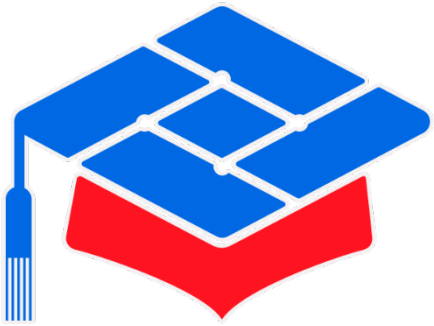


第八届

全国大学生集成电路创新创业大赛

报告类型：设计报告

参赛杯赛：中科芯杯

作品名称：基于 SRAM 的高速共享缓存模块设计

队伍编号：CICC5303

团队名称：野原芯之助队

目录

1 概述 4

1.1 设计背景 4

1.2 设计指标要求 4

2 详细设计说明 6

2.1 Port_in 模块 7

接口信号说明 7

模块设计 8

2.2 Port_mux 模块 9

接口信号说明 9

模块设计 10

2.3 Padding 模块 11

接口信号说明 11

模块设计 12

2.4 Switch 模块 12

2.4.1 Switch_pre 模块 12

接口信号说明 12

模块设计 13

2.4.2 Switch_core 模块 14

接口信号说明 14

模块设计 15

自由指针队列 15

写入状态机 16

队列控制器 17

读出状态机 19

2.5 Port_out 模块 20

接口信号说明 20

模块设计 20

3 验证与测试 21

3.1 验证方法与思路.....21
3.2 验证结果分析.....23
3.2.1 模块级功能仿真.....23
3.2.2 系统级仿真.....26
3.2.3 共享缓存管理模块的 FPGA 设计.....27
4 设计亮点及未来工作.....30
4.1 设计亮点.....30
4.2 未来工作.....30
参考文献.....31

1 概述

1.1 设计背景

随着网络技术的迅猛发展，互联网流量爆炸式增长，企业和家庭对网络带宽的需求不断提升。在这种趋势下，40G 与 100G 网络时代已经到来，要求交换机和路由器等网络设备具备极高的数据存储能力。然而，网络设备在数据包处理过程中，存储管理和调度占用了大量时间，低速缓存能力成为制约网络处理器性能提升的瓶颈。

本赛题旨在设计一款支持高速数据存储的 SRAM 管理模块，以解决存储器资源受数据包长度和数据通道数量限制的问题。通过内存回收和动态调整空间利用率，提升存储效率，并增强数据校验和纠错能力。

1.2 设计指标要求

对赛题要求进行整理，可分为以下 7 个方面：

SRAM 管理

- 管理不少于 32 块 256K bit 的 SRAM 单元，总容量达到 8M bit。
- 支持时钟频率>250MHz，满足高速缓存需求。

端口支持

- 支持 16 个端口同时读写缓存。
- 每端口数据传输带宽达到 1Gbps。
- 每端口支持 8 个优先级队列，按队列缓存数据。
- 端口时序如图 1 所示

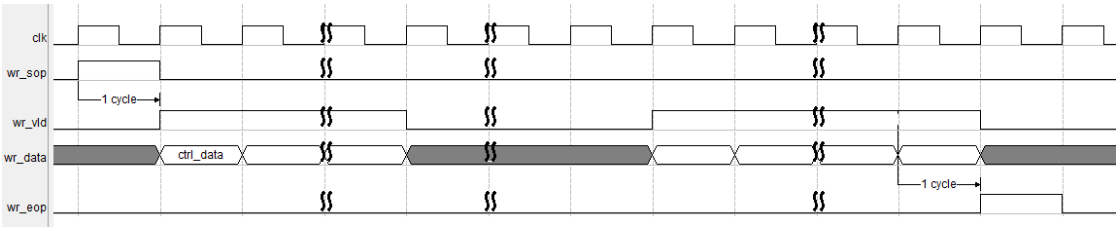


图 1.1 时序图

缓存与调度

- 支持按包缓存和调度数据，数据包长度范围为 64 至 1024 字节，每包帧格式包含控制帧部分（如目的地址、优先级），如图 2 所示

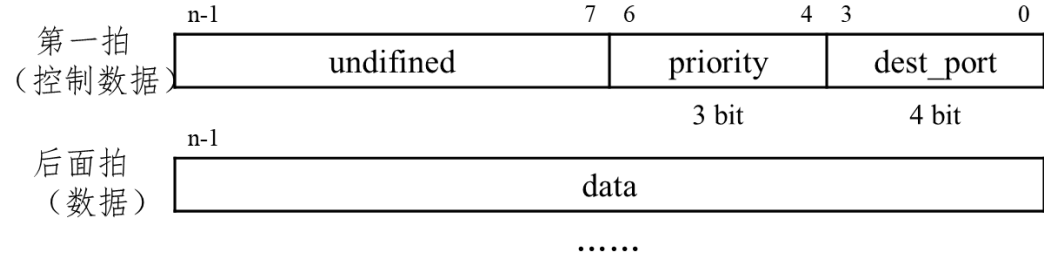


图 1.2 帧格式

- 多端口、多队列动态共享缓存，只要缓存未满，任意端口的任意队列数据都可写入缓存。

并行操作

- 允许多个端口同时进行写入和写出操作，每个端口独立工作，互不影响。

写出与调度

- 写出端口由目的端口域决定，支持单播和多播。
- 支持按 QoS（严格优先级、加权轮询（WRR））调度。

内存管理

- 支持内存回收机制，对已读出的内存空间进行重分配。

数据校验

- 提供数据校验功能以确保数据正确性与完整性。

2 详细设计说明

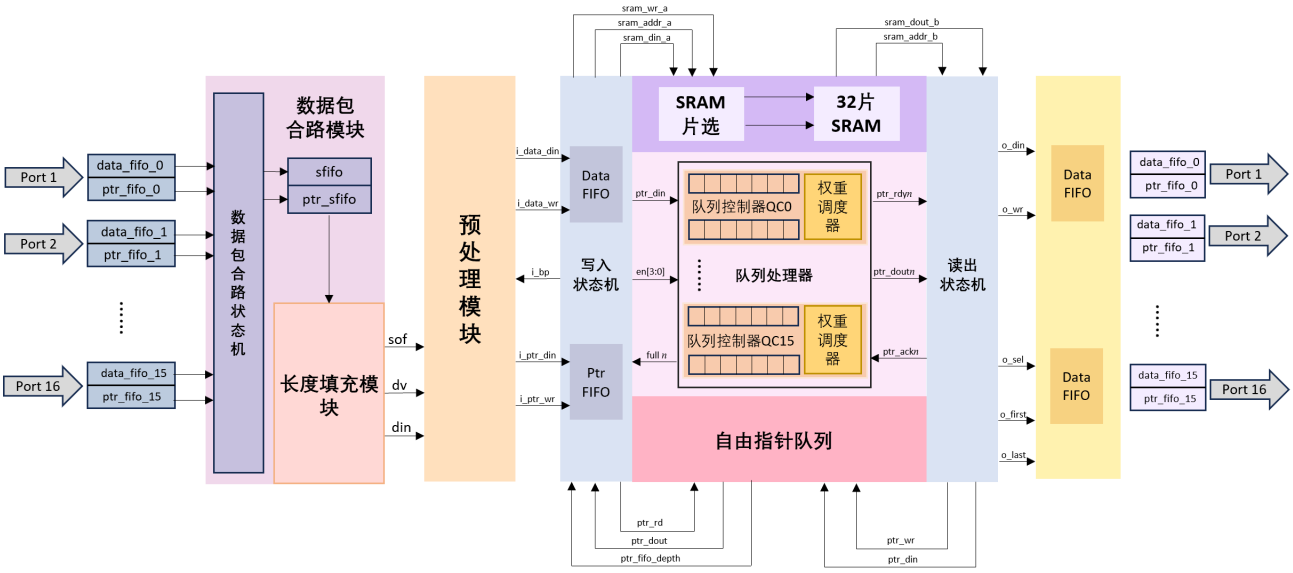


图 2.1 整体架构图 [1]

整体设计框架如图 2.1 所示。数据从端口进入，经过数据合路、长度填充，在预处理模块中被切割成定长信元。然后进入交换核心结构，在自由指针队列、写入状态机和队列处理器的共同协作下，存入 SRAM 共享缓存区。最后，读出状态机将信元从 SRAM 中读出，重组成完整数据包，去除控制帧，并送往目的端口。

