Highly Integrated Media Access Controller Programmer's Guide

57710_57711-PG200-R

5300 Califlornia Avenue • Irvine, CA 92617 • Phone: 949-926-5000 • Fax: 949-926-5203

09/25/09

REVISION HISTORY

Revision	*Date*	*Change Description*
57710_57711-PG200-R	09/25/09	Initial release.

Broadcom Corporation
5300 Calilfornia Avenue
Irvine, CA 92617

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Section 1: Introduction

Throughout this document, the term Broadcom® NetXtreme II® refers to the Broadcom BCM57710/BCM57711 Dual Port 10-Gigabit Ethernet Controllers. The BCM577XX 10-Gigabit Ethernet controllers refer to the Broadcom BCM57710 and the BCM57711 controllers. The reader is urged to view the Broadcom Open Source FreeBSD or Linux® driver as a code reference and companion to this document. The Firmware component of the BCM57710/BCM57711 is vital to the BCM57710/BCM57711 host driver and device functionality. Developers writing custom drivers must be aware of the required hardware and software interface data structures used in specific versions of driver firmware. Changes in the software and firmware are evident in the Open Source drivers. This document primarily covers the device layer two and layer three functionality in host driver software along with an overview of the BCM57710/BCM57711 hardware architecture.

This document includes data structures specific to the device firmware version 4.8.5. See Appendix B "Programming the Non-Volatile Memory" for an example of data structures common to the host driver and device firmware.

Functional Description

Broadcom NetXtreme II includes dedicated hardware and internal microprocessors to process frames. Four on-chip high-performance VLIW STate Optimized RISC Microprocessors (also known as STORMs) enable layer four (L4) and layer five (L5) offload features, including TCP segmentation, full TCP Offload processing, iSCSI offload, and RDMA offload. On the transmit path, the Broadcom NetXtreme II extracts data directly from application buffer space on the host, executes any relevant L5 protocol such as the TCP/IP protocol, adds the required TCP/IP, iSCSI 1.0, or RDMA headers and then transmits the data over the physical medium. On the receive path, the Broadcom NetXtreme II processes the frame up to the highest supported layer. It removes lower-level headers from the incoming frames received from the physical interface and optionally places the data directly to application buffer space, again sparing host CPU resources that would otherwise be required for copying data between user buffers and kernel buffers.

For iSCSI functionality, the Broadcom NetXtreme II provides a hardware offload for the most host-CPU-intensive tasks. For the transmit data path, the Broadcom NetXtreme II adds framing support and calculates the header and data digest. On the receive data path, the Broadcom NetXtreme II strips the framing headers and checks CRC, then stores the data in the designated iSCSI buffers. Broadcom NetXtreme II builds iSCSI command PDUs and processes iSCSI response PDUs, offloading the host CPU from this task. Broadcom NetXtreme II DMA data directly to/ from application buffers, sparing host CPU resources that would otherwise be required for copying data between user buffers and kernel buffers.

The Broadcom NetXtreme II also supports other major features like Receive Side Scaling (RSS) for short-lived TCP connections, guaranteed delivery of management packets, RemotePHY™ functionality, and advanced congestion management. RSS allows for more balanced load sharing in an SMP server, handling a load of short-lived connections. The guaranteed delivery of management packets provides QoS for management traffic and allows management applications running on Remote Management Console (RMC) to talk to BMC even when there is congestion because of CPU inability to service faster than the incoming traffic. On the transmit side, management traffic coming from BMC takes strict priority over data coming from the host. The RemotePHY feature allows the local SerDes to interface to a Remote 10/100/1000BASE-T copper PHY via a backplane and to control and configure the RemotePHY in-band using the SerDes interface supporting the Broadcom Autonegotiation Mode (BAM) which utilizes the IEEE 802.3 next page capabilities in Clause 37. The advanced congestion management functionality like Service Aware Flow Control (SAFC) allows flow control and rate limiting per a given Class of Service.

The integrated 1-Gbps and 10-Gbps MACs are IEEE 802.3-compliant and support 802.1Q VLAN tagging, 802.1p layer 2 priority encoding, and 802.3x full-duplex flow control. The integrated transceivers are fully compatible with the IEEE 802.3

standard for autonegotiation of speed. Additionally, the device supports real-time tracing, loopbacks, and extensive statistics for debugging and diagnostic purposes.

SUPPORTED DEVICES

Broadcom Devices with the following PCIE Device IDs are the focus of this document. The PCIE Device ID value is found in PCIE Configuration space at offset 0x2, or through the GRC address space at PCIE_REG_PCIER_CFG_DEVICE_ID - (Offset: 0x2002; Width: 16).

Table 1: Supported Devices

Device	*PCI Device ID*
BCM57710	0x164e
BCM57711	0x164f

ABBREVIATIONS AND DEFINITIONS

Table 2 shows abbreviations and definitions used in this document.

Table 2: Abbreviations and Definitions

Abbreviation	*Definition*
ACPI	Advanced Configuration and Power Interface
BCN	Backwards Congestion Notification
BMC	Baseboard Management Controller
CAM	Content Addressable Memory
CID	Connection ID
CMOS	Complimentary Metal Oxide Semiconductor
CNIC	Converged NIC
CRC	Cyclic Redundancy Check
DDP	Direct Data Placement
FIC	Fast Input Channel
FOC	Fast Output Channel
GMII	Gigabit Media Independent Interface
GRC	Global Register Controller
GSO	Giant Send Offload
JTAG	Joint Test Action Group
HBA	Host Bus Adapter
IEEE	Institute of Electrical and Electronics Engineers
iSCSI	internet Small Computers System Interface
iSER	iSCSI Extensions for RDMA
IF	Interface
IPG	Inter Packet Gap

Table 2: Abbreviations and Definitions (Cont.)

Abbreviation	*Definition*
IPMI	Intelligent Platform Management
ILAN	Interface Local Area Network
LCID	Local Connection ID
LFSR	Linear Feedback Shift Register
LOM	LAN On Motherboard
LSO	Large Send Offload
MAC	Media Access Control
MCP	Management Communication Processor
MDIO	Management Data Input Output
MIB	Management Information Base
MII	Media Independent Interface
MSI	Message Signal Interrupt
MSIX-X	MSI eXtended
MSS	Maximum Segment Size
NIC	Network Interface Card
OSI	Open Systems Interface
PBF	Packet Builder and Framer
PCS	Physical Coding Sublayer
PHY	Physical Interface
PMA	Physical Medium Attachment
QoS	Quality of Service
PDU	Protocol Data Unit
PCI	Peripheral Connect Interface
PCI-X	PCI-eXtended
PCIe	PCI Express®
RFE	Receive Front End
RDMA	Remote Direct Memory Transfer
RDMAC	RDMA Consortium
RDMAP	RDMA Protocol
RMC	Remote Management Console
RNIC	RDMA capable NIC
ROM	Read Only Memory
RSS	Receive Side Scaling
SAFC	Service Aware Flow Control
SerDes	Serializer and Deserializer
SNMP	Simple Network Management Protocol
Tuple	An ordered set of values.
TCP Tuple	A tuple consisting of the source IP address, destination IP address, source TCP port, and destination TCP port
TSO	TCP Segmentation Offload
ULP	Upper Layer Protocol
UMP	Universal Management Port

Table 2: Abbreviations and Definitions (Cont.)

Abbreviation	*Definition*
VLAN	Virtual LAN
WoL	Wake on LAN
XAUI	10 Gigabit Attachment Unit Interface
XFI	10 Gigabit small Form factor pluggable module Interface
XFP	10 Gigabit small Form factor Pluggable module
XGMII	10 Gigabit Media Independent Interface
XPAK	10 Gigabit Ethernet Pluggable module

Section 2: Hardware Architecture

THEORY OF OPERATIONS

Figure 1 on page 6 shows the major functional blocks and interfaces of Broadcom NetXtreme II BCM57710/BCM57711 dual port 10 Gbps Ethernet controller. Each port includes an integrated SerDes device supporting 1000/2500 BASE-X, 10GBASE-CX4, 10GBASEKX4, and XAUI network interfaces and IEEE 802.3-compliant 1/2.5/10 Gbps MACs. Both ports share common blocks that are tuned for independent packet flows. The device's DMA engine provides DMA transactions from host memory to the device local storage, and vice-versa. The Broadcom NetXtreme II provides a PCIe v2.0 and v1.1-compliant bus interface. The RX MAC moves packets from the PHY into the device internal memory. All incoming packets are checked against a set of rules and then categorized accordingly.

When a packet is transmitted, the TX MAC moves the data from internal memory to the PHY. Both flows operate independently of each other in full-duplex mode. The device implements 128 KB of internal on-chip receive buffer memory and 25 KB per port of internal on-chip transmit buffer memory for temporarily storing the data before it is moved in and out of Ethernet and PCIe interfaces. The internal receive buffer memory is also commonly referred to as big RAM or the Big Receive Buffer (BRB). The device arbiter controls the access to the on-chip buffer memory.

Four high-performance VLIW RISC processors are implemented at strategic places in the device architecture such that they can be used in conjunction with other hardware blocks for L2 processing and supporting L4 and L5 offload features, including TCP segmentation, full TCP Offload, and iSCSI offload. One other RISC processor, referred as the Management Control Processor (MCP), is implemented to execute the boot-code and the firmware supporting driver initialization and shutdown synchronization and advanced management functions like IPMI Pass-Through or UMP or NC-SI. All five RISC processors and the on-chip buffer memory are shared between the two Ethernet ports. For management the UMP interface and SMBus interfaces are supported to connect a BMC with the Ethernet controller. The UMP uses either MII or RMII signaling. The SMBus interface block provides a serial interface that operates at clock speed of up to 400 kHz.

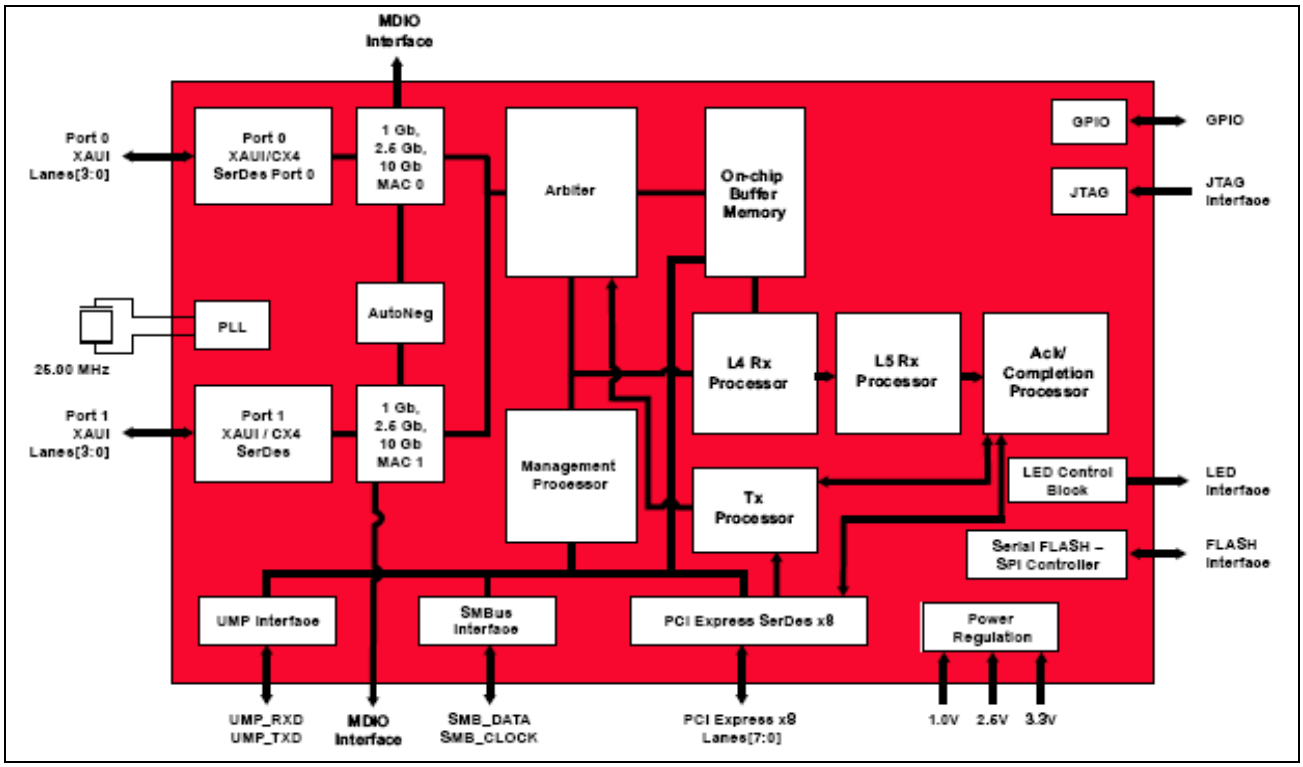

Figure 1: Functional Block Diagram

The BCM57710/BCM57711 device includes a Serial Flash Controller which implements the Serial Peripheral Interface (SPI) to connect to the serial flash devices which are used for storing the boot-code, device info, manufacturing info, and other firmware like PXE Option ROM image, UMP/NC-SI firmware, and IPMI firmware.

The MDIO interface block enables programming and control of the internal PHY (XGXS) and of the external PHY. The LED Control block implements the logic for controlling and driving various Ethernet link and PCIe link status signals. The GPIO block controls the four general purpose I/O pins and two special purpose I/O (SPIO) pins per port. Lastly, the JTAG block implements the IEEE 1149-1compliant JTAG support.

The Broadcom NetXtreme II includes dedicated hardware and processors that process the frames that traverse it and provide its functionality. On the transmit path, the Broadcom NetXtreme II copies the data directly from the highest hierarchy of buffers available (in prioritized order of application buffers, ULP buffers, TCP buffers) on the host; executes, when relevant, the L5 protocols; adds the iSCSI 1.0, or RDMA headers, followed by executing the TCP/IP and adding its headers, relieving the host CPU from these time-consuming operations. On the receive path, the Broadcom NetXtreme II processes the frame up to the highest layer of support present in the frame. Further, it removes the lower level headers from the frames it receives off the wire. It posts the data directly to application buffers, sparing host CPU resources that would otherwise be required for this activity.

For iSCSI, the Broadcom NetXtreme II provides hardware hooks for the most time-consuming tasks. On transmit, the Broadcom NetXtreme II copies data directly from the iSCSI buffers, and computes the header and data CRC when used. On receive, the Broadcom NetXtreme II strips off the framing headers and checks CRC prior to storing the data in the designated iSCSI buffers.

TCP-OFFLOAD

The TCP/IP protocol suite is used to provide transport (L4) services for a wide range of applications. File transfer protocols like CIFS and NFS, to name a few, utilize the services of TCP/IP. For a long time, this protocol suite was run on the host CPU consuming a very high percentage of its resources and leaving little resources for the applications themselves. The BCM57710/BCM57711 includes a TCP Checksum Offload (CSO) feature and TCP Segmentation Offload (TSO) features, also known as Large Send Offload (LSO) that saves host CPU cycles.

The Broadcom NetXtreme II provides an industry-first TCP offload that is carefully architected for integration with the operating system, unlike many of the standalone TCP offloads. In basic terms, the Broadcom NetXtreme II architecture allows the operating system to provide control and management functions (such as connection setup, prevention of denial-of service attack, system resource allocation, selection of the most promising TCP/IP flows for offload, error and exception handling). The Broadcom NetXtreme II fully owns the data processing of TCP/IP flows offloaded to it. The Broadcom NetXtreme II is fully compliant with relevant Internet Engineering Task Force (IETF) RFCs and fully supports modern TCP options like time stamp, window scaling, and so on. The implementation also provides flexibility and robustness against potential changes to TCP algorithms like congestion control by implementing this functionality in firmware, making changes easy to manage.

On transmit, the DMA engine fetches data from the host memory, segments it to the size allowed by the network (MSS or maximum segment size), formats the TCP, IP, 802.2 (Logical Link Control), and Ethernet headers and sends it on the wire. The TCP/IP context that resides on-chip is updated accordingly (all the TCP/IP state variables and timers). If the frame has not reached its destination, the Broadcom NetXtreme II retransmits it according to TCP protocol rules. On receive, every frame is parsed, and if it is a valid frame (i.e., it passes all checks for Ethernet, IP, and TCP frame format and consistency) and it belongs to one of the connections offloaded to the Broadcom NetXtreme II, it is processed for TCP/IP. The processing includes the complete TCP/IP protocol state variables, header removal for Ethernet, IP and TCP. The data is either placed in a temporary anonymous buffer similar to a software stack today or is placed directly at the buffer pre-posted by the application (for example, Zero Copy), if the application had posted such buffer in advance. The Broadcom NetXtreme II fully handles out-of-order reception, including placement of data in the buffers (Zero Copy).

The Broadcom NetXtreme II TCP Offload functionality allows simultaneous operation of up to 1 million fully offloaded TCP connections. The Broadcom NetXtreme II TCP Offload significantly reduces the host CPU utilization while preserving the rich and flexible nature of the soft implementation of the operating system stack. As the TCP control loop is shorter (ACK messages and other control functions handled by hardware), TCP data exchange becomes faster and more efficient and latencies of TCP operation are reduced.

Figure 2 on page 8 shows an example of the basic flow of application data through the networking stack on its way to the wire. This data flow begins with the application writing a block of data (e.g., 64 KB) to the sockets interface. The application may insert application layer headers in the data (e.g., every 8192 bytes) and split it into 8 messages or PDUs. The networking stack further splits this into TCP segments, typically 1500 bytes each (also the typical MTU setting for Ethernet frames) to fit into standard Ethernet frames. The TCP stack then processes TCP and IP and adds the TCP/IP headers into each packet. The packet information is then posted to the NIC's output queue, and the NIC fetches the data from the host memory, calculates checksum and Ethernet CRC, and places it into the NIC buffers for transmission. The NIC transmits the information to the wire and interrupts the CPU to signal completion of the transaction.

Figure 2 on page 8 also illustrates the differences between the existing technologies of checksum offload and TCP segmentation, and between TCP Offload and RDMA. With checksum offload, the NIC calculates and appends the TCP/IP checksum to each outgoing packet, offloading the CPU of this task. TCP segmentation copies the packet header from one packet to the next in a string of like packets, further offloading the CPU of the creation of each packet header when sending large segments of data. TCP Offload even further offloads the CPU by allowing the NIC to handle the tasks associated with reliable transport (windowing, sequence numbers, packet acknowledgements, and so on) and by completely processing the received segments, placing the data in the buffers instead of the host stack. RDMA provides the ultimate offload by allowing

the RNIC to communicate directly with the application by bypassing the kernel, taking application data, creating segments, and handling TCP/IP and lower-layer functions to move the data directly into the receiver's application space. The RDMA's built-in header and data separation also eliminates the need to strip the application/middleware header.

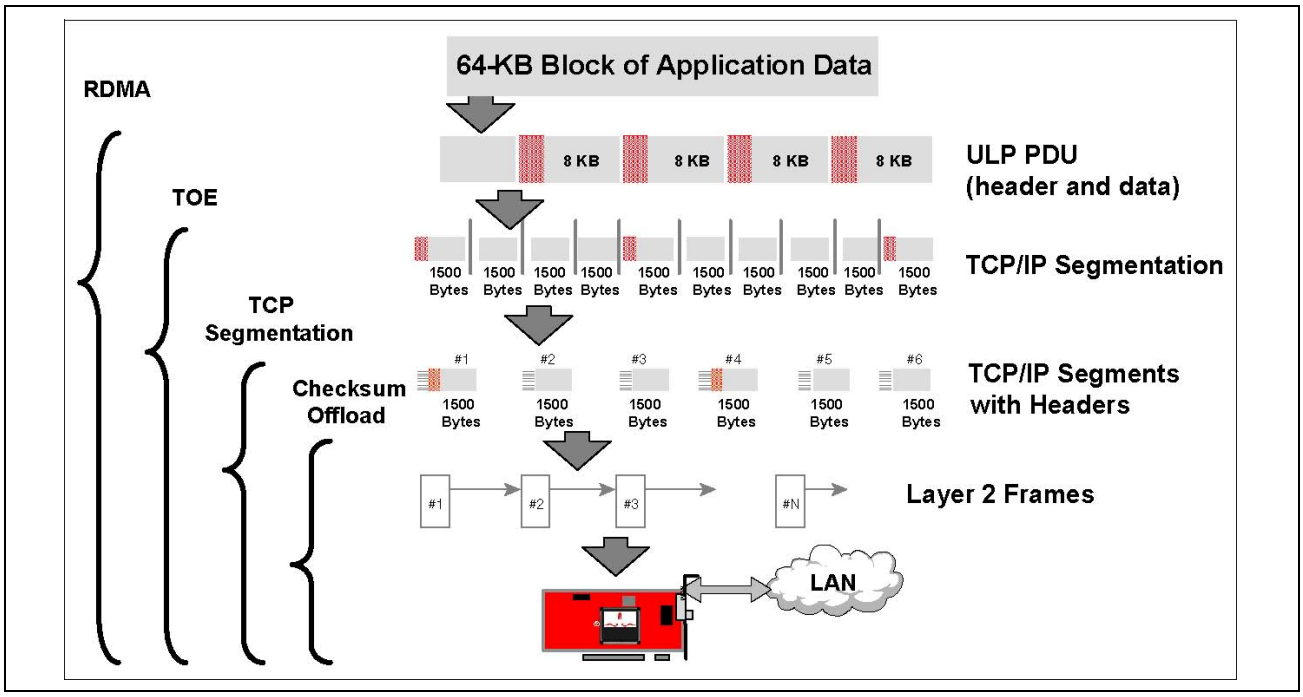

Figure 2: Data Flow From Application Through Wire

iSCSI Offload

The IETF has standardized the Internet Small-Computer Systems Interface (iSCSI). The SCSI is a popular family of protocols that enable systems to communicate with storage devices, using block-level transfer (i.e., address data stored on a storage device that is not a whole file, unlike file-transfer protocols such as NFS or CIFS). iSCSI maps the SCSI request/response application protocols and its standardized command set over TCP/IP networks. For further information on iSCSI, refer to RFC 3720.

As iSCSI utilizes TCP as its sole transport protocol, it greatly benefits from hardware acceleration of the TCP processing (such as when using a TCP Offload). However, iSCSI as a layer 5 protocol has additional mechanisms beyond the TCP layer. Figure 3 on page 9 shows the relationship iSCSI has to TCP and to the SCSI layer. As is custom in the SCSI family, the initiator requests certain operations (e.g., IO read or IO write) from the target in an end-to-end SCSI session using iSCSI.

The iSCSI layer adds several mechanisms above and beyond the TCP transport service. The iSCSI frame shown in Figure 4 on page 9 is an iSCSI PDU embedded as the payload of a standard TCP/IP frame. To further enhance the TCP checksum mechanism, iSCSI optionally uses header and data digest (such as CRC-32c). As iSCSI exposes the use of a named buffer to both the initiator and the target, it also facilitates for direct data placement into these named buffers (for example, zero copy). These mechanisms benefit from hardware acceleration.

The Broadcom NetXtreme II targets best-system performance, maintains system flexibility to changes, and supports current and future OS convergence and integration. Therefore, the Broadcom NetXtreme II iSCSI offload architecture is unique as evident by the split between hardware and host processing. Unlike a monolithic implementation on an HBA, the Broadcom

NetXtreme II focuses mainly on offloading to hardware the time-consuming elements poorly handled by software. A monolithic implementation offloads the complete iSCSI protocol to the HBA. This adds complexity to the HBA in order to process the complete control plane, which is better handled by the host CPU, and limits the integration options with a host operating system (such as when the control plane on the host provides for flexible implementation and robustness against updates to the iSCSI protocol, allowing for virtually no limit on number of outstanding commands and no limit on number of connections per session, and so forth). The Broadcom NetXtreme II accelerates in hardware the iSCSI mechanisms that impact system resource utilization and performance. It supports all of these mechanisms with the support of specially built hardware circuitry and/or firmware executing on-chip. The Broadcom NetXtreme II provides rich and complete iSCSI HBA functionality.

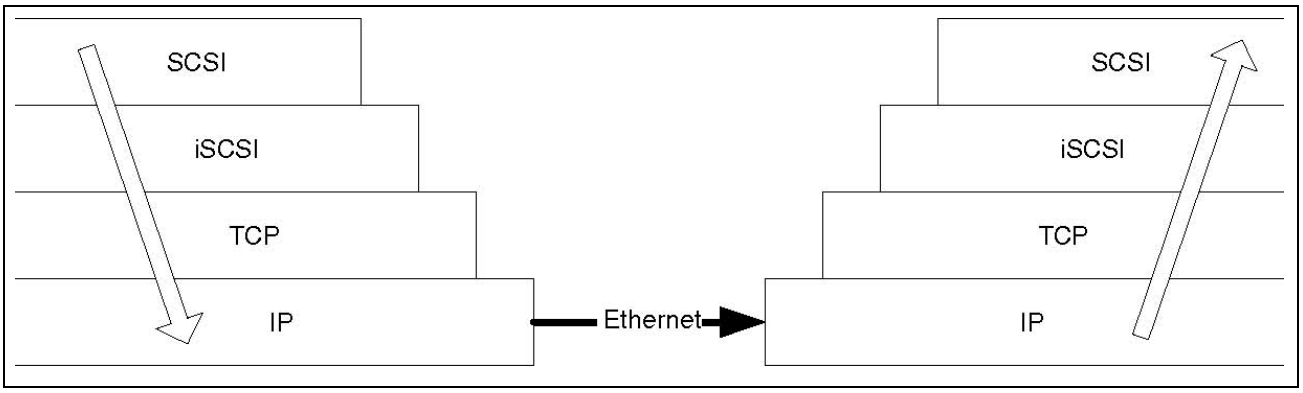

Figure 3: iSCSI Layers

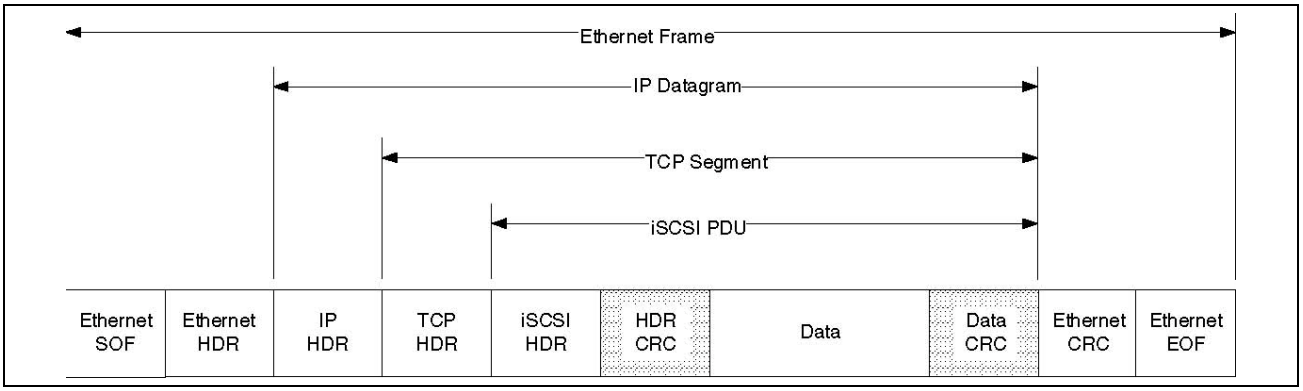

Figure 4: iSCSI Ethernet Frame Encapsulation

The Broadcom NetXtreme II supports the iSCSI frame format, header/data separation, insertion and checking of the iSCSI header and data CRCs, and direct data placement into the named iSCSI buffers. A deep command queue ensures optimal utilization of the network for maximal bandwidth and a high number of I/O Operations Per Second (IOPS) under significantly reduced CPU utilization.

Figure 5 on page 10 depicts a typical data flow between an iSCSI initiator and an iSCSI target. The Broadcom NetXtreme II can be the core of an initiator HBA. The initiator's SCSI driver passes SCSI Control Data Blocks (CDBs) to the iSCSI layer that sits atop the TCP/IP stack. On the target, a storage device (for example, a SCSI disk) sees SCSI commands and has no knowledge of the iSCSI layer. The SCSI disk interprets the SCSI commands, executes them, and sends appropriate data or replies to its local iSCSI layer, which transports them to the iSCSI on the initiator.

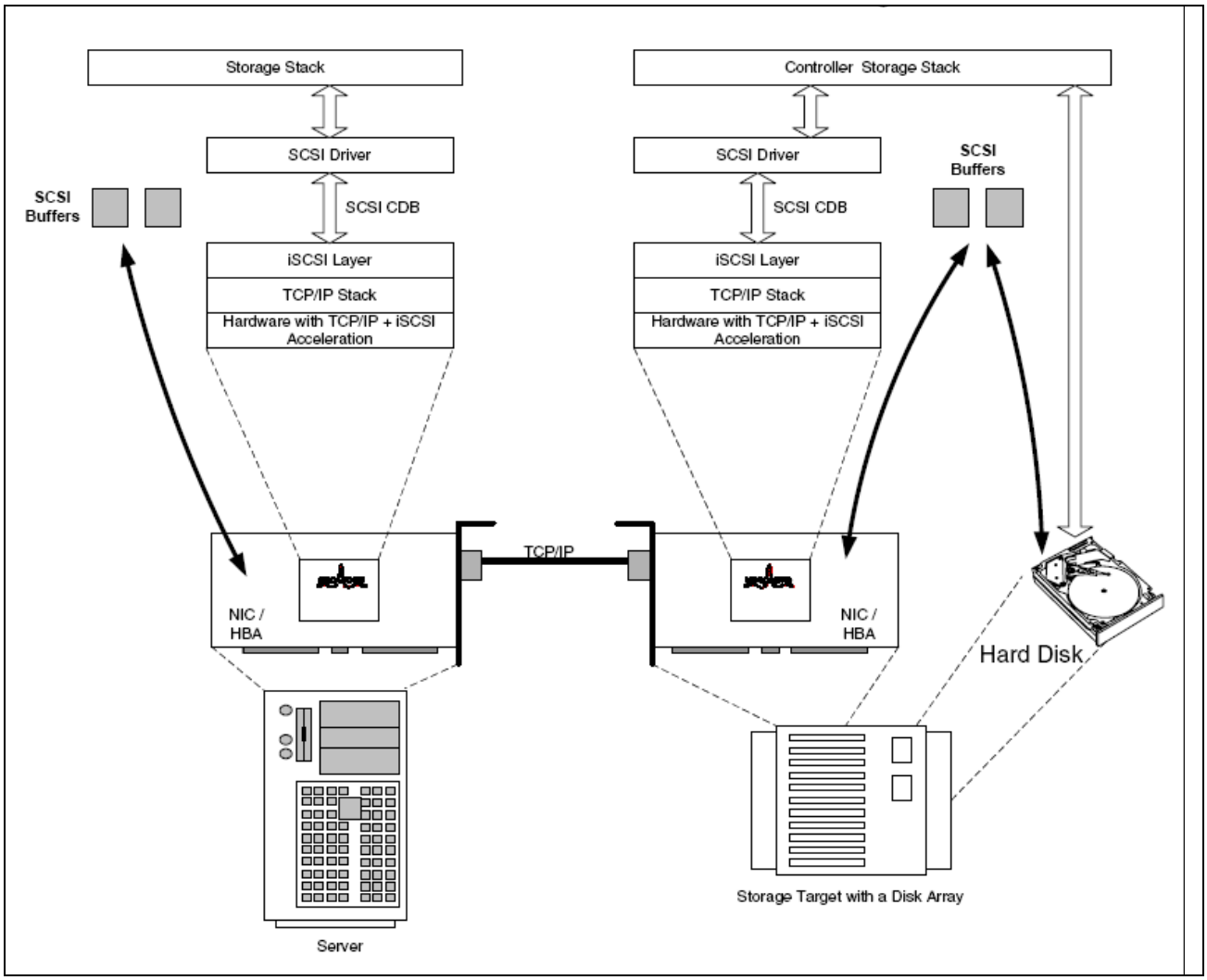

Figure 5: iSCSI Architecture

REMOTE PHY

The goal of traditional pass-through copper switch modules in a Blade Server environment is to provide the same or similar functionality as a local copper PHY. Presently, pass-through modules only support a Gigabit Interface Connection (GBIC) mode of operation. There are several limitations to this approach; The GBIC only supports functionality enabled by the autonegotiation base page exchange of Clause 37 of the IEEE 802.3; limited to duplex, and pause resolution. The speed of 1 Gbps is assumed. Extended features such as 10 Mbps and 100 Mbps, jumbo frame support, Ethernet at wire-speed, and so forth, are not supported and not possible.

Remote PHY addresses the need to provide a similar feature set as a local copper PHY. The remote PHY protocol is PHY vendor independent since all communication is handled via the extension of clause 37 of the IEEE 802.3 specification. Industry standard mechanisms are used to establish the link. The Remote PHY protocol uses the appropriate 1000BASE-KX signaling as defined in the IEEE 802.3ap specification for backplane operation.

The Broadcom Remote PHY SerDes Autonegotiation Mode (BAM) utilizes the IEEE 802.3 next page capabilities in Clause 37 to support the Remote PHY feature. The remote copper PHY features are abstracted by mapping these features onto the exchange of the next pages during the autonegotiation process. Explicit registers and the mapping of features are not required by the link partner. This lowers the cost of ownership as new features can be easily supported by additional capabilities and pages on the standardized autonegotiation exchange of pages. The Remote PHY feature set is supported in hardware in the Broadcom NetXtreme II with the possibility of expanding the feature set under firmware control.

The Broadcom NetXtreme II supports the Remote PHY feature where the SerDes interfaces to a remote 10/100/1000BASE-T copper PHY via a backplane. Since all the PHY management communication is performed in-band using the SerDes interface during autonegotiation, no out-of-band Management Data I/O (MDIO) bus is required. Remote PHY supports all standard functions of a 10/100/1000BASE-T copper PHY such as autonegotiation, forced speeds, Ethernet at wirespeed, and auto MDIX functions.

A remote copper PHY is a stand-alone copper PHY transceiver that connects to the local MAC via a backplane and communicates to another copper transceiver via category 5 (CAT5) copper cabling. The copper interface is governed by Clauses 14, 24, 28, and 40 of IEEE 802.3 for 10T, 100T, autonegotiation, and 1000T, respectively. The SerDes interface is governed by Clauses 36 and 37 of IEEE 802.3 for 1000-X as well as the Broadcom proprietary SerDes BAM.

BASIC OPERATION BETWEEN DEVICE AND REMOTE COPPER PHY

The remote copper PHY operates with two asynchronous autonegotiation processes running. One operates on the copper media and is based on clause 28 of the IEEE 802.3 standard. The other operates on the SerDes media or backplane and is based on Clause 37 of the IEEE 802.3 standard as well as the Broadcom proprietary SerDes BAM. There are some connections between the two autonegotiation processes that allow one to restart the other.

The copper interface will restart the SerDes interface under the following conditions:

- The copper interface receives advertised abilities from its link partner that are different than the advertised abilities transmitted on the SerDes interface. This includes 10,100, 1000T half duplex and full duplex, and pause settings.
- The copper link change (goes up or goes down)
- The copper autonegotiation changes from enabled to disabled and the current SerDes transmitted advertised abilities are different from the register 0x04 SerDes advertisement register.

The SerDes interface will restart the copper interface under the following conditions:

- The SerDes interface receives advertised abilities from its link partner that are different than the advertised abilities transmitted on the copper interface. This includes 10, 100 and 1000T half duplex and full duplex and pause settings.

- The SerDes autonegotiation changes from enabled to disabled and the current copper transmitted advertised abilities are different from the register 0x09 copper advertisement register.

The SerDes interface will exchange autonegotiation base pages between the MAC/switch and the RemotePHY. Following the base page, the next pages are exchanged.

The remote copper PHY updates all of its control settings from the MAC/Switch on entry to HCD/link (highest common denominator) determination. The settings are stored internally by the remote copper PHY until one of the following occurs:

- Another sequence of next pages is exchanged after restarting the SerDes autonegotiation for any reason. If the new next page exchange does not contain a MAC/Switch with remote copper capability, then the remote copper PHY register settings revert back to the default settings. If it is a MAC/Switch with remote copper capability, then any changes will be updated by the remote copper PHY.
- The local SerDes autonegotiation is disabled for any reason and detects valid idles from the serdes link partner (i.e. parallel-detection). The remote copper PHY register settings will revert back to the default settings.
- The RemotePHY SerDes link partner sends SGMII autonegotiation code words and the RemotePHY automatically changes to SGMII mode (if SGMII/GBIC auto-detection is enabled) and a base page is received without a selector mismatch. The remote copper PHY register settings will revert back to the default settings.

The simplified hardware architecture is shown in Figure 6.

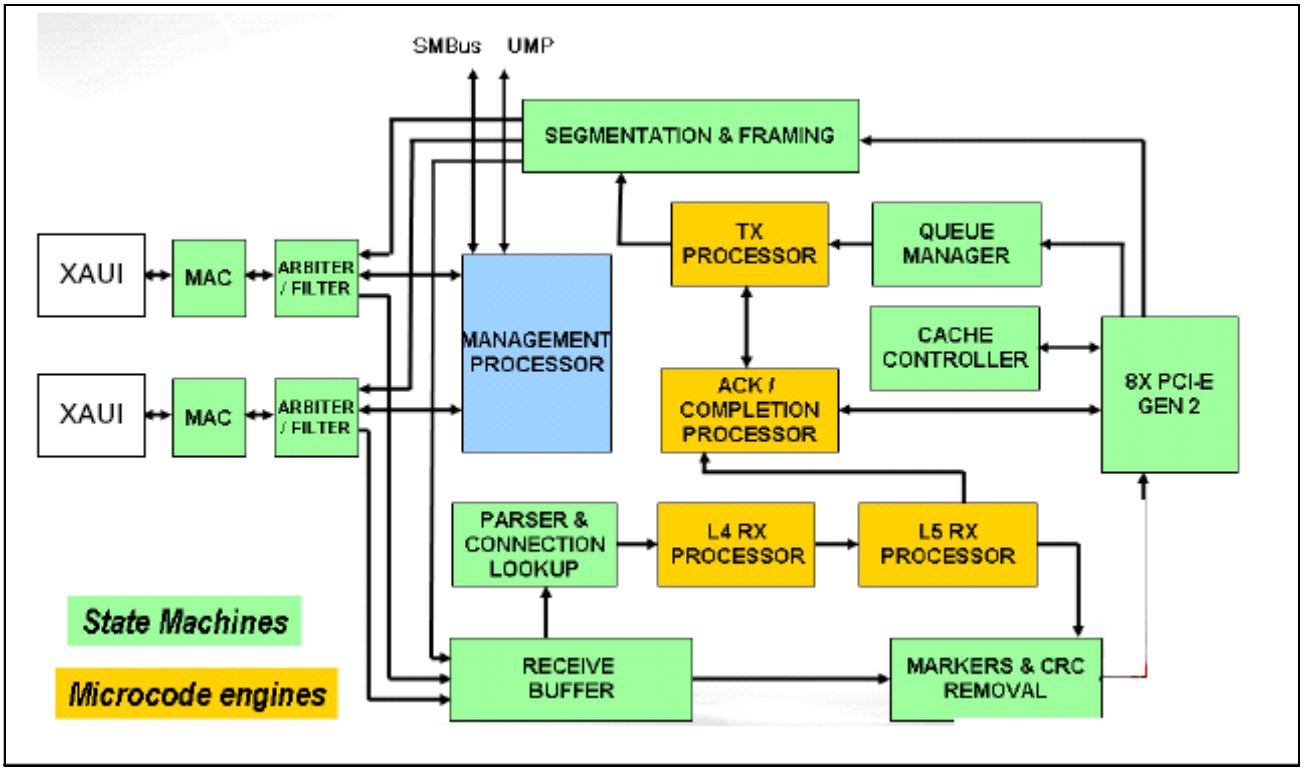

Figure 6: Simplified Hardware Architecture

SERDES

The BCM57710/BCM57711 integrates dual XAUI™/10GBASE-CX4/10GBASE-KX4 transceivers to support the 1-/2.5-/10-Gbps speeds of operation on both the ports. Each of the ports can be individually selected for XAUI or 10GBASE-CX4 or 10GBASE-KX4 or 1000BASE-KX or 2500BASE-KX mode of operation. The XAUI (10-gigabit attachment unit interface) is referenced in the IEEE 802.3ae specification and it consists of four lanes each running at 2.5 Gbps (3.125 Gbaud) for an aggregate bandwidth of 10 Gbps. A single lane of the XAUI can be configured to operate at either 1-Gbit or 2.5-Gbit data rates.

There are two Ethernet ports coincident with the two XAUI ports.

Figure 7 on page 14 illustrates the top level multiplexing of the two XAUI interfaces (Unicore x4 1000/2500). The two 1000/2500 BASE-KX SerDes interfaces shown are not used in the BCM57710 and are not connected in the BCM57711. The two Unicore x4 blocks are connected to two instances of the 10-Gbps MAC (also known as Big MAC) via two 2:1 multiplexers. The 2:1 multiplexers allow connecting any of the two Unicore blocks to any of the two Big MACs. When configured for 1/2.5 Gbps operation, the lane swap logic controls the two Unicore blocks so that a single lane can be controlled by the EMAC.

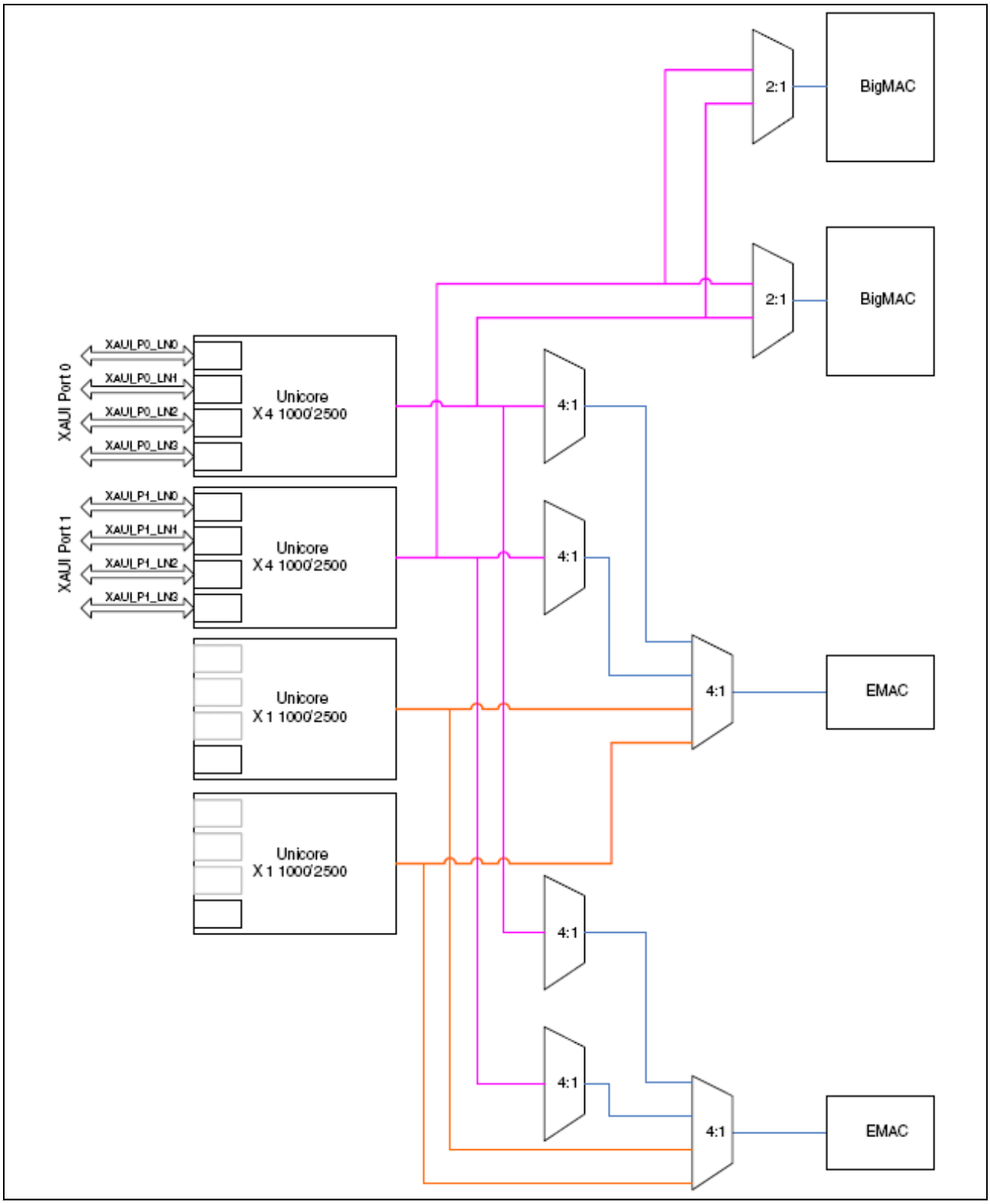

Figure 7: Multiplexing b/w XAUI and 1000BASE-KX/2500BASE-KX

MAC

As shown in Figure 7 on page 14, the BCM57710/BCM57711 implements one EMAC and one Big MAC for each of its two ports. The EMAC is used for 1-Gbps/2.5-Gbps operation and Big MAC is used for 10-Gbps operation. Both the EMAC and Big-MAC perform all of the following Ethernet MAC functionalities.

On the receive side:

- Adapts data presented at the frequency of the network clock to that of the system clock.
- Interprets /Idle/, /Start/, /Term/, and /Error/ special control characters.
- Recognizes the start and end of Ethernet packets.
- Recognizes packet framing errors.
- Strips the preamble from packets.
- Checks the FCS field for compliance to the IEEE CRC function.
- Optionally strips the FCS field from incoming packets.
- Eliminates all incoming packets less than 64 bytes in length.
- Truncates incoming packets of greater than a programmable size limit.
- Detects SAFC control frames and transfers them to the transmit processor.
- Eliminates all incoming MAC Control packets, or transfers them to the transmit processor for congestion management support.
- Asserts an error signal for all packets passed to the system on which errors are detected.
- Recognizes Pause packets and signals the transmit section to hold off data transfers as necessary.
- Detects Link Fault Sequences

On the transmit side:

- Synchronizes data from the system to the network clock.
- Encapsulates the data in a valid Ethernet packet
- Flags outgoing packets of less than 64 bytes in length with an error.
- Truncates outgoing packets of size greater than a programmable limit and flags them with errors.
- As a programmable option per packet, either appends a valid FCS to all packets presented by the system, or keeps the FCS unchanged, or replaces the FCS with a newly computed value.
- Generates and sends Pause packets as per IEEE 802.3 clause 31 either at the request of the system or upon receiving a signal from the receive section of MAC.
- Maintains packet statistics
- Transmits Link Fault Sequences

The EMAC also provides the PHY management interface via the MDC/MDIO pins for all 1-/2.5-/10-Gbps speeds. Each of the EMACs has an MDIO controller. The EMAC has to be out of reset for MDIO to work.

Receive Front End

The Arbiter/Filter, Receive Buffer, Parser and Connection lookup blocks shown in the Figure 6: "Simplified Hardware Architecture," on page 12 constitute the Receive Front End (RFE) of the device. The RFE module receives the traffic from the two ports and the internal loopback port, buffers the data, parses the packet headers, and searches for the connection in the connection database. All the sub-blocks of RFE module are described below.

Network Interface Glue

The Network Interface Glue (NIG) receives packets from two 10G MAC cores, and loop-back packets from the transmitter. This block synchronizes the packets from the MAC cores to the system clock, performs the bus adaptations, and forwards the packets to BRB. It is also responsible for counting packet length, discarding packets in case of backpressure, maintaining per port EOP (end of packet) information, and collecting statistics.

Arbiter/Filter

The arbiter/filter selects the port that gains the access to the BRB, decides whether an incoming packet should be written to the BRB, the Mgmt Rx Buffer or both, or (in the case of congestion management frames) to the transmit processor and selects between the Mgmt Tx packet, Host packet, and debug packet for transmission on the wire. It uses the Round Robin algorithm to provide fairness between all write clients to the BRB.

Big Receive Buffer

The Big Receive Buffer (BRB) is used to buffer packets (up to jumbo packet size) from each of the interfaces until they are parsed and validated by the Parser and the TCP processing engine. The BRB is also used to absorb temporary traffic bursts, where the input packet rate is higher than the device packet processing rate. The BRB is 128 KB in size and supports a 32-Gbps input bandwidth and 32-Gbps output bandwidth. It is logically divided in to 256-byte blocks that are managed by link-lists to optimize utilization.

Every incoming packet is written to the BRB as long as it has space for a new packet coming on a given port. Per port buffer space guaranteed can be configured, while the remaining buffer space is shared between the two network ports and traffic coming on the loopback interface from the transmitter. In parallel, the BRB sends a message to the Parser informing about the new packet and its address in the buffer. The BRB also initiates the Pause frame transmission when it is almost full and Pause is enabled.

Parser

For every incoming packet, after the header is received, the BRB sends a message to the Parser with the corresponding port number, the start block, Flush indication flag and the port tag. This message is queued until it can be handled by the Parser. The queue is global for all the ports. The parser is responsible for parsing the Ethernet, IPv4, IPv6, TCP and UDP headers, calculating the TCP/UDP checksum, calculating the MPA CRC assuming single aligned PDU and for building the messages to the L4 Rx Processor (aka TCP-Receiver). The header parsing includes:

- Ethernet address type recognition
- Validation of version and length fields
- Fragments recognition
- Validation of IP header checksum
- TCP options parsing for time-stamp

After the header parsing, messages are sent to the Context Fetch Controller (CFC). This unit is also commonly referred to as a Memory Management Unit (MMU) for connection information, and to the searcher to calculate Hash on the TCP/IP 4-tuple (for RSS). If the CFC cannot find the connection information in its internal cache, it requests the searcher to do a search for the connection information in host memory and to load the connection context to its internal cache. Note that the parser sends the Search message to the CFC only when the parsing is successful and the received packet is a TCP packet. After the responses are returned from the CFC and the searcher, the Parser builds a message to the TCP-Receiver RISC (aka TCP-Receiver STORM or L4 Rx Processor). The message mainly includes general parameters, header parsing information, TCP parameters, some context parameters from the CFC, and the RSS Hash result from the searcher.

For every non-fragmented TCP packet, the TCP checksum is calculated. In addition, for each TCP packet, the MPA CRC is also calculated simultaneously. The MPA CRC calculation is speculative and it assumes single aligned PDU in the TCP packet. When checksum and MPA CRC calculations are completed, and the EOP information is received from the NIG, the Parser generates a second message to the TCP-Receiver STORM, with the packet length, EOP info and the calculation results. The split of information to TCP-Receiver STORM into two messages is done in order to save latency by starting the TCP processing after the header is received but before the packet is fully received.

SEARCHER

The searcher is responsible for searching the connection ID of the received packet in the searcher database, in cases that it is not cached in the CFC cache. The hardware supports up to 1,000,000 connections in the searcher database which is located in host memory.

The search parameters are:

- IP source address
- IP destination address
- TCP source port
- TCP destination port

The searcher unit supports both IPv4 and IPv6, with different lengths search strings. In both cases it does search on a 4-tuple comprising of TCP source port, TCP destination port, IP source address, and IP destination address fields. The searcher supports add, delete and change commands that the host initiates and search commands that the CFC initiates. In both cases it receives the request from the CFC block and sends the response back to the CFC.

TSTORM (aka L4 Rx Processor or TCP Rx Processor)

The TSTORM is responsible for the TCP processing and for modifying the packet data prior to ULP processing (for example, header stripping and markers removal) for all the received packets. In the case of L4/L5 packets, it performs delineation, i.e. identifies the boundaries of the ULP PDUs according to the specific protocol rules. A message is sent to the USTORM when a complete PDU is ready, containing the ULP header and information about the PDU. In parallel, the TSTORM also removes markers and validates CRC. The TCP receiver sends complete PDUs to the ULP receiver even if they were not received in order. It also sends TCP information to CSTORM which uses that information to calculate the transmission window as well as data to be sent to the far end in the transmitted TCP packets. In the case of L2 packets, it performs classification of the packet into one out of up to 16 receive queues per port.

USTORM (aka L5 Rx Processor or ULP Rx Processor)

The USTORM receives the messages about new PDUs from TSTORM and is responsible for the ULP processing with the flexibility to support different ULP protocols such as RDMA and iSCSI. The ULP processing includes validation of the packet and identifying the address for placing the data over the PCI-E to the host memory. The data is moved from BRB to the calculated address allowing direct data placement to the application buffer without need for additional Host CPU copy. Pre-

fetching data as part of the context load and centralized caching techniques are used to place the data with minimum latency. The ULP processing is done even when the ULPs were received out of order.

XSTORM (aka Tx Processor)

The XSTORM is responsible for the ULP processing, TCP processing, and preparing a transmission command to the PBF (Packet Builder and Framer) unit for all transmit packets. This unit works in conjunction with the Doorbell Queue (DQ) and Timers blocks. The XSTORM is triggered by a doorbell that the host CPU rings with information about additional work. The doorbell address can be mapped to user space application allowing kernel bypass. The DQ processes the doorbell and updates the connection context (after loading the context thorough the cache controller). Decision rules are applied to decide whether the new information should trigger STORM processing (for example, checking the TCP transmission credit). If so, the connection ID enters a queue of connections that wait for processing. The XSTORM will process the connection according to TCP and the ULP protocol of the connection. The XSTORM will then build and send a command to PBF with the information of transmit packet data location in the host memory and how to build ULP and TCP/UDP headers.

CSTORM (aka Ack/Completion Processor)

The CSTORM is responsible for receiving the TCP acknowledge updates received from the far end, calculating the TCP transmission window updates for the transmit unit, calculating which host work requests are completed, and sending completion notifications to the host accordingly. The CSTORM is triggered by the TSTORM upon receiving the TCP acknowledge updates. The CSTORM does TCP processing for calculating the transmission window and updates the Transmitter unit. The CSTORM also sends messages to the transmitter for retransmission and for initiating the necessary TCP protocol related timers. The CSTORM also notifies the Host about completion of tasks (for example, when confirming that data of the write RDMA request was received by the far end) using the appropriate notification mechanism according to the protocol (for example, when generating Completion Queue Element in RDMA and iSCSI, or advancing the transmit ring consumer pointer in L2 and TOE).

SEGMENTATION AND FRAMING UNIT (AKA PACKET BUILDER AND FRAMER-PBF)

The PBF reads data from host buffers through PCIe, segments it to upper layer PDUs such as RDMA or iSCSI and TCP segments, builds the headers, inserts the headers, markers and CRC, and finally transmits the packets. The PBF uses the per connection context only for CRC calculation in parts. This is required for larger than TCP window ULP messages. The PBF also implements the following functions.

- TCP and UDP checksum offload with any size of TCP/IP/Ethernet header of up to 254 bytes.
- Automatic TCP segmentation of large TCP packets when the TCP/IP/Ethernet header size is limited to 254B. This automatic segmentation is also supported for the IPv6 header without options.
- Setting TCP Push flag on the last segment of a PDU.
- Automatic segmentation mode for segmentation of iSCSI PDUs and TOE segments. Single command will generate multiple segments.
- Automatic RDMA DDP slicing in hardware. In this mode the PBF will duplicate the ULP header and advance the offset between the different ULP slices (DDPs).

MARKER AND CRC REMOVAL (AKA ULP PACKET BUILDER-UPB)

The ULP packet builder is a generic packet manipulation unit. It can be used to verify CRC and remove markers, VLAN tags, and CRC from the incoming network traffic. This block is used by the USTORM.

PCIe

The PCIe bus is a high-bandwidth serial bus providing a low-pin count interface. The PCIe block implements the x8 PCIe end-point interface and fully complies with PCIe specification v1.1. The PCIe block is also compliant with PCIe v2.0. The PCIe block is responsible to serve memory access requests to the external host memory through PCIe bus. These can be simple read/write requests or Context-Fetch requests. This block supports configurable address mapping for different functionalities (e.g. doorbells and communication with STORMs). This block can be also used for intra-chip communication between STORM processors. Refer to PCIe section of this document for more details on the PCIe interface and supported features.

Management Control Processor

The Management Control Processor (MCP) is an on-chip high performance RISC processor which executes the boot code for chip initialization and configuration and any optional management firmware to support IPMI Pass-Through or UMP. On every chip reset (soft reset or PCIe reset), the MCP will execute on-chip ROM code which loads the boot code from an external NVRAM to the on-chip MCP scratchpad memory and starts executing the boot code. The boot code initializes and configures some of the chip parameters for proper operation. If the device advanced management feature is enabled, the boot code will also load the management firmware from the external NVRAM and starts executing it. The IPMI Pass-Through firmware uses the SMBus interface to provide a pass-through interface to the BMC on the mother board. For UMP or NC-SI, a 10/100 Mbps MAC is used and MII/RMII interface is supported for connection to BMC. The MCP also supports DMA operations for data transfers between the UMP MAC FIFOs and the Ethernet MAC FIFOs. There are reserved buffers for management packets to/from the external network and this ensures guaranteed delivery of management packets even when there is congestion because of Host traffic.

Device Address Space

This section provides the details of Host memory map and the internal MCP memory map of BCM57710/BCM57711 Ethernet Controller.

Host Bar Memory Map

The BCM57710/BCM57711 Ethernet Controllers support two BARs (see Figure 8 on page 22). The BAR0 memory map includes all the device registers and memory space while the BAR1 memory map is the doorbell address space which depends on the number of connections. Table 3 shows the BAR0 memory map. The GRC space of 4087 KB is expanded in Table 4 on page 20. See the BCM57710/BCM57711 Register Definitions Document for details on various registers supported in the BCM57710/BCM57711 Ethernet Controller.

Table 3: BCM57710/BCM57711 BAR0 Memory Map

Region	*Size*	*Address*
Reserved	8 KB	0x000000–0x001FFF
PCI configuration space (1)	1 KB	0X0020000–0X0023FF
Global Register Control (GRC) space (1)	4087 KB	0x002400–0x3FFFFF
USTORM internal memory (2)	64 KB	0x400000–0x40FFFF
CSTORM internal memory (2)	64 KB	0x410000–0x41FFFF
XSTORM internal memory (2)	64 KB	0x420000–0x42FFFF
TSTORM internal memory (2)	64 KB	0x430000–0x43FFFF

Table 3: BCM57710/BCM57711 BAR0 Memory Map

Region	*Size*	*Address*
IGU internal memory (2)	64 KB	0x440000–0x44FFFF
Reserved	3.68 KB	0x450000–0x7FFFF

Table 4: BCM57710/BCM57711 ***GRC Space Register Offsets***

Region	*Starting Address*
Reserved	0x000000
PCI registers	0X002400
EMAC0 registers	0x008000
EMAC1 registers	0x008400
DBU registers	0x008800
MISC/AEU registers	0x00A000
DBG registers	0x00C000
NIG registers	0x010000
XCM registers	0x020000
PRS registers	0x040000
SRCH registers	0x040400
TSDM registers	0x042000
TCM registers	0x050000
BRB registers	0x060000
MCP registers	0x080000
UPB registers	0x0C1000
CSDM registers	0x0C2000
USDM registers	0x0C4000
CCM registers	0x0D0000
UCM registers	0x0E0000
CDU registers	0x101000
DMAE registers	0x102000
PXP registers	0x103000
CFC registers	0x104000
HC registers	0x108000
PXP2 registers	0x120000
PBF registers	0x140000
XPB registers	0x161000
Timer registers	0x164000
XSDM registers	0x166000
QM registers	0x168000
DQ registers	0x170000
TSEM registers	0x180000
CSEM registers	0x200000
XSEM registers	0x280000

Table 4: BCM57710/BCM57711 ***GRC Space Register Offsets***

Region	*Starting Address*
USEM registers	0x300000

See the "BCM57xx Register Definitions Document" for a detailed description of the registers available through this space.

Note: The layout of internal memory space is defined by firmware running on the appropriate processor. This information is not defined in this document and is subject to change.

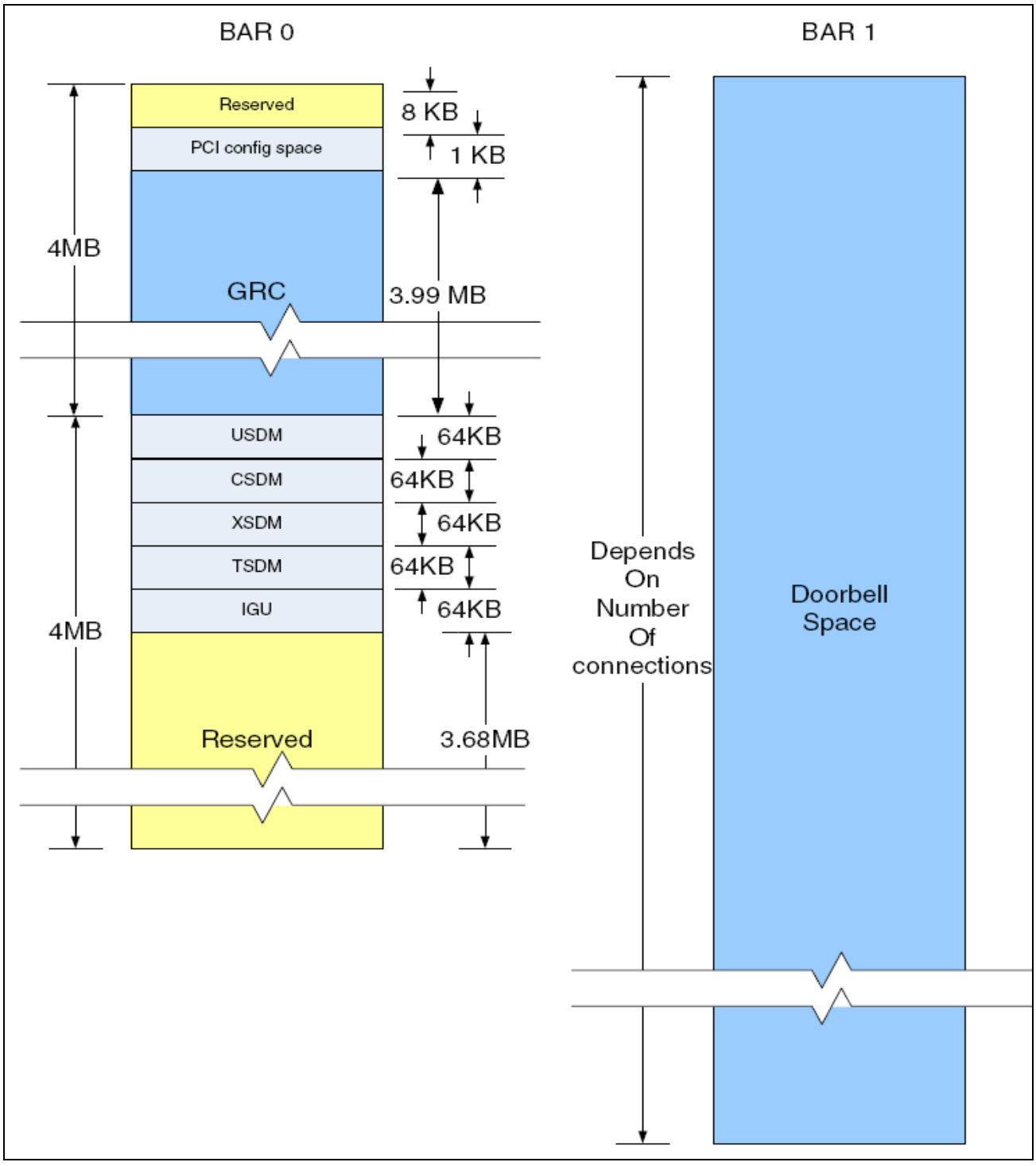

Figure 8: BCM57710/BCM57711 BAR Memory Space

MCP MEMORY MAP

The MCP memory map is shown in Figure 9. The MCP scratch pad in the BCM57710/BCM57711 Ethernet Controller starts at 0x08000000 and its size is 64 KB, so the unused space following the MCP scratch pad starts at 0x08010000.

