CFG, FEB_CPRG)
m_pn	Qualified optic modules information
vpd	Virtual Product Data
nicp	NIC PRTION image
niv	NIV image
niv_profiles	NIV_PROFILE image

Option '-mfw' upgrades the combined management firmware (MFW + SWIM) images. On the first time MFW is upgraded, the BC image will be deleted and replaced by MFW image.

```
1:> nvm upgrade -mfw /nfs/usr/mcp/mf712v64.13
Deleting old BC image ..
Upgrading MFW1 image: to version MFW1 6.4.13
Upgrading MFW2 image: to version MFW2 6.4.13
Upgrading SWIM1 image: to version SWIM1 6.4.13
Upgrading SWIM2 image: to version SWIM2 6.4.13
Upgrading SWIM3 image: to version SWIM3 6.4.13
Upgrading SWIM4 image: to version SWIM4 6.4.13
info burned into nvm successfully.
```

SWIM Shadow feature

When performing an MFW upgrade (using option '-mfw'), eDiag checks if there is enough space in NVRAM for two instances of SWIM images: old and new. If there is enough space, eDiag will not delete the old SWIMs and will leave them in NVRAM.

The motivation for this feature is usage by other upgrade tools running while the operational driver is up. In this case, the old MFW (compatbile with old SWIMs) is still running, so to avoid the need to immediately reboot, the old MFW can continue using the old SWIMs until reboot.

Therefore, when upgrading MFW, SWIMs will be written to to the empty group (the one not including the current-running SWIMs) - group A (images SWIM1-SWIM5) or group B (SWIM1B-SWIM5B).

Other Notes regarding nvm upgrade:

- The latest iSCSI boot file actually contains multiple images, including the iSCSI boot driver itself, iSCSI configuration, and iSCSI configuration utility program.

- Using '-ib' option will burn the iSCSI driver together with the configuration data for the board devices (I_B_CFG and I_B_CFG2 incase of dual port board, only I_B_CFG in case of single port board).

- Option '-ibpe' is the only way to burn the utility program.

- Option 'm_pn' was added to upgrade the qualified optic modules information image.

  The input file should be a text file with the following format:

```
# Each none empty line that does not start with "#" should
# have all 3 module's parameters separated by coma ",":
# module name (up to 16 bytes)
# OUI (3 hex single bytes numbers is xx-xx-xx format)
# part number (up to 16 bytes
#
# it is possible to replace a parameter with "*" to indicate any
# value is
# acceptable
#
JDSU , 00-01-9c , PLRXPLSCS4321N
```

After the upgrade completes, the controller needs to be reset in order to load and run the new firmware.

Examples:

```
1:> nvm upgrade -bc1 bc1.bin
1:> nvm upgrade -mba evmmba.nic
1:> nvm upgrade -ib iscsi.bin
1:> nvm upgrade -ibpe iscsi.bin
1:> nvm upgrade -ib_ipv6 iscsi.bin
1:> nvm upgrade -ib_ipv4n6 iscsi.bin
1:> nvm upgrade -m_pn modules.txt
1:> nvm upgrade -feb fcoe_b.bin
1:> nvm upgrade -feb -c fcoe_b.bin
1:> nvm upgrade -feb -p fcoe_b.bin
1:> nvm upgrade -feb -cp fcoe_b.bin
1:> nvm upgrade -mfw mf712v64.13
```

Note:

NVM upgrade can fail if there is not enough space in NVRAM from the new image. In this case, an error message "not enough room in NVM for this image type" will be shown on screen.

To check if there is enough space in NVM for a new image, you can run "nvm dir", sum up all images byte_cnt including the new image size and check if it does not exceed the $::currrent(NVM_SIZE).

7.32 Pci

The Pci commands enable to search for PCI devices and get their information.

7.32.1 Pci init

Usage: **pci init**

Description: This command restores PCI-related information back to the device. This is useful in NIC manufacturing to use PCI/PCIE extender to shut off the device instead of shutting off the entire system.

7.32.2 Pci scan

Usage: **pci scan**

Description: This command scans all PCI devices of the system.

