Xilinx Embedded RDMA Enabled NIC v3.1

LogiCORE IP Product Guide

Vivado Design Suite

PG332 June 30, 2021

Table of Contents

IP Facts

Chapter 1: Overview

Navigating Content by Design Process 5
Core Overview 5
Feature Summary 7
Applications 10
Unsupported Features 10
Licensing and Ordering 11

Chapter 2: Product Specification

RDMA Enabled NIC 12
Work Requests/Work Queue Entries (WQEs) 13
Work Completions 14
RDMA Queues 14
ERNIC RX Path 16
ERNIC TX Path 21
Standards 24
Performance 24
Resource Utilization 24
Port Descriptions 25
Parameter Descriptions 29
Register Space 30

Chapter 3: Designing with the Core

General Design Guidelines 47
Interrupts 48
Clocking 49
Resets 49

Chapter 4: Design Flow Steps

Customizing and Generating the Core 50
Simulation 51
Synthesis and Implementation 51

Chapter 5: Example Design

Example Design Features. 53
Example Design Limitations 54
Simulating the Example Design. 54
Example Sequence 55

Chapter 6: ERNIC Software Flow

ERNIC Global Configuration. 57
QP1 Creation 58
Memory Registration. 58
RC QP Creation 59
WQE SQ Posting 60
Receiving Incoming RDMA SEND Messages 60
Processing WQE Completions 60
Enabling QP HW Offload 61
QP Deletion 61
QP Fatal Recovery 62

Appendix A: Requesting ERNIC Support Questionnaire

Appendix B: Debugging

Debugging the IP/System 66

Appendix C: Additional Resources and Legal Notices

Xilinx Resources 70
Documentation Navigator and Design Hubs 70
References 71
Revision History 71
Please Read: Important Legal Notices 73

Introduction

The Xilinx® ERNIC™ (Embedded RDMA enabled NIC) IP is an implementation of RDMA over Converged Ethernet (RoCE v2) enabled NIC functionality. This parameterizable soft IP core can work with a wide variety of Xilinx hard and soft MAC IP implementations providing a high throughput, low latency, and completely hardware offloaded reliable data transfer solution over standard Ethernet. The ERNIC IP allows simultaneous connections to multiple remote hosts running RoCE v2 traffic.

Features

- Support for RDMA functionality
  - RoCE v2
  - Packet retransmission on errors are handled in the hardware by the IP.
- 100 Gb/s line rate
- Support for Reliable Connection (RC) RDMA transport service types
- QP1 support for sending and receiving MAD packets
- Hardware handshake mode on user interface to support hardware RDMA applications in the user logic
- Supports incoming and outgoing RDMA SEND, RDMA READ, RDMA WRITE, RDMA SEND WITH IMM, RDMA WRITE WITH IMM, and RDMA SEND WITH INVALIDATE message types.
- Designed to scale up to 255 RDMA Queue pairs[3]
- Support for IPv4 and IPv6 packets
- Support for Explicit Congestion Notification (ECN)
- Supports Priority flow control with different priorities for RoCE and non-RoCE traffic.
- Supports memory registrations and protection domains

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family[1]	Versal™ ACAP Kintex UltraScale+™, Virtex® UltraScale™, Virtex UltraScale+, Zynq® UltraScale+
Supported User Interfaces	AXI4-Lite, AXI4, and AXI4-Stream
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Test Bench	Not Provided
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Not Provided
Supported S/W Driver[2]	Linux kernel drivers and user space libraries for RDMA
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite Vivado IP integrator
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	N/A
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of third-party tools, see the Xilinx Design Tools: Release Notes Guide.
3. For -1 speed grade devices design might have timing violations for more than 64 QP configuration.

Overview

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. This document covers the following design processes:

- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado timing, resource and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Port Descriptions
  - Register Space
  - Clocking
  - Resets
  - Customizing and Generating the Core
  - Example Design

Core Overview

This chapter provides an overview of the ERNIC IP core and details of the applications, licensing requirements, and standards conformance. ERNIC is a soft IP implementing RDMA over a Converged Ethernet (RoCE v2) protocol for embedded target or initiator devices. This implementation is based on the specifications described in *InfiniBand Architecture Specification* Volume 1, Annex A16 RoCE and Annex 17 RoCE V2 [Ref 1].

Figure 1-1 shows the ERNIC and its connections to other IPs in the subsystem.

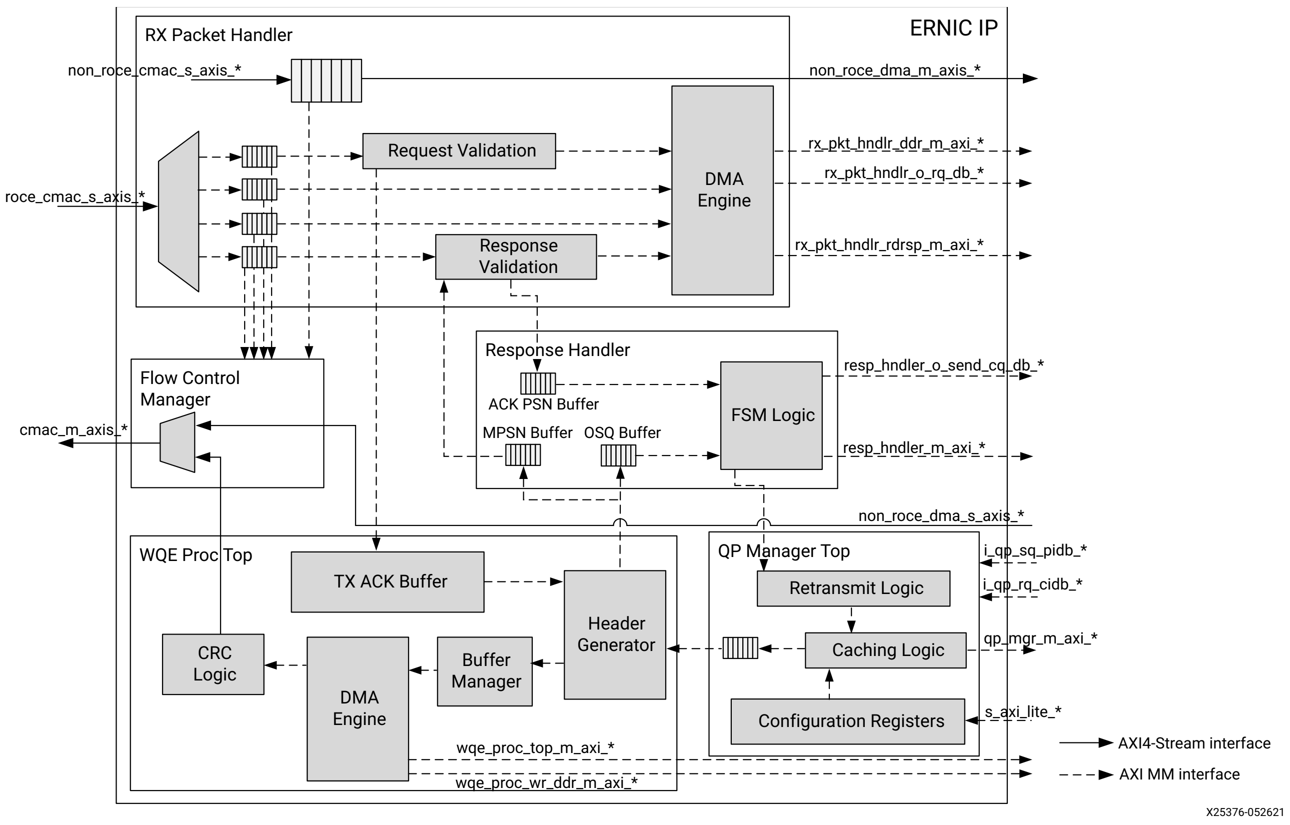

Figure 1-1: **ERNIC IP Block Diagram**

Note: The user logic or target IP that connects to ERNIC is referred to as application and the direction of the arrows is from master to slave.

Apart from the ERNIC IP, the ERNIC subsystem includes the Xilinx Ethernet IP, AXI DMA, and AXI Interconnect among other IPs. On the user application front, the ERNIC IP exposes side band interfaces to allow efficient doorbell exchanges without going through the interconnect.

Each queue is identified with a set of read and write pointers called the Producer Index (write pointer) and Consumer Index (read pointer). The register address locations for these pointers are termed as doorbells in this document. A doorbell exchange or doorbell ringing indicates that the corresponding register location is updated.

Feature Summary

The ERNIC IP interfaces with any Ethernet MAC IP using an AXI4-Stream interface. Access to DDR or any other memory region is necessary for reading and writing various data structures for RDMA packet processing. This connection is achieved using multiple AXI4 interfaces. 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.

The ERNIC IP implements embedded RNIC functionality. As a result, only the following subset of RoCE v2 functionality is implemented compared to a general purpose RNIC:

- Support for RDMA SEND, RDMA READ, RDMA WRITE, RDMA SEND INVALIDATE, RDMA SEND IMMEDIATE, and RDMA WRITE IMMEDIATE for incoming and outgoing packets. Atomic operations are not supported.
- Support for up to 254 connections.
- Scalable design of up to 255 RDMA Queue pairs.

  Note: Default Vivado strategies allow for the timing to pass up to 127 queue pairs. To match the timing for 255 queue pairs, use the Vivado strategy — Performance_refinePlacement.
- Supports dynamic memory registration.
- Hardware handshake mechanism for efficient doorbell exchange with the user application logic.

  Note: When switching in the handshake mode, the upper layer should not have any traffic on that particular QP.

ERNIC Modules

The ERNIC IP consists of the following main modules that are explained in this section.

- QP Manager
- WQE Processor Engine
- RX PKT Handler
- Response Handler
- Flow Control Manager

QP Manager

The QP Manager module houses the configurations for all the QPs and provides an AXI4-Lite interface to the processor. It also arbitrates across various SEND Queues and caches the SEND Work Queue Entries (WQEs). These WQEs are then provided to the WQE processor module for further processing. This module also handles the QP pointer updates in the event of retransmission.

WQE Processor Engine

The WQE Processor Engine reads the cached WQEs from the QP Manager module and handles the following tasks:

- Validates the incoming WQE for any invalid opcode.
- Creates the header for the RDMA packets based on the Payload Max Transfer Unit (PMTU) and programs internal DMA engine.
- It also triggers the DMA to start outgoing packet transfers.

The WQE Processor Engine is also responsible for sending outgoing acknowledgment packets for the incoming RDMA SEND/WRITE requests and read responses for incoming RDMA READ requests.

RX PKT Handler

The RX PKT Handler module receives the incoming RDMA packets. Non-RDMA packets should be filtered out before receiving the RX PKT Handler (**`roce_cmac_s_axis`**) interface. The ERNIC IP handles the following types of incoming RoCE v2 packets:

- RDMA SEND, RDMA WRITE, RDMA READ and response packets for RDMA READ (request sent from ERNIC)
- RDMA SEND with Invalidate, RDMA SEND with immediate, and RDMA WRITE with immediate packets
- Acknowledgment packets for RDMA WRITE/RDMA SEND (request sent from ERNIC)
- Communication management (Management Datagram) packets to QP1

The RX PKT Handler module is responsible for validating the incoming packets. It also triggers outgoing acknowledgment packets for incoming RDMA SEND and RDMA WRITE requests and pushes the packets that pass the validation to the corresponding memory location. The RDMA READ responses are channeled to the target application directly. The module handles the incoming RDMA READ requests and forwards the request to the TX path.

The RX PKT Handler module also decodes the RDMA SEND invalidate/Immediate and RDMA WRITE Immediate packets. The 32-bit data present in either IETH or IMMDT headers is provided on a separate AXI4-Stream interface. Sixty-four bits of data is provided on this streaming interface for every entry. Below table shows encoding of these 64 bits.

Table 1-1: **Invalidate/Immediate Data Entry Structure**

Bits	Field	Description
31:0	ImmDt/R_KEY	Contents of this field is defined by the Type field. This field contains the immediate data or R_KEY in case of send with invalidate.
39:32	Type	This field indicates the type of the incoming packet that triggered this completion. 8'h01 = SEND INVALIDATE 8'h02 = SEND IMMEDIATE 8'h03 = WRITE_IMMEDIATE 8'h04 to 8'hFF = Reserved
47:40	QPID	QP identifier which triggered this entry
55:48	RQ PI pointer	This field is valid only when Type field indicates SEND with immediate or SEND with invalidate commands. This field indicates the receive queue pointer where corresponding SEND command is located.
63:56	Reserved	Reserved

An external hardware logic needs to be implemented to handle this invalidate/immediate data provided on this interface. For example AXI DMA IP to convert streaming interface to AXI Memory mapped interface and write the data to specific location and notify to software or hardware application.

Outgoing Pause requests for RDMA traffic are handled inside this module. When remaining buffer locations reaches to XON condition then it triggers for Pause ON and deasserts when buffer pointer reaches to XOFF condition.

Response Handler

The Response Handler module manages the outstanding queues. These queues hold the information about all packets sent to the remote host but have not yet been acknowledged or responded to. In addition, this module triggers a re-transmission if the remote host sends a Negative Acknowledgment (NAK). If this module does not receive a response from the remote host within a specified time (timeout value), it triggers a timeout related retransmission.

Retransmission is triggered either due to an incoming NACK or an internal timeout. When a retransmission is triggered, ERNIC IP takes care of the following:

1. Identifies the PSN and MSN values to be retransmitted.
2. Rolls back the PSN and MSN values for that queue pair.
3. Rolls back the current SQ pointer to the WQE that needs to be retransmitted.
4. ERNIC retransmits the complete WQE in case of a retry. Partial WQE retransmission is not supported.

Flow Control Manager

This module maintains separate buffers in RX path and generates PFC control signals as per the programmable full conditions of those buffers. Similarly in TX path, it also has two separate buffers for RoCE and non-RoCE. On the assertion of any Pause signals, the module will stop reading from the buffer.

Applications

The ERNIC IP can be used in range of applications that require reliable transfer of packets across the network fabric. A few such applications are listed here:

- Sensor Data Acquisition and transfer over RoCE V2
- Video/Image capture and transfer over RoCE V2
- Remote storage nodes over RoCE V2

Unsupported Features

The ERNIC IP does not support:

- Incoming ATOMIC operations
- Outgoing ATOMIC operations
- Incoming/outgoing RDMA SEND packets of 0 length
- Maximum DMA length for RDMA payload more than 8 MB
- Incoming RDMA READ Requests with DMA Length equal to or less than 4 bytes
- Incoming RDMA READ/WRITE Requests with virtual address not on 64-byte boundary
- Incoming READ Response and WRITE Request Length non-multiple of 64-byte

Licensing and Ordering

This Xilinx LogiCORE™ IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado® Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. To generate a full license, visit the product licensing web page. Evaluation licenses and hardware timeout licenses might be available for this core or subsystem. Contact your local Xilinx sales representative for information about pricing and availability.

License Checkers

If the IP requires a license key, the key must be verified. The Vivado design tools have several license checkpoints for gating licensed IP through the flow. If the license check succeeds, the IP can continue generation. Otherwise, generation halts with an error. License checkpoints are enforced by the following tools:

- Vivado synthesis
- Vivado implementation
- write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Product Specification

The ERNIC IP provides an embedded implementation of a RoCE v2 enabled NIC. The RDMA technology allows for faster movement of data over standard Ethernet while completely offloading CPU bandwidth. The ERNIC IP core comes with SW drivers that can be ported to any Zynq® MPSoC or FPGA devices. This allows the ERNIC IP to function independent of any external processor.

RDMA Enabled NIC

The Xilinx® Embedded RDMA enabled NIC (ERNIC) controller can interface with any user application using one of the following methods:

- Closely integrated HW handshake mechanism which provides a low latency, light weight interface to the target system
- OFED-complaint RDMA hardware drivers and user space libraries providing standard SW API to post work requests to ERNIC

Figure 2-1 shows sample end-to-end system with multiple host CPUs and multiple native NVMe devices talking over a network fabric through the Xilinx ERNIC + NVMEOFABRIC IP subsystem.