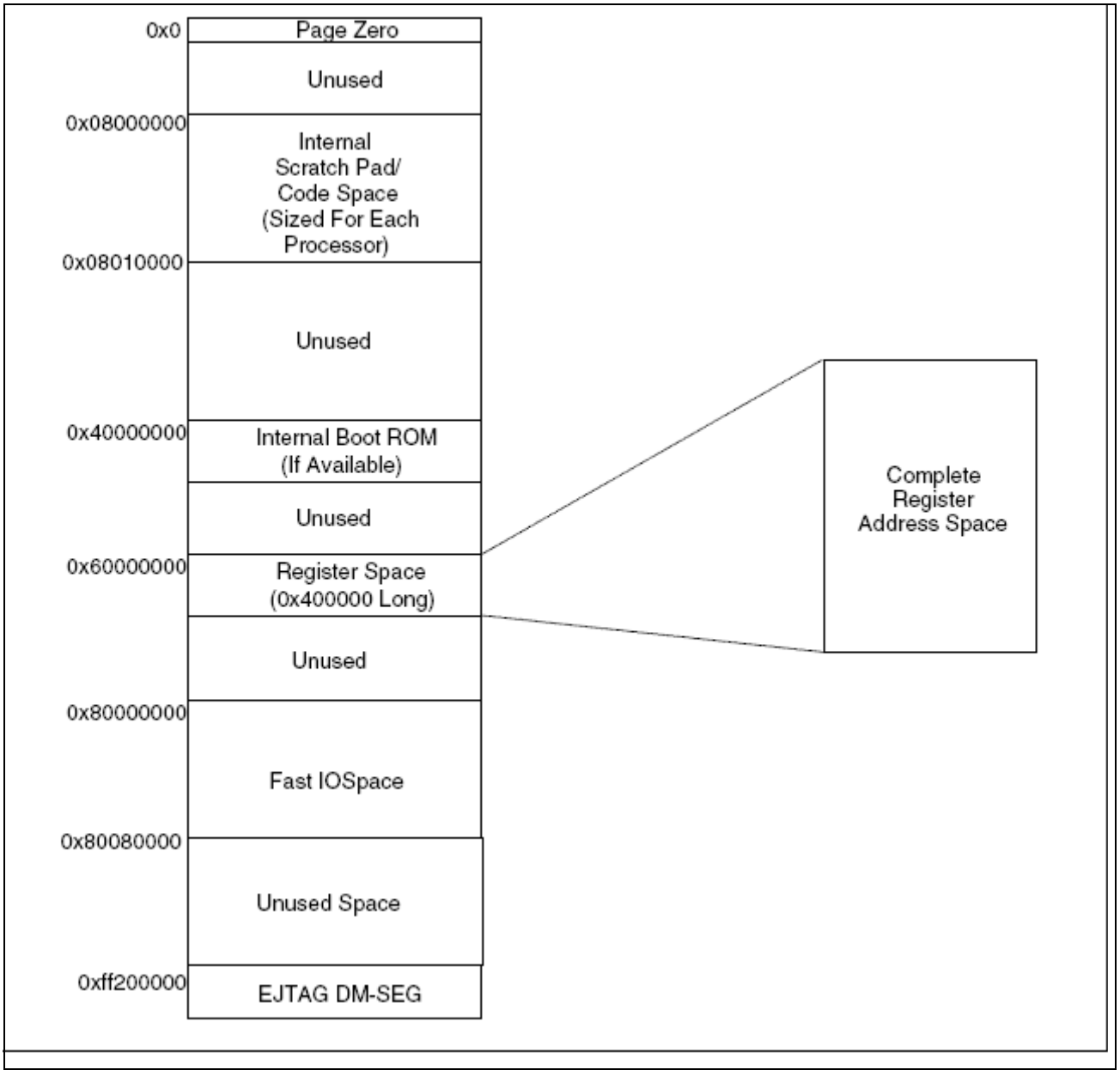

Figure 9: MCP Memory MAP

The MCP can access the device registers through the Register space, which starts at 0x60000000 and ends at 0x603FFFFF. The Unused space following the Register space starts at 0x60400000. The Fast IO space starts at 0x80000000 and ends at 0x8007FFFF.

Section 3: NVRAM Configuration

NVRAM MAP

The NVRAM contains the following information:

- Boot strap
- Code directory
- Manufacturing Information
- Feature configuration information
- VPD-R information
- Licensing information
- Programs (PXE or management firmware images)

The base address for these regions is described in Table 5.

Table 5: NVRAM Map

Base Address	*Description*
0x0000: 0x0013	Boot strap
0x0014: 0x00FF	Code directory
0x0100: 0x044F	Manufacturing Information
0x0450: 0x053F	Feature Configuration Information
0x0540: 0x063F	VPD
0x0640: 0x0817	Licensing Information (384*2 ports = 768 bytes)
0x0818: TBD	Program Images (Bootcode, PXE, IPMI/UMP firmware, and so on)
–	Reserved

Note: NVRAM parameters are stored in Big Endian format.

Table 6: Boot Strap Region

Offset	*Size (Bytes)*	*Description*
0x00	4	Magic number 0x669955AA that is used by the ROM code to identify valid boot strap.
0x04	4	CPU scratch pad physical address where boot strap will be copied to and execute.
0x08	4	Boot strap code len in DWORD. This length includes the 4-byte CRC-32 stored at the end of the boot strap code. This is the length of first phase boot code when 2phase boot code is used. The length of second phase bootcode is available in the header at the beginning of second phase bootcode.
0x0C	4	Offset within NVRAM where boot strap code is found. This should be a physical address as ROM loader code operates only on physical addresses.
0x010	4	CRC-32 of the Boot Strap header (offset 0x00 to 0x0F inclusive). This field should be updated with correct CRC-32 value whenever any of the fields in Boot Strap region is changed.

There is up to 1 KB of ROM code in the Broadcom NetXtreme II family that is executed by the Management Processor (MCP) when the device is removed from reset. The major function of this ROM code is to load the initial boot strap from the NVRAM to the scratch pad of the MCP and execute it.

CODE DIRECTORY

The code directory is a table of executable binaries. Each binary may be loaded onto one of the embedded RISC processors or it may be used by the host CPU. The executable images are located in the NVRAM map region (see "Program Images" on page 43). The code directory region contains 16 directories of code with 12 bytes in each directory.

Table 7: Code Directory Region

Offset	*Size (Bytes)*	*Description*
0x14	4	Directory #0 Code SRAM address, relative to the designated CPU (see attribute field at offset 0x18).
0x18	4	Directory #1 image type, attributes, and length:
		Bit 31-28: Image Type. Following values for image types have been defined. • 0x0: BC2 (2nd phase bootcode) • 0x1: MBA (multiple boot agent host software) • 0x3: UMP (Universal management port firmware) • 0x8: IPMI_INIT (intelligent platform management initiative initialization firmware) • 0x9: IPMI_SERV (IPMI service firmware) • 0xD: ISCSI_BOOT (iSCSI Boot firmware) • 0xE: BC1 (1st phase bootcode) • Others: Reserved Bit 27-24: Target CPU • 0x0: None • 0x1: Host • 0x2: MCP • 0x3–0xf: Reserved
		Bit 23–2: Indicates the length of the image (plus the 4-byte CRC-32). It implies that the maximum size per directory entry is 16 MB.
		Bit 1–0: Reserved
0x1C	4	Directory #1 NVRAM Offset to image
0x20	12	Code Directory #2
0x2C	12	Code Directory #3
0x38	12	Code Directory #4
0x44	12	Code Directory #5
0x50	12	Code Directory #6
0x5C	12	Code Directory #7
0x68	12	Code Directory #8
0x74	12	Code Directory #9
0x80	12	Code Directory #10
0x8C	12	Code Directory #11
0x98	12	Code Directory #12
0xA4	12	Code Directory #13
0xB0	12	Code Directory #14

Table 7: Code Directory Region (Cont.)

Offset	*Size (Bytes)*	*Description*
0xBC	12	Code Directory #15
0xC8	12	Code Directory #16
0xD4	20	Reserved
0xE8	20	Spare part number (similar to the PN field in the VPD)
0xFC	4	CRC-32 of the code directory (offset 0x14 to 0xFB inclusive)

MANUFACTURING INFORMATION

Table 8: Manufacturing Information

Parameter	*Bytes*	*Address*	*Address (hex)*	*Comments*
Format revision	1	256	0x0100	Manufacturing format revision in ASCII. The current version is 'A'.
Reserved	1	257	0x0101	Reserved
Length of manufacturing information	2	258	0x0102	Length of manufacturing information. It is set to 0x400.
Shared Hardware Configuration	Bytes	Address	Offset	Structure is described below.
Part Number	16	260	0x0104	This field is identical to PN in VPD-R region.
Configuration	4	276	0x0114	Bit 0:Mdio_voltage 0 – 1.2V 1 – 2.5V MDIO voltage 1.2V (default) Bit 1: Reserved Bit 2:Port swap 0 – Disabled 1 – Enabled Port swap 0 – do not swap (default) 1 – swap Bootcode will check port swap SPIO and XOR the value of this parameter with the value read from SPIO (set by BIOS according to BIOS menus) and sets the XOR result to the NIG register Reason: There two scenarios for port swap: 1. Statically rename port 0 and port 1 - a board parameter 2. Disable function 0 on BIOS menu. In this case, PCI function 1 becomes the new function 0. These two scenarios can co-exist, which is the reason for the XOR.

Table 8: Manufacturing Information (Cont.)

Parameter	*Bytes*	*Address*	*Address (hex)*	*Comments*
config				Bit 3:Beacon WOL_Enabled 0 – Disabled 1 – Enabled Beacon in WoL 0 – Disabled (default) 1 – Enabled Bits 7-4:reserved Bits 10-8: Management Firmware selection type 0 – Default 1 – NC_SI 2 – UMP 3 – IPMI 4 – SPIO4_NC_SI_IPMI 5 – SPIO4_UMP_IPMI 6 – SPIO4_NC_SI_UMP MFW select 0 – Default (default) 1 – NC-SI 2 – UMP 3 – IPMI 4 – SPIO4 (0-NC-SI, 1-IPMI) 5 – SPIO4 (0-UMP, 1-IPMI) 6 – SPIO4 (0-NC-SI, 1-UMP) 7 – Reserved Bits 15–11: reserved

Table 8: Manufacturing Information (Cont.)

Parameter	*Bytes*	*Address*	*Address (hex)*	*Comments*
config				Bits 19-16:Led Mode 0 – MAC1 1 – PHY1 2 – PHY2 3 – PHY3 4 – MAC2 5 – PHY4 6 – PHY5 7 – PHY6 8 – MAC3 9 – PHY7 10 – PHY9 11 - PHY11 12 - MAC4 13 - PHY8 block LED mode register: 0 – MAC1 1 – PHY1 2 – PHY2 3 – PHY3 4 – MAC2 5 – PHY4 6 – PHY5 7 – PHY6 8 – MAC3 9 – PHY7 10 – PHY9 11 – PHY11 12 – MAC4 13 – PHY8 14 – PHY1 (unused) 15 – PHY1 (unused)
config				Bits 23-20:reserved Bits 29-24:Autonegotiation type Enabled bitmask (6 bits) Bit 24:CL37 Bit 25:CL73 Bit 26:BAM Bit 26:Parallel Detection Bit 28:SGMII Fiber Auto Detect Bit 29:RemotePHY Bit 0: CL37 AN enable mask Bit 1: CL73 AN enable mask Bit 2: BAM enable mask Bit 3: Parallel detection mask Bit 4: SGMII / fiber auto detect mask Bit 5: Remote PHY enable mask Note: If 0, not used when AN is enabled Bits 31-30:reserved

Table 8: Manufacturing Information (Cont.)

Parameter	*Bytes*	*Address*	*Address (hex)*	*Comments*
config2	4	280	0x0118	Bits 7–0: Reserved. Bit 8: PCI Gen2 0 – Disabled 1 – Enabled Enables PCIE gen 2. This option is not applicable to BCM57710. Bits 11-9: reserved Bit 12: SMBus Timing 0 – 100 kHz 1 – 400 kHz SM Bus timing configuration: 0 – 100 kHz 1 – 400 kHz Bit 13: Hide port1 0 – Disabled 1 – Enabled Bit 14: reserved Bit 15: SPIO4 follows PERST 0 – Disabled 1 – Enabled Use SPIO4 as output pin to follow PERST assertion (output low if PERST is asserted)
config2				Bits 18 –16: PCIE Gen2 preemphasis 0 – HW 1 – 0dB 2 – 3_5dB 3 – 6_0dB Set PCIE Gen 2 Preemphasis. This option is not applicable to the BCM57710. Bits 20-19: Fan Failure Enforcement 0 – PHY type 1 – Disabled 2 – Enabled The fan failure mechanism is usually related to the PHY type since the power consumption of the board is determined by the PHY. Currently, a fan is required for most designs with SFX7101, BCM8727 and BCM8481. If a fan is not required for a board which uses one of these PHYs, this field should be set to Disabled. If a fan is required for a different PHY type, this option should be set to Enabled. Bits 22–21:ASPM Support 0 – L0s L1 enabled 1 – L0s disabled 2 – L1 disabled 3 – L0s L1 disabled ASPM Power Management support Bits 31–23:reserved

Table 8: Manufacturing Information (Cont.)

Parameter	*Bytes*	*Address*	*Address (hex)*	*Comments*
power_dissipated	4	284	0x011C	Bits 15-0: reserved Bits 23–16: Power management scale 0 - Unknown 1 – 0.1W 2 – 0.01W 3 – 0.001W Bits 31–24: Power dissipated by common logic
ump_nc_si_config	4	288	0x0120	Bits 1–0: MII mode 0 – MII MAC 1 – MII PHY 2 – RMII UMP/NCSI MII mode Bits 7–2:reserved Bits11–8: Reserved Bits 31–12:reserved
board	4	292	0x0124	Bits 15–0: reserved Bits 23–16: Board revision Bits 31–24: Board version
Reserved	4	296	0x0128	
Port Hardware Configuration	**Bytes**	**Address**	**Offset**	**Structure is described below**
pci_id	4	300	0: 0x012C 1: 0x02BC	Bits 15–0: Vendor Device ID PCI Device ID MCP updates this field to PCIE-IP Bits 31–16:Vendor ID PCI Vendor ID MCP updates this field to PCIE-IP
pci_sub_id	4	304	0: 0x0130 1: 0x02C0	Bits 15–0:Subsystem Vendor ID PCI Subsystem Vendor ID MCP updates this field to PCIE-IP Bits 31–16:Subsystem Device ID PCI Subsystem Device ID MCP updates this field to PCIE-IP
Power Dissipated (D3:D2:D1:D0)	4	308	0: 0x0134 1: 0x02C4	Note that data scale is at 0.1. MCP updates this field to PCIE-IP
Power Consumed (D3:D2:D1:D0)	4	312	0: 0x0138 1: 0x02C8	Note that data scale is at 0.1. MCP updates this field to PCIE-IP

Table 8: Manufacturing Information (Cont.)

Parameter	*Bytes*	*Address*	*Address (hex)*	*Comments*
MAC Address	8	316	0: 0x013C 1: 0x02CC	Primary MAC address. The upper two bytes are unused and must be 0.
iSCSI MAC Address (Upper 16 bits are unused)	8	324	0: 0x0144 1: 0x02D4	The iSCSI MAC address. The upper two bytes are unused and must be 0.
RDMA MAC Address (Upper 16 bits are unused)	8	332	0: 0x014C 1: 0x02DC	The RDMA MAC address. The upper two bytes are unused and must be 0.
SerDes configuration matrix for forced/AN mode (Tx pre-emphasis:Rx equalizer) (16 bits each)	4	340	0: 0x0154 1: 0x02E4	16 bits for each direction (Tx/Rx) - currently not supported by SW
Reserved	64	344	0: 0x0158 1: 0x02E8	
XGXS backplane Rx equalizer matrix coef. (lane4:lane3:lane2:lane 1) (16bits for each lane)	8	408	0: 0x0198 1: 0x0328	Option 75 must be enable so this option will have effect. 4 times 16 bits for all 4 lanes. In case external PHY is present (not direct mode), those values will not take effect on the 4 XGXS lanes. For some external PHYs (such as 8706 and 8726) the values will be used to configure the external PHY – in those cases, not all 4 values are needed.
XGXS backplane Tx pre-emphasis matrix coef. (lane4:lane3:lane2:lane 1) (16bits for each lane)	8	416	0: 0x01A0 1: 0x0330	Option 75 must be enable so this option will have effect. 4 times 16 bits for all 4 lanes. In case external PHY is present (not direct mode), those values will not take effect on the 4 XGXS lanes. For some external PHYs (such as 8706 and 8726) the values will be used to configure the external PHY – in those cases, not all 4 values are needed.
Reserved	256	424	0: 0x01A8 1: 0x0338	
lane_config	4	680	0: 0x02A8 1: 0x0438	Bits 15–0:Lane swap configuration (16 bit) 0x1B1B – 01230123 0x1BE4 – 01233210 0xD8D8 – 31203120 0xE4E4 – 32103210 XGXS lanes order Bits 31–16:reserved

Table 8: Manufacturing Information (Cont.)

Parameter	*Bytes*	*Address*	*Address (hex)*	*Comments*
external_phy_config	4	684	0: 0x02AC 1: 0x043C	Bits 7–0:XGXS external PHY address (8 bits) The external PHY MDC/MDIO address Bits 15–8:XGXS external PHY type 0 – Direct 1 – BCM8071 2 – BCM8072 3 – BCM8073 4 – BCM8705 5 – BCM8706 6 – BCM8726 7 – BCM8481 8 – SFX7101 9 – BCM8727 10 – BCM8727_NOC 255 – Not Connected Value of the external XGXS (four lanes) PHY. This will determine the PHY driver, including register set Bits 23–16:SerDes external PHY address (8 bits) The external PHY MDC/MDIO address Bits 31–24:SerDes external PHY type 0 – Direct 1 – BCM5482 255 – Not connected value of the external SerDes (5th lane) PHY. This will determine the PHY driver, including register set
speed_capability_mask	4	688	0: 0x02B0 1: 0x0440	Bits 15–0: Speed capability mask (for D3) (2 bytes) Bit 0:10M FULL Bit 1:10M HALF Bit 2:100M HALF Bit 2:100M FULL Bit 4:1G Bit 5:2.5G Bit 5:10G Bit 6:12G Bit 8:12.5G Bit 9:13G Bit 10:15G Bit 11:16G Bitmap of supported speeds. These values will be used by bootcode (for WoL and Management)
speed_capability_mask				Bits 31–16:Speed capability mask (for D0) (2 bytes) Bit 16:10M FULL Bit 17:10M HALF Bit 18:100M HALF Bit 18:100M FULL Bit 20:1G Bit 21:2.5G Bit 21:10G Bit 22:12G Bit 24:12.5G Bit 25:13G Bit 26:15G Bit 27:16G Bitmap of supported speeds. These values will be used by the drivers.

Table 8: Manufacturing Information (Cont.)

Parameter	*Bytes*	*Address*	*Address (hex)*	*Comments*
Reserved	8	692	0: 0x02B4 1: 0x0444	Reserved
Checksum	4	1100	0x044C	

FEATURE CONFIGURATION INFORMATION

Table 9: Feature Configuration Region

Shared Hardware Features	*Size (Bytes)*	*Offset (d)*	*Offset (h)*	*Structure Description*
config	4	1104	0x0450	Bit 0:BMC loopback mode 0 – Disabled 1 – Enabled Enables BMC firmware to operate in loopback mode (default: 0). When this bit is set, MFW implements a loopback on BMC packets till first command. Bit 1: Override pre-emphasis configuration 0 – Disabled 1 – Enabled Use the values from options 47 and 48 instead of the HW default values Bit 2: reserved Bit 3: NCSI Package ID assignment method 0 – SPIO 1 – NVRAM Bits 5–4: NCSI Package ID Bits 7–6: Reserved Bits 10–8: Forces SF Mode 0 – MF allowed 1 – Forced SF 2 – SPIO4
Port Hardware Features				
config	4	1108	0: 0x0454 1: 0x04C8	

Table 9: Feature Configuration Region (Cont.)

Shared Hardware Features	Size (Bytes)	Offset (d)	Offset (h)	Structure Description
config				Bits 7–0: BARS size (Bar2: Bar1) 0 – Disabled 1 – 64K 2 – 128K 3 – 256K 4 – 512K 5 – 1M 6 – 2M 7 – 4M 8 – 8M 9 – 16M 10 – 32M 11 – 64M 12 – 128M 13 – 256M 14 – 512M 15 –1G BAR size according to the following decoding for BAR size (bytes): 0 – BAR is disabled 1 – 64K 2 – 128K 3 – 256K 4 – 512K 5 – 1M 6 – 2M 7 – 4M 8 – 8M 9 – 16M 10 – 32M 11 – 64M 12 – 128M 13 – 256M 14 – 512M 15 – 1G

Table 9: Feature Configuration Region (Cont.)

Shared Hardware Features	*Size (Bytes)*	*Offset (d)*	*Offset (h)*	*Structure Description*
config				Bits 23–8: Reserved Bit 24: Magic Packet WoL 0 – Disabled 1 – Enabled Out of the box Wake on LAN enable/disable Bit 25: MBA 0 – Disabled 1 – Enabled MBA enable/disable Bit 26:Mgmt Firmware 0 – Disabled 1 – Enabled Management firmware enable/disable Bit 27: Forces expansion ROM advertise 0 – Disabled 1 – Enabled If this bit is set, advertises expansion ROM even if MBA is disabled Bit 28: Reserved
config				Bits 31-29: Optic Module Vendor Enforcement 0 – No enforcement 1 – Disable Tx laser 2 – Warning message 3 – Power down

Table 9: Feature Configuration Region (Cont.)

Shared Hardware Features	Size (Bytes)	Offset (d)	Offset (h)	Structure Description
wol_config	4	1112	0: 0x0458 1: 0x04CC	Bits 3 – 0: Reserved Bit 4: Enabled WoL on ACPI matching management pattern 0 – Disabled 1 – Enabled The bit configures whether packets that match both ACPI patterns and management filtering rules should generate WoL or not. 1=if match on both WoL and management - set power_on; 0=if match on both WoL & management - don't set power_on. Bits 31-5: Reserved
mba_config	4	1116	0: 0x045C 1: 0x04D0	Bits 2–0: MBA Boot Protocol 0 – PXE 1 – RPL 2 – BOOTP 3 – iSCSI Boot 7 – None Boot agent 00b: PXE 01b: RPL 10b: BOOTP 11b: iSCSI boot Bits 9–3: Reserved Bit 10: MBA hide setup prompt 0 – Disabled 1 – Enabled Enable setup prompt message Bit 11: MBA Setup Hot Key 0 – Ctrl S 1 – Ctrl B Hot-key selection – If it is 0, the hot-key is Ctrl-S; otherwise, it is Ctrl-B.

Table 9: Feature Configuration Region (Cont.)

Shared Hardware Features	*Size (Bytes)*	*Offset (d)*	*Offset (h)*	*Structure Description*
mba_config				Bits 19-12:Expansion ROM size 0 – Disabled 1 – 2K 2 – 4K 3 – 8K 4 – 16K 5 – 32K 6 – 64K 7 – 128K 8 – 256K 9 – 512K 10 – 1M 11 – 2M 12 – 4M 13 – 8M 14 – 16M 15 – 32M Expansion ROM according to the following decoding for BAR size (bytes): 0 – BAR is disabled 1 – 1K 2 – 2K 3 – 4K 4 – 8K 5 – 16K 6 – 32K 7 – 64K 8 – 128K 9 – 256K 10 – 512K 11 – 1M 12 – 2M 13 – 4M 14 – 8M 15 – 16M
mba_config				Bits 23–20: MBA Delay Time (0-15) Message timeout (in seconds). Bits 25–24: MBA Boot Type 0 – Auto 1 – BBS 2 – Int18h 3 – Int19h BIOS bootstrap methods 00b: Auto Detect 01b: BBS 10b: INT18h 11b: INT19h

Table 9: Feature Configuration Region (Cont.)

Shared Hardware Features	Size (Bytes)	Offset (d)	Offset (h)	Structure Description
mba_config				Bits 29-26: MBA Link Speed 0 – Auto 1 – 10M Full 2 – 10M Half 3 – 100M Half 4 – 100M Full 5 – 1G 6 – 2.5G 7 – 10G CX4 MBA link configuration 0: MBA_LINK_SPEED_AUTO 1: MBA_LINK_SPEED_10HD 2: MBA_LINK_SPEED_10FD 3: MBA_LINK_SPEED_100HD 4: MBA_LINK_SPEED_100FD 5: MBA_LINK_SPEED_1GBPS 6: MBA_LINK_SPEED_2_5GBPS 7: MBA_LINK_SPEED_10GBPS_CX4 8: MBA_LINK_SPEED_10GBPS_KX4 9: MBA_LINK_SPEED_10GBPS_KR 10: MBA_LINK_SPEED_12GBPS 11: MBA_LINK_SPEED_12_5GBPS 12: MBA_LINK_SPEED_13GBPS 13: MBA_LINK_SPEED_15GBPS 14: MBA_LINK_SPEED_16GBPS 15: Reserved
mba_config				Bits 31–30: Reserved
Reserved	4	1120	0: 0x0460 1: 0x04D4	
mba_vlan_cfg	4	1124	0: 0x0464 1: 0x04D8	Bits 15–0: MBA VLAN value (16 bits) VLAN value Bit 16:MBA VLAN 0 – Disabled 1 – Enabled Enable VLAN Bits 31–17:reserved
Reserved	4	1128	0: 0x0468 1: 0x04DC	
smbus_config	4	1132	0: 0x046C 1: 0x04E0	Bits 7-0: SM Bus Address {one byte even hex value (for example, b0=0)} Bit 0: Reserved Bits 7–1: Address value of the bus Bits 31–8:reserved
Reserved	4	1136	0: 0x0470 1: 0x04E4	

Table 9: Feature Configuration Region (Cont.)

Shared Hardware Features	*Size (Bytes)*	*Offset (d)*	*Offset (h)*	*Structure Description*
link_config	4	1140	0: 0x0474 1: 0x04E8	Bits 7–0:reserved Bits 10–8:Flow control 0 – Auto 1 – Tx 2 – Rx 3 – Both 4 – None 5 – SAFC Rx 6 – SAFC Tx 7 – SAFC Both Driver default values Bits 15–11:reserved Bits 19–16:Link speed 0 – Auto 1 – 10M Full 2 – 10M Half 3 – 100M Half 4 – 100M Full 5 – 1G 6 – 2.5G 7 – 10G CX4 Driver default speed Bits 23–20: Reserved
link_config				Bits 25–24:Switch connected configuration 0 – 1G switch 1 – 10G switch 2 – auto detect 3 – one-time auto detect Driver default HW interface (10G=XGXS=4 lanes, 1G=SerDes=5th lane) Bits 31–26: Reserved

Table 9: Feature Configuration Region (Cont.)

Shared Hardware Features	*Size (Bytes)*	*Offset (d)*	*Offset (h)*	*Structure Description*
mfw_wol_link_cfg	4	1144	0: 0x0478 1: 0x04EC	Bits 7–0: Reserved Bits 10–8: Mfw Wol Flow control 0 – Auto 1 – Tx 2 – Rx 3 – Both 4 – None 5 – SAFC Rx 6 – SAFC Tx 7 – SAFC Both Bits 15–11:reserved Bits 19–16:Mfw Wol Link speed 0 – Auto 1 – 10M Full 2 – 10M Half 3 – 100M Half 4 – 100M Full 5 – 1G 6 – 2.5G 7 – 10G CX4 Management and Wake on LAN link speed Bits 23–20:reserved
mfw_wol_link_cfg				Bits 25–24: Mfw Wol Switch connected configuration 0 – 1G switch 1 – 10G switch 2 – auto detect 3 – one-time auto detect Management and Wake on LAN HW interface (10G=XGXS=4 lanes, 1G=SerDes=5th lane) Bits 31–26:reserved
Reserved	76	1148	0: 0x047C 1: 0x04F0	
CRC of the feature configuration information block	4	1300	0x053C	

VIRTUAL PRODUCT DATA REGION

Table 10: Virtual Product Data Region

Virtual Product Data (VPD)	*Size (bytes)*	*Offset (d)*	*Offset (h)*	*Structure Description*
Read only VPD string + VPD-R	128	1304	0x0540	Read-Only field contains NULL-terminated product name string. See note1 Read only VPD string is up to 48 bits Note1: supported fields: Product Name: The default value is Broadcom NetXtreme II Ethernet Controller.
VPD-W	128	1432	0x05C0	VPD-R.

PROGRAM IMAGES

The boot code, option ROM firmware, and other value added optional firmware (such as UMP firmware and IPMI firmware) are stored in the Program Images region of NVRAM. All entries in this region are subject to relocation, provided that their corresponding entries in the Code Directory region are appropriately updated. Table 11 shows the Program Images region.

Table 11: Program Images Region

Offset	*Size (bytes)*	*Description*
0x808	Up to 256,000–0x818 bytes	This region contains bootcode, PXE binary, IPMI firmware, UMP firmware, and so forth.

CALCULATING THE CRC 32 CHECKSUM

The following shows the C code for calculating the 32 bit CRC of given data in Manufacturing_Info[] array.

```
typedef unsigned long u32;
#define MANUFACTURING_INFO_SIZE140
#define CRC32_POLYNOMIAL 0xEDB88320
char Manufacturing_Info[ MANUFACTURING_INFO_SIZE ];

void main()
{
u32 crc;
crc = ~util_gen_crc(Manufacturing_Info, MANUFACTURING_INFO_SIZE - 4, 0xffffffff);
*((u32 *)&Manufacturing_Info[MANUFACTURING_INFO_SIZE - 4]) = crc;
}
u32 util_gen_crc (
char *pcDatabuf, // Pointer to data buffer
u32 ulDatalen, // Length of data buffer in bytes
u32 ulCrc_in) // Initial value
{
char data;
u32 idx, bit, crc;
crc = ulCrc_in;
for (idx = 0; idx < ulDatalen; idx++) {
data = *pcDatabuf++;
for (bit = 0; bit < 8; bit++, data >>= 1)
{
crc = (crc >> 1) ^ (((crc ^ data) & 1) ?  CRC32_POLYNOMIAL : 0); } } return crc;
}
   }
}
```

FLASH CONTROLLER

The Broadcom NetXtreme II device has a flexible non-volatile memory system that supports up to 20 MHz Serial Flash through a dedicated four-signal SPI interface. The Flash Controller state machine provides software access to the external Serial SPI Flash memory device. Reads and writes to the Flash memory device are performed by reading and writing MCP_REG_MCPR_NVM_COMMAND registers in the device register space. Figure 10 shows the block diagram of the Flash controller state machine and its interfaces.

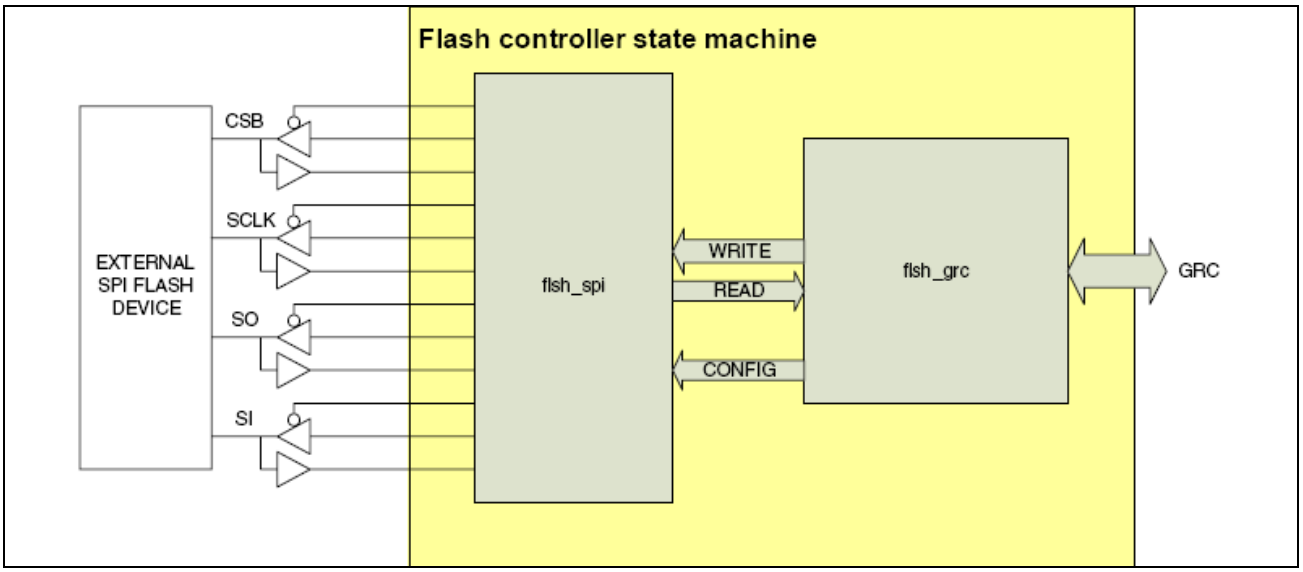

Figure 10: Flash Controller State Machine and Interfaces

Note: For a list of the serial Flash devices supported by a given device, please refer to its latest data sheet.

The main features of this interface include the following:

Provides automated sequencing FSM that implements the following commands:

- 32-bit Word Write
- 32-bit Word Read
- 32-bit String Read
- Page Erase

Provides native support for listed parts with strap pin selection of power-up defaults.

Provides a direct bit-bang control over interface pins.

SELF CONFIGURATION

A feature of the Flash controller is that it is self-configuring on reset, using pull-up and pull-down straps on the external SPI pins. Two Flash device vendors are natively supported: Atmel and ST Microelectronics. However, other vendors may be supported using bit-bang accesses. Configuration is automatically performed upon reset for the natively supported Flash devices.

Software does not need to setup any registers inside Flash device before usage. Device type and device commands are automatically configured based on strap values. The detected strap values are reflected in the MCP_REG_MCPR_NVM_RECONFIG register. The Flash controller hardware will automatically configure device size by issuing a read of the device's status or ID register upon exiting reset. The configured device size can be read from the MCP_REG_MCPR_NVM_CFG4 register. Both CFG4 and RECONFIG registers allow the default auto-configuration settings to be over-ridden by software or firmware. Table 12 shows the NVRAM strapping table. Although there are four straps, allowing for sixteen configurations, these are broken down into only two modes of functional timing and command words, one for each vendor (flash_straps[3:0] (SO,SI,CS,SCLK) = 4'b1XXX = Atmel; flash_straps[3:0] (SO,SI,CS,SCLK) = 4'b0XXX = ST).

Table 12: NVRAM Strapping Table

SO	*SI*	*CS*	*SCLK*	*Device*
0	0	0	0	Reserved
0	0	0	1	Reserved
0	0	1	0	Reserved
0	0	1	1	Reserved
0	1	0	0	Reserved
0	1	0	1	Reserved
0	1	1	0	ST M45PE10 ST M45PE20 ST M45PE40 ST M45PE80
0	1	1	1	Reserved
1	0	0	0	Reserved
1	0	0	1	Reserved
1	0	1	0	Reserved
1	0	1	1	Atmel AT45DB011B Atmel AT45DB021B Atmel AT45DB041B Atmel AT45DB081B Atmel AT45DB161D Atmel AT45DB321C Atmel AT45DB642D
1	1	0	0	Reserved
1	1	0	1	Reserved
1	1	1	0	Reserved
1	1	1	1	Reserved

ATMEL PAGE SIZES

Atmel pages are typically 264 Bytes (for devices 8 Mbit or less) whereas ST pages are 256 bytes, and therefore Atmel accesses require special consideration. The Flash controller state machine supports two modes of reads and writes for these devices, selectable via the MODE_256 bit in the MCP_REG_MCPR_NVM_COMMAND register. One mode of access is used to limit the pages size to 256-byte pages. The other mode is a legacy mode used to support the full 264-byte pages.

MODE_256

When using the 256-byte mode (MODE_256 bit set to 1) with Atmel devices, the address provided in the ADDRESS[23:0] register is broken down into 16 bits of page address (i.e., ADDRESS[23:8]) and 8 bits of byte address (i.e., ADDRESS[7:0]). Since these devices require 9 bits of byte address; the Flash controller hardware will automatically set the ninth bit (i.e., bit[8]) to 0. When writing to the Atmel device, software must ensure the LAST bit is set when writing to byte location 0xFF. When reading from the Atmel device, hardware will automatically close the page after reading from location 0xFF and then open the next page; this will be transparent to software. Larger Atmel devices, up to 64 Mbits are supported. The 16 Mbit and 32 Mbit Flash devices use 528-byte pages; but the page sizes will be limited to 512 bytes in this mode (hardware will always set the bit[9] to 0). Similarly, the 64 Mbit Atmel devices use 1056 Byte pages; but the hardware will always set the bit[10] to 0. The bits 5:4 of the MCP_REG_MCPR_NVM_CFG4 register indicate and control which address bit will be set to 0.

Legacy Mode

When using the 264 Byte legacy mode (MCP_REG_MCPR_NVM_COMMAND.MODE_256 bit set to 0) with Atmel devices, the address provided in the ADDRESS[23:0] register is broken down into 15 bits of page address (i.e., ADDRESS[23:9]) and 9 bits of byte address (i.e., ADDRESS[8:0]). This ADDRESS is basically passed through to the device. To use Legacy mode, it is required to support a logical addressing scheme that allows all supported Flash devices to be accessed in a consistent manner through a single, contiguous range of addresses. An example is provided below illustrating the holes that will be created in the NVRAM physical address map because of page size not aligning to any power of 2.

> ***Example:*** The Atmel® AT45DB011B is organized into 512 pages with 264 bytes per page. It uses nine address bits to reference an individual page (A17 to A9) and nine address bits to access an individual byte within a page (A8 to A0). This means that byte addresses 0 to 263 access actual data while addresses 264 to 511 do not, creating a hole in the address map. (The byte address actually wraps around to 0).

Implementing the logical addressing scheme as discussed above requires logical to physical address mapping. Listed below is the C code for logical to physical address mapping that is required when Legacy Mode is used for accessing the Atmel Flash devices whose page size is not aligned with any power of 2.

Logical to Physical Address Mapping Example Code

The following code is used to translate between logical and physical addresses for the Atmel AT45DB011B Flash device when Legacy Mode is used (MCP_REG_MCPR_NVM_COMMAND.MODE_256 bit set to 0).

```
/* Buffered Atmel Flash (Atmel AT45DB011B) address mapping functions */
#define BUFFERED_FLASH_PAGE_POS  9
#define BUFFERED_FLASH_BYTE_ADDR_MASK  ((1<<BUFFERED_FLASH_PAGE_POS) - 1)
#define BUFFERED_FLASH_PAGE_SIZE  264
#define BUFFERED_FLASH_PHY_PAGE_SIZE  512

typedef unsigned int  u32;

/* Convert a given Logical Address to Physical Address */
u32 nvram_LogicaltoPhysicalAddress (u32 address) {
```

```
u32 ad = address / BUFFERED_FLASH_PAGE_SIZE;
return (ad << BUFFERED_FLASH_PAGE_POS) + (address % BUFFERED_FLASH_PAGE_SIZE);
}

/* Convert a given Physical Address to Logical Address */
u32 nvram_PhysicaltoLogicalAddress (u32 address) {
return (address >> BUFFERED_FLASH_PAGE_POS) * BUFFERED_FLASH_PAGE_SIZE + (address &
BUFFERED_FLASH_BYTE_ADDR_MASK);
```

Note: All of the NVRAM addresses listed in the Code Directory are given as logical addresses. It is up to the application to convert these logical addresses to physical addresses if the NVRAM device requires such translation. The only exception to this rule is the offset field in the NVRAM Bootstrap region. This offset is given as a physical address to reduce the size and complexity of the ROM code executed by the management processor (MCP) when the reset is de-asserted.

Bit Bang and Pass Modes

A manual bitbang mode is supported. Using this mode, software can directly control the SPI pins to the flash via register writes. This includes toggling SCLK. Software must be cautious of the page boundaries when using bitbang mode. The bitbang mode is enabled by setting the MCP_REG_MCPR_NVM_CFG1:BITBANG_MODE bit to 1. When bitbang mode is enabled, the bits [3:0] of MCP_REG_MCPR_NVM_WRITE, MCP_REG_MCPR_NVM_ADDR, and MCP_REG_MCPR_NVM_READ registers are used to control the SPI signals.

A semi-automatic pass mode is also supported. This is a more automated version of the bitbang mode. Using this pass mode, software can send a serial stream of bits to the flash via register writes. As in normal operation, hardware will deassert/ assert the CS_L based on the FIRST/LAST bits in the MCP_REG_MCPR_NVM_COMMAND register and will toggle the clock. However, unlike normal operation in which hardware issues the command/address/data to the device, hardware instead will serially shift out an 8 bit value written by software. The pass mode is enabled by setting the MCP_REG_MCPR_NVM_CFG1:PASS_MODE bit to 1.

PROGRAMMING THE NON-VOLATILE MEMORY

Access to the non-volatile memory interface is controlled through internal configuration, command, and status registers in the NVRAM register block. The NVRAM can be accessed with automated 32-bit read and write commands or configured for bit-bang operation through the NVM control registers. A semaphore register (MCP_REG_MCPR_NVM_SW_ARB) allows up to four software entities to share access to the NVRAM device.

Note: Request level 0 is the highest priority arbitration request level and is reserved for BCM57710/BCM57711 firmware/ boot code. See the Broadcom Linux or FreeBSD drivers for an example of NVRAM Access code.

Section 4: Data Structures

Host Memory L2 Data Structures

This section describes the device data structures that need to be maintained by the L2 device driver in host memory. These data structures are subject to change, and correspond to the firmware version of the device. As an example, see Appendix D "bxe_hsi.h". All host memory structures that are used to pass data between the Broadcom NetXtreme II and the driver must be in locked system memory. Locking prevents memory from being swapped to disk or being moved so that it maintains a consistent physical memory address from the device's point of view. The L2 data structures are defined for use between the host driver software, the device firmware and the device hardware.

Virtual Versus Physical Address Views

Data buffers are used to store payload data that is moving between the Ethernet controller, driver, host operating system, and application. These are always allocated by the operating system. On host operating systems that provide virtual memory, the allocated memory will appear to exist at different addresses (and may even appear to be fragmented) depending on the point of view. In Table 10: "Virtual Product Data Region," on page 43, the driver sees a single, contiguous virtual memory block, while the Broadcom NetXtreme II sees the same memory as four discrete physical memory blocks, because the host operating system will allocate multiple pages (for example, 4 KB on many systems). For the data buffer to be accessible by the Broadcom NetXtreme II hardware, it must be locked into host physical memory and converted into a list of one or more physical address/length pairs. Normally, this physical view of a data buffer is carried in a Buffer Descriptor Chain (BD Chain), which is described in "Buffer Descriptor Chains" on page 52.

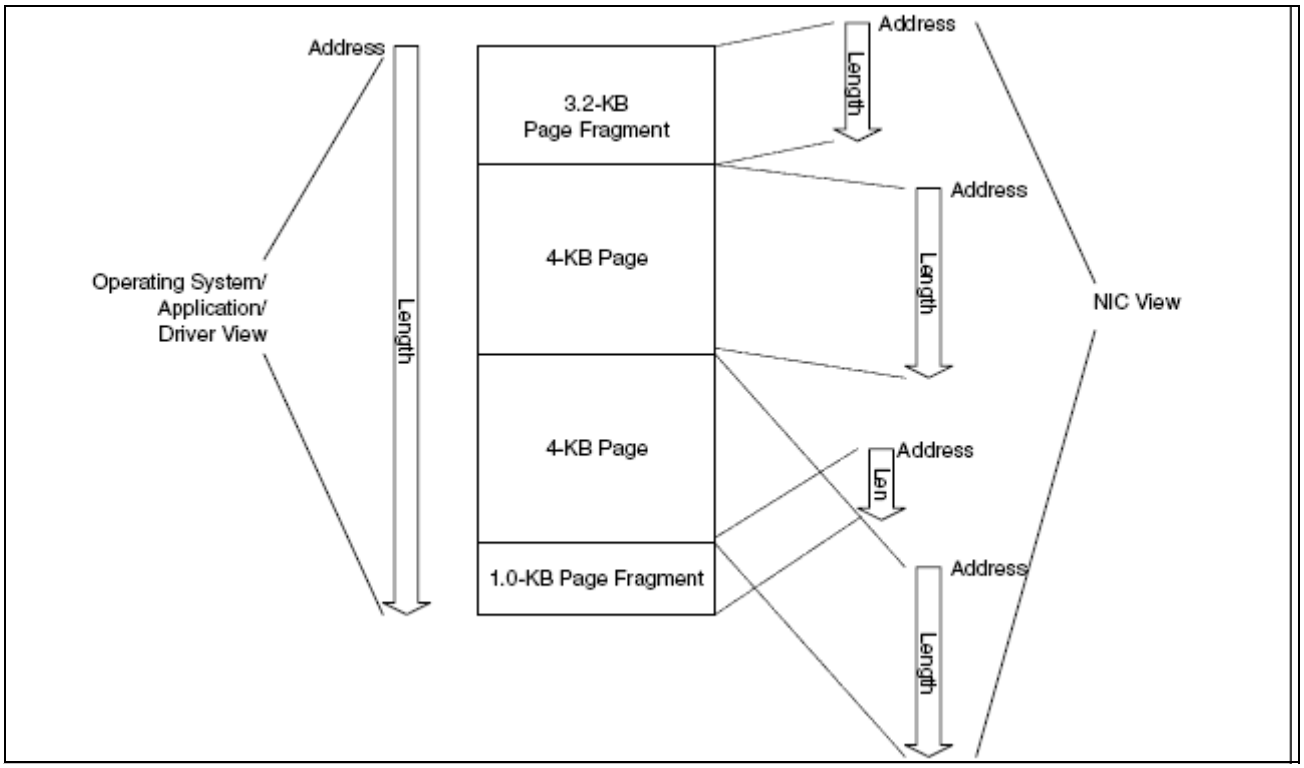

Figure 11: Virtual Address versus Physical Address View

BUFFER DESCRIPTOR CHAINS

A given number of pages will be allocated by the driver and linked together to form a page chain. Each page will hold an integer number of fixed size records (usually 16 bytes), called BDs (Buffer Descriptors). All pages in a chain will be used for storing and passing the BDs between the driver and the device and hence these chains are referred to as BD chains throughout this document. The last BD in each page will point to the first BD in the next page, where the last BD in the last page will point to the first BD of the first page. The motivation for this chain structure is in order to not force allocating a big chunk of contiguous physical memory, but to allow the pages to reside on different physical areas. Figure 12 shows a linked list of free pages, a chain (chain#1) of 3 pages, and another chain (chain#2) of 1 page.

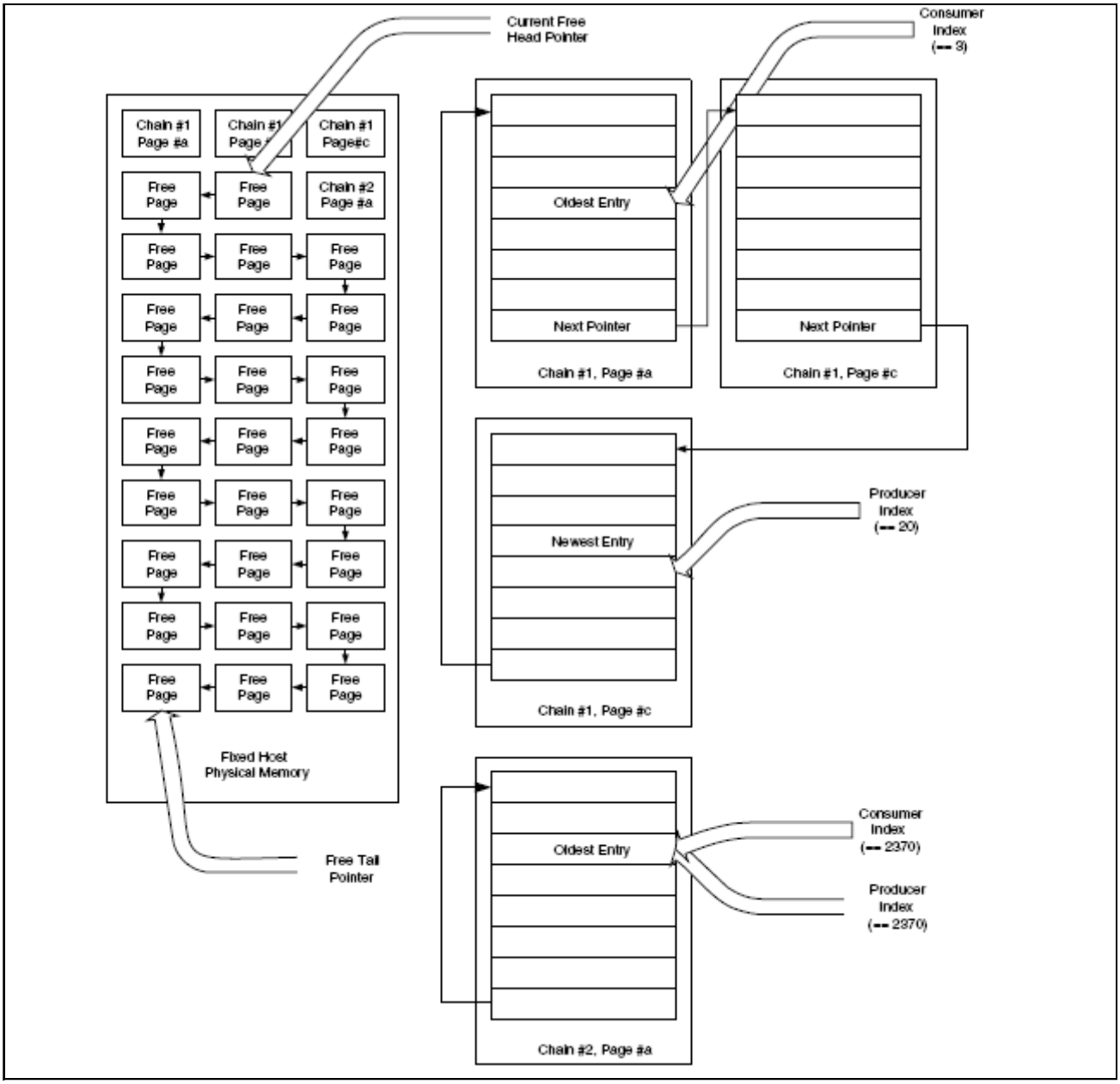

Figure 12: Chain With Multiple Pages

Because the Broadcom NetXtreme II supports bus-mastering, it is able to update chains independently of the driver. The driver maintains one or more Tx BD chains for transmit packets and one or more Rx BD chains for receive packets. In the case of packets being sent out onto the Ethernet, the driver adds BDs to the Tx chain and the device consumes the BDs from Tx chain. In this case, the driver is referred to as a writer and the device is referred to as a reader. In the receive direction, these roles are reversed and the device becomes the writer of the Rx chain while the driver takes on the role of a reader. Because a reader and writer may access a chain independently of each other, the writer will maintain a producer index (also known as the head) that points to the next entry to be written, while the reader will maintain a consumer index (also known as the tail) that will track the next entry to be read. When the producer and consumer indices are the same, the chain is empty. At reset, all indices are 0, and they increment by 1 for every record passed in the chain. If records are skipped at the end of a chain page, the index will be incremented to account for this. The record offset within a chain page can always be determined from the index (index MOD records_per_page). The driver allocates the Rx and Tx BD chains, statically, once at initialization time. They do not shrink or grow dynamically at runtime. A BD chain can be implemented with an array of page pointers. For each BD chain implemented by the driver, the driver should maintain the virtual and physical addresses of all the pages, the producer and consumer indices of the chain, a pointer to hardware updated consumer or producer index of the chain in status block, and other parameters like number of BDs used.

RX BUFFER DESCRIPTOR FORMAT

The firmware today assumes only 1 Rx BD to be used for receiving a single packet. This may change in the future to facilitate the header and data of a packet to be DMAed to separate buffers. With the restriction of 1 Rx BD for a packet, it is required that the host buffer associated with an Rx BD is large enough to receive an Ethernet MTU-sized packet. So, when Jumbo Frames are supported, each host buffer should be a single contiguous buffer of size larger than 9600 Bytes. The Rx BD structure in little endian format is as follows:

```
struct eth_rx_bd
{
uint32_t addr_lo;
uint32_t addr_hi;
};
```

addr_lo: The lower 32-bits of the 64-bit physical address of host memory buffer.

addr_hi: The upper 32-bits of the 64-bit physical address of host memory buffer.

Next Page Rx BD

The last BD in each page of the Rx BD chain, used to point to the address of the next BD at the beginning of the next page of the BD chain, has the following format.

```
struct eth_rx_next_bd

{

uint32_t addr_lo;

uint32_t addr_hi;

uint8_t reserved[8];
};
```

addr_lo: The lower 32-bits of the 64-bit physical address of first BD of the next page in Rx BD chain.

addr_hi: The upper 32-bits of the 64-bit physical address of first BD of the next page in Rx BD chain.

RX COMPLETION QUEUE ENTRY FORMAT

The BCM57710/BCM57711 supports the Rx Completion Queue for indicating the Rx completions (both fast path completions and slow path completions) to the driver. The RCQ (Rx Completion Queue) is also a page chain with each page in the chain containing multiple entries called CQEs (Completion Queue Entries). The device is the writer (producer) to this chain and the driver is the reader (consumer) of this chain. A received packet will consume an Rx BD from the Rx BD chain and produces a fast path CQE in the Rx Completion Queue. A posted slow path request (also referred to as ramrod in this document) by the driver, is processed by the device firmware and completion of the processing is indicated back to the driver by producing a ramrod CQE in the Rx Completion Queue. The number of CQEs in RCQ is usually maintained the same as the number of Rx BDs in the RX BD chain in order to provide completion indications for all the packets that have consumed an Rx BD. For L2, the size of CQE and the size of Rx BD are the same, and hence the number of pages in the RCQ is the same as the number of pages in the Rx BD chain. The structure of Rx CQE in little endian format is described below.

The CQE can be either a fast path CQE or ramrod CQE or the next_page CQE and hence the below union definition for the Rx CQE.

```
union eth_rx_cqe

{
struct eth_fast_path_rx_cqe fast_path_cqe;
struct common_ramrod_eth_rx_cqe ramrod_cqe;
struct eth_rx_cqe_next_page next_page_cqe;

};
```

Fast Path Rx CQE

This fast path CQE is for indicating the completion of an Rx packet. The structure of fast path CQE in little endian format is described below.

```
struct eth_fast_path_rx_cqe {
uint8_t type_error_flags;
uint8_t status_flags;
uint8_t placement_offset;
uint8_t queue_index;
uint32_t rss_hash_result;
uint16_t vlan_tag;
uint16_t pkt_len;
uint16_t len_on_bd;
struct parsing_flags pars_flags;
uint16_t sgl[8];

};
```

type: error_flags: Bit 0 indicates a fast path rx entry when it has a 0 value.

status_flags: See the Appendix of this document for a list of bit definitions.

placement_offset: This field specifies the placement offset in bytes from the start of the Rx buffer.

rss_hash_result: This field is the RSS Toeplitz hash result. Used only when RSS is enabled.

vlan_tag: This field is the Ethernet VLAN tag stripped from a tagged packet. Valid only VLAN tagged packets.

pkt_len: This field is the length of the received packet in bytes, including the headers and Ethernet CRC.

tcp_csum: This field is the TCP checksum value as calculated by the Parser state machine.

parsing_flags: This field specifies the parsing information for a given Rx packet.

```
struct parsing_flags
{
u16_t flags;
};
```

Ramrod Rx CQE

The ramrod CQE indicates completion of a posted slow path event (ramrod) to the device. The structure of ramrod Rx CQE in little endian format is described below.

```
struct common_ramrod_eth_rx_cqe

{
uint8_t ramrod_type;
uint8_t conn_type;
uint16_t reserved1;
uint32_t conn_and_cmd_data;
struct ramrod_data protocol_data;
uint32_t reserved2[4];
};
```

ramrod_type: This indicates whether this entry is a fast path CQE or ramrod CQE.

conn_type: This indicates the connection type. Listed below are the defined connection types. Note that only the last 3 bits of this field are used. For all L2 packets, the conn_type is ETH_CONNECTION_TYPE.

protocol_data: This field specifies the protocol specific data. For Ethernet packets, this field provides the information as specified in the eth_init_ramrod_data structure below.

Next Page Rx CQE

The next page CQE is used for chaining the pages of RCQ. It points to the first CQE of the next page in the RCQ.

```
struct eth_rx_cqe_next_page

{
uint32_t addr_lo;
uint32_t addr_hi;
uint32_t reserved [6];

};
```

addr_lo: The lower 32-bits of the 64-bit physical address of first CQE of the next page in RCQ.

addr_hi: The upper 32-bits of the 64-bit physical address of first CQE of the next page in RCQ.

TX BUFFER DESCRIPTOR FORMAT

The device supports scatter-gather DMA mechanism by supporting the use of 1 or more BDs for transmitting a given packet. The structure of Tx BD in little endian format is described below.

```
struct eth_tx_bd
{

uint32_t addr_lo;

uint32_t addr_hi;

uint16_t nbd;

uint16_t nbytes;

uint16_t vlan;

struct eth_tx_bd_flags bd_flags;

uint8_t general_data;
};
```

addr_lo: The lower 32-bits of the 64-bit physical address of host memory buffer.

addr_hi: The upper 32-bits of the 64-bit physical address of host memory buffer.

nbd: The total number of BDs, including the parse BD, used for a given Tx packet. This field is only relevant in the first BD (also called start BD) of a Tx packet.

nbytes: The size of the Tx buffer in Bytes.

vlan: The 16-bit VLAN Tag information for a given Tx packet. The LSB 12 bits are the VLAN ID, the next MSB bit is the CFI, and the next 3 MSB bits are the 802.1p priority bits.

bd_flags: This structure defines the following BD flags.

```
struct eth_tx_bd_flags
{

uint8_t as_bitfield;
};
```

Tx Parsing Information BD

This Parsing Information BD is required only when either the checksum is offloaded or an LSO/GSO packet is being transmitted. This BD carries the necessary header parsing information of a given Tx packet to aid the device to quickly calculate the required IP/TCP checksums and/or some of the TCP/IP header fields for LSO/GSO packets. The structure of this parsing information BD in little endian format is given below.

```
struct eth_tx_parse_bd

{

uint8_t global_data;

uint8_t tcp_flags;

uint16_t ip_hlen; uint8_t_cs_offset;

uint16_t total_hlen;

uint16_t lso_mss;

uint16_t tcp_pseudo_csum;

uint16_t ip_id;

uint32_t tcp_send_seq;
};
```

global_data: This field carries the global parsing information of a given Tx packet. Listed below are the various bit fields embedded in this field.

tcp_flags: This field specifies TCP parsing information to the device and is relevant only when a given packet is an LSO packet.

ip_hlen: This field specifies the length of IP header, including IP options, in WORDs.

total_hlen: This field specifies the length of Ethernet, IP, and TCP headers.

lso_mss: This field specifies the MSS (Maximum Segment Size) for TCP segmentation. This is relevant only for LSO/GSO packets.

tcp_pseudo_csum: This field carries the pseudo header checksum value calculated with the length field as zero.

ip_id: This field specifies the IP identifier in the IP header of the given packet. Used only with the LSO packet.

tcp_send_seq: This field specifies the TCP Send Sequence Number in the TCP header of the given packet. Used only with the LSO packet.

Next Page Tx BD

The last BD in each page of the Tx BD chain, used to point to address of the next BD at the beginning of the next page of the BD chain, has the following format.

```
struct eth_tx_next_bd

{

uint32_t addr_lo;

uint32_t addr_hi;

uint8_t reserved[8];
};
```

addr_lo: The lower 32-bits of the 64-bit physical address of first BD of the next page in Tx BD chain.

addr_hi: The upper 32-bits of the 64-bit physical address of first BD of the next page in Tx BD chain.

STATUS BLOCK FORMAT

There are a total of 17 status blocks per function: 16 status blocks for fast path, and 1 default status block for slow path. Each of these 16 fast path status blocks contains a USTORM structure and a CSTORM structure. The default status block contains structures for all the 4 STORMs (USTORM, CSTORM, TSTORM, and the XSTORM). In addition, it contains the attention bits. Each of the 17 status blocks has its own status_block_id which is a unique identifier for each port/RSS (if any)/ STORM combination.

The driver allocates the host memory for the status block and configures the device with the physical address of the status blocks in host memory. The device updates the status block in host memory upon the triggering of an event like link status change or the host coalescence triggering upon Rx and/or Tx packet completions. Each of the status blocks contains an index which will increment by 1 with every update of that status block in host memory. The driver also maintains a local copy of each of the status blocks and compares the local copy index with the index in the device status block in order to determine if there are any new events that need to be serviced. The driver checks for updates in the status blocks either at regular intervals in case of polling mode or upon interrupt.

Note that each status block supports multiple protocols. For example; the status block for CPU_ID 3 can support operations for a L2 NIC, an L4 stream and iSCSI operations at the same time.

Fast Path Status Block

The BCM57710/BCM57711 supports up to 16 fast path status blocks. Each of these status blocks has updates from the USTORM and CSTORM. Refer to "bxe_hsi.h" in the appendix, or in the open source driver. Note that the reserved fields are used for alignment purposes. The name "fast path" implies that these status blocks are used within the critical time context of transmitting and receiving packets.

Default Status Block

There is only one default status block per function. The structures are defined in the "bxe_hsi.h" found in the appendix. Note that the fields marked as reserved are used for alignment purposes. The name "slow path" is used to imply tasks including the default status block, which are not as time critical as the network traffic tasks.

Section 5: Host Driver Flows

This section describes the data flows of L2 and L4 traffic through the chip and driver.

DEVICE INITIALIZATION AND SHUTDOWN

Use the Broadcom Open Source Linux or FreeBSD drivers as a companion reference for this section.

The BCM57710/BCM57711 device is initialized in roughly three phases. The first phase occurs before the driver loads and is mentioned here only for reference. The final two phases are controlled by the host driver using the Management Control Processor (MCP) to help in the synchronization of the four device internal STate Optimized RISC Microprocessors (STORM)s and the device firmware.

The first phase occurs after a PCIE reset to the device. It includes the loading of the device boot code from the NVRAM and the boot code writing configuration parameters to the device shared memory in the MCP memory space. The loading of the MCP firmware from non-volatile memory into MCP memory and MCP code execution is also included in this phase.

The second phase begins with the loading of the host driver and includes the host driver writing initialization values to each of the device hardware block registers. This phase also includes the host driver writing the device firmware to STORM memory and should include the host driver/operating system initialization steps and host driver memory allocation for device specific structure initialization. Since the BCM57710/BCM57711 shares common resources between two ports, the second phase also includes the initialization of the device common resources.

The third and final phase begins when the host driver posts a command to the STORM firmware referred to as the initial ramrod command. This is the operation that primes the device to use the host driver allocated memory structures and connection context. Included in this phase is the posting of subsequent ramrod commands for port or function MAC address configuration and multi queue support. The physical layer (PHY) initialization is included in this last phase. For both the device initialization and the device shutdown, the MCP is used as a handshake interface for each step.

MCP INTERFACE

The host driver uses the device Management Communication Processor (MCP) to synchronize the communication between the host driver, the device firmware and the device STORMs during the device initialization process. When the host driver is loaded into host memory, the MCP firmware should already be up and running and available for the host driver to use during the device initialization process.

The host driver can interact with the MCP when using the MCP interface. The interface to the MCP consists of four command/ request and response mailboxes located in the device shared memory where pre-defined request values can be written for the MCP and pre-defined responses can be read for the host driver.

Per Device port the Shared Memory contains one mailbox for requests from the host driver to the MCP and another mailbox for responses from the MCP. The shared memory also contains two additional a pulse mailboxes per port for a heart beat handshake between the host driver and MCP.

To verify that the MCP firmware has loaded properly, the host driver must find the shared memory address by reading from the MISC_REGISTERS_SHARED_MEM_ADDR(0xA2B4) register.

An example of the device shared memory region is defined in the bxe_hsi.h file in the appendix. The MCP/ firmware is not running if the shared memory address read from the MISC_REGISTESR_SHARED_MEM_ADDR is 0 or 0xf.

The defined values used by the host driver to verify the validity of the shared memory are found in the bxe_hsi.h sample file found in the appendix.

Once the shared memory is determined to be valid, the MCP mailbox interface can be used and the device initialization can continue.

Besides being a location for the MCP mailbox interface, the device shared memory contains device parameters loaded from the NVRAM for use during initialization and available to the host driver.

The host driver and the MCP synchronize the initialization process by a sequence of commands and responses or handshakes. The command mailbox is written by the host driver and read by the MCP. The response mailbox is written by the MCP and read by a driver.

The command mailbox is composed of 32-bit message location in device shared memory. The format is described in the following table:

D31	D16	D15	D0
Command		Sequence number	

The response mailbox is composed of 32-bit message located in device shared memory. The format is described in the following table:

D31	D16	D15	D0
Response		Sequence number	

The host driver initiates a device initialization handshake to the MCP by writing a command to the MCP command mailbox. The MCP writes a response to the response mailbox before the host driver begins the initialization process.

Each command includes an incremental sequence number. The response also includes a sequence number field and a value that matches the command sequence number.

The host driver must not post a new command until an MCP response is posted for the previous command by the MCP.

To begin the device initialization process the driver must send a command to the MCP to signal the start of the device initialization process with a value that represents the command for Driver Load (see bxe_hsi.h in the appendix).

The BCM57710/BCM57711 shares internal device resources with multiple ports/functions, so depending on if the shared resources have already been initialized or not, the MCP will respond to the Driver Load command with the response of Load Common or Load Port or Load Function.

The Load Common response from the MCP requires the host driver to begin to initialize the device common resources. The device common resources include all of the device hardware blocks and the device STORMs.

The Load Port or Load Function response from the MCP requires that the driver limit the initialization process to the port or function initialization. The port or function resources are specific entries in the device NIG block and the EMAC and BMAC device blocks as well as the external PHY.

After the host device driver has completed the common and port or function initialization, the host driver writes a command of Load Done. The MCP response of Load Done completes a portion of the initialization process.

