ICT NCIC 系统互连组

HCA CtxMgt 设计文档

V3.0

Revision History

Date	Author	Comment
22/5/2020	Ma Xiaoxiao	第一版 CtxMgt 硬件架构设计
18/6/2020	Ma Xiaoxiao	增加 VirtToPhys 硬件架构设计
29/6/2020	Ma Xiaoxiao	CtxMgt 与 VirtToPhys 的 MetaData 各自独立存储； CtxMgt 仅存放 QP、CQ、EQ 的头指针
03/7/2020	Ma Xiaoxiao	形成 CtxMgt 独立的设计文档，细化各个模块的细节
08/7/2020	Ma Xiaoxiao	更新与 RDMA 引擎的交互信息
26/7/2020	Ma Xiaoxiao	增加与 RDMA 引擎的交互信息，完善消息格式
01/8/2020	Ma Xiaoxiao	修改 CtxMgt 整体设计框架
28/8/2020	Ma Xiaoxiao	根据代码修改 CtxMgt 设计 Parser、Request Controller、Pointer Table、QP-State Table 部分
20/01/2021	Ma Xiaoxiao	重新设计 CtxMgt，将整个上下文按需求整合存储
28/01/2021	Ma Xiaoxiao	完善 CtxMgt 内部子模块设计，更新内部接口，对应代码版本： D:\HPC_LAB\Lab_Project\2019IB_NIC\Design\Project\CtxMgt\ sources_21-01-28
13/09/2021	Ma Xiaoxiao	修改与 rdma 引擎的部分接口，增加了新上下文字段的存取操作，位于 D:\HPC_LAB\Lab_Project\2019IB_NIC\Design\Project\verification \ctxmgt_test\ctxmgt_test_v6\ctxmgt_v6
06/02/2022	Ma Xiaoxiao	按照“22-01-21 IB HCA CtxMgt 上下文响应格式 - 增补”文档修改上下文管理模块的存储信息内容及相应格式，增加对 RoCE 的支持。
22/07/2022	Ma Xiaoxiao	按照“2022-07-22 马潇潇 中断及 EQ 支持.docx”文档修改上下文管理模块的存储信息内容及相应格式，增加对 EQ 的支持。
11/08/2022	Ma Xiaoxiao	按照 sw hw interface 1.4 中 EQC 格式的修改，4.1.1.3 将 inter（8 bits）更新为 msix_vector（16 bits）
12/16/2022	Ma Xiaoxiao	全文更新用于京兆 2

目录

1 CtxMgt 模块功能

ICM Metadata 保存了 ICM 资源的虚拟地址到物理地址的映射表，上下文是网卡在通信过程中需要用到的重要元数据。由于多个模块都会访问 Metadata 和上下文，因此，设计 CxtMgt 模块对上下文部分的 Metadata 和上下文资源的访问进行集中控制。CxtMgt 模块的主要功能包括：

1. 响应 CEU 模块发起的请求，包括：
   - 写、无效 ICM MetaData 中的上下文部分，即 CtxMetaData；
   - 读、写、无效内存中的整个上下文条目；
   - 写、无效 SQ/RQ/CQ/EQ 的队列头指针（非显式的 CEU 请求，是上一种请求的附属操作）；
2. 响应 RDMA 引擎中不同模块发起的请求，包括：
   - 读取上下文条目中的 PSN，ST，Port、PMTU、QP State 等信息；
   - 将 PSN，QP State 等信息写回内存中的上下文条目中；
   - 读取上下文条目中的 SQ/RQ 头指针和 PD，用于读取 WQE；
   - 读取上下文条目中的 CQ/EQ 头指针和 PD，用于生成 CQE、EQE。

需要说明的是，上述请求中提及的 Metadata 和所有队列头指针等数据路径会访问的关键全部保存在 CxtMgt 模块的内部缓存中，整个上下文条目则全部存放在主机内存中。对于 CEU 和 RDMA 引擎而言，其并不关心数据具体的存放位置，只需要向 CxtMgt 提交访问相应资源的请求即可。

2 消息格式定义

由于 CxtMgt 要响应不同模块的不同数据请求，因此本设计中自定义了一种消息包头格式以便 CxtMgt 与其它模块进行数据交互。在实现中我们将**请求与负载分开传输，其中与 CEU 接口的 tuser 用于传输包头，tdata 用于传输数据**。

详细的请求包头消息格式定义如下所示：（左上高位，右下低位）

表 2.1 CxtMgt 模块消息字段定义

CEU 请求	Type	Opcode	Reserved	Addr	Data
	4 bits	4 bits	24 bits	32 bits	64 bits Optional 63:60 state **Optional**

RDMA 请求	Type	Opcode	Addr	Reserved	Data
	4 bits	4 bits	24 bits	32 bits	64 bits Optional 63:60 state **Optional**

字段介绍：

Type：用于 Parser 按照操作对象 CtxMetaData、上下文、以及是否需要返回数据分类；

Opcode：对应读、写、无效、修改操作；

Addr：请求要访问的上下文资源的条目 index（QP 号，CQ 号，EQ 号，ICM 虚拟地址）

Data：对不同的 Type 和 Opcode，负载可能为 CtxMetaData、山下文条目中的某部分字段；

表 2.2Type、Opcode 字段含义说明（注：红色下划线为融合网卡新增加的上下文信息）

AXIS Slave		
Type	**Opcode**	**说明**
RD_QP_CXT	RD_QP_ALL	CEU 使用，读取一个上下文条目 RDMA引擎使用(预留)，读取一个上下文条目
	RD_QP_NPST	RDMA引擎RTC使用，读取NextPSN；QP state；SCQ_lkey；SCQ_Length；CQ PD；CQN
	RD_QP_SST	RDMA引擎Doorbell Processing使用，读QP信息，包括ST：Service Type，链接类型；DstQPN：远端QP号；PKey： PMTU：只使用[7:5]三位，1到5分别表征256到4096；SQ LKey：发送队列LKey；QP PD：SQ，RQ及通信过程中数据区域使用的PD；SQ_Entry_Size_Log：每个SQ数据块的大小，用于预取判定；IB/RoCE：指示当前通信模式；SQ_Length：发送队列长度，用于循环判定
	RD_QP_RST	RDMA引擎EE使用，读取Pkey、PMTU、Expected PSN、RNR Timer、QP State，DstQPN：远端QPN； RQ LKey：接收队列LKey；CQ LKey：完成队列LKey；QP PD：QP保护域；CQ PD：CQ保护域；RQ_Length：接收队列长度； RCQ_Length：接收完成事件长度； RQ_Entry_Size_Log：每个RQ数据块的大小；RLID
	RD_QP_STATE	RDMA引擎WQE Parser使用，查询QP state；SQ_Entry_Size_Log：每个SQ数据块的大小，用于预取判定； SQ_Length：发送队列长度，用于循环判定
	RD_ENCAP	RDMA引擎FrameEncap使用，查询PKey；SL；IB/RoCE；Port；Source IP；Destination IP；Source MAC ([15:0] Source LID)； Destination MAC ([15:0] Destination LID)；

		Source MAC；Destination MAC
WR_QP_CXT	WR_QP_ALL	CEU 使用，写一个 QP 上下文条目
	WR_QP_UAPST	RDMA引擎RRC使用，写UnAcked PSN、QP State
	WR_QP_NPST	RDMA引擎RTC使用，写NextPSN、QP state
	WR_QP_EPST	RDMA引擎EE使用，写Expected PSN、QP State
	WR_QP_STATE	Reserved，写QP State
WR_CQ_CXT	WR_CQ_ALL	CEU 使用，写一个 CQ 上下文条目
	WR_CQ_MODIFY	CEU 使用，修改 CQ 上下文条目
	WR_CQ_INVALID	CEU 使用，将一个 CQ 上下文条目无效
WR_EQ_CXT	WR_EQ_ALL	CEU 使用，写一个 EQ 上下文条目
	WR_EQ_FUNC	CEU 使用，使能或失能 EQ 条目的事件功能
	WR_EQ_INVALID	CEU 使用，将一个 EQ 上下文条目无效
WR_ICMMAP_CXT	WR_ICMMAP_EN	CEU 使用，将上下文资源表写入 CxtMgt
	WR_ICMMAP_DIS	CEU 使用，无效写入的上下文资源表
MAP_ICM_CXT	MAP_ICM_EN	CEU 使用，将上下文资源映射的物理地址写入 CxtMgt
	MAP_ICM_DIS	CEU 使用，无效写入的上下文资源表
RD_CQ_CXT	RD_CQ_ALL(预留)	RDMA 引擎使用，读取一个CQ上下文条目
	RD_CQ_CST	RDMA 引擎RRC使用，读队列信息，包括SCQ_Lkey、NextPSN、UnAckedPSN、QP State；SCQ_Length；CQ PD；CQN；QP PD； RLID
RD_EQ_CXT	RD_EQ_ALL(预留)	RDMA 引擎使用，读取一个EQ上下文条目
	RD_EQ_K(预留)	RDMA 引擎使用，读取一个EQ头指针和PD
AXIS Master		
Type	**Opcode**	**说明**
RD_QP_CXT	RD_QP_ALL	CEU 使用，返回一个上下文条目 RDMA 引擎使用(预留)，返回一个上下文条目
	RD_QP_NPST	RDMA引擎RTC使用，读取NextPSN、QP state
	RD_QP_SST	RDMA引擎Doorbell Processing使用，读QP信息，包括PD、Lkey、Pkey/PMTU、服务类型ST、QPN

