FAST-ANT 2.0网络测试与分析套件

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号 | 修改人 | 修改时间 | 备注 |
| 2.0 | 蒋越 | 2019.05.23 | FAST-ANT 2.0初步设计 |
| 2.1 | 蒋越、吴尚明 | 2019.05.26 | 修改硬件架构设计、基本功能设计 |
| 2.2 | 蒋越、吴尚明 | 2019.05.28 | 根据指导意见修改硬件架构设计；重构设计文档。 |
| 2.2.1 | 蒋越、吴尚明 | 2019.05.31 | 增加架构对比分析；给出读写寄存器信息；增加功能模块概要设计分析。 |
| 2.2.2 | 蒋越、吴尚明 | 2019.06.02 | 修改硬件功能模块顺序；补充硬件功能模块设计。 |
| 2.3 | 蒋越、吴尚明 | 2019.06.03 | 细化各个功能模块设计。 |

FAST-ANT（Agile Network Tester）作为一款基于FPGA/CPU的轻量级网络测试分析器，支持针对吞吐、延迟等相关指标的精确测量。其功能可定制、轻量级、低成本的特点，适合中小型研究机构进行网络性能与功能的测试与分析。

FAST-ANT 1.0将支持的基本功能有：

1. 测量设备对于不同大小、不同协议类型报文的精确/粗略时延；
2. 测量设备对于不同大小、不同协议类型报文的吞吐率；
3. 测量设备对于不同大小、不同协议类型报文的丢包率；
4. 数据统计功能。

但随着实际应用场景复杂化、多样化，FAST-ANT 1.0所提供的测试功能已经难以满足针对新的协议或功能的全面测试，如：多类型流量并发、流量回放以及更高发包速率等。因此，基于上述现状，我们将设计并开发FAST-ANT 2.0来满足新的功能需求和性能指标。

FAST-ANT 2.0将基于FAST-ANT 1.0的基础上，优化重构软硬件设计实现，并在OpenBox-S4上先实现原型系统，后移植至NUDT-NSG团队自主开发的40G NP上。

# 1 FAST-ANT 2.0功能概述

## 1.1 多类型流量并发

FAST-ANT 2.0扩充的第一个功能是支持多类型流量并发。FAST-ANT在测试中常常被用于构造测试环境的背景流量，例如：模拟流量爆发，产生一条高速率流量来测试网络转发设备的丢包率、最大吞吐或时延等，如图1所示。



图1 针对网络设备的单/双端口测试

但是在一些复杂的测试场景中，例如：TSN，需要产生复杂的背景流量，这样的背景流量中可能包含了不同优先级和不同类型的流，如图2所示。FAST-ANT需要向待测设备生成一条多类型流量并发，并且需要按照测试场景的需求来设定流量的发送模式、速率等多项参数。



图2 复杂背景流测试场景

该功能的思路是：用户可以在Linux端通过构造不同的测试报文（报文类型、大小），配置报文发送的相关参数（速率）以及设定报文发送模式（乒乓报文、周期报文），同时指定并发流的发送端口。在FPGA上根据Linux端用户的设定来实现流量的精确、高性能发送，生成符合测试场景需求的复杂背景流量，满足并发、多协议的要求。

## 1.2 流量回放

在FAST-ANT 1.0中，对于捕获到的报文，我们仅统计报文个数、吞吐量以及计算指定类型报文的延迟。当获取了以上统计数据后，捕获到的报文将被丢弃。但在某些测试场景中，例如：在相同测试场景中对比不同算法的效果或不同设备的性能，需要保存某一时间段内所捕获到的所有报文并再次使用；或是在测试结束后，对某一时段产生的流量进行进一步分析来获取网络的某一时段状态。

因此，在FAST-ANT 2.0中，我们将支持流量回放。该功能的思路是：用户需要在Linux端设定捕获报文类型、捕获时间段或捕获报文个数；在FPGA中根据设定的捕获参数，将捕获的报文存储在硬件上，允许用户在Linux端读取存储的报文生成可视化结果并用于后续的分析。

## 1.3 FAST-ANT 1.0测试功能

FAST-ANT 2.0将继续支持FAST-ANT 1.0中所提供的测试功能，包括：时延、吞吐量等相关测试功能和内容。详细功能描述见FAST-ANT 1.0设计文档。

# 2 FAST-ANT 2.0总体设计

FAST-ANT 2.0基于FAST平台设计开发，分别在软硬件层面借助了FAST所提供的数据结构、接口函数以及FPGA OS和UM中的部分代码（根据扩充功能去掉UM流水线中部分不需要的功能模块）。原型系统在OpenBox-S4上进行开发实现。