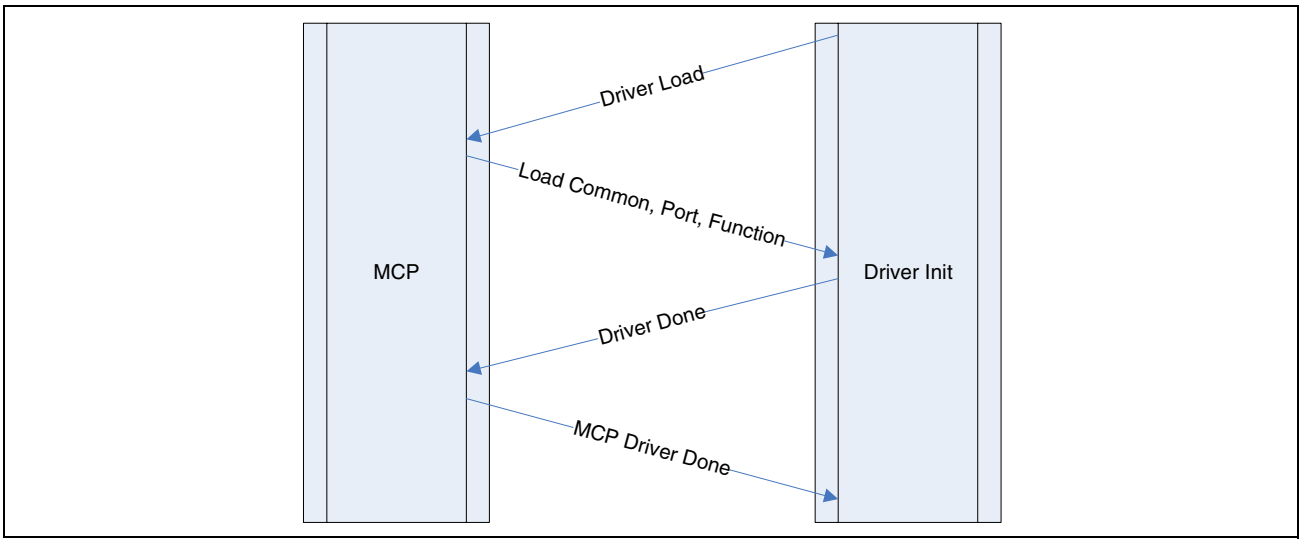

Figure 13: Driver/MCP Handshake

HEART BEAT/PULSE

The heart beat or pulse handshake between the host driver and MCP is used to inform the MCP that the host driver is loaded and running on the device. This allows the MCP to take control the network link should the host driver become unstable and unable to send a pulse or communicate with the MCP. The pulse handshake mailboxes have the same format as the command and response mailboxes.

To disable the driver pulse mechanism, a flag can be set to indicate to the MCP that the host driver is always alive. See the FreeBSD BCM57710/BCM57711 driver.

NIG DRAIN

The drain mode register of the NIG block (0x10060) is used to flush the device of outgoing packets. When the drain mode bit is set, then the drain is activated and the device will not transmit packets. This bit is controlled by both the MCP firmware and by the host driver. When the controlling software entity services a link–up event it will clear this bit. This bit is set when the controlling software entity services a link–down event.

It is important that all the MCP handshake mechanisms are used properly so that the host driver, MCP and device firmware function in unison.

HARDWARE BLOCK INITIALIZATION AND STORM FIRMWARE DOWNLOAD

The MCP Load Common response received by the host driver from the MCP requires that the host driver begin initialization of all the device hardware blocks which includes downloading the firmware for all device STORM s. Most of the common initialization values for each hardware block are pre-defined and tightly coupled with the version of firmware downloaded into the device STORMS.

The device STORM firmware written to STORM memory can be read back for verification of a successful download. The first host driver loaded to use the device always receives the MCP response of Load Common because the first driver loaded must initialize the portion of the device that is used in common by both ports. The first host driver loaded also initializes the first port and function of the device without requesting a Load Port command from the MCP.

When a second host driver is loaded it initiates the Driver Load command to the MCP just like the first host driver, however, the MCP response is always the Load Port response since the common portion of the device is already initialized by the previously loaded driver on the initial port. A second host driver load initializes only the second port or function and not the common part of the device.

The initialization of the device hardware blocks and device STORMS by the host driver is the second phase of the device initialization. The host device driver allocates memory for software structures used by the host driver, device firmware. The physical address of the host memory structures passed to some of the device hardware blocks. The final step of the device initialization occurs in phase three of the device initialization process where the device uses the host driver allocated structure values to validate the initial port setup.

HOST DRIVER INITIALIZATION

The BCM57710/BCM57711 depends on the host driver to allocate host memory for device operations. The host driver must allocate memory and initialize the structures that are needed by the device. This includes the TX and RX buffer descriptor chains and descriptors, the Receive Completion Queue (RCQ) and status block structures and other structures.

The context structure is a very important part of the initialization process. This is a single memory structure that the device needs to establish a context between the STORMS and the host driver to establish a network connection.

The context structure is the only host memory structure required to be on a 4K memory address alignment and it is initialized by the host driver with necessary context data such as the base physical address of the TX/RX descriptor base and other parameters. This context structure is named the eth_context structure found in the example code.

The host driver should allocate the RX descriptors and receive buffers in 16-byte aligned memory address.

RAMROD

A dictionary definition for the word Ramrod reads: *a rod for ramming home the charge in a muzzle-loading firearm*. This word is also used to describe a BCM57710/BCM57711 device mailbox method of passing the index of a command structure in memory directly to the device STORM firmware.

The host driver communicates with the device STORM firmware via mailbox commands called ramrod commands, also known as slow-path commands. A slow-path or ramrod command is used to send a device configuration change to the device STORM firmware. It is called a slow-path because the The new configuration information is sent to each device STORM processor and to the device STORM related hardware blocks. The device STORM response to a ramrod command is a completion status message posted to host memory usually as a completion queue entry (CQE) in the receive completion queue (RCQ). Most of the responses are posted in the RCQ, however the statistics ramrod and the delete port ramrod are completed on the default status block

Before using the ramrod interface, the host driver must first allocate a 4K byte block of memory where the ramrod command descriptor can be placed in host memory. The ramrod descriptors contain fields that are used for various slow path tasks. The host driver initializes a location in the STORM memory with the physical base address of this slow path queue and the starting producer index. The address locations for the slow path queue and producer index in the example code are the following:

XSEM_REG_FAST_MEMORY + XSTORM_SPQ_PAGE_BASE_OFFSET

XSEM_REG_FAST_MEMORY + XSTORM_SPQ_PROD_OFFSET

To post a slow path command, the host driver writes the index to STORM memory of a slow path command descriptor which has been modified with command information. This slow path producer index is placed in STORM memory to initiate the slow path command.

BAR_XSTORM_INTMEM + XSTORM_SPQ_PROD_OFFSET

The device firmware preserves spaces on an internal fast-path ring for ramrod completions and the host driver must keep a count of posted ramrods. When there are not sufficient resources to post new ramrods the host driver must queue the ramrod request until resources become available. Currently the maximum number of slow path commands that can be queued at any time is eight.

The device firmware will consume one CQE for each slow path completion and will not consume a BD. The CQE contains a flag that indicates if the entry is for a ramrod completion or a fast path completion.

The final phase in the device initialization includes posting a ramrod command to the STORM firmware called a port setup ramrod command. The port setup ramrod command causes the device to use information found on the ramrod initialized by the host driver and to DMA completion data to the receive completion queue. This slow path command completion occurs on the fast path completion queue.

The port setup ramrod operation must be the first slow path command posted by the host driver after the firmware download and hardware block initialization completes and after writing the DRV_MSG_CODE_LOAD_DONE handshake command to the MCP.

The port setup ramrod initializes the first or leading connection of the port (cid == 0). It is the first ramrod to be sent on each port after the host driver initialization is finished. It loads the context to the chip, locks it, and initializes all storms.

Ironically, the completion of this slow path command or ramrod operation completes on the fast path receive completion queue (RCQ). ASTORM puts the completion queue entry (CQE) on the RCQ of the leading connection. The command field of the ramrod completion entry is the same as the posted command when the operation completes successfully.

Only after the port setup ramrod completes successfully, can other ramrod operations take place and only sequentially. There can only be one ramrod command pending per function.

Here is a list of all the supported ramrods:

RAMROD_CMD_ID_ETH_PORT_SETUP

Used to setup the leading connection on a port. Completes on the Receive Completion Queue (RCQ) of that port

RAMROD_CMD_ID_ETH_CLIENT_SETUP

Used to setup an additional connection on a port. Completes on the RCQ of the multi-queue/RSS connection being initialized

RAMROD_CMD_ID_ETH_STAT_QUERY

Used to force the storm processors to update the statistics database in host memory. This ramrod is sent on the leading connection CID and completes as an index increment of the CSTORM on the default status block

RAMROD_CMD_ID_ETH_UPDATE

Used to update the state of the leading connection, usually to update the RSS indirection table. It completes on the RCQ of the leading connection

RAMROD_CMD_ID_ETH_HALT

Used when tearing down a connection prior to driver unload. It completes on the RCQ of the multi-queue/RSS connection being torn down. Do not use this on the leading connection

RAMROD_CMD_ID_ETH_SET_MAC

Sets the Unicast/Broadcast/Multicast used by the port. It completes on the RCQ of the leading connection

RAMROD_CMD_ID_ETH_CFC_DEL

Used when tearing down a connection prior to driver unload. It completes on the RCQ of the leading connection (since the current connection has been completely removed from controller memory)

RAMROD_CMD_ID_ETH_PORT_DEL

Used to tear down the leading connection prior to driver unload. It completes as an index increment of the CSTORM on the default status block

RAMROD_CMD_ID_ETH_FORWARD_SETUP

Used for connection offload. It completes on the RCQ of the multi-queue RSS connection that is being offloaded.

DEVICE SHUTDOWN

The process of shutting down the BCM57710/BCM57711 includes communicating with the MCP in a similar manner as the initialization process. The host driver informs the MCP of a shutdown by sending a Driver Unload request to the MCP. The MCP responds to the host driver with an acknowledgement of Common Unload, Port Unload or Function Unload. The following steps are required for the host driver to properly shutdown the BCM57710/BCM57711 device.

- Set the device receive filter to not receive packets
- Clear all MAC addresses from the CAM using the SET_MAC ramrod
- Reset the network link to drain the transmit queue
- Issue the HALT ramrod
- Issue the ETH_CFC_DEL ramrod for each queue connection that is not the leading connection
- Issue the ETH_PORT_DEL ramrod for the leading connection
- Disable device interrupts
- Issue the Driver Unload request to the MCP firmware
- Use the response from the MCP firmware to determine if a common reset or port reset is required
- Reset the common or port resources to a known state

INTERRUPT HANDLING AND ATTENTION

This is a high level description of how to manage BCM57710/BCM57711 interrupts in a host software device driver. Some of the BCM57710/BCM57711 registers and memory locations referenced in this section are subject to change. See the Linux BCM57710/BCM57711 driver or FreeBSD BCM57710/BCM57711 driver for the latest BCM57710/BCM57711 device register address offsets and hardware and firmware dependent memory structures.

INTERRUPT MODES

The BCM57710/BCM57711 supports three interrupt modes: The INTx# mode, Message Signaled Interrupt (MSI) mode and the eXtended Message Signaled Interrupt (MSI-X) mode. The support for these interrupt modes are determined by the system architecture and operating system. Some systems may support all three modes and would allow for the selection and user preference of the interrupt mode. The 57xx supports up to 16 MSI-X vectors and up to 4 MSI vectors.

BCM57710/BCM57711 INTERRUPT GENERATION

Interrupts are generated from two main areas in the BCM57710/BCM57711 which are the attention block and the internal STate Optimized RISC Microprocessor (STORMs) CPUs. The attention block generates interrupts based on configuration settings written to the attention block at device initialization time. The attention interrupts (attentions) are placed mainly on the slow path interrupt. The STORMs generate interrupts when an update of an index register in a status block occurs. Each status block index update may generate an interrupt. These interrupts can be placed on both the slow path and fast path. The slow path and fast path are described below.

STATUS BLOCKS

All status blocks are grouped into two main status blocks: The default status block and TX/RX status blocks. The status blocks are allocated in host memory by the host driver and the device hardware is initialized with a pointer to the physical base address of the status blocks during device initialization time.

The default status block is made up of five different status blocks also referred to as status block segments. These are the Attention, XSTORM, USTORM, TSTORM and CSTORM status block segments. The BCM57710/BCM57711 uses one default status block per port or function. The default status block is also known as the slow path status block (see Table 13 on page 66).

The TX/RX status block which is also referred to as the fast path status block or non-default status block is made up of 2 types of status blocks or status block segments called the USTORM and CSTORM status blocks. The USTORM status block segment is associated with packet receive interrupts and receive interrupt coalescing and the CSTORM status block segment is associated with packet transmit completions and transmit interrupt coalescing (see Table 14 on page 68).

The BCM57710/BCM57711 can support up to 16 fast path status blocks per port or function. This amount of status blocks are used by the BCM57710/BCM57711 to manage up to 16 queues or host CPUs in the BCM57710/BCM57711 Multi-Queue or Receive Side Scaling (RSS) feature.

Each status block segment has a status index field that is incremented after each update from the device. The status index update signifies that an event occurred and needs to be serviced. The host driver must maintain a copy of the hardware status block segments status index to determine which status block segment status index has changed or what event needs to be serviced, When there is a change between the host driver copy and the status index within a status block segment then the status block has changed and the driver must service the event for that particular status block segment.

When the device is passing traffic, the status index value within a status block segment will continually be updated. Each STORM independently updates the status index of a status block segment, therefore the driver must check for index updates after each loop within an ISR.

Table 13: Default (Slow Path) Status Block

31	*24*	*23*	*16*	*15*	*8*	*7*	*0*
Attention Bits							
Attention Bits Acknowledge							
Attention Bits status index				Reserved		Status block id	
USTORM STATUS BLOCK							
USTORM status index				Reserved		Status block id	
CSTORM STATUS BLOCK							
CSTORM status index				Reserved		Status block id	

Table 13: Default (Slow Path) Status Block

31	***24***	***23***	***16***	***15***	***8***	***7***	***0***
XSTORM STATUS BLOCK							
XSTORM status index				Reserved			Status block id
TSTORM STATUS BLOCK							
TSTORM status index				Reserved			Status block id

Table 14: TX/RX (Fast Path) Status Block

31 24	23 16	15 8	7 0
USTORM STATUS BLOCK			
USTORM status index		Reserved	Status block id
CSTORM STATUS BLOCK			
CSTORM status index		Reserved	Status block id

ISR Mode

There are two ISR mode settings in the BCM57710/BCM57711. One setting is referred to as the single ISR mode, which is when the device combines all the interrupts from each of the status block interrupts into one. The default status block plus the potentially sixteen fast path status blocks interrupts are funneled into one single interrupt in the single ISR mode.

The non-single ISR mode allows each status block to independently interrupt the host. For the driver to support this mode each MSI or MSI-X vector must be initialized by the host driver to service a separate status block.

To configure the ISR mode on the BCM57710/BCM57711 the host driver must set the appropriate bit in the HC_REGISTERS_CONFIG_ register during driver initialization for the duration of the driver instance.

Interrupt Configuration and Control

The BCM57710/BCM57711 device uses HC_REGISTERS_CONFIG_0/1 (0/1 depending on the port or function) register that is used to enable and disable interrupts, and to enable the interrupt type during device initialization. For example, to use an INTA# mode ISR configuration in port/function 0, the host driver must clear the MSI_ATTN_EN_0 bit, clear the MSI_MSIX_MEMORY_EN_0 bit, set the SINGLE_ISR_EN_0 bit and set the INT_LINE_EN_0 bit (see Figure 14 on page 70).

Control for allowing and disallowing INTA interrupts generated by function 0 can be done by the setting and clearing the INT_LINE_EN_0 bit respectively.

Note: In the BCM57710/BCM57711 when setting the INT_LINE_EN_x (bit3 in HC_REGISTERS_CONFIG_x) bit, the MSI_MSIX_INT_EN_x (bit 2 in HC_REGISTERS_CONFIG_X) bit must be set to 1 then cleared.

For the host driver use of MSI or MSI-X type interrupts, the MSI_MSIX_MEMORY_EN_0 bit and the MSI_ATTN_EN_0 bit should be enabled.

Allowing and disallowing these interrupts can be controlled using the MSI_MSIX_INT_EN_0 bit (see Figure 14 on page 70).

HOST DRIVER INTERRUPT HANDLER FLOW

The host driver should take the following steps within the ISR:

1. To serialize the interrupts and ensure no other interrupt occurs during the ISR servicing thread, the interrupt generation of all interrupts from the BCM57710/BCM57711 on the current port can be suppressed by disabling the specific bit in the HC_REGISTERS_CONFIG register.
2. The host driver must check to determine if the status index of a status block differs from the host driver copy of the status index by first reading the interrupt status register. To do this the host driver must first read the HC_REGISTERS_COMMAND_REG single_isr_multi_dpc_wmask_ port0 register. This register returns a 32 bit value of which the first 17 bits correspond to events posted on the single default status block and event posted on the up to 16 fast path status blocks. Bit 0 corresponds to a default status block event and bits 1-16 corresponds to an event in fast path status blocks 1-16.
3. When the single_isr_multi_dpc_wmask register is read, the asserted bits are automatically cleared and masked off to disable corresponding interrupts on the device. There is also a single_isr_multi_dpc w/o mask register which automatically clears the asserted bit when read, but the active interrupt is not disabled. In single_isr mode, only two of the 32 bits are valid: bit 0 and bit 1. Bit 0 corresponds to the default status block and bit 1 corresponds to the fast path status block. When the single_isr mode is disabled, bit 0 (default status block) and bits 1-17 (fast path status blocks) are valid.
4. The main reason the non- single_isr mode is used in a driver is to allow for servicing or queuing of multiple independent interrupt tasks. The single_isr_multi_dpc register with mask is part of the HC_REGISTERS_COMMAND_REG register (see Figure 14 on page 70).
5. The following steps can be if the ISR is calling a task queue within a Linux NAPI type driver or a DPC within a Windows miniport driver:
6. Service each event by comparing the driver status index with each storms status block index.
7. Update the index saved by the driver with the one from the status block.
8. Acknowledge the status blocks by updating the consumer index of all the status blocks and set the interrupt enable bit on the last status block to be updated. Setting the interrupt enable bit on the status block allows an interrupt to be generated from that status block the next time the storm updates the index of that status block. The consumer index of a status block and the interrupt enable bit are written to the HC_REGISTERS_COMMAND_REG register.
9. If the interrupts were suppressed in the HC_REGISTERS_CONFIG_ register, then they must be set to allow interrupts before returning from the ISR, Windows DPC or Linux NAPI task.

A flow diagram of the driver interrupt handler is shown in Figure 14.

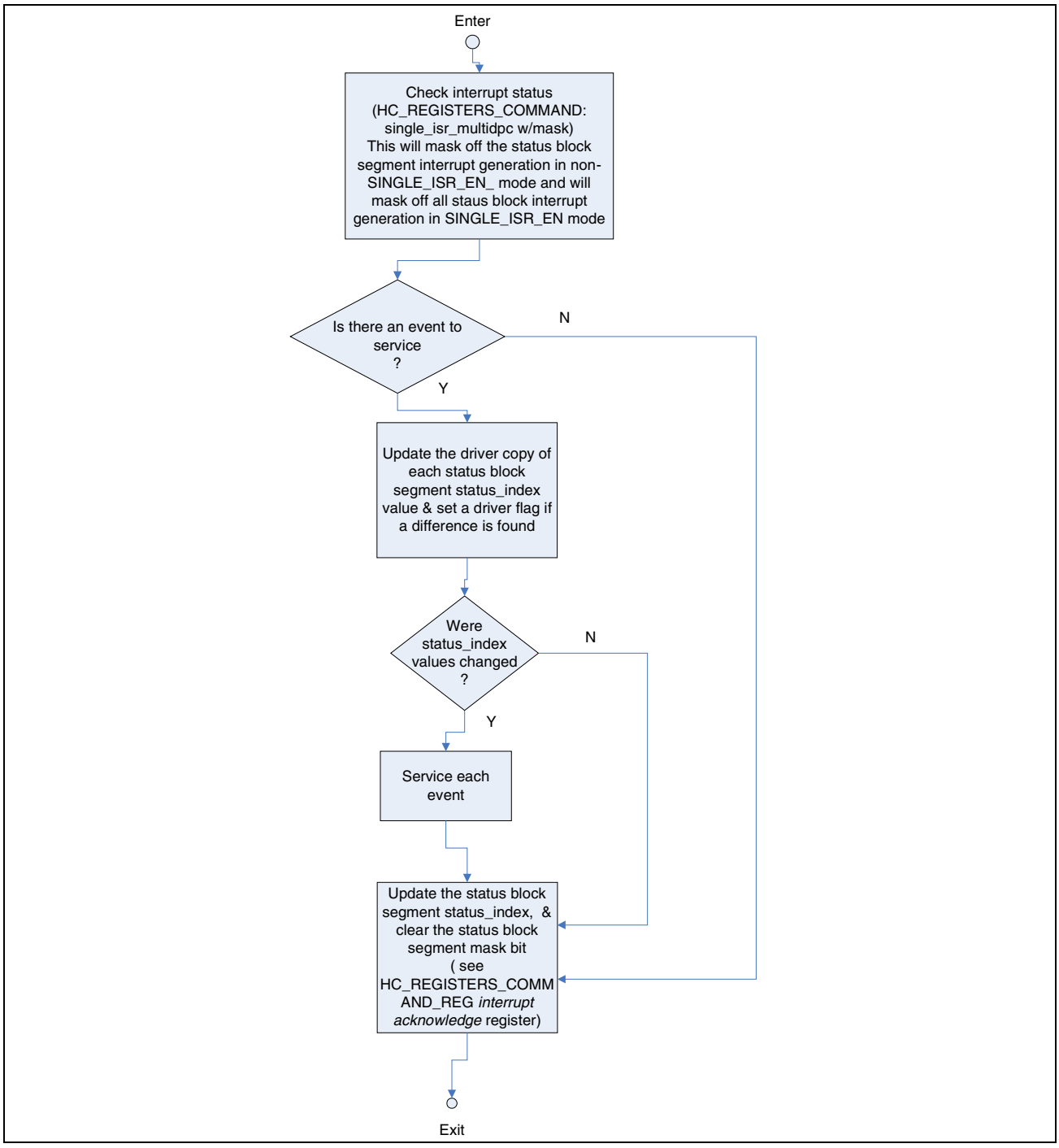

Figure 14: Handle Interrupt Flow

HC REGISTERS

HC_REGISTERS_CONFIG_0 (Offset: 0x108000; Width: 32)

Table 15: HC_REGISTERS_CONFIG_0 (Offset: 0x108000; Width: 32)

Bits	Name	Description	Access Mode	Reset Value
31:13	RESERVED		RO	0x0
12	MSI_MSIX_MEMORY_EN_0	Configuration register function 0; msi/msix memory is enable; if clear rd/wr to/from msi/msix memory is disable.	RW	0x1
11	STATISTIC_COUNTER_EN_0	Configuration register function 0; statistic counters enable; if clear rd/wr to/from statistic counters is disable.	RW	0x0
10:9	MAILBOX_COUNTER_0	2 bits for statistic counters configuration: 00 = no counting; 01 = count only producer updates; 10 = count only consumer updates; 11 = Count both producer and consumer updates.	RW	0x0
8	MSIX_ATTN_EN_0	UNUSED.	RW	0x0
7	MSI_ATTN_EN_0	Configuration register function 0; when this bit is set the msi_attn (form PCIE bar) is enable.	RW	0x1
6	COALESCE_NOW_EN_0	UNUSED.	RW	0x0
5	NOT_DURING_INT_EN_0	UNUSED.	RW	0x0
4	ATTN_BIT_EN_0	Configuration register function 0; if clr no attention msg will be send.	RW	0x0
3	INT_LINE_EN_0	Configuration register function 0; if clr interrupt line (INTA) is disable.	RW	0x0
2	MSI_MSIX_INT_EN_0	Configuration register function 0; if clr no msi/msix msg will be send.	RW	0x0
1	SINGLE_ISR_EN_0	Configuration register function 0; if set HC work in single ISR mode.	RW	0x0
0	BLOCK_DISABLE_0	Configuration register function 0; block enabled; if set the block is disabled; only rd/wr from direct register is enabled when this bit is set.	RW	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108180) - Interrupt Acknowledge Port 0

Table 16: HC_REGISTERS_COMMAND_REG (Offset: 0x108180) - Interrupt Acknowledge Port 0

Bits	Name	Description	Access Mode	Reset Value
31:27	Reserved		WO	0x0
26:25	tmp_dis_enable_cmd	If *tmp_dis_enable_cmd* == 0, set mask bit for *tmp_cmd_status_id*. If *tmp_dis_enable_cmd* == 1, clr mask bit for SB *tmp_cmd_status_id*. Else - ignore	WO	0x0
24	tmp_upd_index_cmd	If set the tmp_index_val value will be written to the appropriate consumer according to (*tmp_cmd_storm_index* and *tmp_cmd_status_id*) .	WO	0x0

Table 16: HC_REGISTERS_COMMAND_REG (Offset: 0x108180) - Interrupt Acknowledge Port 0

Bits	Name	Description	Access Mode	Reset Value
23:21	tmp_cmd_storm_index	Status block idx table Status block idx: 0 Status block name: First status block Status block idx: 1 Status block name Second status block . . . Status block idx: 15 Status block name: 16th status block Status block idx: 16 Status block name: Default status block	WO	0x0
20:16	tmp_cmd_status_id	STORMs/Attn idx table STORMs index: 0 STORM name: USTORM STORMs index: 1 STORM name: CSTORM STORMs index: 2 STORM name: XSTORM (default SB only!) STORMs index: 3 STORM name: TSTORM (default SB only!) STORMs index: 4 STORM name: Attn (default SB only1) STORMs index: 5-7 STORM name: Reserved	WO	0x0
15:0	tmp_index_val	The consumer index	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108184) - Producer Update Port 0

Table 17: HC_REGISTERS_COMMAND_REG (Offset: 0x108184) - Producer Update Port 0

Bits	Name	Description	Access Mode	Reset Value
31:27	RESERVED		WO	0x0
26:25	tmp_dis_enable_cmd	If *tmp_dis_enable_cmd* == 0, set mask bit for *tmp_cmd_status_id*. If *tmp_dis_enable_cmd* == 1, clr mask bit for SB *tmp_cmd_status_id*. Else - ignore	WO	0x0

Table 17: HC_REGISTERS_COMMAND_REG (Offset: 0x108184) - Producer Update Port 0 (Cont.)

Bits	Name	Description	Access Mode	Reset Value
24	tmp_upd_index_cmd	If set the tmp_index_val value will be written to the appropriate producer according to (*tmp_cmd_storm_index* and *tmp_cmd_status_id*).	WO	0x0
23:21	tmp_cmd_storm_index	Status block idx table Status block idx: 0 Status block name: First status block Status block idx: 1 Status block name Second status block . . . Status block idx: 15 Status block name: 16th status block Status block idx: 16 Status block name: Default status block	WO	0x0
20:16	tmp_cmd_status_id	STORMs/Attn idx table STORMs index: 0 STORM name: USTORM STORMs index: 1 STORM name: CSTORM STORMs index: 2 STORM name: XSTORM (default SB only!) STORMs index: 3 STORM name: TSTORM (default SB only!) STORMs index: 4 STORM name: Attn (default SB only1) STORMs index: 5-7 STORM name: Reserved	WO	0x0
15:0	tmp_index_val	The producer index	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108188) - Attention Bit Update Port 0

Table 18: HC_REGISTERS_COMMAND_REG (Offset: 0x108188) - Attention Bit Update Port 0

Bit(s)	Name	Description	Access Mode	Reset Value
31:16	RESERVED		WO	0x0
15:0	tmp_index_val	The attention bit value	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x10818C) - Attention Bit Set Port 0

Table 19: HC_REGISTERS_COMMAND_REG (Offset: 0x10818C) - Attention Bit Set Port 0

Bit(s)	Name	Description	Access Mode	Reset Value
31:16	RESERVED		WO	0x0
15:0	tmp_index_val	Every bit that is set will set the appropriate bit in the attention bit register	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108190) - Attention Bit Clear Port 0

Table 20: HC_REGISTERS_COMMAND_REG (Offset: 0x108190) - Attention Bit Clear Port 0

Bit(s)	Name	Description	Access Mode	Reset Value
31:16	RESERVED		WO	0x0
15:0	tmp_index_val	Every bit that is set will clear the appropriate bit in the attention bit register	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108194) - Coalesce Now Port 0

Table 21: HC_REGISTERS_COMMAND_REG (Offset: 0x108194) - Coalesce Now Port 0

Bit(s)	Name	Description	Access Mode	Reset Value
31:0	RESERVED	Writing to this address will cause the HC to send coalesce now command to all the storms.	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108198) - Single_isr_multi_dpc With Mask Port 0

Table 22: HC_REGISTERS_COMMAND_REG (Offset: 0x108198) - Single_isr_multi_dpc With Mask Port 0

Bit(s)	Name	Description	Access Mode	Reset Value
31:17	RESERVED		RO	0x0
16:0	tmp_index_val	The single ISR register value. Every bit that is set will be cleared in the mask register	RO	0xXX

HC_REGISTERS_COMMAND_REG (Offset: 0x10819C) - Single_isr_multi_dpc Without Mask Port 0

Table 23: HC_REGISTERS_COMMAND_REG (Offset: 0x10819C) - Single_isr_multi_dpc Without Mask Port 0

Bit(s)	Name	Description	Access Mode	Reset Value
31:17	RESERVED		RO	0x0
16:0	tmp_index_val	The single ISR register value.	RO	0xXX

HC_REGISTERS_COMMAND_REG (Offset: 0x108200) - Interrupt Acknowledge Port 1

Table 24: HC_REGISTERS_COMMAND_REG (Offset: 0x108200) - Interrupt Acknowledge Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
31:27	RESERVED		WO	0x0
26:25	tmp_dis_enable_cmd	If *tmp_dis_enable_cmd* == 0, set mask bit for *tmp_cmd_status_id*. If *tmp_dis_enable_cmd* == 1, clr mask bit for SB *tmp_cmd_status_id*. Else - ignore	WO	0x0
24	tmp_upd_index_cmd	If set the tmp_index_val value will be written to the appropriate consumer according to (*tmp_cmd_storm_index* and *tmp_cmd_status_id*).	WO	0x0
23:21	tmp_cmd_storm_index	Status block idx table Status block idx: 0 Status block name: First status block Status block idx: 1 Status block name Second status block . . . Status block idx: 15 Status block name: 16th status block Status block idx: 16 Status block name: Default status block	WO	0x0

Table 24: HC_REGISTERS_COMMAND_REG (Offset: 0x108200) - Interrupt Acknowledge Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
20:16	tmp_cmd_status_id	STORMs/Attn idx table STORMs index: 0 STORM name: USTORM STORMs index: 1 STORM name: CSTORM STORMs index: 2 STORM name: XSTORM (default SB only!) STORMs index: 3 STORM name: TSTORM (default SB only!) STORMs index: 4 STORM name: Attn (default SB only1) STORMs index: 5-7 STORM name: Reserved	WO	0x0
15:0	tmp_index_val	The consumer index	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108204) - Producer Update Port 1

Table 25: HC_REGISTERS_COMMAND_REG (Offset: 0x108204) - Producer Update Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
31:27	RESERVED		WO	0x0
26:25	tmp_dis_enable_cmd	If *tmp_dis_enable_cmd* == 0, set mask bit for *tmp_cmd_status_id*. If *tmp_dis_enable_cmd* == 1, clr mask bit for SB *tmp_cmd_status_id*. Else - ignore	WO	0x0
24	tmp_upd_index_cmd	If set the tmp_index_val value will be written to the appropriate producer according to (*tmp_cmd_storm_index* and *tmp_cmd_status_id*).	WO	0x0

Table 25: HC_REGISTERS_COMMAND_REG (Offset: 0x108204) - Producer Update Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
23:21	tmp_cmd_storm_index	Status block idx table Status block idx: 0 Status block name: First status block Status block idx: 1 Status block name Second status block . . . Status block idx: 15 Status block name: 16th status block Status block idx: 16 Status block name: Default status block	WO	0x0
20:16	tmp_cmd_status_id	STORMs/Attn idx table STORMs index: 0 STORM name: USTORM STORMs index: 1 STORM name: CSTORM STORMs index: 2 STORM name: XSTORM (default SB only!) STORMs index: 3 STORM name: TSTORM (default SB only!) STORMs index: 4 STORM name: Attn (default SB only1) STORMs index: 5-7 STORM name: Reserved	WO	0x0
15:0	tmp_index_val	The producer index	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108208) - Attention Bit update Port 1

Table 26: HC_REGISTERS_COMMAND_REG (Offset: 0x108208) - Attention Bit Update Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
31:16	RESERVED		WO	0x0
15:0	tmp_index_val	The attention bit value	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x10820C) - Attention Bit Set Port 1

Table 27: HC_REGISTERS_COMMAND_REG (Offset: 0x10820C) - Attention Bit Set Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
31:16	RESERVED		WO	0x0
15:0	tmp_index_val	Every bit that is set will set the appropriate bit in the attention bit register	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108210) - Attention Bit Clear Port 1

Table 28: HC_REGISTERS_COMMAND_REG (Offset: 0x108210) - Attention Bit Clear Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
31:16	RESERVED		WO	0x0
15:0	tmp_index_val	Every bit that is set will clear the appropriate bit in the attention bit register	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108214) - Coalesce Now Port 1

Table 29: HC_REGISTERS_COMMAND_REG (Offset: 0x108214) - Coalesce Now Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
31:0	RESERVED	Writing to this address will cause the HC to send coalesce now command for all the storms.	WO	0x0

HC_REGISTERS_COMMAND_REG (Offset: 0x108218) - Single_isr_multi_dpc With Mask Port 1

Table 30: HC_REGISTERS_COMMAND_REG (Offset: 0x108218) - Single_isr_multi_dpc With Mask Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
31:17	RESERVED		RO	0x0
16:0	tmp_index_val	The single ISR register value. Every bit that is set will be clear in the mask register	RO	0xXX

HC_REGISTERS_COMMAND_REG (Offset: 0x10821C) - Single_isr_multi_dpc Without Mask Port 1

Table 31: HC_REGISTERS_COMMAND_REG (Offset: 0x10821C) - Single_isr_multi_dpc Without Mask Port 1

Bit(s)	Name	Description	Access Mode	Reset Value
31:17	RESERVED		RO	0x0
16:0	tmp_index_val	The single ISR register value.	RO	0xXX

ATTENTION SIGNALS

The BCM57710/BCM57711 can generate up to 128 different attention signals each of which can potentially generate a system interrupt when asserted. The attention signal is a mechanism the BCM57710/BCM57711 uses to notify software or firmware that an event has occurred that needs servicing, or that an error has occurred within the device or that a state has changed within the device.

These signals include those generated by each of the BCM57710/BCM57711 STate Optimized RISC Microprocessors (STORMs), most BCM57710/BCM57711 hardware blocks and signals for link state change and GPIO and timer attentions. The attention signals must be routed to a smaller set of attention status bits for easy software handling.

Each attention signal can be configured at device initialization time to assert and cause an attention via a status block update. The attentions signals can also be configured to cause an attention when the device monitors the rising and falling edge of the signal or both the rising and falling edge of the signal.

The 128 attention signals are mapped to the bits found in the MISC_REGISTERS_AEU_ENABLE(1-4)_FUNC_(0/1)_OUT_(1-8) registers that are used to enable and disable each signal. When a bit is set to 1, the corresponding signal is enabled to generate an attention. The attentions are able to generate interrupts when the attention signal is latched by the hardware and the host driver configured route bits are in the proper state.

ATTENTION ROUTING

The 128 signals are routed into 16 attention status bits per port located in the default status block. The default status block contains a sub-block or segment called the attention status block.

Each of the 8 lower bits of the attention register within the attention block segment is designed to map to one of the eight register groups of the MISC_REGISTERS_AEU_ENABLE(1-4)_FUNC(1-4)_OUT_(1-8) registers.

For example; if bit 0 of the attention register in the default status block was asserted/de-asserted for port 0, the host driver would need to check which of the enabled signals in the MISC_REGISTERS_AEU_ENABLE_(1-4)_FUNC_0_OUT_0 registers has changed. If bit 1 of the attention register was asserted/de-asserted for the same port then the signals that caused that attention would be enabled in the MISC_REGISTER_AEU_ENABLE(1-4)_FUNC_0_OUT_1 registers.

The driver will determine which signal caused the assertion/de-assertion by reading the 32 bit value in the MISC_REG_AEU_AFTER_INVERT_(1-4)_FUNC_(0/1) register and using the bit positions that correspond to the enabled attention bits.

SIGNAL MONITORING

The 128 attention signals are grouped into 8 signals that can be configured and monitored by the BCM57710/BCM57711 on the leading or trailing edges or both the leading and trailing edge. This allows the device to generate an interrupt on the attention assert high (1) signal or assert low (0) signal or both.

MASKING

The masking on/off of these attentions can be achieved by the host driver writing a 1/0 respectively to the MISC_REGISTERS_AEU_MASK_ATTN_FUNC_(0/1) register. Each bit in these registers map to each of the attention groups mentioned above.

Dynamic vs. Static Interrupt Groups

Sixteen bits within the MISC_REGISTERS_AEU_MASK_ATTN_FUNC_(0/1) register are available to mask dynamically configured attentions and static, hard-wired attentions. The lower 8 bits of the register are for masking the dynamic attention groups as explained above. The upper 8 bits are for masking the static attentions. The attentions signals that are considered dynamic or static are any attentions that the host driver enables in the MISC_REGISTERS_AEU_ENABLE_(X) register groups. There are 8 groups of these registers and the default status block attention register bits 0-7 map to them. The attentions referred to as static attentions correspond to bits 8-15 of the default status block attention bits. Both dynamic and static attentions are masked with the MISC_REGISTERS_AEU_MASK_ATTN_FUNC(0/1) register. Both types of attentions are configured in the same way, but are handled a little differently because the static attention bits are hard-wired to the registers shown in Table 32 and Table 33.

Table 32: Static Attention Routing for Function 0

Routing Table for Function 0		
Name	***Bit Position***	***Notes:***
NIG attention for port0	D8	See NIG specification for configuration of this signal.
SW timer#4 port0	D9	See misc. for more information regarding SW timer access.
GPIO#2 port0	D10	See misc. for more information regarding SW timer access.
GPIO#3 port0	D11	See misc. for more information regarding SW timer access.
GPIO#4 port0	D12	See misc. for more information regarding SW timer access.
General attn1	D13	Setting/clear of this signal is via AEU GRC.
General attn2	D14	Setting/clear of this signal is via AEU GRC.
General attn3	D15	Setting/clear of this signal is via AEU GRC.

Table 33: Static Attention Routing for Function 1

Routing Table for Function 1		
Name	***Bit Position***	***Notes:***
NIG attention for port1	D8	See NIG specification for configuration of this signal.
SW timer#4 port1	D9	See misc. for more information regarding SW timer access.
GPIO#2 port1	D10	See misc. for more information regarding SW timer access.
GPIO#3 port1	D11	See misc. for more information regarding SW timer access.
GPIO#4 port1	D12	See misc. for more information regarding SW timer access.
General attn1	D13	Setting/clear of this signal is via AEU GRC.
General attn2	D14	Setting/clear of this signal is via AEU GRC.
General attn3	D15	Setting/clear of this signal is via AEU GRC.

Attention Initialization by the Host Driver

The steps taken in the host driver to initialize the attention signals include:

1. The host driver masks off the mask register by writing zeros to the bits in the following register:
   MISC_REGISTERS_AEU_MASK_ATTN_FUNC_(0/1)
   Setting the bits to 1 allows the attentions to propagate to system interrupts. The attention signals should be cleared so that no attention occurs during the device initialization. The E1.5 has two registers that mask the attention bits, one for the 8 LSBs and one for the 8 MSBs.

2. The host driver enables/disables the attention signals of interest in the MISC_REGISTERS_AEU_ENABLE(1-4)_FUNC_(0/1)_OUT_(0-7) registers. The configuration of these registers is by the host driver writing configuration values to these registers.
3. The attentions signals are configured to assert on a leading or trailing edge or both. This configuration is done in the HC_REGISTERS_TRAILING_EDGE(0/1) and HC_REGISTERS_LEADING_EDGE(0/1) registers. When the both registers are configured, the attentions occur on both leading and trailing edge. The signal that is monitored by the BCM57710/BCM57711 on the leading edge is referred to as a signal assertion and the signal monitored by the BCM57710/BCM57711 on the trailing edge is referred to as a signal de-assertion.
   - The bits in these registers correspond directly to the default status block attention segment bits.
   - When both leading and trailing edge are configured, the trailing edge should be configured first. The host driver should also clear the ACK and ATTN bits in the HC registers.
4. The host driver masks on (unmasks) the appropriate bits in the following mask register: MISC_REGISTERS_AEU_MASK_ATTN_FUNC_(0/1).
5. Initialize associated default status block memory and variables. The host driver creates a variable to keep the last known attention state serviced (see Figure 15: "Attention States," on page 83).

Handling Attentions in the Host Driver

After the driver configures the attentions, the enabled attention signals are primed to assert and de-assert. The Linux driver and FreeBSD driver code are configured to monitor both the asserted and de-asserted signal. This is due to the design of the device. The drivers are designed to use the assertion signal to mask off the interrupt and then service most of them on the de-assertion signal.

The attention status block segment in the default status block contains an attention (Attn) register and an attention acknowledgment (Ack) register. The bits of the Attn register are the previously mentioned 16 bits of dynamically and static attentions and are used to indicate a signal attention. The Ack bits correspond to the Attn bits and are for the host driver to acknowledge the asserted or de-asserted signal.

The default status block attention segment is updated by the asserted or de-asserted signal depending on the state of the attention bit and acknowledgment bit of the signal in the attention and acknowledgement registers. It is the responsibility of the host driver to set the appropriate Ack bit when an assertion of de-assertion signal is detected.

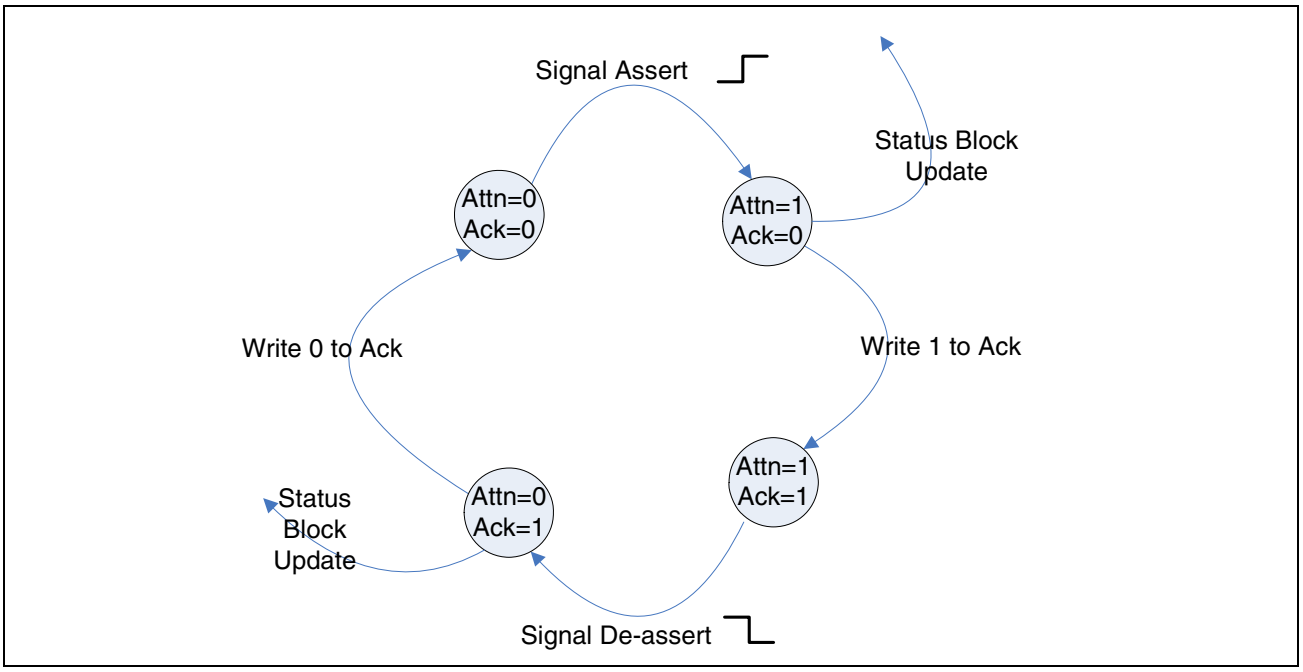

Figure 15: Attention States

Figure 15 represents the attention states and bit settings of the Attn and the Ack registers in the attention segment block of the default status block when the corresponding attention signals are monitored for both the leading and trailing edge latched.

Figure 15 also shows the state transition when the Attn and Ack bits are 0, a signal assertion will update the status block which will generate and interrupt. The signal will not generate another status block update until the host driver writes a 1 to the corresponding signal bit in the Ack register. Once the Ack bit is written, another status block update will follow on a de-asserted signal. Another assertion cannot take place unless the Ack bit is written with a 0 for that de-asserted signal. The BCM57710/BCM57711 device design of the attention signals require that this assertion/de-assertion transition take place. Subsequent status block updates for an attention signal state change and associated interrupt generation cannot occur unless the host driver acknowledges the assertion and de-assertion of each attention signal.

The steps taken by the host driver to service attention interrupts are as follows:

1. The host driver copies the status block attention segment content.
2. The host driver determines that an attention interrupt has occurred by checking the default status block Attention Bits status index.
3. The Attn bits and Ack bits from the attention register are read and compared to each other and to the last state saved by the host driver to determine if any states changes occurred and which bits are in asserted and de-asserted states. The host driver can determine the asserted and de-asserted states of the bits by the following method:
   - Asserted bits = Attn bits & ~Ack bits & ~last saved state.
   - De-asserted bits = ~Attn bits & Ack bits & last saved state
4. Update the asserted state driver variable by adding the current asserted state with the last saved asserted state so the old bit that have not been processed yet are not lost when adding the new bits.
5. The asserted bits are masked off to disable the interrupt by writing zeros to the MISC_REGISTERS_AEU_MASK_ATTN_FUNC_(0/1) register.

6. Handle the hard-wired attentions which include the link attention found in the NIG registers. The NIG register mask is used to disable the interrupt on the NIG if a link attention is asserted. The 57711 device includes an additional register called the MISC_REGISTERS_AEU_MASK_ATTN_FUNC_0/1_MSB register that masks the eight most significant bits of the attention register that can be used in place of clearing the NIG register.
7. Ack the serviced asserted attentions by writing to the HC_REGISTERS_COMMAND attention_bit_set_port_(0/1) register.
8. Handle the de-asserted attentions by searching through the eight groups of enabled attentions. The host driver reads the MISC_REGISTERS_AEU_AFTER_INVERT(1-4)_FUNC(0/1) registers to identify the asserted signals. See the Linux or FreeBSD driver source code.
9. Ack the serviced attentions by writing to the HC_REGISTERS_COMMAND attention_bit_clear_port(0/1) register.
10. If the attention signals have been disabled, ensure that the attention signals of interest are enabled in the MISC_REGISTERS_AEU_ENABLE(1-4)_FUNC_(0/1)_OUT_(0-7) registers.
11. The de-asserted state must be saved to the last saved state to be used in the next interrupt attention service call.
12. Finally Ack the status block index using the status block segment copy taken in step 1 before returning from the ISR. Some attentions that required to be cleared in the specific block that generates the attention may need to be re-enabled such as the attentions generated from the NIG block.

L2 TRANSMIT FLOW

This section describes the ASIC, Firmware, and Driver Flows for a transmit packet.

ASIC/FIRMWARE FLOW

The L2 packet Tx flow inside the ASIC is shown in Figure 16. The various steps in the flow are described below.

1. The doorbell command is how the driver delivers packets to the device to transmit onto the network. The doorbell commands from the driver are queued in the doorbell queue (DQ). For each command, the DQ loads the context to the CFC and then sends the context update to the XCM.
2. The XCM updates the context according to the aggregation rules. It then applies a decision algorithm to either queue the connection in the XQM or wake up the XSTORM (Transmitter STORM) for processing of that connection.
3. When the Max PBF credit required for processing a connection is available, the XCM will request the XQM for the connection to be processed. The XQM selects the next connection to be processed according to the configured arbitration rules, gets the context of that connection from the CFC, and then sends a message and the context to the XSTORM.

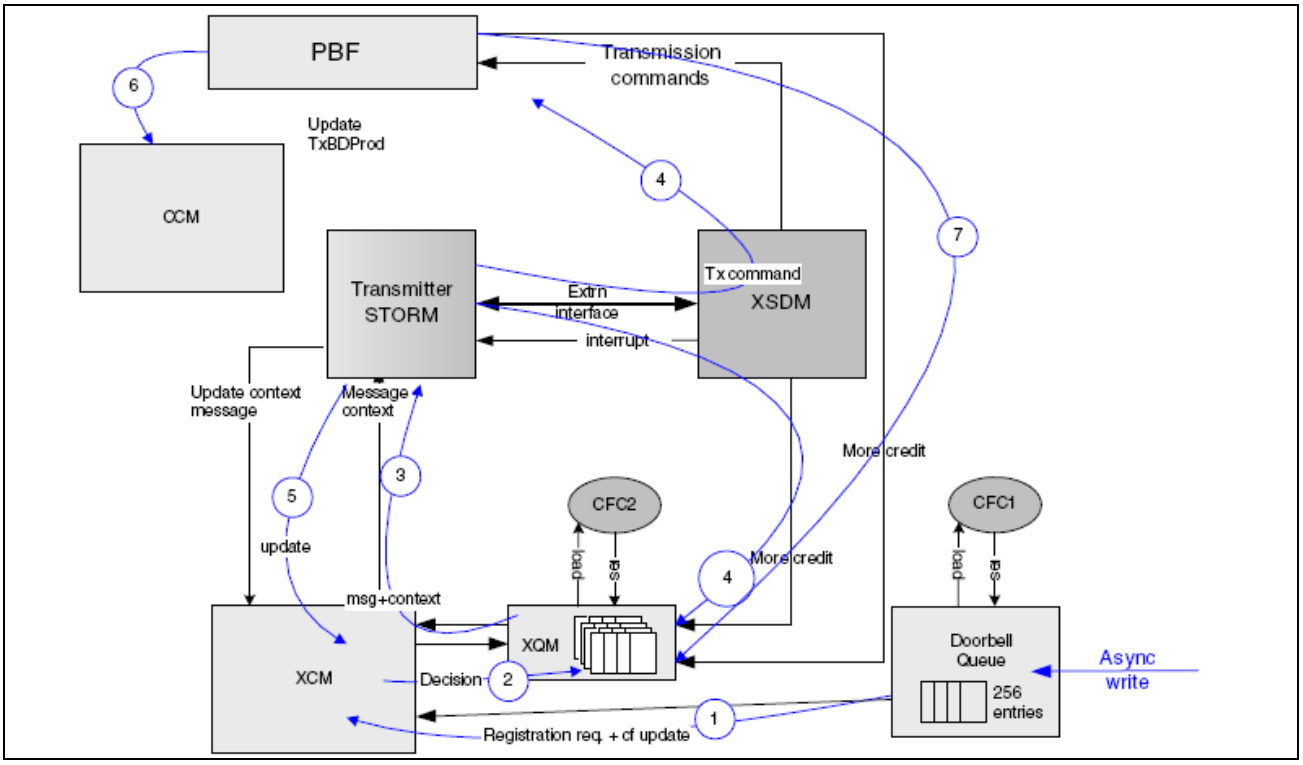

Figure 16: L2 Tx Packet Flow Inside the ASIC

4. After the CFC sends a message and the context to the XSTORM:
   a. The XSTORM requests the XSDM to fetch new BDs to its context whenever there are not enough BDs for at least one packet.

b. The XSTORM sends the Tx Command through the XSDM. In case of an LSO packet with NumOfBdsInPacket > LOCAL_RING_SIZE (13 BDs), the XSTORM may send multiple commands to the PBF.

c. The XSTORM also gives back the PBF credit that was allocated but not necessary for transmitting the given packet.

5. The XSTORM updates the context in the XCM.

6. The PBF sends the packet and sends a message to the CCM. The CCM alerts the CSTORM and the CSTORM will send an update to the HC block indicating the completion of packet transmit operation.

7. Upon reading a command from the PBF command queue, the PBF releases the PBF credit associated with that command back to the XQM.

Driver Flow

As part of the initialization, the driver will allocate host memory for Tx BD and Tx BD chain(s). The driver also maintains local copies of the Producer and Consumer Indices for all the Tx BD chains used. The BCM57710/BCM57711 supports up to 16 Tx BD chains for L2 Tx traffic.

Figure 17 shows the BCM57710/BCM57711 driver Tx flow for L2 packets. For L2 Tx packets, the driver gets the required number of BDs from the free Tx BD list, copies the packet data from the given packet buffers in to BD buffers, prepares the BDs, and rings the device's doorbell. If the given packet is either an LSO packet, GSO packet or TCP/UDP/IP checksum offloaded packet, then the packet data will also be parsed to form the Parsing-Info BD which will be added as the second BD for a given packet.

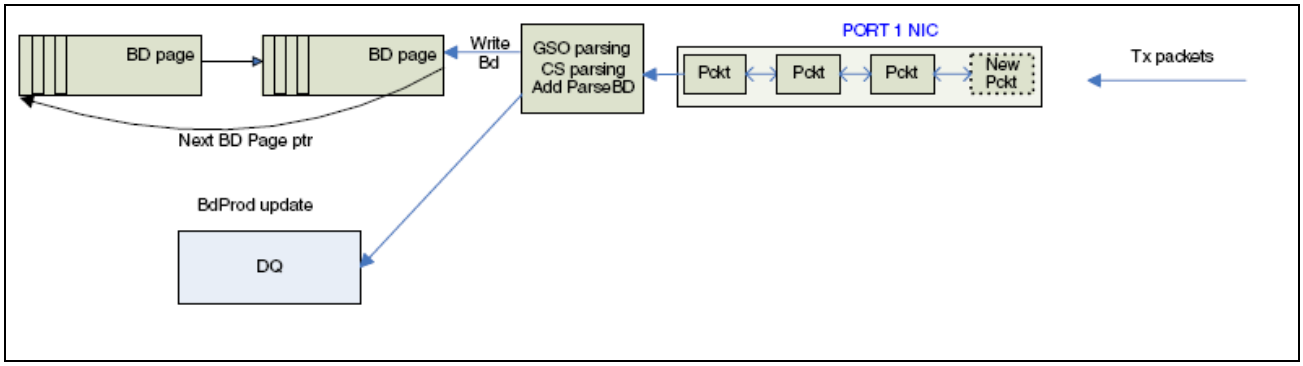

Figure 17: L2 Tx Packet Flow

Listed below are more detailed driver steps for transmitting a given packet.

Handle short buffer merging:

- If a given packet contains multiple short buffers and requires more than the maximum number of BDs that can be fetched by the device for a Tx packet as limited by the XSTORM (which is 13 BDs), then the driver may need to merge the short buffers such that the required number of Tx BDs is less than 13 BDs.
  - The driver should maintain a pre-allocated pool of buffers for facilitating this short buffer merging.
  - For LSO, the driver should join the buffers such that the BD List size per MSS is less than the 13 BDs. See "Large Send Offload" on page 103 for more information on sending LSO packets.

Note: This short buffer copying is not required if the OS can guarantee that a given packet would not use more than 13 buffers.

Get the free Tx BD(s) from Tx BD Chain and increment the local copy of Tx Producer Index.

Prepare the First Tx BD:

- Set the START_BD flag in the bd_flags field.
- If the given packet is a tagged packet, add the VLAN information delivered by the Host OS stack to the first BD and set the VLAN_TAG flag.
- Set the addr_hi and addr_lo fields with the Host Address of Tx Buffer.
- If the IP checksum offload is enabled, set the IP_CSUM flag.
- If the TCP checksum offload is enabled, set the TCP_CSUM flag.
- If the UDP checksum offload is enabled, then also set the TCP_CSUM flag.
- Set the hdr_nbds with the Number of BDs containing Ethernet/IP/TCP headers. Currently the firmware requires one Tx Buffer to contain all the headers, so make sure that all the headers are in the Tx Buffer pointed to by first Tx BD and set this hdr_nbds field to 1.
- Set the length of Tx Buffer into nbytes field.
- Set the nbd field with the number of Tx BDs for this packet. Note that this count also includes the parsing Info BD if it is present.
- If LSO is enabled, set the SW_LSO flag. If the first Tx BD contains headers (Ethernet, IP, and TCP) and data, then split it into to 2 Tx BDs with the first Tx BD containing headers and the second Tx BD containing the data. This split is to aid firmware processing of LSO/GSO packets. For the first BD containing the headers, increment the nbds field and set the nbytes to the value of length of headers. For the second BD containing the data, set the addr_hi and addr_lo fields to the Host Address of Tx Buffer, the nbytes to the length of data, and the SW_LSO flag of bd_flags to

Note: This second Tx BD arrives immediately after the Parsing Info BD in the BD list for a Tx packet.

- Set the END_BD flag if the first BD is also the last BD for a given packet.

Prepare the Parsing-Info BD if the packet is an LSO or CS (Check Sum) offload ('TCP and IP' or 'UDP and IP' or IP) packet: The parsing BD information is used by the XSTORM to configure the PBF block that builds and frames the packet to be given to Phy for transmission on the wire.

- If the packet has LLCSNAP encapsulation, raise the LLC_SNAP_EN flag in global_data field.
- Calculate the Ethernet Header Length and write it to the IP_HDR_START_OFFSET of the global_data field.
- If the packet is a TCP packet, set the tcp_flags as delivered by the Host OS.
- Calculate the IP Header Length and write it to ip_hlen field.
- If the packet is a TCP packet and TCP checksum offload is enabled, write the TCP pseudo-header checksum to the tcp_pseudo_csum field. If the OS stack does not provide this pseudo header checksum, then the driver must calculate it.
- If the packet is an UDP packet, set the UDP_FLG of global_data field. If the UDP checksum is offloaded, then calculate the pseudo header checksum and place it in the UDP checksum field of the packet data.
- Calculate the total length of Ethernet, IP, and TCP (or UDP) headers and write it to total_hlen field.
- If LSO is enabled, do the following:
  - Configure the lso_mss field with the value given by OS.
  - Configure the tcp_send_seq field with the sequence number from the TCP header.
  - Configure the ip_id field with the value from the IP header.
  - Calculate the pseudo header checksum with length field set to 0 and write it to the tcp_pseudo_csum field. Also, set

the PSEUDO_CS_WITHOUT_LEN flag.

Prepare the remaining Tx BDs for the given packet:

- Set the addr_hi and addr_lo fields with the Host Address of Tx Buffer.
- Set the length of Tx Buffer pointed by this BD into nbytes field.
- Set the bd_flags to zero.
- Make sure the END_BD flag is set for the last BD of a given packet.
- Ring the doorbell: The driver sends a doorbell to BCM57710/BCM57711 by writing the Doorbell entry to the doorbell address of L2 connection (aka NIC connection).

Tx Interrupt Handling

The driver maintains a mirror copy for each of the Status Blocks. Upon receiving an interrupt from HC block, the driver compares the updated Status Block to its mirror copy to determine the indices that have changed. Figure 18 shows the high level view of Tx packet completion for both the Legacy INTA# and MSI-X interrupt modes. In the Legacy INTA# mode, the INTA wakes up an arbitrary CPU which starts the main interrupt handler. The main interrupt handler determines the CPUs whose BD Chain indices have changed and schedules Deferred Procedure Calls (DPCs), or task threads, for those CPUs to start their interrupt handlers for completing the Tx packets and freeing the Tx BDs. In the case of the MSI-X mode, the device directly wakes up each of the CPUs whose indices have changed and each CPU that is awaken by the device will start an interrupt handler for completing the Tx packets in its Tx BD Chain and freeing the Tx BDs.

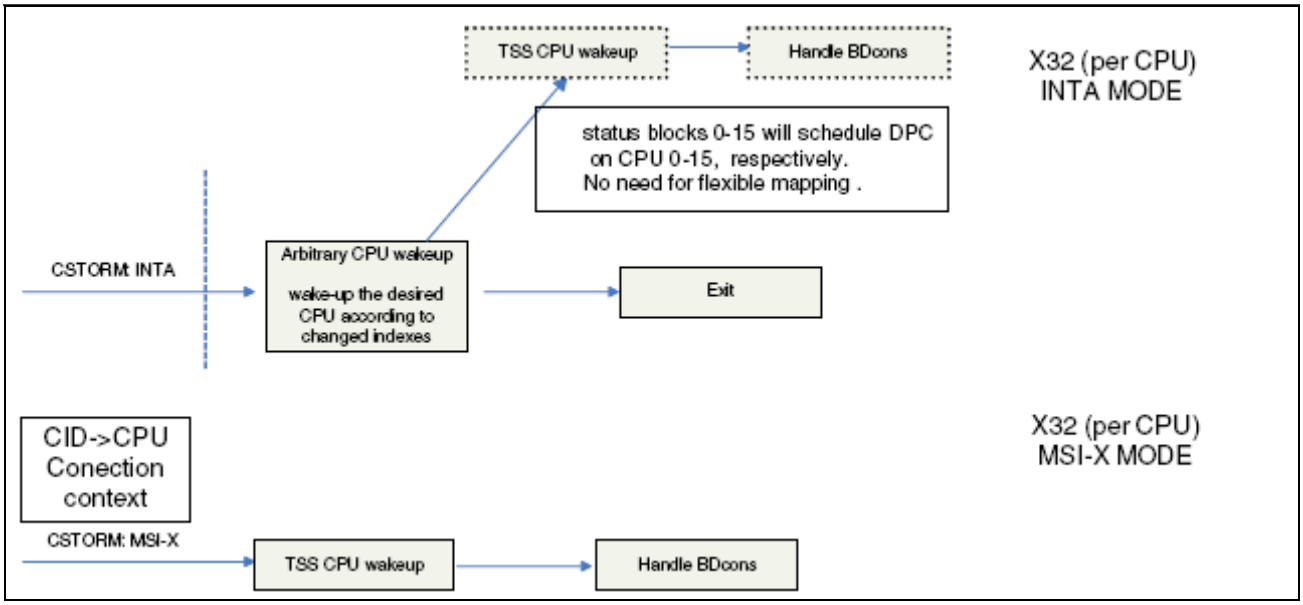

Figure 18: Tx Packet Completion

For each updated BD chain, the driver:

1. Determines if there are packets to be completed by comparing the local copy of Tx BD Chain Consumer Index with the Tx BD Chain Consumer Index in updated Status Block.
2. For each packet, that needs to be completed:
   a. Finds the Number of BDs used for a given Tx packet by reading the nbd field of the first BD of that Tx packet. Note that the local Consumer Index always points to this first BD of a Tx packet that needs to be completed first by the

driver.

b. Releases the DMA mapping of the Tx Buffers used by the BDs and frees them.

c. Frees the Tx BDs by advancing the local Consumer Index of Tx BD Chain by the Number of BDs used for the packet being completed.

d. Sends the completion notification to upper layers.

Note: When using only one Tx BD Chain (No Tx Scaling and Tx Load Distribution), all indices are available in the default status block. When using the per-CPU Tx-BD Chain for Tx Scaling, the per CPU status block should be used for determining the BD Chain hardware consumer indices.

L2 RECEIVE FLOW

This section describes the ASIC, Firmware, and Driver Flows for a receive packet.

ASIC FLOW

All the steps outlined in this procedure are not necessarily sequential as all the state-machines and STORM processors in the datapath are optimized to handle parallel processing for a single packet and many packets wherever possible. However, the device will ensure the in-order delivery of the packets to the host per connection. In the L2 mode, all the packets are treated as one single NIC connection and hence all the L2 packets will be delivered to the Host in the order they are received.

1. The SerDes passes the Rx packet to the NIG and the NIG places it into the BRB.

2. The BRB sends a notification to the Parser as soon as the packet header is placed into the BRB.

3. The Parser parses the header and requests the Searcher to find the corresponding connection.

4. The Searcher determines that it is a NIC connection. Note that all L2 packets are treated as a single NIC connection which has a fixed CID (Connection ID).

5. The Parser requests the CFC to load the TSTORM context using the CID of the leading connection.

6. The Parser allocates a Serial Number for the packet and sends a Packet Start message to the TCM, together with the allocated Serial Number. This serial number is used later by the BRB to release the buffers occupied by a given packet.

7. The Parser starts calculating the TCP checksum of the packet

8. The Packet Start message is a non-aggregative message, and hence TCM passes it directly to the TSTORM.

9. The TSTORM either forwards or drops the packet by checking for any errors in the packet data and applying the unicast, multicast, and VLAN filtering rules. If it decides to forward the packet, it builds a Packet Start message for the USTORM. If it drops the packet, then it also releases the Serial Number and the BRB buffers.