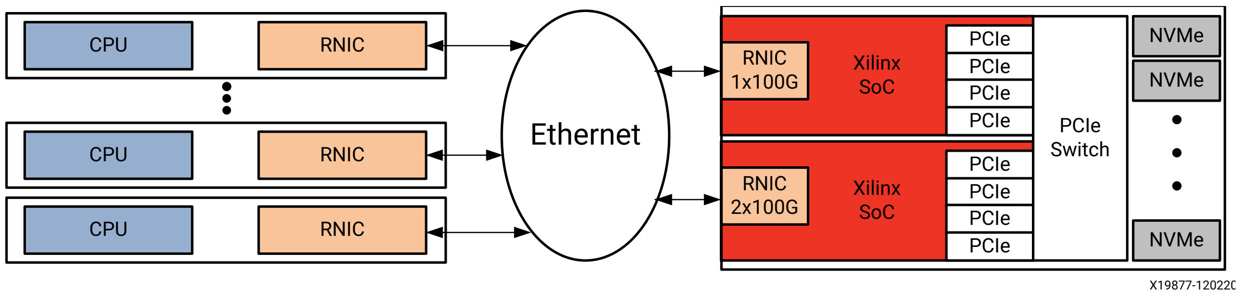

Figure 2-1: **Xilinx ERNIC + NVMe over Interconnect**

The host side RDMA enabled NICs (RNIC) shown in Figure 2-1 need to support RoCE v2 technology. The following sections describe the key data structures used in the ERNIC implementation.

Work Requests/Work Queue Entries (WQEs)

Work Requests are used to submit units of work to the ERNIC IP. The operations which can be posted to the Send Work Queues by the application are:

- RDMA Write
- RDMA WRITE Immediate
- RDMA SEND
- RDMA SEND Immediate
- RDMA Read
- RDMA SEND Invalidate

The following work requests are not allowed:

- ATOMIC
- Bind Memory Window
- Local Invalidate
- Fast Register Physical MR

The Receive Queue work requests need not be posted by application as the ERNIC Hardware automatically re-posts consumed receive buffers as per the configured receive queue depth.

Table 2-1 shows the structure of Send work requests. Each Work Queue Entry (WQE) is 64 bytes in size.

Table 2-1: **WQE Structure**

Bitwidth	Content	Size (B)	Comment
[15:0]	WRID	2	Work Request ID. Unique identifier for every WQE
[31:16]	Reserved	2	
[95:32]	LADDR	8	Local address. Data Buffer address for operations
[127:96]	LENGTH	4	Length of the transfer
[135:128]	OPCODE	1	8’h00 -- RDMA WRITE 8’h01 -- RDMA_WRITE WITH IMMDT 8’h02 -- RDMA SEND 8’h03 -- RDMA SEND WITH IMMDT 8’h04 -- RDMA READ 8’h0C -- RDMA SEND WITH INVALIDATE All other values are reserved.
[159:136]	Reserved	3	3
[223:160]	ROFFSET	8	Remote offset address

Table 2-1: **WQE Structure *(Cont'd)***

Bitwidth	Content	Size (B)	Comment
[255:224]	RTAG	4	Remote TAG
[383:256]	SDATA	16	RDMA SEND Data. If the data to be sent is less than or equal to 16B, this field is used to represent the data.
[511:384]	Reserved	16	-
[415:384]	IMMDT DATA	4	Immediate data to be send in ImmDt header
[511:416]	Reserved	12	Reserved

Work Completions

Work completions are posted for every WQE posted on the Send Queue. Completions are not posted for Receive Queue (RQ) entries, instead doorbells are rung per Queue Pair (QP) at the address pointed to by RQWPTRDBADDi when a new receive buffer is consumed by an incoming RDMA SEND request. The structure of a Completion Queue Entry (CQE) is given in Table 2-2. Each CQE is 4 bytes in size.

Table 2-2: **Completion Queue Entry Structure**

Bitwidth	Content	Size (B)	Comment
[15:0]	WRID	2	Work Request ID. Unique identifier for every WQE
[23:16]	OPCODE	1	• 8'd0 = RDMA WRITE • 8'd1 = RDMA WRITE Immediate • 8'd2 = RDMA SEND • 8'd3 = RDMA SEND Immediate • 8'd4 = RDMA READ • 8'd5 - 8'd11 = RESERVED • 8'd12 = RDMA SEND Invalidate • 8'd13 - 8'd255 = RESERVED
[31:24]	ERRFLAG	1	Error flag. Set to 8'b1 in case QP enters a fatal state

RDMA Queues

The ERNIC IP implements RDMA queues like Receive Queue (RQ), Send Queue (SQ), and Completion Queue (CQ). These queues are referred to as Queue Pairs or QPs. SQ houses the send WQEs posted by the user application.

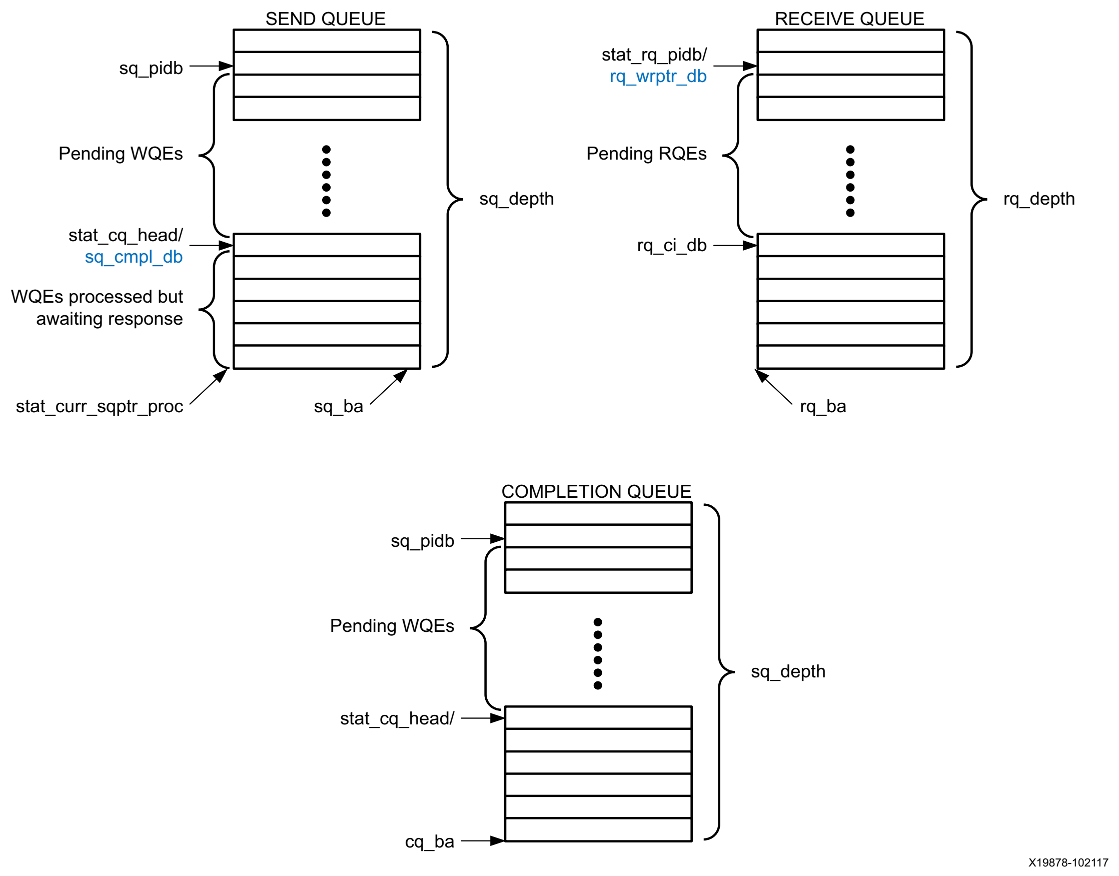

Figure 2-2: **RDMA Queues**

The RQ houses the incoming RDMA SEND packets. The completion queue informs the user application about the completed SEND WQEs. Each of these queues are implemented as circular buffers. Various doorbell and write pointer registers define the current state of these queues. Figure 2-2 shows the different queues and the variables/registers that define the state of the queues. The highlighted variables are indirectly accessed by the ERNIC IP. The register CQDBADDi points to the address for **`sq_cmpl_db`** and the RQWPTRDBADDi register points to the address for the **`rq_wrptr_db`**.

These queues are implemented in memory regions outside the ERNIC IP. The ERNIC accesses the IP through the various AXI master interfaces. See Table 3-2 for details of ERNIC memory requirements.

The next few sections provide a brief overview of the incoming (RX) and outgoing (TX) data flow of the ERNIC IP.

ERNIC RX Path

The ERNIC RX Path gets the packet data from the MAC through the AXI4-Stream interface. All incoming packets are validated and all packet headers that fail packet validation are sent to the error buffer (base address specified by ERRBUFBA[31:0]) if the value of XRNICCONF[5] is set to 1. The header is prefixed with an error syndrome by the ERNIC RX packet handler module as per Table 2-3. These buffers provide useful debug information for incoming packet errors.

Table 2-3: **Packet Validation Error Syndrome**

Error Syndrome Bit Index	Error related to	Precedence	Description and impact	Impact
0	MAC	4	MAC destination address received in the Ethernet header does not match the ERNIC MAC address configured	Packet dropped
1	Reserved			
2	IPv4/IPv6	5	IP Version not as per ERNIC configuration	Packet dropped
3	IPv4	7	IPv4 header length not equal to 20 bytes	Packet dropped
4	Reserved			
5	IPv4	8	Flag bits in IPv4 are not 3'b010	Packet dropped
6	IPv4	8	Fragment Offset in IPv4 are not 0	Packet dropped
7	Reserved			
8	IPv4/IPv6	8	IPv4 destination address in IPv4 header is not matching with ERNIC IPv4 address configured	Packet dropped
9	IPv4	6	IPv4 header checksum error.	Packet dropped
10	IPv4/IPv6	8	IPv4 Total Length field value is not with in range. IPv6 Payload Length field value is not with in range	Packet dropped
11	Reserved			
12	UDP	9	UDP Length field is not consistent with IPv4 Total Length or IPv6 Payload Length	Packet dropped
13	BTH	10	BTH Version is not 4'b0000	Packet dropped
14	BTH	10	QP Destination address in BTH is not supported	Packet dropped
15	BTH	11	QP specified in the destination address of BTH is either not configured or not enabled	Packet dropped
16	BTH	14	Opcode sequence in not correct	NAK-Invalid sent and QP moved to FATAL state

Table 2-3: **Packet Validation Error Syndrome *(Cont'd)***

Error Syndrome Bit Index	Error related to	Precedence	Description and impact	Impact
17	BTH	10	Unsupported or reserved opcode request is received from remote QP	NAk-Invalid sent and QP moved to FATAL state
18	BTH	14	For FIRST and MIDDLE request/response received BTH Pad bits are not 2'b00	Packet dropped
19	Transport layer	14	Transport Layer Payload is inconsistent with PMTU configured for that QP	Packet dropped
20	Internal error	13	Request Queue is full and the packet is dropped	RNR-NAK sent
21	Transport layer	12	Incoming request PSN sequence is not correct	NAK-Sequence error
22	Transport layer	14	AETH Syndrome is malformed.	Packet dropped
23	Transport layer	14	Bad response or NAK-Invalid response received	QP moved to FATAL state
24-26	Reserved			
27	IPv4/IPv6	5	IP4(6) source address in IP4(6) header is not matching with QP configured remote node source address	Packet dropped
28	MAC	4	MAC source address in the Ethernet header does not match the QP configured remote node MAC address	Packet dropped
29	Reserved			
30	Link layer	2	ICRC Error	Packet dropped
31	Link layer	1	FCS Error	Packet dropped

Most of the packet validation errors are handled entirely by hardware and no software intervention is required. However, if an incoming packet causes the QP to enter into a FATAL state, software intervention is required to process the error and to initiate a disconnection. Such errors are available for the SW in the incoming packet error status buffers defined by IPKTERRQBA, IPKTERRQSZ, and IPKTERRQWPTR registers. Each error status buffer entry is 64-bit wide. The format for the error status is as shown in Figure 2-3.

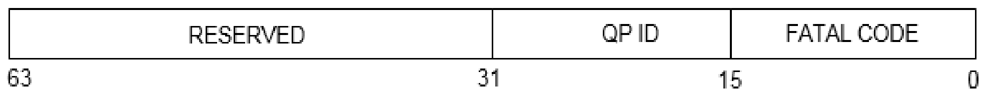

Figure 2-3: **Error Status Format**

Fatal table decoding is shown in Table 2-4.

Table 2-4: **Decoding for FATAL Codes**

FATAL Code	Description	Local/Remote Error
5'b00001	Opcode seq check fail	Locally detected error
5'b00010	Request packet length is not as per PMTU configured OR Pad count check failed	Locally detected error
5'b00011	Both opcode sequence error and packet length OR Pad count error occurred simultaneously	Locally detected error
5'b00100	Unsupported request opcode received but with correct PSN	Locally detected error
5'b00101	QP went into fatal due to WQE Processor	Locally detected error
5'b00110	QP went into fatal due to response handler	Locally detected error
5'b10010	Write packet length error or pad count check failed	Locally detected error
5'b10001	Write opcode sequence check failed	Locally detected error
5'b10011	Both the above errors occurred simultaneously	Locally detected error
5'b10101	R-key check failed or access permission check failed	Locally detected error
5'110110	RETH DMA length check failed	Locally detected error
5'b10111	Both the above errors occurred simultaneously	Locally detected error
5'b10000	Read request resources full	Locally detected error
5'b01100	ACK response opcode is not correct	Locally detected error
5'b01101	RNR-NAK down counter expired	Locally detected error
5'b01010	NAK Invalid/RAE/ROE response received	Remote Error
5'b11111	Read response last packet error	Locally detected error
5'b11001	Bad AETH syndrome	Locally detected error
5'b11110	Read response middle length check failed	Locally detected error
5'b11100	Read response last instead of read resp only	Locally detected error
5'b11011	Bad response error	Locally detected error
5'b01001	RNR counter expired	Locally detected error
5'b11010	Read Response only length check failed	Locally detected error

Incoming RDMA SEND/WRITE/READ requests are expected on the RX side. All other types of packets are response packets for the outgoing requests. The data flow for incoming RDMA SEND requests is shown in Figure 2-4. The direction of arrows show the flow of data. On receiving a valid RDMA SEND incoming packet on a connected QP, the packet content (without the headers) is pushed into the RX Buffer for the relevant QP. The ERNIC rings the RQ Producer Index Doorbell (RQPI DB), to indicate that a new packet is available, either using the side band interface or through the AXI interface. This depends on the configuration of QPCONFi[4]. An acknowledgment is also posted to the remote host at this point. The user application may inform the ERNIC of having consumed the new packet by ringing the RQ consumer Index Doorbell (RQCI DB). On receiving this doorbell, the corresponding RX buffer is made available to be used for new incoming packets.

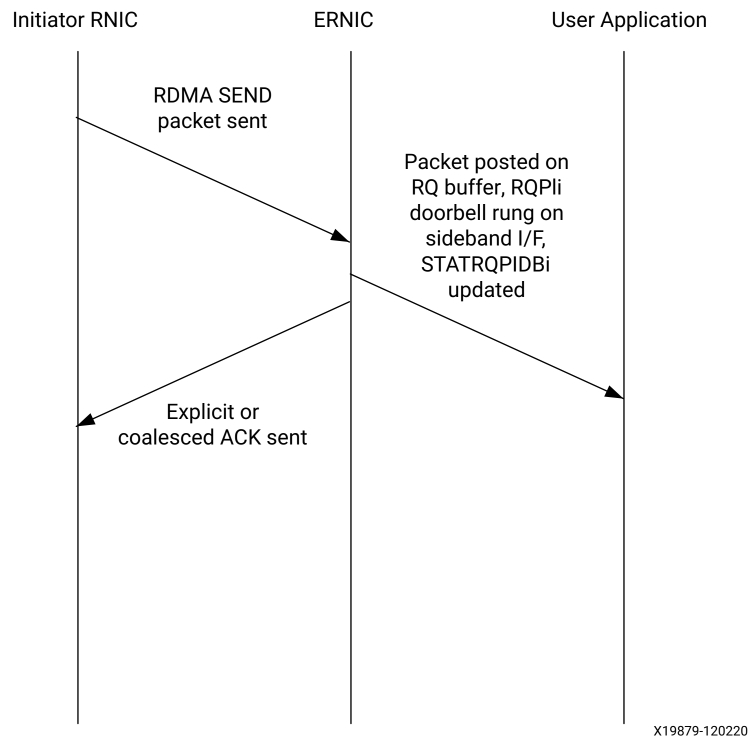
Initiator RNIC
ERNIC
User Application
RDMA SEND packet sent
Packet posted on RQ buffer, RQPIi doorbell rung on sideband I/F, STATRQPIDBi updated
Explicit or coalesced ACK sent
X19879-120220

Figure 2-4: **RDMA SEND RX Data Flow**

For incoming RDMA READ/WRITE request support on any RDMA device and to send a read/write request to other RDMA device in the network, the device should know the address, length, and RKEY of the destination RDMA client. For this purpose, the application layer should implement its own protocol to exchange the address and RKEY values before any data transfers. The examples in Figure 2-5 and Figure 2-6 shows an additional RDMA_SEND operation to exchange the required details for the RDMA READ/WRITE transaction. When the details are exchanged, RNIC will send a RDMA READ/WRITE packet to ERNIC and ERNIC will respond with ACK or RDMA read response. Figure 2-5 shows the flow of data for RDMA READ transaction. Figure 2-6 shows the flow of data for RDMA WRITE transaction.

