ICT NCIC 系统互连组

HCA VirtToPhys 设计文档

V5.0

Revision History

Date	Author	Comment
/7/2020	Ma Xiaoxiao	第一版 VirtToPhys 独立的硬件架构设计文档
/7/2020	Ma Xiaoxiao	补充了 Cache、TPT 查询、TPT 数据结构、Request Scheduler 状态机
/7/2020	Ma Xiaoxiao	修改了本模块和 RDMA 引擎的消息格式，TPT 翻译修改为以单个 sge 为单位进行，修改了 Request Scheduler 状态机
12/9/2020	Ma Xiaoxiao	细化了 CEU Parser、RDMA 引擎四个 Channel 的各自 FIFO 接口、Request Scheduler 调度机制、TPT Cache 部分的设计。
19/9/2020	Ma Xiaoxiao	细化了 TPTMetaData 部分的逻辑设计。
25/9/2020	Ma Xiaoxiao	补充了 TPTMetaData 部分内部数据分布、接口、状态机
10/2020	Ma Xiaoxiao	补充了 TPT Cache 部分内部逻辑，以及与 TPTMetaData 连接，DMA 引擎连接
11/2020	Ma Xiaoxiao	补充了 mpt_ram 实现逻辑，以及 mpt_ram_ctl 逻辑
12/2020	Ma Xiaoxiao	补充了 mtt_ram 实现逻辑，以及 mtt_ram_ctl 逻辑
01/2021	Ma Xiaoxiao	补充了 dma_read_data、dma_write_data 实现逻辑
16/01/2021	Ma Xiaoxiao	结合 mpt_ram 的设计，修改了 dma_read_ctx 实现逻辑 对应 D:\HPC_LAB\Lab_Project\2019IB_NIC\Design\Project\VirtToPhys\source_01.16 代码版本
22/11/2021	Ma Xiaoxiao	修改 mtt 访问网络数据读写死锁问题
21/04/2022	Ma Xiaoxiao	修改 dma_read_data 模块，将 RQ WQE（即 RDMA Engine 的 RWM 通道读 wqe 的请求）的 DMA 读通道与其他数据读请求的通道分离。增加了一组 DMA 读通道。
15/05/2022	Ma Xiaoxiao	将整个 v2p 的架构设计进行了修改，具体而言将 RDMA Engine 的 7 个通道的请求分为了三种 Read WQE、Read DATA、Write。分别对请求进行轮训的调度和 MPT 查询处理，对 MPT 查询结果分为三种 Read WQE、Read DATA、Write 通道，经 mtt_req_scheduler 调度后由 mtt_ram_ctl 处理，进而分为三种 Read WQE、Read DATA、Write 通道进行 DMA 请求处理。

18/08/2022	Ma Xiaoxiao	将整个 v2p 的地址检查和地址翻译查询通路增加了物理地址直接写请求（写 EQE）的支持。具体而言，在 mpt_ram_ctl、mpt_ram 模块接口之间增加了物理地址仿存的标志接口 mpt_eq_addr；mtt_ram_ctl、mtt_ram 模块接口之间增加了物理地址仿存的标志接口 mtt_eq_addr。所有的调度机制不点，物理地址的查询直接提取 RDMA 请求中原来虚拟地址的字段即可。（修改标注为橘黄色衬底）
12/16/2022	Ma Xiaoxiao	全文更新用于京兆 2

目录

1 VritToPhys 模块功能

VritToPhys 管理的内容分为两部分：

（1）ICM MetaData 中的 TPT（MPT、MTT）部分，即 TPTMetaData，用于 MPT、MTT 表项物理地址的查询；

（2）ICM 中的 TPT 部分即 MPT 与 MTT 表的 Cache，该模块对内维护 Cache 的更新，对外将 RDMA 引擎提交的 DMA 请求进行解析，查询访问权限，将虚拟地址转换为物理地址，然后发起内存的读、写操作。

具体功能包括：

1、响应 CEU 模块发起的请求，包括：

- 响应 CEU 发起的写、无效 ICM MetaData 中的 TPT 部分，即 TPTMetaData；
- 响应 CEU 发起的写、无效 TPT 条目。包括命中 Cache 时直接更新 Cache 中的条目，以及不命中时先写到 Cache，完成 Cache 操作后更新 host 端的 TPT 表；

2、响应 RDMA 引擎模块发起的请求，包括：

- 响应 RDMA 引擎中的 Doorbell Processing 模块，根据 lkey 等信息（Memory key、Virtual Addr Offset、Length、Access Flags、PD）翻译物理页面地址、查询访问权限，错误时返回错误信息；无误时返回成功信息，然后发起读取 SQ WQE 的 DMA 请求，并接收 WQE 后转发给 Doorbell Processing 模块；
- 响应 RDMA 引擎中的 WQE Parser（sq wqe）模块，根据 lkey 翻译物理页面地址，错误时返回错误信息；无误时返回成功信息，然后发起读取 SQ WQE 的 DMA 请求，并接收 WQE 后转发给 WQE Parser（sq wqe）模块；
- 响应 RDMA 引擎中的 WQE Parser（network data）模块，根据 lkey 翻译物理页面地址，错误时返回错误信息；无误时返回成功信息，然后发起读取数据负载的 DMA 请求，并接收数据后转发给 WQE Parser（network data）模块；
- 响应 RDMA 引擎中 RTC 模块：完成 CQE 的 DMA 写；
- 响应 RDMA 引擎中 RRC 模块：（1）根据 sge 翻译物理页面地址，若出现异常，返回异常信息；（2）若无异常，发起 DMA 写数据负载请求，将网络数据写入主机内存；
- 响应 RDMA 引擎中 Execution Engine 模块：（1）根据 cq_lkey 翻译物理页面地址，出现异常，返回异常信息；若无异常，发起写 CQE 的 DMA 请求；（2）根据 eq_lkey 翻译物理页面地址，出现异常，返回异常信息；若无异常，发起写 EQE 的 DMA 请求；（3）根据 lkey 翻译物理页面地址，错误时返回错误信息；无误时返回成功信息，然后发起读取 RQ WQE 的 DMA 请求，并接收 WQE 后转发给 Responder Engine 模块；（4）根据 sge 翻译物理页面地址，若出现异常，返回异常信息；若无异常，返回成功信息，然后发起 DMA 读并接收 DMA 数据后转发给 Responder Engine 模块（用于响应端的 RDMA Read 操作）；（5）

根据 sge 翻译物理页面地址，若出现异常，返回异常信息；若无异常，返回成功信息，然后发起 DMA 写数据请求，将网络数据写入主机内存（用于响应端的 RDMA Write 操作）；

3、VirtToPhys 模块内部数据管理，包括：

- 根据 CEU 发起的操作，写、无效 TPTMetaData；
- 根据 CEU 发起的操作，写、无效 TPT 条目。命中 Cache 时直接更新 Cache 中的条目；不命中时，先写入 Cache 再更新主机内存中的 TPT 表项；
- 根据 RDMA 引擎发起的操作，读、对比 TPT 条目：（1）检查内存访问权限 flags、PD、lkey 有效性、访问数据长度有效性等，出现异常时返回对应的错误代码，匹配成功时则返回正确代码（4 bits）；（2）lkey 命中 Cache 时直接读取 Cache 中的条目；不命中时，读取 TPTMetaData，根据需要替换条目，并从内存取回 TPT 表项到 Cache 后更新；

2 消息格式定义

本设计中自定义了一种消息格式以便 VirtToPhys 模块与其它模块进行数据交互。将请求、查询状态响应通道与数据通路（包括接收或者发送的 256 位宽的数据通道，数据通道纯数据没有包头）分离，其中请求头的详细的消息格式定义如下所示：

表 2.1 VirtToPhys 模块对外消息字段定义

包头格式类型	字段（单位 bit）					位宽	说明
Slave1	**Type**	**Opcode**	**R**	**Addr**	**Data**	**128**	接收 CEU 的 TPTMetaData、TPT 表项写、无效请求
	4	4	24	32option	64 option		
Slave2	**Type**	**Opcode**	**Match-Info**			**256**	接收 RDMA 引擎的 TPT 表项查询请求，用于读 WQE，读、写主存网络数据、写主存 CQE、EQE 数据请求
	4	4	32*6				
Master	**State**					**8**	向 RDMA 引擎返回查询 MPT 表项的状态信息；
	8						

Slave2 具体命令格式（右上为最低位）

+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
Reserved																								OpCode				Type			
Flags																															
Protection Domain																															
Key																															

Virtual Address（low）
Virtual Address（high）
Length
Reserved

2.1 字段介绍

Type：按照操作对象读、写 TPTMetaData、TPT 表项、以及是否需要返回数据分类；详细类别见下文 2.2。

Opcode：对应不同 Type 对象的读、写、无效、修改操作；详细类别见下文 2.2。

Addr：请求要访问的 TPTMetaData、TPT 表项资源的 index（MTT index，MPT index，ICM 虚拟地址）。

Match-Info：用于查询 MPT、MTT 表项的数据，1 个 Match-Info 字段包含如下几个字段，由于根据 Doorbell 查询 WQE 是只知道偏移，因此，需要在 **Flags 中【27：26】设置标志位表明是绝对虚拟地址方式还是偏移虚拟地址查询方式**。其余字段与 mpt 表项中字段的分布一致，见 5.7.2

字段	**Flags**	**PD**	**LKey**	**V-Addr**	**Length1**
长度	32 bits	32 bits	32 bits	64 bits	32 bits

Flags 字段定义：

MPT 表项属性标志位

MTHCA_MPT_FLAG_SW_OWNS 【31:28】

ABSOLUTE_ADDR【27】

RELATIVE_ADDR【26】

MTHCA_MPT_FLAG_MIO 【17】

MTHCA_MPT_FLAG_BIND_ENABLE【15】

MTHCA_MPT_FLAG_PHYSICAL【9】

MTHCA_MPT_FLAG_REGION【8】

MPT 表项访问权限标志位

IBV_ACCESS_LOCAL_WRITE【0】

IBV_ACCESS_REMOTE_WRITE【1】

IBV_ACCESS_REMOTE_READ【2】

IBV_ACCESS_REMOTE_ATOMIC【3】

IBV_ACCESS_MW_BIND【4】

IBV_ACCESS_ZERO_BASED【5】

IBV_ACCESS_ON_DEMAND【6】

State：用于描述根据 Match-Info 查询 MPT 表项时，信息匹配检查的状况。暂且提供如

下几种，其他新增错误类型可预留。

State	说明	数值
PD_ERR	PD 错误	8'b00000001
FLAGS_ERR	flags 访问权限错误	8'b00000010
KEY_ERR	Key 无效错误	8'b00000100
LENGTH_ERR	访问数据空间大小错误	8'b00001000
SUCCESS	信息匹配成功	8'b00010000

2.2 Type、Opcode 字段详细说明

表 2.2 Type、Opcode 字段含义说明

对接CEU（Slave1包头格式，位于tuser）		
Type	**Opcode**	**说明**
WR_MPT_TPT	WR_MPT_WRITE	CEU向MPT表中写入一个表项，Addr为mpt_index，Data为空，负载位于tdata
	WR_MPT_INVALID	CEU将MPT表中的一个表项无效，Addr为mpt_index，Data为空，同时无效对应的MTT
WR_MTT_TPT	WR_MTT_WRITE	CEU向MTT表中写入一系列表项，Addr为mtt_num，Data为mtt_start_index，负载位于tdata
	WR_MTT_INVALID	CEU将MTT表中一系列表项无效
WR_ICMMAP_TPT	WR_ICMMAP_EN	CEU将TPT资源表写入VirtToPhys ，Addr为空，Data为空，另128位负载位于tdata
	WR_ICMMAP_DIS	CEU无效写入的TPT资源表，Addr为空，Data为空
MAP_ICM_TPT	MAP_ICM_EN	CEU将TPT资源映射的物理地址写入VirtToPhys，Addr为chunk_num，Data为空，负载位于tdata
	MAP_ICM_DIS	CEU无效写入的TPT资源表 ，Addr为page_cnt，Data为64位虚地址
对接RDMA引擎的请求包头		
Type	**Opcode**	**说明**
RD_REQ_WQE（Slave2格式）	RD_SQ_FWQE	DB Processing请求SQ第一个WQE
	RD_SQ_TWQE	SQ WQE Processing请求SQ后续WQE
	RD_RQ_WQE	Responder Engine请求RQ WQE
RD_REQ_DATA（Slave2格式）	RD_L_NET_DATA	Requester/Responder Engine本地请求读主机内存网络数据
	RD_R_NET_DATA	Requester/Responder Engine远端请求读主机内存网络数据

WR_REQ_DATA（Slave2格式）	WR_L_NET_DATA	Requester/Responder Engine本地请求向主机内存写网络数据
	WR_R_NET_DATA	Requester/Responder Engine远端请求向主机内存写网络数据
	WR_CQE_DATA	Requester/Responder Engine请求向主机内存写CQE数据
	WR_EQE_DATA	Requester/Responder Engine请求向主机内存写EQE数据
无Type字段（Master格式）	PD_ERR	PD错误
	FLAGS_ERR	flags访问权限错误
	KEY_ERR	Key无效错误
	LENGTH_ERR	访问数据空间大小错误
	SUCCESS	信息匹配成功
数据负载格式：均为256bit的纯数据		

3 模块接口定义

3.1 与 CEU 模块接口

使用 128 位宽的 AXI4-Stream Slave，用于接收 CEU 的 TPTMetaData、TPT 表项写、无效请求，包头消息格式见第 2 部分表 2.1 中 **Slave1**。

表 3.1 VirtToPhys 模块与 CEU 模块接口定义

对接模块	接口类型	消息格式	说明
CEU	AXIS Slave	AXIS Slave1	CEU 对 TPTMetaData、TPT 表项写、无效请求
信号线	内容	消息格式	说明
AXIS-tuser	包头	AXIS Slave1	CEU 对 TPTMetaData、TPT 表项写、无效请求包头
AXIS-tdata	数据负载	128 位纯数据	接收 CEU 写 TPTMetaData、TPT 表项的数据。 包含负载的 Type+Opcode 有： WR_MPT_TPT + WR_MPT_WRITE； WR_MTT_TPT + WR_MTT_WRITE； WR_ICMMAP_TPT + WR_ICMMAP_EN； MAP_ICM_TPT + MAP_ICM_EN；

3.2 与 RDMA 引擎模块接口

与 RDMA 引擎连接的 Channel 共 7 组，每组含有 3/4 组 FIFO。均使用 256 位宽。

表 3.2 所示，使用多组 FIFO 接口。其中 FIFO 读端口接收 RDMA 引擎对 WQE 的读请求，对网络数据的读、写请求，对 CQE、EQE 的写请求，消息格式见第 2 部分表 2.1 中 **Slave2**。FIFO 写端口则向 RDMA 引擎返回 WQE、网络数据、返回 TPT 表匹配查询信息，消息格式见第 2 部分表 2.1 中 **Master**。

表 3.2 VirtToPhys 模块与 RDMA 引擎模块接口定义

对接模块	接口类型	消息格式	说明
Doorbell Processing	FIFO 读	Slave2	接收 RDMA 引擎对 SQ WQE 的读请求
	FIFO 写	Master	向 RDMA 引擎返回 TPT 查询信息
	FIFO 写	256 位纯数据	向 RDMA 引擎返回 SQ WQE
WQE Parser（sq wqe）	FIFO 读	Slave2	接收 RDMA 引擎对 SQ WQE 的读请求
	FIFO 写	Master	向 RDMA 引擎返回 TPT 查询信息
	FIFO 写	256 位纯数据	向 RDMA 引擎返回 SQ WQE
WQE Parser（network data）	FIFO 读	Slave2	接收 RDMA 引擎对数据的读请求
	FIFO 写	Master	向 RDMA 引擎返回 TPT 查询信息
	FIFO 写	256 位纯数据	向 RDMA 引擎返回数据
Requester TransControl	FIFO 读	Slave2	接收 RDMA 引擎对完成事件的写请求
	FIFO 写	Master	向 RDMA 引擎返回 TPT 查询信息
	FIFO 读	256 位纯数据	向内存写完成事件
Requester RecvControl	FIFO 读	Slave2	接收 RDMA 引擎对数据的写请求
	FIFO 写	Master	向 RDMA 引擎返回 TPT 查询信息
	FIFO 读	256 位纯数据	向内存写 Read 响应包的数据
Execution Engine（ee）	FIFO 读	Slave2	接收 RDMA 引擎对网络数据的读、写请求
	FIFO 读	256 位纯数据	接收 RDMA 引擎的网络数据
	FIFO 写	Master	返回 TPT 查询信息

	FIFO 写	256 位纯数据	向 RDMA 引擎返回网络数据
Execution Engine（rwm）	FIFO 读	Slave2	接收 RDMA 引擎对 RQ WQE 的读请求
	FIFO 写	Master	向 RDMA 引擎返回 TPT 查询信息
	FIFO 写	256 位纯数据	向 RDMA 引擎返回 RQ WQE

3.3 与 DMA 引擎接口

使用标准的 AXIS 接口。共 4 组读（网络数据、WQE、MPT、MTT 各一组）、3 组写（网络数据/CQE/EQE 单独一组，MPT、MTT 各一组）接口。

其中**写数据时，需要完成数据的拆分工作**，在 tuser 中仅写一个物理地址的描述符，同时在 tdata 中写入对应大小的数据，不需要考虑地址对齐问题，DMA 引擎内部进行处理，提供字节索引的地址、数据长度即可，虚实地址转换模块内部完成跨页引起的空泡问题；**读数据时**，接收到的数据为已经排序完成的数据，即获得的数据与请求顺序一致，本模块**需要完成数据空泡的拼接工作**。

表 3.3 Virt2phys 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	out	提交 DMA 读 MPT 表项请求（tuser）
	AXI4-Stream Slave	in	接收 DMA 读 MPT 表项（tdata）
读 2	AXI4-Stream Master	out	提交 DMA 读 WQE 请求（tuser）
	AXI4-Stream Slave	in	接收 DMA 读 WQE 数据（tdata）
读 3	AXI4-Stream Master	out	提交 DMA 读 MTT 表项请求（tuser）
	AXI4-Stream Slave	in	接收 DMA 读 MTT 表项（tdata）
读 4	AXI4-Stream Master	out	提交 DMA 读网络数据请求（tuser）
	AXI4-Stream Slave	in	接收 DMA 读网络数据（tdata）
写 1	AXI4-Stream Master	out	提交 DMA 写 MPT 表项请求（tuser）
		out	提交 DMA 写 MPT 表项数据（tdata）
写 2	AXI4-Stream Master	out	提交 DMA 写 CQ、EQ、网络数据请求（tuser）
		out	提交 DMA 写 CQ、EQ、网络数据（tdata）
写 3	AXI4-Stream Master	out	提交 DMA 写 MTT 表项请求（tuser）
		out	提交 DMA 写 MTT 表项数据（tdata）

4 总体框架

VirtToPhys 模块内部总体架构如图 4-1 所示，CEU 和 RDMA 引擎中的模块通过不同的 Channel 与 VirtToPhys 进行交互，读、写、无效虚实地址转换关系及 TPTMetaData。

CEU 操作分为两大类由 Parser 区分：(1)写、无效 TPTMetaData，具体命令有 INIT_HCA、CLOSE_HCA、MPI_ICM、UNMPI_ICM，由 TPTMetaData 作出相应的元数据更新；(2) 写、无效 TPT 表项，具体命令有 HW2SW_MPT、SW2HW_MPT、WRITE_MTT，更新 Cache 中的 TPT，然后由 HostOp 发起 DMA 操作写、无效内存中的 TPT。

RDMA 引擎五大类操作：（1）读取 WQE 相关的模块（Doorbell Processing、SQ WQE Processing/Responder Engine）请求描述符的物理地址，返回 WQE；（2）Requester/Responder Engine 模块请求读本地数据的 sge 对应的物理地址，返回 DMA 网络数据；（3）Requester/Responder Engine 模块请求写入本地数据的 sge 对应的物理地址，接收网络数据并 DMA 写入主机内存；(4) Requester/Responder Engine 模块根据 cq_lkey 翻译物理页面地址，发起写 CQE 的 DMA 请求。(5) Requester/Responder Engine 模块根据 eq_lkey 翻译物理页面地址，发起写 EQE 的 DMA 请求。