	RD_QP_RST	RDMA引擎EE使用，读取Pkey、PMTU、PD、Lkey、Expected PSN、RNR Timer、QP State
	RD_QP_STATE	RDMA引擎WQE Parser使用，查询QP State
RD_CQ_CXT	RD_CQ_ALL(预留)	RDMA 引擎使用，读取一个CQ上下文条目
	RD_CQ_CST	RDMA 引擎RRC使用，读队列信息，包括CQ_Lkey、NextPSN、UnAckedPSN、QP State
RD_EQ_CXT	RD_EQ_ALL(预留)	RDMA 引擎使用，返回一个EQ上下文条目
	RD_EQ_K(预留)	RDMA 引擎使用，返回一个EQ头指针和PD

3 模块接口定义

3.1 与 CEU 模块接口

使用 128 位宽的 AXI4-Stream 接口，包括 AXI4-Stream Slave 和 AXI4-Stream Master。其中 AXI4-Stream Slave 接收 CEU 对上下文和对 CtxMetaData 的两类操作。AXI4-Stream Master 则仅返回 CEU 对上下文读请求的上下文数据。

表 3.1 CxtMgt 模块与 CEU 模块包头接口定义

AXI4-Stream Slave——上下文操作				
描述	**Type**	**Opcode**	**Addr**	**Data**
读取 QP 信息	0b0001 RD_QP_CXT	0b0001 RD_QP_ALL	QP 号	\
写一个 QPC	0b0010 WR_QP_CXT	0b0001 WR_QP_ALL	QP 号	\
写一个 CQC	0b0011 WR_CQ_CXT	0b0001 WR_CQ_ALL	CQ 号	\
修改某一个 CQC		0b0010 WR_CQ_MODIFY	CQ 号	\
无效某一个 CQC		0b0011 WR_CQ_INVALID	CQ 号	\
写一个 EQC	0b0100 WR_EQ_CXT	0b0001 WR_EQ_ALL	EQ 号	\
使能/失能一个 EQ 事件功能		0b0010 WR_EQ_FUNC	EQ 号（最高位为 0，该 EQ 有	event_mask (64bit 中低 32

			效，最高位为 1，失效）	位，不做处理)
无效某一个 EQC		0b0100 WR_EQ_INVALID	EQ 号	\
AXI4-Stream Slave——CtxMetaData 操作				
描述	**Type**	**Opcode**	**Addr**	**Data**
创建 CtxMetaData 表	0b0101 WR_ICMMAP_CXT	0b0001 WR_ICMMAP_EN	\	\
无效 ICM CtxMetadata		0b0010 WR_ICMMAP_DIS	\	\
写 CtxMetadata 表项	0b0110 MAP_ICM_CXT	0b0001 MAP_ICM_EN	chunk_num (32 bits)	\
无效 CtxMetadata 表项		0b0010 MAP_ICM_DIS	page_cnt (32bit)	索引 (虚地址) (64 bit)
AXI4-Stream Master——CtxMetaData 操作				
描述	**Type**	**Opcode**	**Addr**	**Data**
响应 CEU QUERY_QP 命令，返回给 QP 信息	0b0001 RD_QP_CXT	0b0001 RD_QP_ALL	QP 号	\

地址映射数据负载格式：

Req header	Seg	type	opcode	R	num	R
	Bit width	4	4	24	32	64
	Value	MAP_ICM_TPT	MAP_ICM_EN	void	chunk_num	void
Req Payload	Bit width	64	64(high 52 addr; 11:0 page num)	重复 chunk_num 个 64virt+64page, 127:0 chunk0;255-128 chunk1 ••		
	Value	Virtual addr	page addr			

chunk_num 的数量不超过 64.（256KB/4KB = 64）

3.2 与 RDMA 引擎模块接口

请求通道使用 128 位宽的 FIFO 接口。与 RDMA 引擎具体连接的模块及所要的上下文信息如下：（橘黄色表示需要读写的信息）。

模块通道	上下文字段	读/写	获取方式
DoorbellProcessing（响应通道 256bit）	Service Type	只读	上下文读取
	Destination QPN	只读	上下文读取
	QP Protection Domain	只读	上下文读取
	SQ LKey	只读	上下文读取
	Port Key	只读	上下文读取
	PMTU	只读	上下文读取
	SQ_Entry_Size_Log	只读	上下文读取
	SQ_Length	只读	上下文读取
WQEParser（响应通道 128 bit）	QP State	只读	上下文读取
	SQ_Entry_Size_Log	只读	上下文读取
	SQ_Length	只读	上下文读取
RequesterTransControl（响应通道 256bit）	**QP State**	**读写**	**上下文读写**
	Next PSN	**读写**	**上下文读写**
	SCQ_LKey	只读	上下文读取
	SCQ_Length	只读	上下文读取
	CQ PD	只读	上下文读取
	CQN	只读	上下文读取
	RLID[15:0]	只读	上下文读取
	EQN	只读	上下文读取
	EQ Size log	只读	上下文读取
	EQ PD	只读	上下文读取
	EQ key	只读	上下文读取
	Msix_vector	只读	上下文读取
RequesterRecvControl（响应通道 256bit）	**QP State**	**读写**	**上下文读写**
	UnAcked PSN	**读写**	**上下文读写**
	Next PSN	只读	上下文读写
	DstQPN	只读	上下文读取
	RLID	只读	上下文读取
	SCQ LKey	只读	上下文读取

	QP PD	只读	上下文读取
	CQ PD	只读	上下文读取
	SCQ Length	只读	上下文读取
	CQN	只读	上下文读取
	EQN	只读	上下文读取
	EQ Size log	只读	上下文读取
	EQ PD	只读	上下文读取
	EQ key	只读	上下文读取
	Msix_vector	只读	上下文读取
ExecutionEngine （响应通道 320 bit）	**Expected PSN**	**读写**	**上下文读写**
	RNR Retry	只读	上下文读取
	QP State	**读写**	**上下文读写**
	RLID	只读	上下文读取
	PMTU	只读	上下文读取
	Port Key	只读	上下文读取
	RQ_Entry_Size_Log	只读	上下文读取
	DstQPN	只读	上下文读取
	RQ LKey	只读	上下文读取
	CQ LKey	只读	上下文读取
	QP PD	只读	上下文读取
	CQ PD	只读	上下文读取
	RQ_Length	只读	上下文读取
	RCQ_Length	只读	上下文读取
	CQN	只读	上下文读取
	EQN	只读	上下文读取
	EQ Size log	只读	上下文读取
	EQ PD	只读	上下文读取
	EQ key	只读	上下文读取
	Msix_vector	只读	上下文读取
FrameEncap （响应通道 256bit）	Port Key	只读	上下文读取
	Service level	只读	上下文读取
	IB/RoCE	只读	上下文读取
	Port	只读	上下文读取
	Source IP	只读	上下文读取

	Destination IP	只读	上下文读取
	Source MAC	只读	上下文读取
	Destination MAC	只读	上下文读取

表 3.2 CxtMgt 模块与 RDMA 引擎模块接口请求包头及响应形式定义（左下高右上低）

Doorbell Processing

RDMA 引擎 Doorbell Processing 请求（cxtmgt_cmd FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读 QP 信息，包括 PD、SEND_Lkey、Pkey/PMTU、服务类型 ST、DestQPN	RD_QP_CXT	RD_QP_SST	QP 号	32*3'b0

RDMA引擎 Doorbell Processing 请求命令回传（cxtmgt_resp FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读 QP 信息，包括 PD、SEND_Lkey、Pkey、PMTU、服务类型 ST、DestQPN	RD_QP_CXT	RD_QP_SST	QP 号	32*3'b0

