RDMA Smart NIC (*RSNIC IP*)

Table of Contents

The Following table shows you the revision history for this document.

Date	Description	Comment
15/08/23	1. Added Chapter 11. Ethernet 2. Ethernet technology 3. Ethernet protocol 4. Ethernet data connection 5. Ethernet supported speed 6. Rsnic IP implementation	
7/13/22	7. Added CMAC Description 8. Added Figure 10-1, RDMA IB Payload Sample 9. Added Figure 10-2, RDMA Memory to Memory Transfer 10. Changed Figure 10-1 to Figure 10-3, CMAC Block Diagram	
7/12/22	1. Added Figure 10-1, CMAC Block Diagram	
7/11/22	1. Add Table 10-1 CMAC Read Operation 2. Add Table 10-2 CMAC Write Operation 3. Add Figure 10-3 CMAC Block Diagram	
7/9/22	1. Added Chapter 10 - CMAC Implementation	
6/6/22	1. Added Figure 9-7, 9-8, 9-9 and 9-10 2. Added Chapter 9, Memory Configuration 3. Updated Memory Map Table 8-1 to include CMAC Base Address 4. Elaborate Figure 7-8-2-1 Relation of UMR to MKey and SQ 5. Elaborate Figure 7-8-3-1 UAR Relation to WQ and Doorbell from HCA and PCI Mapping 6. Added Figure 7-8-3-1, UAR Relation to WQ and Doorbell from HCA and PCI Mapping 7. Added Current Build Memory map Table 8-1 8. Added Figure 7-8-2-1, Relation of UMR to MKey and SQ 9. Changed previous Figure 7-8-3-1 to Figure 7-8-3-2, UAR. 10. Adding Figure 7-8-3-2 Elaborating UAR Functionality	
4/27/22	1. Cleanup heading 2 and 3 in Chapter 4	
4/26/22	1. Added Chapter 8, Memory Map	
4/25/22	1. Added Figure 7-7-2-1, Relation of QP, CQ and EQ	
4/22/22	1. Added Queue Pair (Section 7.5) 2. Added UMR (Section 7.8.2) 3. Added UAR (Section 7.8.3) 4. Added MKey (Section 7.9	

4/20/22	1. Added in Overview Chapter 2, HCA Operation, and Interrupts 2. Added Figure 2-2-1 SQ, RQ, WQ and WQE Relation and Figure 2-3-1	
4/18/22	1. Added in Chapter 7, Host to Device interface, work queues, structure and access, send queue, receive queue, doorbell record, WQE ownership, posting a work request to work queue, posting work request on shared receive queue, work request (WQE) formats, send WQE format,	
4/17/22	1. Added Figure 7-5-1 Communication Interface	
3/30/22	1. Edit Chapter 4 mailbox configuration commands processing to make commands more clearer	
3/24//22	1. Replaced DMA with new implementation, Chapter 6	
3/24/22	1. Added Figure 6-1 DMA Block Diagram	
3/16/22	1. Added Packet Retransmission Figure 4-4-10-1	
3/13/22	1. Added Table 6-3 write descriptor and 6-4 MSI-X.	
3/11/22	1. Changed Register Space to DMA Operations. Added Table 6-1 DMA Descriptor and Table 6-2 Read Descriptor.	
3/2/22	1. Added Table 4-1.21 Create QP CSR with ~31 registers to show all the registers for CREATE QP based on firmware tracing. 2. Added Table 4-15: based on firmware tracing, created a list of 18 mailbox commands called when running the ib_read_bw rdma perftest (client side), including opcode.	
3/1/22	1. Chapter 6 DMA Operation	
2/26/22	1. Add Chapter 14. References	
2/23/22	1. Added Generic MailBox Processing 2. Added CSR: Query HCA Capability, Query Adapter, Query Pages, Manage Pages, SET HCA CAP, QUERY ISSI, SET ISSI, SET DRIVER VERSION, CREATE MKEY. QUERY MKEY. DESTROY MKEY. QUERY SPECIAL CONTEXT. 3. CREATE EQ. DESTROY EQ, QUERY EQ, GEN EQE, CREATE CQ, DESTROY CQ, QUERY CQ, CREATE SQ, CREATE RQ 4. Update details on RDMA address translation/ memory region with details on Virtual to Physical Translation	

	5. Add figure to show host creates a send queue	
2/16/22	1. Added block diagrams for RDMA 2. Added memory protection flow	
2/4/22	1. Initial Release	

Chapter 1. Introduction

The *RSNIC-IP* is an implementation of RDMA over Converged Ethernet (RoCE v2) NIC functionality. This IP core can work with a wide variety of Xilinx hard and soft MAC IP. It provides a high throughput, low latency, and reliable data transfer solution over standard Ethernet. The *RSNIC-IP* allows simultaneous connections to multiple remote hosts running RoCE v2 traffic.

RDMA offers host-offload, host bypass to enable a secure direct memory-to-memory data communication between two applications over the network. RDMA benefits network performance by employing Zero copy applications that can perform data transfers without the involvement of the network software stack. Data is sent and received directly to the buffers without being copied between the network layers. Kernel bypass applications can perform data transfers directly from user space without kernel involvement. No CPU involvement. Applications can access remote memory without consuming any CPU time in the remote server. remote memory server will be read without the intervention from the remote processor.

RDMA differs from the traditional network interface because it bypasses the operating system. This allows programs that implement RDMA to have: Low latency, high throughput, offload CPU. One of the chief technologies to access memory directly without intervention of CPU. Providing direct memory access of one host to the memory of another host without involving the CPU while boosting performance and reducing latency.

RDMA over converged Ethernet or RoCE is also a network protocol that allows remote direct memory access over an Ethernet network. There are two versions of RoCE version 1 and RoCE version 2. RoCE version 1 is an Ethernet link layer protocol, hence allows communications between any two hosts only in the same Ethernet broadcast domain and is no longer widely used. RoCE version 2 is an internet layer protocol which means that like iWARP RoCE version 2 packets can be routed.

In contrast TCP/IP communications typically requires copy operation which adds latency and consumes significant CPU resources. Where RDMA offers zero copy network by enabling network to direct transfer data to or from application memory. Effectively eliminating the need to copy data between application memory and data buffer in the operating system. When an application performs a read or write the application data is directly delivered.

RDMA eliminates the need to copy data between application memory and the data buffers in the operating system. When an application performs an RDMA read or write request, the application data is delivered directly to the network, reducing latency and enabling fast message transfer. Such transfers require no work to be done by CPUs, caches or context switches and transfers continue in parallel with other system operations.

RDMA is used to accelerate performance in many types of applications, including storage fabrics and storage applications such as Windows SMB direct, and storage space direct, as well as NVMe over fabric. RDMA is also used in high performance computing, big data, relational databases, and anywhere there is a need for lower latency, higher bandwidth and decreased CPU utilization. There are multiple industries that use RDMA, including financial services, medical appliances, and cloud computing to name just a few.

In short, RDMA can move data directly from the memory of one computer into that of another without involving either one's operating system. This permits high throughput low latency networking, which is especially useful in massively parallel computing clusters.

1.1 Features

- Support for RoCE v2
- Support RDMA Call Library
- Mimic the Mellanox Infiniband, RoCE v2, and RDMA
- Mellanox Stores the packet in the Host Server
- DMA Uses Virtual Addresses
- Support for memory registration and protection domains

Chapter 2. Overview

This chapter provides an overview of the *RSNIC-IP* core features where the applications will be useful and the standard conformance.
RSNIC-IP is an IP implementation of RDMA over Converged Ethernet (RoCE v2) protocol for embedded target or initiator devices.

2.1 Core Feature Overview

1. Support for RoCE v2

Table 2-1-1: The *RSNIC-IP* Supported features

***RSNIC IP* Supported RoCE v2 Functionalities**	
Number	**Function/Feature**
1	RDMA_SEND
2	RDMA_WRITE
3	RDMA_READ
4	RDMA_WRITE_FIRST
5	RDMA_WRITE_MIDDLE
6	RDMA_WRITE_LAST
7	RDMA_WRITE_LAST_WITH_IMD
8	RDMA_WRITE_ONLY
9	RDMA_WRITE_MIDDLE
10	RDMA_WRITE_ONLY_WITH_IMD
11	WRITE_READ_REQUEST
12	RDMA_READ_RESP_FIRST
13	RDMA_READ_RESP_MIDDLE
14	RDMA_READ_RESP_LAST
15	RDMA_READ_RESP_ONLY
16	READ_ACK
17	READ_PART_ONLY
18	READ_PART_FIRST
19	READ_PART_MIDDLE
20	READ_PART_LAST
21	RDMA_READ_POINTER_REQUEST
22	RDMA_READ_CONSISTENT_REQUEST

23	Support for up to 127 Connections (initial and can be increase along the way)
24	Scalable Design of up to 127 RDMA Queue Pairs (Initial and can be increase along the way)
25	Support Dynamic Memory Registration
26	Hardware Handshake Mechanism for efficient Doorbell exchange with the user application logic

Table 2-1-1: Support the Following Subset of RoCE v2 functionalities

2.2 HCA Operation

After the HCA is initialized and opened, the host software supports send and receive data transfers through Work Requests (WRs) posted to Work Queues (WQs). Each WQ contains a Send Work Queue (SQ) for posted send requests, and a Receive Work Queue (RQ) for posted receive requests. The WR is posted as a Work Queue Entry (WQE) to an SQ/RQ. These WQEs can either cause data to be transmitted or received. WQEs are essentially descriptors that control the source and destination of data movement.

Figure 2-2-1 SQ, RQ, WQ and WQE Relation

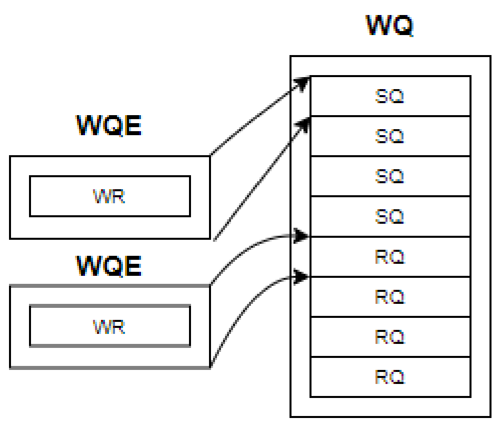


Figure 2-2-1 SQ, RQ, WQ and WQE Relation

2.3 Interrupts

The HCA supports multiple means of generating interrupts - asserting a pin on its physical interface, emulating interrupt pin assertion on the host link (PCI) or generating Message Signaled Interrupts (MSI/MSI-X), enabling software to de-multiplex interrupts to different host consumers.

Each EQ can be configured to generate an interrupt when an EQE is posted to that EQ. Multiple EQs can be mapped to the same interrupt vector (MSI-X) maintaining many-to-one relations between EQs and interrupts.

The relations between WQs, CQs, EQs and different interrupt messages (MSIX vectors) is shown in Figure 2-3-1

Figure 2-3-1 WQs, CQs, EQs and Interrupt Relations

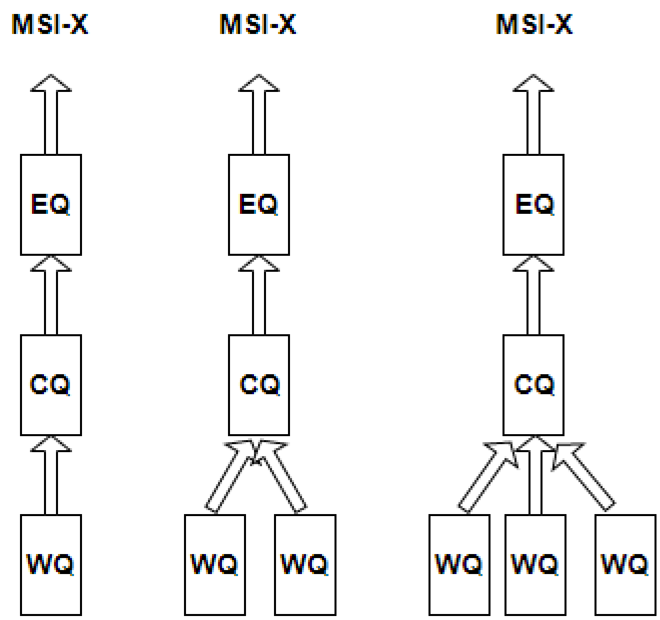


Figure 2-3-1 WQs, CQs, EQs and Interrupt Relations

Asynchronous events - like link state change or various errors - can also cause an event to be posted and an interrupt to be asserted. Each asynchronous event type can be mapped to a specific EQ and optionally generate an interrupt. Hardware does not prevent mapping synchronous and asynchronous events to the same EQ. The user should use common sense while configuring a device and use distinct EQs for asynchronous events.

Chapter 3. Block Diagram

Figure 3-1: *RSNIC-IP* Block Diagram

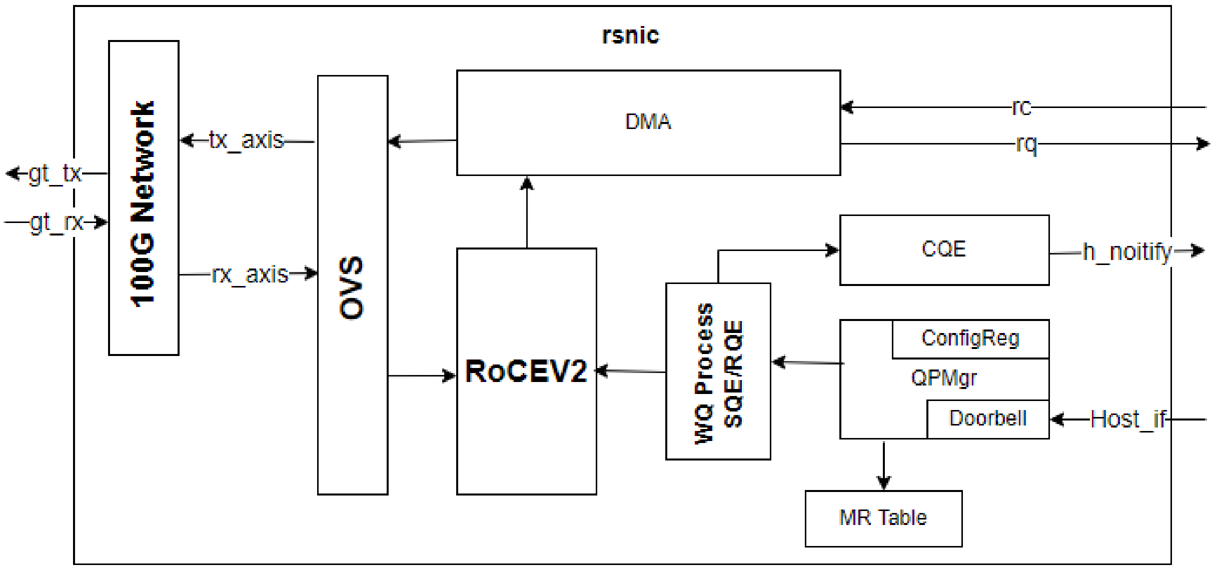


Figure 3-1: *RSNIC-IP* Block Diagram

The *RSNIC-IP* are composed of the several major modules, The QP manager, WQE handler, CQE generator, IBH processor and IP handler.

The IP works on a 512-bit internal datapath that can be completely hardware accelerated without any software intervention for data transfer. All recoverable faults like retransmission due to packet drops are also handled entirely in the hardware.

Figure 3-2: RSNIC IP Sample Application

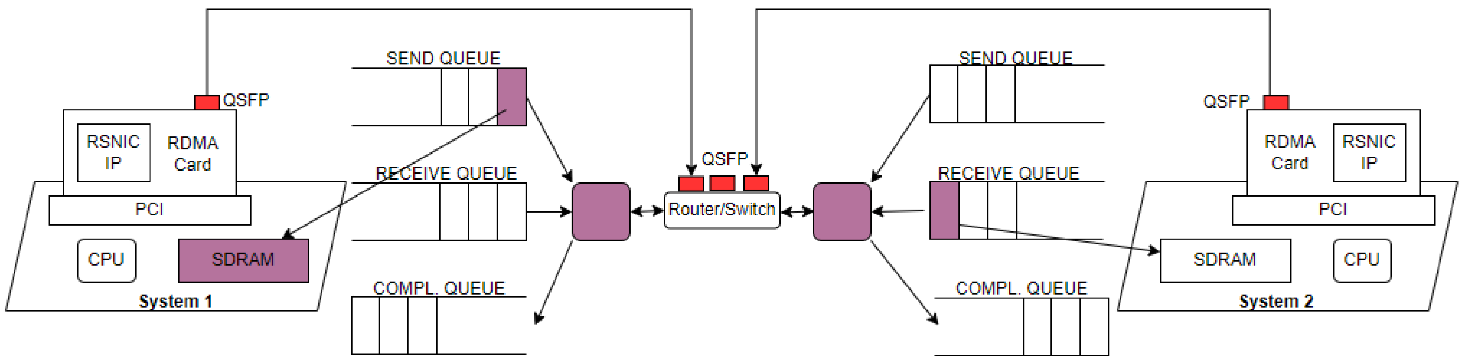


Figure 3-2: RSNIC IP Sample Application

Chapter 4. Design Approach Description

This chapter describes in detail how each function or feature of the *RSNIC-IP* core works.

Figure 4-1: RDMA Block Diagram

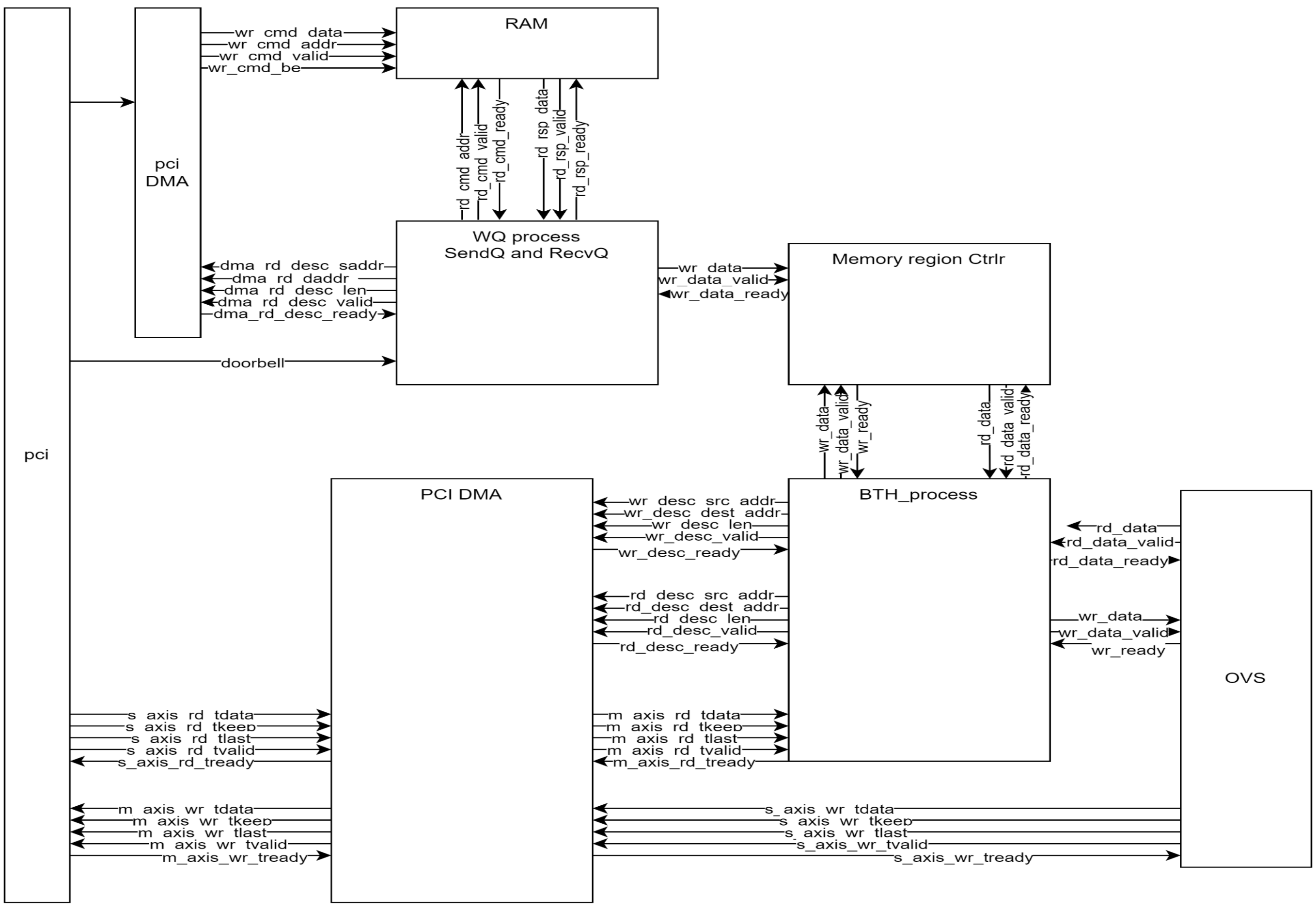


Figure 4-1: RDMA Block Diagram

1.) The Doorbell from the host will be received by the WQ process module.
2.) The WQ process module will trigger the dma transfer of the new WQE to ram.
3.) The WQ process will read the WQE from RAM and pass it to Memory region Ctrl.
4.) Memory region Ctrl will locate the keyCtx entry and validate the requested memory region.
5.) Memory region Ctrl will translate the Virtual address to physical address and pass to the BTH process module.
6.) The process module will create BTH and append it to data or extract the BTH and activate the DMA.