10. The TSTORM sends the Packet Start message to the USTORM.

11. The TSTORM checks in its internal RAM whether a Packet End message for this packet has also arrived. If it finds that the Packet End has also arrived, then it jumps to step 15, otherwise it will continue with the next step.

12. The TSTORM thread goes to sleep and requests the TSDM to be wakened when the Packet End message for the given packet has arrived.

13. The Parser completes the TCP checksum calculation and sends the Packet End message to the TSDM.

14. The TSDM wakes up the TSTORM thread for which Packet End has arrived using the Serial Number in the message.

15. The TSTORM forwards the Parser Packet End message to the USTORM.

16. The USTORM reads the Parser message from its internal memory based on the Serial Number of the packet, and sends a message (through the USDM) to the Parser to release the Serial Number.

17. The USTORM requests the BRB to release the buffers if it decides to drop the packet rather than forwarding it to the Host.

18. The USTORM forms and sends the UPB command to the UPB block (Packet Builder Block used with the USTORM for placing the packet data into host memory) to deliver the packet to the Host.

19. The UPB block requests the PXP block to DMA the packet data from the BRB to Host memory.

20. After the DMA of the packet data is done, the USTORM requests the BRB to release the BRB buffers.

TSTORM

- The TSTORM filters the packets based on Unicast and Multicast MAC Addresses configured into the TSTORM CAM Table Entries. The TCAM supports a total of 96 entries per port.
- The TSTORM filters the packets based on the VLAN configuration of the port. Table 34 summarizes how the TSTORM handles the VLAN filtering.

Table 34: VLAN Filtering Rules

Configured VLAN ID for NIC	*Packet Contains a tag header?*	*VLAN ID in the Tag header*	*TSTORM Action*
Zero (filter bypass)	yes	Any value	Remove VLAN Tag from the packet if the Enable VLAN Removal bit of the port is set. Insert the stripped VLAN tag into BD and set the VLAN bit in the WQE.
Zero (filter bypass)	No	Not applicable	Unset the VLAN bit in the WQE.
Nonzero	Yes	Matches the configured VLAN ID	Remove VLAN Tag from the packet if the Enable VLAN Removal bit of the port is set. Insert the stripped VLAN tag into BD and set the VLAN bit in the WQE.
Nonzero	Yes	Does not match the configured VLAN ID	Drop the packet
Nonzero	No	Not applicable	Drop the packet

- Drops the packets if there is an Rx error as indicated by the NIG block and the port is configured to drop the packets with any Rx error.
- If RSS is enabled, The TSTORM calculates the RSS Hash based on the configured 'RSS Hash Type' and finds the actual CID for that packet using the RSS Indirection Table.
- Sends Start Packet Messages to the USTORM
- Writes End Packet Messages directly to USTORM internal memory using the Serial Number assigned to a packet.

USTORM

- Fetches Rx BDs if the local Rx BD Ring is empty.
- Removes the VLAN tag if the device is configured to perform this action.
- Pads the packet with zeros if a packet becomes a short packet (length less than 64 Bytes) after the VLAN tag is removed. For GVRP and LACP packets, removal of the VLAN field is not allowed, even if the port is configured for VLAN Tag removal. The driver should set the Override VLAN Tag Removal when setting the GVRP and LACP destination MAC addresses (the destination MAC address of GVRP is 0180-C2-00-00-21 and the destination MAC address of LACP is 01-80-C2-00-00-02).

- Checks for any errors (TCP/IP checksum errors and Rx Errors) in Packet End messages and drops the error packets if the port is configured to do so.
- Forms the UPB command and sends it to UPB block so that UPB block can handle the DMA of Packet Data and RCQ entry to the Host Memory. Updates the local RCQ Producer Index and writes it to the HC block.
- Builds RCQE using the information from the Packet Start and Packet End messages and writes the RCQE to RCQ according to the RSS CPU_ID.

DRIVER FLOW

For Rx of L2 packets, the device driver maintains an Rx BD Chain (aka Receive Queue) and an Rx Completion Queue (RCQ). The BCM57710/BCM57711 supports up to 16 Rx BD Chains for L2 traffic. For each BD Chain, the driver maintains an associated RCQ. Multiple Rx Rings can be used to distribute the Rx Load across multiple CPU cores if the Host system has multiple CPU cores. Figure 19 on page 92 shows the high level Driver Flow of Rx packets. Note that only one Rx BD Chain (RQ) and RCQ are shown.

During the initialization, the driver:

- Allocates the memory for RQ and initializes it such that the last Rx BD element of each page in the Chain will point to the next page in the Chain.
- Allocates the memory for RCQ and initializes it such that the last RCQE of each page in the Chain will point to the next page in the Chain.
- Writes the Host memory addresses of RQ and RCQ to the device.
- For each chain (RQ and RCQ), the driver maintains the Consumer and Producer Indices. Initializes all these indices to zero.
- Allocates memory for Rx Buffers. Associates an Rx buffer with each of the Rx BDs by updating the Rx BD fields. All the Rx BDs are now ready and can be given to the device. So update the software copies of RQ Producer Index and RCQ Producer Index with the available number of Rx BDs.
- Give all the prepared Rx BDs to the device by updating the RCQ Producer Index of the TSTORM in the device.

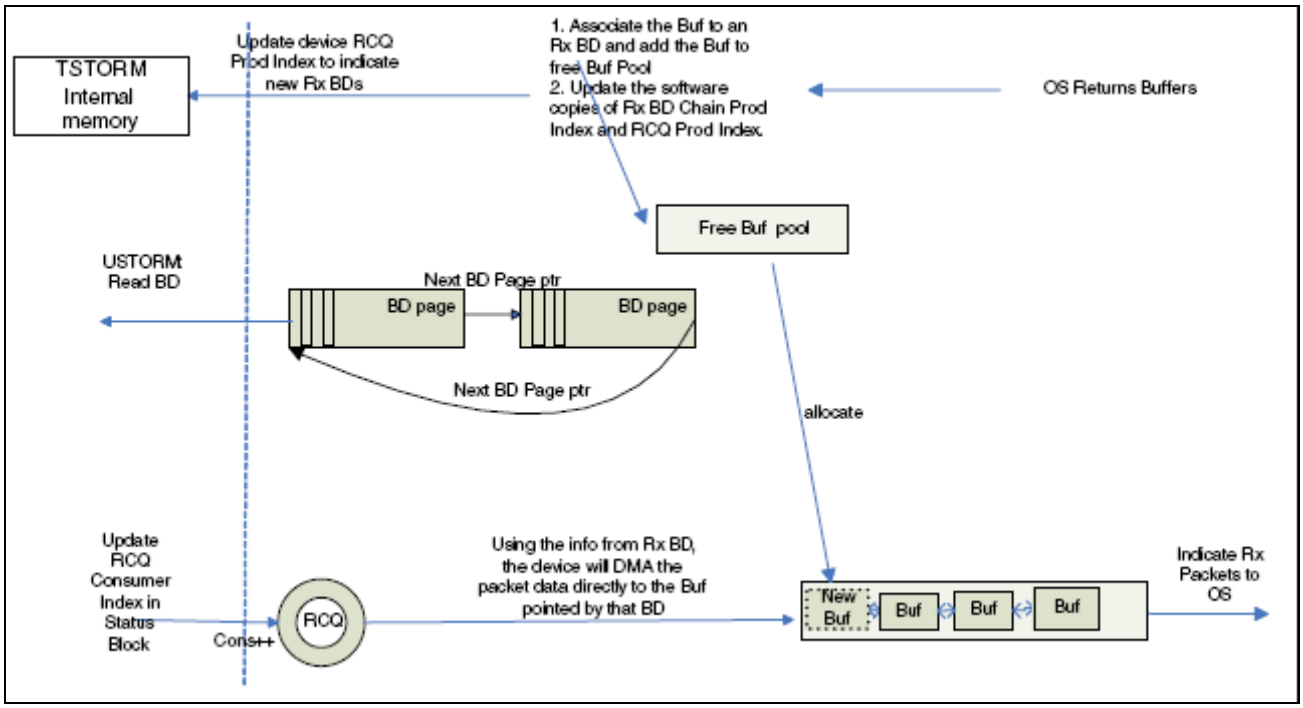

Figure 19: Rx Driver Flow

For each Rx packet the device will use one Rx BD and the RCQE. When the USTORM gets the Packet End message of that packet, it takes an Rx BD from the device's Rx Ring and builds and sends a command to the UPB block using the information from the Rx BD, Packet Start and Packet End messages. The USTORM also takes an entry from the RCQ associated with that Rx Ring and updates that RCQE with all the information required by the driver to complete processing of Rx packet. The UPB will handle the DMA of the packet data to Host memory. After the packet is DMAed to the Host memory and RCQE is written into the Host memory, the HC block will be notified of the updated RCQ Producer Index. When any of the coalescence conditions is met, the HC block will update the Status Block in Host memory and trigger an interrupt. Depending upon the configured mode of the interrupt, the device will generate either the INTA# interrupt or an MSI or MSI-X interrupt. As shown in Figure 20 on page 93, when an INTA# interrupt is used, the interrupt will wake up an arbitrary CPU or a fixed CPU which executes the main interrupt handler and schedules a DPC to the CPU as indicated by the RSS hash results. In MSI or MSI-X mode, the interrupt will directly wake up the CPU as determined by the RSS hash results.

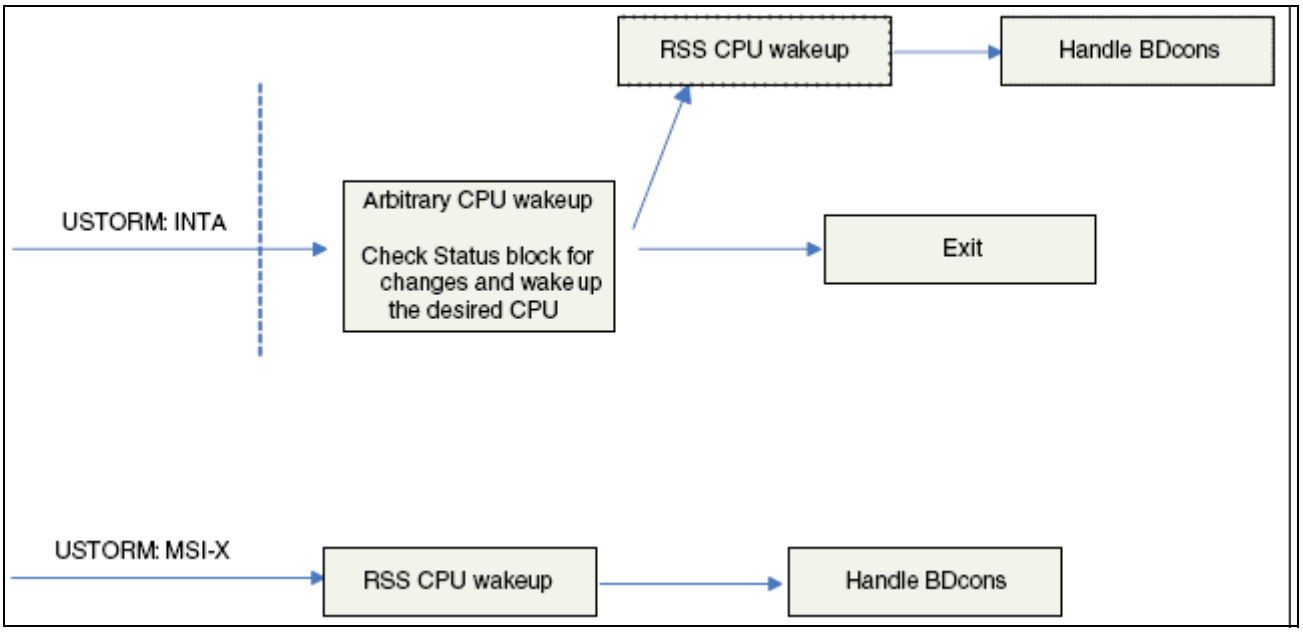

Figure 20: Rx Interrupt Flow

RX INTERRUPT HANDLING

The driver maintains a mirror copy for each of the Status Blocks used. Upon receiving an interrupt from the HC block, the driver compares the software copy of the RCQ Consumer Index with the hardware copy of the RCQ Consumer Index available in the updated Status Block. When the driver detects the software copy is not same as the hardware copy, then it knows there are new packets received and will schedule a DPC or task thread to process the received packets. Listed below is the step-by-step procedure of an Rx interrupt handler for each RCQ that has newly received packets.

1. Acquire the software copies of RQ and RCQ indices.
2. Use the software copy of the RCQ Consumer Index, and get the CQE.
3. Get the Rx Buf used for the Rx packet by using the software Rx BD Chain Consumer Index.
4. Check whether the CQE is a slow path completion entry or Rx completion entry. If the CQE is a slow path entry, then change the driver state depending upon the completed slow path event (such as ramrod). Increment the available SPQ commands, and jumps to step 13 to give the CQE back to the device.
5. If any error flag is set in the CQE, drop the packet and jump to step 11 to reuse the Rx Buf.
6. Acquire the Rx packet length and placement offset information from the CQE. This placement offset field tells the driver about the number of Bytes padded by the FW before the Ethernet header in order to make the L3 header aligned to a 16 Byte boundary.
7. Allocate an Rx Buf and associate it with the Rx BD of the RQ indexed by the Rx BD Producer Index by updating the Rx BD with the Rx Buf Host memory address and length. If the allocation of new Rx Buf fails, then either drop the Rx packet and jump to step 11 for reusing the Rx Buf of the dropped Rx packet or proceed with steps 8-9 and then exit the interrupt handler deferring the replenishing of Rx Bufs until OS returns the Rx Bufs.
8. Extract other information such as the L2 protocol, the VLAN tag when VLAN is enabled, and determine whether the TCP/IP checksum validation is accomplished by the HW. Provide this information to the OS through the OS specific headers associated with the OS specific data structure Rx Buf.

9. Indicate the new packet arrival to the OS.
10. Proceed to step 12.
11. Reuse the Rx Buf used by the Rx packet for the Rx BD of the RQ indexed by the Rx BD Producer Index by updating the Rx BD with the Rx Buf Host memory address and length.
12. Update the software copies of RX BD Chain Consumer Index and Producer Index.
13. Update the software copies of RCQ Producer Index and Consumer Index.
14. Indicates the updated RCQ Producer Index to the device. Recycling of fast path CQE must be accomplished only after recycling the BD.

If the interrupt handler defers the replenishing of the RCQ entries, then the driver will update the RCQ Producer Index only when the OS returns the Rx Bufs as shown in Figure 19 on page 92. In any case, the driver can implement a threshold and replenish the RCQ entries only when the number of available RCQ entries exceeds the threshold. This reduces the number of DMA operations fetching the Rx BDs. When Jumbo Frame support is enabled, as there is no separate Rx ring for receiving Jumbo Frames, the size of all Rx Bufs allocated should be at least the size of configured Ethernet Rx MTU. In this case, if memory usage efficiency is a concern, the driver may implement copying the Rx packets of size less than a given threshold into newly allocated small buffers and releasing Rx Bufs for reuse.

1 For slow path completion, only the CQE is consumed by the device and Rx BD is not consumed.
2 The driver should allocate the receive buffers in 16-byte aligned addresses as required by FW.

INTERRUPT COALESCING

The BCM57710/BCM57711 device has a transmit and a receive interrupt coalescing feature. The BCM57710/BCM57711 device interrupt coalescing functions are initialized by setting a timeout value in microseconds in the CSTORM and USTORM timer offsets for transmits and receive interrupts respectively (CSTORM_SB_HC_TIMEOUT_OFFSET and USTORM_SB_HC_TIMEOUT_OFFSET)

The microsecond value in the CSTORM and USTORM timeout offset has a 12 µs granularity.

For receive interrupt coalescing, the device generates an interrupt for the first incoming packet. At this point, the device sets a timer and collects all packets received within the interval of the time out value.

Example: If the device receives a packet and the interval is configured to be the minimum value (12 µs), and if the subsequent incoming packets arrive within the 12 µs interval, an interrupt is generated only when the timeout expires and not before. If the next incoming packet arrives 12 µs after the previous packet and after the previous timeout value was achieved, then a new interrupt is generated for that new packet and the timer interval begins again.

The transmit interrupt coalescing functions in the same way using transmitted packets.

TRANSPARENT PACKET AGGREGATION

This section describes the theory and implementation of Transparent Packet Aggregation (TPA) for NetXtreme II 10 GbE controllers.

GLOSSARY

- 4-Tuple - A combination of the IP source and destination address along with the TCP source and destination port.
- Aggregation - An aggregation is a TCP stream tracked by the network controller.
- LSO - Large Send Offload
- SGE - Scatter-Gather Entry
- SGQ - Scatter-Gather Queue
- TSO - TCP Segmentation Offload
- TOE - TCP Offload Engine
- TPA - Transparent Packet Aggregation

THEORY OF OPERATIONS

Transparent Packet Aggregation (TPA) is a technology that allows stateless offload of TCP connections to the network controller. Unlike transmit offload technologies such as LSO/TSO (Large Send Offload/TCP Segmentation Offload) which allow the network controller to breakup a single large TCP frame (say 64KB) into multiple, smaller frames appropriate for the physical network layer (say 1500 byte frames), TPA is a receive offload technology which allows the network controller to combine multiple, smaller TCP frames (say 1500 bytes) into a single, larger TCP frame (say 64KB) and pass the resulting larger frame to the network stack.

HOW DOES AGGREGATION WORK?

Whenever the network controller detects an incoming TCP frame it compares the 4-tuple value of that frame to an existing table of tracked TCP streams to determine if the TCP frame is part of an existing stream or a new stream. If it is a new stream the network controller will store the relevant TCP connection information internally and then pass the TCP frame to the host network driver, indicating that a new aggregation has begun. The network controller will then watch for additional TCP frames the previously identified 4-tuple value. When such a frame is received, the controller will perform several checks (see "When to Aggregate?" and "When to Stop Aggregation?" on page 97) to determine frame validity and to decide whether the new frame should be added to an existing aggregation, or be used to start a new aggregation. If the frame should be added to an existing aggregation, then the network controller writes the TCP payload to a buffer in the Scatter Gather Queue (SGQ).

When an aggregation is complete the network controller will indicate to the host network driver that an aggregation has completed and that it should be assembled into a buffer appropriate for the OS network stack. The driver is then responsible for combining the original TCP frame (which includes an updated TCP header) with additional entries from the SGQ to create a single, large TCP frame for the OS.

Implementation of TPA has several advantages in a network driver:

- The OS stack is relieved of the responsibility to verify each individual frame (including verifying the TCP checksum)
- The network controller is responsible for verifying TCP checksums for the entire aggregated frame.

WHEN TO AGGREGATE?

The network controller is responsible for determining when a received TCP frame can be aggregated. Only a frame which satisfies ALL of the following conditions will be aggregated.

- The frame does not contain an IPv4 checksum error.
- The frame does not contain a TCP checksum error.
- The frame does not have a TTL = 0.
- The frame is not larger than the programmed MTU.
- The frame is larger than 80 bytes
- The frame does not have an FCS error.
- The frame does not use LLC SNAP encapsulation
- The frame does not use IP options
- The frame is not an IP fragment
- The frame does not contain any TCP options other than timestamp.
- The frame does not contain any of the following TCP flags: FIN/SYN/RST/URG.

Each NetXtreme II controller supports a maximum number of aggregation queues per port (32 for the BCM57710 and 64 for the BCM57711). If a newly received frame is suitable for aggregation and all existing aggregation queues are already in use the network controller will randomly select an existing aggregation, send an indication to the host network driver that the existing aggregation is complete, and begin a new aggregation for the newly received TCP frame.

WHEN TO STOP AGGREGATION?

The network controller is responsible for determining when a TCP aggregation is complete. If a newly received frame matches ANY of the following conditions the current aggregation will be marked complete and indicated to the network driver.

- The TCP header length of the current frame does not match the TCP header length of the first TCP frame in the aggregation.
- The VLAN ID of the current frame does not match the VLAN ID of the first TCP frame in the aggregation.
- The TCP frame is out of order (i.e. the received sequence number does not match the expected sequence number).
- The acknowledged sequence number of the current frame is less than the acknowledged sequence number of the last frame in the aggregation.
- The expected size of the aggregation with the new frame is greater than the aggregation threshold. The aggregation threshold is calculated as (size_of_bd + size_of_sgl * size_of_sge - mtu).
- The TCP PUSH flag is set.
- Timeout

IMPLEMENTATION ASSUMPTIONS

It is assumed that the reader is familiar with basic operation of the Broadcom NetXtreme II 10Gb network driver. It is also required that the host OS is capable of supporting large received TCP frames. In FreeBSD, for example, the capability to accept large TCP frames is referred to as Large Receive Offload (LRO) and is available in FreeBSD version 6.5 and later..

The BCM57710/BCM57711 controllers use firmware extensively to implement many different features. The version of firmware used may change the size or definition of common host memory data structures such as the Completion Queue Entry (CQE). Developers writing their own driver should pay close attention to the Host Software Interface (HSI) definitions provided in any reference drivers as these definitions may change between firmware releases. Although this document will be update periodically to accommodate any new definitions the developer should always refer to the HSI definitions of the firmware they receive. The remainder of this document includes HSI data structures that were in use with firmware version 4.8.5.

TPA Implementation

The following sections document how to modify a NetXtreme II network driver to add TPA support.

Required Firmware Version

The ability to support TPA was added in BCM57710/BCM57711 firmware version 4.4.9

Firmware Data Structures

When enabling or initializing many features in the NetXtreme II 10Gb network controller the host device driver must update numerous firmware data structures used by the STORM processors in addition to the typical register reads/writes. The following describes which STORM processors are involved with TPA operation and the additional initialization required for that processor.

USTORM

The following USTORM context data structure is modified to enable TPA.

```
struct ustorm_eth_st_context_config {
#if defined(__BIG_ENDIAN)
```

```
    uint8_t flags;
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_MC_ALIGNMENT (0x1<<0)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_MC_ALIGNMENT_SHIFT 0
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_DYNAMIC_HC (0x1<<1)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_DYNAMIC_HC_SHIFT 1
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_TPA (0x1<<2)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_TPA_SHIFT 2
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_SGE_RING (0x1<<3)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_SGE_RING_SHIFT 3
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_STATISTICS (0x1<<4)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_STATISTICS_SHIFT 4
#define __USTORM_ETH_ST_CONTEXT_CONFIG_RESERVED0 (0x7<<5)
#define __USTORM_ETH_ST_CONTEXT_CONFIG_RESERVED0_SHIFT 5
    uint8_t status_block_id;
    uint8_t clientId;
    uint8_t sb_index_numbers;
#define USTORM_ETH_ST_CONTEXT_CONFIG_CQE_SB_INDEX_NUMBER (0xF<<0)
#define USTORM_ETH_ST_CONTEXT_CONFIG_CQE_SB_INDEX_NUMBER_SHIFT 0
#define USTORM_ETH_ST_CONTEXT_CONFIG_BD_SB_INDEX_NUMBER (0xF<<4)
#define USTORM_ETH_ST_CONTEXT_CONFIG_BD_SB_INDEX_NUMBER_SHIFT 4
#elif defined(__LITTLE_ENDIAN)
    uint8_t sb_index_numbers;
#define USTORM_ETH_ST_CONTEXT_CONFIG_CQE_SB_INDEX_NUMBER (0xF<<0)
#define USTORM_ETH_ST_CONTEXT_CONFIG_CQE_SB_INDEX_NUMBER_SHIFT 0
#define USTORM_ETH_ST_CONTEXT_CONFIG_BD_SB_INDEX_NUMBER (0xF<<4)
#define USTORM_ETH_ST_CONTEXT_CONFIG_BD_SB_INDEX_NUMBER_SHIFT 4
    uint8_t clientId;
    uint8_t status_block_id;
    uint8_t flags;
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_MC_ALIGNMENT (0x1<<0)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_MC_ALIGNMENT_SHIFT 0
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_DYNAMIC_HC (0x1<<1)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_DYNAMIC_HC_SHIFT 1
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_TPA (0x1<<2)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_TPA_SHIFT 2
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_SGE_RING (0x1<<3)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_SGE_RING_SHIFT 3
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_STATISTICS (0x1<<4)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_STATISTICS_SHIFT 4
#define __USTORM_ETH_ST_CONTEXT_CONFIG_RESERVED0 (0x7<<5)
#define __USTORM_ETH_ST_CONTEXT_CONFIG_RESERVED0_SHIFT 5
#endif
#if defined(__BIG_ENDIAN)
    uint16_t bd_buff_size;
    uint8_t statistics_counter_id;
    uint8_t mc_alignment_log_size;
#elif defined(__LITTLE_ENDIAN)
    uint8_t mc_alignment_log_size;
    uint8_t statistics_counter_id;
    uint16_t bd_buff_size;
#endif
#if defined(__BIG_ENDIAN)
    uint8_t __local_sge_prod;
    uint8_t __local_bd_prod;
    uint16_t sge_buff_size;
#elif defined(__LITTLE_ENDIAN)
```

```
    uint16_t sge_buff_size;
    uint8_t __local_bd_prod;
    uint8_t __local_sge_prod;
#endif
    uint32_t reserved;
    uint32_t bd_page_base_lo;
    uint32_t bd_page_base_hi;
    uint32_t sge_page_base_lo;
    uint32_t sge_page_base_hi;
};
```

The following fields in the eth_st_context_config structure must be modified when enabling TPA support in the NetXtreme II 10Gb network controller:

- flags - The USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_TPA bit in the flags field is used to enable/disable TPA operation in the device while the USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_SGE_RING bit is used to enable support for the scatter-gather queue. Both of these bits must be set when enabling TPA. These bits should not be set until AFTER the SGQ has been initialized and populated.
- sge_buff_size - The size in bytes of each scatter-gather entry buffer. Typically equivalent to the page size of the host CPU (for example, 4KB for Intel®/AMD® x86 processors).
- sge_page_base_lo - The lower 32 bits of the 64 bit address of the first page in the SGQ.
- sge_page_base_hi - The upper 32 bits of the 64 bit address of the first page in the SGQ.

The following USTORM receive producer data structure is modified to use TPA.

```
struct ustorm_eth_rx_producers {
#if defined(__BIG_ENDIAN)
    uint16_t bd_prod;
    uint16_t cqe_prod;
#elif defined(__LITTLE_ENDIAN)
    uint16_t cqe_prod;
    uint16_t bd_prod;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t reserved;
    uint16_t sge_prod;
#elif defined(__LITTLE_ENDIAN)
    uint16_t sge_prod;
    uint16_t reserved;
#endif
};
```

The following fields in the ustorm_eth_rx_producers structure are used by TPA:

- sge_prod - The sge_prod field is used by the host network driver when adding new entries to the SGQ. It is used in the same manner as the bd_prod or cqe_prod fields are during non-TPA operation and should be written at the same time as the bd_prod or cqe_prod values.

Note: Unlike the Receive Queue (RXQ) or the Completion Queue (CQ), entries of the SGQ may be posted to the host network driver out-of-order. For example, when multiple aggregations are occurring simultaneously, the host network driver may receive a CQE with scatter-gather elements 2, 4, and 6 in the sgl array field. Other scatter-gather elements 0, 1, 3, and 5 may be involved with other aggregations that have not completed. However, just like the RXQ and the CQ, when new elements are posted to the SGQ they must be posted in order (or contiguously) since only a single producer index is used. It is therefore the host network driver's responsibility to monitor the order of completions in the SGQ and only post contiguous scatter-gather entries to the SGQ when updating the sge_prod value.

Host Data Structures

The following host data structures are either added or modified to support TPA.

Scatter Gather Queue

The scatter gather queue (SGQ) is similar to the receive queue (RXQ) in that it is a chunked-list of pointers to host memory. Each chunk of the SGQ is a page of host memory (typically 4KB in size for Intel/AMD x86 systems) which is directly accessed by the network controller (i.e. it must be mapped for DMA by the controller). Each entry in the SGQ, or SGE, is a 64 bit pointer to a host memory buffer except for the last entry on a page which is the Next Page pointer to the next SGQ page. Since the SGQ is a circular linked list the last SGQ page points back to the first SGQ page.

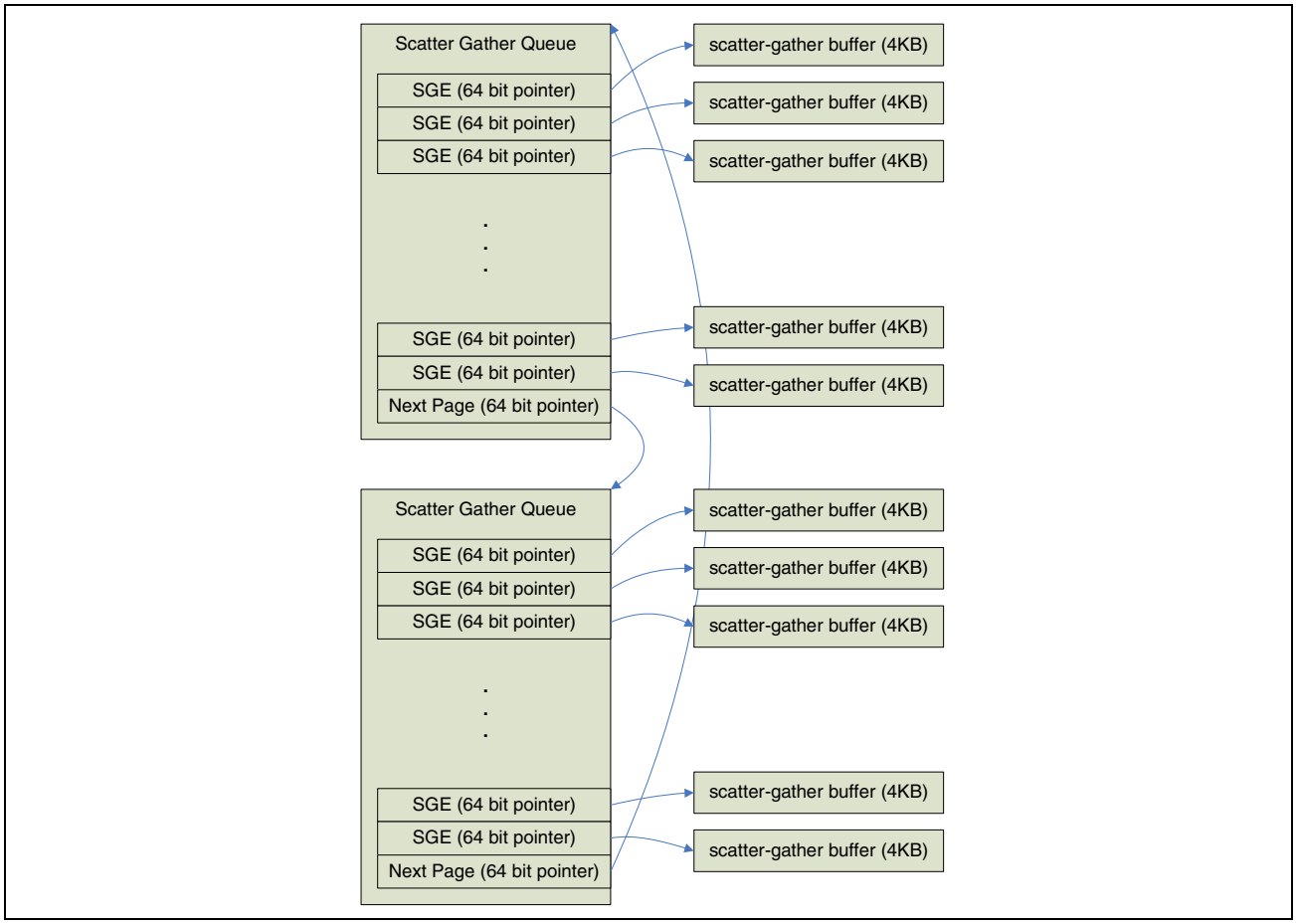

Figure 21: Scatter-Gather Queue Structure

Whereas the Receive Queue (RXQ) generally uses MTU sized buffers (2KB for FreeBSD) the SGQ generally uses buffers that are equivalent to the host CPU page size (4KB for Intel/AMD x86 systems). The size of the buffers used in the SGQ is configurable by the driver but the driver writer should be aware that a maximum of 8 SGE's can be used for an aggregated frame (see "Completion Queue Entry" below.)

Scatter Gather Entry

The scatter-gather entry (SGE) has the following definition:

```
struct eth_rx_sge {
    uint32_t addr_lo;
    uint32_t addr_hi;
};
```

Completion Queue Entry

When a TPA aggregation is started or completed the NetXtreme II 10Gb network controller will provide an indication to the host network driver through a Completion Queue Entry (CQE) in the Completion Queue (CQ). The CQE has the following structure:

```
union eth_rx_cqe {
    struct eth_fast_path_rx_cqe fast_path_cqe;
    struct common_ramrod_eth_rx_cqe ramrod_cqe;
    struct eth_rx_cqe_next_page next_page_cqe;
};
```

Since a receive completion is a fastpath event, the CQE has the following structure:

```
struct eth_fast_path_rx_cqe {
    uint8_t type_error_flags;
#define ETH_FAST_PATH_RX_CQE_TYPE (0x1<<0)
#define ETH_FAST_PATH_RX_CQE_TYPE_SHIFT 0
#define ETH_FAST_PATH_RX_CQE_PHY_DECODE_ERR_FLG (0x1<<1)
#define ETH_FAST_PATH_RX_CQE_PHY_DECODE_ERR_FLG_SHIFT 1
#define ETH_FAST_PATH_RX_CQE_IP_BAD_XSUM_FLG (0x1<<2)
#define ETH_FAST_PATH_RX_CQE_IP_BAD_XSUM_FLG_SHIFT 2
#define ETH_FAST_PATH_RX_CQE_L4_BAD_XSUM_FLG (0x1<<3)
#define ETH_FAST_PATH_RX_CQE_L4_BAD_XSUM_FLG_SHIFT 3
#define ETH_FAST_PATH_RX_CQE_START_FLG (0x1<<4)
#define ETH_FAST_PATH_RX_CQE_START_FLG_SHIFT 4
#define ETH_FAST_PATH_RX_CQE_END_FLG (0x1<<5)
#define ETH_FAST_PATH_RX_CQE_END_FLG_SHIFT 5
#define ETH_FAST_PATH_RX_CQE_RESERVED0 (0x3<<6)
#define ETH_FAST_PATH_RX_CQE_RESERVED0_SHIFT 6
    uint8_t status_flags;
#define ETH_FAST_PATH_RX_CQE_RSS_HASH_TYPE (0x7<<0)
#define ETH_FAST_PATH_RX_CQE_RSS_HASH_TYPE_SHIFT 0
#define ETH_FAST_PATH_RX_CQE_RSS_HASH_FLG (0x1<<3)
#define ETH_FAST_PATH_RX_CQE_RSS_HASH_FLG_SHIFT 3
#define ETH_FAST_PATH_RX_CQE_BROADCAST_FLG (0x1<<4)
#define ETH_FAST_PATH_RX_CQE_BROADCAST_FLG_SHIFT 4
#define ETH_FAST_PATH_RX_CQE_MAC_MATCH_FLG (0x1<<5)
#define ETH_FAST_PATH_RX_CQE_MAC_MATCH_FLG_SHIFT 5
#define ETH_FAST_PATH_RX_CQE_IP_XSUM_NO_VALIDATION_FLG (0x1<<6)
#define ETH_FAST_PATH_RX_CQE_IP_XSUM_NO_VALIDATION_FLG_SHIFT 6
#define ETH_FAST_PATH_RX_CQE_L4_XSUM_NO_VALIDATION_FLG (0x1<<7)
#define ETH_FAST_PATH_RX_CQE_L4_XSUM_NO_VALIDATION_FLG_SHIFT 7
```

```
    uint8_t placement_offset;
    uint8_t queue_index;
    uint32_t rss_hash_result;
    uint16_t vlan_tag;
    uint16_t pkt_len;
    uint16_t len_on_bd;
    struct parsing_flags pars_flags;
    uint16_t sgl[8];
};
```

When the CQE indicates an aggregation start or aggregation end, the eth_fast_path_rx_cqe structure fields have the following meaning:

- type_error_flags
  - The ETH_FAST_PATH_RX_CQE_START_FLG bit indicates that a new aggregation has begun and that the driver should place the newly received frame in the appropriate TPA queue.
  - The ETH_FAST_PATH_RX_CQE_END_FLG bit indicates that an aggregation has completed and that the driver should remove the scatter-gather entries for the frame from the SGQ.
- queue_index - This field indicate which TPA queue is referenced by the current TPA start or end CQE.
- pkt_len - When the ETH_FAST_PATH_RX_CQE_END_FLG bit is set this field indicates the number of bytes for the entire aggregated frame.
- len_on_bd - When the ETH_FAST_PATH_RX_CQE_END_FLG bit is set this field indicates the number of bytes used for the first frame received in the aggregation.
- sgl - The sgl array contains a list of pointers into the SGQ for the remaining data of the aggregated frame. The number of valid sgl pointers used by the aggregated frame can be determined by subtracting the len_on_bd field from the pkt_len_field and calculating the number of pages used to hold the result.

High Level Outline

The following is a high level outline of the steps required to implement TPA.

Initialization

The host network driver should perform the following steps to initialize the network controller for TPA:

- Allocate a number of pages for the scatter gather queue. Map these pages for DMA access. (The supplied reference drivers require that the number of pages in the SGQ be a "power of two", i.e. 1, 2, 4, or 8 pages.)
- Initialize the "Next Page" entries for each SGQ page, creating a circular linked list.
- Allocate the necessary number of scatter-gather buffers (e.g. 255 scatter-gather buffers are used for a 4KB scatter-gather chain page) to fill the SGQ. Map these buffers for DMA access. Update the scatter-gather entries with the physical address of the allocated scatter-gather buffers.
- Update the USTORM context to enable SGE/TPA support, set the size of a scatter-gather buffer, and provide the physical address of the first page of the SGQ.

Fastpath Operation

The host network driver should perform the following steps during normal network operation (i.e. during the interrupt service routine or the deferred procedure call):

- If the CQE has the ETH_FAST_PATH_RX_CQE_START_FLG set, the driver should:
  - Store the frame into an array of frames that are being aggregated (the TPA pool).
  - Set an internal flag to indicate that an aggregation for queue_index is in progress.

  - Not unmap the receive buffer from DMA space until the aggregation is complete as the network controller will need to update the TCP/IP header as the aggregation continues.

- If the CQE has the ETH_FAST_PATH_RX_CQE_END_FLG set, the driver should:
  - Determine the number of SGEs used for the aggregation.
  - Umap the SGE's and the frame in the TAP pool
  - Concatenate the SGE's to the original received frame in the TPA pool and remove those SGE's from the SGQ.
  - Clear an internal flag to indicate that an aggregation for queue_index has completed.
- Replace any SGE's that have been released and map the buffers for DMA.
- Update the sge_prod index if any new scatter-gather buffers can be added to the SGQ, provided that they are contiguous.
- Pass the resulting aggregated frame to the host network stack.

LARGE SEND OFFLOAD

This subsection includes rules that the BCM57710/BCM57711 host driver must follow for the device to support the Large Send Offload (LSO) feature. The BCM57710/BCM57711 handles LSO packets very similar to non-LSO packets except that the BCM57710/BCM57711 device segments large LSO blocks into smaller LSO maximum segment sized packets. Here are additional rules for transmitting LSO blocks that are not included in the normal L2 Transmit Flows section.

- The packet header data must be separated from the data payload. The packet headers include the Ethernet, IP and TCP headers. The packet headers must be placed into a separate buffer descriptor (BD).
- An LSO block maximum segment size (MSS) must not use up more than 13 buffer descriptors (BDs) for per MSS. A maximum number of 13 BDs must hold a byte length that is greater than or equal to the LSO MSS.
- Three BDs of the 13 BD limit include 1 or more packet header BDs, 1 parsing BD (See the L2 Tx flow section) and 1 **last** BD. This can be called the "13 BD window"
- If more than 13 BDs are needed to transmit an MSS packet, then a coalescing buffer BD or bounce buffer BD is necessary to meet the 13 BD limit.
- The host driver must implement a 13 BD sliding window algorithm to ensure the 13 BD limit is enforced as it processes the LSO block.

The host driver must set the appropriate LSO related fields in the Tx BD for the device to recognize that the packet is an LSO packet. See L2 Tx Flows section.

An LSO packet is handled in the following way:

1. Split the packet header data from the packet payload data and put it into a separate BD. The header BD is also the "start" BD (See the L2 Transmit Flows section). If the packet header is already separate, then this BD is labeled the **start** BD.
2. If an entire LSO packet meets the 13 BD window restriction then continue with the steps described in "L2 Transmit Flow" on page 85. If the LSO block contains more than 13 BDs, continue with the following steps
3. Determine how many BDs are required for the first packet byte count to be >= MSS. The Length in bytes used is calculated by counting the total data payload bytes and not the header data of 13 BDs minus the 1 parsing BD + (x amount of header BDs) + 1 last BD must be >= MSS. 13 BDs - (1 parsing BD + 1(or more) header BDs + 1 last BD) is a "13 BD window"
4. Determine the size in bytes of the first 13 BD window
5. Beginning from the first data payload BD, slide the "13 BD window" over 1 BD and determine if [(the previous window

size in bytes - the size of the 1st payload BD) + (size of the newly windowed payload BD)] >= MSS

6. If any 13 BD window (except for the final window) is < MSS a double copy or bounce buffer is needed to meet the 13 BD requirement of the 13 BD window >= MSS
7. Continue until the end of the LSO block. The last window does not have to be >= the MSS

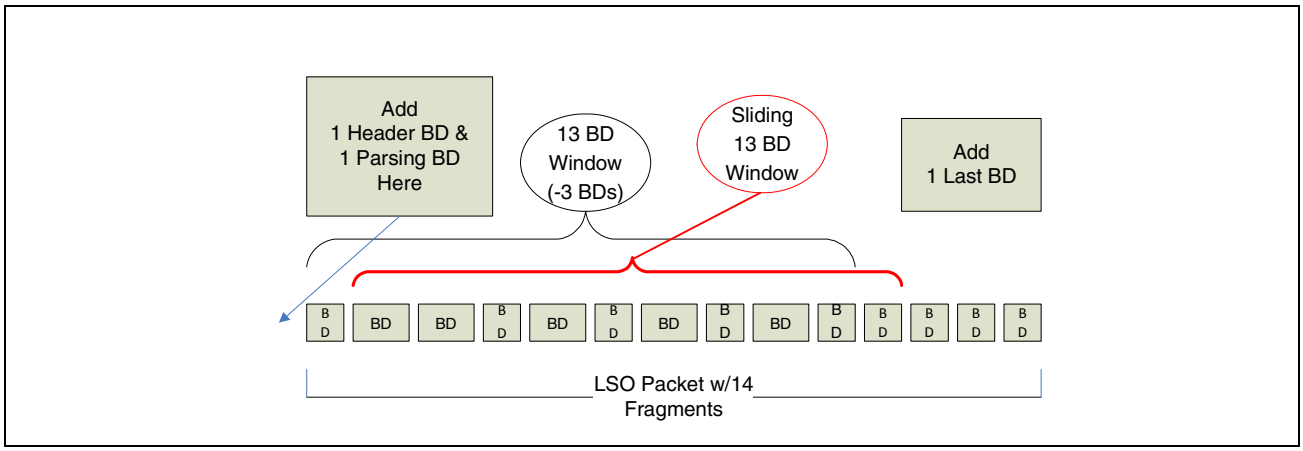

Figure 22: 13 BD Sliding Window

Figure 22 represents a 13 BD sliding window on an LSO block containing more than 13 fragments where the header data was split from the payload data and placed into a separate header BD. The blocks represent buffer descriptors of various sizes. The header data must be split from the payload data and the total byte count within the 13 BD window must not include the header data. By subtracting the 3 new driver-inserted BDs (the header BD, the parsing BD and a last BD) the 13 BD window becomes a window that covers 10 payload BDs. After calculating the first window size, the window slides over 1 BD and a new count is started.

DEVICE STATISTICS

BCM57710/BCM57711 layer 2 device statisitics are gathered by the host driver from the device block statistic registers and from each STORM statistic memory.

The BMAC, EMAC and NIG device blocks contain device statistics that the driver can gather directly from register reads or by using the BCM57710/BCM57711 device DMA engine. The STORM statistics are gathered by first using a RAMROD command to request an update of all STORM internal statistics fields in the STORM memories. The STORM statistics memories can then be read directly by the host driver.

The STORM statistics correspond to individual function statistics when multiple functions are enabled.

The BMAC, EMAC and NIG device block statistics pertain to individual port statistics.

The driver has the responsibility to ensure a 64–bit counter expansion from 32–bit counters.

Statistics gathered by the host driver from the BMAC, EMAC and NIG device blocks must be written to the MCP shared memory by the host driver for use by the device management firmware (MCP). See the FreeBSD BCM577xx driver for implementation details.

The method of gathering the device statistics can be seen in the open source host driver code.

DIRECT MEMORY ACCESS ENGINE (DMAE)

The BCM57710/BCM57711 includes an internal DMAE that allows DMA functionality to and from host memory, BAR mapped memory, and device GRC memory.

THE "GO" REGISTER

Each DMAE command has a "Go" register, which is a distinct GRC address that is allocated for this purpose. When an execution of a command finishes, the PCI/GRC source/destination addresses are retained intact, and the next execution continues from the addresses where the previous execution finished. The CRC continues as if the new command is a continuation of the previous command, except if the opcode indicates to reset the CRC result on either of the source/destination addresses.

THE OPCODE

The opcode format is shown in Table 35.

Table 35: Opcode Format

Field	*Bits*	*Description*	*Significant Width*
SRC	0:0	Whether the source is the PCIe or the GRC. 0- The source is the PCIe 1- The source is the GRC.	1
DST	2:1	The destination of the DMA can be: 0- None 1- PCIe 2- GRC 3- None	2
C-DST	3:3	The destination of the completion: 0- PCIe 1- GRC	1
Completion type	4:4	Whether to write a completion word to the completion destination: 0- Do not write a completion word 1- Write the completion word	1
	5:5	Whether to write a CRC word to the completion destination 0- Do not write a CRC word 1- Write a CRC word	1
	8:6	The CRC word should be taken from the DMAE GRC space from address 9+X, where X is the value in these bits.	3
Endianity mode	10:9	The endianity swapping used when doing the DMA. (see the PCIe HLD spec for a description of endianity modes).	2
Network Port ID	11:11	Which network port ID to present to the PCI request interface	1

Table 35: Opcode Format (Cont.)

Field	Bits	Description	Significant Width
CR - CRC reset	12:12	Reset the CRC result (do not use the previous result as the seed).	1
SR – SRC address reset	13:13	Reset the source address in the next "go" to the same source address of the previous "go". If this bit is 0, then the next "go" continues from the source address where the previous "go" stopped	1
DR – DEST address reset	14:14	Reset the destination address in the next "go" to the same destination address of the previous "go". If this bit is 0, then the next "go" continues from the destination address where the previous "go" stopped.	1

In the completion type field, if both bits 0 and 1 are asserted, first the word is written to the completion address, and then the CRC result is written to the next address.

Endianity mode: Endianity swapping is used when reading/writing from/to PCI.

When consecutive DMAE commands are necessary and when the CRC should be calculated over the entire payload of all the commands, then generally the CR bit should be 0, except for the first command where the CR bit should 1.

ARCHITECTURE

The NetXtreme II controller provides controls for both byte-swapping and word-swapping. The Broadcom NetXtreme II controller operates internally using a 64-bit big-endian architecture, and its internal RISC processors are 64-bit big-endian devices. This differs from many host systems that connect to the NetXtreme II through the little-endian PCI/PCI-X/PCIe bus, and operate with little-endian processors. To accommodate the difference in data representation between the internal and external interfaces, the Broadcom NetXtreme II controller provides several different byte and word swapping options so that both big-endian and little-endian hosts can interface seamlessly over the PCIe interface.

The BCM57710/BCM57711 Ethernet Controller supports the Byte and Word swapping configuration per each of the clients to the PXP (PCI Express logic) block. For SDM (USDM, TSDM, CSDM, XSDM, USDM_DP) and DMAE clients, the Byte and Word swapping is supported per request (2 MSB bits of the Echo field in the request). For all other PXP clients, the swapping mode is determined based on the configuration in per client Endian Mode configuration registers in the PXP and PXP2 blocks.

The following are the registers used for configuring the swap mode for Host accesses to various blocks of the device in BAR space.

- PXP_REGISTERS_HST_USDM_SWAP_MODE (Offset: 0x10300C): Swapping Mode for the Host accesses to USDM block.
- PXP_REGISTERS_HST_CSDM_SWAP_MODE (Offset: 0x10300C): Swapping Mode for the Host accesses to XSDM block.
- PXP_REGISTERS_HST_XSDM_SWAP_MODE (Offset: 0x103014; Swapping Mode for the Host accesses to XSDM block.
- PXP_REGISTERS_HST_TSDM_SWAP_MODE (Offset: 0x103018): Swapping Mode for the Host accesses to TSDM block.
- PXP_REGISTERS_HST_GRC_SWAP_MODE (Offset: 0x103020): Swapping Mode for the Host accesses to GRC block.

- PXP_REGISTERS_HST_DQ_SWAP_MODE (Offset: 0x103024): Swapping Mode for the Host accesses to Door Bell BAR space.
- PXP_REGISTERS_HST_HC_SWAP_MODE (Offset: 0x10301C): Swapping Mode for the Host accesses to BAR memory used by HC block. One example of such BAR memory used by HC block is the MSI-X table space.

The following are the registers for configuring the swap mode for DMA read requests from the different clients to the PCIe block.

- PXP2_REGISTERS_RD_PBF_SWAP_MODE (Offset: 0x1203F4): Swapping Mode for the PBF bus master read requests.
- PXP2_REGISTERS_RD_QM_SWAP_MODE (Offset: 0x1203F8): Swapping Mode for the QM bus master read requests.
- PXP2_REGISTERS_RD_TM_SWAP_MODE (Offset: 0x1203FC): Swapping Mode for the TM bus master read requests.
- PXP2_REGISTERS_RD_SRC_SWAP_MODE (Offset: 0x120400): Swapping Mode for the SRC bus master read requests.
- PXP2_REGISTERS_RD_CDURD_SWAP_MODE (Offset: 0x120404): Swapping Mode for the CDU bus master read requests.

The following are the registers used for configuring the swap mode for DMA write requests from the different clients to the PCIe block.

- PXP2_REGISTERS_RQ_QM_ENDIAN_M (OFFSET: 0X120194): Swapping Mode for the Queue Manager block DMA write requests.
- PXP2_REGISTERS_RQ_TM_ENDIAN_M (OFFSET: 0X120198): Swapping Mode for the Timers block DMA write requests.
- PXP2_REGISTERS_RQ_SRC_ENDIAN_M (OFFSET: 0X12019C): Swapping Mode for the Searcher block DMA write requests.
- PXP2_REGISTERS_RQ_CDU_ENDIAN_M (OFFSET: 0X1201A0): Swapping Mode for the Context Distribution Unit block DMA write requests.
- PXP2_REGISTERS_RQ_DBG_ENDIAN_M (OFFSET: 0X1201A4): Swapping Mode for the Debug block DMA write requests.
- PXP2_REGISTERS_RQ_HC_ENDIAN_M (OFFSET: 0X1201A8): Swapping Mode for the Host Coalescence block DMA write requests.
- PXP2_REGISTERS_RQ_PBF_ENDIAN_M (OFFSET: 0X1201AC): Swapping Mode for the Packet Builder and Framer block DMA write requests.

All the swap mode configuration registers described above support four different swapping modes depending upon the configuration the 2 LSB bits.

- Mode-00: No Byte swapping and No Word (DW) swapping
- Mode-01: Byte swapping and No Word (DW) swapping
- Mode-10: No Byte swapping and Word (DW) swapping
- Mode-11: Byte swapping and Word (DW) swapping

Section 6: PCIe

INTRODUCTION

PCI Express (PCIe) is a third-generation high-performance I/O interconnect, and is used in a wide variety of computing and communication platforms. The BCM57710/BCM57711 supports the x8 PCIe interface, which is fully compliant with the PCI Express Base Specification, Revisions 2.0 and 1.1. The x8 PCIe link has eight differential lanes in each direction, with each lane operating at 2.5 Gbps in Gen 1 and 5.0 Gbps in Gen 2, for an aggregate raw bandwidth of 20 Gbps and 40 Gbps in Gen 1 and Gen 2 respectively, in each direction. The PCIe bus is a serial interconnect and hence the x8 PCIe link requires only 40 pins, with 32 of these used for 16 differential lanes (eight lanes in each direction), with the balance of the pins used for power and grounding. The data on each lane is 8B/10B encoded to allow for clock recovery at the receiver and AC coupling. CRC checking is performed on each lane to ensure data integrity. The bus protocol and I/O electrical design allow for hot-insertion or removal of PCIe devices when a standard connector is used. The PCIe also supports other advanced features like Power Management, Quality of Service, and error handling. See the PCI Express Base Specification, Revisions 2.0 and 1.1, March 28, 2005 for more details.

Broadcom Corporation

5300 Calilfornia Avenue
Irvine, CA 92617
Phone: 949-926-5000
Fax: 949-926-5203

SUPPORTED FEATURES

Table 36: PCIe Features

Feature	*Support and Comments*
Number of Lanes	1,2 ,4 or 8
Option to turn off packing	Supported. Useful when link partner has problems with packing.
Access configuration space even when PCIe reference clock is unavailable	Supported
Aligns the read requests originating from the PCIe to 4 Byte boundaries	Supported
Master single write transactions (without splitting) up to the Max Payload Size	Supported
Target bursts	Supported. User controls maximum length through credits and ability to return error.
I/O transactions	Not supported
Max payload size	512 bytes
Max Read Request size	4 KB
64-bit addressing	Supported
Byte-enable checking	Supported
Number of outstanding requests	Up to 32. Hard-configurable.
Relaxed ordering support	Supported. User controlled
No snoop support	Supported. User controlled
Check crossing of 4k boundary as a receiver	Supported
Transmitter supports INTx messages	Supported
Receiver checks for INTx messages and reports as errors	Supported
Interrupts performed with MSI/MSI-X or INTx messages	Supported BCM57710/BCM57711 A0 does not support MSI.
Transmitter supports PM_PME, PME_TO_Ack TLPs and PM_Enter_L1 DLLP	Supported
Receiver supports PM_Active_State_Nak and PME_Turn_Off	Supported
Receiver discards error messages	Supported
Violations of programming model return completer abort	Supported, user controlled
Support configuration request retry	Supported
May return partial completions	User controlled
Transactions crossing RCB boundaries may use multiple completions	User controlled
Receivers may check for RCB violations	Supported
VC support	Only VC0 supported
TC support	Supported
Virtual channel capability configuration space supported	Supported
Initial posted header credits	Hard configurable, default to 32, can be set to infinite for debug
Initial posted data credits	Hard configurable, default to 512B worth, can be set to infinite for debug

Table 36: PCIe Features (Cont.)

Feature	*Support and Comments*
Initial non-posted header credits	Hard configurable, default to 1, can bet set to infinite for debug
Initial non-posted data credits	Soft configurable, default to infinite
Initial completion header credits	Soft configurable, default to infinite
Initial completion data credits	Soft configurable, default to infinite
Check for violations of initial credits	Supported
Check for too many credits	Supported
Check for finite credits after infinite	Supported
Receiver overflow detect	Supported
Timer for not receiving DLLP	Supported
Recommended FC update timer	Self configurable
Supports ECRC generation/checking	Supported
Transmitter data poisoning support	User controlled
Completion timeout	Supported
Transmitter may choose to have ACKs/NAKs not affect replay	ACK/NAKs do not affect replay.
ACKs/NAKs may be collapsed	Collapsing supported
Recommended DLLP priorities	Recommended priorities supported
Replay timer programmable	Self configurable
Retry buffer sizing	Hard-configurable. Must follow formula and include L0s exit latency
ACK_NAK latency timer programmable	Soft configurable
STP and SDP symbols may be placed on link in same symbol time	Supported
Handling of mutually exclusive control bits	Supported
Number of FTS ordered sets required	Soft-configurable between 0 and 255
Link errors may result in LTSSM transitioning to recovery	Supported
Ability to form x2 link	Supported
Support lane reversal	Supported
Beacon support	Supported
L1 support	Supported
L1 transition from ASPM	Supported
L2 support	Supported
WAKE# support	Supported
PM_PME backpressure deadlock avoidance	Supported
L0s entry time	Soft-configurable (<7 us)
L1 entry time	Soft-configurable
L0s and L1 exit latencies	Soft-configurable (reset to 2-4 us for L0s and 32-64 us for L1)
Endpoint L0s and L1 acceptable latencies	Soft-configurable (reset to 2-4 us for L0s and >64 us for L1)
Advanced error reporting	Supported
Device serial number capability	Supported
Power budgeting capability	Supported

CONFIGURATION SPACE

The PCIe devices support the configuration space of 4 KB as shown in Figure 23. The PCIe configuration space is divided into a PCI 2.3-compatible region, which is the first 256 bytes, and an extended PCIe configuration space region, which consists of the remaining configuration space.

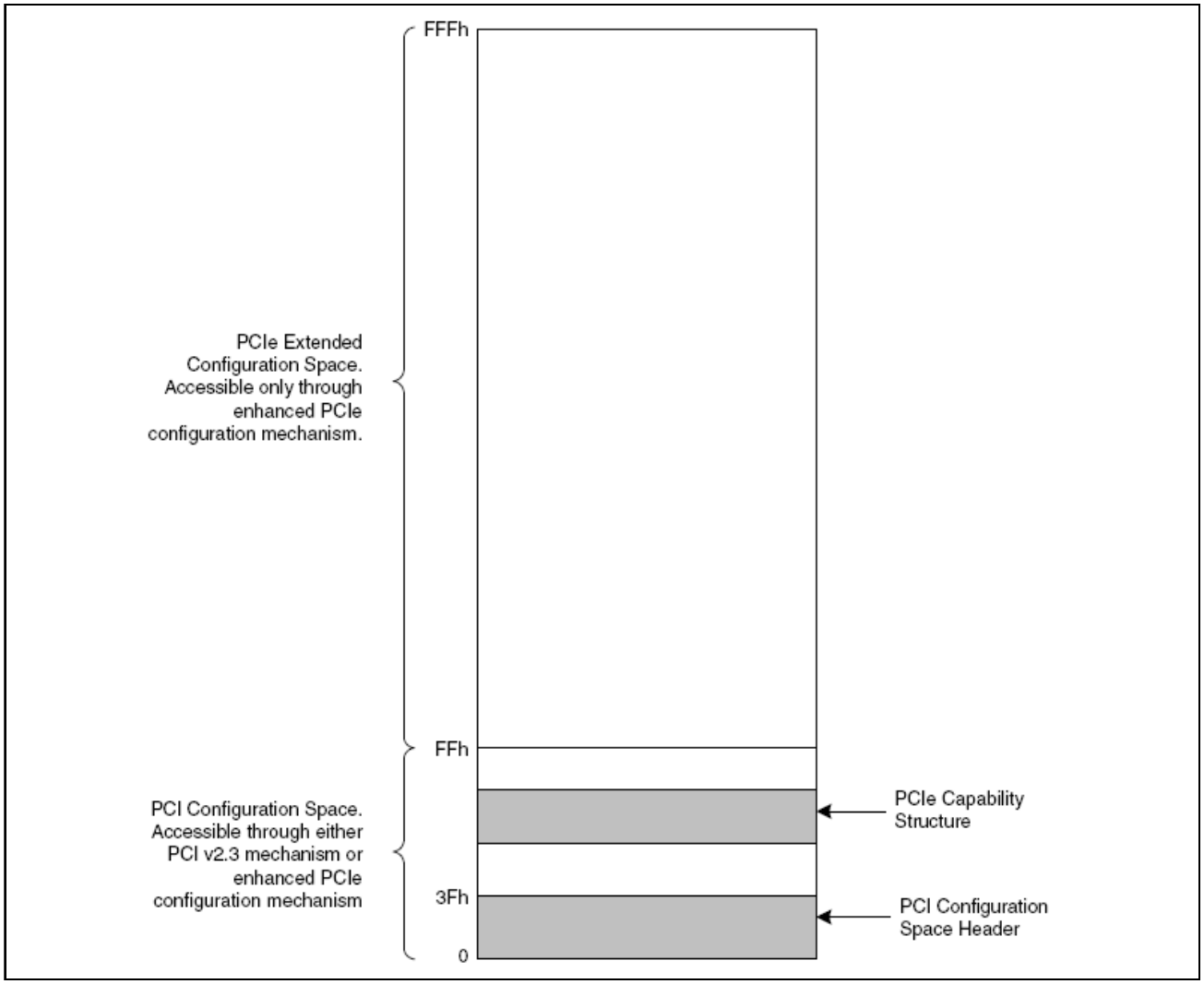

Figure 23: PCIe Configuration Space

There are three types of PCI configuration space registers that may be exposed by any particular PCI device:

- Required
- Capabilities
- Device specific

REQUIRED REGISTERS

Figure 24 shows the PCI configuration space header that occupies the configuration space 0x0 to 0x3F and includes all the Required PCIe registers. There are two formats for configuration space header: Type-0 and Type-1. The format for a given device depends upon the device type. The BCM57710/BCM57711, being a PCIe endpoint device, implements the Type-0 configuration space header. The shaded registers are common for both Type-0 and Type-1 configuration space headers.

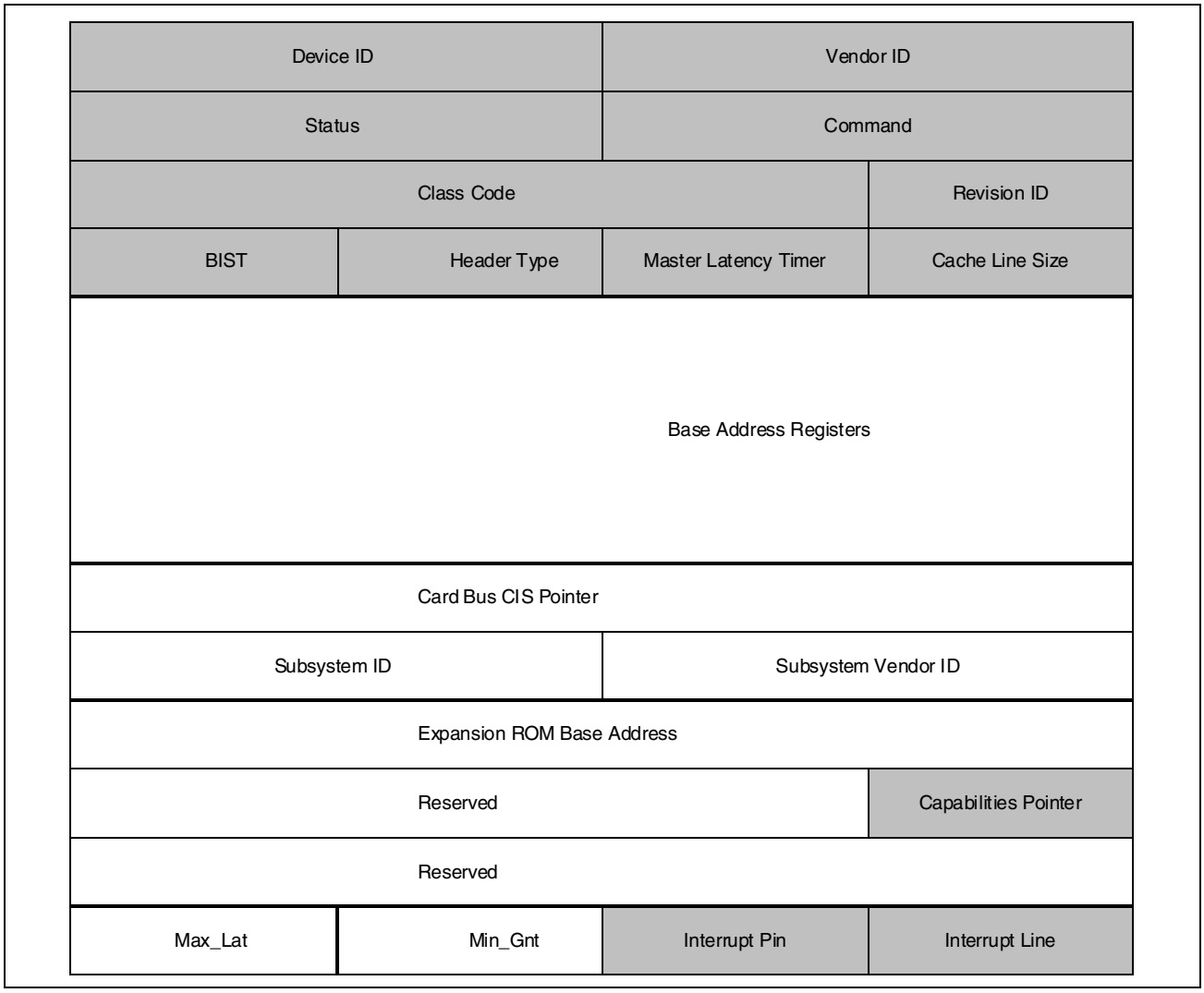

Figure 24: PCIe Type 0 Configuration Space Header

CAPABILITIES REGISTERS

The capabilities registers provide system BIOS and Operating Systems visibility into a set of optional features that devices may implement. The structure and mechanism for chaining auxiliary capabilities is defined in the PCI specification. Both software and BIOS must implement algorithms to fetch and program capabilities fields, accordingly.

