



QLogic Network Adapters

XDiag OEM User's Guide

Firmware Version 5.8

SN0054691-00 A

CONFIDENTIAL

Information furnished in this manual is believed to be accurate and reliable. However, QLogic Corporation assumes no responsibility for its use, nor for any infringements of patents or other rights of third parties which may result from its use. QLogic Corporation reserves the right to change product specifications at any time without notice. Applications described in this document for any of these products are for illustrative purposes only. QLogic Corporation makes no representation nor warranty that such applications are suitable for the specified use without further testing or modification. QLogic Corporation assumes no responsibility for any errors that may appear in this document.

Document Revision History	
Revision 5.8, January 22, 2015	
Changes	Sections Affected
Initial release	

Table of Contents

1	Introduction	
2	Operating Environment	
	MS-DOS	2
	Linux	2
	Unified Extensible Firmware Interface	3
	Option A - LAN on Motherboard Device	3
	Option B - NIC Device	4
	Input File List	5
3	Modes of Operation	
	Manufacturing Mode	6
	Engineering Mode	6
	Command Prompt	6
4	Xdiag Test List	
	Group A - Basic Functional Tests	9
	Register Test (A1)	9
	PCI Configuration Test (A2)	9
	Interrupt Test (A3)	10
	PCI-Express Link Test (A4)	10
	MSI Test (A5)	11
	Memory BIST (A6)	11
	Network Link Test (A7)	12
	MSI-X Test (A8)	12
	Group B - Memory Tests	13
	Group C - Block Tests	15
	CPU Logic and DMA Interface Test (C1)	15
	RBUF Allocation Test (C2)	15
	Content Addressable Memory Access Test (C3)	16
	TPAT Cracker Test (C4)	16
	FIO Register Test (C5)	17
	NVM Access and Reset-Corruption Tests (C6)	17
	Core-Reset Integrity Test (C7)	18

	DMA Engine Test (C8)	18
	VPD Test (C9)	19
	FIO Events Test (C11)	20
	Context Caching Test (C12)	21
	Data Integrity Test (C13)	22
	Receive Side Scaling Hash Test (C14)	22
	Status Block Test (C15)	23
	Memory Parity Test (C16)	23
	CPU Multiplier Test (C17)	24
	RXP CRC Test (C18)	24
	Tx BD Cache Test (C19)	25
	System Timer Accuracy Text (C20)	25
	Boot Code State Test (C21)	26
	Group D - Ethernet Traffic Tests	27
	MAC Loopback Test (D1)	27
	Physical Layer Loopback Test (D2)	28
	External Loopback Test (D3)	29
	Large Sent Offload Test (D4)	30
	EMAC Statistics (D5)	31
	Remote Procedure Call Test (D6)	31
	Universal Management Port MAC Loopback Test (D7)	32
	Universal Management Port External Loopback Test (D8)	33
5	Xdiag Command Line Options and Return Codes	
	Command Line Options	34
	Return Codes	38
6	Engineering Mode Commands	
	bits	44
	clear	44
	set	44
	cpu	45
	ctrl	45
	halt	45
	load	46
	mem	46
	run	47
	select	47
	gpr, step, disasm, fioaddr, uc fioaddr, grc, fill	47
	delay	48

us	48
ms	48
device	49
disp64	51
dma	51
read	52
write	53
driver	54
intr	54
intrcnt	54
load	55
poll	55
unload	55
init, run, pause, shutdown, wol, reset, rxmask, txbatch, submit, stat, statslave	55
exit	56
ftq	56
gpio	57
read	57
write	57
help	58
hex	59
hex16	59
hex32	59
hmem	60
alloc	60
fill	60
.	60
free	61
inuse	61
paddr	61
palloc	62
read	62
show	62
write	63
ioport	64
read	64
write	64
ipmi	65

cfg	65
crc	66
keyhit	66
l2pkt	66
l2spec	67
license	67
secret	67
storekey	67
rmkey	68
display	68
chkkey	69
dump	69
log	70
open	70
close	70
mcast	70
mii	71
[slave <device number>] [copper serdes] read	71
[slave <device number>] [copper serdes] show	71
[slave <device number>] [copper serdes] write	72
rread	72
rshow	72
rwrite	73
nictest	73
nvm	74
cfg	74
crc	75
dir	75
dump	76
fill	76
prg	77
read	78
show	78
write	78
upgrade	79
rescfg	80
usrblk	80
pci	81
init	81

scan	81
search	82
setdut	82
pcicfg	83
read	83
show	83
write	83
qhack	84
rebuf	84
reg	84
read, iread, and dread	84
show and ishow	85
trace	85
write, iwrite and dwrite	85
serialport	86
open	87
redirect	87
close	87
sram	88
fill	88
read	88
show	89
write	89
stats	90
value	90
version	90
watch	90
what	90
wol	90
xfer	91
send	91
receive	91
yield	91
?	91

7 Engineering Mode Macros

blast	93
init	93
loadfw	94
qstat	95

	redirect	95
	reset	96
	sb	96
	tb	97
	txcfg	99
8	Configuration Files	
	config.txt	101
	macaddr.txt	105
	cfgchk.txt	106
A	Appendix A - Tcl Reference	
B	Appendix B-TCL Environment Variables	
	env	110
	toe	110
	drv06_xx	112
	sys	113

1 Introduction

This document provides information on how to use QLogic's xdiag utility for manufacturing and engineering level diagnostics for the QLogic Network line of Ethernet controllers. This manual is intended for engineers, testers, and technicians at QLogic and QLogic's OEM customers and should not be distributed to end-users.

2 Operating Environment

The xdiag utility has been ported to various OS and hardware platforms to support different customer needs. Historically, the utility has been used mostly in DOS environment; thus, the remaining sections of the document tend to be DOS-centric. Moreover, some of the engineering mode commands such as **ioport** may be applicable for DOS only.

MS-DOS

The xdiag 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 1MB.

OS: MS-DOS 6.22

Additional Driver(s): ANSI SYS

To install this driver, simply put ANSI.SYS file in some directory (e.g. C:\DOS) and add another line (e.g. DEVICE=C:\DOS\ANSI.SYS) in the config.sys file. The ANSI driver is used to allow manufacturing mode tests to display a clear indication that a test has passed (using green) or that a test has failed (using red). The xdiag utility will run correctly without the ANSI driver installed.

Software: xdiag.exe

Linux

The xdiag utility operates in a Linux environment, running on kernels specified below. It needs two additional kernel drivers (specified below and included in the package) in order for the xdiag to access the hardware. The main purpose of supporting Linux is for manufacturing use. Some of the command support in engineering mode may not be available.

OS: Linux kernels provided in RHEL 4.x and 5.x; SLES 9.x and 10.x.

Hardware platforms: Intel x86, Intel x64, and PowerPC

Additional driver(s): bxbdrv.ko and b06kdiag.ko

Both drivers are shipped in hybrid forms. Some modules are in source forms while others are in object file format. Users are expected to build the drivers using the source code and object files, and load the drivers on their own. However, scripts are provided in the package to facilitate this.

Limitation: It is required to have bnx2 driver removed from the system for **bxbdrrv** and **b06kdiag** drivers to be loaded and used properly.

Software: xdiag (user mode application) and libtcl8.3.so (shared library)

Unified Extensible Firmware Interface

The xdiag utility operates in a unified extensible firmware interface environment, whether it is an emulated “Duet” (shell running on top of legacy systems) or native platform (where unified extensible firmware interface firmware is part of the system).

OS: UEFI v2.10

Hardware platforms: Intel x64

Additional driver(s): none

Software: efi_xdiag64.exe

If pre-execution environment (PXE)/MBA is enabled in nvram image, it will cause unpredictable results (even system hang) when xdiag regression tests run in unified extensible firmware interface shell. To avoid this problem, there are two approaches that you can try.

If after taking either of the approaches and xdiag still fails, you may try combining both approaches by Option B first, then Option A

Option A - LAN on Motherboard Device

If it is a LAN on motherboard (LOM) device in the system, the system unified extensible firmware interface firmware may have loaded the PXE/ MBA driver for each port of the QLogic Network LOMs (and NICs if present).

Complete the following procedure:

1. Manually unload the PXE/MBA driver at unified extensible firmware interface shell before running xdiag by typing **drivers** at UEFI shell. This shell command displays all drivers loaded.
2. Locate each QLogic PXE/MBA driver instance and run **unload x** where **x** is the QLogic PXE/MBA driver handle appearing in the first column of the corresponding QLogic Network PXE/MBA driver entry from **drivers** command.
3. After unloading all instances of QLogic PXE/MBA driver, run xdiag without system reboot.

Option B - NIC Device

If it is a NIC device in the system, the nvram configuration parameter option 21 has to be set to 0 to disable loading PXE driver.

Complete the following steps to disable PXE driver to be loaded:

1. Go to engineering mode by typing:
`xdiag -b06eng`
2. Type the following command at prompt:
`nvm cfg 21=0`
3. Repeat step 3 for all devices by typing:
`dev x (where x is device number 2, 3, 4...)`
`nvm cfg 21=0`
4. Type the following command to exit xdiag:
`exit`
5. Reboot the system.

If you want to enable PXE driver again, repeat Steps 1-5 with Step 2 and Step 3 changed to: `nvm cfg 21=1`

Input File List

The following files should be in the same location as the **xdiag.exe**. Without these files, some of the diagnostic tests will not function properly. Some other manufacturing features rely on these files as well.

- Firmware image files stored in **diagfw** directory are required when running many of the block diagnostics discussed in [“Xdiag Test List” on page 8](#).
- **config.txt**
This text file is used to specify detailed configuration of the design (for example MAC address, PCI vendor ID, WOL setting) and is generally used for manufacturing. Refer to [“config.txt” on page 101](#) for more details. This file is not required for normal xdiag operation.
- **macaddr.txt**
This text file specifies a range of available MAC addresses (for L2 and iSCSI functionality) to use when programming the MAC address of the QLogic Network 1G controller. As addresses are programmed into the QLogic Network 1G controller, xdiag will update the **macaddr.txt** file and reduce the range of available addresses by the number used. Refer to [“macaddr.txt” on page 105](#) for more details. This file is not required for normal xdiag operation.
- **cfgchk.txt**
This text file specifies some of the configuration settings and serves as a sanity check against the design on the manufacturing floor, making sure that the correct configuration is programmed. Refer to [“cfgchk.txt” on page 106](#) for more details. This file is not required for normal xdiag operation.

3 Modes of Operation

The xdiag utility has two modes of operation, manufacturing mode and engineering mode.

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:\>xdiag`
- `C:\>xdiag -t alc2`
- `C:\>xdiag -mac 11:22:33:44:55:66 -wol 0`

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 xdiag in engineering mode:

- `C:\>xdiag -b06eng`

Command Prompt

When xdiag is executing in engineering mode it will display a command prompt and wait for a user to enter commands (documented in [“Engineering Mode Commands” on page 43](#)).

The command prompt has the following format:

- `<NetXtreme II Device>.<Current NetXtreme II CPU>:>`

For example, a command prompt of **1.NONE:>** indicates that xdiag is currently operating on the first QLogic Network device (as found on the PCI bus) and that none of the internal CPUs have been selected. A command prompt of **3.MCP:>** would indicate that the third QLogic Network device is selected and that the MCP processor of that device is selected.

The xdiag utility can also be used to examine the PCI registers of non-QLogic Network devices (using the **setdut** command described in [“setdut” on page 82](#)). In those cases the command prompt will be displayed as **99.NONE:>**.

4 Xdiag Test List

The xdiag utility provides a wide range of tests to verify specific blocks of functionality within the QLogic Network controller. The tests are divided into 4 groups (labeled A through D), with a variable number of specific tests within each group.

Running the Default Test Suite in Manufacturing Mode

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

```
C:\>xdiag
```

Running the Default Test Suite in Engineering Mode

To run the default test suite in engineering mode, type the following command at the engineering mode command prompt:

```
1.NONE:>nictest
```

Some of the diagnostic tests described in this section will require external configuration for the test to pass (such as an external loopback adapter). These tests must be explicitly enabled to run otherwise they may indicate a false test failure.

Group A - Basic Functional Tests

Register Test (A1)

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

Table 4-1. Register Test (A1)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T a1
Engineering Mode Command	1.NONE:>regtest
Return Codes	0 - Test passed
Possible Failure Causes	Any failure usually indicates a defective controller.
Enabled by Default	Yes
Special Requirements	Some critical registers are not tested as the system and/or the chip may become unstable when the register value is changed.

PCI Configuration Test (A2)

The PCI Configuration Test checks the functionality of the PCI base address register by varying the amount of memory requested by the base address register and verifying that the base address register actually requests the correct amount of memory (without actually mapping the base address register into system memory). Refer to PCI or PCI-E specification for details on the base address register and its addressing space.

Table 4-2. PCI Configuration Test (A2)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T a2
Engineering Mode Command	1.NONE:>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

Interrupt Test (A3)

The Interrupt Test generates a PCI interrupt and verifies that the system receives the interrupt and invokes the correct interrupt service routine. A negative test is also performed to verify that a masked interrupt does not invoke the interrupt service routine.

Table 4-3. Interrupt Test (A3)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T a3
Engineering Mode Command	1.NONE:>intrtest
Return Codes	0 - Test passed
Possible Failure Causes	Any failure could indicate a chip defect, improper PCI interrupt routing, improper chip installation on the system, or a defective host system interrupt controller.
Enabled by Default	Yes
Special Requirements	None

PCI-Express Link Test (A4)

The PCI Express Link Test checks that all supported PCI-E lanes are operating correctly.

Table 4-4. PCI-Express Link Test (A4)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T a4
Engineering Mode Command	1.NONE:>elinktest
Return Codes	0 - Test passed
Possible Failure Causes	A failure may indicate that the chip is defective, the chip is installed in a slot that supports fewer PCI-E lanes than the chip is capable of supporting, or that there is a physical connection problem in one or more PCI-E lanes.
Enabled by Default	Yes (Only for PCI-E devices)
Special Requirements	This test is only applicable to PCI-E devices such as the BCM5708.

MSI Test (A5)

The Message Signaled Interrupt (MSI) Test verifies that an MSI causes an MSI message to be DMA'd to host memory. A negative test is also performed to verify that when MSI is masked it does not write an MSI message to host memory.

Table 4-5. MSI Test (A5)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T a5
Engineering Mode Command	1.NONE:>msitest
Return Codes	0 - Test passed
Possible Failure Causes	A failure may indicate that the chip is defective, the chip is improperly installed on the system, or that it may be related to any failures found in C1 and C8 tests.
Enabled by Default	Yes
Special Requirements	None

Memory BIST (A6)

The Memory Built-In Self Test (BIST) Test invokes the internal chip BIST command to test internal memory.

Table 4-6. Memory BIST (A6)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T a6
Engineering Mode Command	1.NONE:>bisttest
Return Codes	0 - Test passed
Possible Failure Causes	A failure may indicate that the chip is defective.
Enabled by Default	Yes
Special Requirements	None

Network Link Test (A7)

The Network Link Test verifies the physical connection for SerDes devices and is identical to the External Loopback Test (D3).

Table 4-7. Network Link Test (A7)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T a7
Engineering Mode Command	1.NONE:>netlinktest
Return Codes	0 - Test passed
Possible Failure Causes	A failure may indicate that the chip is defective or that no external loopback adapter is installed.
Enabled by Default	No
Special Requirements	This test is only applicable to SerDes controllers such as the BCM5706S or BCM5708S.

MSI-X Test (A8)

The Message Signaled Interrupt (MSI-X) Test verifies that an MSI causes an MSI message to be DMA'd to host memory. A negative test is also performed to verify that when MSI is masked it does not write an MSI message to host memory and the corresponding pending bit is set.

Table 4-8. MSI-X Test (A8)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T a8
Engineering Mode Command	1.NONE:>msixtest
Return Codes	0 - Test passed
Possible Failure Causes	A failure may indicate that the chip is defective, the chip is improperly installed on the system, or that it may be related to any failures found in C1 and C8 tests.
Enabled by Default	Yes (only for 5709)
Special Requirements	None

Group B - Memory Tests

The Group B tests verify all memory blocks of the QLogic Network Adapter by writing various data patterns (0x55aa55aa, 0xaa55aa55, walking zeros, walking ones, address) to each memory location and then reading back the data, comparing it to the value written. The fixed data patterns are used to ensure that no memory bit is stuck high or low, while the walking zeros/ones and address tests are used to ensure that memory writes do not corrupt adjacent memory locations.