RDMA引擎 Doorbell Processing响应数据负载格式（cxtmgt_cxt FIFO）

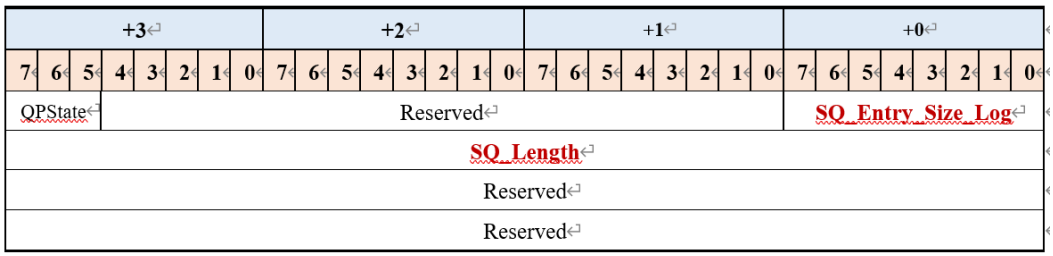
+3								+2								+1								+0							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
DstQPN																								Reserved						ST	
Reserved								PMTU_MSGMAX								PKey															
SQ LKey																															
QP PD																															
Reserved																								SQ_Entry_Size_Log							
SQ_Length																															
Reserved																															
Reserved																															

WQEParser

RDMA 引擎 WQEParser 请求包头（cxtmgt_cmd FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读 QP 信息，QP State	RD_QP_CXT	RD_QP_STATE	QP 号	32*3'b0
RDMA 引擎 WQEParser 请求数据负载（cxtmgt_ctx FIFO）				
读队列请求负载	无负载			

RDMA 引擎 WQEParser 请求命令回传包头（cxtmgt_resp FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读 QP 信息，QP State	RD_QP_CXT	RD_QP_STATE	QP 号	32*3’b0

RDMA 引擎 WQEParser 读响应数据负载（cxtmgt_ctx FIFO）



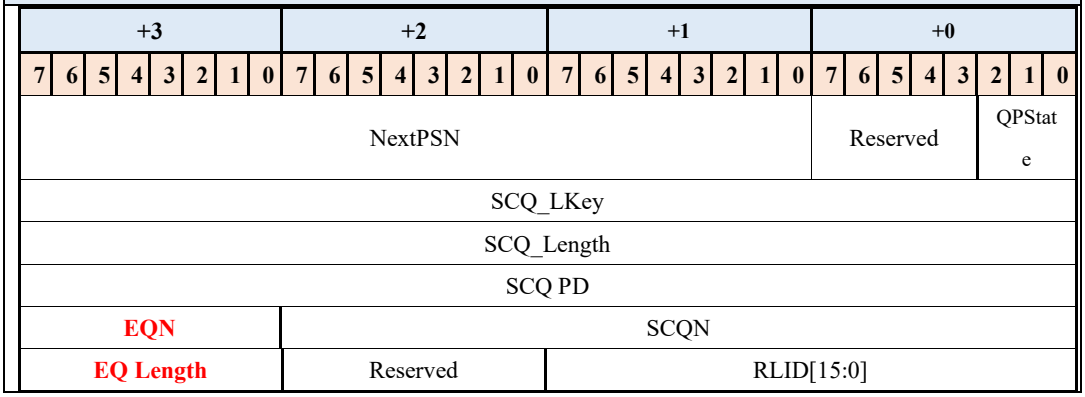
RequesterTransControl

RDMA 引擎 RequesterTransControl 请求包头（cxtmgt_cmd FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读 QP 信息，包括 NextPSN、QP State	RD_QP_CXT	RD_QP_NPST	QP 号	32*3’b0
写 QP 信息，包括 NextPSN、QP State	WR_QP_CXT	WR_QP_NPST	QP 号	32*3’b0

RDMA 引擎 RequesterTransControl 请求数据负载（cxtmgt_ctx FIFO）				
描述	**Reserved(96)**	**Expected PSN(24)**	**Reserved(5)**	**QP State(3)**
读队列请求负载	无负载			
写 QP 请求负载	0	Next PSN	0	State

RDMA 引擎 RequesterTransControl 请求命令回传包头（cxtmgt_resp FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读 QP 信息，包括 NextPSN、QP State	RD_QP_CXT	RD_QP_NPST	QP 号	32*3’b0
~~写 QP 信息，包括 NextPSN、QP State~~	~~WR_QP_CXT~~	~~WR_QP_NPST~~	~~QP 号~~	~~32*3’b0~~

RDMA 引擎 RequesterTransControl 读响应数据负载（cxtmgt_ctx FIFO）



EQ LKey	
EQ PD	
Reserved	Msix_vector

RequesterRecvControl

RDMA 引擎 RequesterRecvControl 请求包头（cxtmgt_cmd FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读队列信息，包括 CQ_Lkey、NextPSN、UnAckedPSN、QP State	RD_CQ_CXT	RD_CQ_CST	QP 号	32*3'b0
写 QP 信息，包括 UnAcked PSN、QP State	WR_QP_CXT	WR_QP_UAPST	QP 号	32*3'b0

RDMA 引擎 RequesterRecvControl 请求数据负载（cxtmgt_ctx FIFO）				
描述	**Reserved(96)**	**Expected PSN(24)**	**Reserved(5)**	**QP State(3)**
读队列请求负载	无负载			
写 QP 请求负载	0	UnAcked PSN	0	State

RDMA 引擎 RequesterRecvControl 请求命令回传包头（cxtmgt_resp FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读队列信息，包括 CQ_Lkey、NextPSN、UnAckedPSN、QP State	RD_CQ_CXT	RD_CQ_CST	QP 号	32*3'b0
~~写 QP 信息，包括 UnAcked PSN、QP State~~	~~WR_QP_CXT~~	~~WR_QP_UAPST~~	~~QP 号~~	~~32*3'b0~~

RDMA 引擎 RequesterRecvControl 读响应数据负载（cxtmgt_ctx FIFO）

+3								+2								+1								+0							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
UnAcked PSN																								Reserved						QPState	
NextPSN																								RLID[7:0]							
DstQPN																								RLID[15:8]							
SCQ LKey																															
QP PD																															
SCQ PD																															
SCQ Length																															
EQN								SCQN																							
EQ LKey																															
EQ Length								Reserved								Msix_vector															
EQ PD																															

Execution Engine

RDMA 引擎 Execution Engine 请求包头（cxtmgt_cmd FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读队列信息，包括 Pkey、PMTU、PD、RECV_LKey、Expected PSN、RNR Timer、QP State	RD_QP_CXT	RD_QP_RST	QP 号	32*3'b0
写 QP 信息，包括 Expected PSN、QP State	WR_QP_CXT	WR_QP_EPST	QP 号	32*3'b0

RDMA 引擎 Execution Engine 请求数据负载（cxtmgt_ctx FIFO）				
描述	**Reserved(96)**	**Expected PSN(24)**	**Reserved(5)**	**QP State(3)**
读队列请求负载	无负载			
写 QP 请求负载	0	PSN	0	State

RDMA 引擎 Execution Engine 请求命令回传包头（cxtmgt_resp FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
读队列信息，包括 Pkey、PMTU、PD、RECV_LKey、Expected PSN、RNR Timer、QP State	RD_QP_CXT	RD_QP_RST	QP 号	32*3'b0

RDMA 引擎 Execution Engine 读响应数据负载（cxtmgt_ctx FIFO）

+3								+2								+1								+0							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Expected PSN																								RNRTimer					QPState		
RLID[7:0]								PMTU_MSGMAX								PKey															
RQ_Entry_Size_Log								DstQPN																							
RQ LKey																															
CQ LKey																															
QP PD																															
CQ PD																															
RQ_Length																															
RCQ_Length																															
RLID[15:8]								RCQN																							
EQ LKey																															
Msix_vector																**EQN**								**EQ Length**							
EQ PD																															

FrameEncap

RDMA 引擎 Execution Engine 请求包头（cxtmgt_cmd FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
查询 PKey；SL；IB/RoCE；Port；Source IP；Destination IP；Source MAC ([15:0] Source LID)；Destination MAC ([15:0] Destination LID)；Source MAC；Destination MAC	RD_QP_CXT	RD_ENCAP	QP 号	32*3'b0

RDMA 引擎 Execution Engine 请求命令回传包头（cxtmgt_resp FIFO）				
描述	**Type(4)**	**Opcode(4)**	**Addr(24)**	**Data**
查询 PKey；SL；IB/RoCE；Port；Source IP；Destination IP；Source MAC ([15:0] Source LID)；Destination MAC ([15:0] Destination LID)；Source MAC；Destination MAC	RD_QP_CXT	RD_ENCAP	QP 号	32*3'b0

RDMA 引擎 Execution Engine 读响应数据负载（cxtmgt_ctx FIFO）

Reserved+3								+2								+1								+0							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
PKey																[7:4]：SL				[3]：IB/RoCE=0				[2:0]：Port							
Source IP																															
Destination IP																															
Source MAC[31:0] ([15:0] Source LID)																															
Destination MAC[15:0] ([15:0] Destination LID)																Source MAC[47:32]															
Destination MAC[47:16]																															
Reserved																															
Reserved																															