4.1 MailBox Configuration Commands Processing

In this section the *RSNIC IP* detailed responses to Host Device Driver Configuration Command Processing will be shown.

Listed below is the Generic MailBox Configuration Command Processing:

Host Driver Side:

1. Fill up the MailBox input buffer in the Host SDRAM.
2. Assert the DoorBell to Notify the Device. This will generate an interrupt to the Device.

Device Side:

3. Firmware will receive the Interrupt DoorBell Notification
4. Firmware will set up the PCI Read DMA to get the content of the MailBox.

PCI Hardware Side

5. *The actual Transfer of Data from Host SDRAM to MailBox input BRAM will happen after the PCI Read DMA setup.*
6. PCI BAR Module will Interrupt the Firmware notifying there is a new MailBox Command in the MailBox Input Buffer.

Firmware Side:

7. Firmware will Process the MailBox Command depending on the OpCode. Every scenario in the OpCode will be discussed separately.
8. Firmware will Fill-up its reply in the MailBox Output Buffer.
9. Firmware will Set up PCI Write DMA to send the MailBox Output Buffer.

PCI Hardware Side

10. *The actual Transfer of Data from MailBox Output Buffer to Host SDRAM will happen after the PCI Write DMA setup.*
11. MSI-X will be sent to Host

Host Driver Side:

12. Driver will Read the response from the Device in the MailBox Output.

4.1.1 Query HCA Capability CSR

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
00h	5	31:27	0x16		log_max_cq_sz	Pls refer to page 198 of PRM
	5	26:22	0x18		log_max_cq	Pls refer to page 198 of PRM
	4	21:18	0x08		log_max_eq	Pls refer to page 198 of PRM
	6	17:12	0x98		log_max_mkey	Pls refer to page 198 of PRM
	8	11:04	0x16		log_max_eq_sz	Pls refer to page 198 of PRM
	1	3	0x00		cache_line_128_byte	Pls refer to page 198 of PRM
	1	2	0x01		start_pad	Pls refer to page 198 of PRM
	1	1	0x01		end_pad	Pls refer to page 198 of PRM
	1	0	0x01		vport_counters	Pls refer to page 199 of PRM
04h	6	31:26	0xD0		log_max_klm_list_sz	Pls refer to page 198 of PRM
	7	25:19	0xC0		log_max_mrw_sz	Pls refer to page 198 of PRM
	8	18:11	0x04		max_indirection	Pls refer to page 198 of PRM
	8	10:03	0x8F		num_ports	Pls refer to page 199 of PRM
	1	2	0x01		nic_flow_table	Pls refer to page 199 of PRM
	1	1	0x01		vport_group_manager	Pls refer to page 199 of PRM
	1	0	0x01		temp_warn_event	Pls refer to page 199 of PRM
08h	4	31:28	0x05		max_tc	Pls refer to page 199 of PRM
	5	27:23	0x1E		log_max_msg	Pls refer to page 199 of PRM
	4	22:19	0x02		cqe_version	Pls refer to page 199 of PRM
	2	18:17	0x01		cmdif_checksum	Pls refer to page 199 of PRM
	1	16	0x01		wq_signature	Pls refer to page 199 of PRM
	1	15	0x01		setr_data_cqe	Pls refer to page 199 of PRM
	1	14	0x01		eth_net_offloads	Pls refer to page 199 of PRM

	1	13	0x01		cq_oi	Pls refer to page 200 of PRM
	1	12	0x01		cq_resize	Pls refer to page 200 of PRM
	1	11	0x01		cq_moderation	Pls refer to page 200 of PRM
	1	10	0x01		cq_eq_remap	Pls refer to page 200 of PRM
	1	9	0x01		scqe_break_moderation	Pls refer to page 200 of PRM
	6	8:3	0x05		uar_sz	Pls refer to page 200 of PRM
	1	2	0x01		bf	Pls refer to page 200 of PRM
	1	1	0x01		driver_version	Pls refer to page 200 of PRM
	1	0	0x01		pad_tx_eth_packet	Pls refer to page 200 of PRM
0Ch	5	31:27	0x10		Log_max_transport_ domain	Pls refer to page 200 of PRM
	5	26:22	0x18		log_max_pd	Pls refer to page 200 of PRM
	8	21:14	0xE0		Log_max_flow_counter_ bulk	Pls refer to page 200 of PRM
	1	13	0x01		modify_tis	Pls refer to page 201 of PRM 0 - Feature not supported
	5	12:08	0x17		log_max_rq	Pls refer to page 201 of PRM 0 - Feature not supported
	5	07:03	0x17		log_max_sq	Pls refer to page 201 of PRM 0 - Feature not supported
	1	2	0x01		basic_cyclic_rev_wqe	Pls refer to page 201 of PRM
	2	1:0			reserved	
10h	16	31:16	0xFF60		max_flow_counter	Pls refer to page 200 of PRM
	5	15:11	0x10		log_max_tir	Pls refer to page 201 of PRM 0 - Feature not supported
	5	10:06	0x17		log_max_tis	Pls refer to page 201 of PRM
	5	05:01	0x17		log_max_rmp	Pls refer to page 201 of PRM 0 - Feature not supported
	1	0			reserved	
14h	5	31:27	0x10		log_max_rqt	Pls refer to page 201 of PRM 0 - Feature not supported
	5	26:22	0xB0		log_max_rqt_sz	Pls refer to page 201 of PRM
	5	21:17	0x00		log_max_tis_per_sq	Pls refer to page 201 of PRM
	5	16:12	0x09		log_max_stride_sz_rq	Pls refer to page 201 of PRM

	5	11:07	0x04		log_min_stride_sz_rq	Pls refer to page 201 of PRM
	5	06:02	0x06		log_max_stride_sz_sq	Pls refer to page 201 of PRM
	2	01:00	0x03		port_type	Pls refer to page 199 of PRM
18h	5	31:27	0x0F		log_max_wq_sz	Pls refer to page 201 of PRM
	5	26:22	0x0C		log_max_vlan_list	Pls refer to page 201 of PRM
	5	21:17	0x0E		log_max_current_mc_list	Pls refer to page 201 of PRM
	5	16:12	0x00		log_max_l2_table	Pls refer to page 202 of PRM
	1	11	0x01		cq_period_start_from_cqe	Pls refer to page 200 of PRM
	8	10:03	0x0C		log_pg_sz	Pls refer to page 200 of PRM
	3	02:00			reserved	
1Ch	16	31:16	0x00		log_uar_page_sz	Pls refer to page 202 of PRM
	16	15:00			reserved	
20h	32	31:00	0x9C		device_frequency_mhz	Pls refer to page 202 of PRM
24h	5	31:27	0x09		log_bf_reg_sz	Pls refer to page 200 of PRM
	5	26:22	0x04		log_max_current_uc_list	Pls refer to page 201 of PRM
	22	21:00			reserved	reserved

Table 4-1: Query HCA Capability CSR

Query HCA Capability CSR Command has a different scenario, adding extra firmware code below is required.

Firmware Side:

1. Firmware will use the Query HCA Capability CSR and fill-up the MailBox Output buffer depending on the op_mod listed below.
   Op_mod:
      Bit[0] indicates Maximum or Current capabilities
      i. 0x0: Maximum
      ii. 0x1: Current
      Bits[15:1] indicates Capability Type
      iii. 0x0: General Device Capabilities
      iv. 0x1: Ethernet Offload Capabilities
      v. 0x7: NIC Flow Table Capabilities

   If Bit[0] = 0, Maximum.
2. Firmware will allow the following of Log (base 2):
   a. log_max_cq_sz - Maximum CQEs allowed in a CQ.
   b. log_max_cq - Maximum number of CQs supported.

c. log_max_eq_sz - Maximum EQEs allowed in an EQ.
d. log_max_mkey - Maximum number of data MKey entries (the number of regions/windows).
e. log_max_eq - Maximum number of EQs.
f. max_indirection - Maximum level of MKey indirection supported.
g. log_max_mrw_sz - Maximum size of a Memory Region/Window.
h. Log_max_klm_list_sz - Maximum indirect klm entries list (in MKey).
i. log_max_msg - Maximum message size in bytes supported by the device.
j. log_max_transport_domain - Maximum number of Transport Domains.
k. log_max_flow_counter_bulk - Maximum number of flow counters that can be queried by a single QUERY_FLOW_COUNTER command.
l. max_flow_counter - Maximum number of flow counters.
m. log_max_stride_sz_rq - Maximum size (in bytes) of RQ stride.
n. log_max_stride_sz_sq - Maximum size (in bytes) of SQ stride.
o. log_max_wq_sz - Maximum number of WQEs allowed on the WQ.
p. log_max_vlan_list - Maximum size of vlan list used in nic_port_context.
q. log_max_current_mc_list - Maximum size of current_mc_mac_address list used in nic_vport_contect.
r. log_max_current_uc_list - Maximum size of current_uc_mac_address list used in nic_vport_context.
s. log_max_l2_table - Maximum size of L2 Table.

4.1.2 Query Adapter CSR

Retrieves specific parameters. This information is used by the host device driver in order to clear interrupt signaling by the device. Table below is the data structure to be sent upon responding to the command.

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
28h	24	15:00	0x00	RO	ieee_vendor_id	IEEE vendor_id. Supported only starting from ISSI==1
	8	31:16				Reserved
2Ch	16	31:16	0x1D00	RO	vsd_vendor_id	PCISIG Vendor ID of the vendor specifying/formatting the VSD. The vsd_vendor_id identifies the management domain of the vsd/psid data. Different vendors may choose different vsd/psid format and encoding as long as they use their assigned

	16	15:00				vsd_vendor_id. The psid format as described below is used in conjunction with Mellanox vsd_vendor_id (15B3h).
30h-3Ch	128	31:00 31:00 31:00 31:00	0x00000000	RO	vsd_contd_psid	This field carries the last sixteen bytes of the VSD field. For Mellanox formatted VSD (vsd_vendor_id=15B3h) the last 16 bytes VSD is used as PSID. The PSID field is a 16-ascii (byte) character string which acts as an HCA Adapter Card ID. The format of the PSID is as follows: Vendor Symbol (VS) - 3 characters Board Type Symbol (BT) - 3 characters Board Version Symbol (BV) - 3 characters Parameter Set Number (PS) - 4 characters Reserved (R) - 3 characters (filled with zeros) The various characters are organized as described in See Table 164, “PSID Character Offsets (Example is in Parentheses),” on page 212. Example: A PSID for Mellanox’s MHXL-CF128-T HCA board is MT_0030000001, where: MT_ is the Mellanox Vendor Symbol 003 is the MHXL-CF128-T Board Type Symbol 000 is the Board Version Symbol 0001 is the Parameter Set Number The byte order is as follows: Bits 31:24 at offset 0 represent the first character (byte), bits 23:16 - second character, etc. Bits 31:24 at offset 4 represent the fourth character, etc.
40h-10Ch	1664 [20h-ECh]	31:00 31:00 . . 31:00 31:00	0x00000000 0x00000000 . . 0x00000000 0x69696969	RW	vsd	Vendor Specific Data. The VSD string that is burnt to the Flash with the Firmware.

Table 4-2: Query Adapter CSR

4.1.3 Query Pages CSR

Offset	No. of Bits	Bits Alloc	Default Values	Access		Description
110h	32	31:00	0x00000000	RW	num_pages	

Table 4-3: Query Pages CSR

Again, Query Pages CSR Command has a different scenario, so adding extra firmware code below is also required.

The process is the same with Query HCA Capability CSR with the addition of firmware process below.

Firmware Side:

1. Firmware will use the Query Pages CSR and fill up the MailBox Output buffer depending on the op_mod listed below.
   opcode modifier:
   0x1: boot_pages
   0x2: init_pages
   0x3: regular_page

4.1.4 Manage Pages CSR

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
114h	32	31:00		RW	output_num_entries	Output Number Entries
118h-11Ch	32	31:00		RW	pas[0]	Physical Address High 0
	20	31:12				Physical Address Low 0
	12	11:00				reserved

120h-124h	32	31:00		RW	pas[1]	Physical Address High 1
	20	31:12				Physical Address Low 1
	12	11:00				reserved

Table 4-4: Manage Pages CSR

Additional firmware process here.

Firmware Side:

1. Firmware will use the Manage Pages CSR and fill up the MailBox Output buffer depending on the op_mod listed below.
   opcode modifier:
   0x0: ALLOCATION_FAIL - SW cannot give pages. No mailbox is valid.
   0x1: ALLOCATION_SUCCESS - SW gives pages to HCA. Input parameter and input MailBoxes are valid.
   0x2: HCA_RETURN_PAGES - SW requests to return pages from HCA back to the host. Input parameter, output parameter and output mailbox are valid.

4.1.5 SET HCA CAP CSR

Firmware code below is the addition after the generic MailBox Processing,

Firmware Side:

1. Firmware will use Table 4-4 HCA Capability CSR and modify based on what has been Read on the Input MailBox.

4.1.6 QUERY ISSI CSR

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
128h	8	31:24				Reserved
	16	23:8	0x00000000	RW	current_issi	Pls refer to page 217 of PRM
12Ch-140h	32	32:0	0x00000000	RW	supported_issi	Pls refer to page 217 of PRM
	32	32:0	0x00000000	RW		Pls refer to page 217 of PRM
	32	32:0	0x00000000	RW		Pls refer to page 217 of PRM
	32	32:0	0x00000002	RW		Pls refer to page 217 of PRM
	32	32:0	0x00000000	RW		Pls refer to page 217 of PRM

Table 4-5: Query ISSI CSR

4.1.7 SET ISSI CSR

Firmware code below is the addition after the generic MailBox Processing,

Firmware Side:

1. Firmware will use the Table 4-5: Query ISSI CSR to respond and put it to Input MailBox.
2. BAD_OPCODE returns indicate that only ISSI = 0 is supported.

4.1.8 SET DRIVER VERSION CSR

The process is the same with Query HCA Capability CSR.

Offset	**No. of Bits**	**Bits Alloc**	**Default Values**	**Access**	**Name**	**Description**
144h-180h	512	31:00 . . 31:00		RW	driver version	The Driver version string is a concatenation of 3 fields separated by ",": • OS_name - generic string • Driver_name - Generic string • Driver_version_number - Concatenation of 3 numbers separated by ".": • Main_version - 3 digits (no need for leading zeros) • Minor_version - 3 digits (mandatory leading zeros) • Sub_minor_version - 6 digits (mandatory leading zeros) The order of the fields is from left (first - OS_name) to right (last - Sub_minor_version) The total length of the driver_version is limited to 64 Bytes (including the end-of-string symbol).

Table 4-6: Set Driver Version CSR

4.1.9 CREATE MKEY CSR

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
184h	24	31:08		RW	mkey_index	MKey index to be returned to Software
	8	07:00				Reserved

Table 4-7: Create MKey CSR

4.1.10 QUERY MKEY CSR

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
188h	1	31		RW	free	0 - memory key is in use. It can be used for address translation 1 - memory key is free. Cannot be used for address translation
	1	30		RW	umr_en	Enable umr operation on this MKey
	1	29		RW	rw	If set, remote write is enabled
	1	28		RW	rr	If set, remote read is enabled
	1	27		RW	lw	If set, local write is enabled
	1	26		RW	lr	If set, local read is enabled. Must be set for all MKeys
	2	25:24		RW	access_mode	0x0: PA - (VA=PA, no translation needed) if set, no virtual to physical address translation is performed for this MKey. Not valid for, block mode MKey, replicated MTT MKey 0x1: MTT - (PA is needed) 0x2: KLMs - (Indirect access
	24	23:00		RW	qpn	must be 0xffffff.
18Ch	8	31:24		RW	mkey[7:0]	Variant part of MKey specified by this MKey context
	24	23:00		RW	pd	Protection domain

190h-194h	32	31:00		RW	start_addr	Start Address - Virtual address where this region/window starts
	32	31:00				
198h-19Ch	32	31:00		RW	len	Region length. Reserved when length64 bit is set (in which case the region length is 2^64B).
	32	31:00				
1A0h	32	31:00		RW	translations_octword_size	Size (in units of 16B) required for this MKey's physical buffer list or SGEs access_mode: MTT - each translation is 8B access_mode: KLM - each SGE is 16B access_mode: PA - reserved Must be a multiple of 4
1A4h	1	31		RW	length64	Enable registering 2^64 bytes per region
	5	30:26		RW	log_entity_size	When access_mode==MTT: log2 of Page size in bytes granularity. otherwise: reserved. Must be >=12
	26	25:00				Reserved

Table 4-8: Query MKey CSR

Firmware code below is the addition after the generic MailBox Processing,

Firmware Side:

1. Firmware will use the Table 4-8: Query MKey CSR to respond and put it to the Input MailBox.

4.1.11 DESTROY MKEY

Destroy MKey means was just to delete the copy of MKey Index that was passed in the input MailBox. It will also follow the generic process flow except that it doesn't return a value but a status.

4.1.12 QUERY SPECIAL CONTEXT CSR

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
1A8h	32	31:00		RW	resd_lkey	The value of the reserved Lkey for Base Memory Management Extension

Table 4-9: Query Special Context CSR

4.1.13 CREATE EQ CSR

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
1ACh	8	31:24		RW	eq_number	EQ Number
	24	23:00				Reserved

Table 4-10: Create EQ CSR

4.1.14 DESTROY EQ CSR

Destroy MKey means was just to delete the copy of EQ Number that was passed in the input MailBox. It will also follow the generic process flow except that it doesn't return a value but a status.