接下来将详细讲述各个模块的设计思路。其中，在接口信号说明部分，为方便阅读，与前级模块相连的信号用紫色表示，与后级模块相连的信号用青色表示，其余信号则用黑色表示。

[1] 由于图中篇幅限制，部分信号名称有所简化或更改，但不影响理解。

2.1 Port_in 模块

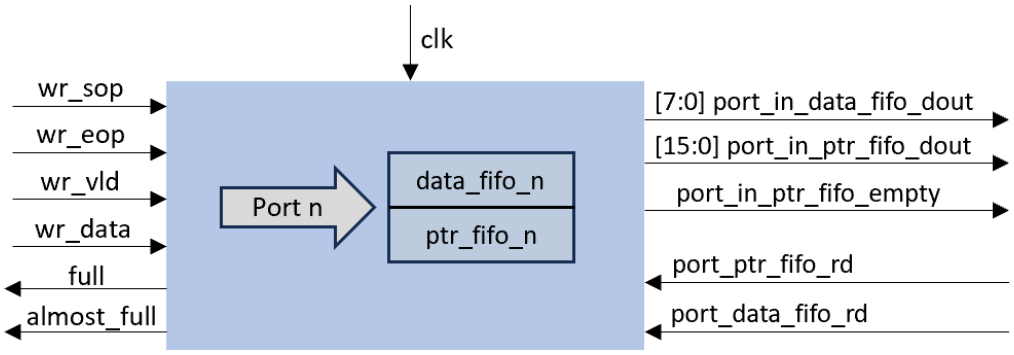


图 2.2 Port_in 模块设计

对于 Port_in 模块，应当满足以下指标要求

a) 能按照赛题要求的时序规范读入数据包。当 wr_sop 拉高一个时钟周期后，如果 wr_vld 也拉高，则开始读入数据包。如果 wr_vld 变低，应当实现数据包传输暂停功能。当 wr_eop 变高电平且 wr_vld 同时变为低电平时，代表一个数据包传输完成。

b) 能够识别数据包的长度。对于长度不在 64-1024 字节的非法长度数据包，应当有异常判别信息，并设计相应的异常应对程序。

c) 端口数据传输带宽达到 1Gbps。

d) 能够实现数据校验功能。

接口信号说明

A 输入信号

rx_clk：1 位。

wr_vld：1 位。数据有效 （原 rx_dv）

[15:0] wr_data：传入数据，位宽更新为 16 位 （原 4 位 rx_d）

wr_vld：写入数据有效信号

wr_data：传入的数据包

wr_sop：数据包开始信号

wr_eop：数据包结束信号

port_data_fifo_rd：后级合路模块传来的 Data FIFO 读信号

port_ptr_fifo_rd：后级合路模块传来的 Ptr FIFO 读信号

B 输出信号

full：资源不足信号

almost_full：资源即将不足信号

[7:0] port_in_data_fifo_dout：写入 Data FIFO 的数据，位宽为 16 位。

[15:0] port_in_ptr_fifo_dout：写入 Ptr FIFO，位宽为 16 位。在本设计中指针均为 16 位。

port_in_ptr_fifo_empty：指针 FIFO 的空信号。

模块设计

该模块集成了一套状态机控制逻辑，以精确管理数据包从接收起始到结束的整个处理流程。

首先，通过检测外部输入的起始(wr_sop)和结束(wr_eop)信号，以及数据有效(wr_vld)标志，模块启动一个精细设计的四状态机来指导操作序列，当检测到 wr_sop 信号为高并且 wr_vld 信号为高时开始接收数据，在接收时若 wr_vld 信号变为低则停止接收，直至 wr_vld 重新变为高时再开始接收。当 wr_eop 信号为高时，表面数据包最后一位已经接收完毕，停止接收。

在接收过程中，使用一个接收计数器(nib_cnt)跟踪接收到的数据位数，该计数器会在每个数据包开始时清零，并根据数据传输的暂停或继续动态调整。数据校验环节引入了 CRC 算法，利用 crc32_8023 模块对数据包执行完整性检查，确保数据无误，其校验结果与一个预定值(CRC_RESULT_VALUE)比对，以标记数据包的校验状态。此外，模块还判断数据包长度的合法性，当长度落于 64~1024 字节之外，长度错误标记会被置为高，进一步增强了错误检测能力。

循环冗余校验（CRC）是用于检测数据损坏的错误检测码。发送数据时，会根据数据内容生成简短的校验和：发送方用发送数据的二进制多项式 t(x) 除以 g(x) ，得到余数 y(x) 作为 CRC 校验码。CRC32 的多项式生成多项式为

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}$$

$$+x^{5}+x^{4}+x^{2}+x+1$$

校验时，以计算的校正结果是否为 0 为据，判断数据帧是否出错。并将其与数据一起发送。接收数据时，将再次生成校验和并将其与发送的校验和进行比较。如果两者相等，则没有数据损坏。

为了高效管理数据流，设计中集成了两个 FIFO（先入先出队列）：数据 FIFO (u_data_fifo) 和指针 FIFO (u_ptr_fifo)。数据 FIFO 负责暂存接收到的有效数据，其写入操作受控于内部产生的写使能信号，同时监控 FIFO 的满或接近满状态，以防数据溢出。而指针 FIFO 则用于存储每个数据包的处理元数据，包括数据包的长度、长度合法性标志以及 CRC 校验通过与否的信息，这些信息对于后续处理至关重要。

2.2 Port_mux 模块

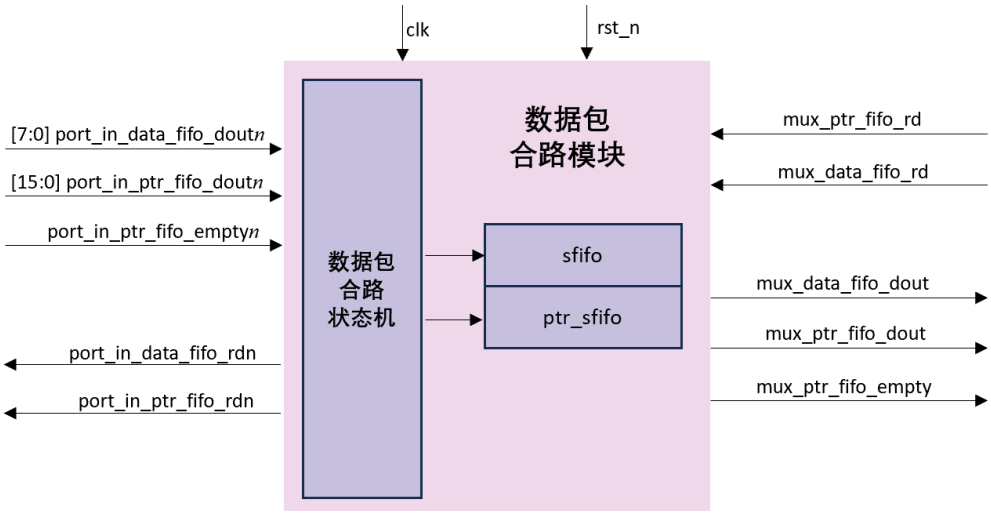


图 2.3 Port_mux 模块设计

接口信号说明

A. 输入信号

clk：时钟信号

rstn：复位信号

[7:0] port_in_data_fifo_dout*n*：每个端口的数据 FIFO 中所存的 8 位宽数据（n=0~15）

[15:0] port_in_ptr_fifo_dout*n*：每个端口的指针 FIFO 中所存的 16 位宽指针（n=0~15）

port_in_ptr_fifo_empty*n*：每个端口的指针 FIFO 空信号

mux_data_fifo_rd：Padding 模块传来的合路数据 FIFO 的读数据信号

mux_ptr_fifo_rd：Padding 模块传来的合路指针 FIFO 的读指针信号

B. 输出信号

port_in_data_fifo_rdn：端口 n 的数据缓存读使能信号

port_in_ptr_fifo_rdn：端口 n 的指针缓存读使能信号

mux_data_fifo_dout：合路后的数据缓存

mux_ptr_fifo_dout：合路后的指针缓存

mux_ptr_fifo_empty：合路后指针 FIFO 空信号

模块设计

该模块的主要功能是对 Port_in 的接收队列进行轮询，从接收队列中读取指针，并根据指针信息将数据包写入合路 FIFO。此外，如果指针中包含错误信息，数据包将被丢弃。

该设计采用公平轮询策略，所有端口具有相同优先级。合路状态机通过设置公平轮询指示寄存器 RR 来确保对各端口队列的公平处理。当 RR 为 0 时，如果 16 个端口队列中至少有一个非空，状态机首先检查端口队列 0 是否有数据包，若有则进行处理，否则依次检查队列 1 到队列 15 是否有数据包。处理完成后，RR 的值加 1，状态机返回空闲状态。然后，模块检查队列 1 是否有数据包，若有则处理，否则依次检查队列 2 到队列 15 和队列 0，处理完成后，RR 的值由 1 变为 2，即每完成一次数据包接收操作后，RR 的值增加 1，指示下一次首先判断的队列。当 RR 值达到 15 时，本轮处理结束后 RR 重置为 0。此设计确保了对所有端口队列的公平对待。

当轮询发现端口 n（n 的取值范围为 0 至 15）的 rx_ptr_fifo_n 非空时，首先从 rx_ptr_fifo_n 中读取一个指针。该指针包含当前接口队列中数据 FIFO 的数据包长度和错误指示信息。如果数据包无误，interface_mux 会根据指针提供的长度值从当前数据 FIFO 中读取数据包，并将其写入 interface_mux 内部的接收队列（即 interface_mux 和 frame_process 之间的接口队列）。如果数据包有误，数据包将被读取并丢弃。

数据包写入 interface_mux 内部接收队列的数据 FIFO 后，需要生成该数据包的指针并写入接收队列的指针 FIFO，该指针中包含当前数据包的长度值。由于 interface_mux 在将数据包写入接收队列时，会丢弃其 CRC-32 校验值，因此写入的数据包长度比读取的数据包长度少 4 字节。