Example:

```
1:> pci scan
X-DIAG format:
Domain Bus Dev Func  Vendor ID Device ID    Class     Base/IO Address    IRQ
====== === === ====  ========= =========  ==========  =================  ===
   0   0   0   0       8086      2774     06:00:00   00000000:00000000  0
   0   0   1   0       8086      2775     06:04:00   00000000:00000000  11
   0   0  27   0       8086      27D8     04:03:00   00000000:FEBFC000  11
   0   0  28   0       8086      27D0     06:04:00   00000000:00000000  11
   0   0  28   4       8086      27E0     06:04:00   00000000:00000000  11
   0   0  28   5       8086      27E2     06:04:00   00000000:00000000  10
   0   0  29   0       8086      27C8     0C:03:00   00000000:00000000  9
   0   0  29   1       8086      27C9     0C:03:00   00000000:00000000  5
   0   0  29   2       8086      27CA     0C:03:00   00000000:00000000  3
   0   0  29   3       8086      27CB     0C:03:00   00000000:00000000  10
   0   0  29   7       8086      27CC     0C:03:20   00000000:FFA80800  9
   0   0  30   0       8086      244E     06:04:01   00000000:00000000  0
   0   0  31   0       8086      27B8     06:01:00   00000000:00000000  0
   0   0  31   1       8086      27DF     01:01:8A   00000000:00000000  11
   0   0  31   2       8086      27C1     01:06:01   00000000:0000FE00  5
   0   0  31   3       8086      27DA     0C:05:00   00000000:00000000  10
   0   1   0   0       10DE      0165     03:00:00   00000000:FC000000  11
   0   2   0   0       14E4      1662     02:00:00   00000000:F7800000  11
   0   2   0   1       14E4      1662     02:00:00   00000000:F6800000  10
   0   4   0   0       14E4      1677     02:00:00   00000000:FBEF0000  10
```

7.32.3 Pci search

Usage: **pci search [**-did <***deviceID***>**] [**-vid <***vendorID***>**] [**-class <***class***>**]**

Description: This command searches the entire PCI/PCIE bus for all devices with device ID, *deviceID*, or with a vendor ID, *vendorID*, or with a class code of *class*. If it finds any device, the corresponding bus number, device number, and function number will be displayed.

Example:

```
1:> pci search -did 0x164c
{6 0 0} {9 0 0}
```

7.32.4 Pci setdut

Usage: **pci setdut <domain> <bus> <device> <function>**

Description: This command sets the address of the device to be used with pcicfg and reg iread/iwrite commands. After this command, the current device will be unset (no device will be set as current). The following conditions must be met so that the operation succeeds: bus < 256, device < 32, function < 8.

When the command "pci setdut" is used with no arguments, it displays the current <domain> <bus> <device> <function>

Examples:

```
1:> pci setdut
0 2 0 0
1:> pci setdut 0 2 0 3
0 2 0 3
```

7.33 Pcicfg

The pcicfg command provides a mechanism to read, modify, display, and write PCI configuration space registers.

7.33.1 pcicfg read

Usage: **pcicfg read** <*addr*>

Description: This command reads data from PCI configuration space using PCI config cycle. A 32-bit value that corresponds to *addr* is returned.

Example:

```
1:> pcicfg read 0
0x164a14e4
```

7.33.2 pcicfg show

Usage: **pcicfg show** <*begin_addr*> < *nbytes*>

Description: This command performs "**pcicfg read**" with the same argument and formats the output to be user-friendly. An additional argument, *nbytes*, (must be a multiple of 4) can be provided to indicate how much data to display

Example:

```
1:> pcicfg show 0 0x20
0000000: 164a14e4 02b0011e 02000001 00002008 da000004 00000000 00000000 00000000
```

7.33.3 pcicfg write

Usage: **pcicfg write** <*addr*> < *value*>

Description: This command treats *value* as a 32-bit value and writes it into the configuration register at *addr*. It returns 1 to indicate the success of the command.

Example:

```
1:> pcicfg write 0x10 0xffffffff
```

7.34 Reg

The reg command enables access to registers for read / write. Access can be done either in a direct way by writing to the register's address the data or indirectly using PCICFG_REG_WINDOW_ADDRESS and PCICFG_REG_WINDOW_DATA to write to the register address this data.

7.34.1 Reg read / dread / iread

Usage: **reg (read | iread | dread)** <*addr*>

Description: This command reads the register's data. The **read** and **dread** perform a direct read register (i.e., use memory cycle to access).The **iread** carries out the operation through indirect register access method (i.e., use PCI config cycles via PCICFG_REG_WINDOW_ADDRESS and PCICFG_REG_WINDOW_DATA).

Examples:

```
1:> reg read 0x400
0x0
```

7.34.2 Reg show / ishow

Usage: **reg (show | ishow)** <*begin_addr*> < *nbytes*>

Description: This command performs "reg read" from the <*begin_addr*>. An additional argument <*nbytes*> (must be a multiple of 4) can be provided to indicate how much data to display. The **reg show** performs a direct read from the registers and **reg ishow** carries out the operation through the indirect register access (i.e., use PCI config cycles via PCICFG_REG_WINDOW_ADDRESS and PCICFG_REG_WINDOW_DATA).