The BCM57710/BCM57711 supports the Power Management capability, VPD capability, MSI capability, MSI-X capability, and PCIe capability in the capabilities chain, which is within the first 256 bytes of the configuration space. The BCM57710/BCM57711 also supports Device Serial Number capability, Advanced Error Reporting capability, Power Budgeting capability, and Virtual Channel capability in the PCIe extended configuration space.

DEVICE-SPECIFIC REGISTERS

Additional PCI configuration space may be used for device-specific registers. However, device-specific registers are not exposed to system software, according to a specification/standard. System software cannot probe device-specific registers without a predetermined understanding of the device and its functionality.

EXPANSION ROM

The Expansion ROM on the Broadcom NetXtreme II family is intended for implementation of Preboot Execution Environment (PXE) and iSCSI boot. The firmware detects the size of option ROM firmware and accordingly advertises the required option ROM size through the Expansion ROM Base Address register of the configuration space header.

OPERATIONAL CHARACTERISTICS

By default, the Expansion ROM is disabled and the firmware must explicitly enable this feature by setting the PCIE_REG_PCIER_CONFIG_2.EXP_ROM_SIZE field to the required expansion ROM size. The BIOS detects whether a PCI device supports an Expansion ROM by writing the value 0xFFFFFFFE to the Expansion ROM Base Address register. If the value is non-0 when the BIOS reads back from this register, this PCI device supports Expansion ROM; otherwise, it is not supported. If the EXP_ROM_SIZE field of the PCIE_REG_PCIER_CONFIG_2 register is set to a value of 0, then Broadcom NetXtreme II returns a value of 0x00000000 when the PCI Expansion ROM Base Address register is read, indicating that Expansion ROM is not supported. Otherwise, it returns a non-0 value that indicates the size of the expansion ROM supported by the NetXtreme II.

If a PCI device supports Expansion ROM, the BIOS will assign an Expansion Base address to the device. It then checks for a valid ROM header (0x55 0xAA as first 2 bytes, and so on) and checksum. If the ROM header and image are valid, the BIOS will copy the Expansion ROM image to Host's Upper Memory Block (UMB) and invoke the initializing entry point.

Accesses to the PCI Expansion ROM are handled by boot code firmware running on the NetXtreme II. When the host generates a PCI access to the Expansion ROM, the address is latched in the PCI Expansion ROM Address register and an EXP_ROM_ATTN is asserted. When the firmware services this attention, it places the expansion ROM data read from NVRAM into the PCI Expansion ROM Data register which places the data onto the PCI bus and completes the Expansion ROM access. In between the time that the EXP_ROM_ATTN is asserted and the final data is written to the PCI Expansion ROM Data register, the PCI block issues retries on the PCI bus to make the requester wait.

Note: The Maximum Expansion ROM size supported by the device is limited by the Maximum NVRAM size supported by the device.

Section 7: Ethernet Link Configuration

OVERVIEW

The BCM57710/BCM57711 Ethernet Controller integrates dual XAUI™/ 10GBASE-CX4/ 10GBASE-KX4 SerDes Transceivers and dual 1000 BASE-KX/ 2500 BASE-KX SerDes Transceivers for supporting 1/ 2.5/ 10 Gbps modes of operation and interfacing with external Phys supporting either Fiber or Copper physical media. This section describes the programming aspects of driver accessing the Phy registers.

MDIO INTERFACE

The MDIO interface is an IEEE standardized Management interface between MAC and PHY devices. It is a serial interface with two signals: MDIO clock (MDC) and bidirectional Serial Data (MDIO) which allows MDIO Address frames, MDIO Write Data frames or MDIO Read Data frames to communicate to the (MDIO Management Device) MMD. This interface is covered in details by Clause 22, Clause 45 and Annex 45A in IEEE 802.3-2005™ specification. The BCM57710/BCM57711 Ethernet Controller supports both Clause 22 and Clause 45 modes of operation of MDIO interface.

CLAUSE 22 OVERVIEW

The management frame formats for the Clause 22 operation are shown in Table 18. The Clause 22 allows up to 32 Phy devices and up to 32 registers per Phy device.

Table 37: Management Frame Format (See IEEE 802.3-2005 Specification)

				Management Frame Fields				
Frame	***PRE***	***ST***	***OP***	***PHYAD***	***REGAD***	***TA***	***DATA***	***IDLE***
Read	1...1	01	10	AAAAA	RRRRR	Z0	DDDDDDDD DDDDDDDD	Z
Write	1...1	01	01	AAAAA	RRRRR	10	DDDDDDDD DDDDDDDD	Z

CLAUSE 45 OVERVIEW

The Clause 45 provides the ability to access more PHY device registers while still retaining logical compatibility with the MDIO interface defined in Clause 22. This clause allows a single MDIO management station (STA), through a single MDIO interface, to access up to 32 Phys (defined as PRTAD in the frame format) consisting of up to 32 MMDs (MDIO Manageable Devices addressed by DEVAD in the frame format) as shown in Figure 25. The MDIO interface can support up to a maximum of 65536 registers in each MMD.

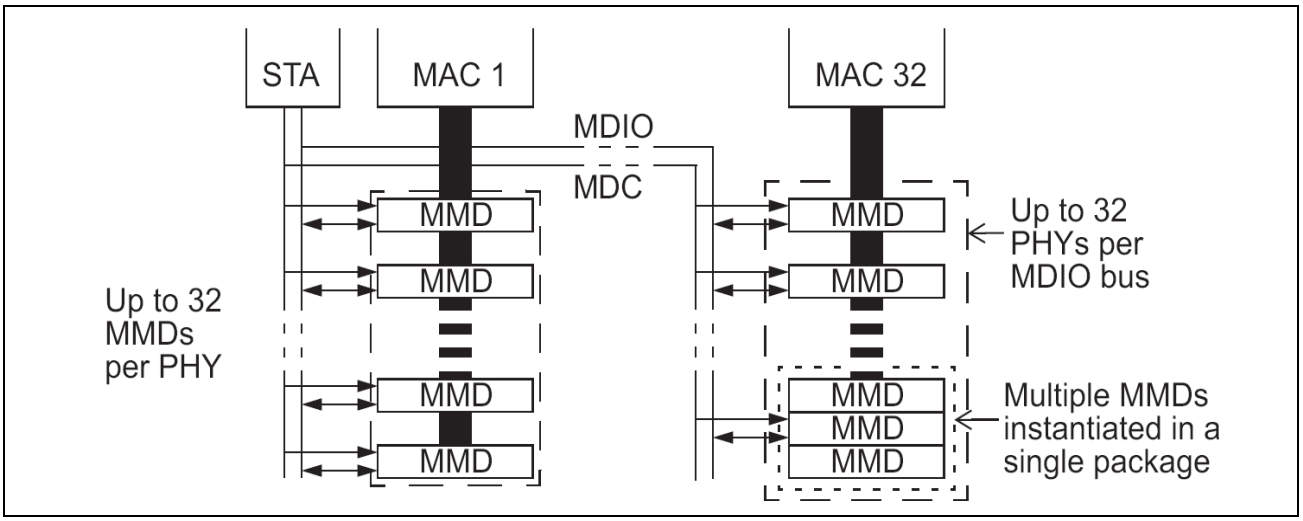

Figure 25: STA and MMD devices (ffrom the IEEE 802.3-2005 Specification)

In Clause 45 additional registers are added to the address space by defining MDIO frames that use an ST (Start) code of '00' as shown in Table 38. Clause 45 defines an Address frame for writing the PHY register address to the Address Register of the MMD. The write, read, and post-read-increment-address frames will access the register whose address is stored in the Address Register of the MMD. The Post-read-increment-address frame allows reading of multiple successive registers from an MMD without having to issue an Address command before every read operation. The MMD will automatically increment the address by one upon receiving a post-read-increment-address frame unless the address is already 65535.

Table 38: Clause 45 MDIO Management Frame Formats

Management Frame Fields								
Frame	***PRE***	***ST***	***OP***	***PRTAD***	***DEVAD***	***TA***	***ADDRESS/DATA***	***IDLE***
Address	1...1	00	00	PPPPP	EEEEE	I0	AAAAAAAAAAAAAAAA	Z
Write	1...1	00	01	PPPPP	EEEEE	I0	DDDDDDDDDDDDDDDD	Z
Read	1...1	00	11	PPPPP	EEEEE	Z0	DDDDDDDDDDDDDDDD	Z
Post-read increment address	1...1	00	10	PPPPP	EEEEE	Z0	DDDDDDDDDDDDDDDD	Z

ACCESSING PHY REGISTERS

The EMAC block of BCM57710/BCM57711 Ethernet Controller implements the device MDIO interface and the device MI communications registers for allowing the firmware and driver to access the Phy registers. The BCM57710/BCM57711 MDIO Management Station (STA) currently can be found in the BCM57710/BCM57711 EMAC block. The MMDs are found in the BCM57710/BCM57711 internal PHY (XGXS) and external PHY if present.

Currently the BCM57710/BCM57711 internal PHY has a default PHY address of 1 and the external PHY of a Broadcom designed NIC has an external PHY address of 0x0, 0x10 or 0x11.

There are three modes in which the MDIO interface signals (MDC/MDIO) can be controlled for communication with the PHY registers; the Auto-Polling Mode, the Bit-Bang mode and the Auto-Access mode. Currently the Broadcom BCM57710/BCM57711 software drivers use the Auto-Access Mode only. The other modes are listed for reference.

AUTO-POLLING MODE

This mode allows the EMAC to automatically poll for link status bit in the PHY periodically and update the MAC link status accordingly. This mode is used for MAC detecting the link status changes from the PHY, especially when the PHY device does not support the link status change interrupt signal. This mode can be enabled or disabled by using the 'Auto Poll' bit in the EMAC_REG_EMAC_MDIO_MODE register.

BIT-BANG MODE

This mode allows the driver or firmware to directly control the signals on MDIO bus through the EMAC_REG_EMAC_MDIO_MODE register. This mode can be enabled or disabled using the BIT_BANG bit of the MDIO_MODE register. When this mode is enabled, the values written to MDC, MDIO_OE, and MDIO bits of the MDIO_MODE register directly control the logic levels of MDC and MDIO signals of the MDIO interface. When using this mode, the software entity controlling the MDIO bus is responsible for providing the proper delay to satisfy timing requirements such as setup time, hold time, clock frequency, etc. The Auto Access mode described next takes this burden off of the software and hence is the recommended method for accessing PHY registers.

AUTO-ACCESS MODE

This mode allows the driver or firmware to access the PHY registers by issuing commands to the device through the EMAC_REG_EMAC_MDIO_COMM register. The BCM57710/BCM57711 Ethernet Controller has a built-in state machine in the EMAC block for controlling the MDIO bus according to the command issued from driver. For ease of reference, the MDIO_COMM register definition is shown in Table 39.

Table 39: EMAC_REG_EMAC_MDIO_MODE - (Offset: (GRCBASE_EMAC0 / GRCBASE_EMAC1) + 0xB4; Width: 32)

Bit	*Name*	*Description*	*Mode*	*Reset*
31	CLAUSE_45	When set to 1 this bit indicates that the current MDIO transaction will be executed as a Clause 45 transaction. When 0, the transaction is executed as a Clause 22 transaction. Value of this bit also determines the meaning of bits specified in bits [27:0] of the EMAC_REG_EMAC_MDIO_COMMregister. This bit must be set to proper value before the link auto-polling function is enabled.	RW	0
30-22	unused4		RO	0
21-16	CLOCK_CNT	This field controls the MDIO clock speed. The output MDIO clock runs at a frequency equal to CORE_CLK/(2*(CLOCK_CNT+1)). A value of 0 is invalid for this register.	RW	0x13
15-14	unused3		RO	0
13	EXT_MDINT	The read value of this bit reflects the current state of the External MDINT input pin. If the interrupt is asserted, this bit will be '0', otherwise, this bit will be '1'.	RO	X
12	MDINT	The read value of this bit reflects the current state of the MDINT input pin from the Copper PHY. If the interrupt is asserted, this bit will be '0', otherwise, this bit will be '1'.	RO	0x1
11	MDC	Setting this bit to '1' will cause the MDC pin to high if the **BIT_BANG** bit is set. Setting this pin low will cause the MDC pin to drive low if the **BIT_BANG** bit is set.	RW	0
10	MDIO_OE	Setting this bit to '1' will cause the MDIO pin to drive the value written to the **MDIO** bit if the **BIT_BANG** bit is set. Setting this bit to zero will make the MDIO pin an input.	RW	0
9	MDIO	The write value of this bit controls the drive state of the MDIO pin if the **BIT_BANG** bit is set. The read value of this bit always reflects the state of the MDIO pin.	RW	0
8	BIT_BANG	If this bit is '1', the MDIO interface is controlled by the **MDIO**, **MDIO_OE**, and **MDC** bits in this register. When this bit is '0', the commands in the EMAC_REG_EMAC_MDIO_COMM register will be executed.	RW	0
7-5	unused2		RO	0
4	AUTO_POLL	Enables auto-polling. When auto-polling is on, the START_BUSY bit in the EMAC_REG_EMAC_MDIO_COMM register must not be set. The interface automatically polls the PHY device and sets the LINK bit in the EMAC_REG_EMAC_STATUS register according to bit 2 of PHY register 1. The PHY address used is that programmed into the PHY_ADDR field of the EMAC_REG_EMAC_MDIO_COMM register.	RW	0
3-2	unused1		RO	0
1	SHORT_ PREAMBLE	If this bit is set, the 32-bit pre-amble will not be generated during autopolling.	RW	0
0	unused0		RO	0

The following is a list of steps for accessing BCM57710/BCM57711 external PHY registers in the Auto Access mode.

MDIO access is possible through the EMAC0 and EMAC1 blocks.

EMAC0 usually addresses XGXS0 and EMAC1 usually addresses EMAC1.

Steps for a Clause 45 Write

1. Set the MDIO mode to Clause45 by setting the appropriate bit in the EMAC0 or EMAC1 register block register MDIO_MODE register
2. Create an MDIO address management frame with the CMD field of 0
3. Send an MDIO address management frame using the EMAC(0/1) MDIO_COMM register and setting the START_BUSY bit to 1
4. Wait for the START_BUSY bit to return to 0
5. Create an MDIO data management frame with the CMD field set to 1
6. Send an MDIO write management frame using the EMAC(0/1) MDIO_COMM register and setting the START_BUSY bit to 1.

Figure 26 shows an example of address and write management frames for an MDIO write procedure with a PHY address of 0, port 0, an MMD address of 2 containing an offset 0x9005 with data 0x1234.

DEVAD = 2

Data = 0x1234

Address Offset = 0x9005

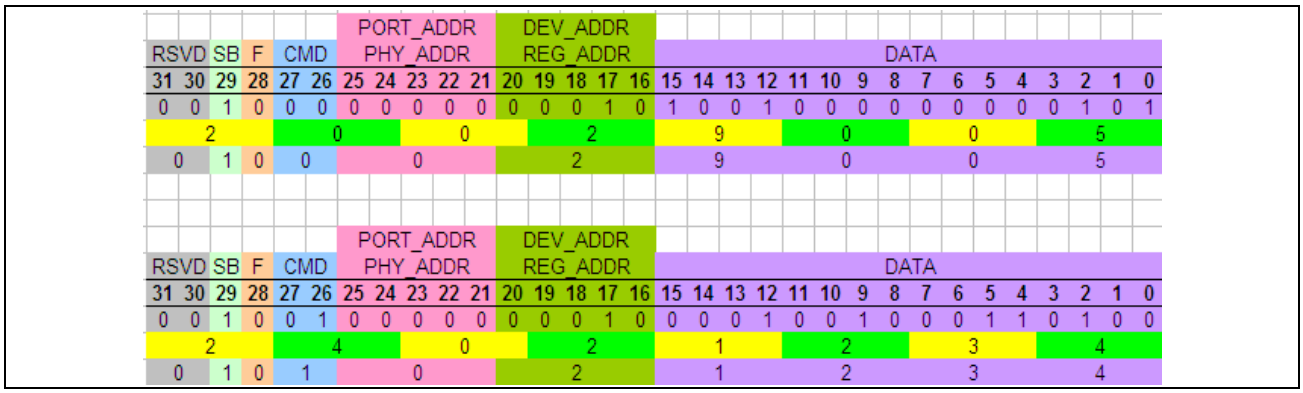

Figure 26: Address and Write Management Frames

The following shows steps for a Clause 45 PHY Read Procedure :

1. Set the MDIO mode to Clause45 by setting the appropriate bit in the EMAC0 or EMAC1 register block register MDIO_MODE register
2. Create an MDIO address management frame with the CMD field of 0
3. Send the MDIO address management frame using the EMAC(0/1) MDIO_COMM register and setting the START_BUSY bit to 1
4. Wait for the START_BUSY bit to return to a value of 0
5. Create an MDIO read management frame with the CMD field value of 3
6. Send the MDIO read management frame using the EMAC(01) MDIO_COMM registers by setting the START_BUSY bit

to 1

7. Wait for the START_BUSY bit to return to a value of 0
8. Read the DATA field from the MDIO_COMM register for the read data

Figure 27 on page 119 shows an example of address and read management frames for an MDIO write procedure with a PHY address of 0, port 0, an MMD address of 2 containing an offset 0x9005 with data 0x1234.

DEVAD = 2

Data = 0x1234

Address Offset = 0x9005

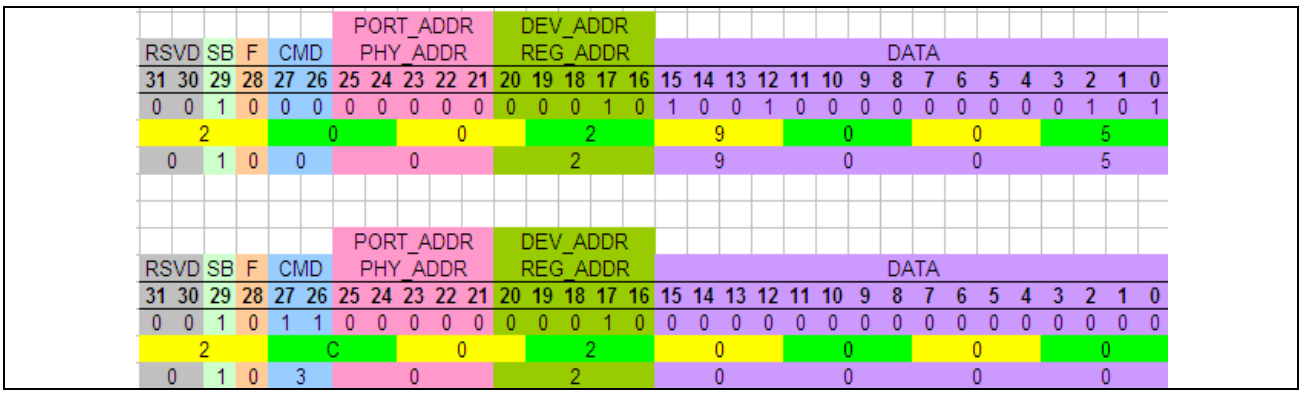

Figure 27: Address and Read Management Frames

Note: Writing to the MDIO_COMM register prior to the completion of a previous MDIO access will yield unpredictable MDIO data: when programming, wait for the START_BUSY bit to clear.

Note: The Auto-polling mode has the lowest priority and it will be stalled any time there is an active operation through the MDIO_COMM register using any other mode.

Note: When Bit-bang mode is enabled, the MDIO_COMM register cannot be read or written, thus avoid using Auto Access mode while Bit-bang mode is enabled.

INTERNAL PHY

The same steps taken to access the BCM57710/BCM57711 external PHY can be taken to access the BCM57710/BCM57711 internal PHY by using a different DEVAD value, and by ensuring that the NIG_REGISTERS_XGXS0_CTRL_MD_DEVAD (Offset: 0x1033C; Width: 32) register is programmed with the same value. In this example, the DEVAD is set to a value of 3 when making a Clause 45 call and the NIG register is programmed with the same value. The XGXS AER register ADDRESS EXTENSION REGISTER BLOCK (OFFSET: 0XFFD0) must be programmed to 0x3801 for the mapping found in this example. Table 40 lists the register blocks available in the BCM57710/BCM57711 XGXS Autonegotiation MMD, DEVAD = 3.

Table 40: Internal PHY Clause 45 Register Blocks

Block Address	*Name*
0000h	IEEE0 AN CTRL Block
0010h	IEEE1 AN ADV Block
0020h	IEEE Reserved
8000h	XGXS Block 0
8010h	XGXS Block 1
8020h	10G TX BERT Block
8030h	10G RX BERT Block
8040h	BcstBERT Block
8050h	PLL Block
8060h	TX0 Bock
8070h	TX1 Block
8080h	TX2 Block
8090h	TX3 Block
80A0h	TXAll Block
80B0h	RX0 Block
80C0h	RX1 Block
80D0h	RX2 Block
80E0h	RX3 Block
80F0h	RXAll Block
8100h	XGXS Block 2
8110h	In Band MDIO Block
8120h	General Purpose Status Block
8130h	10G Parallel Detect Block
8300h	SerDes Digital Block
8310h	Test Block
8320h	Over 1G Block
8330h	Remote PHY Block
8350h	MRBE Block
8370h	Clause 73 User B0 Block
FFD0h	Address Extension Register Block
FFE0h	Combo IEEE 0 Block

Table 40: Internal PHY Clause 45 Register Blocks

Block Address	Name
FFF0h	Combo User B0 Block
All Others	Reserved

Register Notations

In the register description tables, the following notation in the R/W column is used to describe the ability to read or write:

- R/W = Read or write
- RO = Read only
- LH = Latched high (until read)
- LL = Latched low (until read)
- H = Fixed high
- L = Fixed low
- SC = Self-clear after read
- CR = Clear on read Reserved bits must be written as the default value and ignored when read.

IEEE0 Clause 73 Autonegotiation Control Block (Offset: 0x0)

Table 41: IEEE0 Clause 73 Autonegotiation Control Register (Offset: 0x0; Width: 16) AKA (CL73_IEEEB0)

Bit	Name	R/W	Description	Default
15	Reset	R/W		0
14-13	Reserved	R/W	Write as 0, ignore on read.	0
12	Autonegotiation Enable	R/W		0
11-10	Reserved	R/W	Write as 0, ignore on read.	0
9	Restart Autonegotiation	R/W		0
8-0	Reserved	R/W	Write as 0, ignore on read.	0

IEEE0 Clause 73 Autonegotiation Status Register (Offset: 0x1 Width: 16)

Table 42: IEEE0 Clause 73 Autonegotiation Status Register (Offset: 0x1 Width: 16)

Bit	Name	R/W	Description	Default
15-10	Reserved	RO	Ignore on read.	0
9	Parallel Detect Fault	RO	Parallel detection fault.	0
8	Reserved	RO	Ignore on read.	0
7	Extended NP Status	RO	Extended next page will be used.	1
6	Page Received	RO	Page received.	0
5	Autonegotiation Complete	RO	Autonegotiation complete.	0
4	Remote Fault	RO	Remote fault.	0
3	Autonegotiation Ability	RO	Autonegotiation ability.	1
2	Link Status	RO	Link status.	0
1	Reserved	RO	Ignore on read.	0

Table 42: IEEE0 Clause 73 Autonegotiation Status Register (Offset: 0x1 Width: 16)

Bit	Name	R/W	Description	Default
0	Link Partner Autonegotiation Capable	RO	LP autonegotiation capable.	0

IEEE0 Clause 73 Autonegotiation PHY ID MSB Register (Offset: 0x2 Width: 16)

Table 43: IEEE0 Clause 73 Autonegotiation PHY ID MSB Register (Offset: 0x2 Width: 16)

Bit	Name	R/W	Description	Default
15-0	OUI	RO	Bits 18:3 of the organizationally unique identifier.	0143h

IEEE0 Clause 73 Autonegotiation PHY ID LSB Register (Offset: 0x3 Width: 16)

Table 44: IEEE0 Clause 73 Autonegotiation PHY ID LSB Register (Offset: 0x3 Width: 16)

Bit	Name	R/W	Description	Default
15-0	OUI	RO	Bits 24:19 of the organizationally unique identifier.	27h
9-4	Model	RO	Device model number	3Fh
3-0	Revision	RO	Device model number	0h

IEEE0 CL73 Autonegotiation Devices in Package 1 Register (Offset: 0x5; Width: 16)

Table 45: IEEE0 CL73 Autonegotiation Devices in Package 1 Register (Offset: 0x5; Width: 16)

Bit	Name	R/W	Description	Default
15-8	Reserved	RO	Ignore on read.	0
7	AN MMD Present	RO	1 = Autonegotiation MMD present. 0 = Autonegotiation MMD not present.	1
6	TC MMD Present	RO	1 = TC MMD present. 0 = TC MMD not present.	0
5	DTE XS MMD Present	RO	1 = DTE XS MMD present. 0 = DTE XS MMD not present.	0
4	PHY XS MMD Present	RO	1 = PHY XS MMD present. 0 = PHY XS MMD not present.	0
3	PCS MMD Present	RO	1 = PCS MMD present. 0 = PCS MMD not present.	0
2	WIS MMD Present	RO	1 = WIS MMD present. 0 = WIS MMD not present.	0
1	PMD/PMA MMD Present	RO	1 = PMA/PMD MMD present. 0 = PMA/PMD MMD not present.	1
0	Clause 22 Present	RO	1 = Clause 22 present. 0 = Clause 22 not present.	1

IEEE0 CL 73 Autonegotiation Devices in Package 2 (Offset: 0x6; Width: 16)

Table 46: IEEE0 CL 73 Autonegotiation Devices in Package 2 (Offset: 0x6; Width: 16)

Bit	Name	R/W	Description	Default
15	Vendor Specific MMD 2 Present	RO	1 = Vendor specific MMD present. 0 = Vendor specific MMD not present.	0
14	Vendor Specific MMD 1 Present	RO	1 = Vendor specific MMD present. 0 = Vendor specific MMD not present.	0
13	Clause 22 Extension Present	RO	1 = Clause 22 extension present 0 = Clause 22 extension not present	0
12-0	Reserved	RO	Ignore on read.	0

IEEE1 Clause 73 PHY Block (Offset: 0x0010) CL73_IEEEB1

CL73_IEEE1_CL73_AUTONEG_ADVERTISE (Offset: 0x1; Width: 16)

Table 47: CL73_IEEE1_CL73_AUTONEG_ADVERTISE (Offset: 0x1; Width: 16)

Bit	Name	R/W	Description	Default
15:12	Reserved	RO	n/a	0
7	Advertise 10 KR	R/W	Advertise 10G KR	0
6	Advertise 10G KX4	R/W	Advertise 10G KX4	0
5	Advertise 1000M KX	R/W	Advertise 1000M KX	0
4:3	Reserved	N/A	N/A	0
0	Advertise 1000M	R/W	Advertise 1000M	0

IEEE3 Clause 73 PHY Block (Offset: 0x0030)

Reserved

XGXS0 Block (Offset: 0x8000)

Reserved

XGXS1 Block (Offset: 0x8010)

Reserved

10G TX Bert Block (Offset: 0x8020)

Reserved

10G RX Bert Block (Offset: 0x8030)

Reserved

BcstBert Block (Offset: 0x8040)

Reserved

PLL Block (Offset: 0x8050)

Reserved

Transmit 0 Block (Offset: 0x8060)

Reserved

Transmit 1 Block (Offset: 0x8070)

Reserved

Transmit 2 Block (Offset: 0x8080)

Reserved

Transmit 3 Block (Offset: 0x8090)

Reserved

Transmit All Block (Offset: 0x80A0)

TXALL Status 0 Register (Offset: 0x0000; Width: 16)

Table 48: TXALL Status 0 Register (Offset: 0x0000; Width: 16)

Bit	Name	R/W	Description	Default
12:3	Reserved	RO	Ignore on read.	0
2	Transmit G-Loop	RO	G-Loop command (to serializer).	0
1	R-Loop Transmit FIFO Error	RO	R-Loop FIFO error.	0
0	Transmit FIFO error	RO	Transmit FIFO error.	0

TXALL Control 0 Register (Offset: 0x0007; Width: 16)

Table 49: TXALL Control 0 Register (Offset: 0x0007; Width: 16)

Bit	Name	R/W	Description	Default
15	Reserved	R/W	Write as 0, ignore on read.	0
14	Force Transmit Clock	R/W	Force txclk (bypass clock switch).	0
13	Transmit 1G FIFO Reset	R/W	Transmit 1G FIFO reset.	0
12	G-Loop Output Enable	R/W	Output data across serializer when G-Loop is enabled.	0
11-9	Reserved	R/W	Write a 0, ignore on read.	0
8	PRBS Enable	R/W	TX BERT enable.	0
7	Packet Enable	R/W	TX BERT packet mode enable.	0
6	Packet Start	R/W	TX BERT packet start.	0
5	Transmit Polarity Flip	R/W	Flip the analog transmit output.	
4	RTBI Flip	R/W	Flip the RTBI nibble sequence.	0

Table 49: TXALL Control 0 Register (Offset: 0x0007; Width: 16)

Bit	Name	R/W	Description	Default
3	eden_r	R/W	8B/10B enable.	0
2	eden_force_r	R/W	Enables eden_r.	0
1	Transmit Pattern Enable	R/W	Enables Transmit test pattern.	0
0	Transmit MDIO Data Enable	R/W	Enables transmit MDIO data.	

TXALL MDIO Data 0 Register (Offset: 0x0012; Width: 16)

Table 50: TXALL MDIO Data 0 Register (Offset: 0x0012; Width: 16)

Bit	Name	R/W	Description	Default
15-13	txTestMuxSelect	R/W		0
12-10	rlfifo_tstsel	R/W		0
9-0	TxMdioTstData[9:0]	R/W		0

TXALL MDIO Data 1 Register (Offset: 0x0013; Width: 16)

Table 51: TXALL MDIO Data 1 Register (Offset: 0x0013; Width: 16)

Bit	Name	R/W	Description	Default
15-10	Reserved	R/W	ignore on read.	0
9-0	TxMdioTstData[19:10]	R/W		0

TXALL Status 1 Register (Offset: 0x0014; Width: 16)

Table 52: TXALL Status 1 Register (Offset: 0x0014; Width: 16)

Bit	Name	R/W	Description	Default
15-14	Transmit ID	RO	Write on 0, ignore on read.	0
13-0	Reserved	R/W	ignore on read.	0

TXALL BG VCM Register (Offset: 0x0015; Width: 16)

Table 53: TXALL BG VCM Register (Offset: 0x0015; Width: 16)

Bit	Name	R/W	Description	Default
15-14	Reserved	R/W	Write as 0, ignore on read.	0
13	id2c[2]	R/W		0
12	refl_tx	R/W		0
11	refh_tx	R/W		0
10	newbias_en	R/W		0
9	drivermode	R/W		0
8	vddr_bgb	R/W		0
7-6	ticksel	R/W		0
5-4	driver_vcm	R/W		0

Table 53: TXALL BG VCM Register (Offset: 0x0015; Width: 16)

Bit	Name	R/W	Description	Default
3-0	Reserved	R/W	Write as 0, ignore on read.	0

TXALL IBuff 1T2T Register (Offset: 0x0016; Width: 16)

Table 54: TXALL IBuff 1T2T Register (Offset: 0x0016; Width: 16)

Bit	Name	R/W	Description	Default
15-14	icbuf1t	R/W		0
13-11	icbuf2t	R/W		0
10	imin_predrv	R/W		0
9	imax_predrv	R/W		0
8	imode_predrv	R/W		0
7-5	i21mux	R/W		0
4	imin_drv	R/W		0
3	imax_drv	R/W		0
2	imode_drv	R/W		0
1-0	id2c	R/W		0

TXALL Transmit Driver Register (Offset: 0x0017; Width 16)

Table 55: TXALL Transmit Driver Register (Offset: 0x0017; Width 16)

Bit	Name	R/W	Description	Default
15-12	Pre-emphasis	R/W	For BCM5709S AX and BX controllers: 0000 = 0.0dB 1000 = 0.4dB 0100 = 0.8dB 1100 = 1.2dB 0010 = 1.7dB 1010 = 2.2dB 0110 = 2.7dB 1110 = 3.2dB 0001 = 3.9dB 1001 = 4.4dB 0101 = 5.2dB 1101 = 5.8dB 0011 = 6.6dB 1011 = 7.5dB 0111 = 8.4dB 1111 = 9.6dB For BCM5709S CX and BCM5716S controllers: 0000 = 0.0dB 0001 = 0.4dB 0010 = 0.8dB 0011 = 1.2dB 0100 = 1.7dB 0101 = 2.2dB 0110 = 2.7dB 0111 = 3.2dB 1000 = 3.9dB 1001 = 4.4dB 1010 = 5.2dB 1011 = 5.8dB 1100 = 6.6dB 1101 = 7.5dB 1110 = 8.4dB 1111 = 9.6dB	0

Table 55: TXALL Transmit Driver Register (Offset: 0x0017; Width 16)

Bit	Name	R/W	Description	Default
11-8	iDriver	R/W	Specifies the voltage range. For BCM5709S AX & BX controllers: 0000 = 670mV 0001 = 720mV 0010 = 770mV 0011 = 820Mv 0100 = 870mV 0101 = 920mV 0110 = 970mV 0111 = 1020mV 1000 = 1070mV 1001 = 1110mV 1010 = 1160mV 1011 = 1210mV 1100 = 1260mV 1101 = 1310mV 1110 = 1360mV 1111 = 1400mV For BCM5709S CX+ and BCM5716S controllers: 0000 = 670mV 1000 = 720mV 0100 = 770mV 1100 = 820Mv 0010 = 870mV 1010 = 920mV 0110 = 970mV 1110 = 1020mV 0001 = 1070mV 1001 = 1110mV 0101 = 1160mV 1101 = 1210mV 0011 = 1260mV 1011 = 1310mV 0111 = 1360mV 1111 = 1400mV	9h
7-4	ipredriver	R/W		0
3-1	ifullspd	R/W		0
0	icbuf1t	R/W		0

Receive 0 Block (Offset: 0x80B0)

Reserved

Receive 1 Block (Offset: 0x80C0)

Reserved

Receive 2 Block (Offset: 0x80D0)

Reserved

Receive 3 Block (Offset: 0x80E0)

Reserved

Receive All Block (Offset: 0x80F0)

RXALL Receive Status Register (Offset: 0x0; Width: 16)

Table 56: RXALL Receive Status Register (Offset: 0x0; Width: 16)

Bit	*Name*	*R/W*	*Description*	*Default*
15	Signal Detect	R/W		0
14-13	Reserved	R/W	Write as 0, ignore on read.	0
12	RxSeqDone	R/W		0
11	rx_sigdet_II	RO	1 = High to low transition of rx_sigdet (analog output) was detected.	0
10	cx4_sigdet_II	RO	1 = High to low transition of cx4_sigdet was detected.	0
9-0	Reserved	RO	Ignore on read.	0

RXALL Receive Control Register (Offset: 0x01; Width 16)

Table 57: RXALL Receive Control Register (Offset: 0x01; Width 16)

Bit	*Name*	*R/W*	*Description*	*Default*
15	RxSeqRestart	R/W		0
14-13	Reserved	R/W	Write as 0, ignore on read.	0
12	sigDetected_en	R/W		0
11	sigdettRestart_en	R/W		0
10	sigdetMonitor_en	R/W		0
9	overide_sig_en	R/W		0
8	overide_sig_val	R/W		0
7-6	Reserved	R/W	Write as 0, ignore on read.	0
5	phfreq_rst_dis	R/W		0
4	forceRxSeqDone	R/W		0
3	Reserved	R/W	Write as 0, ignore on read.	0
2-0	status_sel	R/W		0

RXALL Receive Timer 1 Register (Offset: 0x02; Width 16)

Table 58: RXALL Receive Timer 1 Register (Offset: 0x02; Width 16)

Bit	Name	R/W	Description	Default
15-8	Reserved	R/W	Ignore on read	0
7-0	CDR Acquisition Time	R/W		0

RXALL Receive Timer 2 Register (Offset: 0x03; Width 16)

Table 59: RXALL Receive Timer 2 Register (Offset: 0x03; Width 16)

Bit	Name	R/W	Description	Default
15-8	Reserved	R/W	Ignore on read	0
7-0	CDR Track Time	R/W		0

RXALL Receive Signal Detect Register (Offset: 0x04; Width 16)

Table 60: RXALL Receive Signal Detect Register (Offset: 0x04; Width 16)

Bit	Name	R/W	Description	Default
15-8	sigdet_time	RO, R/W		0
7	cx4_sigdet_cnt_ld	R/W		0
6	ext_sigdet_en_SM	R/W		0
5	cx4_sigdet_en_SM	R/W		0
4	tpctrl	R/W		0
3-0	testMuxSelect	R/W		0

RXALL Receive CDR Phase Register (Offset: 0x05; Width 16)

Table 61: RXALL Receive CDR Phase Register (Offset: 0x05; Width 16)

Bit	Name	R/W	Description	Default
15	phase_freeze_en	R/W		0
14	phase_freeze_val	R/W		0
13-8	phase_offset	R/W		0
7	phase_override	R/W		0
6	phase_inc	R/W		0
5	phase_dec	R/W		0
4	phase_strobe	R/W		0
3-0	phase_delta	R/W		0

RXALL Receive CDR Frequency Register (Offset: 0x06; Width 16)

Table 62: RXALL Receive CDR Frequency Register (Offset: 0x06; Width 16)

Bit	Name	R/W	Description	Default
15-12	bwsel_prop_trck	R/W		0
11-8	bwsel_intg_trck	R/W		0
7	falling_edge	R/W		0
6	flip_polarity	RO		0
5	freq_override_en	R/W		0
4-0	freq_override_val	R/W		0

RXALL Receive Equalizer Configuration Register (Offset: 0x07; Width)

Table 63: RXALL Receive Equalizer Configuration Register (Offset: 0x07; Width)

Bit	Name	R/W	Description	Default
15-8	Reserved	R/W	Write as 0, ignore on read	0
7-4	bwsel_prop_acq	R/W		0
3-0	bwsel_intg_acq	R/W		0

RXALL Receive Equalizer Force Register (Offset: 0x08; Width 16)

Table 64: RXALL Receive Equalizer Force Register (Offset: 0x08; Width 16)

Bit	Name	R/W	Description	Default
15-0	Reserved	R/W	Write as 0, ignore on read	0

RXALL Receive Control 1G Register (Offset: 0x09; Width 16)

Table 65: RXALL Receive Control 1G Register (Offset: 0x09; Width 16)

Bit	Name	R/W	Description	Default
15	fpat_md	R/W	Fixed pattern mode enable.	0
14	pkt_count_en	R/W	Packet counter enable.	0
13	staMuxRegsDis	R/W	Disable registering of status mux.	0
12	prbs_clr_dis	R/W	Disable PRBS clear-on-read.	0
11	rxd_dec_sel	R/W	Select 8B/10B output for PRBS monitor.	0
10	cgbad_tst	R/W	Define error code as 10'h3FE for PRBS monitor.	0
9	Emon_en	R/W	Enable |E| monitor.	0
8	prbs_en	R/W	Enable PRBS monitor.	0
7	cgbad_en	R/W	Set bit 9 of symbol when bad symbol is detected.	0
6	cstretch	RO	Enable rxck0_1g clock/data phase alignment.	0
5	rtbi_ckflip	RO	Flip RTBI nibble clock phasing.	0
4	rtbi_flip	R/W	Flip RTBI (re-ordering) data sequence.	0

Table 65: RXALL Receive Control 1G Register (Offset: 0x09; Width 16)

Bit	Name	R/W	Description	Default
3	phase_sel	R/W	Phfreqloops phase select.	0
2	Reserved	R/W	Write as 0, ignore on read.	0
1	freq_sel_force	R/W	Force freq_sel.	0
0	freq_sel	R/W	0 = div/1 1 = div/2	0

RXALL Receive Control PCI Express Register (Offset: 0x0A; Width 16)

Table 66: RXALL Receive Control PCI Express Register (Offset: 0x0A; Width 16)

Bit	Name	R/W	Description	Default
15	comma_adj_sync_sel	R/W	Used sync_status as comma_adj_en.	0
14	com_mask_force_r	R/W	Force com_mask_r.	0
13	com_mask	R/W	Defines |K| as 10-bit symbol (used for PCIe).	0
12	sync_status_force_sync	R/W	Derive sync_status from sync acquisition block.	0
11	sync_status_force_r	R/W	Derive sync_status from sync_status_r.	0
10	sync_status_r	RW	MDIO controller sync_status.	0
9	comma_adj_en_force_ext	R/W	Derive comma_adj_en from external pin.	0
8	comma_adj_en_force_sync	R/W	Derive comma_adj_en from sync_status.	0
7	comma_adj_en_force_r	R/W	Derive comma_adj_en from comma_adj_en_r.	0
6	comm._adj_en_r	RO	MDIO controller comma_adj_en.	0
5	link_en_force_r	RO	Enable link_en via link_en_r.	0
4	link_en_r	R/W	Link enable.	0
3	rx_polarity_force_r	R/W	Derive RX polarity from rx_polarity_r.	0
2	rx_polarity_r	R/W	Invert RX polarity.	0
1-0	integ_mode	R/W	Integration mode.	0

RXALL Receive All Status Register (Offset: 0x0B; Width 16)

Table 67: RXALL Receive All Status Register (Offset: 0x0B; Width 16)

Bit	Name	R/W	Description	Default
15-1	Reserved	R/W	Ignore on read	0
0	sigdet	RO		0

RXALL Receive Equalizer Boost Register (Offset: 0x0C; Width 16)

Table 68: RXALL Receive Equalizer Boost Register (Offset: 0x0C; Width 16)

Bit	Name	R/W	Description	Default
15	imode_vcm	R/W		0
14	imin_vcm	R/W		0

Table 68: RXALL Receive Equalizer Boost Register (Offset: 0x0C; Width 16)

Bit	Name	R/W	Description	Default
13	imax_sigdet	R/W		0
12	imode_sigdet	R/W		0
11	imin_sigdet	R/W		0
10	refh_rx	R/W		0
9	refl_rx	R/W		0
8	tport_en	R/W		0
7	vddr_bg	R/W		0
6	sig_pwrdn	R/W		0
5-3	offset_ctrl	R/W		0
2-0	equalizer_ctrl	R/W		0

RXALL Receive Ib Data Equalizer Register (Offset: 0x0D; Width 16)

Table 69: RXALL Receive Ib Data Equalizer Register (Offset: 0x0D; Width 16)

Bit	Name	R/W	Description	Default
15	imax_phase	R/W		0
14	imode_phase	R/W		0
13	imin_phase	R/W		0
12	imax_div2	R/W		0
11	imode_div2	R/W		0
10	imin_div2	R/W		0
9	imax_eqfl	R/W		0
8	imode_eqfl	R/W		0
7	imin_eqfl	R/W		0
6	imax_slic	R/W		0
5	imode_slic	R/W		0
4	imin_slic	R/W		0
3	imax_bufd2c	R/W		0
2	imode_bufd2c	R/W		0
1	imin_bufd2c	R/W		0
0	imax_vcm	R/W		0

RXALL Receive Ib ADC Buffer Register (Offset: 0x0E; Width 16)

Table 70: RXALL Receive Ib ADC Buffer Register (Offset: 0x0E; Width 16)

Bit	Name	R/W	Description	Default
15-12	adc_clkdelay	R/W		0
11	imax_s/n	R/W		0
10	imode_s/n	R/W		0
9	imin_s/n	R/W		0
8	imax_adcdrv	R/W		0

Table 70: RXALL Receive Ib ADC Buffer Register (Offset: 0x0E; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
7	imode_adcdrv	R/W		0
6	imin_adcdrv	R/W		0
5	imax_adc	R/W		0
4	imode_adc	R/W		0
3	imin_adc	R/W		0
2	imax_eqbuf	R/W		0
1	imode_eqbuf	R/W		0
0	imin_eqbuf	R/W		0

XGXS Block 2 (Offset: 0x8100)

XGXS BLOCK 2 RX LANE SWAP (Offset: 0x0; Width 16)

Table 71: XGXS BLOCK 2 RX LANE SWAP (Offset: 0x0; Width 16)

Bit	Name	R/W	Description	Default
15	Rx Lane Swap Enable	R/W	Enables Rx Lane Swap (re-ordering)	strap
14	Rx Ln Swap Force En	R/W	Enables Rx lane re-ordering using bits 13:0	strap
12	Rx Ln Swap Link En	R/W	Enables link ordering	0
12:8	Qx MSB	R/W	MSB of Lane 0-3 Q character	0x8000
7:0	Rx Ln Swap Force X	R/W	Lane 0-3 manual override Rx swap MUX select	0x3210

XGXS BLOCK 2 TX LANE SWAP (Offset: 0x1; Width 16)

Table 72: XGXS BLOCK 2 TX LANE SWAP (Offset: 0x1; Width 16)

Bit	Name	R/W	Description	Default
15	Tx Lane Swap Enable	R/W	Enables Tx Lane Swap	strap
14:8	Reserved	R/W	Ignore on read	N/A
7:0	Tx Ln Swap Force X	R/W	Lane 0-3 manual override Tx swap MUX select	0x3210

XGXS BLOCK 2 UNI-core Mode (Offset: 0x04; Width 16)

Table 73: XGXS BLOCK 2 UNI-core Mode (Offset: 0x04; Width 16)

Bit	Name	R/W	Description	Default
15:8	Reserved	R/W	Reserved	N/A
7:4	Unicore 10gHiG	R/W	XGXS mode bit 4	strap
3:0	Unicore 10gCx4	R/W	XGXS mode bit 0	strap

XGXS BLOCK Test Mode Lane (Offset: 0x05; Width 16)

Table 74: XGXS BLOCK Test Mode Lane (Offset: 0x05; Width 16)

Bit	Name	R/W	Description	Default
15:2	Reserved	R/W	Reserved	N/A
1:0	Slice selector	R/W	XGXS mode bit 4	strap
3:0	Unicore 10gCx4	R/W	XGXS mode bit 0	strap

General Purpose Status Block (Offset: 0x8120)

GP Status Miscellaneous RX Status Register (Offset: 0x0; Width 16)

Table 75: GP Status Miscellaneous RX Status Register (Offset: 0x0; Width 16)

Bit	Name	R/W	Description	Def
15	capture_NP_lh	RO, LH		0
14	teton_brk_link_lh	RO, LH		0
13	UP3_lh	RO, LH		0
12	MP5_lh	RO, LH		0
11	nonMatchingOUI_lh	RO, LH		0
10	matchingOUI_msb_lh	RO, LH		0
9	matchingOUT_lsb_lh	RO, LH		0
8	invalidSeq_lh	RO, LH	Indicates that an invalid message/next page sequence was detected. An example would be the reception of an over-1G message page followed by fewer than two next pages.	0
7	mullMP_lh	RO, LH	Null message page detected.	0
6	remotePhyMP_lh	RO, LH	Remote PHY message page detected.	0
5	nonMatchingMP_lh	RO, LH	Non-Matching next page detected (neither over-1G nor Remote PHY).	0
4	over1gMP_lh	RO, LH	Over-1G message page detected.	0
3	rx_config_is_0_lh	RO, LH	All-zero rx_config detected (this indicates that the link partner initiated an auto-neg sequence.)	0
2	np_toggle_err_lh	RO, LH	Next page toggle error detected.	0
1	mr_np_lh	RO, LH	Indicates reception of a next page.	0
0	mr_bp_lh	RO, LH	Indicates reception of a base page.	0

GP Status XGXS Status 0 Register (Offset 0x01; Width 16)

Table 76: GP Status XGXS Status 0 Register (Offset 0x01; Width 16)

Bit	Name	R/W	Description	Default
15	read_control	RO	mdio_csreg status register control bit.	1
14	Reserved	RO	Ignore on read.	0
13	tx_remote_fault	RO, LH	Remote fault on transmit path.	0
12	rx_remote_fault	RO, LH	Remote fault on receive path.	0
11	txpll_lock	RO	Transmit PLL lock indicator (from analog tx_pll block).	0
10	txd_fifo_err	RO, Sticky	FIFO error for the 4 lanes (from tx_fifo).	0
9	sequencer_done	RO	PLL sequencer.	0
8	sequencer_pass	RO	PLL sequencer finished successfully.	0
7:4	rxferr	RO, Sticky	Per-lane receiver FIFO error indicator.	0
3	Reserved	RO	Ignore on read.	0
2	ckcmp_unflow	RO, Sticky	Clock compensation FIFO underflow.	0
1	ckcmp_ovflow	RO, Sticky	Clock compensation FIFO overflow.	0
0	skew_status	RO	Skew status (from rx_link).	0

GP Status XGXS Status 1 Register (Offset 0x02; Width 16)

Table 77: GP Status XGXS Status 1 Register (Offset 0x02; Width 16)

Bit	Name	R/W	Description	Default
15:12	mode_10g_tx	RO	Mode: 0 = XGXS 1 = XGXS, No Clock Conpensation 6 = IndLn 7 = PCI 8 = XGXS No LSS Q 9 = XGXS No Lss Q, No Clock Compensation A = PBypsDsk B = PBypsNoDsk C = Combo Core Mode (SerDes/Unicore: 10M, 100M, 1G, 2.5G, and auto-neg to XGXS speeds) F = Clocks Off All other values are reserved.	0
11	serdesMode_en_tx	RO	Indicates current mode is SerDes mode (i.e. 10M, 100M, 1G, or 2.5G).	0
10	sgmii_mode	RO	Indicates current mode is SerDes/SGMII mode (i.e. 10M, 100M, or 1G).	0
9	link10g	RO	Indicates XGXS link status.	0
8	linkstat	RO	Indicates XGXS link status.	0
7	autoneg_complete	RO	Indicates autonegotiation is complete.	0

Table 77: GP Status XGXS Status 1 Register (Offset 0x02; Width 16)

Bit	Name	R/W	Description	Default
6	Reserved	RO	Ignore on read.	0
5-0	Actual Speed	RO	Indicates the current UniCore speed: 0 = 10M 1 = 100M 2 = 1G 3 = 2.5G 4 = 5G 5 = 6G 6 = 10G HiG 7 = 10G CX4 8 = 12G 9 = 12.5G A = 13G B = 15G C = 16G All other values are reserved.	0

GP Status XGXS Status 2 Register (Offset 0x03; Width 16)

Table 78: GP Status XGXS Status 2 Register (Offset 0x03; Width 16)

Bit	Name	R/W	Description	Default
15-12	gpwrdwn_rx	R/W		0
11-8	gpwrdwn_tx	R/W		0
7-4	freq_sel_rx	R/W		0
3-0	freq_sel_tx	R/W		0

GP Status 1000X Status 1 Register (Offset: 0x04; Width 16)

Table 79: GP Status 1000X Status 1 Register (Offset: 0x04; Width 16)

Bit	Name	R/W	Description	Default
15	Transmit FIFO Error Detected	RO, LH	1 = Transmit FIFO error detected since last read 0 = No transmit FIFO error detected since last read	0
14	Receive FIFO Error Detected	RO, LH	1 = Receive FIFO error detected since last read 0 = No receive FIFO error detected since last read	0
13	False Carrier Detected	RO, LH	1 = False carrier detected since last read 0 = No false carrier detected since last read	0
12	CRC Error Detected	RO, LH	1 = CRC error detected since last read 0 = No CRC error detected since last read	0
11	Transmit Error Detected	RO, LH	1 = Transmit error code detected since last read 0 = No transmit error code detected since last read	0

Table 79: GP Status 1000X Status 1 Register (Offset: 0x04; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
10	Receive Error Detected	RO, LH	1 = Receive error detected since last read 0 = No receive error detected since last read	0
9	Carrier Extend Error Detected	RO, LH	1 = Carrier extend error detected since last read 0 = No carrier extend error detected since last read	0
8	Early End Extension Detected	RO, LH	1 = Early end extension detected since last read 0 = No early end extension detected since last read	0
7	Link Status	RO, LH	1 = Link status has not changed since last read 0 = Link status has changed since last read	0
6	Receive Pause Resolution	RO	1 = Enable pause receive 0 = Disable pause receive	0
5	Transmit Pause Resolution	RO	1 = Enable pause transmit 0 = Disable pause transmit	0
4-3	Speed Status	RO	0h = 10Mbps 1h = 100Mbps 2h = 1000Mbps 3h = 2500Mbps	0
2	Duplex Status	RO	1 = Full-duplex 0 = Half-duplex	0
1	Link Status	RO	1 = Link is up 0 = Link is down	0
0	SGMII Mode	RO	1 = SGMII mode 0 = Fiber mode (1000Base-X)	0

GP Status 1000X Status 2 Register (Offset: 0x05; Width 16)

Table 80: GP Status 1000X Status 2 Register (Offset: 0x05; Width 16)

Bit	Name	R/W	Description	Default
15	SGMII Mode Change	RO, LH	1 = SGMII mode has changed since last read 0 = SGMII mode has not changed since last read	0
14	Consistency Mismatch	RO, LH	1 = Consistency mismatch detected since last read 0 = No consistency mismatch detected since last read	0
13	Autonegotiation Resolution Error	RO, LH	1 = Autonegotiation HCD error detected since last read 0 = No autonegotiation HCD error detected since last read	0
12	SGMII Selector Mismatch	RO, LH	1 = SGMII selector mismatch detected since last read (autonegotiation page received from link partner with bit 0 = 0 while local device is in SGMII mode) 0 = No SGMII selector mismatch detected since last read	0

Table 80: GP Status 1000X Status 2 Register (Offset: 0x05; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
11	Sync Status Failed	RO, LH	1 = Sync status has failed since last read (synchronization has been lost) 0 = Sync status has not failed since last read	0
10	Sync Status OK	RO, LH	1 = Sync_Status OK detected since last read (synchronization has been achieved) 0 = Sync_Status OK has not been detected since last read	0
9	RUDI_C	RO, LH	1 = RUDI_C detected since last read 0 = RUDI_C has not been detected since last read	0
8	RUDI_L	RO, LH	1 = RUDI_L detected since last read 0 = RUDI_L has not been detected since last read	0
7	RUDI_Invalid	RO, LH	1 = RUDI_Invalid detected since last read 0 = RUDI_Invalid has not been detected since last read	0
6	Link Down from Sync Loss	RO, LH	1 = A valid link went down due to loss of synchronization for over 10ms 0 = No link down due to synchronization loss has been detected	0
5	Idle Detect State	RO, LH	1 = Idle detect state in autonegotiation FSM entered since last read 0 = Idle detect state not entered since last read	0
4	Complete Acknowledge State	RO, LH	1 = Complete acknowledge state in autonegotiation FSM entered since last read 0 = Complete acknowledge state in autonegotiation FSM not entered since last read	0
3	Acknowledge Detect State	RO, LH	1 = Acknowledge detect state in autonegotiation FSM entered since last read 0 = Acknowledge detect state in autonegotiation FSM not entered since last read	0
2	Ability Detect State	RO, LH	1 = Ability detect state in autonegotiation FSM entered since last read 0 = Ability detect state in autonegotiation FSM not entered since last read	0
1	Autonegotiation Error State	RO, LH	1 = Autonegotiation error state in autonegotiation FSM entered since last read 0 = Autonegotiation error state in autonegotiation FSM not entered since last read	0
0	Autonegotiation Enable State	RO, LH	1 = Autonegotiation enable state in autonegotiation FSM entered since last read 0 = Autonegotiation enable state in autonegotiation FSM not entered since last read	0

GP Status 1000X Status 3 Register (Offset: 0x06; Width 16)

Table 81: GP Status 1000X Status 3 Register (Offset: 0x06; Width 16)

Bit	Name	R/W	Description	Default
15-13	Reserved	RO	Ignore on read.	0
12	pd_park_an	RO		0
11	remotePhy_autosel	RO		0
10	latch_linkdown	RO		0
9	sd_filter	RO	1 = Output of signal detect filter is set 0 = Output of signal detect filter is not set	0
8	sd_mux	RO	1 = Input of signal detect filter is set 0 = Input of signal detect filter is not set	0
7	sd_filter_chg	RO, LH	1 = Signal detect has changed since last read 0 = Signal detect has not changed since last read	0
6-0	Reserved			0

GP Status TPOUT 1 Register (Offset: 0x07; Width 16)

Table 82: GP Status TPOUT 1 Register (Offset: 0x07; Width 16)

Bit	Name	R/W	Description	Default
15-0	tpout[15:0]	R/W		0

GP Status TPOUT 2 Register (Offset: 0x08; Width 16)

Table 83: GP Status TPOUT 2 Register (Offset: 0x08; Width 16)

Bit	Name	R/W	Description	Default
15-0	tpout[23:8]	R/W		0

GP Status XGXS Status 3 Register (Offset: 0x09; Width 16)

Table 84: GP Status XGXS Status 3 Register (Offset: 0x09; Width 16)

Bit	Name	R/W	Description	Default
15	link			0
14	link_latchdown			0
13	link_latchdown_10g_o			0
12	pd_park_an			0
11	gpwrdwn_pll			0
10-0	hcd_over_1g			0

GP Status x2500 Status 1 Register (Offset: 0x0A; Width 16)

Table 85: GP Status x2500 Status 1 Register (Offset: 0x0A; Width 16)

Bit	Name	R/W	Description	Default
15	hcd_over_1g_or			0
14	latch_hcd_over_1g			0
13	latchmido			0
12	s_bc_reg_rst			0
11	s_wait2res			0
10	s_wait30ms			0
9	s_clockswit			0
8	s_pllswit			0
7	s_wait4link			0
6	s_complete			0
5	s_lostlink			0
4	s_dead			0
3-0	fail_cnt			0

GP Status Top Autonegotiation Status Register (Offset: 0x0B; Width 16)

Table 86: GP Status Top Autonegotiation Status Register (Offset: 0x0B; Width 16)

Bit	Name	R/W	Description	Default
15-14	Reserved	R/W	Write as 0, ignore on read.	0
13-8	Actual Speed	R/O	00h = 10Mb 01h = 100Mb 02h = 1Gb 03h = 2.5Gb 04h = 5Gb 05h = 6Gb 06h = 10G HiG 07h = 10G CX4 08h = 12G HiG 09h = 12.5Gb 0Ah = 13Gb 0Bh = 15Gb 0Ch = 16Gb 0Dh = 1GBase-KX 0Eh = 10GBase-KX4 All others reserved.	0
7	Receive Pause Resolution	R/O	1 = Enable pause receive 0 = Disable pause receive	0
6	Transmit Pause Resolution	R/O	1 = Enable pause transmit 0 = Disable pause transmit	0

Table 86: GP Status Top Autonegotiation Status Register (Offset: 0x0B; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
5	Clause 73 MRBE Capable	R/O	1 = Indicates that the LP and the LD support MRBE for clause 37 autonegotiation. This bit is asserted when both the LD and LP have successfully exchanged MRBE Clause 73 NPs and, therefore, determined that a switch over to clause 37 autonegotiation will follow.	0
4	Clause 73 Autonegotiation Capable	R/O	1 = The link partner does support autonegotiation 0 = The link partner does not support autonegotiation	0
3	Duplex Status	R/O	1 = Full-duplex 0 = Half-duplex	0
2	Link Status	R/O	1 = Link is up 0 = Link is down	0
1	Clause 37 Autonegotiation Status	R/O	1 = Clause 37 Autonegotiation is complete 0 = Clause 37 Autonegotiation is not complete	0
0	Clause 73 Autonegotiation Status	R/O	1 = Clause 73 Autonegotiation is complete 0 = Clause 73 Autonegotiation is not complete	0

GP Status LP_UP1 Register (Offset: 0x0C; Width 16)

Table 87: GP Status LP_UP1 Register (Offset: 0x0C; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Ignore on read.	0
10-0	lp_adv_over_1g[10:0]	RO		0

GP Status LP_UP2 Register (Offset: 0x0D; Width 16)

Table 88: GP Status LP_UP2 Register (Offset: 0x0D; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Ignore on read.	0
10-0	lp_adv_over_1g[21:11]	RO		0

GP Status LP_UP3 Register (Offset: 0x0E; Width 16)

Table 89: GP Status LP_UP3 Register (Offset: 0x0E; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Ignore on read.	0
10-0	lp_adv_over_1g[31:22]	RO		0

SerDes Digital Block (Offset : 0x8300)

SerDes Digital 1000X Control 1 Register (Offset: 0x0; Width 16)

Table 90: SerDes Digital 1000X Control 1 Register (Offset: 0x0; Width 16)

Bit	Name	R/W	Description	Default
15	Reserved	R/W	Write as 0, ignore on read.	0
14	Disable Signal Detect Filter	R/W	1 = Disable filter for signal detect. 0 = Filter signal detect from pin before using for synchronization.	0
13	Master MDIO PHY Select	R/W	1 = All MDIO write accesses to PHY address "00000" will write this PHY in addition to its own PHY address. 0 = Normal operation.	0
12	SerDes TX Amplitude Override	R/W	1 = Overide SerDes transmit amplitude from register 1*10h bit 14. 0 = Normal operation	0
11	Select RX Packets for Counter	R/W	1 = Select received packets for 0 * 17h counter. 0 = Select CRC packets for 0 * 17h counter.	0
10	Remote Loopback	R/W	1 = Enable remote loopback (only operates at gigabit speed) 0 = Normal operation	0
9	Zero Comma Detector Phase	R/W	1 = Force comma detector phase to zero 0 = Normal operation	0
8	Comma Detection Enable	R/W	1 = Enable comma detection 0 = Disable comma detection	1
7	CRC Checker Disable	R/W	1 = Disable CRC checker by gating the clock to save power 0 = Enable CRC checker	1
6	Disable PLL Powerdown	R/W	1 = PLL will never be powered down. (Use this when the MAC/Switch uses tx_wclk_o ouput.) 0 = PLL will be powered down when 0.11 is set	0
5	SGMII Master Mode	R/W	1 = SGMII mode operates in "PHY Mode", sending out link, speed, and duplex settings from register 0 to link partner. 0 = Normal operation.	0
4	Auto-detection Enable	R/W	1 = Enable auto-detection (fiber and SGMII mode will switch each time an autonegotiation page is received with the wrong selector field in bit 0). 0 = Disable auto-detection (fiber or SGMII mode is set according to bit 0 of this register).	0
3	Invert Signal Detect	R/W	1 = Invert signal detect from pin 0 = Use signal detect from pin	0

Table 90: SerDes Digital 1000X Control 1 Register (Offset: 0x0; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
2	Signal Detect Enable	R/W	1 = Signal detect from pin must be set in order to achieve synchronization. In SGMII the signal detect is always ignored regardless of the setting of this bit. 0 = Ignore signal detect from pin.	1
1	TBI Interface	R/W	1 = Ten-bit interface 0 = GMII Interface	0
0	Fiber Mode 1000x	R/W	1 = Fiber mode (1000-X) 0 = SGMII mode	1

SerDes Digital 1000X Control 2 Register (Offset: 0x01; Width 16)

Table 91: SerDes Digital 1000X Control 2 Register (Offset: 0x01; Width 16)

Bit	Name	R/W	Description	Default
15	Disable Extend_Full-Duplex Only	R/W		0
14	Clear BER Counter	R/W, SC	1 = Clear bit-error-rate counter 0 = Normal operation	0
13	Transmit Idle Jam Sequence Test	R/W	1 = Enable test sequence to SerDes transmitter 0 = Normal operation	0
12	Transmit Packet Sequence Test	R/W	1 = Enable transmit test sequence to SerDes transmitter 0 = Normal operation	0
11	Test Counter	R/W	1 = Increment 0 * 17h counter each clock cycle for testing 0 = Normal operation	0
10	Bypass PCS Transmit	R/W	1 = Bypass PCS transmit operation 0 = Normal operation	0
9	Bypass PCS Receive	R/W	1 = Bypass PCS receive operation 0 = Normal operation	0
8	Disable TRRR Generation	R/W	1 = Disable TRRR generation in PCS transmit 0 = Normal operation	0
7	Disable Carrier Extend	R/W	1 = Disable carrier extension in PCS receive 0 = Normal operation	0
6	Autonegotiation Fast Timers	R/W	1 = Speed up timers in autonegotiation for testing 0 = Normal operation	0
5	Force Transmit Data on Transmit Side	R/W	1 = Allow packets to be transmitted regardless of the condition of the link or synchronization 0 = Normal operation	0
4	Disable Remote Fault Sensing	R/W	1 = Disable automatic sensing of remote faults such as autonegotiation errors 0 = Automatically detect remote faults and send remote fault status to link partner via autonegotiation when fiber mode is selected (SGMII does not support remote faults).	0

Table 91: SerDes Digital 1000X Control 2 Register (Offset: 0x01; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
3	Enable Autonegotiation Error Timer	R/W	1 = Enable autonegotiation error timer. Error occurs when timer expires in ability-detect, ack-detect, or idle-detect. When the error occurs, config words of all zeros are sent until an ability match occurs, the autonegotiation-enable state is entered. 0 = Normal operation	0
2	Filter Force Link	R/W	1 = Sync-status must be set for 10ms before a valid link is established when autonegotiation is disabled. (This is useful in fiber applications where the user does not have the signal detect pin connected to the fiber modules and autonegotiation is turned off.) 0 = Normal operation.	1
1	Disable False Link	R/W	1 = Do not allow link to be established when autonegotiation is disabled and receiving autonegotiation code words. The link will only be established in this case after idles are received. (This bit does not need to be set if bit 0 below is set.) 0 = Normal operation.	1
0	Enable Parallel Detection	R/W	1 = Enable parallel detection. (This will turn autonegotiation on and off as needed to properly link up with the link partner. The idles and autonegotiation code words received from the link partner are used to make this decision). 0 = Disable parallel detection.	1