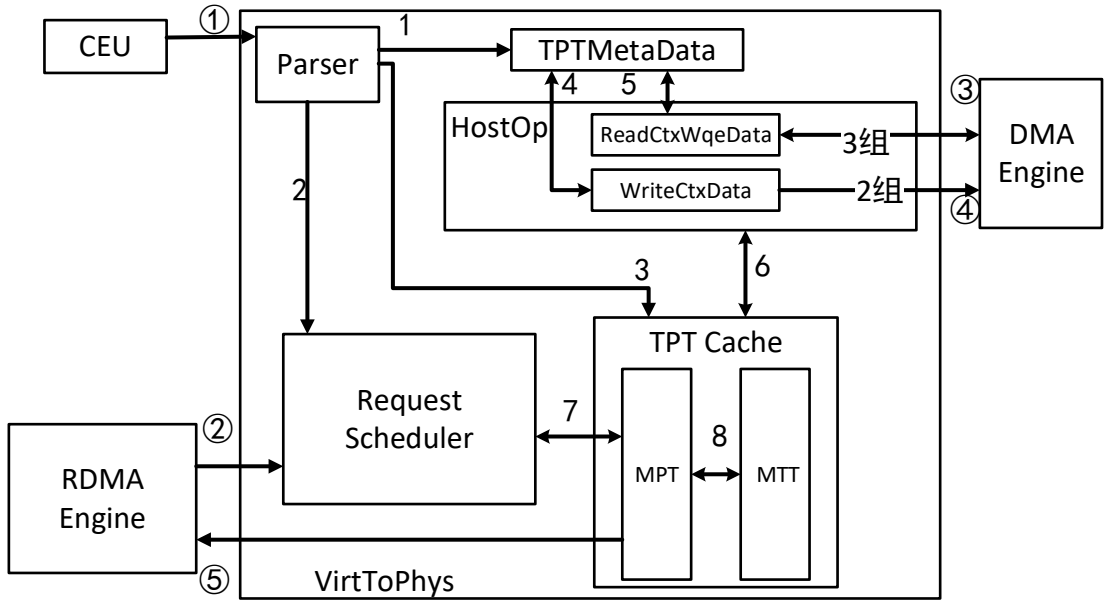


图 4-1 VirtToPhys 总体架构图

4.1 外部模块通路

① 以 2 节表 2.1 Slave1 的消息格式，CEU 发起的 TPTMetaData 及 TPT 表项的写、失效操作；

② 以 2 节表 2.1 Slave2 的消息格式，RDMA 引擎 Doorbell Processing、WQE Parser（sq wqe）、WQE Parser （network data）、Requester TransControl、Requester RecvControl、 Execution Engine（ee）、Execution Engine （rwm）发起内存读写操作，获取 TPT 查询状态信息，完成数据读写。

③ 根据 CEU 或者 RDMA 引擎的请求，VirtToPhys 发起 DMA 读请求，并接收 DMA 返回的数据和读完成描述符；

④ 根据 CEU 或者 RDMA 引擎的请求，VirtToPhys 发起 DMA 写请求并向内存写数据；

⑤ 返回 TPT 查询状态信息、读取的网络数据/WQE。

4.2 内部模块通路

1、对于 CEU 发起的 TPTMetaData 及 TPT 表项的写、失效操作；Parser 以 2 节表 2.1 Slave1 的消息格式提取对 TPTMetaData 操作交给 TPTMetaData 处理；

2、对于 CEU 发起的 TPTMetaData 及 TPT 表项的写、失效操作；Parser 以 2 节表 2.1 Slave1 的消息格式提取对 MPT 表项的操作，交给 Request Scheduler 模块接管；**注：CEU 发起的 MPT 操作优于 RDMA 引擎发起的 MPT 操作**。

3、对于 CEU 发起的 TPTMetaData 及 TPT 表项的写、失效操作；Parser 以 2 节表 2.1 Slave1 的消息格式提取对 MTT 表项的操作，交给 MTT 模块接管；**注：CEU 发起的 MTT 操作优于 RDMA 引擎发起的 MTT 操作**。

4、HostOp 模块进行 DMA 写 TPT 操作前，获取 TPTMetaData 数据；

5、HostOp 模块进行 DMA 读 TPT 操作前，获取 TPTMetaData 数据；

6-1、Request Scheduler 到 HostOp 上行数据流包括：（1）TPT Cache 替换掉的脏块写回主存；（2）TPT 未命中时，发起 DMA 加载 TPT 请求；（3）根据 TPT 查询到的物理地址将网络数据写到主存；（4）根据 TPT 查询到的物理地址将 CQE、EQE 数据写到主存；（5）根据 TPT 查询到的物理地址发起 WQE、网络数据的 DMA 读请求；

6-2、HostOp 到 Request Scheduler 下行数据流包括：（1）TPT 未命中时，DMA 读的 TPT 表项加载到 TPT Cache；（2）DMA 读取的 WQE、网络数据放到对应的请求通道

7、Request Scheduler 模块将 RDMA 引擎不同模块的请求进行调度，对 TPT 表项进行读、写、失效、信息匹配等。

8、MPT 表项到 MTT 表项的查询。

5 子模块

5.1 CEU Parser

5.1.1 功能

CUE Parser 将 CEU 发起的 TPTMetaData、TPT 表项请求分发到 TPTMetaData 模块、Request Controller 模块、MTT 模块的 FIFO（每个模块两个 FIFO，其一用于请求包头，其一用于可能的数据负载，共 6 个 FIFO 在本模块例化），等待各个模块处理。

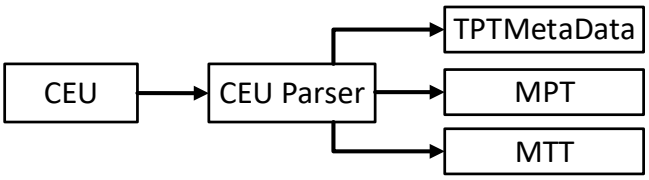


图 5-1 CEU Parser 子模块架构图

5.1.2 接口

5.1.2.1 转发到 TPTMetaData 的请求及负载

请求包头 128 位宽，负载 256 位宽（注：左边高位，右边低位）：

Req 1 header	Seg	type	opcode	R	
	Bit width	4	4	120	
	Value	WR_ICMMAP_TPT	WR_ICMMAP_EN	void	
Req 1 Payload	Bit width	128	56	8	64
	Value	void	mpt_base	log_mpt_sz	mtt_base

Req 2 header	Seg	type	opcode	R
	Bit width	4	4	120
	Value	WR_ICMMAP_TPT	WR_ICMMAP_DIS	void
Req 2 Payload	void			

Req 3 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 3 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			

Req 4 header	Seg	type	opcode	R	num	addr
	Bit width	4	4	24	32	64
	Value	MAP_ICM_TPT	MAP_ICM_DIS	void	page_cnt	virt addr
Req 4 Payload	void					

5.1.2.2 转发到 MPT（经 Request Scheduler 调度）的请求及负载

（注：左边高位，右边低位）：

Req 5 header	Seg	type	opcode	R	addr	R
	Bit width	4	4	24	32	64
	Value	WR_MPT_TPT	WR_MPT_WRITE	void	mpt_index	void
Req 5 Payload	5.7.2 MPT entry 格式,共 56 Byte = 56 * 8，需要 256 位 payload 2 个 cycle					

Req 6 header	Seg	type	opcode	R	addr	R
	Bit width	4	4	24	32	64
	Value	WR_MPT_TPT	WR_MPT_INVALID	void	mpt_index	void
Req 6 No Payload，该命令会在无效 MPT 的同时将对应的所有 MTT 表项无效掉						

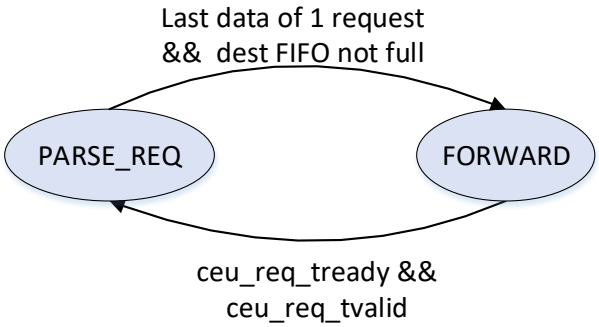
5.1.2.3 转发到 MTT 的请求及负载

（注：左边高位，右边低位）：

Req 7 header	Seg	type	opcode	R	num	index
	Bit width	4	4	24	32	64
	Value	WR_MTT_TPT	WR_MTT_WRITE	void	mtt_num	start_index
Req 7 Payload	5.8.2 MTT entry 格式 64 bit/entry，63:0 entry0; 127:64 entry1 ••• ，位宽 256					

5.1.3 状态机

状态机分为两个状态：



（1） PARSE_REQ：当 ceu_req_tready && ceu_req_tvalid 同时有效，将 ceu 请求的包头（theader）和负载（tdada）分别放到 qv_ceu_req_theader 和 qv_ceu_req_tdata 中，便于下一状态根据 theader 的信息确定是否有负载、要转发的目的 FIFO，下一状态跳转为 FORWARD。

（2） FORWARD：根据 qv_ceu_req_theader 中的信息，将请求包头和负载分别转发到对应的目的请求包头 FIFO 和目的请求负载 FIFO。当 1 个请

求的最后一拍数据到达转发到对应的目的 FIFO 后，下一状态跳转为 PARSE_REQ。

5.2 Request Scheduler

5.2.1 功能

Request Scheduler 完成协调 CEU、RDMA 引擎部件对 MPT 请求的功能

具体实现方式为：

（1）CEU 对 MPT 的更新操作优先级高于其他模块的请求；

（2）使用调度策略从不同的与 RDMA Engine 连接的 ReqFIFO 中读取请求，并更新 Selected Channel Reg，以便 MPT 模块读取对应 Req FIFO，以及 Request Scheduler 决定下一次 Channel 的选择，每次 MPT 读完新的 Selected Channel 信息之后都会将旧的 Selected Channel Reg 中的 Ready 位置零，此时 Request Scheduler 检测到 Ready 为 0 会根据旧的 Selected Channel Reg、现有模块的请求状态、PendingFIFO 的状态决定新的 Selected Channel，该过程参考图 5-3 中绿色箭头

（3）使用 RoundRobin 方式选择 Channel 时要参考 PendingFIFO 中的 Req（Pend Channel Count Reg），不可选择已经有 Req 位于 PendingFIFO 的 Channel，因为这样选择会造成乱序。

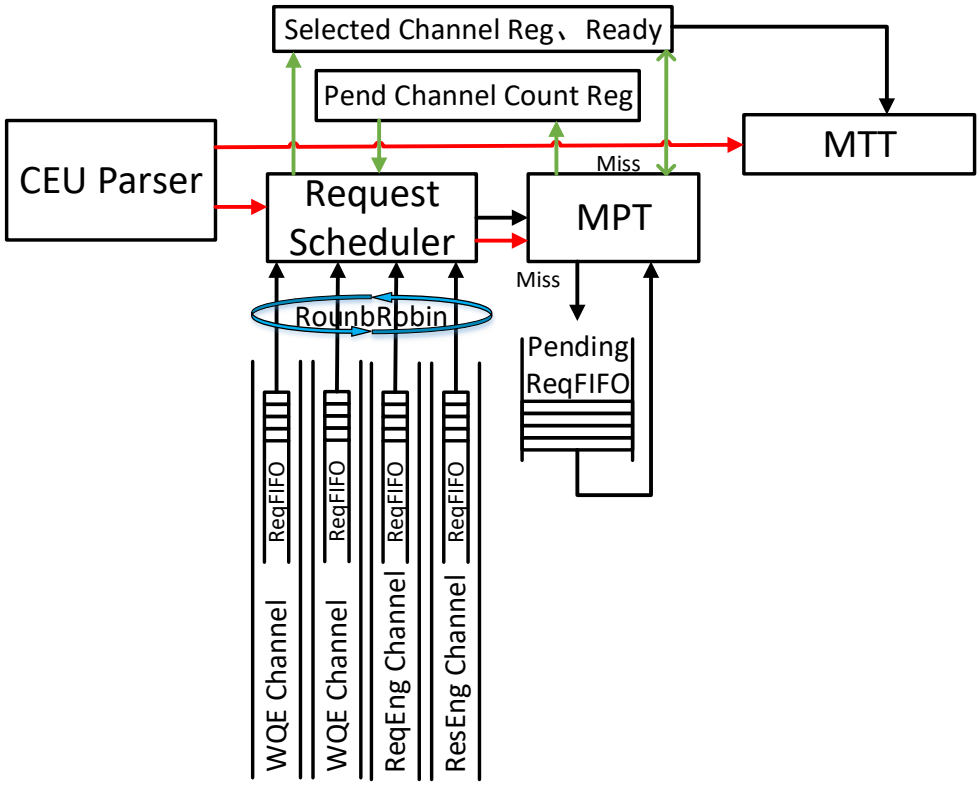


图 5-3 Request Scheduler 与其他模块连接关系

5.2.2 接口

5.2.2.1 与 MPT 模块接口

输出 Selected Channel Reg 格式（9 bit），该寄存器在进行 Req Channel 选择时更新，8 个 Channel 中每次只有 1 个和 Ready 同时为 1，其余 Channel 为 0,。

字段	位	说明
Channel 0	0	CEU
Channel 1	1	Doorbell Processing(WQE)
Channel 2	2	WQE Parser(WQE）
Channel 3	3	WQE Parser(DATA)
Channel 4	4	RequesterTransControl(CQ)
Channel 5	5	RequesterRecvControl(DATA)
Channel 6	6	Execution Engine(RQ WQE)
Channel 7	7	Execution Engine(DATA)
Ready	8	更新 Channel bit 的同时，置 1，表明此时 Reg 已更新，等 MPT 读取；MPT 读取 Req 后会将 Ready 信号置 0，表明此时需要更新

注：由于 request_scheduler 和 mpt 模块都要读写 selected_channel reg，因此增加了 selected_channel_ctl 子模块用于集中控制 selected_channel_reg 的读写。mpt 模块在读取到新的 selected_channe_reg 值之后，需要向该控制模块发送 read_req_already 信号，用以控制 selected_channel_reg 的数值更新。

接收 Pend Channel Count Reg 格式（32 bit），该寄存器组在进行 MPT Miss 或者 TPT 表返回时更新。当 MPT Miss 时，对应的 Channel 计数器+1；当 TPT 表返回时，对应的 Channel 计数器-1。该寄存器组的目的是，当某个 Channel 中有 Req 被挂起在 PendingFIFO 中时，Request Scheduler 便不再调度该 Channel 同的请求，以避免乱序的情况出现。

字段	位	说明
CEU	0:3	CEU 由于 Miss 被挂起的 Req 数量
Doorbell Processing(WQE)	4:7	DB Proc 由于 Miss 被挂起的 Req 数量
WQE Parser(WQE）	8:11	WQE Parser 由于 Miss 被挂起的 Req 数量
WQE Parser(DATA)	12:15	WQE Parser 由于 Miss 被挂起的 Req 数量
RequesterTransControl(CQ)	16:19	RTC 由于 Miss 被挂起的 Req 数量
RequesterRecvControl(DATA)	20:23	RRC 由于 Miss 被挂起的 Req 数量
Execution Engine(RQ WQE)	24:27	Execution Engine 由于 Miss 被挂起的 Req 数量
Execution Engine(DATA)	28:31	Execution Engine 由于 Miss 被挂起的 Req 数量

5.2.2.1 与各个请求模块接口

接收各个请求模块中 ReqFIFO 的 empty 信号，作为是否有请求的依据，用于决策 Selected Channel。

模块	子模块	位宽	in/out	说明
VirtToPhys 内部	CEU Parser	1	in	rd_ceu_req_empty,
RDMA Engine	Doorbell Processing(WQE)	1	in	rd_dbp_req_empty,
	WQE Parser(WQE）	1	in	rd_wp_wqe_req_empty
	WQE Parser(DATA)	1	in	rd_wp_dt_req_empty
	RequesterTransControl(CQ)	1	in	rd_rtc_req_empty
	RequesterRecvControl(DATA)	1	in	rd_rrc_req_empty,
	Execution Engine(RQ WQE)	1	in	rd_ee_req_empty,
	Execution Engine(DATA)	1	in	rd_rqwqe_req_empty,

5.2.3 状态机

该模块无需使用状态机：当 Ready 信号为 0，则按照 RoundRobin 策略，根据旧的 Selected Channel Reg、现有模块的请求 FIFO 是否为空状态、PendingFIFO 的状态决定新的 Selected Channel。

时序图：便于后续修改分析。

5.3 TPTMetaData

5.3.1 功能

TPTMetaData 用于存放 TPT 表项在主机内存中的虚实地址转换关系，由 CEU 对 TPTMetaData 进行修改。

主要功能：

（1）实现 CEU 对 TPTMetaData 的写、无效管理。

（2）对 CEU 发起的 MPT、MTT 表项的写、无效操作，在更新网卡 Cache 的同时，会引发读取对应的 TPTMetaData 中的数据，以获得 TPT 表项在主机内存中的物理地址，然后由 TPTMetaData 直接发起 host 端 TPT 表项的操作。

（3）由于 MPT Cache 替换策略引起的 DMA 写操作，会引发先读取 TPTMetaData 中对应表项的物理地址，然后由 TPTMetaData 直接发起 DMA 写请求。

（4）在处理 RDMA 引擎相关模块的 TPT 相关查询请求，出现的 Cache Miss 时，引发的 DMA 读 host TPT 数据，此时 MPT 模块会发起读取 TPTMetaData 中对应表项的物理地址的请求，然后由 TPTMetaData 直接发起 DMA 读 TPT 表项的请求。

5.3.2 数据格式

TPTMetaData 数据格式，每个条目 52bit 数据+1bit 有效标志位。

ICM_MAP 消息格式

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	virt[i][63:32]																															
04h	virt[i][31:0]																															
08h	page[i][63:32]																															
0Ch	page[i][31:12]																				page_num											

RAM 设计：使用 52 位宽、512 深度的 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 512 深度 1 位宽度的 valid 标志位；

5.3.3 接口

下图为与 TPTMetaData 模块，密切相关的几个模块的重要数据关系，其中：

（1） 黑色线条表示，RDMA 引擎相关模块发起的 TPT 表项相关的操作请求

（2） 紫色线条表示，CEU 发起的 TPTMetaData 相关的写、无效操作

（3） 红色线条表示，CEU 发起的 TPT 表项的写、无效操作，包括网卡 Cache 上的操作，以及引发的读取对应的 TPTMetaData 中的数据，进而发起 host 端 TPT 表项的操作。除此之外，线条 2 还表示可能由于 Cache 替换策略引起的 DMA 写操作（先读取 TPTMetaData 中对应表项的物理地址再发起线条 6 所示的写）

（4） 蓝色线条表示，在处理 RDMA 引擎相关模块的 TPT 相关查询请求，出现的 Cache Miss 时，引发的 DMA 读 host TPT 数据（先读取 TPTMetaData 中对应表项的物理地址再发起线条 5 所示的 DMA 读）。***注意***：MPT Cache Miss 之后，MPT 模块会进行如下操作：1）将对应的 Request 放入 PendingFIFO，以备在 DMA 返回读取的 MPT 表项后，核实 MPT 表项填入 Cache；2）通知 TPTMetaData 模块，查询缺失 MPT 表项的物理地址，然后发起 DMA 读 MPT 请求；3）接收 DMA Read Ctx 的 MPT 数据，查询对应的 MTT 索引地址 4）通知 MTT 模块，由 MTT 模块通知 TPTMetaData 查询对应的 MTT 物理地址，发起 DMA 读 MTT 请求，准备接收来自 DMA 读取的 MPT 表项对应的 MTT 表项；5）MTT 模块接收到 MTT 表项后，处理的 PendingFIFO，更新 Pending Channel Count Reg 的值。***注意：当 MPT 缺失后，发起读 Host MPT 表项时，待获取 MPT 数据后随即发起读该 MPT 表项对应的 MTT 表项***。

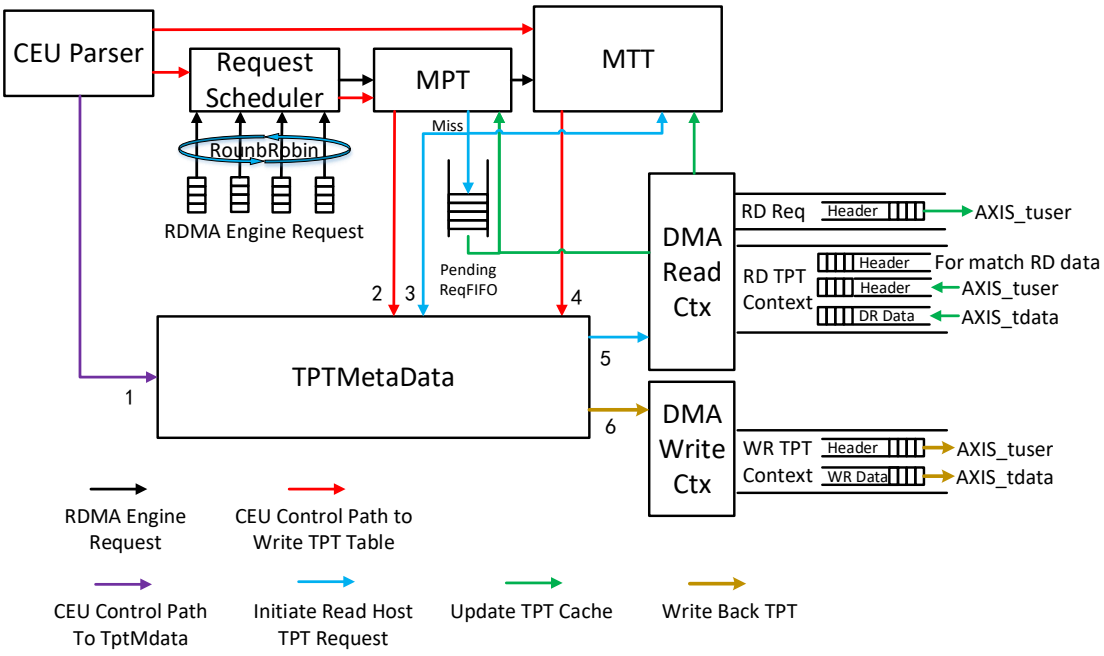


图 5-4 TPTMetaData 与其他模块连接关系

与 TPTMetaData 接口的模块主要有 CEU Parser、MPT、MTT、DMA Read CTX、DMA Write Ctx。

5.3.3.1 CEU Parser 接口

如图 5-4 线条 1 所示，用于 CEU 更新 TPTMetaData 表项

接口格式为请求包头 128 位宽，负载 256 位宽（注：左边高位，右边低位）：

初始化 TPTMetaData 表：INIT HCA

Req 1 header	Seg	type	opcode	R	
	Bit width	4	4	120	
	Value	WR_ICMMAP_TPT	WR_ICMMAP_EN	void	
Req 1 Payload	Bit width	128	56	8	64
	Value	void	mpt_base	log_mpt_sz	mtt_base

