ICT NCIC 系统互连组

RDMA 引擎 详细设计文档

目录

RDMA 引擎 详细设计文档.....1
1 概述.....9
1.1 文档说明.....9
1.2 整体架构.....9
2 描述符处理（SendWQEProcessing）.....11
2.1 模块功能.....11
2.2 模块架构.....11
2.3 模块接口.....11
2.4 子模块设计.....11
2.4.1 门铃模块（DoorbellProcessing）.....11
2.4.2 描述符调度器（WQEScheduler）.....13
2.4.3 描述符处理（WQEParser）.....16
2.4.4 描述符记录表（WQEIndicatorTable）.....20
2.4.5 数据对齐模块（DataPack）.....21
3 请求引擎（RequestEngine）.....25
3.1 模块功能.....25
3.2 模块架构.....25
3.3 模块接口.....25
3.4 子模块设计.....25
3.4.1 请求方发送传输控制（RequesterTransControl）.....25
3.4.2 定时器模块（TimerControl）.....35
3.4.3 多队列模块（MultiQueue）.....38
3.4.4 请求包生成（ReqPktGen）.....49
3.4.5 请求方接收传输控制.....50
4 响应引擎（ResponderEngine）.....66
4.1 模块功能.....66
4.2 模块架构.....66
4.3 模块接口.....66
4.4 子模块设计.....66
4.4.1 请求执行引擎（ExecutionEngine）.....66
4.4.2 接收队列描述符管理（RQWQEManager）.....75
4.4.3 写地址记录表（WriteAddrTable）.....79
4.4.4 响应包生成（ResponsePacketGen）.....80
5 包头解析（HeaderParser）.....82

5.1 模块功能.....82
5.2 模块架构.....82
5.3 模块接口.....82
5.4 状态机设计.....83
5.4.1 状态说明.....83
5.4.2 状态转移图.....83
5.4.3 状态转移说明.....83
6 位宽转换（BitWidthTrans）.....85
6.1 模块功能.....85
6.2 模块架构.....85
6.3 模块接口.....85
6.4 子模块设计.....85
6.4.1 256To64Trans.....85
6.4.2 64To256Trans.....87
7 附录.....89
7.1 各种命令/响应格式.....89
7.1.1 门铃格式说明.....89
7.1.2 上下文访问及响应格式.....89
7.1.3 内存读写命令及响应格式.....90
7.1.4 定时器事件格式.....91
7.1.5 散列表管理模块命令及响应格式.....91
7.1.6 接收描述符管理模块及响应格式.....92
7.2 模块之间传递的元数据.....94
7.2.1 DoorbellProcessing 与 WQEScheduler.....94
7.2.2 WQEScheduler 与 WQEParser.....94
7.2.3 WQEParser 与 DataPack.....95
7.2.4 DataPack 与 RequesterTransControl.....97
7.2.5 RequesterTransControl 与 ReqPktGen.....97
7.2.6 RequesterRecvControl 与 ReqPktGen.....98
7.2.7 HeaderParser 与 RequesterRecvControl.....98
7.2.8 HeaderParser 与 ExecutionEngine.....99
7.2.9 ExecutionEngine 与 RespPktGen.....99
7.3 各模块用到的上下文信息及获取方式.....100

图目录

图 1.1 RDMA 引擎整体架构图.....10
图 2.1 描述符处理模块架构图.....11
图 2.2 门铃处理模块架构图.....12
图 2.3 门铃与描述符对应关系示意图.....14
图 2.4 WQEScheduler 状态机.....15
图 2.5 WQEParser 状态机.....19
图 2.6 气泡问题示意图.....22
图 2.7 DataPack 状态机.....23
图 3.1 请求引擎架构图.....25
图 3.2 数据包重传方式划分.....29
图 3.3 RequesterTransControl 状态转移图.....34
图 3.4 定时器原理图.....36
图 3.5 定时器更新时序图 1.....38
图 3.6 定时器更新时序图 2.....38
图 3.7 多队列模块示意图.....39
图 3.8 多队列机制.....39
图 3.9 多队列元素插入操作流程图.....45
图 3.10 多队列元素删除操作流程图.....45
图 3.11 RequesterControl 插入元素的时序图.....46
图 3.12 RequesterContro 预取缓存地址时序图.....47
图 3.13 RequesterControl 插入元素的时序图.....47
图 3.14 RequestPacketGen 状态转移图.....50
图 3.15 UPSN/RPSN/NPSN 相对位置示意图.....51
图 3.16 RDMA Read Response 丢失 PSN 示意图.....51
图 3.17 RequesterRecvControl 状态机-第 1 部分.....60
图 3.18 RequesterRecvControl 状态机-第 2 部分.....61
图 3.19 RequesterRecvControl 状态机-第 3 部分.....62
图 3.20 RequesterRecvControl 状态机-第 4 部分.....63
图 3.21 ScatterEntryManager 模块架构图.....64
图 3.22 ScatterEntryManager 状态转移图.....65
图 4.1 响应引擎架构图.....66
图 4.2 请求方包序列号检查示意图.....67
图 4.3 包头检查流程及对应错误码.....68
图 4.4 接受传输控制模块状态机-第 1 部分.....72

图 4.5 接受传输控制模块状态机-第 2 部分.....73
图 4.6 接受传输控制模块状态机-第 3 部分.....74
图 4.7 接受传输控制模块状态机-第 4 部分.....75
图 4.8 接收描述符格式.....76
图 4.9 接收描述符管理模块状态机.....78
图 4.10 响应包生成模块状态图.....81
图 5.1 包头解析器架构示意图.....82
图 5.2 包头解析器状态机.....83
图 6.1 位宽转换模块示意图.....85
图 6.2 256To64BitTrans 模块状态机.....87

表目录

表 1.1 RDMA 引擎支持的服务类型与操作类型 9
表 2.1 DoorbellProcessing 模块接口 12
表 2.2 DoorbellProcessing 传递的元数据格式 13
表 2.3 WQEScheduler 模块接口 14
表 2.4 WQEScheduler 状态转移说明 15
表 2.5 WQEParser 模块接口 17
表 2.6 WQEParser 状态转移说明 19
表 2.7 描述符记录表表项 20
表 2.8 WQEIndicatorTable 20
表 2.9 DataPack 模块接口 22
表 2.10 DataPack 状态转移说明 23
表 3.1 包头字段及对应元数据获取方式 27
表 3.2 RequesterTransControl 模块接口 30
表 3.3 RequesterTransControl 状态说明 33
表 3.4 RequesterTransControl 状态转移说明 34
表 3.5 定时器控制模块事件格式 ..... **错误!未定义书签。**
表 3.6 定时器控制模块事件说明 ..... **错误!未定义书签。**
表 3.7 Loss Timer 双端口 RAM 格式 36
表 3.8 RNR Timer 双端口 RAM 格式 36
表 3.9 TimerControl 模块接口 36
表 3.10 RPB List Table 表格式 40
表 3.11 RPB Element Table 表格式 40
表 3.12 RPB Free FIFO 格式 40
表 3.13 REB List Table 模块接口 40
表 3.14 REB Element Table 表格式 41
表 3.15 REB Free FIFO 表格式 41
表 3.16 SWPB List Table 表格式 41
表 3.17 SWPB Element Table 表格式 41
表 3.18 SWPB Free FIFO 格式 42
表 3.19 SWPB Free FIFO 模块接口 42
表 3.20 MultiQueue 模块接口 42
表 3.21 MultiQueue 缓冲区开销统计表 48
表 3.22 ReqPktGen 模块接口 49
表 3.23 ScatterEntryManager 命令类型 54

表 3.24 RequesterRecvControl 模块接口 55
表 3.25 RequesterRecvControl 状态说明 58
表 3.26 ScatterEntryManager 模块接口.....64
表 3.27 RequesterRecvControl 状态说明 65
表 4.1 ExecutionEngine 模块接口 69
表 4.2 ExecutionEngine 状态说明 71
表 4.3 RQWQEManager 模块接口.....76
表 4.4 RQWQEManager 状态说明.....77
表 4.5 WQEParser 状态转移说明.....78
表 4.6 Write Addr Table 表格式 79
表 4.7 WriteAddrTable 模块接口.....80
表 4.8 ResponsePacketGen 模块接口 80
表 4.9 ResponsePacketGen 状态说明 81
表 5.1 HeaderParser 模块接口 82
表 5.2 HeaderParser 状态转移说明 83
表 6.1 256To64Trans 模块接口.....85
表 6.2 64To256Trans 模块接口.....87
表 7.1 门铃格式.....89
表 7.2 DoorbellProcessing 模块接口 89
表 7.3 上下文读取命令格式.....89
表 7.4 内存读写命令格式.....90
表 7.5 内存读写命令字段说明.....90
表 7.6 内存读写响应格式.....90
表 7.7 内存读写响应各字段说明.....90
表 7.8 定时器设定事件格式.....91
表 7.9 定时器设定事件各字段说明.....91
表 7.10 定时器超时事件格式.....91
表 7.11 定时器设定事件各字段说明.....91
表 7.12 散列表管理模块命令格式.....92
表 7.13 散列表管理模块命令说明.....92
表 7.14 散列表管理模块响应格式.....92
表 7.15 散列表管理模块响应说明.....92
表 7.16 接收描述符管理模块命令格式.....92
表 7.17 接收描述符管理模块命令说明.....93
表 7.18 接收描述符管理模块响应格式.....93

表 7.19 接收描述符管理模块响应说明.....93
表 7.20 DoorbellProcessing 传递给 WQEScheduler 的元数据格式.....94
表 7.21 DoorbellProcessing 传递给 WQEScheduler 的元数据字段说明.....94
表 7.22 WQEParser 传递给 DataPack 的公有元数据格式.....95
表 7.23 DoorbellProcessing 传递给 WQEScheduler 的公有元数据字段说明.....95
表 7.24 WQEParser 传递给 DataPack 的散列表项格式.....96
表 7.25 DoorbellProcessing 传递给 WQEScheduler 的公有元数据字段说明.....96
表 7.26 WQEParser 传递给 DataPack 的 RDMA 操作元数据格式.....96
表 7.27 DoorbellProcessing 传递给 WQEScheduler 的 RDMA 操作元数据字段说明.....97
表 7.28 WQEParser 传递给 DataPack 的原子操作元数据格式.....97
表 7.29 DoorbellProcessing 传递给 WQEScheduler 的原子操作元数据字段说明.....97
表 7.30 RequesterTransControl 传递给 ReqPktGen 的包头元数据类型说明.....98
表 7.31 RequesterRecvControl 传递给 ReqPktGen 的包头元数据类型说明.....98
表 7.32 HeaderParser 传递给 RequesterRecvControl 的包头元数据类型说明.....99
表 7.33 ExecutionEngine 传递给 RespPktGen 的元数据格式.....99
表 7.34 ExecutionEngine 传递给 RespPktGen 的元数据说明.....99
表 7.35 各模块与上下文信息对照.....100

1 概述

1.1 文档说明

本文档给出 IB HCA 架构中 RDMA 引擎的硬件架构详细设计方案，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。

RDMA 引擎的设计遵循标准 InfiniBand 协议（版本 1.3），向上兼容 libibverbs 通信库。在本设计中，RDMA 引擎支持的服务类型和操作类型如所示。

表 1.1 RDMA 引擎支持的服务类型与操作类型

服务类型 / 操作类型	RC	UC	UD
Send/Recv	√	√	√
RDMA Write	√	√	×
RDMA Read	√	×	×
Atomics	√	×	×

1.2 整体架构

RDMA 引擎的整体架构如图 1.1 所示，主要可以划分为四个部分：1.描述符处理；2.请求引擎；3.响应引擎；4.包头解析与位宽转换。

整个数据流可以按照请求方/响应方、发送/接收划分为 4 部分：

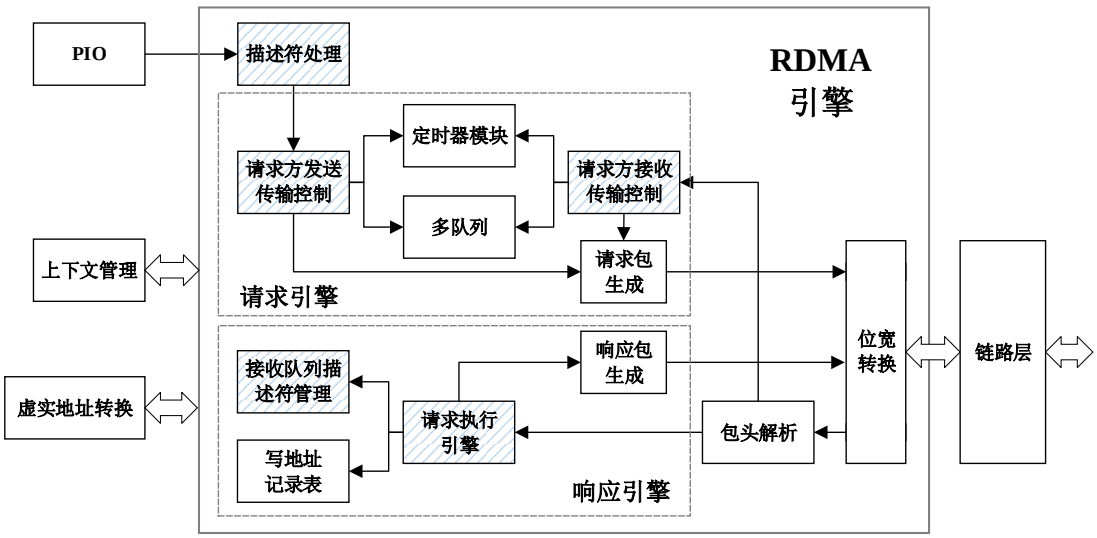
1.**请求方发送处理路径（Requester Transmit Path）**：描述符处理模块接收主机下发的门铃，根据门铃中携带的信息从内存中获取描述符，再根据操作类型从内存中获取数据传递给请求方发送传输控制模块进行数据传输层的处理；发送传输控制模块根据当前服务类型与操作类型进行数据的缓存与分段，并更新相关的发送状态信息，将元数据和负载交由请求包生成模块发送；由于当前链路层使用的接口位宽为 64-bit，而 RDMA 引擎内部总线位宽为 256-bit，因此，还需要位宽转换模块进行数据转换与对齐。

2.**在请求方接收处理路径（Requester Recv Path）**：包头解析模块将链路层上传的数据包进行解析，不断提取各个包头，生成元数据，将元数据和负载一同传递给请求方接收传输控制模块；请求方接收传输控制模块根据当前收到的数据包携带的包头信息和连接状态，对数据包进行不同的处理，例如写入内存（Read 响应包）、丢弃（重复包）或重传（错误指示包）。

3.在响应方接收处理路径（Responder Recv Path）：请求执行引擎根据数据包的类型和当前连接状态执行请求，例如写入内存（Send/Write 请求包），从内存中获取数据（Read/Atomics 包）、丢弃（错误包）等。

4.在响应方发送处理路径（Responder Send Path）：请求执行引擎将执行结果传递给响

应包生成模块，由响应包生成模块构造不同类型的响应包，例如 ACK（Send/Write 的响应），RDMA Read Response 包（Read 的响应）。



图中阴影部分模块需要与上下文管理单元和虚实地址转换模块进行数据交互，为简洁起见，省略了这些连接线，后文中的图均遵循此规范

图 1.1 RDMA 引擎整体架构图

2 描述符处理（SendWQEProcessing）

2.1 模块功能

描述符处理模块完成的功能为门铃读取、描述符读取、描述符调度、描述符解析、数据读取、数据打包。每个子功能由对应的子模块完成。该模块的输入是由 PIO 模块下发的门铃，向传输层发送控制模块输出的是请求的元数据和其对应的数据负载。

2.2 模块架构

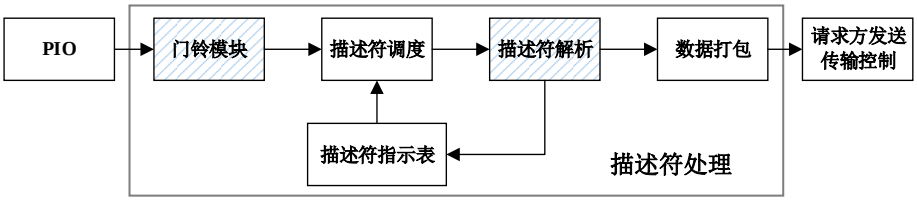


图 2.1 描述符处理模块架构图

2.3 模块接口

由内部子模块直接引出外部接口。

2.4 子模块设计

2.4.1 门铃模块（DoorbellProcessing）

2.4.1.1 模块功能

DoorbellProcessing 完成三个处理过程：1.从与 PIO 模块对接的门铃队列中读取门铃；2.根据门铃中携带的信息获取 QP 上下文信息；3.根据上下文信息读取当前 SQ 中待处理的 WQE。

2.4.1.2 模块架构

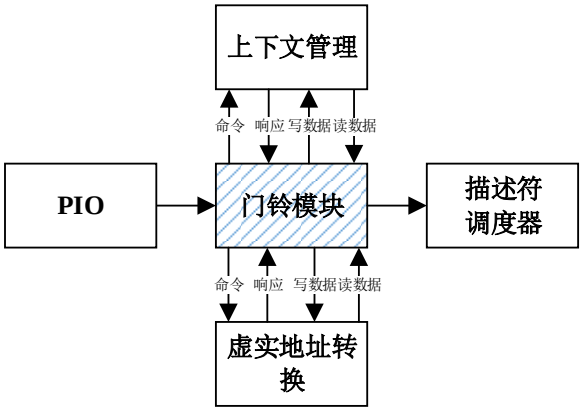


图 2.2 门铃处理模块架构图

需要说明的是，门铃模块与上下文管理和虚实地址转换之间都有四组接口，分别对应于命令接口，响应接口，读数据接口和写数据接口。

对于上下文的更新，写命令与写数据同步发出，不需要等待响应；对于虚实地址转换模块的访问，由于需要等待访问权限检查结果，因此，先发出命令后需要等待响应返回，权限检查通过后才能够发出写数据。

其它所有模块与上下文和虚实地址转换模块的交互都遵循此设计。

2.4.1.3 模块接口

表 2.1 DoorbellProcessing 模块接口

模块名	DoorbellProcessing			
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_pio_tvalid	输入	1	PIO	门铃数据获取
o_pio_tready	输出	1		
iv_pio_tdata	输入	64		
o_cxtmgt_cmd_tvalid	输出	1	CxtMgt	读取上下文
i_cxtmgt_cmd_tready	输入	1		
ov_cxtmgt_cmd_tdata	输出	128		
i_cxtmgt_resp_tvalid	输入	1		上下文响应
o_cxtmgt_resp_tready	输出	1		
iv_cxtmgt_resp_tdata	输入	128		
o_vtp_cmd_tvalid	输出	1	VirtToPhys	读命令接口
i_vtp_cmd_tready	输入	1		
ov_vtp_cmd_tdata	输出	256		

i_vtp_resp_tvalid	输入	1		读响应接口（返回权限检查结果）
o_vtp_resp_tready	输出	1		
iv_vtp_resp_tdata	输入	8		
i_vtp_wqe_tvalid	输入	1		读数据接口
o_vtp_wqe_tready	输出	1		
iv_vtp_wqe_tdata	输入	256		
o_wqe_tvalid	输出	1	WQEScheduler	前递 WQE
i_wqe_tready	输入	1		
ov_wqe_tdata	输出	256		
o_metadata_tvalid	输出	1		前递元数据
i_metadata_tready	输入	1		
ov_metadata_tdata	输出	96		

表 2.2 DoorbellProcessing 传递的元数据格式

2.4.1.4 状态机设计

Doorbell 功能比较简单，不需要实现状态机，划分为三段流水线处理即可（以 Pn 代表第 n 级流水）：

P1：若门铃队列非空，获取门铃，并根据门铃中携带的 QP 号从上下文管理模块中读取发送队列（SQ）的内存区域首地址 VA，以及该区域相关的 LKey；

P2：接收上下文管理模块返回的 VA 和 LKey，根据门铃中包含的信息发起对 WQE 的内存读请求；

P3：接收读响应，将 QPN 和 WQE 传递给描述符调度器进行处理。

2.4.2 描述符调度器（WQEScheduler）

2.4.2.1 模块功能

发送描述符的读取有两种方式：1.门铃模块根据门铃信息读取；2.描述符解析模块根据描述符里的字段从内存中读取。因此，每个门铃可以对应着多个描述符的提交。

图给出了提交门铃与描述符的对应关系示意。假设门铃 0 对应于提交描述符 0~2，门铃 1 对应于提交描述符 3。当门铃 0 和门铃 1 下发到网卡时，门铃模块首先处理门铃 0，从内存中读取描述符 0，描述符 1 的读取必须由描述符解析模块等到描述符 0 从内存返回后才能进行；而门铃模块在发起描述符 0 的读取后，就开始处理门铃 1，此时如果不加以限制，描述符 3 将先于描述符 1 和 2 被处理，违背了描述符的保序原则。

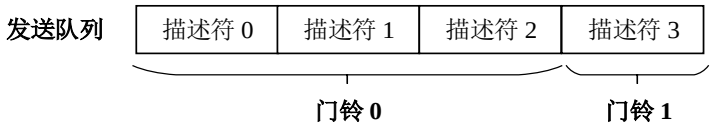


图 2.3 门铃与描述符对应关系示意图

描述符调度器的功能就是为了实现保序性。门铃模块读回的描述符和描述符解析模块读回的描述符将被缓存至两个队列中，描述符调度器将根据一定的规则从两个队列中调度描述符向描述符解析模块传递。

调度规则定义如下：记门铃模块读回的描述符队列为 Q1，描述符解析模块读回的描述符队列为 Q2。优先调度 Q2 中的描述符处理，如果 Q2 为空，调度 Q1 中的描述符。在处理 Q1 中的队首描述符时，查找该描述符所属 QP 在描述符记录表中的对应标志位是否清空（描述符记录表见 2.4.4），如果清空，则可以调度该描述符，如果对应标志位被置位，则继续调度 Q2。

2.4.2.2 模块接口

表 2.3 WQEScheduler 模块接口

模块名	WQEScheduler			
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_wqe_from_db_empty	输入	1	DoorbellProcessing	从门铃处理模块获取描述符
o_wqe_from_db_rd_en	输出	1		
ov_wqe_from_db_data	输出	256		
i_md_from_db_empty	输入	1		从门铃处理模块获取元数据
o_md_form_db_rd_en	输出	1		
ov_md_from_db_data	输出	256		
i_vtp_wqe_empty	输入	1	VirtToPhys	获取描述符解析模块读回的描述符
o_vtp_wqe_rd_en	输出	1		
iv_vtp_wqe_data	输入	256		
i_wqe_prog_full	输入	1	WQEParser	输出调度后的描述符
o_wqe_wr_en	输出	1		
ov_wqe_data	输出	256		
i_md_fwd_prog_full	输入	1		向描述符解析模块前递元数据
o_md_fwd_wr_en	输出	1		

ov_md_fwd_data	输出	256		
i_md_back_empty	输入	1		从描述符解析模块获取元数据
o_md_back_rd_en	输出	1		
iv_md_back_data	输入	64		
ov_wit_rd_addr	输出	14	WQEIndicatorTable	从描述符记录表中读取表项
iv_wit_rd_data	输入	1		

2.4.2.3 状态机设计

2.4.2.3.1 状态转移图

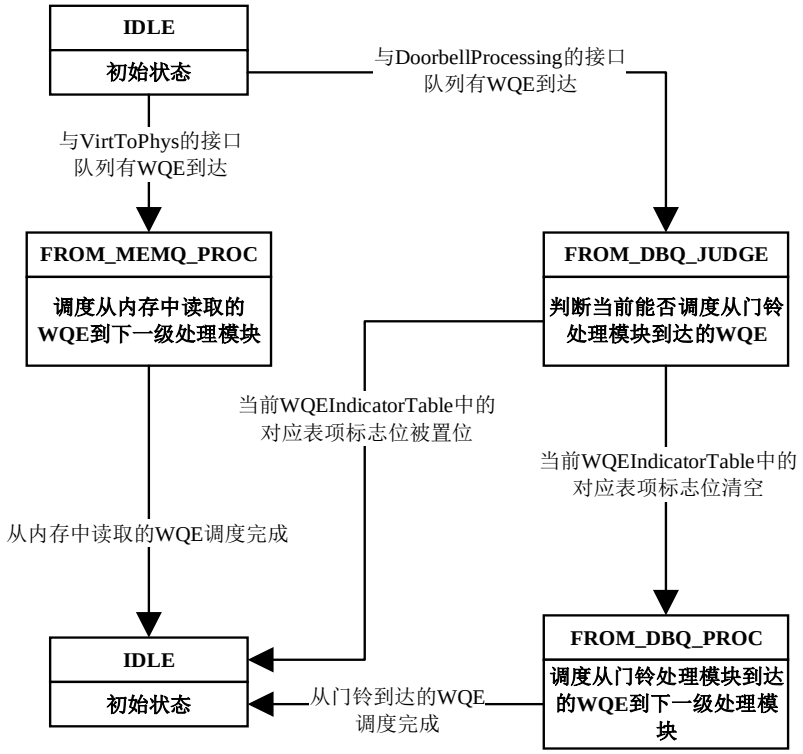


图 2.4 WQEScheduler 状态机

2.4.2.3.2 状态转移说明

需要说明的是，状态转移说明并不覆盖状态机中所有的状态跳转，只针对个别状态机图中难以详细表示的关键跳转条件进行补充说明。后续所有状态机描述均遵循此约定。

表 2.4 WQEScheduler 状态转移说明

现态	次态	转移条件
FROM_DBQ_JUDGE	FROM_DBQ_PROC	当前在调度 Q1 中的描述符，根据其 QP 号从描述符表中读取对应标志位，标志位清零
	IDLE	当前在调度 Q1 中的描述符，根据其 QP 号从描述符

		表中读取对应标志位，标志位置位

2.4.3 描述符处理（WQEParser）

描述符处理模块接收从描述符调度器传来的描述符，解析描述符中的各个描述符段（Seg），针对不同的段进行相应的操作。不同段的具体格式参见文档《》。

2.4.3.1 模块功能

2.4.3.1.1 处理不同字段获取相应元数据

处理 NextSeg：所有发送描述符中都会携带该段，需要从该段中获取的信息为下一个描述符的相关信息和当前操作携带的立即数（如果有立即数）。

如果下一个描述符的大小不为 0，则发起对下一个描述符的读取请求，同时，将描述符记录表中对应表项的标志位置位；如果下一个描述符的大小为 0，则将描述符记录表中对应表项的标志位清空，并处理下一个字段。

