ICT NCIC 互连架构组

以太网引擎 详细设计文档

V1.2

Revision History

Date	Author	Comment
01/10/2021	李剑雄	以太网引擎详细设计文档，包括子模块的详细设计方案
15/12/2021	李剑雄	更新了新版设计文档
23/02/2022	李剑雄	更新新版设计文档
15/12/2022	李剑雄	更新新版设计文档

目录

图目录

表目录

1 概述

1.1 文档说明

本文档给出以太网引擎的硬件架构详细设计方案，包含寄存器管理单元、以太网和RoCE 引擎发送路径、接收路径，旨在实现以下目标：1.定义每个模块的处理功能；2.定义不同模块之间接口；3.给出每个模块的状态机实现。

1.2 整体架构

下图是网卡的总体设计，主要分为寄存器管理单元、以太网引擎、描述符管理三个部分。网卡向上与 PCIe 引擎、内核驱动进行交互，获取描述符数据，接受驱动的配置请求，进行队列信息管理，在主机内存和本地 ram 间进行数据包的接收和发送；向下与 MAC 模块和 PHY 模块对接，发送和接受完整的以太网帧。

网卡主要通过 DMA、IO 两种方式与主机交互。PCIe 引擎对 TLP 包处理后，分为 DMA、IO 两类数据，再将 DMA 与 IO 数据发送给相应模块中。系统 IO 在网卡内转换为 AXI 信号，通过 AXI crossbar 模块与网卡各个寄存器管理单元进行对接，DMA 则与网卡内数据包 FIFO 与描述符 FIFO 对接，完成数据包和描述符收发。

在接收方向，网卡从 MAC 中获取到描述符后，将数据包发往接收引擎，通过 hash 模块后，决定数据包由哪个队列进行处理，然后向相应队列获取描述符信息，将数据包 DMA 至描述符所指向的内存空间中。

在发送方向，主机通过 IO 向网卡发送数据包发送请求，通过队列调度模块调度后，将请求描述符信息发往发送引擎，发送引擎根据描述符将数据包 DMA 至网卡 FIFO，经过校验和计算后，发送给 MAC。

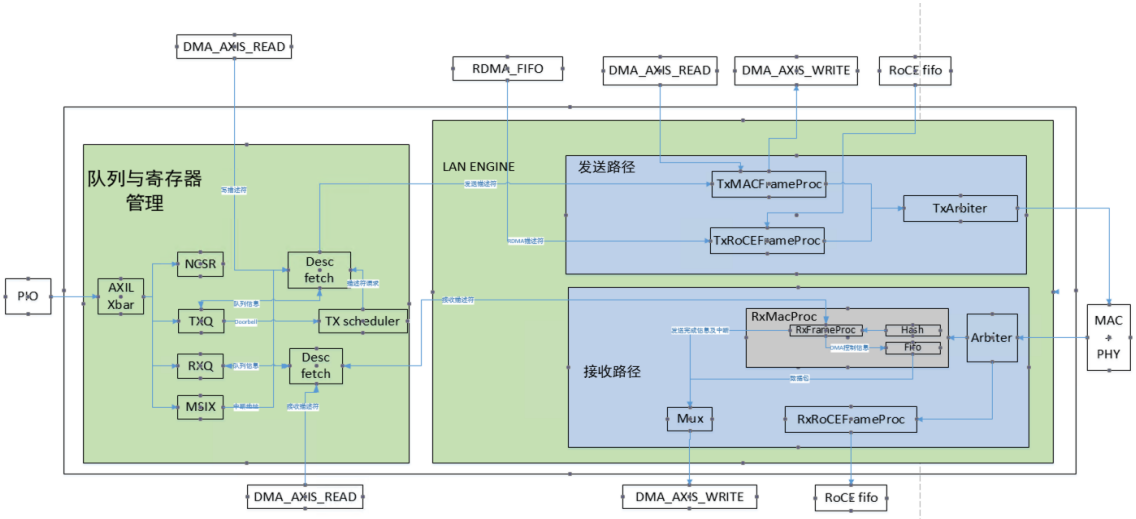


图 1-1 以太网引擎整体架构图

本网卡通过将队列信息与描述符数据进行拆分的方式，队列信息保存在网卡，描述符保

存在内存中，减少硬件资源占用。网卡中仅存储队列基址、队列指针、队列活动状态、队列大小等信息，每个队列仅占用 16B 空间。当要进行数据收发时，先根据队列的基址和指针，向内存中获取描述符，再将描述符发送给以太网引擎。队列工作流程如下：

接收方向：初始化时，驱动会根据队列大小开辟相应的缓冲区，队列内的描述符会指向相应的缓冲地址。然后驱动通过 IO 将队列头尾指针、大小、基址等信息写入网卡。当一个新的数据包到来时，以太网引擎会向队列管理模块索要数据包要发送到的内存地址，队列管理模块会根据其队列地址，和队列当前的指针，向内存中发起 DMA 请求，获取描述符信息（缓冲区地址和长度），然后将地址和长度信息发送给以太网引擎。

发送方向：初始化和接收方向类似。工作时，当一个新的数据包要发送时，驱动会更新相应队列指针，当队列管理模块检测到指针变动时，会获取指针指向的描述符信息（数据包地址和长度），发送给以太网引擎，以太网引擎根据数据包地址和长度，向内存中获取数据包，处理后发送。

网卡同时提供了 RSS（Received Side Scaling）服务，可以通过驱动配置队列匹配方式，将数据包引导入指定的队列中（默认采用 hash）。一方面，主机可以通过在内核中将队列与应用所在的核心绑定，降低数据包在不同 NUMA 域拷贝带来的性能损耗，并充分利用 DCA（Direct Cache Access），让应用可以直接从 L3 cache 中获取数据包，降低延迟，并且将不同流引导到不同的 CPU 核心上，充分调度 CPU 所有的计算资源。另一方面，可以对 RSS 进行扩展，将流导入不同的模块中（如在网计算模块），完成相应功能卸载。

2 模块外部接口定义

对接模块：PIO			
接口	输入/输出	位宽	说明
clk	输入	1	时钟信号
rst	输入	1	复位信号
awvalid_m	input	24	标准 AXI 接口
awaddr_m	input	3	标准 AXI 接口
awready_m	output	1	标准 AXI 接口
wvalid_m	input	1	标准 AXI 接口
wdata_m	input	32	标准 AXI 接口
wstrb_m	input	4	标准 AXI 接口
wready_m	output	1	标准 AXI 接口
bvalid_m	output	1	标准 AXI 接口
bready_m	input	2	标准 AXI 接口
arvalid_m	input	1	标准 AXI 接口
araddr_m	input	1	标准 AXI 接口
arready_m	output	24	标准 AXI 接口
rvalid_m	output	3	标准 AXI 接口
rdata_m	output	1	标准 AXI 接口
rready_m	output	1	标准 AXI 接口

对接模块：DMA			
接口	输入/输出	位宽	说明
rx_desc_dma_req_valid	output	1	DMA 接收描述符读请求接口
rx_desc_dma_req_last	output	1	
rx_desc_dma_req_data	output	256	
rx_desc_dma_req_head	output	128	
rx_desc_dma_req_ready	input	1	
rx_desc_dma_rsp_valid	input	1	DMA 接收描述符读响应接口
rx_desc_dma_rsp_last	input	1	
rx_desc_dma_rsp_data	input	256	
rx_desc_dma_rsp_head	input	128	

rx_desc_dma_rsp_ready	output	1	
tx_desc_dma_req_valid	output	1	DMA 发送描述符读请求接口
tx_desc_dma_req_last	output	1	
tx_desc_dma_req_data	output	256	
tx_desc_dma_req_head	output	128	
tx_desc_dma_req_ready	input	1	
tx_desc_dma_rsp_valid	input	1	DMA 发送描述符读响应接口
tx_desc_dma_rsp_data	input	1	
tx_desc_dma_rsp_head	input	256	
tx_desc_dma_rsp_last	input	128	
tx_desc_dma_rsp_ready	output	1	
rx_axis_wr_valid	output	1	RX 方向 DMA MAC 帧、完成信息、中断写请求接口
rx_axis_wr_last	output	1	
rx_axis_wr_data	output	256	
rx_axis_wr_head	output	128	
rx_axis_wr_ready	input	1	
tx_frame_req_valid	output	1	DMA 发送 MAC 帧读请求接口
tx_frame_req_last	output	1	
tx_frame_req_data	output	256	
tx_frame_req_head	output	128	
tx_frame_req_ready	input	1	
tx_frame_rsp_valid	input	1	DMA 发送 MAC 帧读响应接口
tx_frame_rsp_data	input	1	
tx_frame_rsp_head	input	256	
tx_frame_rsp_last	input	128	
tx_frame_rsp_ready	output	1	
tx_axis_wr_valid	output	1	TX 方向 DMA 完成事件写、中断写请求接口
tx_axis_wr_data	output	1	
tx_axis_wr_head	output	256	
tx_axis_wr_last	output	128	
tx_axis_wr_ready	input	1	

对接模块：RDMA			
接口	**输入/输出**	**位宽**	**说明**

i_tx_desc_empty	input	1	RDMA 发送描述符读取接口
iv_tx_desc_data	input	192	
o_tx_desc_rd_en	output	1	
i_roce_empty	input	1	RDMA 发送路径流量读取接口
iv_roce_data	input	256	
o_roce_rd_en	output	1	
i_roce_prog_full	input	1	RDMA 接收路径流量读取接口
ov_roce_data	input	256	
o_roce_wr_en	output	1	

对接模块：MAC			
接口	**输入/输出**	**位宽**	**说明**
axis_rx_valid	1	input	MAC RX 接口
axis_rx_last	1	input	
axis_rx_data	256	input	
axis_rx_data_be	32	input	
axis_rx_ready	1	output	
axis_rx_start	1	input	
axis_rx_user	7	input	
axis_tx_valid	1	output	MAC TX 接口
axis_tx_last	1	output	
axis_tx_data	256	output	
axis_tx_data_be	32	output	
axis_tx_ready	1	input	
axis_tx_start	1	output	
axis_tx_user	7	output	

3 寄存器管理单元（CSR Manager）

3.1 模块功能

寄存器管理单元如图 3-1 所示，主要可以划分为四个子模块：1.axi crossbar；2.TX 和 RX 队列管理；3.网卡硬件信息寄存器模块（NCSR）；4.中断向量管理模块（MSIX）。

外部与 PIO 和描述符调度模块对接。

在初始化时，驱动首先读写 NCSR 模块，获取网卡基本信息，如队列长度，队列 bar 空间基地址偏移等；紧接着对中断向量（MSIX manager 模块）进行配置，写入中断地址和中断信息；最后对 TX RX 队列（TXQ RXQ manager 模块）进行初始化。