2.3 Padding 模块

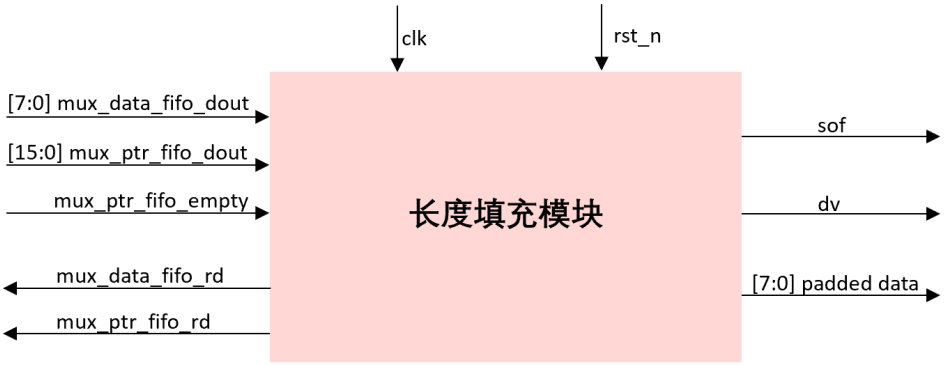


图 2.4 Padding 模块设计

接口信号说明

A 输入信号

clk：时钟信号

rstn：复位信号

[7:0] mux_data_fifo_dout：合路后的数据，位宽为 8

[15:0] mux_ptr_fifo_dout：合路指针

mux_ptr_fifo_empty：合路指针空信号

B 输出信号

mux_data_fifo_rd：合路数据 FIFO 读信号

mux_ptr_fifo_rd：合路指针 FIFO 读信号

sof：数据包开始信号

dv：数据有效信号

[7:0] padded_data：填充后的数据，位宽为 8

模块设计

完成合路后，需要对数据包进行进一步处理，提取出目的端口信息，并将数据包的长度填充到 64 字节的整数倍，便于后续模块将数据包切分为大小为 64 字节的定长信元。

2.4 Switch 模块

2.4.1 Switch_pre 模块

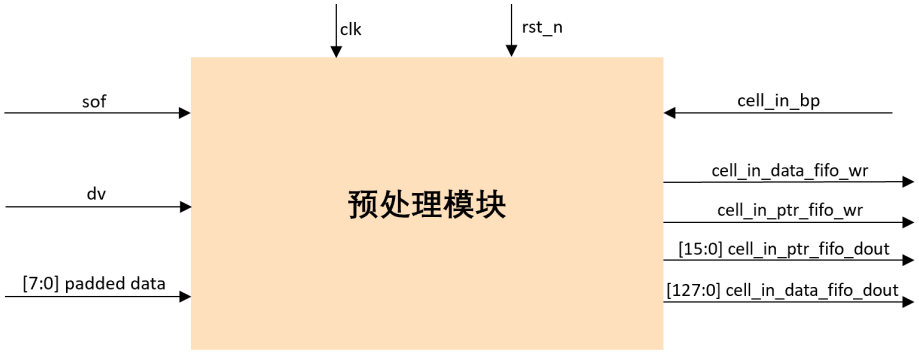


图 2.5 switch_pre 模块设计

接口信号说明

A 输入信号

clk：时钟信号

rstn：复位信号

sof：数据包开始信号

dv：数据有效信号

[7:0] padded_din：填充模块传来的数据

cell_in_bp：给填充模块的反压信号

B 输出信号

[127:0] cell_in_data_fifo_dout：经过位宽变换、数据包切割的数据

cell_in_data_fifo_wr：数据 FIFO 写使能信号

[15:0]　cell_in_ptr_fifo_dout：记录变换后信息的指针信号

cell_in_ptr_fifo_wr：指针 FIFO 写使能信号

模块设计

Padding 模块向该模块传入 3 个信号：用于指示数据包开始的 sof、有效信号 dv 和数据包 din。其中 din 位宽为 8 位，格式如下图所示：

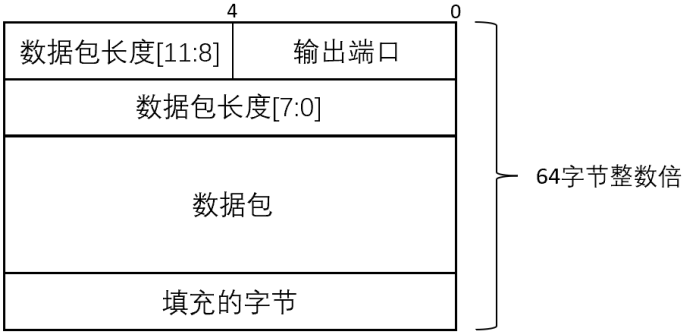


图 2.6 传入 switch_pre 的数据包结构

模块将接收到的数据包位宽变为 128 位，便于提高处理速率，增大交换容量。此外，还将数据包切割成多个大小为 64 字节的定长信元，每个信元由 4 拍组成。经过该模块处理后的数据包结构如图 2.7 所示

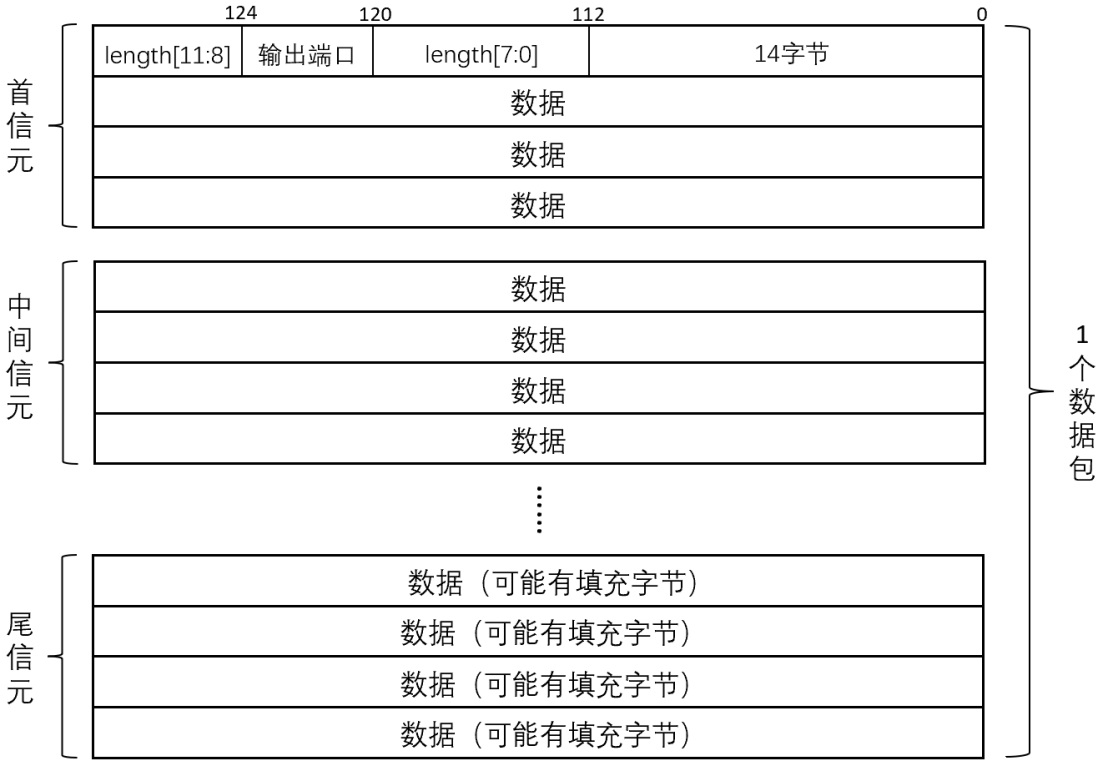


图 2.7 经过 Switch_pre 处理的数据包结构

将变换后的信息写入 i_cell_ptr_fifo_dout. 该指针的格式为:

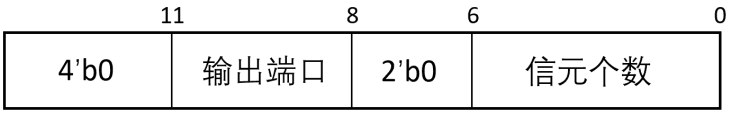


图 2.8 指针格式

2.4.2 Switch_core 模块

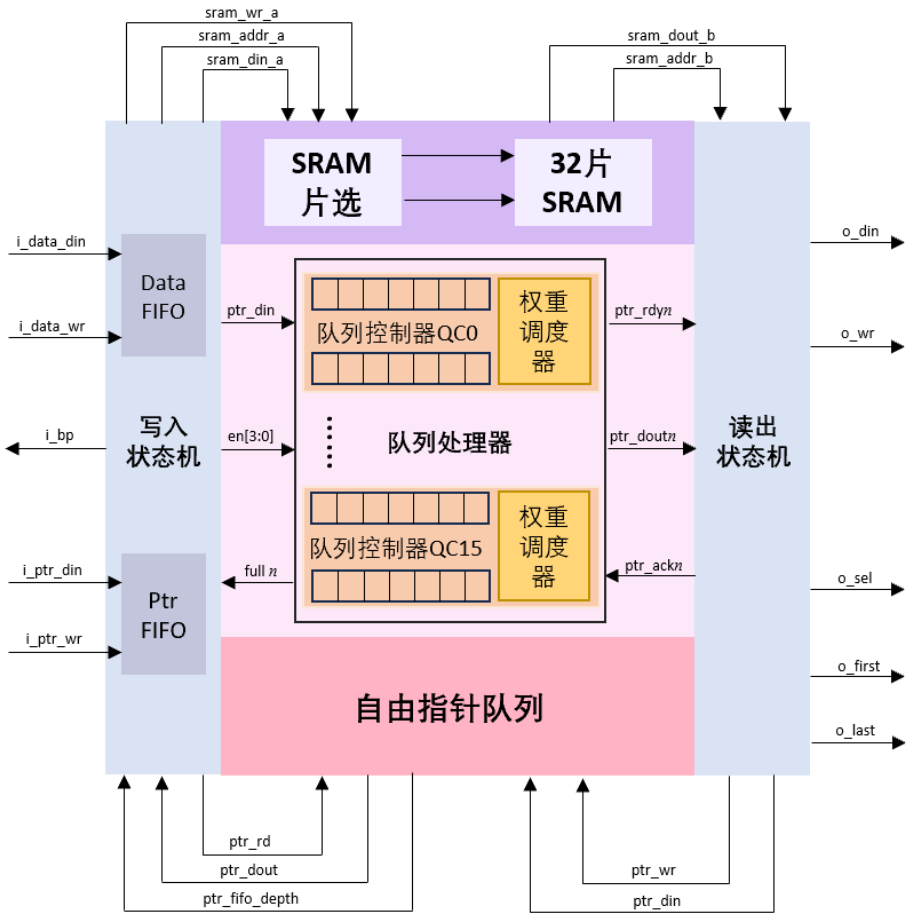


图 2.9 交换核心结构

接口信号说明

A 输入信号

clk：时钟信号

rstn：复位信号

[127:0] cell_in_data_fifo_din：经过位宽变换、数据包切割的数据

cell_in_data_fifo_wr：数据 FIFO 写使能信号

[15:0] cell_in_ptr_fifo_din：记录变换后信息的指针信号

cell_in_ptr_fifo_wr：指针 FIFO 写使能信号

[3:0]　　cell_out_bp：给后级 switch_core 的反压信号

B 输出信号

cell_in_bp：给前级 switch_pre 的反压信号

cell_out_fifo_wr：输出数据写使能

[3:0] cell_out_fifo_sel：FIFO 片选信号

[127:0] cell_out_fifo_din：输出数据

cell_out_first：首信元指示信号

cell_out_last：尾信元指示信号

模块设计

定长信元到达 switch_core 后，将被写入数据缓冲区。自由指针队列给出可用的 SRAM 存储区域编号。写入状态机根据此编号，生成相应 SRAM 中的对应地址，将数据包以信元为单位，按照此地址存入 SRAM。同时，写入状态机生成一个记录数据包存储信息（SRAM 地址、首尾信元标志等）的指针，并根据数据包的优先级字段和目的端口域，将指针交付给相应端口的队列控制器中相应的优先级队列。

自由指针队列

图 2.10 自由指针队列设计框图

本 IP 设计方案中，数据包以信元为单位存储在 SRAM 中。无论去往哪个端口的数据包都可以存到 SRAM 共享缓存区中。一片 SRAM 大小为 256 K bit，一个信元大小为 64 Byte，一片 SRAM 中可存放 512 个信元。故将每片 SRAM 划分成 512 个存储区域，则 32 片 SRAM 可分为 16384 个存储区域，对应一个深度同样为 16384 的自由指针队列，指针值为存储区域编号，范围为 0~16383。在初始

化过程中，所有 SRAM 均空闲，故将指针值写入自由指针队列。由于数据包此时的位宽为 128 位，SRAM 的位宽也为 128 位，需要 4 个 SRAM 单位存储一个信元（即一个 SRAM 存储区域）。因此，除了使用自由指针的存储区域编号，还应当在低位加上 2 位计数值（00~11），拼接后的信号才是真正使用的 SRAM 地址。

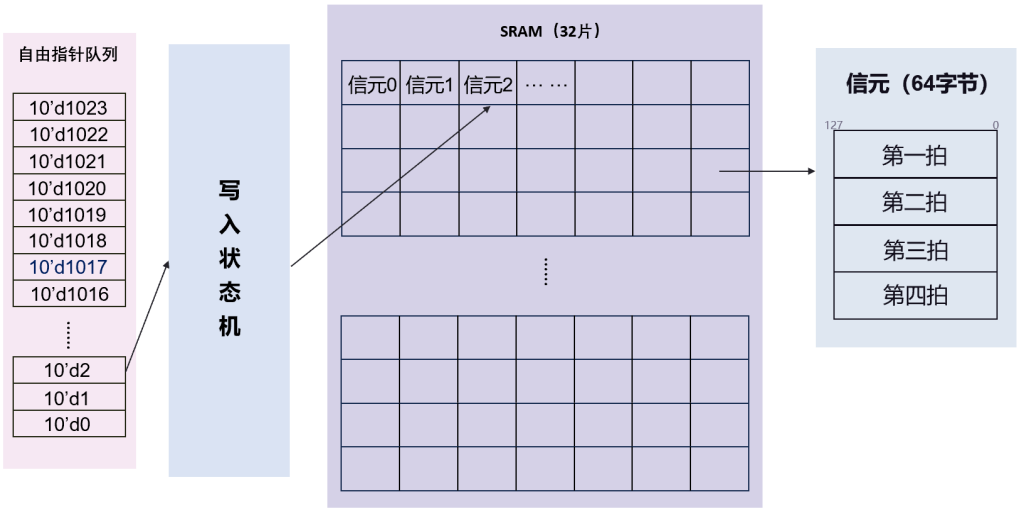


图 2.11 写入 SRAM

写入状态机

写入状态机接收并缓存由前级模块 switch_pre 传入的信元，主要完成申请自由指针、将信元存储进 SRAM，以及自由指针写入队列控制器等工作。

状态 0 为初始状态，将检查指针 FIFO 是否有数据且队列控制器和空闲队列是否准备好。如果条件满足，读取一个数据单元和指针，并初始化相关变量。

状态 1 为写入 SRAM 第一阶段。减少剩余数据单元计数，并将当前数据单元地址和内容写入 SRAM。同时，更新队列控制器的指针信息，标记写入操作。

在第二阶段到第四阶段中，逐步完成数据单元写入 SRAM 的过程，更新相关计数器。如果还有剩余信元，返回到第一阶段继续写入，否则结束本次写入过程，返回初始状态。

其中，传给队列控制器的指针信号将在队列控制器模块设计中具体阐述。

队列控制器

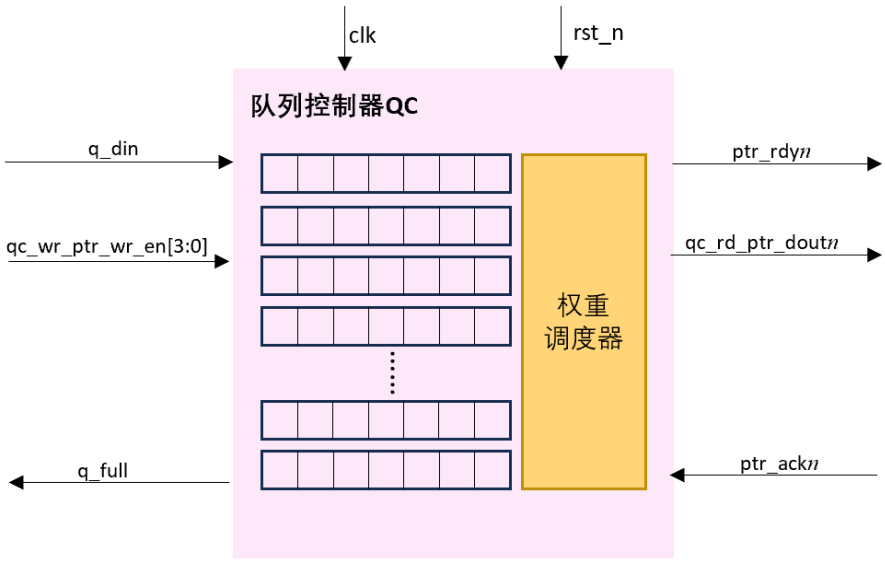


图 2.12 队列控制器设计

该模块采用链表结构来实现先入先出的逻辑队列。链表存储区由一片 SRAM 实现，此外定义了一组内部寄存器实现链表结构，包括 head 寄存器（指向链表首地址的寄存器）、tail 寄存器（指向链表尾地址的寄存器）、信元深度计数器和数据帧深度计数器。链表存储区的深度与自由指针的深度相同。head 寄存器始终指向链表的首部，而 tail 寄存器始终指向链表的尾部。信元深度计数器记录当前逻辑队列中的信元数量，数据帧深度计数器记录当前逻辑队列中的完整数据帧数量。

每次写入操作信元深度计数器都要加 1；每次读出操作信元深度计数器都要减 1。如果写入的信元是一个数据包的尾信元，则数据包深度计数器加 1；如果读出的信元是一个数据包的尾信元，则数据包深度计数器加 1。

每个链表存储单元除了存储指针值，还需要记录该指针指向信元的首、尾状态信息，以表明该指针所指向的信元是一个数据包的首信元、中间信元，还是尾信元。因此，在指针的取值范围对应的位宽的基础上增加 2 位。最终，链表存储区位宽为 16 位，位 15 是尾信元指示位，置 1 是表示是数据包尾信元；位 14 是首信元指示位，置 1 是表示是数据包首信元；位 9~位 13 为片选信号位，由写入状态机给出；位 0~位 8 是自由指针队列给出的存储区域编号。位 0~位 13 结合在

一起为 SRAM 存储地址。指针格式如图 2.13 所示，该指针由写入状态机生成并传入。

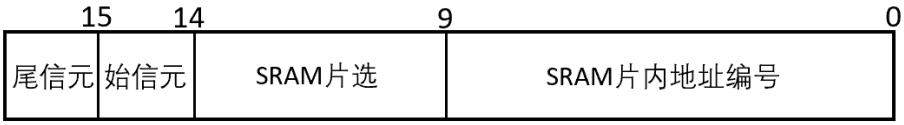


图 2.13 指针结构

每次写入操作信元深度计数器都要加 1；每次读出操作信元深度计数器都要减 1。如果写入的信元是一个数据包的尾信元，则数据包深度计数器加 1；如果读出的信元是一个数据包的尾信元，则数据包深度计数器加 1。