Example:

```
1:> reg show 0x408
0000400:........ ........ 02000a9a
1:> reg show 0x400 0x20
0000400:00000000 00000000 02000a9a c40000ff 0a000064 0a000064 01020304 11121314
```

7.34.3 Reg write / dwrite / iwrite

Usage: **reg (write | iwrite)** <*addr*> < *value*>

Description: This command treats *value* as a 32-bit value and writes it into the register at *addr*. This command returns 1 for success. The **write** and **dwrite** perform the operation through direct register access method (i.e., use memory cycle to access). The **iwrite** carries out the operation through the indirect register access (i.e., use PCI config cycles via PCICFG_REG_WINDOW_ADDRESS and PCICFG_REG_WINDOW_DATA).

Example:

```
1:> reg write 0x808 0x01234567
```

7.35 serialport

The serialport command allows eDiag to control the system's serial port and redirect console input and output through the serial port.

7.35.1 open

Synopsis: serialport open <*comport*> < *baudrate*> < *settings*>

Description: This command attempts to open a communication port for I/O to be redirected to a dumb terminal. The terminal setting is configurable. The baud rate can be optionally specified as 1200, 4800, 9600, 19200, 38400, 57600, or 115200 (default). The setting can also be optionally specified with either 7 or 8 data bits, any parity setting (odd, even, or none), and 1 or 2 stop bits. The default setting is 8N1.

Example:

```
1:> serialport open 2
opened port 2 settings: 115200, 8N1
1:>
```

Synopsis: serialport redirect

Description: This command moves the console input and output to the serial port that is open by "**serialport open"** command. As soon as the command is executed, all input and output are redirected to that port selected in "**serialport open**" command. It is also necessary to designate which port I/O and statistic information goes by setting the $::sys(IO_PORT) and $::sys(STAT_PORT) to the appropriate value.

Returns: Nothing

Example:

```
1:> serialport redirect
```

7.35.2 close

Synopsis: serialport close <*comport*>

Description: This command closes the communication port specified by *comPort* and restores all console input and output to the standard display. This command is usually issued at the remote terminal.

Returns: Nothing

Example:

```
1:> serialport close
1:>
```

7.36 Sb

The "sb" commands enable to view the status blocks. The status blocks are used by the hardware to inform the driver of events such as interrupts. The status blocks are mapped on the host memory. There are two types of status blocks: *Default status block* and *RSS status block*. There are 16 RSS status blocks (with indexes 0 -15).

Each status block includes:
Consumer IDs – The hardware can alert the host driver of events.
Status index- Used by the driver to see if there was a change in the status block.
Block ID – The status block ID.

7.36.1 sb cons_id

Usage: **sb cons_id <RSS index (0-15)> <storm prefix (u or c)>**

Description: This command returns the consumer IDs of the requested storm section (ustorm or cstorm) of the requested RSS status block storm (index 0-15).

Examples:

```
1:> sb cons_id 0 c
0x0 0x0 0x20 0x0
```

7.36.2 sb status_index

Usage: **sb status_index <RSS index (0-15)> <storm prefix (u or c)>**

Description: This command returns the status index of the requested storm section (ustorm or cstorm) of the requested RSS status block storm (index 0-15).

Examples:

```
1:> sb status_index 0 u
0x0
```

7.36.3 sb block_id

Usage: **sb block_id <RSS index (0-15)> <storm prefix (u or c)>**

Description: This command returns the block ID of the requested storm section (ustorm or cstorm) of the requested RSS status block storm (index 0-15).

Examples:

```
1:> sb block_id 0 c
0x20
```

7.36.4 sb show

Usage: **sb show <RSS index (0-15)>**

Description: This command shows all the available information on the requested RSS status block storm (index 0-15).

Examples:

```
1:> sb show 0
Status block 0 show result:
u-storm consumer ids: 0 0 0 0
u-storm status index 0
u-storm block id 0
c-storm consumer ids: 0 0 0 0
c-storm status index 0
c-storm block id 20
```

7.36.5 sb default

Usage: **sb default**

Description: This command shows all the available information on the default status block storm.

Examples:

```
1:> sb default
u-storm consumer ids: 0 0 0 0
u-storm status index 0
u-storm block id 10
c-storm consumer ids: 0 0 0 0
c-storm status index 0
c-storm block id 30
t-storm consumer ids: 0 0 0 0
t-storm status index 0
t-storm block id 70
x-storm consumer ids: 0 0 0 0
x-storm status index 0
x-storm block id 50
```

7.37 Spio

The "spio" commands enable the user to read/write to SPIO pins.