本章首先在2.1和2.2小节介绍FAST-ANT 2.0的软件架构设计和硬件架构设计。其次，介绍FAST-ANT 2.0中所使用的数据分组结构。最后，通过介绍多类型流量并发和流量回放这两个测试功能的运行流程来进一步说明FAST-ANT 2.0架构设计思想。

## 2.1 FAST-ANT 2.0软件架构设计

FAST-ANT 2.0软件端负责接收用户在可视化界面上选定的测试功能和配置的测试参数，并在测试完成后以图表和数据文字相结合的方式展示相应的测试结果。其整体结构如图3所示。

FAST-ANT 2.0的软件端部分基于FAST库所提供的API和数据结构，通过Linux NIC同硬件中LCM模块进行消息交互来完成测试模式和测试参数的配置工作。在继承了FAST-ANT 1.0的基础上，将软件端从ARM移植至标准Linux机器上，并增加可视化界面，简化测试仪的使用流程，提升用户体验。同时，软件端在标准的Linux机器上实现便于用户进行移植和扩展。



图3 FAST-ANT 2.0软件架构图

首先，用户在提供的FAST-ANT GUI界面中选定测试功能并配置相对应的测试参数。GUI中的输入值将传递到FAST-ANT OS层，并映射到对应的FAST数据结构中；接着，将调用相应的FAST API和库，以报文的形式将相关测试参数配置到FPGA中并运行相应的测试功能；最后，FAST-ANT将同样通过调用相应的FAST API读取相关FPGA内寄存器内和存储内存上的数据，并在FAST-ANT OS层经过处理后以图标和数据文字的形式反馈至GUI界面，形成最终的测试结果的展示。

## 2.2 FAST-ANT 2.0硬件架构设计

在本章内容中，首先将对FAST-ANT 2.0的硬件架构进行概述；接着，给出硬件架构设计图并阐述核心硬件模块的功能；最后，展示UM流水线的接口信号图并详细介绍接口信息。

### 2.2.1 硬件架构对比

在给出FAST-ANT 2.0架构设计之前，我们分析一下不同硬件架构设计的优点和不足，综合对比后给出最优的架构设计方案。

原有的FAST架构采用的是单一流水线设计来通过插入不同的硬件功能模块，从而支持不同的功能。控制信息的读取通过一条环形通路实现，轮询每一个硬件功能模块，判断是否为目标模块中的目标寄存器，从而进行修改。而在FAST-ANT 2.0中，由于引入了新的测试功能，原有的FAST-ANT 1.0的硬件架构已无法很好地支撑。因此，需要进行一定的调整修改才能更好地适配新增加的测试功能。

关于数据通路，我们可以仍然采用原有FAST架构的单流水线设计，也可以采用多流水线并行设计；关于控制通路，我们也可以仍然采用原有的环形控制通路，也可以采用集中控制的方式。因此，总共可以组合出4种不同的硬件架构设计方案。接下来，我们将对这四种硬件架构设计方案进行分析比较。

1. **双流水线、集中控制**

硬件架构设计如图4所示。



图4 双流水线、集中控制的硬件架构设计图

***优点：***

1. 特定处理需求的分组有特定的功能流水线，不需要再经过冗余的模块。例如：流量回放功能的数据报文仅需要经过UDC和SCM模块；

2. PGM、OSM、UDC以及SCM模块的相关参数寄存器由LCM模块进行统一配置，即：通过少数的FAST-Pkt便可以完成对所有寄存器的读写操作。

***不足：***

1. SCM模块位于UDC模块之后，如果想最大限度地复用FAST-ANT 1.0中SCM的源代码，则需要在UDC模块中实现额外的FAST-Metadata的生成功能，而这个功能和UDC自身所要提供的功能无关，增加了额外的工作量和开发难度。

1. **双流水线、环形控制**

硬件架构设计如图5所示。



图5 双流水线、环形控制的硬件架构设计图

***不足：***

1. 在DMUX模块中需要额外的转换操作，即：对收到的包含控制信息的FAST-Pkt进行解析，提取出控制信息并按照环形控制通路的消息格式进行转发；

2. 一个环形控制通路的报文一次只能对一个寄存器进行读写操作。假设在UM流水线内的所有模块共有20个寄存器，则控制通路至少需要依次发送20个环形控制通路报文，那么DMUX则需要封装解封装报文20次，效率较低。