4.1.15 QUERY EQ CSR

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
1B0h	4				status	EQ status 0x0: OK 0xA: EQ_WRITE_FAILURE Valid for the QUERY_EQ command only
	4				st	Event delivery state machine 0x9: ARMED 0xA: FIRED other: reserved Reserved for CREATE_EQ.
	24				uar_page	UAR page this EQ can be accessed through (ringing EQ DoorBells)
1B4h	1				ec	Is set, all EQEs are written (collapsed) to first EQ entry

	1				oi	When set, overrun ignore is enabled.
	6				page_offset	Must be 0
	24				consumer_counter	Consumer counter. The counter is incremented for each EQE polled from the EQ. Must be 0x0 in EQ initialization. Indicates last consumer counter seen by HW (valid for the QUERY_EQ command only).
1B8h	24				producer_counter	Producer Counter. The counter is incremented for each EQE that is written by the HW to the EQ.EQ overrun is reported if Producer_counter + 1 equals to Consumer_counter and a EQE needs to be added.Maintained by HW (valid for the QUERY_EQ command only) Should be 0x0 in EQ initialization
	8				intr	MSI-X table entry index to be used to signal interrupts on this EQ. Reserved if MSI-X is not enabled in the PCI configuration header.
1BCh	5				log_page_size	Log (base 2) of page size in units of 4KByte
	5				log_eq_size	Log (base 2) of the EQ size (in entries).
	22					Reserved

Table 4-11: Query EQ CSR

4.1.16 GEN EQE

Generate EQE Command means was just to generate EQE from the EQ number passed in the input MailBox. It will also follow the generic process flow, it doesn't return a value but a status.

4.1.17 CREATE CQ CSR

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
1C0h	24	31:08		RW	cqn	CQ Number
	8	07:00				Reserved

4.1.18 QUERY CQ CSR

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
1C4h	4	31:28		RW	status	CQ status 0x0: OK 0x9: CQ_OVERFLOW 0xA: CQ_WRITE_FAIL Valid for the QUERY_CQ
	3	27:25			cqe_sz	CQE size 0: BYTES_64 1: BYTES_128
	1	24			cc	If set, all CQEs are written (collapsed) to first CQ entry
	24	23:00			uar_page	UAR page this CQ can be accessed through (ringing CQ DoorBells)
1C8h	1	31			scqe_break_moderation_en	When set, solicited CQE (CQE.SE flag is enabled) breaks Completion Event Moderation. CQE causes immediate EQE generation. Supported only if HCA_CAP. scqe_break_moderation==1.
	1	30			oi	When set, overrun ignore is enabled. When set, updates of CQ consumer counter (poll for completion) or Request completion notifications (Arm CQ) DoorBells should

						not be rung on that CQ.
	1	29			cqe_compression_en	When set, CQE compressing feature is enabled for that CQ.Must be zero when cqe size is 128 byte (cqe_sz == 1).
	2	28:27			cq_period_mode	0: upon_event - cq_period timer restarts upon event generation. 1: upon_cqe - cq_period timer restarts upon completion generation. Supported only when HCA_CAP.cq_period_start_from_cqe==
	4	26:23			st	Event delivery state machine 0x6: SOLICITED_NOTIFICATION_ REQUEST_ARMED 0x9: NOTIFICATION_REQUEST_ARMED 0xA: FIRED other: reserved Valid for the QUERY_CQ command only. Reserved for CREATE_CQ.
	6	22:17			page_offset	Must be 0
	5	16:12			log_cq_size	Log (base 2) of the CQ size (in entries). Maximum CQ size is 222 CQEs (max log_cq_size is 22)
	12	11:00			cq_period	Event Generation moderation timer in 1 usec granularity 0 - CQ moderation disabled
1CCh	24	31:08			last_notified_index	Last_notified_indx. Maintained by HW. Valid for QUERY_CQ command only. This field is for debug only purpose and is subject to change.
	8	07:00			c_eqn	EQ this CQ reports completion events to
1D0h	2	31:30			mini_cqe_res_format	Mini Cqe responder format. valid only when cqe_compression==1. 0: Responder Mini CQE consists from: Byte Count and RX hash result. 1: Responder Mini CQE consists from: Byte Count and HW Checksum value.

	24	29:06			last_solicit_index	Solicit_producer_indx. Maintained by HW. Valid for QUERY_CQ command only. This field is for debug only purpose and is subject to change.
	5	05:01			log_page_size	Log (base 2) of page size in units of 4KByte
	1	0				Reserved
1D4h	24	31:08			consumer_counter	Consumer counter. The counter is incremented for each CQE polled from the CQ. Must be 0x0 in CQ initialization. Indicates last consumer counter seen by HW (valid for the QUERY_CQ command only).
	8	07:00				
1D8h	24	31:08			producer_counter	Producer Counter. The counter is incremented for each CQE that is written by the HW to the CQ. CQ overrun is reported if Producer_counter + 1 equals to Consumer_counter and a CQE needs to be added. Maintained by HW (valid for the QUERY_CQ command only)
	8	07:00				
1DCh	16	31:16			cq_max_count	Event Generation Moderation counter, 0 - CQ moderation disabled
	16	15:00				Reserved
1E0h-1E4h	64	63:00			dbr_addr	CQ DB Record physical address

Table 4-12: Query CQ CSR

4.1.19 CREATE SQ CSR

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
SQ Context						
	1	31			rlkey	Reserved LKey enable. When set the reserved LKey can be used on the SQ.
	3	30:28			min_wqe_inline_mode	Sets the inline mode for the SQ. min_wqe_inline_mode defines the minimal required inline headers in Eth Segment of the SQ’s WQEs. If Eth Segment of the WQE does not contain the required inlined headers, the packet is silently dropped. Note: SQ.min_wqe_inline_mode must be >= Nic_vport.min_sq_wqe_inline_mode.
	4	27:24			state	SQ state 0x0: RST 0x1: RDY 0x3: ERR For QUERY_SQ - the current state is returned For MODIFY_SQ - the requested state is specified
	24	23:00			user_index	"User_index - an opaque identifier which software sets, which will be reported to the Completion Queue. Reserved if HCA_CAP.cqe_version==0."
	1	31			fre	Fast Register Enable. When set, the SQ supports Fast Register WQEs.
	1	30			flush_in_error_en	If set, and when SQ transitions into error state, the hardware will flush in error WQEs that were posted and WQEs that will be posted to SQ. Otherwise, when SQ enters an error state HW is not forced to flush in error all the WQEs.
	24	29:6			cqn	Completion Queue Number
	6	5:0				reserved
	16	31:16			tis_lst_sz	The number of entries in the list of TISes. This is the list of Transport Interface Send (TISes) that are associated with this SQ

	16	15:00				reserved
	24	31:8			tis_num[0]	List of TIS numbers. Note: in this revision of PRM, only a single TIS is supported
	8	7:0				reserved
Work Queue Context						
	24	31:8			pd	Protection Domain. Used for accessing data through WQEs (scatter/gather)
	1	7			wq_signature	If set, WQE signature will be checked on this WQ.
	2	6:5			end_padding_mode	Scattering end of incoming send message (or raw Eth packet) 0: END_PAD_NONE - scatter as-is 1: END_PAD_ALIGN - pad to cache line alignment other reserved Note that padding does not go beyond the receive WQE scatter entry length.
	5	4:0			page_offset	Page offset in offset in quanta of (page_size / 64)
	4	31:28			wq_type	WQ type 0x0: WQ_LINKED_LIST - Linked List 0x1: WQ_CYCLIC - Cyclic Descriptors Send Queue WQ must be set to 1.
	24	27:4			uar_page	UAR number allocated for ringing DoorBells for this WQ. For RQ this field is required only if cd_slave is enabled otherwise this field is reserved.
	4	3:0			log_wq_stride	The size of a WQ stride equals 2^log_wq_stride.
	32	31:00			dbr_addr[63:32]	Physical address bits of DB Record
	32	31:00			dbr_addr[31:0]	Physical address bits of DB Record
	32	31:00			hw_counter	Current HW stride index Points to the next stride to be consumed by HW). Bits [31:16] are reserved. This field is available through the QUERY commands only. Otherwise reserved.

	32	31:00			sw_counter	Current SW WQ WQE index Points to the next stride to be produced by SW. Bits [31:16] are reserved. This field is available through the QUERY commands only. Otherwise reserved.
	16	31:16			lwm	Limit Water Mark when WQE count drops below this limit, an event is fired. 0- Disabled
	5	15:11			log_wq_pg_sz	Log (base 2) of page size in units of 4Kbyte
	5	10:06			log_wq_sz	WQ size in Bytes is 2^(log_wq_size + log_wq_stride) log_wq_sz must be less than HCA_CAP.log_max_wq_size
	6	05:00				reserved
	32	63:32			pas[...]	Array of physical address structure (PAS) that map the work queue
	20	31:12				
	12	11:00				reserved

Table 4-13: Query SQ CSR

4.1.20 CREATE RQ CSR

The process is the same with Query HCA Capability CSR.

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name	Description
RQ Context						
	1	31			rlkey	Reserved LKey enable. When set the reserved LKey can be used on the SQ.
	1	30			vlan_strip_disable	VLAN Stripping Disable 0 - strip VLAN from incoming Ethernet frames 1 - do not strip VLAN from incoming Ethernet frames

	1	29			flush_in_error_en	If set, and when SQ transitions into error state, the hardware will flush in error WQEs that were posted and WQEs that will be posted to SQ. Otherwise, when SQ enters an error state HW is not forced to flush in error all the WQEs.
	24	28:6			user_index	User_index - an opaque identifier which software sets, which will be reported to the Completion Queue. Reserved if HCA_CAP.cqe_version==0.
	5	5:0				reserved
	4	31:28			state	SQ state 0x0: RST 0x1: RDY 0x3: ERR For QUERY_RQ - the current state is returned For MODIFY_RQ - the requested state is specified
	4	27:24			mem_rq_type	Memory RQ Type 0x0: MEMORY_RQ_INLINE - Inlined memory queue 0x1: MEMORY_RQ_RMP - RMP, RQ points to a remote memory pool
	24	23:0			user_index	User_index - an opaque identifier which software sets, which will be reported to the Completion Queue. Reserved if HCA_CAP.cqe_version==0.
	24	31:08			rmpn	RMP number. Reserved for non RMP RQs
	8	07:00				reserved
	24	31:08			cqn	Completion Queue Number
	8	07:00				reserved
Work Queue Context, See Work Queue Context table in Create SQ CSR						

Table 4-14: Query RQ CSR

4.1.21 CREATE QP

For creating QP, kindly follow the link regarding QP Context and HCA view. For more information, kindly click the appropriate link. Chapter 9. QP Context, for host to device or HCA view, kindly see 7.2 Queue Pair

Table 4-1: List Of MailBox Command Called when running IB

List of mailbox command called when running the ib_read_bw rdma perftest (client side)		
OpCode	**Command Name**	**Description**
0x802	ALLOC_UAR	User Address Register (Allocated from Idx 0x0 - 0x7)
0x816	ALLOC_TRANSPORT	
0x754	QUERY_NIC_VPORT_CONTEXT	
0x101	QUERY_ADAPTER	
0x805	ACCESS_REG	
0x800	ALLOC_PD	
0x400	CREATE_CQ	
0x500	CREATE_QP	Used bfregion 0xC, uar_index 0x0
0x502	RST2INIT_QP	
0x503	INIT2RTR_QP	
0x504	RTR2RTS_QP	
0x403	MODIFY_CQ	
0x50a	2RST_QP	
0x501	DESTROY_QP	
0x401	DESTROY_CQ	
0x801	DEALLOC_PD	
0x817	DEALLOC_TRANSPORT_DOMAIN	
0x803	DEALLOC_UAR	

Table 4-15: List Of MailBox Command Called when running IB

4.2 Address Translation and Protection

Memory Protection flow is as follow:

1. LKEY/RKEY from WQE are used to locate the memory region entry and validate the memory region.
2. QPN from memory region entries are used to validate that the memory region is associated with the right QPN.
3. Virtual address and length from WQE are used to validate the associated virtual address and length are within the boundary.
4. Check the access rights on the memory region.
5. Perform the virtual to Physical translation.

The address translation offset table defines the protection attributes and physical address of the system memory being accessed.

The address is used to locate the Memory Translation Table (MTT)

The data structure for protection attributes:
- Access type
- Protection Domain

The size of the Memory Region Protection Table has dependencies with the Memory Translation Table.

The Memory Translation Table needs to be contiguous.

The Memory Translation Table can be shared by more than one queue. Depending on the application, multiple Memory Region Protection Tables can provide different protection attributes to the same Memory Translation Table.

Translation Steps:
1. Boundary check:
    a. Address is valid
    b. Address is invalid if BAR/address offset cannot be divided by boundary (e.g. 4K)
    c. Address is invalid if BAR/address is > Limit (e.g. 32) * boundary (e.g. 4K) as example if address is > 32*4K address is invalid
    d. The data structure for boundary check attribute :
        i. Base
        ii. Offset
        iii. Length
2. Operation check:
    a. Find the Queue : Find the number of queue using BAR/address/offset divided by boundary (e.g. 4K)
    b. Verify if the queue is already allocated;
    c. Memory protection check: Verify if the queue belongs to the BAR
    d. Find the MKey: Find the index to identify the MKey (Memory Key with 32-bit address space) from the Memory Protection Table (MPT). The MKey should already be programmed and can be changed by the application.

3. If Boundary and Operation checks are successful:
    a. Translate virtual address to physical address: Memory region base address - virtual address -> select 32-bit Mkey + offset + queue number (direct mode or indirect mode).
    b. The data structure for address translation attribute :
        i. page size
        ii. physical address

After protection attributes are checked and the address boundaries checked. The address translation uses the offset to map virtual to the physical address.

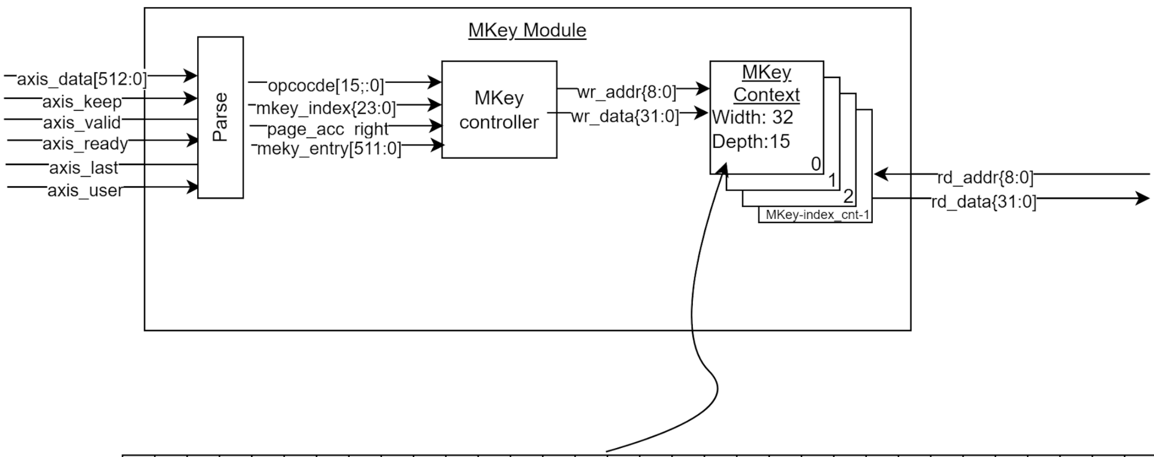
The physical address is used to locate the physical entry to provide the physical page address.

The page address is concatenated with the offset of the physical base address to form the physical system address.

Figure 4-2 MKey Context

Memory	Width -MKey (a)	Mkey Context Depth (b)	MKey_index_cnt (c)	Memory entry (b*c)	Total bit	XCZU19EG Memory (Mb)
MKey_	32	15	16,777,216	251,658,240	8,053,063,680	70.6

Parameter	Default Value	Description
MKey_index_cnt	TBD/1024	1)The parameter indicates the number of MKey index. 2)For example, if Mkey_index_cnt =523, there will be 512 entries of MKey. 3)The CREATE_MKEY mkey_index (offset 0X08) is 24-bit to support 16 Million (16,777,216 index) 4) For FPGA implementation assume 1024 Mkey. The Mkey memory is:1024*64B



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Offset
	fre															umr_e		rw	rr	lw	lr	access mode										00h
qpn																								mkey[7:0]								04h
																																08h
length64							pd																									0Ch
start_addr																																10-14h
len																																18-1Ch
																																20h
																																24h
																																28h
																																2Ch
																																30h
translations_octword_size																																34h
																											log_entity_size					38h
																																3Ch

Figure 4-3 MKey Context

4.4 RSNIC IP Modules

The *RSNIC-IP* has the following internal modules:.

- QP Manager
- WQE Handler
- CQE Generator
- IBH Processor
- IP Handler
- MR
- DMA

4.4.1 QP Manager

The QP Manager module main task is to handle the incoming doorbell from the host and schedule the work request. It handles the configuration for all the Queue Pairs thru an AXI4-Lite interface. It also decides across various SEND Queues and Caches the SEND Work Entries (WQEs). These WQEs are then provided to the WQE processor module for further processing. This module also handles the Queue Pair pointer updates in the event of retransmission.

Figure 4.4-1-1: QP Manager Module

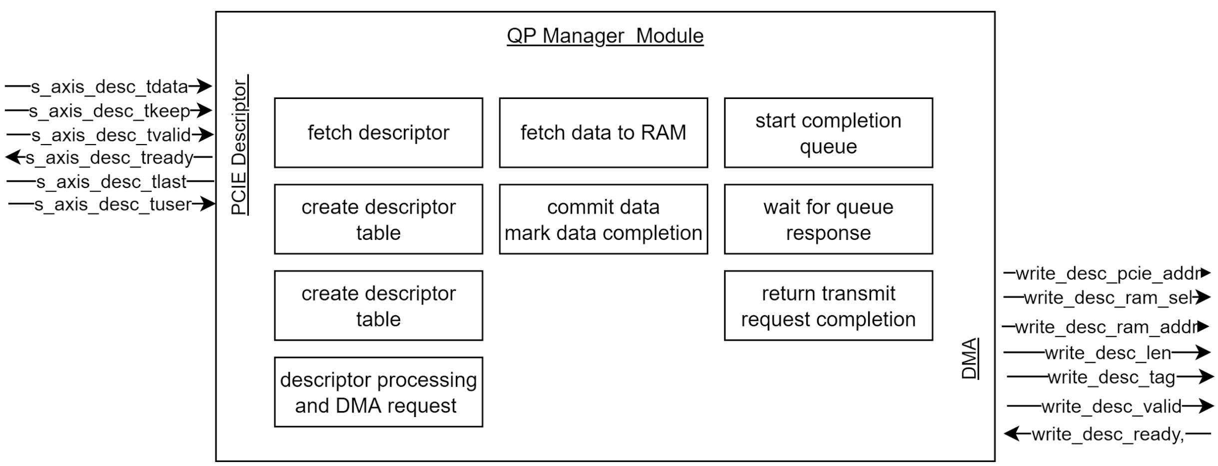


Figure 4-4-1-1: QP Manager Module

4.4.2 WQE Handler

The WQE Engine executes the work request from the QP Manager module and handles the following tasks:

1. Validates the incoming WQE for any invalid opcode, and
2. Check what kind of this Work Queue command is, whether it is sent or received. A send queue contains a pointer to buffer that has to be sent to the client and a receive queue contains a pointer to a buffer that will hold all the incoming messages.

WQ also handles the request engine queue from the host device driver. Upon receiving the CREATE_RQ as discussed in chapter 12 of the PRM, An input structure from the request is being managed by the engine.

Table 4.4-2-1: WQE Structure

Pls refer to page 78 of PRM

offset	UMR Work Request Format (WQE)
10-18	UMR Control Segment LR. LW, RR, RW
0	Byte Count
	LKey
	Local addr
	Local addr
N	Byte Count
	LKey
	Local addr
	Local addr

Table 4-4-2-1: WQE Structure

4.4.3 CQE Generator

The Completion Queue Entries generator module is responsible for creating completion entries and putting completion entries to the completion queue to notify the host that the requested work has been completed.