每个输出端口存在具有不同优先级的多个队列，为了实现对不同优先级队列的有效调度，需要设计专用的权重调度器，以保证不同优先级的队列得到不同的输出带宽，从而保证高优先级业务的服务质量．本文设计的权重调度器在 WRR 调度算法的基础上加以改编、创新，其基本结构如图 2.14 所示。

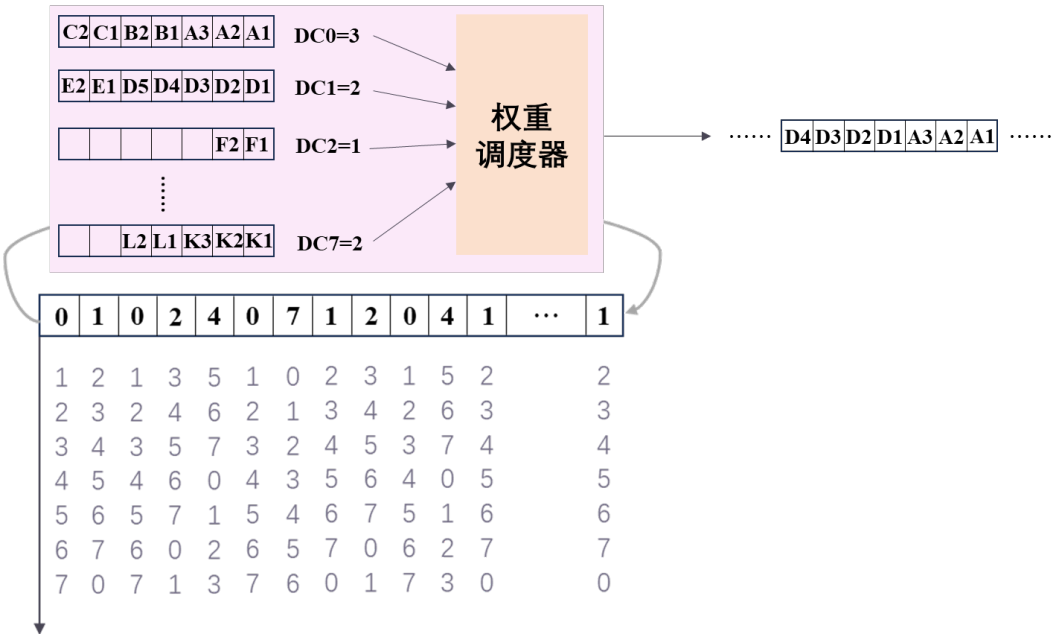


图 2.14 权重调度器设计方案

如上图示，每个队列控制器内包括 8 个具有不同优先级的队列，队列中的每个方格代表一个内部信元。相同的字母编号代表同一数据包。可见，一个数据包通常包括数量不等的多个信元。在每个优先级队列中配置了一个深度计数器 DC，记录着该队列中完整数据包的个数。队列调度器根据预先的配置，依次轮询每个队列，只有当该队列计数器深度不为 0 时，才能够输出一个完整数据包所包括的多个信元。

本设计在每个权重调度器中都配置了一条长度为 M 的权重带。在电路中，由编号为 0 至 M-1 的一组寄存器实现，每个寄存器中写入的是一个对应的优先级编号，另外设计了一个从 0 至 M-1 循环移动的指针，其值用于从 M 个寄存器中选择出对应的寄存器并读出其中存储的优先级编号。如图 2.14 所示，指针值为 0 时，对应优先级配置寄存器中预先配置的是 0，表示需首先查看 Pri0 队列中是否有完整数据包，如果有则调度出其队首的分组包含的所有信元，如果该优先级队列为空，则根据配置，按照从 Pri1-Pri7 的顺序依次查看。当指针值为 M-1 时，优先级配置寄存器中存储的为 1，表示首先查看 Pri1 队列中是否有完整的分组，完成调度后，指针值又恢复为 0 号位的值，开始新一轮调度操作。

该此方案使得配置优先级队列的带宽变得直观而方便。例如在 M 个寄存器中，Pri0 出现了 K 次，那么调度器为其分配的带宽为总带宽的 K/M。由于每个队列首部的数据包长度不同，同一次调度，可能分组的长度差别很大，就单次调度来说存在着带宽分配的不公平性。但考虑到整个交换结构采用共享存储方式，分组存储在片外容量较大的主存储区中，从统计来看，可以认为不同优先级中业务流的分布特征相同，此时这种权重带宽分配方式是统计准确的。

这种权重调度机制具有实现简单，配置灵活的特点，由于每次完整的调度一个分组的所有信元，降低了输出端口重组操作的复杂度，有效降低了资源消耗。

读出状态机

读出状态机用于从 SRAM 中读出信元并传给下一级，主要完成读取队列控制器中的指针、读取信元、归还自由指针等工作。

在初始状态中，检查是否有指针读取请求且输出没有背压。如果条件满足，进入下一状态，准备读取数据单元。状态 1 读取指针信息。从队列控制器中读取指针信息并更新选择信号，然后启动 SRAM 读取操作。

状态 2 至 4 为读取 SRAM 第一阶段到第三阶段，在这一过程中，逐步完成数据单元从 SRAM 的读取过程，更新相关计数器，将读取到的数据单元准备发送到对应的输出端口，并更新队列控制器的计数。

2.5 Port_out 模块

接口信号说明

A 输入信号

clk：时钟信号

rstn：复位信号

cell_out_data_fifo_wr：数据 FIFO 写使能

[127:0] cell_out_data_fifo_din：数据输入

cell_out_data_first：首信元指示位

cell_out_data_last：尾信元指示位

ptr_fifo_rd：指针 FIFO 读信号

data_fifo_rd：数据 FIFO 读信号

B 输出信号

o_cell_data_fifo_bp：数据反压指示

[15:0] ptr_fifo_dout：输出指针

ptr_fifo_empty：指针 FIFO 空信号

[7:0] data_fifo_dout：输出数据

模块设计

该模块的设计方案与电路结构与输入端口基本对偶，在此不做赘述。但相较于输入端口，有一些功能值得关注。

该模块主要实现以下功能：

1. 缓存并重新组合从 switch_core 输出到某个端口的信元，还原原始数据包。
2. 将数据包的位宽转换为 8 位，保持与输如 IP 时的数据包位宽一致。
3. 移除填充字节。
4. 发送 CRC 校验字节。

3 验证与测试

3.1 验证方法与思路

在完成工程的设计部分后，为了确保所做设计的正确性与准确性，需要针对已完成的设计进行仿真、验证，确保系统具有良好的准确性和稳定性，并对系统的吞吐率与资源占用情况进行分析总结。

首先应编写 Testbench，对模块进行 RTL 行为级仿真，之后进行系统级仿真。然后，编写时序约束与引脚约束，进行综合后仿真。在经过布局布线后，可通过进行静态时序分析、查看时序报告、芯片功耗等方式进一步优化设计。整体验证方案如图 3.1 所示。

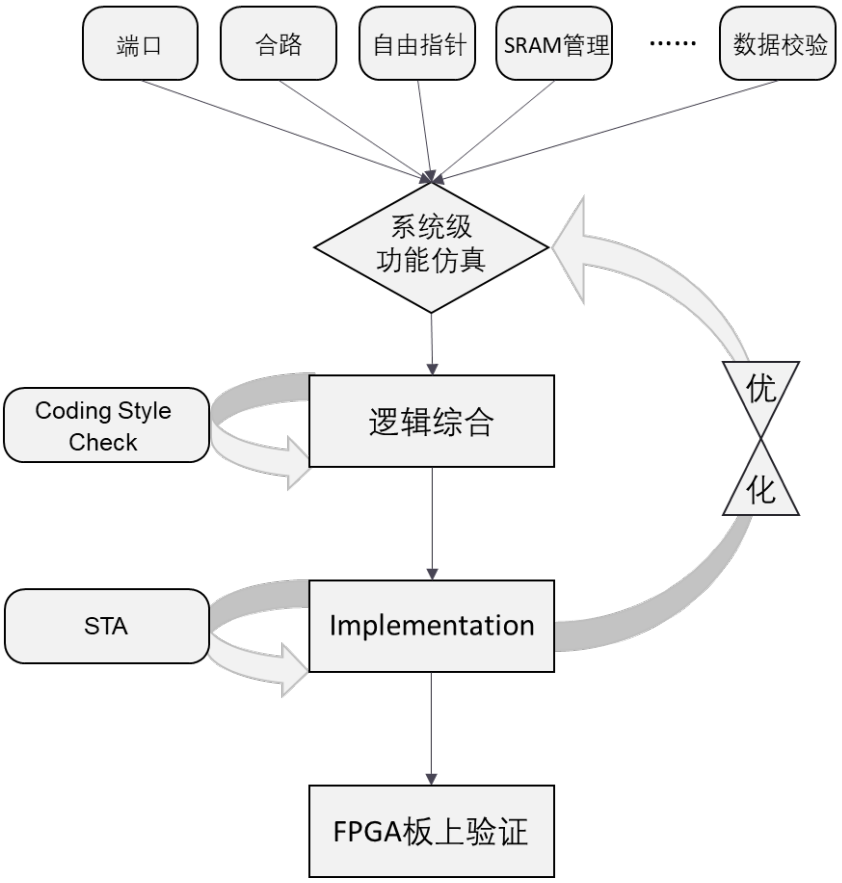


图 3.1 整体验证方案

本文使用的测试平台为 Vivado 2023.1。

模块级功能仿真设计思路

在设计过程中，针对每个分立模块编写 testbench，搭建测试平台，检验模块的设计是否符合要求。在设计的初期阶段不使用特殊底层元件可以提高代码的可读性、可维护性，又可以提高仿真效率，且容易被重用。本文在设计过程中对每一个模块均进行了功能仿真，并全部通过测试。 由于篇幅原因，在此不一一列出，具体可查阅工程文件。在单个模块通过功能性测试后，首先对功能相近、联系紧密的模块进行联合仿真，验证模块间的交互是否满足要求，这样可以得到比系统仿真更多的细节，及时优化重点模块间的交互，增加系统仿真准确率。