1. TXP scratchpad (B1)
2. TPAT scratchpad (B2)
3. RXP scratchpad (B3)
4. COM scratchpad (B4)
5. CP scratchpad (B5)
6. MCP scratchpad (B6)
7. TAS header buffer (B7)
8. TAS payload buffer (B8)
9. RBUF via GRC (B9)
10. RBUF via indirect access (B10)
11. RBUF Cluster list (B11)
12. TSCH List (B12)
13. CSCH List (B13)
14. RV2P scratchpads (B14)
15. TBDC memory (B15)
16. RBDC memory (B16)
17. CTX page table (B17)
18. CTX memory (B18)
19. CTX CAM memory (B19) (only for 5709)
20. CTX Mirror memory (B20) (only for 5709)
21. CTX Usage Count memory (B21) (only for 5709)
22. FTQ memory (B22) (only for 5709)
23. MCP Egress memory (B23) (only for 5709)
24. MCP Ingress memory (B24) (only for 5709)
25. History Buffer memory (B25) (only for 5709)

Table 4-9. Group B Memory Tests

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T bX (where X is replaced by the specific test)
Engineering Mode Command	1.NONE:>memtest
Return Codes	0 - Test passed
Possible Failure Causes	A failure may indicate that the chip is defective or that the chip was improperly installed on the system (especially for PCI/PCI-X devices such as BCM5706).
Enabled by Default	Yes
Special Requirements	The memtest command available in engineering mode cannot select individual tests.

Group C - Block Tests

CPU Logic and DMA Interface Test (C1)

The CPU Logic and DMA Interface Test verifies the basic logic functionality of all the on-chip CPUs. It also exercises the DMA interface exposed to those CPUs. The internal CPU will try to initiate DMA activities (both read and write) to system memory, then compare the values to confirm that the DMA operation completed successfully.

Table 4-10. CPU Logic and DMA Interface Test (C1)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T c1
Engineering Mode Command	1.NONE:>cputest
Return Codes	0 - Test passed
Possible Failure Causes	A failure may indicate that the test firmware could not be located, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	Requires the presence of a special test firmware file inside the diagfw directory of the xdiag distribution.

RBUF Allocation Test (C2)

The RBUF Allocation Test verifies the RX buffer allocation interface by allocating and releasing buffers, checking that the RBUF block maintains an accurate count of the allocated and free buffers.

Table 4-11. RBUF Allocation Test (C2)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T c2
Engineering Mode Command	1.NONE:>rbufatest
Return Codes	0 - Test passed
Possible Failure Causes	A failure may indicate that the chip is defective
Enabled by Default	Yes
Special Requirements	None

Content Addressable Memory Access Test (C3)

The content addressable memory access test verifies the content addressable memory block by performing read, write, add, modify, and cache hit tests on the content addressable memory associative memory.

Table 4-12. Content Addressable Memory Access Test (C3)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c3</code>
Engineering Mode Command	<code>1.NONE:>rluptest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure may indicate that the chip is defective.
Enabled by Default	Yes
Special Requirements	None

TPAT Cracker Test (C4)

The TPAT Cracker Test verifies the packet cracking logic block (the ability to parse TCP, IP, and UDP headers within an Ethernet frame) as well as the checksum/CRC offload logic. In this test, packets are submitted to the chip as if they were received over Ethernet and the TPAT block cracks the frame (identifying the TCP, IP, and UDP header data structures) and calculates the checksum/CRC. The TPAT block results are compared with the values expected by xdiag and any errors are displayed.

Table 4-13. TPAT Cracker Test (C4)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c4</code>
Engineering Mode Command	<code>1.NONE:>tpatctest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure may indicate that the test firmware could not be located, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	Requires the presence of a special test firmware file inside the diagfw directory of the xdiag distribution.

FIO Register Test (C5)

The Fast-IO (FIO) Register Test verifies the register interface that is only exposed to the internal CPUs.

Table 4-14. FIO Register Test (C5)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c5</code>
Engineering Mode Command	<code>1.NONE:>diagfwtest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure may indicate that the test firmware could not be located or that the chip is defective.
Enabled by Default	Yes
Special Requirements	Requires the presence of a special test firmware file inside the diagfw directory of the xdiag distribution.

NVM Access and Reset-Corruption Tests (C6)

The NVM Access and Reset-Corruption Test verifies non-volatile memory accesses (both read and write) initiated by one of the internal CPUs. It also tests for appropriate access arbitration among multiple entities (CPUs). Finally, it checks for possible NVM corruption by issuing a chip reset while the NVM block is servicing data.

Table 4-15. NVM Access and Reset-Corruption Tests (C6)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c6</code>
Engineering Mode Command	<code>1.NONE:>nvmtest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure may indicate that the test firmware could not be located, that an error was encountered on the NVM interconnect, that the NVM device is defective, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	Requires the presence of a special test firmware file inside the diagfw directory of the xdiag distribution. If this test is run in a continuous loop the write test may reduce the effective lifetime of the NVRAM device.

Core-Reset Integrity Test (C7)

The Core-Reset Integrity Test verifies that the chip performs its reset operation correctly by resetting the chip multiple times, checking that the bootcode and the internal xdiag driver loads/unloads correctly.

Table 4-16. Core-Reset Integrity Test (C7)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c7</code>
Engineering Mode Command	<code>1.NONE:>resettest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure may indicate that the bootcode was not previously programmed correctly, that an error was encountered on the NVM interconnect, that the NVM device is defective, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	None

DMA Engine Test (C8)

The DMA Engine Test verifies the functionality of the DMA Engine block by performing numerous DMA read and write operations to various system and internal memory locations (and byte boundaries) with varying lengths (1 byte to over 4KB) crossing CRC checks are performed to ensure data integrity. The DMA write test also verifies that DMA writes do not corrupt the neighboring host memory.

Table 4-17. DMA Engine Test (C8)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c8</code>
Engineering Mode Command	<code>1.NONE:>dmatest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure may indicate that the test firmware could not be located, that an error was encountered on the PCI/PCI-E bus, that the chip is improperly installed on the system, that there are issues with the system chipset, or that the chip is defective.
Enabled by Default	Yes

Table 4-17. DMA Engine Test (C8)

Feature	Description
Special Requirements	Requires the presence of a special test firmware file inside the diagfw directory of the xdiag distribution.

VPD Test (C9)

The Vital Product Data (VPD) test exercises the VPD interface using PCI configuration cycles and requires a proper bootcode to be programmed into the non-volatile memory. If no VPD data is present (for example, the VPD NVM area is all zeros), the test will first initialize the VPD data area with non-zero data before starting the test and restore the original data after the test completes.

Table 4-18. VPD Test (C9)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c9</code>
Engineering Mode Command	<code>1.NONE:>vpdtest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure in this test may indicate that the bootcode was not previously programmed correctly, that an error was encountered on the NVM interconnect, that the NVM device is defective, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	None

FIO Events Test (C11)

The FIO Events Test verifies that the event bits in the CPU's Fast IO interface are triggering correctly when particular chip events occur such as:

- a VPD request initiated by the host
- an expansion ROM request initiated by the host
- a timer event generated internally
- toggling any GPIO bits
- accessing NVM

Table 4-19. FIO Events Test (C11)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c11</code>
Engineering Mode Command	<code>1.NONE:>fioevtstest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure in this test may indicate that the test firmware could not be located, that an error was encountered on the NVM interconnect, that the NVM device is defective, or that the chip is defective.
Enabled by Default	Yes
Special Requirements	This test requires the presence of a special test firmware file inside the diagfw directory of the xdiag distribution.

Context Caching Test (C12)

The Context Caching Test verifies the context cache blocks with the following tests:

- locking mechanism
- setting different context cache attributes
- using different individual processor to perform GRC and FIO reads/writes/verify
- using different pairs of processors to perform GRC and FIO reads/writes/verify
- using all processors to do GRC and FIO reads/writes/verify

Table 4-20. Context Caching

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c12</code>
Engineering Mode Command	<code>1.NONE:>ctxtest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure may indicate that the test firmware could not be located, that an error was encountered on the context cache reads/writes/verify or other context cache logic, or that the chip is defective.
Enabled by Default	Yes (only for 5709)
Special Requirements	Requires the presence of a special test firmware file inside the diagfw directory of the xdiag distribution.

Data Integrity Test (C13)

The test verifies that individual hardware blocks have the ability to detect possible data corruption as packet information are passed between blocks. The test artificially injects errors at various blocks, making sure the downstream block can detect the error.

Table 4-21. Data Integrity (C13)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c13</code>
Engineering Mode Command	<code>1.NONE:>dataintgtest</code>
Return Codes	<code>0 - Test passed; -1 - N/A</code>
Possible Failure Causes	Failures are usually due to hardware problems. Keep in mind that the chip may still function properly with these failures since the feature is really an integrity check.
Enabled by Default	No
Special Requirements	Available only on 5709 (5709 and 57709)

Receive Side Scaling Hash Test (C14)

This test verifies the chip's ability to correctly hash TCP/IP information on packets to appropriately interrupt a particular host processor on a multiprocessor system. Various combinations of TCP/IP packet information are used to make sure the block handles them properly.

Table 4-22. Receive Side Scaling Hash Test (C14)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c14</code>
Engineering Mode Command	<code>1.NONE:>hashtest</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure in this test usually indicates a hardware problem. However, users need to make sure that a proper device is being put under this test.
Enabled by Default	No
Special Requirements	None

Status Block Test (C15)

The test checks for behavior of per-CPU status block.

Table 4-23. Status Block Test (C15)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T c15
Engineering Mode Command	1.NONE:>hcsbtest
Return Codes	0 - Test passed; -1 - N/A
Possible Failure Causes	Any failure in this test usually indicates a hardware problem.
Enabled by Default	No
Special Requirements	Available only on 5709 (5709 and 57709)

Memory Parity Test (C16)

This test performs memory parity check on the hardware, making sure the hardware can correctly detect parity problems on all on-chip memory.

Table 4-24. Memory Parity Test (C16)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T c16
Engineering Mode Command	1.NONE:>mparitytest
Return Codes	0 - Test passed; -1 - N/A
Possible Failure Causes	Any failure on this test usually means a hardware problem.
Enabled by Default	No
Special Requirements	Available only on 5709 (5709 and 57709)

CPU Multiplier Test (C17)

This test checks the internal multipliers used by some of the internal processors.

Table 4-25. CPU Multiplier Test (C17)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T c17
Engineering Mode Command	1.NONE:>multtest
Return Codes	0 - Test passed
Possible Failure Causes	Any failure in this test indicates a possible hardware problem.
Enabled by Default	No
Special Requirements	None

RXP CRC Test (C18)

This test checks the CRC engine used by one of the internal processors.

Table 4-26. RXP CRC Test (C18)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T c18
Engineering Mode Command	1.NONE:>rxpcrctest
Return Codes	0 - Test passed; -1 - N/A
Possible Failure Causes	Any failure in this test indicates a possible hardware problem.
Enabled by Default	No
Special Requirements	Available only on 5709 (5709 and 57709)

Tx BD Cache Test (C19)

This test fills the BD cache memory with some patterns and makes sure the searches are successful.

Table 4-27. Tx BD Cache Test (C19)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T c19
Engineering Mode Command	1.NONE:>tbdctest
Return Codes	0 - Test passed; -1 - N/A
Possible Failure Causes	Any failure in this test indicates a possible hardware problem.
Enabled by Default	No
Special Requirements	Available only on 5709 (5709 and 57709)

System Timer Accuracy Text (C20)

This test checks the accuracy of the timers at various granularities, making sure that the timers can fire off at the expected time.

Table 4-28.

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T c20
Engineering Mode Command	1.NONE:>timertest
Return Codes	0 - Test passed
Possible Failure Causes	Any failure in this test indicates a possible hardware problem.
Enabled by Default	No
Special Requirements	None

Boot Code State Test (C21)

This test checks if the boot code firmware module is still running and responding to host driver commands.

Table 4-29. Boot Code State Test (C21)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T c21</code>
Engineering Mode Command	<code>1.NONE:>bcstatetest</code>
Return Codes	<code>0 - Test passed; -1 - N/A</code>
Possible Failure Causes	Failure in this test means that the boot code is not responding to host driver commands.
Enabled by Default	Yes
Special Requirements	None

Group D - Ethernet Traffic Tests

MAC Loopback Test (D1)

The MAC Loopback test will enable MAC loopback mode in the QLogic Network controller and transmit 5000 Layer-2 packets of various sizes. As the packets are received back by xdiag they are checked for errors. Packets are returned through the MAC receive path and never reach the physical layer.

Table 4-30. MAC Loopback Test (D1)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T d1
Engineering Mode Command	1.NONE:>mac1b_test
Return Codes	0 - Test passed
Possible Failure Causes	A failure 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.

Physical Layer Loopback Test (D2)

The Physical Layer Loopback test will enable physical layer loopback mode in the QLogic Network controller and transmit 5000 Layer-2 packets of various sizes. As the packets are received back by xdiag they are checked for errors. Packets are returned through the physical layer receive path and never reach the wire.

For 5709 **S** part in NIC design, this test will run up to all four media (PHY0, PHY1, SERDES0, SERDES1) per port. There is an environment variable **XDIAG_PHY_MASK** that controls which media to be tested. That variable can be set in DOS by **set XDIAG_PHY_MASK=1111**.

In this example, all four media will be tested per port. The value of the five digit variable is decoded as follows in the corresponding left to right order:

Override control	alternate SERDES	alternate COPPER	default SERDES	default COPPER
------------------	------------------	------------------	----------------	----------------

When override control digit is **1**, the next four digits control which media to be tested. When override control digit is **0**, the next four digits will be ignored and the default strapped medium will be tested. As an example, to skip test of alternate **COPPER PHY**, the **XDIAG_PHY_MASK** should be set to **11011**. The **0** in the third digit, corresponds to alternate **COPPER**, will skip the test of alternate **COPPER PHY** and test the other three media.

For **MAC0**, the default **COPPER PHY** is **PHY0**, the default **SERDES** is **SERDES0**. The alternate **COPPER PHY** is **PHY1** and the alternate **SERDES** is **SERDES1**. For **MAC1**, the default **COPPER PHY** is **PHY1**, the default **SERDES** is **SERDES1**. The alternate **COPPER PHY** is **PHY0** and the alternate **SERDES** is **SERDES0**.

Table 4-31. Physical Layer Loopback Test (D2)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T d2
Engineering Mode Command	1.NONE:>phylb_test
Return Codes	0 - Test passed
Possible Failure Causes	A failure 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.

External Loopback Test (D3)

The External Loopback test will transmit a varying number and size of Layer-2 packets. As the packets are received back by xdiag they are checked for errors. The number of packets used for the test is based on the link speed being tested. For 10Base-T, only 1000 packets are used; for 100Base-T, 5,000; for gigabit traffic, 20,000 packets are used.

For 5709 **S** part in NIC design, this test will run up to all four media (**PHY0**, **PHY1**, **SERDES0**, **SERDES1**) per port. There is an environment variable **XDIAG_PHY_MASK** that controls which media to be tested. That variable can be set in DOS by **set XDIAG_PHY_MASK=11111**. In this example, all four media will be tested per port.

The value of the five digit variable is decoded as follows in the corresponding left to right order:

Override control	alternate SERDES	alternate COPPER	default SERDES	default COPPER
------------------	------------------	------------------	----------------	----------------

When override control digit is **1**, the next four digits control which media to be tested. When override control digit is **0**, the next four digits will be ignored and the default strapped medium will be tested. As an example, to skip test of alternate **COPPER PHY**, the **XDIAG_PHY_MASK** should be set to **11011**. The 0 in the third digit, corresponds to alternate **COPPER**, will skip the test of alternate **COPPER PHY** and test the other three media.

For **MAC0**, the default **COPPER PHY** is **PHY0**, the default **SERDES** is **SERDES0**. The alternate **COPPER PHY** is **PHY1** and the alternate **SERDES** is **SERDES1**. For **MAC1**, the default **COPPER PHY** is **PHY1**, the default **SERDES** is **SERDES1**. The alternate **COPPER PHY** is **PHY0** and the alternate **SERDES** is **SERDES0**.

Table 4-32. External Loopback Test (D3)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T d3</code> (all supported link speeds will be tested)
Engineering Mode Command	<code>1.NONE:>extlb_???m_test</code> (where ??? corresponds to the specific link speed to test: 10, 100, 1000, 2500)
Return Codes	0 - Test passed

Table 4-32. External Loopback Test (D3)

Feature	Description
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.

Large Sent Offload Test (D4)

The large sent offload test verifies the functionality of the QLogic Network's Large Send Offload support by enabling MAC loopback mode and transmitting large TCP packets. As the packets are received back by xdiag they are checked for proper segmentation (according to the selected MSS size) and any other errors.

Table 4-33. Large Sent Offload Test (D40)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T d4</code>
Engineering Mode Command	<code>1.NONE:>lsotest</code>
Return Codes	<code>0 - Test passed</code>
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
Enabled by Default	Yes
Special Requirements	The controller should not be connected to a network.