SerDes Digital 1000X Control 3 Register (Offset: 0x02; Width 16)

Table 92: SerDes Digital 1000X Control 3 Register (Offset: 0x02; Width 16)

Bit	Name	R/W	Description	Default
15	Disable Packet Misalign	R/W		0
14	Receive FIFO GMII Reset	R/W		0
13	Disable TX CRS	R/W	1 = Disable generating CRS from transmitting in half-duplex. Only receiving will generate CRS. 0 = Normal operation	0
12	Invert External PHY CRS	R/W	1 = Invert "receive CRS from PHY" pin 0 = Use "receive CRS from PHY" pin	0
11	External PHY CRS Mode	R/W	1 = User external pin for the PHY's "receive only" CRS output. (Useful in SGMII 10/100 half-duplex environments in order to reduce the collision domain latency. Requires a PHY which generates a "receive only" CRS output to a pin.)	0
10	Jam False Carrier Mode	R/W	1 = Change false carriers received into packets with preamble only. (Not necessary if MAC uses CRS to determine collision.) 0 = Normal operation	0
9	Block Transmit Enable Mode	R/W	1 = Block TXEN when necessary to guarantee an IPG of at least 6.5 bytes in 10/100 mode, 7 bytes in 1000 mode. 0 = Normal operation	0
8	Force Transmit FIFO On	R/W	1 = Force transmit FIFO to free-run in gigabit mode. (Requires clk_in and tx_wclk_o to be frequency locked.) 0 = Normal operation	0

Table 92: SerDes Digital 1000X Control 3 Register (Offset: 0x02; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
7	Bypass Transmit FIFO 1000	R/W	1 = Bypass transmit FIFO in gigabit mode. (Useful for fiber of gigabit only environments where the MAC is using the tx_wclk_o as the clk_in port. User must meet timing to the tx_wclk_o domain.) 0 = Normal operation	0
6	Frequency Lock Elasticity TX	R/W	1 = Minimum FIFO latency to properly handle a clock which is frequency locked but out of phase. (Overrides bits 2:1 of this register.) 0 = Normal operation	0
5	Frequency Lock Elasticity RX	R/W	1 = Minimum FIFO latency to properly handle a clock which is frequency locked but out of phase. (Not necessary if MAC uses CRS to determine collision. Overrides bits 2:1 of this register.) 0 = Normal operation	0
4	Early Preamble RX	R/W	1 = Send extra bytes of preamble to avoid FIFO latency. (Not necessary if MAC uses CRS to determine collision.) 0 = Normal operation	0
3	Early Preamble TX	R/W	1 = Send extra bytes of preamble to avoid FIFO latency. (Used in half-duplex environments to reduce collision domain latency. MAC must send 5 bytes of preamble or less to avoid non-compliant behavior.) 0 = Normal operation	0
2-1	FIFO Elasticity TX/RX	R/W	00 = Support packets up to 5KB 01 = Support packets up to 10KB 1X = Supports packets up to 13.5KB	1h
0	Transmit FIFO Reset	R/W	1 = Reset transmit FIFO. FIFO will remain in reset until the bit is cleared with a software write. 0 = Normal operation	0

SerDes Digital 1000X Control 4 Register (Offset: 0x03; Width 16)

Table 93: SerDes Digital 1000X Control 4 Register (Offset: 0x03; Width 16)

Bit	Name	R/W	Description	Default
15-14	Reserved	R/W	Write as 0, ignore on read.	0
13	disable_resolution_err_restart	R/W		0
12	enable_last_resolution_err	R/W		0
11	tx_config_reg_sel	R/W		0
10	zero_rxdgmii	R/W		0
9	clear_linkdown	R/W		0
8	latch_linkdown_enable	R/W		0
7	link_force	R/W		0
6	Reserved	R/W	Write as 0, ignore on read.	0
5	lp_next_page_sel	R/W		0
4	np_count_ClrnBp	R/W		0
3	np_count_ClrnRd	R/W		0
2-0	MiscRxStatus_sel	R/W		0

SerDes Digital 1000X Status 1 Register (Offset: 0x04; Width 16)

Table 94: SerDes Digital 1000X Status 1 Register (Offset: 0x04; Width 16)

Bit	Name	R/W	Description	Default
15	Transmit FIFO Error Detected	RO, LH	1 = Transmit FIFO error detected since last read 0 = No transmit FIFO error detected since last read	0
14	Receive FIFO Error Detected	RO, LH	1 = Receive FIFO error detected since last read 0 = No receive FIFO error detected since last read	0
13	False Carrier Detected	RO, LH	1 = False carrier detected since last read 0 = No false carrier detected since last read	0
12	CRC Error Detected	RO, LH	1 = CRC error detected since last read 0 = No CRC error detected since last read	0
11	Transmit Error Detected	RO, LH	1 = Transmit error code detected since last read (rx_data_error state in PCS receive FSM) 0 = No transmit error code detected since last read	0
10	Receive Error Detected	RO, LH	1 = Receive error detected since last read (early_end in PCS receive FSM) 0 = No receive error detected since last read	0
9	Carrier Extend Error Detected	RO, LH	1 = Carrier extend error detected since last read (extend_err in PCS receive FSM) 0 = No carrier extend error detected since last read	0
8	Early End Extension Detected	RO, LH	1 = Early end extension detected since last read (early_end_ext in PCS receive FSM) 0 = No early end extension detected since last read	0
7	Link Status Change Detected	RO, LH	1 = Link status has changed since last read 0 = Link status has not changed since last read	0
6	Pause Resolution Receive Side	RO	1 = Enable pause receive 0 = Disable pause receive	0
5	Pause Resolution Transmit Side	RO	1 = Enable pause transmit 0 = Disable pause transmit	0
4-3	Speed Status	RO	11 = 2.5Gb 10 = 1Gb 01 = 100Mb 00 = 10Mb	0
2	Duplex Status	RO	1 = Full-duplex 0 = Half duplex	0
1	Link Status	RO	1 = Link is up 0 = Link is down	0
0	SGMII Mode	RO	1 = SGMII mode 0 = Fiber mode (1000-X)	0

SerDes Digital 1000X Status 2 Register (Offset: 0x05; Width 16)

Table 95: SerDes Digital 1000X Status 2 Register (Offset: 0x05; Width 16)

Bit	Name	R/W	Description	Default
15	sgmii_mode_change			0
14	consistency_mismatch			0
13	autoneg_resolution_err			0
12	sgmii_selector_msimatch			0
11	sync_status_fail			0
10	sync_status_ok			0
9	rudi_c			0
8	rudi_i			0
7	rudi_invalid			0
6	link_went_down_from_loss_of_sync			0
5	idle_detect_state			0
4	complete_acknowledge_state			0
3	acknowledge_detect_state			0
2	ability_detect_state			0
1	an_error_state			0
0	an_enable_state			0

SerDes Digital 1000X Status 3 Register (Offset: 0x06; Width 16)

Table 96: SerDes Digital 1000X Status 3 Register (Offset: 0x06; Width 16)

Bit	Name	R/W	Description	Default
15-13	Reserved	R/W	Write as 0, ignore on read.	0
12	pd_park_an			0
11	remotePhy_autosel			0
10	latch_linkdown			0
9	sd_filter			0
8	sd_mux			0
7	sd_filter_chg			0
6-0	Reserved	R/W	Write as 0, ignore on read.	0

SerDes Digital CRC Err and Rx Packet Counter Register (Offset: 0x07; Width 16)

Table 97: SerDes Digital CRC Err and Rx Packet Counter Register (Offset: 0x07; Width 16)

Bit	Name	R/W	Description	Default
15-8	bit_error_rate_counter			0
7-0	crc_err_rx_pkt_cntr			0

SerDes Digital Miscellaneous 1 Register (Offset: 0x08; Width 16)

Table 98: SerDes Digital Miscellaneous 1 Register (Offset: 0x08; Width 16)

Bit	Name	R/W	Description	Default
15-13	Reference Clock Select	R/W	0 = 25MHz 1 = 100MHz 2 = 125MHz 3 = 156.25MHz 4 = 187.5MHz	3
12	Force PLL Mode AFE Select	R/W	Forces on force_pll_mode_afe[2:0]	0
11	Reserved	R/W	Write as 0, ignore on read.	0
10-8	force_pll_mode_afe[2:0]	R/W	Determines PLL clock multiplier factor: 0 = x16 1 = x20 2 = x25 3 = x25 4 = x26 5 = x30 6 = x32 7 = x40	0
7	Reserved	R/W	Write as 0, ignore on read.	0
6	Transmit Underrun 1000 Disable	R/W		0
5	Force Lane Mode	R/W	1 = Forces the lane mode to be derived from independent lane control registers (overrides autonegotiation). 0 = Normal operation	0
4	Force Speed Select	R/W	Forces the speed selected in bits [3:0] of this register.	0
3-0	Force Speed	R/W	0 = 2.5Gb 1 = 5Gb 2 = 6Gb 3 = 10G (HiG) 4 = 10G (CX4) 5 = 12G 6 = 12.5G 7 = 13G 8 = 15G 9 = 16G All others reserved	0

SerDes Digital Miscellaneous 2 Register (Offset: 0x09; Width 16)

Table 99: SerDes Digital Miscellaneous 2 Register (Offset: 0x09; Width 16)

Bit	Name	R/W	Description	Default
15	rxckpl_sel_combo			0
14	rxck_mii_gen_sel_force			0

Table 99: SerDes Digital Miscellaneous 2 Register (Offset: 0x09; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
13	rxck_mii_gel_sel_val			0
12	rlpbk_sw_force			0
11	rlpbk_RxRst_en			0
10	clkSigdet_bypass			0
9	clk41_bypass			0
8-6	Reserved			0
5	pma_pmd_forced_speed_enc_en			0
4	fifo_err_cya			0
3	an_txdisablePhase			0
2	an_rxSeqStartDis			0
1	an_txdisable_ln			0
0	an_deadTrap			0

SerDes Digital Pattern Generation Control Register (Offset: 0x0A; Width 16)

Table 100: SerDes Digital Pattern Generation Control Register (Offset: 0x0A; Width 16)

Bit	Name	R/W	Description	Default
15	Reserved	R/W		0
14	tx_err	R/W	1 = Set txer=1 during CRC portion of packet 0 = Normal operation	0
13	skip_crc	R/W	1 = Do not append 32 bit CRC to end of packet 0 = Normal operation	0
12	en_crc_checker_fragment_err_det	R/W	1 = Enable CRC checker to detect CRC errors on packets of any size (1 byte or more) 0 = Normal operation (CRC checker only detects CRC errors on packets of at least 72 bytes)	0
11-9	ipg_select	R/W	0 = Invalid 1 = IPG of 6 bytes 2 = IPG of 10 bytes 3 = IPG of 14 bytes 4 = IPG of 18 bytes 5 = IPG of 22 bytes 6 = IPG of 26 bytes 7 = IPG of 20 bytes	4
8-3	pkt_size	R/W	0 = Invalid 1 = 256 bytes 2 = 512 bytes 3 = 768 bytes 4 = 1024 bytes … 3Fh = 16,128 bytes	4
2	single_pass_mode	R/W	1 = Only send 1 packet and stop. 0 = Send packets while bit 1 of this register is set	0

Table 100: SerDes Digital Pattern Generation Control Register (Offset: 0x0A; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
1	run_pattern_gen	R/W	1 = A rising edge on this bit while the pattern generator is in the idle state will start sending packets. If the single_pass_mode bit is set then a single packet will be sent and the idle state will be entered, otherwise packets will be sent until this bit is cleared. At this point the current packet will finish transmitting and then ether the idle state. Note: A valid link must be established before sending packets. 0 = Do no send packets.	0
0	sel_pattern_gendata	R/W	1 = Send idles or pattern generator data into transmit FIFO (ignore MAC transmit data) 0 = Normal operation	0

SerDes Digital Pattern Generation Status Register (Offset: 0x0B; Width 16)

Table 101: SerDes Digital Pattern Generation Status Register (Offset: 0x0B; Width 16)

Bit	Name	R/W	Description	Default
15-4	Reserved	R/W		0
3	pattern_gen_active	RO	1 = Pattern generator is still sending packets 0 = Pattern generator is idle	0
2-0	pattern_gen_fsm	RO	0 = Idle 1 = Transmit preamble 2 = Transmit data 3 = Transmit SFD 4 = IPG 5 = IPG 2 (allows FSM to be grey-coded) 6 = Transmit CRC 7 = Reserved	0

SerDes Digital Test Mode Register (Offset: 0x0C; Width 16)

Table 102: SerDes Digital Test Mode Register (Offset: 0x0C; Width 16)

Bit	Name	R/W	Description	Default
15	disable_reset_cnt	R/W		0
14	clear_packet_counters	R/W		0
13-12	Reserved	R/W		0
11-6	test_monitor_mode2	R/W		0
5-0	test_monitor_mode1	R/W		0

SerDes Digital Transmit Packet Count Register (Offset: 0x0D; Width 16)

Table 103: SerDes Digital Transmit Packet Count Register (Offset: 0x0D; Width 16)

Bit	Name	R/W	Description	Default
15-0	txpktcnt	RO	Transmit packet counter status register (rolls over).	0

SerDes Digital Receive Packet Count Register (Offset: 0x0E; Width 16)

Table 104: SerDes Digital Receive Packet Count Register (Offset: 0x0E; Width 16)

Bit	Name	R/W	Description	Default
15-0	rxpktcnt	RO	Receive packet counter status register (rolls over).	0

Over 1G Block (Offset: 0x8320)

Over 1G Digital Control 30 Register (Offset: 0x0; Width 16)

Table 105: Over 1G Digital Control 30 Register (Offset: 0x0; Width 16)

Bit	Name	R/W	Description	Default
15-0	an_lostlink_cnt[15:0]	R/W	After losing link, the PHY must wait for this number of 25 MHz clock cycles before restarting autoneg.	0X7180

Over 1G Digital Control 31 Register (Offset: 0x1; Width 16)

Table 106: Over 1G Digital Control 31 Register (Offset: 0x1; Width 16)

Bit	Name	R/W	Description	Default
15-0	an_switch_cnt[15:0]	R/W	After entering the autoneg IDLE state, the PHY must wait this number of 25MHz clock cycles before commanding the PLL to switch.	0X7180

Over 1G Digital Control 32 Register (Offset: 0x2; Width 16)

Table 107: Over 1G Digital Control 32 Register (Offset: 0x2; Width 16)

Bit	Name	R/W	Description	Default
15-0	an_link_cnt[15:0]	R/W	After losing link, the PHY must continue looking for link for this number of 25 MHz clock cycles before restarting the autoneg process.	0X25A0

Over 1G Digital Control 33 Register (Offset: 0x3; Width 16)

Table 108: Over 1G Digital Control 33 Register (Offset: 0x3; Width 16)

Bit	Name	R/W	Description	Default
15-8	an_switch_cnt[23:16]	R/W	After entering the autoneg IDLE state, the PHY must wait this number of 25 MHz clock cycles before commanding the PLL to switch.	0X0B

Table 108: Over 1G Digital Control 33 Register (Offset: 0x3; Width 16)

Bit	Name	R/W	Description	Default
7:0	an_link_cnt[23:16]		After losing link, the PHY must continue looking for link for this number of 25 MHz clock cycles before restarting the autoneg process.	0X26

Over 1G Digital Control 34 Register (Offset: 0x04; Width 16)

Table 109: Over 1G Digital Control 34 Register (Offset: 0x04; Width 16)

Bit	Name	R/W	Description	Default
15:5	mp_number[10:0]	R/W	The message page (MP) ID for "Over 1Gb" next pages.	0x80
4	no_fail_cnt	R/W	1 = No limit to the number of 2.5G autoneg iterations 0 = The an_fail_cnt field specifies the number of autoneg iterations that must be performed before the higher speed advertisement is no longer advertised.	0
3:0	an_fail_cnt[3:0]		Specifies the number of autoneg iterations that must be performed before the higher speed advertisement is no longer advertised.	0x4

Over 1G Digital Control 35 Register (Offset: 0x05; Width 16)

Table 110: Over 1G Digital Control 35 Register (Offset: 0x05; Width 16)

Bit	Name	R/W	Description	Default
15:0	an_ignoreLink_cnt[15:0]	R/W	The PHY must ignore the link status for this number of 25 MHz clock cycles while concluding autonegotiation.	0x71B0

Over 1G Digital Control 36 Register (Offset: 0x06; Width 16)

Table 111: Over 1G Digital Control 36 Register (Offset: 0x06; Width 16)

Bit	Name	R/W	Description	Default
15:8	an_lostLink_cnt[23:16]	R/W	After losing link, the PHY must wait for this number of 25 MHz clock cycles before restarting autoneg.	0x0B
7:0	an_ignoreLink_cnt[23:16]		The PHY must ignore the link status for this number of 25 MHz clock cycles while concluding autonegotiation.	0x0B

Over 1G TPOUT 1 Register (Offset: 0x07; Width 16)

Table 112: Over 1G TPOUT 1 Register (Offset: 0x07; Width 16)

Bit	Name	R/W	Description	Default
15:0	tpout[15:0]	R/W	Select from register 0*1C.9:0	0

Over 1G TPOUT 2 Register (Offset: 0x08; Width 16)

Table 113: Over 1G TPOUT 2 Register (Offset: 0x08; Width 16)

Bit	Name	R/W	Description	Default
15:0	tpout[23:8]	R/W	Select from register 0*1C.9:0	0

Over 1G Unformatted Page 1 Register (Offset: 0x09; Width 16)

Table 114: Over 1G Unformatted Page 1 Register (Offset: 0x09; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-0	mr_adv_over_1g[10:0]	R/W	Over 1Gb abilities advertised (first next page). bit[0] = 2.5Gb bit[1] = 5Gb bit[2] = 6Gb bit[3] = 10Gb bit[4] = 10Gb CX4 bit[5] = 12Gb bit[6] = 12.5Gb bit[7] = 13Gb bit[8] = 15Gb bit[9] = 16Gb	1h

Over 1G Unformatted Page 2 Register (Offset: 0x0A; Width 16)

Table 115: Over 1G Unformatted Page 2 Register (Offset: 0x0A; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-0	mr_adv_over_1g[21:11]	R/W	General purpose [21:11] (second next page).	0

Over 1G Unformatted Page 3 Register (Offset: 0x0B; Width 16)

Table 116: Over 1G Unformatted Page 3 Register (Offset: 0x0B; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-0	mr_adv_over_1g[31:22]	R/W	General purpose [31:22] (third next page).	0

Over 1G Link Partner Unformatted Page 1 Register (Offset: 0x0C; Width 16)

Table 117: Over 1G Link Partner Unformatted Page 1 Register (Offset: 0x0C; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved			0

Table 117: Over 1G Link Partner Unformatted Page 1 Register (Offset: 0x0C; Width 16)

Bit	Name	R/W	Description	Default
10-0	lp_adv_over_1g[10:0]			

Over 1G LP_UP 2 Register Offset: 0x0D; Width 16)

Table 118: Over 1G LP_UP 2 Register Offset: 0x0D; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved			0
10-0	lp_adv_over_1g[21:11]			

Over 1G LP_UP 3 Register (Offset: 0x0E; Width 16)

Table 119: Over 1G LP_UP 3 Register (Offset: 0x0E; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved			0
10-0	lp_adv_over_1g[31:22]			

Remote PHY Block (Offset: 0x 8330)

Multirate Backplane Ethernet (MRBE) Block (Offset: 0x8350)

MRBE Message Page 5 Next Page Control Register (Offset: 0x0; Width 16)

Table 120: MRBE Message Page 5 Next Page Control Register (Offset: 0x0; Width 16)

Bit	Name	R/W	Description	Default
15:4	Reserved	R/W	Write as 0, ignore on read.	0
3	Next Page Software Override Enable	R/W	1 = Enable next page software override 0 = Normal operation	0
2	Unformatted Page 3 Enable	R/W	1 = Send unformatted pages 1, 2, and 3 0 = Send unformatted pages 1 and 2 only	0
1	T2 Mode	R/W	1 = Enable T2 mode 0 = Normal operation	0
0	MRBE Mode	R/W	1 = Enable MRBE mode 0 = Normal operation	0

MRBE Link Timer Offset 1 Register (Offset: 0x01; Width 16)

Table 121: MRBE Link Timer Offset 1 Register (Offset: 0x01; Width 16)

Bit	Name	R/W	Description	Default
15:8	sgmii_offset[7:0]	R/W		0x14
7:0	max_offset[7:0]	R/W		0x7E

MRBE Link Timer Offset 2 Register (Offset: 0x02; Width 16)

Table 122: MRBE Link Timer Offset 2 Register (Offset: 0x02; Width 16)

Bit	Name	R/W	Description	Default
15:8	sgmii_up_offset[7:0]	R/W		0x3D
7:0	link_down_offset[7:0]	R/W		0x3D

MRBE Link Timer Offset 3 Register (Offset: 0x03; Width 16)

Table 123: MRBE Link Timer Offset 3 Register (Offset: 0x03; Width 16)

Bit	Name	R/W	Description	Default
15:8	break_link_offset[7:0]	R/W		0x3D
7:0	np_link_offset[7:0]	R/W		0x3D

MRBE OUI MSB Field Register (Offset: 0x04; Width 16)

Table 124: MRBE OUI MSB Field Register (Offset: 0x04; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-0	oui[23:13]	R/W	OUI bits [23:13].	0

MRBE OUI LSB Field Register (Offset: 0x05; Width 16)

Table 125: MRBE OUI LSB Field Register (Offset: 0x05; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-0	oui[12:2]	R/W	OUI bits[12:2]	0x2BD

MRBE Field Register (Offset: 0x06; Width 16)

Table 126: MRBE Field Register (Offset: 0x06; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-9	oui[1:0]	R/W	OUT bits[1:0]	0x3
8-0	mrbe_field	R/W	0 = None, used to transfer OUI 1 = SerDes autonegotiation	0x0

MRBE UD Field Register (Offset: 0x07; Width 16)

Table 127: MRBE UD Field Register (Offset: 0x07; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-0	ud_field	R/W	0 = Over 1Gb 1 = Remote copper PHY 2 = Remote SerDes PHY 3 = Autoneg MDIO 4 = In-Band MDIO 5 = Remote BN page	0

MRBE Link Partner OUI MSB Field Register (Offset: 0x08; Width 16)

Table 128: MRBE Link Partner OUI MSB Field Register (Offset: 0x08; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-0	lp_oui[23:13]	R/W	LP OUI bits[23:13]	0

MRBE Link Partner OUI LSB Field Register (Offset: 0x09; Width 16)

Table 129: MRBE Link Partner OUI LSB Field Register (Offset: 0x09; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-0	lp_oui[12:2]	R/W	LP OUI bits[12:2]	0

MRBE Link Partner MRBE Field Register (Offset: 0x0A; Width 16)

Table 130: MRBE Link Partner MRBE Field Register (Offset: 0x0A; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0
10-9	lp_oui[1:0]	R/W	LP OUI bits[1:0]	0
8-0	lp_mrbe_field	R/W	0 = None, used to transfer OUI 1 = SerDes autonegotiation	0

MRBE Link Partner UD Field Register (Offset: 0x0B; Width 16)

Table 131: MRBE Link Partner UD Field Register (Offset: 0x0B; Width 16)

Bit	Name	R/W	Description	Default
15-11	Reserved	R/W	Write as 0, ignore on read.	0

Table 131: MRBE Link Partner UD Field Register (Offset: 0x0B; Width 16)

Bit	Name	R/W	Description	Default
10-0	lp_ud_field	R/W	0 = Over 1Gb 1 = Remote copper PHY 2 = Remote SerDes PHY 3 = Autoneg MDIO 4 = In-Band MDIO 5 = Remote BN page	0

Clause 73 User B0 (CL73_UserB0) Block (Offset: 0x 8370)

CL73_UserB0 Control 1 Register (Offset: 0x00; Width 16)

Table 132: CL73_UserB0 Control 1 Register (Offset: 0x00; Width 16)

Bit	Name	R/W	Description	Default
15-13	Reserved	R/W	Write as 0, ignore on read.	0
12	cl73_lossOfSyncFail_en	R/W	Enables exit from an AN_GOOD_CHECK state upon loss of sync_status.	0
11	cl73_parDet_dis	R/W	Disables CL73 parallel detect.	0
10	cl73_allowCl37AN	R/W	Allows CL73 auto-neg when link resolves to KX.	0
9	longParDetTimer_dis	R/W	Long parallel detect timer disable (use 50ms timer instead).	0
8	linkFailTimer_dis	R/W	Disable link fail inhibit timer enable.	0
7	linkFailTimerQual_en	R/W	Qualify link with link fail inhibit timer enable.	0
6	cl73_nonce_match_over	R/W	Clause 73 nonce match override.	0.
5	cl73_noncr_match_val	R/W	Clause 73 nonce match value.	0
4	couple_w_cl73_restart_wo_link_fail	R/W	Undocumented.	0
3	couple_w_cl73_restart	R/W	Undocumented.	0
2	couple_w_cl37_restart	R/W	Undocumented.	0
1	cl73_Ustat1_muxsel	R/W	Undocumented.	0
0	force_cl73_tx_omux_en	RW	Undocumented.	0

CL73_UserB0 Status 1 Register (Offset: 0x01; Width 16)

Table 133: CL73_UserB0 Status 1 Register (Offset: 0x01; Width 16)

Bit	Name	R/W	Description	Default
15:10	Reserved	RO	Write as 0, ignore on read.	0
9:0	Arbitration Finite State Machine	RO	Undocumented	0

CL73_UserB0 MRBE Control 1 Register (Offset: 0x02; Width 16)

Table 134: CL73_UserB0 MRBE Control 1 Register (Offset: 0x02; Width 16)

Bit	Name	R/W	Description	Default
15	Clause 73 MRBE Enable	R/W	1 = Enable MRBE autonegotiation 0 = Disable MRBE autonegotiation	0
14	Clause 73 MRBE Station Manager Enable	R/W	1 = Enable MRBE station manager 0 = Disable MRBE station manager	1
13	Clause 73 MRBE Next Page After Base Page Enable	R/W	1 = Send MRBE next pages immediately after the base page 0 = Send MRBE next pages following software next pages	1
12	Clause 73 MRBE Test Message Page 5 Halt Enable	R/W	1 = Enable MRBE MP5 test 0 = Disable MRBE MP5 test	0
11	Clause 73 MRBE Test Message Page 5 Halt Step	R/W	The MP5 halt step value.	0
10	Reserved	R/W	Write as 0, ignore on read.	0
9:0	User Defined Code Field [41:32]	R/W	MRBE user defined code field [41:32].	0

CL73_UserB0 MRBE Control 2 Register (Offset: 0x03; Width 16)

Table 135: CL73_UserB0 MRBE Control 2 Register (Offset: 0x03; Width 16)

Bit	Name	R/W	Description	Default
15:0	User Defined Code Field [31:16]	R/W	MRBE user defined code field [31:16].	0

CL73_UserB0 MRBE Control 3 Register (Offset: 0x04; Width 16)

Table 136: CL73_UserB0 MRBE Control 3 Register (Offset: 0x04; Width 16)

Bit	Name	R/W	Description	Default
15:0	User Defined Code Field [15:0]	R/W	MRBE user defined code field [15:0].	0

CL73_UserB0 MRBE Status 1 Register (Offset: 0x05; Width 16)

Table 137: CL73_UserB0 MRBE Status 1 Register (Offset: 0x05; Width 16)

Bit	Name	R/W	Description	Default
15:10	Reserved	RO	Ignore on read.	0
9:0	Link Partner User Defined Code Field [41:32]	RO	The link partner's user defined code field [41:32].	0

CL73_UserB0 MRBE Status 2 Register (Offset: 0x06; Width 16)

Table 138: CL73_UserB0 MRBE Status 2 Register (Offset: 0x06; Width 16)

Bit	Name	R/W	Description	Default
15:0	Link Partner User Defined Code Field [31:16]	RO	The link partner's user defined code field [31:16].	0

CL73_UserB0 MRBE Status 3 Register (Offset: 0x07; Width 16)

Table 139: CL73_UserB0 MRBE Status 3 Register (Offset: 0x07; Width 16)

Bit	Name	R/W	Description	Default
15:0	Link Partner User Defined Code Field [15:0]	RO	The link partner's user defined code field [15:0].	0

Address Extension Register (AER) Block (Offset: 0xFFD0)

AER Address Extension Register (Offset: 0x0E; Width 16)

Table 140: AER Address Extension Register (Offset: 0x0E; Width 16)

Bit	Name	R/W	Description	Default
15-0	Address Expansion Register	R/W	Selects an embedded/companion MMD. When not in independent channel mode (single PHY mode) there are 4 MMD's. The top level XGXS MMD is identified by the DEVAD_STRAP. The PMA/PMD device is identified by DEVAD=1. The Clause 73 Autonegotiation device is identified by DEVAD=7. The Clause 22 Combo core/SGMII device is identified with a Clause 22 MDIO access. When configured as four independent channels each channel has 3 devices. The PMA/PMD devices is identified by DEVAD=1. The Clause 73 Autonegotiation device is identified by DEVAD=7. The Clause 22 Combo core/SGMII device is identified with a Clause 22 MDIO access. The MDI registers for all MMDs can be accessed through Clause 22 PHY register space by setting the AER field to the appropriate value. The top level XGXS registers can be accessed by setting Address Expansion Register [15:11] to the DEVAD_STRAP value. 0x0000 = DEVAD/ComboCore 0x0800 = PMA/PMD 0x3800 = Autonegotiation MMD (7h) All other values are reserved.	0

IEEE Combination Block (Offset 0xFFE0)

IEEE_Combo MII Control Register (Offset: 0x0; Width 16)

Table 141: IEEE_Combo MII Control Register (Offset: 0x0; Width 16)

Bit	Name	R/W	Description	Default
15	Reset	R/W, SC	1 = PHY Reset 0 = Normal operation	0
14	Internal Loopback	R/W	1 = Global loopback enabled 0 = Normal operation	0
13	Speed Selection (LSB)	R/W	See Speed Selection (MSB)	0
12	Autonegotiation Enable	R/W	1 = Autonegotiation enabled 0 = Autonegotiation disabled	1
11	Power Down	R/W	1 = Low Power Mode 0 = Normal operation	0
10	Reserved	RO	Ignore on read.	0
9	Restart Autonegotiation	R/W, SC	1 = Restart autonegotiation 0 = Normal operation	0
8	Manual Duplex Mode	R/W	1 = Full duplex 0 = Half duplex	1
7	Collision Test Enabled	R/W	1 = Collision test mode enabled 0 = Collision test mode disabled	0
6	Speed Selection (MSB)	R/W	11 = Reserved 10 = SGMII 1000Mb/s 01 = SGMII 100Mb/s 00 = SGMII 10Mb/s	0
5-0	Reserved	RO	Ignore on read.	0

IEEE0 MII Status Register (Offset: 0x01; Width 16)

Table 142: IEEE0 MII Status Register (Offset: 0x01; Width 16)

Bit	Name	R/W	Description	Default
15	100Base-T4 Capable	RO, L	1 = 100Base-T4 capable 0 = Not 100Base-T4 capable	0
14	100Base-X Full-Duplex Capable	RO, L	1 = 100Base-X full-duplex capable 0 = Not 100Base-X full-duplex capable	0
13	100Base-X Half-Duplex Capable	RO, L	1 = 100Base-X half-duplex capable 0 = Not 100Base-X half-duplex capable	0
12	10Base-T Full-Duplex Capable	RO, L	1 = 10Base-T full-duplex capable 0 = Not 10Base-T full-duplex capable	0
11	10Base-T Half-Duplex Capable	RO, L	1 = 10Base-T half-duplex capable 0 = Not 10Base-T half-duplex capable	0
10	100Base-T2 Full-Duplex Capable	RO, L	1 = 100Base-T2 full-duplex capable 0 = Not 100Base-T2 full-duplex capable	0

Table 142: IEEE0 MII Status Register (Offset: 0x01; Width 16) (Cont.)

Bit	Name	R/W	Description	Default
9	100Base-T2 Half-Duplex Capable	RO, L	1 = 100Base-T2 half-duplex capable 0 = Not 100Base-T2 half-duplex capable	0
8	Extended Status	RO, H	1 = Extended status information in register 0Fh	1
7	Reserved	RO	Ignore on read.	0
6	Management Frames Preamble Suppression	RO, H	1 = PHY will accept management frames with preamble suppressed 0 = PHY will not accept management frames with preamble suppressed	1
5	Autonegotiation Complete	RO	1 = Autonegotiation complete 0 = Autonegotiation in progress	0
4	Remote Fault	RO, LH	1 = Remote fault detected 0 = No remote fault detected	0
3	Autonegotiation Ability	RO, H	1 = Autonegotiation capable 0 = Not autonegotiation capable	1
2	Link Status	RO, LL	1 = Link is up 0 = Link is down	0
1	Jabber Detect	RO, L	1 = Jabber condition detected 0 = No jabber condition detected	0
0	Extended Capability	RO, H	1 = Extended register capabilities supported 0 = Basic register set capabilities only	1

IEEE0 PHY Identifier MSB Register (Offset: 0x02; Width 16)

Table 143: IEEE0 PHY Identifier MSB Register (Offset: 0x02; Width 16)

Bit	Name	R/W	Description	Default
15-0	OUI	RO	Bits 18:3 of the organizationally unique identifier.	0143h

IEEE0 PHY Identifier LSB Register (Offset: 0x03; Width 16)

Table 144: IEEE0 PHY Identifier LSB Register (Offset: 0x03; Width 16)

Bit	Name	R/W	Description	Default
15:10	OUI	RO	Bits 24:19 of the organizationally unique identifier.	2Fh
9:4	Model	RO	Device model number	3Fh
3:0	Revision	RO	Device revision number	0h

IEEE0 Autonegotiation Advertisement Register (Offset: 0x04; Width 16)

Table 145: IEEE0 Autonegotiation Advertisement Register (Offset: 0x04; Width 16)

Bit	Name	R/W	Description	Default
15	Next Page	R/W	1 = Next page ability supported 0 = Next page ability not supported	0
14	Reserved	RO	Ignore on read	0
13:12	Remote Fault	R/W	01 = Link failure 00 = Advertise no remote fault	0
11:9	Reserved	R/W	Write as 0, ignore on read	0
8:7	Pause	R/W	01 = Advertise symmetric pause capable 00 = Advertise not pause capable	1
6	Half-Duplex	R/W	1 = Advertise half-duplex capable 0 = Advertise not half-duplex capable	1
5	Full-Duplex	R/W	1 = Advertise full-duplex capable 0 = Advertise not full-duplex capable	1
4:0	Reserved	R/W	Write as 0, ignore on read	0

IEEE0 Autonegotiation Link Partner Ability Register (Offset: 0x05; Width 16)

Table 146: IEEE0 Autonegotiation Link Partner Ability Register (Offset: 0x05; Width 16)

Bit	Name	R/W	Description	Default
15	Next Page	RO	1 = Link partner has Next Page ability. 0 = Link partner does not have Next Page ability.	0
14	Acknowledge	RO	1 = Link partner has received link code word. 0 = Link partner has not received link code word.	0
13-12	Remote Fault	RO	01 = Link partner has detected remote fault. 00 = Link partner has not detected remote fault.	0
11-9	Reserved	RO	Ignore on read.	0
8-7	Pause Capable	RO	01 = Link partner is symmetric pause capable. 00 = Link partner is not pause capable.	0
6	Half Duplex Capable	RO	1 = Link partner is half duplex capable 0 = Link partner is not half duplex capable	0
5	Full Duplex Capable	RO	1 = Link partner is full duplex capable 0 = Link partner is not full duplex capable	0
4-1	Reserved	RO	Ignore on read.	0
0	SGMII Mode	RO	1 = SGMII Mode 0 = Fiber mode	0

IEEE0 Autonegotiation Expansion Register (Offset: 0x06; Width 16)

Table 147: IEEE0 Autonegotiation Expansion Register (Offset: 0x06; Width 16)

Bit	Name	R/W	Description	Default
15-3	Reserved	RO	Ignore on read.	0

Table 147: IEEE0 Autonegotiation Expansion Register (Offset: 0x06; Width 16)

Bit	Name	R/W	Description	Default
2	Next Page Ability	RO, L	1 = Local device is next page capable. 0 = Local device is not next page capable.	0
1	Page Received	RO, LH	1 = New link code word has been received 0 = New link code word has not bee received.	0
0	Reserved	RO	Ignore on read.	0

IEEE0 Autonegotiation Next Page Register (Offset: 0x07; Width 16)

Table 148: IEEE0 Autonegotiation Next Page Register (Offset: 0x07; Width 16)

Bit	Name	R/W	Description	Default
15	Next Page	R/W	0 = Last page 1 = Additional next page(s) follow	0
14	Acknowledge	R/W	Acknowledge	0
13	Message Page	R/W	0 = Unformatted page 1 = Message Page	0
12	Acknowledge 2	R/W	Acknowledge	0
11	Toggle	R/W	Opposite value of bit in the previous page.	0
10-0	Message or Unformatted Code Field	R/W	400h = Over 1G message page 410h = Remote copper PHY message page 411h = MDIO register write message page 412h = MDIO register read request message page 413h = MDIO register response message page	0

IEEE0 Autonegotiation Link Partner Next Page Register (Offset: 0x08; Width 16)

Table 149: IEEE0 Autonegotiation Link Partner Next Page Register (Offset: 0x08; Width 16)

Bit	Name	R/W	Description	Default
15	Next Page	R/W	0 = Last page 1 = Additional next page(s) follow	0
14	Acknowledge	R/W		0
13	Message Page	R/W	0 = Unformatted page 1 = Message Page	0
12	Acknowledge 2	R/W		0
11	Toggle	R/W	Opposite value of bit in the previous page.	0
10-0	Message or Unformatted Code Field	R/W	400h = Over 1G message page 410h = Remote copper PHY message page 411h = MDIO register write message page 412h = MDIO register read request message page 413h = MDIO register response message page	0

