RDMA Smart NIC (*RSNIC IP*) v1.0

Table of Contents

Chapter 1. Introduction

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

Features

- Support for RoCE v2
- Support RDMA Call Library
- 100Gb/s line rate
- Support for reliable Connection (RC) RDMA transport service type
- QP1 support for sending and receiving MAD packets
- Connection Management in Host Server CPU
- No Firmware Needed
- Mimic the Mellanox Infiniband, RoCE v2, and RDMA
- Hardware handshake mode on user interface to support hardware RDMA applications in the user logic
- Mellanox Stores the packet in the Host Server
- DMA Uses Virtual Addresses
- Support incoming and outgoing RDMA
    - RDMA SEND
    - RDMA WRITE
    - RDMA READ
    - RDMA_WRITE_FIRST
    - RDMA_WRITE_MIDDLE
    - RDMA_WRITE_LAST
    - RDMA_WRITE_LAST_WITH_IMD
    - RDMA_WRITE_ONLY
    - RDMA_WRITE_ONLY_WIT_IMD
    - RDMA_READ_REQUEST
    - RDMA_READ_RESP_FIRST
    - RDMA_READ_RESP_MIDDLE
    - RDMA_READ_RESP_LAST
    - RDMA_READ_RESP_ONLY
    - RDMA_ACK
    - RDMA_PART_ONLY
    - RDMA_PART_FIRST
    - RDMA_PART_MIDDLE
    - RDMA_PART_LAST
    - RDMA_READ_POINTER_REQUEST
    - RDMA_READ_CONSISTENT_REQUEST
- Designed to scale up to 255 RDMA Queue pairs

- Support for IPv4 and IPv6 packets
- Support for explicit Congestion Notification (ECN)
- Support for memory registration and protection domains
- 500 QPs in hardware cache
- Retransmission buffer on host DDR
- Congestion control on QPs level
- Follow the following modules:
    - QP Manager
    - WQE handler
    - CQE generator
    - IBH Processor
    - IP Handler

Chapter 2. Overview

This chapter provides an overview of the *RSNIC IP* core and details of the applications, and standards conformance. *RSNIC IP* is an IP implementation RDMA over a Converged Ethernet (RoCE v2) protocol for embedded target or initiator devices.

Figure 2-1: Shows the *RSNIC IP* and its connections to other IPs in the subsystem