7.37.1 Spio read

Usage: **spio read** *[optional: < pin>]*

Description: This command reads from GPIO pin 0,1,2 ..7. If no <pin> input all pins are read.

Examples:

```
1:> spio read 1
1:> 0
1:> spio read
1:> 0: 0 1 0 1 1 1 1 1
```

7.37.2 Spio write

Usage: **spio write** *< pin> < value>*

spio write *< pin_n> {< value_n> <value_n+1> <value_n+2>}*

Description: This command writes to SPIO pin 0,1,2 ..7 the <value> 0 or 1. There is also the option to write from <pin_n> the values <value_n> <value_n+1> <value_n+2> to the next pins.

Notes: Pin 0,1 connected to VAUX enable/disable accordingly
Pin 2 is read-only, Pin 3 is not connected
Pin 6,7 are connected to UMP 0,1

Examples:

```
1:> spio write 0 1
1:> spio write 0 {1 0 1}      -> write pin0=1 pin1=0 pin2=1
1:> spio write 4 {1 1 1}      -> write pin4=1 pin4=1
```

7.38 value

Usage: **value** <*value*>

Description: This command displays the value *value* in hexadecimal, binary, and decimal format.

Example:

```
1:> value 50
0x00000032 00000000 00000000 00000000 00110010   50
1:> value 0x4b
0x0000004b 00000000 00000000 00000000 01001011   75
```

7.39 version

Usage: **version**

Description: This command displays the version string for eDiag.

Example:

```
0:> version
ediag version: 4.0.4
```

7.40 xfer

The xfer command is used to send files through a serial port connection.

7.40.1 send

Usage: **xfer send** <*send_file*>

Description: This command is used to upload file *send_file* from device to a laptop computer running Hyperterminal via the serial port. The path of *send_file* is relative to "$env(DIAG_ROOT)/tmp" directory unless absolute path is specified. If *send_file* already exits, it will be overwritten. Note that the serial port has to be ready to use by issuing "**serialport open"** command.

7.40.2 receive

Usage: **xfer receive** <*rcv_file*>

Description: This command is used to download the file *rcv_file* from a laptop computer running Hyperterminal to the device. The path of *rcv_file* is relative to "$env(DIAG_ROOT)/tmp" directory unless absolute path is specified. Usually the terminal software issues the "**rz**" command to eDiag and user does not have to run the "**xfer receive**" command. If *rcv_file* already exits, it will be overwritten. . Note that the serial port has to be ready to use by issuing "**serialport open"** command. The directory $env(DIAG_ROOT)/tmp" must exist in order to receive the file if absolute path is not used.

8. Engineering Mode Macros

Engineering Mode Macros are very similar to Engineering Mode Commands in that they can both be executed from the engineering mode command prompt. The only effective difference is that the command is implemented completely in TCL and generally makes use of the Engineering Mode Commands.

8.1 qstat

Usage: **qstat**

Description: This macro displays the statistics gathered information from the Everest controller registers and firmware.

Example: `1:> qstat`

```
Tx Packets                          :                  100
Tx Unicast Packets                  :                    0
Tx Multicast Packets                :                    0
Tx Broadcast Packets                :                  100
Tx Octets                           :                11350
Rx Packets                          :                  100
Rx Unicast Packets                  :                    0
Rx Multicast Packets                :                    0
Rx Broadcast Packets                :                  100
Rx Octets                           :                11350
Dot3 FCS Errors                     :                    0
Dot3 Internal Mac Transmit Errors   :                    0
Dot3 Carrier Sense Errors           :                    0
Dot3 Alignment Errors               :                    0
Dot3 Single Collision Frames        :                    0
Dot3 Multiple Collision Frames      :                    0
Dot3 Deferred Transmissions         :                    0
Dot3 Excessive Collisions           :                    0
Dot3 Late Collisions                :                    0
Ether Collisions                    :                    0
In Frames L2 Filter Discards        :                    0
In TTL0 Discards                    :                    0
In Overflow Discards                :                    0
In MBUF Discards                    :                    0
In Errors                           :                    0
In No Brb Buffer                    :                    0
In False Carrier Errors             :                    0
Out Discards                        :                    0
```

8.2 reset_chip

Usage: **reset_chip**

Description: This macro resets the chip and puts it to a stable state.