整个数据流可以按照发送/接收划分为 2 部分：

1.发送处理路径（Transmit Path）：要进行数据包发送时，驱动修改发送队列头指针，经由 PIO，更新网卡中 TXQ 中相应队列的头指针。TXQ 管理模块检测到队列头指针更新后，生成 doorbell，传送给描述符调度模块。对描述符进行调度后，由描述符调度模块向 TXQ 发起队列信息请求，获取队列地址、队列指针、完成队列地址以及中断号等信息，TXQ 返回队列信息后，更新自己的队列指针，将队列尾指针加一，完成队列头指针加一。

2.接收处理路径（Receive Path）：数据包到来后，接收引擎会经由描述符调度模块向 RXQ 发起队列信息请求，RXQ 将相应的队列地址、当前指针、完成队列地址以及中断号信号返回给描述符调度模块，返回队列信息后，更新自己的队列指针，将队列尾指针加一，完成队列头指针加一。

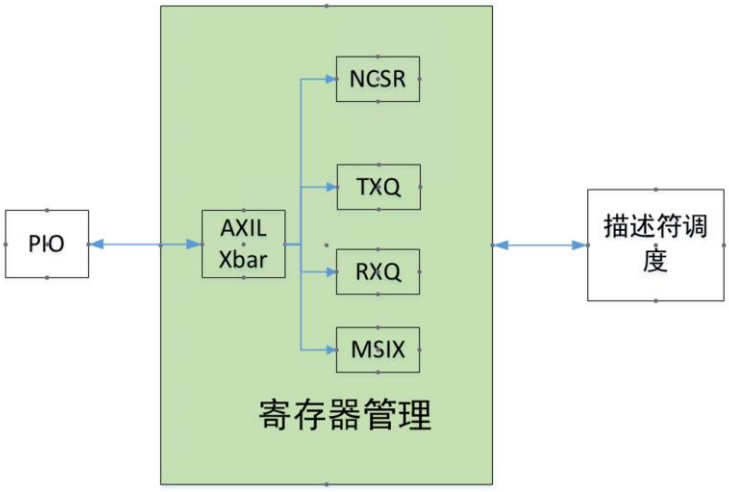


图 3-1 寄存器管理模块

3.2 模块接口

由子模块引出

3.3 子模块设计

3.3.1 AXIS 地址分配模块（AXIS Xbar）

3.3.1.1 模块功能

采用 synopsys axi IP，将 axi 总线按地址进行划分，将 PIO 信息根据地址传给不同 slave 模块。目前 slave 模块有 4 个，分别为

- 发送队列管理
- 接收队列管理
- ncsr 管理
- MSIX 中断

在使用时，axi 中有很多冗余的信号，如 burst 相关信号，暂时不用，只使用最基础的信号。

注：在使用时，读写中 last 信号一定要使用，否则会造成模块阻塞，因为驱动每次使用 iowrite32 或者 ioread32 发起寄存器读写，因此每次只改写一个寄存器，last 可以直接等于 valid 位， 如 assign wlast = wvalid。

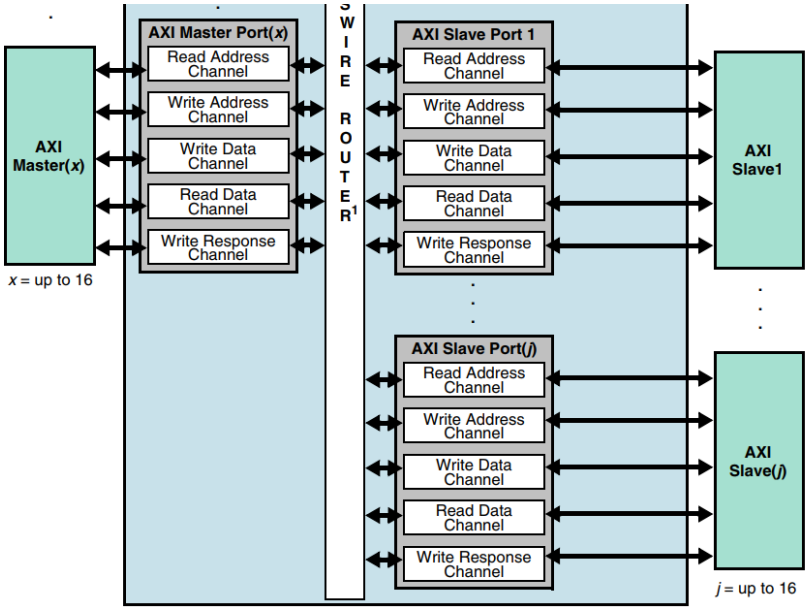


图 3-2 synopsys axi ip

3.3.1.2 模块接口

表 3-1 AXIS Xbar 模块接口

模块名	AXIS Xbar			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明

clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
awvalid_m	input	1	与 PIO 接口	标准 AXI 接口
awaddr_m	input	32		标准 AXI 接口
awready_m	output	1		标准 AXI 接口
wvalid_m	input	1		标准 AXI 接口
wdata_m	input	32		标准 AXI 接口
wstrb_m	input	4		标准 AXI 接口
wready_m	output	1		标准 AXI 接口
bvalid_m	output	1		标准 AXI 接口
bready_m	input	1		标准 AXI 接口
arvalid_m	input	1		标准 AXI 接口
araddr_m	input	32		标准 AXI 接口
arready_m	output	1		标准 AXI 接口
rvalid_m	output	1		标准 AXI 接口
rdata_m	output	32		标准 AXI 接口
rready_m	input	1		标准 AXI 接口
awvalid_csr	input	1	与内部 CSR 管理模块接口，目前设置有 4 个。表中只列出一组 slave 信号	标准 AXI 接口
awaddr_csr	input	32		标准 AXI 接口
awready_csr	output	1		标准 AXI 接口
wvalid_csr	input	1		标准 AXI 接口
wdata_csr	input	32		标准 AXI 接口
wstrb_csr	input	4		标准 AXI 接口
wready_csr	output	1		标准 AXI 接口
bvalid_csr	output	1		标准 AXI 接口
bready_csr	input	1		标准 AXI 接口
arvalid_csr	input	1		标准 AXI 接口
araddr_csr	input	32		标准 AXI 接口
arready_csr	output	1		标准 AXI 接口
rvalid_csr	output	1		标准 AXI 接口
rdata_csr	output	32		标准 AXI 接口
rready_csr	input	1		标准 AXI 接口

表 3-2 axl crossbar 接口

3.3.1.3 CSR 划分

寄存器地址 24bit	寄存器名称	[31：0]位 DATA 32bit	Component
nic_csr			
32'h11000	FW_ID_REG	[31:0]	interface id
32'h11004	IF_FEATURE_REG	[0:0]	rx rss enable
		[8:8]	tx checksum enable
		[9:9]	rx checksum enable
		[10:10]	rx hash enable
32'h10010	TX_QUEUE_INDEX_WIDTH_REG	[31:0]	发送队列数量
32'h10014	TX_QM_BASE_ADDR_REG	[31:0]	发送队列地址偏移
32'h10018	TX_CPL_QUEUE_INDEX_WIDTH_REG	[31:0]	发送完成队列数量
32'h1001C	TX_CQM_BASE_ADDR_REG	[31:0]	发送完成队列地址偏移
32'h10020	RX_QUEUE_INDEX_WIDTH_REG	[31:0]	接受队列数量
32'h10024	RX_QM_BASE_ADDR_REG	[31:0]	接收队列地址偏移
32'h10028	RX_CPL_QUEUE_INDEX_WIDTH_REG	[31:0]	接收完成队列数量
32'h1002C	RX_CQM_BASE_ADDR_REG	[31:0]	接收完成队列地址偏移
32'h10030	RSS mask	[31:0]	RSS mask
32'h10034	TX MTU	[31:0]	TX MTU
32'h10038	RX MTU	[31:0]	RX MTU
rx queue csr			
24'h11000	lower address	[31:0]	低位地址
24'h11004	upper address	[31:0]	高位地址
24'h11008	log size	[31:0]	块大小
24'h1100C	cpl queue index	[31:0]	完成事件号
24'h11010	head ptr	[31:0]	头指针
24'h11018	tail ptr	[31:0]	尾指针
rx cpl queue csr			
24'h11020	lower address	[31:0]	低位地址
24'h11024	upper address	[31:0]	高位地址
24'h11028	log size	[31:0]	块大小
24'h1102C	interrupt index	[31:0]	中断号
24'h11030	head ptr	[31:0]	头指针
24'h11038	tail ptr	[31:0]	尾指针
tx queue csr			
24'h12000	lower address	[31:0]	低位地址
24'h12004	upper address	[31:0]	高位地址
24'h12008	log size	[31:0]	块大小
24'h1200C	cpl queue index	[31:0]	完成事件号
24'h12010	head ptr	[31:0]	头指针
24'h12018	tail ptr	[31:0]	尾指针
tx cpl queue csr			
24'h11020	lower address	[31:0]	低位地址
24'h11024	upper address	[31:0]	高位地址

24'h11028	log size	[31:0]	块大小
24'h1102C	interrupt index	[31:0]	中断号
24'h11030	head ptr	[31:0]	头指针
24'h11038	tail ptr	[31:0]	尾指针
msix csr			
24'h20000	Msix vector	[31:0]	Msix vecoter
24'h30000	Msix pendding	[31:0]	Msix pendding

表 3-3 CSR 划分

图 3-3 CSR 划分

3.3.2 网卡硬件信息寄存器管理模块（NCSR）

3.3.2.1 模块功能

维护网卡内部状态寄存器信息，如发送队列数量，队列基地址及队列地址偏移等。详见表 3-3 CSR 划分。

3.3.2.2 模块接口

标准 AXI 总线接口。

模块名	NCSR			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号

rst	输入	1	/	复位信号
awvalid_csr	input	1	与 axi crossbar	标准 AXI 接口
awaddr_csr	input	32		标准 AXI 接口
awready_csr	output	1		标准 AXI 接口
wvalid_csr	input	1		标准 AXI 接口
wdata_csr	input	32		标准 AXI 接口
wstrb_csr	input	4		标准 AXI 接口
wready_csr	output	1		标准 AXI 接口
bvalid_csr	output	1		标准 AXI 接口
bready_csr	input	1		标准 AXI 接口
arvalid_csr	input	1		标准 AXI 接口
araddr_csr	input	32		标准 AXI 接口
arready_csr	output	1		标准 AXI 接口
rvalid_csr	output	1		标准 AXI 接口
rdata_csr	output	32		标准 AXI 接口
rready_csr	input	1		标准 AXI 接口