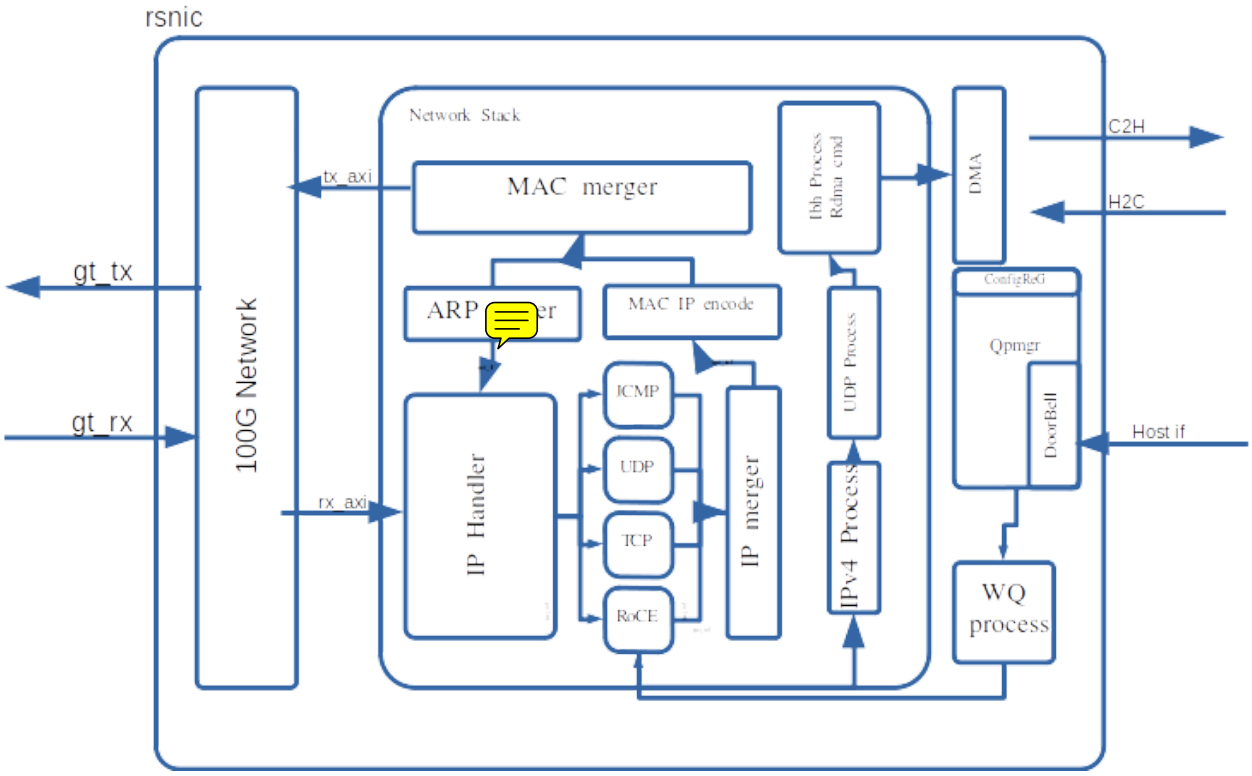


Figure 2-1: **RSNIC IP Block Diagram**

The *RSNIC IP* is composed of the 5 major modules, The QP manager, WQE handler, CQE generator, IBH processor and IP handler.

The *RSNIC 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 *RSNIC IP* implements the following subset of RoCE v2 functionalities:

- RDMA SEND
- RDMA WRITE
- RDMA READ
- RDMA_WRITE_FIRST
- RDMA_WRITE_MIDDLE
- RDMA_WRITE_LAST
- RDMA_WRITE_LAST_WITH_IMD
- RDMA_WRITE_ONLY
- RDMA_WRITE_ONLY_WIT_IMD
- RDMA_READ_REQUEST
- RDMA_READ_RESP_FIRST
- RDMA_READ_RESP_MIDDLE
- RDMA_READ_RESP_LAST
- RDMA_READ_RESP_ONLY
- RDMA_ACK
- RDMA_PART_ONLY
- RDMA_PART_FIRST
- RDMA_PART_MIDDLE
- RDMA_PART_LAST
- RDMA_READ_POINTER_REQUEST
- RDMA_READ_CONSISTENT_REQUEST for incoming and outgoing packets
- Support for up to 254 connections.
- Scalable design of up to 255 RDMA Queue pairs.
- Supports dynamic memory registration.
- Hardware handshake mechanism for efficient doorbell exchange with the user application logic.