1. **单流水线、集中控制**

硬件架构设计如图6所示。



图6 单流水线、集中控制的硬件架构设计图

***优点：***

1. 可以最大限度地复用SCM模块和FAST-TSN中的令牌桶调度，降低开发难度，减少工作量；

2. LCM模块统一管理寄存器读写，突出“中心控制”的设计思想。

***不足：***

1. 当UM内部寄存器较多时，如何优化LCM内部的多选一是提高性能的关键。

1. **单流水线、环形控制**

硬件架构设计设计如图7所示



图7 单流水线、环形控制的硬件架构设计图

***不足：***

1. 当UDC从RAM中读取了存储的报文后，需要经过FAST-TSN的令牌桶调度部分。报文将被写入BufM中，而这将会对令牌桶的调度和流量整形造成影响。

综上所述，我们将采用硬件设计方案三——单流水线、集中控制，最大限度地复用已有功能模块的基础上，进行统一的寄存器读写管理。详细的硬件架构设计见2.2.2小节。

### 2.2.2 硬件架构概述

FAST-ANT 2.0的硬件总体架构采用了FAST架构设计，如图8所示。PGM、ESW和SCM模块分别在FAST 3.0、FAST-ANT 1.0以及FAST-TSN的基础上进行重构；LCM和UDC是为了支持FAST-ANT 2.0的新功能而专门开发的；同时，FAST-ANT 2.0继承了FAST-TSN中流量调度部分，即：EOS模块，支持令牌桶进行流量整形，这将在多类型流量并发中使用。

其中，核心功能模块包含以下几部分：



图8 FAST-ANT 2.0硬件架构设计图

***LCM（Local Control Module）模块：***本地控制模块，用于接收来自软件端配置的参数报文，解析报文后将相关配置参数写入对应的寄存器。同时，支持直接读取硬件功能模块内寄存器的值（如：计数器等），并封装成报文发送至软件端。

***UDC（User Defined Counter）模块：***用户定义计数器模块，用于支持流量回放功能。对于进入UDC模块的测试流量（即从非软件端互连口进入），UDC收集十万个报文写入UM流水线外部的RAM中，并在内部维护写入报文个数计数器，从而支持用户获取硬件中的存储情况。同时，用户可以通过配置参数来指定读取某一个报文或连续多个报文。

***SCM（Statistics Collection Module）模块：***收集统计模块，用于根据用户配置的捕获条件（如：流量类型、流量数目），对特定一些流进行统计计数，包括：吞吐率、流量大小、报文个数以及延迟等。在SCM模块内，将使用RAM来存储1024个延迟的值。在测试中或测试结束后，软件端可以读取这些延迟值来分析网络或网络设备的状态。参数的配置和统计结果的读取均通过LCM模块完成。

***PGM（Packet Generation Module）模块：***报文生成模块，用于根据用户配置参数来生成并发流量。PGM通过接收软件端构造的不同类型报文以及根据不同流的发送速率所构造的发送调度表，按照需求生成多类型的并发流量。软件端配置的不同类型报文存储在PGM内的RAM上。同时，采用轮询查找的方式，对调度表进行轮询，判断当前时刻哪一条流满足生成的条件，从而决定哪一类报文被从RAM中读出并发送。

***ESW（Epipe Switch Wrapper）模块：***流水线交换解析模块，用于解析分组类型、流量监管、转发动作生成以及生成TSN-Metadata（见2.4小节）和FAST架构中的Metadata。为了最大限度支持对FAST-ANT 1.0中SCM模块的设计和实现，ESW将对进入模块内的流量进行区分。对于PGM模块产生的流量生成TSN-Metadata并用于后面的流量整形以及调度；另外，ESW还支持用户配置分组转发的目的端口。

***EOS（Epipe Output Schedule）模块：***流水线输出调度模块，用于实现简化的CQF转发。主要负责支持核心的TSN元数据的循环队列转发功能，从而确保TSN分组调度输出的确定性延时以及带宽预约分组的基于令牌桶的流量整形功能的实现。在FAST-ANT 2.0中，EOS通过使用令牌桶算法对生成的并发流量进行流量整形。