3.3.3 发送队列寄存器管理（TxQueueManager）

3.3.3.1 模块功能

模块负责维护发送队列信息，包括队列基址、头指针、尾指针、完成队列指针等。接收 PIO 下发的发送队列信息。等待描述符调度模块的队列请求，然后将待发送队列的队列信息交付描述符调度模块，待描述符获取成功，更新发送队列尾指针及完成队列指针。

模块内通过一系列的 ram 保存队列信息，在 PIO 或者描述符请求后对队列信息进行更改。

因为有的操作会对队列相同信息进行读写，因此在实现中，优先级顺序为 PIO 写> PIO 读>描述符请求。

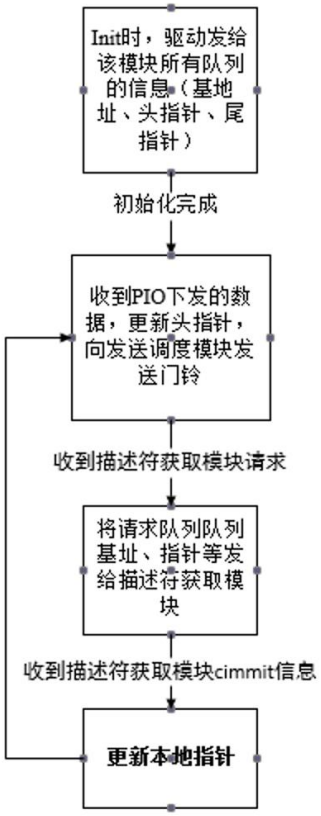


图 3-4 发送队列寄存器管理模块工作流程

3.3.3.2 模块接口

表 3-4 TxQueueManager 模块接口

模块名	TxQueueManager			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
awvalid_tx	input	1	与 PIO 接口 与 PIO 接口	标准 AXI 接口
awaddr_tx	input	32		标准 AXI 接口
awready_tx	output	1		标准 AXI 接口
wvalid_tx	input	1		标准 AXI 接口
wdata_tx	input	32		标准 AXI 接口
wstrb_tx	input	4		标准 AXI 接口
wready_tx	output	1		标准 AXI 接口
bvalid_tx	output	1		标准 AXI 接口
bready_tx	input	1		标准 AXI 接口

arvalid_tx	input	1		标准 AXI 接口
araddr_tx	input	32		标准 AXI 接口
arready_tx	output	1		标准 AXI 接口
rvalid_tx	output	1		标准 AXI 接口
rdata_tx	output	32		标准 AXI 接口
rready_tx	input	1		标准 AXI 接口
desc_dequeue_req_qnum	input	16	与描述符调度模块接口输入	请求队列号
desc_dequeue_req_valid	input	1		请求 valid
desc_dequeue_req_ready	output	1		返回有效
desc_dequeue_resp_qnum	output	16	与描述符调度模块接口输出	队列号
desc_dequeue_resp_qindex	output	16		队列指针
desc_dequeue_resp_desc_addr	output	64		描述符地址
desc_dequeue_resp_cpl_addr	output	64		Cpl 描述符地址
desc_dequeue_resp_msi	output	16		Msix 中断号
desc_dequeue_resp_status	output	8		描述符状态
desc_dequeue_resp_valid	output	1		有效位
doorbell_queue	output		描述符调度模块	门铃队列号
doorbell_valid	output			门铃有效为

3.3.4 接收寄存器管理（RxQueueManager）

3.3.4.1 模块功能

模块负责维护接收队列信息，包括队列基址、头指针、尾指针、完成队列指针等。接收 PIO 下发的发送队列信息。等待描述符调度模块的队列请求，然后将待发送队列的队列信息交付描述符调度模块，待描述符获取成功，更新发送队列尾指针及完成队列指针。

其他设计同发送寄存器管理模块。

3.3.4.2 模块接口

同发送寄存器管理模块

模块名	TxQueueManager			
模块说明	手工编码			
接口	**输入/输出**	**位宽**	**对接模块**	**说明**
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号

awvalid_tx	input	1	与 PIO 接口 与 PIO 接口	标准 AXI 接口
awaddr_tx	input	32		标准 AXI 接口
awready_tx	output	1		标准 AXI 接口
wvalid_tx	input	1		标准 AXI 接口
wdata_tx	input	32		标准 AXI 接口
wstrb_tx	input	4		标准 AXI 接口
wready_tx	output	1		标准 AXI 接口
bvalid_tx	output	1		标准 AXI 接口
bready_tx	input	1		标准 AXI 接口
arvalid_tx	input	1		标准 AXI 接口
araddr_tx	input	32		标准 AXI 接口
arready_tx	output	1		标准 AXI 接口
rvalid_tx	output	1		标准 AXI 接口
rdata_tx	output	32		标准 AXI 接口
rready_tx	input	1		标准 AXI 接口
desc_dequeue_req_qnum	input	16	与描述符调度模块接口输入	请求队列号
desc_dequeue_req_valid	input	1		请求 valid
desc_dequeue_req_ready	output	1		返回有效
desc_dequeue_resp_qnum	output	16	与描述符调度模块接口输出	队列号
desc_dequeue_resp_qindex	output	16		队列指针
desc_dequeue_resp_desc_addr	output	64		描述符地址
desc_dequeue_resp_cpl_addr	output	64		Cpl 描述符地址
desc_dequeue_resp_msi	output	16		Msix 中断号
desc_dequeue_resp_status	output	8		描述符状态
desc_dequeue_resp_valid	output	1		有效位

3.3.5 MSIX 寄存器管理（MSIX）

3.3.5.1 模块功能

接收 PIO 下发的中断向量及中断信息，以及 pendding 位设置（pendding 暂时全 0）并储存，在包发送或接收完成后，由发送接收引擎来查询中断向量的中断地址及信息，进行中断请求。中断向量配置见 3.3.5.3 节

3.3.5.2 模块接口

表 3-5 TxQueueManager 模块接口

模块名	TxQueueManager			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
awvalid_msix	input	1	与 PIO 接口 与 PIO 接口	标准 AXI 接口
awaddr_msix	input	32		标准 AXI 接口
awready_msix	output	1		标准 AXI 接口
wvalid_msix	input	1		标准 AXI 接口
wdata_msix	input	32		标准 AXI 接口
wstrb_msix	input	4		标准 AXI 接口
wready_msix	output	1		标准 AXI 接口
bvalid_msix	output	1		标准 AXI 接口
bready_msix	input	1		标准 AXI 接口
arvalid_msix	input	1		标准 AXI 接口
araddr_msix	input	32		标准 AXI 接口
arready_msix	output	1		标准 AXI 接口
rvalid_msix	output	1		标准 AXI 接口
rdata_msix	output	32		标准 AXI 接口
rready_msix	input	1		标准 AXI 接口
tx_irq_req_msix	input	16	与发送引擎接口 输入	请求中断号
tx_irq_req_valid	input	1		请求 valid
tx_irq_req_ready	output	1		返回有效
tx_irq_rsp_msg	output	32	与发送引擎输出	中断信息
tx_irq_rsp_addr	output	64		中断地址
tx_irq_rsp_valid	output	1		有效
tx_irq_rsp_ready	output	1		ready
rx_irq_req_msix	input	16	与接收引擎接口 输入	请求中断号
rx_irq_req_valid	input	1		请求 valid
rx_irq_req_ready	output	1		返回有效
rx_irq_rsp_msg	output	32	与接收引擎输出	中断信息
rx_irq_rsp_addr	output	64		中断地址
rx_irq_rsp_valid	output	1		有效

rx_irq_rsp_ready	output	1		ready

3.3.5.3 MSIX 中断配置

MSIX 支持 2048 个中断向量，并且可以要求中断向量不连续。MSI 中断最高支持 32 个中断请求，并且要求中断向量连续，因此对于多队列网卡，为每个队列都配置一个中断号的话，采用 MSIX。若不采用 MSIX，那么可能会面临中断向量不足的情况，需要硬件实现 event 事件等方式进行让多队列共享同一个中断，但是 event 事件会多一次 dma 写请求。因此我们采用 MSIX。

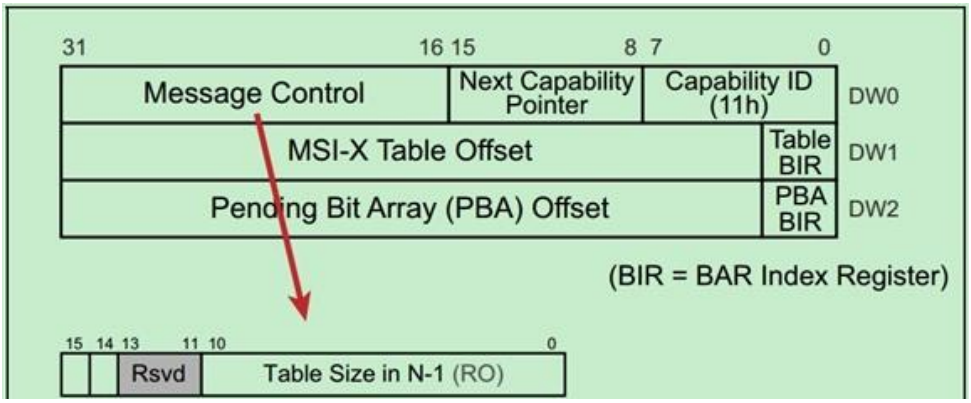


图 3-5 MSIX capability

在 PCIE 设备的 11 号 capability 中，存放 MSIX capability，其中包含了 MSIX table offset 和 pending bit array offset，里面存放了 MSIX vector 和 pending bit 在 bar 空间中的地址偏移。这两个字段可以在 Xillinx pcie core 中配置，如设置 MSIX 中断数量，在 bar 空间的地址偏移。设置好这些属性后，就可以系统中通过 lspci 扫描到相应的设置了。