4.4.4 IBH Processor

The Infinib-and Header (IBH) processor module is assigned to execute or trigger the DMA once the construction of the RoCE v2 packet including all the required layers like Ethernet Header, UDP Header, IB Header and make this RDMA command as a payload. The IBH also covers the retransmission once it encounters an error from CRC or caused by timeout.

4.4.5 IP Handler

The IP Handler module receives the incoming RDMA packets and filters out the non-RMDA packets.

4.4.5.1 RDMA Memory Region

Figure 4-4-5-1 Memory Region

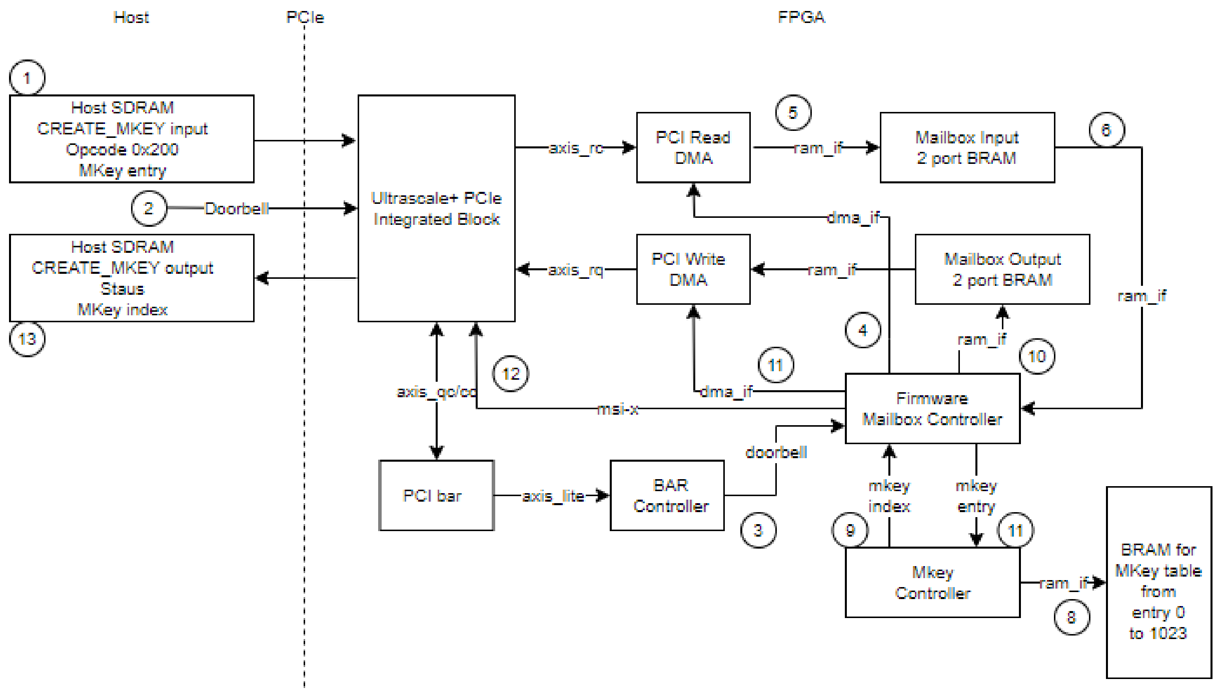


Figure 4-4-5-1 Memory Region

1. Host driver fills up the CREATE_MKEY to the mailbox input buffer in the Host SDRAM.
2. Host can assert the doorbell to notify the device.
3. Firmware Mailbox Controller will receive the doorbell notification.
4. Firmware Mailbox Controller will set up PCI Read DMA to get the mailbox input.
5. The PCI Read DMA transfers data from Host SDRAM to Mailbox input BRAM.
6. Firmware Mailbox Controller reads the data from Mailbox input BRAM.
7. Firmware Mailbox Controller provides the payload MKey Entry to the Mkey Controller.
8. Mkey Controller will store the Mkey Entry into the Mkey Table and get the index used.
9. Mkey Controller will provide the used Mkey index.
10. Firmware Mailbox Controller will respond using the Mailbox Output buffer.
11. Firmware Mailbox Controller will set up PCI Write DMA to send the mailbox output.
12. The PCI Write DMA transfers data from Mailbox output to Host SDRAM.
13. Firmware Mailbox Controller sends MSI-X to notify the Host.
14. Host Driver will read the status and mkey index from the Mailbox output.

4.4.6 CREATE SQ

Figure 4-4-6-1: Create SQ

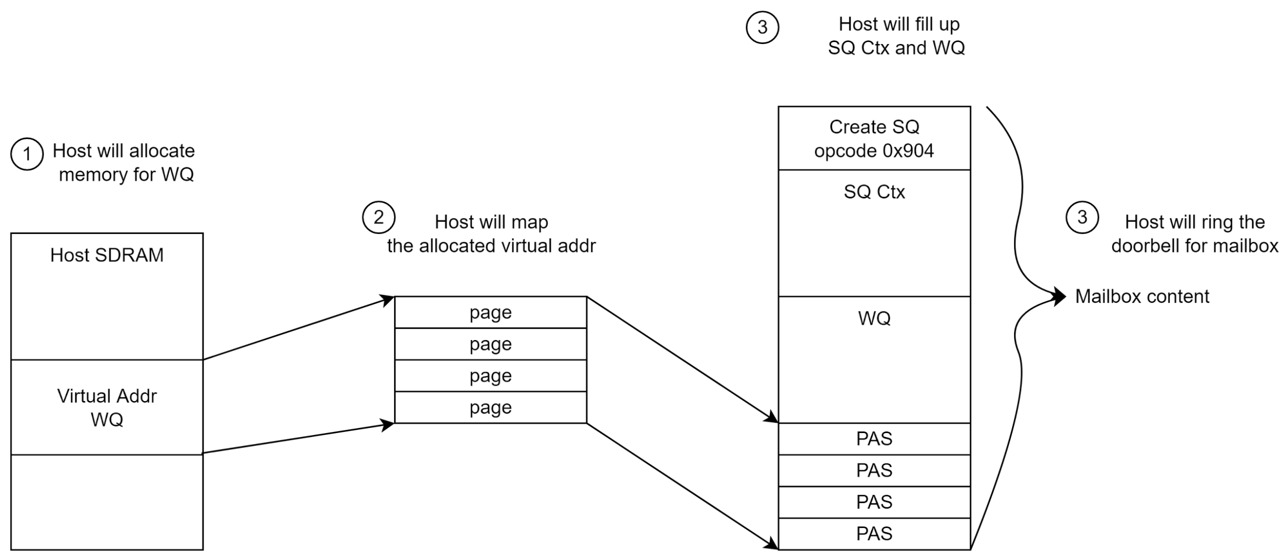


Figure 4-4-6-1: Create SQ

4.4.7 DMA

Figure 4-4-7-1: DMA

PCIe read DMA and PCIe write DMA will get the address and length for the descriptor which are used to prepare the data. The PCIe DMA will track the request from the PCIe and once the request is ready will let the Network read/write the data once it's ready.

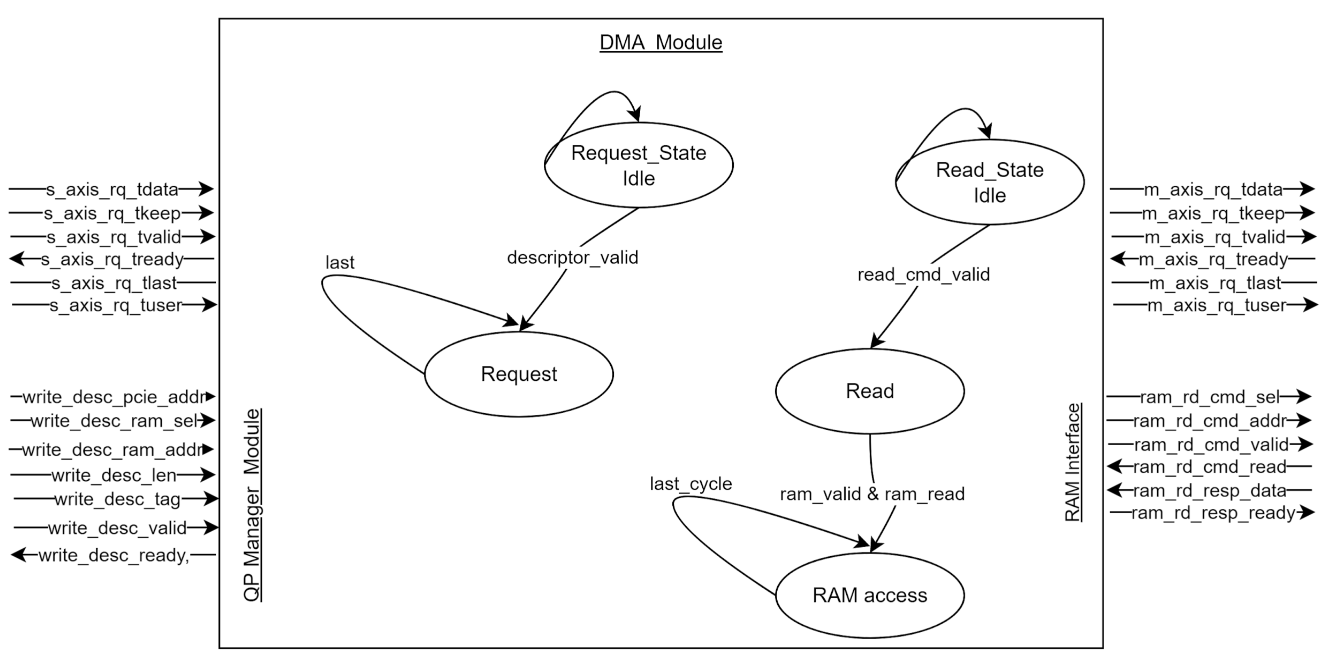


Figure 4-4-7-1: DMA

4.4.8 Completion Queue

The completion queue engine supports one completion queue per MSI-X vector per enabled PCI Express function and a maximum of MAX_COMPLETION_QUEUE.

Each completion queue is associated with a separated interrupt . The number of completion queues that software uses is the minimum number of MSI-X vectors available.

Software uses multiple completion queues to distribute the completion work. Each completion queue is individually assigned a completion via the CREATE or MODIFY operation.

The completion queues are a group of entries located in a virtually continuous buffer in host memory.

Completion queues entries should match the maximum number of active queues that are assigned. Each completion queue guarantees that it generates a maximum of one completion entry without acknowledgement that the entry has been consumed.

Completion queues are not checked for overflow conditions so it is necessary for software to size the number of entries correctly or risk completion events can be corrupted.

The start condition for software shows the completion queue index is set to 0x0.

The queue engine writes to the completion queue index specified by the head pointer.
Once an interrupt is received that tells a new completion queue is available, software reads the completion queue index and increments the completion queue index.

Software will process all valid completion queues until software reaches an invalid entry.
The software will track the index of the invalid entry in the completion queue index.

The completion entries that come later with a valid completion queue entry will use the value on the doorbell page ACK register to be written to notify the completion engine the entry is available for use by the hardware. The hardware will increment on-chip tail pointer content

Figure 4-4-8-1 Queue

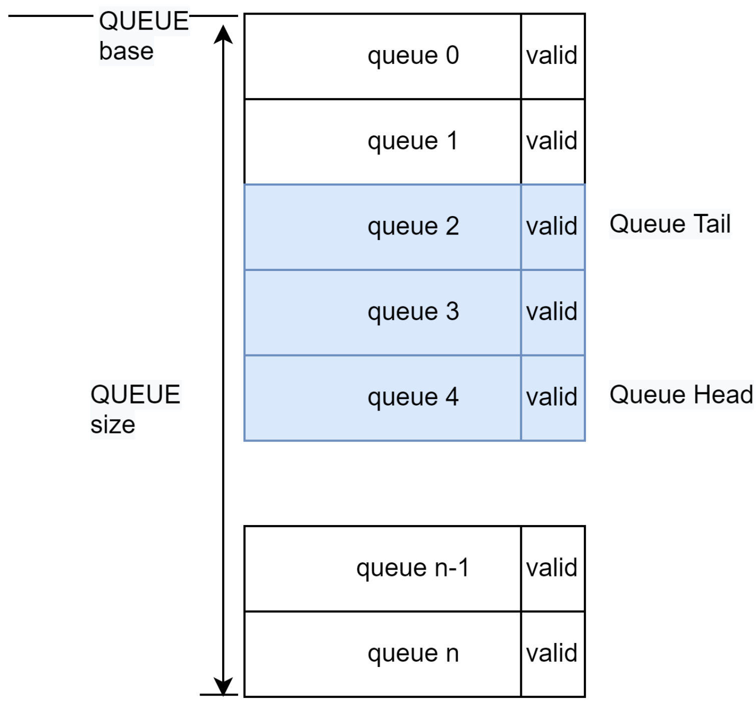


Figure 4-4-8-1: Queue

Figure 4-4-8-2: Completion Queue Module

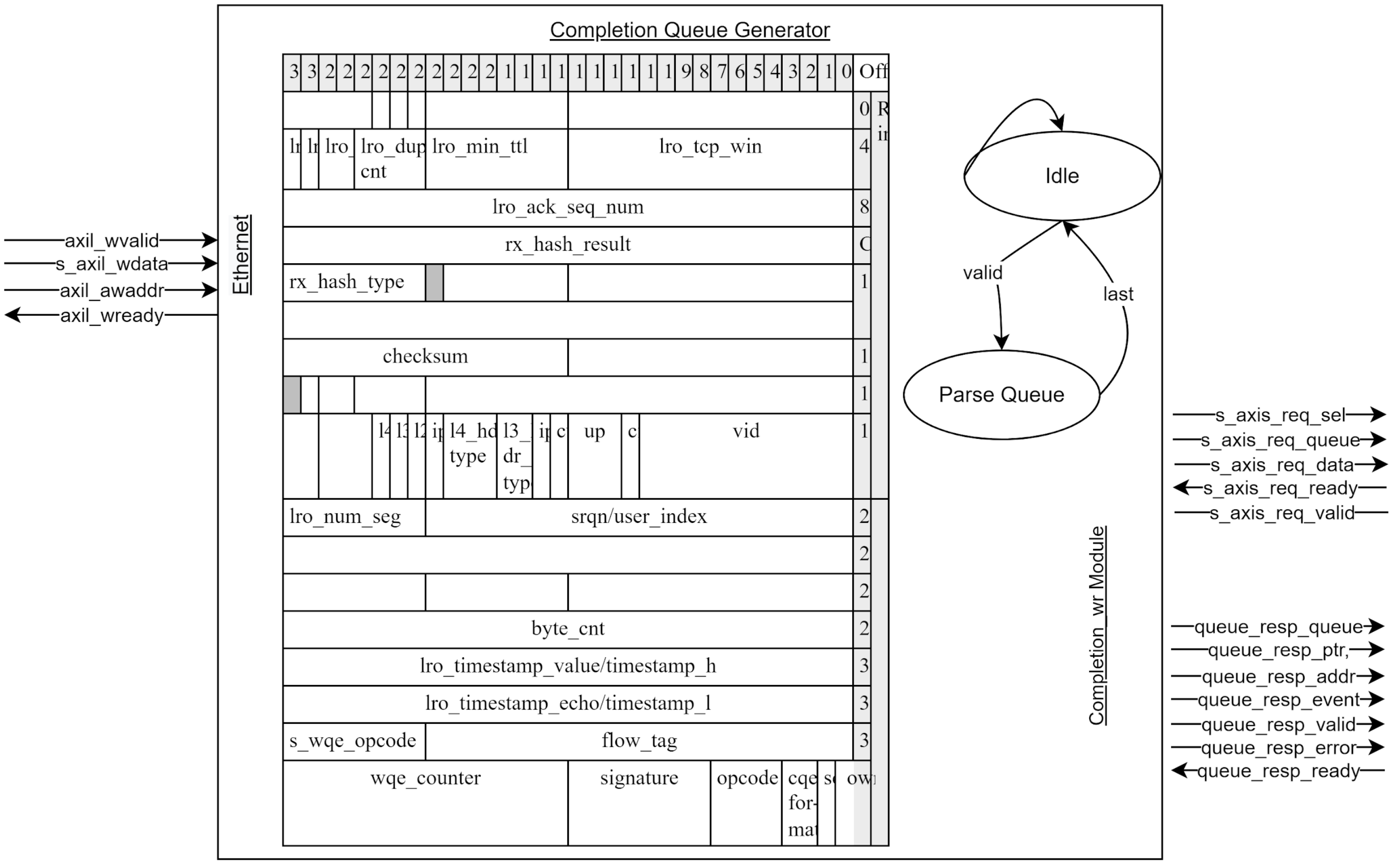


Figure 4-4-8-2: Completion Queue Module

4.4.9 Virtual to Physical Translation

This section will detail the translation of virtual address to physical address. Shown in Figure 4-4-9-1 is a scenario of virtual address to physical address translation.

Figure 4-4-9-1: Virtual to physical translation

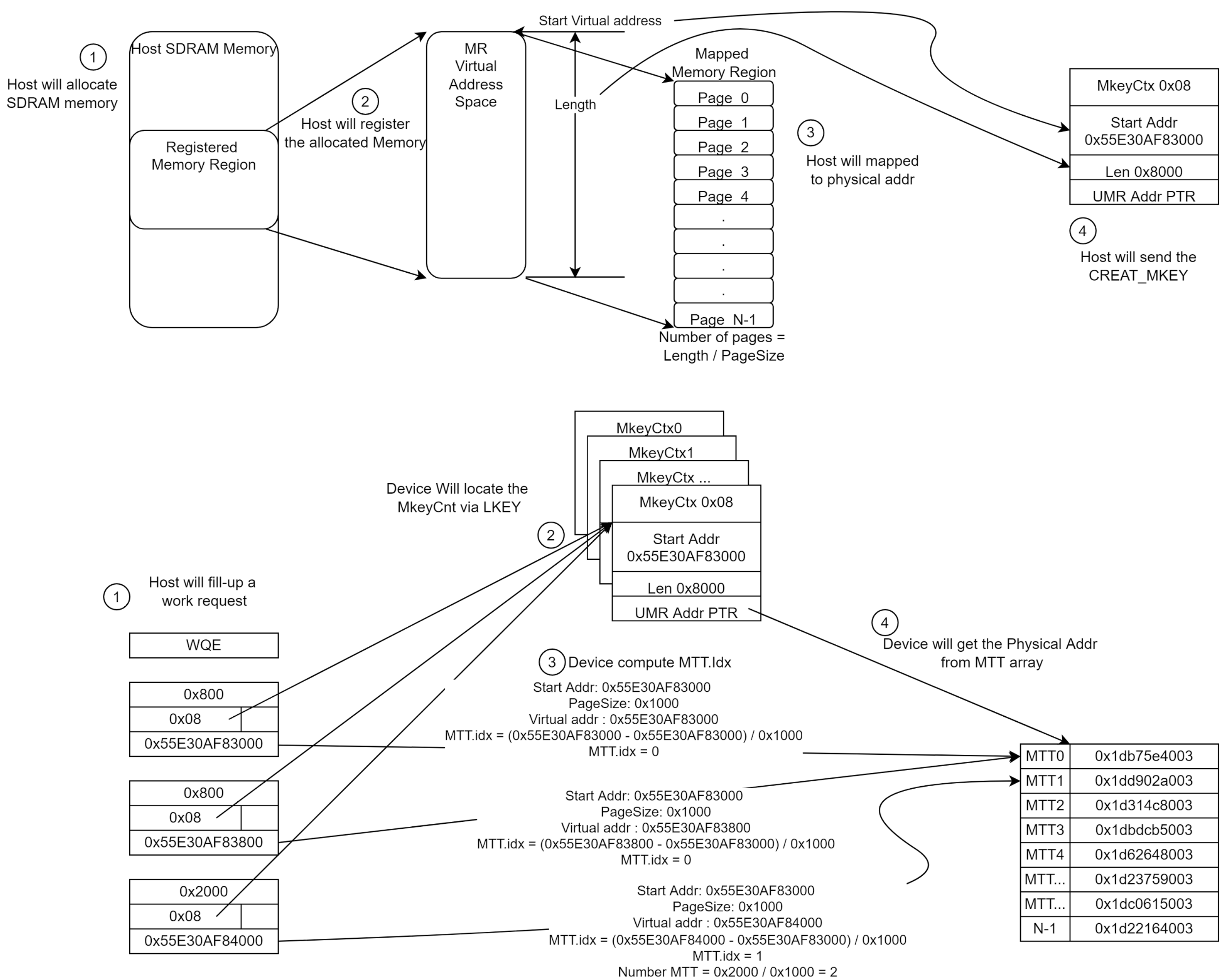


Figure 4-4-9-1: Virtual to Physical Translation

4.4.10 Packet Retransmission

This section will detail the packet retransmission by rebuilding WQE to retransmit the packet. Shown in Figure 4-4-10-1 is a scenario of packet retransmission.