EMAC Statistics (D5)

The EMAC Statistics test verifies that the basic statistics information maintained by the chip is correct by enabling MAC loopback mode and sending Layer-2 packets of various sizes.

Table 4-34. EMAC Statistics (D5)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T d5
Engineering Mode Command	1.NONE:>stattest
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.

Remote Procedure Call Test (D6)

The remote procedure call test verifies the Receive Path Catch-up block by sending packets to different transmit chains. The packets will traverse the remote procedure call logic (though not the entire MAC block) and return to the receive buffers as received packets. This is another loopback path that is used by Layer-4 and Layer-5 traffic within the MAC block. As packets are received back by xdiag they are checked for errors.

Table 4-35. Remote Procedure Call Test (D6)

Feature	Description
Manufacturing Mode Command	C:\>xdiag -t abcd -T d6
Engineering Mode Command	1.NONE:>rpc_test
Return Codes	0 - Test passed

Table 4-35. Remote Procedure Call Test (D6)

Feature	Description
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.

Universal Management Port MAC Loopback Test (D7)

The universal management port MAC Loopback test verifies the universal management port block in MAC loopback mode. It exercises different functions in universal management port block. Including frame type pre-fetch register, ingress and egress path isolation, flow control, operation in fifo cut through mode, VLAN tag deletion and insertion, traffic test with different packet size.

Table 4-36. Universal Management Port MAC Loopback Test (D7)

Features	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T d7</code>
Engineering Mode Command	<code>1.NONE:>ump_mac1b_test</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure in this test, depends on the area of verification, may indicate the specific function is missing, MAC loopback failure or that the chip is defective.
Enabled by Default	No
Special Requirements	The controller should not be connected to a network.

Universal Management Port External Loopback Test (D8)

The universal management port MAC Loopback test verifies the universal management port block in external loopback mode. It exercises different functions in universal management port block. Including frame type pre-fetch register, ingress and egress path isolation, flow control, operation in fifo cut through mode, VLAN tag deletion and insertion, traffic test with different packet size.

Table 4-37. Universal Management Port External Loopback Test (D8)

Feature	Description
Manufacturing Mode Command	<code>C:\>xdiag -t abcd -T d8</code>
Engineering Mode Command	<code>1.NONE:>ump_extlb_test</code>
Return Codes	<code>0 - Test passed</code>
Possible Failure Causes	A failure in this test, depends on the area of verification, may indicate 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.

5 Xdiag Command Line Options and Return Codes

Command Line Options

The following command line options are supported in xdiag:

Table 5-1. Command Line Options

Options	Description
t <grps/tests>	Disable certain tests/groups. Default is to run all tests. For example, -t a1c2 will disable Register test (A1) and RBUF allocation test (C2). User can specify a list of tests to be disabled.
T <grps/tests>	Enable certain tests/groups. For example, -T a1c2 will enable Register test (A1) and RBUF allocation test (C2). User can specify a list of tests to be enabled. All tests are enabled by default except A4, A7, and D3.
C <device#>	Select device number to run. Default is all devices for diagnostic tests.
slave <device#>	Statically select device number to operate in slave mode. By slave mode, the device is simply echoing all received packets, and not operable for other purposes. Default is none. If 0 , the slave device number changes depending on the current device number. For example, if current device is 1 , the slave will be 3 for a four-port testing setup. The mappings are 1 => 3, 2 => 4, 3 => 1, 4 => 2. This mode is supported only for limited manufacturing environment.
I <iteration #>	Specify how many iterations of the selected tests need to run. Default is 1 .
pwd <password>	Specify the password in order to update firmware

Table 5-1. Command Line Options

Options	Description
fb <bc_image>	Specify the bin file to update bootcode.
Fmba <mba_image>	Specify the bin file to update MBA code.
Fipmi <ipmi_image>	Specify the bin file to update IPMI code.
Fump <ump_image>	Specify the bin file to update UMP code
fncsi <ump_image>	Specify the bin file to update NCSI code.
Fib <ib_image>	Specify the bin file to update iSCSI boot code.
Fib_ipv6 <ib_image>	Specify the bin file to update IPV6 iSCSI boot code.
Fib_ipv4n6 <ib_image>	Specify the bin file to update combo IPV4 & IPV6 iSCSI boot code.
Fibc	Program iSCSI configuration block, used with -fib -fib_ipv6 -fib_ipv4n6 <ib_image> only.
Fibp	Program iSCSI configuration software, used with -fib -fib_ipv6 -fib_ipv4n6 <ib_image> only.
F	Force to upgrade image without checking version.
Wol <1 0>	Enable (1) or disable (0) magic packet Wake on Lan (WoL).
Mba <1 0>	Enable (1) or disable (0) Multiple Boot Agent (MBA) (replaced -pxe).
Mfw <1 0>	Enable (1) or disable (0) management firmware (mfw).
Mbas <n>	Configure MBA/PXE link speed: auto (0), 10half (1), 10full(2), 100half(3), 100full(4), 1000full(6).
Mbap <n>	Configure MBA protocol: PXE (0), RPL (1), BOOTP(2), iSCSI boot(3).
Mbav <1 0>	Enable (1) or disable (0) MBA VLAN.
Mbavval <n>	Configure MBA VLAN value (applicable only when MBA VLAN is enabled, and the value must be less than 65536.
Rc <script>	Specify a script file to source after starting xdiag.
Checkrc	If the script file specified by rc <script> finishes with error, it will exit xdiag with error code returned from executing the script file.

Table 5-1. Command Line Options

Options	Description
Cof	Allow tests to continue on failure.
Fnm <nvm_image>	Specify nvm image file to program to NVM. Requires -pwd and -c or -idmatch .
Virtnvm <filename>	NVM image file read/view only operation in engineering mode. File can be accessed through NVM commands on dev 0 .
Viewnvmfile <filename>	NVM image file read/view only operation in command line mode.
Idmatch	Enable matching of VID, DID, SVID, and SSID from image file with device's IDs. Used with -fnvm <raw_image> only. This option applies to all devices.
Noidmatch	Disable matching of VID, DID, SVID, and SSID from image file with device's IDs. Used with -fnvm <raw_image> only.
Mac <mac_addr>	Specify primary MAC address in the form of xx:xx:xx:xx:xx:xx . Requires -pwd and -c .
iscsimac <mac_addr>	Specify iSCSI MAC address in the form of xx:xx:xx:xx:xx:xx . Requires -pwd and -c .
m	Display the MAC addresses (primary and iSCSI) and allow user to enter new MAC addresses, which have to be consecutive. Requires -pwd and -c .
ver	Display information on devices and xdiag version.
Log <logfile>	Log the tests' execution into the specified log file.
B06eng	Enter engineering mode. A detailed description of the commands available in engineering mode is available in "Engineering Mode Commands" on page 43 .
Sysop	Run a set of tests required by the QLogic SYSOP group. This test checks for proper soldering on pin connection.
Autom	Generate iSCSI and secondary MAC addresses. Work with -m in sysop mode.
Noinit	Start xdiag without pre-selecting or initializing any device.

Table 5-1. Command Line Options

Options	Description
Fmac <file>	Specify a text file that contains the MAC address range to be programmed. Requires -pwd and -c . Refer to “ macaddr.txt ” on page 105 for more information on the format of the macaddr.txt file.
Pnchk	Check part number against value input from keyboard, used with -sysop .
Cfgchk <file>	Check NVRAM configuration against description from a file. The contents of the file is similar to cfgchk.txt . Refer to “ cfgchk.txt ” on page 106 for more information on the format of the cfgchk.txt file.
Mkey <file>	Program manufacturing license key from the specified file (filename cannot start with -).
Ukey <file>	Program license key from the specified file (filename cannot start with -).
Chkkey <sm su smu>	Validate manufacturing license key (sm), upgrade license key (su), or both (smu).
Arg	Provide arbitrary arguments used by any customized scripts executed with the -rc command line option.
Nvmchk	Perform NVRAM image CRC checks.
Force_smbus	Override SMBus address, used with -fnvm .
Force_ibcfg	Override iSCSI boot cfg block, used with -fnvm .
Force_febcfg	Override FcoE boot cfg block, 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 .
raw	Program complete NVM image without skipping any field. When this option is specified, all -force_x and -skip_x will be overridden. Used with -fnvm .
@filename	A file that contains xdiag command line options. Xdiag will parse the file content. Example: xdiag@fn.txt
wpnvm	Enable HW write protection function on ATMEL D flash
slotchk	Primary and Secondary ASIC slot configuration check

Table 5-1. Command Line Options

Options	Description
smbus_addr	Change smbus address
mbabr	Configure MBA boot retry count.
noL1	Prevent PCIE L1-entry.
help	Display the command line options.

Return Codes

When xdiag exits and returns to a DOS command prompt the following return codes are supported:

Table 5-2. Return Codes

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	PCI-E link test fails
ERR_MSI_TEST	05	MSI test fails
ERR_BIST_TEST	06	Memory BIST test fails
ERR_NETLINK_TEST	07	Network link test fails
ERR_MEM_TXP_TEST	08	TXP scratchpad memory test fails
ERR_MEM_TPAT_TEST	09	TPAT scratchpad memory test fails
ERR_MEM_RXP_TEST	10	RXP scratchpad memory test fails
ERR_MEM_COM_TEST	11	COM scratchpad memory test fails
ERR_MEM_CP_TEST	12	CP scratchpad memory test fails
ERR_MEM_MCP_TEST	13	MCP scratchpad memory test fails
ERR_MEM_THBUF_TEST	14	THBUF memory test fails
ERR_MEM_TPBUF_TEST	15	TPBUF memory test fails

Table 5-2. Return Codes

Return Code	Value	Description
ERR_MEM_RBUF_TEST	16	Rx buffer memory test fails
ERR_MEM_RBUF_IND_TEST	17	Rx buffer (indirect) memory test fails
ERR_MEM_RBUF_LIST_TEST	18	Rx buffer cluster memory test fails
ERR_MEM_TSCH_LIST_TEST	19	TSCH memory test fails
ERR_MEM_CSCH_LIST_TEST	20	CSCH memory test fails
ERR_MEM_RV2P_TEST	21	RV2P scratchpad memory test fails
ERR_MEM_TBDC_TEST	22	TBDC memory test fails
ERR_MEM_RBDC_TEST	23	RBDC memory test fails
ERR_MEM_CTX_PGTBL_TEST	24	Context table memory test fails
ERR_MEM_CTX_TEST	25	Context memory test fails
ERR_CPU_TEST	26	CPU and DMA interface test fails
ERR_RBUF_ALLOC_TEST	27	Rx buffer allocation test fails
ERR_CAM_TEST	28	CAM test fails
ERR_TIMER_TEST	29	Timer test fails
ERR_TPATC_TEST	30	TPAT cracker test fails
ERR_FIO_REG_TEST	31	FIO register test fails
ERR_NVRAM_TEST	32	NVRAM test fails
ERR_RESET_TEST	33	Reset test fails
ERR_DMAE_TEST	34	DMA engine test fails
ERR_VPD_TEST	35	VPD test fails
ERR_HC_TEST	36	Host coalescing test fails
ERR_FIOEVT_TEST	37	FIO event 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

Table 5-2. Return Codes

Return Code	Value	Description
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 address 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/NCSI 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/NCSI 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

Table 5-2. Return Codes

Return Code	Value	Description
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_CTX_TEST	82	CTX test fails
ERR_MEM_CTX_CAM_TEST	83	CTX CAM memory test fails
ERR_MEM_CTX_MIRROR_TEST	84	CTX mirror memory test fails
ERR_MEM_CTX_USG_CNT_TEST	85	CTX usage count memory test fails
ERR_MEM_FTQ_TEST	86	FTQ memory test fails
ERR_MEM_MCP_EG_FIFO_TEST	87	MCP egress fifo memory test fails
ERR_MEM_MCP_ING_FIFO_TEST	88	MCP ingress fifo memory test fails
ERR_MEM_HBBUF_TEST	89	History buffer memory test fails
ERR_MSIX_TEST	90	MSIX test fails
ERR_DATINTG_TEST	91	Data Integrity test fails
ERR_HASH_TEST	92	RSS Hash test fails

Table 5-2. Return Codes

Return Code	Value	Description
ERR_HCSB_TEST	93	Status Block test fails
ERR_MPARITY_TEST	94	Parity mode test fails
ERR_MULT_TEST	95	CPU multiplier test fails
ERR_RXPCRC_TEST	96	RxP CRC test fails
ERR_TBDC_TEST	97	Tx BD Cache test fails
ERR_UMP_MACLB_TEST	98	UMP MAC loopback test fails
ERR_UMP_EXTLB_TEST	99	UMP external loopback test fails
ERR_ASIC_CONFIG	100	Primary and Secondary ASIC configuration error
ERR_SMBUS_CONFIG	101	Set smbus address failed
ERR_KEY_NA	102	License key not applicable (5716)
ERR_MED_UNDEF	103	Media not defined (5709/16)
ERR_CFG_MBA_BOOT_RETRY	104	Fail to configure boot retry count
ERR_BC_TEST	105	Boot code state test fails

6 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” on page 109](#) for additional information on TCL.) In addition to the core TCL commands such as **set**, **return**, **expr**, **if/else**, **while**, additional commands have been added to provide specific support for interacting with the QLogic Network 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 xdiag in engineering mode use the following DOS command:

```
C:\>xdiag -b06eng
```

Numeric values used in engineering mode commands can be represented in decimal form (such as 16) or in hexadecimal form (such as 0x10). Engineering mode 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 xdiag working directory. For example, to load the firmware on floppy drive **A:** while xdiag is running on the **C:\xdiag** directory, the proper command would be **cpu load a:/filename.bin**.

bits

The **bits** command is used to create numbers with specific bit positions either set or clear.

clear

Synopsis: **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**, setting the remaining bits to **1**, and then returns the value.

Returns: The hexadecimal representation of the bit position arguments set as **0**.

Example:

```
1.NONE:> bits clear 0 31
0x7fffffff
1.MCP:>
```

set

Synopsis: **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**, setting the remaining bits to **0**, and then returns the value.

Returns: The hexadecimal representation of the bit position arguments set as **1**.

Example:

```
1.NONE:> bits set 2 31
0x80000004
1.MCP:>
```

cpu

The **cpu** command is used to perform operations that are specific to a CPU in the QLogic Network controller. The command will be performed against the CPU displayed in the xdiag command prompt (see [“Command Prompt” on page 6](#) for details on the engineering mode command prompt format).

ctrl

Synopsis: `cpu ctrl <reg> <value>`

Description: With no register name **reg** {**mode**, **state**, **evmask**, **pc**, **inst**, **bp**} and no value specified, this command displays the CPU control register contents. When reg and value are typed, it will attempt to write value into the specified reg.

Returns: None

Example:

```
1.MCP:> cpu ctrl
mode: 80008880 state: 80000000 evmask: 00000500 pc:080050c0
inst: 0e0013d4 bp: 0000001d
1.MCP:>
1.MCP:> cpu ctrl mode 0x0000c880
mode: 00008880
1.MCP:>
```

halt

Synopsis: `cpu halt`

Description: This command halts the selected CPU.

Returns: None

Example:

```
1.MCP:> cpu halt
1.MCP:>
```

load

Synopsis: `cpu load <fw_filename>`

Description: This command loads firmware for the selected CPU. Make sure that forward slash is used to specify path for firmware file.

Returns: None

Example:

```
1.NONE:> cpu select com
1.COM:> cpu load diagfw/regress/dma.h
Writing Text section to address 0x8000000 offset 0x0c5c
Writing Rodata section to address 0x0 offset 0x0000
Writing Data section to address 0x8000c80 offset 0x0000
Writing SBSS section to address 0x0 offset 0x0000
Writing BSS section to address 0x0 offset 0x0000
1.COM:>
```

mem

Synopsis: `cpu mem <addr> <value1 [value2 ...]>`

Description: When **addr** and **value** are specified, this command writes **value** into selected CPU memory scratchpad at address **addr**. If only **addr** is specified, it displays 64 bytes of the selected CPU scratchpad memory starting at **addr**. If no parameter is specified, it displays 64 bytes of selected CPU scratchpad memory starting at **addr** plus the number of DWORDS written since the last write command.