处理 RaddrSeg：RDMA 请求和原子操作会携带该字段，是远端内存区域的描述符；

处理 AtomicSeg：原子操作携带该段，描述原子操作的源数据；

处理 UDSeg：UD 服务类型用到该段，用于生成 BTH 头部中的 Q_Key 等信息；

处理 InlineSeg：Inline 数据发送模式下使用该段，包含一个 32 位的数值；

处理 DataSeg：Send/RDMA Write/RDMA Read 操作均会携带该段，用于描述源节点的内存区域。对于 Send/RDMA Write，根据该段中携带的 Gather Entry List，从内存中读取待发送的数据；对于 RDMA Read，将该段中携带的 Scatter Entry List 传递给传输层控制模块，用于后续将 RDMA Read 响应包写入源节点内存。

2.4.3.1.2 错误检查与处理

QP 状态检查：若当前 QP 的状态为 Error，则当前 WQE 将不会被处理，直接为其生成元数据，元数据中的错误码为“Flush in Error”；若当前 QP 的状态非 Error，则进行后续检查。

操作类型检查：由于不同的服务类型所允许的操作类型是不一样的，因此，在处理一个 WQE 之前，必须对其进行操作类型检查。如果 QP 使用的服务类型不支持当前的操作类型，则应当进行错误处理。WQE 对应的 QP 支持的操作类型保存在上下文信息中，因此，在处理一个 WQE 时，需要向 CxtMgt 获取 QP 中支持的操作类型。

错误处理即为出错的 WQE 生成元数据，元数据中的错误码为“Unsupported OpCode”。需要注意的是，当 WQE Processing 检查到错误时，不会将 QP 的状态修改为 Error，这是因为当 QP 状态变为 Error，Send Path Processing 会认定此时开始收到的 Request 都应该以错误状态完成。但实际上，在错误 WQE 之前提交的 WQE（例如某些请求暂存在 WQE Processing 和 Send Path Processing 之间的 FIFO 中尚未被处理）应该被正确执行。因此，WQE Processing

在检测到错误时，不转换 QP 状态，而是将错误信息传递给 Send Path Processing，由该模块检查到错误信息后再切换到状态。

本地内存访问检查：WQE Processing 在发起内存请求时，需要向 VirtToPhys 传递 LKey 以便 VirtToPhys 进行本地内存访问检查。检查的内容主要包括两个方面：

1.权限检查：当前内存区域的读写权限是否包含要求的权限；

2.边界检查：当前内存区域是否包含请求的地址区域。

检查结果由 VirtToPhys 返回。如果检查失败，则会产生一个 B 类错误。此时，在发给 Send Path Processing 的元数据中指示该错误。

请求长度检查：对于 RC/UC 服务，检查请求中的本地缓冲区和目的缓冲区大小是否小于 2GB。对于 RDMA Read/Write，检查其 SGL Entry 中的长度和 mthca_raddr_seg 中的长度是否相等且小于 2GB；对于 Send 操作，如果使用了 mthca_inline_seg，检查该字段中包含的长度是否小于 2GB，如果使用了 mthca_data_seg，则对所有 mthca_data_seg 中的长度进行累加，判断是否小于 2GB。对于 UD 服务，其只支持 Send，因此，检查其 mthca_inline_seg 或 mthca_data_seg 中的长度是否小于 PMTU。

如果检查失败，则将错误码包含在元数据中，发送给 Send Path Processing。

2.4.3.1.3 元数据传递

从 WQEScheduler 接收的元数据：

向 WQEScheduler 写入的元数据：

向 DataPack 写入的元数据：

2.4.3.2 模块接口

表 2.5 WQEParser 模块接口

模块名	WQEParser			
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_wqe_empty	输入	1	WQEScheduler	获取描述符
iv_wqe_data	输入	128		
o_wqe_rd_en	输出	1		
i_md_fwd_empty	输入	1		获取描述符元数据
iv_md_fwd_data	输入	64		
o_md_fwd_rd_en	输出	1		
i_md_back_prog_full	输入	1		回传元数据（由描述符

o_md_back_wr_en	输出	1		调度器使用）
ov_md_back_data	输出	64		
o_wit_wr_en	输出	1	WQEIndicatorTable	描述符记录表接口
ov_wit_wr_addr	输出	10		
ov_wit_wr_data	输出	25		
i_md_to_dp_prog_full	输入	1	DataPack	传递请求元数据
o_md_to_dp_wr_en	输出	1		
ov_md_to_dp_data	输出	256		
i_inline_prog_full	输入	1		传递 inline 数据
o_inline_wr_en	输出	1		
ov_inline_data	输出	256		
i_entry_prog_full	输入	1		传递 Scatter Entry
o_entry_wr_en	输出	1		
ov_entry_data	输出	128		
i_atomics_prog_full	输入	1		传递原子操作元数据
o_atomics_wr_en	输出	1		
ov_atomics_data	输出	128		
i_raddr_prog_full	输入	1		传递远端地址元数据
o_raddr_wr_en	输出	1		
ov_raddr_data	输出	128		
i_cxtmgt_cmd_prog_full	输入	1	CxtMgt	上下文读写命令接口
o_cxtmgt_cmd_wr_en	输出	1		
ov_cxtmgt_cmd_data	输出	128		
i_cxtmgt_resp_empty	输入	1		上下文响应接口
iv_cxtmgt_resp_data	输入	128		
o_cxtmgt_resp_rd_en	输出	1		
o_vtp_wqe_cmd_wr_en	输出	1	VirtToPhys	内存描述符读取接口
i_vtp_wqe_cmd_prog_full	输入	1		
ov_vtp_wqe_cmd_data	输出	256		
i_vtp_wqe_resp_empty	输入	1		内存描述符响应接口
o_vtp_wqe_resp_rd_en	输出	1		
iv_vtp_wqe_resp_data	输入	4		
o_vtp_nd_cmd_wr_en	输出	1		网络数据读取接口
i_vtp_nd_cmd_prog_full	输入	1		

ov_vtp_nd_cmd_data	输出	256		
i_vtp_nd_resp_empty	输入	1		网络数据响应接口
o_vtp_nd_resp_rd_en	输出	1		
iv_vtp_nd_resp_data	输入	4		

2.4.3.3 状态机设计

2.4.3.3.1 状态转移图

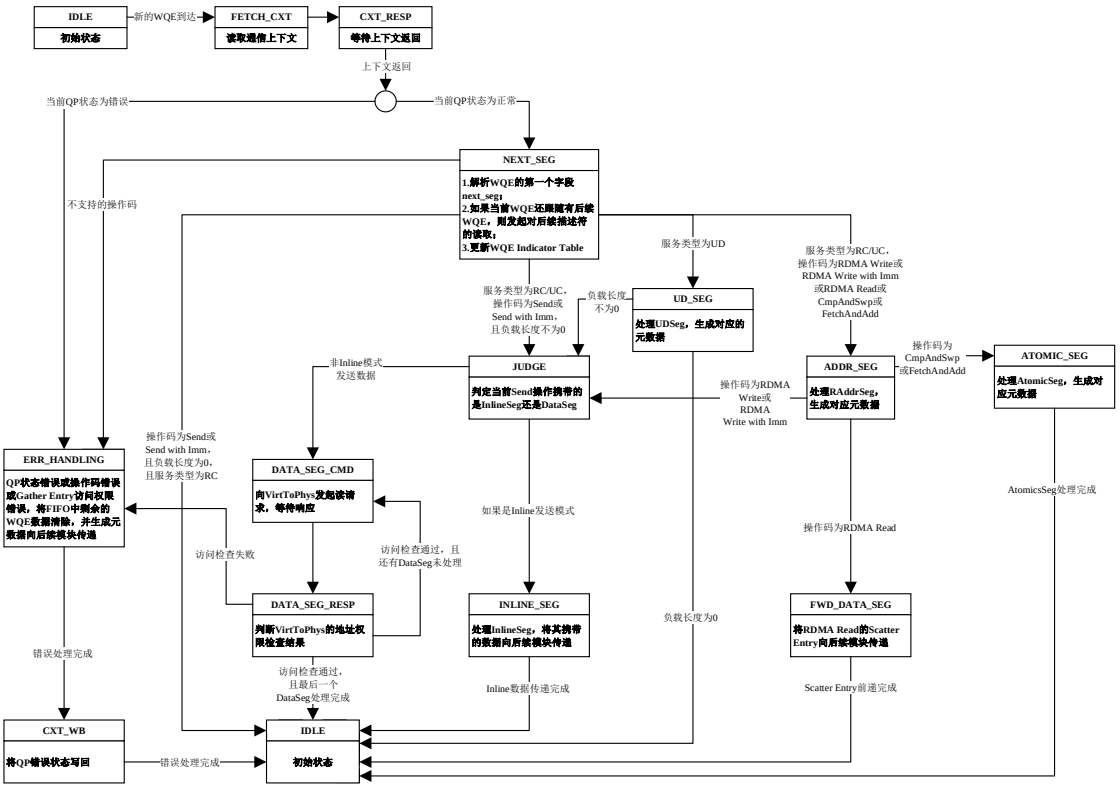


图 2.5 WQEParser 状态机

2.4.3.3.2 状态转移说明

表 2.6 WQEParser 状态转移说明

现态	次态	转移条件
NEXT_SEG	IDLE	如果当前发送操作为 Send，且发送数据长度为 0。需要注意的是，在描述符中并不携带需要发送的数据长度，我们只知道每个 WQE 的长度。因此，我们维护了一个计数器，每次处理一个 Seg 就将计数器自减，如果处理 NextSeg 时已经是描述符中的最后一个段了，则说明当前操作不携带数据。
ADDR_SEG	JUDGE	操作码为 RDMA Write 或 RDMA Write with Imm，进

		入发送数据方式判定阶段。
	FWD_DATA_SEG	操作码为 RDMA Read，进入 Scatter Entry 前递阶段。
	ATOMIC_SEG	操作码为 FetchAndAdd 或者 CmpAndSwp，进入 AtomicSeg 处理阶段。
JUDGE	DATA_SEG_CMD	当前 Send/RDMA Write 操作使用 Gather Entry 方式发送数据，进入内存读命令阶段。判定方式是：当第一个 DataSeg 的最高位位 1 时，表明是 Inline 操作，否则是 Gather Entry 方式。
	INLINE_SEG	当前 Send/RDMA Write 操作使用 Inline 方式发送数据，处理描述符中携带的发送数据。

2.4.4 描述符记录表（WQEIndicatorTable）

2.4.4.1 模块功能

该模块用于协同实现描述符处理的保序，其内部维护了一个记录表，表项内容为 1 比特的标志位，供描述符调度器和描述符解析模块之间进行同步。具体同步方式见 2.4.2.1。

在具体实现时，描述符记录表内部例化了一个 SDP（Single Dual Port RAM），一个端口被描述符调度器用来读取对应表项，另一个端口被描述符解析模块用来更新对应表项。

表 2.7 描述符记录表表项

索引号	表项内容（1-bit）
0	flag（0 或 1）
1	flag（0 或 1）
… …	… …
16383	flag（0 或 1）

2.4.4.2 模块接口

表 2.8 WQEIndicatorTable

模块名	WQEScheduler			
模块说明	IPCore：Simple Dual Port RAM；深度：16384：读写宽度：1			
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

wr_en	输入	1	WQEParser	表项更新接口
wr_addr	输入	14		
wr_data	输入	1		
rd_en	输入	1	WQEScheduler	表项读取接口
rd_addr	输入	14		
rd_data	输出	1		

2.4.5 数据对齐模块（DataPack）

2.4.5.1 模块功能

DataPack 模块接收由 WQEParser 模块传递的数据，将多个数据段中可能存在的气泡消除，将其对齐到 256-bit 的数据缓冲队列中，将其交付给传输层模块处理；同时，对于一些出错的请求，DataPack 还需要将已经读回的错误数据从与 VirtToPhys 的接口队列中清除。

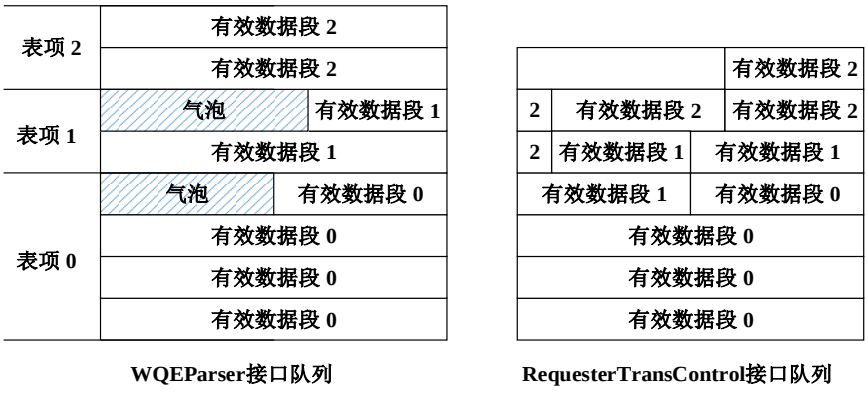
2.4.5.1.1 数据对齐

DataPack 需要打包的网络数据有两个来源：

其一是 Inline 数据，由 WQEParser 直接从描述符中提取得到，这部分数据的位宽是 128 比特，DataPack 需要将其对齐到 256 比特，该过程较为简单，不再赘述；

其二是 Gather 数据，由 WQEParser 根据描述符中的 Gather Entry 从内存中读取。由于待发送的数据是根据多个 Entry 从内存中获取的，而每个 Entry 描述的数据区域不一定是 256 比特对齐的，因此，由不同 Entry 读回的数据段之间可能会存在“气泡”，DataPack 需要将数据段中的气泡消除掉，然后将其拼接为连续的数据负载。

图 2.6 气泡问题示意图图 2.6 展示了该问题。左图是 DataPack 与 WQEParser 对接的数据 FIFO，右图是 DataPack 与 RequesterTransControl 的接口队列。图中 WQEParser 通过三个表项读回了待发送的数据，除了数据段 2，其余两个数据段均没有与 256 比特对齐，因此，数据段 0 和数据段 1 的末尾存在气泡。DataPack 在消除气泡后将连续的数据写入下级模块的对接 FIFO 中。



WQEParser接口队列　　RequesterTransControl接口队列

图 2.6 气泡问题示意图

2.4.5.1.2 错误数据清除

在 WQEParser 处理描述符时，有可能遇到出错的 Gather Entry，此时，由于出错表项之前的读请求已经发出，因此，会有一部分数据被读回。DataPack 需要将这部分错误数据清除，然后将相应的元数据传递给后续模块。

错误数据的清除较为简单，当 DataPack 收到元数据时，如果元数据中指示当前请求中有无效表项，则 DataPack 根据已经收到的表项将读回的数据清除即可。

2.4.5.2 模块接口

表 2.9 DataPack 模块接口

模块名	DataPack			
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_md_from_wp_empty	输入	1	WQEParser	获取请求元数据接口
o_md_from_wp_rd_en	输出	1		
iv_md_from_wp_data	输入	256		
i_inline_empty	输入	1		获取 Inline 数据
o_inline_rd_en	输出	1		
iv_inline_data	输入	256		
i_entry_from_wp_empty	输入	1		获取 SGL Entry
o_entry_from_wp_rd_en	输出	1		
iv_entry_from_wp_data	输入	128		
i_atomics_from_wp_empty	输入	1		获取原子操作元数据
o_atomics_from_wp_rd_en	输出	1		
iv_atomics_from_wp_data	输入	128		
i_raddr_from_wp_empty	输入	1		获取远端地址元数据
o_raddr_from_wp_rd_en	输出	1		
iv_raddr_from_wp_data	输入	128		
i_sgl_empty	输入	1	VirtToPhys	获取由 SGL 读取的数据
o_sgl_rd_en	输出	1		
iv_sgl_data	输入	256		
i_entry_to_sre_prog_full	输入	1	RequesterTransControl	传递 Scatter Entry

o_entry_to_sre_wr_en	输出	1		
ov_entry_to_sre_data	输出	128		
i_atomics_to_sre_prog_full	输入	1		传递原子操作元数据
o_atomics_to_sre_wr_en	输出	1		
ov_atomics_to_sre_data	输出	128		
i_raddr_to_sre_prog_full	输入	1		传递远程地址元数据
o_raddr_to_sre_wr_en	输出	1		
ov_raddr_to_sre_data	输出	128		
i_nd_to_sre_prog_full	输入	1		传递请求数据
o_nd_to_sre_wr_en	输出	1		
ov_nd_to_sre_data	输出			
i_md_to_sre_prog_full	输入	1		传递请求元数据
o_md_to_sre_wr_en	输出	1		
ov_md_to_sre_data	输出	256		

2.4.5.3 状态机设计

2.4.5.3.1 状态转移图

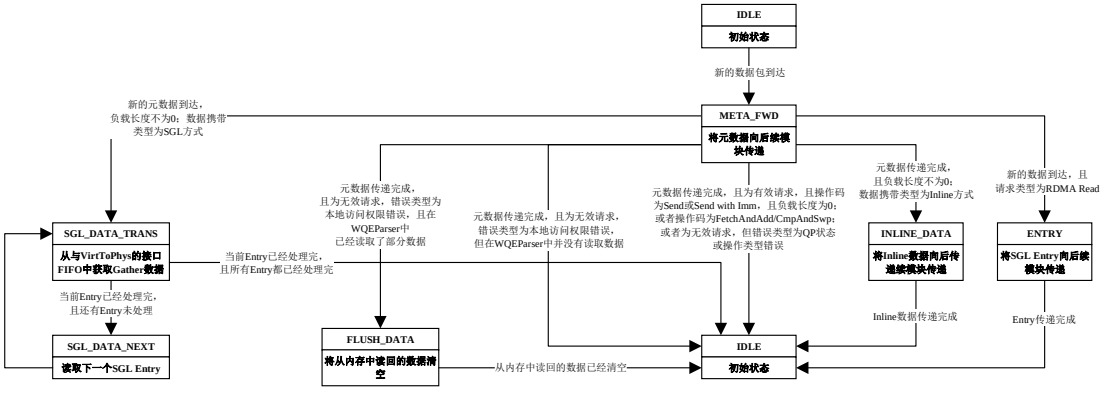


图 2.7 DataPack 状态机

2.4.5.3.2 状态转移说明

表 2.10 DataPack 状态转移说明

现态	次态	转移条件
IDLE	META_FWD	新的请求到达，向后续模块转发元数据
META_FWD	ENTRY	元数据传递完成，请求类型为 RDMA Read
	INLINE_DATA	元数据传递完成，负载长度不为 0，数据负载读取方式为 Inline

	FLUSH_DATA	元数据传递完成，且为无效请求，错误类型为本地访问权限错误，且在 WQEParser 中已经读回了一部分数据
	SGL_DATA_TRANS	元数据传递完成，请求负载长度不为 0，且数据数据携带方式为 SGL，且请求有效
	IDLE	1.元数据传递完成，且为有效请求，且操作码为 Send 或 Send with Imm，且负载长度为 0； 2.操作码为 FetchAndAdd 或 CmpAndSwp； 3.无效请求，错误码为 QP 状态错误或操作类型错误； 4.无效请求，错误码为本地访问权限错误，且在 WQEParser 中没有读到数据。
SGL_DATA_TRANS	SGL_DATA_NEXT	当前 Entry 对应的数据已经处理完成，且还有 Entry 未处理
	IDLE	当前 Entry 对应的数据已经处理完，且所有 Entry 都已经处理完
SGL_DATA_NEXT	SGL_DATA_TRANS	当前 Entry 解析完成
FLUSH_DATA	IDLE	从内存中读回的无效数据已经清空

3 请求引擎（RequestEngine）

3.1 模块功能

请求引擎的功能主要可以划分为以下两个部分：1.处理要发送的 Request；2.处理接收到的 Response。

对 Request 的处理包含以下方面：

1.将待发送的负载进行分段，为每段负载生成元数据，以便请求包生成模块生成对应的包头；

2.提供可靠性传输保证，对于 RC 服务，将每个发送的数据包暂存起来，以便丢包重传；

3.对错误的请求进行处理。

对 Response 的处理包含以下方面：

1.对于 RC 服务，如果是 ACK 响应包，则根据响应包中的 PSN 释放缓冲区；如果是 RDMA Read 响应包，还需要通过 Scatter List 将响应包中的数据写回内存；

2.如果检测到丢包，或者定时器超时，需要将发送缓冲区中的数据进行重传。

上述功能将由不同的子模块实现。

3.2 模块架构

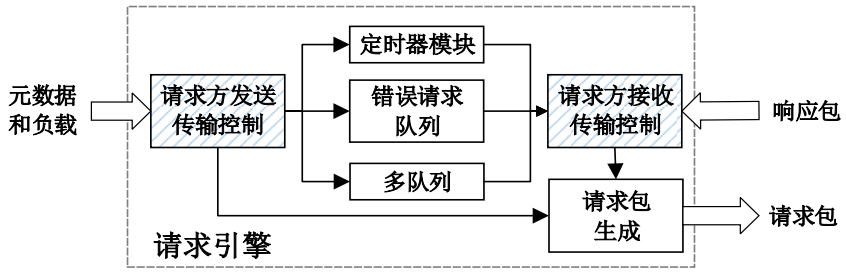


图 3.1 请求引擎架构图

3.3 模块接口

由内部子模块直接引出外部接口。

3.4 子模块设计

3.4.1 请求方发送传输控制（RequesterTransControl）

3.4.1.1 模块功能

请求方发送传输控制模块需要对接收到的元数据进行解析，针对不同的服务类型和操作

类型进行相应的处理。其处理流程如下：

1.对于每个请求的元数据，该模块首先需要读取通信上下文信息，将上下文信息缓存在本地寄存器中；

2.将数据负载按照 PMTU（Path Maximum Transfer Unit）进行分段，为切分后的每段负载生成元数据；

3.针对不同的服务类型进行不同的传输控制处理；

4.将更新后的上下文信息写回。

本节将逐次介绍每个功能的具体实现方式。

3.4.1.1.1 发送状态信息读取

在开始处理一个请求时，需要获取通信上下文，其主要作用在于由上下文信息生成对应对应的元数据。

对于上下文信息的获取，本设计中采用的是以下设计方案，对于属性为只读的上下文，由门铃模块读取后沿着发送路径逐次传递给后续模块使用；对于属性为读写的上下文，由各个模块在使用时自行读取。各个模块需要用到的上下文信息见附录。

3.4.1.1.2 数据负载分段及元数据生成

3.4.1.1.2.1 分段机制

数据负载分段指的是当待发送的负载长度超过 PMTU 后，需要将负载按照 PMTU 的长度进行切分，切分后的每段负载都将被封装为一个独立的数据包传输。需要进行分段的操作类型包括：Send，Send with Immediate，RDMA Write，RDMA Write with Immediate。对于 RDMA Read 和 Atomics 操作而言，其请求包不携带负载，因此不需要进行分段。

在实际实现中，为每个需要分段的请求维护一个长度计数器，计数器初值设定为消息长度。如果计数器大小不为 0，则进行分段，为当前数据生成一个包头，然后将计数器自减，进入下一次判断。详细状态跳转见状态机设计及说明。

在分段过程中，大部分包头数据是通过读取已有信息获取的，只有 Send 和 RDMA Write 的操作码（Operation Code）需要根据消息长度信息和当前分段状态计算得到，以 Send 操作为例，其生成规则如下：

1.如果待发送消息的长度小于等于 1 个 PMTU，则数据包的 OpCode 为 Send Only；

2.如果待发送消息的长度大于 1 个 PMTU，小于等于两个 PMTU，则数据包的 OpCode 分别为 Send First，Send Last；

3.如果待发送消息的长度大于两个 PMTU，则数据包的 OpCode 分别为 Send First，Send Middle 和 Send Last。

3.4.1.1.2.2 包头元数据生成

在每次进行数据分段产生一个新的数据包时，都需要为其生成对应的包头元数据。不同

请求操作类型的包头元数据及生成方式如所示。

表 3.1 包头字段及对应元数据获取方式

包头	对应操作/服务类型	包头字段	包头字段元数据来源
BTH（Base Transport Header）	所有服务类型的所有操作	Operation Code	根据元数据传递的操作码和消息长度计算得到。
		Solicited Event	置 1
		MigReq	置 0
		Pad Count	根据消息长度计算得到，用于指示数据包的最后一个 4 字节中的无效数据长度
		Transport Header Version	置 0
		Partition Key	元数据传递
		Destination QP	元数据传递
		FECN	置 0
		BECN	置 0
		Reserve	置 0
		ACK Request	置 1，该位表征是否要求 Responder 对当前数据包发送 ACK 确认包，目前的实现中每个数据包都需要显式确认。
		Reserve	置 0
		Packet Sequence Number	包序列号，从本地寄存器中获取。
Datagram Extended Transport Header（DETH）	UD 服务类型添加该包头	Queue Key	元数据传递
		Reserved	置 0
		Dest QP	元数据传递
RDMA Extended Transport Header	RDMA Read/Write 添加该包头	Virtual Address	元数据传递
		Remote Key	元数据传递
		DMA Length	元数据传递
Atomic Extended Transport Header	原子操作添加该包头	Virtual Address	元数据传递
		Remote Key	元数据传递

		Swap/Add Data	元数据传递
		Compare Data	元数据传递

3.4.1.1.3 发送传输控制

RequesterTransControl 负责对每个数据包进行发送传输控制，其对于可靠性传输服务和不可靠性传输服务采用不同的控制策略。

相对于 RC 服务，UC/UD 不需要进行可靠性保证，且其不必相应丢包等错误，因此，传输控制非常简单。如果当前 QP 状态正常，且请求有效，就可以对数据进行分段，封装数据包，直接推送到网络中；而对于 RC 服务，在发送之前必须要进行各种发送条件的判定，还需要进行可靠性保证，处理较为复杂，本节重点对可靠性传输的发送部分设计进行阐述。

3.4.1.1.3.1 错误请求的处理

IB 协议规定错误请求的处理也必须保序，当发送传输控制模块检测到当前请求出错时，不能直接为其生成完成事件。这是因为在错误请求之前可能已经提交了多个请求，必须确保之前提交的请求全部处理完成后才能够处理错误请求。因此，当发送传输控制检测到错误请求时，将该请求暂存在 Bad Request 缓冲区中，等待其之前的请求全部处理完成后再提交完成事件。

3.4.1.1.3.2 可靠性传输

在处理一个正常的发送请求时（且 QP 状态正常），需要进行以下判定：

1.是否有足够的发送缓冲区可用；2.如果请求设定了 Fence 位，前置 RDMA Read/Atomics 请求是否已经处理完成。当上述条件同时成立，当前请求才能正常发送。