3.3 与 DMA 引擎接口

1 组读接口、1 组写接口，包头位宽 128bit，数据位宽 256bit，其中 AXIS 的 tuser 用于

传输包头，tdata 用于传输数据。

表 3.3 CxtMgt 模块与 DMA 引擎模块接口定义

对接模块：DMA Engine			
读写组	**信号**	**in/out**	**描述**
读 1	AXI4-Stream Master dma_cm_rd_req	out	提交 DMA 读上下文表项请求（tuser）
	AXI4-Stream Slave dma_cm_rd_rsp	in	接收 DMA 读上下文表项（tdata）
写 1	AXI4-Stream Master dma_cm_wr_req	out	提交 DMA 写上下文表项请求（tuser）
		out	提交 DMA 写上下文表项数据（tdata）

4 模块架构

4.1 数据分布

1．存放 ICM MetaData 中的上下文部分，含有 QP（SQ、RQ） Context、CQ Context、EQ Context 的虚拟地址（HCA 私有）、大小、物理页面地址、页大小信息，用于对主机内存中的上下文进行读、写操作，CtxMData 具体分布见子模块设计；

2．存放 QPC、CQC、EQC 信息，其中 QP 上下文的 lkey、pd、WQE base、PMTU 等信息都是通过同一个 QP（2^13 个）号查询，其中 CQ（2^13 个）、EQ（32 个）的 lkey、pd 分别通过 CQ 号和 EQ 号查询。在该版本数据通路的设计中使用的的上下文信息为整个上下文信息的一个子集，在实现中将上下文中数据通路会读写的关键数据全部放在网卡上。

4.2 数据来源说明

4.2.1 QPC 信息说明

RDMA 引擎在进行数据通信中用到的关键信息（lkey、pd、WQE base、PMTU 等）在 QPC 中的位置（大端模式，在每个 cycle 传输中，左上高位右下低位，不足位宽，低位补 0；**注：表中有底纹的信息为存在网卡中的信息，含黄色底纹、红色底纹**）。

QPC 中数据通路常用的关键信息共(12+9)*4-2=82 个字节，按照 2^13 个 QP 计算，共需要 82*2^13= **656KB** 空间。

offset	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
00h	opt_param_mask（软件修改的 Mask）																															
04h	reserved																															

08h	state	flags	servtype	mig state	flags	flags	flags
0Ch	mtu_msgmax		rq_entry_sz_log	sq_entry_sz_log		rlkey_arbel_sched_queue	
10h	usr_page						
14h	local_qpn						
18h	remote_qpn（Dest QPN）						
1Ch	port_pkey 端口号[26:24] pkey_index[6:0]						
20h	rnr_retry		g_mylmc	reserved			
24h	ackto		mgid_index	static_rate		hop_limit	
28h	sl_tclass_flowlabel						
2Ch	rgid[127:96]						
30h	rgid[95:64]						
34h	rgid[63:32]						
38h	rgid[31:0]						
3Ch	dlid(dmac[15:0])			slid(smac[15:0])			
40h	smac[47:16]						
44h	dmac[47:16]						
48h	sip						
4Ch	dip						
50h	reserved(alt_path)						
54h	reserved(alt_path)						
58h	reserved(alt_path)						
5Ch	QP pd						
60h	wqe_base						
64h	wqe_lkey						
68h	reserved （params1）						
6Ch	next_send_psn(Next PSN)						
70h	cqn_snd						
74h	snd_wqe_base_lkey（SQ Lkey）						
78h	snd_wqe_length（SQ Length）						
7Ch	last_acked_psn(UnAckedPSN)						
80h	ssn						
84h	rnr_nextrecvpsn(Expected PSN)						
88h	ra_buff_indx						

8Ch	cqn_rcv	
90h	rcv_wqe_base_lkey	
94h	rcv_wqe_length	
98h	qkey	
9Ch	rmsn	
A0h		
A4h		
A8h		
ACh		
B0h		
B4h		
B8h		
BCh	rq_wqe_counter	sq_wqe_counter

各个字段含义

Offset	Bits	Name	Description	Access
00h	31:0	opt_param_mask	本次命令要修改属性的 mask，详见表 A1-3	WR
08h	31:0	flags	QP 状态标志	WR
0Ch	31:24	mtu_msgmax	MTU [7:5] & 消息最大值[4:0] `enum ib_mtu { IB_MTU_256 = 1, IB_MTU_512 = 2, IB_MTU_1024 = 3, IB_MTU_2048 = 4, IB_MTU_4096 = 5 };` maxmsg 为字节数的 log 值	WR
	23:16	rq_entry_sz_log	RQ 中一个 WQE 条目的大小(byte)，以 2 为底数	WR
	15:8	sq_entry_sz_log	SQ 中一个 WQE 条目的大小（byte），以 2 为底数	WR
	7:0	rlkey_arbel_sched_queue	（目前没用）	WR
10h	31:0	usr_page	pfn of UAR page（目前没用）	WR
14h	31:0	local_qpn	该 QP 的 QP 号	WR
18h	31:0	remote_qpn	远端的 QP 号，用于连接服务类型	WR
1Ch	31:0	port_pkey	端口号[26:24] pkey index[6:0]/	WR
20h	31:24	rnr_retry	3bit 的请求方接收到远端 RNR NAK 后重发的次数（在报告错误之前）。7 代表无限重发。	WR
	23:16	g_mylmc	has grh[7:7]：是否使用 GRH local mask control[6:0]：用于向端口指定 LID	WR

			（目前没用）	
24h	31:24	ackto	ack timeout（目前没用）	WR
	23:16	mgid_index	sgid_index,端口 GID 表的 index（目前没用）	WR
	15:8	static_rate	获得端口静态速率（目前没用）	WR
	7:0	hop_limit	数据包经历的跳数限制（目前没用）	WR
28h	31:0	sl_tclass_flowlabel	sl & traffic class & flow label（目前没用）	WR
2Ch	31:0	rgid[127:96]	目的 GID（目前没用）	WR
30h	31:0	rgid[95:64]	目的 GID（目前没用）	WR
34h	31:0	rgid[63:32]	目的 GID（目前没用）	WR
38h	31:0	rgid[31:0]	目的 GID（目前没用）	WR
3Ch	31:16	dlid	目的 LID，或目的 MAC 的低 16 位（仅在 RoCE 模式下有用）	WR
	15:0	slid	源 LID，或源 MAC 的低 16 位（仅在 RoCE 模式下有用）	WR
40h	31:0	smac[47:16]	源 MAC 的高 32 位（仅在 RoCE 模式下有用）	WR
44h	31:0	dmac[47:16]	目的 MAC 的高 32 位（仅在 RoCE 模式下有用）	WR
48h	31:0	sip	源 IP（仅在 RoCE 模式下有用）	WR
4Ch	31:0	dip	目的 IP（仅在 RoCE 模式下有用）	WR
5Ch	31:0	pd	QP 所在保护域	WR
60h	31:0	wqe_base	（目前没用）	WR
64h	31:0	wqe_lkey	QP 队列所在 Memory Region 的 lkey（目前没用）	WR
6Ch	31:0	next_send_psn	下一个要发送的消息的 PSN（包序列号）	WR
70h	31:0	cqn_snd	SQ 的 CQ 号	WR
74h	31:0	snd_wqe_base_l	发送队列的基地址的 lkey	WR
78h	31:0	snd_wqe_len	发送队列的总长度（byte）	WR
7Ch	31:0	last_acked_psn	之前 ACK 的 PSN	WR
80h	31:0	ssn	（目前没用）	WR
84h	31:0	rnr_nextrecvpsn	Recv not Ready[31:24] & ePSN[23:0]	WR
88h	31:0	ra_buff_indx	（目前没用）	WR
8Ch	31:0	cqn_rcv	RQ 的 CQ 号	WR
90h	31:0	rcv_wqe_base_l	接收队列的基地址的 lkey	WR
94h	31:0	rcv_wqe_len	接收队列的总长度（byte）	WR
98h	31:0	qkey	在数据报服务类型中用于验证远端发送方对本地接收队列的访问权限，须在接收队列 WQE 提交前建立好（目前没用）	WR
9Ch	31:0	rmsn	（目前没用）	WR
A0h	31:16	rq_wqe_counter	（目前没用）	WR
	15:0	sq_wqe_counter	（目前没用）	WR

4.2.2 CQC 信息说明

CQ 的 lkey、pd 在 CQC 中的位置：

CQ 的最大数量设置为 2^13，与 QP 的最大数量相同。CQC 中数据通路常用的关键信息共 13 个字节。**注：表中有底纹的信息为存在网卡中的信息**。

offset	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
00h	flags																															
04h	start[63:32]																															
08h	start[31:0]																															
0Ch	logsize								usrpage																							
10h	comp_eqn																															
14h	CQ pd																															
18h	lkey																															
1Ch	last_notified_index																															
20h	solicit_producer_index																															
24h	consumer_index																															
28h	producer_index																															
2Ch	cqn																															
30h	ci_db																															
34h	state_db																															