数据合路 Port_mux 模块与 Padding 模块的联合仿真

合路模块与数据填充模块关系紧密，共同为数据进入交换核心结构做出充分准备，故进行二者的联合仿真。具体的仿真方案如表 3.1 所示。

表 3.1 数据合路与填充模块的联合仿真方案

验证内容	验证方法及说明	验证结果
正确进行数据合路	在 testbench 中，多端口同时发送数据包，观察合路状态机的输出是否符合公平轮询的结果。	可以正确进行数据合路。
正确进行数据填充	在 testbench 中，发送长度为 64 字节整数倍和长度不为 64 字节整数倍的数据包，观察输出。	能将数据包填充为 64 字节的整数倍。
能够丢弃错误数据包	发送 CRC 校验错误、含有长度错误指示（数据包长度不在 64-1024 字节范围内）的数据包。	错误数据包不会被传入下一级。
能将接受的数据包正确传给下一级	观察填充模块输出是否正确；在 testbench 中，将接收数据包和指针分别写入后级 FIFO 中。	模块输出正确，符合时序规范，能够正确写入后级电路。

Switch 模块的总体验证

Switch 模块将预处理、队列管理器、SRAM 管理和自由指针等单元整合起来，形成一个完整的交换结构，是设计的核心部分。该联合仿真可为系统级测试探路。由于前期已经针对各单元做过测试，该模块仿真时采用黑箱验证，在编写 testbench 在时主要针对该数据包是否能够到达对应的端口。为此，在初始化数据包时需要在数据包的头部补充上数据长度信息以及目的端口信息，模拟传入数

系统级功能仿真设计思路

整体 Testbench 编写思路如下，考虑到发送方与交换机是异步时序，在初始时为每个端口添加了与交换机不同频率的时钟（如下图）。

```
//时钟频率设定
always #2     clk=~clk;//交换机内时钟
always #10    rx_clk_0=~rx_clk_0;//输入端口时钟
always #10    rx_clk_1=~rx_clk_1;
always #10    rx_clk_2=~rx_clk_2;
always #10    rx_clk_3=~rx_clk_3;
always #10    rx_clk_4=~rx_clk_4;
always #10    rx_clk_5=~rx_clk_5;
always #10    rx_clk_6=~rx_clk_6;
always #10    rx_clk_7=~rx_clk_7;
always #20    rx_clk_8=~rx_clk_8;
always #20    rx_clk_9=~rx_clk_9;
always #20    rx_clk_10=~rx_clk_10;
always #20    rx_clk_11=~rx_clk_11;
always #20    rx_clk_12=~rx_clk_12;
always #20    rx_clk_13=~rx_clk_13;
always #20    rx_clk_14=~rx_clk_14;
always #20    rx_clk_15=~rx_clk_15;
```

图 3.2 交换机与各端口时钟

在相关使能信号进行初始化后，编写了三次 task，模拟数据包从 X 端口传入，发送至 Y 端口。

3.2 验证结果分析

3.2.1 模块级功能仿真

数据合路 Port_mux 模块与 Padding 模块的联合仿真

以端口 1,9,15 有数据包进入为例。模拟端口 1 和端口 9 同时有数据进入的情景。由于数据包长度不一，端口 1 数据包首先发完。

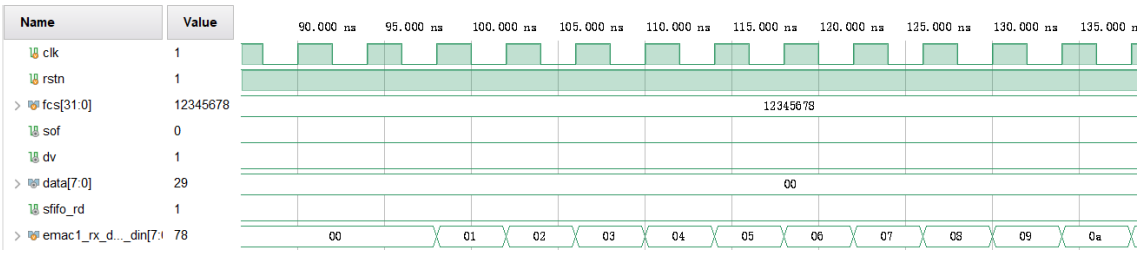


图 3.3 端口 1 数据包写入

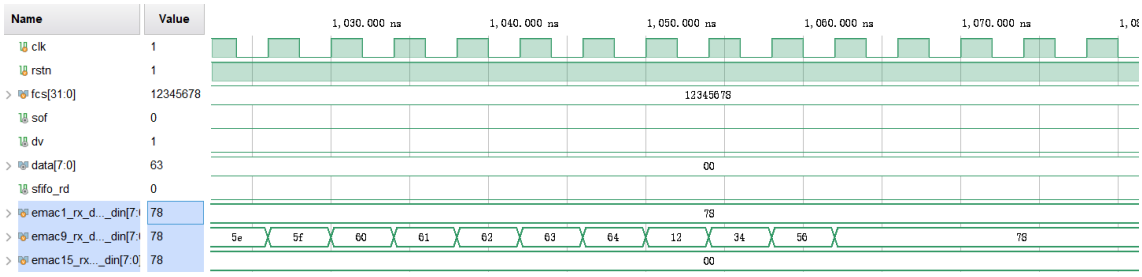


图 3.4 端口 9 数据包写入

数据合路状态机按照公平轮询，首先将端口 1 的数据读入，然后读入端口 9 的数据。数据经过填充，发送给交换核心。如图 3.5 所示，sof 与 dv 信号拉高，将端口 1 的数据进行输出。端口 1 数据输出完成后，进行端口 9 数据的输出。