RSNIC IP Modules

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

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

QP Manager

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

WQE Handler

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

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

CQE Generator

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

IBH Processor

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

IP Handle

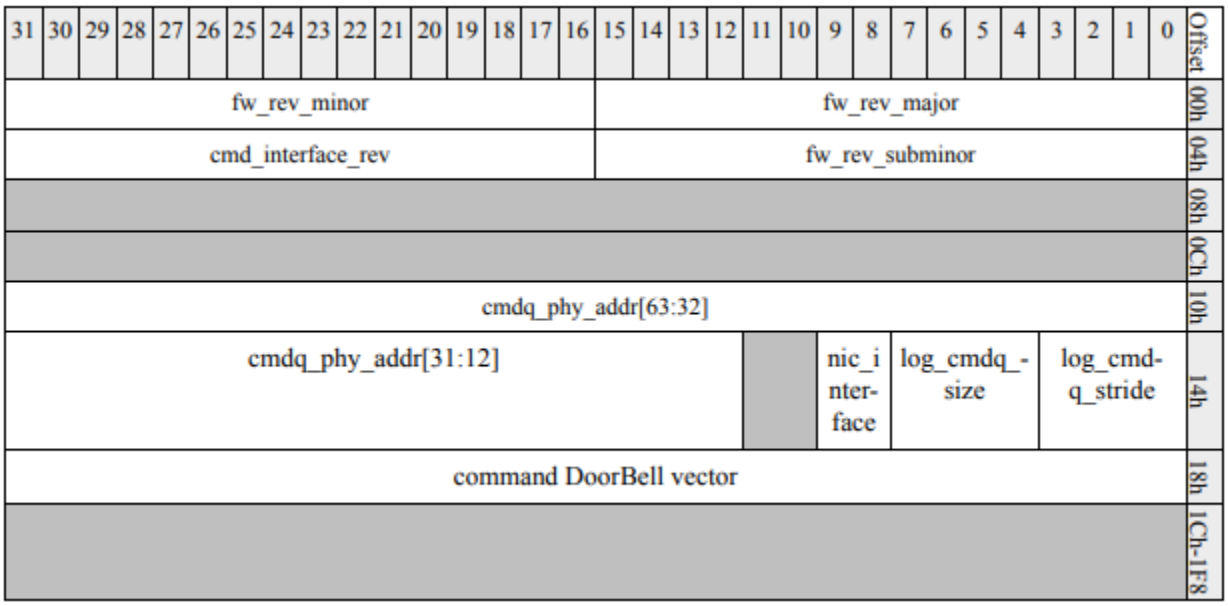
The IP Handler module receives the incoming RDMA packets and filters out the non-RMDA packets.
The *RSNIC IP* handles the following types of incoming RoCE v2 packets with the transport type Reliable Connection “RC”:

RDMA SEND
RDMA WRITE
RDMA READ
RDMA_WRITE_FIRST
RDMA_WRITE_MIDDLE
RDMA_WRITE_LAST
RDMA_WRITE_LAST_WITH_IMD

RDMA_WRITE_ONLY
RDMA_WRITE_ONLY_WIT_IMD
RDMA_READ_REQUEST
RDMA_READ_RESP_FIRST
RDMA_READ_RESP_MIDDLE
RDMA_READ_RESP_LAST
RDMA_READ_RESP_ONLY
RDMA_ACK
RDMA_PART_ONLY
RDMA_PART_FIRST
RDMA_PART_MIDDLE
RDMA_PART_LAST
RDMA_READ_POINTER_REQUEST
RDMA_READ_CONSISTENT_REQUEST