本节结合重传策略和发送缓冲区划分的方式，对发送判定条件的处理进行阐述，同时，给出请求方在 ACK/NAK 协议中的操作。

3.4.1.1.3.2.1 重传策略

对于 RC 服务，在检测到丢包时，需要对数据包进行重传。重传的方式可以按照两个不同的维度进行划分：数据包是否缓存在网卡上、重传整个消息或单个数据包。因此，重传机制的实现可以有四种方式，如下图所示。

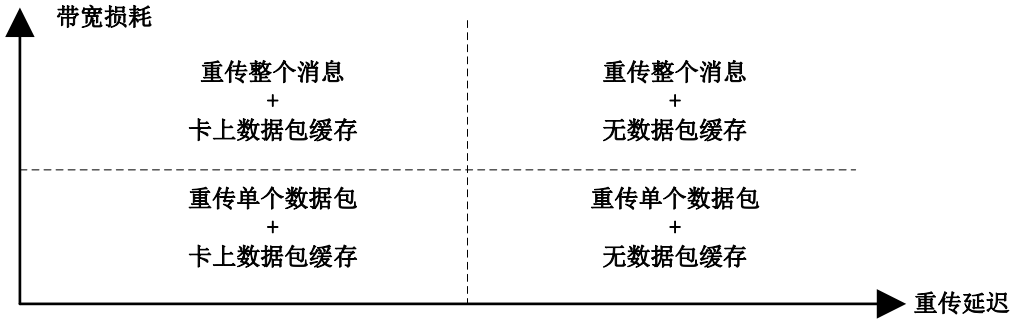


图 3.2 数据包重传方式划分

1.重传整个消息+无数据包缓存：当发送方检测到丢包后，将丢失的数据包对应的整个消息，以及该消息之后发送的所有消息都重传。重传的数据根据卡上缓存的描述符信息从内存中重新获取。重传整个消息的方式也称为 Go-Back-0。由于多了一次 PCIe Read RTT 的时间，这种方式的重传延迟开销较大，同时，由于每次重传都至少需要重传一个完整的消息，对带宽的损耗较高。

2.重传整个消息+卡上数据包缓存：与 1 不同的地方在于，卡上要缓存整个消息，这在大消息通信时，如果不借助片外存储几乎是不可实现的，目前没有这种方式的具体实现。

3.重传单个数据包+无数据包缓存：每次重传时，只从丢失的数据包开始重传，这种重传策略也称为 Go-Back-N。带宽损耗小，但多了 PCIe 的延迟。

4.重传单个数据包+卡上数据包缓存：相对于其它几种方式重传开销最小，但这种方式有几个弊端。首先，其实现机制比较复杂，需要对缓冲区进行动态管理；其次，增加了一部分卡上的缓存开销；同时，在网络规模较大的情况下，如果响应方 ACK 延迟太高，会导致发送方缓冲区长时间不能释放，阻塞后续消息的发送。

不同的重传策略也会影响 ACK/NAK 的具体实现方式。对于 Go-Back-0 而言，发送方只在每个消息的最后一个数据包上将 AckReq 置 1，要求响应方对整个消息进行确认，这种方式对接收带宽的损耗较小；对于 Go-Back-N 而言，发送方要求在每个数据包上都将 AckReq 置 1，要求响应方对每个数据包进行确认。

在目前的实现中，我们采用的是第四种实现方式，请求方对缓冲区进行动态管理，同时，要求每个发出的请求包都必须有对应的 ACK 响应包。

3.4.1.1.3.2.2 缓冲区的划分

发送缓冲区划分为以下几部分：

1.Send/RDMA Write 数据包缓冲区，存放 Send 和 RDMA Write 请求的数据包，包括包头元数据和数据负载；

2.RDMA Read 数据包缓冲区，存放 RDMA Read 请求的包头元数据；

3.RDMA Read 散列表缓冲区，存放 RDMA Read 请求的 Scatter Entry。

每个缓冲区都使用 Block RAM 实现，缓冲区的具体组织形式将在 MultiQueue 模块中给

出详细设计。

3.4.1.1.3.2.3 发送判定

缓冲区判定：如果是 Send/RDMA Write 请求，在消息判定阶段（RC_JUDGE_MSG）不进行缓冲区大小的判定，缓冲区判定留在数据分段阶段进行判定；如果是 RDMA Read 请求，在消息判定阶段需要检查 RDMA Read 数据包缓冲区是否有空余空间，同时，还需要检查 RDMA Read 散列表缓冲区是否有足够的空间来存放 Read 请求的 Scatter Entry。

Fence 位判定：在缓冲区划分时，我们将 RDMA Read 请求的缓冲区和其它缓冲区分离，主要原因是便于在判定 Fence 位时，直接检查 RDMA Read 请求缓冲区是否为空即可。

3.4.1.1.3.2.4 PSN 的处理

对于每个 RC 服务的数据包，需要根据其操作类型来确定其包头中填入的 PSN 值，规则如下：

1.对于 RDMA Read 之外的所有数据包，将当前的 Next PSN 填入 BTH，并将 Next PSN 加 1；

2.对于 RDMA Read 请求，则根据当前请求的数据长度和 PMTU 的大小计算出 RDMA Read Response 的个数为 n，将 Next PSN 填入包头，并将 Next PSN 更新为（Next PSN + n）。

3.4.1.1.3.2.5 激活定时器

在极端情况下，如果网络中的数据包都丢失了，必须依赖定时器机制来检测丢包的发生。实际实现时，请求传输控制模块每次发送数据包时都会发出激活命令，具体命令的执行由定时器控制模块负责。定时器设计见。

3.4.1.2 模块接口

表 3.2 RequesterTransControl 模块接口

模块名	RequesterTransControl			
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_atomics_from_dp_empty	输入	1	DataPack	原子操作元数据接口
o_atomics_from_dp_rd_en	输出	1		
iv_atomics_from_dp_data	输入	128		
i_raddr_from_dp_empty	输入	1		远程地址元数据接口
o_raddr_from_dp_rd_en	输出	1		
iv_raddr_from_dp_data	输入	128		

iv_entry_from_dp_data	输入	128		Scatter Entry 接口
i_entry_from_dp_empty	输入	1		
o_entry_from_dp_rd_en	输出	1		
i_md_from_dp_empty	输入	1		请求元数据接口
o_md_from_dp_rd_en	输出	1		
iv_md_from_dp_data	输入	256		
i_nd_from_dp_empty	输入	1		请求数据接口
o_nd_from_dp_rd_en	输出	1		
iv_nd_from_dp_data	输入	256		
o_cxtmgt_cmd_wr_en	输出	1	CxtMgt	上下文命令接口
i_cxtmgt_cmd_prog_full	输入	1		
ov_cxtmgt_cmd_data	输出	128		
i_cxtmgt_resp_empty	输入	1		上下文响应接口
o_cxtmgt_resp_rd_en	输出	1		
iv_cxtmgt_resp_data	输入	128		
i_br_prog_full	输入	1	RequesterRecvControl	出错请求传递接口
o_br_wr_en	输出	1		
ov_br_data	输出	64		
i_header_to_rpg_prog_full	输入	1	ReqPktGen	请求元数据接口
o_header_to_rpg_wr_en	输出	1		
ov_header_to_rpg_data	输出	336		
i_nd_to_rpg_prog_full	输入	1		网络数据传递接口
o_nd_to_rpg_wr_en	输出	1		
ov_nd_to_rpg_data	输出	256		
i_te_prog_full	输入	1	TimerControl	定时器事件接口
o_te_wr_en	输出	1		
ov_te_data	输出	63		
o_rpb_list_wea	输出	1	MultiQueue	Read Packet Buffer 元数据读写接口
ov_rpb_list_addra	输出	14		
ov_rpb_list_dina	输出	19		
iv_rpb_list_douta	输入	19		
o_rpb_element_wea	输出	1		Read Packet Buffer 数据读写接口
ov_rpb_element_addra	输出	9		
ov_rpb_element_dina	输出	265		

iv_rpb_element_douta	输入	265		
iv_rpb_free_data	输入	9		Read Packet Buffer 可用地址读接口
o_rpb_free_rd_en	输出	1		
i_rpb_free_empty	输入	1		
iv_rpb_free_data_count	输入	10		
ov_reb_list_addra	输出	14		Read Entry Buffer 元数据读写接口
ov_reb_list_dina	输出	29		
iv_reb_list_douta	输入	29		
o_reb_list_wea	输出	1		
ov_reb_element_addra	输出	14		Read Entry Buffer 数据读写接口
ov_reb_element_dina	输出	142		
iv_reb_element_douta	输入	142		
o_reb_element_wea	输出	1		
iv_reb_free_data	输入	14		Read Enetry Buffer 可用地址读接口
o_reb_free_rd_en	输出	1		
i_reb_free_empty	输入	1		
iv_reb_free_data_count	输入	15		
ov_swpb_list_addra	输出	14		Send/Write Packet Buffer 元数据读写接口
ov_swpb_list_dina	输出	25		
iv_swpb_list_douta	输入	25		
o_swpb_list_wea	输出	1		
ov_swpb_element_addra	输出	12		Send/Write Packet Buffer 数据读写接口
ov_swpb_element_dina	输出	268		
iv_swpb_element_douta	输入	268		
o_swpb_element_wea	输出	1		
iv_swpb_free_data	输入	12		Send/Write Packet Buffer 可用地址读接口
o_swpb_free_rd_en	输出	1		
i_wpb_free_empty	输入	1		
iv_swpb_free_data_count	输入	13		

3.4.1.3 状态机设计

由于跟传输层相关的状态机设计都较为复杂，因此，本节分别给出状态机的每个状态说明，状态转移图和状态转移说明。

3.4.1.3.1 状态说明

表 3.3 RequesterTransControl 状态说明

状态划分	状态说明
IDLE	初始状态
FETCH_CXT	读取上下文信息
RESP_CXT	等待上下文信息返回
FLUSH	将与上级模块对接的 FIFO 中的数据清空
RC_BAD_REQ	生成 Bad Request，将其写入与 RequesterRecvControl 对接的 FIFO 中
UCUD_CPL	生成为 UC/UD 服务请求生成完成事件
UCUD_SEG	对当前 UC/UD 请求进行分段，并为每段负载生成相应的元数据
UCUD_FWD	当前请求的负载长度不为 0，需要将负载传递给后续打包模块
RC_JUDGE_MSG	判定当前 RC 服务请求是否能够发送，判定条件已在前置小节说明
RC_SEG	将当前请求的负载按照 PMTU 大小进行分段，并为每段负载生成元数据
RC_FWD	将负载传递至下一级处理模块
RC_STORE_ENTRY	将 RDMA Read 的 Scatter Entry 写入缓冲区中
CXT_WB	将更新后的上下文信息写回

3.4.1.3.2 状态转移图

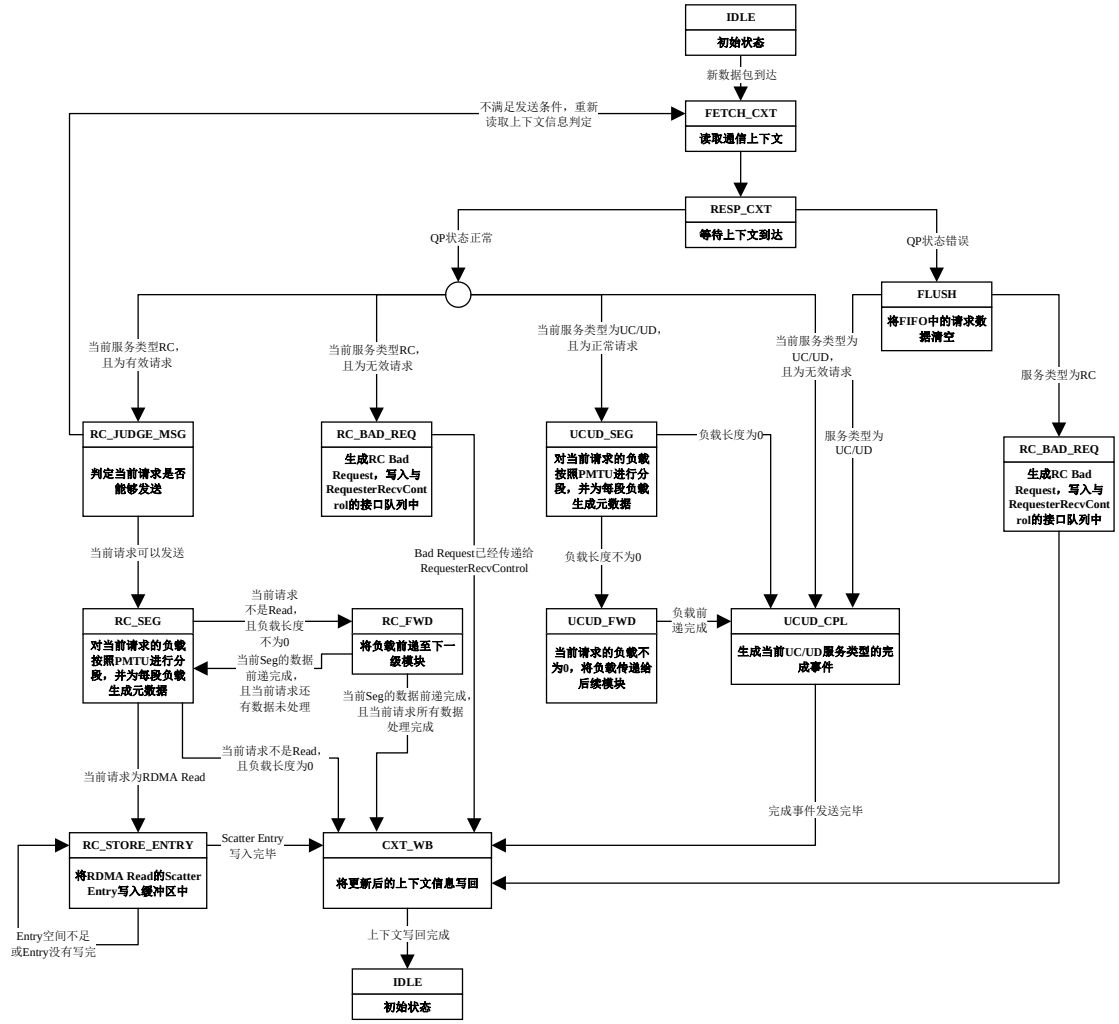


图 3.3 RequesterTransControl 状态转移图

3.4.1.3.3 状态转移说明

表 3.4 RequesterTransControl 状态转移说明

现态	次态	转移条件
RESP_CXT	FLUSH	上下文返回，QP 状态错误，将 FIFO 中的错误数据清除掉
	RC_JUDGE_MSG	上下文返回，QP 状态正常，服务类型为 RC，且为有效请求
	RC_BAD_REQ	上下文返回，QP 状态正常，服务类型为 RC，且为无效请求
	UCUD_SEG	上下文返回，QP 状态正常，服务类型为 UC/UD，且为有效请求
FLUSH	RC_BAD_REQ	服务类型为 RC，错误数据已清空

	UCUD_CPL	服务类型为 UC/UD，错误类型已清空
RC_JUDGE_MSG	FETCH_CXT	发送条件判定失败，重新读取状态信息进行判定
	RC_SEG	发送条件判定成功，请求可以发送。进入数据分段。
RC_SEG	RC_FWD	当前请求不是 Read，且负载长度不为 0
	RC_STORE_ENTRY	当前请求是 Read
	CXT_WB	当前请求不是 Read，且负载长度为 0
RC_FWD	RC_SEG	当前 Seg 前递完成，且当前请求还有数据未处理
	CXT_WB	当前 Seg 前递完成，且当前请求所有数据处理完成
RC_STORE_ENTRY	CXT_WB	Scatter Entry 全部写入缓冲区
	RC_STORE_ENTRY	Entry 缓冲区空间不足或还有 Entry 没有处理

3.4.2 定时器模块（TimerControl）

3.4.2.1 模块功能

请求方和接收方之间的 ACK/NAK 协议能够协助双方检测到具体出错或丢失的数据包，但是在极端情况下，如果数据包全部丢失，仅凭 ACK/NAK 协议是无法感知到丢包的。因此，必须设置定时器来实现超时机制，保证请求方能够检测到丢包事件。

对于每个 QP，请求引擎在运行过程中需要维护两个定时器：丢包超时重传定时器（Loss Timer）和 RNR（Receiver-Not-ready）重传定时器（RNR Timer）。前者用于检测最早传输的且尚未被确认的数据包是否丢失，如果丢失，则进行重传；后者用于在收到 RNR NAK 后让 RDMA 引擎等待一段时间后重新发送请求。TimerControl 的功能就是维护这两个定时器，并在定时器超时后生成超时事件，通知请求接收传输控制模块（RequesterRecvControl）进行相应事件的处理。

3.4.2.1.1 定时器的事件接口

外部模块通过发送事件与接收事件与定时器控制模块进行交互。

发送给定时器的事件包括：1.激活，启动对应 QP 的定时器；2.关闭，将对应 QP 的定时器关闭；3.重置，将定时器复位至初值，重新开始计时。

定时器发出的事件包括：1.定时器超时。

发送事件和接收事件格式和说明见 7.1.4。

3.4.2.2 模块架构

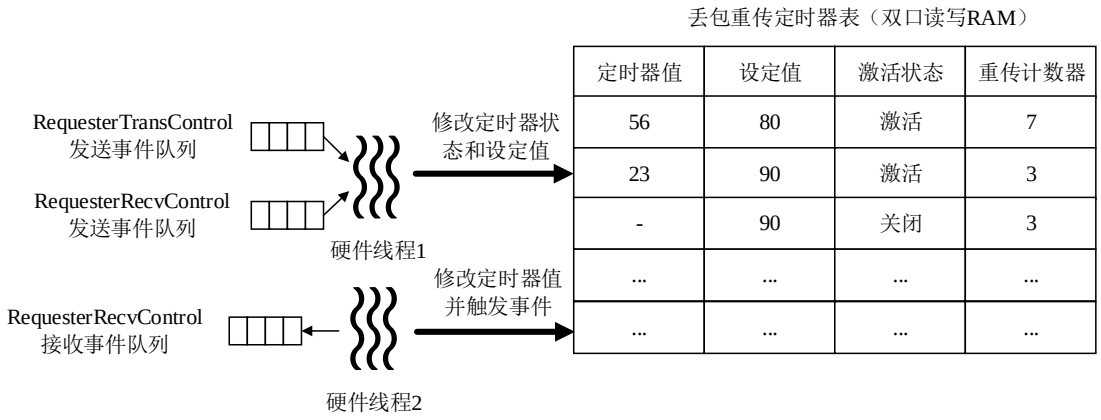


图 3.4 定时器原理图

实际实现时，Loss Timer 和 RNR Timer 均设定了个双端口 RAM，每个 RAM 都有对应的两个硬件线程去接收外部模块的事件以及发送超时事件。双端口 RAM 的配置信息如下表所示。两个双端口 RAM 采用完全一致的配置。

表 3.5 Loss Timer 双端口 RAM 格式

索引号	表项内容（23-bit）				
	当前值（22~15）	设定值（14~7）	激活状态（6）	重传次数（5~3）	重传阈值（2~0）
0	56	80	1	2	5
… …	… …	… …	… …	… …	
16383	79	90	1	3	7

表 3.6 RNR Timer 双端口 RAM 格式

索引号	表项内容（23-bit）				
	当前值（22~15）	设定值（14~7）	激活状态（6）	重传次数（5~3）	重传阈值（2~0）
0	56	80	1	2	5
… …	… …	… …	… …	… …	
16383	79	90	1	3	7

3.4.2.3 模块接口

表 3.7 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
i_tc_te_wr_en	输入	1	RequesterTransControl	RequesterTransControl 定时器设置接口
iv_tc_te_data	输入	64		
o_tc_te_prog_full	输出	1		
i_rc_te_loss_wr_en	输入	1	RequesterRecvControl	RequesterRecvControl 丢包定时器设置接口
iv_rc_te_loss_data	输入	64		
o_rc_te_loss_prog_full	输出	1		
i_rc_te_rnr_wr_en	输入	1		RequesterRecvControl 的 RNR 定时器设置接口
iv_rc_te_rnr_data	输入	64		
o_rc_te_rnr_prog_full	输出	1		
o_loss_expire_empty	输出	1		丢包定时器超时事件接口
ov_loss_expire_data	输出	31		
i_loss_expire_rd_en	输入	1		
o_rnr_expire_empty	输出	1		RNR 定时器超时事件接口
ov_rnr_expire_data	输出	31		
i_rnr_expire_rd_en	输入	1		

3.4.2.4 状态机设计

定时器不需要用状态机实现，本节给出定时器设计的原理和实现过程中的一些注意事项。

3.4.2.4.1 工作原理

线程 1 从地址 0 开始迭代访问每个定时器，该线程的功能是计时和触发超时事件。当访问到定时器表中的某个表项时，首先判定当前定时器是否处于激活状态，如果不是激活状态，则处理下一个定时器；如果是激活状态，则判定当前重传计数器的值是否归零，如果归零，则生成超过重传次数事件给 RequesterRecvControl 模块修改 QP 状态；如果未归零，则将重传次数计数器减 1，触发超时重传事件，通知 RequesterRecvControl 重传最早发出而未被确认的数据包。

线程 2 接收 RequesterTransControl 和 RequesterRecvControl 的请求事件，对定时器的状态和数值进行更新。RequesterTransControl 的事件是激活定时器，并设定定时器的阈值；RequesterRecvControl 的请求是重置定时器的值或关闭定时器。对两个模块的事件调度没有优先级的要求，实现时采用了轮询调度的方式。

实际上，RequesterTransControl 并不需要在每次发送数据包时都激活定时器，只需要在发送时检测一下当前发送缓冲区，如果发送缓冲区为空，将将定时器激活；如果非空，代表定时器已经处于激活状态，可以不做任何处理。

3.4.2.4.2 定时器的粒度

线程 2 修改重传定时器条目的时序如下图所示：

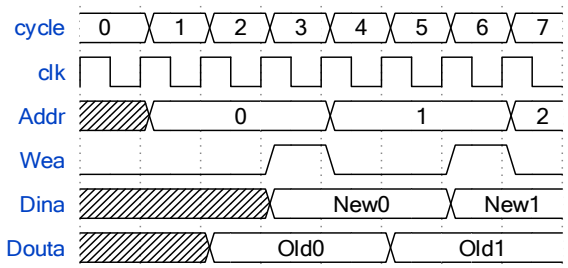


图 3.5 定时器更新时序图 1

在 c1（第 1 个时钟周期），线程 2 给出定时器地址，在 c2，输出定时器值，在 c3，将更新的定时器值写回。每个定时器的更新需要 3 个时钟周期完成。

对于 16384 个 QP，若每个定时器的更新需要 3 个时钟周期，则每个定时器的计时粒度 16384 * 3 * 时钟周期，假设使用 500MHz 的时钟频率，则计时粒度为 96us。而 IB 协议规定的超时重传粒度最小为 4.096μs，最大为 $4.096 * 2^{31}$μs，即 8388s。96us 处于可接受范围，但还是应该尽可能降低计时器粒度。

上述计时器设定方法的粒度太大主要在于需要轮询 16384 个表项，可以通过拆分计时器表的方法进行轮询，例如拆成 4 个表，计时器粒度就变成 24us。但拆分计时器的问题在于同一时刻可能出现 4 个超时事件，此时需要对 4 个超时事件进行仲裁，然后写入与 RecvControl 的事件队列中，相比于 1 个表，增加了 3 个额外的时钟周期。

第二种方法可以将定时器的更新流水起来，观察到第 2 个时钟周期并不需要给出地址，因此，在第 2 个时钟周期我们将地址设为下一个定时器的地址，这样就可以在 4 个时钟周期内完成两个定时器的更新。这种方式下的时序图如下：

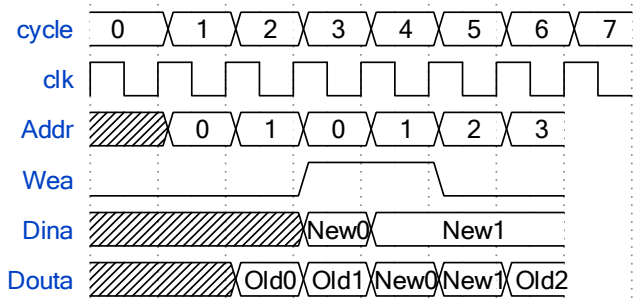


图 3.6 定时器更新时序图 2

这种方式下，计时器的粒度由 96μs 变为 64μs。如果要满足 4.096μs 的最小粒度，还需要将定时器表拆分成 16 个。再由线程 3 对每个定时器表的事件进行调度。

目前实现中定时器的最小粒度先按照 64μs 进行设定。

3.4.3 多队列模块（MultiQueue）

3.4.3.1 模块功能

多队列模块用于暂存已经发出但尚未被确认的网络数据包（Outstanding Packets），其本身并不实现任何处理功能，仅仅是多个 BRAM 和 FIFO 的集合。RequesterTransControl 在发送请求时将数据包暂存在缓冲区中，RequesterRecvControl 在接收到响应数据包后，根据响应包类型释放发送缓冲区或从缓冲区中读取数据进行重传。

3.4.3.2 模块架构

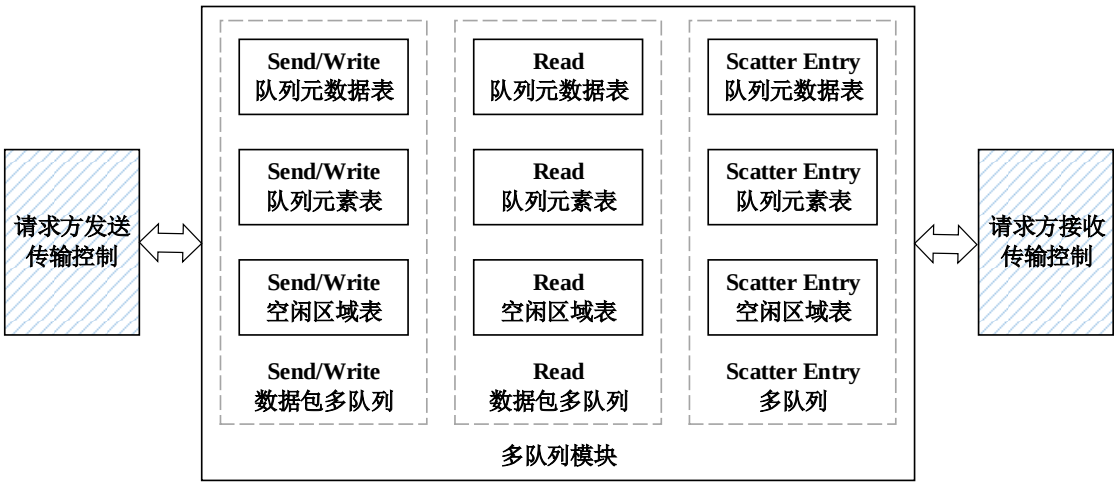


图 3.7 多队列模块示意图

多队列模块中包含三组队列，分别是 Send/Write 数据包多队列、Read 数据包多队列以及 Scatter Entry 多队列，划分成三组多队列的原因已经在前置章节阐述。