8.3 Parse_rdf

Usage: **parse_rdf -f <RDF file> [optional: -b <Block name> -skip <Block to skip> -e < ending block>]**

Description: This macro parses the registers RDF file (everest.rdf) into global arrays to be used.

The –b option enables to parse only one block.
The –skip option enables to skip a block.
The –e option enables to stop parsing at the ending block and not at the end of file.

8.4 block_offset

Usage: **block_offset –b <block name>**

Description: This macro returns the block's offset from BAR by its name.

Example:

```
1:> parse_rdf -f everest.rdf
Completed parsing everest.rdf successfully
0
1:> block_offset -b BRB1
0x060000
```

8.5 block_regs

Usage: **block_regs –b <block name>**

Description: This macro returns the list of registers of this block.

Example:

```
1:> parse_rdf -f everest.rdf
Completed parsing everest.rdf successfully
0
1:> block_regs -b BRB1
BRB1_REGISTERS_MAXIMUM_NUMBER_OF_FULL_BLOCKS 0x000600b4
BRB1_REGISTERS_PORT_0_GUARANTIED 0x000600e0
BRB1_REGISTERS_PORT_2_GUARANTIED 0x000600e8
BRB1_REGISTERS_PORT_4_GUARANTIED 0x000600f0
```

8.6 reg_read

Usage: **reg_read -n <regName>**

Description: This macro reads a register's value by its name.

Example:

```
1:> parse_rdf -f everest.rdf
Completed parsing everest.rdf successfully
0
1:> reg_read -n PBF_REGISTERS_NUM_SENT_PKTS_P0
0x64
```

8.7 reg_offset

Usage: **reg_offset -n <regName>**

Description: This macro gets the register's offset address from BAR by its name.

Example:

```
1:> reg_offset -n PBF_REGISTERS_NUM_SENT_PKTS_P0
0x00140150
```

8.8 reg_size

Usage: **reg_size -n <regName>**

Description: This macro gets the register's size (in bits) by its name.

Example:

```
1:> reg_size -n PBF_REGISTERS_NUM_SENT_PKTS_P0
24
```

8.9 reg_type

Usage: **reg_type -n <regName>**

Description: This macro gets the register's access type (RW/R/ST/WB) by its name.

Example:

```
1:> reg_type -n PBF_REGISTERS_NUM_SENT_PKTS_P0
ST
```

8.10 reset_value

Usage: **reset_value -n <regName>**

Description: This macro gets the register's reset value by its name.

Example:

```
1:> reset_value -n PBF_REGISTERS_NUM_SENT_PKTS_P0
0x0
```

8.11 reg_info

Usage: **reg_info -n <regName>**

Description: This macro prints all the registers information, including: offset, size, type and reset value..

Example:

```
1:> reg_info -n PBF_REGISTERS_NUM_SENT_PKTS_P0
offset: 0x00140150 size: 24 type: ST
```

8.12 read_regs

Usage: **read_regs -out <outfile> [-all -b <blockName> -noPrint]**

Description: This macro reads all the registers values and print them into a file

The -all options enables to read all registers (this is the default)

The -b <blockname> option reads only this block registers

The -noPrint options disables any extra print-outs

Example:

```
1:> read_regs -b PBF
name          offset    size   type   value
PBF_REGISTERS_CCM_AGG_DEC_TYPE    0x00140130    4   RW  0x5
PBF_REGISTERS_CCM_HDR  0x0014012c    12  RW  0x0
PBF_REGISTERS_CCM_INIT_CRD    0x001400c0    4   RW  0x6
PBF_REGISTERS_CCM_MSG_CNT  0x0014016c    24  ST  0x0
PBF_REGISTERS_CMDS_RCVD_ON_P0     0x00140188    24  ST  0x0
PBF_REGISTERS_CMDS_RCVD_ON_P1     0x0014018c    24  ST  0x0
```

8.13 idle_chk

Usage: idle_chk <-ignoreAssert>

Description: This macro is used to verify that the chip is in the IDLE state. This script is useful for running via UART incase the machine on which the chip is running is stuck.

8.14 Phy

Usage: phy <port> <reg> [value]

Description: Read/Write PHY register.

In case of read, no need for the [value] field.

8.15 extphy

Usage: extphy <port> <dev> <reg> [value]

Description: Read/Write external PHY register.

In case of read, no need for the [value] field.

9. Configuration files

The eDiag Diagnostic tool supports various configuration files which are used to configure or verify various Everest device parameters during manufacturing. This section will document the usage and syntax for each of these files.

The **macaddr.txt** provides a range of MAC address values that can be used to program the primary MAC and iSCSI MAC address onto the device for OEM manufacturing.