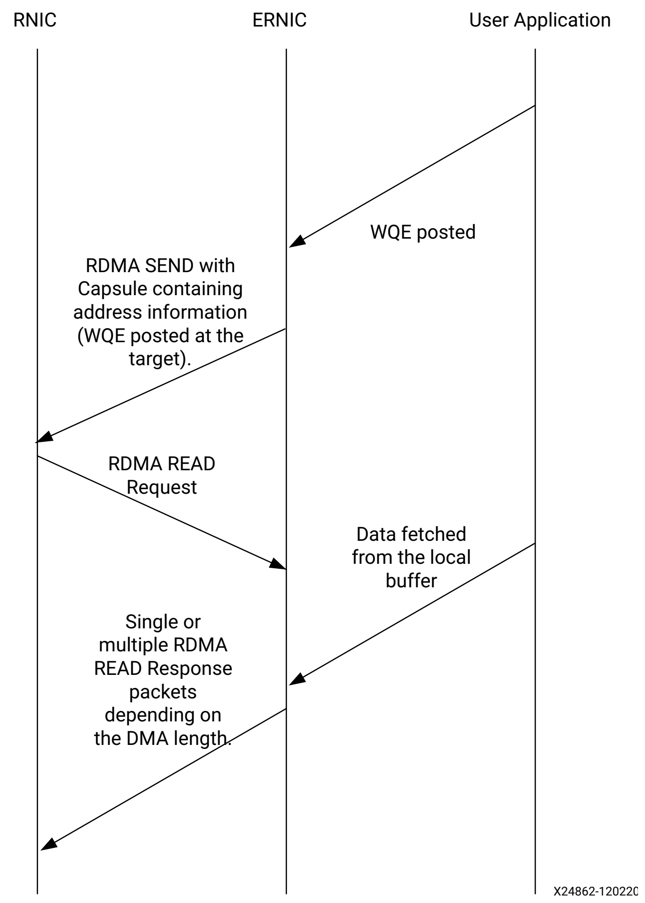
RNIC
ERNIC
User Application
WQE posted
RDMA SEND with Capsule containing address information (WQE posted at the target).
RDMA READ Request
Data fetched from the local buffer
Single or multiple RDMA READ Response packets depending on the DMA length.
X24862-120220

Figure 2-5: **RDMA READ Request RX Data Flow**

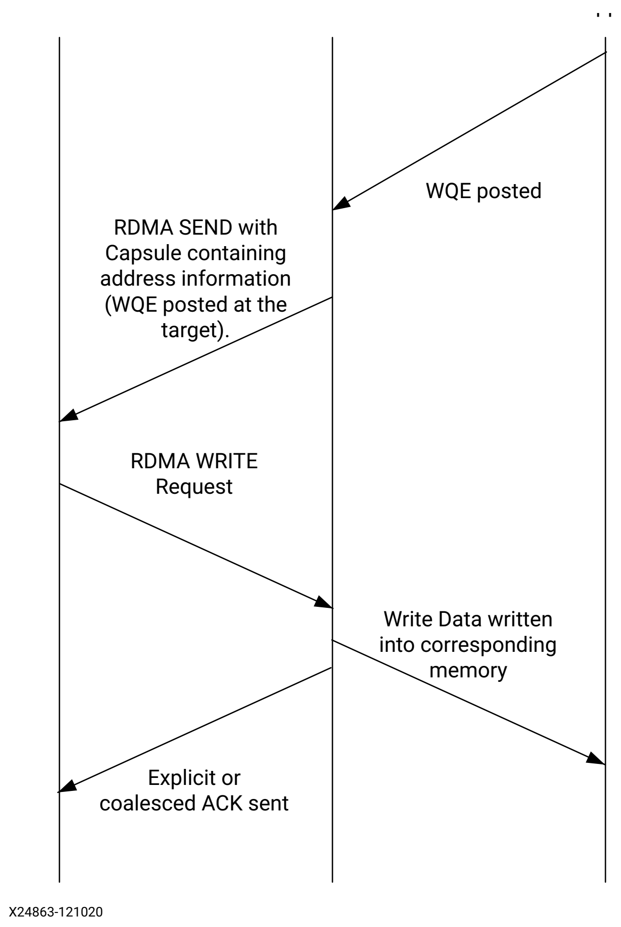
WQE posted
RDMA SEND with Capsule containing address information (WQE posted at the target).
RDMA WRITE Request
Write Data written into corresponding memory
Explicit or coalesced ACK sent
X24863-121020

Figure 2-6: **RDMA WRITE Request RX Data Flow**

ERNIC TX Path

The TX Path consists of outgoing RDMA READ, RDMA WRITE transactions, and ACK packets for incoming RDMA SEND/WRITE requests and responses for incoming RDMA READ requests. Based on the SQPIi doorbell, the Send Work Queue requests are processed. The DMA module is configured for data transfers for all outgoing transactions. The TX data flow for RDMA WRITE/SEND is shown in Figure 2-7.

The user application requests the ERNIC IP to transmit an RDMA WRITE/SEND/READ packet by posting a WQE on the SEND Queue for a particular QP and ringing the corresponding SQ Producer Index Doorbell (SQPI DB). The ERNIC processes the WQE and pulls data for RDMA WRITE/SEND commands based on the information provided in the WQE. This data along with relevant headers is pushed out on the link. Once an acknowledgment is received from the remote host, the ERNIC informs the user application of the successful completion of the WQE by posting a CQE, based on the configuration of QPCONFi[5], and by posting the completion count through the side band interface. The CQHEAD for the corresponding QP is also updated. For RDMA READ requests, the RX packet handler intimates the TX path. For this communication, an outstanding read request queue is available for each QP. The depth of the queue is determined by a parameter for incoming request resources. When the outstanding read request queue is full, it is indicated to the RX path. Any further requests to the QP will result in an NAK-Invalid and the QP is moved to fatal state.

Note: The direction of arrows shows the flow of data.

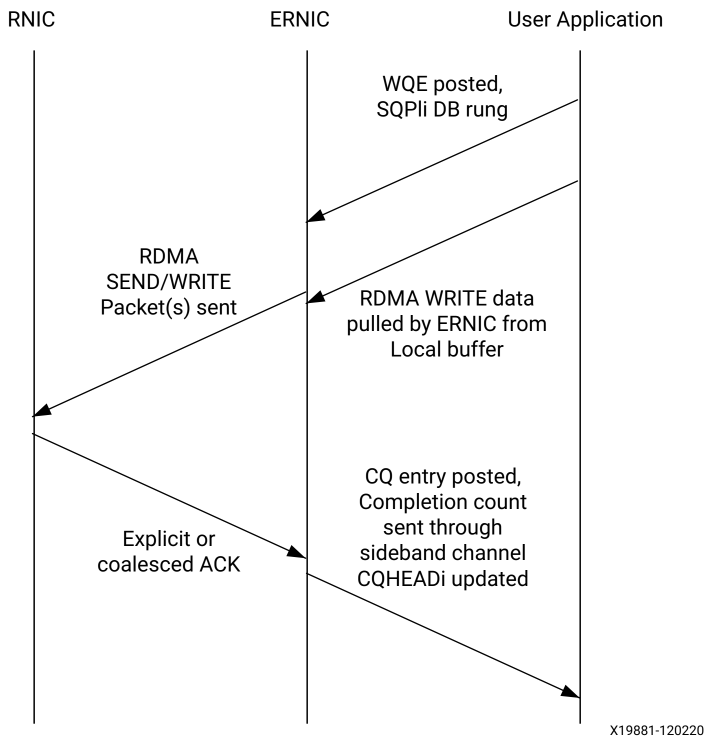
RNIC
ERNIC
User Application
WQE posted, SQPli DB rung
RDMA SEND/WRITE Packet(s) sent
RDMA WRITE data pulled by ERNIC from Local buffer
CQ entry posted, Completion count sent through sideband channel CQHEADi updated
Explicit or coalesced ACK
X19881-120220

Figure 2-7: **RDMA SEND/WRITE Data Flow**

Figure 2-8 shows the flow of RDMA READ requests to the remote host from the ERNIC.

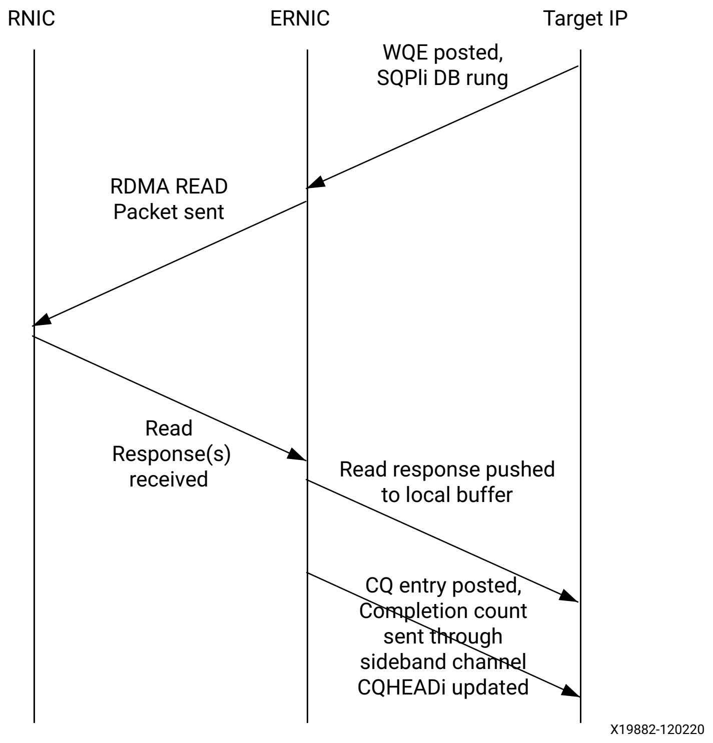
RNIC
ERNIC
Target IP
WQE posted, SQPIi DB rung
RDMA READ Packet sent
Read Response(s) received
Read response pushed to local buffer
CQ entry posted, Completion count sent through sideband channel CQHEADi updated
X19882-120220

Figure 2-8: **RDMA READ Data Flow**

Figure 2-8 shows the flow for RDMA READ requests to the remote host from the ERNIC. The user application requests the ERNIC IP to transmit an RDMA READ request to the remote host by posting a WQE on the SEND Queue for a particular QP and ringing the corresponding SQ Producer Index Doorbell (SQPI DB). ERNIC processes the WQE and creates the request packet with relevant headers and pushes it out on the link. Once the response data packets are received from the remote host, ERNIC writes the data (after removing the headers) to the local buffer address provided in the WQE. It then informs the user application of the successful completion of the WQE by posting a CQE, based on the configuration of QPCONFi[5], and by posting the completion count through the side band interface. The CQHEAD for the corresponding QP is also updated.

Standards

This implementation is based on the standard and specifications described in *InfiniBand Architecture Specification* Volume 1, Annex A16 RoCE and Annex 17 RoCE V2 [Ref 1].

Performance

The ERNIC IP is designed with an internal data path throughput of up to 100 Gb/s at a frequency of 200 MHz.

For more details about resource utilization, see the Performance and Resource Utilization.

Resource Utilization

This section summarizes the estimated maximum performance for various modules within the ERNIC IP. The data is separated into a table per device family. Each row describes a test case. The columns are divided into test parameters and results. The test parameters include the part information and the core-specific configuration parameters. Any configuration parameters that are not listed have their default values; any parameters with a blank value are disabled or set automatically by the IP core. Consult the product guide for this IP core for a list of GUI parameter and user parameter mappings.

- Resource use numbers are taken from the utilization report issued at the end of an implementation using the Out-of-Context flow in Vivado Design Suite.
- The Out-of-Context IP constraints include HD.CLK_SRC properties as required to ensure correct hold timing closure. These properties are enabled using the following Tcl command: **`set_param ips.includeClockLocationConstraints true`**
- The frequencies used for clock inputs are stated for each test case.
- LUT numbers do not include LUTs used as pack-thrus, but include LUTs used as memory.
- Default Vivado Design Suite 2018.1 settings are used. You can improve on these numbers using different settings. However, because the surrounding circuitry will affect placement and timing, these numbers might not repeat in a larger design.

For more details about resource utilization, see the Performance and Resource Utilization.

Port Descriptions

Table 2-5 table describes the ports and their interface definitions

Table 2-5: **ERNIC IP Ports**

Name	I/O	Width	Clock Domain	Description
Clocking and Reset				
m_axi_aclk	I	1	AXI4	AXI4 clock
m_axi_aresetn	I	1		AXI4 reset (active-Low)
s_axi_lite_aclk	I	1	AXI4	AXI4-Lite clock
s_axi_lite_aresetn	I	1		AXI4-Lite reset (active-Low)
system_resetn	O	1		System reset
cmac_rx_clk	I	1		RX user clock output from CMAC
cmac_rx_rst	I	1		RX reset for user logic from CMAC
cmac_tx_clk	I	1		TX user clock output from CMAC
cmac_tx_rst	I	1		TX reset for user logic from CMAC
Response Handler AXI4 Master Interface				
resp_hndler_m_axi_*	I/O		AXI4	This interface is used by the response handler to write completions in the completion queue present in the DDR. See Appendix A of the Vivado Design Suite: AXI Reference Guide (UG1037) [Ref 8] for more information on the AXI4 signal.
RX Packet Handler AXI4 Master Interface to DDR				
rx_pkt_hndler_ddr_m_axi*	I/O		AXI4	This interface is used by the RX packet handler module to push the data from incoming MAD, SEND, and RDMA WRITE packets to RQ buffer in the DDR. Supports only 64B aligned transaction. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.
RX Packet Handler AXI4 Master Read Response Interface				
rx_pkt_hndler_rdrsp_m_axi*	I/O		AXI4	This interface is used by the RX packet handler to write the data received in read responses to the DDR. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.

Table 2-5: **ERNIC IP Ports *(Cont'd)***

Name	I/O	Width	Clock Domain	Description
AXI4-Stream Slave Interface for Incoming RoCE Traffic				
roce_cmac_s_ axis_*	I/O		AXI4-Stream	This interface provides RoCE packets from the network interface to the IP. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.
AXI4-Stream Slave Interface for Incoming Non-RoCE Traffic				
non_roce_cmac_s_ axis_*	I/O		AXI4-Stream	This interface provides non-RoCE packets from the network interface to the IP. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.
Non-RoCE AXI4-Stream Interface from DMA Module				
non_roce_dma_s_axis_*	I/O		AXI4-Stream	Incoming non-RDMA path from DMA module to IP. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.
Non-RoCE AXI4-Stream Interface to DMA Module				
non_roce_dma_m_axis_*	I/O		AXI4-Stream	AXI4 Outgoing non-RDMA path from IP to DMA module. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.
CMAC AXI4-Stream Interface				
cmac_m_axis_*	I/O		cmac_tx_clk	This interface is used to send out RoCE and non-RoCE packets from ERNIC to CMAC.
WQE Processor AXI Master Interface				
wqe_proc_top_m_axi_*	I/O		AXI4	This interface is used by the WQE processor engine to read the data sent in outgoing RDMA SEND/WRITE and READ responses from the local buffer. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.
WQE Processor AXI4 Master Interface to DDR				
wqe_proc_wr_ddr_m_axi_*	I/O		AXI4	This interface is used by the WQE processor to write the data of outgoing write packets in the write retry buffer. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.