Returns: None

Example:

```
1.MCP:> cpu mem 0
00000000: 00000011 ffffffff 1e3c78f0 1e3c78f0 .....<x..<x.
00000010: 55555555 00000000 00000000 0000000d UUUU.....
00000020: 62633120 312e332e 32000000 01030205 bc1.1.3.2.....
00000030: 00000000 10000003 00000000 0000000d .....
1.MCP:> cpu mem 4 0x22222222
1.MCP:> cpu mem
00000008: 1e3c78f0 1e3c78f0 55555555 00000000 .<x..<x.UUUU....
00000018: 00000000 0000000d 62633120 312e332e .....bc1.1.3.
00000028: 32000000 01030205 00000000 10000003 2.....
00000038: 00000000 0000000d 0000000d 3c020800 .....<...
1.MCP:> cpu mem 0
00000000: 00000011 22222222 1e3c78f0 1e3c78f0 ...."""".<x..<x.
00000010: 55555555 00000000 00000000 0000000d UUUU.....
00000020: 62633120 312e332e 32000000 01030205 bc1.1.3.2.....
00000030: 00000000 10000003 00000000 0000000d .....
1.MCP:>
```

run

Synopsis: `cpu run`

Description: This command starts the selected CPU if it has been halted.

Returns: None

Example: `1.MCP:> cpu run`
`1.MCP:>`

select

Synopsis: `cpu select <cpu_name>`

Description: This command selects the CPU to be manipulated. The available CPUs are **TXP**, **TPAT**, **RXP**, **COM**, **CP**, and **MCP**. **NONE** can also be the argument to select no particular CPU.

Returns: The selected CPU

Example: `1.NONE:> cpu select mcp`
`MCP`
`1.MCP:>`

gpr, step, disasm, fioaddr, uc fioaddr, grc, fill

Description: These commands are for QLogic internal use only.

delay

The **delay** command is used to introduce fixed time delays when executing scripts.

us

Synopsis: `delay us <count>`

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

Returns: None

Example: `delay us 5`

ms

Synopsis: `delay ms <count>`

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

Returns: None

Example: `delay ms 10`

device

Synopsis: `device [devNum [-no_display | hlb_on | hlb_off [poll]]]`

Description: Without an argument, this command displays a brief configuration summary for all of the devices that xdiag has detected. When **devNum** is provided, a different device will be selected and a brief configuration summary will be displayed. Optionally, a device can be into a **slave** mode by setting **hlb_on** flag. By that, the driver is loaded on that device, and it is operating as a host loopback device (echoing all Rx packets back to the wire). The slave cannot be selected as current device. For example, when a device is configured as slave device, xdiag cannot select that device using this command until it is brought out of the slave mode by setting the **hlb_off** flag. Only one device can be selected as slave at any given time. Currently, **hlb_on** allows loopback for L2 traffic only. When poll is specified with **hlb_on**, the device will operate in polling mode.
If the **-no_display** option is provided, the device summary page will not be displayed.

Returns: None

Example:

```
2.NONE:> device
C: Brd:Rv Bus PCI Spd Base IRQ MAC FmwVer Configuration
1:5706C:A1 04:01:00 PCI-64 66 0xDA00 5 0010180466FD 0.3.6 W,Mp,auto
2:5706C:A1 05:01:00 PCI-64 66 0xFA00 5 001018046742 0.3.6 W,Mp,auto
2.NONE:> device 1
C: Brd:Rv Bus PCI Spd Base IRQ MAC FmwVer Configuration
1:5706C:A1 04:01:00 PCI-64 66 0xDA00 5 0010180466FD 0.3.6 W,Mp,auto
1.NONE:> device 1 hlb_on
Select another device before putting this into host loopback mode.
1.NONE:> device 2 hlb_on
1.NONE:> device 2
Can't select the device; already selected as host loopback device.
1.NONE:> device 2 hlb_off
1.NONE:>
```

The configuration summary for device 1 in the example is interpreted as follows:

Column	Value	Meaning
1	1	Device #
2	5706C	Device ID and wire type (C for copper and S for serdes)
3	A1	Device revision number
4	04:01:00	PCI bus #:device #:function #
5	PCI-64	PCI bus running in 64bit width
6	66	PCI bus speed is 66Mhz
7	0xDA00	Byte[3:2] of 64 bit base IO address. Note that byte [7:4] and byte[1:0] are 0.
8	5	IRQ #
9	0010180466fD	Primary MAC address
10	0.3.6	Firmware revision
11	W, Mp, auto	Current device configuration W = WoL enabled M = Multiple boot agent (MBA) enabled Mp = MBA boot protocol is PXE Mr = MBA boot protocol is RPL Mb = MBA boot protocol is BOOTP Mi = MBA boot protocol is iSCSI boot Mn = MBA boot protocol is none auto = auto negotiation 10HD = 10 Mb half duplex (MBA configured) 10FD = 10 Mb full duplex (MBA configured) 100HD = 100 Mb half duplex (MBA configured) 100FD = 100 Mb full duplex (MBA configured) 1000HD = 1000 Mb half duplex (MBA configured) 1000FD = 1000 Mb full duplex (MBA configured)

disp64

Synopsis: **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.

Returns: The 64-bit value.

Example:

```
1.MCP:> disp64 signed 9 2
38654705666
1.MCP:>
1.MCP:> disp64 unsigned 15 4
64424509444
1.MCP:>
```

dma

The **dma** command allows explicit DMA operations to be performed from the command prompt. Note that only the CP and COM CPUs support the **dma** command. Before running this command, **cpu select**, **cpu load**, and **cpu run** must have been previously executed.

read

Synopsis: `dma read -vaddr <32bitaddr> -paddr <32bitadd_h> <32bitadd_l> -length <32bitvalue> -byteswap -pattern <pattern>`

Description: This command directs the chip firmware to perform DMA read operation from host memory to chip memory by providing the host address (either a virtual address `-vaddr <32bitaddr>` or a physical address `-paddr <32bitadd_h> <32bitadd_l>`), a byte count (`-length <32bitvalue>`), and byte swapping (`-byteswap`, little endian vs. big endian). Under MS-DOS, the physical and virtual addresses are always the same.

If data pattern (`-pattern <pattern>`) is specified, the host memory is initialized with different data pattern before starting the DMA. When the firmware completes the operation, a CRC is returned within the firmware logic. This allows the command to compare the computed CRC value with the firmware calculated CRC value to ensure data integrity through DMA. The DMA operation is chip-centric, for example, DMA read means that the chip is grabbing data from the host memory. The following are the possible values of *pattern*:

PatternType - Meaning

- 0 - no data initialization
- 1 - increment each byte value by 1, starting at 0.
- 2 - initialize each double word to 0xdeadbeef.
- 3 - initialize each double word to 0x55ff55ff

Returns: 0 if the operation completed successfully, non-zero otherwise.

Example:

```
1.MCP:> cpu select com
COM
1.COM:> cpu load diagfw/regress/dma.h
Writing Text section to address 0x8000000 offset 0x0c5c
Writing Rodata section to address 0x0 offset 0x0000
Writing Data section to address 0x8000c80 offset 0x0000
Writing SBSS section to address 0x0 offset 0x0000
Writing BSS section to address 0x0 offset 0x0000
1.COM:> cpu run
1.COM:> set addr [hmem alloc 512]
1.COM:> hmem show $addr 0x100
11d8c40:..... deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8c60:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8c80:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8ca0:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8cc0:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8ce0:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8d00:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8d20:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8d40:deadbeef deadbeef
1.COM:> dma read -vaddr $addr -paddr $addr -length 24 -pattern 3
vaddr=0x11d8c48
paddr=0x11d8c48
length=24
pattern=3
the crc generated on the host is 08x12bclbf2
going to bm0=011d8c48 bml=00000018 bm2=00000000
at address 0x0800003c written with word 0x011d8c48
at address 0x08000040 written with word 0x00000018
at address 0x08000044 written with word 0x00000000
got bm back
dma_req return value is =0 the cmd_stat fio reg =e
coming back bm0=80004 bml=12bclbf2 bm2=100018
return value from the dma_read =0
0
1.COM:>
```

write

Synopsis: `dma write -vaddr <32bitaddr> -paddr <32bitadd_h> <32bitadd_l> -length <32bitvalue> -byteswap -pattern <pattern type>`

Description: This command directs the chip firmware to perform DMA write operation from chip memory to host memory by providing the host address (either a virtual address `-vaddr <32bitaddr>` or a physical address `-paddr <32bitadd_h> <32bitadd_l>`), byte count (`-length <32bitvalue>`), byte swapping (`-byteswap`, little endian vs. big endian), and data pattern (`-pattern <pattern type>`). Under MS-DOS, the physical and virtual addresses are always the same.

When the firmware completes the operation, a CRC is returned within the firmware logic. This allows the command to compare the computed CRC value with the firmware calculated CRC value to ensure data integrity through DMA. The DMA operation is chip-centric, for example, DMA write means that the chip is writing data to the host memory. The following are the possible values of *pattern type*:

PatternType - Meaning

- 0 - no data initialization
- 1 - increment each byte value by 1, starting at 0.
- 2 - initialize each double word to 0xdeadbeef.
- 3 - initialize each double word to 0x55ff55ff.

Returns: 0 if the operation completed successfully, non-zero otherwise.

Example:

```
1.MCP:> cpu select com
COM
1.COM:> cpu load diagfw/regress/dma.h
Writing Text section to address 0x8000000 offset 0x0c5c
Writing Rodata section to address 0x0 offset 0x0000
Writing Data section to address 0x8000c80 offset 0x0000
Writing SBSS section to address 0x0 offset 0x0000
Writing BSS section to address 0x0 offset 0x0000
1.COM:> cpu run
1.COM:> set addr [hmem alloc 512]
1.COM:> hmem show $addr 0x100
11d8c40:..... deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8c60:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8c80:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8ca0:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8cc0:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8ce0:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8d00:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8d20:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8d40:deadbeef deadbeef
1.COM:> dma write -vaddr $addr -paddr $addr -length 24 -pattern 1
bm0=11d8c48 bml=18 bm2=1 bm3=0
at address 0x0800003c written with word 0x011d8c48
at address 0x08000040 written with word 0x00000018
at address 0x08000044 written with word 0x00000001
at address 0x08000048 written with word 0x00000000
at address 0x0800004c written with word 0x00000001
got bm back
dma_req=0 cmd_stat=c
dma_stat=4 bml=8295a696 bm2=50000
the status coming back is okay!
the host crc is = 08x8295a696
return value from the dma_write = 0
0
1.COM:> hmem show $addr 0x100
11d8c40:..... 00010203 04050607 08090a0b 0c0d0e0f 10111213 14151617
11d8c60:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8c80:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8ca0:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8cc0:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8ce0:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8d00:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8d20:deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef deadbeef
11d8d40:deadbeef deadbeef
```

driver

The xdiag utility includes an internal network driver that is used for sending and receiving Ethernet traffic. This driver operates similarly to an OS-based driver in that it can be loaded and unloaded on demand. Directly modifying QLogic Network registers while the driver is loaded may result in unpredictable operation of the controller.

intr

Synopsis: `driver intr`

Description: This command sets the driver to run in hardware interrupt event driven mode.

Returns: Returns **1** if the operation completed successfully.

Example:

```
1.MCP:> driver intr
1
1.MCP:
```

intrcnt

Synopsis: `driver intrcnt`

Description: This command returns the number of hardware interrupt events that have occurred.

Returns: The number of hardware interrupt events.

Example:

```
1.NONE:>
1.NONE:> driver intrcnt
0
1.NONE:> driver load
RUN
1.NONE:> driver intrcnt
1
1.NONE:>
```

load

Synopsis: `driver load`

Description: This command loads and starts the driver.

Returns: The new driver state (**FOUND**, **SETUP**, **INIT**, **RUN**, **SHUTDOWN**, **PAUSED**, or **WOL**).

Example:

```
1.MCP:> driver load
RUN
1.MCP:>
```

poll

Synopsis: `driver poll`

Description: This command sets the driver to run in hardware polling event mode.

Returns: None

Example:

```
1.MCP:> driver poll
1.MCP:>
```

unload

Synopsis: `driver unload`

Description: This command stops the driver, unloads it, and frees the memory.

Returns: The new driver state (**FOUND**, **SETUP**, **INIT**, **RUN**, **SHUTDOWN**, **PAUSED**, or **WOL**).

Example:

```
1.MCP:> driver unload
SETUP
1.MCP:>
```

init, run, pause, shutdown, wol, reset, rxmask, txbatch, submit, stat, statslave

Synopsis: `init, run, pause, unpause, shutdown, wol, reset, rxmask, offload, txbatch, submit, stat, statslave <device number>`

Description: These commands are for QLogic internal use only.

exit

Synopsis: `exit`

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

Returns: None

Example: 1.MCP:> `exit`
C:\>

ftq

The QLogic Network controller uses Flow-Through Queues (or FTQs) to connect various state machines within the controller. The **ftq** commands allow an engineer to look at the state of the various FTQs.

Synopsis: `ftq dump`

Description: This command displays the information of all the flow-through queues inside the controller which helps identify if a particular block within the controller is hung or not responding.

Returns: None

Example: 1.MCP:> `ftq dump`

FTQ	Command	Control	Depth_Now	Max_Depth	Valid_Cnt
rlup :	0	8000	0	8	0
rxp :	0	100000	0	100	0
rxpc :	0	8000	0	8	0
rv2pp :	0	10000	0	10	0
rv2pm :	0	20000	0	20	0
rv2pt :	0	20000	0	20	0
rdma :	0	10000	0	10	0
tsch :	0	20000	0	20	0
tbdr :	0	4000	0	4	0
txp :	0	10000	0	10	0
tdma :	0	10000	0	10	0
tpat :	0	10000	0	10	0
tas :	0	10000	0	10	0
comx :	0	10000	0	10	0
comt :	0	20000	0	20	0
com :	0	10000	0	10	0

1.MCP :>

gpio

The **gpio** command allows the user to directly manipulate the state of the GPIOs supported on the QLogic Network 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. Systems using the BCM5708 should not manipulate GPIO2.

read

Synopsis: `gpio read <pin>`

Description: This command queries a certain GPIO *pin*. The pin can be 0 – 7.

Returns: The current value of the GPIO (0 or 1).

Example:

```
1.MCP:> gpio read 1
1
1.MCP:>
```

write

Synopsis: `gpio write <pin> <value>`

Description: This command writes value to a certain GPIO pin. pin can be 0 – 7.

Returns: Returns **1** if the operation completed successfully.

Example:

```
1.MCP:> gpio write 1 0
1
1.MCP:>
```

help

Synopsis: help

Description: Displays all high level commands.

Returns: None

Example:

```
1.MCP:> help
Cmd group-tcl commands
Type "help Cmd_Name" to get detailed information about a command
hex          Returns value in hexadecimal form
bits         Creates a 32-bit value.
value        Display a value in hex, bin, and dec.
device       Select and/or display device
reg          Access chip registers
sram         Access on-chip memory/scratchpad
nvm          Access non-volatile memory.
mii          Access PHY registers
pci          Scan/select PCI devices, or initialize current pci
             device's config space.
pcicfg       Access register via PCI config cycle.
gpio         Creates a 32-bit value.
version      Returns the version string of diagnostic.
serialport   Serial port I/O.
cpu          Debugger for on-chip MIPS processor
l2spec       Layer 2 specification.
l2pkt        Generate and manipulate layer 2 packets.
what         Display detail information.
stats        Manipulates device/driver statistics.
watch        Executes & displays TCL script(s) result(s).
yield        Allow background process to run.
driver       Manipulate the driver state and settings.
hmem         Display host memory
delay        Force Tcl to wait.
dma          Perform DMA access
ftq          Perform FTQ access
log          Performs command/output logging
qhack        Quick hack to try out your function.
keyhit       Check for keyboard activity, usually for script use.
xfer         Upload/download files via serial port.
mcast        Multicast address import/export to driver.
wol          Configure Wake On-Lan setting.
disp64       Present a string form of a 64-bit value in decimal.
rbuf         Stress test the rbuf.
nictest      Run NIC functional tests.
ioport       Access host I/O ports
license      Manipulate license keys.
ipmi         IPMI-PT manipulation.
1.MCP:>
```

hex

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

hex16

Synopsis: **hex16** <value>

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

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

Example:

```
1.MCP:> hex16 20
0014
1.MCP:>
```

hex32

Synopsis: **hex32** <value>

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

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

Example:

```
1.MCP:> hex32 20
00000014
1.MCP:>
```

hmem

The **hmem** command provides direct access to host memory.

alloc

Synopsis: **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. The **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**. In the Linux environment, **hmem alloc** allocates memory from user space.

Returns: The address of the allocated memory.

Example: 1.MCP:> hmem alloc 100 My100bytes
0x15bae58
1.MCP:>

fill

Synopsis: **hmem fill** <low32addr> [<high32addr>] <byte_count> <pattern>

Description: This command fills host memory with **byte_count** of bytes at specified address. The **byte_count** value must be a multiple of 4. 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.

