ICT NCIC 系统互连组

DMA 引擎设计文档

v1.5

Revision History

Date	Author	Comment	Version
29/10/2020	康宁	1. 完成了 DMA 引擎的基本功能	V1.0
23/06/2021	康宁	1. 添加了 Head 接口的字段说明	V1.1
07/05/2022	康宁	1. 修改了 DMA 读响应端处理逻辑	V1.2
23/06/2022	康宁	1.更新了对 DMA tag 数目选择的说明	V1.3
28/06/2022	康宁	1.更新了新版读通道设计方案	V1.4
01/11/2022	康宁	1.使用多个 tag 队列进行已分配 tag 的暂存	V1.5

1 DMA 引擎总体设计

1.1 需求分析

在 IB HCA 中，DMA 引擎是 HCA 访问内存的接口模块，其完成了跨时钟域处理、请求的分包、对齐、格式转换，响应的重排，以及响应包的重组等工作。为便于后期 HCA 与其它设备（如专用硬件加速器等）进行连接，HCA 中需要一个统一的数据传输接口层进行对接。因此，我们设计并实现了一个 DMA 引擎，下面将对 DMA 引擎的设计进行详细说明。

1.2 总体设计

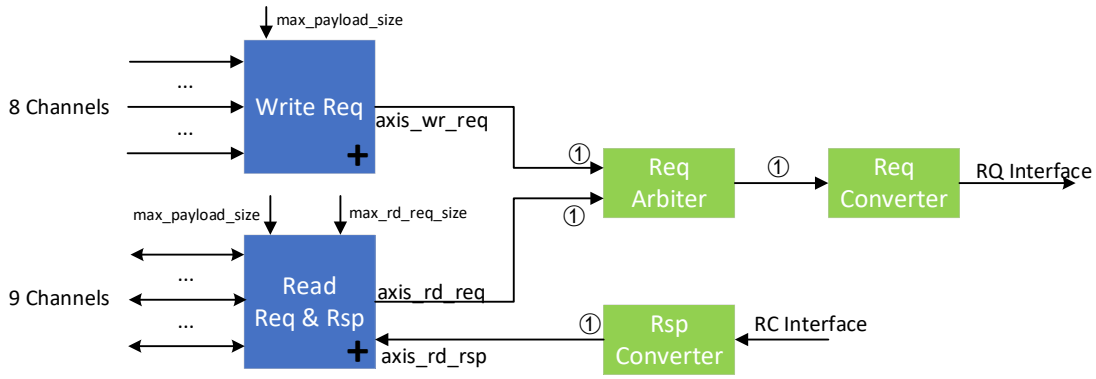

图 1-1 DMA 引擎总体结构

如图 1-1 所示，为 DMA 引擎的总体结构。DMA 引擎共分为 5 个模块，分别为 DMA 写请求处理模块（Write Req），DMA 读请求响应处理模块（Read Req & Resp），请求仲裁模块（Req Arbiter），请求转换模块（Req Converter）及响应转换模块（Rsp Converter）。其中，DMA 写请求处理模块用于处理 HCA 中的内存写请求；DMA 读请求相应模块用于处理 HCA 中的内存读请求以及相应的读响应；请求仲裁模块用于仲裁 HCA 发出的 Host Memory 读请求和 Host Memory 写请求，发送给 PCIe 控制器；请求转换模块和响应转换模块都用于转换 AXIS 数据包格式为 PCIe 数据包格式，与 PCIe 控制器对接；响应转换模块都用于转换 PCIe 数据包格式为 AXIS 数据包格式，转发 PCIe 控制器收到的读响应。第 2 章对各个模块的设计进行了详细的说明。

1.3 对外接口及使用注意

类型	信号	in/out	位宽
写请求	dma_wr_req_valid	in	1
	dma_wr_req_last	in	1

	dma_wr_req_head	in	128
	dma_wr_req_data	in	256
	dma_wr_req_ready	out	1
读请求	dma_rd_req_valid	in	1
	dma_rd_req_last	in	1
	dma_rd_req_head	in	128
	dma_rd_req_data	in	256
	dma_rd_req_ready	out	1
读响应	dma_rd_rsp_valid	out	1
	dma_rd_rsp_last	out	1
	dma_rd_rsp_head	out	128
	dma_rd_rsp_data	out	256
	dma_rd_rsp_ready	in	1