Table 2-5: **ERNIC IP Ports *(Cont'd)***

Name	I/O	Width	Clock Domain	Description
AXI4-Lite Slave Interface for Register Programming				
s_axi_lite_*	I/O		AXI4-Lite	This interface is used by the processor to configure the ERNIC registers. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.
QP Manager AXI4 Master Interface				
qp_mgr_m_axi_*	I/O		AXI4	This interface is used by the QP manager to fetch the send queue WQEs from the DDR. See Appendix A of the *Vivado Design Suite: AXI Reference Guide* (UG1037) [Ref 8] for more information on the AXI4 signal.
Invalidate or Immediate Data AXI4-Stream Interface				
ieth_immdt_m_axis_*	I/O		AXI4S	This interface is used to provide IETH or Immdt headers along with 32-bit additional data to external hardware logic. The use of this information is left to user application.
HW Handshake Ports for RQ Doorbells				
rx_pkt_hndler_o_rq_db_data	O	32	AXI4	RDMA-SEND Producer Index Doorbell Value from ERNIC
rx_pkt_hndler_o_rq_db_addr	O	10	AXI4	RDMA-SEND Producer Index Doorbell Address (4 Bytes per QP; for 127 QPs)
rx_pkt_hndler_o_rq_db_data_valid	O	1	AXI4	RDMA-SEND Producer Index Doorbell Valid. When this signal is asserted High, rx_pkt_hndler_o_rq_db_addr and rx_pkt_hndler_o_rq_db_data are valid. Until rx_pkt_hndler_i_rq_db_rdy is not sampled High, this signal remains asserted and, rx_pkt_hndler_o_rq_db_addr and rx_pkt_hndler_o_rq_db_data signals will hold the same values.
rx_pkt_hndler_i_rq_db_rdy	I	1	AXI4	Ready from the target signaling data and address are accepted
HW Handshake Ports for CQ Doorbells				
resp_hndler_o_send_cq_db_cnt	O	32	AXI4	Send WQE Completion queue Doorbell count from ERNIC
resp_hndler_o_send_cq_db_addr	O	10	AXI4	Send WQE Completion queue Doorbell address (4 Bytes per QP; for 127 QPs)

Table 2-5: **ERNIC IP Ports *(Cont'd)***

Name	I/O	Width	Clock Domain	Description
resp_hndler_o_send_cq_db_cnt_valid	O	1	AXI4	Send WQE Completion Doorbell Valid. When this signal is asserted High, resp_hndler_o_send_cq_db_addr and resp_hndler_o_send_cq_db_cnt are valid. Until resp_hndler_i_send_cq_db_rdy is not sampled High, this signal remains asserted and, resp_hndler_o_send_cq_db_addr and resp_hndler_o_send_cq_db_cnt signals will hold the same values.
resp_hndler_i_send_cq_db_rdy	I	1	AXI4	Send WQE Completion Doorbell Ready. Ready signal should go High when the target application accepts the current doorbell transaction.
HW Handshake Ports for SQ PI Doorbells				
i_qp_sq_pidb_hndshk	I	16	AXI4	Send WQE Producer Index Doorbell Value from target application
i_qp_sq_pidb_wr_addr_hndshk	I	32	AXI4	Send WQE Producer Index Doorbell Address
i_qp_sq_pidb_wr_valid_hndshk	I	1	AXI4	Send WQE Producer Index Doorbell Valid. After the target application posts WQE(s), it should assert this signal High with valid i_qp_sq_pidb_wr_addr_hndshk and i_qp_sq_pidb_hndshk. Target application should keep this signal asserted and hold i_qp_sq_pidb_wr_addr_hndshk and i_qp_sq_pidb_hndshk signals until it samples o_qp_sq_pidb_wr_rdy as High.
o_qp_sq_pidb_wr_rdy	O	1	AXI4	Send WQE Producer Index Doorbell Ready. Ready signal asserted High when ERNIC accepts the Doorbell value.
HW Handshake Ports for RQ CI Doorbells				
i_qp_rq_cidb_hndshk	I	16	AXI4	RDMA-SEND Consumer Index Doorbell value from target application
i_qp_rq_cidb_wr_addr_hndshk	I	32	AXI4	RDMA-SEND Consumer Index Doorbell register address
i_qp_rq_cidb_wr_valid_hndshk	I	1	AXI4	RDMA-SEND Consumer Index Doorbell Valid. After target application processes incoming RDMA-SEND command(s), it should assert this signal High with valid i_qp_rq_cidb_wr_addr_hndshk and i_qp_rq_cidb_hndshk. Target application should keep this signal asserted and hold i_qp_rq_cidb_wr_addr_hndshk and i_qp_rq_cidb_hndshk signals until it samples o_qp_rq_cidb_wr_rdy as High.
o_qp_rq_cidb_wr_rdy	O	1	AXI4	RDMA SEND Consumer Index Doorbell value is accepted by ERNIC

Table 2-5: **ERNIC IP Ports *(Cont'd)***

Name	I/O	Width	Clock Domain	Description
Priority Flow Control Ports				
stat_rx_pause_req[8:0]	I	9	cmac_rx _clk	Pause request signal. This signal gets asserted by CMAC IP for valid quanta period.
ctl_tx_pause_req[8:0]	O	9	cmac_tx _clk	This bus gets asserted when buffer thresholds are between XON and XOFF. priority bit is set through priority register.
ctl_tx_resend_pause	O	1	cmac_tx _clk	This signal is hardwired to 0 and can be connected to CMAC IP signal.
Interrupts				
rnic_intr	O	1	AXI4	This bit is set when any one of the interrupts in the register INTEN occurs.
Debug Counter Enabling Signals				
o_global_dbg_cnt_en	O	1	AXI4	Enables debug signals
o_global_dbg_cnt_clr	O	1	AXI4	Clears the debug counters

Parameter Descriptions

As many features in the ERNIC controller design are controlled using parameters, the controller implementation can be uniquely tailored using only the resources required for the desired functionality. This approach also achieves the best possible performance with the lowest resource usage.

Table 2-6 lists the parameter descriptions and default values.

Table 2-6: **ERNIC Parameters**

Parameter Name	Description	Default Value
C_NUM_QP	Maximum number of Queue Pairs. The minimum number of queue pairs is 8 and the maximum number is 256	8
C_M_AXI_ADDR_WIDTH	AXI Master address width (supported values are 32 and 64)	32
C_M_AXI_ID_WIDTH	AXI Master ID width	1
C_MAX_WR_RETRY_DATA_BUF_DEPTH	Maximum number of data buffers for RDMA WRITE in case of a retransmission. Each RDMA data buffer is 4	512
C_EN_WR_RETRY_DATA_BUF	Enable storing RDMA WRITE data in DDR for retransmission	1
C_EN_INITIATOR_LITE	Enable incoming RDMA READ and RDMA WRITE packets	0
C_EN_DEBUG_PORTS	Enabling the debug ports for debug purposes	0

Register Space

All the ERNIC registers are synchronous to the AXI4-Lite domain. Any bits not specified in register tables below are considered reserved and return the values as 0 upon read. The power-on reset values of control registers are 0 unless specified in the definition. You should always write the reserved locations with a 0 unless stated otherwise. Only address offsets are listed in the table below and the base address is configured by the AXI interconnect at system level. The contents of the protection domain table are used for header validation as shown in the following figure.

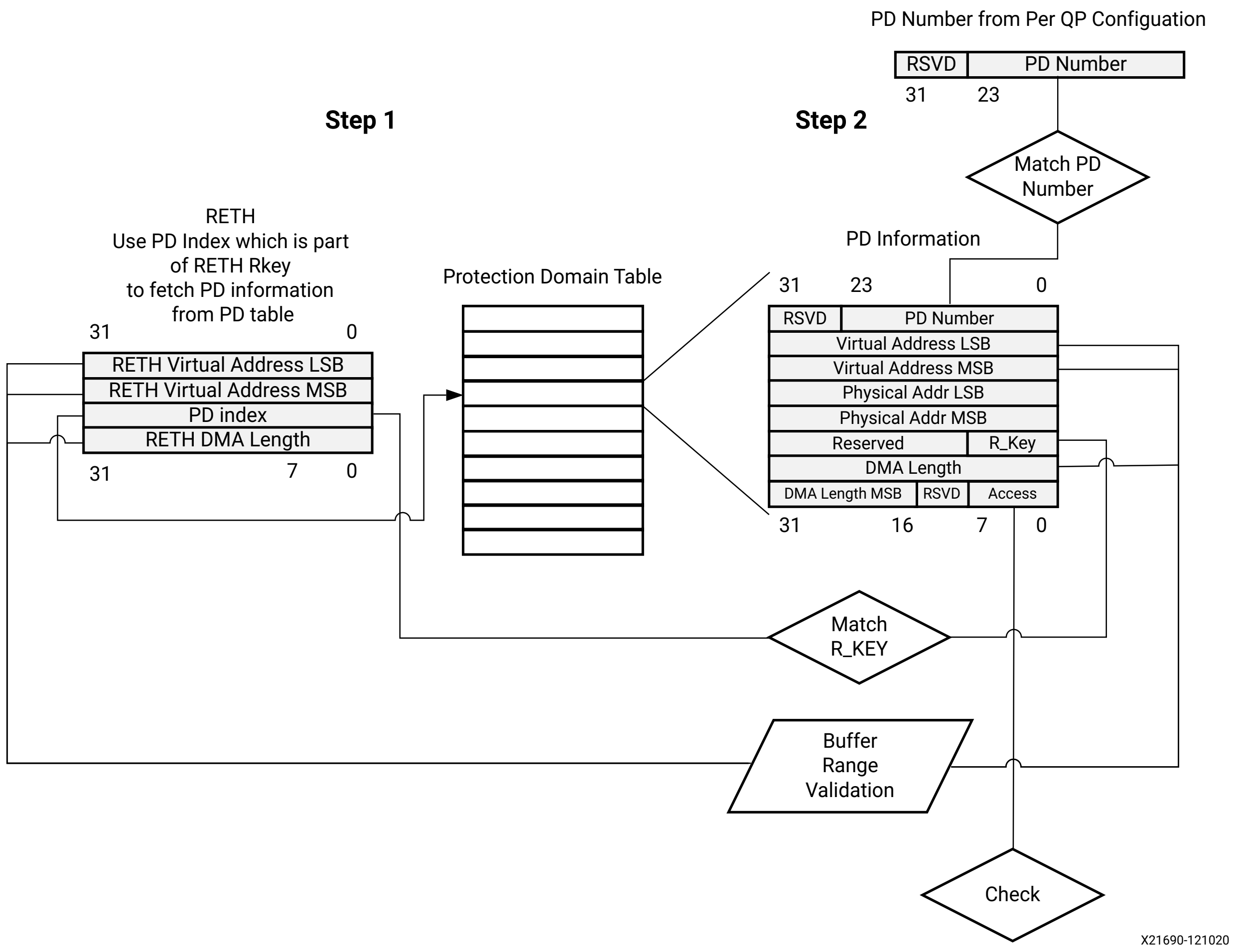

Figure 2-9: **PDT Table Implementation**

Table 2-7: **ERNIC Registers Details**

Address	Access	Register Name	Details
0x00 + ((i) x 0x0100) Where i=0 to 255	RW	Protection Domain Number (PDPDNUM)	Protection Domain Number from the protection domain table Width = 24 Default value = 0
0x04 + ((i) x 0x0100)	RW	Virtual address LSB (VIRTADDRLSB)	This register provides the Virtual Add of the buffer base address LSB. Width = 32 Default value = 0
0x08 + ((i) x 0x0100)	RW	Virtual address MSB (VIRTADDRMSB)	This register provides the Virtual Add of the buffer base address MSB. Width = 32 Default value = 0
0x0C + ((i) x 0x0100)		Buffer Base address LSB (BUFBASEADDRLSB)	This register provides the Write/Read buffer base address MSB. Width = 32 Default value = 0
0x10 + ((i) x 0x0100)	RW	Buffer base address MSB (BUFBASEADDRMSB)	This register provides the Write/Read buffer base address LSB. Width = 32 Default value = 0
0x14 + ((i) x 0x0100)	RW	Buffer R_Key (BUFRKEY)	This register provides the Write/Read buffer R_KEY register. Width = 8 Default value = 0
0x18 + ((i) x 0x0100)	RW	Write/Read buffer length (WRRDBUFLEN)	This register provides the Write/Read buffer length LSB register. Width = 32 Default value = 0

0x1C + ((i) x 0x0100)	RW	Access Description (ACCESSDESC)	This register provides [7:0] the Access description of the Protection Domain and [31:16] Write/Read buffer length MSB register. Default value is 0.
			• [3:0]: Access type
			  ◦ 4'b0000: READ Only
			  ◦ 4'b0001: Write Only
			  ◦ 4'b0010: Read and Write
			  ◦ Other values: Not supported
			• [15:4]: Reserved
			• [31:16]: w_r_buf_len_msb. Register provides the Write/Read buffer length [47:32]

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x20000	RW	XRNIC Configuration (XRNICCONF)	This register provides the basic global (not QP specific) configuration controls for the ERNIC IP • [0]: ERNIC Enable • [1]: Reserved • [2]: Reserved • [4:3]: TX ACK generation. ◦ 00 – (default) ACK only generated on explicit ACK request in the incoming packet or on timeout ◦ 01 – ACK generated only on timeout ◦ 10 – ACK generated only on explicit ACK request in the incoming packet • [5]: Error buffer enable. This bit is set to enable the error buffer • [7:6]: Reserved • [15:8]: Number of QPs enabled. Maximum sequential number • [31:16]: UDP source port for outgoing packets
0x20004	RW	XRNIC Advance Configuration (XRNICADCONF)	This register provides advanced configuration controls for the ERNIC IP. • [0]: SW override enable. Allows SW write access to the following Read Only Registers – CQHEADn, STATCURRSQPTRn, and STATRQPIDBn (where is the QP number) • [1]: Reserved • [2]: retry_cnt_fatal_dis • [15:3]: Reserved • [19:16]: Base count width ◦ Approximate number of system clocks that make 4096us. ◦ For 400 MHz clock -->Program decimal 11 ◦ For 200 MHz clock --> Program decimal 10 ◦ For 125 MHz clock --> Program decimal 09 ◦ For 100 MHz clock --> Program decimal 09 ◦ For N MHz clock ---> Value should be CLOG2(4.096 *N) • [20:23]: Reserved • [31:24] Software Override QP Number
0x20008	RW	XRNIC_BUF_ THRESHOLD_ROCE	This register provides XON and XOFF Configuration for RoCE Traffic. • [15:0]: XON Value for RoCE • [31:16]: XOFF Value for RoCE Default value = 0x965A
0x2000C	RW	XRNIC_PAUSE_CONF	This register provides PFC configuration. • [0]: Pause enable RoCE • [1]: Pause enable non-RoCE • [7:4]: Pause priority for RoCE (Binary encoding) • [11:8]: Pause priority for non-RoCE (Binary encoding) • [12]: Disable priority check (applicable only for TX data path) Default value = 0x800