无效 TPTMetaData 表： CLOSE HCA

Req 2 header	Seg	type	opcode	R
	Bit width	4	4	120
	Value	WR_ICMMAP_TPT	WR_ICMMAP_DIS	void
Req 2 Payload	void			

写入 TPTMetaData 表项虚实地址映射：ICM MAP

Req 3 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 3 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			

无效 TPTMetaData 表项虚实地址映射：ICM UNMAP

Req 4 header	Seg	type	opcode	R	num	addr
	Bit width	4	4	24	32	64
	Value	MAP_ICM_TPT	MAP_ICM_DIS	void	page_cnt	virt addr
Req 4 Payload	void					

5.3.3.2 MPT 接口

如图 5-4 线条 2、3 所示，用于 MPT 发起查询 TPTMetaData 表项请求，请求包括由 CEU 写 MPT 表项引起的 Host MPT 表项同步写回所需的查询物理地址（线条 2）、MPT 自身 Cache 策略引起的 MPT 表项写回查询物理地址（线条 2）、RDMA 引擎查询 MPT Cache 表项 Miss 引起的读 Host MPT、MTT 表项所需的查询物理地址（线条 3）。

接口格式：请求包头 99 位宽（注：左边高位，右边低位）：

注：MPT 负载为 256 位宽，直接与 DMA Write Ctx 模块对接

Req header	Seg	opcode	number	index
	Bit width	3	32	64
	Value	见下表	写 mpt 时为 1，写 mtt 时为 mtt 数量	mpt_lkey(32) mtt_index(64)

根据 INIT HCA 命令中的提供 mpt_base 字段，加上移位后的 Lkey（左移 MPT 表项的大小）即是 TPTMetaData 虚实地址映射中的虚地址，据此可以查到对应的物理页面。然后根据 page_num 字段、lkey 计算物理页面的偏移。查询 MTT 的过程与此类似。

Opcode 字段	位宽	模快	说明
MPT Read	3	MPT	由于 Cache Miss 读 host MPT 表项
MPT Write	3	MPT	由于 CEU 更新或者 Cache 策略写 host MPT 表项
MPT Invalid	3	MPT	CEU 无效 host MPT 表项
MTT Read	3	**MTT**	由于 Cache Miss 读取到 host MPT 后，读 MTT 表项
MTT Write	3	MTT	由于 CEU 更新写 host MTT 表项
MTT Invalid	3	MTT	CEU 无效 host MPT 表项时，同时无效对应的 MTT 表项

5.3.3.3 MTT 接口

如图 5-4 线条 4 所示，用于 MTT 发起查询 TPTMetaData 表项请求，请求包括由 CEU 写 MTT 表项引起的 Host MTT 表项同步写回所需的查询物理地址（线条 4）。。

接口格式同 5.4.3.2 MPT 接口

注：MTT 负载为 256 位宽，直接与 DMA Write Ctx 模块对接

5.3.3.4 DMA Read Ctx 接口

如图 5-4 线条 5 所示，用于 RDMA 引擎查询 MPT Cache 表项 Miss 引起的读 Host MPT、MTT 表项。

接口格式为请求包头 163 位宽（注：左边高位，右边低位）：

（共 2 个位宽 163 深度 32 的 FIFO 在本模块例化，分别用于 MPT 和 MTT）

Req header	Seg	index	opcode	length	physical addr
	Bit width	64	3	32	64
	Value	req_num	read mpt 001/read mtt 101	长度	mpt/mtt_addr

5.3.3.5 DMA Write Ctx 接口

如图 5-4 线条 6 所示，用于：（1）由 CEU 写 MPT、MTT 表项引起的 Host MPT、MTT 表项同步写回；（2）MPT 自身 Cache 策略引起的 MPT 表项写回

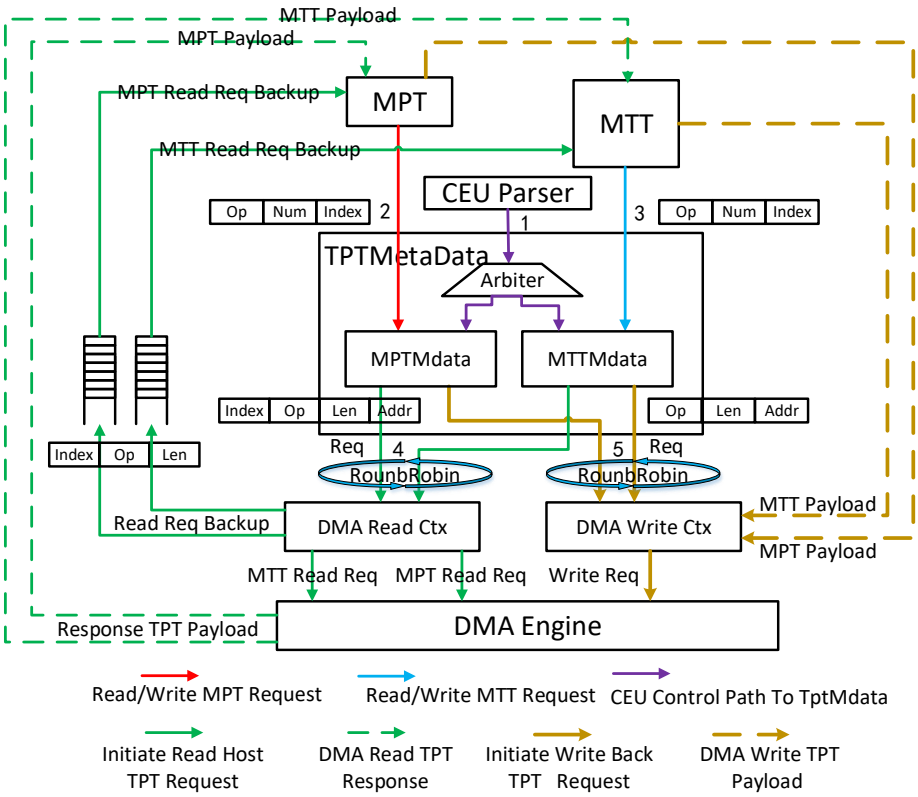
接口格式为请求包头 99 位宽（注：左边高位，右边低位）：

（共 2 个位宽 99 深度 32 的 FIFO 在本模块例化，分别用于 MPT 和 MTT）

Req header	Seg	opcode	length	physical addr
	Bit width	3	32	64
	Value	write mpt 010/write mtt 110	长度	mpt/mtt_addr

5.3.4 内部结构及连接关系

通过 Sort 模块分类，将数据分为 MPTMdata 和 MTTMdata 两部分存储，分别满足 MPT/MTT 查询物理地址，然后发起 DMA 读、写操作（读写请求各两个 FIFO，调度处理由 DMA 读写模块决定）。



5.3.4.1 接口

Sort（编码中的 ceu_tptm_proc）到 MPTMdata 接口：Request header 104 bit FIFO；Payload 256 bit FIFO

Message	Seg	type	opcode	num	Addr	
	Width	4	4	32	64	
INIT HCA	header	WR_ICMMAP_TPT	WR_ICMMAP_EN	void	56 mpt_base	8 log_mpt_sz
	Payload	void				
CLOSE HCA	header	WR_ICMMAP_TPT	WR_ICMMAP_DIS	void		
	Payload	void				
ICM MAP	header	MAP_ICM_TPT	MAP_ICM_EN	chunk_num	void	
	Payload	重复 chunk_num 个 high64virt+low64page(high 52 phy addr； 11:0 page_num 物理页表数量), 127:0 chunk0;255-128 chunk1 ••				
ICM UNMAP	header	MAP_ICM_TPT	MAP_ICM_DIS	page_cnt	virt addr	
	Payload	void				

Sort（编码中的 ceu_tptm_proc）到 MTTMdata 接口：Request header 104 bit FIFO；Payload

256 bit FIFO

Message	Seg	type	opcode	num	Addr
	Width	4	4	32	64
INIT HCA	header	WR_ICMMAP_TPT	WR_ICMMAP_EN	void	mtt_base
	Payload	void			
CLOSE HCA	header	WR_ICMMAP_TPT	WR_ICMMAP_DIS	void	
	Payload	void			
ICM MAP	header	MAP_ICM_TPT	MAP_ICM_EN	chunk_num	void
	Payload	重复 chunk_num 个 64virt+64page(high 52 addr； 11:0 page size), 127:0 chunk0;255-128 chunk1 ••			
ICM MAP	header	MAP_ICM_TPT	MAP_ICM_DIS	page_cnt	virt addr
	Payload	void			

5.3.4.2 mptmdata

功能描述：

（1） 根据 ceu_tptm_proc 分发的 ceu 请求，完成 mptmdata 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 5.4.4.1

（2） 响应 MPT 模块发起的查询 mptmdata 表项请求，请求包括由 CEU 写 MPT 表项引起的 Host MPT 表项同步写回所需的查询物理地址、MPT 自身 Cache 策略引起的 MPT 表项写回查询物理地址、RDMA 引擎查询 MPT Cache 表项 Miss 引起的读 Host MPT 表项所需的查询物理地址

接口格式：请求包头 99 位宽（注：左边高位，右边低位）：

Req header	Seg	opcode	number	index
	Bit width	3	32	64
	Value	见下表	写 mpt 时为 1	**mpt_lkey(32) low 32bit**

字段介绍：

Opcode 字段	位宽	模快	说明
MPT Read	3	MPT	由于 Cache Miss 读 host MPT 表项
MPT Write	3	MPT	由于 CEU 更新或者 Cache 策略写 host MPT 表项
MPT Invalid	3	MPT	CEU 无效 host MPT 表项

查询方法：

ceu 命令涉及到的地址查询：在 mptmdata 实现中，我们使用 virtual addr – mpt_base（从 INIT HCA 请求包头中 mpt_base 后 8 位补 0 获得）获得的 64 位

值，取其中的【20:12】共 9 位，即 512 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 512 深度共 9 位）

mpt 请求中涉及到的地址查询：移位后的 Lkey（***左移 MPT 表项的大小，即 64B，6 位***）减去 INIT HCA 命令中的提供 mpt_base 字段，再取其中的【20:12】位即是 TPTMetaData 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 lkey 的低 12 位计算物理页面的偏移。查询 MTT 的过程与此类似。

（3） 根据 MPT 的请求，以及查询结果，分别向 DMA Write Ctx、DMA Read Ctx 模块发起 DMA Write 或者 DMA Read MPT 表项请求。

（注：左边高位，右边低位）：

DMA Read Ctx Req header	Seg	index	opcode	length	physical addr
	Bit width	64	3	32	64
	Value	req_num	read mpt 001/read mtt 101	长度	mpt/mtt_addr

DMA Write Ctx Req header	Seg	opcode	length	physical addr
	Bit width	3	32	64
	Value	Write mpt	长度	mpt_addr

空间描述

mpt 每个表项大小为 56B，mthca 中 QP 数量的参考值为 2^16，mpt 表项数量的参考值为 2^17，mtt 表象数量的参考值为 2^20。

mptmdata 空间大小：***该设计欲支持 16K，即 2^14 个 QP***，按照 2^15 个 56B（按 64B 计算）的 mpt 表项计算，需要的 ICM 空间为 2^15*2^6=2^21B，即 2MB。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 mpt 数量为 256KB/64B=2^12，若共 2^15 个 mpt，则需要 2^3 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，***mptmdata 表项最多有 2MB/4KB=512 个***。

使用 SingleDualPortRAM 模块例化为 52 位宽、512 深度、地址位宽为 9 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 512 深度 1 位宽度的 valid 标志位；

SingleDualPortRAM 模块时序关系为：写数据时，wr_en、wr_addr、wr_data 同时变化；读数据时，rd_en 拉高之后，下一个时钟获取数据。

5.3.4.3 mttmdata

功能描述：

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

接口见 5.4.4.1

（2） 响应 MTT 模块发起的查询 mttmdata 表项请求，请求包括由 CEU 写 MTT 表项引起的 Host MPT 表项同步写回所需的查询物理地址；MPT 自身 Cache 策略导致的 MPT 表项写回同时引发对应的 MTT 写回查询物理地址；RDMA 引擎查询 MPT Cache 表项 Miss 引起的读 Host MPT 表项返回后，紧接着由 MTT 发起读取对应的 host 中的 MTT 表项所需的查询物理地址

接口格式：请求包头 99 位宽（注：左边高位，右边低位）：

Req header	Seg	opcode	number	index
	Bit width	3	32	64
	Value	见下表	mtt 数量	mtt_index(64)

字段介绍

Opcode 字段	位宽	模快	说明
MTT Read	3	MTT	由于 Cache Miss 读取到 host MPT 后，读 MTT 表项
MTT Write	3	MTT	由于 CEU 更新写 host MTT 表项
MTT Invalid	3	MTT	CEU 无效 host MPT 表项时，同时无效对应的 MTT 表项

查询方法：

根据 INIT HCA 命令中的提供 mtt_base 字段，加上移位后的 Lkey（左移 MTT 表项的大小 8B）即是 TPTMetaData 虚实地址映射中的虚地址，据此可以查到对应的物理页面。然后根据 num 字段、lkey 计算物理页面的偏移。

（3） 根据 MTT 的请求，以及查询结果，分别向 DMA Write Ctx、DMA Read Ctx 模块发起 DMA Write 或者 DMA Read MTT 表项请求。

（注：左边高位，右边低位）：

DMA Read Ctx Req header	Seg	index	opcode	length	physical addr
	Bit width	64	3	32	64
	Value	req_num	read mpt 001/read mtt 101	长度	mpt/mtt_addr

DMA Write Ctx Req header	Seg	opcode	length	physical addr
	Bit width	3	32	64
	Value	Write mpt	长度	mpt_addr

空间描述

mttmdata 空间大小：mtt 每个表项大小为 8B——64 位的物理地址，***该设计欲支持 16K，即 2^14 个 QP***，按照 2^18 个 8B 的 mtt 表项计算，需要的 ICM 空间为 2^18*2^3=2^21B，即 2MB。按照内核态驱动的 ICM 分配机制——mthca_alloc_icm_table（linux-

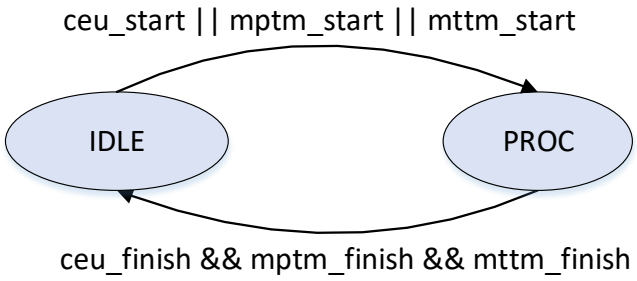
5.4.2\drivers\infiniband\hw\mthca\mthca_memfree.c）。将 ICM 分成 256KB 大小的 chunk，每个 chunk 中含有的 mpt 数量为 256KB/8B=2^15，若共 2^18 个 mpt，则需要 2^3 个 chunk。考虑到 chunk 内的数据可能不是连续的情况，按 4KB 页大小计算，***mttmdata 表项最多有 2MB/4KB=512 个***。

使用 SingleDualPortRAM 模块例化深度为 512，宽度为 52，地址位宽为 9 位的 RAM。

5.3.5 状态机

5.3.5.1 tptmdata 状态机

tptmdata 顶层模块，用于连接 ceu_tptm_proc、mptm、mttm 三个子模块，发起不同子模块的处理信号，当所有模块处理结束后，tptmdata 模块处于 IDLE 状态

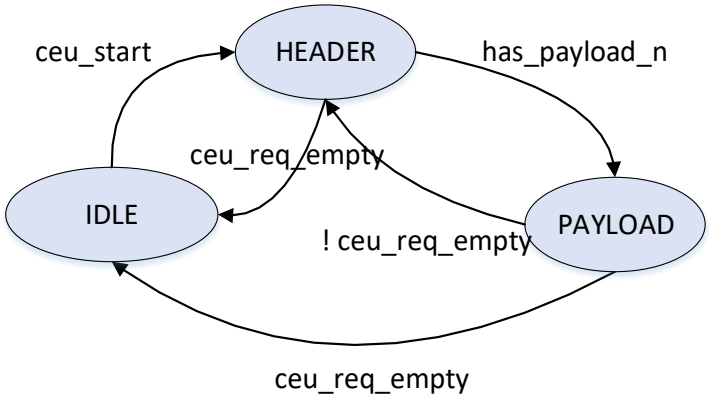


IDLE：空闲状态，当 ceu_parser/mpt/mtt 模块中有模块的请求非空时，进入 PROC 状态；

PROC：处理状态，ceu_start、mptm_start、mttm_start 分别代表三个子模块开始处理对应请求的信号，各个子模块处理各自的请求，当所有的子模块都没有任务处理时，tptmdata 模块再次回归 IDLE 状态。

5.3.5.2 ceu_tptm_proc 子状态机

用于将 CEU 发起的 tptmdata 数据写、失效请求，分为 mptm 和 mttm 两部分数据，分别分发到 mptm、mttm 两个子模块处理。（**注：该模块会比较请求或者负载中虚拟地址的范围，用于判断该请求对应的子模块为 mptm 还是 mttm**）



IDLE：空闲状态，当 ceu_start 信号为高时，进入 HEADER 状态

HEADER：包头处理状态，解析 ceu 发起的 4 种 tptmdata 请求，解析并分别转到对应的

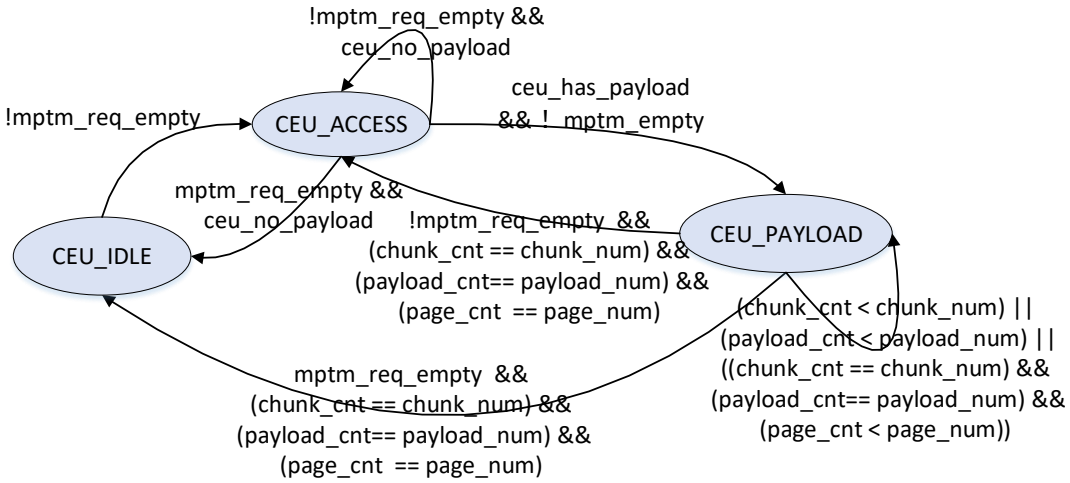
mptm/mttm 子模块。若解析请求包头发现为 MAP_ICM_EN 请求，则进入 PAYLOAD 状态

command	type	opcode	destination
INIT HCA	WR_ICMMAP_TPT	WR_ICMMAP_EN	mptm、mttm
CLOSE HCA	WR_ICMMAP_TPT	WR_ICMMAP_DIS	mptm、mttm
ICM MAP	WR_ICM_TPT	MAP_ICM_EN	比较 addr
ICM UNMAP	WR_ICM_TPT	MAP_ICM_DIS	比较 addr

PAYLOAD：包负载处理状态，用于 MAP_ICM_EN 包，对一定数量的虚实地址转换表项，根据存储的 mptm_base/mttm_base 确定要转发的目的子模块。