Returns: None

Example: 1.MCP:> hmem fill 0x6000 20 0x12121212
1.MCP:>

free

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

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

Returns: None

Example: 1.MCP:> hmem fill 0x6000 20 0x12121212
1.MCP:>

inuse

Synopsis: **hmem inuse**

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

Returns: None

Example: 1.MCP:> hmem inuse
(112,015bae54/00000000) 015bae54/00000000 100 bytes(Virt)
My100bytes
Inside () is only for debugging. The next value is the virtual
address, next oen being physical (in DOS virtual == physical).
1.MCP:>

paddr

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

Description: This command returns a list of high and low 32-bits of the physical address of a buffer with virtual address (*high32addr* << 32 + *low32addr*) and is only applicable when memory is allocated using the **hmem palloc** command.

Returns: The physical address of the buffer.

Example: 1.MCP:> hmem palloc 200 Mybuf
0x15baf10
1.MCP:>paddr 0x15baf10
0x0 0x15baf10
1.MCP:>

palloc

Synopsis: `hmem palloc <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. The **description_string** is limited to 30 characters with no space in between. In the Linux environment, **hmem palloc** allocates memory from kernel space.

Returns: The virtual address of the buffer.

Example:

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

read

Synopsis: `hmem read <low32addr> [<high32addr>]`

Description: This command reads the **DWORD** from the address specified by (`highaddr32 << 32 + low32addr`) in host memory.

Returns: The value read from the specified host memory address.

Example:

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

show

Synopsis: `hmem show <low32addr> [<high32addr>] <byte_count>`

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

Returns: None

Example:

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

write

Synopsis: `hmem write <low32addr> [<high32addr>] <value>`

Description: This command writes the **DWORD** specified by the **value** to the host memory address specified by $(\text{highaddr32} \ll 32 + \text{low32addr})$.

Returns: None

Example:

```
1.MCP:> hmem write 0 0x34343434
1.MCP:>
```

ioport

The **ioport** command provides direct access to system I/O address space.

read

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

Description: This command reads a **BYTE**, **WORD**, or **DWORD** from the port address **addr**.

Returns: The **BYTE**, **WORD**, or **DWORD** value read.

Example: 1.MCP:> ioport read byte 0xcf8
0xff
1.MCP:>

write

Synopsis: **ioport read byte|word|dword** <addr> <value>

Description: This command writes the **BYTE**, **WORD**, or **DWORD** value to the port address **addr**.

Returns: None

Example: 1.MCP:> ioport write byte 0xcf8 0x55
1.MCP:>

ipmi

The **ipmi** command is only applicable to systems that are using optional IPMI firmware for the QLogic Network controller.

cfg

Synopsis: `ipmi cfg <option=choice>`

Description: This command implements both an interactive and a scripted mode of operation that allows a user to view/modify the QLogic Network controller IPMI configuration. When invoked without any options, the user is presented with a listing of the current IPMI configuration and is given the opportunity to change the configuration and save the results. When invoked with options, only that IPMI configuration setting is modified. If the IPMI configuration image does not already exist in NVRAM one will be created.

Returns: None

Example:

```
1.NONE:> ipmi cfg
1. Link speed {10/100(0),10(1),100(2),1000(3),all(4)} : 10/100
2. Link duplexity {Full(0), Half(1)} : Full
3. Link pause capability {Enable(0), Disable(1)} : Enabled
4. Link Auto-neg {Enable(0), Disable(1)} : Enabled
Choice (<option>=<value>, save, cancel) : cancel
Config not saved.
1.NONE:> ipmi cfg 1=1
Saving config...
Programming from: 0x00011534 to 0x0001175c
Config saved.
1.NONE:>
```

crc

Synopsis: `ipmi crc [-fix]`

Description: Checks the integrity of the IPMI configuration table in NVRAM. The optional argument **-fix** is used to force the CRC check correction. Note that this CRC is used by the IPMI-PT firmware to check the integrity of this table. Should the CRC check fail, the IPMI-PT firmware will not be properly loaded. If the IPMI configuration table does not already exist in NVRAM, one will be created.

Returns: The CRC of the IPMI configuration table.

Example:

```
1.NONE:> ipmi crc
OK 0xd797fd56
1.NONE:>
```

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: Returns **0** when a key has been pressed.

Example:

```
keyhit
keyhit yn
```

l2pkt

Description: This command is for QLogic internal use only.

l2spec

Description: This command is for QLogic internal use only.

license

The **license** command is used for installing and verifying QLogic licensing support.

secret

Synopsis: **license secret** *<filename>*

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

Returns: None

Example: 1.NONE:> **license secret oem_ss.bin**
1.NONE:>

storekey

Synopsis: **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.

Returns: None

Example: 1.NONE:> **license storekey -m oem_mkey.bin**
1.NONE:>

rmkey

Synopsis: `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.

Returns: None

Example: 1.NONE:> `license rmkey -m`
1.NONE:>

display

Synopsis: `license display -m|-u|-c`

Description: This command displays the license content of either the manufacturing key (-m), the upgrade key (-u), or the currently effective key (-c). If the key does not exist an error will be displayed.

Returns: None

Example: 1.NONE:> `license display -m`

0. Max TOE conn (65535 for unlimited)	: 1
1. Max RDMA conn (65535 for unlimited)	: 0
2. Max iSCSI initiator conn (65535 for unlimited)	: 0
3. Max iSCSI target conn (65535 for unlimited)	: 0
4. Serial number	: 0
5. Expiration date (yyyymmdd or never)	: 20051231
6. 2.5G capability (1 for enable)	: 0

1.NONE:> `license display -u`
Key is either corrupted or does not exist.
1.NONE:>

chkkey

Synopsis: `license chkkey [-s|-m|-u|-c]`

Description: This command checks the integrity of the license keys as well as the shared secret inside the NVRAM (equivalent to `-s`, `-m`, and `-u`). If an optional switch is provided, only the shared secret (`-s`), the manufacturing key (`-m`), the upgrade key (`-u`), or the currently effect key (`-c`, located on scratch pad rather than NVRAM) 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: Returns **OK** if the secret/key is valid, otherwise nothing.

Example:

```
1.NONE:> license chkkey -m
OK
1.NONE:> license chkkey
Shared secret      :OK
Manuf key          :OK
Upgrade key        :Invalid/Nonexistent
1.NONE:>
```

dump

Synopsis: `license dump -m|-u|-c`

Description: This command dumps the details of a license key in a form of a **Tcl** list object. Each object consists of two items, the name of the license attribute and the value for that attribute. This is normally used in a script to capture the license details for verification. If the license does not exist or is invalid, a string **Key is either corrupted or does not exist** will be returned instead of the expected Tcl list.

Returns: A **Tcl** list object with license attributes if the key is valid, otherwise an error string is returned.

Example:

```
1.NONE:> set x [license dump -m]
{toe 65535} {rdma 0} {iscsi_i 65535} {iscsi_t 0} {sn 0} {exp_date never} {2g5 0}
1.NONE:> set x
{toe 65535} {rdma 0} {iscsi_i 65535} {iscsi_t 0} {sn 0} {exp_date never} {2g5 0}
1.NONE:> set x [lic dump -u]
Key is either corrupted or does not exist.
1.NONE:> set x
Key is either corrupted or does not exist.
1.NONE:>
```

log

The **log** command is used to log commands and results to a file.

open

Synopsis: **log open <logfile>**

Description: This command opens a log file and starts logging all displayed items to the log file.

Returns: None

Example:

```
1.NONE:> set x [license dump -m]
{toe 65535} {rdma 0} {iscsi_i 65535} {iscsi_t 0} {sn 0} {exp_date
never} {2g5 0}
1.NONE:> set x
{toe 65535} {rdma 0} {iscsi_i 65535} {iscsi_t 0} {sn 0} {exp_date
never} {2g5 0}
1.NONE:> set x [lic dump -u]
Key is either corrupted or does not exist.
1.NONE:> set x
Key is either corrupted or does not exist.
1.NONE:>
```

close

Synopsis: **log close**

Description: This command closes and saves the log file.

Returns: None

Example:

```
1.MCP:> log close
1.MCP:>
```

mcast

Description: This command is for QLogic internal use only.

mii

The **mii** command allows access to the PHY registers through MII.

[slave <device number>] [copper|serdes] read

Synopsis: **mii read** <addr>

Description: This command reads data from a PHY register at **addr**. The slave option applies the command to the slave device if it has been configured. The **copper** or **serdes** option is used for dual port device such as 5709. It overrides the current medium to specify which medium to access.

Returns: The 16-bit value read.

Example:

```
1.NONE:> mii read 0
0x1140
1.NONE:>
```

[slave <device number>] [copper|serdes] show

Synopsis: **mii show** <begin_addr> [cnt]

Description: This command performs the same operation as the **mii read** command except that it formats the output in a more user-friendly manner. The optional argument **cnt** specifies the number of registers to display. The slave option applies the command to the slave device if it has been configured. The **copper** or **serdes** option is used for dual port device such as 5709. It overrides the current medium to specify which medium to access.

Returns: None

Example:

```
1.MCP:> mii show 0
00: 1000
1.MCP:> mii show 0 0x10
00: 1000 796d 0020 6151 0de1 4de1 0007 2001
08: 0000 0300 0000 0000 0000 0000 0000 3000
```

[slave <device number>] [copper|serdes] write

Synopsis: `mii write <addr> <value>`

Description: This command writes the value into the PHY register at address **addr**. The **slave** option applies the command to the slave device if it has been configured. The **copper** or **serdes** option is used for dual port device such as 5709. It overrides the current medium to specify which medium to access.

Returns: Returns **1** for success.

Example:

```
1.NONE:> mii write 0x12 0x0123
1
1.NONE:>
```

rread

Synopsis: `mii rread <addr>`

Description: This command writes the value into the PHY register at address **addr**. The **slave** option applies the command to the slave device if it has been configured. The **copper** or **serdes** option is used for dual port device such as 5709. It overrides the current medium to specify which medium to access.

Returns: Returns **1** for success.

Example:

```
1.NONE:> mii write 0x12 0x0123
1
1.NONE:>
```

rshow

Synopsis: `mii rshow <begin_addr> [cnt]`

Description: This command performs the same operation as the **mii rread** command except that it formats the output in a more user-friendly manner. The optional argument **cnt** specifies the number of registers to display. It works only for remote PHY configuration.

Returns: None

Example:

```
1.MCP:> mii rshow 0
00: 1000
1.MCP:> mii rshow 0 0x10
00: 1000 796d 0020 6151 0de1 4de1 0007 2001
08: 0000 0300 0000 0000 0000 0000 0000 3000
```


rewrite

Synopsis: `mii rewrite <addr> <value>`

Description: This command writes the value into the remote PHY register at address **addr**. It works only for remote PHY configuration.

Returns: The 16-bit value read.

Example:

```
1.NONE:> mii write 0x12 0x0123
1
1.NONE:>
```

nictest

Synopsis: `nictest [[-t <groups>] [-T <groups>] [-i <num>] [-cof] [-log <filename>]]`

Description: This command performs the functional testing of the chip. The various test groups are described in [“Xdiag Test List” on page 8](#). The optional **-T** *<groups>* parameter specifies which groups to include in the test while the **-t** *<groups>* parameter specifies which groups to exclude from the test. The **-i** *<num>* parameter specifies the number of iterations of **nictest** to run (with **0** indicating that the tests should run indefinitely) and the **-cof** parameter specifies that the testing should continue after an error is encountered. The **-log** *<filename>* specifies a log file where the test execution results will be written.

Returns: None

Example:

```
1.NONE:> nictest -t ab
Tests are running iteration 1
Group C. Block Tests
C01. CPUs' logic and DMA interface Test.....: passed
C02. RBUF Allocation Test.....: passed
C03. CAM Access Test.....: passed
C04. TPAT Cracker Test.....: passed
C05. FIO Register/Context Test.....: passed
C06. NVM Access and Reset-Corruption Test.....: passed
C07. Core-Reset Integrity Test.....: passed
C08. DMA Engine Test.....: passed
Group D. Data Tests
D01. MAC Loopback Test.....: passed
D02. PHY Loopback Test.....: passed
D04. LSO Test.....: passed
D05. EMAC Statistics Test.....: passed
1.COM:>
```

nvm

The **nvm** command provides access to the QLogic Network NVRAM.

cfg

Synopsis: **nvm cfg** [*<option=choice>*] [*-otherfn*]

Description: This command implements both an interactive and a scripted mode of operation that allows a user to view and modify the QLogic Network 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. When invoked with **-otherfn** option, **nvm cfg** command accesses configuration information of the other port on a dual port device. The **nvm cfg** command on **dev 0** provides read only access to NVM image file specified by **-virtnvm** option. This command cannot be invoked when the driver is loaded (**driver load** command). This command only displays and allows applicable configuration parameters depending on the current device. For example, EPB related parameters are shown only on 5708 devices. Port swapping parameter is only shown on 5709/16 devices.

Refer to the QLogic Network NVRAM Configuration Application Note for additional details on each of the configurable options below and QLogic's recommendations for settings. The example below is not by any means the complete list of parameters. Rather, it is served only as an example. Refer to ["config.txt" on page 101](#) for a list of currently supported parameters.

Returns: None

Example:

```
1.NONE:> nvm cfg
BCM95706A0...
1: MAC Address                : 00:10:18:04:21:1c
2: Power Dissipated (D3:D2:D1:D0) : 10:0:0:100
3: Power Consumed (D3:D2:D1:D0)  : 10:0:0:100
4: Vendor ID                  : 14E4
5: Vendor Device ID           : 164A
6: Subsystem Vendor ID        : 14E4
7: Subsystem Device ID        : 164A
...
...
43: UMP/NCSI PHY Timing (5708 B0 and after) {Disable(0),
Enable(1)} : Disabled
44: SMBus Address {one byte even hex value (i.e.bit0=0)}: 0x00
46: Serdes TxActrl3 value (pre-emphasis, idriver, etc;
5708 fiber only) { 16-bit, or Disabled(0) } : 0000
47: UMP Echo mode { Disabled(0), Enable(1) } : Disabled
48: Allow gigabit link on Vaux {Disabled(0), Enable(1)} : Disabled
...
...
Choice (<option>=<value>, save, cancel): cancel
Config not saved.
1.NONE:> nvm cfg 21=1
Saving config...
Config saved.
```

crc

Synopsis: `nvm crc [-fix <entry>]`

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

Returns: None

Example:

```

1.NONE:> nvm crc
Region Start Length Content Computed Result
BOOTSTRAP 0x00000000 0x00000014 0x3A1090F5 0x3A1090F5 OK
DIRECTORY 0x00000014 0x000000EC 0xA24171BA 0xA24171BA OK
HW_INFO 0x00000100 0x00000100 0x149B145C 0x149B145C OK
FEAT_INFO 0x00000200 0x00000100 0x174951E7 0x174951E7 OK
MKEY_INFO 0x00000400 0x00000100 0x00000000 0xF15963A6 Mismatch
BC1 0x00000800 0x00000C30 0x61D84A1D 0x61D84A1D OK
BC2 0x00002000 0x00001F98 0xA985B8CF 0xA985B8CF OK
MBA 0x0000C400 0x00010840 0x27794857 0x27794857 OK
1.NONE:> nvm crc -fix mkey_info
Fixed 0xf15963a6

```

dir

Synopsis: `nvm dir [-delete <entry>]`

Description: Displays a listing of the firmware programmed in NVRAM, optionally deleting a particular image. The `nvm dir` command on **dev 0** provides read only access to NVM image file specified by `-virtnvm` option.

Returns: None

Example:

```

1.NONE:> nvm dir
Image SRAM Addr  NVM Offset Byte      Cnt  CPU  Version
BC1  0x08006C00 0x00000800 0x00000C30 MCP  bc1  0.3.6
BC2  0x08000000 0x00002000 0x00001F98 MCP  bc2  0.3.6
MBA  0x00000000 0x0000C400 0x00010840 HOST  1.1.2
1.NONE:> nvm dir -delete MBA
1.NONE:>

```

dump

Synopsis: `nvm dump [-l] [-a | -s] [<offset> <length>] <filename>`

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. 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.

Returns: None

Example:

```
1.NONE:> nvm dump a:/nvm.bin
1.NONE:>
```

fill

Synopsis: `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.