字段含义

Offset	Bits	Name	Description	Access
00h	31:0	flags	CQ 的相关配置属性，详见表 3-2-34	WR
04h	31:0	start[63:32]	CQ 队列的起始虚拟地址（高 32 位）	WR
08h	31:0	start[31:0]	CQ 队列的起始虚拟地址（低 32 位）（目前没用）	WR
0Ch	31:24	logsize	CQ 队列可存放 CQE 的个数，以 2 为底数	WR
	23:0	usrpage	UAR 页面指针（目前没用）	WR
10h	31:0	comp_eqn	与该 CQ 关联的完成事件队列的 EQ 号（目前没用）	WR
14h	31:0	pd	与该 CQ 关联的 PD 的 PD 号（目前没用）	WR
18h	31:0	lkey	CQ 队列的 lkey	WR
2Ch	31:0	cqn	该 CQ 的 CQ 号	WR

4.2.3 EQC 信息说明

EQ 的 lkey、pd 在 EQC 中的位置：EQ 的最大数量参考 mthca 网卡设定为 32。**注：表中有底纹的信息为存在网卡中的信息**。

offset	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
00h	flags																															
04h	start[63:32]																															
08h	start[31:0]																															
0Ch	logsize_usrpage																															
10h																																
14h																	Intr（改为 msix_vector）															
18h	pd																															
1Ch	lkey																															
20h																																
24h																																
28h	consumer_index																															
2Ch	producer_index																															

4.3 整体架构

子模块简介

ceu_parser：（1）若是 CEU 发起的上下文读操作，则交给 CtxMetaData 模块查询并发起对内存对应的上下文读操作；（2）若是写、失效上下文操作，则交给 CtxMetaData 模块查询并发起对内存对应的上下文写、失效（写 0）操作；同时需要向 Requester Controller 模块发起请求，并将写的数据传递给 Key QPC Data 模块；（3）若是 CEU 发起的 CtxMetaData 写、失效操作，则交给 CtxMDataOp 模块对对应的 CtxMetaData 表项进行操作；

request controller：对 CEU、RDMA 引擎访问网卡上关键数据信息的请求进行选择调度，其中 CEU 的请求具备最高的优先级，RDMA 引擎的请求 采用轮训的方式进行调度；Key QPC Data 模块根据本模块的调度从对应的通道中读取 cmd 请求，并将请求通过 resp FIFO 返回命令。

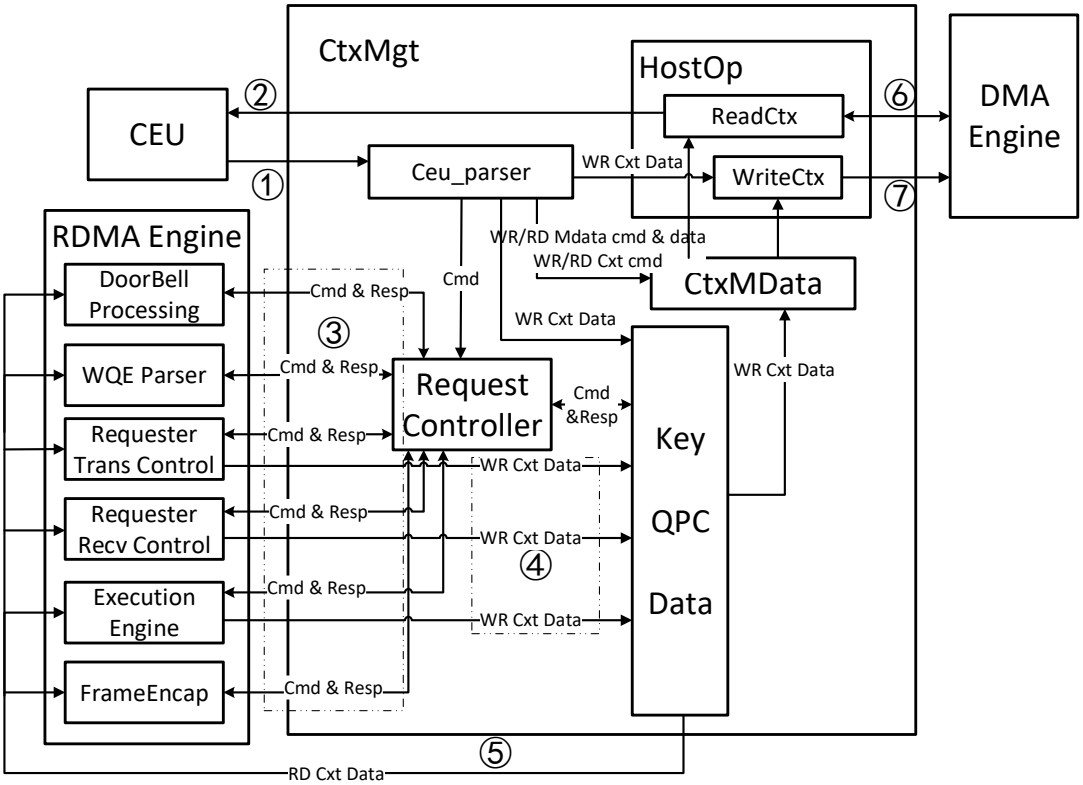
ctxmdata：用于存储查询上下文的物理地址信息。初始化配置过程中，接收 ceu_parser 发起的 metadata 数据的写、无效等操作；数据通信过程中，接收 ceu_parser 读/写上下文的请求，获取要访问的上下文的物理地址，并向 readctx/writectx 模块发起读写请求；当 RDMA 引擎中的某些通道发起写上下文的请求时，ctxmdata 模块会接收 key_qpc_data 的写请求，查询到对应上下文的物理地址，并通过 writectx 模块发起 DMA 写请求。

key_qpc_data：用于存储数据通信过程中关键的上下文信息。具体数据内容如 4.1.2 中介绍。该模块根据 request controller 模块的调度结果，从对应的请求通道中，读取 cmd 请求并在 resp 通道中返回请求命令（RDMA 模块的请求），若是写上下文请求，还需要读取 cxt 内容，以备将数据写入存储区；若是读请求，则需要通过 cxt 通道将读取到的数据返回到对应

的请求模块中。

readctx：用于发起 DMA 读请求。接收 ctxmdata 模块发起的 DMA 读请求，并将读请求以 axis 的格式传递给 DMA 引擎。并且针对 CEU 发起的读 QPC 命令，将整个 QPC 封装成对应的接口格式，传递给 CEU 模块。

writectx：用于发起 DMA 写请求。接收 ctxmdata 模块发起的 DMA 写请求；若是 CEU 的写请求，需要从 ceu_parser 的输出通道中获取要写回的数据；若是 key_qpc_data 模块的写请求，则需要从 key_qpc_data 获取要写回的数据。



4.4 外部模块通路

① 以 3.1 节表 3.1 AXIS Slave 的消息格式，CEU 发起上下文读、写、失效操作，CtxMetaData 写、失效操作；

② 以 3.1 节表 3.1 AXIS Master 的消息格式，CtxMgt 将上下文数据返回给 CEU；

③ 以 3.2 节表 3.2 的 cmd/resp 消息格式，接收 RDMA 引擎中 Doorbell Processing、WQE Parser、Requester Trans Control、Requester Recv Control、Execution Engine 模块的 cmd fifo 中的请求，并通过 resp fifo 对命令进行回传确认

④ 以 3.2 节表 3.2 的 cxt 消息格式，获取 RDMA 引擎中 Requester Trans Control、Requester Recv Control、Execution Engine 模块发起的上下文写操作的数据负载，在 key_qpc_data 模块中，将数据写入对应的数据区；

⑤ 以 3.2 节表 3.2 的 cxt 消息格式，向 RDMA 引擎中 Doorbell Processing、WQE Parser、Requester Trans Control、Requester Recv Control、Execution Engine 模块返回读取上下文信息的数据负载；

⑥ 根据 CEU 或者 RDMA 引擎对上下文的请求，CtxMgt 发起 DMA 读上下文请求并接收 DMA 返回的上下文数据；

⑦ 根据 CEU 或者 RDMA 引擎对上下文的请求，CtxMgt 发起 DMA 写请求并向内存写上下文信息；

5 子模块设计

5.1 ceu_parser

5.1.1 模块功能

（1）若是 CEU 发起的上下文读操作，则交给 CtxMetaData 模块查询并发起对内存对应的上下文读操作；（2）若是写、失效上下文操作，则交给 CtxMetaData 模块查询并发起对内存对应的上下文写、失效（写 0）操作；同时需要向 Requester Controller 模块发起请求，并提取出对应的数据传递给 Key QPC Data 模块；（3）若是 CEU 发起的 CtxMetaData 写、失效操作，则交给 CtxMDataOp 模块对对应的 CtxMetaData 表项进行操作；