Initialization Segment

The initialization segment is located at offset 0 of the BAR



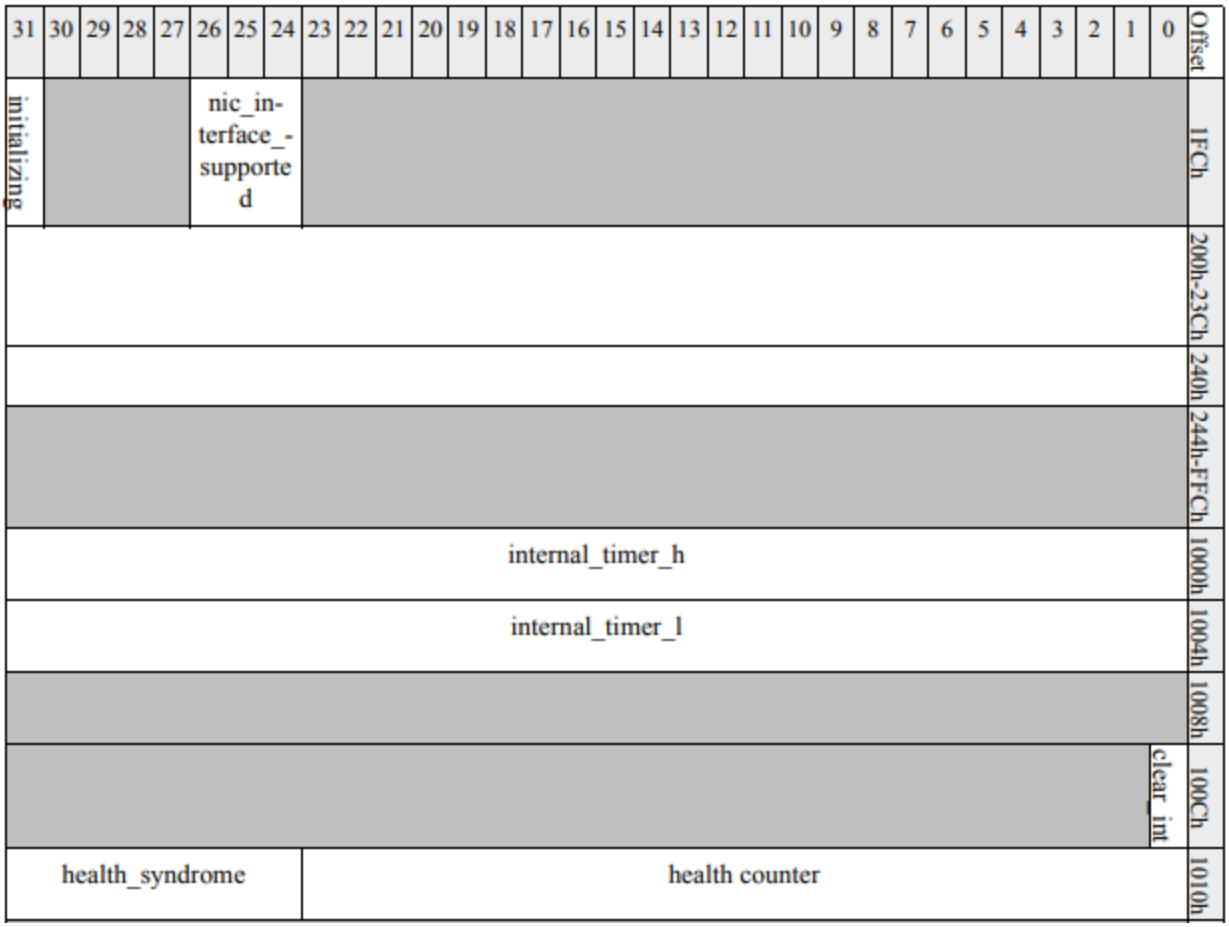


Table 2-1: Initialization Segment

Offset	Bits	Name	Description	Access
0000h	31:16	fw_rev_minor	Firmware Revision - Minor	RO
	15:0	fw_rev_major	Firmware Revision - Major	RO

Offset	Bits	Name	Description	Access
0004h	31:16	cmd_interface_rev	Command Interface Interpreter Revision ID This number is bumped up every time a non-backward-compatible change is done for the command interface.	RO
	15:0	fw_rev_subminor	Firmware Sub-minor version (Patch level)	RO
0010h	31:0	cmdq_-phy_addr[63:32]	Physical address of the command queue record. This field should not be modified after INIT_HCA until TEARDOWN_HCA.	RW
0014h	31:12	cmdq_-phy_addr[31:12]	Physical address of the command queue record. This field should not be modified after INIT_HCA until TEARDOWN_HCA.	RW
	9:8	nic_interface	NIC interface mode 0x0: full_driver 0x1: disabled	WO
	7:4	log_cmdq_size	Log number of cmdqs available	RO
	3:0	log_cmdq_stride	Stride between start of each cmdq	RO
0018h	31:0	command DoorBell vector	Bit per command in the cmdq. When the bit is set, when writing this vector to the device, the command is moved to HW ownership (HW need to execute the command). bit 0 is related to the command in offset 0 in the command queue etc. The valid bits in this vector are [number of commands-1..0] When driver is in No DRAM NIC mode, this field must not be written.	WO
01FC	31	initializing	1 - device still in initializing state. 0 - device is ready to receive commands.	RO
	26:24	nic_interface_supported	Bitmask indicating which ***nic_interface*** modes are supported 0: full_driver 1: disabled	RO
1000h	31:0	internal_timer_h	MSBs of the current internal timer value.	RO
1004h	31:0	internal_timer_l	LSBs of the current internal timer value.	RO
100Ch	0	clear_int	Writing 1 to this register will clear the interrupt (will always use intA)	WO