9.1 macaddr.txt

The **macaddr.txt** file is used to program the primary MAC and iSCSI MAC address of all Everest devices during OEM manufacturing. It consists of a range of MAC addresses that are automatically updated after the MAC address is programmed and is invoked from the eDiag command-line as follows:

eDiag –fmac macaddr.txt

Note that the MAC addresses must be consecutive and that the primary MAC address will be assigned first. After the MAC addresses have been programmed the **macaddr.txt** file will be updated such that the "**mac_addr_start**" value is incremented by two or four (depending of the number of devices on board) to reflect the allocated addresses. For example, if **mac_addr_start** has a value of 042240 before **ediag –fmac macaddr.txt** is run, after the command is executed, **mac_addr_start** will have a value of 042244. When "**mac_addr_start**" value equals "**mac_addr_end**" value, it means that the range of addresses provided when the file is first used have all been taken and a new range has to be specified.

Below is a description of each field in **macaddr.txt**.

PARAMETER	DESCRIPTION
mac_addr_pref	Specify the first 3 bytes of MAC address. Values are in hexadecimal without "0x" prefix.
mac_addr_start	Specify the start of the address range. The first value forms the last 3 bytes of primary MAC address and the next consecutive value forms the last 3 bytes of iSCSI MAC address. Values are in hexadecimal without "0x" prefix.
mac_addr_end	Specify the end of the address range. Values are in hexadecimal without "0x" prefix.

The following is a sample macaddr.txt file.

```
mac_addr_pref   =  001018
mac_addr_start  =  042240
mac_addr_end    =  042246
```

10. Common eDiag Tasks

This section will provide examples of common tasks that are performed with eDiag.

10.1 Writing scripts

The command set can be written in a form of a script to allow regression using the "-rc" command line option. Keep in mind that although the command can be abbreviated at the command prompt, the script requires the commands to be fully spelled out. This promotes the readability of the script and avoids ambiguity.

Scripts can include jointed commands separated by ;

For example:

1:> nvm dir MBA; puts "Testing"

Note:

Some commands set the TCL interpreter result, thus running them with a second joined command will cause overwrite of the first command result.

Thus it is not recommended to use the following commands within joined commands:

"nvm prg", "nvm read", "device –l", "driver get_context", "gpio read", "spio read", "hmem alloc/palloc", "l2pkt", "l4data", "l4fpath", "l4spath", "multiSend", "pci search", "sb", "license chkkey/display".

10.2 Configuring and Exporting an Everest NVRAM Image

This task will focus on installing the Everest bootcode on a blank NVRAM, performing the initial configuration of the controller, and saving the contents of NVRAM to a separate file for use as a "Golden Master" for manufacturing.

Start eDiag in engineering mode with the command-line "ediag –b10eng". Configure NVM default setting using "nvm cfg <board type>". The bootcode is programmed using the "nvm upgrade" command. Dump the NVM content into file using "nvm dump <filename.img>"

Examples:

```
1:> nvm cfg A1020G
1:> nvm upgrade -F -bc1 bc1.bin
1:> nvm dump image.img
write_bin_file: file write 262144 bytes; buf=007f50d0
```

10.3 Checking NVM configuration and content

This task will describe how to check NVM configuration using "nvm cfg" and "nvm chk" commands. Checking NVM contet using "nvm dir" and "nvm crc" commands.

The "nvm cfg –dump filename.txt" enables to dump the NVM configuration into a file.

The "nvm chk <board type>" command checks the current NVM configuration against the expected default NVM settings for this board type.

The "nvm chk <board type> -fix" enables to fix the differences in NVM configuration against the expected default settings.

Examples:

```
1:> nvm cfg –dump tmp.txt

1:> nvm chk T1001

The following errors were found in the configuration:

ID 40 for port 0: "Serdes external PHY address (8 bits)" mismatch:

   Expected 10

   Existing 0

1:> nvm chk T1001 –fix


1:> nvm dir
     Image   SRAM Addr  NVM Offset    Byte Cnt     CPU       Version
-------------------------------------------------------------------------
       BC1  0x08000010  0x00000818  0x000029B4     MCP  BC1 4.0.2.0D
       BC2  0x08008180  0x00003244  0x00006B6C     MCP  BC2 0.30.22.
       UMP  0x08000010  0x00009DB0  0x00007D18     MCP  UMP 0.28.21.
1:> nvm crc
    Region       Start      Length     Content    Computed  Result
-------------------------------------------------------------------------
 BOOTSTRAP  0x00000000  0x00000014  0xD2CC5FC1  0xD2CC5FC1  OK
 DIRECTORY  0x00000014  0x000000EC  0x751CD84D  0x751CD84D  OK
   HW_INFO  0x00000100  0x00000350  0xB5DF17EB  0xB5DF17EB  OK
 FEAT_INFO  0x00000450  0x000000F0  0x2CB3B527  0x2CB3B527  OK
       BC1  0x00000818  0x000029B4  0x2E7ECABE  0x2E7ECABE  OK
       BC2  0x00003244  0x00006B6C  0x09E42778  0x09E42778  OK
       UMP  0x00009DB0  0x00007D18  0x3D914929  0x552A1424  Mismatch
```