每组多队列实际上包含三组缓冲区：元数据表、元素表以及空闲区与表，其中队列元数据表记录了每个 QP 所使用的队列的头（Head）/尾（Tail）指针，以及当前队列是否为空（Empty）。之所以存放一个空标志位，是因为在 Head 等于 Tail 时，队列可能为空或者只有一个元素，因此，需要该标志位来指示当前队列的状态；元素表用于存放具体的数据包；空闲区域表用于存放元素表中的空闲数据块地址。下图给出了三者的对应关系。

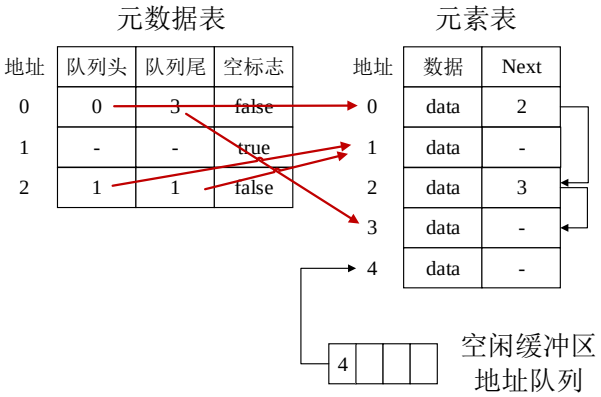


图 3.8 多队列机制

在初始化时，ListTable 中的所有 Head 和 Tail 被置 0，Empty 标记位被置为 1。FreeList

被写入从 0 到 n 的地址项（n 为 ElementTable 中的表项数量）。

每个缓冲区的表项格式和使用的 BRAM 大小如下表所示。

表 3.8 RPB List Table 表格式

索引号	表项内容（19-bit）		
	18	17~9	8~0
0	Empty	Tail	Head
1	Empty	Tail	Head
… …	… …	… …	… …
16383	Empty	Tail	Head

表 3.9 RPB Element Table 表格式

索引号	表项内容（265-bit）	
	264~256	255~0
0	Next Addr	RDMA Read Packet
1	Next Addr	RDMA Read Packet
… …	… …	… …
511	Next Addr	RDMA Read Packet

表 3.10 RPB Free FIFO 格式

模块名	RPBFreeFIFO		
模块说明	IP Core，由 Vivado 生成；深度：512；宽度：9		
接口	输入/输出	位宽	对接模块
clk	输入	1	/
srst	输入	1	/
din	输入	9	RequesterTransControl
wr_en	输入	1	
prog_full	输出	1	
data_count	输出	10	
rd_en	输入	1	RequesterRecvControl
empty	输出	1	
dout	输出	9	

表 3.11 REB List Table 模块接口

索引号	表项内容（29-bit）		
	28	27~13	13~0

0	Empty	Tail	Head
1	Empty	Tail	Head
… …	… …	… …	… …
16383	Empty	Tail	Head

表 3.12 REB Element Table 表格式

索引号	表项内容（142-bit）	
	141~128	**127~0**
0	Next Addr	Scatter Entry
1	Next Addr	Scatter Entry
… …	… …	… …
16383	Next Addr	Scatter Entry

表 3.13 REB Free FIFO 表格式

模块名	REBFreeFIFO		
模块说明	IP Core，由 Vivado 生成；深度：16384；宽度：14		
接口	**输入/输出**	**位宽**	**对接模块**
clk	输入	1	/
srst	输入	1	/
din	输入	14	RequesterTransControl
wr_en	输入	1	
prog_full	输出	1	
data_count	输出	15	
rd_en	输入	1	RequesterRecvControl
empty	输出	1	
dout	输出	14	

表 3.14 SWPB List Table 表格式

索引号	表项内容（25-bit）		
	24	**23~12**	**11~0**
0	Empty	Tail	Head
1	Empty	Tail	Head
… …	… …	… …	… …
16383	Empty	Tail	Head

表 3.15 SWPB Element Table 表格式

索引号	表项内容（268-bit）	
	267~256	255~0
0	Next Addr	Send/RDMA Write Packet（Header 或 Payload）
1	Next Addr	Send/RDMA Write Packet
... …	… …	… …
4095	Next Addr	Send/RDMA Write Packet

表 3.16 SWPB Free FIFO 格式

模块名	SWPBFreeFIFO		
模块说明	IP Core，由 Vivado 生成；深度：4096；宽度：12		
接口	输入/输出	位宽	对接模块
clk	输入	1	/
srst	输入	1	/
din	输入	14	RequesterTransControl
wr_en	输入	1	
prog_full	输出	1	
data_count	输出	15	
rd_en	输入	1	RequesterRecvControl
empty	输出	1	
dout	输出	14	

表 3.17 SWPB Free FIFO 模块接口

3.4.3.3 模块接口

表 3.18 MultiQueue 模块接口

模块名	MultiQueue			
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_rpb_list_wea	输入	1	RequesterTransControl	Read Packet Buffer 元数据接口
iv_rpb_list_addra	输入	14		
iv_rpb_list_dina	输入	19		
ov_rpb_list_douta	输出	19		
i_rpb_element_wea	输入	1		Read Packet Buffer 数据

iv_rpb_element_addra	输入	9		接口
iv_rpb_element_dina	输入	265		
ov_rpb_element_douta	输出	265		
i_reb_list_wea	输入	1		Read EntryBuffer 元数据接口
iv_reb_list_addra	输入	14		
iv_reb_list_dina	输入	29		
ov_reb_list_douta	输出	29		
i_reb_element_wea	输入	1		Read Entry Buffer 数据接口
iv_reb_element_addra	输入	14		
iv_reb_element_dina	输入	142		
ov_reb_element_douta	输出	142		
i_swpb_list_wea	输入	1		Send/Write Packet Buffer 元数据接口
iv_swpb_list_addra	输入	14		
iv_swpb_list_dina	输入	25		
ov_swpb_list_douta	输出	25		
i_swpb_element_wea	输入	1		Send/Write Packet Buffer 数据接口
iv_swpb_element_addra	输入	12		
iv_swpb_element_dina	输入	268		
ov_swpb_element_douta	输出	268		
iv_rpb_free_din	输入	9		RDMA Packet Buffer 可用地址接口
i_rpb_free_wr_en	输入	1		
o_rpb_free_prog_full	输出	1		
ov_rpb_free_data_count	输出	10		
iv_reb_free_din	输入	14		Read Entry Buffer 可用地址接口
i_reb_free_wr_en	输入	1		
o_reb_free_prog_full	输出	1		
ov_reb_free_data_count	输出	15		
iv_swpb_free_din	输入	12		Send/Write Packet Buffer 可用地址接口
i_swpb_free_wr_en	输入	1		
o_swpb_free_prog_full	输出	1		
ov_swpb_free_data_count	输出	13		
i_rpb_list_web	输入	1	RequesterRecvControl	Read Packet Buffer 元数据接口
iv_rpb_list_addrb	输入	14		
iv_rpb_list_dinb	输入	19		

ov_rpb_list_doutb	输出	19		
i_rpb_element_web	输入	1		Read Packet Buffer 数据接口
iv_rpb_element_addrb	输入	9		
iv_rpb_element_dinb	输入	265		
ov_rpb_element_doutb	输出	265		
i_reb_list_web	输入	1		Read EntryBuffer 元数据接口
iv_reb_list_addrb	输入	14		
iv_reb_list_dinb	输入	29		
ov_reb_list_doutb	输出	29		
i_reb_element_web	输入	1		Read Entry Buffer 数据接口
iv_reb_element_addra	输入	14		
iv_reb_element_dinb	输入	142		
ov_reb_element_doutb	输出	142		
i_swpb_list_web	输入	1		Send/Write Packet Buffer 元数据接口
iv_swpb_list_addrb	输入	14		
iv_swpb_list_dinb	输入	25		
ov_swpb_list_doutb	输出	25		
i_swpb_element_web	输入	1		Send/Write Packet Buffer 数据接口
iv_swpb_element_addrb	输入	12		
iv_swpb_element_dinb	输入	268		
ov_swpb_element_doutb	输出	268		
o_rpb_free_empty	输出	1		RDMA Packet Buffer 可用地址接口
ov_rpb_free_data	输出	9		
i_rpb_free_rd_en	输入	1		
o_reb_free_empty	输出	1		Read Entry Buffer 可用地址接口
ov_reb_free_data	输出	14		
i_reb_free_rd_en	输入	1		
o_swpb_free_empty	输出	1		Send/Write Packet Buffer 可用地址接口
ov_swpb_free_data	输出	12		
i_swpb_free_rd_en	输入	1		

3.4.3.4 实现相关

3.4.3.4.1 队列元素的插入与删除

多队列模块本身负责对队列的操作，仅仅提供基本的读写接口，对队列元素的插入与释

放由 RequesterTransControl 和 RequesterRecvControl 模块控制。

当 RequesterTransControl 模块需要发送一个数据包时，根据数据包的大小和 FreeList 中的可用地址数量（由 FIFO 的 data_count 计算得到）判定是否有足够的缓冲区来缓存当前数据包。插入数据的处理流程如下图所示。

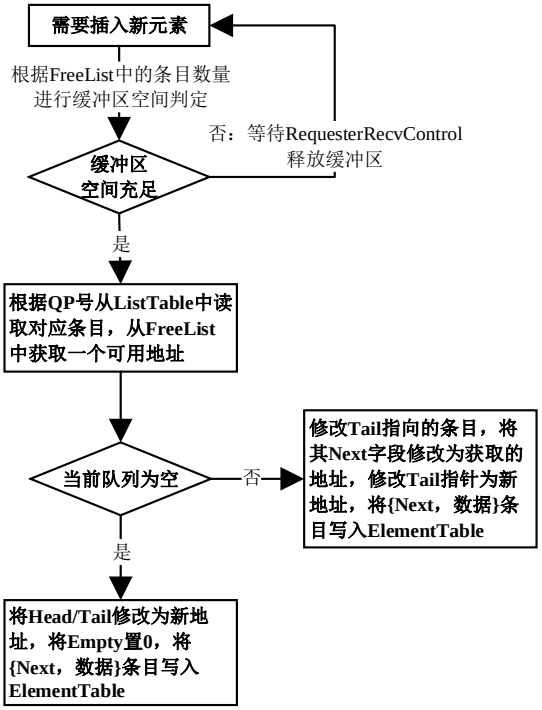


图 3.9 多队列元素插入操作流程图

当 RequesterRecvControl 接收到一个响应包时，需要根据响应包的 PSN 释放缓冲区。释放数据的处理流程如下图所示。

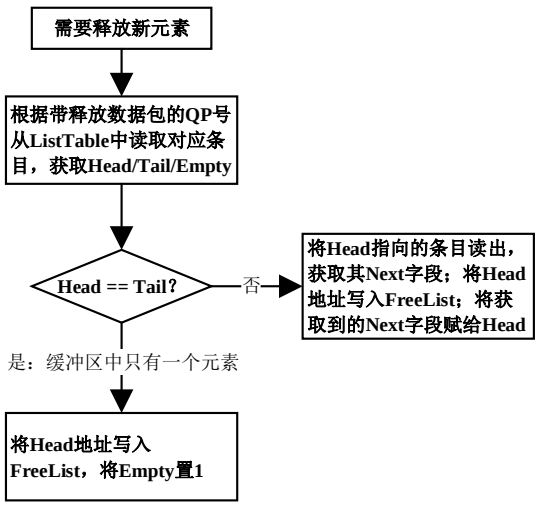


图 3.10 多队列元素删除操作流程图

3.4.3.4.2 时序图

3.4.3.4.2.1 插入元素时序图

RequesterTransControl 插入元素的时序图如下图所示。

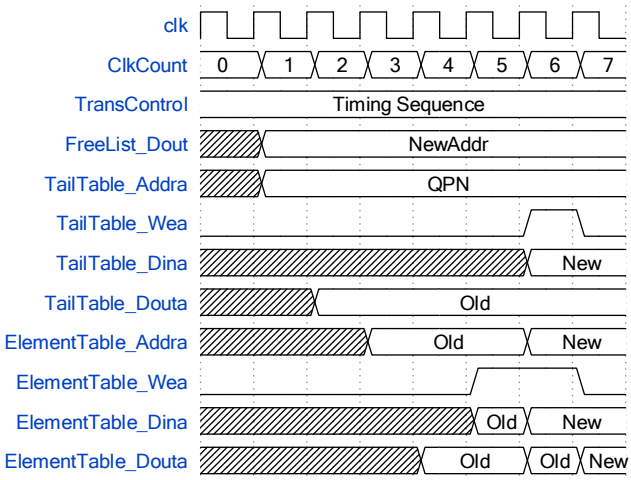


图 3.11 RequesterControl 插入元素的时序图

在第 1 个时钟周期，FreeList_Dout 给出了新的可用条目地址。TailTable_Addra 为 QPN，索引对应条目。在第 2 个时钟周期，ListTable_Dout 变化为旧的 Tail，同时，将 Tail 赋给 ElementTable_Addra 的寄存器。在第 3 个时钟周期，ElementTable_Addra 变化为旧的 Tail。在第 4 个时钟周期，我们从 ElementTable_Douta 得到了旧的 Tail 元素。在第 5 个时钟周期，我们更新旧的 Element 的 Next 字段指向新的元素（即 FreeList_Dout 的输出），将其写入 ElementTble。在第 6 个时钟周期，将新的 Element 写入 ElementTable。同时，在第 6 个时钟周期，将更新后的 ListTable 的 Tail 指针写回。需要说明的是，对于 ElementTable_Douta 而言，图示在第 4，5 个时钟周期输出的 Old 和第 6 个时钟周期输出的 Old 并不相同。第 6 个时钟周期输出的 Old 元素中的 Next 字段已经被修改过了。在上述示例中，省略了对 HeadTable 的访问，实际上，需要读出 Head 和 Empty 来判断是否需要更新 Head 和 Empty 位，其更新过程与 TailTable 处于相同的时钟周期，不再赘述。

对于每个消息，只有在第一个数据包时需要给出 QPN 并获取 Tail 和旧的 Tail 元素（即 1，2，3，4 时钟周期），后续数据包的 Tail 更新，ElementTable 更新只需要两个时钟周期（即 5，6 时钟周期）。因为在插入下一个元素时，旧的 Element 就是我们在第 6 个时钟周期写回的元素，可以直接拿到。

从上述分析可以看出，由于在插入新元素时，既要将原本队尾元素的 Next 字段更新后重新写入，又要将新的条目加入，因此，每次插入一个新元素时需要两个时钟周期，这对于连续的数据流传输相当于吞吐量下降了一半，这显然是不可接受的，解决该问题可以采用以下方案：

之所以要更新队尾元素的 Next 字段是因为队尾元素在被缓存时不知道下一个缓存的地址。在实际实现时，在处理数据包的第一个 256 比特（即数据包包头）时，从 FreeList 队列中读取两个地址，记为 A1，A2。将数据缓存到 A1 地址，同时将数据的 Next 字段记为 A2。在处理下一个 256 比特时，从 FreeList 队列中读取地址 A3，将数据缓存到 A2，并将数据的

Next 字段标记为 A3.依次类推，后续数据的处理依次类推。这种方式的时序图如下所示。

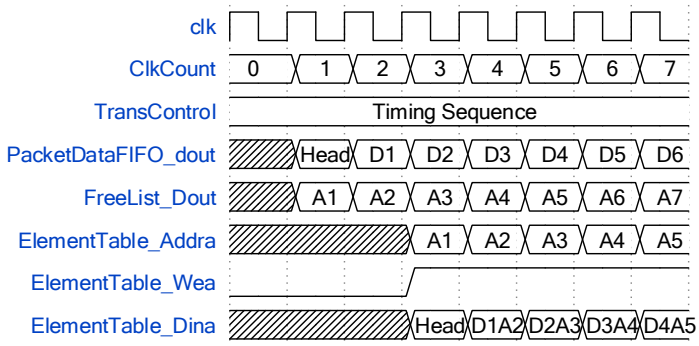


图 3.12 RequesterContro 预取缓存地址时序图

3.4.3.4.2.2 释放元素时序图

RequesterRecvControl 释放队列元素的时序图如下图所示。

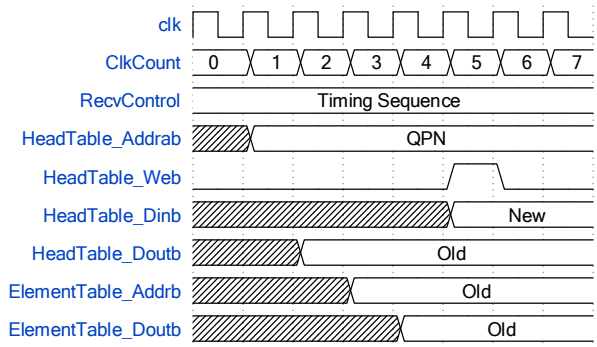


图 3.13 RequesterControl 插入元素的时序图

在第 1 个时钟周期给出 QPN，索引 HeadTable。第 2 个时钟周期 HeadTable 输出 Head 的值，第 3 个时钟周期给出 ElementTable_Addrb，第 4 个时钟周期 ElementTable 输出 Old 元素，第 5 个时钟周期将 Old 元素的 Next 字段赋给 Head 指针并写回 ListTable 中。

在上述示意中，省略了对 TailTable 的访问。实际上，需要读取 Tail 指针来判断当前是否在处理最后一个元素。

3.4.3.4.3 解决“写写”冲突

假设 RequesterTransControl 和 RequesterRecvControl 在处理同一个 QP，即两者需要访问 ListTable 中的相同条目。在某个时钟周期，TransControl 和 RecvControl 都需要对 LIstTable 中的条目进行更新，例如，RequesterTransControl 需要移动 Head，RequesterRecvControl 需要移动 Tail。此时，如果 ListTable 中的各个字段物理上存储在一个缓冲区中，则 RequesterTransTransControl 写入的条目为<更新后的 Head，旧的 Tail，旧的 Empty>，RequesterRecvControl 写入的条目为<旧的 Head，新的 Tail，新的 Empty>。此时会造成对同一个地址数据写入的冲突。

为了避免“写写冲突”，实现中将 Head，Tail，Empty 字段分开存储。在大部分情况下，

TransControl 只会访问 Tail 字段，RecvControl 只会访问 Head 字段和 Empty 字段。但有一种情况比较特殊，就是在 TransControl 加入第一个元素的时候，TransControl 需要去移动 Head 指针指向第一个元素，同时需要更新 Empty 位。此时，由于是在处理第一个元素，RecvControl 不可能去写该条目，因此，这种“写写冲突”也不存在。

3.4.3.4.4 解决“读写”冲突

正常情况下，RequesterTransControl 和 RequesterRecvControl 不会出现“读写冲突”，因为 TransControl 访问的是 Tail 指针以及 Tail 指针指向的 ElementTable 中的元素，而 RecvControl 访问的是 Head 指针以及 Head 指针指向的 ElementTable 中的元素，二者并没有交集。但是，在一种特殊情况下，即队列中只剩下一个元素时，两者的数据访问会出现交集，从而产生“读写冲突”。

在队列中只剩下一个元素时需要进行特殊处理，因为此时 RequesterTransControl 和 RequesterRecvControl 可能会有共享变量的交互。假设当前队列中只剩下一个元素，RequesterTransControl 需要插入新元素，而 RequesterRecvControl 需要释放当前元素，假设两者均从时钟周期 1 开始执行。在第 2 个时钟周期，RequesterRecvControl 读出了 Tail 和 Head 以及 Empty，判定认为当前 Head 等于 Tail 且 Empty 不为 1，就会在第 5 个时钟周期将 Empty 置为 1。但实际上，在这个过程中，RequesterTransControl 会插入新的元素，由于 RequesterRecvControl 感知不到 Tail 的更新，在两个操作序列执行完成后，队列中仍然有一个元素，此时就会造成 Empty 标志位与队列实际状态不符，导致下次 RequesterTransControl 在插入新元素时，判定队列为空，会去修改 Head 指针的指向。

为了规避该问题，在实际实现时，RequesterTransControl 和 RequesterRecvControl 在获取到 Head，Tail 和 Empty 时，如果发现 Head 与 Tail 不相等，则直接忽略掉 Empty 位；只有在 Head 与 Tail 相等时，才会根据 Empty 位来判定当前是否为空队列。

3.4.3.4.5 缓冲区开销

下表给出了各个缓冲区的宽度、深度及大小（表格中不对 Head，Tail，Empty 做区分）。

表 3.19 MultiQueue 缓冲区开销统计表

缓冲区	宽度	深度	大小（Mb）
RPBListTable	19	16384	0.296
RPBElementTable	265	512	0.129
RPBFreeList	9	512	0.004
REBListTable	29	16384	0.453
REBElementTable	142	16384	2.218
REBFreeList	14	16384	0.218
SWPBListTable	19	16384	0.296

SWPBElementTable	265	512	0.129
SWPBFreeList	9	512	0.004
SWPDListTable	25	16384	0.390
SWPDElementTable	268	4096	1.046
SWPBFreeList	12	4096	0.046
合计	/	/	3.75

3.4.4 请求包生成（ReqPktGen）

3.4.4.1 模块功能

请求包生成模块从RequesterTransControl模块接收元数据，根据元数据生成对应的包头，并将包头和数据向 256 比特宽度的 FIFO 中对齐。

3.4.4.2 模块接口

表 3.20 ReqPktGen 模块接口

模块名	RequestPacketGen			
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_tc_header_empty	输入	1	RequesterTransControl	RequesterTransControl 传递数据包元数据
iv_tc_header_data	输入	336		
o_tc_header_rd_en	输出	1		
i_tc_nd_empty	输入	1		RequesterTransControl 传递数据包负载
iv_tc_nd_data	输入	256		
o_tc_nd_rd_en	输出	1		
i_rc_header_empty	输入	1	RequesterRecvControl	RequesterTransControl 传递数据包元数据
iv_rc_header_data	输入	336		
o_rc_header_rd_en	输出	1		
i_rc_nd_empty	输入	1		RequesterTransControl 传递数据包负载
iv_rc_nd_data	输入	256		
o_rc_nd_rd_en	输出	1		
i_trans_prog_full	输入	1	256To64BitTrans	向位宽转换模块传递数据，256 比特指示数据

o_trans_wr_en	输出	1		包开始，257 比特指示数据包结束，中间数据这两个比特均为 0
ov_trans_data	输出	258		

3.4.4.3 状态机设计

该模块状态机非常简单，不再赘述。

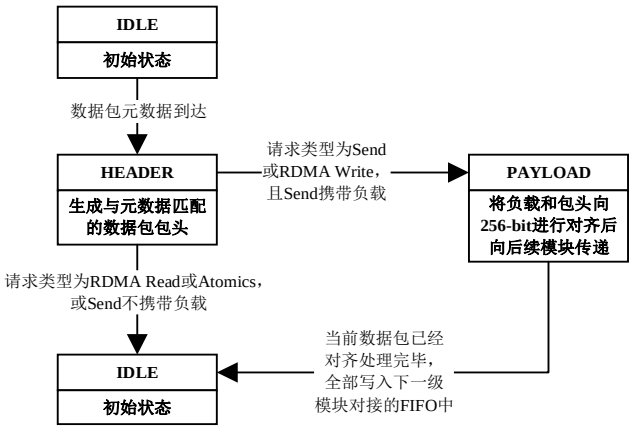


图 3.14 RequestPacketGen 状态转移图

3.4.5 请求方接收传输控制

3.4.5.1 模块功能

请求方接收传输控制模块是整个 RDMA 引擎中最为复杂的一个模块，其主要功能是对接收到的响应包进行检测，根据响应包中携带的 ACK 或 NAK 对数据缓冲区进行释放或对数据包进行重传，该模块还要将 RDMA Read Response 的数据写入内存中；另一方面，该模块接收定时器事件，根据事件类型进行重传或 QP 状态切换。本小节将详细介绍其处理机制。

3.4.5.1.1 说明

为方便后文讨的阐述，本小节首先对一些术语做出说明。

ACK 包指代 BTH 中的操作码为 ACK，且 AETH 中的 Syndrome 也为 ACK；NAK 包指代 BTH 中的操作码为 ACK，且 AETH 中的 Syndrome 为 NAK 相关错误；Read Response 指代 BTH 中的操作码为 RDMA Read Response First/Middle/Last/Only。

UPSN（UnAcked PSN）指的是当前 QP 对应缓冲区中最早发出的且未被确认的数据包的 PSN，RPSN（Received PSN）指的是当前处理的响应包中携带的 PSN，NPSN（Next PSN）指的是 RequesterTransControl 的下一个待发数据包的 PSN。

三者之间的关系如下图所示。

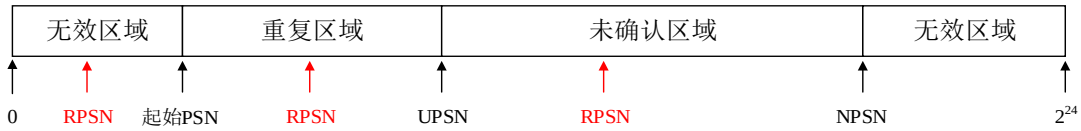


图 3.15 UPSN/RPSN/NPSN 相对位置示意图

其中，RPSN 可能落在无效区域、重复区域和未确认区域。对于前两者，响应包将被直接丢弃，不做任何处理，后文重点阐述 RPSN 落在未确认区域的情况。

3.4.5.1.2 ACK 包的处理

当收到一个 ACK 包时，有以下三种情况需要区分对待：

1.RPSN 小于 UPSN

此时说明收到了重复的数据包，直接将该数据包丢弃即可。

2.RPSN 等于 UPSN

此时说明收到的 ACK 恰好对应着最早发出而未被确认的数据包，将 UPSN 对应的数据包释放即可。

3.RPSN 大于 UPSN，且未被确认的请求中没有 RDMA Read 请求

此时说明序号小于等于 RPSN 的请求都已经被响应方处理完成，但是返回的 ACK 丢失了。这种情况下，由于我们不需要接收 RDMA Read 的响应数据，可以直接将 UPSN 至 RPSN 之间的所有数据包都释放。