表 1-1 DMA 引擎与 HCA 的三类接口

offset	+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
0Ch	chnl_num																								Type							
08h	Addr[63:32]																															
04h	Addr[31:0]																															
00h	Length																															

表 1-2 Head 接口字段布局

Offset	Bits	Name	Description	Access
00h	31:0	Length	请求数据的长度	WR
04h	64	Addr	请求数据的起始地址	WR
0Ch	7:0	Type	类型字段，标志请求类型	WR
0Ch	31:24	Chnl_num	该通道的通道号，由 DMA Engine 自动添加	WR

表 1-3 Head 接口字段说明

作为 HCA 内存访问接口，DMA 引擎一方面与 HCA 进行交互，获取读写请求、返回读响应，另一方面与 PCIe 控制器交互，完成读写请求及响应的转发。因此，DMA 引擎接口可以分为两个部分：与 HCA 交互部分和与 PCIe 交互部分。与 HCA 交互部分包括三类接口：写请求接口、读请求接口、读响应接口。三类接口信号见表 1-1。其中，写请求接口、读请求响应接口的数量可以任意指定（读请求接口和读响应接口数量一致，但可以与写请求接口数量不同）。在图 1-1 中，我们假定有四个写请求接口以及四个读请求响应接口， 分别与 HCA 的 CEU、Context Management、Virtual to Physics（context）、Virtual to Physics（Data）连接。为便于解释，在下文中，我们将对 DMA 引擎与 HCA 的接口数量及连接做同样的假定。

类型	信号	in/out	位宽
写请求	m_axis_rc_tvalid	in	1
	m_axis_rc_tlast	in	1
	m_axis_rc_tuser	in	128
	m_axis_rc_tkeep	in	8
	m_axis_rc_tdata	in	256
	m_axis_rc_tready	out	1
读请求	s_axis_rq_tvalid	out	1
	s_axis_rq_tlast	out	1
	s_axis_rq_tuser	out	128
	s_axis_rq_tkeep	out	8
	s_axis_rq_tdata	out	256
	s_axis_rq_tready	in	1

表 1-4 DMA 引擎与 PCIe 的接口

与 PCIe 交互部分的接口目前采用 Virtex-7 FPGA Gen3 Integrated Block for PCI Express v2.2 IP 的接口，该接口信号见表 1-2。接口含义及其时序关系见 Xilinx 产品文档。

2 各模块设计

2.1 写请求处理模块设计

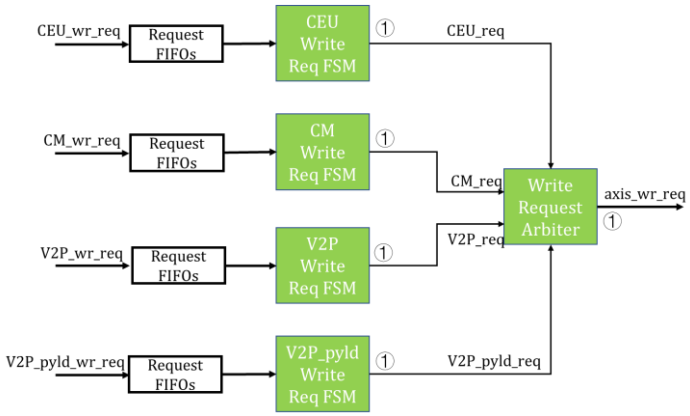

图 2-1 写请求模块总体结构

写请求模块结构如图 2-1 所示，当前有四个写请求接口。写请求处理模块包含三个子单元：写请求异步 FIFO 单元，写请求状态机单元，写请求仲裁单元。其中，写请求仲裁单元将在 2.3 节中进行说明。

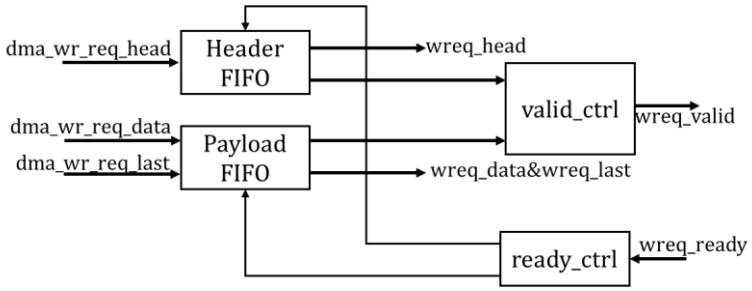

图 2-2 写请求 FIFO 单元结构

写请求异步 FIFO 单元用于跨时钟域的处理操作，目前由于 HCA 的时钟频率低于 PCIe，为保证 PCIe 可以连续收到一个整包，该单元采用“存储转发”策略，即存完一个整包后（读到 last 位有效）再进行输出。为支持乒乓操作，该 FIFO 单元存储空间的大小为 8KB（两个页）。写请求 FIFO 单元的结构如图 2-2 所示。