Returns: None

Example:

```
1.NONE:> nvm fill 0 0x100 0xdeadbeef
Programming from: 0x00000000 to 0x00000100
done
1.NONE:> nvm fill 0 0x10 addr
Programming from: 0x00000000 to 0x00000010
Done
1.NONE:>
```

prg

Synopsis: `nvm prg [-l | -mem | -skip_lic | -force_lic | -force_mac | -force_smbus | -force_ibcfg | -force_febcfg | -skip_ec | -skip_sn | -raw | -no_chip_chk | -skip_options <option_list> <offset> [<filename> | <hostaddr>] [<size>]`

Description: This command programs NVRAM starting at the 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. By default, when programming the NVRAM image from offset **0**, the MAC addresses and the backup MAC addresses (used by some OEM) of the boards are preserved. The option **-force_mac** allows user to override this default. Similarly, the **-force_smbus** allows users to program the **SMBus** address that is in the file (or host memory), while the this address value is preserved by default. The option **-force_ibcfg** allows user to override iSCSI boot cfg block. The option **-force_febcfg** allows user to override Fibre Channel over Ethernet boot cfg block. The option **-skip_ec** and option **-skip_sn** allow user to preserve EC or SN field in VPD block. Depending on OEMs, the default behavior of license block programming is different. The option of **-skip_lic** (or **-force_lic**) allows users to override default to skip (or always program) the license key blocks when programming the NVRAM image from offset 0. Usually, the device ID is checked against the image to make sure a proper image is being program. If **-no_chip_chk** is provided, this check will be skipped. The option **-raw** allows user to program a complete NVM image without skipping any field. After the programming is completed, the controller needs to be reset to reflect the new NVRAM contents.

The option **-skip_options** is used to preserve a list of specific configuration parameters. The parameter list is a string composed of a list of numbers, delimited by commas (for example **1,9,55**). The numbers correspond to the **nvm cfg** options (see “[config.txt](#)” on page 101). The function of this option basically captures the configuration values prior to programming, programs the NVRAM based on other options specified, and re-applies the captured values back. Thus, it can override the **-raw** option to preserve whatever configuration parameters.

Although the backup MAC address fields are preserved as a default behavior, xdiag has an exception to zero them out if the fields have all F's. This helps addressing the manufacturing issue, preventing all F's in these fields when the device leaves the manufacturing floor.

Returns: None

Example:

```
1.NONE:> nvm prg 0 nvmmimage.bin
1.NONE:> set options 1,9,55; nvm prg -raw -skip_options $options
0 nvmmimage.bin
1.NONE:>
```

read

Synopsis: `nvm read <addr>`

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

Returns: The value read.

Example:

```
1.NONE:> nvm read 0
0x669955aa
1.NONC:>
```

show

Synopsis: `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.

Returns: None

Example:

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

write

Synopsis: `nvm write <addr> <value1 [value2...]>`

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

Returns: None

Example:

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

upgrade

Synopsis: `nvm upgrade [-F] -bc|-mba|-ipmi|-ump|-ib|-feb|-ncsi <filename>|ccm`

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

OPTION	NAME OF FIRMWARE OR BOOT IMAGE
bc	boot code
mba [-p <pfm_val>] [-d]	MBA (PXE) code
ipmi	IPMI code
ump	UMP firmware
ncsi	NCSI firmware
ib [-c -p -cp]	iSCSI boot driver
ib_ipv6 [-c -p -cp]	IPV6 iSCSI boot driver
ib_ipv4n6 [-c -p -cp]	IPV4/IPV6 iSCSI boot driver
feb [-c -p -cp]	Fibre Channel over Ethernet boot driver
ccm	CCM image

Note that all other parameters are for QLogic internal use only. For iSCSI boot image, there are addition options can be specified. They are **-c**, **-p**, and **-cp**. The latest iSCSI boot file actually contains multiple images, including iSCSI boot driver itself, iSCSI configuration, and iSCSI configuration utility program. By default, the driver portion is always programmed (subject to version check or **-F** option). The configuration portion is programmed only when such image does not exist on the current device, but this default can be overridden by **-c** option. The utility program is normally not programmed into the NVRAM unless **-p** option is specified. The **-cp** option simply forces to program both the configuration and the utility program. After the upgrade completes, the controller needs to be reset in order to load and run the new firmware. Same for Fibre Channel over Ethernet boot image.

For MBA image, the **-p** option updates the MBA image so to tell MBA to use the embedded PFN information. The **-d** option bypasses the device check, allowing MBA images of other devices to be programmed. All these options are used by engineers for debugging only.

Returns: None

Example:

```
1.NONE:> nvm upgrade -bc1 bc1.bin
get_file_size: file: bc1.bin; size=3192
read_bin_file: file read 3192 bytes; buf=00b43e30
BC Version: bc1 0.3.6 (0x00030605)
update bootstrap: start=800; byte_cnt=3120; crc=0x3a1090f5
The upgrade won't take effect until the next chip reset
1.NONE:> nvm upgrade -bc2 bc2.bin
get_file_size: file: bc2.bin; size=8160
read_bin_file: file read 8160 bytes; buf=00b49a30
The upgrade won't take effect until the next chip reset
1.NONE:> nvm upgrade -mba b06mmmba.nic
get_file_size: file: b06mmmba.nic; size=67644
read_bin_file: file read 67644 bytes; buf=00b5320c
```

rescfg

Synopsis: `nvm rescfg [<option=choice>]`

Description: This command implements both an interactive and a scripted mode of operation that allows a user to view and modify the QLogic Network controller resource configuration. When invoked without any options, the user is presented with a listing of the current NVRAM resource configuration and is given the opportunity to change the configuration and save the results. When invoked with options, only that NVRAM resource configuration setting is modified. This command cannot be invoked when the driver is loaded (**driver load** command).

Refer to the QLogic Network NVRAM Configuration Application Note for additional details on each of the configurable options below and QLogic's recommendations for settings. The example below is not by any means the complete list of parameters. Rather, it is served only as an example. ["config.txt" on page 101](#) has a list of currently supported parameters.

Returns: None

Example:

```
1.NONE:> nvm rescfg
BCM95706A0...
1: ISCSI pending tasks : 20
Choice (<option>=<value>, save, cancel): cancel
Config not saved.
1.NONE:> nvm rescfg 1=32
Saving config...
Config saved.
```

usrblk

Synopsis: `nvm usrblk <byte_cnt>`

Description: This command creates a user defined NVRAM image, and the size is determined by the argument `<byte_cnt>` (in bytes). If the image already exists, this command allows user to expand the image in size as long as the `<byte_cnt>` argument has a greater value than the current image size. To extract content from this user defined block, software must traverse the NVRAM directory structure to locate the image. Similar to other NVRAM images, this image is appended with a 4-byte CRC. Any software that attempts to change the content must also update it with the correct the CRC to ensure its integrity.

Returns: None

Example:

```
1.NONE:> nvm usrblk 100
1.NONE:> nvm dir
```

Image	SRAM Addr	NVM Offset	Byte Cnt	CPU	Version
BC1	0x08000010	0x00000600	0x00002340	MCP	bc1 1.4.0
BC2	0x080047C0	0x00002940	0x00003430	MCP	bc2 1.4.0
L2_RXP	0x08000000	0x00005D70	0x00000428	RXP	rxp 1.4.0
USR_BLK	0xFFFFFFFF	0x00006198	0x00000068	HOST	

pci

The **pci** commands provide a mechanism to select, discover, and initialize PCI devices.

init

Synopsis: **pci init** *[device number]*

Description: This command configures PCI related information (such as BAR address, latency timer) in the controller. This may be useful in situations where the controller is installed into a running system because it does not require that the system be reset to recognize and use the controller. Device number specifies from which device the configuration starts.

Returns: None

Example: 1.MCP:> pci init 1
1.MCP:>

scan

Synopsis: **pci scan**

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

Returns: None

Example: 1.NONE:> pci scan

Domain	Bus	Dev	Func	Vendor ID	Dev ID	Class	Base/IO Address	IRQ
0	0	0	0	8086	3590	06:00:00	00000000:00000000	0
0	0	2	0	8086	3595	06:04:00	00000000:00000000	5
0	0	3	0	8086	3596	06:04:00	00000000:00000000	5
0	0	4	0	8086	3597	06:04:00	00000000:00000000	5
0	0	6	0	8086	3599	06:04:00	00000000:00000000	5
0	0	29	0	8086	24D2	0C:03:00	00000000:00000000	5
0	0	29	1	8086	24D4	0C:03:00	00000000:00000000	11
0	0	29	2	8086	24D7	0C:03:00	00000000:00000000	10
0	0	29	3	8086	24DE	0C:03:00	00000000:00000000	5
0	0	29	7	8086	24DD	0C:03:20	00000000:D8000000	11
0	0	30	0	8086	244E	06:04:00	00000000:00000000	0
0	0	31	0	8086	24D0	06:01:00	00000000:00000000	0
0	0	31	1	8086	24DB	01:01:8 ^a	00000001:00000001	255
0	0	31	3	8086	24D3	0C:05:00	00000000:00000000	10
0	2	0	0	8086	0329	06:04:00	00000000:00000000	0
0	2	0	1	8086	0326	08:00:20	00000000:D8100000	0
0	2	0	2	8086	032 ^a	06:04:00	00000000:00000000	0
0	2	0	3	8086	0327	08:00:20	00000000:D8101000	0
0	4	1	0	14E4	164A	02:00:00	00000000:DA000004	5
0	7	1	0	1002	4752	03:00:00	00002001:DD000000	10
0	7	1	7	1002	4752	03:00:00	00002001:DD000000	10

search

Synopsis: `pci search [-did <deviceID>] [-vid <vendorID>] [-class <class>]`

Description: This command searches the entire PCI/PCI-E bus for all devices with a device ID **deviceID** or with a vendor ID **vendorID** or with a class code of **class**.

Returns: An ordered TCL list that represents the PCI bus, device, and function number of the matching device.

Example:

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

setdut

Synopsis: `pci setdut [<domain>] <bus> <dev> <func>`

Description: This command designates which PCI device to manipulate using **pcicfg** commands and requires the bus number, device number, and the function number of the device desired. If the selected device is a QLogic Network controller the command prompt will change to indicate which QLogic Network device is selected, otherwise the command prompt will display the device number as **99**.

Returns:

Example:

```
1.NONE:> pci setdut 4 1 0
4 1 0
99.NONE:> pcicfg read 0
0x164a14e4
```

pcicfg

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

read

Synopsis: **pcicfg read** <addr>

Description: This command reads data from PCI configuration register specified by **addr** using PCI configuration cycles.

Returns: The 32-bit value of the PCI configuration register.

Example: 1.NONE:> pcicfg read 0
0x164a14e4

show

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

Description: This command performs the same operation as the **pcicfg 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.

Returns:

Example: 1.NONE:> pcicfg show 4
0000000: 02b0011e
1.NONE:> pcicfg show 0 0x20
0000000: 164a14e4 02b0011e 02000001 00002008 da000004
00000000 00000000 00000000
1.NONE:>

write

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

Description: This command writes the 32-bit value to the PCI configuration register **addr**.

Returns: Returns **1** for success.

Example: 1.NONE:> pcicfg write 0x10 0xffffffff
1
1.NONE:>

qhack

Description: This command is for QLogic internal use only.

rebuf

Description: This command is for QLogic internal use only.

reg

The `reg` command provides a mechanism to read, modify, display, and write internal QLogic Network registers.

read, iread, and dread

Synopsis: `reg [slave <device number>] (read | iread | dread) <addr>`

Description: This command reads internal QLogic Network register data from the register at address **addr**. When the `read` subcommand is used, the actual method used to read the data over the PCI bus is determined by `xdiag`. The **iread** subcommand will force `xdiag` to perform the read using the indirect register access method (for example, using PCI configuration cycles via registers 0x78 and 0x80) while the `dread` subcommand will force `xdiag` to perform the read using the direct register access method (for example, using memory cycles to access BAR mapped memory). The **slave** option applies the command to the slave device if it has been configured.

Returns: The 32-bit value of the QLogic Network internal register.

Example:

```
1.NONE:> reg read 0x400
0x0
1.NONE:>
```

show and ishow

Synopsis: **reg** [**slave** <device number>] (**show** | **ishow**) <begin_addr> [<nbytes>]

Description: This command performs the same operation as the **reg 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 registers to display.

When the **show** subcommand is used, the actual method used to read the data over the PCI bus is determined by xdiag. The **ishow** subcommand will force xdiag to perform the reads using the indirect register access method (for example, using PCI configuration cycles via registers 0x78 and 0x80). The **slave** option applies the command to the slave device if it has been configured.

Returns: The 32-bit value of the QLogic Network internal register.

Example:

```
1.NONE:> reg show 0x408
0000400:..... 02000a9a
1.NONE:> reg show 0x400 0x20
0000400:00000000 00000000 02000a9a c40000ff 0a000064 0a000064
01020304 11121314
1.NONE:>
```

trace

Description: This subcommand is for QLogic internal use only.

write, iwrite and dwrite

Synopsis: **reg** [**slave** <device number>] (**write** | **iwrite** | **dwrite**) <addr> <value>

Description: This command writes internal QLogic Network registers at the **addr** with the 32-bit value. When the **write** subcommand is used, the actual method used to write the data over the PCI bus is determined by xdiag. The **iwrite** subcommand will force xdiag to perform the write using the indirect register access method (for example, using PCI configuration cycles via registers 0x78 and 0x80) while the **dwrite** subcommand will force xdiag to be perform the write using the direct register access method (for example, using memory cycles to access BAR mapped memory). The **slave** option applies the command to the slave device if it has been configured.

Returns: Returns 1 for success.

Example:

```
1.NONE:> reg write 0x808 0x01234567
1
1.NONE:>
```

serialport

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

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 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.

Returns: None

Example:

```
1.NONE:> serialport open 2
opened port 2 settings: 115200, 8N1
1.NONE:>
```

redirect

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: None

Example:

```
1.NONE:> serialport redirect
```

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: None

Example:

```
1.NONE:> serialport close
1.NONE:>
```

sram

The **sram** command provides a mechanism to read, modify, display, and write to QLogic Network internal memory.

fill

Synopsis: **sram fill** <addr> <bytecount> <dword_pattern>

Description: This command fills a particular CPU's scratchpad memory starting at address **addr** with the data pattern **dword_pattern** for a length of **bytecount** bytes. The CPU should be previously selected by the **cpu select** command.

Returns: None

Example:

```
1.NONE:>cpu select mcp
1.MCP:> sram fill 4 20 0x12121212
done
1.MCP:>
```

read

Synopsis: **sram read** <addr>

Description: This command reads the 32-bit data from the location **addr** of a particular CPU scratchpad memory. If the xdiag command prompt CPU selection is **NONE** then the command is identical to **reg read**.

Returns: The 32-bit value of the QLogic Network internal memory.

Example:

```
1.NONE:> sram read 0
0x164c14e4
1.NONE:> cpu select mcp
MCP
1.MCP:> sram read 0
0xa000008
```


show

Synopsis: **sram show** [<begin_addr> [< nbytes>]]

Description: This command performs the same operation as the **sram 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. If *begin_addr* is not specified, the **sram show** command will use the previous (*begin_addr* + *nbytes*) as the new beginning address.

Returns: None

Example:

```
1.MCP:> sram show 0x10
0000000:..... 62633220
1.MCP:> sram show 0 0x40
0000000:0a000008 00000000 00000000 0000000d 62633220 302e332e
36000000 00030605
0000020:00000000 10000003 00000000 0000000d 0000000d 3c020800
24421fc4 3c030800
```

write

Synopsis: **sram write** <addr> < value>

Description: This command writes the 32-bit value to the location *addr* of a particular CPU scratchpad memory. If the current selection is **NONE**, the command is identical to **reg write**.

Returns: Returns **1** for success.

Example:

```
1.MCP:> sram write 0 0x0
1
1.MCP:>
```

stats

Description: This command is for QLogic internal use only.

value

Synopsis: **value** <value>

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

Returns: None

Example:

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

version

Synopsis: **version**

Description: This command displays the version string for xdiag.

Returns: None

Example:

```
0:> version
xdiag version: 0.8.7
```

watch

Description: This command is for QLogic internal use only.

what

Description: This command is for QLogic internal use only.

wol

Description: This command is for QLogic internal use only.

xfer

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

send

Synopsis: **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 exists, it will be overwritten. Note that the serial port has to be ready to use by issuing **serialport open** command.

Returns: None

Example: `xfer send ../stdscrpt/5706/12tx.tcl`

receive

Synopsis: **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 xdiag and user does not have to run the **xfer receive** command. If **rcv_file** already exists, 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.

Returns:

Example: `xfer receive ../stdscrpt/5706/12tx.tcl`

yield

Description: This command is for QLogic internal use only.

?

Description: This command displays a listing of the internal and macro commands supported by xdiag. [“Engineering Mode Macros” on page 92](#) provides additional documentation for the macro commands.

7 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.

blast

Synopsis: `blast`

Description: This macro allows the user to perform transmit and receive stress tests on the QLogic Network controller. (Refer to the description of `txcfg` in [“txcfg” on page 99](#) for details on configuring the type of transmitted traffic.) Before the blast command can be loaded, the user must manually load the xdiag internal Ethernet driver with the driver load command documented in [“load” on page 46](#). The following blast options are supported:

- `f` - Check the packet content on every `n` pkt (see `-k` option)
- `h` - Enable host loopback (this disables `-t` & `-r`)
- `<n>` - Run for `<n>` iterations (0=>infinite) (def=1)
- `k<n>` - Perform Rx check on every `<n>` pkt (def=0: no check)
- `p` - Use polling instead of interrupt
- `n<n>` - Number of packets to be transmitted (def=100)
- `r` - Enable Rx
- `t` - Enable Tx
- `u` - Update display
- `x` - Check CRC on every `n` pkt (see `-k` option)

Returns: None

Example:

init

Synopsis: `init`

Description: This macro initializes the chip to a stable state and is part of the reset macro (described in [“reset” on page 96](#)).