Appendix A – Tcl Reference

The eDiag utility draws extensively on the Tool Command Language (or Tcl) for its engineering mode interface, allowing a rich environment for developing and implementing scripts. Please refer to http://www.tcl.tk for additional information on the basic Tcl command syntax. The eDiag utility is based on the Tcl command interpreter v8.3.3.

Below there are two TCL tutorials and manuals that can be of use to beginners:

http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html

http://www.astro.princeton.edu/~rhl/Tcl-Tk_docs/tcl8.0a1/contents.html

Appendix B – TCL Environment Variables

The eDiag utility provides numerous environment settings for users to develop their own testing and configuration scripts. Tcl environment variables are accessed using the syntax $::<array_name>(<array_idx>).

B.1 env

This environment variable inherits the setting from DOS. Depending upon what is set while in the DOS environment, users can also retrieve the DOS setting via this variable. For example, $::env(COMSPEC) could be C:\COMMAND.COM.

B.2 current

This variable maintains information for the currently selected device. Note that these variables are meant to be read-only, and they will change as users switch from one device to another.

- $::current(DID) – the device ID of selected device (e.g. "5730")
- $::current(CHIP_REV) – the chip revision of the selected device (e.g. "A0")
- $::current(DEV) – currently selected device (value = 0, 1, …)
- $::current(DRV_STATE) – the current state of the driver (e.g. "SETUP", "RUNNING", etc.)
- $::current(MAC_ADDR) – the MAC address of the selected device (e.g. "001018010B23")
- $::current(PHY_TYPE) – the PHY medium type of the device (e.g. "COPPER", "SERDES")
- $::current(TOTAL_DEV) – total number of devices (e.g. 1, 2)
- $::current(FUNC) – stores the function number of the current device.
- $::current(NVM_SIZE) – stores the NVMRAM size in bytes (524288 in case of 4Mbit NVM, 1048576 in case of 8Mbit NVM).

B.3 sys

The sys environment variable is used to maintain state information used by eDiag. Changing any of these variables will change the behavior of eDiag.

- $sys(IO_PORT) – the COM port number that the I/O will go to. This is usually set for the command "**serialport open**".
- $sys(STAT_PORT) – the COM port number that the statistic window will go to. This is usually set for the command "**serialport open**".
- $sys(ARG) stores the arbitrary argument for any internal test scripts to use. The command line switch "**-arg**", followed by a string must be included at the time **eDiag** is invoked.

- $sys(DO_DEF_SEL) – stores the status of the device selection. If set default device (1) is selected. Otherwise, no device is selected.
- $sys(NO_PCI) – If set, then PCI bus is used to access GRC, other wise Debug UART should be used while accessing GRC.

Appendix C – External PHYs

eDiag supports different external PHYs:

BCM8072

BCM8073

BCM8705

BCM8706

BCM8726

BCM8727

BCM8481

BCM84823

BCM54640

SFX7101

Some of the external PHYs support firmware upgrade using "ext_phy_fw upgrade".

The external PHYs which support firmware upgrade are:

- BCM8073
- BCM8726
- SFX7101
- BCM8727
- BCM8481
- BCM84823
- BCM84833
- BCM8706

Appendix D – Setting up NIC PARTITIONING

The NIC PARTITION feature allows up to 8 functions per port.

To set up NIC PARTITIONING, user should enter "nvm cfg" menu and select the "7. npar" menu.