Figure 4-4-10-1: Packet retransmission

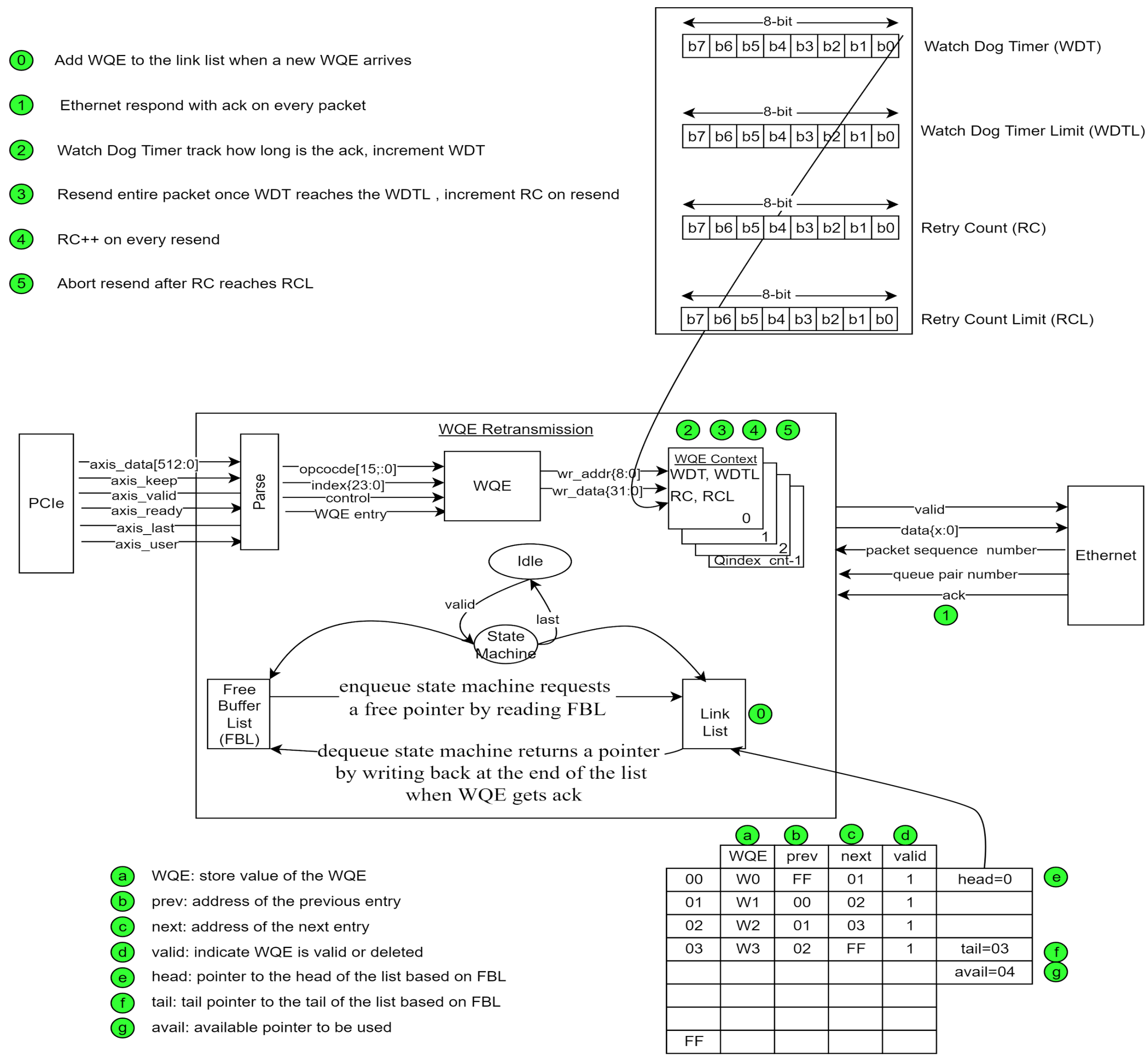


Figure 4-4-10-1: Packet Retransmission

Chapter 5. PCI Interface

The device accesses the host processor through a 16x PCI Express 3.0 bus, capable of consuming/driving data at a rate of 16Gbyte/s full-duplex.
Each function (be it physical or virtual) exposes a single BAR called. A prefetchable BAR used to post control data path commands. Both the initialization segment and the UAR pages are implemented on this BAR. This is BAR0 in the PCI header.

5.1 PCI Initialization Sequence

- PCI Resource Start
- PCI Enable Device (Initialize device before it's used by a driver)
- Request BAR (Reserve PCI I/O and Memory Resources, BAR0 0xC0010000)
- PCI Set Master (Enable Bus Mastering)

Figure 5-1: PCI Initialization

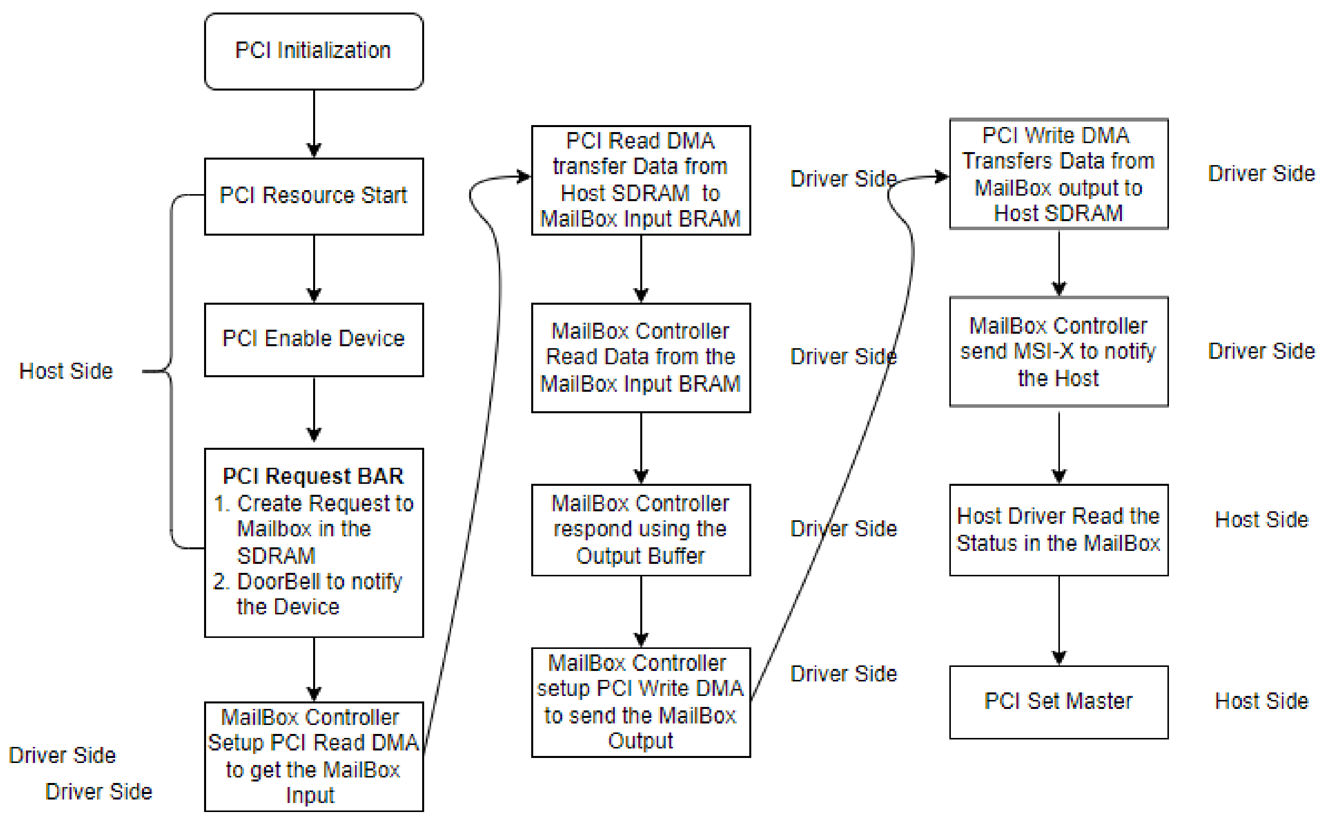


Figure 5-1: PCI Initialization

5.2 Capabilities Reporting

PCI Express Base Specification defines a set of capabilities which should be reported when made available by the device. These capabilities are reported through a special structure of linked-list items. The first capability is pointed in offset 0x34 in the configuration space, and the last capability is indicated by setting its “next capability” pointer to 0x00.

5.3 Initialization Segment

The initialization segment is located at offset 0 of HCA BAR. The device uses this segment in the initialization flow and to perform command doorbell.

Table 5-3-1: Internal data structure

Offset	No. of Bits	Bits Alloc	Default Values	Access	Name
00h	16	31:16	0x1600	RO	fw_rev_major
	16	15:00	0x0E00	RO	fw_rev_minor
04h	16	31:16	0x0500	RO	fw_rev_subminor
	16	15:00	0xB40F	RO	cmd_interface
08h	32	31:00	0x01000000	RW	cmdq_phy_addr[63:32]
0Ch	4	31:28	0x00	RW	log_cmdq_stride
	4	27:24	0x00	RW	log_cmdq_sz
	2	23:22	0x00	RW	nic_interface_supported
	22	21:00	0x56010	RW	cmdq_phy_addr[31:12]
10h	32	31:00	0x01000000	RW	command doorbell vector
14h	3	31:29	0x00	RW	nic_interface_supported
	1	28	0x00	RW	initializing
	28	27:00			reserved
18h	32	31:00		RW	internal_timer_h
1Ch	32	31:00		RW	internal_timer_l

Table 5-3-1: Internal Data Structure for PCI Initialization segment

Table 5-3-2: Config Space Initial Values

(Register values are based on bus trace provide on 02/06)

<table>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>Off set</th></tr>
<tr><td colspan="8" rowspan="2">Vendor ID
0x15B3</td><td>0</td></tr>
<tr><td>1</td></tr>
<tr><td colspan="8" rowspan="2">Device ID
0x1015</td><td>2</td></tr>
<tr><td>3</td></tr>
<tr><td>IDSEL Steeping /Wait Cycle Control
0</td><td>Parity Error Response
0</td><td>VGA Palette Snoop
0</td><td>Memory Write and Invalidate
0</td><td>Special Cycle Enable
0</td><td>Bus Master Enable
1</td><td>Memory Space Enable
1</td><td>I/O Space Enable
0</td><td>4</td></tr>
<tr><td colspan="5">Reserved
00</td><td>Interrupt Disable
1</td><td>Fast Back to Back Transactions
0</td><td>SERR# Enable
0</td><td>5</td></tr>
<tr><td>Fast Back to Back Transactions Capable
0</td><td>Reserved
0</td><td>66 Mhz Capable
0</td><td>Capabilities List
1</td><td>Interrupt Status
0</td><td colspan="3">Reserved
0</td><td>6</td></tr>
<tr><td>Detected Parity Error
0</td><td>Signaled System Error
0</td><td>Received Master Abort
0</td><td>Received Target Abort
0</td><td>Signaled Target Abort
0</td><td colspan="2">DEVSEL Timing
0</td><td>Master Data Parity Error
0</td><td>7</td></tr>
<tr><td colspan="8">Revision ID
00</td><td>8</td></tr>
<tr><td colspan="8" rowspan="3">Class Code
020000 (PCI Device)</td><td>9</td></tr>
<tr><td>10</td></tr>
<tr><td>11</td></tr>
<tr><td colspan="8">Cache Line size
10</td><td>12</td></tr>
<tr><td colspan="8">Master Latency Timer
00</td><td>13</td></tr>
<tr><td colspan="8">Header Type
80</td><td>14</td></tr>
<tr><td colspan="8">BIST
00</td><td>15</td></tr>
</table>

Base Address Register 0 E200000C	16 17 18 19
Base Address Register 1 00000000	20 21 22 23
Base Address Register 2 00000000	24 25 26 27
Base Address Register 3 00000000	28 29 30 31
Base Address Register 4 00000000	32 33 34 35
Base Address Register 5 00000000	36 37 38 39
Cardbus CIS Pointer 00000000	40 41 42 43
Subsystem Vendor ID 15B3	44 45
Subsystem ID 0057	46 47
Expansion ROM Base Address FCFC0000	48 49

	50
	51
Capabilities Pointer 60	52
Reserved 000000	53
	54
	55
Reserved 00000000	56
	57
	58
	59
Interrupt Line 0A	60
Interrupt Pin 01	61
Min_Gnt 00	62
Max_lat 00	63

Table 5-3-2: Config Space Initial Values

Chapter 6. DMA Operations

At the most basic level, the PCIe® DMA engine typically moves data between host memory and memory that resides in the FPGA which is often (but not always) on an add-in card. When data is moved from host memory to the FPGA memory, It is called DMA Write. When data is moved from FPGA Memory to Host Memory, it is called DMA Read.

These terms help delineate which way data is flowing (as opposed to using read and write which can get confusing very quickly). The PCIe DMA engine is simply moving data to or from PCIe address locations.

In typical operation, an application in the host must move data between the FPGA and host memory. To accomplish this transfer, the host sets up buffer space in system memory and creates descriptors that the DMA engine uses to move the data.

Table 6-1: DMA Descriptor

DMA Descriptor			
Bit Index	**Data Direction**	**Name**	**Description**
	In	clk	Clock
	In	rst	Reset
	In	en	Enable
3:0	In	we	Write Enable
15:0	In	addr	Address
31:0	In	din	Data In
31:0	Out	dout	Data Out

Table 6-1: DMA Descriptor

Table 6-2: Read Descriptor

Read Descriptor			
Bit Index	**Data Direction**	**Name**	**Description**
63:0	Out	read_desc_pcie_addr	Read Descriptor Pcie Address
18:0	Out	read_desc_ram_addr	Read Descriptor RAM Address
15:0	Out	read_desc_len	Read Descriptor Length
14:0	Out	read_desc_tag	Read Descriptor Tag

	Out	read_desc_valid	Read Descriptor Valid
	In	read_desc_ready	Read Descriptor Ready
14:0	In	read_desc_status_tag	Read Descriptor Status Tag
	In	read_desc_status_valid	Read Descriptor Status Valid

Table 6-2: Read Descriptor

Table 6-3: Write Descriptor

Write Descriptor			
Bit Index	**Data Direction**	**Name**	**Description**
63:0	Out	write_desc_pcie_addr	Write Descriptor Pcie Address
18:0	Out	write_desc_ram_addr	Write Descriptor RAM Address
15:0	Out	write_desc_len	Write Descriptor Length
14:0	Out	write_desc_tag	Write Descriptor Tag
	Out	write_desc_valid	Write Descriptor Valid
	In	write_desc_ready	Write Descriptor Ready
14:0	In	write_desc_status_tag	Write Descriptor Status Tag
	In	write_desc_status_valid	Write Descriptor Status Valid

Table 6-3: Write Descriptor

Table 6-4: msi-x

msi-x			
Bit Index	**Data Direction**	**Name**	**Description**
	In	cfg_interrupt_msi_sent	Config Interrupt msi Sent
31:0	Out	cfg_interrupt_msi_data	Config Interrupt msi-x Data
63:0	Out	cfg_interrupt_msix_address	Config Interrupt msi-x Data
	Out	cfg_interrupt_msix_int	Config Interrupt msi-x Int

Table 6-4: msi-x

Figure 6-1: H/W Architecture Diagram

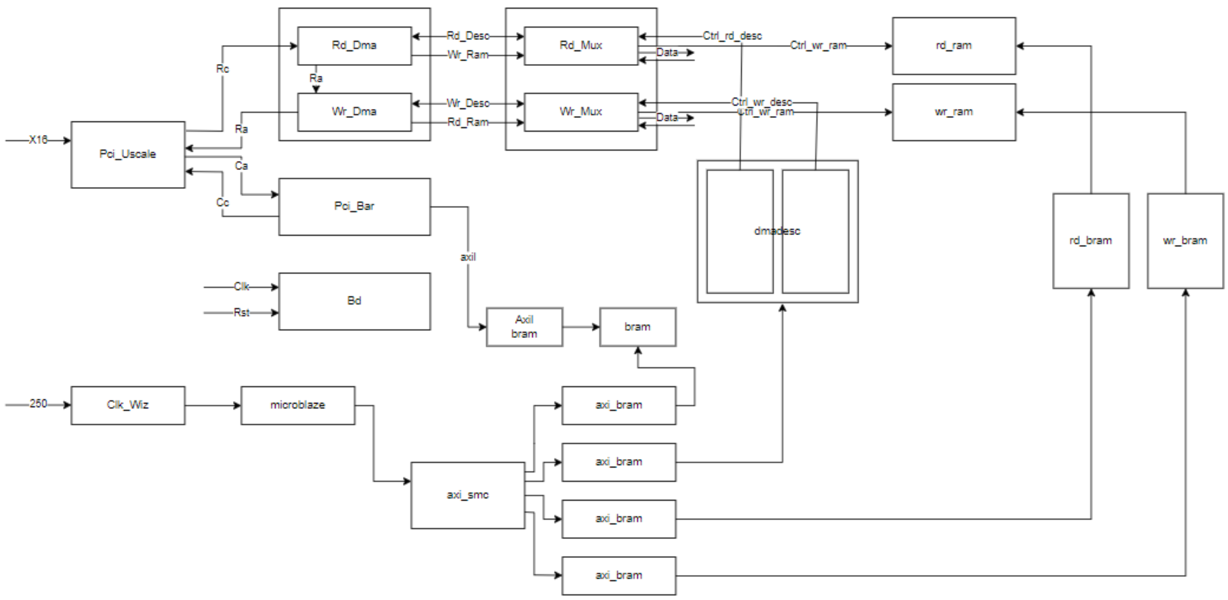


Figure 6-1: H/W Architecture Diagram

In Figure 6-1, the arrow direction represents master to slave. The top part is RTL and the bottom part is inside the cpu bd (Block Design). There are four cpu bd AXIL to BRAM interfaces going to RTL. The memory map is the following:

0xc000_0000 - 32KB BAR0 (dual port BRAM)
0xc001_0000 - DMA descriptors to control the PCI DMA
0xc002_0000 - 16KB Read RAM access (for Mailbox Input)
0xc003_0000 - 16KB Write RAM access (for Mailbox Output)

Chapter 7. Host to Device Interface

7.1 Relations of Queues and Events

Work Request (SQ or RQ) is posted to the HCA by writing a list of one or more Work Queue Elements (WQE) to the WQ and ringing the DoorBell, notifying the HCA that request has been posted.