4.RPSN 大于 UPSN，且未被确认的请求中有 RDMA Read 请求

此时说明序号小于等于 RPSN 的请求都已经被响应方处理完成，但是返回的 ACK 丢失了。这种情况下，由于有 RDMA Read Response 请求丢失，因此，我们需要重传 RDMA Read 请求以便响应方重新发送读响应数据。这种情况如下图所示。

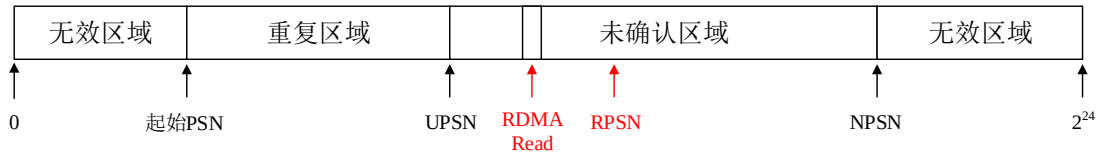


图 3.16 RDMA Read Response 丢失 PSN 示意图

记最早发出的未被确认的 RDMA Read 请求的 PSN 为 RdPSN，在 UPSN 和 RdPSN 之前的请求可以被确认，而 RdPSN 和 NPSN 之间的所有数据包则需要重传。

需要说明的是，在实际实现时，使用了 RPBListTable 中的元数据和标志位判定是否在 RPSN 之前还有 Read 请求。将 RDMA Read 和 Send/RDMA Write 的缓冲区分离实现就是为了简化对 RDMA Read 的判定。

3.4.5.1.3 NAK 包的处理

如果收到了 NAK 包，则说明响应方在处理请求时遇到了错误。NAK 的错误类型分为以下几类：

1.RNR（Resources Not Ready）：接收方没有足够的资源来处理当前请求，一般发生在接收方处理 Send 请求时找不到可用的描述符；

2.PSN Sequence Error：接收方收到的数据包 PSN 出现了错误，一般由发送路径上的丢包引起；

3.Invalid Request：接收方收到了非预期的请求，一般发生于接收方不支持数据包的操作码；

4.Remote Access Error：接收方执行请求时遇到了访问错误，一般发生于请求包的 R_Key 非法；

5.Remote Operational Error：接收方本身处理请求时出现了错误，一般发生于接收方的描述符指定的内存区域太小，不足以容纳请求方的数据。

对于这 5 种类型的 NAK，接收方在处理时不仅要考虑不同的包类型，还要考虑 RPSN 和 UPSN 之间的相对关系。我们对这 5 类不同的 NAK 包的处理给出详细说明。

3.4.5.1.3.1.1 NAK RNR 错误

如果收到了 RNR 包，则说明响应方的资源并没有准备好，此时需要等待一段时间后再开始重传，该错误属于可恢复性错误。

如果当前 UPSN 等于 RPSN，说明丢失的恰好是最早发出的未被确认的数据包，此时根据 RNR 数据包中的定时器值，发送设置定时器事件到定时器控制模块；

如果当前 UPSN 大于 RPSN，且缓冲区中尚有 Read 请求，则将 Read 请求之前的所有数据包释放，并重传 Read 请求到当前 UPSN 之间的所有数据包，并发送定时器事件；

如果当前 UPSN 大于 RPSN，且缓冲区中没有 Read 请求，则将 UPSN 之前的所有数据包释放，并发送定时器事件；

与 NAK PSN 处理过程不同的地方在于，在 UPSN 之前的请求释放后，并不是马上开启重传，而是必须等待 RNR 定时器到时后才能触发重传。

3.4.5.1.3.2 NAK PSN 错误/Remote Access 错误/Invalid Request 错误

这三类错误属于可恢复性错误，在响应方检查到该错误时，仅仅会返回一个 NAK 包，而不会切换自己的 QP 状态。返回的 NAK 包中携带的 PSN 是当前响应方的 Expected PSN。

在收到 NAK PSN 错误时，如果当前 UPSN 等于 RPSN，说明丢失的恰好是最早发出的未被确认的数据包，此时将 UPSN 到 NPSN 之间的所有数据包重传，同时发送计时器重置请求到定时器管理模块；

如果当前 UPSN 小于 RPSN，且缓冲区中尚有 Read 请求，则将 Read 请求之前的所有数据包释放，并重传 Read 请求到 NPSN 之间的所有数据包；

如果当前 UPSN 小于 RPSN，且缓冲区中没有 Read 请求，则将 UPSN 之前的所有数据包释放，并重传 UPSN 到 NPSN 之间的所有数据包。

需要说明的是，之所以在重传时要从 UPSN 重传到 NPSN，是因为响应方一旦检测到一

个数据包出现了这三类错误，其后续数据包会直接被全部丢弃。

3.4.5.1.3.3 NAK Remote Operational 错误

本类错误属于响应方自身的错误，与请求方无关，在遇到这种错误时，响应方除了会返回一个 NAK 之外，还会将自己的 QP 状态切换到 QP Error，因此，当请求方收到本类 NAK 之后，没有必要再进行重传，因为重传的请求也会被响应方直接丢弃。

如果 UPSN 等于 RPSN，则说明当前 NAK 之前的所有请求都已经执行完成，将缓冲区中 RPSN 之前的请求包全部释放，同时，将 RPSN 到 NPSN 之间的所有数据包清空，并为其生成完成事件，完成事件中填入“Flush in Error”的错误码；

如果 UPSN 小于 RPSN，且缓冲区中没有 Read 请求，则将 RPSN 之前的所有请求直接释放掉；同时，将 RPSN 到 NPSN 之间的所有数据包清空，并为其生成完成事件，完成事件中填入“Flush in Error”的错误码；

如果 UPSN 小于 RPSN，且缓冲区中还有 Read 请求，则将 UPSN 到 Read 请求之间的所有数据包释放掉；同时，将 RPSN 到 NPSN 之间的所有数据包清空，并为其生成完成事件，完成事件中填入“Flush in Error”的错误码；

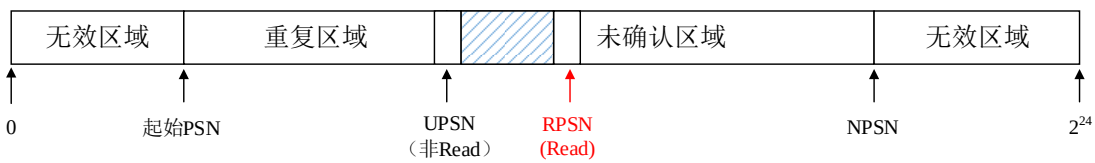
3.4.5.1.3.4 RDMA Read Response 的处理

RDMA 读响应的处理较为复杂，不仅要考虑对缓冲区的释放，数据包的重传，还要将响应数据写入内存，同时，RDMA Read Response First/Middle/Only/Last 的处理过程还存在着较大区别，本节给出 RDMA 读响应的详细设计。

3.4.5.1.3.4.1 请求释放与重传

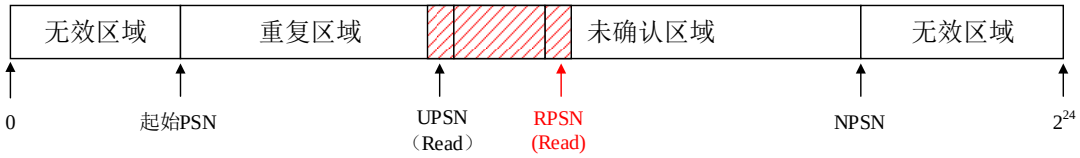
当 UPSN 与 RPSN 相等，说明当前 Read 请求之前的所有数据包都已经处理完成，此时可以直接将响应数据写入内存；

当 UPSN 小于 RPSN，且此时操作码为 First 或 Only，并且 UPSN 指示的请求不是 RDMA Read，此时不论缓冲区中还有没有待释放的 RDMA Read 请求，都需要先进行隐式数据包释放。将 UPSN 到最早发出的 RDMA Read 请求之前的数据包全部释放。所谓“最早发出的未被确认的 RDMA Read 请求”，有可能是当前待处理的 Read 请求，也有可能是之前未被确认的请求，下图给出了这种情况的示意。不管阴影区域可能有 Read 请求，此时首先都需要进行隐式释放。如果阴影区域中有 RDMA Read，说明此时有前置 RDMA Read 响应丢失，则需要从丢失的 RDMA Read 请求开始重传，一致重传到当前 RPSN 对应的请求。

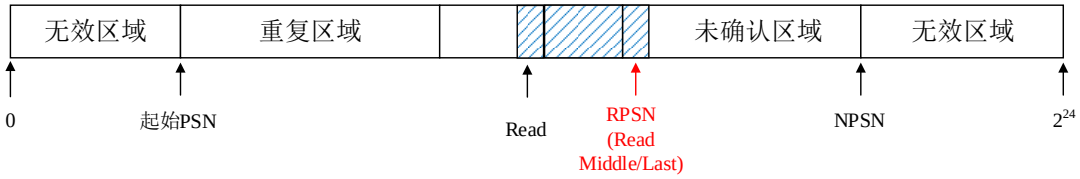


当 UPSN 小于 RPSN，且操作码为 First 或者 Only，并且 UPSN 指示的请求是 RDMA Read（情形 1），或者 UPSN 小于 RPSN，且操作码为 Middle 或 Last（情形 2），在这两种情

形下，均说明有前置请求丢失，需要进行请求重传。下图给出了情形 1 和情形 2 的示意。图中红色阴影为重传区域。



在情形 1 中，由于 UPSN 指向的请求是 RDMA Read，而 RPSN 大于 UPSN，且 RPSN 指向的 RDMA First/Only，说明在当前缓冲区中存在着两个及以上的 Read 请求，但此时只收到了后发出的 Read 请求，则前置 Read 请求必然丢失了。此时需要将最早发出的未被确认的 Read 请求到 RPSN 的所有请求重传。

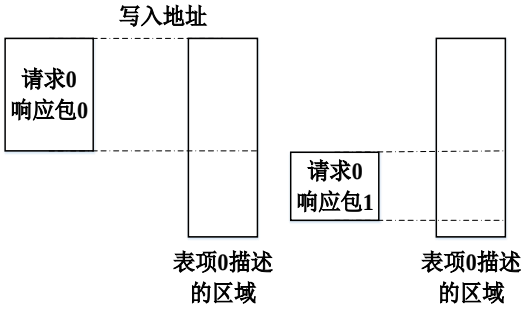


在情形 2 中，RPSN 指向的是 Middle/Last。当 RequesterRecvControl 确认了一个 First 后，将 UPSN 同步为当前 First 的 RPSN + 1。因此，在后续收到 Middle/Last 时，如果 RPSN 大于 UPSN，则必然说明出现了响应包的丢失。图中的蓝色阴影区域是一个 RDMA Read 请求占用的 PSN 区域。此时需要将当前处理的 Read 请求重传。

3.4.5.1.3.4.2 响应包的写入

在完成缓冲区的释放和丢包的重传后，需要将数据包写入内存区域。响应包的写入依赖于 Scatter Entry，在发起 RDMA 请求时，我们在缓冲区中保存了请求对应的散列表，此时，需要从散列表中读取相应的表项进行数据的写入。这里需要解决的问题有以下几点：

问题 1：表项和数据包并不是一一对应的，即有可能一个数据包要使用多个表项，且不同数据包可能要使用同一个表项区域的不同部分。因此，当一个数据包处理完成，需要记录下其当前已经使用的表项区域，以便下个数据包从该区域的空余空间开始写入。下图给出了这种情况的说明。



为了方便控制，在实际实现时，对 Scatter Entry 的管理由一个专用的模块负责，该模块向 RequesterRecvCcontrol 提供对 Scatter Entry 的访问接口，访问接口如下表所示：

表 3.21 ScatterEntryManager 命令类型

命令类型	说明	命令格式
FETCH_ENTRY	获取一个可用的 Scatter Entry	
UPDATE_ENTRY	将缓冲区队首的 Entry 更新	
RELEASE_ENTRY	将缓冲区队首的 Entry 释放	

需要说明的是，UPDATE_ENTRY 是将当前缓冲区队首的 Entry 进行更新，将 Entry 中的 VA 字段修改为新的 VA。举例而言，写入一个数据包之前，首先获取一个 Entry，假设其 VA 为 0x10000000，长度为 8KB，数据包大小为 4KB，则该数据包写入后，VA 将变为 0x10001000，并被写回队首 Entry。收到的下一个响应包将从 0x10001000 开始写入。

ScatterEntryManager 的详细设计见。

问题 2：将数据按照表项写入时，要将数据向 256 比特对齐，此问题原理与 DataPack 需要解决的问题相同，在此不再赘述。

3.4.5.1.4 定时器事件的处理

在定时器控制模块中，我们实现了两个定时器，分别对应于丢包检测和 RNR 错误处理，两者的处理机制和事件接口完全一致，只是事件类型不同，这里只对丢包检测事件进行阐述。

定时器发送的事件类型可以分为两类：1.定时器超时，但未超过重传次数；2.定时器超时，且超过了重传次数。

在收到第一类事件时，直接重传 UPSN 对应的请求，同时，向定时器控制模块发送定时器重置命令；在收到第二类事件时，表明当前已经不能再进行重传了，将 UPSN 到 NPSN 之间的请求全部以错误状态结束，并将 QP 的状态切换为 Error。

3.4.5.1.5 错误请求（Bad Request）的处理

错误请求来自于 RequesterTransControl 在执行过程中检查到的请求错误。当 RequesterRecvControl 收到 BadReq 后，需要检查发送缓冲区中的所有请求是否都已经被清空，如果还有请求，则必须等待所有请求都被确认后才能处理 Bad Req 请求。

在处理 Bad Req 时，直接为当前 Bad Req 生成完成事件，并将 QP 状态切换为 Error。

3.4.5.2 模块接口

表 3.22 RequesterRecvControl 模块接口

模块名	RequesterEngine			
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
o_rc_te_loss_wr_en	输出	1	TimerControl	丢包计时器设定接口

ov_rc_te_loss_data	输出	64		
i_rc_te_loss_prog_full	输入	1		
o_rc_te_rnr_wr_en	输出	1		RNR 计时器设定接口
ov_rc_te_rnr_data	输出	64		
i_rc_te_rnr_prog_full	输入	1		
i_loss_expire_empty	输入	1		丢包计时器超时事件
iv_loss_expire_data	输入	32		
o_loss_expire_rd_en	输出	1		
i_rnr_expire_empty	输入	1		RNR 计时器超时事件
iv_rnr_expire_data	输出	32		
o_rnr_expire_rd_en	输出	1		
o_rpb_list_web	输出	1		Read Packet Buffer 元数据读写接口
ov_rpb_list_addrb	输出	14		
ov_rpb_list_dinb	输出	19		
iv_prb_list_doutb	输入	19		
o_rpb_element_web	输出	1		Read Packet Buffer 数据读写接口
ov_rpb_element_addrb	输出	9		
ov_rpb_element_dinb	输出	265		
iv_rpb_element_doutb	输入	265		
ov_rpb_free_data	输出	9		Read Packet Buffer 可用地址写接口
o_rpb_free_wr_en	输出	1		
i_rpb_free_prog_full	输入	1		
ov_reb_list_addrb	输出	14	MultiQueue	Read Entry Buffer 元数据读写接口
ov_reb_list_dinb	输出	29		
iv_reb_list_doutb	输入	29		
o_reb_list_web	输出	1		
ov_reb_element_addrb	输出	14		Read Entry Buffer 数据读写接口
ov_reb_element_dinb	输出	142		
iv_reb_element_doutb	输入	142		
o_reb_element_web	输出	1		
ov_reb_free_data	输出	14		Read Enetry Buffer 可用地址写接口
o_reb_free_wr_en	输出	1		
i_reb_free_prog_full	输入	1		
ov_swpb_list_addrb	输出	14		Send/Write Packet

ov_swpb_list_dinb	输出	25		Buffer 元数据读写接口
iv_swpb_list_doutb	输入	25		
o_swpb_list_web	输出	1		
ov_swpb_element_addrb	输出	12		Send/Write Packet Buffer 数据读写接口
ov_swpb_element_dinb	输出	268		
iv_swpb_element_doutb	输入	268		
o_swpb_element_web	输出	1		
ov_rpb_free_data	输出	12		Send/Write Packet Buffer 可用地址写接口
o_rpb_free_wr_en	输出	1		
i_rpb_free_prog_full	输入	1		
i_header_from_hp_empty	输入	1	HeaderParser	响应包包头元数据
o_header_from_hp_rd_en	输出	1		
iv_header_from_hp_data	输入	240		
iv_nd_from_hp_data	输入	256		响应包负载
i_nd_from_hp_empty	输入	1		
o_nd_from_hp_rd_en	输出	1		
i_header_to_rpg_prog_full	输入	1	ReqPktGen	请求包包头元数据
o_header_to_rpg_wr_en	输出	1		
ov_header_to_rpg_data	输出	256		
i_nd_to_rpg_prog_full	输入	1		网络数据传递接口
o_nd_to_rpg_wr_en	输出	1		
ov_nd_to_rpg_data	输出	256		
o_cxtmgt_cmd_wr_en	输出	1	CxtMgt	上下文读取命令接口
i_cxtmgt_cmd_prog_full	输入	1		
ov_cxtmgt_cmd_data	输出	128		
i_cxtmgt_resp_empty	输入	1		上下文响应接口
o_cxtmgt_resp_rd_en	输出	1		
iv_cxtmgt_resp_data	输入	128		
i_cxtmgt_cxt_empty	输入	1		上下文数据返回接口
o_cxtmgt_cxt_rd_en	输出	1		
iv_cxtmgt_cxt_data	输入	128		
o_cxtmgt_cxt_wr_en	输出	1		上下文数据写入接口
i_cxtmgt_cxt_prog_full	输入	1		
ov_cxtmgt_cxt_data	输出	128		

o_vtp_cmd_wr_en	输出	1	VirtToPhys	内存写请求命令接口
i_vtp_cmd_prog_full	输入	1		
ov_vtp_cmd_data	输出	256		
i_vtp_resp_empty	输入	1		权限检查响应
o_vtp_resp_rd_en	输出	1		
iv_vtp_resp_data	输入	8		
o_vtp_wqe_wr_en	输出	1		内存写请求数据
i_vtp_wqe_prog_full	输入	1		
ov_vtp_wqe_data	输出	256		

3.4.5.3 状态机设计

3.4.5.3.1 状态说明

表 3.23 RequesterRecvControl 状态说明

状态划分	状态说明
IDLE	初始状态
FETCH_CXT	读取上下文信息
RESP_CXT	等待上下文信息返回
TIMER_RETRANS	将 UPSN 对应的数据包重传
FLUSH	将 UPSN 到 NPSN 之间的所有数据包清空，并为每个数据包生成完成事件
BAD_REQ	处理前置模块发来的出错请求
CXT_WB	将上下文信息写回
CLEAR_TIMER	将对应 QP 的定时器关闭
SILENT_DROP	将当前数据包丢弃，不做任何处理
ACK_RELEASE_NORMAL	将 UPSN 到 RPSN 的请求全部释放
ACK_RELEASE_EXCEPTION	将 UPSN 到最早发出的未被确认的 RDMA Read 请求之间的所有数据包释放，并记最后释放的数据包的 PSN 为 IPSN
ACK_LOSS_RETRANS	将 IPSN 到 NPSN 之间的数据包全部重传
RESET_TIMER	如果是 PSN 错误，重置 Loss 定时器；如果是 RNR 错误，重置 RNR 定时器
NAK_IMPLICIT_RELEASE	将 UPSN 到（RPSN-1）的请求进行隐式释放，如果缓冲区中有 Read，则释放到 Read 停止，记 Read 的 PSN 为 ReadPSN，置 IPSN 为（ReadPSN-1）；如果缓冲区中没有 Read，则释放到（RPSN-1）停止，置 IPSN 为 RPSN：

	1.如果 Syndrome 为 PSN Error，置 JPSN 为（NPSN-1）； 2.如果 Syndrome 为 RNR Error，置 JPSN 为（RPSN-1）； 3.如果 Syndrome 为 Fatal Error，置 JPSN 为（RPSN-1）；
NAK_RETRANS	如果 JPSN≥IPSN，重传 IPSN，IPSN 自增 1
READ_IMPLICIT_RELEASE	对最早发起的未完成的 Read 之前的请求进行隐式释放
READ_RETRANS	对请求进行重传：1.如果是 First 或者 Only，将最早发出的未被确认的 Read 请求到 RPSN 的所有请求重传；2.如果是 Middle 或 Last，将缓冲区中的 Read 请求重传
READ_FETCH_ENTRY	获取一个可用的 Scatter Entry
READ_RESP_ENTRY	判断读回的 Entry
READ_SCATTER	将数据包写入 Entry 指定的内存区域
READ_UPDATE_ENTRY	将更新后的 Entry 写回
READ_FLUSH	将未处理完的 Read 响应包从 FIFO 中清空
READ_RELEASE_ENTRY	通知 ScatterEntryManager 将当前 Entry 释放： 1.如果 Entry 空间小于未写入的数据包大小，将当前 Entry 释放； 2.如果 Entry 空间等于未写入的数据包大小，且包类型为 Read Last 或 Read Only，将当前请求的所有 Entry 释放； 3.如果 Entry 空间等于未写入的数据包大小，且包类型为 Read First 或 Read Middle，将当前 Entry 释放。