5.3.5.3 mptm_proc 子状态机

用于处理 ceu_tptm_proc 请求的状态机



CEU_IDLE：空闲状态。当 mptm_req_proc FIFO 非空时，进入 CEU_ACCESS 状态

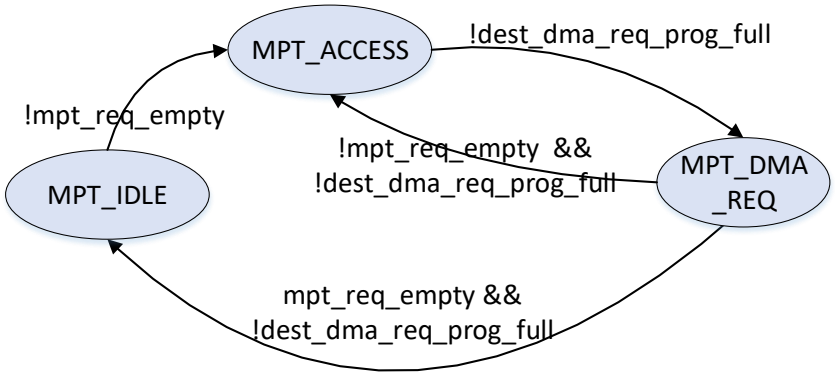
CEU_ACCESS：访问 mptmdata 状态。（1）执行 ceu 发起的 4 种 mptmdata 请求，如下表，若没有包负载（加粗的请求有负载），则进入 IDLE 状态；若有包负载（加粗的请求有负载），则进入 CEU_PAYLOAD；若还有其他请求则继续停留在 CEU_ACCESS 状态；

command	type	opcode
INIT HCA	WR_ICMMAP_TPT	WR_ICMMAP_EN
CLOSE HCA	WR_ICMMAP_TPT	WR_ICMMAP_DIS
ICM MAP	**WR_ICM_TPT**	**MAP_ICM_EN**
ICM UNMAP	WR_ICM_TPT	MAP_ICM_DIS

CEU_PAYLOAD：针对上表中加粗的请求，将包负载根据包头中虚实地址映射数量 chunk_num 信息，放到 mptmdata ram 中。注意负载为 256 位宽，RAM 为 52 位宽（物理地址 52 位、reg 有效位 1 位（有效位使用单独的 reg 数组记录）），因此每个时钟的负载需要两次判断、写入的操作才能完成，对于一次判断加操作而言，每 128 位的负载还要根据 page_num 的数量判断要发起写 mptmdata RAM 的次数，每次写入的 RAM 地址需要根据 chunk 的基地

址和 page_num 计算，写入的物理地址需要根据 page_cnt 和 chunk 的物理计算。

用于处理 mpt 请求的状态机



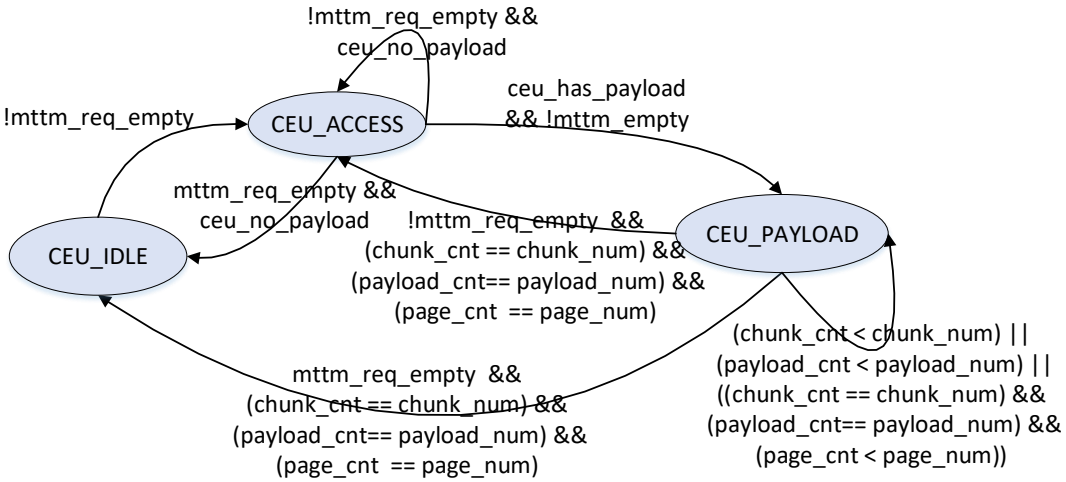
MPT_IDLE：空闲状态。当 mptm_start 信号为高时，进入 MPT_ACCESS 状态

MPT_ACCESS：访问 mptmdata 状态，读取对应表项，获取对应 MPT 表项的物理地址。

MPT_DMA_REQ：发起 DMA 请求状态。获取 mptmdata RAM 读出的数据，随即根据 opcode 类型，向对应的 DMA write 或者 DMA read FIFO 发出请求（注：由于每次读写 mpt 请求的数量都为 1 个 MPT 表项大小（64B），而 4KB 恰好是 64B 的整数倍，因此不存在 4KB 跨页问题）。若目标 FIFO 队列未满（即成功发出写请求），并且外部没有新的请求，则进入 MPT_IDLE 状态；若目标 FIFO 队列未满（即成功发出写请求），并且外部有新的请求，则进入 MPT_ACCESS 状态；否则停留在 MPT_DMA_REQ 状态。

5.3.5.4 mttm_proc 子状态机

用于处理 ceu_tptm_proc 请求的状态机



CEU_IDLE：空闲状态。当 mttm_req_proc FIFO 非空时，进入 CEU_ACCESS 状态

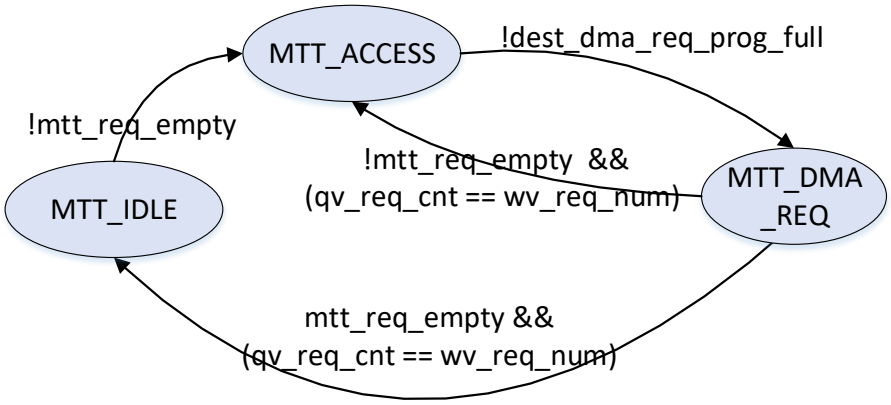
CEU_ACCESS：访问 mttmdata 状态。（1）执行 ceu 发起的 4 种 mttmdata 请求，如下表，

若没有包负载（加粗的请求有负载），则进入 IDLE 状态；若有包负载（加粗的请求有负载），则进入 CEU_PAYLOAD；若还有其他请求则继续停留在 CEU_ACCESS 状态；

command	type	opcode
INIT HCA	WR_ICMMAP_TPT	WR_ICMMAP_EN
CLOSE HCA	WR_ICMMAP_TPT	WR_ICMMAP_DIS
ICM MAP	**WR_ICM_TPT**	**MAP_ICM_EN**
ICM UNMAP	WR_ICM_TPT	MAP_ICM_DIS

CEU_PAYLOAD：针对上表中加粗的请求，将包负载根据包头中虚实地址映射数量 chunk_num 信息，放到 mttmdata ram 中。注意负载为 256 位宽，RAM 为 52 位宽（物理地址 52 位、reg 有效位 1 位（有效位使用单独的 reg 数组记录）），因此每个时钟的负载需要两次判断、写入的操作才能完成，对于一次判断加操作而言，每 128 位的负载还要根据 page_num 的数量判断要发起写 mptmdata RAM 的次数。

用于处理 mtt 请求的状态机



MTT_IDLE：空闲状态。当 mttm_start 信号为高，即 mtt 请求不为空时，进入 MTT_ACCESS 状态

MTT_ACCESS：访问 mttmdata 状态，读取对应表项，获取对应 MTT 表项的物理地址。

MTT_DMA_REQ：发起 DMA 请求状态。获取 mttmdata RAM 读出的数据，随即根据 opcode 类型，向对应的 DMA write 或者 DMA read FIFO 发出一定数量的请求（请求的数量与 mtt 的数量和 mtt 的初始 index 有关，每跨 4KB 的物理页，则需要发起新的 DMA 请求，因为 DMA 引擎目前设计的接口每次传输的数据大小在 4KB 以内）。若目标 FIFO 队列未满（即成功发出写请求），并且外部没有新的请求，则进入 MTT_IDLE 状态；若目标 FIFO 队列未满（即成功发出写请求），并且外部有新的请求，则进入 MTT_ACCESS 状态；否则停留在 MTT_DMA_REQ 状态。

5.4 HostOp

5.4.1 功能

HostOp 模块用于完成与 DMA 引擎的接口交互，实现对两大类数据的 Host 端主存读、写操作，一共包含四个子模块：

（1）读、写上下文（TPT）表项；（**dma read ctx、dma write ctx**）

（2）对读网络数据、WQE；写网络数据、CQE、EQE；（**dma read data、dma write data**）

5.4.2 dma read ctx

5.4.2.1 功能

（1） 接收 TPTMetaData 模块发起的 DMA 读 MPT、MTT 表项的请求，分别将 MPT/MTT 读请求转化成 DMA 引擎接口的模式，与 DMA 引擎进行交互；

（2） 将 DMA 读请求元数据分别发送到与 MPT 模块连接的 FIFO 中，以备对 DMA 引擎返回的读响应数据进行处理，将数据分别放到 MPT 模块的对应地址中。（MPT RAM 实现中使用的是非阻塞的方式，需要对读 DMA 请求元数据的备份）

（3） MTT RAM 实现中使用的是阻塞的方式，不需要进行读 DMA 请求元数据的备份

5.4.2.2 接口

TPTMetaData 接口

请求包头 163 位宽（注：左边高位，右边低位）：

共两个 FIFO 接口分别来自 MPTM 和 MTTM

Req header	Seg	index	opcode	length	physical addr
	Bit width	64	3	32	64
	Value	req_num	read mpt 001/read mtt 101	长度	mpt/mtt_addr

注：DMA Read Ctx 返回的 256 位宽 MPT/MTT 负载，直接与 TPT Arbiter 模块对接

MPT/MTT（mpt_ram、mtt_ram）接口

用于存放读请求元数据备份

Seg	index	opcode	length
Bit width	64	3	32
Value	req_num	read mpt 001/read mtt 101	长度

DMA 引擎接口

读请求接口：axis 接口，使用 128 位宽的 head，256 位宽的数据

读响应接口：axis 接口，使用 128 位宽的 head，256 位宽的数据

DMA 引擎请求包头格式

DMA Req header	Seg	Reserved	addr	Reserved	Byte length
	Bit width	127:96	95:32	31:12	11:0
	Value	void	mpt/mtt addr	void	Byte length

5.4.2.3 状态机

该模块无需状态机控制，只要 mttm/mttm 请求读 mpt/mtt 的请求 FIFO 非空，则按照 DMA 引擎的接口发出 DMA 读请求，同时将读请求元数据分别存入与 mpt/mtt 模块相连接的 backups FIFO，以备 mpt/mtt 模块处理 DMA 引擎返回的 mpt/mtt 数据。

5.4.3 dma write ctx

5.4.3.1 功能

（1） 接收 TPTMetaData 模块发起的 DMA 写 MPT、MTT 表项的请求，根据写请求的地址和数据长度，从 MPT/MTT 模块中获取 MPT/MTT 表项数据，按照 DMA 引擎接口向 DMA 引擎发起写 MPT/MTT 请求；

5.4.3.2 接口

TPTMetaData 接口

请求包头 99 位宽（注：左边高位，右边低位）：

DMA Write Ctx Req header	Seg	opcode	length	physical addr
	Bit width	3	32	64
	Value	Write mpt	长度	mpt_addr

注：MPT 和 MTT 的 256 位宽负载，在此与 DMA Write Ctx 模块对接，由 DMA Write 模块根据请求包头到对应的模块读负载。

MPT（mpt_ram）接口

从 mpt 模块接收 256 位宽的 mpt payload 数据

MTT（mtt_ram）接口

从 mtt 模块接收 256 位宽的 mtt payload 数据

DMA 引擎接口

写请求接口：axis 接口，使用 128 位宽的 head，256 位宽的数据

DMA 引擎请求包头格式

DMA Req header	Seg	Reserved	addr	Reserved	Byte length
	Bit width	127:96	95:32	31:26	25:0
	Value	void	mpt/mtt addr	void	Byte length

5.4.3.3 状态机

该模块无需状态机控制，只要 mttm/mttm 请求写 mpt/mtt 的请求 FIFO 非空，则按照 DMA 引擎的接口发出 DMA 写请求。（注：该模块需要考虑将 MPT/MTT 模块提供的 256 位宽数据进行切分，需要用到 rest_legnth、tmp_data 寄存器来解决计算和存储同一个请求所携带的数据长度和数据拼接问题）

5.4.4 dma read data、dma read wqe（完全独立的两个处理模块及通道）

5.4.4.1 功能

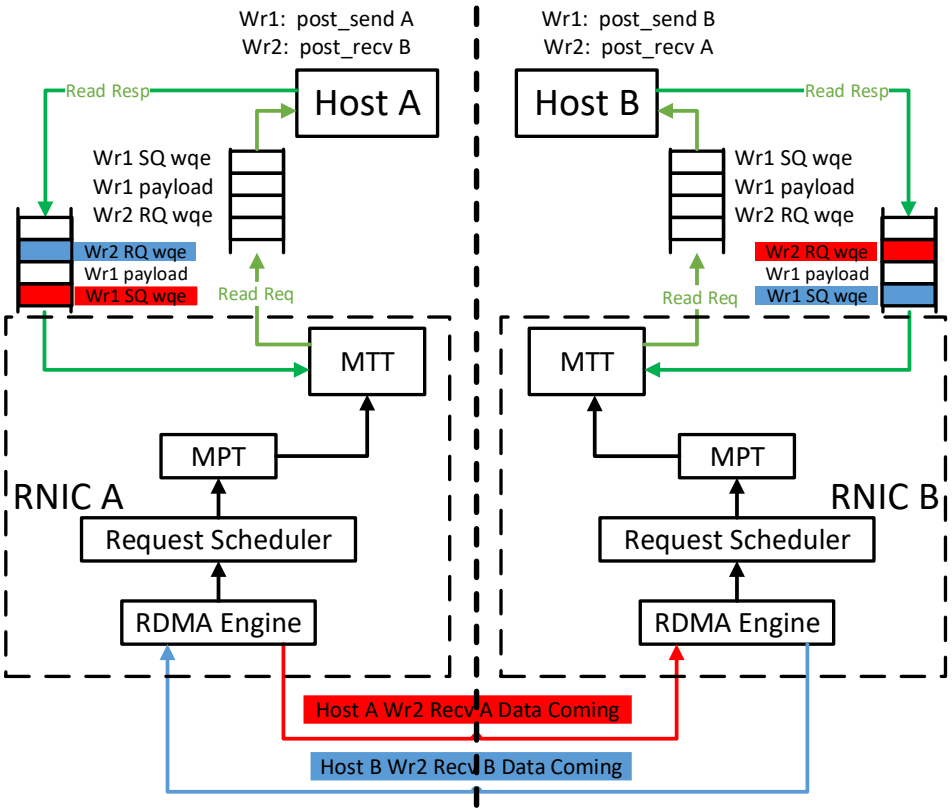
（1） 接收 MTT 模块发起的 DMA 读网络数据、WQE 的请求，根据读请求的地址和数据长度，将读请求切分成 4KB 边界的请求，与 DMA 引擎进行交互；

（2） 将读请求的元数据信息备份，用于对读响应包进行处理

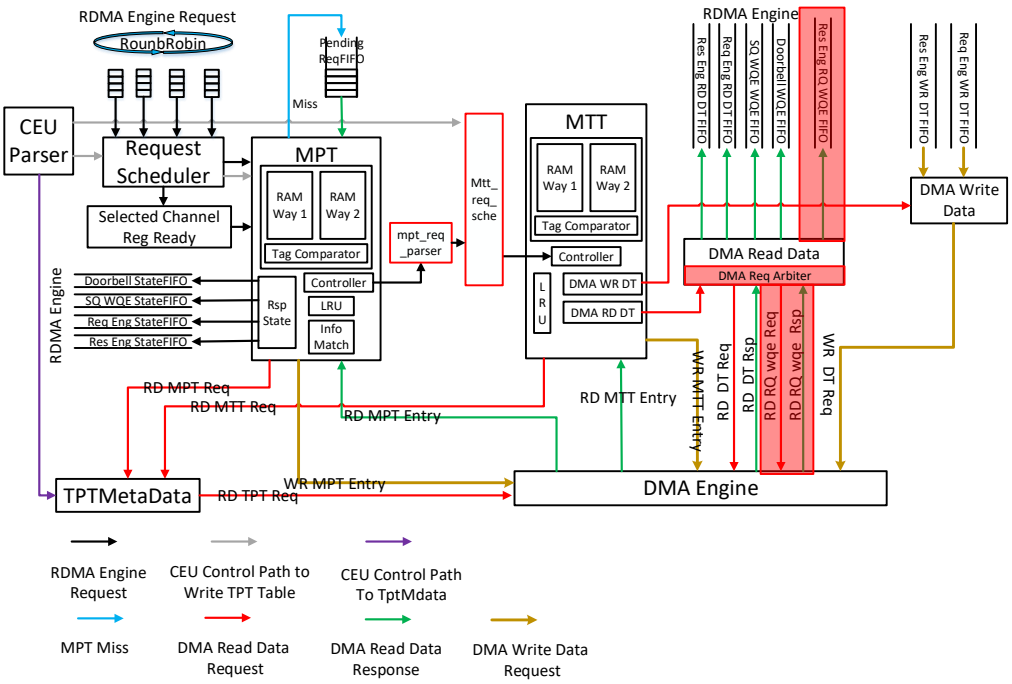
（3） 接收 DMA 引擎返回的网络数据、WQE，将数据拼接成无边界的数据格式，分别返回到 RDMA 引擎相连接的各个 Channel 的 FIFO 通道。

（4） 非阻塞设计优化：将读 RQ WQE 的通道与其他读数据的通道分离，使用单独的 DMA 通道进行读 RQ WQE 的管理

阻塞问题说明：通信流程中大模块间因为 RQ WQE 被网路数据阻塞无法返回的死锁



解决：将读 RQ WQE 的通道与其他读数据的通道分离，使用单独的 DMA 通道进行读 RQ WQE 的管理



5.4.4.2 接口

MTT 接口

请求包头 134 位宽（注：左边高位，右边低位）：

Req header	Seg	total ength	opcode	dest	tmp length	physical addr
	Bit width	32	3	3	32	64
	Value	总数据长度	RD_DT/ RD_DT_FIRST	目的通道	该页面长度	物理地址

RDMA Request Channel 接口

256 位宽纯数据

信号	I/O	位宽	描述	对接模块
RD DT FIFO	Out	256	返回读取的数据	Doorbell Processing(WQE) db
RD DT FIFO	Out	256	返回读取的数据	WQE Parser(WQE）wp_wqe
RD DT FIFO	Out	256	返回读取的数据	WQE Parser(DATA) wp_nd
RD DT FIFO	Out	256	返回读取的数据	Execution Engine(WQE) rwm
RD DT FIFO	Out	256	返回读取的数据	Execution Engine(DATA) ee

DMA 引擎接口

dma 读数据请求（AXIS）

信号	I/O	位宽	描述	对接模块
dma_v2p_dt_rd_req_valid	Out	1	有效	DMA Engine
dma_v2p_dt_rd_req_last	Out	1	最后一拍	DMA Engine
dma_v2p_dt_rd_req_data	Out	256	请求数据	DMA Engine
dma_v2p_dt_rd_req_head	Out	128	请求包头	DMA Engine
dma_v2p_dt_rd_req_ready	In	1	准备就绪	DMA Engine

dma 读数据响应（AXIS）

信号	I/O	位宽	描述	对接模块
dma_v2p_dt_rd_rsp_valid,	In	1	有效	DMA Engine
dma_v2p_dt_rd_rsp_last ,	In	1	最后一拍	DMA Engine
dma_v2p_dt_rd_rsp_data ,	In	256	数据负载	DMA Engine
dma_v2p_dt_rd_rsp_head ,	In	128	数据包头	DMA Engine
dma_v2p_dt_rd_rsp_ready,	Out	1	准备就绪	DMA Engine

dma 读 wqe 请求（AXIS）

信号	I/O	位宽	描述	对接模块
dma_v2p_wqe_rd_req_valid	Out	1	有效	DMA Engine
dma_v2p_wqe_rd_req_last	Out	1	最后一拍	DMA Engine
dma_v2p_wqe_rd_req_data	Out	256	请求数据	DMA Engine

dma_v2p_wqe_rd_req_head	Out	128	请求包头	DMA Engine
dma_v2p_wqe_rd_req_ready	In	1	准备就绪	DMA Engine

dma 读 wqe 响应（AXIS）

信号	I/O	位宽	描述	对接模块
dma_v2p_wqe_rd_rsp_valid,	In	1	有效	DMA Engine
dma_v2p_wqe_rd_rsp_last ,	In	1	最后一拍	DMA Engine
dma_v2p_wqe_rd_rsp_data ,	In	256	数据负载	DMA Engine
dma_v2p_wqe_rd_rsp_head ,	In	128	数据包头	DMA Engine
dma_v2p_wqe_rd_rsp_ready,	Out	1	准备就绪	DMA Engine