Offset	Bits	Name	Description	Access
1010h	31:24	health_syndrome	Syndrome 0x1: FW_INTERNAL_ERR - assert triggered in FW code 0x7: DEAD_IRISC - Irisc not responding 0x8: HW_FATAL_ERR 0x9: FW_CRC_ERR 0xA: ICM_FETCH_PCI_ERR 0xB: ICM_PAGE_ERR 0xC: ASYNCHRONOUS_EQ_BUF_OVERRUN 0xD: EQ_IN_ERR 0xE: EQ_INV 0xF: FFSER_ERR 0x10: HIGH_TEMP_ERR	RO

Table 2-2: Initialization Segment Field Description

Table 2-3: Configuration Register

Table 2-3a: **Context**

Bitwidth	Content	Size (b)	Description
[2:0]	QpState	3	Queue Pair State
[26:3]	LocQpn	24	Local Queue Pair Number
[50:27]	RemPsn	24	Remote PSN
[74:51]	LocPsn	24	Local PSN
[90:75]	RemRkey	48	Remote RKey
[122:91]	RemValAddrL	16	Remote Valid Address Low
[138:123]	RemValAddrH	16	Remote Valid Address High

Table 2-3b: **Connection**

Bitwidth	Content	Size (b)	Description
[15:0]	LocQpn	16	Local QPN
[39:16]	RemQpn	24	Remote QPN
[167:40]	RemIpAddr	128	Remote IP Address
[183:168]	RemUdpPrt	16	Remote UDP Port

Table 2-4 Shows the structure of Send work requests. Each Work Queue Entry (WQE) is 64 bytes in size.

Table 2-4: WQE Structure

Bitwidth	Content	Size (b)	Description
[2:0]	OpCode	3	Operation Code
[6:3]	LocAddr	4	Local Address
[74:27]	OrgAddr	48	Original Address
[122:75]	TgtAddr	48	Target Address
[154:123]	Size	32	Size

Register Space

All the *RSNIC IP* 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.