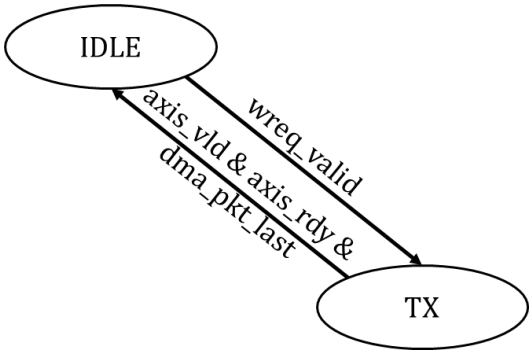

图 2-3 写请求状态机状态转换图

写请求状态机单元写请求完成的功能包括：数据双字对齐（数据位宽重组）、按照 PCIe 最大 payload 大小进行分包、元数据格式改写。写请求状态机单元包括一个写请求状态机，以及一个数据位宽重组流水线。其中，写请求状态机状态转换图如图 2-3 所示。该状态机结合剩余发送数据量寄存器，共同实现了数据包的分割。数据位宽重组流水线采用输入寄存器、中间寄存器、输出寄存器三级流水线的方式（见图 2-4），实现了 HCA 写请求数据的双字对齐。

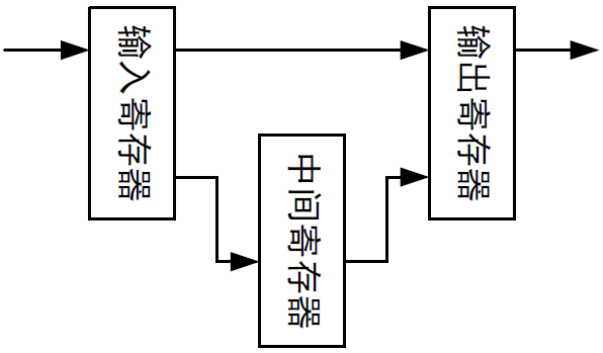

图 2-4 数据位宽重组三级流水线结构

2.2 读请求响应处理模块

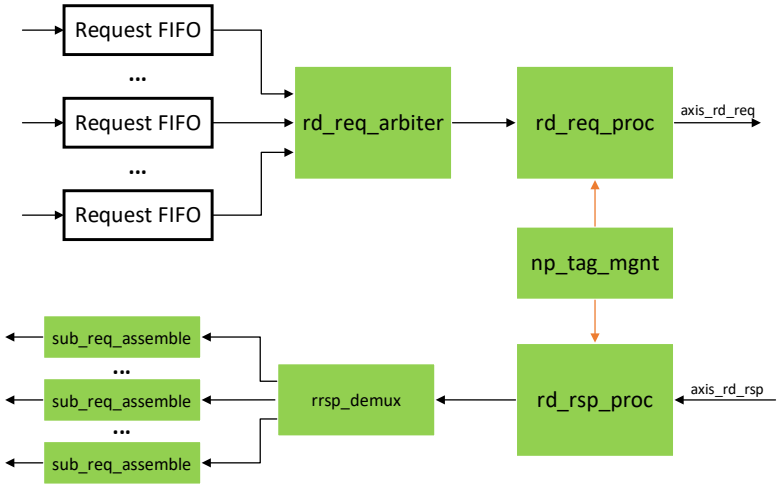

图 2-5 读请求响应处理模块结构

读请求响应处理模块结构如图 2-5 所示，当前包含 9 个读通道。读请求响应处理模块包含 7 个子单元：读请求 FIFO、读请求仲裁器单元、读请求处理单元、tag 管理单元、读响应处理单元，以及 reorder buffer。其中，读请求 FIFO 中需要添加 chnl_num 字段，用来标志该请求所属的通道；读请求仲裁单元，与通用的请求仲裁模块相同，将在 2.3 节进行说明；读请求处理单元（rd_req_proc），用来对请求进行 DW 对齐，获得 tag 标签，并按照 max_rd_req_size 分割请求；tag 管理单元（np_tag_mgnt），用来为读请求分配 tag，回收读响应的 tag，并暂存该请求的 dw_len，chnl 信息；读响应处理单元（rd_rsp_proc），用来将收到的读响应包根据 tag 标签存储到对应的缓冲区中，同时合并多个子响应包为一个响应包；dma_demux，将存储的响应分配到对应通道中去；子请求重组单元（sub_req_assemble），将多个子请求的响应包合并为同一个响应包。