5.4.4.3 状态机

模块无状态机控制，只要 mtt_ram_ctl 请求读网络数据的请求 FIFO 非空，则按照 DMA 引擎的接口发出 DMA 读请求，同时将读请求元数据存入本模块的 backups FIFO 中，以备本模块接收来自 DMA 引擎返回的数据时，按照请求信息处理 DMA 引擎返回的网络数据，将数据返回到 rdma 引擎的不同请求通道中。（注：该模块需要考虑将 DMA 引擎通过 axis 返回的 256 位宽数据进行切分或者拼接才能发送给 rdma 引擎不同的通道，需要用到 total_rest_length、tmp_rest_legnth、tmp_data、offset 寄存器来解决计算和存储同一个 rdma 请求、同一个 mtt 表项请求所携带的数据长度和数据拼接问题）

该模块的整体实现方式与 dma read ctx 模块相似，但是需要考虑更多的是，除了将 DMA 引擎返回的数据拼接成 4KB 为界限的数据外，还需要将同一个 rdma 请求（或者说是同一个 mpt 查表引发的请求）的数据合并成无空泡数据。

5.4.5 dma write data

5.4.5.1 功能

（1）接收 MTT 模块发起的 DMA 写网络数据、CQE、EQE 的请求，轮询调度读请求，根据写请求的地址和数据长度，将写请求切分成 4KB 边界的请求，与 DMA 引擎进行交互；

（2）接收 RDMA 引擎各个 Channel 提供的网络数据、CQE、EQE 数据，将数据切分成 4KB 边界的数据格式交给 DMA 引擎。

5.4.5.2 接口

MTT 接口

请求包头 134 位宽（注：左边高位，右边低位）：

Req header	Seg	total ength	opcode	dest	tmp length	physical addr
	Bit width	32	3	3	32	64

	Value	总数据长度	WR_DT/ WR_DT_ FIRST	目的通道	该页面长度	物理地址

RDMA Request Channel 接口

256 位宽纯数据

信号	I/O	位宽	描述	对接模块
WR DT FIFO	In	256	读取要写入的数据	RequesterTransControl(CQ) rtc
WR DT FIFO	In	256	读取要写入的数据	RequesterRecvControl(DATA) rrc
WR DT FIFO	In	256	读取要写入的数据	Execution Engine(DATA) ee

DMA 引擎接口

dma 读数据请求（AXIS）

信号	I/O	位宽	描述	对接模块
dma_v2p_dwr_req_valid,	Out	1	有效	DMA Engine
dma_v2p_dwr_req_last ,	Out	1	最后一拍	DMA Engine
dma_v2p_dwr_req_data ,	Out	256	请求数据	DMA Engine
dma_v2p_dwr_req_head ,	Out	128	请求包头	DMA Engine
dma_v2p_dwr_req_ready	In	1	准备就绪	DMA Engine

5.4.5.3 状态机

该模块无需状态机控制，只要 mtt_ram_ctl 请求写网络数据的请求 FIFO 非空，则按照 DMA 引擎的接口发出 DMA 写请求。（注：该模块需要考虑将 rdma 引擎模块提供的 256 位宽数据进行切分或者拼接，需要用到 total_rest_length、tmp_rest_legnth、tmp_data、offset 寄存器来解决计算和存储同一个 rdma 请求、同一个 mtt 表项请求所携带的数据长度和数据拼接问题）

该模块的整体实现方式与 dma write ctx 模块相似，但是需要考虑更多的是，除了将数据拆分成 4KB 为界限的数据，还需要考虑这个数据是属于同一个 rdma 请求（或者说是同一个 mpt 查表引发的请求）。

5.5 TPT Cache

5.5.1 存储空间

参考 CX-4 的 Page Entry Table 大小为 8Mbit，目前考虑使用 64Byte 的 Cache line，1MB Cache 空间，写命中时使用写穿透策略，写缺失时使用写分配策略，替换使用 LRU 策略。

具体问题及解决方案：

5.5.1.1 Cache 空间大小分配问题：

按照 RDMA 引擎支持的 QP 数量为 2^14 计算，参考 mthca 中 QP：MPT：MTT=1:2:16

的数量关系。设置 MPT 数量 2^15，MTT 数量 2^18，每个 MPT 大小 64B，每个 MTT 大小 8B，则 MPT+MTT 总空间大小为：2^22B＝4MB，CX-4 网卡表项 Cache 是 8Mbit，我们若也采用 1MB Cache 大小，计划 MPT 和 MTT Cache 空间分别占用 512KB，MPT 存放 8K 个条目，MTT 存放 64K 个条目。（考虑使用可参数化配置的设计，Cache 总空间大小可参数化配置）

5.5.1.2 组相联数量问题：

目前考虑使用 2 路组相联。（设计中考虑使用可参数化组相联个数，其中不同组相联个数时 LRU 设计需要使用链表维护）

5.5.1.3 地址索引及存储布局参数化设计：

参数配置：设总 Cache 大小为 C MB，组相联个数为 w。则：MPT/MTT Cache 空间大小各为 C/2 MB。

1、MPT

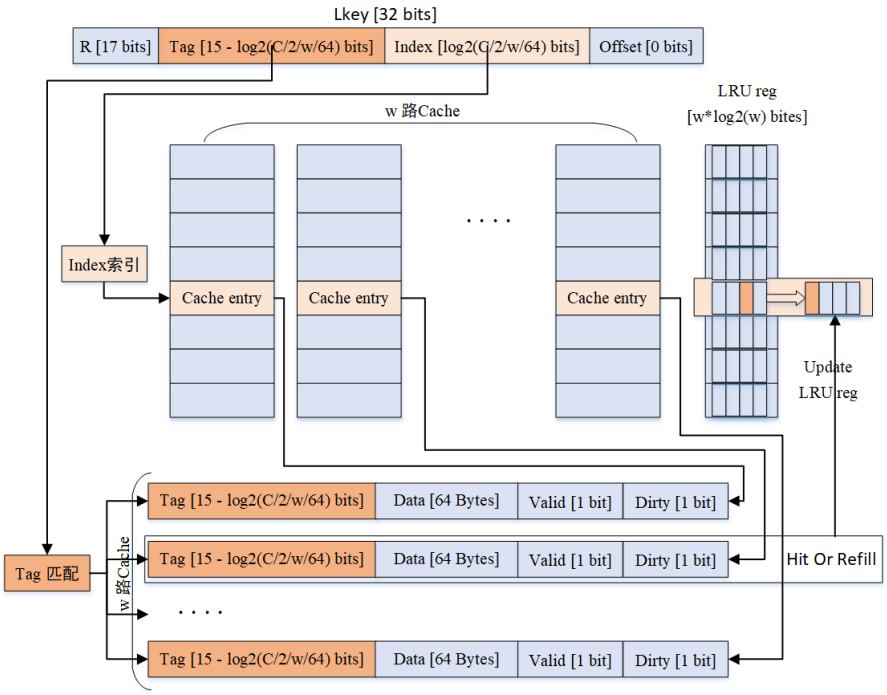
地址索引字段计算

- valid-addr 有效索引长度计算：MPT 总条目数量为 2^15，使用 32 位 Lkey 查询的有效索引长度为 15 位。
- offset 字段长度计算：Cache 行大小为 64B，刚好与 MPT 条目大小相等，因此不需要 Cache 行内的 offset 字段。
- Index 字段长度计算：log2（C/2/w/64）；（***注：该字段直接用于读写 RAM 的地址***）
- tag 字段长度计算：valid-addr – offset - log2（C/2/w/64）= 15 - log2（C/2/w/64）

具体字段分布如下

seg in lkey	reserved	tag	index	offset
width	17	15 - log2（C/2/w/64）	log2（C/2/w/64）	0

RAM 存储分布



2、MTT

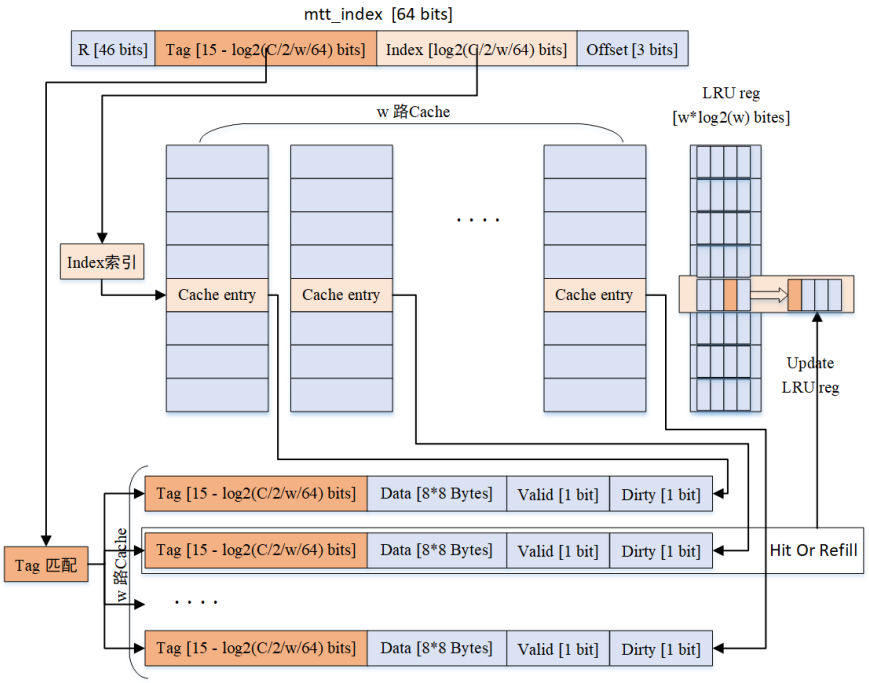
地址索引字段计算

- valid-addr 有效索引长度计算：MTT 总条目数量为 2^18，使用 64 位 mtt_seg 查询的有效索引长度为 18 位。
- offset 字段长度计算：若 Cache 行大小为 64B，而 MTT 条目大小为 8B，因此要 Cache 行内的 offset 字段长度为 3 位。
- Index 字段长度计算：log2（C/2/w/64）；（***注：该字段直接用于读写 RAM 的地址***）
- tag 字段长度计算：valid-addr – offset - log2（C/2/w/64）= 15 - log2（C/2/w/64）

具体字段分布如下：

seg in mtt_index	reserved	tag	index	offset
width	46	15 - log2（C/2/w/64）	log2（C/2/w/64）	3

RAM *存储分布*



问题

如果使用 64B Cache 行大小，那么每个 Cache 行中有 8 个 MTT 表项，当其中的某一个表项 UNMAP 之后，整个 Cache 行，变为 invalid 和 dirty，当有读请求访问其中的某个有效的 mtt 表项时，查询到的结果依然是 invalid，需要重新读？还是说为每个 mtt 设置一个 valid 标志位？或者将 mtt 的 Cache 行大小设置为 8B？

解决：MTT 是否 valid 与 Cache 行是否 valid 无关，MTT valid 表示 MTT 表项是否是有效的物理地址，当 CEU 执行 UNMAP 之后，MPT 会无效掉，此时 MPT 所指向的 MTT 不会被访问到，若被访问到则是属于未定义行为，因此对于 UNMAP 指令，MTT Cache 不需要做操作；而 Cache valid 位表示该 Cache 行中是否有数据，与其中的 MTT 表项是否有效无关。因此，MTT Cache 行可以使用 64B 大小，仅用 1 bit 表示 Cache 行 valid 即可。

5.5.2 管理机制

总体而言，Cache 使用写回机制，LRU 替换策略。

Cache 读：RDMA 各个请求通道读取 TPT 表项时，如果命中，则按照 5.5.4 的查询机制进行 MPT 表项 Match_info 匹配，最终在 MTT 中获取物理地址，然后发起对应的 DMA 操作；若未命中，将对应的请求放到 PendingFIFO 中缓存，同时更新 Pend Channel Count Reg，然后查询 TPTMetaData 中对应表项的物理地址（图 5-5 棕色箭头），接下来如图 5-5 中绿色箭头所示，发起 DMA 读请求，当接收到 DMA TPT 表项后，则从 PendingFIFO 中取出之前 miss 的请求优先处理，并且更新 Pend Channel Count Reg、MPT、MTT。

Cache 写：Cache 主要是由 CEU 发起的 TPT 表项的更新和无效。当 CEU 更新 TPT 表项时，将对应的 TPT 表项值更新，同时将其置为脏块写回到主存，写回时，查询 TPTMetaData

中对应表项的物理地址（图 5-5 棕色箭头），接下来如图 5-5 中蓝色箭头所示，发起 DMA 写 TPT 表项请求。

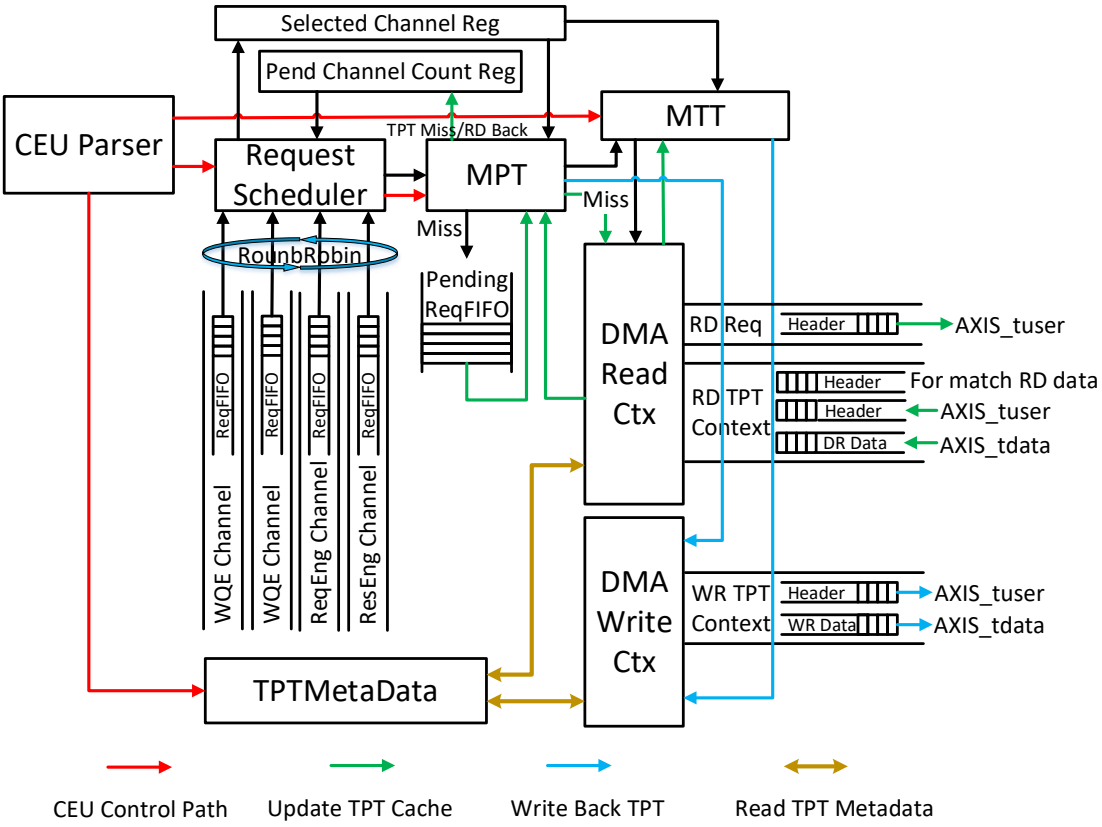


图 5-5 TPT Cache 管理机制

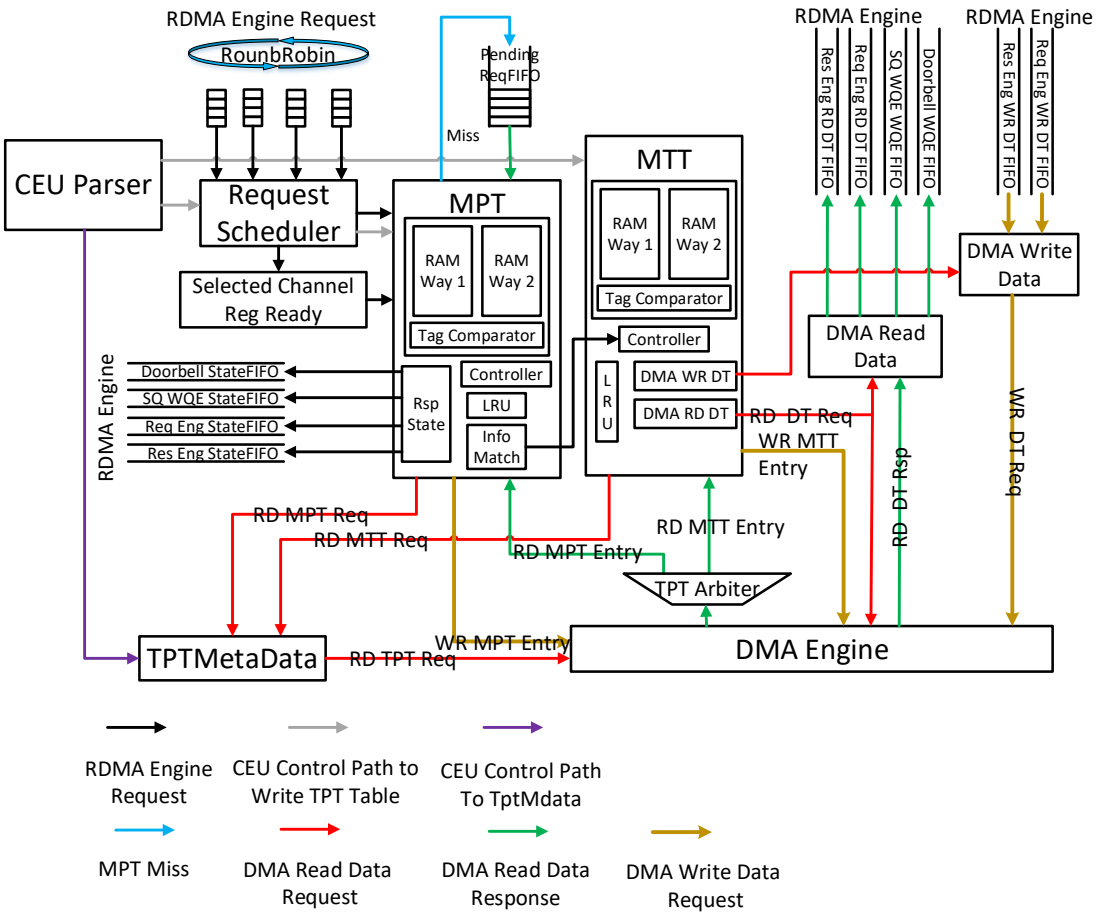


图 5-5 TPT Cache 总体设计及连接关系

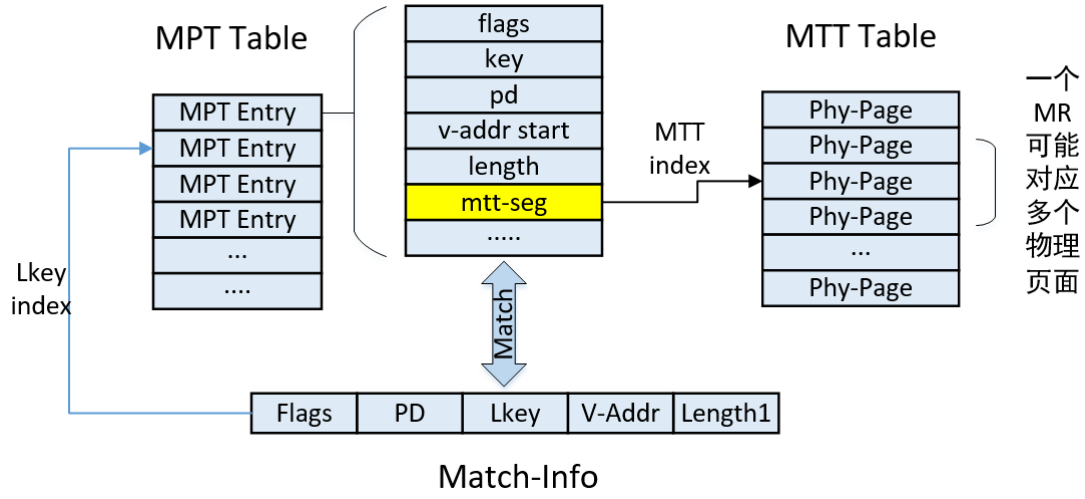
5.5.3 查询机制

地址查询分为 Direct 寻址模式（即一个 MKey 寻址一个 MR 区域）和 Indrect 模式（一个 MKey 寻址多个 MR 区域）。其中 Indrect 模式用于 UMR（User-Mode Memory Registration）拓展功能，在此实现中，暂且未实现 UMR 的 Indirect 模式，下文介绍均以 Direct 寻址模式介绍。

查询 MPT 的索引为 Match-Info 中的 lkey， Match-Info 包括以下字段

字段	Flags	PD	LKey	V-Addr	Length1
长度	32 bits	32 bits	32 bits	64 bits	32 bits

查询过程如下：



1、 Lkey 索引 MPT 表项：

通过 Lkey 索引 MPT 表项中的条目，若无命中则发起 DMA 操作，获取 TPT 表项；若命中，获取 MPT Entry；

2、 MPT Match-Info 检查：

- 检查 MPT 中的 mr 对应的 pd 信息与 QP 对应的 pd 是否匹配，若错误返回 pd 错误信息；
- 检查访问权限，检查请求的 flags 与 MPT 表项中的 flags 是否匹配，若错误返回访问权限错误信息；
- 长度边界检查：（1）绝对虚拟地址方式，假设 V-Addr >= v-addr start 且 V-Addr + Length1 < v-addr start + length，则地址边界检查通过；否则返回边界错误信息；（2）偏移虚拟地址方式，**注：通过 Doorbell 读 wqe 时，V-Addr 为偏移，因此比较标准为** V-Addr + Length1 < length。