This section describes the structure and management of WQs and the WQE format.

HCA WQ is an object containing the following entities:

1. SQ/RQ Context - Contains control information required by the device to execute I/O operations on that Context. These contexts are configured by SW at creation time.
2. Work Queue Buffer - A virtually-contiguous memory buffer allocated when creating the SQ/RQ:
   a. Send Queue - A virtually-contiguous circular buffer accessible by user-level software and used to post send requests. Maximum supported SQ size is retrieved by the QUERY_HCA_CAP command (log_max_qp_sz).
   b. Receive Queue - A virtually-contiguous circular buffer accessible by user-level software and used to post and receive requests. Maximum supported RQ size is retrieved by the QUERY_HCA_CAP command (log_max_qp_sz).
3. DoorBell Record - A structure containing information of the most recently-posted Work Request.

7.2 Work Queues Structure and Access

The device WQ is a virtually-contiguous memory buffer used by SW to post I/O requests (WQEs) for HCA execution. A WQ is composed of WQE Basic Blocks (WQEBBs); WQE size is modulo of WQEBB size.

Each WQ buffer contains the Send WQ and Receive WQ adjacently. The RQ resides in the beginning of the buffer; Figure 7-2-1 illustrates the structure of Work Queue Buffer.

The buffer must be aligned on WQEBB/Stride (the larger of the two); RQ must be aligned to 64B even if RQ stride is smaller than 64B. RQ can be of zero size. The offset of WQE in the first page (page_offset) is delivered to the device while configuring the SQ/RQ.

Figure 7-2-1: Work Queue Buffer Structure

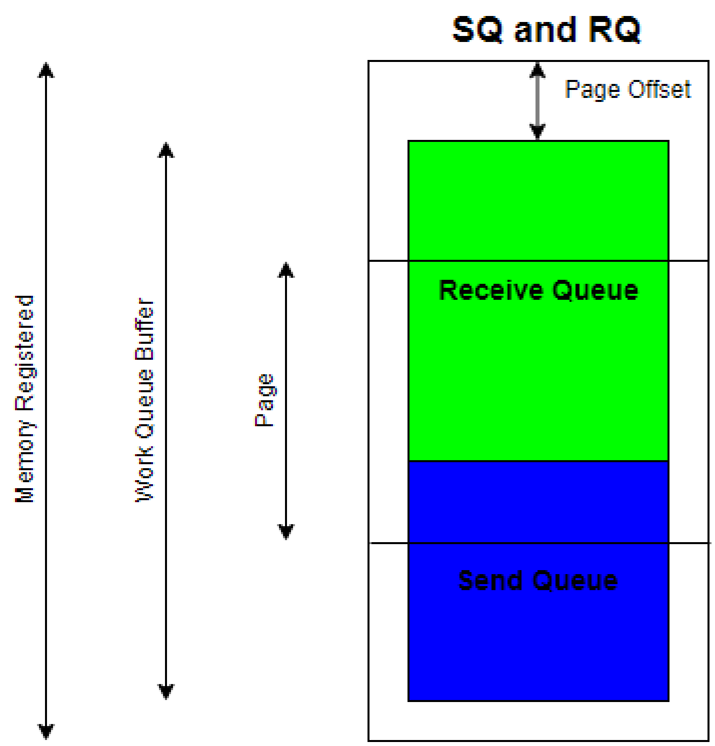


Figure 7-2-1: Work Queue Buffer Structure

7.3 Send Queue

The size of Send WQ is specified in units of Work Queue Basic Blocks (WQEBBs). The size of a WQEBB is 64 bytes. The size (or depth) of the SQ is a power-of-two number of WQEBBs, and therefore the queue itself is a power-of-two size (in bytes). The maximum SQ size in WQEs can be retrieved through the QUERY_HCA_CAP command (2log_max_qp_sz).

Figure 7-3-1 Send Work Queue With 3 WQEs Posted

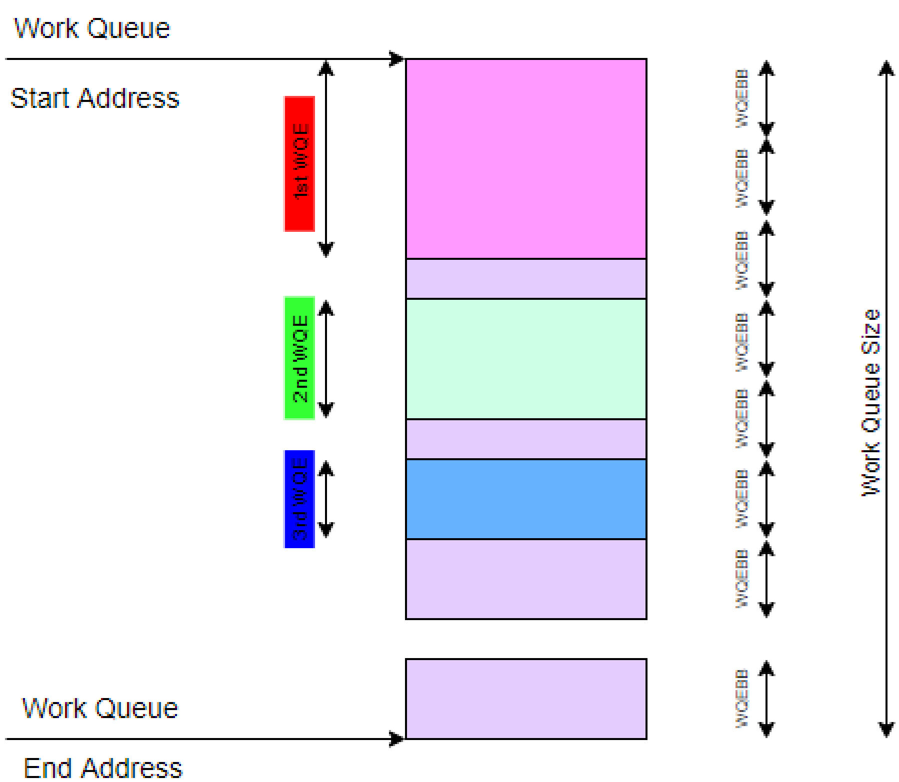


Figure 7-3-1: Send Work Queue With 3 WQEs Posted

SQ WQEs are identified by sq_wqebb_counter, which starts at zero and advances with each WQE post by the number of WQEBBs this WQE occupies and is stored in Send Doorbell Record. sq_wqebb_counter wraps around 0xFFFF.

7.4 Receive Queue

RQ is organized as a circular buffer containing Receive WQEs. Receive WQEs are placed at fixed stride in the Receive WQ buffer and are executed in the order of their appearance in the buffer. The stride is specified in the WQ in a multiple of 16-byte chunks; the number of these chunks must be a power of two. The maximum RQ size in strides can be retrieved through the QUERY_HCA_CAP command (2log_max_qp_sz). Maximum WQE size supported is retrieved by QUERY_HCA_CAP command. Can be 512B or smaller.

Each 16-byte chunk of Receive WQE contains a single scatter pointer and byte count. If WQ stride size is larger than the actual WQE, the scatter pointers list can be terminated with scatter pointer specifying a zero value in the byte_count field and L_Key = 0x00000100. This scatter pointer is interpreted by HW as end of the scatter list and is referred thereafter as Null scatter pointer.

Figure 7-4-1: Receive Work Queue With 3 WQEs Posted

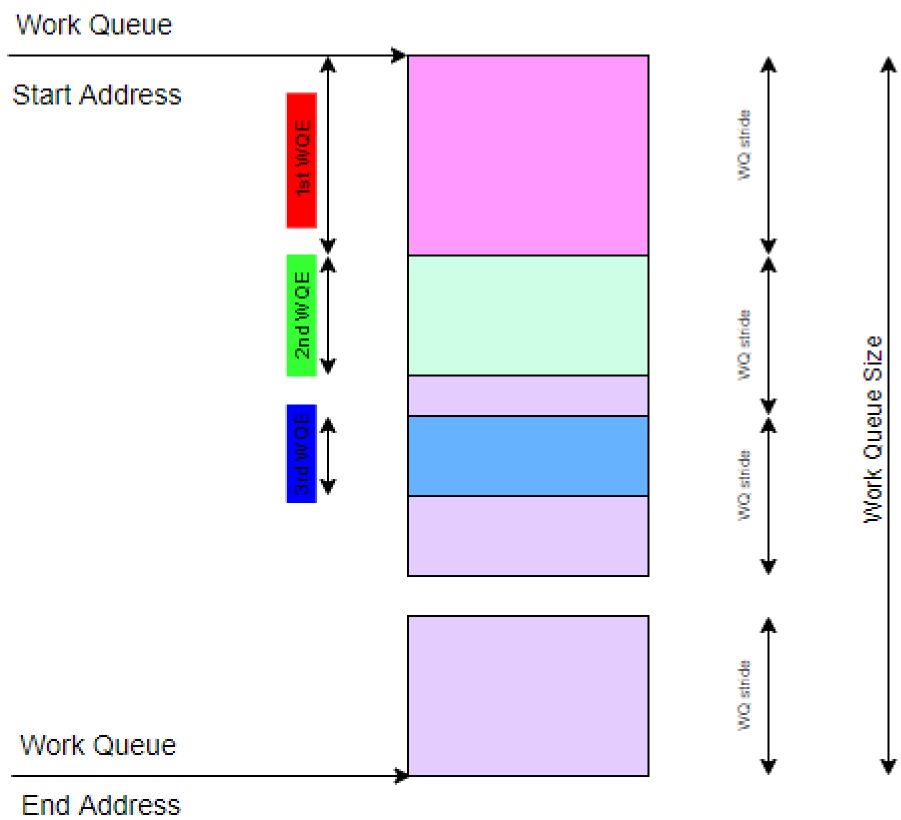


Figure 7-4-1: Receive Work Queue With 3 WQEs Posted

RQ WQEs are identified by the rq_wqe_counter, which starts at 0 and advances by 1 each time a WQE is posted to the RQ and stored to Receive Doorbell Record. rq_wqe_counter wraps around 0xFFFF.

7.5 Queue Pair

The QP is the virtual interface that the hardware provides to an IBA consumer; it serves as a virtual communication port for the consumer. The operation on each QP is independent from the others. Each QP provides a high degree of isolation and protection from other QP operations and other consumers. Thus a QP can be considered a private resource assigned to a single consumer. Please click the link for QP Context [Chapter 9. QP Context](#).

Figure 7-5-1: Communication Interface

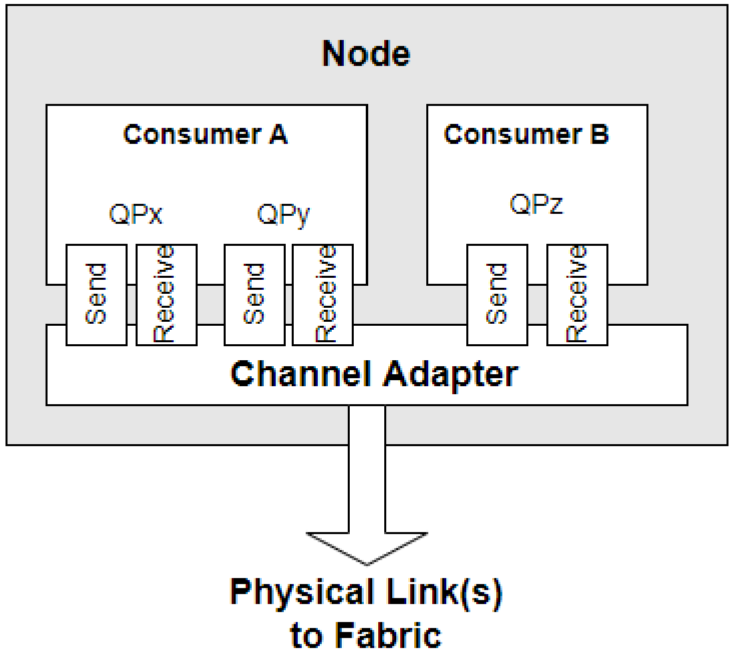


Figure 7-5-1: Communication Interface

The consumer creates this virtual communication port by allocating a QP and specifying its class of service. Communication takes place between a source QP and a destination QP. For connection oriented service, each QP is tightly bound to exactly one other QP, usually on a different node. The consumer initiates any communication establishment necessary to bind the QP with the destination QP and configures the QP context with certain information such as Destination LID, service level, and negotiated operating limits.

The consumer posts work requests to a QP to invoke communication through that QP.

7.6 Create Queue Pair

Creates a QP for the specified HCA. A set of initial QP attributes must be specified by the Consumer.

Figure 7-6-1: Create Queue Pair

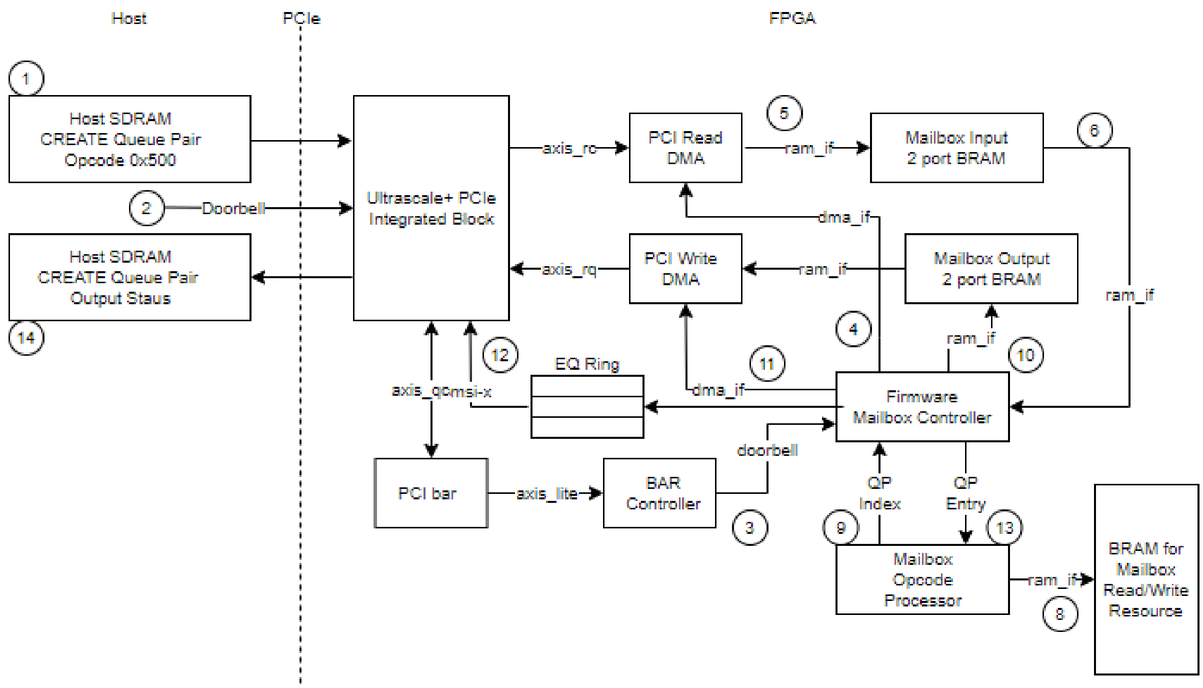


Figure 7-6-1: Create Queue Pair

1. Host driver fills up the Create Queue Pair (Allocated Memory Address and QPN number) to the mailbox input buffer in the HOST SDRAM.
    - Its Memory Allocation is depending on when the received queue and SQ are on the same offset or separate from memory allocation.
    - If there is Receive Queue Count, the next 4k alignment is the Send Queue. That's how the Memory is partitioned.
    - If the Offset or RQ Count is 0, it means the base for the QP is the Send Queue.
2. Host can assert the Doorbell (Command Index, Doorbell BAR Offset) to notify the device.
3. Firmware Mailbox Controller will receive the Doorbell notification.
4. The Firmware Mailbox Controller will set-up PCI Read DMA to get the mailbox input.
5. The PCI Read DMA transfers data from Host SDRAM to Mailbox input BRAM.
6. Firmware Mailbox Controller reads the data from Mailbox input BRAM.
7. Firmware Mailbox Controller provides the payload Create Queue Pair Controller.
8. Queue Pair Controller will store the Create Queue Pair Entry into the Queue Pair Table and get the index used.
9. Mailbox Opcode Processor will provide the payload used Queue Pair index.
10. Firmware Mailbox Controller will respond using the Mailbox Input buffer.
11. The PCI Write DMA transfers data from mailbox output to Host SDRAM.
12. Firmware Mailbox Controller send EQ Entry and MSI-X to notify the Host

13. Firmware Mailbox Controller will provide the payload QP entry.
14. Host Driver will read the status and Queue Pair index from the mailbox output

7.7 Completion Queue

A CQ can be used to multiplex work completions from multiple work queues across queue pairs on the same HCA.

This method shall support Completion Queue (CQ) as the notification mechanism for Work Request completions. A CQ shall have zero or more work queue associations. Any CQ shall be able to service send queues, receive queues, or both. Work queues from multiple QPs shall be able to be associated with a single CQ.

7.7.1 Creating a Completion Queue

Completion Queues are created through the Channel Interface as shown in Figure 7-4.

The maximum number of Completion Queue Entries (CQEs) that may be outstanding on a CQ must be specified when the CQ is created; this is known as the CQ's capacity. The actual capacity is returned through the Channel Interface. If the number of CQEs outstanding on a CQ is equal to its capacity, and another entry is added, the CQ overflows. It is the responsibility of the Consumer to ensure that the capacity chosen is sufficient for the Consumer's operations; it must, in any case, arrange to handle an error resulting from CQ overflow.

7.7.2 Completion Queue Attribute

The only Completion Queue attribute is the capacity of the CQ. This attribute can be retrieved through the Query Completion Queue Verb.

Note that no Verb is provided which retrieves a CQ's set of associated Work Queues; the consumer is responsible for keeping track of this information, if needed.