3.4.5.3.2 状态转移图

由于该模块的状态转移图较为复杂，因此，从 RESP_CXT 开始对不同的状态转移进行切分，形成多个状态转移子图。

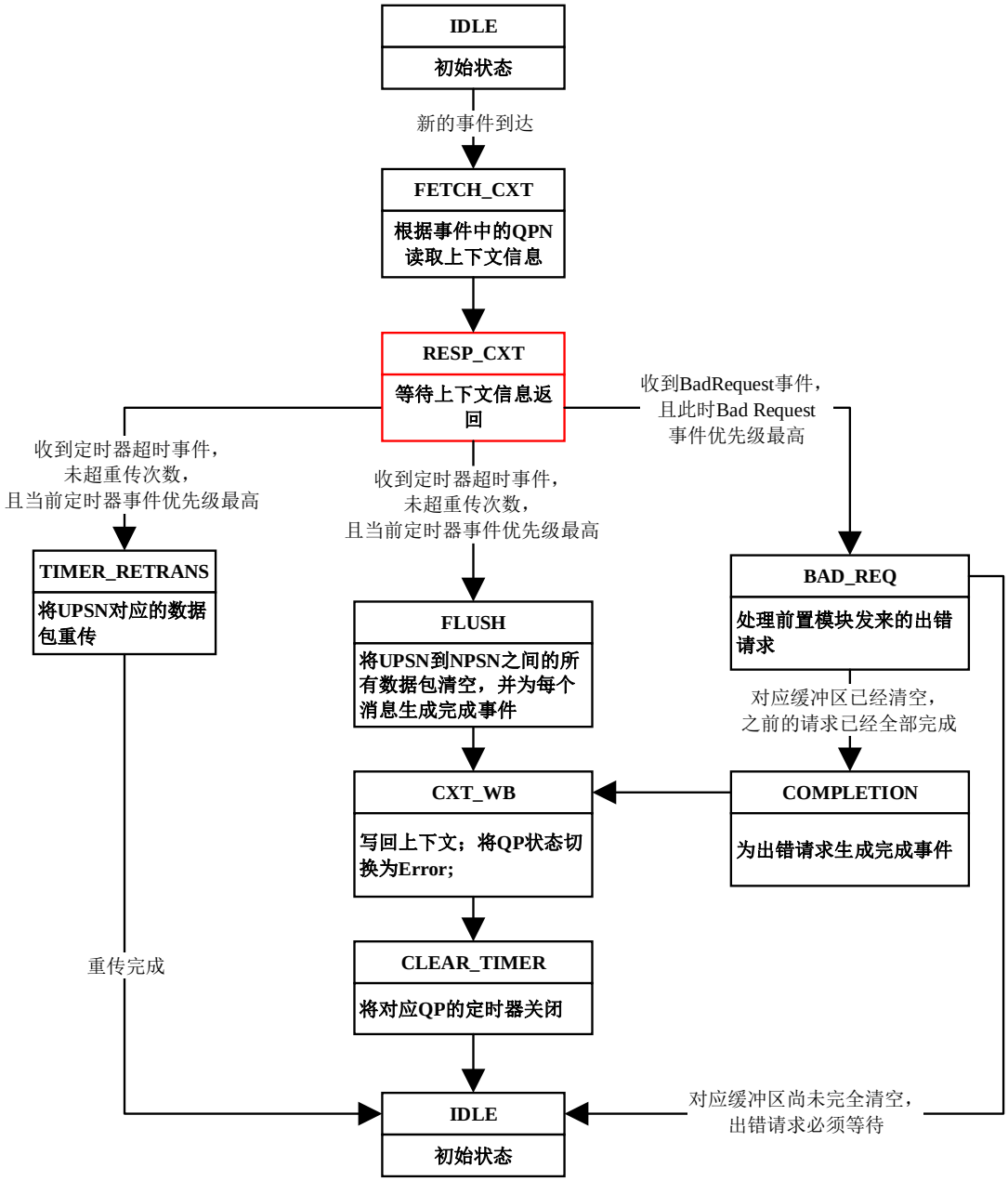


图 3.17 RequesterRecvControl 状态机-第 1 部分

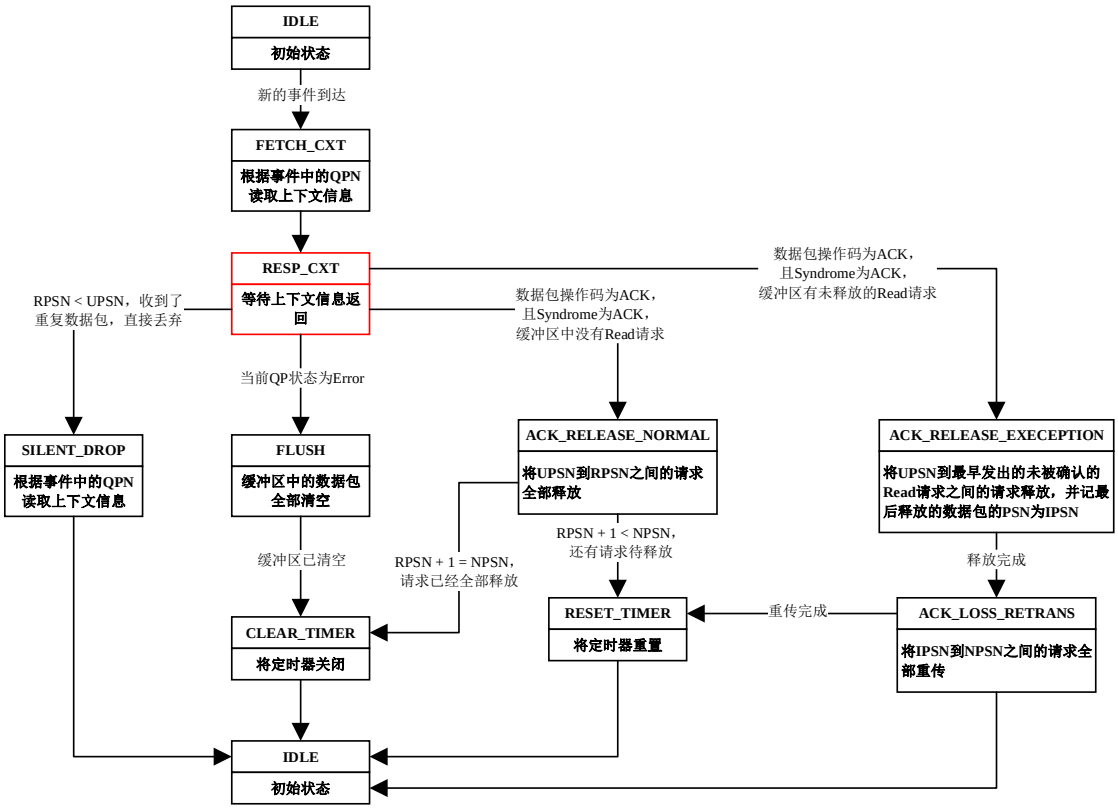


图 3.18 RequesterRecvControl 状态机-第 2 部分

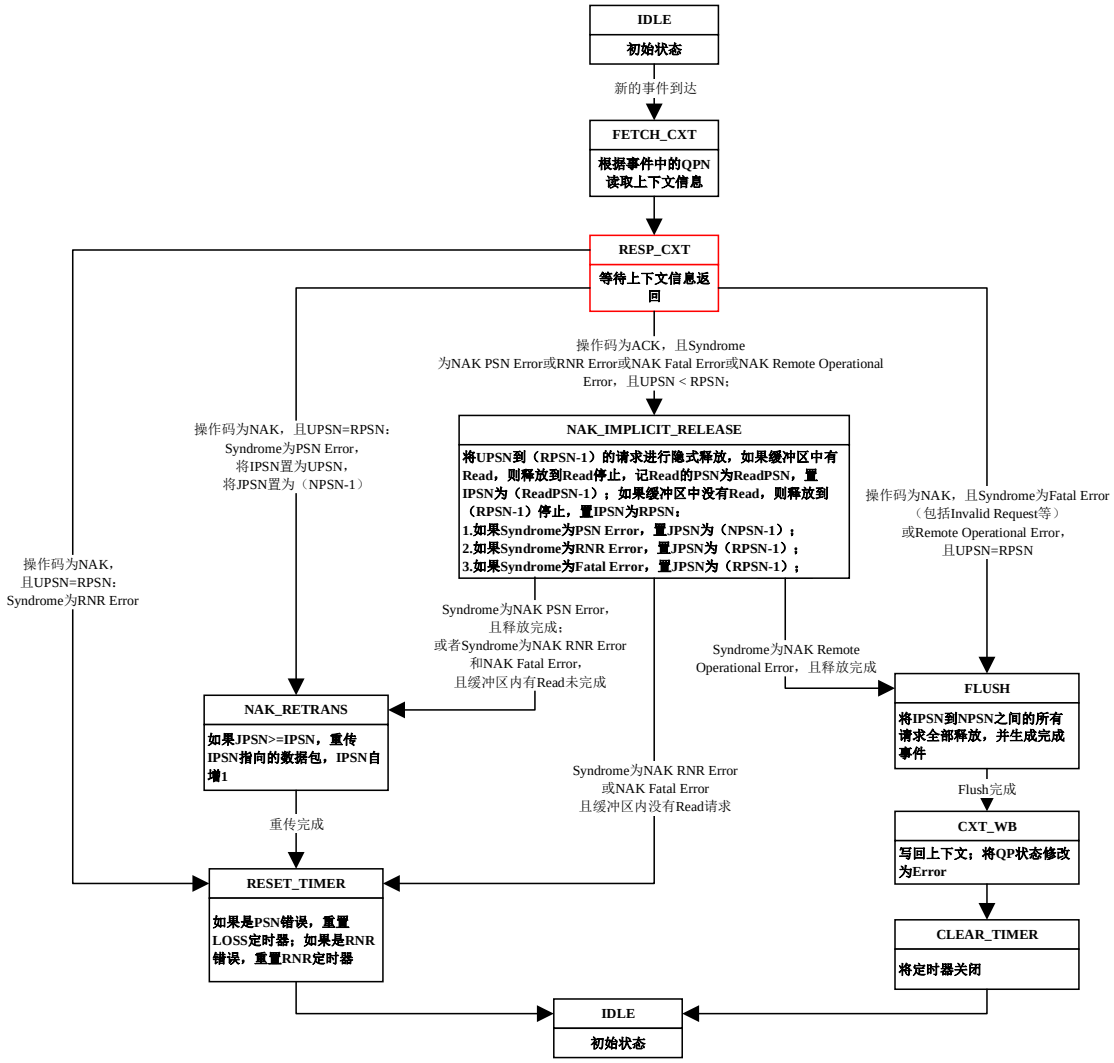


图 3.19 RequesterRecvControl 状态机-第 3 部分

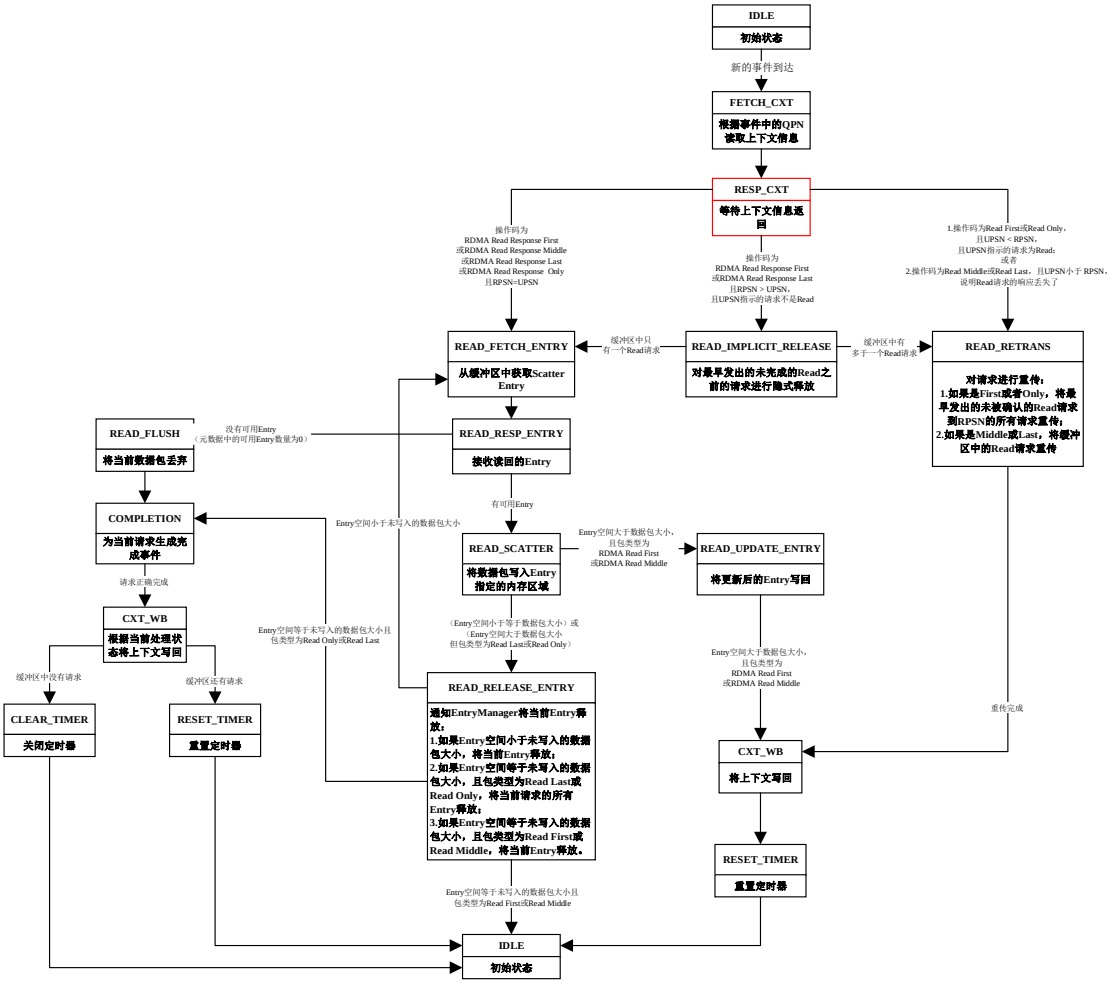


图 3.20 RequesterRecvControl 状态机-第 4 部分

3.4.5.3.3 状态转移说明

状态转移图中已详细标注，不再赘述。

3.4.5.4 子模块设计

3.4.5.4.1 散列表项管理模块（Scatter Entry Manager）

3.4.5.4.1.1 模块功能

散列表管理模块接管散列表的访问请求，对表项内容进行获取、更新和释放。需要注意的是，不仅 RequesterRecvControl 模块会访问散列表，RequesterTransControl 也会访问散列表，即在发起 RDMA 请求时将 Scatter Entry 插入缓冲区队列。由于缓冲区使用了 TDP（True Dual Port RAM），且两个模块分别访问队首和队尾元素，因此，不会造成冲突。

3.4.5.4.1.2 模块架构

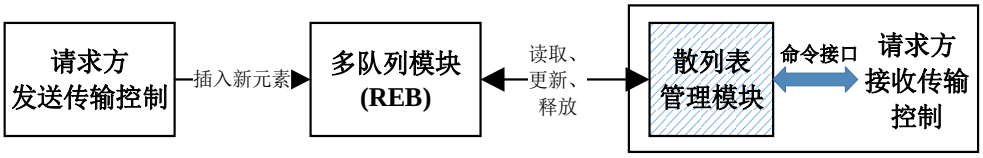


图 3.21 ScatterEntryManager 模块架构图

3.4.5.4.1.3 模块接口

表 3.24 ScatterEntryManager 模块接口

模块名	ScatterEntryManager 模块接口			
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
ov_reb_list_addrb	输出	14	MultiQueue	Read Entry Buffer 元数据读写接口
ov_reb_list_dinb	输出	29		
iv_reb_list_doutb	输入	29		
o_reb_list_web	输出	1		
ov_reb_element_addrb	输出	14		Read Entry Buffer 数据读写接口
ov_reb_element_dinb	输出	142		
iv_reb_element_doutb	输入	142		
o_reb_element_web	输出	1		
ov_rpb_free_data	输出	14		Read Entry Buffer 可用地址队列写接口
o_rpb_free_wr_en	输出	1		
i_rpb_free_prog_full	输入	1		
i_cmd_empty	输入	1	RequesterRecvControl 内部信号	Entry 管理命令接口
o_cmd_rd_en	输出	1		
iv_cmd_data	输入	32		
i_resp_prog_full	输入	1		Entry 管理响应接口
o_resp_wr_en	输出	1		
ov_resp_data	输出	32		
i_entry_prog_full	输入	1		Entry 条目返回接口
o_entry_wr_en	输出	1		
ov_entry_data	输出	128		
i_entry_empty	输入	1		Entry 条目写入接口
o_entry_rd_en	输出	1		

iv_entry_data	输入	128		

3.4.5.4.1.4 状态机设计

3.4.5.4.1.4.1 状态说明

表 3.25 RequesterRecvControl 状态说明

状态划分	状态说明
IDLE	初始状态
FETCH_ENTRY	从 Scatter Entry 缓冲区队首获取一个 Entry 并返回：如果有可用 Entry，返回获取到的 Entry 和状态码；否则返回错误状态。
UPDATE_ENTRY	更新队首的 Entry，将新的虚拟地址写入，无返回值。
RELEASE_ENTRY	根据命令中的操作码和 Entry 个数，将队列中对应个数的 Entry 释放掉，无返回值。

3.4.5.4.1.4.2 状态转移图

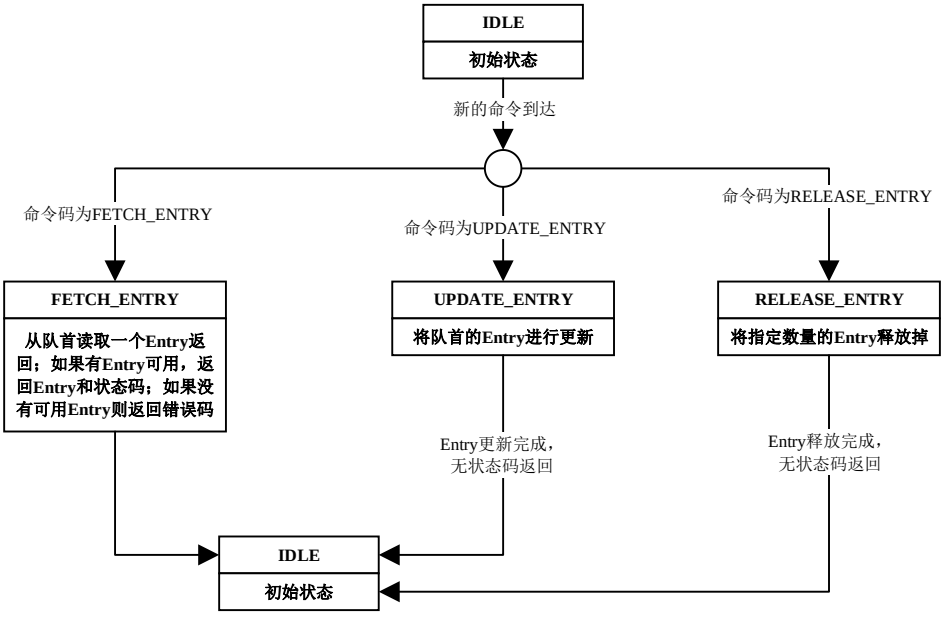


图 3.22 ScatterEntryManager 状态转移图

4 响应引擎（ResponderEngine）

4.1 模块功能

响应引擎完成对请求的处理，其主要功能可以划分为以下两部分：1.处理接收到的Request；2.生成待发送的Response。

对Request的处理又包含以下方面：1.检查接收到的数据包的包头，判断数据包是否有效；2.根据操作类型的不同，将数据写入主机内存或从主机内存中读取数据；3.根据操作类型的不同，生成完成事件。

对Response的处理包含以下方面：1.生成Send/RDMA Write的ACK；2.生成RDMA Read的Response。

上述不同功能分别由不同的子模块完成：Request的处理由请求执行模块、接收描述符管理模块和请求写地址模块协同完成；响应的处理由响应生成模块完成。

本章将详细介绍响应引擎的各部分功能实现。

4.2 模块架构

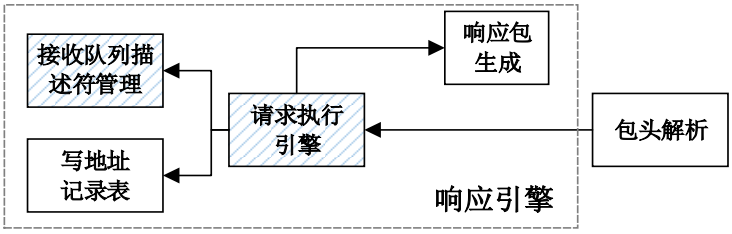


图 4.1 响应引擎架构图

4.3 模块接口

由内部子模块直接引出外部接口。

4.4 子模块设计

4.4.1 请求执行引擎（ExecutionEngine）

4.4.1.1 模块功能

请求执行引擎是响应引擎的核心模块，按顺序完成对包头的检查、请求的执行及完成事件的生成。

4.4.1.1.1 包头检查

每当 ExecutionEngine 收到一个数据包后，都需要读取上下文，并根据当前 QP 的状态决定是否要对数据包做进一步的处理。

如果当前 QP 状态处于 Reset，Initialize 或 Error，则 TCU 直接将收到的数据包丢弃，否则进行后续包头检查。

4.4.1.1.1.1 Q_Key 检查

4.4.1.1.1.2 操作类型检查

4.4.1.1.1.3 包序列号检查

响应引擎在可靠性保证中要完成的工作相对于请求引擎要简单很多，在响应引擎的视角上，其看到的 PSN 区间划分如下图所示。

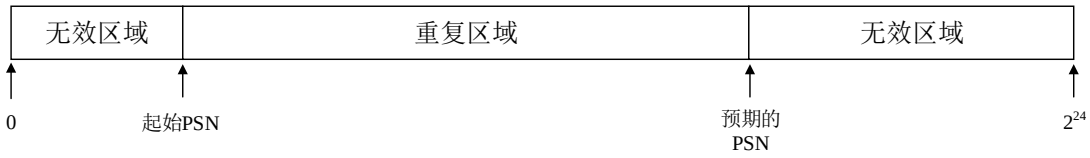


图 4.2 请求方包序列号检查示意图

对于每个接收到的请求数据包，执行引擎提取其 PSN（记为 SPSN）进行判定：

1.若 SPSN 落在无效区域，则说明请求方发送的请求出现了丢包。响应引擎生成一个 NAK 返回给 Requester，NAK 中的 PSN 为当前预期 PSN（记为 EPSN）。同时，将当前收到的数据包丢弃；

2.若 SPSN > 起始 PSN 且 SPSN < 预期 PSN，说明当前数据包是一个重复的请求。出现重复请求的原因是发送方的定时器超时，导致其重传请求。在这种情况下，定时器超时只可能是返回的 ACK 丢失或者 ACK 延迟过高。这两种情况不需要区分对待，采用相同的处理方式。

如果收到的请求是 Send 或者 RDMA Write，则发送一个 ACK，其携带的 PSN 为（EPSN – 1），以通知 Requester 在 EPSN 之前的数据包已经全部处理完成，可以进行确认；如果收到的 Request 是 RDMA Read，则由于 RDMA Read 的 Response 可能丢失，需要重新执行 RDMA Read 请求，并将数据返回。返回的数据包中携带的 PSN 为[SPSN, SPSN + n]，其中 n 是 RDMA Read 需要返回的数据包个数。

4.4.1.1.1.4 包头检查流程图

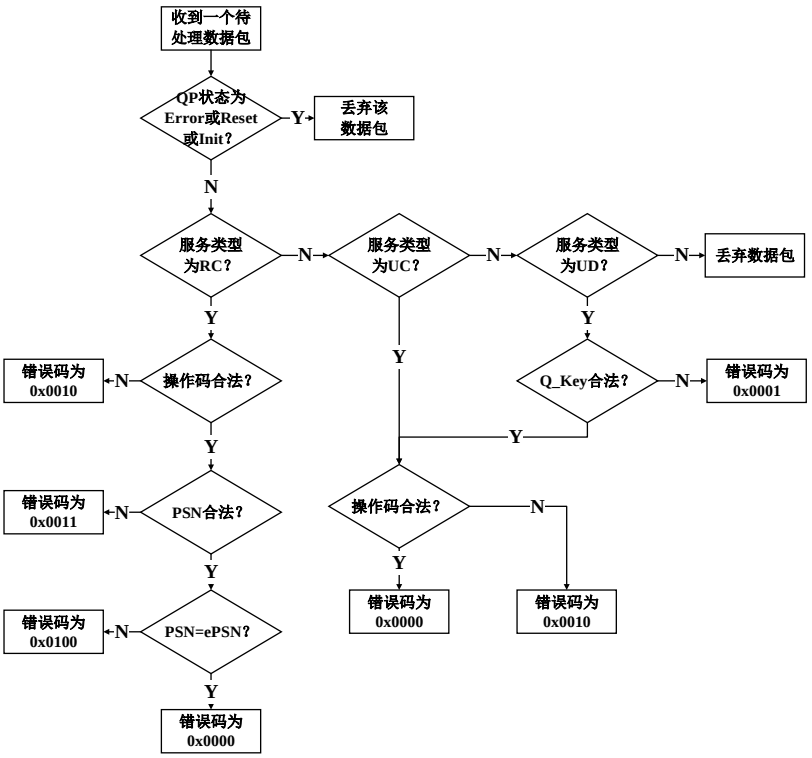


图 4.3 包头检查流程及对应错误码

4.4.1.1.2 请求的执行

4.4.1.1.2.1 执行 RDMA Read 请求

RDMA Read 请求的执行较为简单，只要包头检查通过就可以进入执行流程。从包头中提取出虚拟地址，R_Key 和长度，生成 DMA 请求发送给虚实地址转换模块即可。

需要注意的是，虚实地址转换模块有可能返回检查失败的结果，此时，ExecutionEngine 应该将错误码添加到元数据中，传递给响应包生成模块，便于其生成 NAK。

4.4.1.1.2.2 执行 RDMA Write 请求/RDMA Write with Immediate 请求

RDMA Write 包含两种操作类型，普通的 RDMA Write 和 RDMA Write with Immediate，两者除了将数据写入响应方内存外，后者的执行还需要消耗一个接收描述符，对描述符的获取与 Send 操作的机制相同，留在 Send 操作中阐述。本节仅讨论 RDMA Write 的写入机制。

在收到一个 RDMA Write 数据包后，需要将其携带的负载写入指定的内存地址。对于一个请求的第一个 RDMA Write 包（RDMA Write First/Only），其请求地址携带在其包头的 RETH 字段中，因此可以直接提取出虚拟地址发送 DMA 请求。但对于其它 RDMA Write 包（RDMA Write Middle/Last），其包头中不携带任何地址信息，因此，在处理一个请求的第一个 Write 包时，就需要将其地址存放下来，以便处理后续数据包时按照保存的地址写入。

因此，响应引擎中实现了一个地址表，该表的索引是 QP 号，每个表项保存着当前处理的 RDMA Write 请求的写地址和 R_Key。其处理流程如下：

1.在处理 RDMA Write First 时，提取 RETH 中的虚拟地址 VA，R_Key。将（VA＋PMTU）和 R_Key 存入地址保存表；

2.在处理 RDMA Write Middle/Last 时，根据 QP 号索引地址保存表，读取表项中的 VA 和 R_Key，将数据包写入指定的地址中，并将更新后的地址写回。

具体表项格式见地址保存表模块。

4.4.1.1.2.3 执行 Send/Send with Immediate 请求

在执行 Send 请求时，需要使用散列表将数据写入内存，与 RDMA Read 响应包的处理类似，我们设计了一个模块专门接管对接收方描述符的管理，并提供一系列访问命令。ExecutionEngine 需要用到的访问命令包括获取一个可用 Entry、更新一个 Entry、释放一个 Entry、释放一个 WQE。

与 Read 响应包的 Entry 管理命令不同，ExecutionEngine 本身并不知道接收描述符中携带了多少表项，因此，不能在释放命令中指定要释放的 Entry 数据，而是通过释放 WQE 的命令指示接收描述符管理模块对 WQE 进行释放。

在使用 Entry 将数据包写入内存时，仍然存在着数据向 256 比特对齐的问题，与 DataPack 类似，在此不再赘述。

由于响应包的 Entry 是 RequesterTransControl 写入的，而 Send 包的 Entry 需要从内存中获取，因此，其管理机制要复杂很多，详细设计见接收描述符管理模块。

RDMA Write with Immediate 的执行也需要消耗接收方 WQE，但是其并不使用其中的散列表，仅仅是获取一个 Entry 后，就将整个 WQE 释放。

4.4.1.2 模块接口

表 4.1 ExecutionEngine 模块接口

模块名	Execution 模块接口			
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
o_rwm_cmd_wr_en	输出	1	RQWQEManager	命令接口
ov_rwm_cmd_data	输出	128		
i_rwm_cmd_prog_full	输入	1		
i_rwm_resp_empty	输入	1		响应接口
iv_rwm_resp_data	输入	32		
o_rwm_resp_rd_en	输出	1		
o_rwm_entry_wr_en	输出	1		表项返回接口

ov_rwm_entry_data	输出	128		
i_rwm_entry_prog_full	输入	1		
i_rwm_entry_empty	输入	1		表项写入接口
iv_rwm_entry_data	输出	128		
o_rwm_entry_rd_en	输出	1		
o_wat_wr_en	输出	1	WriteAddrTable	写地址保存表读写接口
ov_wat_wr_data	输出	128		
ov_wat_addra	输出	14		
ov_wat_addrb	输出	14		
iv_wat_rd_data	输入	128		
o_rpg_md_wr_en	输出	1	RespPktGen	响应包元数据接口
ov_rpg_md_data	输出			
i_rpg_md_prog_full	输入	1		
i_header_empty	输入	1	HeaderParser	请求包包头接口
iv_header_data	输入	336		
o_header_rd_en	输出	1		
i_nd_empty	输入	1		网络数据接口
iv_nd_data	输入	1		
o_nd_rd_en	输出	256		
o_vtp_cmd_tvalid	输出	1	VirtToPhys	内存写请求命令接口
i_vtp_cmd_tready	输入	1		
ov_vtp_cmd_tdata	输出	256		
i_vtp_resp_tvalid	输入	1		权限检查响应
o_vtp_resp_tready	输出	1		
iv_vtp_resp_tdata	输入	8		
o_vtp_nd_tvalid	输出	1		内存写请求数据
i_vtp_nd_tready	输入	1		
ov_vtp_nd_tdata	输出	256		
o_cxtmgt_cmd_wr_en	输出	1	CxtMgt	上下文请求接口
i_cxtmgt_cmd_prog_full	输入	1		
ov_cxtmgt_cmd_data	输出	128		
i_cxtmgt_resp_empty	输入	1		上下文响应接口
o_cxtmgt_resp_rd_en	输出	1		
iv_cxtmgt_resp_data	输入	128		

i_cxtmgt_cxt_empty	输入	1		上下文数据返回接口
o_cxtmgt_cxt_rd_en	输出	1		
iv_cxtmgt_cxt_data	输入	128		
o_cxtmgt_cxt_wr_en	输出	1		上下文数据写入接口
i_cxtmgt_cxt_prog_full	输入	1		
ov_cxtmgt_cxt_data	输出	128		

4.4.1.3 状态机设计

4.4.1.3.1 状态说明

表 4.2 ExecutionEngine 状态说明

状态划分	状态说明
IDLE	初始状态
FETCH_CXT	读取上下文信息
RESP_CXT	等待上下文信息返回
GEN_RESP	生成响应包的元数据，传递给响应包生成模块
DROP	将当前数据包直接丢弃
CXT_WB	将上下文信息写回
ENTRY_FETCH	向接收描述符管理模块请求一个散列表项
ENTRY_RESP	等待表项返回
ENTRY_RELEASE	将队首表项释放
SEND_SCATTER	将数据包的数据写入 Entry 描述的区域
ENTRY_UPDATE	将未用完的 Entry 更新并写回
WQE_RELEASE	指示描述符管理模块将当前处理的 WQE 释放
VTP_RESP	等待 VTP 回复权限检查结果
VTP_CMD	向 VTP 发起内存写请求
WRITE_DATA	将数据写入内存
STORE_ADDR	保存更新后的写地址
ASYNC_ERROR	提交 Asynchronous 错误