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x20010	RW	MAC XRNIC Address LSB (MACXADDLSB)	This is the MAC Address for local (ERNIC) device and should be configured before the XRNICCONF[0] is set to 1. • [31:0]: MAC local address LSB
0x20014	RW	MAC XRNIC Address MSB (MACXADDMSB)	MAC Address for local (ERNIC) device. Should be configured before the XRNICCONF[0] is set to 1. • [15:0]: MAC local address MSB • [31:16]: Reserved
0x20018	RW	XRNIC_BUF_ THRESHOLD_NON_ ROCE	This register provides XON and XOFF configuration for non-RoCE traffic. • [15:0]: XON value for non-RoCE • [31:16]: XOFF value for non-RoCE Default value = 0x965A
0x20020	RW	IPv6 Address 1 (IPv6XADD1)	IPv6 address [32:0] for local (ERNIC) device. Should be configured before the XRNICCONF[0] is set to 1. [31:0]: IP address [31:0]
0x20024	RW	IPv6 Address 2 (IPv6XADD2)	IPv6 address [63:32] for local (ERNIC) device. Should be configured before the XRNICCONF[0] is set to 1. [31:0]: IP address [63:32]
0x20028	RW	IPv6 Address 3 (IPv6XADD3)	IPv6 address [95:64] for local (ERNIC) device. Should be configured before the XRNICCONF[0] is set to 1. [31:0]: IP address [95:64]
0x2002C	RW	IPv6 Address 4 (IPv6XADD4)	IPv6 address [127:96] for local (ERNIC) device. Should be configured before the XRNICCONF[0] is set to 1. [31:0]: IP address [127:96]
0x20060	RW	Error Buffer Base Address (ERRBUFBA)	This register provides the base address for Error buffers. The ERNIC IP updates these buffers with incoming packets that fail validation. The writes to this buffer for all validation errors are enabled by writing a 1 to XRNICCONF[5]. If this bit is disabled, only packets that cause the QP to move to a FATAL state are written to the error buffer. • [31:0]: Error buffer base address LSB 32 bits
0x20064	RW	Error Buffer Base Address msb (ERRBUFBAMSB)	This register provides the msb base address for Error buffers. • [31:0]: Error buffer base address MSB 32 bits
0x20068	RW	Error Buffer Size (ERRBUFSZ)	This register provides the size of the error buffer • [15:0]: Number of Error buffers • [31:16]: Size of each Error buffer
0x2006C	RO	Error Buffer write pointer (ERRBUFWPTR)	This register is updated by the ERNIC IP when any error packet is written to the error buffer. The pointer informs the driver of the number of error packets received. • [15:0]: Write pointer doorbell for Error Buffer
0x20070	RW	IPv4 XRNIC Address (IPv4XADD)	IPv4 address for local (ERNIC) device. • [31:0]: IPv4 address

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x20078	RW	Outgoing Pkt Error status Queue Base address (OPKTERRQBA)	This register is used to configure the base address for outgoing packet error status queue. See, ERNIC RX Path for details of the outgoing packet error status queue. [31:0]: Outgoing Pkt Error Status Queue base address
0x2007C	RW	Outgoing Pkt Error status Queue Base address (OPKTERRQBAMSB)	This register is used to configure the msb base address for outgoing packet error status queue. See, ERNIC RX Path for details of the outgoing packet error status queue. [63:32]: Outgoing Pkt Error Status Queue base address msb
0x20080	RW	Outgoing Error status Q size (OUTERRSTSQSZ)	[15:0]: Number of entries in outgoing error Q.
0x20084	RW	Outgoing Error status Q write pointer Doorbell (OPTERRSTSQQPTRDB)	Outgoing Error status Q write pointer doorbell. The hardware keeps writing to the buffer in a circular fashion. Does not take care of buffer overflow. Needs to be handled in SW.
0x20088	RW	Incoming Pkt Error Status Queue Base address (IPKTERRQBA)	This register is used to configure the base address for incoming packet error status queue. See, ERNIC RX Path for details of the incoming packet error status queue. [31:0]: Incoming Pkt Error Status Queue base address
0x2008C	RW	Incoming Pkt Error Status Queue Base address MSB (IPKTERRQBAMSB)	This register is used to configure the msb base address for incoming packet error status queue.
0x20090	RW	Incoming Pkt Error Status Queue Size (IPKTERRQSZ)	This register is used to configure the size of the incoming packet error status queue. • [15:0]: Number of incoming error pkt status queue entries • [31:16]: Reserved
0x20094	RO	Incoming Pkt Error Status write pointer (IPKTERRQWPTR)	This status register provides information on the number of incoming error packets received by the ERNIC IP. [15:0]: Write pointer doorbell for incoming error status queue. ERNIC IP writes to the queue in a circular manner without taking care of overflow. This needs to be handled in SW.
0x200A0	RW	Data Buffer Base Address (DATBUFBA)	These data buffers are used by the ERNIC to save all outgoing RDMA WRITE data until it is acknowledged by the remote host. In the event of retransmission, the retried data is pulled from these buffers. [31:0]: Data Buffer base address
0x200A4	RW	Data Buffer Base Address MSB (DATBUFBAMSB)	These data buffers are used by the ERNIC to save all outgoing RDMA WRITE data until it is acknowledged by the remote host. In the event of retransmission, the retried data is pulled from these buffers. [63:32]: Data Buffer base address MSB
0x200A8	RW	Data Buffer Size (DATBUFSZ)	This register is used to configure the size of the data buffers. • [15:0]: Number of data buffers • [31:16]: Data buffer size in bytes

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x200AC	RW	Connect IP configuration (CON_IO_CONF)	This register is used to ring the doorbell when hardware handshake is just enabled. • [15:0]: Reserved • [23:16]: Queue pair ID • [30:24]: Reserved • [31]: Connect IO ready indication. This bit is read only and when set to 1'b1 indicates that the connect IO configuration is done and ready to accept for another QPID
0x200B0	RW	Response Error pkt buffer base address (RESPERRPKTBA)	These error buffers are used by the ERNIC to save all error response pkt msb base address. the retried addresses are pulled from these buffers. [31:0]: Response Error pkt Buffer base address msb
0x200B4	RW	Response Error pkt buffer base address msb (RESPERRPKTBAMSB)	These error buffers are used by the ERNIC to save all error response pkt msb base address. the retried addresses are pulled from these buffers. [31:0]: Response Error pkt Buffer base address msb
0x200B8	RW	Response Error pkt buffer size address (RESPERRSZ)	These error buffers are used by the ERNIC to save all error response pkt size. It is used during retry. [31:0]: Response Error Buffer size address
0x200BC	RW	Response Error pkt buffer size address (RESPERRSZMSB)	These error buffers are used by the ERNIC to save all error response pkt size address msb. It is used during retry. [63:32]: Response Error Buffer size address msb
Global Status Registers			
0x20100	RO	Incoming SEND/Read Response Pkt count (INSRRPKTCNT)	This is a status register which provides information about incoming RDMA SEND and RDMA READ response packets. • [15:0]: Incoming SEND packet count • [31:16]: Incoming Read Response packet count
0x20104	RO	Incoming ACK/MAD Pkt count (INAMPKTCNT)	This is a status register which provides information about incoming acknowledgment and Management Datagram (MAD) packets. • [15:0]: Incoming ACK packet count • [31:16]: Incoming MAD packet count
0x20108	RO	Outgoing IO (SEND/READ/WRITE) Pkt count (OUTIOPKTCNT)	This is a status register which provides information about outgoing RDMA SEND and RDMA READ/WRITE packets. • [15:0]: Outgoing SEND packet count • [31:16]: Outgoing RDMA READ/WRITE packet count
0x2010C	RO	Outgoing ACK/MAD Pkt count (OUTAMPKTCNT)	This is a status register which provides information about outgoing acknowledgment and MAD packets. • [15:0]: Outgoing ACK packet count • [31:16]: Outgoing MAD packet count

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x20110	RO	Last incoming pkt (LSTINPKT)	This status register provides details of the last incoming packet received. • [7:0]: Opcode of the last incoming packet • [15:8]: QPID of the last incoming packet • [31:16]: PSN LSB bits of the last incoming packet
0x20114	RO	Last outgoing pkt (LSTOUTPKT)	This status register provides the details of the last outgoing packet. • [7:0]: Opcode of the last outgoing packet • [15:8]: QPID of the last outgoing packet • [31:16]: PSN LSB bits of the last outgoing packet
0x20118	RO	Incoming Invalid/ Duplicate pkt count (ININVDUPCNT)	This status register provides information on incoming invalid or duplicate packets. • [15:0]: Incoming Invalid packet count • [31:16]: Incoming Duplicate packet count
0x2011C	RO	Incoming NAK pkt Status (INNCKPKTSTS)	This status register is provides details of incoming Responder Not Ready (RNR) NAK, Retry Count expired or other NAK received. • [7:0]: QPID for incoming NAK packet • [15:8]: Reserved • [31:16]: Incoming NAK packet count
0x20120	RO	Outgoing RNR pkt Status (OUTRNRPKTSTS)	This status register is provides details of outgoing Responder Not Ready (RNR) NAK sent. • [7:0]: QPID for outgoing RNR Packet • [15:8]: Reserved • [31:16]: Count of RNR Packets sent
0x20124	RO	WQE Processor Status (WQEPROCSTS)	This status register provides the status of WQE processor engine and is used for debug purposes. • [2:0]: WQE processor state • [3]: reserved • [4]: retry buffers unavailable • [5]: internal FIFO empty status • [6]: Buffer manager busy signal • [11:7]: reserved • [12]: Back pressure indication from CMAC. Sticky bit • [15:12]: Header buffer manager FSM state • [23:16]: Buffer tail pointer • [31:24]: Buffer head pointer
0x2012C	RO	QP Manager Status (QPMSTS)	This status register provides the status of QP manager and is used for debug purposes. • [0]: WQE Cache full • [1]: WQE Cache empty • [15:2]: Reserved • [31:16]: Count of WQEs processed

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x20130	RO	Incoming all/dropped pkt count (INALLDRPPKTCNT)	This status register is provides details of all incoming and dropped packets. • [15:0]: All incoming packet count • [31:16]: Incoming dropped packet count
0x20134	RO	Incoming NAK pkt count (INNAKPKTCNT)	This status register is provides details of all incoming NAK packets. • [15:0]: Incoming NAK packet count • [31:16]: Reserved
0x20138	RO	Outgoing NAK pkt count (OUTNAKPKTCNT)	This status register is provides details of all outgoing NAK packets. • [15:0]: Outgoing NAK packet count • [31:16]: Reserved
0x2013C	RO	Response handler Status (RESPHNDSTS)	This status register provides the status of Response Handler module and is used for debug purposes. • [7:0]: Arbitrated QP Index • [13:8]: Response handler FSM state • [16:14]: Reserved • [17]: Acks to process • [31:18]: Reserved
0x20140	RO	Retry count status (RETRYCNTSTS)	This status register provides details of retransmitted packets. [31:0] Retry count
0x20174	RO	Incoming CNP packet count (INCNPPKTCNT)	Incoming CNP packet count
0x20178	RO	Outgoing CNP packet count (OUTCNPPKTCNT)	Outgoing CNP packet count
0x2017C	RO	Outgoing read response packet count (OUTRDRSPPKTCNT)	Outgoing read response packet count
0x20180	RW	Interrupt Enable (INTEN)	This register provides the configuration control for interrupts generated by the ERNIC IP. • [0]: Incoming packet validation error interrupt enable • [1]: Incoming MAD packet received interrupt enable • [2]: Reserved • [3]: RNR NACK generated interrupt enable • [4]: WQE completion interrupt enable (asserted for QPs for which QPCONF[3] bit is set) • [5]: Illegal opcode posted in SEND Queue interrupt enable • [6]: RQ Packet received interrupt enable (asserted for QPs for which QPCONF[2] bit is set • [7]: Fatal error received interrupt enable • [8]: Interrupt enable for CNP scheduling

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x20184	W1C	Interrupt Status (INTSTS)	This status register provides the cause of an asserted interrupt. • [0]: Incoming packet validation error interrupt • [1]: Incoming MAD packet received interrupt • [2]: Reserved • [3]: RNR NACK generated interrupt • [4]: WQE completion interrupt (asserted for QPs for which QPCONF[3] bit is set) • [5]: Illegal opcode posted in SEND Queue interrupt • [6]: RQ Packet received interrupt (asserted for QPs for which QPCONF[2] bit is set) • [7]: Fatal error received interrupt • [8]: Interrupt enable for CNP scheduling
0x20190	WC1	Rcv Q interrupt status (bitwise) QP 0-31 (RQINTSTS1)	This register provides the RQ interrupt status for QPs 0 to 31. Bit [i] provides the interrupt status for RQi where i=0 to 31. [i]: Interrupt status for RQ i
0x20194	WC1	Rcv Q interrupt status (bitwise) QP 32-63 (RQINTSTS2)	This register provides the RQ interrupt status for QPs 32 to 63. Bit [i] provides the interrupt status for RQ(i+32) where i=0 to 31. [i]: Interrupt status for RQ(i+32)
0x20198	WC1	Rcv Q interrupt status (bitwise) QP 64-95 (RQINTSTS3)	This register provides the RQ interrupt status for QPs 64 to 95. Bit [i] provides the interrupt status for RQ(i+64) where i=0 to 31. [i]: Interrupt status for RQ(i+64)
0x2019C	WC1	Rcv Q interrupt status (bitwise) QP 96-127 (RQINTSTS4)	This register provides the RQ interrupt status for QPs 96 to 127. Bit [i] provides the interrupt status for RQ(i+96) where i=0 to 31. [i]: Interrupt status for RQ(i+96)
0x201A0	WC1	Rcv Q interrupt status (bitwise) QP 128-159 (RQINTSTS5)	This register provides the RQ interrupt status for QPs 128 to 159. Bit [i] provides the interrupt status for RQ(i+128) where i=0 to 31. [i]: Interrupt status for RQ(i+128)
0x201A4	WC1	Rcv Q interrupt status (bitwise) QP 160-191 (RQINTSTS6)	This register provides the RQ interrupt status for QPs 160 to 191. Bit [i] provides the interrupt status for RQ(i+160) where i=0 to 31. [i]: Interrupt status for RQ(i+160)
0x201A8	WC1	Rcv Q interrupt status (bitwise) QP 192-223 (RQINTSTS7)	This register provides the RQ interrupt status for QPs 192 to 223. Bit [i] provides the interrupt status for RQ(i+192) where i=0 to 31. [i]: Interrupt status for RQ(i+192)
0x201AC	WC1	Rcv Q interrupt status (bitwise) QP 224-255 (RQINTSTS8)	This register provides the RQ interrupt status for QPs 224 to 255. Bit [i] provides the interrupt status for RQ(i+224) where i=0 to 31. [i]: Interrupt status for RQ(i+224)
0x201B0	WC1	Completion Q interrupt status (bitwise) QP 0-31 (CQINTSTS1)	This register provides the CQ interrupt status for QPs 0 to 31. Bit [i] provides the interrupt status for CQi where i=0 to 31. [i]: Interrupt status for CQ i
0x201B4	WC1	Completion Q interrupt status (bitwise) QP 32-63 (CQINTSTS2)	This register provides the CQ interrupt status for QPs 32 to 63. Bit [i] provides the interrupt status for CQ(i+32) where i=0 to 31. [i]: Interrupt status for CQ(i+32)

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x201B8	WC1	Completion Q interrupt status (bitwise) QP 64-95 (CQINTSTS3)	This register provides the CQ interrupt status for QPs 64 to 95. Bit [i] provides the interrupt status for CQ(i+64) where i=0 to 31. [i]: Interrupt status for CQ(i+64)
0x201BC	WC1	Completion Q interrupt status (bitwise) QP 96-127 (CQINTSTS4)	This register provides the CQ interrupt status for QPs 96 to 127. Bit [i] provides the interrupt status for CQ(i+96) where i=0 to 31. [i]: Interrupt status for CQ(i+96)
0x201C0	WC1	Completion Q interrupt status (bitwise) QP 128-159 (CQINTSTS5)	This register provides the CQ interrupt status for QPs 128 to 159. Bit [i] provides the interrupt status for CQ(i+128) where i=0 to 31. [i]: Interrupt status for CQ(i+128)
0x201C4	WC1	Completion Q interrupt status (bitwise) QP 160-191 (CQINTSTS6)	This register provides the CQ interrupt status for QPs 160 to 191. Bit [i] provides the interrupt status for CQ(i+160) where i=0 to 31. [i]: Interrupt status for CQ(i+160)
0x201C8	WC1	Completion Q interrupt status (bitwise) QP 192-223 (CQINTSTS7)	This register provides the CQ interrupt status for QPs 192 to 223. Bit [i] provides the interrupt status for CQ(i+192) where i=0 to 31. [i]: Interrupt status for CQ(i+192)
0x201CC	WC1	Completion Q interrupt status (bitwise) QP 224-255 (CQINTSTS8)	This register provides the CQ interrupt status for QPs 224 to 255. Bit [i] provides the interrupt status for CQ(i+224) where i=0 to 31. [i]: Interrupt status for CQ(i+224)
0x201D0	RO	CNP Packed scheduled interrupt status (bitwise) QP 0-31 (CNPSCHDSTS1REG)	This register provides the CNP scheduled interrupt for QPs 0 to 31.
0x201D4	RO	CNP Packed scheduled interrupt status (bitwise) QP 32-63 (CNPSCHDSTS2REG)	This register provides the CNP scheduled interrupt for QPs 32 to 64.
0x201D8	RO	CNP Packed scheduled interrupt status (bitwise) QP 64-95 (CNPSCHDSTS3REG)	This register provides the CNP scheduled interrupt for QPs 64 to 95.
0x201DC	RO	CNP Packed scheduled interrupt status (bitwise) QP 96-127 (CNPSCHDSTS4REG)	This register provides the CNP scheduled interrupt for QPs 96 to 127.
0x201E0	RO	CNP Packed scheduled interrupt status (bitwise) QP 128-159 (CNPSCHDSTS5REG)	This register provides the CNP scheduled interrupt for QPs 128 to 159.