Figure 7-7-2-1: Relation of QP, CQ and EQ

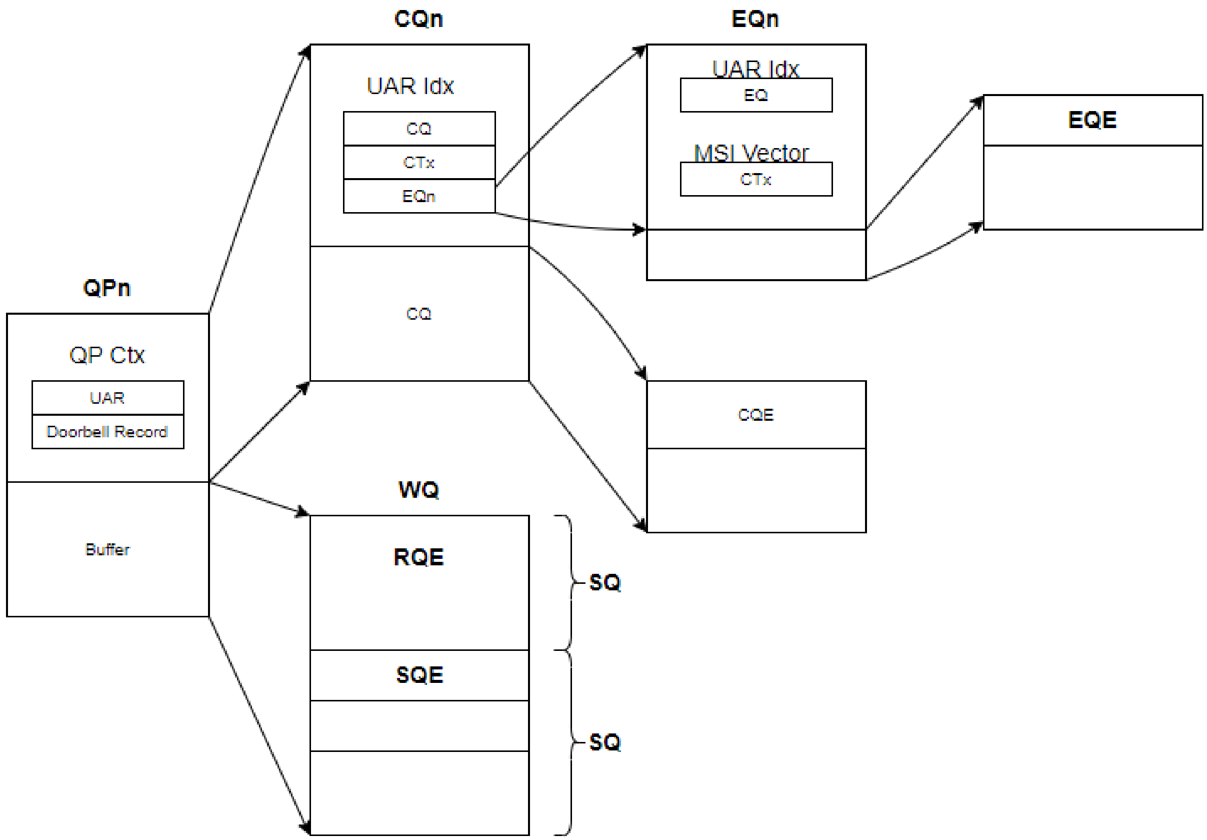


Figure 7-7-2-1: Relation of QP, CQ and EQ

7.8 Address Translation and Protection Enhancement

7.8.1 Lightweight Memory Registration

Lightweight memory registration enables the creation of virtually-contiguous address spaces out of disjoint (non-contiguous) chunks of memory region(s) already registered with the HCA.

With the adapter device, this is done with a User Mode Memory Registration (UMR) Work Request posted to a SQ. This is the “right way” to execute lightweight memory registration on the device.

7.8.2 User-Mode Memory Registration (UMR)

UMR is a mechanism to alter the address translation properties of MKeys by posting Work Requests on SQs. The key advantage of this mechanism versus prior fast registration implementations is that this operation can be executed by non-privileged software and the granularity (size and alignment) of memory sections specified by the KLM entries are not constrained (in contrast to page or block granularity previously required).

Figure 7-8-2-1: UMR Relation to MKey and SQ

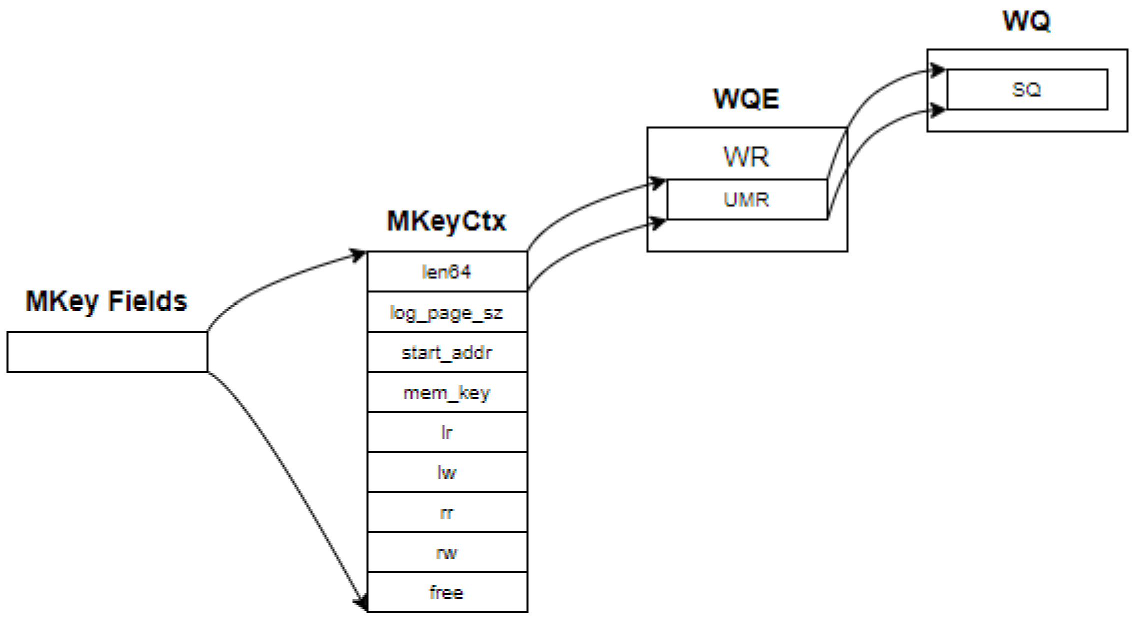


Figure 7-8-2-1: UMR Relation to MKey and SQ

7.8.3 User Access Region (UAR)

The isolated, protected and independent direct access to the HCA HW by multiple processes is implemented via User Access Region (UAR) mechanism.

Each function exposes a single BAR called HCA BAR. Both the initialization segment and the UAR pages are implemented on this BAR. This is BAR0 in the PCI header.

The UAR is part of PCI address space that is mapped for direct access to the HCA from the CPU. UAR is comprised of multiple pages, each page containing registers that control the HCA operation. UAR mechanism is used to post execution or control requests to the HCA. It is used by the HCA to enforce protection and isolation between different processes

Figure 7-8-3-1: UAR Relation to WQ and Doorbell from HCA and PCI Mapping

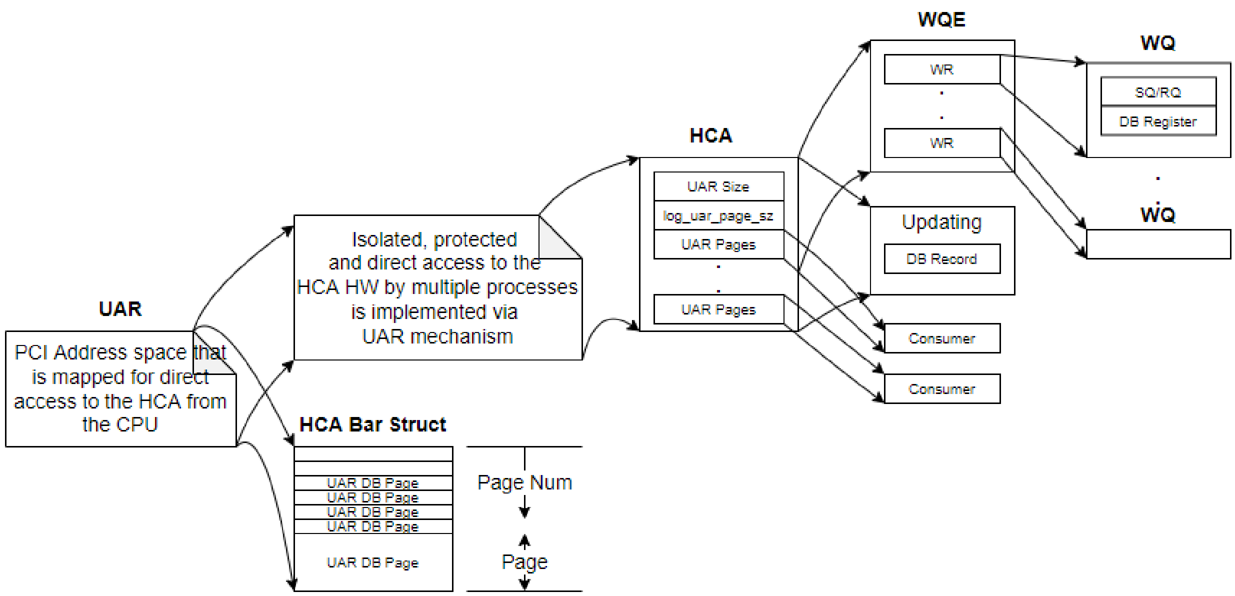


Figure 7-8-3-1: UAR Relation to WQ and Doorbell from HCA and PCI Mapping

Figure 7-8-3-2: Elaborating UAR Functionality

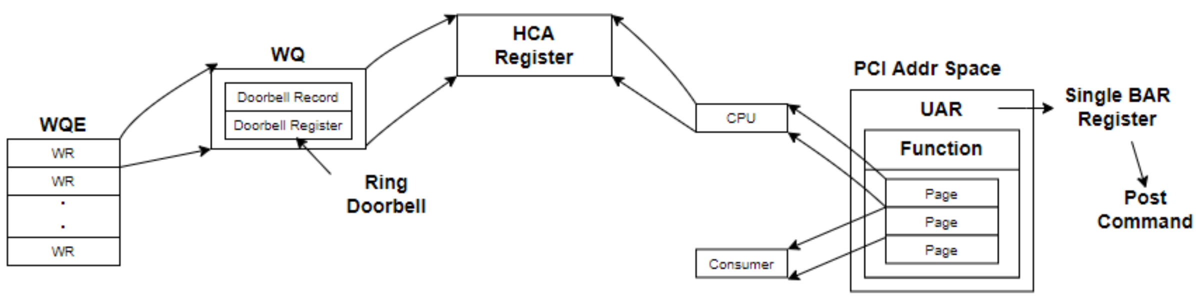


Figure 7-8-3-2: Elaborating UAR Functionality

7.9 MKey

An MKey is a 32-bit address space identifier consisting of two fields - index and key. The Index field identifies the MKey. The key field can be altered by the application program for various reasons such as a generation identifier when an MKey is reused.

Chapter 8. Memory Map

Current build memory map.

Table 8-1: Current Build Memory Map

Memory Map		
Address	**Capacity**	**Description**
0xC0000000	64K	BAR 0 (bram0)
0xC0010000	64K	DMA Descriptor (bram1)
0xC0020000	64K	Mailbox Input RAM (rd_ram or bram2))
0xC0030000	64K	Mailbox Output RAM (wr_ram or bram 3))
0xC0040000		Rx_ram or bram4
0xC0050000		Tx_ram or bram5
0xC0060000		CMAC Base Address or bram6

Table 8-1: **Current Build Memory Map**

Chapter 9. Memory Configuration

This chapter will show the memory computation or configuration.

Figure 9-1: WQEBB Memory Computation

How Much Memory will be consumed for 1024 WQEBB			
QP Per WQEBB	Target Number of WQEBB	Bytes Per WQEBB	Total Memory
256	1024	64	16MB

Figure 9-1: WQEBB Memory Computation

Figure 9-2: Physical Address Space

Physical Address Space (PAS)				
1024 WQEBB	Contiguous Physical Address	Total PAS	Physical Address Size	Total Memory
16MB	4KB	4KB	8B	32KB

Figure 9-2: Physical Address Space

Figure 9-3: MKey Memory Computation

MKey - If we want to support 2GB of Memory per 1 MKey?				
Target Memory	Bytes Per Page	Total PAS	Physical Address Size	MKey
2GB	4KB	512K	8B	4MB

Figure 9-3: MKey Memory Computation

Figure 9-4: MKey Context Memory Computation

MKey Context Memory Computation			
MKey	Number of QP Context (Words)	Bytes per Word	Total Memory
512	64	4	132KB

Figure 9-4: MKey Context Memory Computation

Figure 9-5: CQ Context Memory Computation

CQ Context Memory Computation			
Number of CQ	Number of QP Context (Words)	Bytes per Word	Total Memory
256	64	4	64KB

Figure 9-5: CQ Context Memory Computation

Figure 9-6: EQ Context Memory Computation

EQ Context Memory Computation			
Number of CQ	Number of QP Context (Words)	Bytes per Word	Total Memory
16	64	4	4KB

Figure 9-6: EQ Context Memory Computation

Figure 9-7: CQ Memory Computation

CQ is a Ring - Here is the Computation			
Number of CQ	CQE Per CQ	Bytes per CQE	Total Memory
256	512	64	8MB

Figure 9-7: CQ Memory Computation

Figure 9-8: EQ Memory Computation

EQ is a Ring - Here is the Computation			
Number of EQ	EQE per REQ	Bytes per EQE	Total Memory
16	1024	64	1MB

Figure 9-8: EQ Memory Computation

Figure 9-9: Total Memory Computation

Total Memory									
QP	CQ	EQ	MKey	PAS	MKey Context	QP Context	CQ Context	EQ Context	Total Memory
16MB	8MB	1MB	4MB	50KB	132KB	64KB	64KB	4KB	29.5MB

Figure 9-9: Total Memory Computation

Figure 9-10: CQ and EQ Ring

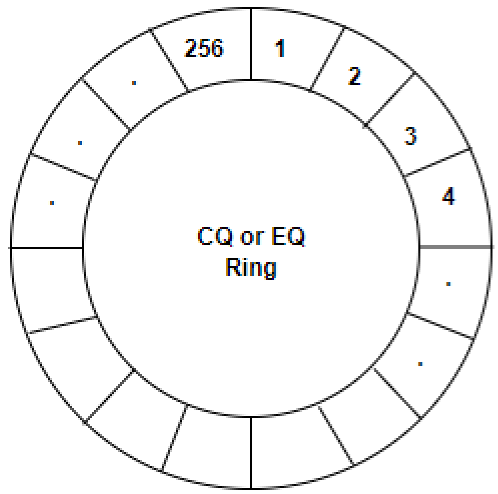


Figure 9-10: CQ and EQ Ring

Chapter 10. CMAC Implementation

RSNIC-IP uses UltraScale+ Integrated 100G ethernet from Xilinx for CMAC or CAUI Media Access Controller implementation. Using UltraScale and *RSNIC-IP* guaranteed high-performance interconnect for communications. CMAC is the module being used in ethernet which *RSNIC-IP* implemented by 4 lanes 25G each following RoCEv2 standard.

Figure 10-1: **RDMA IB Payload Sample**

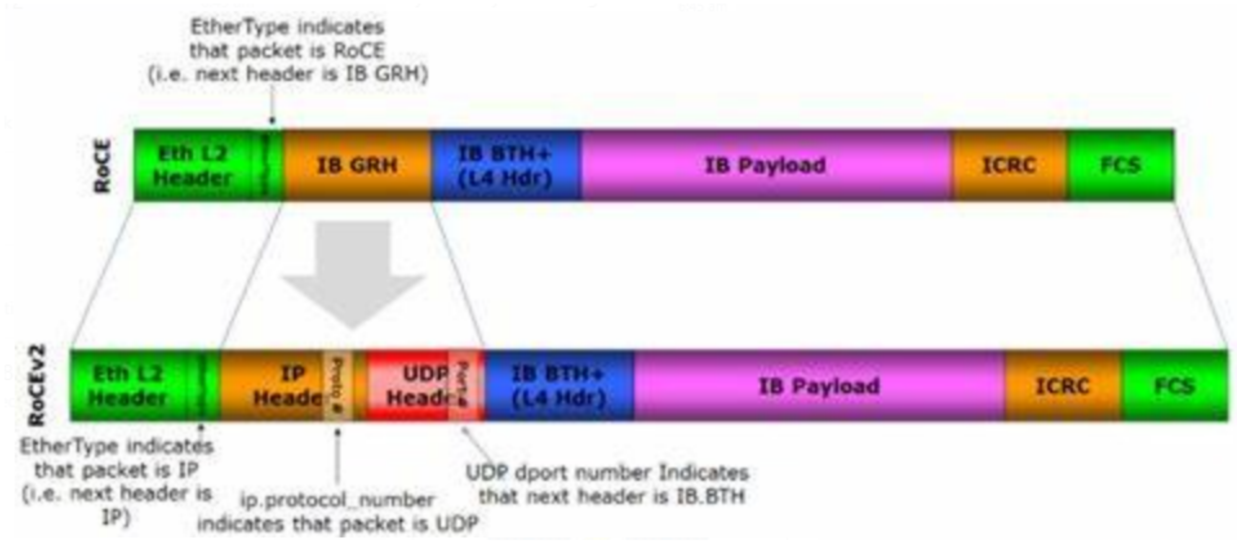


Figure 10-1: **RDMA IB Payload Sample**

Figure 10-2: RDMA Memory to Memory Transfer

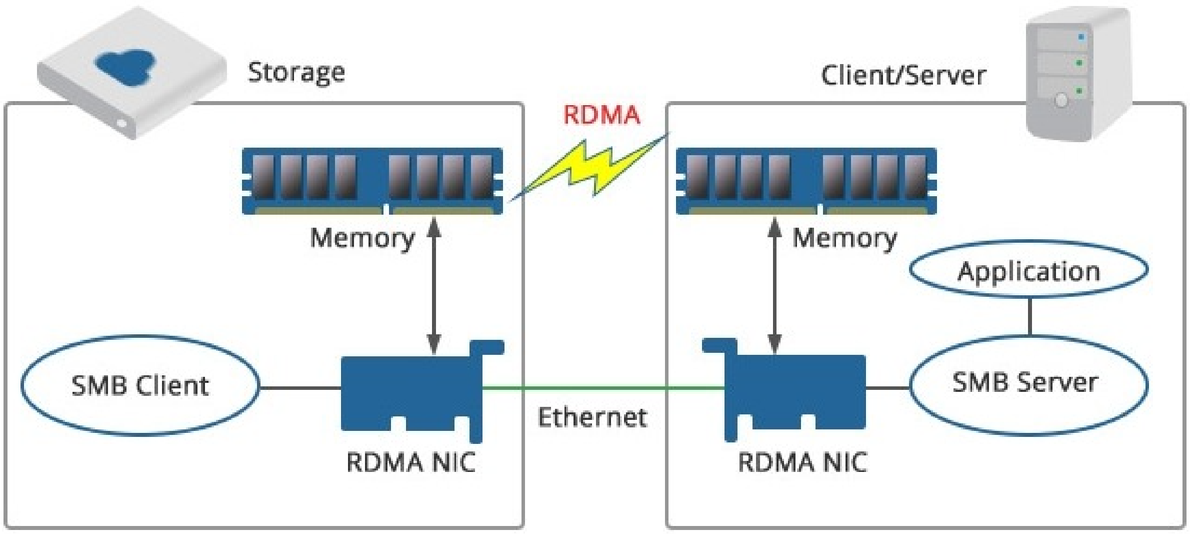


Figure 10-2 **RDMA Memory to Memory Transfer**

Table 10-1: CMAC Read Operation

CMAC Read Operation				
Offset	**Bit Index**	**Data Direction**	**Name**	**Description**
0x00	13:0	Out	rx_desc_ram_addr	Receive Descriptor RAM Addr
0x04	16:0	Out	rx_desc_len	Receive Descriptor Length
0x08	27:0	Out	rx_desc_tag	Receive Descriptor Tag
0x0C	31:0	Out	rx_desc_valid	Receive Descriptor Valid
0x10	16:0	Out	rx_desc_status_len_reg	Receive Descriptor Status Length Register
0x14	27:0	Out	rx_desc_status_tag_reg	Receive Descriptor Status Tag Register
0x18	31:0	Out	rx_desc_status_valid_reg	Receive Descriptor Status Valid Register
0x20	13:0	Out	tx_desc_ram_addr	Transmit Descriptor RAM Addr
0x24	16:0	Out	tx_desc_len	Transmit Descriptor Length
0x28	27:0	Out	tx_desc_tag	Transmit Descriptor Tag
0x2C	31:0	Out	tx_desc_valid	Transmit Descriptor Valid
0x30	27:0	Out	tx_desc_status_tag_reg	Transmit Descriptor Status Tag Register
0x34	31:0	Out	tx_desc_status_valid_reg	Transmit Descriptor Status Valid Register