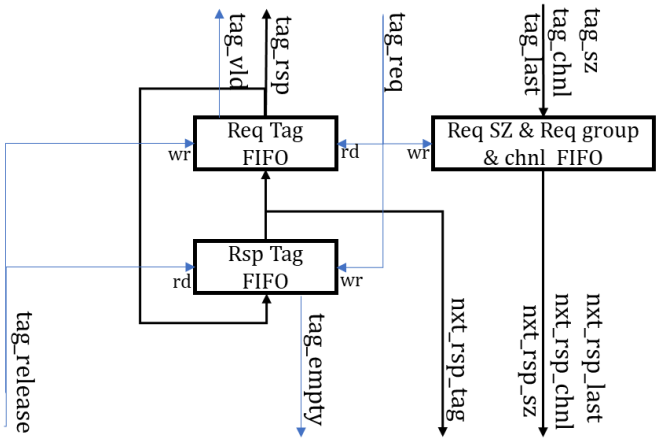

图 2-6 tag 管理单元

Tag 管理单元结构如图 2-6 所示，该单元内部共包含 3 个子结构：Req Tag Fifo，Rsp Tag Fifo，以及 Req SZ & Req group & chnl FIFO。Req Tag Fifo 存储待分配的 tag，需要在初始化时进行预分配；Rsp Tag FIFO 中存储着已分配的 tag; Req SZ & Req group & chnl FIFO 存储一些与请求该 tag 相关的边带信息，包括请求的 dw_len，chnl，以及是否为最后一个子请求的信息。

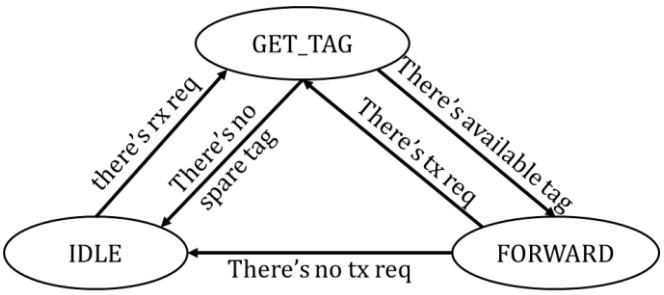

图 2-7 读请求处理结构状态转移

读请求处理结构主要完成的功能包括：读请求元数据格式的改写、向 tag 管理结构申请 tag、读请求按照 PCIe 最大请求限制拆分。读请求处理结构状态转换图如图 2-7 所示。该结构共包含三个状态：IDLE、GET_TAG、FORWARD。IDLE 状态主要用于等待 HCA 发起读请求操作，当收到一个读请求时，跳转到 GET_TAG 状态；GET_TAG 状态主要用于获取 tag，当获取到一个有效的 tag 时，跳转到 FORWARD 状态，同时，该状态还对读请求改写为双子对齐格式；FORWARD 状态主要用于转发读请求，若请求大小超过最大限制，则将其拆分为多个请求包。该单元与原来的 read_request 单元几乎相同，只是添加了通道字段，并在 tag 请求的过程中将这一字段发送给了 np_tag_mgnt 模块。

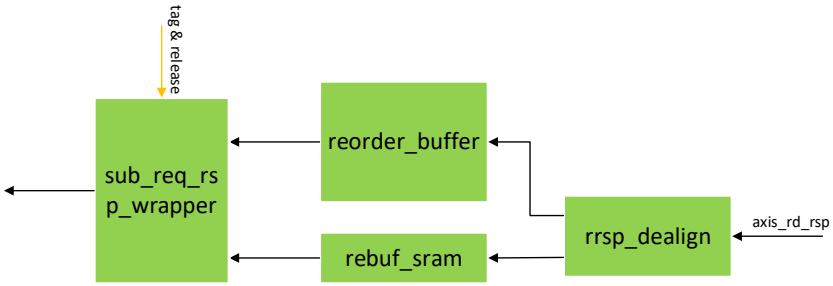

图 2-8 响应处理单元

读响应处理单元主要用于接收读响应的子响应包（包括转换元数据格式）、并将其缓存到 reorder buffer 中。如图 2-8 所示，该单元共包含四个子结构，分别为 rrsp_dealign，reorder_buffer，rebuf_sram，subreq_rsp_wrapper。