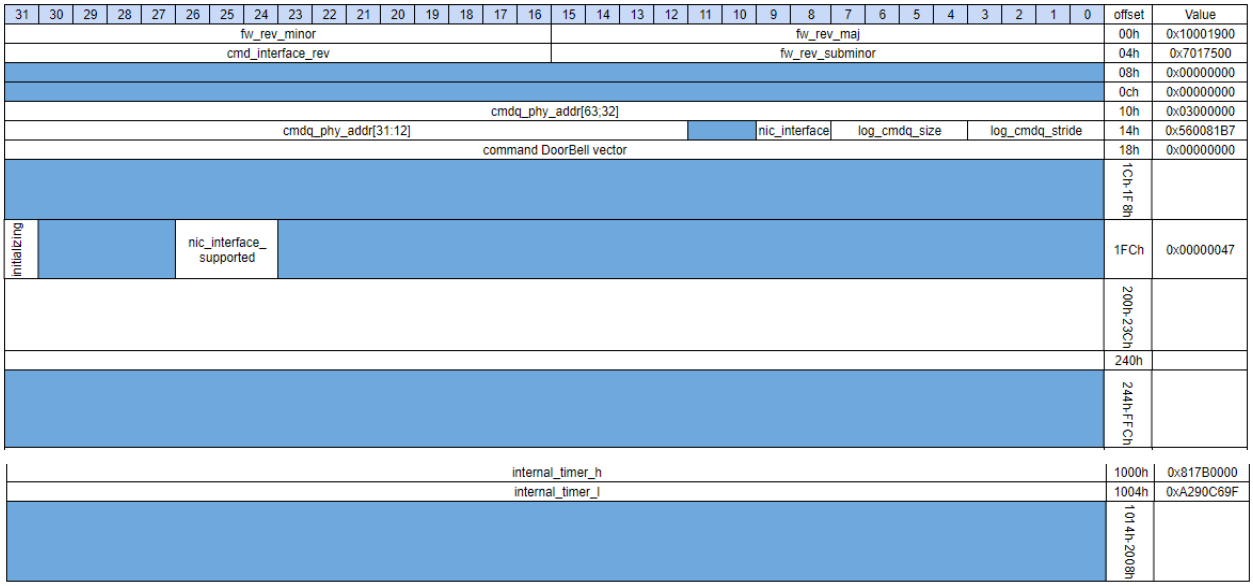


Figure 2-2 PCIe Initialization Values

Offset	Bits	Name	Description	Access
0000h	31:16	fw-rev_minor	Firmware Revision - Minor	RO
	15:00	fw_rev_major	Firmware Revision - Major	RO
0004h	31:16	cmd_interface_rev	Command Interface Interpreter Revision ID This number is bumped up every time a non-backward-compatible change is done for the command interface.	RO
	15:00	fw_rev_subminor	Firmware Sub-minor version (Patch level)	RO
0010h	31:00	cmdq_- phy_addr[63:32]	Physical address of the command queue record. This field should not be modified after INIT_HCA until TEARDOWN_HCA.	RW
0014h	31-12	cmdq_- phy_addr[31:12]	Physical address of the command queue record. This field should not be modified after INIT_HCA until TEARDOWN_HCA.	RW
	09-08	nic_interface	NIC interface mode 0x0: full_driver 0x1: disabled	WO
	07:04	log_cmdq_size	Log number of cmdqs available	RO
	03:00	log_cmdq_stride	Stride between start of each cmdq	RO
0018h	31:00	command DoorBell vector	Bit per command in the cmdq. When the bit is set, when writing this vector to the device, the command is moved to HW ownership (HW need to execute the command). bit 0 is related to the command in offset 0 in the command queue etc. The valid bits in this vector are [number of com□mands-1..0] When driver is in No DRAM NIC mode, this field must not be written.	WO
01fCh	31	initializing	1 - device still in initializing state. 0 - device is ready to receive commands.	RO
	26:24	nic_interface_ supported	Bitmask indicating which nic_interface modes are supported 0: full_driver 1: disabled	RO
1000h	31:00	internal_timer_h	MSBs of the current internal timer value	RO
1004h	31-00	internal_timer_l	LSBs of the current internal timer value	RO
100Ch	0	clear_int	Writing 1 to this register will clear the interrupt (will always use intA)	WO
1010h	31:24	health_syndrome	Syndrome 0x1: FW_INTERNAL_ERR - assert triggered in FW code 0x7: DEAD_IRISC - Irisc not responding 0x8: HW_FATAL_ERR 0x9: FW_CRC_ERR 0xA: ICM_FETCH_PCI_ERR 0xB: ICM_PAGE_ERR 0xC: ASYNCHRONOUS_EQ_BUF_OVERRUN 0xD: EQ_IN_ERR 0xE: EQ_INV 0xF: FFSER_ERR 0x10: HIGH_TEMP_ERR	RO