Appendix A: bxe_hsi.h

```
/*-
 * Copyright (c) 2007-2008 Broadcom Corporation
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Neither the name of Broadcom Corporation nor the name of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written consent.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS'
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS
 * BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */


struct license_key {
   uint32_t reserved[6];

#if defined(__BIG_ENDIAN)
   uint16_t max_iscsi_init_conn;
   uint16_t max_iscsi_trgt_conn;
#elif defined(__LITTLE_ENDIAN)
   uint16_t max_iscsi_trgt_conn;
   uint16_t max_iscsi_init_conn;
#endif

   uint32_t reserved_a[6];
};


#define PORT_00
#define PORT_11
#define PORT_MAX2

/****************************************************************************
 * Shared HW configuration    *
```

```
 ***************************************************************************/
struct shared_hw_cfg { /* NVRAM Offset */
   /* Up to 16 bytes of NULL-terminated string */
   uint8_t  part_num[16];/* 0x104 */

   uint32_t config;    /* 0x114 */
#define SHARED_HW_CFG_MDIO_VOLTAGE_MASK     0x00000001
#define SHARED_HW_CFG_MDIO_VOLTAGE_SHIFT    0
#define SHARED_HW_CFG_MDIO_VOLTAGE_1_2V     0x00000000
#define SHARED_HW_CFG_MDIO_VOLTAGE_2_5V     0x00000001
#define SHARED_HW_CFG_MCP_RST_ON_CORE_RST_EN    0x00000002

#define SHARED_HW_CFG_PORT_SWAP     0x00000004

#define SHARED_HW_CFG_BEACON_WOL_EN    0x00000008

#define SHARED_HW_CFG_MFW_SELECT_MASK    0x00000700
#define SHARED_HW_CFG_MFW_SELECT_SHIFT    8
   /* Whatever MFW found in NVM
      (if multiple found, priority order is: NC-SI, UMP, IPMI) */
#define SHARED_HW_CFG_MFW_SELECT_DEFAULT    0x00000000
#define SHARED_HW_CFG_MFW_SELECT_NC_SI    0x00000100
#define SHARED_HW_CFG_MFW_SELECT_UMP    0x00000200
#define SHARED_HW_CFG_MFW_SELECT_IPMI    0x00000300
   /* Use SPIO4 as an arbiter between: 0-NC_SI, 1-IPMI
     (can only be used when an add-in board, not BMC, pulls-down SPIO4) */
#define SHARED_HW_CFG_MFW_SELECT_SPIO4_NC_SI_IPMI   0x00000400
   /* Use SPIO4 as an arbiter between: 0-UMP, 1-IPMI
     (can only be used when an add-in board, not BMC, pulls-down SPIO4) */
#define SHARED_HW_CFG_MFW_SELECT_SPIO4_UMP_IPMI     0x00000500
   /* Use SPIO4 as an arbiter between: 0-NC-SI, 1-UMP
     (can only be used when an add-in board, not BMC, pulls-down SPIO4) */
#define SHARED_HW_CFG_MFW_SELECT_SPIO4_NC_SI_UMP    0x00000600

#define SHARED_HW_CFG_LED_MODE_MASK    0x000f0000
#define SHARED_HW_CFG_LED_MODE_SHIFT    16
#define SHARED_HW_CFG_LED_MAC1    0x00000000
#define SHARED_HW_CFG_LED_PHY1    0x00010000
#define SHARED_HW_CFG_LED_PHY2    0x00020000
#define SHARED_HW_CFG_LED_PHY3    0x00030000
#define SHARED_HW_CFG_LED_MAC2    0x00040000
#define SHARED_HW_CFG_LED_PHY4    0x00050000
#define SHARED_HW_CFG_LED_PHY5    0x00060000
#define SHARED_HW_CFG_LED_PHY6    0x00070000
#define SHARED_HW_CFG_LED_MAC3    0x00080000
#define SHARED_HW_CFG_LED_PHY7    0x00090000
#define SHARED_HW_CFG_LED_PHY9    0x000a0000
#define SHARED_HW_CFG_LED_PHY11     0x000b0000
#define SHARED_HW_CFG_LED_MAC4    0x000c0000
#define SHARED_HW_CFG_LED_PHY8    0x000d0000

#define SHARED_HW_CFG_AN_ENABLE_MASK    0x3f000000
#define SHARED_HW_CFG_AN_ENABLE_SHIFT    24
#define SHARED_HW_CFG_AN_ENABLE_CL37    0x01000000
#define SHARED_HW_CFG_AN_ENABLE_CL73    0x02000000
#define SHARED_HW_CFG_AN_ENABLE_BAM    0x04000000
```

```
#define SHARED_HW_CFG_AN_ENABLE_PARALLEL_DETECTION  0x08000000
#define SHARED_HW_CFG_AN_EN_SGMII_FIBER_AUTO_DETECT 0x10000000
#define SHARED_HW_CFG_AN_ENABLE_REMOTE_PHY    0x20000000

   uint32_t config2;   /* 0x118 */
   /* one time auto detect grace period (in sec) */
#define SHARED_HW_CFG_GRACE_PERIOD_MASK     0x000000ff
#define SHARED_HW_CFG_GRACE_PERIOD_SHIFT    0

#define SHARED_HW_CFG_PCIE_GEN2_ENABLED     0x00000100

   /* The default value for the core clock is 250MHz and it is
      achieved by setting the clock change to 4 */
#define SHARED_HW_CFG_CLOCK_CHANGE_MASK     0x00000e00
#define SHARED_HW_CFG_CLOCK_CHANGE_SHIFT    9

#define SHARED_HW_CFG_SMBUS_TIMING_100KHZ    0x00000000
#define SHARED_HW_CFG_SMBUS_TIMING_400KHZ    0x00001000

#define SHARED_HW_CFG_HIDE_PORT1    0x00002000

#define SHARED_HW_CFG_WOL_CAPABLE_DISABLED    0x00000000
#define SHARED_HW_CFG_WOL_CAPABLE_ENABLED    0x00004000

   /*  Output low when PERST is asserted */
#define SHARED_HW_CFG_SPIO4_FOLLOW_PERST_DISABLED      0x00000000
#define SHARED_HW_CFG_SPIO4_FOLLOW_PERST_ENABLED      0x00008000

   uint32_t power_dissipated;/* 0x11c */
#define SHARED_HW_CFG_POWER_DIS_CMN_MASK    0xff000000
#define SHARED_HW_CFG_POWER_DIS_CMN_SHIFT    24

#define SHARED_HW_CFG_POWER_MGNT_SCALE_MASK    0x00ff0000
#define SHARED_HW_CFG_POWER_MGNT_SCALE_SHIFT    16
#define SHARED_HW_CFG_POWER_MGNT_UNKNOWN_SCALE    0x00000000
#define SHARED_HW_CFG_POWER_MGNT_DOT_1_WATT    0x00010000
#define SHARED_HW_CFG_POWER_MGNT_DOT_01_WATT    0x00020000
#define SHARED_HW_CFG_POWER_MGNT_DOT_001_WATT    0x00030000

   uint32_t ump_nc_si_config;/* 0x120 */
#define SHARED_HW_CFG_UMP_NC_SI_MII_MODE_MASK    0x00000003
#define SHARED_HW_CFG_UMP_NC_SI_MII_MODE_SHIFT    0
#define SHARED_HW_CFG_UMP_NC_SI_MII_MODE_MAC    0x00000000
#define SHARED_HW_CFG_UMP_NC_SI_MII_MODE_PHY    0x00000001
#define SHARED_HW_CFG_UMP_NC_SI_MII_MODE_MII    0x00000000
#define SHARED_HW_CFG_UMP_NC_SI_MII_MODE_RMII    0x00000002

#define SHARED_HW_CFG_UMP_NC_SI_NUM_DEVS_MASK    0x00000f00
#define SHARED_HW_CFG_UMP_NC_SI_NUM_DEVS_SHIFT    8

#define SHARED_HW_CFG_UMP_NC_SI_EXT_PHY_TYPE_MASK   0x00ff0000
#define SHARED_HW_CFG_UMP_NC_SI_EXT_PHY_TYPE_SHIFT  16
#define SHARED_HW_CFG_UMP_NC_SI_EXT_PHY_TYPE_NONE   0x00000000
#define SHARED_HW_CFG_UMP_NC_SI_EXT_PHY_TYPE_BCM5221 0x00010000

   uint32_t board;     /* 0x124 */
```

```
#define SHARED_HW_CFG_BOARD_REV_MASK    0x00FF0000
#define SHARED_HW_CFG_BOARD_REV_SHIFT    16

#define SHARED_HW_CFG_BOARD_MAJOR_VER_MASK    0x0F000000
#define SHARED_HW_CFG_BOARD_MAJOR_VER_SHIFT    24

#define SHARED_HW_CFG_BOARD_MINOR_VER_MASK    0xF0000000
#define SHARED_HW_CFG_BOARD_MINOR_VER_SHIFT    28

   uint32_t reserved;  /* 0x128 */

};


/******************************************************************************
 * Port HW configuration    *
 ******************************************************************************/
struct port_hw_cfg {    /* port 0: 0x12c  port 1: 0x2bc */

   uint32_t pci_id;
#define PORT_HW_CFG_PCI_VENDOR_ID_MASK    0xffff0000
#define PORT_HW_CFG_PCI_DEVICE_ID_MASK    0x0000ffff

   uint32_t pci_sub_id;
#define PORT_HW_CFG_PCI_SUBSYS_DEVICE_ID_MASK    0xffff0000
#define PORT_HW_CFG_PCI_SUBSYS_VENDOR_ID_MASK    0x0000ffff

   uint32_t power_dissipated;
#define PORT_HW_CFG_POWER_DIS_D3_MASK    0xff000000
#define PORT_HW_CFG_POWER_DIS_D3_SHIFT    24
#define PORT_HW_CFG_POWER_DIS_D2_MASK    0x00ff0000
#define PORT_HW_CFG_POWER_DIS_D2_SHIFT    16
#define PORT_HW_CFG_POWER_DIS_D1_MASK    0x0000ff00
#define PORT_HW_CFG_POWER_DIS_D1_SHIFT    8
#define PORT_HW_CFG_POWER_DIS_D0_MASK    0x000000ff
#define PORT_HW_CFG_POWER_DIS_D0_SHIFT    0

   uint32_t power_consumed;
#define PORT_HW_CFG_POWER_CONS_D3_MASK    0xff000000
#define PORT_HW_CFG_POWER_CONS_D3_SHIFT     24
#define PORT_HW_CFG_POWER_CONS_D2_MASK    0x00ff0000
#define PORT_HW_CFG_POWER_CONS_D2_SHIFT     16
#define PORT_HW_CFG_POWER_CONS_D1_MASK    0x0000ff00
#define PORT_HW_CFG_POWER_CONS_D1_SHIFT     8
#define PORT_HW_CFG_POWER_CONS_D0_MASK    0x000000ff
#define PORT_HW_CFG_POWER_CONS_D0_SHIFT     0

   uint32_t mac_upper;
#define PORT_HW_CFG_UPPERMAC_MASK    0x0000ffff
#define PORT_HW_CFG_UPPERMAC_SHIFT    0
   uint32_t mac_lower;

   uint32_t iscsi_mac_upper;  /* Upper 16 bits are always zeroes */
   uint32_t iscsi_mac_lower;

   uint32_t rdma_mac_upper;   /* Upper 16 bits are always zeroes */
```

```
   uint32_t rdma_mac_lower;

   uint32_t serdes_config;
#define PORT_HW_CFG_SERDES_TX_DRV_PRE_EMPHASIS_MASK      0x0000FFFF
#define PORT_HW_CFG_SERDES_TX_DRV_PRE_EMPHASIS_SHIFT      0

#define PORT_HW_CFG_SERDES_RX_DRV_EQUALIZER_MASK      0xFFFF0000
#define PORT_HW_CFG_SERDES_RX_DRV_EQUALIZER_SHIFT      16


   uint32_t Reserved0[16];    /* 0x158 */

   /*  for external PHY, or forced mode or during AN */
   uint16_t xgxs_config_rx[4];    /* 0x198 */

   uint16_t xgxs_config_tx[4];    /* 0x1A0 */

   uint32_t Reserved1[64];    /* 0x1A8 */

   uint32_t lane_config;
#define PORT_HW_CFG_LANE_SWAP_CFG_MASK    0x0000ffff
#define PORT_HW_CFG_LANE_SWAP_CFG_SHIFT     0
#define PORT_HW_CFG_LANE_SWAP_CFG_TX_MASK    0x000000ff
#define PORT_HW_CFG_LANE_SWAP_CFG_TX_SHIFT    0
#define PORT_HW_CFG_LANE_SWAP_CFG_RX_MASK    0x0000ff00
#define PORT_HW_CFG_LANE_SWAP_CFG_RX_SHIFT    8
#define PORT_HW_CFG_LANE_SWAP_CFG_MASTER_MASK    0x0000c000
#define PORT_HW_CFG_LANE_SWAP_CFG_MASTER_SHIFT    14
   /* AN and forced */
#define PORT_HW_CFG_LANE_SWAP_CFG_01230123    0x00001b1b
   /* forced only */
#define PORT_HW_CFG_LANE_SWAP_CFG_01233210    0x00001be4
   /* forced only */
#define PORT_HW_CFG_LANE_SWAP_CFG_31203120    0x0000d8d8
   /* forced only */
#define PORT_HW_CFG_LANE_SWAP_CFG_32103210    0x0000e4e4

   uint32_t external_phy_config;
#define PORT_HW_CFG_SERDES_EXT_PHY_TYPE_MASK    0xff000000
#define PORT_HW_CFG_SERDES_EXT_PHY_TYPE_SHIFT    24
#define PORT_HW_CFG_SERDES_EXT_PHY_TYPE_DIRECT    0x00000000
#define PORT_HW_CFG_SERDES_EXT_PHY_TYPE_BCM5482     0x01000000
#define PORT_HW_CFG_SERDES_EXT_PHY_TYPE_NOT_CONN    0xff000000

#define PORT_HW_CFG_SERDES_EXT_PHY_ADDR_MASK    0x00ff0000
#define PORT_HW_CFG_SERDES_EXT_PHY_ADDR_SHIFT    16

#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_MASK    0x0000ff00
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_SHIFT    8
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_DIRECT    0x00000000
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_BCM8071    0x00000100
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_BCM8072    0x00000200
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_BCM8073    0x00000300
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_BCM8705    0x00000400
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_BCM8706    0x00000500
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_BCM8726    0x00000600
```

```
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_BCM8481    0x00000700
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_SFX7101    0x00000800
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_FAILURE    0x0000fd00
#define PORT_HW_CFG_XGXS_EXT_PHY_TYPE_NOT_CONN    0x0000ff00

#define PORT_HW_CFG_XGXS_EXT_PHY_ADDR_MASK    0x000000ff
#define PORT_HW_CFG_XGXS_EXT_PHY_ADDR_SHIFT    0

   uint32_t speed_capability_mask;
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_MASK    0xffff0000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_SHIFT    16
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_10M_FULL    0x00010000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_10M_HALF    0x00020000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_100M_HALF   0x00040000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_100M_FULL   0x00080000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_1G    0x00100000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_2_5G    0x00200000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_10G    0x00400000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_12G    0x00800000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_12_5G    0x01000000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_13G    0x02000000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_15G    0x04000000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_16G    0x08000000
#define PORT_HW_CFG_SPEED_CAPABILITY_D0_RESERVED    0xf0000000

#define PORT_HW_CFG_SPEED_CAPABILITY_D3_MASK    0x0000ffff
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_SHIFT    0
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_10M_FULL    0x00000001
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_10M_HALF    0x00000002
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_100M_HALF   0x00000004
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_100M_FULL   0x00000008
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_1G    0x00000010
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_2_5G    0x00000020
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_10G    0x00000040
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_12G    0x00000080
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_12_5G    0x00000100
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_13G    0x00000200
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_15G    0x00000400
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_16G    0x00000800
#define PORT_HW_CFG_SPEED_CAPABILITY_D3_RESERVED    0x0000f000

   /*  A place to hold the original MAC address as a backup */
   uint32_t backup_mac_upper;    /* 0x2B4 */

   uint32_t backup_mac_lower;    /* 0x2B8 */
};


/******************************************************************************
 * Shared Feature configuration     *
 ******************************************************************************/
struct shared_feat_cfg { /* NVRAM Offset */

   uint32_t config;    /* 0x450 */
#define SHARED_FEATURE_BMC_ECHO_MODE_EN     0x00000001
```

```
   /*  Use the values from options 47 and 48 instead of the HW default
     values */
#define SHARED_FEAT_CFG_OVERRIDE_PREEMPHASIS_CFG_DISABLED     0x00000000
#define SHARED_FEAT_CFG_OVERRIDE_PREEMPHASIS_CFG_ENABLED      0x00000002

#define SHARED_FEAT_CFG_NCSI_ID_METHOD_SPIO      0x00000000
#define SHARED_FEAT_CFG_NCSI_ID_METHOD_NVRAM      0x00000008

#define SHARED_FEAT_CFG_NCSI_ID_MASK      0x00000030
#define SHARED_FEAT_CFG_NCSI_ID_SHIFT      4
   /*  Override the OTP back to single function mode */
#define SHARED_FEATURE_MF_MODE_DISABLED     0x00000100

};


/*****************************************************************************
 * Port Feature configuration    *
 *****************************************************************************/
struct port_feat_cfg {    /* port 0: 0x454  port 1: 0x4c8 */

   uint32_t config;
#define PORT_FEATURE_BAR1_SIZE_MASK    0x0000000f
#define PORT_FEATURE_BAR1_SIZE_SHIFT    0
#define PORT_FEATURE_BAR1_SIZE_DISABLED     0x00000000
#define PORT_FEATURE_BAR1_SIZE_64K    0x00000001
#define PORT_FEATURE_BAR1_SIZE_128K    0x00000002
#define PORT_FEATURE_BAR1_SIZE_256K    0x00000003
#define PORT_FEATURE_BAR1_SIZE_512K    0x00000004
#define PORT_FEATURE_BAR1_SIZE_1M    0x00000005
#define PORT_FEATURE_BAR1_SIZE_2M    0x00000006
#define PORT_FEATURE_BAR1_SIZE_4M    0x00000007
#define PORT_FEATURE_BAR1_SIZE_8M    0x00000008
#define PORT_FEATURE_BAR1_SIZE_16M    0x00000009
#define PORT_FEATURE_BAR1_SIZE_32M    0x0000000a
#define PORT_FEATURE_BAR1_SIZE_64M    0x0000000b
#define PORT_FEATURE_BAR1_SIZE_128M    0x0000000c
#define PORT_FEATURE_BAR1_SIZE_256M    0x0000000d
#define PORT_FEATURE_BAR1_SIZE_512M    0x0000000e
#define PORT_FEATURE_BAR1_SIZE_1G    0x0000000f
#define PORT_FEATURE_BAR2_SIZE_MASK    0x000000f0
#define PORT_FEATURE_BAR2_SIZE_SHIFT    4
#define PORT_FEATURE_BAR2_SIZE_DISABLED     0x00000000
#define PORT_FEATURE_BAR2_SIZE_64K    0x00000010
#define PORT_FEATURE_BAR2_SIZE_128K    0x00000020
#define PORT_FEATURE_BAR2_SIZE_256K    0x00000030
#define PORT_FEATURE_BAR2_SIZE_512K    0x00000040
#define PORT_FEATURE_BAR2_SIZE_1M    0x00000050
#define PORT_FEATURE_BAR2_SIZE_2M    0x00000060
#define PORT_FEATURE_BAR2_SIZE_4M    0x00000070
#define PORT_FEATURE_BAR2_SIZE_8M    0x00000080
#define PORT_FEATURE_BAR2_SIZE_16M    0x00000090
#define PORT_FEATURE_BAR2_SIZE_32M    0x000000a0
#define PORT_FEATURE_BAR2_SIZE_64M    0x000000b0
#define PORT_FEATURE_BAR2_SIZE_128M    0x000000c0
#define PORT_FEATURE_BAR2_SIZE_256M    0x000000d0
```

```
#define PORT_FEATURE_BAR2_SIZE_512M    0x000000e0
#define PORT_FEATURE_BAR2_SIZE_1G    0x000000f0
#define PORT_FEATURE_EN_SIZE_MASK    0x07000000
#define PORT_FEATURE_EN_SIZE_SHIFT    24
#define PORT_FEATURE_WOL_ENABLED    0x01000000
#define PORT_FEATURE_MBA_ENABLED    0x02000000
#define PORT_FEATURE_MFW_ENABLED    0x04000000

   uint32_t wol_config;
   /* Default is used when driver sets to "auto" mode */
#define PORT_FEATURE_WOL_DEFAULT_MASK    0x00000003
#define PORT_FEATURE_WOL_DEFAULT_SHIFT    0
#define PORT_FEATURE_WOL_DEFAULT_DISABLE    0x00000000
#define PORT_FEATURE_WOL_DEFAULT_MAGIC    0x00000001
#define PORT_FEATURE_WOL_DEFAULT_ACPI    0x00000002
#define PORT_FEATURE_WOL_DEFAULT_MAGIC_AND_ACPI     0x00000003
#define PORT_FEATURE_WOL_RES_PAUSE_CAP    0x00000004
#define PORT_FEATURE_WOL_RES_ASYM_PAUSE_CAP    0x00000008
#define PORT_FEATURE_WOL_ACPI_UPON_MGMT     0x00000010

   uint32_t mba_config;
#define PORT_FEATURE_MBA_BOOT_AGENT_TYPE_MASK    0x00000007
#define PORT_FEATURE_MBA_BOOT_AGENT_TYPE_SHIFT    0
#define PORT_FEATURE_MBA_BOOT_AGENT_TYPE_PXE    0x00000000
#define PORT_FEATURE_MBA_BOOT_AGENT_TYPE_RPL    0x00000001
#define PORT_FEATURE_MBA_BOOT_AGENT_TYPE_BOOTP    0x00000002
#define PORT_FEATURE_MBA_BOOT_AGENT_TYPE_ISCSIB     0x00000003
#define PORT_FEATURE_MBA_BOOT_AGENT_TYPE_NONE0x00000007
#define PORT_FEATURE_MBA_RES_PAUSE_CAP    0x00000100
#define PORT_FEATURE_MBA_RES_ASYM_PAUSE_CAP    0x00000200
#define PORT_FEATURE_MBA_SETUP_PROMPT_ENABLE    0x00000400
#define PORT_FEATURE_MBA_HOTKEY_CTRL_S    0x00000000
#define PORT_FEATURE_MBA_HOTKEY_CTRL_B    0x00000800
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_MASK    0x000ff000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_SHIFT    12
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_DISABLED    0x00000000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_2K    0x00001000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_4K    0x00002000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_8K    0x00003000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_16K    0x00004000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_32K    0x00005000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_64K    0x00006000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_128K    0x00007000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_256K    0x00008000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_512K    0x00009000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_1M    0x0000a000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_2M    0x0000b000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_4M    0x0000c000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_8M    0x0000d000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_16M    0x0000e000
#define PORT_FEATURE_MBA_EXP_ROM_SIZE_32M    0x0000f000
#define PORT_FEATURE_MBA_MSG_TIMEOUT_MASK    0x00f00000
#define PORT_FEATURE_MBA_MSG_TIMEOUT_SHIFT    20
#define PORT_FEATURE_MBA_BIOS_BOOTSTRAP_MASK    0x03000000
#define PORT_FEATURE_MBA_BIOS_BOOTSTRAP_SHIFT    24
#define PORT_FEATURE_MBA_BIOS_BOOTSTRAP_AUTO    0x00000000
```

```
#define PORT_FEATURE_MBA_BIOS_BOOTSTRAP_BBS    0x01000000
#define PORT_FEATURE_MBA_BIOS_BOOTSTRAP_INT18H    0x02000000
#define PORT_FEATURE_MBA_BIOS_BOOTSTRAP_INT19H    0x03000000
#define PORT_FEATURE_MBA_LINK_SPEED_MASK    0x3c000000
#define PORT_FEATURE_MBA_LINK_SPEED_SHIFT    26
#define PORT_FEATURE_MBA_LINK_SPEED_AUTO    0x00000000
#define PORT_FEATURE_MBA_LINK_SPEED_10HD    0x04000000
#define PORT_FEATURE_MBA_LINK_SPEED_10FD    0x08000000
#define PORT_FEATURE_MBA_LINK_SPEED_100HD    0x0c000000
#define PORT_FEATURE_MBA_LINK_SPEED_100FD    0x10000000
#define PORT_FEATURE_MBA_LINK_SPEED_1GBPS    0x14000000
#define PORT_FEATURE_MBA_LINK_SPEED_2_5GBPS    0x18000000
#define PORT_FEATURE_MBA_LINK_SPEED_10GBPS_CX4    0x1c000000
#define PORT_FEATURE_MBA_LINK_SPEED_10GBPS_KX4    0x20000000
#define PORT_FEATURE_MBA_LINK_SPEED_10GBPS_KR    0x24000000
#define PORT_FEATURE_MBA_LINK_SPEED_12GBPS    0x28000000
#define PORT_FEATURE_MBA_LINK_SPEED_12_5GBPS    0x2c000000
#define PORT_FEATURE_MBA_LINK_SPEED_13GBPS    0x30000000
#define PORT_FEATURE_MBA_LINK_SPEED_15GBPS    0x34000000
#define PORT_FEATURE_MBA_LINK_SPEED_16GBPS    0x38000000

   uint32_t bmc_config;
#define PORT_FEATURE_BMC_LINK_OVERRIDE_DEFAULT    0x00000000
#define PORT_FEATURE_BMC_LINK_OVERRIDE_EN    0x00000001

   uint32_t mba_vlan_cfg;
#define PORT_FEATURE_MBA_VLAN_TAG_MASK    0x0000ffff
#define PORT_FEATURE_MBA_VLAN_TAG_SHIFT    0
#define PORT_FEATURE_MBA_VLAN_EN    0x00010000

   uint32_t resource_cfg;
#define PORT_FEATURE_RESOURCE_CFG_VALID    0x00000001
#define PORT_FEATURE_RESOURCE_CFG_DIAG    0x00000002
#define PORT_FEATURE_RESOURCE_CFG_L2    0x00000004
#define PORT_FEATURE_RESOURCE_CFG_ISCSI    0x00000008
#define PORT_FEATURE_RESOURCE_CFG_RDMA    0x00000010

   uint32_t smbus_config;
   /* Obsolete */
#define PORT_FEATURE_SMBUS_EN    0x00000001
#define PORT_FEATURE_SMBUS_ADDR_MASK    0x000000fe
#define PORT_FEATURE_SMBUS_ADDR_SHIFT    1

   uint32_t reserved1;

   uint32_t link_config;    /* Used as HW defaults for the driver */
#define PORT_FEATURE_CONNECTED_SWITCH_MASK    0x03000000
#define PORT_FEATURE_CONNECTED_SWITCH_SHIFT    24
   /* (forced) low speed switch (< 10G) */
#define PORT_FEATURE_CON_SWITCH_1G_SWITCH    0x00000000
   /* (forced) high speed switch (>= 10G) */
#define PORT_FEATURE_CON_SWITCH_10G_SWITCH    0x01000000
#define PORT_FEATURE_CON_SWITCH_AUTO_DETECT    0x02000000
#define PORT_FEATURE_CON_SWITCH_ONE_TIME_DETECT    0x03000000

#define PORT_FEATURE_LINK_SPEED_MASK    0x000f0000
```

```
#define PORT_FEATURE_LINK_SPEED_SHIFT    16
#define PORT_FEATURE_LINK_SPEED_AUTO    0x00000000
#define PORT_FEATURE_LINK_SPEED_10M_FULL    0x00010000
#define PORT_FEATURE_LINK_SPEED_10M_HALF    0x00020000
#define PORT_FEATURE_LINK_SPEED_100M_HALF    0x00030000
#define PORT_FEATURE_LINK_SPEED_100M_FULL    0x00040000
#define PORT_FEATURE_LINK_SPEED_1G    0x00050000
#define PORT_FEATURE_LINK_SPEED_2_5G    0x00060000
#define PORT_FEATURE_LINK_SPEED_10G_CX4     0x00070000
#define PORT_FEATURE_LINK_SPEED_10G_KX4     0x00080000
#define PORT_FEATURE_LINK_SPEED_10G_KR    0x00090000
#define PORT_FEATURE_LINK_SPEED_12G    0x000a0000
#define PORT_FEATURE_LINK_SPEED_12_5G    0x000b0000
#define PORT_FEATURE_LINK_SPEED_13G    0x000c0000
#define PORT_FEATURE_LINK_SPEED_15G    0x000d0000
#define PORT_FEATURE_LINK_SPEED_16G    0x000e0000

#define PORT_FEATURE_FLOW_CONTROL_MASK    0x00000700
#define PORT_FEATURE_FLOW_CONTROL_SHIFT     8
#define PORT_FEATURE_FLOW_CONTROL_AUTO    0x00000000
#define PORT_FEATURE_FLOW_CONTROL_TX    0x00000100
#define PORT_FEATURE_FLOW_CONTROL_RX    0x00000200
#define PORT_FEATURE_FLOW_CONTROL_BOTH    0x00000300
#define PORT_FEATURE_FLOW_CONTROL_NONE    0x00000400

   /* The default for MCP link configuration,
      uses the same defines as link_config */
   uint32_t mfw_wol_link_cfg;

   uint32_t reserved[19];

};


/******************************************************************************
 * Device Information      *
 ******************************************************************************/
struct shm_dev_info {    /* size */

   uint32_t    bc_rev; /* 8 bits each: major, minor, build */       /* 4 */

   struct shared_hw_cfg shared_hw_config;      /* 40 */

   struct port_hw_cfg port_hw_config[PORT_MAX];     /* 400*2=800 */

   struct shared_feat_cfg shared_feature_config;        /* 4 */

   struct port_feat_cfg port_feature_config[PORT_MAX];/* 116*2=232 */

};


#define FUNC_00
#define FUNC_11
#define FUNC_22
#define FUNC_33
```

```
#define FUNC_44
#define FUNC_55
#define FUNC_66
#define FUNC_77
#define E1_FUNC_MAX2
#define E1H_FUNC_MAX8

#define VN_0  0
#define VN_1  1
#define VN_2  2
#define VN_3  3
#define E1VN_MAX1
#define E1HVN_MAX4


/* This value (in milliseconds) determines the frequency of the driver
 * issuing the PULSE message code.  The firmware monitors this periodic
 * pulse to determine when to switch to an OS-absent mode. */
#define DRV_PULSE_PERIOD_MS250

/* This value (in milliseconds) determines how long the driver should
 * wait for an acknowledgement from the firmware before timing out.  Once
 * the firmware has timed out, the driver will assume there is no firmware
 * running and there won't be any firmware-driver synchronization during a
 * driver reset. */
#define FW_ACK_TIME_OUT_MS5000

#define FW_ACK_POLL_TIME_MS1

#define FW_ACK_NUM_OF_POLL(FW_ACK_TIME_OUT_MS/FW_ACK_POLL_TIME_MS)

/* LED Blink rate that will achieve ~15.9Hz */
#define LED_BLINK_RATE_VAL480

/****************************************************************************
 * Driver <-> FW Mailbox    *
 ****************************************************************************/
struct drv_port_mb {

   uint32_t link_status;
   /* Driver should update this field on any link change event */

#define LINK_STATUS_LINK_FLAG_MASK0x00000001
#define LINK_STATUS_LINK_UP0x00000001
#define LINK_STATUS_SPEED_AND_DUPLEX_MASK0x0000001E
#define LINK_STATUS_SPEED_AND_DUPLEX_AN_NOT_COMPLETE(0<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_10THD(1<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_10TFD(2<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_100TXHD(3<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_100T4(4<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_100TXFD(5<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_1000THD(6<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_1000TFD(7<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_1000XFD(7<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_2500THD(8<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_2500TFD(9<<1)
```

```
#define LINK_STATUS_SPEED_AND_DUPLEX_2500XFD(9<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_10GTFD(10<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_10GXFD(10<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_12GTFD(11<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_12GXFD(11<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_12_5GTFD(12<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_12_5GXFD(12<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_13GTFD(13<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_13GXFD(13<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_15GTFD(14<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_15GXFD(14<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_16GTFD(15<<1)
#define LINK_STATUS_SPEED_AND_DUPLEX_16GXFD(15<<1)

#define LINK_STATUS_AUTO_NEGOTIATE_FLAG_MASK0x00000020
#define LINK_STATUS_AUTO_NEGOTIATE_ENABLED0x00000020

#define LINK_STATUS_AUTO_NEGOTIATE_COMPLETE0x00000040
#define LINK_STATUS_PARALLEL_DETECTION_FLAG_MASK0x00000080
#define LINK_STATUS_PARALLEL_DETECTION_USED0x00000080

#define LINK_STATUS_LINK_PARTNER_1000TFD_CAPABLE0x00000200
#define LINK_STATUS_LINK_PARTNER_1000THD_CAPABLE0x00000400
#define LINK_STATUS_LINK_PARTNER_100T4_CAPABLE0x00000800
#define LINK_STATUS_LINK_PARTNER_100TXFD_CAPABLE0x00001000
#define LINK_STATUS_LINK_PARTNER_100TXHD_CAPABLE0x00002000
#define LINK_STATUS_LINK_PARTNER_10TFD_CAPABLE0x00004000
#define LINK_STATUS_LINK_PARTNER_10THD_CAPABLE0x00008000

#define LINK_STATUS_TX_FLOW_CONTROL_FLAG_MASK0x00010000
#define LINK_STATUS_TX_FLOW_CONTROL_ENABLED0x00010000

#define LINK_STATUS_RX_FLOW_CONTROL_FLAG_MASK0x00020000
#define LINK_STATUS_RX_FLOW_CONTROL_ENABLED0x00020000

#define LINK_STATUS_LINK_PARTNER_FLOW_CONTROL_MASK0x000C0000
#define LINK_STATUS_LINK_PARTNER_NOT_PAUSE_CAPABLE(0<<18)
#define LINK_STATUS_LINK_PARTNER_SYMMETRIC_PAUSE(1<<18)
#define LINK_STATUS_LINK_PARTNER_ASYMMETRIC_PAUSE(2<<18)
#define LINK_STATUS_LINK_PARTNER_BOTH_PAUSE(3<<18)

#define LINK_STATUS_SERDES_LINK 0x00100000

#define LINK_STATUS_LINK_PARTNER_2500XFD_CAPABLE0x00200000
#define LINK_STATUS_LINK_PARTNER_2500XHD_CAPABLE0x00400000
#define LINK_STATUS_LINK_PARTNER_10GXFD_CAPABLE 0x00800000
#define LINK_STATUS_LINK_PARTNER_12GXFD_CAPABLE 0x01000000
#define LINK_STATUS_LINK_PARTNER_12_5GXFD_CAPABLE0x02000000
#define LINK_STATUS_LINK_PARTNER_13GXFD_CAPABLE 0x04000000
#define LINK_STATUS_LINK_PARTNER_15GXFD_CAPABLE 0x08000000
#define LINK_STATUS_LINK_PARTNER_16GXFD_CAPABLE 0x10000000

   uint32_t port_stx;

   uint32_t stat_nig_timer;
```

```
    uint32_t reserved[1];

};


struct drv_func_mb {

    uint32_t drv_mb_header;
#define DRV_MSG_CODE_MASK0xffff0000
#define DRV_MSG_CODE_LOAD_REQ0x10000000
#define DRV_MSG_CODE_LOAD_DONE0x11000000
#define DRV_MSG_CODE_UNLOAD_REQ_WOL_EN0x20000000
#define DRV_MSG_CODE_UNLOAD_REQ_WOL_DIS 0x20010000
#define DRV_MSG_CODE_UNLOAD_REQ_WOL_MCP 0x20020000
#define DRV_MSG_CODE_UNLOAD_DONE0x21000000
#define DRV_MSG_CODE_DIAG_ENTER_REQ0x50000000
#define DRV_MSG_CODE_DIAG_EXIT_REQ0x60000000
#define DRV_MSG_CODE_VALIDATE_KEY0x70000000
#define DRV_MSG_CODE_GET_CURR_KEY0x80000000
#define DRV_MSG_CODE_GET_UPGRADE_KEY0x81000000
#define DRV_MSG_CODE_GET_MANUF_KEY0x82000000
#define DRV_MSG_CODE_LOAD_L2B_PRAM0x90000000

#define BIOS_MSG_CODE_LIC_CHALLENGE0xff010000
#define BIOS_MSG_CODE_LIC_RESPONSE0xff020000
#define BIOS_MSG_CODE_VIRT_MAC_PRIM0xff030000
#define BIOS_MSG_CODE_VIRT_MAC_ISCSI0xff040000

#define DRV_MSG_SEQ_NUMBER_MASK 0x0000ffff

    uint32_t drv_mb_param;

    uint32_t fw_mb_header;
#define FW_MSG_CODE_MASK0xffff0000
#define FW_MSG_CODE_DRV_LOAD_COMMON0x10100000
#define FW_MSG_CODE_DRV_LOAD_PORT0x10110000
#define FW_MSG_CODE_DRV_LOAD_FUNCTION0x10120000
#define FW_MSG_CODE_DRV_LOAD_REFUSED0x10200000
#define FW_MSG_CODE_DRV_LOAD_DONE0x11100000
#define FW_MSG_CODE_DRV_UNLOAD_COMMON0x20100000
#define FW_MSG_CODE_DRV_UNLOAD_PORT0x20110000
#define FW_MSG_CODE_DRV_UNLOAD_FUNCTION 0x20120000
#define FW_MSG_CODE_DRV_UNLOAD_DONE0x21100000
#define FW_MSG_CODE_DIAG_ENTER_DONE0x50100000
#define FW_MSG_CODE_DIAG_REFUSE 0x50200000
#define FW_MSG_CODE_DIAG_EXIT_DONE0x60100000
#define FW_MSG_CODE_VALIDATE_KEY_SUCCESS0x70100000
#define FW_MSG_CODE_VALIDATE_KEY_FAILURE0x70200000
#define FW_MSG_CODE_GET_KEY_DONE0x80100000
#define FW_MSG_CODE_NO_KEY0x80f00000
#define FW_MSG_CODE_LIC_INFO_NOT_READY0x80f80000
#define FW_MSG_CODE_L2B_PRAM_LOADED0x90100000
#define FW_MSG_CODE_L2B_PRAM_T_LOAD_FAILURE0x90210000
#define FW_MSG_CODE_L2B_PRAM_C_LOAD_FAILURE0x90220000
#define FW_MSG_CODE_L2B_PRAM_X_LOAD_FAILURE0x90230000
#define FW_MSG_CODE_L2B_PRAM_U_LOAD_FAILURE0x90240000
```

```
#define FW_MSG_CODE_LIC_CHALLENGE0xff010000
#define FW_MSG_CODE_LIC_RESPONSE0xff020000
#define FW_MSG_CODE_VIRT_MAC_PRIM0xff030000
#define FW_MSG_CODE_VIRT_MAC_ISCSI0xff040000

#define FW_MSG_SEQ_NUMBER_MASK0x0000ffff

   uint32_t fw_mb_param;

   uint32_t drv_pulse_mb;
#define DRV_PULSE_SEQ_MASK0x00007fff
#define DRV_PULSE_SYSTEM_TIME_MASK0xffff0000
   /* The system time is in the format of
    * (year-2001)*12*32 + month*32 + day. */
#define DRV_PULSE_ALWAYS_ALIVE0x00008000
   /* Indicate to the firmware not to go into the
    * OS-absent when it is not getting driver pulse.
    * This is used for debugging as well for PXE(MBA). */

   uint32_t mcp_pulse_mb;
#define MCP_PULSE_SEQ_MASK0x00007fff
#define MCP_PULSE_ALWAYS_ALIVE0x00008000
   /* Indicates to the driver not to assert due to lack
    * of MCP response */
#define MCP_EVENT_MASK0xffff0000
#define MCP_EVENT_OTHER_DRIVER_RESET_REQ0x00010000

   uint32_t iscsi_boot_signature;
   uint32_t iscsi_boot_block_offset;

   uint32_t drv_status;
#define DRV_STATUS_PMF0x00000001

   uint32_t virt_mac_upper;
#define VIRT_MAC_SIGN_MASK0xffff0000
#define VIRT_MAC_SIGNATURE0x564d0000
   uint32_t virt_mac_lower;

};


/****************************************************************************
 * Management firmware state    *
 ****************************************************************************/
/* Allocate 440 bytes for management firmware */
#define MGMTFW_STATE_WORD_SIZE    110

struct mgmtfw_state {
   uint32_t opaque[MGMTFW_STATE_WORD_SIZE];
};


/****************************************************************************
 * Multi-Function configuration     *
 ****************************************************************************/
```

```
struct shared_mf_cfg {

   uint32_t clp_mb;
#define SHARED_MF_CLP_SET_DEFAULT    0x00000000
   /* set by CLP */
#define SHARED_MF_CLP_EXIT    0x00000001
   /* set by MCP */
#define SHARED_MF_CLP_EXIT_DONE     0x00010000

};

struct port_mf_cfg {

   uint32_t dynamic_cfg;/* device control channel */
#define PORT_MF_CFG_E1HVN_TAG_MASK    0x0000ffff
#define PORT_MF_CFG_E1HVN_TAG_SHIFT    0
#define PORT_MF_CFG_DYNAMIC_CFG_ENABLED     0x00010000
#define PORT_MF_CFG_DYNAMIC_CFG_DEFAULT     0x00000000

   uint32_t reserved[3];

};

struct func_mf_cfg {

   uint32_t config;
   /* E/R/I/D */
   /* function 0 of each port cannot be hidden */
#define FUNC_MF_CFG_FUNC_HIDE    0x00000001

#define FUNC_MF_CFG_PROTOCOL_MASK    0x00000007
#define FUNC_MF_CFG_PROTOCOL_ETHERNET    0x00000002
#define FUNC_MF_CFG_PROTOCOL_ETHERNET_WITH_RDMA     0x00000004
#define FUNC_MF_CFG_PROTOCOL_ISCSI    0x00000006
#define FUNC_MF_CFG_PROTOCOL_DEFAULT\
   FUNC_MF_CFG_PROTOCOL_ETHERNET_WITH_RDMA

#define FUNC_MF_CFG_FUNC_DISABLED    0x00000008

   /* PRI */
   /* 0 - low priority, 3 - high priority */
#define FUNC_MF_CFG_TRANSMIT_PRIORITY_MASK    0x00000300
#define FUNC_MF_CFG_TRANSMIT_PRIORITY_SHIFT    8
#define FUNC_MF_CFG_TRANSMIT_PRIORITY_DEFAULT    0x00000000

   /* MINBW, MAXBW */
   /* value range - 0..100, increments in 100Mbps */
#define FUNC_MF_CFG_MIN_BW_MASK     0x00ff0000
#define FUNC_MF_CFG_MIN_BW_SHIFT    16
#define FUNC_MF_CFG_MIN_BW_DEFAULT    0x00000000
#define FUNC_MF_CFG_MAX_BW_MASK     0xff000000
#define FUNC_MF_CFG_MAX_BW_SHIFT    24
#define FUNC_MF_CFG_MAX_BW_DEFAULT    0x64000000

   uint32_t mac_upper;/* MAC */
#define FUNC_MF_CFG_UPPERMAC_MASK    0x0000ffff
```

```
#define FUNC_MF_CFG_UPPERMAC_SHIFT    0
#define FUNC_MF_CFG_UPPERMAC_DEFAULT    FUNC_MF_CFG_UPPERMAC_MASK
   uint32_t mac_lower;
#define FUNC_MF_CFG_LOWERMAC_DEFAULT    0xffffffff

   uint32_t e1hov_tag;/* VNI */
#define FUNC_MF_CFG_E1HOV_TAG_MASK    0x0000ffff
#define FUNC_MF_CFG_E1HOV_TAG_SHIFT    0
#define FUNC_MF_CFG_E1HOV_TAG_DEFAULT    FUNC_MF_CFG_E1HOV_TAG_MASK

   uint32_t reserved[2];

};

struct mf_cfg {

   struct shared_mf_cfgshared_mf_config;
   struct port_mf_cfgport_mf_config[PORT_MAX];
   struct func_mf_cfgfunc_mf_config[E1H_FUNC_MAX];

};


/*****************************************************************************
 * Shared Memory Region     *
 *****************************************************************************/
struct shmem_region {       /*   SharedMem Offset (size) */

   uint32_t  validity_map[PORT_MAX];  /* 0x0 (4*2 = 0x8) */
#define SHR_MEM_FORMAT_REV_ID    ('A'<<24)
#define SHR_MEM_FORMAT_REV_MASK     0xff000000
   /* validity bits */
#define SHR_MEM_VALIDITY_PCI_CFG    0x00100000
#define SHR_MEM_VALIDITY_MB    0x00200000
#define SHR_MEM_VALIDITY_DEV_INFO    0x00400000
#define SHR_MEM_VALIDITY_RESERVED    0x00000007
   /* One licensing bit should be set */
#define SHR_MEM_VALIDITY_LIC_KEY_IN_EFFECT_MASK     0x00000038
#define SHR_MEM_VALIDITY_LIC_MANUF_KEY_IN_EFFECT    0x00000008
#define SHR_MEM_VALIDITY_LIC_UPGRADE_KEY_IN_EFFECT  0x00000010
#define SHR_MEM_VALIDITY_LIC_NO_KEY_IN_EFFECT    0x00000020
   /* Active MFW */
#define SHR_MEM_VALIDITY_ACTIVE_MFW_UNKNOWN    0x00000000
#define SHR_MEM_VALIDITY_ACTIVE_MFW_IPMI    0x00000040
#define SHR_MEM_VALIDITY_ACTIVE_MFW_UMP     0x00000080
#define SHR_MEM_VALIDITY_ACTIVE_MFW_NCSI    0x000000c0
#define SHR_MEM_VALIDITY_ACTIVE_MFW_NONE    0x000001c0
#define SHR_MEM_VALIDITY_ACTIVE_MFW_MASK    0x000001c0

   struct shm_dev_infodev_info; /* 0x8     (0x438) */

struct license_keydrv_lic_key[PORT_MAX];/* 0x440 (52*2=0x68) */

   /* FW information (for internal FW use) */
   uint32_t  fw_info_fio_offset;    /* 0x4a8       (0x4) */
   struct mgmtfw_statemgmtfw_state;       /* 0x4ac     (0x1b8) */
```

```
    struct drv_port_mbport_mb[PORT_MAX];     /* 0x664 (16*2=0x20) */
#if !defined(b710) /* BXE_UPSTREAM */
    struct drv_func_mbfunc_mb[E1H_FUNC_MAX];
#else
    struct drv_func_mbfunc_mb[E1_FUNC_MAX];  /* 0x684 (44*2=0x58) */
#endif

#if !defined(b710) /* BXE_UPSTREAM */
    struct mf_cfgmf_cfg;
#endif

};                          /* 0x6dc */


struct emac_stats {
    uint32_t     rx_stat_ifhcinoctets;
    uint32_t     rx_stat_ifhcinbadoctets;
    uint32_t     rx_stat_etherstatsfragments;
    uint32_t     rx_stat_ifhcinucastpkts;
    uint32_t     rx_stat_ifhcinmulticastpkts;
    uint32_t     rx_stat_ifhcinbroadcastpkts;
    uint32_t     rx_stat_dot3statsfcserrors;
    uint32_t     rx_stat_dot3statsalignmenterrors;
    uint32_t     rx_stat_dot3statscarriersenseerrors;
    uint32_t     rx_stat_xonpauseframesreceived;
    uint32_t     rx_stat_xoffpauseframesreceived;
    uint32_t     rx_stat_maccontrolframesreceived;
    uint32_t     rx_stat_xoffstateentered;
    uint32_t     rx_stat_dot3statsframestoolong;
    uint32_t     rx_stat_etherstatsjabbers;
    uint32_t     rx_stat_etherstatsundersizepkts;
    uint32_t     rx_stat_etherstatspkts64octets;
    uint32_t     rx_stat_etherstatspkts65octetsto127octets;
    uint32_t     rx_stat_etherstatspkts128octetsto255octets;
    uint32_t     rx_stat_etherstatspkts256octetsto511octets;
    uint32_t     rx_stat_etherstatspkts512octetsto1023octets;
    uint32_t     rx_stat_etherstatspkts1024octetsto1522octets;
    uint32_t     rx_stat_etherstatspktsover1522octets;

    uint32_t     rx_stat_falsecarriererrors;

    uint32_t     tx_stat_ifhcoutoctets;
    uint32_t     tx_stat_ifhcoutbadoctets;
    uint32_t     tx_stat_etherstatscollisions;
    uint32_t     tx_stat_outxonsent;
    uint32_t     tx_stat_outxoffsent;
    uint32_t     tx_stat_flowcontroldone;
    uint32_t     tx_stat_dot3statssinglecollisionframes;
    uint32_t     tx_stat_dot3statsmultiplecollisionframes;
    uint32_t     tx_stat_dot3statsdeferredtransmissions;
    uint32_t     tx_stat_dot3statsexcessivecollisions;
    uint32_t     tx_stat_dot3statslatecollisions;
    uint32_t     tx_stat_ifhcoutucastpkts;
    uint32_t     tx_stat_ifhcoutmulticastpkts;
    uint32_t     tx_stat_ifhcoutbroadcastpkts;
```

```
    uint32_t     tx_stat_etherstatspkts64octets;
    uint32_t     tx_stat_etherstatspkts65octetsto127octets;
    uint32_t     tx_stat_etherstatspkts128octetsto255octets;
    uint32_t     tx_stat_etherstatspkts256octetsto511octets;
    uint32_t     tx_stat_etherstatspkts512octetsto1023octets;
    uint32_t     tx_stat_etherstatspkts1024octetsto1522octets;
    uint32_t     tx_stat_etherstatspktsover1522octets;
    uint32_t     tx_stat_dot3statsinternalmactransmiterrors;
};


struct bmac_stats {
    uint32_t     tx_stat_gtpkt_lo;
    uint32_t     tx_stat_gtpkt_hi;
    uint32_t     tx_stat_gtxpf_lo;
    uint32_t     tx_stat_gtxpf_hi;
    uint32_t     tx_stat_gtfcs_lo;
    uint32_t     tx_stat_gtfcs_hi;
    uint32_t     tx_stat_gtmca_lo;
    uint32_t     tx_stat_gtmca_hi;
    uint32_t     tx_stat_gtbca_lo;
    uint32_t     tx_stat_gtbca_hi;
    uint32_t     tx_stat_gtfrg_lo;
    uint32_t     tx_stat_gtfrg_hi;
    uint32_t     tx_stat_gtovr_lo;
    uint32_t     tx_stat_gtovr_hi;
    uint32_t     tx_stat_gt64_lo;
    uint32_t     tx_stat_gt64_hi;
    uint32_t     tx_stat_gt127_lo;
    uint32_t     tx_stat_gt127_hi;
    uint32_t     tx_stat_gt255_lo;
    uint32_t     tx_stat_gt255_hi;
    uint32_t     tx_stat_gt511_lo;
    uint32_t     tx_stat_gt511_hi;
    uint32_t     tx_stat_gt1023_lo;
    uint32_t     tx_stat_gt1023_hi;
    uint32_t     tx_stat_gt1518_lo;
    uint32_t     tx_stat_gt1518_hi;
    uint32_t     tx_stat_gt2047_lo;
    uint32_t     tx_stat_gt2047_hi;
    uint32_t     tx_stat_gt4095_lo;
    uint32_t     tx_stat_gt4095_hi;
    uint32_t     tx_stat_gt9216_lo;
    uint32_t     tx_stat_gt9216_hi;
    uint32_t     tx_stat_gt16383_lo;
    uint32_t     tx_stat_gt16383_hi;
    uint32_t     tx_stat_gtmax_lo;
    uint32_t     tx_stat_gtmax_hi;
    uint32_t     tx_stat_gtufl_lo;
    uint32_t     tx_stat_gtufl_hi;
    uint32_t     tx_stat_gterr_lo;
    uint32_t     tx_stat_gterr_hi;
    uint32_t     tx_stat_gtbyt_lo;
    uint32_t     tx_stat_gtbyt_hi;

    uint32_t     rx_stat_gr64_lo;
```

```
    uint32_t     rx_stat_gr64_hi;
    uint32_t     rx_stat_gr127_lo;
    uint32_t     rx_stat_gr127_hi;
    uint32_t     rx_stat_gr255_lo;
    uint32_t     rx_stat_gr255_hi;
    uint32_t     rx_stat_gr511_lo;
    uint32_t     rx_stat_gr511_hi;
    uint32_t     rx_stat_gr1023_lo;
    uint32_t     rx_stat_gr1023_hi;
    uint32_t     rx_stat_gr1518_lo;
    uint32_t     rx_stat_gr1518_hi;
    uint32_t     rx_stat_gr2047_lo;
    uint32_t     rx_stat_gr2047_hi;
    uint32_t     rx_stat_gr4095_lo;
    uint32_t     rx_stat_gr4095_hi;
    uint32_t     rx_stat_gr9216_lo;
    uint32_t     rx_stat_gr9216_hi;
    uint32_t     rx_stat_gr16383_lo;
    uint32_t     rx_stat_gr16383_hi;
    uint32_t     rx_stat_grmax_lo;
    uint32_t     rx_stat_grmax_hi;
    uint32_t     rx_stat_grpkt_lo;
    uint32_t     rx_stat_grpkt_hi;
    uint32_t     rx_stat_grfcs_lo;
    uint32_t     rx_stat_grfcs_hi;
    uint32_t     rx_stat_grmca_lo;
    uint32_t     rx_stat_grmca_hi;
    uint32_t     rx_stat_grbca_lo;
    uint32_t     rx_stat_grbca_hi;
    uint32_t     rx_stat_grxcf_lo;
    uint32_t     rx_stat_grxcf_hi;
    uint32_t     rx_stat_grxpf_lo;
    uint32_t     rx_stat_grxpf_hi;
    uint32_t     rx_stat_grxuo_lo;
    uint32_t     rx_stat_grxuo_hi;
    uint32_t     rx_stat_grjbr_lo;
    uint32_t     rx_stat_grjbr_hi;
    uint32_t     rx_stat_grovr_lo;
    uint32_t     rx_stat_grovr_hi;
    uint32_t     rx_stat_grflr_lo;
    uint32_t     rx_stat_grflr_hi;
    uint32_t     rx_stat_grmeg_lo;
    uint32_t     rx_stat_grmeg_hi;
    uint32_t     rx_stat_grmeb_lo;
    uint32_t     rx_stat_grmeb_hi;
    uint32_t     rx_stat_grbyt_lo;
    uint32_t     rx_stat_grbyt_hi;
    uint32_t     rx_stat_grund_lo;
    uint32_t     rx_stat_grund_hi;
    uint32_t     rx_stat_grfrg_lo;
    uint32_t     rx_stat_grfrg_hi;
    uint32_t     rx_stat_grerb_lo;
    uint32_t     rx_stat_grerb_hi;
    uint32_t     rx_stat_grfre_lo;
    uint32_t     rx_stat_grfre_hi;
    uint32_t     rx_stat_gripj_lo;
```

```
    uint32_t     rx_stat_gripj_hi;
};


union mac_stats {
    struct emac_statsemac_stats;
    struct bmac_statsbmac_stats;
};


struct mac_stx {
    /* in_bad_octets */
    uint32_t     rx_stat_ifhcinbadoctets_hi;
    uint32_t     rx_stat_ifhcinbadoctets_lo;

    /* out_bad_octets */
    uint32_t     tx_stat_ifhcoutbadoctets_hi;
    uint32_t     tx_stat_ifhcoutbadoctets_lo;

    /* crc_receive_errors */
    uint32_t     rx_stat_dot3statsfcserrors_hi;
    uint32_t     rx_stat_dot3statsfcserrors_lo;
    /* alignment_errors */
    uint32_t     rx_stat_dot3statsalignmenterrors_hi;
    uint32_t     rx_stat_dot3statsalignmenterrors_lo;
    /* carrier_sense_errors */
    uint32_t     rx_stat_dot3statscarriersenseerrors_hi;
    uint32_t     rx_stat_dot3statscarriersenseerrors_lo;
    /* false_carrier_detections */
    uint32_t     rx_stat_falsecarriererrors_hi;
    uint32_t     rx_stat_falsecarriererrors_lo;

    /* runt_packets_received */
    uint32_t     rx_stat_etherstatsundersizepkts_hi;
    uint32_t     rx_stat_etherstatsundersizepkts_lo;
    /* jabber_packets_received */
    uint32_t     rx_stat_dot3statsframestoolong_hi;
    uint32_t     rx_stat_dot3statsframestoolong_lo;

    /* error_runt_packets_received */
    uint32_t     rx_stat_etherstatsfragments_hi;
    uint32_t     rx_stat_etherstatsfragments_lo;
    /* error_jabber_packets_received */
    uint32_t     rx_stat_etherstatsjabbers_hi;
    uint32_t     rx_stat_etherstatsjabbers_lo;

    /* control_frames_received */
    uint32_t     rx_stat_maccontrolframesreceived_hi;
    uint32_t     rx_stat_maccontrolframesreceived_lo;
    uint32_t     rx_stat_bmac_xpf_hi;
    uint32_t     rx_stat_bmac_xpf_lo;
    uint32_t     rx_stat_bmac_xcf_hi;
    uint32_t     rx_stat_bmac_xcf_lo;

    /* xoff_state_entered */
    uint32_t     rx_stat_xoffstateentered_hi;
```

```
uint32_t     rx_stat_xoffstateentered_lo;
/* pause_xon_frames_received */
uint32_t     rx_stat_xonpauseframesreceived_hi;
uint32_t     rx_stat_xonpauseframesreceived_lo;
/* pause_xoff_frames_received */
uint32_t     rx_stat_xoffpauseframesreceived_hi;
uint32_t     rx_stat_xoffpauseframesreceived_lo;
/* pause_xon_frames_transmitted */
uint32_t     tx_stat_outxonsent_hi;
uint32_t     tx_stat_outxonsent_lo;
/* pause_xoff_frames_transmitted */
uint32_t     tx_stat_outxoffsent_hi;
uint32_t     tx_stat_outxoffsent_lo;
/* flow_control_done */
uint32_t     tx_stat_flowcontroldone_hi;
uint32_t     tx_stat_flowcontroldone_lo;

/* ether_stats_collisions */
uint32_t     tx_stat_etherstatscollisions_hi;
uint32_t     tx_stat_etherstatscollisions_lo;
/* single_collision_transmit_frames */
uint32_t     tx_stat_dot3statssinglecollisionframes_hi;
uint32_t     tx_stat_dot3statssinglecollisionframes_lo;
/* multiple_collision_transmit_frames */
uint32_t     tx_stat_dot3statsmultiplecollisionframes_hi;
uint32_t     tx_stat_dot3statsmultiplecollisionframes_lo;
/* deferred_transmissions */
uint32_t     tx_stat_dot3statsdeferredtransmissions_hi;
uint32_t     tx_stat_dot3statsdeferredtransmissions_lo;
/* excessive_collision_frames */
uint32_t     tx_stat_dot3statsexcessivecollisions_hi;
uint32_t     tx_stat_dot3statsexcessivecollisions_lo;
/* late_collision_frames */
uint32_t     tx_stat_dot3statslatecollisions_hi;
uint32_t     tx_stat_dot3statslatecollisions_lo;

/* frames_transmitted_64_bytes */
uint32_t     tx_stat_etherstatspkts64octets_hi;
uint32_t     tx_stat_etherstatspkts64octets_lo;
/* frames_transmitted_65_127_bytes */
uint32_t     tx_stat_etherstatspkts65octetsto127octets_hi;
uint32_t     tx_stat_etherstatspkts65octetsto127octets_lo;
/* frames_transmitted_128_255_bytes */
uint32_t     tx_stat_etherstatspkts128octetsto255octets_hi;
uint32_t     tx_stat_etherstatspkts128octetsto255octets_lo;
/* frames_transmitted_256_511_bytes */
uint32_t     tx_stat_etherstatspkts256octetsto511octets_hi;
uint32_t     tx_stat_etherstatspkts256octetsto511octets_lo;
/* frames_transmitted_512_1023_bytes */
uint32_t     tx_stat_etherstatspkts512octetsto1023octets_hi;
uint32_t     tx_stat_etherstatspkts512octetsto1023octets_lo;
/* frames_transmitted_1024_1522_bytes */
uint32_t     tx_stat_etherstatspkts1024octetsto1522octets_hi;
uint32_t     tx_stat_etherstatspkts1024octetsto1522octets_lo;
/* frames_transmitted_1523_9022_bytes */
uint32_t     tx_stat_etherstatspktsover1522octets_hi;
```

```
    uint32_t     tx_stat_etherstatspktsover1522octets_lo;
    uint32_t     tx_stat_bmac_2047_hi;
    uint32_t     tx_stat_bmac_2047_lo;
    uint32_t     tx_stat_bmac_4095_hi;
    uint32_t     tx_stat_bmac_4095_lo;
    uint32_t     tx_stat_bmac_9216_hi;
    uint32_t     tx_stat_bmac_9216_lo;
    uint32_t     tx_stat_bmac_16383_hi;
    uint32_t     tx_stat_bmac_16383_lo;

    /* internal_mac_transmit_errors */
    uint32_t     tx_stat_dot3statsinternalmactransmiterrors_hi;
    uint32_t     tx_stat_dot3statsinternalmactransmiterrors_lo;

    /* if_out_discards */
    uint32_t     tx_stat_bmac_ufl_hi;
    uint32_t     tx_stat_bmac_ufl_lo;
};


#define MAC_STX_IDX_MAX     2

struct host_port_stats {
    uint32_t    host_port_stats_start;

    struct mac_stx mac_stx[MAC_STX_IDX_MAX];

    uint32_t    brb_drop_hi;
    uint32_t    brb_drop_lo;

    uint32_t    host_port_stats_end;
};


struct host_func_stats {
    uint32_t     host_func_stats_start;

    uint32_t     total_bytes_received_hi;
    uint32_t     total_bytes_received_lo;

    uint32_t     total_bytes_transmitted_hi;
    uint32_t     total_bytes_transmitted_lo;

    uint32_t     total_unicast_packets_received_hi;
    uint32_t     total_unicast_packets_received_lo;

    uint32_t     total_multicast_packets_received_hi;
    uint32_t     total_multicast_packets_received_lo;

    uint32_t     total_broadcast_packets_received_hi;
    uint32_t     total_broadcast_packets_received_lo;

    uint32_t     total_unicast_packets_transmitted_hi;
    uint32_t     total_unicast_packets_transmitted_lo;

    uint32_t     total_multicast_packets_transmitted_hi;
```

```
    uint32_t     total_multicast_packets_transmitted_lo;

    uint32_t     total_broadcast_packets_transmitted_hi;
    uint32_t     total_broadcast_packets_transmitted_lo;

    uint32_t     valid_bytes_received_hi;
    uint32_t     valid_bytes_received_lo;

    uint32_t     host_func_stats_end;
};


#define BCM_5710_FW_MAJOR_VERSION4
#define BCM_5710_FW_MINOR_VERSION6
#define BCM_5710_FW_REVISION_VERSION18
#define BCM_5710_FW_ENGINEERING_VERSION 0
#define BCM_5710_FW_COMPILE_FLAGS1


/*
 * attention bits
 */
struct atten_def_status_block {
   uint32_t attn_bits;
   uint32_t attn_bits_ack;
   uint8_t status_block_id;
   uint8_t reserved0;
   uint16_t attn_bits_index;
   uint32_t reserved1;
};


/*
 * common data for all protocols
 */
struct doorbell_hdr {
   uint8_t header;
#define DOORBELL_HDR_RX (0x1<<0)
#define DOORBELL_HDR_RX_SHIFT 0
#define DOORBELL_HDR_DB_TYPE (0x1<<1)
#define DOORBELL_HDR_DB_TYPE_SHIFT 1
#define DOORBELL_HDR_DPM_SIZE (0x3<<2)
#define DOORBELL_HDR_DPM_SIZE_SHIFT 2
#define DOORBELL_HDR_CONN_TYPE (0xF<<4)
#define DOORBELL_HDR_CONN_TYPE_SHIFT 4
};

/*
 * doorbell message sent to the chip
 */
struct doorbell {
#if defined(__BIG_ENDIAN)
   uint16_t zero_fill2;
   uint8_t zero_fill1;
   struct doorbell_hdr header;
#elif defined(__LITTLE_ENDIAN)
```

```
    struct doorbell_hdr header;
    uint8_t zero_fill1;
    uint16_t zero_fill2;
#endif
};


/*
 * IGU driver acknowledgement register
 */
struct igu_ack_register {
#if defined(__BIG_ENDIAN)
    uint16_t sb_id_and_flags;
#define IGU_ACK_REGISTER_STATUS_BLOCK_ID (0x1F<<0)
#define IGU_ACK_REGISTER_STATUS_BLOCK_ID_SHIFT 0
#define IGU_ACK_REGISTER_STORM_ID (0x7<<5)
#define IGU_ACK_REGISTER_STORM_ID_SHIFT 5
#define IGU_ACK_REGISTER_UPDATE_INDEX (0x1<<8)
#define IGU_ACK_REGISTER_UPDATE_INDEX_SHIFT 8
#define IGU_ACK_REGISTER_INTERRUPT_MODE (0x3<<9)
#define IGU_ACK_REGISTER_INTERRUPT_MODE_SHIFT 9
#define IGU_ACK_REGISTER_RESERVED (0x1F<<11)
#define IGU_ACK_REGISTER_RESERVED_SHIFT 11
    uint16_t status_block_index;
#elif defined(__LITTLE_ENDIAN)
    uint16_t status_block_index;
    uint16_t sb_id_and_flags;
#define IGU_ACK_REGISTER_STATUS_BLOCK_ID (0x1F<<0)
#define IGU_ACK_REGISTER_STATUS_BLOCK_ID_SHIFT 0
#define IGU_ACK_REGISTER_STORM_ID (0x7<<5)
#define IGU_ACK_REGISTER_STORM_ID_SHIFT 5
#define IGU_ACK_REGISTER_UPDATE_INDEX (0x1<<8)
#define IGU_ACK_REGISTER_UPDATE_INDEX_SHIFT 8
#define IGU_ACK_REGISTER_INTERRUPT_MODE (0x3<<9)
#define IGU_ACK_REGISTER_INTERRUPT_MODE_SHIFT 9
#define IGU_ACK_REGISTER_RESERVED (0x1F<<11)
#define IGU_ACK_REGISTER_RESERVED_SHIFT 11
#endif
};


/*
 * Parser parsing flags field
 */
struct parsing_flags {
    uint16_t flags;
#define PARSING_FLAGS_ETHERNET_ADDRESS_TYPE (0x1<<0)
#define PARSING_FLAGS_ETHERNET_ADDRESS_TYPE_SHIFT 0
#define PARSING_FLAGS_VLAN (0x1<<1)
#define PARSING_FLAGS_VLAN_SHIFT 1
#define PARSING_FLAGS_EXTRA_VLAN (0x1<<2)
#define PARSING_FLAGS_EXTRA_VLAN_SHIFT 2
#define PARSING_FLAGS_OVER_ETHERNET_PROTOCOL (0x3<<3)
#define PARSING_FLAGS_OVER_ETHERNET_PROTOCOL_SHIFT 3
#define PARSING_FLAGS_IP_OPTIONS (0x1<<5)
#define PARSING_FLAGS_IP_OPTIONS_SHIFT 5
```

```
#define PARSING_FLAGS_FRAGMENTATION_STATUS (0x1<<6)
#define PARSING_FLAGS_FRAGMENTATION_STATUS_SHIFT 6
#define PARSING_FLAGS_OVER_IP_PROTOCOL (0x3<<7)
#define PARSING_FLAGS_OVER_IP_PROTOCOL_SHIFT 7
#define PARSING_FLAGS_PURE_ACK_INDICATION (0x1<<9)
#define PARSING_FLAGS_PURE_ACK_INDICATION_SHIFT 9
#define PARSING_FLAGS_TCP_OPTIONS_EXIST (0x1<<10)
#define PARSING_FLAGS_TCP_OPTIONS_EXIST_SHIFT 10
#define PARSING_FLAGS_TIME_STAMP_EXIST_FLAG (0x1<<11)
#define PARSING_FLAGS_TIME_STAMP_EXIST_FLAG_SHIFT 11
#define PARSING_FLAGS_CONNECTION_MATCH (0x1<<12)
#define PARSING_FLAGS_CONNECTION_MATCH_SHIFT 12
#define PARSING_FLAGS_LLC_SNAP (0x1<<13)
#define PARSING_FLAGS_LLC_SNAP_SHIFT 13
#define PARSING_FLAGS_RESERVED0 (0x3<<14)
#define PARSING_FLAGS_RESERVED0_SHIFT 14
};


struct regpair {
   uint32_t lo;
   uint32_t hi;
};


/*
 * dmae command structure
 */
struct dmae_command {
   uint32_t opcode;
#define DMAE_COMMAND_SRC (0x1<<0)
#define DMAE_COMMAND_SRC_SHIFT 0
#define DMAE_COMMAND_DST (0x3<<1)
#define DMAE_COMMAND_DST_SHIFT 1
#define DMAE_COMMAND_C_DST (0x1<<3)
#define DMAE_COMMAND_C_DST_SHIFT 3
#define DMAE_COMMAND_C_TYPE_ENABLE (0x1<<4)
#define DMAE_COMMAND_C_TYPE_ENABLE_SHIFT 4
#define DMAE_COMMAND_C_TYPE_CRC_ENABLE (0x1<<5)
#define DMAE_COMMAND_C_TYPE_CRC_ENABLE_SHIFT 5
#define DMAE_COMMAND_C_TYPE_CRC_OFFSET (0x7<<6)
#define DMAE_COMMAND_C_TYPE_CRC_OFFSET_SHIFT 6
#define DMAE_COMMAND_ENDIANITY (0x3<<9)
#define DMAE_COMMAND_ENDIANITY_SHIFT 9
#define DMAE_COMMAND_PORT (0x1<<11)
#define DMAE_COMMAND_PORT_SHIFT 11
#define DMAE_COMMAND_CRC_RESET (0x1<<12)
#define DMAE_COMMAND_CRC_RESET_SHIFT 12
#define DMAE_COMMAND_SRC_RESET (0x1<<13)
#define DMAE_COMMAND_SRC_RESET_SHIFT 13
#define DMAE_COMMAND_DST_RESET (0x1<<14)
#define DMAE_COMMAND_DST_RESET_SHIFT 14
#define DMAE_COMMAND_E1HVN (0x3<<15)
#define DMAE_COMMAND_E1HVN_SHIFT 15
#define DMAE_COMMAND_RESERVED0 (0x7FFF<<17)
#define DMAE_COMMAND_RESERVED0_SHIFT 17
```

```
    uint32_t src_addr_lo;
    uint32_t src_addr_hi;
    uint32_t dst_addr_lo;
    uint32_t dst_addr_hi;
#if defined(__BIG_ENDIAN)
    uint16_t reserved1;
    uint16_t len;
#elif defined(__LITTLE_ENDIAN)
    uint16_t len;
    uint16_t reserved1;
#endif
    uint32_t comp_addr_lo;
    uint32_t comp_addr_hi;
    uint32_t comp_val;
    uint32_t crc32;
    uint32_t crc32_c;
#if defined(__BIG_ENDIAN)
    uint16_t crc16_c;
    uint16_t crc16;
#elif defined(__LITTLE_ENDIAN)
    uint16_t crc16;
    uint16_t crc16_c;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t reserved2;
    uint16_t crc_t10;
#elif defined(__LITTLE_ENDIAN)
    uint16_t crc_t10;
    uint16_t reserved2;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t xsum8;
    uint16_t xsum16;
#elif defined(__LITTLE_ENDIAN)
    uint16_t xsum16;
    uint16_t xsum8;
#endif
};


struct double_regpair {
    uint32_t regpair0_lo;
    uint32_t regpair0_hi;
    uint32_t regpair1_lo;
    uint32_t regpair1_hi;
};


/*
 * The eth storm context of Ustorm (configuration part)
 */
struct ustorm_eth_st_context_config {
#if defined(__BIG_ENDIAN)
    uint8_t flags;
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_MC_ALIGNMENT (0x1<<0)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_MC_ALIGNMENT_SHIFT 0
```

```
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_DYNAMIC_HC (0x1<<1)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_DYNAMIC_HC_SHIFT 1
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_TPA (0x1<<2)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_TPA_SHIFT 2
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_SGE_RING (0x1<<3)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_SGE_RING_SHIFT 3
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_STATISTICS (0x1<<4)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_STATISTICS_SHIFT 4
#define __USTORM_ETH_ST_CONTEXT_CONFIG_RESERVED0 (0x7<<5)
#define __USTORM_ETH_ST_CONTEXT_CONFIG_RESERVED0_SHIFT 5
   uint8_t status_block_id;
   uint8_t clientId;
   uint8_t sb_index_numbers;
#define USTORM_ETH_ST_CONTEXT_CONFIG_CQE_SB_INDEX_NUMBER (0xF<<0)
#define USTORM_ETH_ST_CONTEXT_CONFIG_CQE_SB_INDEX_NUMBER_SHIFT 0
#define USTORM_ETH_ST_CONTEXT_CONFIG_BD_SB_INDEX_NUMBER (0xF<<4)
#define USTORM_ETH_ST_CONTEXT_CONFIG_BD_SB_INDEX_NUMBER_SHIFT 4
#elif defined(__LITTLE_ENDIAN)
   uint8_t sb_index_numbers;
#define USTORM_ETH_ST_CONTEXT_CONFIG_CQE_SB_INDEX_NUMBER (0xF<<0)
#define USTORM_ETH_ST_CONTEXT_CONFIG_CQE_SB_INDEX_NUMBER_SHIFT 0
#define USTORM_ETH_ST_CONTEXT_CONFIG_BD_SB_INDEX_NUMBER (0xF<<4)
#define USTORM_ETH_ST_CONTEXT_CONFIG_BD_SB_INDEX_NUMBER_SHIFT 4
   uint8_t clientId;
   uint8_t status_block_id;
   uint8_t flags;
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_MC_ALIGNMENT (0x1<<0)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_MC_ALIGNMENT_SHIFT 0
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_DYNAMIC_HC (0x1<<1)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_DYNAMIC_HC_SHIFT 1
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_TPA (0x1<<2)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_TPA_SHIFT 2
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_SGE_RING (0x1<<3)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_SGE_RING_SHIFT 3
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_STATISTICS (0x1<<4)
#define USTORM_ETH_ST_CONTEXT_CONFIG_ENABLE_STATISTICS_SHIFT 4
#define __USTORM_ETH_ST_CONTEXT_CONFIG_RESERVED0 (0x7<<5)
#define __USTORM_ETH_ST_CONTEXT_CONFIG_RESERVED0_SHIFT 5
#endif
#if defined(__BIG_ENDIAN)
   uint16_t bd_buff_size;
   uint8_t statistics_counter_id;
   uint8_t mc_alignment_log_size;
#elif defined(__LITTLE_ENDIAN)
   uint8_t mc_alignment_log_size;
   uint8_t statistics_counter_id;
   uint16_t bd_buff_size;
#endif
#if defined(__BIG_ENDIAN)
   uint8_t __local_sge_prod;
   uint8_t __local_bd_prod;
   uint16_t sge_buff_size;
#elif defined(__LITTLE_ENDIAN)
   uint16_t sge_buff_size;
   uint8_t __local_bd_prod;
   uint8_t __local_sge_prod;
```

```
#endif
    uint32_t reserved;
    uint32_t bd_page_base_lo;
    uint32_t bd_page_base_hi;
    uint32_t sge_page_base_lo;
    uint32_t sge_page_base_hi;
};

/*
 * The eth Rx Buffer Descriptor
 */
struct eth_rx_bd {
    uint32_t addr_lo;
    uint32_t addr_hi;
};

/*
 * The eth Rx SGE Descriptor
 */
struct eth_rx_sge {
    uint32_t addr_lo;
    uint32_t addr_hi;
};

/*
 * Local BDs and SGEs rings (in ETH)
 */
struct eth_local_rx_rings {
    struct eth_rx_bd __local_bd_ring[16];
    struct eth_rx_sge __local_sge_ring[12];
};

/*
 * The eth storm context of Ustorm
 */
struct ustorm_eth_st_context {
    struct ustorm_eth_st_context_config common;
    struct eth_local_rx_rings __rings;
};

/*
 * The eth storm context of Tstorm
 */
struct tstorm_eth_st_context {
    uint32_t __reserved0[28];
};

/*
 * The eth aggregative context section of Xstorm
 */
struct xstorm_eth_extra_ag_context_section {
#if defined(__BIG_ENDIAN)
    uint8_t __tcp_agg_vars1;
    uint8_t __reserved50;
    uint16_t __mss;
#elif defined(__LITTLE_ENDIAN)
```

```
    uint16_t __mss;
    uint8_t __reserved50;
    uint8_t __tcp_agg_vars1;
#endif
    uint32_t __snd_nxt;
    uint32_t __tx_wnd;
    uint32_t __snd_una;
    uint32_t __reserved53;
#if defined(__BIG_ENDIAN)
    uint8_t __agg_val8_th;
    uint8_t __agg_val8;
    uint16_t __tcp_agg_vars2;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __tcp_agg_vars2;
    uint8_t __agg_val8;
    uint8_t __agg_val8_th;
#endif
    uint32_t __reserved58;
    uint32_t __reserved59;
    uint32_t __reserved60;
    uint32_t __reserved61;
#if defined(__BIG_ENDIAN)
    uint16_t __agg_val7_th;
    uint16_t __agg_val7;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_val7;
    uint16_t __agg_val7_th;
#endif
#if defined(__BIG_ENDIAN)
    uint8_t __tcp_agg_vars5;
    uint8_t __tcp_agg_vars4;
    uint8_t __tcp_agg_vars3;
    uint8_t __reserved62;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __reserved62;
    uint8_t __tcp_agg_vars3;
    uint8_t __tcp_agg_vars4;
    uint8_t __tcp_agg_vars5;
#endif
    uint32_t __tcp_agg_vars6;
#if defined(__BIG_ENDIAN)
    uint16_t __agg_misc6;
    uint16_t __tcp_agg_vars7;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __tcp_agg_vars7;
    uint16_t __agg_misc6;
#endif
    uint32_t __agg_val10;
    uint32_t __agg_val10_th;
#if defined(__BIG_ENDIAN)
    uint16_t __reserved3;
    uint8_t __reserved2;
    uint8_t __da_only_cnt;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __da_only_cnt;
    uint8_t __reserved2;
```

```
    uint16_t __reserved3;
#endif
};

/*
 * The eth aggregative context of Xstorm
 */
struct xstorm_eth_ag_context {
#if defined(__BIG_ENDIAN)
    uint16_t __bd_prod;
    uint8_t __agg_vars1;
    uint8_t __state;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __state;
    uint8_t __agg_vars1;
    uint16_t __bd_prod;
#endif
#if defined(__BIG_ENDIAN)
    uint8_t cdu_reserved;
    uint8_t __agg_vars4;
    uint8_t __agg_vars3;
    uint8_t __agg_vars2;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __agg_vars2;
    uint8_t __agg_vars3;
    uint8_t __agg_vars4;
    uint8_t cdu_reserved;
#endif
    uint32_t __more_packets_to_send;
#if defined(__BIG_ENDIAN)
    uint16_t __agg_vars5;
    uint16_t __agg_val4_th;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_val4_th;
    uint16_t __agg_vars5;
#endif
    struct xstorm_eth_extra_ag_context_section __extra_section;
#if defined(__BIG_ENDIAN)
    uint16_t __agg_vars7;
    uint8_t __agg_val3_th;
    uint8_t __agg_vars6;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __agg_vars6;
    uint8_t __agg_val3_th;
    uint16_t __agg_vars7;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t __agg_val11_th;
    uint16_t __agg_val11;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_val11;
    uint16_t __agg_val11_th;
#endif
#if defined(__BIG_ENDIAN)
    uint8_t __reserved1;
    uint8_t __agg_val6_th;
```

```
    uint16_t __agg_val9;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_val9;
    uint8_t __agg_val6_th;
    uint8_t __reserved1;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t __agg_val2_th;
    uint16_t __agg_val2;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_val2;
    uint16_t __agg_val2_th;
#endif
    uint32_t __agg_vars8;
#if defined(__BIG_ENDIAN)
    uint16_t __agg_misc0;
    uint16_t __agg_val4;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_val4;
    uint16_t __agg_misc0;
#endif
#if defined(__BIG_ENDIAN)
    uint8_t __agg_val3;
    uint8_t __agg_val6;
    uint8_t __agg_val5_th;
    uint8_t __agg_val5;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __agg_val5;
    uint8_t __agg_val5_th;
    uint8_t __agg_val6;
    uint8_t __agg_val3;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t __agg_misc1;
    uint16_t __bd_ind_max_val;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __bd_ind_max_val;
    uint16_t __agg_misc1;
#endif
    uint32_t __reserved57;
    uint32_t __agg_misc4;
    uint32_t __agg_misc5;
};

/*
 * The eth extra aggregative context section of Tstorm
 */
struct tstorm_eth_extra_ag_context_section {
    uint32_t __agg_val1;
#if defined(__BIG_ENDIAN)
    uint8_t __tcp_agg_vars2;
    uint8_t __agg_val3;
    uint16_t __agg_val2;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_val2;
    uint8_t __agg_val3;
```

```
    uint8_t __tcp_agg_vars2;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t __agg_val5;
    uint8_t __agg_val6;
    uint8_t __tcp_agg_vars3;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __tcp_agg_vars3;
    uint8_t __agg_val6;
    uint16_t __agg_val5;
#endif
    uint32_t __reserved63;
    uint32_t __reserved64;
    uint32_t __reserved65;
    uint32_t __reserved66;
    uint32_t __reserved67;
    uint32_t __tcp_agg_vars1;
    uint32_t __reserved61;
    uint32_t __reserved62;
    uint32_t __reserved2;
};

/*
 * The eth aggregative context of Tstorm
 */
struct tstorm_eth_ag_context {
#if defined(__BIG_ENDIAN)
    uint16_t __reserved54;
    uint8_t __agg_vars1;
    uint8_t __state;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __state;
    uint8_t __agg_vars1;
    uint16_t __reserved54;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t __agg_val4;
    uint16_t __agg_vars2;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_vars2;
    uint16_t __agg_val4;
#endif
    struct tstorm_eth_extra_ag_context_section __extra_section;
};

/*
 * The eth aggregative context of Cstorm
 */
struct cstorm_eth_ag_context {
    uint32_t __agg_vars1;
#if defined(__BIG_ENDIAN)
    uint8_t __aux1_th;
    uint8_t __aux1_val;
    uint16_t __agg_vars2;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_vars2;
```

```
    uint8_t __aux1_val;
    uint8_t __aux1_th;
#endif
    uint32_t __num_of_treated_packet;
    uint32_t __last_packet_treated;
#if defined(__BIG_ENDIAN)
    uint16_t __reserved58;
    uint16_t __reserved57;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __reserved57;
    uint16_t __reserved58;
#endif
#if defined(__BIG_ENDIAN)
    uint8_t __reserved62;
    uint8_t __reserved61;
    uint8_t __reserved60;
    uint8_t __reserved59;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __reserved59;
    uint8_t __reserved60;
    uint8_t __reserved61;
    uint8_t __reserved62;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t __reserved64;
    uint16_t __reserved63;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __reserved63;
    uint16_t __reserved64;
#endif
    uint32_t __reserved65;
#if defined(__BIG_ENDIAN)
    uint16_t __agg_vars3;
    uint16_t __rq_inv_cnt;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __rq_inv_cnt;
    uint16_t __agg_vars3;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t __packet_index_th;
    uint16_t __packet_index;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __packet_index;
    uint16_t __packet_index_th;
#endif
};

/*
 * The eth aggregative context of Ustorm
 */
struct ustorm_eth_ag_context {
#if defined(__BIG_ENDIAN)
    uint8_t __aux_counter_flags;
    uint8_t __agg_vars2;
    uint8_t __agg_vars1;
    uint8_t __state;
```

```
#elif defined(__LITTLE_ENDIAN)
    uint8_t __state;
    uint8_t __agg_vars1;
    uint8_t __agg_vars2;
    uint8_t __aux_counter_flags;
#endif
#if defined(__BIG_ENDIAN)
    uint8_t cdu_usage;
    uint8_t __agg_misc2;
    uint16_t __agg_misc1;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_misc1;
    uint8_t __agg_misc2;
    uint8_t cdu_usage;
#endif
    uint32_t __agg_misc4;
#if defined(__BIG_ENDIAN)
    uint8_t __agg_val3_th;
    uint8_t __agg_val3;
    uint16_t __agg_misc3;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_misc3;
    uint8_t __agg_val3;
    uint8_t __agg_val3_th;
#endif
    uint32_t __agg_val1;
    uint32_t __agg_misc4_th;
#if defined(__BIG_ENDIAN)
    uint16_t __agg_val2_th;
    uint16_t __agg_val2;
#elif defined(__LITTLE_ENDIAN)
    uint16_t __agg_val2;
    uint16_t __agg_val2_th;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t __reserved2;
    uint8_t __decision_rules;
    uint8_t __decision_rule_enable_bits;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __decision_rule_enable_bits;
    uint8_t __decision_rules;
    uint16_t __reserved2;
#endif
};

/*
 * Timers connection context
 */
struct timers_block_context {
    uint32_t __reserved_0;
    uint32_t __reserved_1;
    uint32_t __reserved_2;
    uint32_t flags;
#define __TIMERS_BLOCK_CONTEXT_NUM_OF_ACTIVE_TIMERS (0x3<<0)
#define __TIMERS_BLOCK_CONTEXT_NUM_OF_ACTIVE_TIMERS_SHIFT 0
#define TIMERS_BLOCK_CONTEXT_CONN_VALID_FLG (0x1<<2)
```

```
#define TIMERS_BLOCK_CONTEXT_CONN_VALID_FLG_SHIFT 2
#define __TIMERS_BLOCK_CONTEXT_RESERVED0 (0x1FFFFFFF<<3)
#define __TIMERS_BLOCK_CONTEXT_RESERVED0_SHIFT 3
};

/*
 * structure for easy accessbility to assembler
 */
struct eth_tx_bd_flags {
   uint8_t as_bitfield;
#define ETH_TX_BD_FLAGS_VLAN_TAG (0x1<<0)
#define ETH_TX_BD_FLAGS_VLAN_TAG_SHIFT 0
#define ETH_TX_BD_FLAGS_IP_CSUM (0x1<<1)
#define ETH_TX_BD_FLAGS_IP_CSUM_SHIFT 1
#define ETH_TX_BD_FLAGS_TCP_CSUM (0x1<<2)
#define ETH_TX_BD_FLAGS_TCP_CSUM_SHIFT 2
#define ETH_TX_BD_FLAGS_END_BD (0x1<<3)
#define ETH_TX_BD_FLAGS_END_BD_SHIFT 3
#define ETH_TX_BD_FLAGS_START_BD (0x1<<4)
#define ETH_TX_BD_FLAGS_START_BD_SHIFT 4
#define ETH_TX_BD_FLAGS_HDR_POOL (0x1<<5)
#define ETH_TX_BD_FLAGS_HDR_POOL_SHIFT 5
#define ETH_TX_BD_FLAGS_SW_LSO (0x1<<6)
#define ETH_TX_BD_FLAGS_SW_LSO_SHIFT 6
#define ETH_TX_BD_FLAGS_IPV6 (0x1<<7)
#define ETH_TX_BD_FLAGS_IPV6_SHIFT 7
};

/*
 * The eth Tx Buffer Descriptor
 */
struct eth_tx_bd {
   uint32_t addr_lo;
   uint32_t addr_hi;
   uint16_t nbd;
   uint16_t nbytes;
   uint16_t vlan;
   struct eth_tx_bd_flags bd_flags;
   uint8_t general_data;
#define ETH_TX_BD_HDR_NBDS (0x3F<<0)
#define ETH_TX_BD_HDR_NBDS_SHIFT 0
#define ETH_TX_BD_ETH_ADDR_TYPE (0x3<<6)
#define ETH_TX_BD_ETH_ADDR_TYPE_SHIFT 6
};

/*
 * Tx parsing BD structure for ETH,Relevant in START
 */
struct eth_tx_parse_bd {
   uint8_t global_data;
#define ETH_TX_PARSE_BD_IP_HDR_START_OFFSET (0xF<<0)
#define ETH_TX_PARSE_BD_IP_HDR_START_OFFSET_SHIFT 0
#define ETH_TX_PARSE_BD_CS_ANY_FLG (0x1<<4)
#define ETH_TX_PARSE_BD_CS_ANY_FLG_SHIFT 4
#define ETH_TX_PARSE_BD_PSEUDO_CS_WITHOUT_LEN (0x1<<5)
#define ETH_TX_PARSE_BD_PSEUDO_CS_WITHOUT_LEN_SHIFT 5
```

```
#define ETH_TX_PARSE_BD_LLC_SNAP_EN (0x1<<6)
#define ETH_TX_PARSE_BD_LLC_SNAP_EN_SHIFT 6
#define ETH_TX_PARSE_BD_NS_FLG (0x1<<7)
#define ETH_TX_PARSE_BD_NS_FLG_SHIFT 7
    uint8_t tcp_flags;
#define ETH_TX_PARSE_BD_FIN_FLG (0x1<<0)
#define ETH_TX_PARSE_BD_FIN_FLG_SHIFT 0
#define ETH_TX_PARSE_BD_SYN_FLG (0x1<<1)
#define ETH_TX_PARSE_BD_SYN_FLG_SHIFT 1
#define ETH_TX_PARSE_BD_RST_FLG (0x1<<2)
#define ETH_TX_PARSE_BD_RST_FLG_SHIFT 2
#define ETH_TX_PARSE_BD_PSH_FLG (0x1<<3)
#define ETH_TX_PARSE_BD_PSH_FLG_SHIFT 3
#define ETH_TX_PARSE_BD_ACK_FLG (0x1<<4)
#define ETH_TX_PARSE_BD_ACK_FLG_SHIFT 4
#define ETH_TX_PARSE_BD_URG_FLG (0x1<<5)
#define ETH_TX_PARSE_BD_URG_FLG_SHIFT 5
#define ETH_TX_PARSE_BD_ECE_FLG (0x1<<6)
#define ETH_TX_PARSE_BD_ECE_FLG_SHIFT 6
#define ETH_TX_PARSE_BD_CWR_FLG (0x1<<7)
#define ETH_TX_PARSE_BD_CWR_FLG_SHIFT 7
    uint8_t ip_hlen;
    uint8_t cs_offset;
    uint16_t total_hlen;
    uint16_t lso_mss;
    uint16_t tcp_pseudo_csum;
    uint16_t ip_id;
    uint32_t tcp_send_seq;
};

/*
 * The last BD in the BD memory will hold a pointer to the next BD memory
 */
struct eth_tx_next_bd {
    uint32_t addr_lo;
    uint32_t addr_hi;
    uint8_t reserved[8];
};

/*
 * union for 3 Bd types
 */
union eth_tx_bd_types {
    struct eth_tx_bd reg_bd;
    struct eth_tx_parse_bd parse_bd;
    struct eth_tx_next_bd next_bd;
};

/*
 * The eth storm context of Xstorm
 */
struct xstorm_eth_st_context {
    uint32_t tx_bd_page_base_lo;
    uint32_t tx_bd_page_base_hi;
#if defined(__BIG_ENDIAN)
    uint16_t tx_bd_cons;
```

```
    uint8_t statistics_data;
#define XSTORM_ETH_ST_CONTEXT_STATISTICS_COUNTER_ID (0x7F<<0)
#define XSTORM_ETH_ST_CONTEXT_STATISTICS_COUNTER_ID_SHIFT 0
#define XSTORM_ETH_ST_CONTEXT_STATISTICS_ENABLE (0x1<<7)
#define XSTORM_ETH_ST_CONTEXT_STATISTICS_ENABLE_SHIFT 7
    uint8_t __local_tx_bd_prod;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __local_tx_bd_prod;
    uint8_t statistics_data;
#define XSTORM_ETH_ST_CONTEXT_STATISTICS_COUNTER_ID (0x7F<<0)
#define XSTORM_ETH_ST_CONTEXT_STATISTICS_COUNTER_ID_SHIFT 0
#define XSTORM_ETH_ST_CONTEXT_STATISTICS_ENABLE (0x1<<7)
#define XSTORM_ETH_ST_CONTEXT_STATISTICS_ENABLE_SHIFT 7
    uint16_t tx_bd_cons;
#endif
    uint32_t db_data_addr_lo;
    uint32_t db_data_addr_hi;
    uint32_t __pkt_cons;
    uint32_t __gso_next;
#if defined(__BIG_ENDIAN)
    uint8_t __reserved1;
    uint8_t safc_group_num;
    uint8_t safc_group_en;
    uint8_t __is_eth_conn;
#elif defined(__LITTLE_ENDIAN)
    uint8_t __is_eth_conn;
    uint8_t safc_group_en;
    uint8_t safc_group_num;
    uint8_t __reserved1;
#endif
    union eth_tx_bd_types __bds[13];
};

/*
 * The eth storm context of Cstorm
 */
struct cstorm_eth_st_context {
#if defined(__BIG_ENDIAN)
    uint16_t __reserved0;
    uint8_t sb_index_number;
    uint8_t status_block_id;
#elif defined(__LITTLE_ENDIAN)
    uint8_t status_block_id;
    uint8_t sb_index_number;
    uint16_t __reserved0;
#endif
    uint32_t __reserved1[3];
};

/*
 * Ethernet connection context
 */
struct eth_context {
    struct ustorm_eth_st_context ustorm_st_context;
    struct tstorm_eth_st_context tstorm_st_context;
    struct xstorm_eth_ag_context xstorm_ag_context;
```

```
    struct tstorm_eth_ag_context tstorm_ag_context;
    struct cstorm_eth_ag_context cstorm_ag_context;
    struct ustorm_eth_ag_context ustorm_ag_context;
    struct timers_block_context timers_context;
    struct xstorm_eth_st_context xstorm_st_context;
    struct cstorm_eth_st_context cstorm_st_context;
};


/*
 * Ethernet doorbell
 */
struct eth_tx_doorbell {
#if defined(__BIG_ENDIAN)
    uint16_t npackets;
    uint8_t params;
#define ETH_TX_DOORBELL_NUM_BDS (0x3F<<0)
#define ETH_TX_DOORBELL_NUM_BDS_SHIFT 0
#define ETH_TX_DOORBELL_RESERVED_TX_FIN_FLAG (0x1<<6)
#define ETH_TX_DOORBELL_RESERVED_TX_FIN_FLAG_SHIFT 6
#define ETH_TX_DOORBELL_SPARE (0x1<<7)
#define ETH_TX_DOORBELL_SPARE_SHIFT 7
    struct doorbell_hdr hdr;
#elif defined(__LITTLE_ENDIAN)
    struct doorbell_hdr hdr;
    uint8_t params;
#define ETH_TX_DOORBELL_NUM_BDS (0x3F<<0)
#define ETH_TX_DOORBELL_NUM_BDS_SHIFT 0
#define ETH_TX_DOORBELL_RESERVED_TX_FIN_FLAG (0x1<<6)
#define ETH_TX_DOORBELL_RESERVED_TX_FIN_FLAG_SHIFT 6
#define ETH_TX_DOORBELL_SPARE (0x1<<7)
#define ETH_TX_DOORBELL_SPARE_SHIFT 7
    uint16_t npackets;
#endif
};


/*
 * ustorm status block
 */
struct ustorm_def_status_block {
    uint16_t index_values[HC_USTORM_DEF_SB_NUM_INDICES];
    uint16_t status_block_index;
    uint8_t func;
    uint8_t status_block_id;
    uint32_t __flags;
};

/*
 * cstorm status block
 */
struct cstorm_def_status_block {
    uint16_t index_values[HC_CSTORM_DEF_SB_NUM_INDICES];
    uint16_t status_block_index;
    uint8_t func;
    uint8_t status_block_id;
```

```
    uint32_t __flags;
};

/*
 * xstorm status block
 */
struct xstorm_def_status_block {
    uint16_t index_values[HC_XSTORM_DEF_SB_NUM_INDICES];
    uint16_t status_block_index;
    uint8_t func;
    uint8_t status_block_id;
    uint32_t __flags;
};

/*
 * tstorm status block
 */
struct tstorm_def_status_block {
    uint16_t index_values[HC_TSTORM_DEF_SB_NUM_INDICES];
    uint16_t status_block_index;
    uint8_t func;
    uint8_t status_block_id;
    uint32_t __flags;
};

/*
 * host status block
 */
struct host_def_status_block {
    struct atten_def_status_block atten_status_block;
    struct ustorm_def_status_block u_def_status_block;
    struct cstorm_def_status_block c_def_status_block;
    struct xstorm_def_status_block x_def_status_block;
    struct tstorm_def_status_block t_def_status_block;
};


/*
 * ustorm status block
 */
struct ustorm_status_block {
    uint16_t index_values[HC_USTORM_SB_NUM_INDICES];
    uint16_t status_block_index;
    uint8_t func;
    uint8_t status_block_id;
    uint32_t __flags;
};

/*
 * cstorm status block
 */
struct cstorm_status_block {
    uint16_t index_values[HC_CSTORM_SB_NUM_INDICES];
    uint16_t status_block_index;
    uint8_t func;
    uint8_t status_block_id;
```

```
    uint32_t __flags;
};

/*
 * host status block
 */
struct host_status_block {
    struct ustorm_status_block u_status_block;
    struct cstorm_status_block c_status_block;
};


/*
 * The data for RSS setup ramrod
 */
struct eth_client_setup_ramrod_data {
    uint32_t client_id;
    uint8_t is_rdma;
    uint8_t is_fcoe;
    uint16_t reserved1;
};


/*
 * L2 dynamic host coalescing init parameters
 */
struct eth_dynamic_hc_config {
    uint32_t threshold[3];
    uint8_t hc_timeout[4];
};


/*
 * regular eth FP CQE parameters struct
 */
struct eth_fast_path_rx_cqe {
    uint8_t type_error_flags;
#define ETH_FAST_PATH_RX_CQE_TYPE (0x1<<0)
#define ETH_FAST_PATH_RX_CQE_TYPE_SHIFT 0
#define ETH_FAST_PATH_RX_CQE_PHY_DECODE_ERR_FLG (0x1<<1)
#define ETH_FAST_PATH_RX_CQE_PHY_DECODE_ERR_FLG_SHIFT 1
#define ETH_FAST_PATH_RX_CQE_IP_BAD_XSUM_FLG (0x1<<2)
#define ETH_FAST_PATH_RX_CQE_IP_BAD_XSUM_FLG_SHIFT 2
#define ETH_FAST_PATH_RX_CQE_L4_BAD_XSUM_FLG (0x1<<3)
#define ETH_FAST_PATH_RX_CQE_L4_BAD_XSUM_FLG_SHIFT 3
#define ETH_FAST_PATH_RX_CQE_START_FLG (0x1<<4)
#define ETH_FAST_PATH_RX_CQE_START_FLG_SHIFT 4
#define ETH_FAST_PATH_RX_CQE_END_FLG (0x1<<5)
#define ETH_FAST_PATH_RX_CQE_END_FLG_SHIFT 5
#define ETH_FAST_PATH_RX_CQE_RESERVED0 (0x3<<6)
#define ETH_FAST_PATH_RX_CQE_RESERVED0_SHIFT 6
    uint8_t status_flags;
#define ETH_FAST_PATH_RX_CQE_RSS_HASH_TYPE (0x7<<0)
#define ETH_FAST_PATH_RX_CQE_RSS_HASH_TYPE_SHIFT 0
#define ETH_FAST_PATH_RX_CQE_RSS_HASH_FLG (0x1<<3)
#define ETH_FAST_PATH_RX_CQE_RSS_HASH_FLG_SHIFT 3
```

```
#define ETH_FAST_PATH_RX_CQE_BROADCAST_FLG (0x1<<4)
#define ETH_FAST_PATH_RX_CQE_BROADCAST_FLG_SHIFT 4
#define ETH_FAST_PATH_RX_CQE_MAC_MATCH_FLG (0x1<<5)
#define ETH_FAST_PATH_RX_CQE_MAC_MATCH_FLG_SHIFT 5
#define ETH_FAST_PATH_RX_CQE_IP_XSUM_NO_VALIDATION_FLG (0x1<<6)
#define ETH_FAST_PATH_RX_CQE_IP_XSUM_NO_VALIDATION_FLG_SHIFT 6
#define ETH_FAST_PATH_RX_CQE_L4_XSUM_NO_VALIDATION_FLG (0x1<<7)
#define ETH_FAST_PATH_RX_CQE_L4_XSUM_NO_VALIDATION_FLG_SHIFT 7
   uint8_t placement_offset;
   uint8_t queue_index;
   uint32_t rss_hash_result;
   uint16_t vlan_tag;
   uint16_t pkt_len;
   uint16_t len_on_bd;
   struct parsing_flags pars_flags;
   uint16_t sgl[8];
};


/*
 * The data for RSS setup ramrod
 */
struct eth_halt_ramrod_data {
   uint32_t client_id;
   uint32_t reserved0;
};


/*
 * The data for statistics query ramrod
 */
struct eth_query_ramrod_data {
#if defined(__BIG_ENDIAN)
   uint8_t reserved0;
   uint8_t collect_port;
   uint16_t drv_counter;
#elif defined(__LITTLE_ENDIAN)
   uint16_t drv_counter;
   uint8_t collect_port;
   uint8_t reserved0;
#endif
   uint32_t ctr_id_vector;
};


/*
 * Place holder for ramrods protocol specific data
 */
struct ramrod_data {
   uint32_t data_lo;
   uint32_t data_hi;
};

/*
 * union for ramrod data for Ethernet protocol (CQE) (force size of 16 bits)
 */
```

```
union eth_ramrod_data {
    struct ramrod_data general;
};


/*
 * Rx Last BD in page (in ETH)
 */
struct eth_rx_bd_next_page {
    uint32_t addr_lo;
    uint32_t addr_hi;
    uint8_t reserved[8];
};


/*
 * Eth Rx Cqe structure- general structure for ramrods
 */
struct common_ramrod_eth_rx_cqe {
    uint8_t ramrod_type;
#define COMMON_RAMROD_ETH_RX_CQE_TYPE (0x1<<0)
#define COMMON_RAMROD_ETH_RX_CQE_TYPE_SHIFT 0
#define COMMON_RAMROD_ETH_RX_CQE_RESERVED0 (0x7F<<1)
#define COMMON_RAMROD_ETH_RX_CQE_RESERVED0_SHIFT 1
    uint8_t conn_type;
    uint16_t reserved1;
    uint32_t conn_and_cmd_data;
#define COMMON_RAMROD_ETH_RX_CQE_CID (0xFFFFFF<<0)
#define COMMON_RAMROD_ETH_RX_CQE_CID_SHIFT 0
#define COMMON_RAMROD_ETH_RX_CQE_CMD_ID (0xFF<<24)
#define COMMON_RAMROD_ETH_RX_CQE_CMD_ID_SHIFT 24
    struct ramrod_data protocol_data;
    uint32_t reserved2[4];
};

/*
 * Rx Last CQE in page (in ETH)
 */
struct eth_rx_cqe_next_page {
    uint32_t addr_lo;
    uint32_t addr_hi;
    uint32_t reserved[6];
};

/*
 * union for all eth rx cqe types (fix their sizes)
 */
union eth_rx_cqe {
    struct eth_fast_path_rx_cqe fast_path_cqe;
    struct common_ramrod_eth_rx_cqe ramrod_cqe;
    struct eth_rx_cqe_next_page next_page_cqe;
};


/*
 * common data for all protocols
```

```
 */
struct spe_hdr {
   uint32_t conn_and_cmd_data;
#define SPE_HDR_CID (0xFFFFFF<<0)
#define SPE_HDR_CID_SHIFT 0
#define SPE_HDR_CMD_ID (0xFF<<24)
#define SPE_HDR_CMD_ID_SHIFT 24
   uint16_t type;
#define SPE_HDR_CONN_TYPE (0xFF<<0)
#define SPE_HDR_CONN_TYPE_SHIFT 0
#define SPE_HDR_COMMON_RAMROD (0xFF<<8)
#define SPE_HDR_COMMON_RAMROD_SHIFT 8
   uint16_t reserved;
};

/*
 * Ethernet slow path element
 */
union eth_specific_data {
   uint8_t protocol_data[8];
   struct regpair mac_config_addr;
   struct eth_client_setup_ramrod_data client_setup_ramrod_data;
   struct eth_halt_ramrod_data halt_ramrod_data;
   struct regpair leading_cqe_addr;
   struct regpair update_data_addr;
   struct eth_query_ramrod_data query_ramrod_data;
};

/*
 * Ethernet slow path element
 */
struct eth_spe {
   struct spe_hdr hdr;
   union eth_specific_data data;
};


/*
 * doorbell data in host memory
 */
struct eth_tx_db_data {
   uint32_t packets_prod;
   uint16_t bds_prod;
   uint16_t reserved;
};


/*
 * Common configuration parameters per function in Tstorm
 */
struct tstorm_eth_function_common_config {
#if defined(__BIG_ENDIAN)
   uint8_t leading_client_id;
   uint8_t rss_result_mask;
   uint16_t config_flags;
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV4_CAPABILITY (0x1<<0)
```

```
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV4_CAPABILITY_SHIFT 0
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV4_TCP_CAPABILITY (0x1<<1)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV4_TCP_CAPABILITY_SHIFT 1
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV6_CAPABILITY (0x1<<2)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV6_CAPABILITY_SHIFT 2
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV6_TCP_CAPABILITY (0x1<<3)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV6_TCP_CAPABILITY_SHIFT 3
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_MODE (0x7<<4)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_MODE_SHIFT 4
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_DEFAULT_ENABLE (0x1<<7)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_DEFAULT_ENABLE_SHIFT 7
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_VLAN_IN_CAM (0x1<<8)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_VLAN_IN_CAM_SHIFT 8
#define __TSTORM_ETH_FUNCTION_COMMON_CONFIG_RESERVED0 (0x7F<<9)
#define __TSTORM_ETH_FUNCTION_COMMON_CONFIG_RESERVED0_SHIFT 9
#elif defined(__LITTLE_ENDIAN)
   uint16_t config_flags;
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV4_CAPABILITY (0x1<<0)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV4_CAPABILITY_SHIFT 0
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV4_TCP_CAPABILITY (0x1<<1)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV4_TCP_CAPABILITY_SHIFT 1
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV6_CAPABILITY (0x1<<2)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV6_CAPABILITY_SHIFT 2
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV6_TCP_CAPABILITY (0x1<<3)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_IPV6_TCP_CAPABILITY_SHIFT 3
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_MODE (0x7<<4)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_RSS_MODE_SHIFT 4
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_DEFAULT_ENABLE (0x1<<7)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_DEFAULT_ENABLE_SHIFT 7
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_VLAN_IN_CAM (0x1<<8)
#define TSTORM_ETH_FUNCTION_COMMON_CONFIG_VLAN_IN_CAM_SHIFT 8
#define __TSTORM_ETH_FUNCTION_COMMON_CONFIG_RESERVED0 (0x7F<<9)
#define __TSTORM_ETH_FUNCTION_COMMON_CONFIG_RESERVED0_SHIFT 9
   uint8_t rss_result_mask;
   uint8_t leading_client_id;
#endif
   uint16_t vlan_id[2];
};

/*
 * parameters for eth update ramrod
 */
struct eth_update_ramrod_data {
   struct tstorm_eth_function_common_config func_config;
   uint8_t indirectionTable[128];
};


/*
 * MAC filtering configuration command header
 */
struct mac_configuration_hdr {
   uint8_t length;
   uint8_t offset;
   uint16_t client_id;
   uint32_t reserved1;
```

```
};

/*
 * MAC address in list for ramrod
 */
struct tstorm_cam_entry {
   uint16_t lsb_mac_addr;
   uint16_t middle_mac_addr;
   uint16_t msb_mac_addr;
   uint16_t flags;
#define TSTORM_CAM_ENTRY_PORT_ID (0x1<<0)
#define TSTORM_CAM_ENTRY_PORT_ID_SHIFT 0
#define TSTORM_CAM_ENTRY_RSRVVAL0 (0x7<<1)
#define TSTORM_CAM_ENTRY_RSRVVAL0_SHIFT 1
#define TSTORM_CAM_ENTRY_RESERVED0 (0xFFF<<4)
#define TSTORM_CAM_ENTRY_RESERVED0_SHIFT 4
};

/*
 * MAC filtering: CAM target table entry
 */
struct tstorm_cam_target_table_entry {
   uint8_t flags;
#define TSTORM_CAM_TARGET_TABLE_ENTRY_BROADCAST (0x1<<0)
#define TSTORM_CAM_TARGET_TABLE_ENTRY_BROADCAST_SHIFT 0
#define TSTORM_CAM_TARGET_TABLE_ENTRY_OVERRIDE_VLAN_REMOVAL (0x1<<1)
#define TSTORM_CAM_TARGET_TABLE_ENTRY_OVERRIDE_VLAN_REMOVAL_SHIFT 1
#define TSTORM_CAM_TARGET_TABLE_ENTRY_ACTION_TYPE (0x1<<2)
#define TSTORM_CAM_TARGET_TABLE_ENTRY_ACTION_TYPE_SHIFT 2
#define TSTORM_CAM_TARGET_TABLE_ENTRY_RDMA_MAC (0x1<<3)
#define TSTORM_CAM_TARGET_TABLE_ENTRY_RDMA_MAC_SHIFT 3
#define TSTORM_CAM_TARGET_TABLE_ENTRY_RESERVED0 (0xF<<4)
#define TSTORM_CAM_TARGET_TABLE_ENTRY_RESERVED0_SHIFT 4
   uint8_t client_id;
   uint16_t vlan_id;
};

/*
 * MAC address in list for ramrod
 */
struct mac_configuration_entry {
   struct tstorm_cam_entry cam_entry;
   struct tstorm_cam_target_table_entry target_table_entry;
};

/*
 * MAC filtering configuration command
 */
struct mac_configuration_cmd {
   struct mac_configuration_hdr hdr;
   struct mac_configuration_entry config_table[64];
};


/*
 * MAC address in list for ramrod
```

```
 */
struct mac_configuration_entry_e1h {
   uint16_t lsb_mac_addr;
   uint16_t middle_mac_addr;
   uint16_t msb_mac_addr;
   uint16_t vlan_id;
   uint16_t e1hov_id;
   uint8_t client_id;
   uint8_t flags;
#define MAC_CONFIGURATION_ENTRY_E1H_PORT (0x1<<0)
#define MAC_CONFIGURATION_ENTRY_E1H_PORT_SHIFT 0
#define MAC_CONFIGURATION_ENTRY_E1H_ACTION_TYPE (0x1<<1)
#define MAC_CONFIGURATION_ENTRY_E1H_ACTION_TYPE_SHIFT 1
#define MAC_CONFIGURATION_ENTRY_E1H_RDMA_MAC (0x1<<2)
#define MAC_CONFIGURATION_ENTRY_E1H_RDMA_MAC_SHIFT 2
#define MAC_CONFIGURATION_ENTRY_E1H_RESERVED0 (0x1F<<3)
#define MAC_CONFIGURATION_ENTRY_E1H_RESERVED0_SHIFT 3
};

/*
 * MAC filtering configuration command
 */
struct mac_configuration_cmd_e1h {
   struct mac_configuration_hdr hdr;
   struct mac_configuration_entry_e1h config_table[32];
};


/*
 * approximate-match multicast filtering for E1H per function in Tstorm
 */
struct tstorm_eth_approximate_match_multicast_filtering {
   uint32_t mcast_add_hash_bit_array[8];
};


/*
 * Configuration parameters per client in Tstorm
 */
struct tstorm_eth_client_config {
#if defined(__BIG_ENDIAN)
   uint8_t max_sges_for_packet;
   uint8_t statistics_counter_id;
   uint16_t mtu;
#elif defined(__LITTLE_ENDIAN)
   uint16_t mtu;
   uint8_t statistics_counter_id;
   uint8_t max_sges_for_packet;
#endif
#if defined(__BIG_ENDIAN)
   uint16_t drop_flags;
#define TSTORM_ETH_CLIENT_CONFIG_DROP_IP_CS_ERR (0x1<<0)
#define TSTORM_ETH_CLIENT_CONFIG_DROP_IP_CS_ERR_SHIFT 0
#define TSTORM_ETH_CLIENT_CONFIG_DROP_TCP_CS_ERR (0x1<<1)
#define TSTORM_ETH_CLIENT_CONFIG_DROP_TCP_CS_ERR_SHIFT 1
#define TSTORM_ETH_CLIENT_CONFIG_DROP_TTL0 (0x1<<2)
```

```
#define TSTORM_ETH_CLIENT_CONFIG_DROP_TTL0_SHIFT 2
#define TSTORM_ETH_CLIENT_CONFIG_DROP_UDP_CS_ERR (0x1<<3)
#define TSTORM_ETH_CLIENT_CONFIG_DROP_UDP_CS_ERR_SHIFT 3
#define __TSTORM_ETH_CLIENT_CONFIG_RESERVED1 (0xFFF<<4)
#define __TSTORM_ETH_CLIENT_CONFIG_RESERVED1_SHIFT 4
   uint16_t config_flags;
#define TSTORM_ETH_CLIENT_CONFIG_VLAN_REMOVAL_ENABLE (0x1<<0)
#define TSTORM_ETH_CLIENT_CONFIG_VLAN_REMOVAL_ENABLE_SHIFT 0
#define TSTORM_ETH_CLIENT_CONFIG_E1HOV_REMOVAL_ENABLE (0x1<<1)
#define TSTORM_ETH_CLIENT_CONFIG_E1HOV_REMOVAL_ENABLE_SHIFT 1
#define TSTORM_ETH_CLIENT_CONFIG_STATSITICS_ENABLE (0x1<<2)
#define TSTORM_ETH_CLIENT_CONFIG_STATSITICS_ENABLE_SHIFT 2
#define TSTORM_ETH_CLIENT_CONFIG_ENABLE_SGE_RING (0x1<<3)
#define TSTORM_ETH_CLIENT_CONFIG_ENABLE_SGE_RING_SHIFT 3
#define __TSTORM_ETH_CLIENT_CONFIG_RESERVED0 (0xFFF<<4)
#define __TSTORM_ETH_CLIENT_CONFIG_RESERVED0_SHIFT 4
#elif defined(__LITTLE_ENDIAN)
   uint16_t config_flags;
#define TSTORM_ETH_CLIENT_CONFIG_VLAN_REMOVAL_ENABLE (0x1<<0)
#define TSTORM_ETH_CLIENT_CONFIG_VLAN_REMOVAL_ENABLE_SHIFT 0
#define TSTORM_ETH_CLIENT_CONFIG_E1HOV_REMOVAL_ENABLE (0x1<<1)
#define TSTORM_ETH_CLIENT_CONFIG_E1HOV_REMOVAL_ENABLE_SHIFT 1
#define TSTORM_ETH_CLIENT_CONFIG_STATSITICS_ENABLE (0x1<<2)
#define TSTORM_ETH_CLIENT_CONFIG_STATSITICS_ENABLE_SHIFT 2
#define TSTORM_ETH_CLIENT_CONFIG_ENABLE_SGE_RING (0x1<<3)
#define TSTORM_ETH_CLIENT_CONFIG_ENABLE_SGE_RING_SHIFT 3
#define __TSTORM_ETH_CLIENT_CONFIG_RESERVED0 (0xFFF<<4)
#define __TSTORM_ETH_CLIENT_CONFIG_RESERVED0_SHIFT 4
   uint16_t drop_flags;
#define TSTORM_ETH_CLIENT_CONFIG_DROP_IP_CS_ERR (0x1<<0)
#define TSTORM_ETH_CLIENT_CONFIG_DROP_IP_CS_ERR_SHIFT 0
#define TSTORM_ETH_CLIENT_CONFIG_DROP_TCP_CS_ERR (0x1<<1)
#define TSTORM_ETH_CLIENT_CONFIG_DROP_TCP_CS_ERR_SHIFT 1
#define TSTORM_ETH_CLIENT_CONFIG_DROP_TTL0 (0x1<<2)
#define TSTORM_ETH_CLIENT_CONFIG_DROP_TTL0_SHIFT 2
#define TSTORM_ETH_CLIENT_CONFIG_DROP_UDP_CS_ERR (0x1<<3)
#define TSTORM_ETH_CLIENT_CONFIG_DROP_UDP_CS_ERR_SHIFT 3
#define __TSTORM_ETH_CLIENT_CONFIG_RESERVED1 (0xFFF<<4)
#define __TSTORM_ETH_CLIENT_CONFIG_RESERVED1_SHIFT 4
#endif
};


/*
 * MAC filtering configuration parameters per port in Tstorm
 */
struct tstorm_eth_mac_filter_config {
   uint32_t ucast_drop_all;
   uint32_t ucast_accept_all;
   uint32_t mcast_drop_all;
   uint32_t mcast_accept_all;
   uint32_t bcast_drop_all;
   uint32_t bcast_accept_all;
   uint32_t strict_vlan;
   uint32_t vlan_filter[2];
   uint32_t reserved;
```

```
};


/*
 * common flag to indicate existance of TPA.
 */
struct tstorm_eth_tpa_exist {
#if defined(__BIG_ENDIAN)
    uint16_t reserved1;
    uint8_t reserved0;
    uint8_t tpa_exist;
#elif defined(__LITTLE_ENDIAN)
    uint8_t tpa_exist;
    uint8_t reserved0;
    uint16_t reserved1;
#endif
    uint32_t reserved2;
};


/*
 * rx rings pause data for E1h only
 */
struct ustorm_eth_rx_pause_data_e1h {
#if defined(__BIG_ENDIAN)
    uint16_t bd_thr_low;
    uint16_t cqe_thr_low;
#elif defined(__LITTLE_ENDIAN)
    uint16_t cqe_thr_low;
    uint16_t bd_thr_low;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t cos;
    uint16_t sge_thr_low;
#elif defined(__LITTLE_ENDIAN)
    uint16_t sge_thr_low;
    uint16_t cos;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t bd_thr_high;
    uint16_t cqe_thr_high;
#elif defined(__LITTLE_ENDIAN)
    uint16_t cqe_thr_high;
    uint16_t bd_thr_high;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t reserved0;
    uint16_t sge_thr_high;
#elif defined(__LITTLE_ENDIAN)
    uint16_t sge_thr_high;
    uint16_t reserved0;
#endif
};


/*
```

```
 * Three RX producers for ETH
 */
struct ustorm_eth_rx_producers {
#if defined(__BIG_ENDIAN)
    uint16_t bd_prod;
    uint16_t cqe_prod;
#elif defined(__LITTLE_ENDIAN)
    uint16_t cqe_prod;
    uint16_t bd_prod;
#endif
#if defined(__BIG_ENDIAN)
    uint16_t reserved;
    uint16_t sge_prod;
#elif defined(__LITTLE_ENDIAN)
    uint16_t sge_prod;
    uint16_t reserved;
#endif
};


/*
 * per-port SAFC demo variables
 */
struct cmng_flags_per_port {
    uint8_t con_number[NUM_OF_PROTOCOLS];
#if defined(__BIG_ENDIAN)
    uint8_t fairness_enable;
    uint8_t rate_shaping_enable;
    uint8_t cmng_protocol_enable;
    uint8_t cmng_vn_enable;
#elif defined(__LITTLE_ENDIAN)
    uint8_t cmng_vn_enable;
    uint8_t cmng_protocol_enable;
    uint8_t rate_shaping_enable;
    uint8_t fairness_enable;
#endif
};


/*
 * per-port rate shaping variables
 */
struct rate_shaping_vars_per_port {
    uint32_t rs_periodic_timeout;
    uint32_t rs_threshold;
};


/*
 * per-port fairness variables
 */
struct fairness_vars_per_port {
    uint32_t upper_bound;
    uint32_t fair_threshold;
    uint32_t fairness_timeout;
};
```

```
/*
 * per-port SAFC variables
 */
struct safc_struct_per_port {
#if defined(__BIG_ENDIAN)
    uint16_t __reserved1;
    uint8_t __reserved0;
    uint8_t safc_timeout_usec;
#elif defined(__LITTLE_ENDIAN)
    uint8_t safc_timeout_usec;
    uint8_t __reserved0;
    uint16_t __reserved1;
#endif
    uint16_t cos_to_pause_mask[MAX_COS_NUMBER];
};


/*
 * Per-port congestion management variables
 */
struct cmng_struct_per_port {
    struct rate_shaping_vars_per_port rs_vars;
    struct fairness_vars_per_port fair_vars;
    struct safc_struct_per_port safc_vars;
    struct cmng_flags_per_port flags;
};


/*
 * Protocol-common statistics collected by the Xstorm (per client)
 */
struct xstorm_per_client_stats {
    struct regpair total_sent_bytes;
    uint32_t total_sent_pkts;
    uint32_t unicast_pkts_sent;
    struct regpair unicast_bytes_sent;
    struct regpair multicast_bytes_sent;
    uint32_t multicast_pkts_sent;
    uint32_t broadcast_pkts_sent;
    struct regpair broadcast_bytes_sent;
    uint16_t stats_counter;
    uint16_t reserved0;
    uint32_t reserved1;
};


/*
 * Common statistics collected by the Xstorm (per port)
 */
struct xstorm_common_stats {
 struct xstorm_per_client_stats client_statistics[MAX_X_STAT_COUNTER_ID];
};
```

```
/*
 * Protocol-common statistics collected by the Tstorm (per port)
 */
struct tstorm_per_port_stats {
   uint32_t mac_filter_discard;
   uint32_t xxoverflow_discard;
   uint32_t brb_truncate_discard;
   uint32_t mac_discard;
};


/*
 * Protocol-common statistics collected by the Tstorm (per client)
 */
struct tstorm_per_client_stats {
   struct regpair total_rcv_bytes;
   struct regpair rcv_unicast_bytes;
   struct regpair rcv_broadcast_bytes;
   struct regpair rcv_multicast_bytes;
   struct regpair rcv_error_bytes;
   uint32_t checksum_discard;
   uint32_t packets_too_big_discard;
   uint32_t total_rcv_pkts;
   uint32_t rcv_unicast_pkts;
   uint32_t rcv_broadcast_pkts;
   uint32_t rcv_multicast_pkts;
   uint32_t no_buff_discard;
   uint32_t ttl0_discard;
   uint16_t stats_counter;
   uint16_t reserved0;
   uint32_t reserved1;
};

/*
 * Protocol-common statistics collected by the Tstorm
 */
struct tstorm_common_stats {
   struct tstorm_per_port_stats port_statistics;
 struct tstorm_per_client_stats client_statistics[MAX_T_STAT_COUNTER_ID];
};

/*
 * Protocol-common statistics collected by the Ustorm (per client)
 */
struct ustorm_per_client_stats {
   struct regpair rcv_error_bytes;
   uint32_t no_buff_discard;
   uint16_t stats_counter;
   uint16_t reserved0;
};

/*
 * Protocol-common statistics collected by the Ustorm
 */
struct ustorm_common_stats {
 struct ustorm_per_client_stats client_statistics[MAX_U_STAT_COUNTER_ID];
```

```
};

/*
 * Eth statistics query structure for the eth_stats_query ramrod
 */
struct eth_stats_query {
    struct xstorm_common_stats xstorm_common;
    struct tstorm_common_stats tstorm_common;
    struct ustorm_common_stats ustorm_common;
};


/*
 * per-vnic fairness variables
 */
struct fairness_vars_per_vn {
    uint32_t protocol_credit_delta[NUM_OF_PROTOCOLS];
    uint32_t vn_credit_delta;
    uint32_t __reserved0;
};


/*
 * FW version stored in the Xstorm RAM
 */
struct fw_version {
#if defined(__BIG_ENDIAN)
    uint8_t engineering;
    uint8_t revision;
    uint8_t minor;
    uint8_t major;
#elif defined(__LITTLE_ENDIAN)
    uint8_t major;
    uint8_t minor;
    uint8_t revision;
    uint8_t engineering;
#endif
    uint32_t flags;
#define FW_VERSION_OPTIMIZED (0x1<<0)
#define FW_VERSION_OPTIMIZED_SHIFT 0
#define FW_VERSION_BIG_ENDIEN (0x1<<1)
#define FW_VERSION_BIG_ENDIEN_SHIFT 1
#define FW_VERSION_CHIP_VERSION (0x3<<2)
#define FW_VERSION_CHIP_VERSION_SHIFT 2
#define __FW_VERSION_RESERVED (0xFFFFFFFF<<4)
#define __FW_VERSION_RESERVED_SHIFT 4
};


/*
 * FW version stored in first line of pram
 */
struct pram_fw_version {
    uint8_t major;
    uint8_t minor;
    uint8_t revision;
```

```
    uint8_t engineering;
    uint8_t flags;
#define PRAM_FW_VERSION_OPTIMIZED (0x1<<0)
#define PRAM_FW_VERSION_OPTIMIZED_SHIFT 0
#define PRAM_FW_VERSION_STORM_ID (0x3<<1)
#define PRAM_FW_VERSION_STORM_ID_SHIFT 1
#define PRAM_FW_VERSION_BIG_ENDIEN (0x1<<3)
#define PRAM_FW_VERSION_BIG_ENDIEN_SHIFT 3
#define PRAM_FW_VERSION_CHIP_VERSION (0x3<<4)
#define PRAM_FW_VERSION_CHIP_VERSION_SHIFT 4
#define __PRAM_FW_VERSION_RESERVED0 (0x3<<6)
#define __PRAM_FW_VERSION_RESERVED0_SHIFT 6
};


/*
 * a single rate shaping counter. can be used as protocol or vnic counter
 */
struct rate_shaping_counter {
    uint32_t quota;
#if defined(__BIG_ENDIAN)
    uint16_t __reserved0;
    uint16_t rate;
#elif defined(__LITTLE_ENDIAN)
    uint16_t rate;
    uint16_t __reserved0;
#endif
};


/*
 * per-vnic rate shaping variables
 */
struct rate_shaping_vars_per_vn {
    struct rate_shaping_counter protocol_counters[NUM_OF_PROTOCOLS];
    struct rate_shaping_counter vn_counter;
};

/*
 * The send queue element
 */
struct slow_path_element {
    struct spe_hdr hdr;
    uint8_t protocol_data[8];
};


/*
 * eth/toe flags that indicate if to query
 */
struct stats_indication_flags {
    uint32_t collect_eth;
    uint32_t collect_toe;
};

/* ToDo: Are these still needed? */
```

```
typedef struct drv_fw_mb          drv_fw_mb_t;
typedef struct shared_hw_cfg      shared_hw_cfg_t;
typedef struct port_hw_cfg        port_hw_cfg_t;
typedef struct shared_feat_cfg    shared_feat_cfg_t;
typedef struct port_feat_cfg      port_feat_cfg_t;
typedef struct dev_info           dev_info_t;
typedef struct mgmtfw_state       mgmtfw_state_t;
typedef struct shmem_region       shmem_region_t;
```