Returns: Returns 1 for success.

Example:

```
1.NONE:> init
C: Brd:Rv Bus PCI Spd Base IRQ MAC FmwVer Configuration
1:5706C:A1 04:01:00 PCI-64 66 0xDA00 5 001018044800 0.3.6
W,Mp,auto
Program EMAC-EXT_LINK_POL must be 0 here!
Program TBDR-in wrong offset 0x5004 & MAX_BURST not implemented
Program HC
status block addr: 0089b580
statistic block addr: 0089b6d8
Program DMA -- 3 wchan & 5 rchan & DMAE workaround
HC in collect mode.
Go except EMAC & Timer.
1
```

loadfw

Synopsis: loadfw

Description: This macro loads firmware to all the QLogic Network's CPUs except for MCP and is identical to the method used by device drivers. Note that the firmware files located in **fw** directory are required for this to work.

Returns: None

Example:

```
1.NONE:> loadfw
Loading firmware from C:/WORK/BCM5706/DIAG/00807R/fw
Wrote 368 instructions to PROC 1 from file
C:/WORK/BCM5706/DIAG/00807R/fw/rv2p/rv2p_p1.h of 390 lines.
Wrote 763 instructions to PROC 2 from file
C:/WORK/BCM5706/DIAG/00807R/fw/rv2p/rv2p_p2.h of 785 lines.
Writing Text section to address 0x8000000 offset 0x4030
Writing Rodata section to address 0x0 offset 0x0000
Writing Data section to address 0x8004060 offset 0x0000
Writing SBSS section to address 0x0 offset 0x0000
Writing BSS section to address 0x0 offset 0x0000
Writing Text section to address 0x8000000 offset 0x1030
Writing Rodata section to address 0x0 offset 0x0000
Writing Data section to address 0x8001060 offset 0x0000
Writing SBSS section to address 0x0 offset 0x0000
Writing BSS section to address 0x0 offset 0x0000
Writing Text section to address 0x8000000 offset 0x22fc
Writing Rodata section to address 0x0 offset 0x0000
Writing Data section to address 0x8002320 offset 0x0000
Writing SBSS section to address 0x0 offset 0x0000
Writing BSS section to address 0x0 offset 0x0000
Writing Text section to address 0x8000000 offset 0x4660
Writing Rodata section to address 0x8004660 offset 0x0018
Writing Data section to address 0x80046a0 offset 0x0000
Writing SBSS section to address 0x0 offset 0x0000
Writing BSS section to address 0x0 offset 0x0000
Writing Text section to address 0x8000000 offset 0x4860
Writing Rodata section to address 0x8004860 offset 0x00b0
Writing Data section to address 0x8004940 offset 0x0050
Writing SBSS section to address 0x0 offset 0x0000
Writing BSS section to address 0x0 offset 0x00001
1.CP:>
```

qstat

Synopsis: qstat

Description: This macro displays a short block of statistics gathered information from statistic block memory and from the firmware.

- **slave** <device number> - Displays statistics of specified slave device

Returns: None

Example:

```
1.NONENONE:> qstat
Txed Packets           : 11243584798490967245
Rxed Packets           : 11243584798490967245
Control Packets        : 2617851085
etherStatsCollisions   : 3735928559
d3StatsFCSErrors       : 3735928559
d3StatsExcessiveCol    : 3735928559
ifInDiscards           : 2058812348
d3StatsInt1MacTxErrors : 3735928559
Tx Dropped             : N/A
Txed Octets            : 16045690984833335023
Rxed Octets            : 16045690984833335023
FW Dropped
```

redirect

Synopsis: redirect ?comport?

Description: This macro is used to open a serial port and redirect console input and output through that port. The optional comport argument specifies the COM port number, with a default being 1.

Returns: None

Example:

```
1.NONE:> redirect 1
Opened port 1 settings: 115200,8N1
Serial port 1 is used for I/O redirection
Serial port 1 is used for watch/stat redirection
```

reset

Synopsis: reset

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

Returns: Returns 1 for success.

Example:

```
1.NONE:> reset
C: Brd:Rv Bus PCI Spd Base IRQ NVM MAC FmwVer Configuration
1:5706C:A1 04:01:0 P64 66 0xDA00 5 128a 001018044800 0.3.6 W,Mp,auto
Program EMAC-EXT_LINK_POL must be 0 here!
Program TBDR-in wrong offset 0x5004 & MAX_BURST not implemented
Program HC
status block addr: 0089b580
statistic block addr: 0089b6d8
Program DMA -- 3 wchan & 5 rchan & DMAE workaround
HC in collect mode.
Go except EMAC & Timer.
1
```

sb

Synopsis: sb

Description: This macro displays the status block maintained in host memory.

Returns: Returns 1 for success.

Example:

```
1.NONE:> sb
-----
STATUS BLOCK
-----
attn_bits : dead beef
attn_bits_ack : dead beef
tx0 tx1 : dead beef
tx2 tx3 : dead beef
rx0 rx1 : dead beef
rx2 rx3 : dead beef
rx4 rx5 : dead beef
rx6 rx7 : dead beef
rx8 rx9 : dead beef
rx10 rx11 : dead beef
rx12 rx13 : dead beef
rx14 rx15 : dead beef
comp cmd : dead beef
idx - : dead ----
-----
1
```




Synopsis: tb

Description: This macro displays the statistic block maintained in host memory.

Returns: None

Example:

```
1.NONE:> tb
```

```
-----
STATISTICS BLOCK
-----
```

```
If_HC_In_Octets      : deadbeef deadbeef (1604569098483335023)
If_HC_In_Bad_Octets  : deadbeef deadbeef (1604569098483335023)
If_HC_Out_Octets     : deadbeef deadbeef (1604569098483335023)
If_HC_Out_Bad_Octets : deadbeef deadbeef (1604569098483335023)
If_HC_In_Ucast_Pkts  : deadbeef deadbeef (1604569098483335023)
If_HC_In_Multicast_Pkts : deadbeef deadbeef (1604569098483335023)
If_HC_In_Broadcast_Pkts : deadbeef deadbeef (1604569098483335023)
If_HC_Out_Ucast_Pkts : deadbeef deadbeef (1604569098483335023)
If_HC_Out_Multicast_Pkts : deadbeef deadbeef (1604569098483335023)
If_HC_Out_Broadcast_Pkts : deadbeef deadbeef (1604569098483335023)
-----
```

```
Dot3Stats_internal_mac_transmit_errors : deadbeef (3735928559)
```

```
Dot3Stats_CarrierSense_Errors : deadbeef (3735928559)
```

```
Dot3Stats_FCS_Errors : deadbeef (3735928559)
```

```
Dot3Stats_Alignment_Errors : deadbeef (3735928559)
```

```
more..., 'q' to quit
```

```
Dot3Stats_Single_Collision_Frames      : deadbeef (3735928559)
Dot3Stats_Multiple_Collision_Frames     : deadbeef (3735928559)
Dot3Stats_Deferred_Transmissions        : deadbeef (3735928559)
Dot3Stats_Excessive_Collisions          : deadbeef (3735928559)
Dot3Stats_Late_Collisions                : deadbeef (3735928559)
EtherStats_Collisions                   : deadbeef (3735928559)
EtherStats_Fragments                    : deadbeef (3735928559)
EtherStats_Jabbers                      : deadbeef (3735928559)
EtherStats_Undersize_Pkts                : deadbeef (3735928559)
EtherStats_Oversize_Pkts                 : deadbeef (3735928559)
EtherStats_Pkts_Rx_64_Octets             : deadbeef (3735928559)
EtherStats_Pkts_Rx_65_Octets_to_127_Octets : deadbeef (3735928559)
EtherStats_Pkts_Rx_128_Octets_to_255_Octets : deadbeef (3735928559)
EtherStats_Pkts_Rx_256_Octets_to_511_Octets : deadbeef (3735928559)
EtherStats_Pkts_Rx_512_Octets_to_1023_Octets : deadbeef (3735928559)
EtherStats_Pkts_Rx_1024_Octets_to_1522_Octets : deadbeef (3735928559)
EtherStats_Pkts_Rx_1523_Octets_to_9022_Octets : deadbeef (3735928559)
-----
```

```
more..., 'q' to quit
```

```
EtherStats_Pkts_Tx_64_Octets            : deadbeef (3735928559)
EtherStats_Pkts_Tx_65_Octets_to_127_Octets : deadbeef (3735928559)
```

7-Engineering Mode Macros

tb

```
EtherStats_Pkts_Tx_128_Octets_to_255_Octets    : deadbeef (3735928559)
EtherStats_Pkts_Tx_256_Octets_to_511_Octets    : deadbeef (3735928559)
EtherStats_Pkts_Tx_512_Octets_to_1023_Octets   : deadbeef (3735928559)
EtherStats_Pkts_Tx_1024_Octets_to_1522_Octets  : deadbeef (3735928559)
EtherStats_Pkts_Tx_1523_Octets_to_9022_Octets  : deadbeef (3735928559)
Xon_Pause_Frames_Received                      : deadbeef (3735928559)
Xoff_Pause_Frames_Received                     : deadbeef (3735928559)
Out_Xon_Sent                                   : deadbeef (3735928559)
Out_Xoff_Sent                                  : deadbeef (3735928559)
Flow_Control_done                             : deadbeef (3735928559)
Mac_Control_Frames_Received                    : deadbeef (3735928559)
Xoff_State_Entered                            : deadbeef (3735928559)
If_In_Frames_L2_Filter_Discards                 : deadbeef (3735928559)
If_In_Rule_Checker_Discards                     : deadbeef (3735928559)
If_In_FTQ_Discards                             : deadbeef (3735928559)
If_In_MBUF_Discards                            : deadbeef (3735928559)
-----
more..., 'q' to quit
If_In_Rule_Checker_P4_Hit                      : deadbeef (3735928559)
Catchup_In_Rule_Checker_Discards               : deadbeef (3735928559)
Catchup_In_FTQ_Discards                        : deadbeef (3735928559)
Catchup_In_MBUF_Discards                       : deadbeef (3735928559)
Catchup_In_Rule_Checker_P4_Hit                 : deadbeef (3735928559)
Gen_Stat_00                                    : deadbeef (3735928559)
Gen_Stat_01                                    : deadbeef (3735928559)
Gen_Stat_02                                    : deadbeef (3735928559)
Gen_Stat_03                                    : deadbeef (3735928559)
Gen_Stat_04                                    : deadbeef (3735928559)
Gen_Stat_05                                    : deadbeef (3735928559)
Gen_Stat_06                                    : deadbeef (3735928559)
Gen_Stat_07                                    : deadbeef (3735928559)
Gen_Stat_08                                    : deadbeef (3735928559)
Gen_Stat_09                                    : deadbeef (3735928559)
Gen_Stat_10                                    : deadbeef (3735928559)
Gen_Stat_11                                    : deadbeef (3735928559)
-----
more..., 'q' to quit
Gen_Stat_12                                    : deadbeef (3735928559)
Gen_Stat_13                                    : deadbeef (3735928559)
Gen_Stat_14                                    : deadbeef (3735928559)
Gen_Stat_15                                    : deadbeef (3735928559)
```

txcfg

Synopsis: txcfg

Description: This macro allows the user to configure different parameters which are used to generate Ethernet frames for use with the blast -t macro.

Returns: None

Example:

```
0.NONE:> txcfg
1: Source MAC                      : 00:10:18:00:00:00
2: Destination MAC                 : ff:ff:ff:ff:ff:ff
3: Min Packet Length               : 64
4: Max (fixed) Packet Length       : 1518
5: Packet Length Mode {fixed(0),random(-1),inc(x>0)} : fixed
6: Packet Type {EthII(0),802.3raw(1),llc(2),SNAP(3)} : ethernetII
7: Protocol Field {None(0),IP(1),ARP(2),RARP(3),flow(4)} : none
8: Layer 4 Type {random(0),TCP(1),UDP(2)}             : tcp
9: Source IP                       : 4.3.2.1
10: Destination IP                 : 1.2.3.4
11: Source Port                    : 100
12: Destination Port               : 200
13: IP Option Length (32-bit words) : 0
14: Randomize IP Option {true(1),false(0)}            : true
15: TCP Option Length (32-bit words) : 0
16: Randomize TCP Option {true(1),false(0)}           : false
17: Pattern {random(0),inc(1),0xff(2),0x00(3),
    0x55aa(4),0xaa55(5)} : increment
18: IP Checksum Offload {sw(0),hw(1),specified(2)}    : software
19: IP Checksum Value               : 0x0000
20: TCP/UDP Checksum Offload {sw(0),hw(1),specified(2)} : software
21: TCP/UDP Checksum Value          : 0x0000
22: Insert VLAN Tag {true(1),false(0)}               : false
23: VLAN Tag Insertion Offload {hw(1),sw(0)}          : hardware
24: VLAN Tag Value                   : 0x0000
25: CRC Offload {hardware(1),software(0)}             : hardware
26: CRC Pre-generated Value         : 0x00000000
27: BD Arrangement {size1 size2 size3 ...}           : 2000
0: Exit
Choice:
```

8 Configuration Files

The xdiag utility supports various configuration files which are used to configure or verify various QLogic Network device parameters during manufacturing. This section will document the usage and syntax for each of these files.

The **config.txt** contains many parameters to configure the device. 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. The **cfgchk.txt** provides a sanity check for OEM customers to verify the validity of certain configuration parameters.

config.txt

The **config.txt** file is used to configure all of the supported NVRAM options of the QLogic Network controller and should be used when the OEM does not maintain a **Golden Image** file of the NVRAM contents.

The **config.txt** file acts exactly like a TCL script and is invoked from the xdiag command-line as follows:

```
xdiag -rc config.txt
```

The parameters programmed will not take effect until the QLogic Network controller is reset. If the **BAR size** option is changed the system must be rebooted to allow the system BIOS to allocate memory for the new PCI BAR setting.

Refer to the QLogic Network NVRAM Configuration Application Note for additional details on each of the configurable options below and QLogic's recommendations for settings.