3、 获取物理地址：

（1）、（a）绝对虚拟地址方试，取 V-Addr 的高 52 位和 v-addr start 的高 52 位做差得到数值 n；（b）偏移虚拟地址方式，V-Addr 的高 20 位就是数值 n。n 表明本次访问的页面是该 MR 中的第 n 个页面。因此，访问的 MTT 页表项应该是在从 mtt-seg 开始的第 n 项开始取。设该 MTT 页表项地址记作 PTA。

（2）、通过 PTA 读取当前页表项保存的 Phy-Page（即物理页面的基地址）。若当前 PFN 是本次处理的第一个页表项，则页内偏移 Offset = V-Addr 的低 12 位，否则，Offset = 0。检查 Offset + Length1 <= 4KB 是否成立。若成立，则从读/写 Length 长度的数据交给 DMA 引擎处理，本次请求处理结束；若不成立，说明还要访问下一个页面，需要读取下一个页表项，则需要读/写（4KB - Offset）长度的数据，交给 DMA 引擎处理。然后将 Length 置为（Length - （4KB - Offset）），将 PTA 置为（PTA + 4），转步骤 3（2）。

11.2 问题：

MPT/MTT 要参数化

MLNX 的网卡是 128 路，可能使用 CAM

5.6 MPT

5.6.1 接口

5.6.1.1 Request Scheduler 接口

1）Selected Channel Reg

读取 Selected Channel Reg 格式（9bit），8 个 Channel 中每次只有 1 个和 Ready 同时为 1，其余 Channel 为 0,决定从哪个 req_fifo 中获取请求，并将 ready 位清零 MPT 根据 Request Scheduler 的 Selected Channel Reg 中的 Ready 信号确定此时的 Selected Channel 是否有效，若有效，则根据 Selected Channel 从对应的 ReqFIFO 中获取 Req，同时将 mpt_read reg 置 0，表明 Selected Channel ReqFIFO 读出完毕，需要更新 Selected Channel Reg。（注：selected_channel_reg 由 selected_channel_ctl 模块集中控制 request_scheduler 和 MPT 模块对 ready 位的读写 ）最后进行 5.6.2 中 Cache 读写的策略；若 Ready 无效，则等待。

字段	位	说明
Channel 0	0	CEU
Channel 1	1	Doorbell Processing(WQE)
Channel 2	2	WQE Parser(WQE）
Channel 3	3	WQE Parser(DATA)
Channel 4	4	RequesterTransControl(CQ)
Channel 5	5	RequesterRecvControl(DATA)
Channel 6	6	Execution Engine(RQ WQE)
Channel 7	7	Execution Engine(DATA)
Ready	8	更新 Channel bit 的同时，置 1，表明此时 Reg 已更新，等 MPT 读取；MPT 读取 Req 后会将 Ready 信号置 0，表明此时需要更新

2）Pend Channel Count Reg

写入 Pend Channel Count Reg 格式（32 bit），该寄存器组在进行 MPT Miss 或者 TPT 表返回时更新。当 MPT Miss 时，对应的 Channel 计数器+1；当 TPT 表返回时，对应的 Channel 计数器-1。该寄存器组的目的是，当某个 Channel 中有 Req 被挂起在 PendingFIFO 中时，Request Scheduler 便不再调度该 Channel 同的请求，以避免乱序的情况出现。

字段	位	说明

CEU	0:3	CEU 由于 Miss 被挂起的 Req 数量
Doorbell Processing(WQE)	4:7	DB Proc 由于 Miss 被挂起的 Req 数量
WQE Parser(WQE）	8:11	WQE Parser 由于 Miss 被挂起的 Req 数量
WQE Parser(DATA)	12:15	WQE Parser 由于 Miss 被挂起的 Req 数量
RequesterTransControl(CQ)	16:19	RTC 由于 Miss 被挂起的 Req 数量
RequesterRecvControl(DATA)	20:23	RRC 由于 Miss 被挂起的 Req 数量
Execution Engine(RQ WQE)	24:27	Execution Engine 由于 Miss 被挂起的 Req 数量
Execution Engine(DATA)	28:31	Execution Engine 由于 Miss 被挂起的 Req 数量

5.6.1.2 Request Channel 接口

经过 request scheduler 的调度，通过 selected_channel reg 决定从对应的 req_fifo 中读取请求，连接的 request channel 有：

1）接收 CUE_parser 写、无效 MPT 表项的请求

Req 5 header	Seg	type	opcode	R	addr	R
	Bit width	4	4	24	32	64
	Value	WR_MPT_TPT	WR_MPT_WRITE	void	mpt_index	void
Req 5 Payload	5.5.2 MPT entry 格式,共 56 Byte = 56 * 8，需要 256 位 payload 2 个 cycle					

Req 6 header	Seg	type	opcode	R	addr	R
	Bit width	4	4	24	32	64
	Value	WR_MPT_TPT	WR_MPT_INVALID	void	mpt_index	void
Req 6 No Payload，该命令会在无效 MPT 的同时将对应的所有 MTT 表项无效掉						

2）接收 RDMA 引擎读写数据请求、向 RDMA 引擎返回 State 信息

RDMA 引擎中的 7 个通道的：

（1）req_fifo 的 rd_en 和 dout 信号

（2）state_fifo 的 prog_full、wr_en、wr_din 信号

Doorbell Processing(WQE)
WQE Parser(WQE）
WQE Parser(DATA)
RequesterTransControl(CQ)
RequesterRecvControl(DATA)
Execution Engine(RQ WQE)
Execution Engine(DATA)

5.6.1.3 向 TPTMetaData 模块发起读、写 MPT 的查询请求

请求包头 99 位宽（注：左边高位，右边低位）：

Req header	Seg	opcode	number	index
	Bit width	3	32	64
	Value	wr/rd	写 mpt 时为 1，写 mtt 时为 mtt 数量	mpt_lkey(32) mtt_index(64)

5.6.1.4 DMA 相关接口

1）向 dma_write_ctx 模块传输要写回的 MPT 表项数据

MPT 表项数据为纯 256 位宽的 FIFO

2）接收来自 dma_read_ctx 模块的读请求元数据

用于存放读请求元数据备份

Seg	index	opcode	length
Bit width	64	3	32
Value	req_num	read mpt 001/read mtt 101	长度

3）接收来自 DMA 引擎的 MPT 表项数据

读响应接口：axis 接口，使用 128 位宽的 head，256 位宽的数据

DMA 引擎请求包头格式

DMA Req header	Seg	Reserved	addr	Reserved	Byte length
	Bit width	127:96	95:32	31:12	11:0
	Value	void	mpt/mtt addr	void	Byte length

5.6.1.5 MTT 模块接口

向 MTT 模块发起查询 MTT 表项请求

使用 mtt 索引查询物理地址，然后由 MTT 模块发起读、写数据（WQE、CQ 网络数据）请求（mpt_req_mtt FIFO）

MTT Req header	Seg	Src	Op	mtt_index	V-addr	Byte length
	Bit width	164:162	161:160	159:96	95:32	31:0
	Value	见下表	Read/Write	mtt 索引	virt addr	Byte length

（注：该处的 V-addr 均为处理后的相对地址偏移）

为便于其他模块（MPT/MTT/DMA Read Data/DMA Write Data）获知正在处理 Req 所属通道，进而明确获取外部数据或者向外部输出数据的 FIFO，MPT 会在 Req 头部增加 3bit REQ_SOUR 作为标识。

REQ_SOUR 字段	Value
CEU	000
Doorbell Processing(WQE)	001

WQE Parser(WQE）	010
WQE Parser(DATA)	011
RequesterTransControl(CQ)	100
RequesterRecvControl(DATA)	101
Execution Engine(RQ WQE)	110
Execution Engine(DATA)	111

5.6.2 数据分布

mthca 设计中每个 MPT 条目长度为 16*4=64 字节，除去 reserved 字段，该版设计每个 MPT 条目的有效长度为 14*4=56 字节，但为了方便计算，在进行虚实地址映射、TPTMetaData 表项计算时，按照 64B 计算。MPT 表项的数量在初始化时确定。

（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）

offset	+0 Byte	+1 Byte	+2 Byte	+3 Byte	Description
00h	flags				MPT 表项属性标志位 MTHCA_MPT_FLAG_SW_OWNS 【31:28】 MTHCA_MPT_FLAG_MIO 【17】 MTHCA_MPT_FLAG_BIND_ENABLE【15】 MTHCA_MPT_FLAG_PHYSICAL【9】 MTHCA_MPT_FLAG_REGION【8】 MPT 表项访问权限标志位 IBV_ACCESS_LOCAL_WRITE【0】 IBV_ACCESS_REMOTE_WRITE【1】 IBV_ACCESS_REMOTE_READ【2】 IBV_ACCESS_REMOTE_ATOMIC【3】 IBV_ACCESS_MW_BIND【4】 IBV_ACCESS_ZERO_BASED【5】 IBV_ACCESS_ON_DEMAND【6】
04h	page_size				该 MPT 条目指向的页大小
08h	key				Memory Key，与 MPT 最大条目数相与就是该 MPT 条目表内偏移
0Ch	pd				该 MPT 表项所在的保护域号
10h	start[63:32]				该 MPT 表项起始的虚拟地址
14h	start[31:0]				

18h	**length[63:32]**	该 MPT 表项保护的地址空间大小
1Ch	**length[31:0]**	
20h	lkey	reserved
24h	window_count	reserved
28h	window_count_limit	reserved
2Ch	**mtt_seg[63:32]**	该 MPT 表指向的 MTT 表偏移
30h	**mtt_seg[31:0]**	
34h	mtt_sz	reserved
38h	**2 bit**	**Valid、Dirty**

其中加粗的部分表示，需要查询或者对比 MPT 是否有误的重要字段。

5.6.3 子模块设计

5.6.3.1 mpt_ram

用于存放 MPT 表项，暂时使用两路 RAM，mpt_ram 模块设计采用不同通道之间无阻塞通道内阻塞的实现方式，即当来自不同请求通道的 mpt_ram 请求出现 cache miss 时，不影响其他通道的请求，但是由于同一通道内部的保序需求，当同一通道内的请求出现 cache miss 时，同一通道的请求会阻塞。

功能

（1）每次读、写一个 MPT 表项；

（2）读 MPT 表项缺失时：a、将读缺失状态 state、缺失的地址信息 miss_addr 通过 FIFO 返回给 mpt_ram_ctl 模块；b、向 mptmdata 模块发起读请求

（3）读 MPT 表项命中时：a、将读命中状态 state、读命中的数据 hit_data 通过 FIFO 返回给 mpt_ram_ctl 模块；

（4）写 MPT 表项命中：a、将写命中状态 state 通过 FIFO 返回给 mpt_ram_ctl 模块；b、更新对应的 MPT Cacheline

（5）写 MPT 表项缺失：a、将写缺失状态 state 通过 FIFO 返回给 mpt_ram_ctl 模块；b、根据 LRU 替换策略将要替换出去的脏 MPT 表项写入 dma_wr_mpt FIFO 以备 dma_write_ctx 模块发起 dma 写操作；c、根据 LRU 替换策略使用将要替换出去的脏 MPT 表项的索引地址向 mptmdata 模块发起写内存的请求；d、使用新的 MPT 数据更新对应的 MPT Cacheline。

接口

输出可查询信号：lookup_allow_in

输入查询信息：

信号	I/O	位宽	描述	对接模块
lookup_rden	In	1	读使能	mpt_ram_ctl
lookup_wren	In	1	写使能	mpt_ram_ctl
lookup_wdata	In	64B	写入数据	mpt_ram_ctl
lookup_index	In	12	查询索引	mpt_ram_ctl
lookup_tag	In	3	查询 tag	mpt_ram_ctl
lookup_stall	In	1	阻塞 mpt 查询流水	mpt_ram_ctl
mpt_base_addr	In	64	mpt 表虚拟基地址	mptmdata
mpt_eq_addr	In	1	物理地址	mpt_ram_ctl

输出查询结果：

信号	I/O	位宽	描述	对接模块
lookup_state	Out	3	{miss，hit，idle}	mpt_ram_ctl
lookup_ldst	Out	1	1 write/0 read	mpt_ram_ctl
state_valid	Out	1	1 查询状态有效；0 查询状态无效（stall）	mpt_ram_ctl
lookup state FIFO	Out	5	{lookup_state, lookup_ldst, state_valid}	mpt_ram_ctl
hit_data FIFO	Out	64B	输出读命中 MPT 数据（rd、dout，empty）	mpt_ram_ctl
miss_addr FIFO	Out	32	输出缺失的备份地址（rd、dout，empty）	mpt_ram_ctl
dma_wr_mpt FIFO	Out	32B	输出要写回的 mpt 负载（rd、dout，empty）	dma_write_ctx
mptm_req FIFO	Out	99	输出缺失 mpt 的读请求 输出写回 mpt 的写请求（rd、dout，empty）	mptmdata

地址索引字段计算

参数配置：设总 Cache 大小为 C MB，组相联个数为 w。则：MPT/MTT Cache 空间大小各为 C/2 MB。

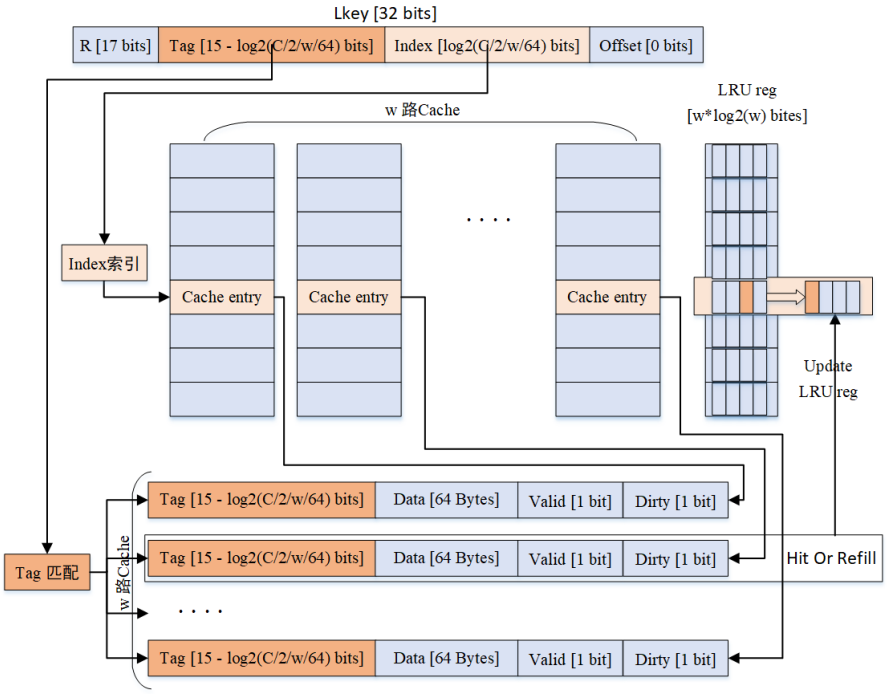
- valid-addr 有效索引长度计算：MPT 总条目数量为 2^15，使用 32 位 Lkey 查询的有效索引长度为 15 位。
- offset 字段长度计算：Cache 行大小为 64B，刚好与 MPT 条目大小相等，因此不需要 Cache 行内的 offset 字段。
- Index 字段长度计算：log2（C/2/w/64）；（***注：该字段直接用于读写 RAM 的地址***）
- tag 字段长度计算：valid-addr – offset - log2（C/2/w/64）= 15 - log2（C/2/w/64）

具体字段分布如下

seg in lkey	reserved	tag	index	offset
width	17	15 - log2（C/2/w/64）	log2（C/2/w/64）	0

RAM 存储分布

该设计中使用 2 路 BRAM，每一路 BRAM 宽度 64B。



处理流程

- Look up：读取 tag 和 valid 位，用于 update 阶段判断是否命中及有效；读出所有路的 Data，将数据（读取的 Cache 数据，地址，操作请求，写数据）传递给下一流水级，供 update 阶段命中时返回读数据，以及 Miss 时，replace 阶段替换时传递出去。
- Match & Update & Refill：获取上一级流水传递的操作类型、请求索引、读到的 tag valid 信息以及各路 data，判断是否有效、是否命中。

  （1）读命中时，返回 hit 命中信号，输出命中的 data，更新 LRU 信息留到下一流水级做（该流水级只对 LRU 进行读操作，对 LRU 的写操作均在下一流水级，避免写冲突）；

  （2） 读缺失时，向 mptmdata 模块发起读取 mpt 表项的请求；然后输出 miss 信号，输出缺失索引地址，供控制模块进行如下操作：a）refill 时获取备份的缺失信息，填入到对应的表项中；

  （注：读缺失时，无需拉高 LRU、Dirty 的读使能信号以及将读取到的 Cache 数据传到下一流水级，因为 refill 回来的数据到达后，必将发生 write miss，由于非阻塞的操作，期间可能有其他读写命中导致 LRU、Dirty 位的只发生变化，因此需要在第二流水级再次读 LRU、Dirty，这样，在首次读缺失时就读 LRU、Dirty，并在

下一流水级将替换的数据写回完全没有必要，只需在 write miss 发生时进行 LRU、Dirty 位的读，以及下一流水级进行替换写回即可。）

（3）写命中时，返回 write hit 命中信号，不做其他处理。将要写的数据传递给下一流水级，在下一流水级更新 LRU、Dirty 信息，将数据写入命中的位置；（仅在一个流水级中做数据写操作，这样不会发生写冲突）

（4）写缺失时，返回 write miss 信号，拉高 LRU、Dirty 的读使能信号，同时将要写的数据以及读取到的 Cache 数据传到下一流水级，以备下一流水级完成数据写入和脏块替换；

（5）Refill 时，需要当做一种特殊的写处理!!! 需要考虑替换的 Cache，不能直接放到原来替换的位置，因为在 DMA 读数据的过程中会出现其他操作改变了原来 LRU 的值。因此 refill 操作相当于一种写缺失时的操作，需要拉高 LRU、Dirty 的读使能信号，同时将读取到的 Cache 数据传到下一流水级。并且将需要 refill 的数据传递到下一流水级

- Select & Replace & Refill：

（1）当 write miss 发生时，在该流水级进行 replace，根据 LRU 以及 valid 信息，选出被替换路的 Cache 行，将从上一流水级接收到的对应的某一 Cache 行替写回；同时根据 LRU 选择对应的路，然后需要更新 Tag、Valid、Dirty、LRU；

（2）在 refill 操作中，进行与 write miss 同样的操作，需要替换写回，并将 refill 的数据填写到 LRU 策略选择的对应路的 Cache 中，同时更新 Tag、Valid、Dirty、LRU；

（3）若是 write hit，则将数据放入命中的 cache，同时更新 Dirty 和 LRU；

（4）在 read hit 操作时，在流水级，需要将对应的 LRU 寄存器更新。

	Stage	**Look up**	**Match & Update & Refill**				**Select & Replace & Refill**		
域	Tag	读2路的Tag	Read Hit	Read Miss	Write Hit	Write Miss（Refill）		Refill，Write 更新Tag	Read Hit
	V	读2路的V	-	-	-	-	-	更新V	
	MPT Data	读2路Data	输出MPT	-	-	传递MPT	-	更新Data	
	Dirty	-	-	~~读Dirty~~	-	读Dirty	miss时，选择LRU, Dirty=1，V=1的替换	更新Dirty	
	LRU	-	–	~~读LRU~~	-	读LRU		更新LRU	更新LRU
备份	读缺失地址信息备	-	–	备份	–	–	–	–	–
DMA信号	DMA读请求索引	-	–	输出读Miss MPT的请求	–	–	–	读取读请求元数据(index)	
	DMA读响应	-	–	–	–	–	–	读取MPT数据	

	DMA写请求索引	-	-	-	-		写替换MPT的请求	-	
	DMA写负载	-	-	-	-		替换的MPT负载，256的FIFO需要2 clk	-	

注：

（1）橘红色底纹表示需要与外部模块通过 FIFO 对接，FIFO 的 prog_full 信号需要作为流水级之间握手信号之一；

（2）高亮的表示，在流水级 2 和流水级 3 中都会有可能对 mptmdata 模块发起请求，因此当流水级 1 是读请求&流水级 2 中发生 write miss 时，需要对流水线进行阻塞，防止流水级 2 和流水级 3 对同一 FIFO 写，引发写竞争。

5.6.3.2 mpt_ram_ctl

功能

与 mpt_ram、request_schedule、selected_channel_ctl、ceu_parser、request channel、dma_rd_mpt_resp 信号对接，将外部的数据访问请求，以 mpt_ram 接口的形式放到对应的 reg 中，并根据 mpt_ran 返回的查询状态信息、查询获得的数据、pend_req FIFO 中的数据、dma_rd_mpt_resp 返回的 mpt 数据进行 MPT 请求检查，并将检查结果反馈到对应的请求状态查询通道，同时将获得的 mtt_seg 信息以 mpt_req_mtt FIFO 的格式发往 mtt_ram_ctl 进行下一步的查询。