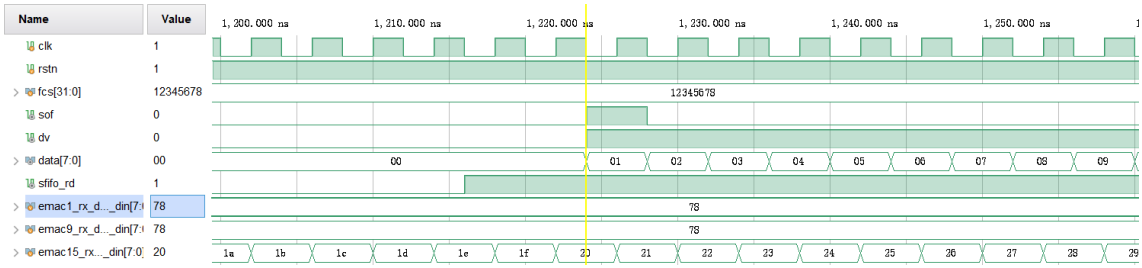


图 3.5 端口 1 数据开始传送给交换核心

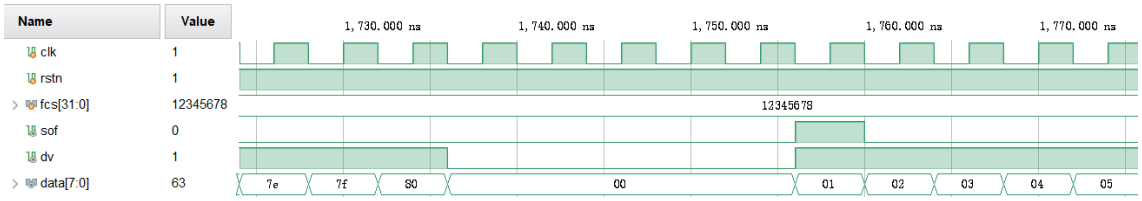


图 3.6 端口 1 传送结束，端口 9 的数据传给交换核心

由于 9 端口的数据包长度为 100 字节，不是 64 的倍数，该数据包在填充模块中需要进行填充处理，原数据与填充处理后的数据如下图所示。

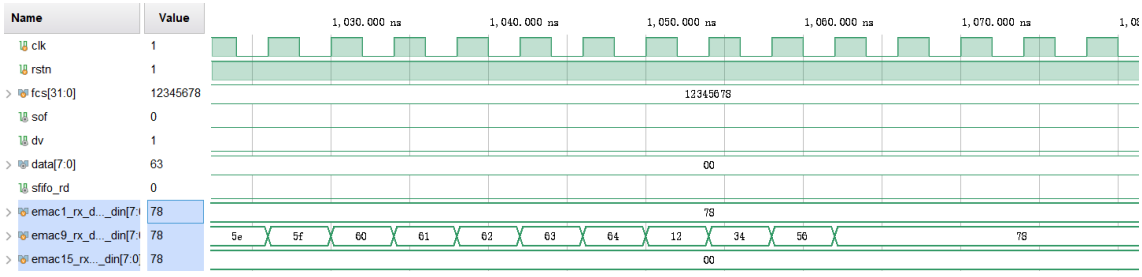


图 3.7 端口 9 的数据包长度为 100 字节

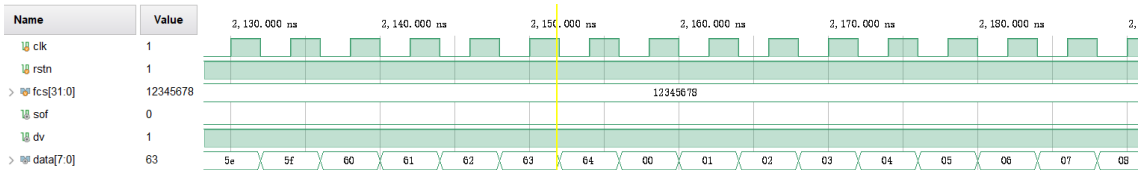


图 3.8 在 100 字节后开始填充至 128 字节

交换核心的总体功能验证

此模块有 3 个输入参量，data_sof 表示数据的开始，data_dv 表示数据有效，data_in 即为输入数据包的内容，如下图所示，

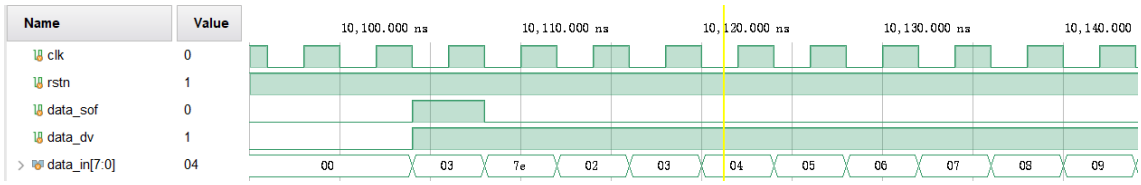


图 3.9 switch 模块输入

037e 表明到端口 3，长度为 0x07e 字节，0x02 开始为数据内容，从下图可以看到端口 3 开始有数据输出，同时有为 0x81 字节长的数据包发向端口 1，

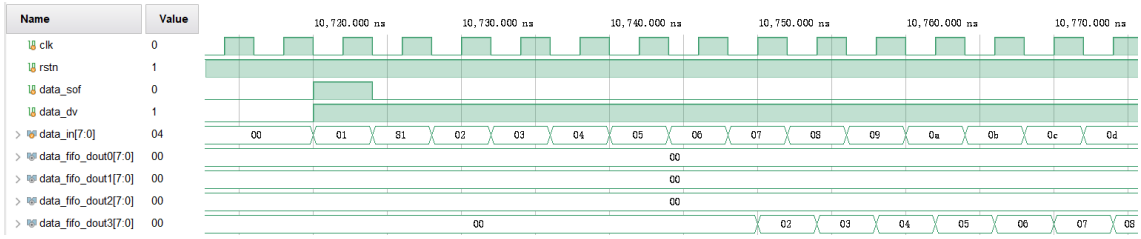


图 3.10 从端口 3 输出

然后端口 1 的输出如下图所示，

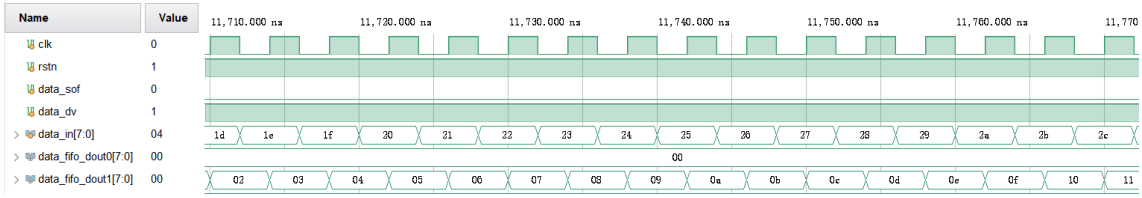


图 3.11 端口 1 输出

符合从任意端口进来，从设定好的端口输出的要求。

3.2.2 系统级仿真

从端口 1 向端口 11 发送数据。如图 3.12 和图 3.13 所示。

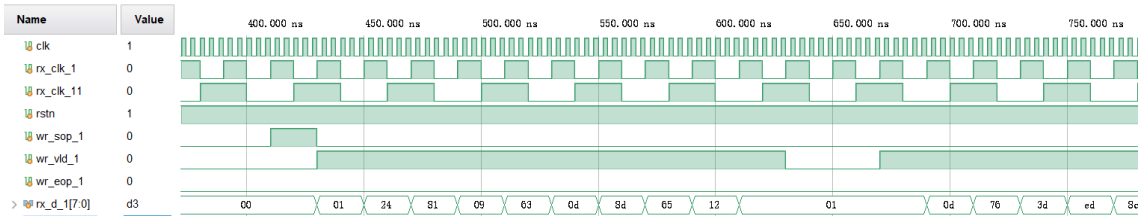


图 3.12 端口 1 的数据包输入开始

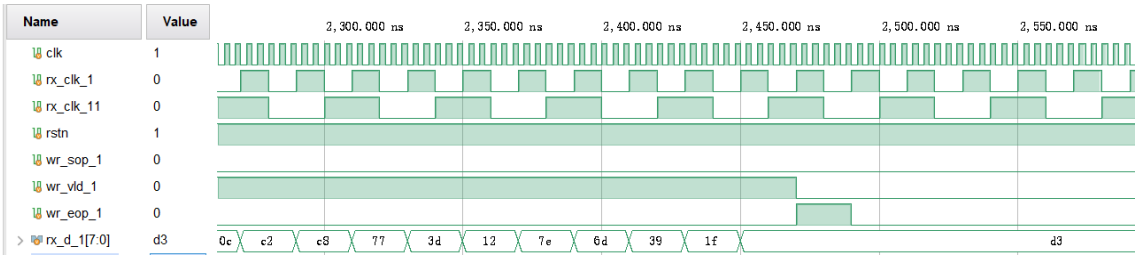


图 3.13 端口 1 的数据包传输结束

向端口 1 发送数据包。wr_sop_1，拉高一周期后 wr_vld_1 拉高，开始传输数据包。中途 wr_vld_1 变低两个周期，数据传输暂停，等待 wr_vld_1 变高后继续。传输结束后，wr_eop_1 拉高的同时，wr_vld_1 变低。符合赛题时序要求。