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x201E4	RO	CNP Packed scheduled interrupt status (bitwise) QP 160-191 (CNPSCHDSTS6REG)	This register provides the CNP scheduled interrupt for QPs 160 to 191.
0x201E8	RO	CNP Packed scheduled interrupt status (bitwise) QP 192-223 (CNPSCHDSTS7REG)	This register provides the CNP scheduled interrupt for QPs 192 TO 223.
0x201EC	RO	CNP Packed scheduled interrupt status (bitwise) QP 224-255 (CNPSCHDSTS8REG)	This register provides the CNP scheduled interrupt for QPs 224 TO 255.

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
Per QP Registers			
0x20200 + ((i-1) x 0x0100)	RW	QP Configuration QPi (QPCONFi)	This register defines the basic configuration of QP. This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). Register offset for per-QP is 0x20200, 0x20204 and so on for the subsequent offsets. • [0]: QP enable – Should be set to 1 for all active QPs. A disabled QP will not be able to receive or transmit packets. • [2]: RQ interrupt enable – When enabled, allows the receive queue interrupt to be generated for every new packet received on the receive queue • [3]: CQ interrupt enable – When enabled, allows the completion queue interrupt to be generated for every send work queue entry completion • [4]: HW Handshake disable – This bit when reset to 0 enables the HW handshake ports for doorbell exchange. If set, all doorbell values are exchanged through writes through the AXI4 or AXI4-Lite interface. • [5]: CQE write enable – This bit when set, enables completion queue entry writes. The writes are disabled when this bit is reset. CQE writes can be enabled to debug failed completions. • [6]: QP under recovery. This bit need to be set in the fatal clearing process. • [7]: QP configured for IPv4 or IPv6 ◦ 0 - IPv4 ◦ 1 - IPv6 • [10:8]: Path MTU ◦ 000 – 256B (default) ◦ 001 – 512B ◦ 010 – 1024B ◦ 011 – 2048B ◦ 100 - 4096B ◦ 101 to 111 - Reserved • [31:16]: RQ Buffer size (in multiple of 256B). This is the size of each buffer element in the request and not the size of the entire request. The programmed value should be the power of 2 for expected behavior.
0x20204 + ((i-1) x 0x0100)	RW	QP advanced configuration QPi (QPADVCONFi)	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). This register is not present for QP1. • [5:0]: Traffic class (keep default reset value) • [15:8]: Time to live • [31:16]: Partition Key
0x20208 + ((i-1) x 0x0100)	RW	Rcv Q Buffer base address QPi (RQBAi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register provides the base address of the RDMA Receive queue buffer. [31:8]: Receive Q Buffer Base address addr (256 B aligned). 256 B is the total (individual rq buffer element size)* rq_depth

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x202C0 + (i-1) x 0x0100)	RW	Rcv Q Buffer base address QPi (RQBAMSBi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register provides the msb base address of the RDMA Receive queue buffer. [63:32]: Receive Q Buffer Base address addr (256 B aligned). 256 B is the total (individual rq buffer element size)* rq_depth
0x20210 + ((i-1) x 0x0100)	RW	SEND Q base address QPi (SQBAi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register provides the base address of the RDMA Send queue buffer. [31:5]: Send Q base address (32 B aligned)
0x202C8 + ((i-1) x 0x0100)		SEND Q base address msb QPi (SQBAMSBi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register provides the msb base address of the RDMA Send queue buffer. [63:32]: Send Q base address (32 B aligned)
0x20218 + ((i-1) x 0x0100)	RW	CQ base address QPi (CQBAi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register provides the base address of the RDMA Completion queue buffer. [31:5]: Send CQ base address (32 B aligned)
0x202D0 + ((i-1) x 0x0100)	RW	CQ base address msb QPi (CQBAMSBi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register provides the msb base address of the RDMA Completion queue buffer. [63:32]: Send CQ base address (32 B aligned)
0x20220 + ((i-1) x 0x0100)	RW	Rcv Q Write pointer DB address QPi (RQWPTRDBADDi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register provides the address of the Receive Queue doorbell register. Upon reception of a new incoming RDMA SEND packet, the ERNIC IP updates the RQ doorbell values in the address pointed to by this register. [31:0]: Rcv Q write pointer Doorbell address
0x20224 + ((i-1) x 0x0100)	RW	Rcv Q Write pointer DB address msb QPi (RQWPTRDBADDMSBi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register provides the msb address of the Receive Queue doorbell register. Upon reception of a new incoming RDMA SEND packet, the ERNIC IP updates the RQ doorbell values in the address pointed to by this register. [63:32]: Rcv Q write pointer Doorbell address msb
0x20228 + ((i-1) x 0x0100)	RW	CQ DB address QPi (CQDBADDi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). this register provides the address of the Completion Queue doorbell register. Upon completion of a new SEND Work queue entry, the ERNIC IP updates the CQ doorbell values in the address pointed to by this register. [31:0]: Send CQ Doorbell address
0x2022C + ((i-1) x 0x0100)	RW	CQ DB address QPi (CQDBADDMSBi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). this register provides the msb address of the Completion Queue doorbell register. Upon completion of a new SEND Work queue entry, the ERNIC IP updates the CQ doorbell values in the address pointed to by this register. [63:32]: Send CQ Doorbell address

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x20230 + ((i-1) x 0x0100)	RO	CQ head pointer QPi (CQHEADi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This status register gives the Send completion Queue doorbell value and provides information about the Send WQEs that have been completed. This is the doorbell value that is written by the ERNIC IP to the address pointed to by the CQDBADDi register. • [15:0]: CQ head pointer • [31:0]: Reserved
0x20234 + ((i-1) x 0x0100)	RW	RQ Consumer Index QPi (RQCIi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register is updated by the target application on consuming a new receive queue entry. Once consumed, the RQ entry can be over written by the ERNIC IP. The RQCI index can also be updated through the side band interface. • [15:0]: Rcv Q Consumer Index Doorbell • [31:16]: Reserved
0x20238 + ((i-1) x 0x0100)	RW	SQ Producer index QPi (SQPIi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register is updated by the target application when it posts a new Send Work Queue entry. The SQPI can also be updated through the side band interface. The Send Queue is a circular buffer. The CQHEAD register for the corresponding QP provides information about the Work Queue entries that have been completed and can be over written by the target application. • [15:0]: Send Q Producer Index Doorbell • [31:16]: Reserved
0x2023C + ((i-1) x 0x0100)	RW	Q Depth QPi (QDEPTHi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register defines the queue depths for send, completion and receive queues. • [15:0]: Send Q depth (Send CQ will of the same depth) • [31:16]: Receive Q depth
0x20240 + ((i-1) x 0x0100)	RW	SEND Q PSN for QPi (SQPSNi)	This register is initialized at connection time by the SW. After that the HW updates it for every outgoing packet and should not be updated by the SW. This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). This register does not exist for QP1. [23:0]: Send Q PSN
0x20244 + ((i-1) x 0x0100)	RW	Last RQ req for QPi (LSTRQREQi)	This register provides the last incoming RQ packet details. This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). This register does not exist for QP1. • [23:0]: Rcv Q PSN • [31:24]: Rcv Q opcode
0x20248 + ((i-1) x 0x0100)	RW	Destination QP configuration for QPi (DESTQPCONFi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register is configured at connection time by the SW and provides the remote QPID connected to this QP. All outgoing packets from this QP are sent with this QPID as the destination QPID. [23:0]: Destination Connected QPID

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x20250 + ((i-1) x 0x0100)	RW	MAC destination address LSB QPi (MACDESADDLSBi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register is configured at connection time by the SW and provides the MAC address of the remote host connected to this QP. All outgoing packets from this QP are sent with this MAC address (LSB) as the destination MAC address. [31:0]: MAC destination address LSB
0x20254 + ((i-1) x 0x0100)	RW	MAC destination address MSB QPi (MACDESADDMSBi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). This register is configured at connection time by the SW and provides the MAC address of the remote host connected to this QP. All outgoing packets from this QP are sent with this MAC address (MSB) as the destination MAC address. [15:0]: MAC destination address MSB
0x20260 + ((i-1) x 0x0100)	RW	IP destination address 1 QPi (IPDESADDR1i)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). If the remote host for this QP is configured using the IPv4 protocol, then this register indicates an IPv4 address. If the remote host is configured using the IPv6 protocol, this register then indicates the IPv6 destination address. [31:0]: IP Destination address LSB1
0x20264 + ((i-1) x 0x0100)	RW	IP destination address 2 QPi (IPDESADDR2i)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). If the remote host is configured using the IPv6 protocol, this register indicates an IPv6 destination address [63:32]. If the remote host of this QP is configured using the IPv4 protocol, then this register is not used. [31:0]: IP Destination address LSB2
0x20268 + ((i-1) x 0x0100)	RW	IP destination address 3 QPi (IPDESADDR3i)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). If the remote host is configured using the IPv6 protocol, this register indicates an IPv6 destination address [95:64]. IF the remote host of this QP is configured using the IPv4 protocol, then this register is not used. [31:0]: IP Destination address MSB1
0x2026C + ((i-1) x 0x0100)	RW	IP destination address 4 QPi (IPDESADDR4i)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). If the remote host is configured using the IPv6 protocol, this register indicates an IPv6 destination address [127:96]. If the remote host of this QP is configured using the IPv4 protocol, then this register is not used. [31:0]: IP Destination address MSB2

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x2024C + ((i-1) x 0x0100)	RW	Timeout Configuration Register for QPi (TIMEOUTCONFi)	This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). It provides the timeout configuration QPi. This register does not exist for QP1. • [5:0]: Timeout value • [7:6]: Reserved • [10:8]: Maximum retry count • [13:11]: Maximum RNR retry count • [15:14]: Reserved • [20:16]: RNR NACK Timeout value for outgoing packets • [31:21]: Reserved
0x20280 + ((i-1) x 0x0100)	RO	Status Sender sequence number QPi (STATSSNi)	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). This register does not exist for QP1. [23:0]: Current outgoing SSN (same as outgoing MSN)
0x20284 + ((i-1) x 0x0100)	RO	Status Message sequence number QPi (STATMSN)	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). This register does not exist for QP1. [23:0]: Current expected incoming MSN
0x20288 + ((i-1) x 0x0100)	RO	Status QPi (STATQPi)	This register provides the QP status for every QP. This register is generated based on the parameter C_NUM_QP (where i = 1 to C_NUM_QP). • [0]: QP in fatal status • [1]: Rcv Q ovfl (outgoing RNR NACK) • [2]: Send Q full • [3]: Outstanding Q full • [4]: CQ FIFO full • [8:5]: Reserved • [9]: Send Q empty. This bit signifies that there are no SQEs left to process. However, it does not imply that all the SEND WQEs were acknowledged by the remote host. • [10]: Outstanding Q empty • [11]: QP packet was retried • [15:12]: Reserved • [22:16]: NACK syndrome received (Read/Write) • [23]: Reserved • [26:24]: Current retry count (Read/Write) • [27]: Reserved • [30:28]: Current RNR nack count for inc resp packets (Read/ Write) • [31]: Reserved
02x028C + ((i-1) x 0x0100)	RO	Status Current SQ ptr under process (STATCURSQPTRi)	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). [15:0]: Current SQ pointer that is under process. This shows the number of WQEs that are outstanding and awaiting a response from the remote host.

Table 2-7: **ERNIC Registers Details *(Cont'd)***

Address	Access	Register Name	Details
0x20290 + ((i-1) x 0x0100)	RO	Status Response PSN for QPi (STATRESPSNi)	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). This register does not exist for QP1. [23:0]: Expected response PSN
0x20294 + ((i-1) x 0x0100)	RO	Status RQ buffer current address for QPi (STATRQBUFCAi)	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). [31:8]: Receive Q Buffer current addr (256 B aligned)
0x202D8 + ((i-1) x 0x0100)	RO	Status RQ buffer current msb address for QPi (STATRQBUFCAMSBi)	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). [63:32]: Receive Q Buffer current addr msb (256 B aligned)
0x20298 + ((i-1) x 0x0100)	RO	Status of WQEs posted to QPi (STATWQEi)	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). [15:0]: Count of WQEs pushed by QP MGR for this QP
0x2029C + ((i-1) x 0x0100)	RO	Status RQ Producer index DB QPi (STATRQPIDBi)	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). [15:0]: Rcv Q Producer Index DB
0x202B0 + ((i-1) x 0x0100)	RW	Protection domain number	This register is generated based on the parameter C_NUM_QP (where i = 2 to C_NUM_QP). This register is 24-bit and contains the PD number assigned to the QP. ***Note:*** Only one memory region can be associated per QP.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the ERNIC IP core.

General Design Guidelines

A typical ERNIC subsystem would include one or more MAC hard/soft IPs. An AXI interconnect would also be required to connect the various AXI interfaces exposed by the ERNIC IP. A DRAM can be a part of the solution which would require a DRAM controller to be instantiated as well.

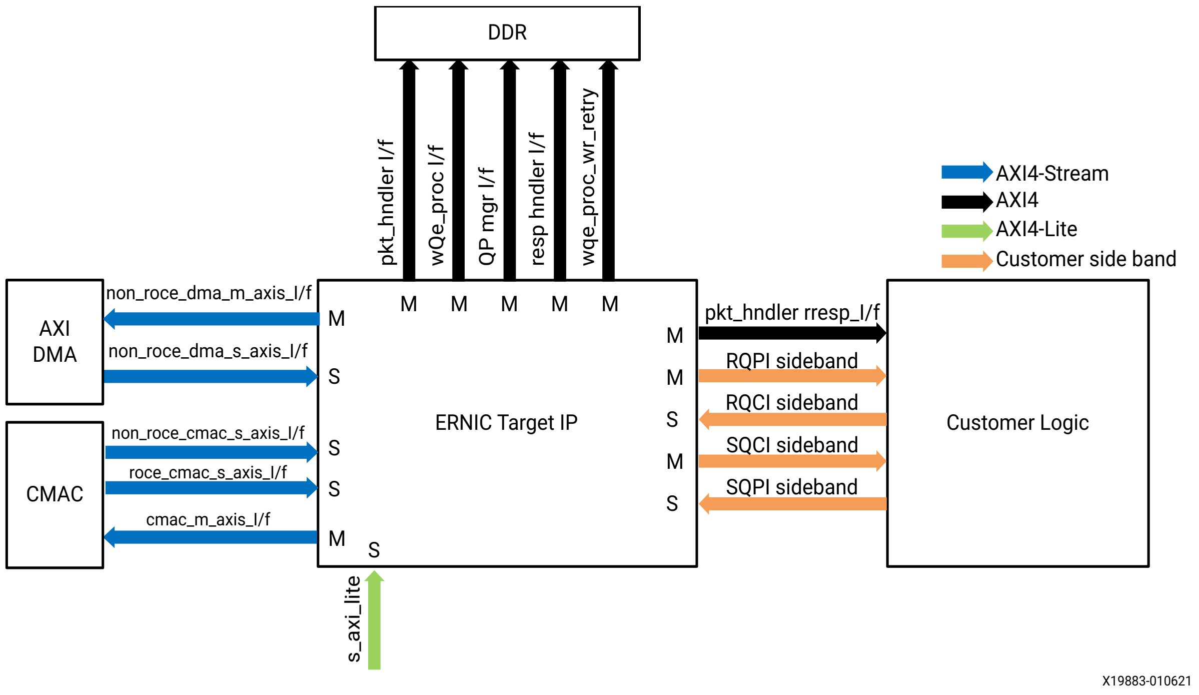

Figure 3-1: **ERNIC Interfaces**

Figure 3-1 shows the various ERNIC IP interfaces. The interfaces shown as interfacing with DDR may interface with any memory mapped region. Refer to Table 2-5 for details on each of these interfaces. The AXI4 and the AXI4-Stream interfaces are 512 bits wide and are mainly used for data transfers. The ERNIC IP provides sideband interfaces to allow for efficient exchange of queue pair related doorbells. These side band interfaces can be enabled or disabled for each queue pair (QP) based on the configuration.