具体功能实现分为 dma 响应及 mpt 查询处理、查询状态处理两个状态机完成，将本节状态机部分详细介绍。

接口

信号	I/O	位宽	描述	对接模块
pend_channel_cnt	Out	32	通知调度模块，miss 的 channel	request_scheduler
selected_channel_reg	In	9	读取调度的 channel	selected_channel_ctl
req_read_already	Out	1	通知已读取请求	selected_channel_ctl
ceu req header FIFO	In	128	ceu 写、无效 mpt 包头	ceu_parser
ceu req payload FIFO	In	256	ceu 写 mpt 表项负载	ceu_parser
req FIFO	In	256	请求包头	Doorbell Processing(WQE)
state FIFO	Out	8	返回查询状态	Doorbell Processing(WQE)
req FIFO	In	256	请求包头	WQE Parser(WQE）
state FIFO	Out	8	返回查询状态	WQE Parser(WQE）

req FIFO	In	256	请求包头	WQE Parser(DATA)
state FIFO	Out	8	返回查询状态	WQE Parser(DATA)
req FIFO	In	256	请求包头	Requester Trans Control(CQ)
state FIFO	Out	8	返回查询状态	Requester Trans Control(CQ)
req FIFO	In	256	请求包头	RequesterRecvControl(DATA)
state FIFO	Out	8	返回查询状态	RequesterRecvControl(DATA)
req FIFO	In	256	请求包头	Execution Engine(RQ WQE)
state FIFO	Out	8	返回查询状态	Execution Engine(RQ WQE)
req FIFO	In	256	请求包头	Execution Engine(DATA)
state FIFO	Out	8	返回查询状态	Execution Engine(DATA)
mpt_base_addr	In	64	mpt 表虚拟基址	mptmdata
req backup FIFO	In	99	dma 读请求备份	dma_read_ctx
dma rsp mpt data	In	256	dma 返回 mpt 数据	dma engine（AXIS-tadta）
dma rsp mpt data	In	128	dma 返回 mpt 的包头	dma engine（AXIS-theader）
lookup_allow_in	In	1	可以查询信号	mpt_ram
lookup_rden	Out	1	读使能	mpt_ram
lookup_wren	Out	1	写使能	mpt_ram
lookup_wdata	Out	64B	写入数据	mpt_ram
lookup_index	Out	12	查询索引	mpt_ram
lookup_tag	Out	3	查询 tag	mpt_ram
lookup_stall	Out	1	阻塞 mpt 查询流水	mpt_ram
mpt_eq_addr	Ouit	1	物理地址	mpt_ram
lookup_state	In	3	{miss，hit，idle}	mpt_ram
lookup_ldst	In	2	1 write/0 read	mpt_ram
state_valid	In	1	1 查询状态有效；0 查询状态无效（stall）	mpt_ram
lookup state FIFO	In	5	{lookup_state, lookup_ldst, state_valid}	mpt_ram_ctl
hit_data FIFO	In	64B	输出读命中 MPT 数据（rd、	mpt_ram

			dout，empty）	
miss_addr FIFO	In	32	输出缺失的备份地址（rd、dout，empty）	mpt_ram
mpt_req _mtt FIFO	Out	165	查询 mtt 请求	mtt_ram_ctl

1、dma 响应及 mpt 查询状态机

mpt_idle state：

空闲，若有 dma 响应数据返回，则进入 mpt_rsp_proc 转态；若无 dma 响应返回，且 selected_channel reg 有效，即表明有新的 request_channel 存在请求是，进入 mpt_lookup 转态处理。

mpt_rsp_proc state：

读取：（1）DMA 引擎返回的 MPT 数据；（2）PendingFIFO 中的请求信息；（3）miss_addr FIFO 中的 index 信息，以备发起 Cache 写请求时填写到 mpt_ram 中；（4）dma_read_ctx 模块的 backup FIFO，用于拼接 dma 返回的 mpt 数据

输出：（1）将获取的 PendingFIFO 中的请求与 mpt 表项进行匹配，根据匹配结果，向对应通道的 state_fifo 返回状态信息；（2）匹配无误的情况下需要根据 mtt_seg 字段向 mtt 模块发起请求（分为度数据请求和写数据请求两个通道）；（3）将 mpt 表项发送到 mpt_ram 完成 RAM 的更新

mpt_lookup state：

读取：（1）selected_channel_reg 对应的请求 FIFO 中的请求，若是 CEU 发起的写 mpt 表项的请求，还需要读取 CEU 模块的 payload；

输出：（1）根据请求信息，将请求对应的 index、tag、wdata、wren、rden 信息填到 mpt_ram 模块的接口；（2）将 rdma 引擎发起的读 mpt 请求放到 rd_mpt_req_fifo 中，以备查询状态处理状态机获取查询信息。

2、查询状态处理状态机

rd_state state：

空闲，若 state FIFO 非空，则读取 StateFIFO 中查询结果。若是读缺失，且 mpt_rsp_stall 无效，则进入 mpt_miss_proc 状态处理；若是读命中，且 mpt_rsp_stall 无效，则进入 mpt_hit_proc 状态处理。

（***注：需要考虑 mpt_rsp_stall***，因为当有 MPT DMA 响应包回来时，会更新 pend_channel_cnt reg，并会向对应请求模块的 rep_stateFIFO 返回 info match 的状态信息，在消息检查无误的情况下还会想 mtt_ram_ctl 模块写入请求。如果不考虑 mpt_rsp_stall 的话，在本状态机处理时，会出现与 mpt_rsp_proc 处理同时写 pend_channel_cnt、rep_state FIFO、mpt_req _mtt FIFO 的写冲突）

mpt_miss_proc state：

读取：（1）rd_mpt_req_fifo 中的查询信息

输出：（1）将 pend_cnt_reg 值更新；（2）将读 miss 的查询信息放入 PendingFIFO 中，以备 mpt_rsp_proc 状态处理；~~（3）若是 key 不合法，则将结果向对应的 channel state FIFO 反馈；~~

mpt_hit_proc state：

读取：（1）hit_data FIFO 中的 MPT 数据；（2）rd_mpt_req_fifo 中的查询信息；

输出：（1）将查询信息与 mpt 表进行对比查询，将检查结果向对应的 channel state FIFO 反馈；若是 key 不合法，同样将结果向对应的 channel state FIFO 反馈；（2）若是检查为 SUCCESS 状态，则根据 mpt 表项中的 mtt_seg 信息，向 mtt_ram_ctl 发起请求（分为度数据请求和写数据请求两个通道）

5.7 MTT

5.7.1 接口

5.7.1.1 Request Channel 接口

1）接收 CUE_parser 写 MTT 表项的请求

（注：左边高位，右边低位）：

Req header 7	Seg	type	opcode	R	num	index
	Bit width	4	4	24	32	64
	Value	WR_MTT_TPT	WR_MTT_WRITE	void	mtt_num	start_index
Req Payload 7	5.8.2 MTT entry 格式 64 bit/entry，63:0 entry0; 127:64 entry1 ••• 位宽 256					

2）接收 RDMA 引擎写数据的数据负载；向 RDMA 引擎发送读取数据的数据负载

Upload Channel

Requester TransControl（CQE）、Requester RecvControl（READ DATA）模块与 VirtToPhys 模块的通道接口均使用相同的 Upload Channel。

Upload Channel 包含 1 个数据 FIFO：WR DT FIFO，用于缓存请求写入主存的网络数据/CQE/EQE（256 位纯数据），与 DMA Write 直接相连，DMA Write 根据 MTT 的查询的结果和对应 mpt_req_mtt FIFO 中 Src 的通道信息找到对应的 WR DT FIFO，然后对数据进行拆分成物理数据地址块的大小，发起写网络数据/CQE/EQE 请求。

Download Channel

Doorbell Processing（WQE）、WQE Parser（WQE）、WQE Parser（DATA）、Execution Engine

（RQ WQE）模块与 VirtToPhys 模块的通道接口使用 Download Channel。

Download Channel 包含 1 个数据 FIFO：RD DT FIFO，用于返回请求的网络数据（256 位纯数据），与 DMA Read 直接相连，DMA Read 根据 MTT 的查询的结果发起读 WQE/网络数据请求，并将接收到的 DMA 网络数据/WQE 去除空泡，根据 mpt_req_mtt FIFO 中 Src 的通道信息找到对应的 RD DT FIFO，按请求的顺序返回数据。

Up_Down Channel

Execution Engine（SEND/WRITE UPDATA; READ DATA）模块与 VirtToPhys 模块的通道接口使用 Up_Down Channel。

Up_Down Channel 包含 2 个数据 FIFO：（1）WR DT FIFO（2）RD DT FIFO。此两个通道的功能与 Upload Channel、Download Channel 中介绍的功能一致。

RDMA 引擎中的 7 个通道的接口信号：

（1）RD DT FIFO 的 rd_en 和 dout 信号

（2）WR DT FIFO 的 prog_full、wr_en、wr_din 信号

RDMA Engine Channel	WR DT FIFO	RD DT FIFO
Doorbell Processing(WQE) db	×	√
WQE Parser(WQE）wp_wqe	×	√
WQE Parser(DATA) wp_nd	×	√
RequesterTransControl(CQ) rtc	√	×
RequesterRecvControl(DATA) rrc	√	×
Execution Engine(RQ WQE) rwm	√	×
Execution Engine(DATA) ee	√	√

5.7.1.2 向 TPTMetaData 模块发起读、写 MPT 的查询请求

请求包头 99 位宽（注：左边高位，右边低位）：

Req header	Seg	opcode	number	index
	Bit width	3	32	64
	Value	wr/rd	写 mtt 时为 mtt 数量	mtt_index(64)

5.7.1.3 DMA 相关接口

1）向 dma_write_ctx 模块传输要写回的 MTT 表项数据

MTT 表项数据为纯 256 位宽的 FIFO

2）接收来自 dma_read_ctx 模块的读请求元数据

用于存放读请求元数据备份

Seg	index	opcode	length

Bit width	64	3	32
Value	req_num	read mpt 001/read mtt 101	长度

3）接收来自 DMA 引擎的 MTT 表项数据（dma_v2p_mpt_rd_rsp）

读响应接口：axis 接口，使用 128 位宽的 head，256 位宽的数据

4）向 DMA 写数据通道发起 Upload 写请求（dma_v2p_dt_wr_req）

写数据请求（theader）+写数据负载（tdata），即附带从 rdma 引擎获取的要写入内存的数据

DMA 引擎请求包头格式

DMA Req header	Seg	Reserved	addr	Reserved	Byte length
	Bit width	127:96	95:32	31:12	11:0
	Value	void	mpt/mtt addr	void	Byte length

5）向 DMA 写数据通道发起 Download 读数据请求（dma_v2p_dt_rd_req）

6）从 DMA 读数据响应通道获取读回的网络数据（dma_v2p_dt_rd_rsp）

获取内存中的数据响应，并将数据拼接成对应的格式，反馈到对应的 rdma 请求通道中。

5.7.1.4 MPT 模块接口

MPT 向 MTT 模块发起查询 MTT 表项请求

使用 mtt 索引查询物理地址，然后由 MTT 模块发起读、写数据（WQE、CQ 网络数据）请求

MTT Req header	Seg	Src	Op	mtt_index	V-addr	Byte length
	Bit width	164:162	161:160	159:96	95:32	31:0
	Value	见下表	Read/Write	mtt 索引	virt addr	Byte length

（注：该处的 V-addr 均为处理后的相对地址偏移）

为便于其他模块（MPT/MTT/DMA Read Data/DMA Write Data）获知正在处理 Req 所属通道，进而明确获取外部数据或者向外部输出数据的 FIFO，MPT 会在 Req 头部增加 3bit REQ_SOUR 作为标识。

REQ_SOUR 字段	Value
CEU	000
Doorbell Processing(WQE)	001
WQE Parser(WQE）	010
WQE Parser(DATA)	011
RequesterTransControl(CQ)	100
RequesterRecvControl(DATA)	101
Execution Engine(RQ WQE)	110
Execution Engine(DATA)	111

5.7.2 数据分布

mthca 每个条目长度为 **8 字节**的物理地址页面。MTT 表项的数量在初始化时确定。

offset	+0 Byte	+1 Byte	+2 Byte	+3 Byte	Description
00h	mtt_phy_addr[i][63:32]				物理页面地址高 32 位
04h	mtt_phy_addr[i][31:0]				物理页面地址低 32 位
08h	2 bit				Valid、Dirty

操作系统在分配 MR 对应的 mtt 表项时，以 mtt_seg 为单位进行 2 的幂次方大小来分配。举例：若分配一个 5MB 大小的 MR，设 mtt_seg 为 4，每个物理页面大小为 4KB，则一个 mtt_seg 对应的物理空间大小为 16KB，这样在实际的分配过程中，由于 2^8*16KB=4MB，2^9*16KB=8MB，则实际需要分配 2^9 个 mtt_seg，2^9*4 个 mtt 表项。

5.7.3 子模块设计

5.7.3.1 mtt_ram

用于存放 MTT 表项，暂时使用两路 RAM。目前实现的 mtt_ram，暂时使用阻塞的方式，不区分不同请求通道的 mtt 请求。

功能

（1）每次读、写一个 Cache Line 中的一个或多个 MTT 表项（通过 offset 字段标记起始地址，通过 num 字段标记该 Cache 行中自 offset 起始的 MTT 表项数量）；

（2）读 MTT 表项缺失时：a、将读缺失状态 state 信息返回给 mtt_ram_ctl 模块，阻塞后续请求；b、向 mttmdata 模块发起读请求（以 Cache line 为粒度）；c、等待读取的是 MTT 表项返回后，refill 表项，并把数据输出到 mtt_ram_ctl 模块

（3）读 MTT 表项命中时：a、将读命中状态 state、读命中的数据 hit_data 通过返回给 mtt_ram_ctl 模块；

（4）写 MTT 表项命中：a、将写命中状态 state 返回给 mtt_ram_ctl 模块；b、更新对应的 MTT Cacheline

（5）写 MTT 表项缺失：a、将写缺失状态 state 返回给 mtt_ram_ctl 模块；b、根据 LRU 替换策略将要替换出去的脏 MTT 表项写入 dma_wr_mtt FIFO 以备 dma_write_ctx 模块发起 dma 写操作；c、根据 LRU 替换策略使用将要替换出去的脏 MPT 表项的索引地址向 mttmdata 模块发起写内存的请求；d、从内存中读取缺失的 Cacheline 内容；e、使用新的 MTT 数据更新对应的 Cacheline 中的。

接口

输出可查询信号：lookup_allow_in

输入查询信息：

信号	I/O	位	描述	对接模块

		宽		
lookup_rden	In	1	读使能	mtt_ram_ctl
lookup_wren	In	1	写使能	mtt_ram_ctl
lookup_wdata	In	32B	写入数据	mtt_ram_ctl
lookup_offset	In	2	Cache Line 内首个 mtt 的偏移	mtt_ram_ctl
lookup_num	In	3	同一 Cache Line 中读写 mtt 的数量	mtt_ram_ctl
lookup_index	In	12	查询索引	mtt_ram_ctl
lookup_tag	In	3	查询 tag	mtt_ram_ctl
lookup_stall	In	1	阻塞 mtt 查询流水	mtt_ram_ctl
mtt_eq_addr	In	1	物理地址	mtt_ram_ctl

输出查询结果：

信号	I/O	位宽	描述	对接模块
lookup_state	Out	3	{miss，hit，idle}	mtt_ram_ctl
lookup_ldst	Out	1	1 write/0 read	mtt_ram_ctl
state_valid	Out	1	1 查询状态有效；0 查询状态无效（stall）	mtt_ram_ctl
hit_rdata	Out	32B	输出读命中 MTT 数据	mtt_ram_ctl
miss_rdata	Out	32B	输出读缺失后，从 dma 返回的数据	mtt_ram_ctl
dma_wr_mtt FIFO	Out	32B	输出要写回的 mtt 负载（rd、dout，empty）	dma_write_ctx
mttm_req FIFO	Out	99	输出缺失 mtt 的读请求 输出写回 mtt 的写请求（rd、dout，empty）	mttmdata

输入 dma 读响应信息：

信号	I/O	位宽	描述	对接模块
dma_v2p_mtt_rd_rsp axis 信号	In	256	{tvalid、tdata、tlast、thead}	dma_engine

地址索引字段计算

- valid-addr 有效索引长度计算：MTT 总条目数量为 2^18，使用 64 位 mtt_seg 查询的有效索引长度为 18 位。
- offset 字段长度计算：Cache 行大小为 32B，而 MTT 条目大小为 8B，因此要 Cache 行内的 offset 字段长度为 2 位。（注：为满足一次可以读写一个 Cache line 中的多个 mtt 表项，添加 num 字段——2 位，用于标记读取自 offset 之后的多少个 mtt）
- Index 字段长度计算：log2（C/2/w/32）；（***注：该字段直接用于读写 RAM 的地址***）
- tag 字段长度计算：valid-addr – offset - log2（C/2/w/32）= 16 - log2（C/2/w/32）

具体字段分布如下：取 C=1MB，w=2

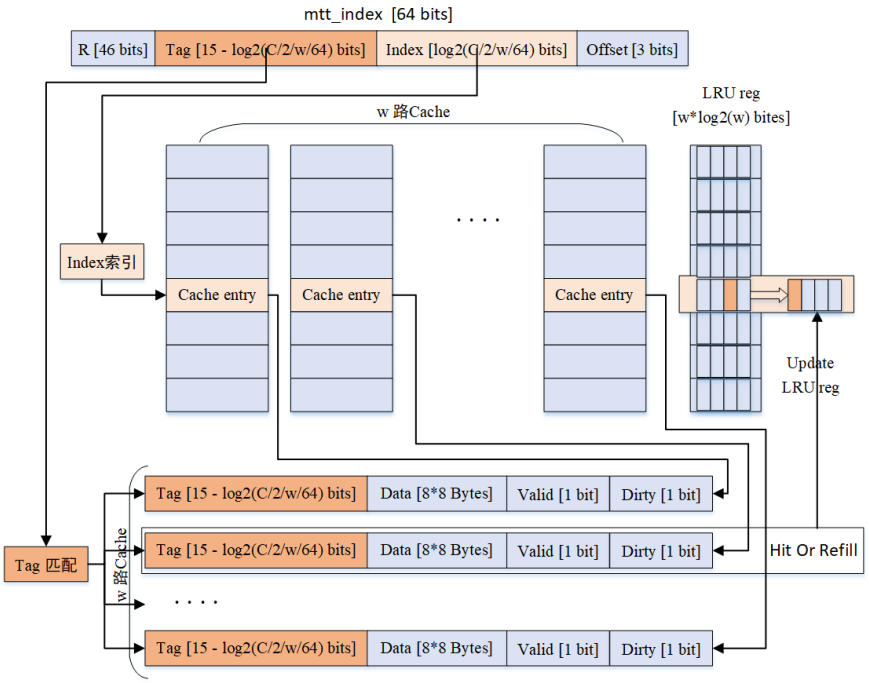
seg in mtt_index	reserved	tag	index	offset
width	46	16 - log2（C/2/w/32）	log2（C/2/w/32）	2
width	46	3	13	2

RAM 存储分布

该设计中使用 2 路 BRAM，每一路 BRAM 含有 4 个 Bank，每个 Bank 的宽度 8B，即每个 Bank 的一个条目刚好为一个 mtt 表项的大小，4 个 Bank 组成一个 Cache line，满足每次可以同时读取同一 Cache line 中的多个 mtt 表项。

way0：bank3，bank2，bank1，bank0；分别对应 offset：3，2，1，0

way1：bank7，bank6，bank5，bank4；分别对应 offset：3，2，1，0



处理流程

- Look up：读取 tag 和 valid 位，用于 update 阶段判断是否命中及有效；读出所有路的 Data，将数据（读取的 Cache 数据，index，tag，num，操作请求，要写的数据）传递给下一流水级，供第二阶段命中时返回对应的命中数据，以及 Miss 时，向 mttmdata 发起读请求，第三阶段替换时传递出去。
- Match & Update & Refill：获取上一级流水传递的操作类型、请求索引、num、读到的 tag valid 信息以及各路 data，判断是否有效、是否命中。

  （1）读命中时，返回 hit 命中信号，输出命中的 mtt data，更新 LRU 信息留到下一流水级做（该流水级只对 LRU 进行读操作，对 LRU 的写操作均在下一流水级，避免写冲突）；

  （2）读缺失时，输出 miss 信号，暂存缺失的索引地址、路号、num，进行如下操作：a）向 mttmdata 模块发起 MTT 读请求；b）等待 dma 引擎返回 mtt 表项数据，

并将返回的 mtt 数据输出给控制模块；c）refill 时将备份的缺失地址信息和返回的 mtt 数据传给下一流水级，以备填入到对应的表项中；d）拉高 LRU、Dirty 的读使能信号以及将旧的 Cache 数据传到下一流水级以备替换。

（3）写命中时，返回 write hit 命中信号，不做其他处理。将要写的数据传递给下一流水级，在下一流水级更新 LRU、Dirty 信息，将数据写入命中的位置；（仅在一个流水级中做数据写操作，这样不会发生写冲突）