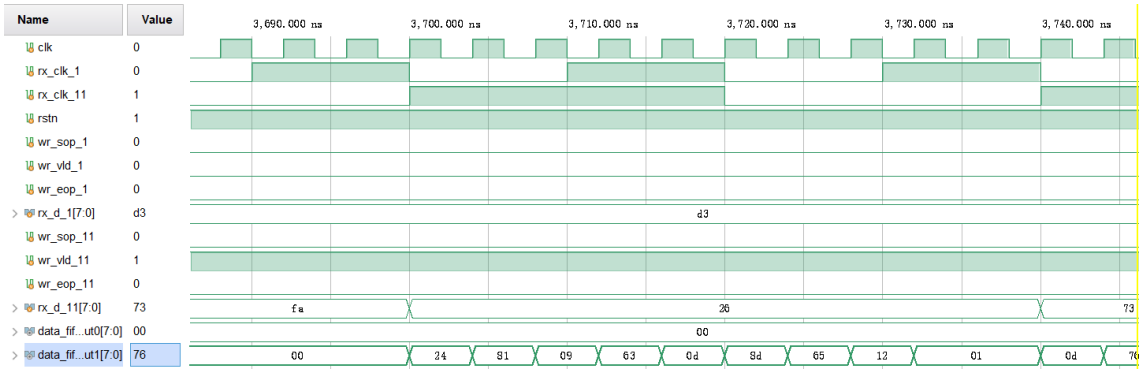


图 3.14 从端口 11 读出数据

从图 3.14 可以看到，数据能够正确从端口 11 读出，且第一个字节（即控制帧字节）被移除。

接下来，从端口 11 向端口 9 发送数据。

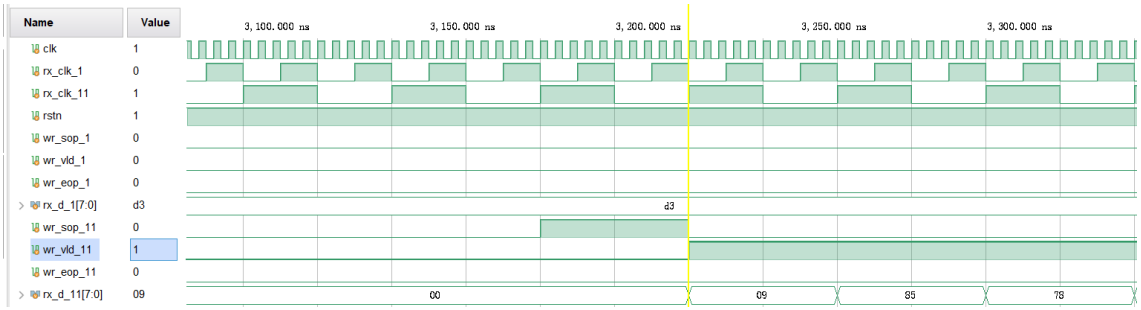


图 3.15 从端口 11 发送数据包

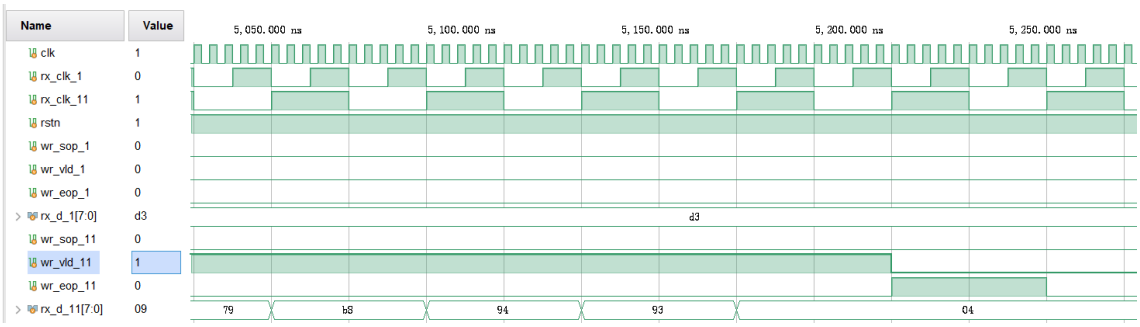


图 3.16 端口 11 发送数据包结束

端口 9 成功接收到数据。

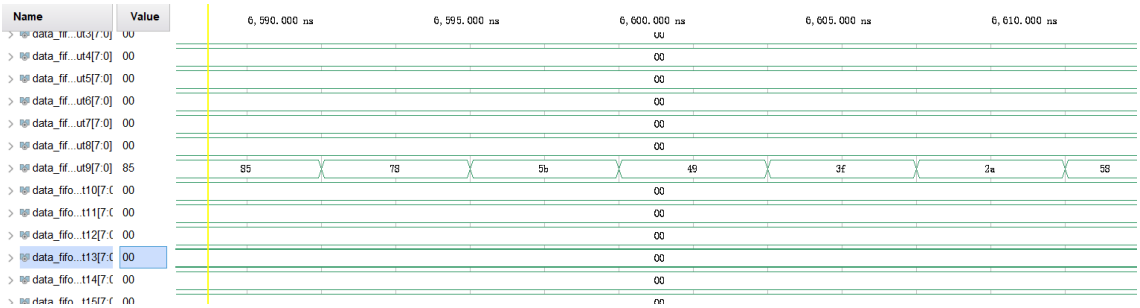


图 3.17 端口 9 接收到数据

3.2.3 共享缓存管理模块的 FPGA 设计

本文的硬件设计是在 Artix-7 系列的 FPGA 上实现的，芯片型号为 XC7A200T-2FBG676C。该款 FPGA 不仅性能优越，而且价格实惠，针对无线、有线、广播、工业，用户以及通信等行业中的低成本的小型应用，Artix-7 无疑是十分理想的选择。在上一节使用仿真器成功实现模块级和系统级的功能仿真之后，本节将完成在 FPGA 上最后的硬件设计。由于时间较为局促，目前仅进行仿真，未烧录至开发板。

根据设计，编写时序约束。参考数据手册，为设计的端口对应上 FPGA 的引脚，配置完对应引脚和电平后，生成 XDC 文件，完成引脚约束。在完成综合与实现后，可以查看 Vivado 的各项结果报告。

FPGA 片上资源占用率和设计性能分析

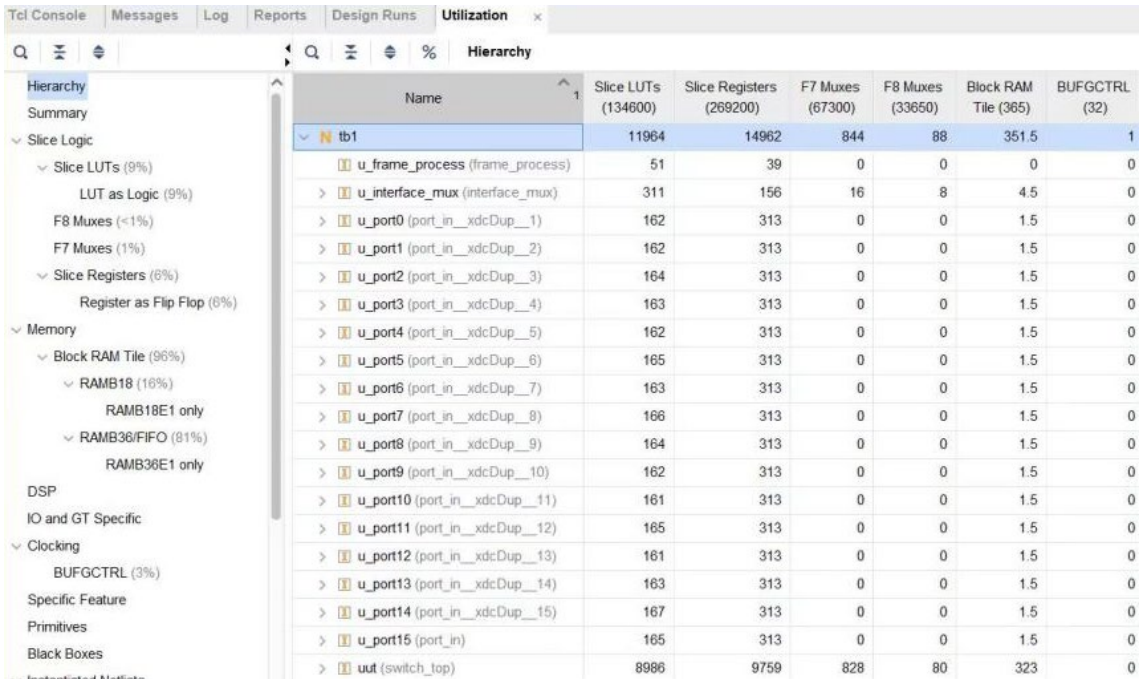
完成综合以及布局与布线之后，不仅可以对该多端口共享存储器的资源占用情况进行记录，而且还可以对静态时序与运行速度进行分析。具体分析数据结果如下表 3.2 和表 3.3 所示。各个端口及模块的资源占用率如图 3.18 所示

表 3.2 片上资源利用率（估计值）

器件	使用数量	可用数量	使用率 %
LUTs	11964	134600	8
Flip Flops	14962	269200	5.56
BRAMs	351	365	96
F7 Muxes	844	67300	1.25
F8 Muxes	88	33650	0.26
BUFGCTRL	1	32	3.12

表 3.3 设计性能（估计值）

指标	数值
WNS (ns)	3.92
Speed（MHz）	255.1



Name	Slice LUTs (134600)	Slice Registers (269200)	F7 Muxes (67300)	F8 Muxes (33650)	Block RAM Tile (365)	BUFGCTRL (32)
tb1	11964	14962	844	88	351.5	1
u_frame_process (frame_process)	51	39	0	0	0	0
u_interface_mux (interface_mux)	311	156	16	8	4.5	0
u_port0 (port_in__xdcDup__1)	162	313	0	0	1.5	0
u_port1 (port_in__xdcDup__2)	162	313	0	0	1.5	0
u_port2 (port_in__xdcDup__3)	164	313	0	0	1.5	0
u_port3 (port_in__xdcDup__4)	163	313	0	0	1.5	0
u_port4 (port_in__xdcDup__5)	162	313	0	0	1.5	0
u_port5 (port_in__xdcDup__6)	165	313	0	0	1.5	0
u_port6 (port_in__xdcDup__7)	163	313	0	0	1.5	0
u_port7 (port_in__xdcDup__8)	166	313	0	0	1.5	0
u_port8 (port_in__xdcDup__9)	164	313	0	0	1.5	0
u_port9 (port_in__xdcDup__10)	162	313	0	0	1.5	0
u_port10 (port_in__xdcDup__11)	161	313	0	0	1.5	0
u_port11 (port_in__xdcDup__12)	165	313	0	0	1.5	0
u_port12 (port_in__xdcDup__13)	161	313	0	0	1.5	0
u_port13 (port_in__xdcDup__14)	163	313	0	0	1.5	0
u_port14 (port_in__xdcDup__15)	167	313	0	0	1.5	0
u_port15 (port_in)	165	313	0	0	1.5	0
uut (switch_top)	8986	9759	828	80	323	0

图 3.18 各模块及端口的资源占用率

4 设计亮点及未来工作

4.1 设计亮点

1 数据和指针分离存储，更有利于读取数据包的状态和数据的处理。

2 自由指针队列的循环使用使得对 SRAM 空间的管理更为高效，资源利用率高。

3 调度算法基于 WRR 创新，降低了数据流量大时低优先级数据包长时间得不到调度的风险、便于配置队列带宽、延时低、片上资源消耗少。

4 工程中所有的状态机均采用混合类型而非传统的米利和摩尔型，更适合设计复杂状态机，使代码可读性更强。信号的命名规范，代码可读性高，便于维护。

4.2 未来工作

1 进一步扩展验证与测试。

2 探索 DWRR 等更多调度算法，并结合目前的算法进行优化。

3 整体模块略多，将尝试精简。

4 进行 FPGA 板上验证。

参考文献

[1] 郑振, 乔庐峰, 陈庆华, 等. 星载 IP 交换机中变长调度 Clos 交换结构的设计[J]. 通信技术, 2016, 49: 361-367.

[2] 乔庐峰, 陈庆华. Verilog HDL 数字系统设计与验证：以太网交换机案例分析[M]. 电子工业出版社, 2021.