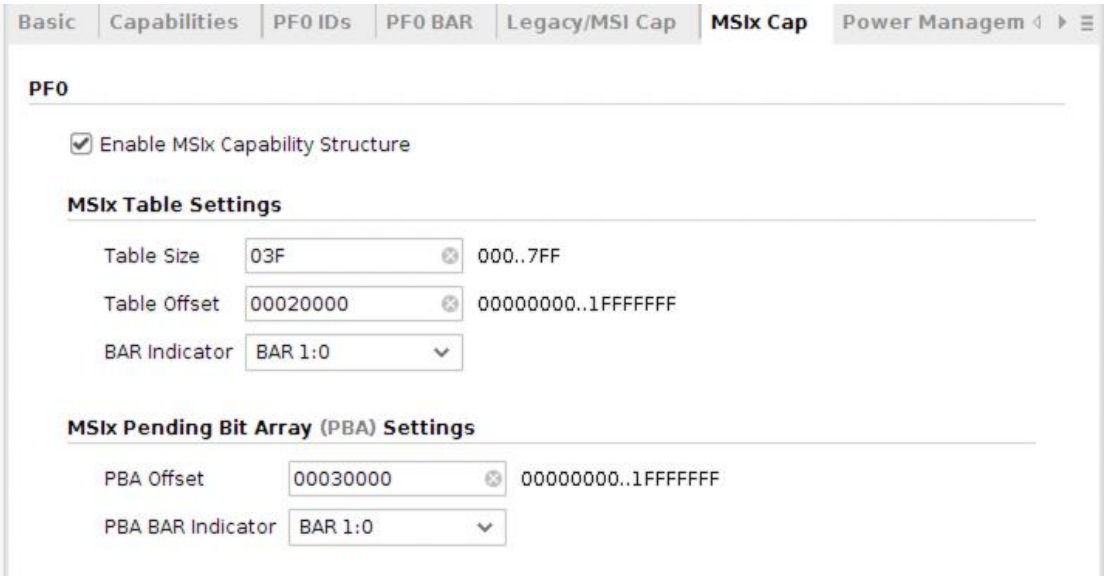


图 3-6 MSIX capability 配置

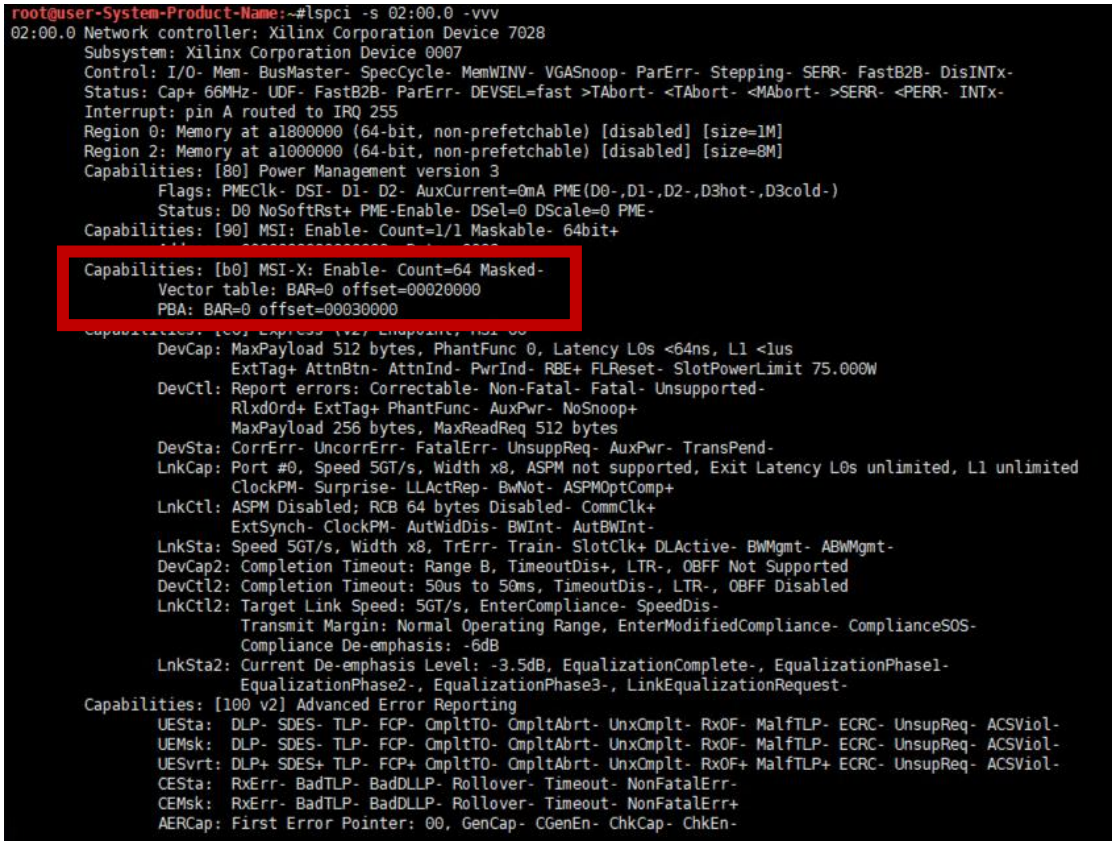

root@user-System-Product-Name:~#lspci -s 02:00.0 -vvv
02:00.0 Network controller: Xilinx Corporation Device 7028
Subsystem: Xilinx Corporation Device 0007
Control: I/O- Mem- BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B- DisINTx-
Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
Interrupt: pin A routed to IRQ 255
Region 0: Memory at a1800000 (64-bit, non-prefetchable) [disabled] [size=1M]
Region 2: Memory at a1000000 (64-bit, non-prefetchable) [disabled] [size=8M]
Capabilities: [80] Power Management version 3
Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
Status: D0 NoSoftRst+ PME-Enable- DSel=0 DScale=0 PME-
Capabilities: [90] MSI: Enable- Count=1/1 Maskable- 64bit+
Capabilities: [b0] MSI-X: Enable- Count=64 Masked-
Vector table: BAR=0 offset=00020000
PBA: BAR=0 offset=00030000
DevCap: MaxPayload 512 bytes, PhantFunc 0, Latency L0s <64ns, L1 <1us
ExtTag+ AttnBtn- AttnInd- PwrInd- RBE+ FLReset- SlotPowerLimit 75.000W
DevCtl: Report errors: Correctable- Non-Fatal- Fatal- Unsupported-
RlxdOrd+ ExtTag+ PhantFunc- AuxPwr- NoSnoop+
MaxPayload 256 bytes, MaxReadReq 512 bytes
DevSta: CorrErr- UncorrErr- FatalErr- UnsuppReq- AuxPwr- TransPend-
LnkCap: Port #0, Speed 5GT/s, Width x8, ASPM not supported, Exit Latency L0s unlimited, L1 unlimited
ClockPM- Surprise- LLActRep- BwNot- ASPMOptComp+
LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- CommClk+
ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
LnkSta: Speed 5GT/s, Width x8, TrErr- Train- SlotClk+ DLActive- BWMgmt- ABWMgmt-
DevCap2: Completion Timeout: Range B, TimeoutDis+, LTR-, OBFF Not Supported
DevCtl2: Completion Timeout: 50us to 50ms, TimeoutDis-, LTR-, OBFF Disabled
LnkCtl2: Target Link Speed: 5GT/s, EnterCompliance- SpeedDis-
Transmit Margin: Normal Operating Range, EnterModifiedCompliance- ComplianceSOS-
Compliance De-emphasis: -6dB
LnkSta2: Current De-emphasis Level: -3.5dB, EqualizationComplete-, EqualizationPhase1-
EqualizationPhase2-, EqualizationPhase3-, LinkEqualizationRequest-
Capabilities: [100 v2] Advanced Error Reporting
UESta: DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF- MalfTLP- ECRC- UnsupReq- ACSViol-
UEMsk: DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF- MalfTLP- ECRC- UnsupReq- ACSViol-
UESvrt: DLP+ SDES+ TLP- FCP+ CmpltTO- CmpltAbrt- UnxCmplt- RxOF+ MalfTLP+ ECRC- UnsupReq- ACSViol-
CESta: RxErr- BadTLP- BadDLLP- Rollover- Timeout- NonFatalErr-
CEMsk: RxErr- BadTLP- BadDLLP- Rollover- Timeout- NonFatalErr+
AERCap: First Error Pointer: 00, GenCap- CGenEn- ChkCap- ChkEn-

图 3-7 MSIX 中断

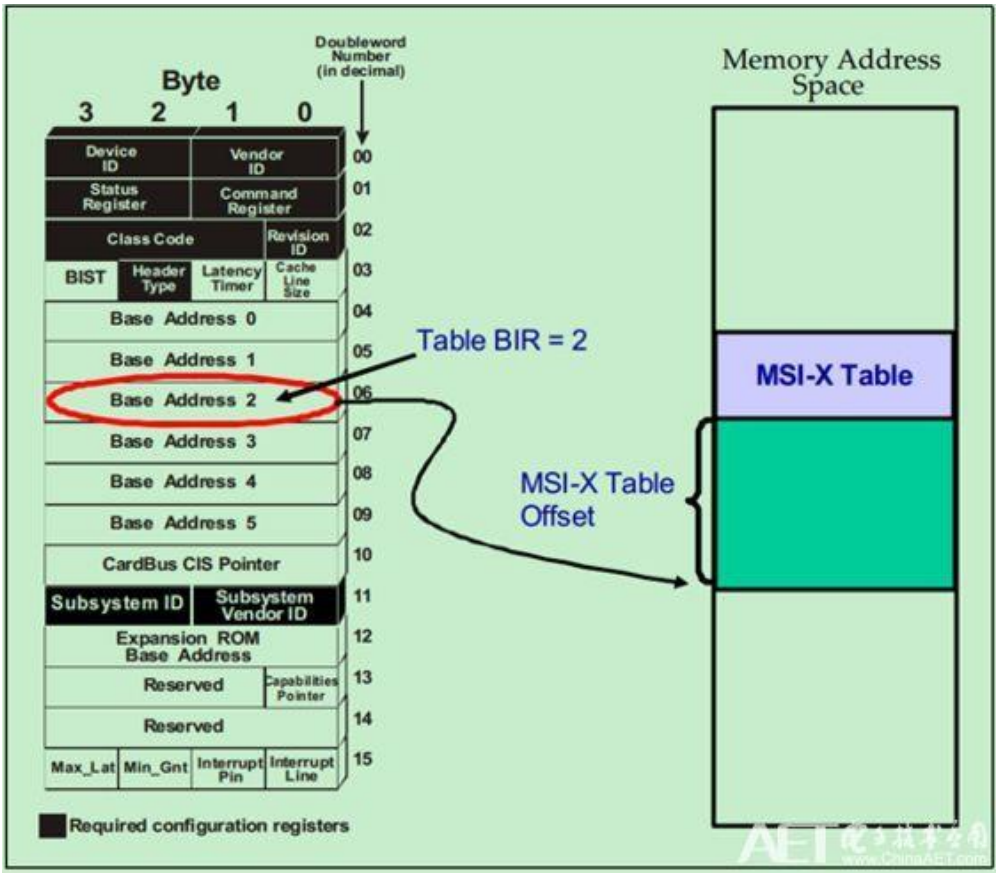


图 3-8 msix table 与 bar 空间对应图

DW3	DW2	DW1	DW0	
Vector Control	Message Data	Upper Address	Lower Address	Entry 0
Vector Control	Message Data	Upper Address	Lower Address	Entry 1
Vector Control	Message Data	Upper Address	Lower Address	Entry 2
....	....	....	....	
....	....	....	....	
Vector Control	Message Data	Upper Address	Lower Address	Entry N-1