The following is a sample **config.txt** file from one port of the 5709 device:

```
# config.txt
# This file contains the NVRAM configuration for QLogic Network devices
# To incorporate these settings into a device, simply invoke xdiag as
# follows.
# xdiag -rc config.txt
# 1: [P] MAC Address : 00:10:18:3e:14:48
# 2: [P] NIC Power Dissipated (D3:D2:D1:D0) : 25:0:0:25
# 3: [P] NIC Power Consumed (D3:D2:D1:D0) : 25:0:0:25
# 4: [P] Vendor ID : 14E4
# 5: [P] Vendor Device ID : 1639
# 6: [P] Subsystem Vendor ID : 14E4
# 7: [P] Subsystem Device ID : 0906
# 8: [S] Product Name : QLogic Ethernet Controller
# 9: [S] Part Number : BCM95709A0906G
# 10: [S] Engineering Change : 116870-11
# 11: [S] Serial Number : 0123456789
# 12: [S] Manufacturing ID : 14e4
# 13: [P] Led Mode { MacMode(0), PhyModel(1),
# PhyMode2(2),..., PhyMode7(7) } : PhyMode9
# 14: [S] Design Type: {NIC(0), LOM(1)} : NIC
# 16: [P] BAR size: {Disabled(0), 64K(1), 128K(2),
# ...1M(5),...16M(9), ...64M(11), ...1G(15)} : 32M
# 17: [P] Magic Packet WoL { Disable(0), Enable(1) } : Enabled
# 18: [P] WoL link speed { Auto(0), ...1000FD(6) } : Auto
# 21: [P] MBA { Disable(0), Enable(1) } : Disabled
# 22: [P] MBA Link Speed { Auto(0), ...1000FD(6),
# 2500FD(8)} : Auto
# 23: [P] MBA Boot Protocol { PXE(0), RPL(1), BOOTP(2),
# iSCSI boot(3), reserved(4-6), None(7) } : PXE
# 24: [P] MBA Boot Type { Auto(0), BBS(1), Int18(2),
```

8-Configuration Files

config.txt

```
#          Int19(3) }                                     : Auto
# 25: [P] MBA Delay Time (0-15) (0 defaults to 4 seconds,
#          15 is no delay, 1-14 is delay value in second) : 5
# 26: [P] MBA Setup Hot Key {Ctrl-S(0), Ctrl-B(1)}       : Ctrl-S
# 27: [P] MBA hide setup prompt {Disable(0), Enable(1)}  : Disabled
# 28: [S] Expansion ROM size { Disabled(0),
#          ...64k(7),...16M(15) }                        : 128k
# 29: [S] Mgmt Firmware { Disable(0), Enable(1) }       : Disabled
# 33: [P] iSCSI MAC Address                             : 00:10:18:3e:14:49
# 34: [P] MBA VLAN { Disable(0), Enable(1) }            : Disabled
# 35: [P] MBA VLAN value                               : 0
# 36: [S] Vaux current overdraw { Disable(0), Enable(1) } : Disabled
# 37: [S] UMP/NCSI uses RMII { Disable/MII only(0),
#          Enable(1) }                                   : Disabled
# 39: [S] Spare Number                                 : BCM95709A0906G
# 43: [S] UMP/NCSI PHY Timing (5708 B0 and after)
#          {Disable(0),Enable(1)}                       : Disabled
# 44: [P] SMBus Address {one byte even hex value
#          (i.e.bit0=0)}                                : 0xfe
# 47: [S] UMP Echo mode { Disabled(0), Enable(1) }      : Disabled
# 48: [S] Allow gigabit link on Vaux
#          {Disabled(0), Enable(1)}                     : Disabled
# 49: [P] Default link setting (serdes only) {Autoneg(0),
#          forced 1G(3), forced 2.5G(4), AN w/1G(19),
#          AN w/2.5G(20)}                               : Autoneg
# 58: [S] Read only VPD Vendor Specific Data (V0)       :
# 59: [S] Port swap { Disable(0), Enable(1) }
#          (5709/16 only)                               : Disabled
# 61: [S] PCIE Gen2 (5709/16 only) { Disable(0),
#          Enable(1) }                                   : Enabled
# 62: [S] Function hide {None(0), Reserved(1), Func1(2)}
#          (5709/16 only)                               : None
# 63: [S] Mgmt FW load choice { Any(0), NCSI(1), UMP(2),
#          IPMI(3), NCSI/IPMI(4), UMP/IPMI(5), NCSI/UMP(6) }
#          (4, 5, 6 are determined by SPIO4 (0/1), 5709
#          & 5716 only)                                 : Any
# 64: [S] NCSI Package ID assignment method { SPIO(0),
#          NVRAM(1) } (5709/16 only)                    : SPIO
# 65: [S] NCSI Package ID assigned value { 0-3 } (5709/16) : 0
# 66: [S] SMBus Timing { 100kHz(0), 400kHz(1) } (5709/16) : 100kHz
# 67: [S] PCIE power budget data 0 (Bits 20-0) (5709/16) : 0x000000
# 68: [S] PCIE power budget data 1 (Bits 20-0) (5709/16) : 0x000000
# 69: [S] PCIE power budget data 2 (Bits 20-0) (5709/16) : 0x000000
# 70: [S] PCIE power budget data 3 (Bits 20-0) (5709/16) : 0x000000
# 71: [S] PCIE power budget data 4 (Bits 20-0) (5709/16) : 0x000000
# 72: [S] PCIE power budget data 5 (Bits 20-0) (5709/16) : 0x000000
# 73: [S] PCIE power budget data 6 (Bits 20-0) (5709/16) : 0x000000
# 74: [S] PCIE power budget data 7 (Bits 20-0) (5709/16) : 0x000000
# 76: [P] Prevent PCIE relax ordering {Disable(0),
```

```
#          Enable(1)} (5709/16)                : Disabled
# 77: [P] Primary backup MAC Address            : 00:00:00:00:00:00
# 78: [P] Force Expansion ROM advertisement { Disable(0),
#          Enable(1) }                          : Disabled
nvm cfg \
    "2=25:0:0:25"\
    "3=25:0:0:25"\
    "4=14E4"\
    "5=1639"\
    "6=14E4"\
    "7=0906"\
    "8=QLogic Ethernet Controller"\
    "9=BCM95709A0906G"\
    "10=116870-11"\
    "11=0123456789"\
    "12=14e4"\
    "13=11"\
    "14=0"\
    "16=10"\
    "17=1"\
    "18=0"\
    "21=0"\
    "22=0"\
    "23=0"\
    "24=0"\
    "25=5"\
    "26=0"\
    "27=0"\
    "28=8"\
    "29=0"\
    "34=0"\
    "35=0"\
    "36=0"\
    "37=0"\
    "39=BCM95709A0906G"\
    "43=0"\
    "44=0xfe"\
    "47=0"\
    "48=0"\
    "49=0"\
    "58=\"\""\
    "59=0"\
    "61=1"\
    "62=0"\
    "63=0"\
    "64=0"\
    "65=0"\
    "66=0"\
    "67=0x000000"
```

```
"68=0x000000"\  
"69=0x000000"\  
"70=0x000000"\  
"71=0x000000"\  
"72=0x000000"\  
"73=0x000000"\  
"74=0x000000"\  
"76=0"\  
"78=0"\  

```


macaddr.txt

The **macaddr.txt** file is used to program the primary MAC and iSCSI MAC address onto the QLogic Network controller 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 xdiag command-line as follows:

```
xdiag -fmac macaddr.txt
```

Note that the two MAC addresses must be consecutive and that the primary MAC address will be assigned first. After the two addresses have been programmed the **macaddr.txt** file will be updated such that the **mac_addr_start** value is incremented by two to reflect the allocated addresses. For example, if **mac_addr_start** has a value of **042238** before **xdiag -fmac macaddr.txt** is run, after the command is executed, **mac_addr_start** will have a value of **04223A**. 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 = 042238
mac_addr_end = 042246
```

cfgchk.txt

The **cfgchk.txt** file is used to allow an OEM to verify the configuration of certain NVRAM configurable parameters and optional software/firmware that may be included in the QLogic Network NVRAM. The configuration check is invoked from the xdiag command-line as follows:

```
xdiag -cfgchk cfgchk.txt
```

Any mismatches will be reported to the operator and generate a return code that can be checked by other DOS applications. (See [“Return Codes” on page 38](#) for details on supported return codes.)

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

Parameter	Description
mac_prefix	The first 3 bytes of MAC address. It is in the form of xx:xx:xx where xx is in hexadecimal without 0x prefix. This is to verify the value of mac_addr_pref in macaddr.txt .
system_device_id	The system device ID. Values are in hexadecimal without 0x prefix. This is to verify the value of Vendor Device ID in config.txt .
subsystem_vendor_id	The subsystem vendor ID. Values are in hexadecimal without 0x prefix. This is to verify the value of Subsystem Vendor ID in config.txt .
subsystem_device_id	The subsystem device ID. Values are in hexadecimal without 0x prefix. This is to verify the value of Subsystem Device ID in config.txt .
chip_rev	The chip revision. Value is in a form of A0 . This is to verify the correct revision of the chip is used in manufacturing.
bootcode_version	The boot code version. It is in the form of a string without any white spaces. This is to verify the proper boot code version.
mba_version	The multiple boot agent version. It is in the form of a string without any white spaces. This is to verify the proper multiple boot agent version.
mba_enabled	Is MBA enabled? It has a value of either yes (enable) or no (disable). This is to verify MBA in config.txt .
wol_enabled	Is WoL enabled? It has a value of either yes (enable) or no (disable). This is to verify Magic Packet WoL in config.txt .

Parameter	Description
<code>max_toe_conn</code>	This specifies the maximum number of TOE connections allowed in the manufacturing key. If specified, it is assumed that this is enabled.
<code>max_rdma_conn</code>	This specifies the maximum number of RDMA connections allowed in the manufacturing key. If specified, it is assumed that this is enabled.
<code>max_iscsi_initiator_conn</code>	This specifies the maximum number of iSCSI initiator connections allowed in the manufacturing key. If specified, it is assumed that this is enabled.
<code>max_iscsi_target_conn</code>	This specifies the maximum number of iSCSI target connections allowed in the manufacturing key. If specified, it is assumed that this is enabled.
<code>max_iser_initiator_conn</code>	This specifies the maximum number of iSER initiator connections allowed in the manufacturing key. If specified, it is assumed that this is enabled.
<code>max_iser_target_conn</code>	This specifies the maximum number of iSER target connections allowed in the manufacturing key. If specified, it is assumed that this is enabled.
<code>iscsi_boot_enabled</code>	This specifies whether iSCSI boot is allowed or not in the manufacturing key. The possible values are yes or no .
<code>iscsi_full_acceleration_enabled</code>	This specifies whether iSCSI full acceleration is allowed or not in the manufacturing key. The possible values are yes or no .
<code>iscsi_header_digest_enabled</code>	This specifies whether iSCSI header digest is allowed or not in the manufacturing key. The possible values are yes or no .
<code>iscsi_body_digest_enabled</code>	This specifies whether iSCSI body digest is allowed or not in the manufacturing key. The possible values are yes or no .
<code>2_5g_capability</code>	This specifies whether 2.5G speed (SERDES only) is allowed or not in the manufacturing key. The possible values are yes or no .
<code>expiration_date</code>	This specifies when the license expires as indicated in the manufacturing key. The date format is either never or yyyymmdd .

The following is a sample cfgchk.txt file. Note that the license related items are commented out in the example.

```
# MAC Prefix: in the form of xx:xx:xx
mac_prefix = 00:10:18
# System IDs: in a form of hex without "0x" prefix
system_device_id = 164c
subsystem_vendor_id = 14e4
subsystem_device_id = 164c
chip_rev = B1
# Firmware and MBA versions: in a form of a string without any
white spaces
bootcode_version = 1.0.5
mba_version = 2.2.1

# Configuration: either "yes" or "no"
mba_enabled = yes
wol_enabled = yes
# License connection information: in a form of integer
# max_toe_conn = 1234
# max_rdma_conn = 2345
# max_iscsi_initiator_conn = 3456
# max_iscsi_target_conn = 4567
# max_iser_initiator_conn = 5678
# max_iser_target_conn = 6789

#License configuration information: either "yes" or "no"
# iscsi_boot_enabled = yes
# iscsi_full_acceleration_enabled = no
# iscsi_header_digest_enabled = yes
# iscsi_body_digest_enabled = no
# 2_5g_capability = yes
#License expiration information: either "never" or "yyyymmdd"
# expiration_date = 20010101
```

A Appendix A - Tcl Reference

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

B

Appendix B-TCL Environment Variables

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

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`.

toe

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.

<code>\$::toe(BASE_ADDR)</code>	The base address of the selected device (for example: 0xffbe0000).
<code>\$::toe(BASE_ADDR_HIGH)</code>	The top 32-bits of the base address of the selected device (for example: 0x0)
<code>\$::toe(BASE_ADDR_LOW)</code>	The bottom 32-bits of the base address of the selected device (for example: 0xffbe0000).
<code>\$::toe(BOARD)</code>	The board number of the selected device (for example: 5708).
<code>\$::toe(BUS)</code>	The bus, device, and function number of the selected device (for example: 00:0b:0).
<code>\$::toe(BUS_TYPE)</code>	The type of PCI bus on which the selected device resides (for example: PCIX-64, PCIE-4).
<code>\$::toe(CHIP_BUILD)</code>	Always ASIC .
<code>\$::toe(CHIP_REV)</code>	The chip revision of the selected device (for example: A0).

<code>\$::toe(CPU)</code>	The currently selected CPU (for example: NONE , TXP).
<code>\$::toe(DEV)</code>	Currently selected device (value = 0, 1, ...).
<code>\$::toe(DIAG_VER)</code>	The version of this diagnostic program in string (for example: 0.8.7).
<code>\$::toe(DID)</code>	The PCI device ID (for example: 0x164a).
<code>\$::toe(DRV_STATE)</code>	The current state of the driver (for example: SETUP , RUN).
<code>\$::toe(FW_VER)</code>	The version of the firmware residing in the selected device (for example: 0.3.6).
<code>\$::toe(IPMI_VER)</code>	The version of the IPMI firmware residing in the selected device (for example: 1.6.0).
<code>\$::toe(IRQ)</code>	The IRQ number for the selected device (for example: 10).
<code>\$::toe(MAC_ADDR)</code>	The MAC address of the selected device (for example: 001018010B23).
<code>\$::toe(MAX_SPEED)</code>	The maximum data rate of the selected device (for example: 1000).
<code>\$::toe(MBA)</code>	An indicator of whether or not MBA is installed and enabled on the selected device (for example: 1 → enabled).
<code>\$::toe(MBA_SPEED)</code>	The speed of MBA (for example: auto , 10 HD).
<code>\$::toe(MBA_VER)</code>	The version of MBA image installed on the device (for example: 1.1.2).
<code>\$::toe(MFW)</code>	An indicator of whether management firmware is installed and enabled on the selected device (for example: 1 → enabled).
<code>\$::toe(NVM_SIZE)</code>	The flash size on the selected device (for example: 135168).
<code>\$::toe(NVM_TYPE)</code>	The type of flash on the selected device (for example: BUFFERED).
<code>\$::toe(PCI_SPEED)</code>	The speed of the PCI bus on which the selected device resides (for example: 33).
<code>\$::toe(PHY_TYPE)</code>	The PHY medium type of the device (for example: COPPER, SERDES).

<code>\$::toe(PORT0)</code>	port 0 or port 1 of the 5709 dual port device (for example: 1 → port 0, 0 → port 1).
<code>\$::toe(RG_VER)</code>	Register specification version of the device (for example: reg_spec.r17).
<code>\$::toe(SSID)</code>	The PCI subsystem ID of the device (for example: 0x164a).
<code>\$::toe(SVID)</code>	The PCI subsystem vendor ID of the device (for example: 0x14e4).
<code>\$::toe(TOTAL_DEV)</code>	Total number of of 5706/5708/5709 devices (for example: 3).
<code>\$::toe(UMP_VER)</code>	The version of the UMP firmware residing in the selected device (for example: 1.1.6).
<code>\$::toe(VID)</code>	The PCI vendor ID of the device (for example: 0x14e4).
<code>\$::toe(WOL)</code>	An indicator of whether or not WOL is enabled on the selected device (for example: 0 → disabled).

drv06_xx

The **drv06_xx** environment variable is used to maintain state information used by the internal xdiag device driver. The **xx** represents the device number shown in the xdiag command prompt. For example, if device **1** is selected, the **xx** would be **01**, making the variable name **drv06_01**. These variables are similar to those found in the registry in Windows environment. Any changes to these variables are not effective until the driver is loaded (via **driver load** command). If it is already loaded when the change occurs, it will not be effective until the next load.

<code>\$::drv06_xx(mac_addr)</code>	The primary MAC address to be used to override the default one stored in non-volatile memory.
<code>\$::drv06_xx(mtu)</code>	The MTU size of a packet.
<code>\$::drv06_xx(num_rchans)</code>	The number of DMA read channels used by the chip (default: 5)
<code>\$::drv06_xx(num_wchans)</code>	The number of DMA write channels used by the chip (default: 3)
<code>\$::drv06_xx(req_medium)</code>	The medium type (for example: 0 for auto-negotiate, 0xfe for MAC loopback, 0xfd for PHY loopback)

sys

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

<code>\$sys(BATCH)</code>	Determines whether the diagnostic is running in interactive or batch mode. In batch mode, pagination will be turned off.
<code>\$sys(BG_YIELD)</code>	Determines whether or not to invoke yield in the background.
<code>\$sys(DISP_BYTE)</code>	Determines how memory content is displayed, whether by dwords or by bytes.
<code>\$sys(IO_PORT)</code>	The COM port number that the I/O will go to. This is usually set for the command serialport open .
<code>\$sys(STAT_PORT)</code>	The COM port number that the statistic window will go to. This is usually set for the command serialport open .
<code>\$sys(ARG)</code>	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 xdiag is invoked.
<code>\$sys(NO_INIT)</code>	Tells xdiag whether to perform some basic chip initialization upon chip reset. The command line switch -noinit will cause this value to be set to 1 (0 otherwise). This may be useful for debugging the chip's state since it will prevent xdiag from modifying any of the QLogic Network's registers.



Corporate Headquarters QLogic Corporation 26650 Aliso Viejo Parkway Aliso Viejo, CA 92656 949.389.6000 www.qlogic.com
International Offices UK | Ireland | Germany | France | India | Japan | China | Hong Kong | Singapore | Taiwan

© 2015 QLogic Corporation. Specifications are subject to change without notice. All rights reserved worldwide. QLogic and the QLogic logo are registered trademarks of QLogic Corporation. All other brand and product names are trademarks or registered trademarks of their respective owners. Information supplied by QLogic Corporation is believed to be accurate and reliable. QLogic Corporation assumes no responsibility for any errors in this brochure. QLogic Corporation reserves the right, without notice, to make changes in product design or specifications.

CONFIDENTIAL