4.4.1.3.2 状态转移图

与 RequesterRecvControl 类似，ExecutionEngine 的执行流程较为复杂，因此，将其状态机切分成不同的部分。

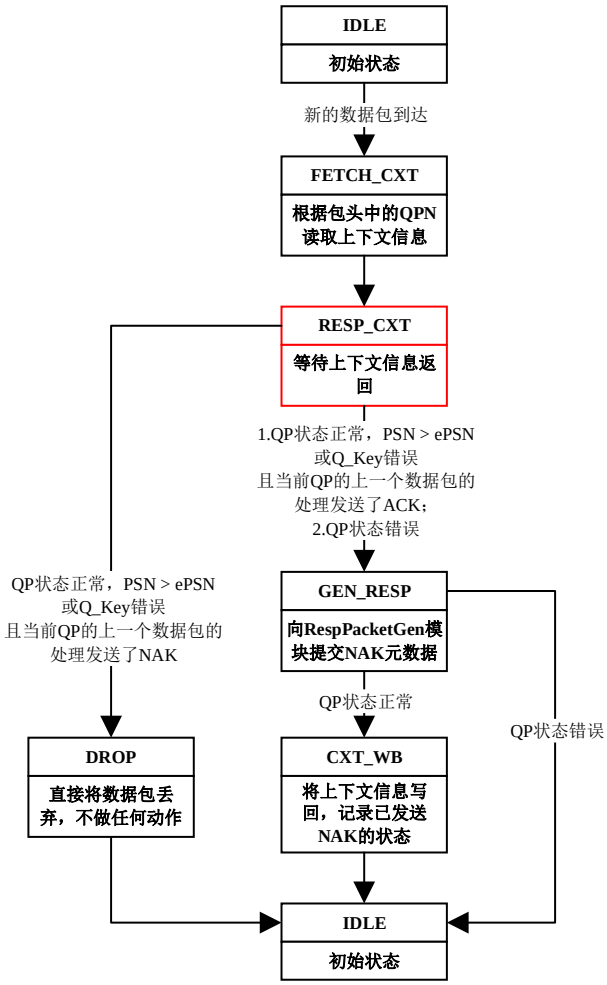


图 4.4 接受传输控制模块状态机-第 1 部分

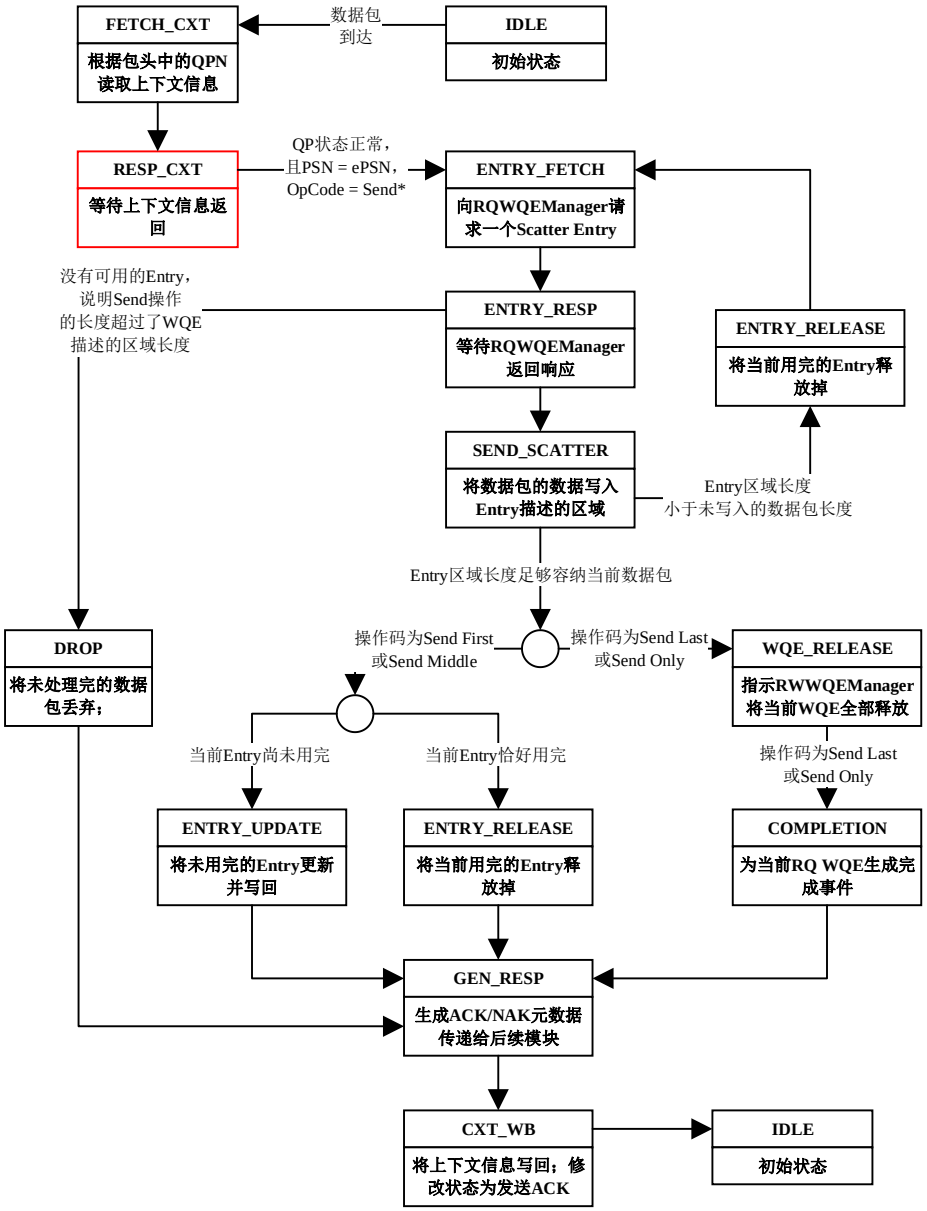


图 4.5 接受传输控制模块状态机-第 2 部分

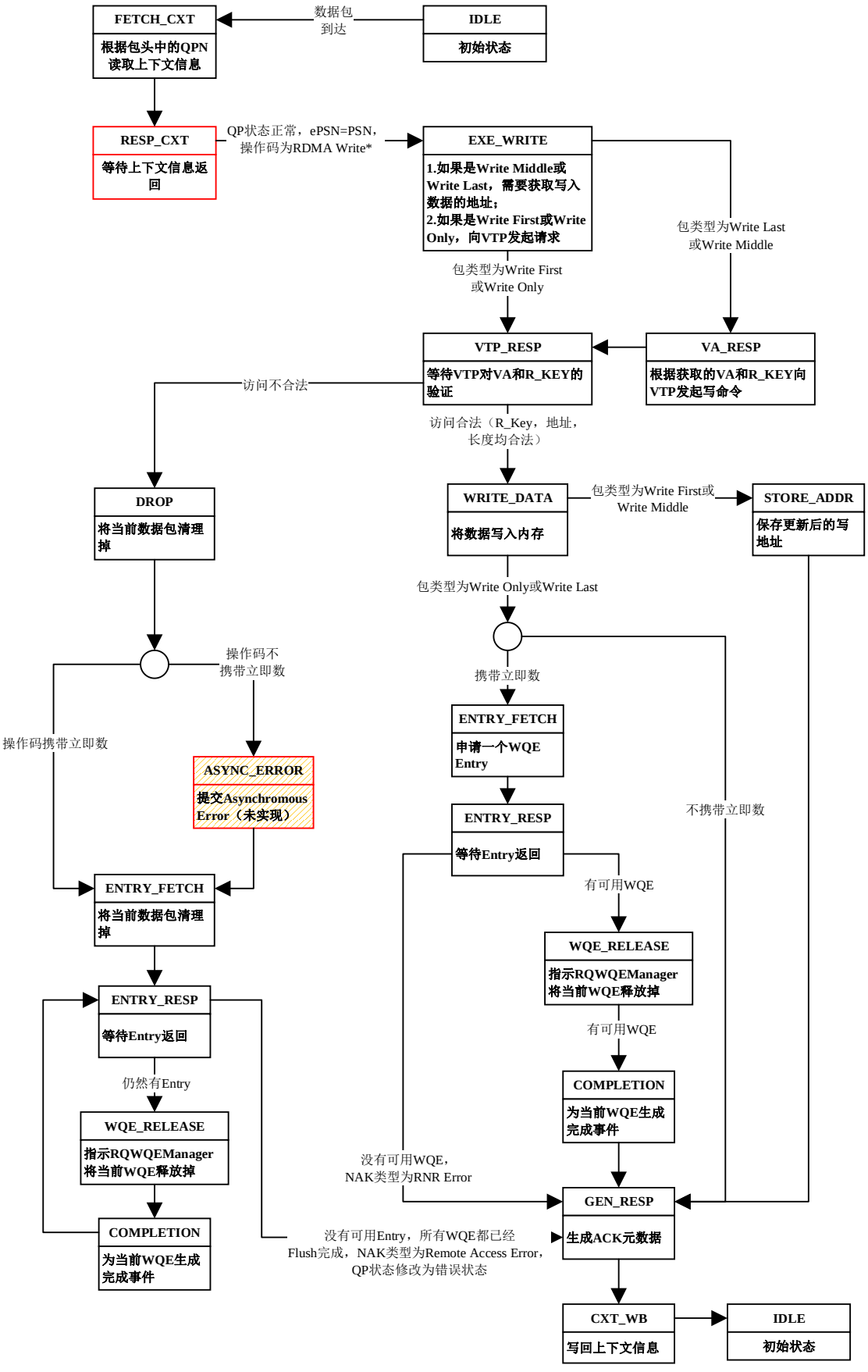


图 4.6 接受传输控制模块状态机-第 3 部分

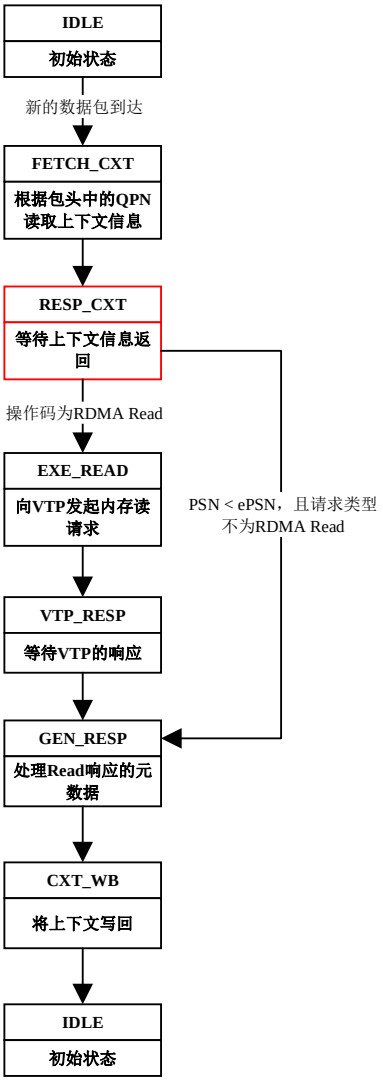


图 4.7 接受传输控制模块状态机-第 4 部分

4.4.2 接收队列描述符管理（RQWQEManager）

4.4.2.1 模块功能

描述符管理模块接收执行引擎发起的描述符访问请求，模块内部维护了一个描述符缓冲区，该缓冲区用于存放从内存中预取的描述符，缓冲区的管理延续多队列的管理方式。在目前的实现中，描述符缓冲区设置在本模块内部，没有划分到多队列模块中统一管理。

当执行引擎发起对描述符的请求时，RQWQEManager 首先在本地的缓冲区中查找是否有预取的 WQE，如果有则将 WQE 中的 Entry 返回，否则需要从内存中的 RQ 中获取描述符进行缓存。

在请求引擎中，主机使用门铃机制来通知请求引擎读取发送描述符，门铃中包含了待读取的 WQE 的相关信息（WQE 的偏移以及 WQE 的大小）。但在响应方并没有门铃机制，响应方使用接收到的 Send 数据包作为门铃来触发对描述符的读取。但 Send 数据包本身只包含 QP 号，而没有任何关于 WQE 的信息，因此，RQWQEManager 需要实现额外的机制来获

取 WQE。

WQE Length 的获取：提交到 RQ 中的 WQE 具有一个共同特性，即每个 WQE 最后都会有一个全 0 的字段（128 比特），该字段可以作为不同描述符之间的定界符。

WQE Offset 的获取：RQWQEManager 自行维护一个 WQE Offset 的偏移表，每次需要取 WQE 时，就从当前 Offset 开始，读取固定长度的数据，然后在处理完读回的数据后再将 Offset 进行更新。

下图是 RQ WQE 的布局。

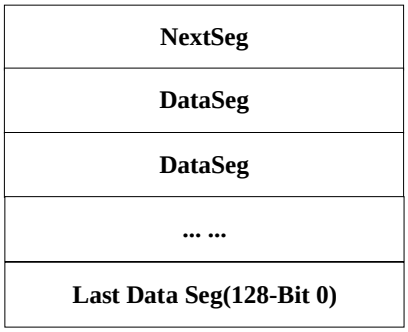


图 4.8 接收描述符格式

其中 NextSeg 携带了下一个描述符的大小信息，DataSeg 就是 Scatter Entry，每个 RQ WQE 的最后 128 比特是全 0。在 Mellanox Connect-X4 网卡中，DataSeg 的数量为 30，因此，每次获取 WQE 时，只要发起的读请求大小为 512B，就可以读到至少一个有效的 WQE。

当读回一个 WQE 时，需要对读回的数据进行解析，按照 WQE 之间的定界符对 WQE 进行划分。详细的处理流程在状态机设计中给出。

4.4.2.2 模块接口

表 4.3 RQWQEManager 模块接口

模块名	RQWQEManager			
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_ee_cmd_empty	输入	1	ExecutionEngine	描述符管理命令接口
iv_ee_cmd_data	输入			
o_ee_cmd_rd_en	输出	1		
i_ee_resp_prog_full	输入	1		描述符管理响应接口
o_ee_resp_wr_en	输出	1		
ov_ee_resp_data	输出			
o_vtp_cmd_wr_en	输出	1	VirtToPhys	内存读请求命令接口

i_vtp_cmd_prog_full	输入	1		
ov_vtp_cmd_data	输出	256		
i_vtp_resp_empty	输入	1		权限检查响应
o_vtp_resp_rd_en	输出	1		
iv_vtp_resp_data	输入	8		
i_vtp_wqe_empty	输入	1		内存读请求数据
o_vtp_wqe_rd_en	输出	1		
iv_vtp_wqe_data	输入	256		

4.4.2.3 状态机设计

4.4.2.3.1 状态说明

表 4.4 RQWQEManager 状态说明

状态划分	状态说明
IDLE	初始状态
CMD_FETCH_ENTRY	从缓冲区中读取 Entry 返回；如果没有可用 Entry，则发起内存读请求（512B）
CMD_UPDATE_ENTRY	更新队首的 Entry
CMD_RELEASE_ENTRY	释放队首的 Entry
CMD_RELEASE_WQE	释放队首的 WQE
FLUSH_DATA	将 VTP 对接的 FIFO 中尚未读完的数据清除；之所以需要清除一部分数据，是因为读取的 512B 数据可能有不完整的 WQE，这部分数据是不能写入 WQE 缓冲区的，因此要从 FIFO 中清除
MEM_RESP	等待内存读数据返回
WQE_SPLIT	处理内存中返回的数据，将 WQE 写入缓冲区

4.4.2.3.2 状态转移图

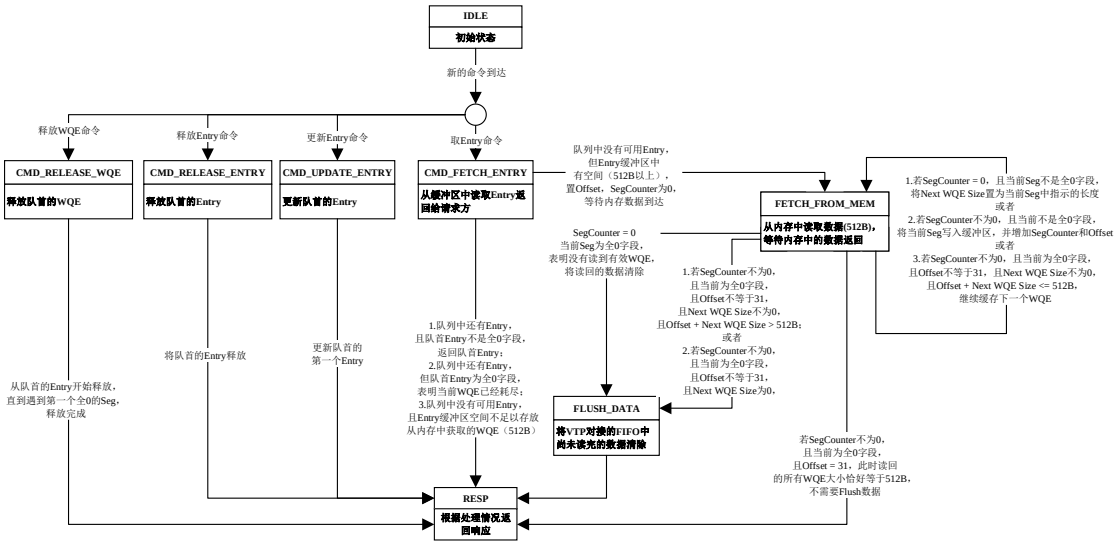


图 4.9 接收描述符管理模块状态机

4.4.2.3.3 状态转移说明

表 4.5 WQEParser 状态转移说明

现态	次态	转移条件
CMD_FETCH_ENTRY	MEM_RESP	当前队列中没有可用 Entry，但缓冲区中的空间超过 512B，Offset 为 0，SegCounter 为 0，发起对内存的访问请求
	RESP	1.队列中还有可用 Entry，且队首 Entry 不是全 0 字段，返回队首 Entry； 2.队列中还有 Entry，但队首 Entry 为全 0 字段，表明当前 WQE 已经耗尽；返回 Entry 耗尽错误； 3.队列中没有可用 Entry，且 Entry 缓冲区空间不足以容纳从内存中获取的 WQE（512B），返回缓冲区耗尽错误。
MEM_RESP	WQE_SPLIT	内存数据返回：
WQE_SPLIT	FLUSH_DATA	1.SegCounter = 0，且当前 SegCounter 为全 0 字段，说明没有读到有效的 WQE，将读回的数据清除； 2.SegCounter 不为 0，且当前为全 0 字段，且 Offset 不等于 31，且 Next WQE Size 不为 0，且 Offset+Next WQE Size > 512B，说明下一个要处理的 WQE 没有完全读回，将读回的剩余数据清除； 3.SegCounter 不为 0，且当前为全 0 字段，且 Offset 不为 31，且 Next WQE Size 为 0，说明当前已经将

		RQ 中的所有 WQE 读回，将读回的剩余数据（0 字段）清空。
	RESP	若 SegCounter 不为 0，且当前为全 0 字段，且 Offset 为 31，此时读回的所有 WQE 大小之和恰好为 512B，不需要 Flush 数据，直接返回队首 WQE。
	WQE_SPLIT	1.若 SegCounter = 0，且当前 Seg 不是全 0 字段， 将 Next WQE Size 置为当前 Seg 中指示的长度 或者 2.若 SegCounter 不为 0，且当前不是全 0 字段， 将当前 Seg 写入缓冲区，并增加 SegCounter 和 Offset 或者 3.若 SegCounter 不为 0，且当前为全 0 字段， 且 Offset 不等于 31，且 Next WQE Size 不为 0， 且 Offset + Next WQE Size <= 512B， 继续缓存下一个 WQE
CMD_UPDATE_ENTRY	RESP	更新队首的第一个 Entry 完成
CMD_RELEASE_ENTRY	RESP	队首元素释放完成
CMD_RELEASE_WQE	RESP	从队首的 Entry 开始释放，直到释放到第一个全 0 的字段，释放完成

4.4.3 写地址记录表（WriteAddrTable）

4.4.3.1 模块功能

写地址记录表内部是一个 Single Dual Port RAM，用于存放 RDMA Write 请求的写地址，其本身只提供 RAM 的读写接口，不实现任何逻辑功能。

4.4.3.2 模块架构

写地址表表项格式如下表所示。

表 4.6 Write Addr Table 表格式

索引号	表项内容（96-bit）	
	R_Key（95~64）	VirtualAddress（63~0）
0	……	……
1	……	……
……	……	……
16383	……	……

4.4.3.3 模块接口

表 4.7 WriteAddrTable 模块接口

模块名	WriteAddrTable			
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_wat_wr_en	输入	1	ExecutionEngine	写地址表读写控制接口
iv_wat_wr_data	输入	128		
iv_wat_addra	输入	14		
iv_wat_addrb	输入	14		
ov_wat_rd_data	输出	128		

4.4.4 响应包生成（ResponsePacketGen）

4.4.4.1 模块功能

响应包生成模块从执行引擎读取元数据，由元数据生成对应的响应包；如果元数据指示 RDMA Read 请求响应，还需要从与 VTP 的接口 FIFO 中读取从内存中返回的数据，将数据分割为多个 RDMA Read Response 包，并为其生成响应包头。

将 RDMA Read Response 数据包分段的过程与请求分段类似：

1.若数据总量小于等于 PMTU，则生成的响应包为 RDMA Read Resposne Only；

2.若数据总量小于等于 2 个 PMTU，则生成的响应包为 First 和 Last；

3.若数据总量大于 2 个 PMTU，则生成的响应包为 First，Middle 和 Last。

4.4.4.2 模块接口

表 4.8 ResponsePacketGen 模块接口

模块名	ResponsePacketGen 模块接口			
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_md_empty	输入	1	ExecutionEngine	响应元数据传递接口
iv_md_data	输入	96		

o_md_rd_en	输出	1		
i_nd_empty	输入	1		读响应数据传递接口
iv_nd_data	输入	256		
o_nd_rd_en	输出	1		
i_trans_prog_full	输入	1	256To64Trans	数据包的第一个 256 比特最高两位为 2'b01，最后一个 256 比特的最高两位为 2'b10。
o_trans_wr_en	输出	1		
ov_trans_data	输出	258		

4.4.4.3 状态机设计

4.4.4.3.1 状态说明

表 4.9 ResponsePacketGen 状态说明

状态划分	状态说明
IDLE	初始状态
HEADER	生成 RETH 头部
READ_SEG	根据负载长度生成 Read Response 的头部
READ_PAYLOAD	读取 Payload，与 Header 一起向 256 比特对齐

4.4.4.3.2 状态转移图

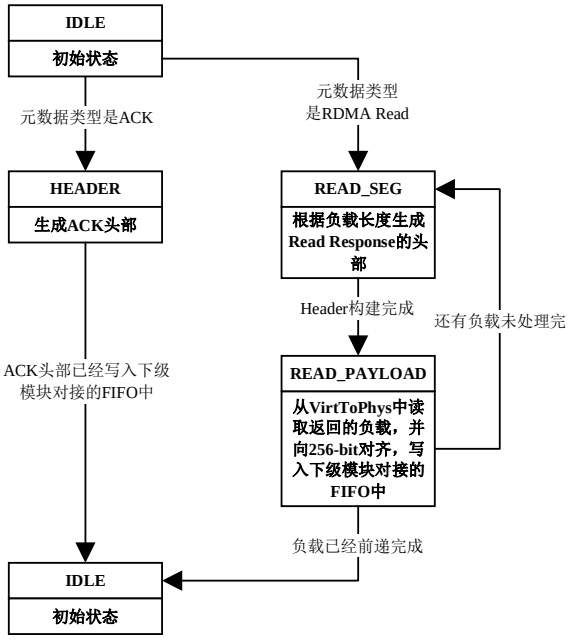


图 4.10 响应包生成模块状态图

5 包头解析（HeaderParser）

5.1 模块功能

包头解析模块对收到的数据包进行解析，将包头与负载分离，提取包头中的元数据并将其传递给对应处理单元。

5.2 模块架构

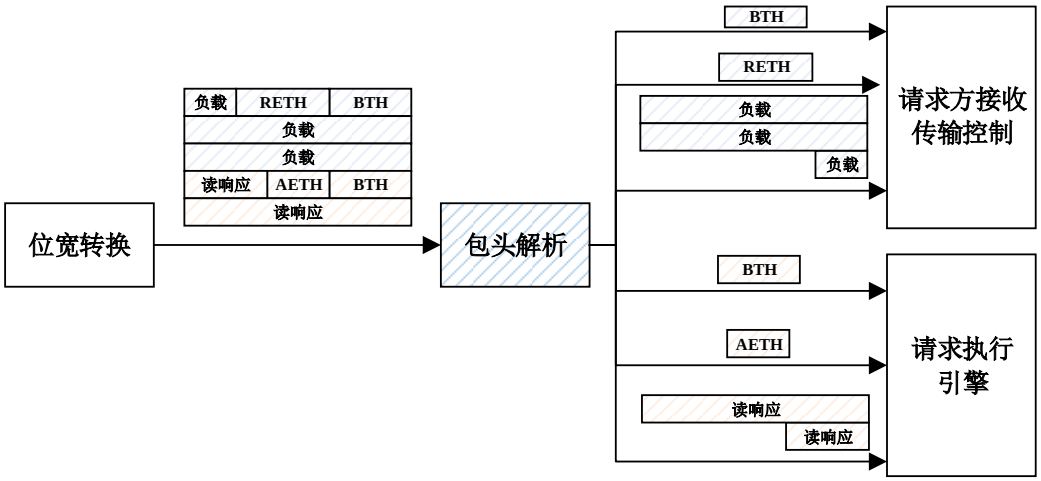


图 5.1 包头解析器架构示意图

如图所示，包头解析模块接收的是连续的网络数据流，经过解析后，包头和负载被分离，转发到不同的对接 FIFO 中。

5.3 模块接口

表 5.1 HeaderParser 模块接口

模块名	HeaderParser 模块接口			
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	RequesterRecvControl	时钟信号
rst	输入	1		复位信号
i_header_to_rrc_prog_full	输入	1		包头元数据接口
o_header_to_rrc_wr_en	输出	1		
ov_header_to_rrc_data	输出	240		
ov_nd_to_rrc_data	输入	256		网络数据接口
i_nd_to_rrc_prog_full	输入	1		

o_nd_to_rrc_wr_en	输出	1		
i_to_ee_header_prog_full	输入	1	Execution Engine	包头元数据接口
ov_to_ee_header_data	输出	336		
o_to_ee_header_wr_en	输出	1		
i_to_ee_nd_prog_full	输入	1		网络数据接口
ov_to_ee_nd_data	输出	1		
o_nd_wr_en	输出	256		
i_bit_trans_empty	输入	1	64To256Trans	数据包的第一个 64 比特最高两位为 2’b01，最后一个 64 比特的最高两位 2’b10。
iv_bit_trans_data	输入	256		
o_bit_trans_rd_en	输出	1		