图 3-9 中断向量格式

MSIX 采用 tlp msg 方式发起中断，在使用时，首先进行初始化，调用 pci_enable_msix_range 函数使能 MSIX 中断，返回得到可以使用的中断向量数目，再调用 pci_request_irq 函数，为每个中断进行注册，该操作为 bar 空间中的 MSIX table 中每一个 MSIX vector（见图 3-8 与图 3-9）写入相应的中断地址和中断信息，并绑定中断函数。经过配置后，硬件就可以发起 MSIX 中断了。

在硬件实现时，软件会根据 PCIe capability 中的 misx table 偏移，逐次向每一个 table 项

写入信息，因此 MSIX manager 模块要根据下发的 PIO 读写请求，根据地址区分要读写的 table 表项（table 表项在硬件内部为一个 ram），将配置信息存入内部的 table 中。

4 描述符调度模块（Descriptor Schedule）

4.1 模块功能

描述符调度整体架构如图 4-1 所示，主要可以划分为两个部分：1.发送调度模块（Tx scheduler）；2.描述符获取模块（Desc fetch）。

整个数据流可以按照发送/接收划分为 2 部分：

1.发送处理路径（Transmit Path）：当有数据包发送时，会通过 PIO 修改寄存器管理模块 TXQ 中相应队列的队头指针，然后会发送一个 doorbell 给发送调度模块，发送调度模块对发送请求进行调度后，向描述符调度模块发起发送请求，描述符调度模块根据队列号，向状态管理单元获取队列信息（描述符地址、指针、完成队列指针），拿到队列信息后，根据计算的描述符地址，发起 DMA 请求，获取描述符信息，将描述符发给 LAN Engine。

2.接收处理路径（Receive Path）：当一个数据包到来后，LAN Engine 发起描述符获取请求，描述符调度模块根据请求的队列号，向状态管理单元 RXQ 中获取队列信息（描述符地址、指针、完成队列指针），拿到描述符地址后发起 DMA 请求，获取描述符，将描述符返回 LAN Engine。

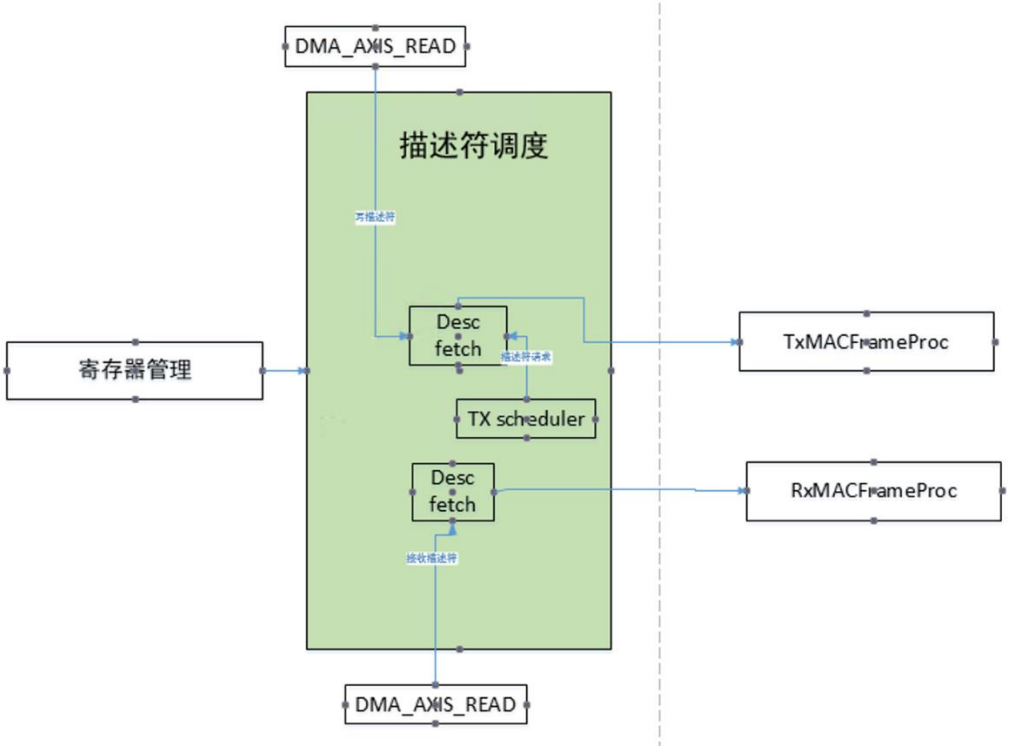


图 4-1 描述符调度设计

4.2 模块接口

由子模块引出

4.3 子模块设计

4.3.1 发送调度模块（Tx scheduler）

4.3.1.1 模块功能

接收驱动下发的 doorbell，对各个待发送队列按照调度算法进行调度。将被选中的队列号发给描述符调度模块。

目前采用的时 RoundRobin 轮训方式。

模块内部有一个请求队列，当有门铃信息时，将门铃信息插入队列，若 desc fetch 模块空闲，便给其发送队列请求。

4.3.1.1 模块接口

表 4-1 TxQueueManager 模块接口

模块名	TxQueueManager			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
doorbell_queue	input	16	与 TXQ 接口	门铃队列号
doorbell_valid	input	1		门铃有效
desc_req_qnum	output	16	与 desc fetch 接口	描述符请求队列号
desc_req_valid	output	1		有效位
desc_req_ready	input	32		Ready 位

4.3.2 描述符获取模块（Descriptor Fetch Module）

4.3.2.1 模块功能

接收 LEN Engine 或者 Tx scheduler 发来的描述符请求，根据其队列号，向 TxQ 或者 RxQ 模块请求队列信息，拿到队列信息（描述符地址，指针，CPL 地址）后，发起 DMA 请求，获取描述符数据，最后将描述符数据发回 LEN Engine。该模块例化两次，在接收方向和发送方向各例化一次。

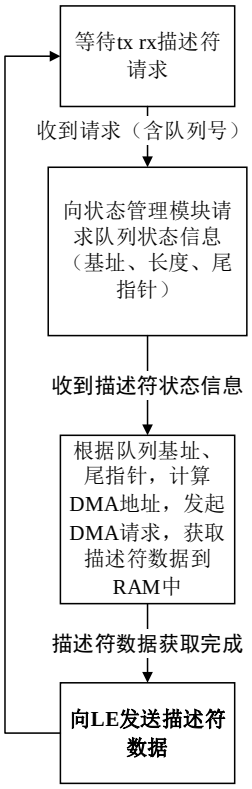


图 4-2 描述符获取模块工作流程

4.3.2.1.1 异步读取

需要注意的是，提交 DMA 读请求和等待 DMA 数据返回是异步的过程。如下图所示，只要有新的描述符到达，就可以向 DMA 引擎提交数据读请求，而不必等之前的 DMA 读请求完成。当读响应到达后，将描述符交给 LEN Engine。

在代码实现时，采用流水线的方式，通过维持一个状态管理 table 对各个请求进行管理，

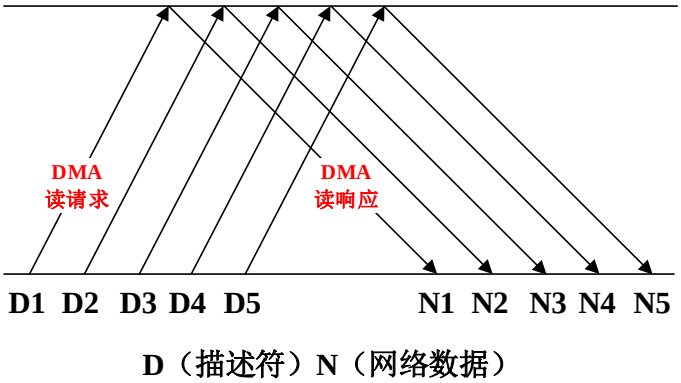
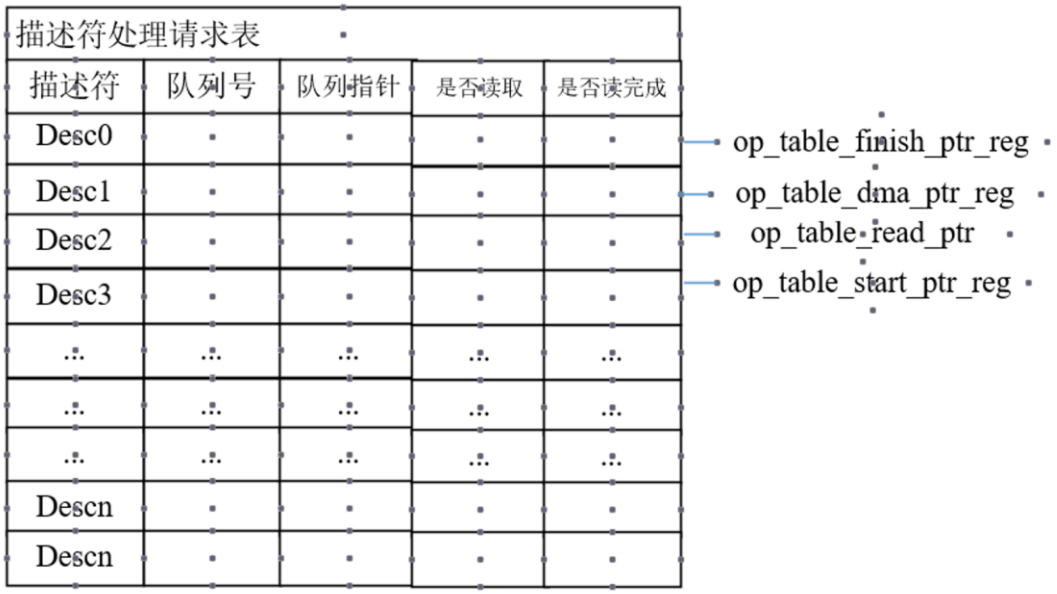


图 4-3 异步读取



每当请求到来时，存下请求的队列号，并将 start 指针自增，当 read 指针小于 start 指针后，便将 read 指针所指的队列号，向队列管理单元请求队列信息，返回队列信息后，read 指针自增；当 dma 指针检测到 read 指针大于它后，便将 dma 指针所指 entry 项，获取描述符地址，然后发起 dma 请求获取描述符；最后当 finish 指针发现 dma 指针小于它后，将 finish 指针所指 entry 返回 LEN engine；