The ERNIC IP has one AXI4-Lite slave interface to access the register space. The details of the memory map required for this slave interface is shown in the following table.

Table 3-1: **Address Space Allocation Requirement for Slave Interfaces**

Slave Interface	Size	Unit	Description
ERNIC AXI4-Lite slave interface	256	KB	ERNIC register interface

Apart from these, the IP also requires some memory regions to be allocated for some specific data structures. These memory regions may be mapped to a local DRAM or an AXI BRAM or any other memory mapped slave. Ensure that there is adequate bandwidth on these memory interfaces based on the line rate that ERNIC should achieve.

Table 3-2: **ERNIC IP Memory Requirement**

Memory Region	Size	Unit	Description
Error buffer	64	KB	256 packets of 256 bytes each. Packets that fail packet validation are sent to the error buffer along with 4 bytes of error syndrome
Send Queue	2	MB	256 QPs of depth 128 locations and each SQE is of 64 bytes each
Receive Queue	32	MB	256 QPs of depth 128 locations and each RQE is of 1024 bytes each
Send completion Queue	128	KB	256 QPs of depth 128 locations and each CQE is of 4 bytes each
Write Retry buffers	16	MB	Buffers of 4K size each for 256 QPs and each QP with up to 16 outstanding transactions

Notes:

1. Use the Vivado implementation strategy — Performance_RefinePlacement — when using the ERNIC with 256 QPs.

Interrupts

ERNIC IP provides a single interrupt line, which is generated on different interrupts status signals defined in INTEN register. Each of these interrupt lines can be enabled by writing a 1 to the corresponding bits of the Interrupt Enable INTEN register. On receiving an ORed interrupt the SW can read the INTSTS register to know the cause of the interrupt.

Interrupt bits 4 and 6 inform the drivers about a WQE completion or an incoming RDMA SEND respectively. These interrupts can be enabled or disabled on a per QP basis. Bit [2] of the QPCONFi registers allows selective enabling of Receive Queue interrupts per QP. Similarly, bit [3] of the QPCONFi registers allows selective enabling of Send Completion Queue interrupts. In general QPs that require SW handling should have this option enabled. The QPs that are directly handled by the hardware will be informed through the hardware handshake ports and corresponding interrupt enable bits can be disabled. Such QPs should have the hardware handshake disable QPCONFi[4] bit reset to 0.

The RQINTSTSn and CQINTSTSn registers provide bitwise information about the QPs that have a pending RQ or CQ entry to be serviced.

These registers should be read by the SW on receiving an RQ or CQ interrupt respectively to know the QPs that are required to be serviced. These interrupt status should be cleared upon successful handling by writing a 1 to the respective bits.

Clocking

Two clocks are exposed at the top of the ERNIC IP. These are: AXI4 clock and AXI4-Lite clock. All the registers accesses work on the AXI4-Lite clock while the rest of the logic works on AXI4 clock. Typically, the AXI4 clock would be of higher frequency (up to 200 MHz) while the AXI4-Lite interface could be clocked at a lower frequency (divided AXI4 clock with the divided clock edges aligning with the AXI4 clock). However, these clocks are treated as synchronous inside the ERNIC IP and are expected to be generated from the same clock source.

Resets

The ERNIC IP requires two active-Low resets that are synchronized to the two clock domains, respectively.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 2]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite.

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 2] for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values change, see the description of the parameter in this chapter. To view the parameter value, run the **`validate_bd_design`** command in the Tcl Console.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the tool bar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 4].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). The layout depicted here might vary from the current version.

Output Generation

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5].

IMPORTANT: *For cores targeting Xilinx 7 series FPGAs or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 3].

Example Design

This chapter contains information about the ERNIC core example design. The example design consists of the following modules:

- Clock and Reset Generator (simulation only)
- Register Configuration Module
- Send, Read Response, and ACK Generator
- WQE generator
- TX Path Checker
- RX Path Checker

Figure 5-1 shows the top-level example design architecture.

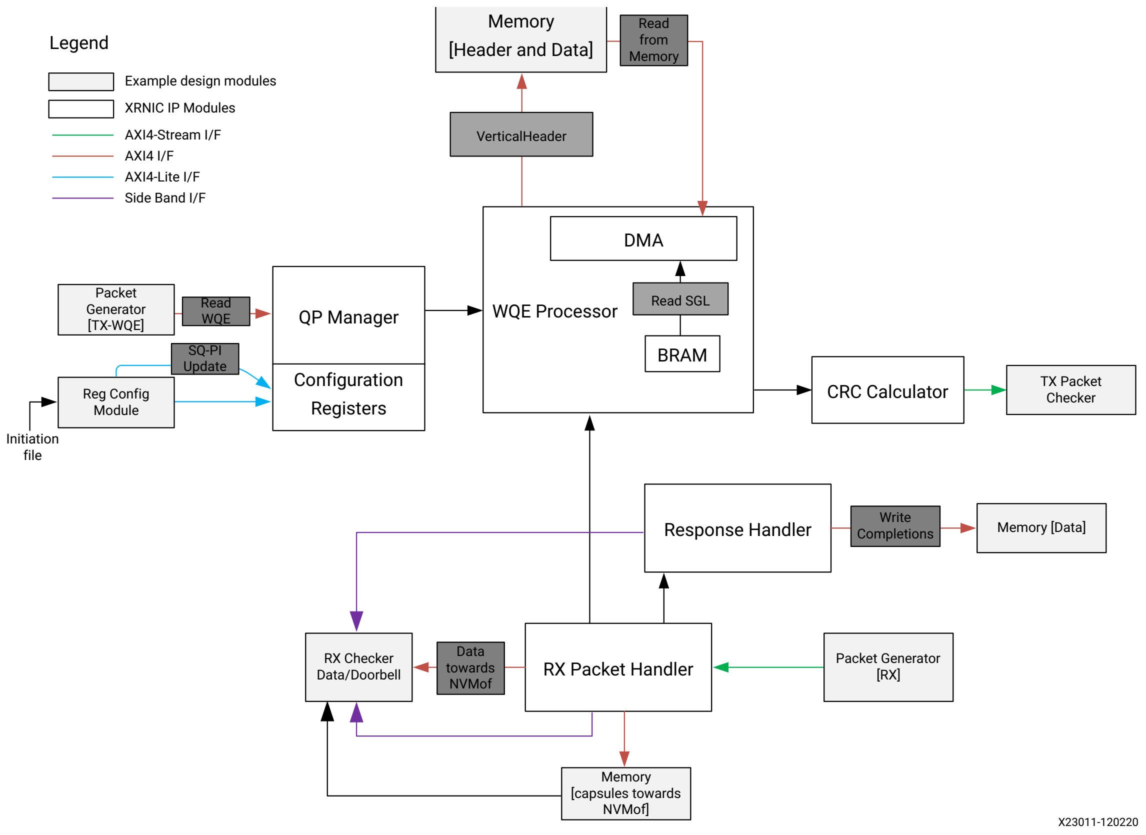

Figure 5-1: **Example Design Architecture**

Apart from the ERNIC IP, the example design integrates the following modules:

- **Register Configuration Module**: This module configures all the required registers of the ERNIC IP.
- **Packet Generator Module**: This generates the following types of packets:
    - SEND Packets
    - RDMA Read Response Packets for all the Read Requests from the ERNIC IP
    - ACK Packets for all the RDMA Write requests from the ERNIC IP
    - RDMA READ and RDMA WRITE
- **Example Design Features**: RDMA READ and RDMA WRITE
- **RX Checker Module**: This module checks the capsules (from SEND packets) and payload (from Read Response) received from the ERNIC IP. This module also checks the number of door bells rang on the RQ side band interface. The payload size of Read Response packets, RDMA write requests is 256 bytes. Capsule size in the SEND packets is 80 bytes.
- **WQE Generator Module**: This is responsible for generating the work queue requests and SQ PI doorbell updates to the ERNIC IP.
- **TX Checker Module**: This checks the data transmitted by the ERNIC IP over the streaming interface.
- **Initiator WQE Module**: This fetches the data from DDR when RDMA READ occurs.
- **Initiator Checker Modules**:
    - Data checker which checks the data received for RDMA WRITE operation.
    - RDMA READ RESPONSE packet check for RDMA READ operation and ACK packet check for RDMA WRITE operation.

Example Design Features

The example design exercises the features of the ERNIC IP. The following incoming packets are sent to the ERNIC IP:

- RDMA SEND
- RDMA Read Response
- ACK Packets
- RDMA READ
- RDMA WRITE

The following outgoing packets are checked by the example design:

- RDMA Read Requests
- RDMA Write Requests
- RDMA READ Response
- ACK Packets

The following are the hardware handshake paths of ERNIC:

- Only IPV4 Packets are exercised
- Only six QPs are exercised (QP2 to QP7)

Example Design Limitations

The ERNIC example design has the following limitations:

- Example design does not exercise Retry path
- Example design exercises only Hardware handshake path (QPCONFi[4] is set to 0 for all QPs)
- Example design does not exercise IPV6 packets
- Example design does not generate any connection management (MAD) packets to QP1
- Example design limits the number of each supported packet type transactions to 8

Simulating the Example Design

For more information on Simulation, see the Vivado Design Suite User Guide: Logic Simulation (UG900) [Ref 5].

Simulation Results

The simulation script compiles the ERNIC example design and supporting simulation files. It then runs the simulation and checks to ensure that it completed successfully. If the test passes, then the following message is displayed:

```
Test Completed Successfully
```

If the test fails, then the following message is displayed:

```
ERROR: Test Failed
```

If the test hangs, then the following message is displayed:

```
ERROR: Test did not complete (timed-out)
```

Example Sequence

The demonstration test bench performs the following tasks:

- All the required ERNIC registers are configured by the Register Configuration Module through AXI4-Lite interface.
- There are three operations being handled in example test bench:

  a. RDMA SEND:

     - Packet Generator module generates eight SEND packets to the QP2 to QP7.
     - RX Path Checker of example design, checks for the data integrity on the capsule transferred from the ERNIC along with the door bells rang.

  b. RDMA RD REQUEST:

     - Example design has a predefined RDMA Read work Queue entry packets. The WQE Generator module rings the SQPI doorbell and sends the work queue entry packets when requested by the ERNIC.
     - The TX Checker Module checks the necessary fields of the RDMA read request received.
     - After successful validation of the request, the Packet Generator module sends RDMA Read Response to the ERNIC IP.
     - The RX Path Checker checks for the data integrity on the payload transferred by the ERNIC along with the door bells rang.

  c. RDMA WRITE REQUEST:

     - Example design has a predefined RDMA Write work Queue entry Packets. The WQE Generator module rings the SQPI doorbell and sends the work queue entry packets when requested by the ERNIC.
     - The TX Checker Module checks the necessary fields of the RDMA write request received.
     - Upon successful validation of the request, the Packet Generator module sends ACK packet to the ERNIC IP.

  d. RDMA WRITE:

     - For Initiator functionality, RDMA write packets are sent to the ERNIC module which writes payload data to the RX path and the acknowledgment (ACK packet) is received on the TX path.

- The ACK packet and data is checked through the checker modules.

e. RDMA READ:

- For initiator Functionality, RDMA Read packets are sent to ERNIC module which reads the data from DDR and sends the read response on the TX path.
- The Read response packets are checked through checker module.

Chapter 6

ERNIC Software Flow

The ERNIC IP should be initialized and configured before it can be used for sending and receiving RDMA traffic. This involves performing the following steps in sequence:

1. ERNIC IP global configuration.
2. Memory region registration (for RDMA client application which share the memory region to remote hosts).
3. QP1 creation for sending and receiving RDMA connection management (CM) packets.
4. RC QP creation.
5. CQ creation.
6. Posting WQEs for RDMA operations.
7. Processing completions.
8. QP deletion.
9. QP fatal recovery.

ERNIC Global Configuration

The following registers should be configured for ERNIC global configuration.

- Configuration of error buffers queue. Allocate memory and configure the base address to ERRBUFBA and queue depth and size to ERRBUFSZ registers. The register ERRBUFWPTR is used by the hardware to indicate the SW about the new entries in the ERROR buffer queue.
- Configure incoming packet error status queue. This queue gives the status of incoming packets with errors. To configure this queue, allocate memory and write the base address to IPKTERRQBA, queue depth and size to IPKTERRQSZ registers. IPKTERRQWPTR register is used by the hardware to indicate the SW about the new entries in the queue.
- Configure response error packet buffer. This involves writing memory base address to RESP_ERR_PKT_BUF_BA, queue depth and size to RESP_ERR_BUF_SZ and a DDR address to RESP_ERR_BUF_WRPTR for the hardware to indicate the SW about the response error buffer writes happened.

- Enable interrupts by writing to INTEN register.
- Allocate memory for CQ and RQ doorbells for all the QPs to be created. The doorbell memory for individual CQ and RQ are taken from offsets of this memory and configured in the corresponding ERNIC registers during the QP creation (see the QP1 Creation).
- Configure source MAC address in the MACXADDLSB, MACXADDMSB registers.
- Configures source IP address. If the IP version is IPv6, then registers IPv6XADD1-4 are written with interface's IPv6 address, otherwise IPv4 address is written to IPv4XADD register.
- Configures number of QPs supported by the ERNIC design, UDP port and enables the ERNIC by writing to XRNICCONF register.

QP1 Creation

QP1 is a special QP in the ERNIC IP. This QP is designated for exchanging the MAD packets with remote hosts for establishing the connection for RC QPs. QP1 must be configured first before creation of any other RC QP. See the *InfiniBand Architecture Specification* Volume 1, Annex A16 RoCE and Annex 17 RoCE V2 [Ref 1] for more information on QP1. The following steps are required to configure and enable QP1:

1. Allocate memory for queues (RQ, SQ, CQ), and program the respective base address registers.
   - RQ base address is written to RQBAi register. SQ base address is written to SQBAi register and queue depths to QDEPTHi.
   - CQ base address is written to CQBAi register.
   - Configures each RQ buffer size by writing to QPCONFi.
2. Pick up memory for both SQ completion doorbell and RQ write pointers from the pre-allocated doorbell memory pool as part of ERNIC initialization (see ERNIC Global Configuration)
   - SQ completion doorbell address to CQDBADDi register.
   - RQ write pointer doorbell address to RQWPTRDBADDi register.

Memory Registration

Memory must be registered with ERNIC hardware before it is exchanged with remote hosts for doing the RDMA operations (incoming RDMA READ and incoming RDMA WRITE). The following steps are important to register a memory with ERNIC:

1. Allocate physical memory and create a virtual address mapping to it.
2. Create a protection domain number and write to PDPDNUM register at a free slot in the PD table.
3. Write the virtual address of the memory region in VIRTADDRLSB and VIRTADDRMSB registers at the same slot in PD table.
4. Write the physical address of the memory region to BUFBASEADDRLSB and BUFBASEADDRMSB registers.
5. Create an R_KEY and configure it to BUFFERKEY register.
6. Write memory region length to WRRDBUFLEN register and access permission (REMOTE READ or WRITE) to ACCESSDESC registers.

RC QP Creation

Once the QP1 is enabled, RC QPs can be created by exchanging the CM MAD packets on QP1. See the “Communication Management” chapter in the *InfiniBand Architecture Specification* Volume 1, Annex A16 RoCE and Annex 17 RoCE V2 [Ref 1] for details on CM MAD packets. For creating any RC QP, the following steps need to be performed:

1. Allocate memory for queues (RQ, SQ, CQ), and program the respective base address registers.
    - RQ base address is written to RQBAi register, SQ base address is written to SQBAi register and queue depths to QDEPTHi.
    - CQ base address is written to CQBAi register.
    - Configure the each RQ buffer size by writing to QPCONFi.
2. Pick up memory for both SQ completion doorbell and RQ write pointers from the pre-allocated doorbell memory pool as part of ERNIC initialization.
    - SQ completion doorbell address to CQDBADDi register.
    - RQ write pointer doorbell address to RQWPTRDBADDi register.
3. When the QP state transitions from RESET to RTS during the connection establishment, configure the following information:
    a. Write remote host MAC address to MACDESADDLSBi and MACDESADDMSBi registers.
    b. Configure DESTQPCONFi with destination QP ID.
    c. Configure IP address to IPDESADDR1i - IPDESADDR4i if IPv6 is being used otherwise, write only IPDESADDR1i with IPv4 address. Configure the IP address version in the QPCONFi.

d. Configure send queue PSN by writing to SQPSNi.

e. Configure last receive queue PSN by writing to LSTRQREQi

f. Configure PMTU by writing to QPCONFi register.

g. Configure the following by writing to TIMEOUTCONFi register.