5.1.2 接口定义

CEU 的 Parser 接口

对接子模块	通道	In/Out	位宽	字段			
request controller	req fifo.empty	Out	35	Type 4	Opcode 4	Source 3	QPN 24
key qpc data	req fifo	Out	35	Type 4	Opcode 4	Source 3	QPN 24
ctxmdata	req fifo	Out	128	见 3.1			
ctxmdata	mdata fifo	Out	256	metadata payload			
writectx	ctxdata fifo	Out	256	context payload			
key qpc data	data fifo	Out	384	context payload			
CEU	axis	In	128tuesr; 256tdata	见表 3.1			

5.1.3 上下文解析

QP Context

2^13 个。mthca 每个条目长度为 64*4=256 字节，，该版设计 QPC 条目的有效长度为 **164** 字节（256 bit 位宽即 32 字节要 6 个时钟）。格式参考 4.1.1

CQ Context

2^13 个。mthca 每个条目长度为 16*4=64 字节，除去 reserved3 暂不实现的字段，该版设计 CQC 条目的有效长度为 **56** 字节（256 bit 即 32 字节位宽要 2 个时钟）。格式参考 4.1.1

EQ Context

32 个。mthca 每个条目长度为 16*4=64 字节，除去 error_eqn、reserved 暂不实现的字段，该版设计 EQC 条目的有效长度 **48** 字节（256 bit 即 32 字节位宽要 2 个时钟）。格式参考 4.1.1

5.1.4 状态机设计

PARSE_REQ：判断并获取 CEU 的请求和第一个 clk 的数据，进入 TRANS_REQ_DATA 状态；

TRANS_REQ_DATA：对获得的 CEU 请求包头进行分析，根据请求类型，判断是否有数据负载，按内部接口定义，重新组织内部接口的请求包头和数据负载。具体操作有：（1）若是 CEU 发起的上下文读操作，直接将请求转发给 CtxMetaData 模块查询并发起对内存对应的上下文读操作；（2）若是写、失效上下文操作，则交给 CtxMetaData 模块查询并发起对内存对应的上下文写、失效（写 0）操作；同时需要向 Requester Controller 模块发起请求，并提取出对应的数据传递给 Key QPC Data 模块；（3）若是 CEU 发起的 CtxMetaData 写、失效操作，则交给 CtxMDataOp 模块对对应的 CtxMetaData 表项进行操作；

5.2 request controller

5.2.1 模块功能

对 CEU、RDMA 引擎访问网卡上关键数据信息的请求进行选择调度，其中 CEU 的请求具备最高的优先级，RDMA 引擎的请求采用轮训的方式进行调度；Key QPC Data 模块根据本模块的调度选择寄存器 selected_channel 从对应的通道中读取 cmd 请求，并将请求通过 resp FIFO 返回命令。

selected_channel 寄存器在进行 Req Channel 选择时更新，7 个 Channel 中每次只有 1 个 Channel 和 Valid 信号同时为 1，其余 Channel 为 0,。

字段	位偏移	说明
Channel 0	0	ceu_parser
Channel 1	1	Doorbell Processing
Channel 2	2	WQE Parser
Channel 3	3	Requester Trans Control
Channel 4	4	Requester Recv Control
Channel 5	5	Execution Engine
Channel 6	6	Frame Encap
Valid	7	更新 Channel bit 的同时，置 1，表明此时 Reg 已更新；key_qpc_data 读取 Req 后会将返回 receive_req 信号，导致 Valid 置 0，表明此时需要更新

5.2.2 接口定义

对接模块	信号	In/Out	位宽
ceu_parser	req fifo empty	In	1
Doorbell Processing	cmd fifo .empty	In	1
WQE Parser	cmd fifo .empty	In	1
Requester Trans Control	cmd fifo .empty	In	1
Requester Recv Control	cmd fifo .empty	In	1
Execution Engine	cmd fifo .empty	In	1
Frame Encap	cmd fifo .empty	In	1
selected channel ctl	selected_channel	Out	8
key qpc data	receive_req	In	1

5.3 ctxmdata

5.3.1 模块功能

用于存储查询上下文的物理地址信息。初始化配置过程中，接收 ceu_parser 发起的 metadata 数据的写、无效等操作；数据通信过程中，接收 ceu_parser 读/写上下文的请求，获取要访问的上下文的物理地址，并向 readctx/writectx 模块发起读写请求；当 RDMA 引擎中的某些通道发起写上下文的请求时，ctxmdata 模块会接收 key_qpc_data 的写请求，查询到对应上下文的物理地址，并通过 writectx 模块发起 DMA 写请求。

调度策略：优先处理 cue_parser 的请求，其次处理 key_ctx_data 的请求

5.3.2 接口定义

对接子模块	通道	In/Out	位宽	字段
ceu_parser	req fifo	In	128	见 3.1
ceu_parser	mdata fifo	In	256	metadata payload
writectx	req_ctx fifo	Out	128	Type4+ Opcode4+ Source3+ R21+data32+Addr64
readctx	req fifo	Out	108	Addr(64)+Len(12)+QPN(32)
key_qpc_data	req_ctx fifo	In	128	Type4+ Opcode4+R24+QPN32+data64

5.3.3 内部数据布局

每个表项的内容如下：其实就是类似 MTT

字段	Page_base	Valid

位宽（bits）	52	1

5.3.3.1 qpctxmdata

空间描述

qpctxmdata 空间大小：若***设计欲支持 16K，即 2^14 个 QP***，按照 2^14 个 256B（164B 按 2 的幂次则按 256B 计算）QPC 表项计算，需要的 ICM 空间为 2^14*2^8=2^22B，即 4MB。按照内核态驱动的 ICM 分配机制——mthca_alloc_icm_table（linux-5.4.2\drivers\infiniband\hw\mthca\mthca_memfree.c）。将 ICM 分成 256KB 大小的 chunk，每个 chunk 中含有的 QPC 数量为 256KB/256B=2^10，若共 2^14 个 QPC，则需要 2^4 个 chunk。考虑到 chunk 内的数据可能不是连续的情况，按 4KB 页大小计算，***qpctxmdata 表项最多要有 4MB/4KB=1024 个***。

使用 SingleDualPortRAM 模块例化为 52 位宽、1024 深度、地址位宽为 10 位的 RAM，因为这里实际上存储的是 64 位物理地址，而物理地址是以 4KB 为大小划分的，因此，低 12 位全为 0，没有存储的必要；以及 1024 深度 1 位宽度的 valid 标志位；

功能描述：

（1） 根据 ceu_parser 分发的 ceu 请求，完成 qpctxmdata 的表创建、表无效、表项填写、表项无效。对于表项是否有效，使用了一个 512 深度 1 位宽的列表，对应 512 个表项是否有效。

接口见 3.1

（2） 响应 ceu_parser 发起的查询 qpctmdata 表项的请求，即 CEU 创建、修改、无效、读取 QPC 表项的请求，需要获取物理地址实现对主机内存的写、读操作。

查询方法：*ceu ICM 命令涉及到的地址查询*：在 qpctxmdata 实现中，我们使用***virtual addr –qpc_base（从 INIT HCA 请求包头中 qpc_base 后 8 位补 0 获得）获得的 64 位值，取其中的【21:12】共 10 位，即 1024 深度的 RAM 中的地址。***（因为虚实地址映射是以 4KB 页大小为单位的，所以低 12 位无用，而 1024 深度共 10 位）

（3） 响应 keq_qpc_data 模块发起的查询 qpctxmdata 表项请求，即 RDMA 引擎修改 QPC 中部分字段引起的上下文变化，需要将内存中的相应字段更新。

查询方法：*key_qpc_data 或 CEU 请求中使用 QP 号的地址查询*：移位后的 QPN（***左移 QPC 表项的大小，即 256B，8 位***），再取其中的***【21:12】***位即是 qpctxmdata 虚实地址映射中的 RAM 地址，据此可以查到对应的物理页面。移位后的 QPN 的低 12 位计算物理页面的偏移。查询 cqctxmata、eqctxmdata 的过程与此类似。

（4） 根据 ceu_parse、key_qpc_data 的请求，以及查询到的物理地址结果，分别向 DMA Write Ctx、DMA Read Ctx 模块发起 DMA Write 或者 DMA Read QPC 表项请求。

5.3.3.2 cqctxmdata

空间描述

cqctxdata 空间大小：CQ 数量最多定为 2^14 个。mthca 每个条目长度为 16*4=64 字节，除去 reserved3 暂不实现的字段，该版设计 CQC 条目的有效长度为 **56** 字节，按照 2^14 个 64B（56B 按照 2 的幂次取 64B 计算）的 CQC 表项计算，需要的***ICM 空间为2^14*2^6=2^20B，即 1MB***。按照内核态驱动的 ICM 分配机制——mthca_alloc_icm_table（linux-5.4.2\drivers\infiniband\hw\mthca\mthca_memfree.c）。将 ICM 分成 256KB 大小的 chunk，每个 chunk 中含有的 CQC 数量为 256KB/64B=2^12，若共 2^14 个 CQC，则需要 2^2 个 chunk。考虑到 chunk 内的数据可能不是连续的情况，按 4KB 页大小计算，***cqctxmdata 表项最多要有 1MB/4KB=256 个***。