4.3.2.2 接口设计

表 4-2 TimerControl 模块接口

模块名	TimerControl			
模块说明	手工编码			
接口	**输入/输出**	**位宽**	**对接模块**	**说明**
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
desc_req_qnum	input	16	描述符请求仲裁模块	队列号
desc_req_valid	input	1		Valid 位
desc_req_ready	output	1		Ready 位
desc_dequeue_req_qnum	output	16	队列寄存器管理模块输出	请求队列号
desc_dequeue_req_valid	output	1		Valid 位
desc_dequeue_req_ready	output	1		Ready 位
desc_dequeue_resp_qnum	input	16	队列寄存器管理模块输入	队列号
desc_dequeue_resp_qindex	input	16		队列指针
desc_dequeue_resp_desc_addr	input	64		描述符地址

desc_dequeue_resp_cpl_addr	input	64		CPL 地址
desc_dequeue_resp_msi	input	16		中断号
desc_dequeue_resp_status	input	8		描述符状态
desc_dequeue_resp_valid	input	1		Valid 位
desc_rsp_status	output	8	LEN Engine	描述符状态
desc_rsp_qnum	output	16		队列号
desc_rsp_qindex	output	16		队列指针
desc_rsp_length	output	32		描述符长度
desc_rsp_dma_addr	output	64		包地址
desc_rsp_cpl_addr	output	64		Cpl 地址
desc_rsp_msi	output	16		中断号
desc_rsp_valid	output	1		Valid 位
desc_rsp_ready	input	1		Ready 位
desc_dma_req_valid	output	1	Dma 读请求	Dma 接口
desc_dma_req_last	output	1		Dma 接口
desc_dma_req_data	output	256		Dma 接口
desc_dma_req_head	output	128		Dma 接口
desc_dma_req_ready	input	1		Dma 接口
desc_dma_rsp_valid	input	1	Dma 读响应	Dma 接口
desc_dma_rsp_last	input	1		Dma 接口
desc_dma_rsp_data	input	256		Dma 接口
desc_dma_rsp_head	input	128		Dma 接口
desc_dma_rsp_ready	output	1		Dma 接口

5 以太网引擎（LAN Engine）

5.1 模块功能

负责 mac 帧接收和发送。

分为接收和发送两个方向。

发送方向上，1、对于以太网包，获取到描述符后，根据描述符内包地址和长度，发起 dma 请求，将包从内存拉取到 fifo 中，然后发送给 mac。2、对于 roce 包，首先向 roce 描述符 fifo 中获取 roce 描述符，然后从 roce 数据包 fifo 中拉取 roce 包负载（fifo 中只有 payload 部分，需要和描述符中的 mac 地址、ip 地址等拼成完整的以太网包），和描述符中的信息，一起拼成完整的以太网包，然后加入校验和，发送给 mac。

接收方向上，首先根据包的类型和端口进行判断，如类型为 udp 包，端口为 4791，则判断为 roce 包，否则为以太网包。1、对于以太网包，数据包到来后，向描述符调取模块获取描述符，根据描述符内包地址和长度，发起 dma 写请求，将包从内部 fifo 中，发给主机内存。2、对于 roce 包，当包到来时，对其 IP 和 UDP 进行校验，校验成功后，将负载部分发给 roce 接收 fifo。

5.2 模块架构

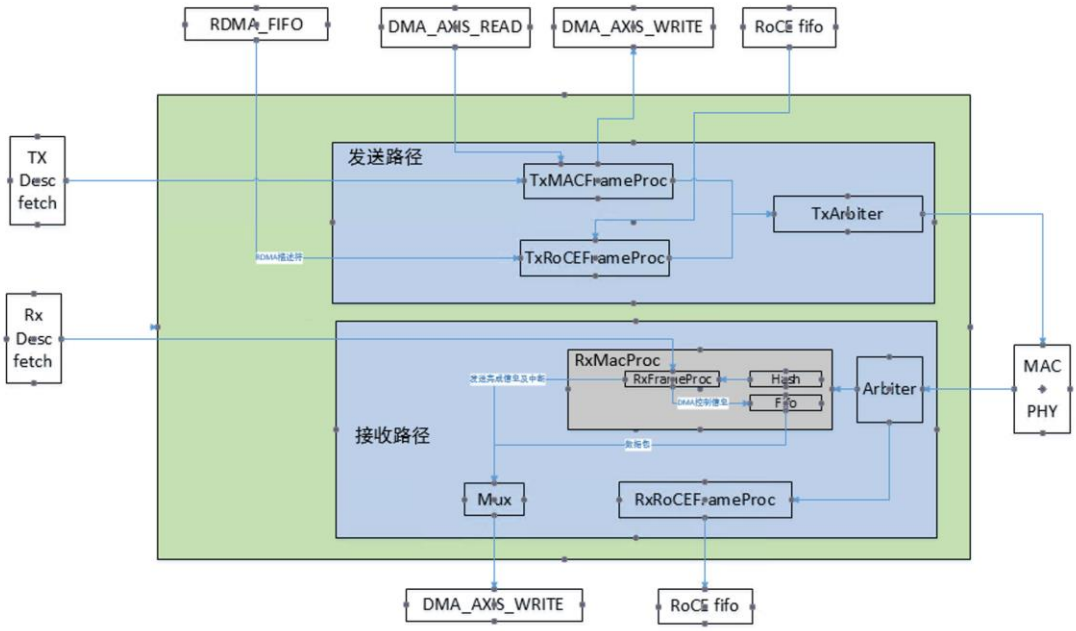


图 5-1 以太网引擎架构

5.3 模块接口

模块接口由子模块引出

5.4 子模块设计

5.4.1 以太网包发送引擎

5.4.1.1 模块功能

获取描述符调度模块发来的描述符帧，根据描述内容，向内存发起 dma 请求，将数据包拉取到 fifo 中，然后将包发送往 mac。

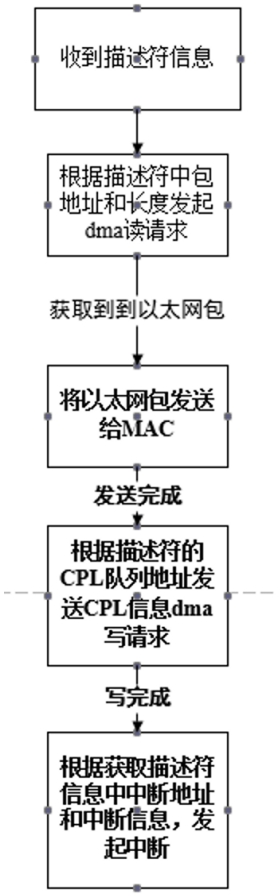


图 5-2 以太网发送引擎工作流程

在发送时，采用流水线方式，分为获取描述符，拉取数据包、发送 cpl、中断信息四级。在驱动发送数据包时，对同一个包所有的分片都拉取到了连续地址空间里，所以一个包对应一个描述符。每发送一个包，队列头指针自增 1，在获取描述符信息时，只进行一次 dma 请求。

5.4.1.2 模块接口

模块名	RX Engine
模块说明	手工编码

接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
tx_desc_qnum	input	16	描述符调度模块	队列号
tx_desc_qindex	input	16		队列指针
tx_desc_frame_len	input	32		数据包长度
tx_desc_dma_addr	input	64		数据包 dma 地址
tx_desc_cpl_addr	input	64		完成信息地址
tx_desc_msix	input	16		中断号
tx_desc_valid	input	1		Valid
tx_desc_ready	Output	1		Ready
tx_frame_req_valid	Output	1	Dma 数据包读请求	数据包读有效
tx_frame_req_last	Output	1		Last
tx_frame_req_data	Output	256		Data（为空）
tx_frame_req_head	Output	128		Head
tx_frame_req_ready	input	1		Ready
tx_frame_rsp_valid	input	1	Dma 数据包读响应	Valid
tx_frame_rsp_last	input	1		Last
tx_frame_rsp_data	input	256		Data
tx_frame_rsp_head	input	128		Head
tx_frame_rsp_ready	Output	1		Ready
axis_tx_valid	Output	1	与 mac 接口	Valid
axis_tx_last	Output	1		Last
axis_tx_data	Output	256		数据位
axis_tx_data_be	Output	32		字节有效
axis_tx_ready	input	1		Ready
axis_tx_user	Output	7		长度字段
axis_tx_start	Output	1		Start
tx_axis_wr_valid	input	1	Dma 完成信息、中断写接口	Valid
tx_axis_wr_data	input	256		Last
tx_axis_wr_head	input	128		Data
tx_axis_wr_last	input	1		Head
tx_axis_wr_ready	Output	1		Ready

5.4.1.3 以太网描述符

描述符格式定义如下：

表 5-1 以太网描述符

+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
Length																Reserved															
Buffer addr[63:0]																															
Buffer addr[127:64]																															

字段说明如下表所示：

表 5-2 以太网描述符字段说明

字段	说明
Length	内存 Buffer 长度
Buffer addr	Buffer 地址

5.4.2 以太网包接收引擎

5.4.2.1 模块功能

当数据包到来时，根据数据包 ip 地址及端口号的四元组进行 hash，得到要放入的队列号。用队列号向描述符调度模块发起请求，得到描述符数据后，将包 dma 进主机内存，然后发起完成信息写以及中断请求。

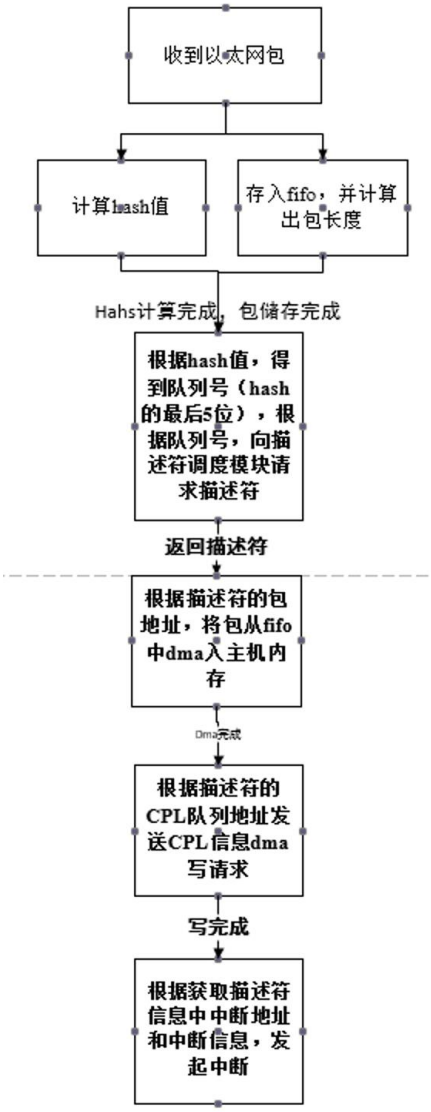


图 5-3 以太网接收模块工作流程

5.4.2.2 模块接口

模块名	RX Engine			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
rx_desc_req_qnum	Output	16	描述符请求	队列号
rx_desc_req_valid	Output	1		Valid

rx_desc_req_ready	input	1		Ready
rx_desc_rsp_status	input	8	描述符调度模块	描述符状态
rx_desc_rsp_qnum	input	16		队列号
rx_desc_rsp_qindex	input	16		队列指针
rx_desc_rsp_length	input	32		数据包长度
rx_desc_rsp_dma_addr	input	64		数据包 dma 地址
rx_desc_rsp_cpl_addr	input	64		完成信息地址
rx_desc_rsp_msix	input	16		中断号
rx_desc_rsp_valid	input	1		Valid
rx_desc_rsp_ready	Output	1		Ready
axis_rx_valid	input	1	与 mac 接口	Valid
axis_rx_last	input	1		Last
axis_rx_data	input	256		数据位
axis_rx_data_be	input	32		字节有效
axis_rx_ready	input	1		Ready
axis_rx_user	input	7		长度字段
axis_rx_start	input	1		Start
rx_axis_wr_valid	input	1	Dma 数据包、完成信息、中断写接口	Valid
rx_axis_wr_data	input	256		Last
rx_axis_wr_head	input	128		Data
rx_axis_wr_last	input	1		Head
rx_axis_wr_ready	Output	1		Ready