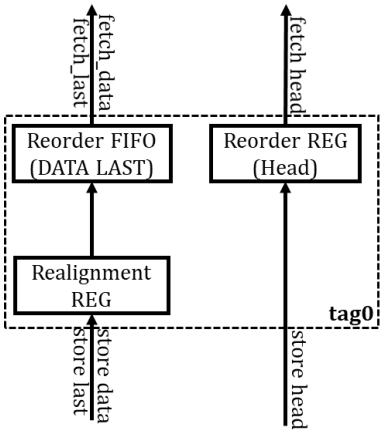

图 2-9 响应数据重排缓存结构

响应数据重排缓存结构（tag_rebuf）主要用于缓存重排的响应包、缓存并重组同一请求的多个子响应包。响应数据重排缓存结构的结构如图 2-9 所示，该结构包括数据包头的存储、数据的存储、以及一个位宽重组流水线（类似于图 2-4 所示结构）。

子请求重组单元（sub_req_assemble）用于将读请求单元发送读请求时分割的多个子读请求的响应包重组。该单元复用了 tag_rebuf 的结构。

2.3 仲裁单元

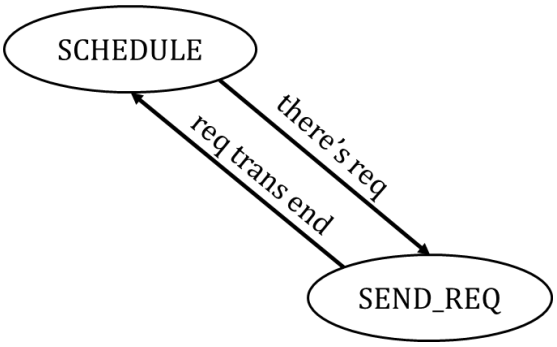

图 2-10 仲裁单元状态机状态转移

DMA 引擎的结构中共包含三个仲裁单元，这三个仲裁单元都采用相同的逻辑，即轮询仲裁。仲裁单元状态机的状态转换如图 2-10 所示。该单元状态机的状态转移包括 2 种状态：SCHEDULE、SEND_REQ。SCHEDULE 状态用于选择请求输出的端口，当有请求到来时，基于轮询策略选择合适的请求通道，并跳转到 SEND_REQ 状态；SEND_REQ 状态用于发送请求，当请求传输到最后一拍时，跳转到 SCHEDULE 状态。

3 DMA 引擎参数分析

3.1 读通道端 tag 数目的分配

在新版设计中，多个通道共享所有 tag 因此不需要考虑每个通道的 tag 数目分配。下面说明我们需要分配的 tag 数目。在 HanGu RNIC 中，我们的设计带宽为 100Gbps，PCIe 带宽为 128Gbps。根据《Understanding PCIe performance for end-host networking》，读取 128bytes 的数据大约需要 1000ns 的延时。

HanGu RNIC 中，我们需要实现 100Gbps 的 Host Memory 端数据获取能力。我们取 PCIe 的读延时为 1 μs，则 1 μs 内网卡数据读通道需要向 Host memory 发起读取的数据量至少为 100Gbps x 1 μs = 100kb。又假定 max_rd_req_size 不超过 256 bytes（128，256 bytes），即每个 tag 可以读取 256 bytes（128 bytes）的数据，那么 1 μs 内共需要分配 100kb / (256B x 8) = 50 个 tag（128 bytes 下为 100 个 tag）。综上，数据读通道需要的 tag 数量的最小值为 50 个。我们设置为 64 个。

3.2 DMA 引擎中的存储空间占用

3.2.1 DMA 写通道

对于每一个写通道，都需要占用一个 FIFO 来存储一个写请求包。该 FIFO 的大小为 384 X 128 = 49152bits。目前，DMA Engine 支持的写通道数为 8 个，所以写通道总的 SRAM 使用量为 384 X 128 X 8 = 393216 bits。