（4）写缺失时，返回 write miss 信号，暂存缺失的索引地址、路号、num，进行如下操作：a）向 mttmdata 模块发起 MTT 读请求；b）等待 dma 引擎返回 mtt 表项数据，并将返回的 mtt 数据输出给控制模块；c）拉高 LRU、Dirty 的读使能信号以及将旧的 Cache 数据传到下一流水级以备替换；d）将返回的新 mtt 数据传递给下一流水级，以备拼接成新的 Cacheline 写入 Cache 中。

- Select & Replace & Refill：

（1）当读缺失发生时，在该流水级进行 replace，根据 LRU 以及 valid 信息，选出被替换路的 Cache 行，将从上一流水级接收到的对应的某一 Cache 行替写回；同时根据 LRU 选择对应的路，使用新返回的 mtt 数据填入到对应的表项中，然后更新 Tag、Valid、LRU；

（2）当写缺失发生时，在该流水级进行 replace，根据 LRU 以及 valid 信息，选出被替换路的 Cache 行，将从上一流水级接收到的对应的某一 Cache 行替写回；同时根据 LRU 选择对应的路，使用新返回的 mtt 数据与旧的 Cache 数据拼接成新的 Cacheline 填入到对应的表项中，然后更新 Tag、Valid、Dirty、LRU；

（3）若是写命中，则将数据放入命中的 cache，同时更新 Dirty 和 LRU；

（4）若是读命中，将对应的 LRU 寄存器更新。

	Stage	**Look up**	**Match & Update & Refill**				**Select & Replace & Refill**				
域	Tag	读2路的Tag	Read Hit	Read Miss (wait refill)	Write Hit	Write Miss (wait refill)	Selec t& Replace	Read Hit	Read Miss 更新	Write Hit	Write Miss 更新
	V	读2路的V	-	-	-	-	-		更新		更新
	MTT Data	读2路Data	输出	返回后输出	传递	传递	-		更新	更新	更新
	Dirty	-	-	读Dirty	-	读Dirty	miss时，选择LRU, 且Dirty=1，V=1			更新	更新
	LRU	-	–	读LRU	-	读LRU		更新	更新	更新	更新
存	读缺失地址	-	–	备份	–	备份	–	–			–
DMA信号	DMA读请求（mttmdata）	-	–	输出读Miss MTT请求	–	输出读Miss MTT请求	–				
	DMA读响应	-	–	读取MTT数据	–	读取MTT数据	–				
	DMA写请求	-	–	–	–		写替换MTT请求	-			
	DMA写负载	-	–	–	–		替换的MPT负载	–			

注：

（1）橘红色底纹表示需要与外部模块通过 FIFO 对接，FIFO 的 prog_full 信号需要作为

流水级之间握手信号之一；

（2）高亮的表示，在流水级 2 和流水级 3 中都会有可能对 mttmdata 模块发起请求，因此当流水级 2 中发生 write miss 时，需要对流水线进行阻塞，防止流水级 2 和流水级 3 对同一 FIFO 写，引发写竞争。

5.7.3.2 mtt_ram_ctl

与 ceu_parser 、 mpt_rd_req_parser 、 mpt_rd_wqe_req_mtt_cl 、 mtt_req_scheduler 、 mpt_wr_req_parser、mtt_ram、dma_read_data、dma_write_data 和 request channel 中的数据传输 FIFO 对接，将外部的数据访问请求，以 mtt_ram 接口的形式放到对应的 reg 中。同时将请求中的关键信息以及通道信息放入 FIFO 中，然后根据查询的信息将上行网络数据或下行的 WQE 和网络数据分别放到对应的不同的数据通道中。

接口

信号	I/O	位宽	描述	对接模块
ceu req header FIFO	In	128	ceu 无效 mtt 包头	ceu_parser
ceu req payload FIFO	In	256	ceu 写 mtt 表项负载	ceu_parser
mtt_base_addr	In	64	mtt 表虚拟基址	mttmdata
mpt_wr_req_mtt_cl FIFO	In	198	mpt 查询 mtt cacheline 请求	mpt_wr_req_parser
mpt_rd_req_mtt_cl FIFO	In	198	mpt 查询 mtt cacheline 请求	mpt_rd_req_parser
mpt_rd_wqe_req_mtt_cl FIFO	In	198	mpt 查询 mtt cacheline 请求	mpt_rd_req_parser
new_selected_channel	In	4	输入的 mtt 调度选择通道	mtt_req_scheduler
block_valid;	Out	3	输出阻塞的通道	mtt_ram_ctl
rd_wqe_block_info;	Out	198	输出需要暂存的阻塞信息	mtt_ram_ctl
wr_data_block_info;	Out	198	输出需要暂存的阻塞信息	mtt_ram_ctl
rd_data_block_info;	Out	198	输出需要暂存的阻塞信息	mtt_ram_ctl
unblock_valid;	Out	3	读入的阻塞的通道	mtt_ram_ctl
rd_wqe_block_reg;	In	198	读入需要处理的阻塞信息	mtt_ram_ctl
wr_data_block_reg;	In	198	读入需要处理的阻塞信息	mtt_ram_ctl
rd_data_block_reg;	In	198	读入需要处理的阻塞信息	mtt_ram_ctl
mtt_read_wqe FIFO	Out	134	dma 读 WQE 请求	dma_read_wqe
mtt_read_data FIFO	Out	134	dma 读数据请求	dma_read_data
mtt_write_data FIFO	Out	134	dma 写数据请求	dma_write_data

lookup_allow_in	In	1	可以查询信号	mtt_ram
lookup_rden	Out	1	读使能	mtt_ram
lookup_wren	Out	1	写使能	mtt_ram
lookup_wdata	Out	32B	写入数据	mtt_ram
lookup_index	Out	12	查询索引	mtt_ram
lookup_tag	Out	3	查询 tag	mtt_ram
lookup_offset	Out	2	Cache Line 内首个 mtt 的偏移	mtt_ram
lookup_num	Out	3	Cache Line 中读写 mtt 的数量	mtt_ram
lookup_stall	Out	1	阻塞 mtt 查询流水	mtt_ram
lookup_state	In	3	{miss，hit，idle}	mtt_ram
lookup_ldst	In	2	1 write/0 read	mtt_ram
state_valid	In	1	1 查询状态有效；0 查询状态无效（stall）	mtt_ram
hit_rdata	In	32B	输出读命中的 mtt 数据	mtt_ram
miss_rdata	Out	32B	读缺失，从 dma 返回的数据	mtt_ram

1、ceu 请求处理状态机

ceu_req_idle state：

空闲状态。若 ceu 的请求 FIFO 非空，则进入 ceu_req_proc 状态进行 ceu 的请求处理；否则，则继续处于空闲状态。

ceu_req_proc state：

读取：（1）ceu 请求包头；（2）ceu 请求负载

输出：（1）向 mtt_ram 模块对应的信号线上输出 ram 操作请求

若 ceu 的请求 FIFO 非空，则依然处于 ceu_req_proc 状态进行 ceu 的请求处理；否则进入空闲状态。

关键问题：（1）CEU 对 mtt_ram 请求拆分问题

设置 left_mtt_num、payload_offset、ceu_req_mtt_addr、get_ceu_payload（暂存 FIFO 输出数据，可以与 FIFO dout 数据拼接）变量。共四种情况决定是否读取 ceu_payload FIFO：

- 同一 ceu req 的最后一个 mtt_ram 请求，若有新请求需要读取新 req 的 payload：
  left_mtt_num+ceu_req_mtt_addr[1:0] <= 4 （一次 mtt 请求即可）
  payload_offset + left_mtt_num <= 4 （无需拼接数据）
- 同一 ceu req 的 mtt_ram 请求处理中：
  left_mtt_num+ceu_req_mtt_addr[1:0] > 4 （剩余多次 mtt 请求）
  payload_offset + 4- ceu_req_mtt_addr[1:0] >= 4（需要拼接数据，即该次 mtt_ram 请求用尽 get_ceu_payload）
- 同一 ceu req 的最后一个 mtt_ram 请求，但需要将旧的数据从 FIFO 中读出：

left_mtt_num+ceu_req_mtt_addr[1:0] <= 4（一次 mtt 请求即可）

payload_offset + left_mtt_num > 4（需要拼接数据，即该次 mtt_ram 请求用尽 get_ceu_payload）

2、mpt 查询处理状态机

mpt 请求处理状态机的设计简单化，不需要考虑跨 Cacheline 的问题，跨 Cacheline 的问题在 mpt_req_parser 中解决，mpt_req_parser 中需要提前对 DMA 的请求的数据长度以及 MTT_ram 访问的偏移、数量进行拆分。并提供读写主存数据的标志位以及 RDMA 引擎请求通道的标志位。

rd_mpt_req state：

优先判断 block_reg 是否有值，若有阻塞信息，则优先读取；若无阻塞信息则从对应的 fifo 中读请求信息，并且目前没有 ceu 的请求在处理（***is_ceu_req_processing***），且 mtt_ram 处于可查询状态（***lookup_allow_in***），则进入 init_lookup 发起查询 mtt_ram 请求。

（注：需要考虑 is_ceu_req_processing，因为当有 ceu 请求在处理时，会引发查询 mtt_ram 的请求，会与 mpt 请求引发的 mtt_ram 查询请求冲突）

init_lookup state：

读取：（1）寄存器中的 mpt 请求的查询信息

输出：若 mtt 与 dma_read/write_data 接口 FIFO 未满（1）将发起 lookup 查询

进入 lookup_result_proc 状态，将查询结果反馈给与 dma 引擎接口的模块处理。

lookup_result_proc state：

读取：（1）mtt_ram 输出的查询结果；

输出：（1）维护同一请求的标记和操作，将请求拆分成 4K 页面为界限的 dma 读写请求并发往 dma_read_data、dma_read_wqe、dma_write_data 接口 FIFO。

若同一请求处理完毕，则进入 **rd_mpt_req;** 若同一请求的处理未完毕，但是输出的通道已经 full，此时将未处理完的请求放到对应 fifo 的寄存器中，然后进入 **rd_mpt_req**，待调度新的请求处理。

5.8 MPT Parser

5.8.1 功能

将一个 MPT 请求所衍生出来的所有读或者写主存数据的 MTT 请求拆分成写、读 WQE、读数据，进行细粒度的调度。保证查询不阻塞。

具体如下：

（1） 将 mpt_ram_ctl 模块输出到 mtt_ram_ctl 的 mpt_req_mtt fifo 通道分解成 mpt_rd_req_parser, mpt_rd_wqe_req_parser, mpt_wr_req_parser 3 个三个 fifo 通道。

（2） 在图 6-1 的红框所示部分 mpt_ram_ctl 和 mtt_ram_ctl 之间增加 MPT 请求拆解模块 mpt_req_parser（分为 mpt_rd_req_parser、mpt_rd_wqe_req_parser 和 mpt_wr_req_parser 3 个子模块分别处理读数据、读 WQE 和写请求的拆解）

（3） MTT 请求调度模块 mtt_req_scheduler。分别将拆解的请求进行调度。

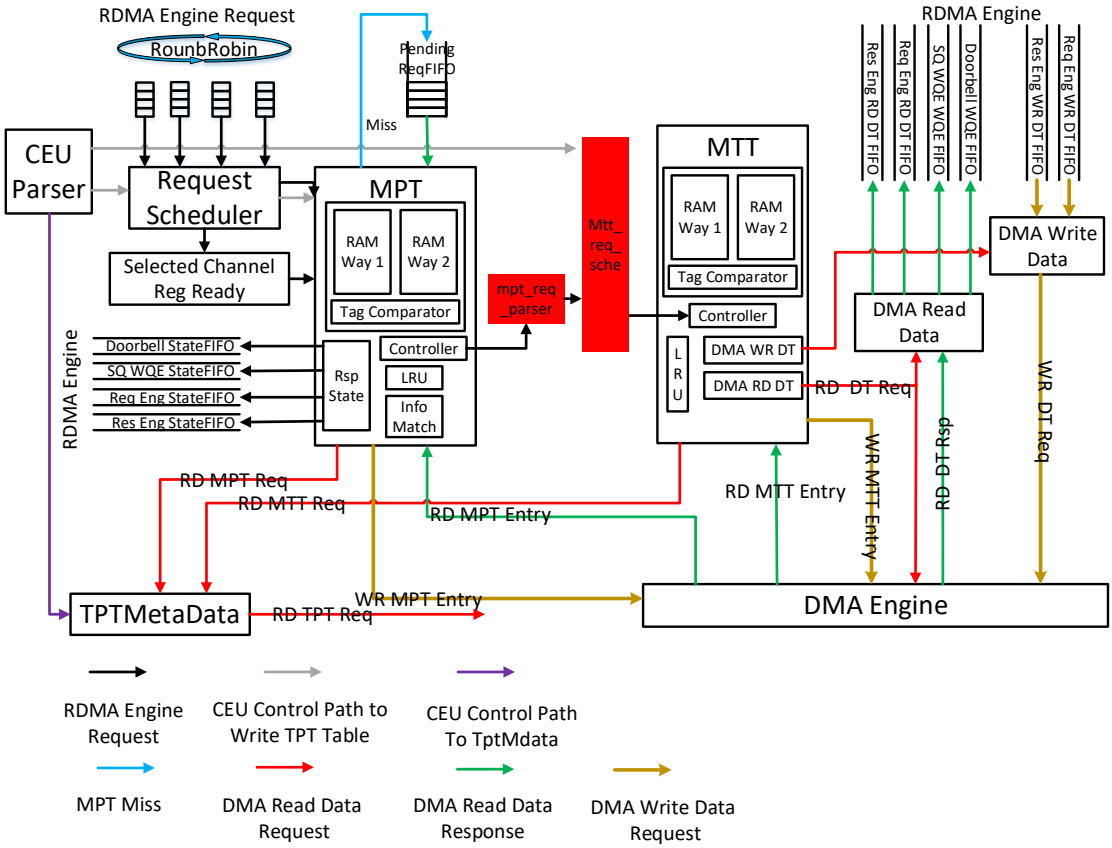


图 6-2 MPT Parser 与 MTT Cache 连接关系

5.8.2 MPT 模块接口

使用 mtt 索引查询物理地址，然后由 MTT 模块发起读、写数据（WQE、CQ 网络数据）请求

MTT Req header	Seg	Src	mtt_index	V-addr	Byte length
	Bit width	162:160	159:96	95:32	31:0
	Value	见下表	mtt 索引	virt addr	Byte length

（注：该处的 V-addr 均为处理后的相对地址偏移）

为便于其他模块（MPT/MTT/DMA Read Data/DMA Write Data）获知正在处理 Req 所属通道，进而明确获取外部数据或者向外部输出数据的 FIFO，MPT 会在 Req 头部增加 3bit REQ_SOUR 作为标识。

REQ_SOUR 字段	Value
CEU	000
Doorbell Processing(WQE)	001
WQE Parser(WQE）	010
WQE Parser(DATA)	011
RequesterTransControl(CQ)	100

RequesterRecvControl(DATA)	101
Execution Engine(RQ WQE)	110
Execution Engine(DATA)	111

5.8.3 子模块设计

5.8.3.1 mpt_rd_req_parser

与 mpt_ram_ctl 中的 mpt_rd_req_mtt FIFO 对接，将外部的数据访问请求，以 mtt cache line 为单位向 mtt_ram_ctl 的 FIFO 写入请求接口。

接口

信号	I/O	位宽	描述	对接模块
mpt_rd_req_mtt FIFO	In	163	mpt 查询 mtt 请求	mpt_ram_ctl
mpt_rd_req_mtt_cl FIFO	Out	198	mpt 查询 mtt cacheline 请求	mtt_ram_ctl

mpt_rd_req_mtt_cl FIFO 格式

Seg	Src	total length	opcode	mtt_index	V-addr	tmp length
Bit width	197:195	194:163	162:160	159:96	95:32	31:0
Value	同 6.3.1 中的 REQ_SOUR 表	总数据长度	RD_DT/ RD_DT_FIRST	mtt 索引	virt addr	Byte length

状态机

rd_mpt_req state：

若 mpt_rd_req_mtt FIFO 非空，mpt_rd_req_mtt_cl 未满，则读取 mpt_req_mtt FIFO 中的请求，进入 init_lookup 发起查询 mtt_ram 请求。

init_lookup state：

读取：（1）寄存器中的 mpt 请求的查询信息

输出：若 mpt_rd_req_mtt_cl 未满，将 mpt_ram_ctl 发出的请求拆分成以 mtt cacheline 为单位的请求，写入 mpt_rd_req_mtt_cl。

5.8.3.2 mpt_rd_wqe_req_parser

与 mpt_ram_ctl 中的 mpt_rd_wqe_req_mtt FIFO 对接，将外部的数据访问请求，以 mtt cache line 为单位向 mtt_ram_ctl 的 FIFO 写入请求接口。

接口

信号	I/O	位宽	描述	对接模块
mpt_rd_wqe_req_mtt FIFO	In	163	mpt 查询 mtt 请求	mpt_ram_ctl

mpt_rd_wqe_req_mtt_cl FIFO	Out	198	mpt 查询 mtt cacheline 请求	mtt_ram_ctl

mpt_rd_wqe_req_mtt_cl FIFO 格式

Seg	Src	total length	opcode	mtt_index	V-addr	tmp length
Bit width	197:195	194:163	162:160	159:96	95:32	31:0
Value	同 6.3.1 中的 REQ_SOUR 表	总数据长度	RD_WQE/ RD_WQE_FIRST	mtt 索引	virt addr	Byte length

状态机

rd_mpt_req state：

若 mpt_rd_req_mtt FIFO 非空，mpt_rd_wqe_req_mtt_cl 未满，则读取 mpt_rd_wqe_req_mtt FIFO 中的请求，进入 init_lookup 发起查询 mtt_ram 请求。

init_lookup state：

读取：（1）寄存器中的 mpt 请求的查询信息

输出：若 mpt_rd_wqe__req_mtt_cl 未满，将 mpt_ram_ctl 发出的请求拆分成以 mtt cacheline 为单位的请求，写入 mpt_rd_wqe_req_mtt_cl。

5.8.3.3 mpt_wr_req_parser

与 mpt_ram_ctl 中的 mpt_wr_req_mtt FIFO 对接，将外部的数据访问请求，以 mtt cache line 为单位向 mtt_ram_ctl 的 FIFO 写入请求接口。

接口

信号	I/O	位宽	描述	对接模块
mpt_wr_req_mtt FIFO	In	163	mpt 查询 mtt 请求	mpt_ram_ctl
mpt_wr_req_mtt_cl FIFO	Out	198	mpt 查询 mtt cacheline 请求	mtt_ram_ctl

mpt_wr_req_mtt_cl FIFO 格式

Seg	Src	total length	opcode	mtt_index	V-addr	tmp length
Bit width	197:195	194:163	162:160	159:96	95:32	31:0
Value	同 6.3.1 中的 REQ_SOUR 表	总数据长度	WR_DT/ WR_DT_FIRST	mtt 索引	virt addr	Byte length

5.8.3.4 mtt_req_scheduler

按照 Round-Robin 的方式将拆解的来自 mpt_wr_req_mtt_cl FIFO，mpt_rd_req_mtt_cl FIFO，mpt_rd_wqe_req_mtt_cl FIFO 的 mtt cacheline 级别的请求进行调度。（RoundRobin）

接口

信号	I/O	位宽	描述	对接模块
mpt_wr_req_mtt_cl FIFO	In	198	empty、rd_en 信号	mpt_wr_req_parser
mpt_rd_req_mtt_cl FIFO	In	198	empty、rd_en 信号	mpt_rd_req_parser
mpt_rd_wqe_req_mtt_cl FIFO	In	198	empty、rd_en 信号	mpt_rd_req_parser
new_selected_channel	Out	4	输出的调度选择通道	mtt_ram_ctl
block_valid;	in	3	阻塞的通道	mtt_ram_ctl
rd_wqe_block_info;	in	198	输入需要暂存的阻塞信息	mtt_ram_ctl
wr_data_block_info;	in	198	输入需要暂存的阻塞信息	mtt_ram_ctl
rd_data_block_info;	in	198	输入需要暂存的阻塞信息	mtt_ram_ctl
unblock_valid;	in	3	读取的阻塞的通道	mtt_ram_ctl
rd_wqe_block_reg;	Out	198	输出暂存的阻塞信息	mtt_ram_ctl
wr_data_block_reg;	Out	198	输出暂存的阻塞信息	mtt_ram_ctl
rd_data_block_reg;	Out	198	输出暂存的阻塞信息	mtt_ram_ctl

block_valid;　说明 | bit 2 read WQE | bit 1 write data | bit 0 read data |

unblock_valid; 说明 | bit 2 read WQE | bit 1 write data | bit 0 read data |

new_selected_channel 说明：

Bit	Description
3	valid
2	mpt_rd_wqe_req_mtt_cl
1	mpt_wr_req_mtt_cl
0	mpt_rd_req_mtt_cl

逻辑：

block_valid 对应的通道阻塞，则将阻塞信息填入 block_reg

unblock_valid 对应的通道有效，则将阻塞信息的寄存器 block_reg 清零

每次使用轮询的方式调度三个通道：mtt_ram_ctl 对每个调度的通道优先判断 block_reg 是否有值，若有阻塞信息，则优先读取处理；若无阻塞信息则从对应的 fifo 中读请求信息进行处理。