  - Configure RNR retry count and RNR retry timeout
  - Configure retry timeout and retry timeout

h. PD number that the QP is associated with is configured by writing to Protection Domain number register.

WQE SQ Posting

- The QP can now be used for posting work requests for RDMA operations. To do that prepare the WQE in the format described in Table 2-1 and copy it to SQ memory.
- Ring the doorbell by incrementing SQPIi by 1 and writing the value to the SQPIi register.

Receiving Incoming RDMA SEND Messages

Data received in the incoming SEND messages is copied to RQ buffers by ERNIC IP. The software can retrieve this data by doing the following steps:

1. Read the RQWPTRDBADDi value. If there is a change from previous value, then process the received data by reading the data from RQBAi+offset. This offset is calculated based on RQ buffer size and the value in RQWPTRDBADDi.
2. After processing the received messages, increment RQCIi register value to indicate to the hardware that buffer is consumed so it can be used for further incoming messages.

Processing WQE Completions

ERNIC IP indicates the completion of WQE posted to it, by incrementing the CQHEADi register. Software should poll this register to check for any completions for the previously posted WQEs.

Enabling QP HW Offload

The RC QPs can be enabled for HW handshake mode, where the external hardware application can send and receive RDMA messages on this QP. The offloads the software posting of RDMA messages to external hardware application. To enable the HW handshake mode on a QP, the following steps need to be performed:

1. Enable software override by writing to XRNIC_ADV_CONF.
2. Reset the QP pointers by writing 0 to SQPIi, CQHEADi, and STATCURSQPTRi.
3. Configure remaining RQs to be processed by getting the difference between RQCIi and RQWPTRDBADDi and write the value to STATRQPIDBi at lower 2B.
4. Enable HW offload by writing to QPCONFi.
5. Disable software override by writing to XRNIC_ADV_CONF.

QP Deletion

An RC can be deleted when it is no longer in use (either because of locally initiated disconnect or from the remote peer). The following steps needs to be performed for deleting the QP and removing its configuration:

1. Wait for SQ and outstanding queues to become empty. The status bits are in STATQPi register.
2. Wait for all completions received for WQEs in SQ. This is done by checking SQPIi and CQHEADi registers.
3. Enable the software override by writing to XRNIC_ADV_CONF register and disable the QP by writing to QPCONFi register.
4. Reset the QP pointers by writing 0 to RQWPTRDBADDi, SQPIi, CQHEADi, RQCIi, STATRQPIDBi, STATCURSQPTRi, SQPSNi, LSTRQREQi, STATMSN, and then disable the QP and kept it in recovery mode by writing QPCONFi.
5. After resetting pointers, software override should be disabled by writing to XRNIC_ADV_CONF.
6. After this, software should free memory allocated for SQ, RQ, and CQs.

QP Fatal Recovery

The following steps describe the recovery process for a QP that entered the fatal condition. It describes how to clear the existing traffic on the QP and re-initialize it so that it can be reused.

Steps to clear traffic on QP:

1. On detecting QP under the FATAL interrupt, read the IPKTERRQBA register to know the Incoming Pkt Error Status Queue base address written by the driver.
2. Read this base address to know the QP FATAL status and decide which QP went into FATAL (Bits[31:16]) and check QP FATAL status code (Bits[15:0]), see Table 2-4.
3. Stop pushing any further SQ PI doorbells.
4. Set the “QP under recovery” bit in the QPCONFi register to 1.
5. Read the STATQPi register to check “send Q empty” and “outstanding Q empty” bits to become 1.
6. Poll the CQHEADi register to check its value is the same as SQPIi register.
7. Poll the RESPHNDSTS register for “sq pici db check en” —16th bit to be set.
8. Set the QPCONFi register “QP enable” bit to 0 and “QP under recovery” bit to 1.

Steps to reinitialize the QP:

1. Poll the CQHEADi register to check its value is the same as SQPIi register.
2. Poll the RESPHNDSTS register for “sq pici db check en”—16th bit to be set.
3. Set the “SW OVERWRIDE” bit in the XRNICADCONF register to 1.
4. Initialize the following QP registers to 0:
    - STATRQPIDBi
    - STATRQBUFCAi
    - STATRQBUFCAMSBi
    - RQCIi
    - STATCURSQPTRi
    - SQPIi
    - SQPSNi
    - LSTRQREQi
    - STATMSN

- CQHEADi

5. Poll the CQHEADi register to check its value is 0.
6. Initialize the following register with the new value:
    - SQPSNi
    - LSTRQREQi
7. Initialize the following Ethernet side registers:
    - MACDESADDMSBi
    - MACDESADDLSBi
    - IPDESADDR1i
    - IPDESADDR2i
    - IPDESADDR3i
    - IPDESADDR4i
8. Re-configure the IP version in the QPCONFi.
9. Re-initialize the "RNR nack count" and "retry count" in the STATQPi register.
10. Set the ACCESSDESC register "access type" for that QP to `'b10`.
11. Re-program the QPCONFi register by re-initializing fields like "RQ interrupt enable", "CQ interrupt enable", "PMTU", and "HW handshake disable". Selectively enable "CQE write enable" to debug error completions and re-initialize "RQ Buffer size".
12. Re-program the QPADVCONFi register by randomizing "Traffic class", "Time to live", and "Partition key".
13. Set "QP EN" to 1 and "QP under recovery" to 0 in QPCONFi register.
14. Clear the "'SW OVERWRIDE" bit in the XRNICADCONF register to 0.

Requesting ERNIC Support Questionnaire

The following is a questionnaire to be filled before requesting ERNIC support.

Table A-1: **ERNIC Support Questionnaire**

Question	Answer
ERNIC IP Configuration	
ERNIC and Vivado release version	
Hardware GUI parameter values	
Hardware handshake mode enabled/disabled	
ERNIC System Level Connectivity	
ERNIC ↔ ERNIC	
ERNIC ↔ Switch ↔ ERNIC	
ERNIC ↔ Mellanox	
ERNIC ↔ Switch ↔ Mellanox	
ERNIC ↔ BFM (Simulation)	
Traffic Pattern	
Opcode – RDMA_READ/RDMA_WRITE/RDMA_SEND/ IMMEDIATE/INVALIDATE/Mix of packets with different Opcodes	
Transfer size (DMA length and PMTU)	
Traffic Direction	
ERNIC sending packets (TX)	
ERNIC receiving packets (RX)	
Both	
Issue Type	
Hang ERNIC is not sending ACKs ERNIC is not processing the WQEs ERNIC is not processing the incoming requests/ responses	
Performance drop	
Packet drop	
Excessive retries	

Table A-1: **ERNIC Support Questionnaire *(Cont'd)***

Question	Answer
Unexpected fatal – Check if a Fatal response is sent by the Host	
Fatal recovery	
FCS error – Check the behavior of packet FIFO between ERNIC → CMAC	
Other	
Congestion	
PFC is enabled in ERNIC and CMAC?	
What are the PFC counter values from CMAC?	
Software Related Questions	
Issue with reference design delivered by Xilinx?	
Connection establishment is done via MAD packets?	
Other	
System block diagram	
Details of any custom blocks connected to ERNIC	
Register configuration details Configuration register dump Debug register dump	
Simulation/Wireshark dump	

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. Before reaching out to Xilinx Support, fill out these basic questionnaire provided in Appendix A.

TIP: *If the IP generation halts with an error, there might be a license issue. See License Checkers in Chapter 1 for more details.*

Debugging the IP/System

The ERNIC IP has a number of status registers implemented to allow for extensive debug in case of a failure. These registers along with counter information from the initiator RNIC, for example Mellanox, can provide vital clues to debug any failure. The debug tools available to the system debugger working with ERNIC IP are:

- ERNIC complete register dump
- Promiscuous or packet sniffer mode counters (from the initiator RNIC),
- HW counters (from the initiator RNIC).

The steps to dump the different logs are explained in the following sections.

Complete Dump of the ERNIC Registers

If you are using smart cable, perform these steps:

1. Change the connect command directive with appropriate IP address of the Smart cable using the following command:

   ```
   $ connect -host <smartlink IP address>
   ```

2. Change the target using the following command:

   ```
    target -set -nocase -filter {name =~ "<processor name>"}
   ```

3. Read the ERNIC IP register values using the following command:

   ```
   $ mrd -force <ERNIC IP offset> <number of locations to read>
   ```

Enable promisc Mode on the Initiator RNIC

1. Enable promisc mode on the initiator RNIC using the command:

   ```
   $ ifconfig <interface name> promisc
   $ netstat -i
   ```

2. At the end of the test, dump all the counter information from the initiator RNIC using the command:

   ```
   $ ethtool -S <RNIC Card name> > ethtool.log
   ```

HW Debug Counters for Mellanox RNIC

To access and dump hardware debug counters for Mellanox RNIC, use the command:

```
$ cd /sys/class/infiniband/mlx5_0/ports/1/hw_counters/
$ for file in `ls .`; do echo -n "${file}:"; cat $file; done
```

Note: For more information on hardware debug counters for Mellanox, see DOC-2572 on the Mellanox community.

Some quick debug checks that you can do to ensure that the system is clean are listed here.

- Check the ethtool.log file for any link failures or CRC failures. Any non-zero value in these two counters points towards an unstable link and can be the cause of failure. Two snippets from the `ethtool.log` file are listed here.

  Sample 1:

  ```
  rx_wqe_err: 0
  rx_mpwqe_filler: 0
  rx_mpwqe_frag: 0
  rx_buff_alloc_err: 0
  rx_cqe_compress_blks: 0
  rx_cqe_compress_pkts: 0
  link_down_events_phy: 4
  rx_out_of_buffer: 0
  rx_vport_unicast_packets: 5
  rx_vport_unicast_bytes: 422
  tx_vport_unicast_packets: 10
  tx_vport_unicast_bytes: 714
  rx_vport_multicast_packets: 46
  rx_vport_multicast_bytes: 7608
  ```

  Sample 2:

  ```
  rx_vport_rdma_multicast_bytes: 0
  tx_vport_rdma_multicast_packets: 0
  tx_vport_rdma_multicast_bytes: 0
  tx_packets_phy: 320587336
  rx_packets_phy: 324924215
  rx_crc_errors_phy: 0
  tx_bytes_phy: 27657272230
  rx_bytes_phy: 467673988652
  tx_multicast_phy: 59
  ```

```
tx_broadcast_phy: 32
rx_multicast_phy: 0
rx_broadcast_phy: 9
rx_in_range_len_errors_phy: 0
```

- Check the **`hw_counters`** on the initiator side. These counters give a picture of all fatal/ non-fatal errors seen by the initiator. A sample of the counters:

```
duplicate_request:0
implied_nak_seq_err:0
lifespan:10
local_ack_timeout_err:0
out_of_buffer:0
out_of_sequence:0
packet_seq_err:0
rnr_nak_retry_err:0
rx_atomic_requests:0
rx_read_requests:5
rx_write_requests:108307999
```

- Check the following register locations from the ERNIC register dump. The QP Status (STATQPi) registers for all enabled QPs provide a status of the different QPs. Check if the QP FATAL status is set to 1 in any of the QP status registers. For example, the QP Status register for QP 5:

```
0x84020688:   30620601    ? QP Fatal is set to 1
```

- If the QP is in FATAL state, no transactions are performed from this QP and the QP gets disconnected. Bits[22:16] in the same register provide the last AETH syndrome received from the initiator. In many cases the QP might go into FATAL state due to a NAK syndrome received from the initiator. The NAK syndrome helps you understand the failure being seen by the initiator RNIC card. In the above example, the AETH syndrome of **`0x62`** indicates a "Remote Access Error" from the initiator. The decoding of the AETH syndrome is provided in the Infiniband Architecture Specification Volume 1 (Release 1.2.1). For NAK code details in this specification, see Table 43: AETH Syndrome field and Table 44: NAK Codes.

- Check the Incoming and outgoing NAK count registers ((INNACKPKTCNT) and (OUTNACKPKTCNT)) at offset **`0x134`** and **`0x138`** for the number of incoming NAK syndromes seen and number of NAK syndromes sent out. This number should normally correlate with the number seen from the **`hw_counters`** seen at the initiator. In general not all NAK codes are fatal. However, all NAK codes lead to retries and can lower the overall performance of the system. A high number of NAK codes can be a cause of concern.

- The total number of retries initiated by the target can be known from the Retry count status register (RETRYCNTSTS) at offset **`0x140`**. Normally this number will match with the incoming NAK count. In case this number is more than the incoming NAK count value, it might be due to timeouts. Timeouts happen when the responder (in this case, the initiator RNIC) does not respond to a request in a given time. The timeout value is configured in the Timeout Configuration register (TIMEOUTCONF). This timeout interval is implemented as per the InfiniBand™ Architecture Specification Volume 1

(Release 1.2.1) clause C9-141. It might be worthwhile to try and increase the timeout interval and check if the number of retries is reduced.

- ERNIC register offset **`0x6C`** (ERRBUFWPTR) indicates Error buffer write pointer. This register gives the number of error packets received. Each error packet will be stored in the address location provided in Error buffer base address (ERRBUFBA) register (offset **`0x60`**). Each entry in this buffer will be given with error syndrome. See ERNIC RX Path for details. The rows highlighted in yellow enlist the conditions that will cause the QP to go into FATAL state.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

For a glossary of technical terms used in Xilinx documentation, see the Xilinx Glossary.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter **`docnav`**.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this product guide:

1. *InfiniBand Architecture Specification* Volume 1, Annex A16 RoCE and Annex 17 RoCE V2
2. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
3. *Vivado Design Suite User Guide: Designing with IP* (UG896)
4. *Vivado Design Suite User Guide: Getting Started* (UG910)
5. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
6. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
7. *Vivado Design Suite User Guide: Implementation* (UG904)
8. *Vivado Design Suite User Guide:* AXI Reference Guide (UG1037)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/30/2021	3.1	Updated size to 256 in Table 3-1.
06/16/2021	3.1	• Added Versal ACAP. • Added timing violation note in Features. • Updated Figure 1-1. • Added non-RDMA packet description in RX PKT Handler. • Added retransmission description in Response Handler. • Updated maximum DMA length and added Read response descriptions in Unsupported Features. • Updated Table 2-4. • Added description for rx_pkt_hndler_ddr_m_axi* in Table 2-5. • Updated description for 0x20088, 0x2013C, 0x20200 + ((i-1) x 0x0100), and 0x202B0 + ((i-1) x 0x0100) in Table 2-7. • Updated size in Table 3-1. • Added QP Fatal Recovery. • Added Appendix A, Requesting ERNIC Support Questionnaire.
02/26/2021	3.0	Updated Resource Utilization link.

Date	Version	Revision
01/21/2021	3.0	• Updated Priority flow control in Features. • Added Navigating Content by Design Process section. • Added Flow Control Manager in Feature Summary. • Updated Figure 1-1. • Added RDMA READ/WRITE request description in ERNIC RX Path. • Updated Figure 2-6. • Updated CMAC RX/TX descriptions and RoCE descriptions in Table 2-5. • Updated Figure 2-9. • Updated Table 2-7. • Updated Figure 3-1. • Updated address for Rcv Q Buffer base address QPi (RQBAMSBi). • Updated descriptions #2 and #3 in Complete Dump of the ERNIC Registers.
06/10/2020	2.0	Added new chapter Chapter 6, ERNIC Software Flow
12/09/2019	1.0	Updated Table 2-5 and Table 3-2.
07/10/2019	1.0	Minor updates
12/05/2018	1.0	Initial Xilinx release.

Please Read: Important Legal Notices