Figure 2-2 Initialization Field Description

```
mlx4_caps structure
        u64                     fw_ver;
        u32                     function;
        int                     num_ports;
        int                     vl_cap[MLX4_MAX_PORTS + 1];
        int                     ib_mtu_cap[MLX4_MAX_PORTS + 1];
        __be32                  ib_port_def_cap[MLX4_MAX_PORTS + 1];
        u64                     def_mac[MLX4_MAX_PORTS + 1];
        int                     eth_mtu_cap[MLX4_MAX_PORTS + 1];
        int                     gid_table_len[MLX4_MAX_PORTS + 1];
        int                     pkey_table_len[MLX4_MAX_PORTS + 1];
        int                     trans_type[MLX4_MAX_PORTS + 1];
        int                     vendor_oui[MLX4_MAX_PORTS + 1];
        int                     wavelength[MLX4_MAX_PORTS + 1];
        u64                     trans_code[MLX4_MAX_PORTS + 1];
        int                     local_ca_ack_delay;
        int                     num_uars;
        u32                     uar_page_size;
        int                     bf_reg_size;
        int                     bf_regs_per_page;
        int                     max_sq_sg;
        int                     max_rq_sg;
        int                     num_qps;
        int                     max_wqes;
        int                     max_sq_desc_sz;
        int                     max_rq_desc_sz;
        int                     max_qp_init_rdma;
        int                     max_qp_dest_rdma;
        int                     max_tc_eth;
        struct mlx4_spec_qps    *spec_qps;
        int                     num_srqs;
        int                     max_srq_wqes;
        int                     max_srq_sge;
        int                     reserved_srqs;
        int                     num_cqs;
        int                     max_cqes;
        int                     reserved_cqs;
        int                     num_sys_eqs;
        int                     num_eqs;
        int                     reserved_eqs;
        int                     num_comp_vectors;
        int                     num_mpts;
        int                     max_fmr_maps;
        int                     num_mtts;
```

```
int                     fmr_reserved_mtts;
int                     reserved_mtts;
int                     reserved_mrws;
int                     reserved_uars;
int                     num_mgms;
int                     num_amgms;
int                     reserved_mcgs;
int                     num_qp_per_mgm;
int                     steering_mode;
int                     dmfs_high_steer_mode;
int                     fs_log_max_ucast_qp_range_size;
int                     num_pds;
int                     reserved_pds;
int                     max_xrcds;
int                     reserved_xrcds;
int                     mtt_entry_sz;
u32                     max_msg_sz;
u32                     page_size_cap;
u64                     flags;
u64                     flags2;
u32                     bmme_flags;
u32                     reserved_lkey;
u16                     stat_rate_support;
u8                      port_width_cap[MLX4_MAX_PORTS + 1];
int                     max_gso_sz;
int                     max_rss_tbl_sz;
int             reserved_qps_cnt[MLX4_NUM_QP_REGION];
int                     reserved_qps;
int             reserved_qps_base[MLX4_NUM_QP_REGION];
int             log_num_macs;
int             log_num_vlans;
enum mlx4_port_type port_type[MLX4_MAX_PORTS + 1];
u8                      supported_type[MLX4_MAX_PORTS + 1];
u8                suggested_type[MLX4_MAX_PORTS + 1];
u8                default_sense[MLX4_MAX_PORTS + 1];
u32                     port_mask[MLX4_MAX_PORTS + 1];
enum mlx4_port_type possible_type[MLX4_MAX_PORTS + 1];
u32                     max_counters;
u8                      port_ib_mtu[MLX4_MAX_PORTS + 1];
u16                     sqp_demux;
u32                     eqe_size;
u32                     cqe_size;
u8                      eqe_factor;
u32                     userspace_caps; /* userspace must be aware of these */
```

```
u32                     function_caps;  /* VFs must be aware of these */
u16                     hca_core_clock;
u64                     phys_port_id[MLX4_MAX_PORTS + 1];
int                     tunnel_offload_mode;
u8                      rx_checksum_flags_port[MLX4_MAX_PORTS + 1];
u8                      phv_bit[MLX4_MAX_PORTS + 1];
u8                      alloc_res_qp_mask;
u32                     dmfs_high_rate_qpn_base;
u32                     dmfs_high_rate_qpn_range;
u32                     vf_caps;
bool                    wol_port[MLX4_MAX_PORTS + 1];
struct mlx4_rate_limit_caps rl_caps;
u32                     health_buffer_addrs;
```