Then enable option 114 "MAC partition global cfg". After changing that value, a new NPAR NVRAM image will be generated.

```
1:> nvm cfg
                           NVRAM Configuration Groups:
                           ---------------------------
1 . board config
2 . board i/o
3 . dual phy
4 . features
5 . link settings
6 . management fw
7 . npar
8 . pcie
9 . phy
10 .pre-boot
11 .vf


Select: (q to quit)
7
                              Group: npar (Group 7)
--------------------------------------------------------------------------------
114: [S] MAC partition global cfg
        {Disabled(0), Enabled(1)}                       : Enabled
115: [P] MAC partition flow control
        {auto(0), tx_only(1), rx_only(2), both(3),
         none(4)}                                       : auto
116: [P] MAC partition physical link speed
        {1G(0), 2.5G(1), 10G(2)}                        : 1G
118: [F] MAC partition func flags
        {enabled(0x1), ethernet(0x2),
         iSCSI_offload(0x4), FCOE_offload(0x8)}         : 0
119: [F] MAC partition func bandwidth weight            : 0
120: [F] MAC partition func max bandwidth               : 0
121: [F] MAC partition func net MAC addr                : 00:00:00:00:00:00
122: [F] MAC partition func iscsi MAC addr              : 00:00:00:00:00:00
123: [F] MAC partition func fcoe MAC addr               : 00:00:00:00:00:00
136: [F] mac partition func fcoe node wwn mac addr      : 00:00:00:00:00:00
137: [F] mac partition func fcoe port wwn mac addr      : 00:00:00:00:00:00
138: [S] niv image version                              : 0
```

Additional settings required:

- Install the latest MBA image (evpxe.nic) from the following folder:

  \\brcm-irv\dfs\projects\nseg\rels\bcm57710\drivers\mba\<VERSION>\fw30_clp

- Enable MBA via "nvm cfg 22=1"
- Set nvm cfg option 73 "Forct SF Mode" to Switch Indpet(3).

Then, there are two options to change NPAR values:

- Using MBA - reboot the system, and during boot, press "Ctrl+S". Choose the "NIC PARTITION" menu and enable it on both devices.

- Using eDiag – manually change the NPAR values detailed above, as required (using "nvm cfg <field>=<new value>"). Note that some of the fields are per port and per function. The only change will be to the value relevant to the current device's port/function.

Appendix E – Setting up NIC PARTITIONING Switch Dependent (SD) Mode

The NIC PARTITION SD feature allows up to 8 functions per port.

To set up NIC PARTITIONING SD mode, user should enter "nvm cfg" menu and select "7. npar" menu.

Then change the value of option 138 "niv image version" to 1. After changing that value, two new NPAR-SD NVRAM images will be generated.

```
1:> nvm cfg
                            NVRAM Configuration Groups:
                            ---------------------------
1 . board config
2 . board i/o
3 . dual phy
4 . features
5 . link settings
6 . management fw
7 . npar
8 . pcie
9 . phy
10 .pre-boot
11 .vf


Select: (q to quit)
7
                                Group: npar (Group 7)
--------------------------------------------------------------------------------
114: [S] MAC partition global cfg
        {Disabled(0), Enabled(1)}                       : Disabled
138: [S] niv image version                              : 1
139: [P] niv port flow control
        {auto(0), tx_only(1), rx_only(2), both(3),
         none(4)}                                       : auto
140: [P] niv port physical link speed
        {1G(0), 2.5G(1), 10G(2)}                        : 1G
142: [P] niv port profiles list                         :
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
143: [F] niv func bw weight                             : 0
144: [F] niv func max bandwidth                         : 0
145: [F] niv func vif type
        {enabled(0x1), ethernet(0x2),
         iSCSI_offload(0x4), FCOE_offload(0x8)}         : 0
146: [F] niv func remote boot enabled
        {Disabled(0), Enabled(1)}                       : Disabled
148: [F] niv func net mac addr                          : 00:00:00:00:00:00
149: [F] niv func iscsi mac addr                        : 00:00:00:00:00:00
150: [F] niv func fcoe mac addr                         : 00:00:00:00:00:00
more.., 'q' to quit
151: [F] niv func fcoe node wwn mac addr                : 00:00:00:00:00:00
152: [F] niv func fcoe port wwn mac addr                : 00:00:00:00:00:00
153: [F] niv func profile name                          :
```

Additional settings required:

- Install the latest MBA image (evpxe.nic) from the following folder:

  \\brcm-irv\dfs\projects\nseg\rels\bcm57710\drivers\mba\<VERSION>\fw30_clp

- Enable MBA via "nvm cfg 22=1"
- Set nvm cfg option 73 "Forct SF Mode" to NIV mode(4).

Then, there are two options to change the NPAR-SD values:

- Using MBA - reboot the system, during boot press "Ctrl+S". Choose the "VNTAG configuration" menu and enable it on both devices.
- Using eDiag – manually change the NPAR-SD values detailed above as required (using "nvm cfg <field>=<new value>"). Note that some of the fields are per port and per function. The only change will be to the value relevant to the current device's port/function.