3.2.2 DMA 读通道

对于每一个读通道，在请求端，需要维护一个宽度为 DMA_HEAD_WIDTH（128bits），深度为 8 的 req_fifo，用来存储待处理的读请求。该 FIFO 的大小为 128 X 8 = 1024bits;在响应端，需要维护一个宽度为 C_DATA_WIDTH（256bits），深度为 128 的 rsp_fifo，用来存储返回的响应数据，该 FIFO 的大小为 256 X 128 = 32768bits。上述计算是在单个通道的情况下进行的计算，DMA Engine 共包含 9 个读通道，所以上述 FIFO 需要的 SRAM 的总用量为（req_fifo + rsp_fifo）* 9 = （1024 + 32768）* 9 = 304128bits。

此外，每个读通道都有大小不等的 reorder_buffer，该 buffer 大小与 tag 数量有关，每个 tag 需要占用 256 * 8 bits 的空间。根据 3.1 节所知，DMA Engine 共需 64 个 Tag，因此，reorder_buffer 占用的空间大小为 256 * 8 * 64 = 131072 bits。

综上，读通道共需要的 SRAM 的面积为：435200 bits，其中，可以使用寄存器搭建的大小为：131072 bits + 1024 bits * 9 = 140288 bits。

表 3-1 读通道面积占用

测试版本	读通道占用 SRAM 大小
base	5.94Mb (5940224 bit)
重构	2.40Mb (2401280 bit)
max_rd_req_sz 最大支持 256 byte	0.44Mb (435200 bit)

3.2.3 异步 FIFO

由于需要与 PCIe 进行交互，DMA engine 使用了 3 个异步 FIFO 用来实现与 PCIe 控制器间的跨时钟处理。这 3 个 FIFO 的大小分别为：（64+32）*32=3072bits，（8+8+256+1）* 32 = 8736bits 和（8+32+256+1）*32=9504bits。总 SRAM 占用量为：21312bits，其中，该占用量全部为使用寄存器搭建的。

3.2.4 存储空间占用总量

基于上面的分析，可以得到 SRAM 占用总量如表 3-2 所示。

表 3-2 DMA 总体存储面积占用

测试版本	DMA 占用 SRAM 大小
base	6.35Mb (6354752 bit)
重构	2.82Mb (2815808 bit)
max_rd_req_sz 最大支持 256 byte	0.85Mb (850028 bit)

3.3 DMA Engine 带宽分析

为了测试我们设计的 DMA Engine 是否可以跑满带宽，我们对 DMA Engine 进行了仿真测试。测试环境如表 3-4 所示。

表 3-4 DMA 带宽测试环境

参数名	值
pcie_clk	625MHz（1.6ns）
rdma_clk	500MHz（2ns）
读请求数据量	4096 bytes
写请求数据量	4096 bytes
读响应包最大大小	256 bytes
max_payload_size	256 bytes
max_rd_req_size	256 bytes

根据上述测试环境我们得到了如表 3-5 所示的带宽测试结果。

表 3-6 DMA Engine 带宽测试结果

测试版本	值
DMA 写（没有读）	103.78Gbps
DMA 写（包含读）	101.61Gbps
DMA 读（base）	77.36Gbps
DMA 读（优化）	90.50Gbps
DMA 读（重构）	99.94Gbps
DMA 读（base - 128 bytes）	56.94Gbps
DMA 读（优化 - 128 bytes）	56.94Gbps
DMA 读（重构 – 128 bytes）	51.79Gbps

4 DMA Engine 的后续优化

4.1 优化带宽利用率

4.1.1 读响应端乱序的带宽利用率优化

在读响应端乱序重排的过程中，在当前状态机设计中，写入乱序缓冲区和从乱序缓冲区读数的过程被合并考虑了，因此当乱序问题出现时，可能会存在阻塞流水线的问题（仅仅阻塞几个周期，不会饥饿）。

为解决这一问题，我们需要分离乱序重排缓冲区的写过程和读过程，使用两个状态机来控制这一过程。

4.2 优化存储面积

4.2.1 读响应端乱序的存储面积优化

在当前设计中，我们使用小块 FIFO 来作为重排缓冲区。然而，该方法分离了各个缓冲区，使得每个缓冲区没有办法用 SRAM 实现，只能用寄存器搭。此外，对于不同大小的 max_rd_req_sz，合并缓冲区可以在充分利用每块缓冲区的情况下，实现同样的带宽性能。