使用 SingleDualPortRAM 模块例化深度为 512，宽度为 52，地址位宽为 8 位的 RAM。

功能描述：

（1） 根据 ceu_parser 分发的 ceu 请求，完成 cqctxmdata 的表创建、表无效、表项填写、表项无效。

接口见 3.1

（2） 响应 ceu_parser 发起的查询 cqctmdata 表项的请求，即 CEU 创建、修改、无效、CPC 表项的请求，需要获取物理地址实现对主机内存的写操作。

查询方法：*ceu 命令涉及到的地址查询*：在 cqctxmdata 实现中，我们使用 virtual addr –cqc_base（从 INIT HCA 请求包头中 cqc_base 后 8 位补 0 获得）获得的 64 位值，取其中的***【19:12】***共 8 位，即 256 深度的 RAM 中的地址。（因为虚实地址映射是以 4KB 页大小为单位的，所以低 12 位无用，而 256 深度共 8 位）

查询方法：*CEU 请求中使用 QP 号的地址查询*：移位后的 CQN（***左移 QPC 表项的大小，即 64B，6 位***），再取其中的***【21:12】***位即是 qpctxmdata 虚实地址映射中的 RAM 地址，据此可以查到对应的物理页面。移位后的 CQN 的低 12 位计算物理页面的偏移。查询 eqctxmdata 的过程与此类似。

（3） ceu_parser 的请求，以及查询结果，向 DMA Write Ctx 模块发起 DMA Write CQC 表项请求。

5.3.3.3 eqctxmdata

空间描述

eqctxmdata 空间大小：EQ 最多支持 32 个。mthca 每个条目长度为 16*4=64 字节，除去 error_eqn、reserved 暂不实现的字段，该版设计 EQC 条目的有效长度 **48** 字节，***该设计欲支持 32 个，即 2^5 个 EQ***，按照 2^5 个 64B（48B 按照 2 的幂次取 64B）的 EQC 表项计算，需要的 ICM 空间为 2^5*2^6=2^11B，即 2KB。按 4KB 页大小计算，***eqctxmdata 表项仅需要 1 个宽度为 53（52 位地址，1 位有效位）的寄存器即可***。

功能描述：

（1） 根据 ceu_parser 分发的 ceu 请求，完成 eqctxmdata 的表创建、表无效、表项填写、表项无效。

接口见 3.1

（2） 响应 ceu_parser 发起的查询 eqctmdata 表项的请求，即 CEU 创建、修改、无效、EPC 表项的请求，需要获取物理地址实现对主机内存的写操作。

查询方法：*获取寄存器值*

（3） 根据 ceu_parser 的请求，以及查询结果，向 DMA Write Ctx 模块发起 DMA Write EQC 表项请求。

5.3.4 状态机设计

RCV_REQ：根据 ceu_parser 和 key_qpc_data 请求 fifo 的是否为空读取请求，优先选择 ceu_parser 的请求读取，其次选择 key_qpc_data 的请求处理。

MDT_PROC：分为读、写 qpctxmdata、cqctxmdata、eqctxmdata 三种 RAM，根据 ceu_parser 或者 key_qpc_data 请求的命令，完成对元数据的操作。具体有：

1）引起读 qpctxmdata 操作有：（1）ceu 读 QPC；（2）ceu 写整个 QPC；（3）key_qpc_data 写 state、PSN 的操作

2）引起写 qpctxmdata 操作有：（1）ceu 进行 MAP_ICM_EN 操作，valid 置位；（2）ceu 进行 MAP_ICM_DIS 操作，valid 位进行清零；（3）ceu 进行 WR_ICMMAP_EN，存放 qpc_base；（4）ceu 进行 WR_ICMMAP_DIS，将所有 qpctxmdata 重置

3）引起读 cqctxmdata 操作有：（1） ceu 写整个 CQC、修改 CQC、无效 CQC

4）引起写 cqctxmdata 操作有：（1）ceu 进行 MAP_ICM_EN 操作；（2）ceu 进行 MAP_ICM_DIS 操作，valid 位进行清零；（3）ceu 进行 WR_ICMMAP_EN，存放 cqc_base；（4）ceu 进行 WR_ICMMAP_DIS，将所有 cqctxmdata 重置

5）引起读 eqctxmdata 操作有：（1） ceu 写整个 EQC、修改 EQC、无效 EQC

6）引起写 eqctxmdata 操作有：（1）ceu 进行 MAP_ICM_EN 操作；（2）ceu 进行 MAP_ICM_DIS 操作，valid 位进行清零；（3）ceu 进行 WR_ICMMAP_EN，存放 eqc_base；（4）ceu 进行 WR_ICMMAP_DIS，将所有 eqctxmdata 重置

DMA_PROC：需要进行 DMA 请求的操作，向 dma_read_ctx、dma_write_ctx 模块发起对应的 DMA 读、写请求。具体的操作有：

1）引起 DMA 读操作的有：（1）ceu 读 QPC；

2）引起 DMA 写操作的有：（1）ceu 写整个 QPC；（2）ceu 写整个 CQC、修改 CQC、无效 CQC；（3）ceu 写整个 EQC、修改 EQC、无效 EQC ；（4）key_qpc_data 写 state、PSN 的操作

5.4 key_qpc_data

5.4.1 模块功能

用于存储数据通信过程中关键的上下文信息。具体数据格式内容如 4.1.2 中介绍。该模块功能如下：

（1）根据 request controller 模块的调度结果，从对应的请求通道中，读取 cmd 请求并在 resp 通道中返回请求命令。**注：对于 RDMA 引擎的 RTC、RRC、EE 写上下文的请求不返回命令；**

（2）若是写上下文请求，还需要读取 context 负载通道里的内容，以备将数据写入存储区，对 RDMA 引擎写请求，还会产生对 ctxmdata 模块的请求，以获取物理地址进而产生 DMA 请求；

（3）若是 RDMA 引擎发起的读请求，则需要通过 cxt 通道将读取到的数据返回到对应的请求模块中。

5.4.2 内部数据读写

该部分中的所有数据使用单端口 RAM IP 存储。

RDMA 引擎数据访问读写请求（CEU 均包含读写请求权限，在此不统计 CEU 命令）

Req type	Req Op	module
RD_QP_CTX	RD_QP_NPST	RTC
RD_QP_CTX	RD_QP_SST	Doorbell processing
RD_QP_CTX	RD_QP_RST	Execution Engine
RD_QP_CTX	RD_QP_STATE	WQE Parser
RD_QP_CTX	RD_ENCAP	FrameEncap
RD_CQ_CTX	RD_CQ_CST	RRC

Key info	位宽	Req type	Req Op	Module	W/R
QPC Key Data					
state	4	RD_QP_CTX	RD_QP_STATE	WQE Parser	R
		RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_QP_CTX	RD_QP_RST	Execution Engine	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
		WR_QP_CXT	WR_QP_NPST	RTC	W
		WR_QP_CXT	WR_QP_EPST	Execution Engine	W
		WR_QP_CXT	WR_QP_UAPST	RRC	W
servtype	8	RD_QP_CTX	RD_QP_SST	Doorbell processing	R
mtu_msgmax	8	RD_QP_CTX	RD_QP_SST	Doorbell processing	R
		RD_QP_CTX	RD_QP_RST	Execution Engine	R
rnr_retry	8	RD_QP_CTX	RD_QP_RST	Execution Engine	R