Table 10-1: **CMAC Read Operation**

Table 10-2: CMAC Write Operation

CMAC Write Operation				
Offset	**Bit Index**	**Data Direction**	**Name**	**Description**
0x00	18:0	In	rx_desc_ram_addr	Receive Descriptor RAM Addr
0x04	15:0	In	rx_desc_len	Receive Descriptor Length
0x08	4:0	In	rx_desc_tag	Receive Descriptor Tag
0x0C	0:0	In	rx_desc_valid	Receive Descriptor Valid
0x10	15:0	In	rx_desc_status_len_reg	Receive Descriptor Status Length Register
0x14	4:0	In	rx_desc_status_tag_reg	Receive Descriptor Status Tag Register
0x18	0:0	In	rx_desc_status_valid_reg	Receive Descriptor Status Valid Register
0x20	18:0	In	tx_desc_ram_addr	Transmit Descriptor RAM Addr
0x24	15:0	In	tx_desc_len	Transmit Descriptor Length
0x28	4:0	In	tx_desc_tag	Transmit Descriptor Tag
0x2C	0:0	In	tx_desc_valid	Transmit Descriptor Valid
0x30	4:0	In	tx_desc_status_tag_reg	Transmit Descriptor Status Tag Register
0x34	0:0	In	tx_desc_status_valid_reg	Transmit Descriptor Status Valid Register

Table 10-2: **CMAC Write Operation**

Figure 10-3: CMAC Block Diagram

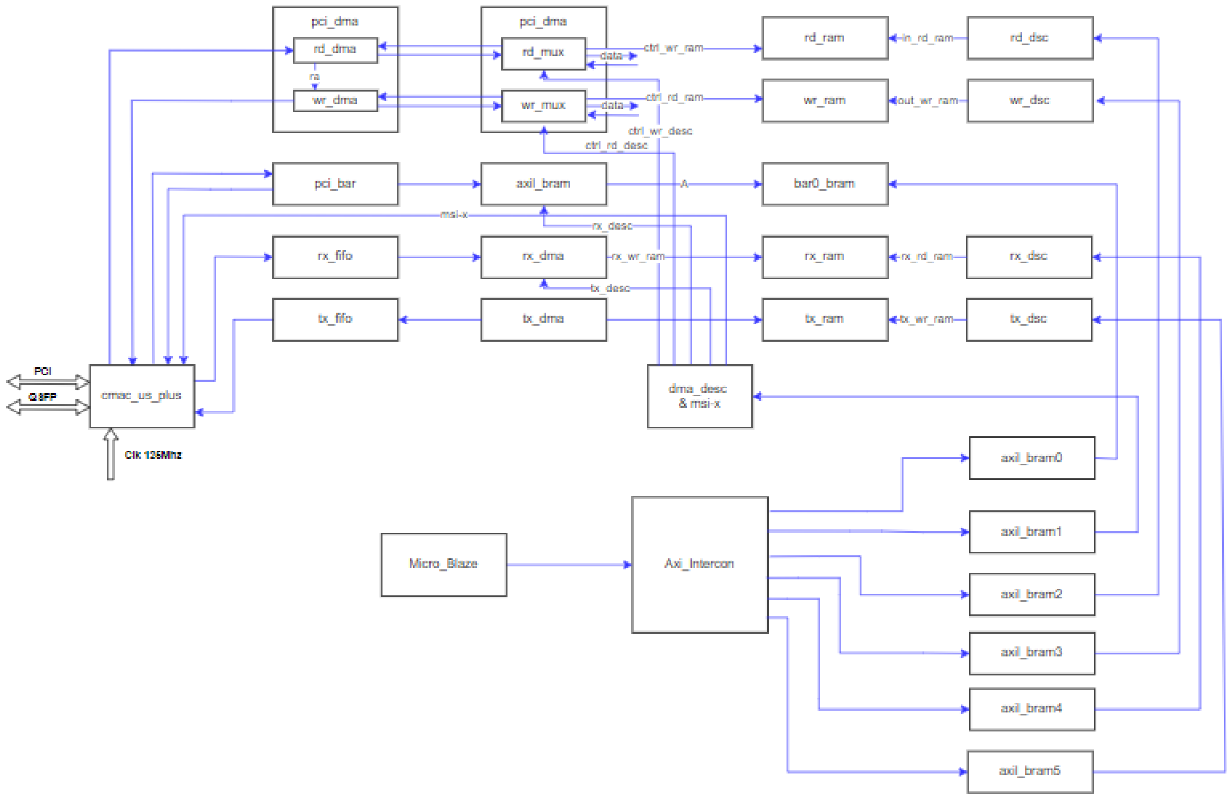


Figure 10-3: **CMAC Block Diagram**

Chapter 11. Ethernet

RSNIC IP has two communication ports, one is the PCI which is the Xilinx PCIE Controller/Phy and the other one is the Ethernet which is the cmac_us_plus Ethernet. In this chapter we will focus on Ethernet only and how it is being implemented in the *RSNIC IP.*

11.1 Ethernet Technology

Ethernet is a networking technology that includes the protocol, port, and cable to plug the server or desktop/laptop into a local area network (LAN). It provides a straightforward user interface that facilitates the connection of several devices, including switches, routers, and PCs. With a router and just a few Ethernet connections, it is possible to construct a local area network (LAN) that enables users to communicate between all connected devices. This is because laptops have Ethernet connectors, into which cables are inserted, and the other end is linked to routers.

Most Ethernet devices are compatible with Ethernet connections and devices that run at slower speeds. However, the connection speed will be determined by the weakest components.

11.2 Ethernet Protocol

The Ethernet protocol employs a star topology or linear bus, which is the basis for the IEEE 802.3 standard. In the OSI network structure, this protocol works both the physical layer and data link layer, the first two levels. Ethernet divides the data connection layer into two distinct layers: the logical link control tier and also the medium access control (MAC) tier.

11.3 Ethernet Data Connection

The data connection layer in a network system is primarily concerned with transmitting data packets from one node to the other. Ethernet employs an access mechanism known as CSMA/CD (Carrier Sense Multiple Access/Collision Detection) to enable each computer to listen to the connection before delivering data across the network.

Ethernet also transmits data using two components: packets and frames. The frame contains the sent data payload as well as the following:

- Both the MAC and physical addresses of the sender and recipient
- Error correction data for identifying transmission faults
- Information on Virtual LAN (VLAN) tagging, as well as the quality of service

Each frame is encapsulated in packets that comprise many bytes of data to set up the connection and identify the frame’s commencement point.

An Ethernet connection encompasses the following:

- **The Ethernet protocol**: It is a series of standards that governs how data is sent between Ethernet components as explained before
- **The Ethernet port**: Ethernet ports (commonly known as jacks or sockets) aer openings on computer network infrastructure into which one may plug in Ethernet cables. It supports cables with RJ-45 connectors. The Ethernet connector on the majority of computers serves to connect the equipment to a wired connection. The Ethernet port of a computer is linked to an Ethernet network adapter, also known as an Ethernet card, mounted on the motherboard. A router may contain numerous Ethernet ports to support various wired network devices.
- **Ethernet network adapter**: An Ethernet adapter is a chip or card that fits into a slot on the motherboard and allows a computer to connect to a local area network (LAN).
- **An Ethernet cable**: Ethernet cable, often known as a network cable, links your computer to a modem, router, or network switch. The Ethernet cable consists of the RJ-45 connection, the internal cabling, and a plastic jacket.

11.4 Ethernet Supported Speed

It is because our Ethernet uses UltraScale+ 100G Ethernet from Xilinx for CMAC or CAUI Media Access Controller implementation, our type of connection is 10 Gigabit Ethernet. The newest iteration of Ethernet, 10 Gigabit Ethernet, offers a data throughput of 10Gbit/s (10,1000 Mbit/s) via an optic or twisted pair connection. 10 GBASE-LX4, 10 GBASE-ER, or 10 GBASE-SR built on an optical fiber connection could reach up to 10,000 meters in distance (6.2 miles). The twisted pair option requires a cable of exceptional quality (Cat-6a or Cat-7). Ethernet 10 Gbit/s is mainly utilized for backbone networks in high-end operations that demand significant data speeds.

We can see in Figure 3.2 how the Ethernet is being connected. Figure 11.4 Below is the infiniband QSFP + Copper Cable 10g DAC Cisco Cable 1m.

Figure 11.4 QSFP + Copper Cable

11.5 *RSNIC IP Implementation*

The *RSNIC IP* Ethernet firmware implementation supported features are the following:

- Fetching of available WQE for ethernet requests.
- Parsing of Multi-Packet Send WQE (MPSW) offload. Figure 11.5 shows the WQE with MPSW format, The MPSW enables sending multiple fixed sized packets using single WQE. The MPSW contains a pointer to the packets. The firmware breaks into packets according to Max segment Size in WQE. When the WQE is completed, the firmware will generate a single CQE.
- Internet Protocol (IP Address)
  - IPv4
  - IPv6
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- Media Access Controller (MAC) Address

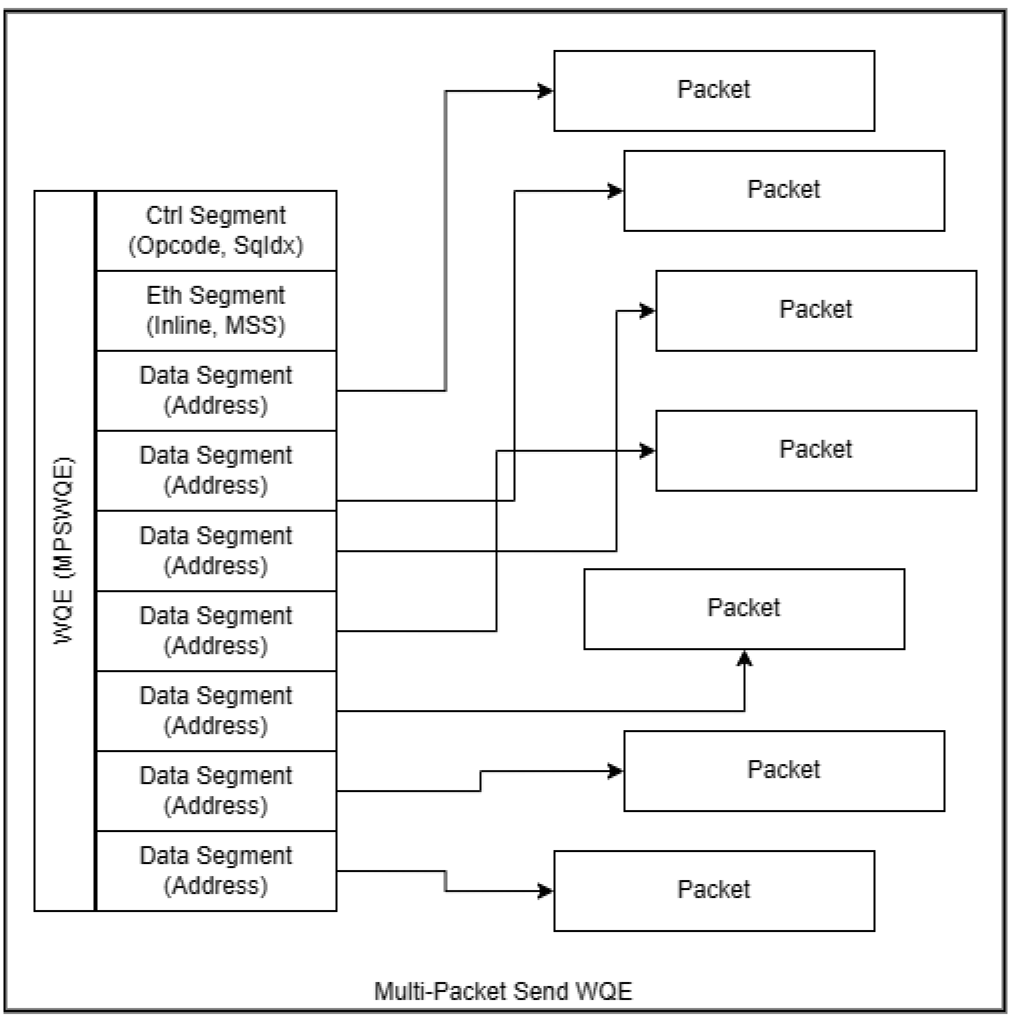


Figure 11.5 Multi-Packet Send WQE format.

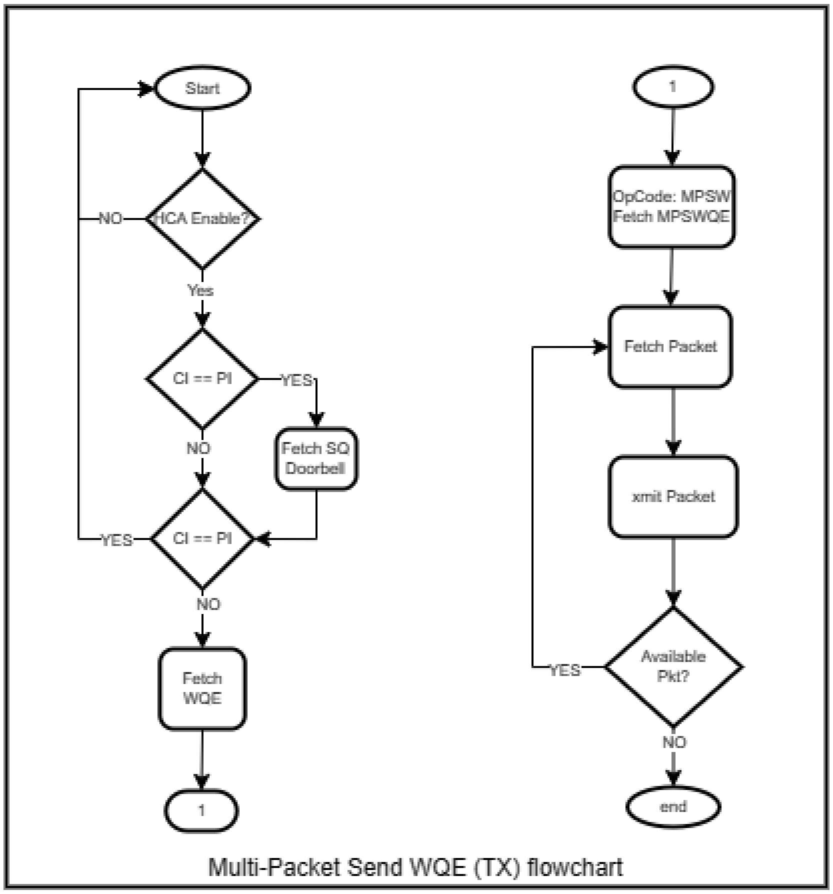


Figure 11.5-1 Multi-Packet Send WQE (TX) Flowchart.

The Figure 11.5-1 shows the process flow for transmitting the packet going out the NIC. The process will wait until the HCA is enabled, which is done during driver initialization. Once the HCA is enabled the firmware will check if there is an available work request by comparing the consumer index (CI) and producer index (PI). When the CI and PI are equal the firmware will fetch the Doorbell record for updated consumer index (CI) to make sure that there is no miss work request. And if there is an available work request the firmware will fetch WQE from the assigned Send Queue (SQ), then the firmware will process the operations code (opcode = MPSWQE). The firmware breaks the packet content of the MPSWQE, by getting the address where the packet was located then the firmware will initiate the fetching of the packet going in to NIC memory, then the firmware will issue the transmittal of packet going out of NIC via net-dma.

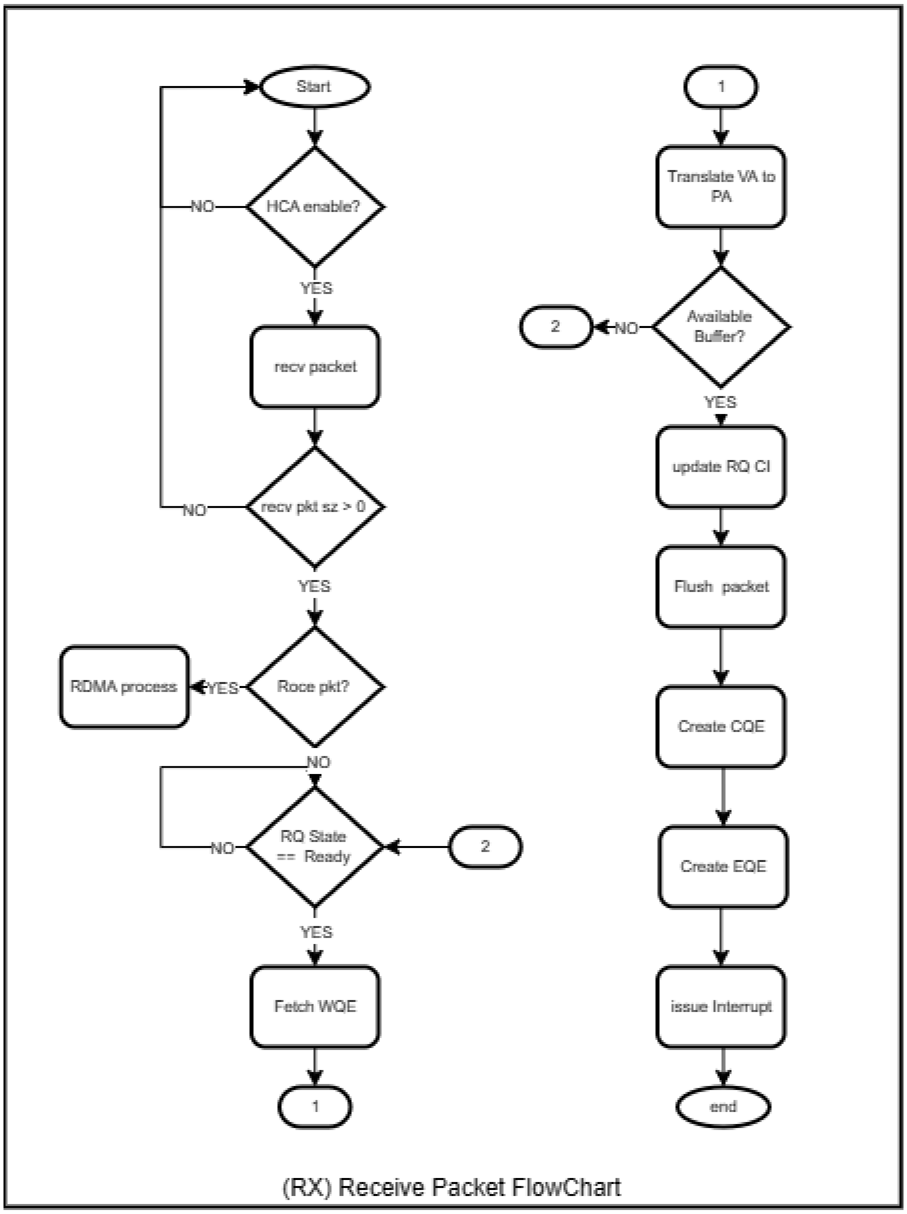


Figure 11.5-2 (RX) Receive PacketFlowchart for ethernet packet.

Figure 11.5-2 shows the process of receiving a packet from the network. The firmware will wait until the HCA is enabled. Once the HCA is enabled the firmware will initiate the rx net-dma to get the available packet. There is an available packet if the size returned from the DMA is not zero. The firmware will check if the packet is Roce or Ethernet. The firmware will start the receive routine for the Ethernet packet, First the firmware will check if the RQ is in ready state, then the firmware will Fetch the WQE (WQE Receive format). The WQE contains the Byte Count, Local key and virtual address, the firmware will translate the virtual address to PA address using the Memory translation table (MTT). Once the firmware gets the Physical Address PA the received packet will DMA'ed to the given Physical Address (PA). Then the firmware will generate the CQE and EQE before the message Signal Interrupt (MSI) is triggered.