Appendix B: Programming the Non-Volatile Memory

Access to the non-volatile memory interface is controlled through internal configuration, command, and status registers in the NVRAM register block. The NVRAM can be accessed with automated 32-bit read and write commands or configured for bit-bang operation through the NVM control registers. A semaphore register (MCP_REG_MCPR_NVM_SW_ARB) allows up to four software entities to share access to the NVRAM device.

Note: Request level 0 is the highest priority arbitration request level and is reserved for BCM57710/BCM57711C firmware/bootcode.

NVRAM Access Example Code

The C code given below uses the MODE_256 (MCP_REG_MCPR_NVM_COMMAND MODE_256 bit set to 1) for accessing the Flash devices. So no logical to physical address mapping or vice versa is necessary. lm_device_t is a driver structure for storing and sharing the device specific information across driver functions.

```
#define LM_STATUS_SUCCESS                   0
#define LM_STATUS_FAILURE                   1
#define LM_STATUS_BUSY                      18

/* NVRAM flags for nvram_write_dword and nvram_read_dword. */
#define NVRAM_FLAG_NONE                         0x00
#define NVRAM_FLAG_SET_FIRST_CMD_BIT            0x01
#define NVRAM_FLAG_SET_LAST_CMD_BIT             0x02
#define NVRAM_FLAG_BUFFERED_FLASH               0x04

#define GRCBASE_MCP   0x080000

#define MCP_REG_MCPR_NVM_COMMAND                            0x6400
        #define MCPR_NVM_COMMAND_RST                        (1L<<0)
        #define MCPR_NVM_COMMAND_RST_BITSHIFT               0
        #define MCPR_NVM_COMMAND_DONE                       (1L<<3)
        #define MCPR_NVM_COMMAND_DONE_BITSHIFT              3
        #define MCPR_NVM_COMMAND_DOIT                       (1L<<4)
        #define MCPR_NVM_COMMAND_DOIT_BITSHIFT              4
        #define MCPR_NVM_COMMAND_WR                         (1L<<5)
        #define MCPR_NVM_COMMAND_WR_BITSHIFT                5
        #define MCPR_NVM_COMMAND_ERASE                      (1L<<6)
        #define MCPR_NVM_COMMAND_ERASE_BITSHIFT             6
        #define MCPR_NVM_COMMAND_FIRST                      (1L<<7)
        #define MCPR_NVM_COMMAND_FIRST_BITSHIFT             7
        #define MCPR_NVM_COMMAND_LAST                       (1L<<8)
        #define MCPR_NVM_COMMAND_LAST_BITSHIFT              8
        #define MCPR_NVM_COMMAND_WREN                       (1L<<16)
        #define MCPR_NVM_COMMAND_WREN_BITSHIFT              16
        #define MCPR_NVM_COMMAND_WRDI                       (1L<<17)
        #define MCPR_NVM_COMMAND_WRDI_BITSHIFT              17
        #define MCPR_NVM_COMMAND_RD_ID                      (1L<<20)
        #define MCPR_NVM_COMMAND_RD_ID_BITSHIFT             20
        #define MCPR_NVM_COMMAND_RD_STATUS                  (1L<<21)
```

```
        #define MCPR_NVM_COMMAND_RD_STATUS_BITSHIFT         21
        #define MCPR_NVM_COMMAND_MODE_256                   (1L<<22)
        #define MCPR_NVM_COMMAND_MODE_256_BITSHIFT          22

#define MCP_REG_MCPR_NVM_STATUS                             0x6404
        #define MCPR_NVM_STATUS_SPI_FSM_STATE               (0x1fL<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_BITSHIFT      0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_IDLE    (0L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_IDLE_BITSHIFT   0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CMD0    (1L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CMD0_BITSHIFT   0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CMD1    (2L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CMD1_BITSHIFT   0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CMD_FINISH0   (3L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CMD_FINISH0_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CMD_FINISH1    (4L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CMD_FINISH1_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_ADDR0    (5L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_ADDR0_BITSHIFT   0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_WRITE_DATA0    (6L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_WRITE_DATA0_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_WRITE_DATA1    (7L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_WRITE_DATA1_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_WRITE_DATA2    (8L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_WRITE_DATA2_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_DATA0  (9L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_DATA0_BITSHIFT0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_DATA1  (10L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_DATA1_BITSHIFT   0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_DATA2  (11L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_DATA2_BITSHIFT   0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID0    (12L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID0_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID1    (13L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID1_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID2    (14L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID2_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID3    (15L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID3_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID4    (16L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_READ_STATUS_RDID4_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CHECK_BUSY0    (17L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_CHECK_BUSY0_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_ST_WREN    (18L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_ST_WREN_BITSHIFT 0
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_WAIT    (19L<<0)
        #define MCPR_NVM_STATUS_SPI_FSM_STATE_SPI_WAIT_BITSHIFT   0

#define MCP_REG_MCPR_NVM_WRITE                              0x6408
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE              (0xffffffffL<<0)
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_BITSHIFT     0
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_BIT_BANG  (0L<<0)
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_BIT_BANG_BITSHIFT 0
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_SI       (1L<<0)
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_SI_BITSHIFT 0
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_SO       (2L<<0)
```

```
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_SO_BITSHIFT 0
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_CS_B     (4L<<0)
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_CS_B_BITSHIFT 0
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_SCLK     (8L<<0)
        #define MCPR_NVM_WRITE_NVM_WRITE_VALUE_SCLK_BITSHIFT 0

#define MCP_REG_MCPR_NVM_ADDR                                   0x640c
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE                    (0xffffffL<<0)
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_BITSHIFT           0
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_BIT_BANG   (0L<<0)
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_BIT_BANG_BITSHIFT 0
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_SI         (1L<<0)
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_SI_BITSHIFT 0
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_SO         (2L<<0)
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_SO_BITSHIFT 0
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_CS_B       (4L<<0)
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_CS_B_BITSHIFT 0
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_SCLK       (8L<<0)
        #define MCPR_NVM_ADDR_NVM_ADDR_VALUE_SCLK_BITSHIFT 0

#define MCP_REG_MCPR_NVM_READ                                   0x6410
        #define MCPR_NVM_READ_NVM_READ_VALUE                    (0xffffffffL<<0)
        #define MCPR_NVM_READ_NVM_READ_VALUE_BITSHIFT           0
        #define MCPR_NVM_READ_NVM_READ_VALUE_BIT_BANG   (0L<<0)
        #define MCPR_NVM_READ_NVM_READ_VALUE_BIT_BANG_BITSHIFT 0
        #define MCPR_NVM_READ_NVM_READ_VALUE_SI         (1L<<0)
        #define MCPR_NVM_READ_NVM_READ_VALUE_SI_BITSHIFT 0
        #define MCPR_NVM_READ_NVM_READ_VALUE_SO         (2L<<0)
        #define MCPR_NVM_READ_NVM_READ_VALUE_SO_BITSHIFT 0
        #define MCPR_NVM_READ_NVM_READ_VALUE_CS_B       (4L<<0)
        #define MCPR_NVM_READ_NVM_READ_VALUE_CS_B_BITSHIFT 0
        #define MCPR_NVM_READ_NVM_READ_VALUE_SCLK       (8L<<0)
        #define MCPR_NVM_READ_NVM_READ_VALUE_SCLK_BITSHIFT 0

#define MCP_REG_MCPR_NVM_CFG1                                   0x6414
        #define MCPR_NVM_CFG1_FLASH_MODE                        (1L<<0)
        #define MCPR_NVM_CFG1_FLASH_MODE_BITSHIFT               0
        #define MCPR_NVM_CFG1_BUFFER_MODE                       (1L<<1)
        #define MCPR_NVM_CFG1_BUFFER_MODE_BITSHIFT              1
        #define MCPR_NVM_CFG1_PASS_MODE                         (1L<<2)
        #define MCPR_NVM_CFG1_PASS_MODE_BITSHIFT                2
        #define MCPR_NVM_CFG1_BITBANG_MODE                      (1L<<3)
        #define MCPR_NVM_CFG1_BITBANG_MODE_BITSHIFT             3
        #define MCPR_NVM_CFG1_STATUS_BIT                        (0x7L<<4)
        #define MCPR_NVM_CFG1_STATUS_BIT_BITSHIFT               4
        #define MCPR_NVM_CFG1_SPI_CLK_DIV                       (0xfL<<7)
        #define MCPR_NVM_CFG1_SPI_CLK_DIV_BITSHIFT              7
        #define MCPR_NVM_CFG1_SEE_CLK_DIV                       (0x7ffL<<11)
        #define MCPR_NVM_CFG1_SEE_CLK_DIV_BITSHIFT              11
        #define MCPR_NVM_CFG1_STRAP_CONTROL_0                   (1L<<23)
        #define MCPR_NVM_CFG1_STRAP_CONTROL_0_BITSHIFT          23
        #define MCPR_NVM_CFG1_PROTECT_MODE                      (1L<<24)
        #define MCPR_NVM_CFG1_PROTECT_MODE_BITSHIFT             24
        #define MCPR_NVM_CFG1_FLASH_SIZE                        (1L<<25)
        #define MCPR_NVM_CFG1_FLASH_SIZE_BITSHIFT               25
        #define MCPR_NVM_CFG1_FW_USTRAP_1                       (1L<<26)
```

```
        #define MCPR_NVM_CFG1_FW_USTRAP_1_BITSHIFT          26
        #define MCPR_NVM_CFG1_FW_USTRAP_0                   (1L<<27)
        #define MCPR_NVM_CFG1_FW_USTRAP_0_BITSHIFT          27
        #define MCPR_NVM_CFG1_FW_USTRAP_2                   (1L<<28)
        #define MCPR_NVM_CFG1_FW_USTRAP_2_BITSHIFT          28
        #define MCPR_NVM_CFG1_FW_USTRAP_3                   (1L<<29)
        #define MCPR_NVM_CFG1_FW_USTRAP_3_BITSHIFT          29
        #define MCPR_NVM_CFG1_FW_FLASH_TYPE_EN              (1L<<30)
        #define MCPR_NVM_CFG1_FW_FLASH_TYPE_EN_BITSHIFT     30
        #define MCPR_NVM_CFG1_COMPAT_BYPASSS                (1L<<31)
        #define MCPR_NVM_CFG1_COMPAT_BYPASSS_BITSHIFT       31

#define MCP_REG_MCPR_NVM_CFG2                               0x6418
        #define MCPR_NVM_CFG2_ERASE_CMD                     (0xffL<<0)
        #define MCPR_NVM_CFG2_ERASE_CMD_BITSHIFT            0
        #define MCPR_NVM_CFG2_STATUS_CMD                    (0xffL<<16)
        #define MCPR_NVM_CFG2_STATUS_CMD_BITSHIFT           16
        #define MCPR_NVM_CFG2_READ_ID                       (0xffL<<24)
        #define MCPR_NVM_CFG2_READ_ID_BITSHIFT              24

#define MCP_REG_MCPR_NVM_CFG3                               0x641c
        #define MCPR_NVM_CFG3_BUFFER_RD_CMD                 (0xffL<<0)
        #define MCPR_NVM_CFG3_BUFFER_RD_CMD_BITSHIFT        0
        #define MCPR_NVM_CFG3_WRITE_CMD                     (0xffL<<8)
        #define MCPR_NVM_CFG3_WRITE_CMD_BITSHIFT            8
        #define MCPR_NVM_CFG3_READ_CMD                      (0xffL<<24)
        #define MCPR_NVM_CFG3_READ_CMD_BITSHIFT             24

#define MCP_REG_MCPR_NVM_SW_ARB                             0x6420
        #define MCPR_NVM_SW_ARB_ARB_REQ_SET0                (1L<<0)
        #define MCPR_NVM_SW_ARB_ARB_REQ_SET0_BITSHIFT       0
        #define MCPR_NVM_SW_ARB_ARB_REQ_SET1                (1L<<1)
        #define MCPR_NVM_SW_ARB_ARB_REQ_SET1_BITSHIFT       1
        #define MCPR_NVM_SW_ARB_ARB_REQ_SET2                (1L<<2)
        #define MCPR_NVM_SW_ARB_ARB_REQ_SET2_BITSHIFT       2
        #define MCPR_NVM_SW_ARB_ARB_REQ_SET3                (1L<<3)
        #define MCPR_NVM_SW_ARB_ARB_REQ_SET3_BITSHIFT       3
        #define MCPR_NVM_SW_ARB_ARB_REQ_CLR0                (1L<<4)
        #define MCPR_NVM_SW_ARB_ARB_REQ_CLR0_BITSHIFT       4
        #define MCPR_NVM_SW_ARB_ARB_REQ_CLR1                (1L<<5)
        #define MCPR_NVM_SW_ARB_ARB_REQ_CLR1_BITSHIFT       5
        #define MCPR_NVM_SW_ARB_ARB_REQ_CLR2                (1L<<6)
        #define MCPR_NVM_SW_ARB_ARB_REQ_CLR2_BITSHIFT       6
        #define MCPR_NVM_SW_ARB_ARB_REQ_CLR3                (1L<<7)
        #define MCPR_NVM_SW_ARB_ARB_REQ_CLR3_BITSHIFT       7
        #define MCPR_NVM_SW_ARB_ARB_ARB0                    (1L<<8)
        #define MCPR_NVM_SW_ARB_ARB_ARB0_BITSHIFT           8
        #define MCPR_NVM_SW_ARB_ARB_ARB1                    (1L<<9)
        #define MCPR_NVM_SW_ARB_ARB_ARB1_BITSHIFT           9
        #define MCPR_NVM_SW_ARB_ARB_ARB2                    (1L<<10)
        #define MCPR_NVM_SW_ARB_ARB_ARB2_BITSHIFT           10
        #define MCPR_NVM_SW_ARB_ARB_ARB3                    (1L<<11)
        #define MCPR_NVM_SW_ARB_ARB_ARB3_BITSHIFT           11
        #define MCPR_NVM_SW_ARB_REQ0                        (1L<<12)
        #define MCPR_NVM_SW_ARB_REQ0_BITSHIFT               12
        #define MCPR_NVM_SW_ARB_REQ1                        (1L<<13)
```

```
        #define MCPR_NVM_SW_ARB_REQ1_BITSHIFT               13
        #define MCPR_NVM_SW_ARB_REQ2                        (1L<<14)
        #define MCPR_NVM_SW_ARB_REQ2_BITSHIFT               14
        #define MCPR_NVM_SW_ARB_REQ3                        (1L<<15)
        #define MCPR_NVM_SW_ARB_REQ3_BITSHIFT               15

#define MCP_REG_MCPR_NVM_ACCESS_ENABLE                      0x6424
        #define MCPR_NVM_ACCESS_ENABLE_EN                   (1L<<0)
        #define MCPR_NVM_ACCESS_ENABLE_EN_BITSHIFT          0
        #define MCPR_NVM_ACCESS_ENABLE_WR_EN                (1L<<1)
        #define MCPR_NVM_ACCESS_ENABLE_WR_EN_BITSHIFT       1

#define NVRAM_TIMEOUT_COUNT   30000

typedef unsigned int  U32;
typedef U32 u32_t;
typedef u32_t lm_status_t;


/****************************************************************************
 * This function is called during the driver initialization and is used to     *
 * determine Flash device type, size, and page_size parameters which are       *
 * required when accessing the Flash device                                    *
 ****************************************************************************/
static lm_status_t lm_get_nvm_info(lm_device_t *pdev)
{
   u32_t val = 0;
   REG_RD(pdev,GRCBASE_MCP,MCP_REG_MCPR_NVM_CFG4,&val);
   pdev->hw_info.flash_spec.total_size = NVRAM_1MB_SIZE << (val &
MCPR_NVM_CFG4_FLASH_SIZE);
   pdev->hw_info.flash_spec.page_size = NVRAM_PAGE_SIZE;
   return LM_STATUS_SUCCESS;
}


/****************************************************************************
 * This function acquires the NVRAM arbitration lock.                          *
 ****************************************************************************/
static lm_status_t acquire_nvram_lock(lm_device_t *pdev)
{
    lm_status_t lm_status;
    u32_t j, cnt;
    u32_t val;

    cnt = NVRAM_TIMEOUT_COUNT;
    val = 0;

    /* Request access to the flash interface. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_SW_ARB, MCPR_NVM_SW_ARB_ARB_REQ_SET1);

    /* Wait in a loop until access is granted or timeout */
    for(j = 0; j < cnt*10; j++)
    {
        REG_RD(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_SW_ARB,&val);
        if(val & MCPR_NVM_SW_ARB_ARB_ARB1)
        {
```

```
            break;
        }

        mm_wait(5);
    }

    if(val & MCPR_NVM_SW_ARB_ARB_ARB1)
    {
        lm_status = LM_STATUS_SUCCESS;
    }
    else
    {
        lm_status = LM_STATUS_BUSY;
    }

    return lm_status;
} /* acquire_nvram_lock */


/********************************************************************************
 * This function releases the NVRAM arbitration lock.                           *
 ********************************************************************************/
static void release_nvram_lock(lm_device_t *pdev)
{
    u32_t j, cnt;
    u32_t val;

    cnt = NVRAM_TIMEOUT_COUNT;
    val = 0;

    /* Relinquish nvram interface. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_SW_ARB, MCPR_NVM_SW_ARB_ARB_REQ_CLR1);

    /* Wait for completion */
    for(j = 0; j < cnt; j++)
    {
     REG_RD(pdev,GRCBASE_MCP, MCP_REG_MCPR_NVM_SW_ARB,&val);
     if(!(val & MCPR_NVM_SW_ARB_ARB_ARB1))
        {
            break;
        }

        mm_wait(5);
    }
} /* release_nvram_lock */


/********************************************************************************
 * Sends Write Enable command on SPI interface                                  *
 ********************************************************************************/
static lm_status_t enable_nvram_write(lm_device_t *pdev)
{
   u32_t val, j, cnt;
   lm_status_t lm_status;

   /* Need to clear DONE bit separately. */
```

```
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, MCPR_NVM_COMMAND_DONE);

    /* Issue a write enable command. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, MCPR_NVM_COMMAND_DOIT |
MCPR_NVM_COMMAND_WREN);

    cnt = NVRAM_TIMEOUT_COUNT;
    lm_status = LM_STATUS_BUSY;

        /* Wait for completion */
    for(j = 0; j < cnt; j++)
    {
              /* OS specific function for a delay of 5 uS. */
        mm_wait(pdev, 5);

        REG_RD(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, &val);
        if(val & MCPR_NVM_COMMAND_DONE)
        {
            lm_status = LM_STATUS_SUCCESS;
            break;
        }
    }
    return lm_status;
} /* enable_nvram_write */


/************************************************************************
 * Sends Write Disable command on SPI interface.                        *
 ************************************************************************/
static lm_status_t disable_nvram_write(lm_device_t *pdev)
{
    lm_status_t lm_status;
    u32_t cnt,j,val;

    /* Need to clear DONE bit separately. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, MCPR_NVM_COMMAND_DONE);

    /* Issue a write enable command. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, MCPR_NVM_COMMAND_DOIT |
MCPR_NVM_COMMAND_WRDI);

    cnt = NVRAM_TIMEOUT_COUNT;
    lm_status = LM_STATUS_BUSY;

    for(j = 0; j < cnt; j++)
    {
              /* OS specific function for a delay of 5 uS. */
        mm_wait(pdev, 5);

        REG_RD(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, &val);
        if(val & MCPR_NVM_COMMAND_DONE)
        {
            lm_status = LM_STATUS_SUCCESS;
            break;
        }
    }
```

```
    return lm_status;
} /* disable_nvram_write */


/****************************************************************************
 * Enable accessing of NVRAM interface registers.                           *
 ****************************************************************************/
static lm_status_t enable_nvram_access(lm_device_t *pdev)
{
    u32_t val;

    REG_RD(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_ACCESS_ENABLE, &val);

    /* Enable both bits, even on read. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_ACCESS_ENABLE, val |
MCPR_NVM_ACCESS_ENABLE_EN | MCPR_NVM_ACCESS_ENABLE_WR_EN);

    return LM_STATUS_SUCCESS;
} /* enable_nvram_access */


/****************************************************************************
 * Disable accessing of NVRAM interface registers.                          *
 ****************************************************************************/
static lm_status_t disable_nvram_access(lm_device_t *pdev)
{
    u32_t val;

    REG_RD(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_ACCESS_ENABLE, &val);

    /* Disable both bits, even after read. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_ACCESS_ENABLE, val &
~(MCPR_NVM_ACCESS_ENABLE_EN | MCPR_NVM_ACCESS_ENABLE_WR_EN));

    return LM_STATUS_SUCCESS;
} /* disable_nvram_access */


/****************************************************************************
 * Read a DWORD from NVRAM. This function should be called only after nvram *
 * lock is acquired and nvram interface register access is enabled          *
 ****************************************************************************/
static lm_status_t nvram_read_dword(lm_device_t *pdev, u32_t offset, u32_t *ret_val, u32_t
nvram_flags)
{
    lm_status_t lm_status;
    u32_t cmd_flags;
    u32_t val;
    u32_t j, cnt;

    /* Build the command word. */
    cmd_flags = nvram_flags | MCPR_NVM_COMMAND_DOIT ;

    /* Need to clear DONE bit separately. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, MCPR_NVM_COMMAND_DONE);
```

```
    /* Address of the NVRAM to read from. */
   REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_ADDR, offset & MCPR_NVM_ADDR_NVM_ADDR_VALUE);

    /* Issue a read command. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, cmd_flags);

    cnt = NVRAM_TIMEOUT_COUNT;

    /* Wait for completion. */
    lm_status = LM_STATUS_BUSY;
    for(j = 0; j < cnt; j++)
    {
        mm_wait(pdev, 5);
     REG_RD(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, &val);
        if(val & MCPR_NVM_COMMAND_DONE)
        {
        REG_RD(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_READ, &val);

            /* Change to little endian if required. */
            #if defined(LITTLE_ENDIAN)
            val = ((val & 0xff) << 24) | ((val & 0xff00) << 8) |
                ((val & 0xff0000) >> 8) | ((val >> 24) & 0xff);
            #endif

            *ret_val = val;

            lm_status = LM_STATUS_SUCCESS;

            break;
        }
    }

    return lm_status;
} /* nvram_read_dword */


/****************************************************************************/
 * Write a DWORD into NVRAM. This function should be called only after nvram   *
 * lock is acquired and nvram register access is enabled                       *
 ****************************************************************************/
static lm_status_t nvram_write_dword(lm_device_t *pdev, u32_t offset, u32_t val, u32_t
nvram_flags)
{
    lm_status_t lm_status;
    u32_t cmd_flags;
    u32_t j, cnt;

    /* Build the command word. */
    cmd_flags = nvram_flags | MCPR_NVM_COMMAND_DOIT | MCPR_NVM_COMMAND_WR;

    /* Change to little endian if required. */
    #if defined(LITTLE_ENDIAN)
    val = ((val & 0xff) << 24) | ((val & 0xff00) << 8) |
        ((val & 0xff0000) >> 8) | ((val >> 24) & 0xff);
    #endif
```

```
    /* Need to clear DONE bit separately. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, MCPR_NVM_COMMAND_DONE);

    /* Write the data. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_WRITE, val);

    /* Address of the NVRAM to write to. */
   REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_ADDR, offset & MCPR_NVM_ADDR_NVM_ADDR_VALUE);

    /* Issue the write command. */
    REG_WR(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, cmd_flags);

    cnt = NVRAM_TIMEOUT_COUNT;

    /* Wait for completion. */
    lm_status = LM_STATUS_BUSY;
    for(j = 0; j < cnt; j++)
    {
        mm_wait(pdev, 5);

        REG_RD(pdev, GRCBASE_MCP, MCP_REG_MCPR_NVM_COMMAND, &val);
        if(val & MCPR_NVM_COMMAND_DONE)
        {
            lm_status = LM_STATUS_SUCCESS;
            break;
        }
    }

    return lm_status;
} /* nvram_write_dword */


/****************************************************************************
 * Read the buf_size number of DWORDs from NVRAM into ret_buf[]             *
 ****************************************************************************/
lm_status_t lm_nvram_read(lm_device_t *pdev, u32_t offset, u32_t *ret_buf, u32_t buf_size)
{
    lm_status_t lm_status;
    u32_t cmd_flags;

    /* Return Failure if either buf_size or offset is not DWORD aligned.*/
    if((buf_size & 0x03) || (offset & 0x03))
    {
        return LM_STATUS_FAILURE;
    }

    /* Return Failure if the given offset+buf_size crosses the total NVRAM size.*/
    /* total_size is determined during the nvram initialization of the driver
       initialization based on the detected NVRAM type*/
    if(offset + buf_size > pdev->hw_info.flash_spec.total_size)
    {
        return LM_STATUS_FAILURE;
    }

    /* Request access to the flash interface. */
    lm_status = acquire_nvram_lock(pdev);
```

```
    if(lm_status != LM_STATUS_SUCCESS)
    {
        return lm_status;
    }

    /* Enable access to flash interface */
    lm_status = enable_nvram_access(pdev);
    if(lm_status != LM_STATUS_SUCCESS)
    {
    return lm_status;
    }

    /* Read the first word. Note the use if MODE_256 bit*/
    cmd_flags = MCPR_NVM_COMMAND_FIRST | MCPR_NVM_COMMAND_MODE_256;
    while(buf_size > sizeof(u32_t) && lm_status == LM_STATUS_SUCCESS)
    {
        lm_status = nvram_read_dword(pdev, offset, ret_buf, cmd_flags);

        /* Advance to the next dword. */
        offset += sizeof(u32_t);
        ret_buf++;
        buf_size -= sizeof(u32_t);
     cmd_flags = 0;
    }

    if(lm_status == LM_STATUS_SUCCESS)
    {
        cmd_flags |= MCPR_NVM_COMMAND_LAST;
        lm_status = nvram_read_dword(pdev, offset, ret_buf, cmd_flags);
    }

    /* Disable access to flash interface */
    disable_nvram_access(pdev);

    /* Release the NVRAM access arbitration lock */
    release_nvram_lock(pdev);

    return lm_status;
} /* lm_nvram_read */


/*******************************************************************************
 * Write the buf_size number of DWORDs from data_buf[] into NVRAM              *
 *******************************************************************************/
lm_status_t lm_nvram_write(lm_device_t *pdev, u32_t offset, u32_t *data_buf, u32_t buf_size)
{
    lm_status_t lm_status;
    u32_t cmd_flags;
    u32_t written_so_far, page_start, page_end, data_start, data_end;
    u32_t idx, *ptr32, addr;
    static u32_t flash_buffer[NVRAM_PAGE_SIZE/4];

    /* Return Failure if offset is not DWORD aligned.*/
    if(offset & 0x03)
    {
        return LM_STATUS_FAILURE;
```

```
    }

    /* Return Failure if the given offset+buf_size crosses the total NVRAM size.*/
    /* pdev->hw_info.flash_spec.total_size is determined during the nvram
       initialization of the driver initialization based on the detected NVRAM type*/
    if(offset + buf_size > pdev->hw_info.flash_spec.total_size)
    {
        return LM_STATUS_FAILURE;
    }

    lm_status = LM_STATUS_SUCCESS;

    written_so_far = 0;
    ptr32 = data_buf;

    while (written_so_far < buf_size)
    {
        /* Find the page_start addr */
        page_start = offset + written_so_far;

       /* page_size is determined during the nvram initialization of the driver
          initialization based on the detected NVRAM type*/
        page_start -= (page_start % pdev->hw_info.flash_spec.page_size);

        /* Find the page_end addr */
        page_end = page_start + pdev->hw_info.flash_spec.page_size;

        /* Find the data_start addr */
        data_start = (written_so_far==0) ?  offset : page_start;

        /* Find the data_end addr */
        data_end = (page_end > offset + buf_size) ?
                   (offset+buf_size) : page_end;

        /* Read the whole page into the buffer (non-buffer flash only) */
        lm_status = lm_nvram_read (pdev, page_start, flash_buffer, pdev-
>hw_info.flash_spec.page_size);
        if(lm_status != LM_STATUS_SUCCESS)
        return lm_status;

        /* Request access to the flash interface. */
    lm_status = acquire_nvram_lock(pdev);
    if(lm_status != LM_STATUS_SUCCESS)
        return lm_status;

    /* Enable access to flash interface */
    lm_status = enable_nvram_access(pdev);
    if(lm_status != LM_STATUS_SUCCESS)
    {
        release_nvram_lock(pdev);
        return lm_status;
    }

    /* Enable writes to flash interface (unlock write-protect) */
    lm_status = enable_nvram_write(pdev);
    if(lm_status != LM_STATUS_SUCCESS)
```

```
    {
        disable_nvram_access(pdev);
        release_nvram_lock(pdev);
        return lm_status;
    }

        /* Loop to write back the buffer data from page_start to data_start */
        cmd_flags = MCPR_NVM_COMMAND_FIRST | MCPR_NVM_COMMAND_MODE_256;
        idx = 0;
        for (addr=page_start; addr<data_start; addr+=4, idx++)
        {
            /* Write back only for non-buffered flash */
            nvram_write_dword(pdev, addr, flash_buffer[idx], cmd_flags);
            cmd_flags = 0;
        }

        /* Loop to write the new data from data_start to data_end */
        for (addr=data_start; addr<data_end; addr+=4, idx++)
        {
            if (addr==(page_end-4))             {
                cmd_flags = MCPR_NVM_COMMAND_LAST;
            }
            nvram_write_dword(pdev, addr, *ptr32, cmd_flags);
            cmd_flags = 0;
            ptr32++;
        }

        /* Loop to write back the buffer data from data_end to page_end */
        for (addr=data_end; addr<page_end; addr+=4, idx++)
        {
            /* Write back only for non-buffered flash */
            if (addr == page_end-4)
            {
                cmd_flags = MCPR_NVM_COMMAND_LAST;
            }
            nvram_write_dword(pdev, addr, flash_buffer[idx], cmd_flags);
            cmd_flags = 0;
        }

    /* Disable writes to flash interface (lock write-protect) */
    disable_nvram_write(pdev);

    /* Disable access to flash interface */
    disable_nvram_access(pdev);

    release_nvram_lock(pdev);

        /* Increment written_so_far */
        written_so_far += data_end - data_start;
    } // while

    return lm_status;

} /* lm_nvram_write */
```

Broadcom Corporation

5300 Calilfornia Avenue
Irvine, CA 92617
Phone: 949-926-5000
Fax: 949-926-5203