5.4.2.3 hash 子模块设计

hash 模块负责计算包的 hash 值，当包到来时，首先判断包是否是 ip 包，若是，提取 sip 和 dip，采用 toeplitz 算法计算 hash 值。然后再判断包是否是 tcp 包或者 udp 包，若是，提取 sport 和 dport，再计算 hash，最后与 ip 的 hash 进行异或。

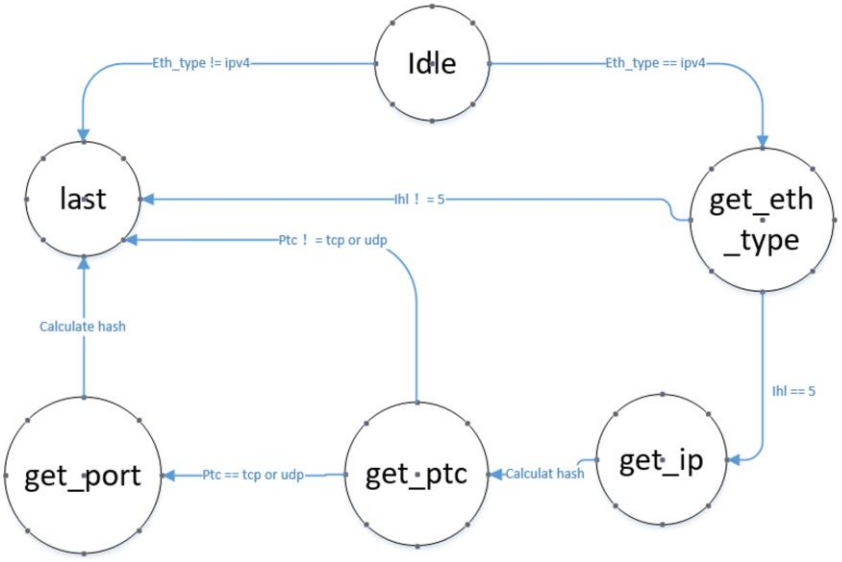


图 5-4 hash 状态机

模块名	RX Engine			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
axis_rx_valid	input	1	输入数据包	Valid
axis_rx_last	input	1		Last
axis_rx_data	input	256		Data
axis_rx_data_be	input	32		字节有效
Hash_key	Input	320	Hash_key	随机的 hash key
crx_hash	Output	32	输出数据	Hash 值
crx_hash_type	Output	4		Hash type
crx_hash_valid	Output	1		Valid

表 5-3 hash 模块接口

5.4.3 roce 包发送引擎

5.4.3.1 模块功能

从 roce 描述符队列中获取描述符，与 roce 数据包 fifo 中的 roce 数据包负载合并为一个完整的以太网包，并加入校验信息后，发送给 mac。

模块设计如图 5-5 所示，当 TXRoCEDescProc 模块检测到描述符 FIFO 不为空时，便从 FIFO 中获取一个描述符，描述符定义见 5.4.3.3，然后交给 RoCE 发送处理模块 TxRoCEFrameProc 模块，由发送处理模块从 RoCE 数据包 FIFO 中将数据包负载（RoCE 数据包 FIFO 内只有负载部分，RoCE 采用 udp 协议，因此 FIFO 中只含有 udp 负载，需要根据描述符中的信息拼成完整的以太网包）拉至本模块，然后根据描述符内的源目的 MAC、IP，和数据包负载等信息拼成一个完整的以太网包，并进行校验，将校验和添加至相应字段，然后发往 MAC。

注：以太网采用的是大端序，拼接时也要按照大端序进行拼接。

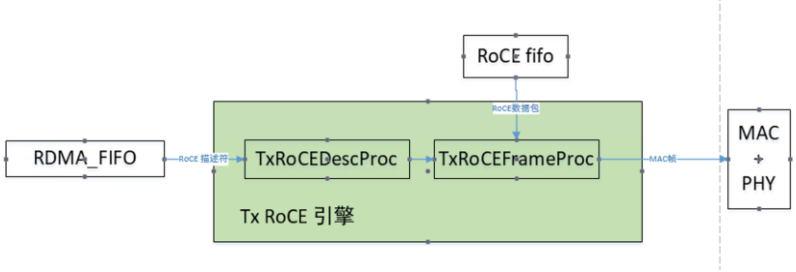


图 5-5 RoCE 发送引擎

5.4.3.2 模块接口

模块名	RX Engine			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
i_tx_desc_empty	input	1	Roce 描述符队列	队列空
iv_tx_desc_data	input	192		描述符数据
o_tx_desc_rd_en	Output	1		队列读
i_roce_empty	input	1	Dma 数据包读请求	队列空
iv_roce_data	input	256		数据包负载数据
o_roce_rd_en	Output	1		队列读
axis_tx_valid	Output	1	与 mac 接口	Valid
axis_tx_last	Output	1		Last
axis_tx_data	Output	256		数据位
axis_tx_data_be	Output	32		字节有效

axis_tx_ready	input	1		Ready
axis_tx_user	Output	7		长度字段
axis_tx_start	Output	1		Start

5.4.3.3 处理 RoCE 发送数据描述符

描述符格式定义如下：

表 5-4 RoCE 发送描述符

+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
Length																Reserved												DTYP			
Source MAC																															
Destination MAC																Source MAC															
Destination MAC																															
Source IP																															
Destination IP																															

字段说明如下表所示：

表 5-5 RoCE 发送描述符字段说明

字段	说明
DTYP	描述符类型（RoCE 流量为 4’b1111）
Length	RoCE 报文长度
Source/Destination MAC	源/目的 MAC
Source/Destination IP	源/目的 IP

5.4.3.4 发送数据帧处理（TxRoCEFrameProc）

5.4.3.4.1 模块功能

RoCE 流量处理需要为数据负载添加以太网头部、IP 头部和 UDP 头部。每个 RDDESC 指示一个 UDP 报文，且该报文的程度不会超过 MTU 限制。因此，本模块不需要再对其进行分段处理。

5.4.3.4.2 L2 头部

源/目的 MAC：由 RDDESC 中获取；

类型字段：0x0800，表明是上层协议是 IP。

5.4.3.4.3 IP 头部

在 UDP 处理中，MSS = MTU - 8B（UDP 头部）- 20B（IP 头部）。RDMA 引擎在处理数据时，每次向 LE 提交的发送请求的数据长度不会超过 MSS，即数据分段已经由 RDMA 引擎完成，IP 层只需要为当前数据添加一个网络层包头即可。

IP 头部中的各个字段生成规则如下：

版本号：4；

首部长度：IP 报文头部的长度。固定部分的长度（20 字节）和可变部分的长度之和。对于 RoCE 流量，可变部分的长度为 0；

服务类型：默认为 0；

总长度：IP 报文头部加上 UDP 头部长度数据负载部分的长度；

标识：每次发送一个 UDP 报文，该值加 1。同一个 UDP 报文中的不同分片中的标识相同。该标识可以由 LE 保存在一个寄存器中，初始值为 0；

标志：标识当前分片是否为上层报文中的最后一个分片。对于 RoCE 报文，始终为 0；

分片偏移：标识当前分片在上层报文中的偏移。对于 RoCE 报文，始终为 0；

生存时间：初始值设定为 64；

协议：指定上层协议。UDP 协议对应的协议号为 17；

首部检查和：计算 IP 头部的检查和。

源/目的 IP：由 RoCE 描述符指定。

5.4.3.4.4 UDP 头部

源/目的端口均填入 4791，UDP 报文长度从 RDDESC 中获得。

在完成上述处理后，将描述符丢弃，生成的数据帧传递给 MAC 层使用。

5.4.3.4.5 检查和计算

IPv4 首部格式如下图所示：

表 5-6 IPv4 头部

+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
IP Header																															
Total Length																Type of Service								Header Length				Version			
Fragmentation Offset														M F	D F	0	Identification														
Header Checksum																	Protocol								Time to Live						
Source IP																															
Destination IP																															

Options (0 ~ 40Bytes)
Data

计算过程如下：

1) 将 IP 首部检查和字段全部填充为 0；

2) 对每 16bit（2 字节）进行二进制反码求和；如果求和结果有进位，则将进位加到最低位上去；

3) 将求得的结果填入检查和字段。

包头其它字段保持不变。

- L4 检查和计算

表 5-7 UDP 头部

+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
UDP Header																															
Destination Port																Source Port															
Payload Length																Checksum															

TCP 和 UDP 的检查和计算过程相同：

1) 生成伪首部，伪首部格式如下：

表 5-8 TCP/UDP 伪首部

+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
TCP Pseudo Header																															
Source IP																															
Destination IP																															
tcpl																ptcl								mbz							