remote_qpn	32	RD_QP_CTX	RD_QP_SST	Doorbell processing	R
		RD_QP_CTX	RD_QP_RST	Execution Engine	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
port_pkey 端口号[26:24] pkey_index[6:0]	32	RD_QP_CTX	RD_QP_SST	Doorbell processing	R
		RD_QP_CTX	RD_ENCAP	FrameEncap（Port）	R
		RD_QP_CTX	RD_ENCAP	FrameEncap（Pkey）	R
		RD_QP_CTX	RD_QP_RST	Execution Engine	R
QP pd	32	RD_QP_CTX	RD_QP_SST	Doorbell processing	R
		RD_QP_CTX	RD_QP_RST	Execution Engine	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
sl_tclass_flowlabel	32	RD_QP_CTX	RD_ENCAP	FrameEncap	R
next_send_psn (Next PSN)	32	RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
		WR_QP_CXT	WR_QP_NPST	RTC	W
qp_cqn_send	32	RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
snd_wqe_base_lkey	32	RD_QP_CTX	RD_QP_SST	Doorbell processing	R
last_acked_psn (UnAckedPSN)	32	RD_CQ_CTX	RD_CQ_CST	RRC	R
		WR_QP_CXT	WR_QP_UAPST	RRC	W
rnr_nextrecvpsn (Expected PSN)	32	RD_QP_CTX	RD_QP_RST	Execution Engine	R
		WR_QP_CXT	WR_QP_EPST	Execution Engine	W
rcv_wqe_base_lkey	32	RD_QP_CTX	RD_QP_RST	Execution Engine	R
rq_entry_sz_log	8	RD_QP_CTX	RD_QP_RST	Execution Engine	R
sq_entry_sz_log	8	RD_QP_CTX	RD_QP_SST	Doorbell processing	R
		RD_QP_CTX	RD_QP_STATE	WQE Parser	R
smac；slid(smac[15:0])	48	RD_QP_CTX	RD_ENCAP	FrameEncap	R
dmac；dlid(dmac[15:0])	48	RD_QP_CTX	RD_QP_NPST	RTC（dlid）	R
		RD_QP_CTX	RD_QP_RST	EE（dlid）	R
		RD_QP_CTX	RD_ENCAP	FrameEncap	R
		RD_CQ_CTX	RD_CQ_CST	RRC（dlid）	R
sip	32	RD_QP_CTX	RD_ENCAP	FrameEncap	R
dip	32	RD_QP_CTX	RD_ENCAP	FrameEncap	R
snd_wqe_length （SQ Length）	32	RD_QP_CTX	RD_QP_SST	Doorbell processing	R
		RD_QP_CTX	RD_QP_STATE	WQE Parser	R

cqn_rcv	32	RD_QP_CTX	RD_QP_RST	Execution Engine	R
rcv_wqe_length（RQ Length）	32	RD_QP_CTX	RD_QP_RST	Execution Engine	R
CQC Key Data					
cq_sz_log（send）	8	RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
cq_pd（send）	32	RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
cq_lkey（send）	32	RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
cq_sz_log（recv）	8	RD_QP_CTX	RD_QP_RST	Execution Engine	R
cq_pd（recv）	32	RD_QP_CTX	RD_QP_RST	Execution Engine	R
cq_lkey（recv）	32	RD_QP_CTX	RD_QP_RST	Execution Engine	R
EQC Key Data					
eq_sz_log（send）	8	RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
eq_pd（send）	32	RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
eq_lkey（send）	32	RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
eqn（send）	8	RD_QP_CTX	RD_QP_NPST	RTC	R
		RD_CQ_CTX	RD_CQ_CST	RRC	R
eq_sz_log（recv）	8	RD_QP_CTX	RD_QP_RST	Execution Engine	R
eq_pd（recv）	32	RD_QP_CTX	RD_QP_RST	Execution Engine	R
eq_lkey（recv）	32	RD_QP_CTX	RD_QP_RST	Execution Engine	R
eqn（recv）	8	RD_QP_CTX	RD_QP_RST	Execution Engine	R

5.4.3 接口定义

对接模块	信号	In/Out	位宽
request controller	selected_channel	In	8
request controller	receive_req	Out	1
ceu_parser	cxtdata fifo1	In	384
ceu_parser	cxtdata fifo2	In	384
ceu_parser	cmd fifo	In	35

Doorbell Processing	cmd fifo	In	128
WQE Parser	cmd fifo	In	128
Requester Trans Control	cmd fifo	In	128
Requester Recv Control	cmd fifo	In	128
Execution Engine	cmd fifo	In	128
Doorbell Processing	resp fifo	Out	128
WQE Parser	resp fifo	Out	128
Requester Trans Control	resp fifo	Out	128
Requester Recv Control	resp fifo	Out	128
Execution Engine	resp fifo	Out	128
Requester Trans Control	cxt fifo	In	128
Requester Recv Control	cxt fifo	In	128
Execution Engine	cxt fifo	In	128
Doorbell Processing	cxt fifo	Out	256
WQE Parser	cxt fifo	Out	128
Requester Trans Control	cxt fifo	Out	192
Requester Recv Control	cxt fifo	Out	256
Execution Engine	cxt fifo	Out	320
Frame Encap	cxt fifo	Out	256
ctxmdata	req_ctx fifo	Out	128

5.4.4 状态机设计

RD_REQ：根据 request_controller 的调取选择 ceu_parser 或者 RDMA 引擎子模块的请求 fifo 读取请求命令。若是 RDMA 或者 CEU 写请求，则进入 DATA_WR 状态；若是 RDMA 的读请求，则拉高对应 RAM 的读使能信号，然后进入 RESP_OUT 状态。

DATA_WR：对需要进行上下文关键数据写入的请求，执行关键上下文数据的写入更新。若是 CEU 的写请求，则进入 RD_REQ 状态；若是 RDMA 写请求，则进入 RESP_OUT 状态。具体有：

1）ceu_parser：（1）写整个 QPC 引起的写 384 位关键上下文信息；（2）ceu 写整个 CPC 引起的写 CQ_lkey；

2）RDMA RTC：（1）写 NextPSN & QP state（2）暂存数据负载，以备 RESP_OUT 状态发起 ctxmdata 的请求

3）RDMA RRC：（1）写 UnAckedPSN、QP State（2）暂存数据负载，以备 RESP_OUT 状态发起 ctxmdata 的请求

4）RDMA EE：（1）写 Expected PSN、QP State;（2）暂存数据负载，以备 RESP_OUT

状态发起 ctxmdata 的请求

RESP_OUT：对 RDMA 请求返回 cmd 请求响应；对 RDMA 读请求返回 ctx 数据响应；对 RDMA 引擎写请求，产生对 ctxmdata 模块的请求，以获取物理地址进而产生 DMA 请求。

5.5 readctx

5.5.1 模块功能

用于发起 DMA 读请求。接收 ctxmdata 模块发起的 DMA 读请求，并将读请求以 axis 的格式传递给 DMA 引擎。并且针对 CEU 发起的读 QPC 命令，将整个 QPC 封装成对应的接口格式，传递给 CEU 模块。

5.5.2 接口定义

对接模块	通道	I/O	位宽	字段
ctxmdata	req fifo	In	108	Addr(64)+Len(12)+QPN(32)
CEU	axis ceu_rsp	Out	128tuser; 256tdata	见表 3.1
DMA Engine	axis dma_cm_rd_req	Out	128tuser; 256tdata	{32'b0,64addr,20'b0,12length}; {256'b0)
DMA Engine	axis dma_cm_rd_rsp	In	128tuser; 256tdata	{32'b0,64addr,20'b0,12length}; {context data}

5.5.3 状态机设计

该模块未实现，当接收到 ctxmdata 的 dma 读请求时，只存在一种情况，即 CEU 引起的读 QPC 整个条目的请求。

模块内部，实现了一个读请求元数据备份队列（first_word_full_through），在将 dma 读请求发给 DMA 引擎时，同时将读请求的元数据{type，opcode、QPN}暂存到读请求元数据备份队列中。在接收到 DMA 引擎的读响应数据时，将读请求元数据队列中的信息添加到响应 CEU 的 AXIS header 中，在 DMA 引擎同一个读响应的最后一拍（last 信号拉高）时，读使能读请求元数据备份队列。

5.6 writectx

5.6.1 模块功能

用于发起 DMA 写请求。接收 ctxmdata 模块发起的 DMA 写请求；若是 CEU 的写请求，需要从 ceu_parser 的输出通道中获取要写回的数据；若是 key_qpc_data 模块的写请求，则需要从 key_qpc_data 获取要写回的数据。

5.6.2 接口定义

对接子模块	通道	In/Out	位宽	字段
ctxmdata	req_ctx fifo	In	128	Type4+ Opcode4+ Source3+Addr64+data53
ceu_parser	ctxdata fifo	In	256	context payload
DMA Engine	axis dma_cm_wr_req	Out	128tuser; 256tdata	{32'b0,64addr,20'b0,12length}; {256'context data)

5.6.3 状态机设计

PARSE_REQ：获取 ctxmdata 的 dma 写请求，解析该请求是否含有额外的数据负载。具体有 3 种情况：

1）ceu_parser：（1）写整个 QPC、CQC、EQC 请求，分别含有 6/2/2 个数据负载。

2）ceu_parser：（1）修改 EQC mask 数据，修改的 flags 数据位于请求包头中；（2）失效 CQC、EQC，无数据负载，但是要写 0

3）Key_qpc_data：（1）写 NextPSN & QP state；（2）写 ExpectPSN & QP state；（3）写 UnackedPSN & QP state。这三种情况皆没有额外的数据负载，数据皆在请求包头中，从包头中提取出数据负载，生成 dma 写请求即可。

REQ_OUT：根据解析出来的结果，同请求包头中提取数据，发起代码写请求；读取额外的数据负载，根据解析出来的数据负载的数量和长度（从请求包头的 type、opcode 字段可以判断），生成 dma 写请求