5.4 状态机设计

5.4.1 状态说明

每个状态就是对不同包头字段的解析，不再赘述。

5.4.2 状态转移图

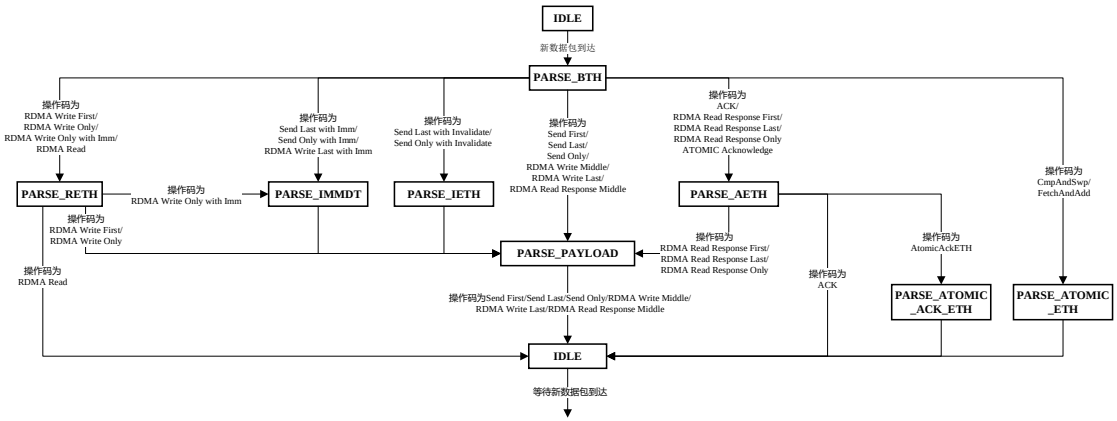


图 5.2 包头解析器状态机

5.4.3 状态转移说明

表 5.2 HeaderParser 状态转移说明

现态	次态	转移条件
PARSE_BTH	PARSE_RETH	操作码为 RDMA Write First/RDMA Write Only/

		RDMA Write Only with Imm/RDMA Read
	PARSE_IMMDT	操作码为 Send Last with Imm/Send Only with Imm/ RDMA Write Last with Imm
	PARSE_IETH	操作码为 Send Last with Invalidate/ Send Only with Invalidate
	PARSE_PAYLOAD	操作码为 Send First/Send Last/Send Only/RDMA Write Middle/ RDMA Write Last/RDMA Read Response Middle
	PARSE_AETH	操作码为 ACK/RDMA Read Response First/ RDMA Read Response Last/RDMA Read Response Only/ATOMIC Acknowledge
	PARSE_ATOMIC_ETH	操作码为 CmpAndSwp/FetchAndAdd
PARSE_RETH	PARSE_PAYLOAD	操作码为 RDMA Write First/RDMA Write Only
	PARSE_IMMDT	操作码为 RDMA Write Only with Imm
	IDLE	操作码为 RDMA Read
PARSE_IMMDT	PARSE_PAYLOAD	直接跳转
PARSE_IETH	PARSE_PAYLOAD	直接跳转
PARSE_AETH	PARSE_PAYLOAD	操作码为 RDMA Read Response First/ RDMA Read Response Last/RDMA Read Response Only
	PARSE_ATOMIC_ACK_ETH	操作码为 Atomic AckETH
PARSE_ATOMIC_ACK_ETH	IDLE	直接跳转
PARSE_ATOMIC_ETH	IDLE	直接跳转
PARSE_PAYLOAD	IDLE	负载解析完成

6 位宽转换（BitWidthTrans）

6.1 模块功能

由于当前 RDMA 引擎对接的链路层使用的是 64 比特位宽的 FIFO，而 RDMA 引擎的内部位宽是 256bite，因此，在 RDMA 引擎和链路层之间需要增加一个位宽转换模块，将 64 比特数据转换为 256 比特数据，或者将 256 比特数据转换为 64 比特数据。

6.2 模块架构

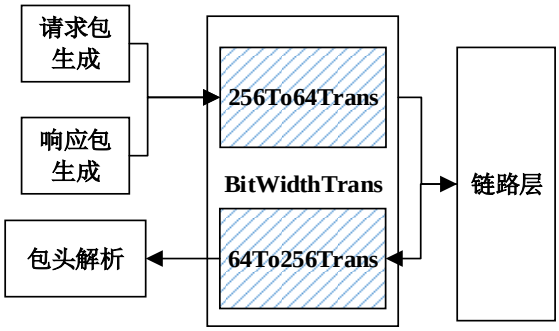


图 6.1 位宽转换模块示意图

6.3 模块接口

由内部子模块直接引出外部接口。

6.4 子模块设计

6.4.1 256To64Trans

6.4.1.1 模块功能

调度请求包生成模块和响应包生成模块发出的数据包，将 256 比特数据写入 64 比特 FIFO。

6.4.1.2 模块接口

表 6.1 256To64Trans 模块接口

模块名	256To64Trans			
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_req_trans_empty	输入	1	ReqPktGen	256 比特指示数据包开始，257 比特指示数据包结束，中间数据这两个比特均为 0
o_req_trans_rd_en	输出	1		
iv_req_trans_data	输入	258		
i_resp_trans_empty	输入	1	RespPktGen	256 比特指示数据包开始，257 比特指示数据包结束，中间数据这两个比特均为 0
o_resp_trans_rd_en	输出	1		
iv_resp_trans_data	输入	258		
i_outbound_pkt_prog_full	输入	1	LinkLayer	72 比特由 1 字节的控制信息和 8 字节的数据构成，控制信息字段说明如下： S：包开始 T：包结束 E：包结束 error ValidByte：指示当前 64B 中的有效字节数。目前未使用。 VL：Virtual Lane，范围 0~3
o_outbound_pkt_wr_en	输出	1		
ov_outbound_pkt_data	输出	72		

6.4.1.3 状态机设计

该模块设计较为简单，只给出状态转移图。

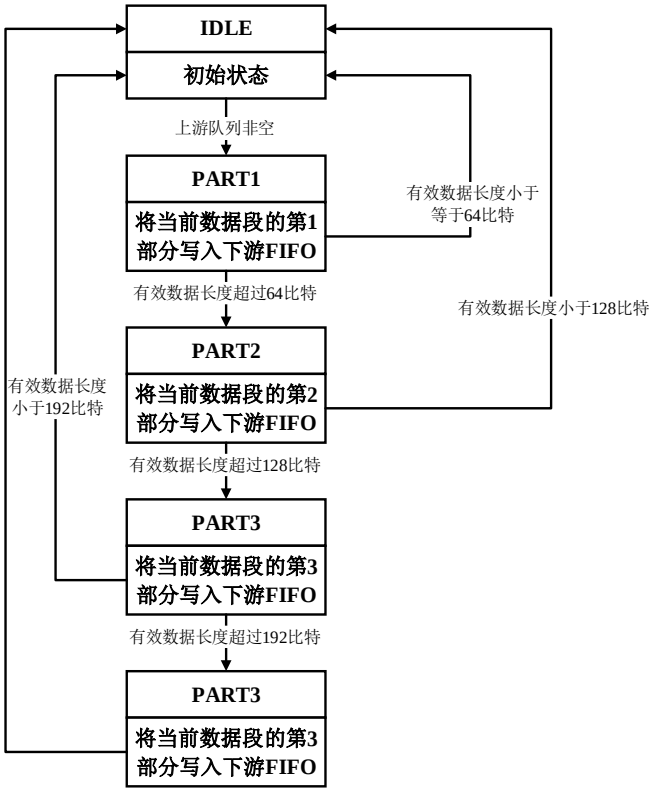


图 6.2 256To64BitTrans 模块状态机

6.4.2 64To256Trans

6.4.2.1 模块功能

将 64 比特数据包拼成 256 比特数据包。

6.4.2.2 模块接口

表 6.2 64To256Trans 模块接口

模块名	64To256TransTrans			
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_inbound_pkt_empty	输入	1	LinLayer	72 比特由 1 字节的控制信息和 8 字节的数据构成，控制信息字段说明

o_inbound_pkt_rd_en	输出	1		如下： S：包开始 T：包结束 E：包结束 error ValidByte：指示当前 64B 中的有效字节数。目前未使用。 VL：Virtual Lane，范围 0~3
iv_inbound_pkt_data	输入	72		
i_to_hp_prog_full	输入	1	HeaderParser	数据包的第一个 256 比特，最高两位为 2'b01，数据包的最后 256 比特，最高两位为 2'b10。
o_to_hp_wr_en	输出	1		
ov_to_hp_data	输出	258		

6.4.2.3 状态机设计

本模块不需要状态机实现，只需要维护两个寄存器，一个 Counter，一个 DataReg。当有数据到达时，进行如下判断和处理：

1.如果当前数据携带着包尾标记，则根据 Counter 值，将当前数据和 DataReg 进行拼接后一并写入 256 比特的队列中，将 Counter 清零；

2.如果当前数据不携带包尾标记，且 Counter 不为 3，则根据当前 Counter 值，将数据和 DataReg 进行拼接，并写回 DataReg 暂存；

3.如果当前数据不携带包尾标记，且 Counter 为 3，则将数据和 DataReg 进行拼接后，一并写入 256 比特的队列中，将 Counter 清零。

7 附录

7.1 各种命令/响应格式

7.1.1 门铃格式说明

SQ 的门铃格式如下表所示。

表 7.1 门铃格式

+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
offset																										F	opcode				
QPN																								size							

各字段含义如下表所示：

表 7.2 DoorbellProcessing 模块接口

字段	说明
opcode	用户指定的操作码，需要注意的是，这里的操作码并不是 BTH 头部中携带的操作码。BTH 的操作码要根据用户指定的操作码以及数据分段情况自行计算。
F	Fence 位，顾名思义，该标志位起到屏障作用，当该位被置 1，则当前请求必须等待其之前提交的所有请求执行完成才能开始执行。
offset	待读取的描述符在发送队列中的偏移，以 16B 为单位。
size	待读取的描述符大小，以 16B 为单位。
QPN	待处理的请求对应 QP 的编号。

7.1.2 上下文访问及响应格式

7.1.2.1 命令格式及说明

上下文读取命令格式如下表所示，后文涉及到上下文处理的命令与响应均遵循该格式。

表 7.3 上下文读取命令格式

+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
32’h00000000																															
32’h00000000																															
32’h00000000																															
Type				OpCode				QPN																							

OpCode 和 Type 字段的说明如下表所示：

7.1.2.2 响应格式及说明

7.1.3 内存读写命令及响应格式

7.1.3.1 内存读写命令格式

表 7.4 内存读写命令格式

+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																															
Virtual Address																															
Length																															
Reserved																															

各字段说明如下表：

表 7.5 内存读写命令字段说明

字段	说明
Type	
OpCode	
Flags	
Protection Domain	
Key	
Virtual Address	
Length	

7.1.3.2 内存读写响应格式

表 7.6 内存读写响应格式

+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																												State			

表 7.7 内存读写响应各字段说明

字段	说明
State	PD_ERR（4’b0100）：PD 错误

	FLAGS_ERR（4’b0011）：访问权限错误 KEY_ERR（4’b0010）：Key 无效错误 LENGTH_ERR（4’b0001)：访问数据空间大小错误 SUCCESS(4’b0000)：访问检查通过

7.1.4 定时器事件格式

7.1.4.1 定时器设定事件格式

表 7.8 定时器设定事件格式

+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
QPN																								OpCode							
Reserved																				TimerValue								CV			

表 7.9 定时器设定事件各字段说明

字段	说明
OpCode	Set Timer（8’h00）：设定定时器值并启动定时器 Stop Timer（8’hFF）：关闭定时器 Restart Timer（8’h01)：将定时器重置为设定值，并重置重传次数
QPN	需要操作的定时器所属 QP 号
CounterValue(CV)	需要设定的重传次数值
TimerValue	需要设定的定时器值

7.1.4.2 定时器超时事件格式

表 7.10 定时器超时事件格式

+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
QPN																								Event							

表 7.11 定时器设定事件各字段说明

字段	说明
Event	Timer Expired（8’h00）：定时器超时，未超过重传次数 Counter Expired（8’hFF）：定时器超时，且超过了重传次数
QPN	需要发送超时事件的 QP 号

7.1.5 散列表管理模块命令及响应格式

7.1.5.1 命令格式

表 7.12 散列表管理模块命令格式

+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
QPN																								Number						Op	
Virtual Address[31:0]																															
Virtual Address[63:0]																															
R_Key																															
Length																															

表 7.13 散列表管理模块命令说明

字段	说明
Op	命令码 Fetch Entry（2’h00）：获取队首 Entry Update Entry（2’h01）：更新队首 Entry Release Entry（2’h10）：释放指定数量的 Entry
Number	需要释放的 Entry 数量
QPN	需要处理的 QP 号

7.1.5.2 响应格式

表 7.14 散列表管理模块响应格式

+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
QPN																								Resp							
Virtual Address[31:0]																															
Virtual Address[63:0]																															
R_Key																															
Length																															

表 7.15 散列表管理模块响应说明

字段	说明
Resp	响应码 Valid Entry（8’h00）：有可用 Scatter Entry Invalid Entry（8’h01）：无可用 Scatter Entry
QPN	需要处理的 QP 号

7.1.6 接收描述符管理模块及响应格式

7.1.6.1 命令格式

表 7.16 接收描述符管理模块命令格式

+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
QPN																								Reserved							Op
Protection Domain																															
RQ_LKey																															
Entry_VA[31:0]																															
Entry_VA[63:32]																															
Entry_LKey																															
Entry_Length																															

表 7.17 接收描述符管理模块命令说明

字段	说明
Op	命令码 Fetch Entry（2’b00）：获取队首 Entry Update Entry（2’b01）：更新队首 Entry Release Entry（2’b10）：释放队首的 Scatter Entry Release WQE（2’b11）：释放队首的 WQE
QPN	需要处理的 QP 号
PD	RQ 保护域，由 EE 提供
RQ_LKey	RQ LKey，由 EE 提供
Entry_LKey	散列表 LKey，由 EE 写回
Entry_VA	散列表地址，由 EE 写回
Entry_Length	散列表长度，由 EE 写回

7.1.6.2 响应格式

表 7.18 接收描述符管理模块响应格式

+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
QPN																								Resp							
VA[31:0]																															
VA[63:32]																															
RKey																															
Length																															

表 7.19 接收描述符管理模块响应说明

字段	说明
Resp	响应码 Valid Entry（8’h00）：有可用 Scatter Entry Invalid Entry（8’h01）：无可用 Scatter Entry

	Invalid WQE（8’h020）：无可用 WQE Invalid Entry 代表的是当前 Entry 数量不足容纳整个数据包，而 Invalid WQE 代表的是 Resource Not Ready，两者返回的错误码是不同的。
QPN	需要处理的 QP 号

7.2 模块之间传递的元数据

7.2.1 DoorbellProcessing 与 WQEScheduler

表 7.20 DoorbellProcessing 传递给 WQEScheduler 的元数据格式

+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
Source QPN																								ST		F		OpCode			
Destination QPN																								Reserved							
Protection Domain																															
LKey																															
P_Key																															
Reserved								PMTU								WQE Size															

表 7.21 DoorbellProcessing 传递给 WQEScheduler 的元数据字段说明

字段	说明
OpCode	用户指定的请求操作码
F	Fence，与门铃中的含义一致
Source QPN	源 QP 号
Destination	目的 QP 号（RC/UC 的目的 QP 号由上下文取得，UD 由描述符携带）
Protection Domain	保护域
LKey	内存访问的校验码
P_Key	Partition Key，生成 BTH 包头时需要用到
WQE Size	待处理的描述符大小
PMTU	Path Maximum Transfer Unit，用于消息分段 MTU_256（8’h00）：分段大小为 256B MTU_512（8’h01）：分段大小为 512B MTU_1024（8’h02）：分段大小为 1024B MTU_2048（8’h03）：分段大小为 2048B MTU_4096（8’h04）：分段大小为 4096B

7.2.2 WQEScheduler 与 WQEParser

元数据格式与 7.2.1 保持一致。

7.2.3 WQEParser 与 DataPack

WQEParser 向 DataPack 传递的元数据有两种：1.公有元数据，即所有请求都会用到的元数据；2.操作相关的元数据，即特定操作才会用到的元数据。

7.2.3.1 公有元数据

表 7.22 WQEParser 传递给 DataPack 的公有元数据格式

+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
PMTU								LegalEntryNum								ReqState				Flags				ST			OpCode				
Source QPN																								PMTU							
Reserved								Destination QPN																							
MsgSize																															
Q_Key																															
Protection Domain																															
Partition Key																															
Immediate Data																															

表 7.23 DoorbellProcessing 传递给 WQEScheduler 的公有元数据字段说明

字段	说明
OpCode	用户指定的请求操作码
ST	Service Type，指定当前 QP 服务类型 Reliable Connection（3'b000） Unreliable Connection（3'b001） Unreliable Datagram（3'b010）
Flags	请求标志位 Flags[0]：Fence 位 Flags[1]：Inline 标志位 Flags[2~7]：Reserved
ReqState	当前描述符执行状态 QP_STATE_ERR（4'b0001）：QP 状态错误 QP_OPCODE_ERR（4'b0010）：当前操作码类型错误 QP_LOCAL_ACCESS_ERR（4'b0100）：本地内存访问错误 QP_NORMAL（4'b0000）：正常请求
LegalEntryNum	有效的 Scatter Gather Entry 数目。对于 RDMA Read，表示携带的 Scatter Entry 数量；对于 Send/Write，表示已经发起的从内存中读取数据的

	Gather Entry 数量
PMTU	Path Maximum Transfer Unit MTU_256（8'h00）：分段大小为 256B MTU_512（8'h01）：分段大小为 512B MTU_1024（8'h02）：分段大小为 1024B MTU_2048（8'h03）：分段大小为 2048B MTU_4096（8'h04）：分段大小为 4096B
Source QPN	源 QP 号
Destination QPN	目的 QP 号（RC/UC 的目的 QP 号由上下文取得，UD 由描述符携带）
MsgSize	消息大小，上限为 2GB
Q_Key	用于 UD 服务校验
Protection Domain	保护域，用于访问内存时使用
Partition Key	生成 BTH 时使用
Immediate Data	立即数，将于 IMMDT 包头中携带

7.2.3.2 操作相关元数据

散列表数据

表 7.24 WQEParser 传递给 DataPack 的散列表项格式

+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																															
LKey																															
Virtual Address																															
Virtual Address																															

表 7.25 DoorbellProcessing 传递给 WQEScheduler 的公有元数据字段说明

字段	说明
Length	Memory Region 长度
LKey	Memory Region 访问校验码
Virtual Address	Memory Region 虚拟地址

RDMA 操作相关元数据

表 7.26 WQEParser 传递给 DataPack 的 RDMA 操作元数据格式

+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
Virtual Address																															
Virtual Address																															
RKey																															

Reserved

表 7.27 DoorbellProcessing 传递给 WQEScheduler 的 RDMA 操作元数据字段说明

字段	说明
Virtual Address	远端内存虚拟地址
RKey	远端内存访问校验码

表 7.28 WQEParser 传递给 DataPack 的原子操作元数据格式

+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
Swap/Add Data																															
Swap/Add Data																															
Compare																															
Compare																															

表 7.29 DoorbellProcessing 传递给 WQEScheduler 的原子操作元数据字段说明

字段	说明
Swap/Add Data	原子操作要用到的操作数
Compare	CmpAndSwp 操作用到的比较数

7.2.4 DataPack 与 RequesterTransControl

与 7.2.3 小节保持一致。

7.2.5 RequesterTransControl 与 RequesterRecvControl

表 7.30 RequesterTransControl 传递给 RequesterRecvControl 的元数据格式

+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
ErrType								QPN																							

表 7.31 RequesterTransControl 传递给 RequesterRecvControl 的元数据说明

字段	说明
QPN	发生错误的 QP 编号
ErrType	错误类型码：

7.2.6 RequesterTransControl 与 ReqPktGen

两个模块之间传递的包头元数据如下表所示。

表 7.32 RequesterTransControl 传递给 ReqPktGen 的包头元数据类型说明

请求包/响应包	操作类型	包头组成	包头长度（B）
请求包	RC Send	<BTH>	12
	RC Send with Immediate	<BTH, ImmDt>	16
	RDMA Write First	<BTH, RETH>	28
	RDMA Middle/Last/Only	<BTH>	12
	RDMA Last with Imm	<BTH, Immdt>	16
	RDMA Write Only with Immediate	<BTH, RETH, ImmDt>	32
	RDMA Read	<BTH, RETH>	28
	Atomics	<BTH, AtomicsETH>	40
	UD Send	<BTH, DETH>	20
	UD Send with Immediate	<BTH, DETH, ImmDt>	24

包头总线向最长的原子操作包头对齐，数据包长度信息利用了 BTH 中的保留字段，因此，请求包包头元数据的长度为 40 字节，共 320 比特。

7.2.7 RequesterRecvControl 与 ReqPktGen

两个模块之间传递的包头元数据如下表所示。

表 7.33 RequesterRecvControl 传递给 ReqPktGen 的包头元数据类型说明

请求包/响应包	操作类型	包头组成	包头长度（B）
请求包	RC Send	<BTH>	12
	RC Send with Immediate	<BTH, ImmDt>	16
	RDMA Write First	<BTH, RETH>	28
	RDMA Middle/Last/Only	<BTH>	12
	RDMA Last with Imm	<BTH, Immdt>	16
	RDMA Write Only with Immediate	<BTH, RETH, ImmDt>	32
	RDMA Read	<BTH, RETH>	28
	Atomics	<BTH, AtomicsETH>	40
	UD Send	<BTH, DETH>	20
	UD Send with Immediate	<BTH, DETH, ImmDt>	24

包头总线向最长的原子操作包头对齐，数据包长度信息利用了 BTH 中的保留字段，因此，请求包包头元数据的长度为 40 字节，共 320 比特。

7.2.8 HeaderParser 与 RequesterRecvControl

两个模块之间传递的包头元数据如下表所示。

表 7.34 HeaderParser 传递给 RequesterRecvControl 的包头元数据类型说明

请求方/响应方	操作类型	包头组成	包头长度（B）
响应包	ACK	<BTH, AETH>	20
	Read Resp First/Only/Last	<BTH, AETH>	20
	Read Resp Middle	<BTH>	12
	Atomic Response	<BTH, AETH, AomicAck>	28

包头总线向最长的原子操作响应包头对齐，同时，需要增加 2 字节的长度描述信息，因此，请求包包头元数据的长度为 30 字节，共 240 比特。

7.2.9 HeaderParser 与 ExecutionEngine

与 0 小节保持一致。

7.2.10 ExecutionEngine 与 RespPktGen

表 7.35 ExecutionEngine 传递给 RespPktGen 的元数据格式

+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
OpCode								S	M	PC		TVer				P_Key															
F	B	Reserve						DestQP																							
A	Reserve							PSN																							
Reserved																PMTU															
MsgSize																															
Syndrome								MSN																							

表 7.36 ExecutionEngine 传递给 RespPktGen 的元数据说明

字段	说明
Type	响应类型 ACK（8’h01）：包括 ACK 和 NAK，具体类型由 Syndrome 指示 Read（8’b10）：指示需要生成 Read 响应包
PMTU	指定分包大小
MSN	已经处理完的消息编号
Syndrome	指示当前 ACK 包的子类型
MsgSize	RDMA Read 返回的消息长度

7.3 各模块用到的上下文信息及获取方式

表 7.37 各模块与上下文信息对照

模块	上下文字段	读/写	获取方式
DoorbellProcessing	Source QPN	只读	上下文读取
	Destination QPN	只读	上下文读取
	Protection Domain	只读	上下文读取
	LKey	只读	上下文读取
	Partition Key	只读	上下文读取
WQEScheduler	/	/	/
WQEParser	LKey	只读	元数据传递
	Protection Domain	只读	元数据传递
	Source QPN	只读	元数据传递
	QP State	只读	上下文读取
DataPack	/	/	/
RequesterTransControl	Source QPN	只读	元数据传递
	Destination QPN	只读	元数据传递
	Protection Domain	只读	元数据传递
	CQ LKey	只读	上下文读取
	Partition Key	只读	元数据传递
	QP State	只读	上下文读取
	Next PSN	读写	上下文读写
	Retry Count	只读	上下文读取
	Transport Timeout	只读	上下文读取
RequesterRecvControl	QP State	读写	上下文读写
	UnAcked PSN	读写	上下文读写
	CQ Lkey	只读	上下文读取
ReqPktGen	/	/	/
TimerControl	/	/	/
MultiQueue	/	/	/
HeaderParser	/	/	/
RespPktGen	/	/	/
ExecutionEngine	RQ LKey	只读	上下文读取
	CQ Lkey	只读	上下文读取

	QP State	读写	上下文读写
	Protection Domain	只读	上下文读取
	Partition Key	只读	上下文读取
	Expected PSN	读写	上下文读写
	PMTU	只读	上下文读取
接收队列描述符管理	RQ LKey	只读	上下文读写
	Protection Domain	只读	上下文读写
写地址记录表	/	/	/
位宽转换	/	/	/

表 7.38 Doorbell 读取 CxtMgt 命令格式

+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																															
Reserved																															
Reserved																															
Type				OpCode				QPN																							

表 7.39 Doorbell 读取 CxtMgt 响应格式

+3								+2								+1								+0							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
DstQPN																								Reserved						ST	
PMTU																PKey															
LKey																															
Protection Domain																															

表 7.40 RequesterTransControl 写入 CxtMgt 数据格式

+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
NextPSN																								Reserved						QPState	

表 7.41 ExecutionEngine 读取 CxtMgt 数据格式

+3								+2								+1								+0							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Expected PSN																								RNRTimer						QPState	

LKey	
Protection Domain	
PMTU	PKey

表 7.42 RequesterRecvControl 读取 CxtMgt 数据格式

Reserved+3								+2								+1								+0							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
UnAcked PSN																								Reserved						QPState	
NextPSN																								Reserved							
CQ LKey																															