其中源、目的 IP 由 IP 首部取得，mbz 全部置为 0，ptcl 为 17，tcpl 置为 TCP 包头长度加上数据负载长度；

2) 将校验和置为 0；

3) 将伪首部部分，TCP 包头部分，以及数据部分划分为 16 位一组的数据，将这些数据逐个进行累加，进位的部分加到最低位上；

4) 最后将得到的结果取反，填入 TCP 首部的校验和部分。

包头其它字段保持不变。

在完成上述处理后，将数据帧传递给 MAC 层，将该数据帧对应的所有描述符传递给 TxCQandIRQ 模块。

5.4.3.5 checksum 子模块设计

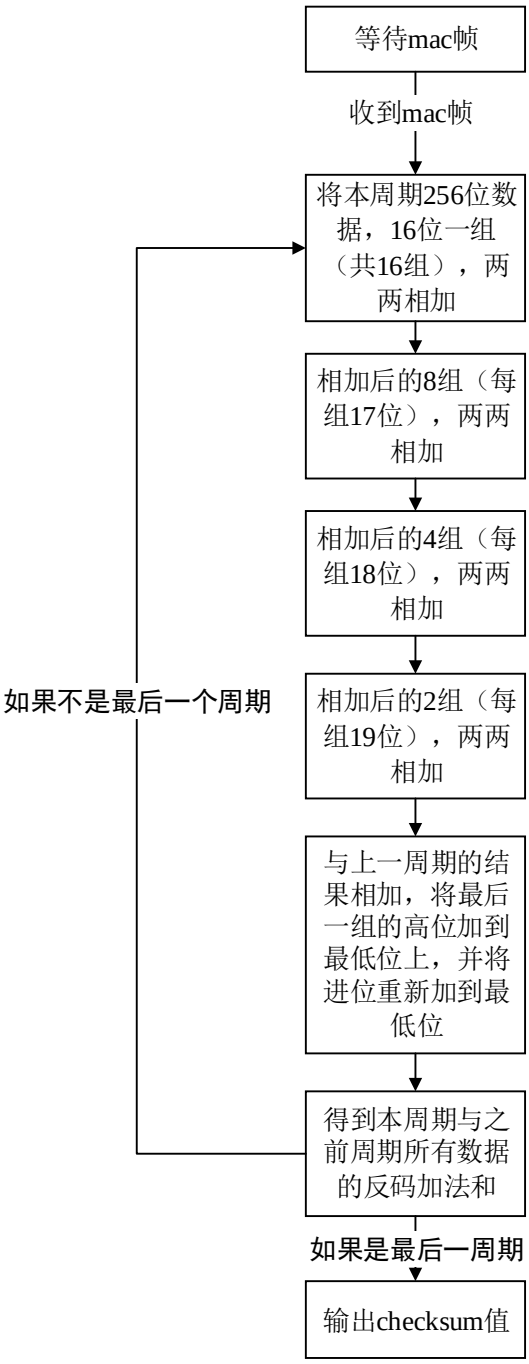


图 5-6 checksum 模块设计

checksum 进行校验时，不区分 udp 或者 ip，只通过传递给模块的 data 和 data_be 信号完成校验，因此外部模块需要调整相关信号。

如在 ip 校验，传入完整数据包时，要通过修改 data_be 掩码，将除了 ip 包头的其他的数据 data_be 位赋值为 0，即可得到 ip 包头的校验。

信号在校验 udp 时，需在传输 data 信号时，额外传输一拍 udp 伪首部（或者直接将 mac 头的部分改为伪首部），输入模块中。

checksum 模块通过 REVERSE 参数修改大端序或者小端序，默认为大端序。。

模块名	RX Engine			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
csum_data_valid	input	1	输入数据包	Valid
csum_data_last	input	1		Last
csum_data	input	256		Data
csum_data_be	input	32		字节有效
csum_out	Output	16	输出数据	Hash 值
csum_out_valid	Output	1		Hash type

表 5-9 hash 模块接口

5.4.4 Roce 包接收引擎（RxRoCEFrameProc）

5.4.4.1 模块功能

获取 MAC 发来数据包，对数据包进行校验，校验通过后对负载部分发送给 roce 接口 fifo 中。

校验过程与 RoCE 发送引擎相同，在处理时连同 checksum 字段一起校验，若计算的校验和结果全为 1，即 16'FFFF，则证明数据无误。

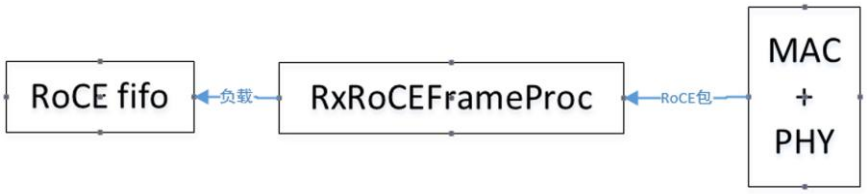


图 5-7 RoCE 包接收引擎

5.4.4.2 模块接口

模块名	RX Engine			
模块说明	手工编码			
接口	输入/输出	位宽	对接模块	说明
clk	输入	1	/	时钟信号

rst	输入	1	/	复位信号
axis_rx_valid	input	1	与 mac 接口	Valid
axis_rx_last	input	1		Last
axis_rx_data	input	256		数据位
axis_rx_data_be	input	32		字节有效
axis_rx_ready	input	1		Ready
axis_rx_user	input	7		长度字段
axis_rx_start	input	1		Start
i_roce_prog_full	input	1	数据包 fifo 写入接口	Fifo 满
ov_roce_data	Output	256		数据
o_roce_wr_en	Output	1		写使能

5.4.5 接收仲裁器

5.4.5.1 模块功能

根据包的 ip 首部的协议字段和 udp 首部的端口字段判断，如果协议字段为 17，源端口目的端口都为 4791，则将包转发到 roce 接收引擎，否则转发给以太网接收引擎。

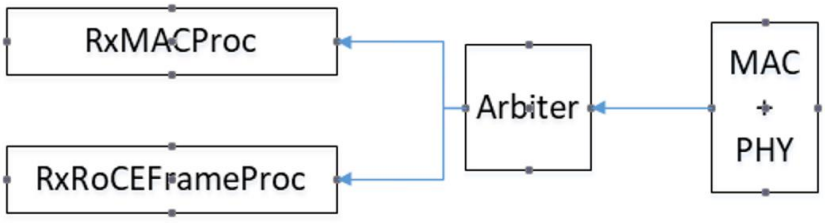


图 5-8 接收仲裁模块

5.4.5.2 模块接口

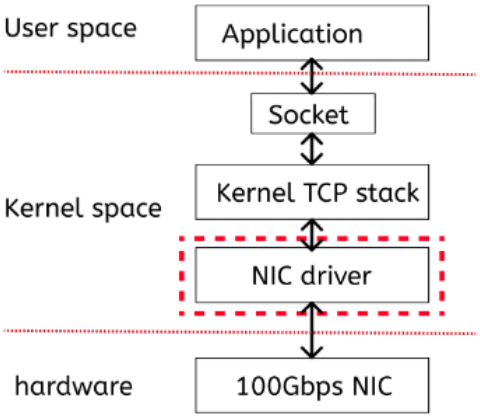
模块名	RX Engine			
模块说明	手工编码			
接口	**输入/输出**	**位宽**	**对接模块**	**说明**
clk	输入	1	/	时钟信号
rst	输入	1	/	复位信号
axis_rx_valid	input	1	MAC 接收接口	Valid
axis_rx_last	input	1		Last

axis_rx_data	input	256		数据
axis_rx_data_be	input	32		Byte enable
axis_rx_ready	Output	1		Ready
axis_rx_user	input	7		包长度
axis_rx_start	input	1		起始位
axis_rx_out_valid	input	2	与 roce 接收引擎 或者以太网包接 收引擎	Valid
axis_rx_out_last	input	2		Last
axis_rx_out_data	input	256		数据
axis_rx_out_data_be	input	32		Byte enable
axis_rx_out_ready	Output	2		Ready
axis_rx_out_user	input	14		包长度
axis_rx_out_start	input	2		起始位

6 软件驱动

6.1 驱动功能

驱动在系统中的定位如图所示，驱动向下与网卡对接，负责硬件初始化、硬件数据交互，向上与内核协议栈对接，将要发送的数据包信息传递给网卡，同时把接收到的数据包传递给内核协议栈。



网卡驱动在系统中定位

队列管理是网卡驱动的核心部分。驱动通过环形队列方式管理内存缓冲，保持与硬件队列的同步。为了减少内存占用，防止内存中没有大段连续内存的情况，适配更多的主机，环形队列中采用二级指针的方式，每个队列项为一个指针，指向为即将到来的数据包开辟的 MTU 大小的缓冲区。

为了保证网卡准确的将数据包 DMA 进指定的内存空间，同时释放已经发送的数据包的内存，驱动必须保持与网卡硬件队列的同步。在硬件中，维护队列的头尾指针，接收一个数据包，尾指针加一。驱动在中断函数中获取数据包时，先读取网卡相应队列的尾指针，更新驱动上的尾指针。当驱动检测到环形队列空余项不足时，补充一定的描述符，更新驱动头指针，同时将头指针下发给网卡硬件，更新硬件头指针。

6.2 驱动工作流程

下面以接收方向为例对网卡工作流程进行介绍。

1) 当数据包到达网卡后，经过 RSS，送入相应队列中，若队列有空余的空间（队列尾指针不等于头指针），队列的尾指针（tail_ptr）加一。数据包通过 DMA 送入队列描述符指向的内存中，然后向内存写入完成事件。

2) 数据包 DMA 完成后，网卡向 CPU 发起中断。CPU 根据中断号调用相应的中断处理函数，开启 polling 流程。

3) 在 polling 函数中，首先通过 IORead32 函数（对应网卡的 PIO）读取网卡队列的尾

指针（tail_ptr），若驱动内保存的队列尾指针（soft_tail_ptr）与网卡硬件的尾指针（tail_ptr）不同，则表明网卡消耗了一个描述符，网卡接收了一个新的数据包。之后驱动获取 soft_tail_ptr 中的数据包，拿到其完成事件中的数据包长度、校验和等信息，将数据包封装为 skb（内核协议栈处理数据包的数据结构），然后将 skb 送入内核协议栈。

4) polling 流程结束后，驱动判断剩余描述符数量。若描述符数量不足（head_ptr 与 tail_ptr 之差小于一定阈值），则将描述符补充至 1024，为新的 描述符开辟新的内存空间用于接收数据包，同时 head_ptr 增加相应数量，然后写入网卡。

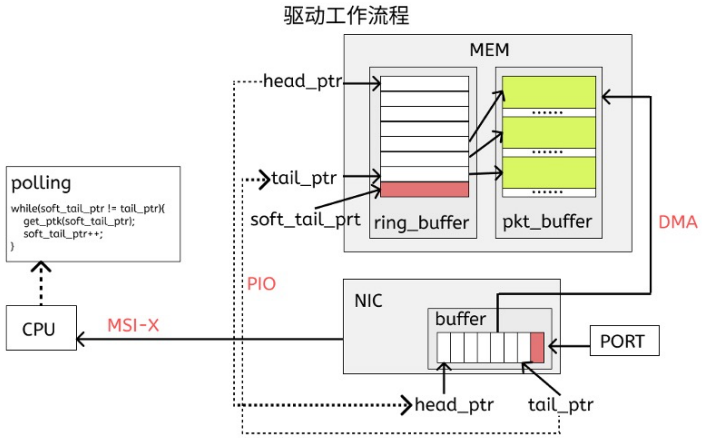


图 4-4 驱动工作流程