关于FAST-TSN的更多细节内容可参考《FTS硬件详细设计文档0516》，更多细节部分在此不再赘述。

### 2.2.2 UM接口信号定义

FPGA OS与UM的接口信号图如图9所示。



图9UM接口信号示意图

根据FAST-ANT 2.0的整体架构，UM的接口信号定义如表1所示：

表1 UM接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **方向** | **位宽** | **备注** |
| FPGA OS Ingress - UM接口信号定义 | | | |
| clk | Input | 1 | 时钟 |
| um\_timestamp | Input | 64 | 进入时间戳 |
| rst\_n | Input | 1 | 复位信号 |
| pktin\_data\_wr | Input | 1 | 报文数据写信号 |
| pktin\_data | Input | 134 | 报文数据 |
| pktin\_data\_valid | Input | 1 | 报文数据标志位 |
| pktin\_data\_valid\_wr | Input | 1 | 报文数据标志位写信号 |
| pktin\_ready | Output | 1 | UM数据Ready信号 |
| UM - FPGA OS Egress接口信号定义 | | | |
| pktout\_data\_n\_wr | Output | 1 | 输出报文写信号，n为0~3 |
| pktout\_data\_n | Output | 134 | 输出报文数据，n为0~3 |
| pktout\_data\_n\_valid | Output | 1 | 输出报文标志位，n为0~3 |
| pktout\_data\_n\_valid\_wr | Output | 1 | 输出报文标志位写信号，n为0~3 |
| pktout\_ready | Input | 1 | FPGA OS数据Ready信号 |
| UM - RAM接口信号定义 | | | |
| um2ram\_wr | Input | 1 | UM写RAM的写信号 |
| um2ram\_wr\_addr | Input | 26 | UM写RAM的写地址 |
| um2ram\_wr\_data | Input | 134 | UM写入RAM的数据 |
| um2ram\_rd | Input | 1 | UM读RAM的读信号 |
| um2ram\_rd\_addr | Input | 26 | UM读RAM的读地址 |
| ram2um\_rd\_data | Output | 134 | RAM返回UM的数据 |

## 2.3 FAST-ANT 2.0数据分组结构

输入和输出数据分组采用了FAST报文的数据结构，由Metadata头部（包括Metadata 0，Metadata 1）和有效数据分组两部分共同组成，格式如图10所示。

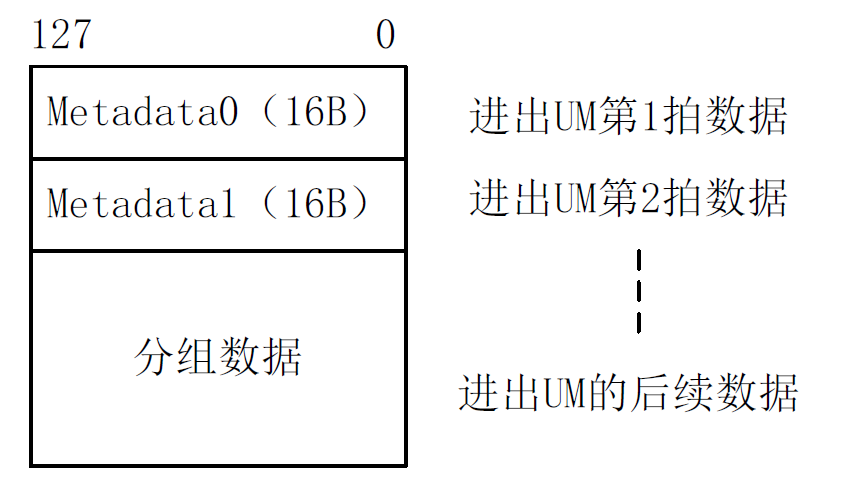


图10 分组数据结构示意图

Metadata在FAST报文的前32字节携带，每一个分组紧出UM的第一拍为16字节的Metadata 0，第二拍为Metadata 1。其中，Metadata 0由FPGA OS生成，数据字段意义已经提前定义好，为用户提供了在UM中处理报文所能用到的相关信息，如表2所示：

表2 FAST Metadata 0定义表

|  |  |  |
| --- | --- | --- |
| **字段** | **名称** | **备注** |
| FAST\_MD[127] | pkttype | 分组输入类型，0: 数据报文，1: 控制报文 |
| FAST\_MD[126] | pktdst | 分组目的，0: 网络接口输出，1: 送CPU |
| FAST\_MD[125:120] | inport | 分组输入端口号 |
| FAST\_MD[119:118] | outtype | 输出类型，  00:单播 01:组播  10:泛洪 11:从输入接口输出 |
| FAST\_MD[117:112] | output | 输出，  单播：分组输出端口ID  组播/泛洪：表地址索引 |
| FAST\_MD[111:109] | priority | 优先级，分组优先级 |
| FAST\_MD[108] | discard | 丢弃位 |
| FAST\_MD[107:96] | len | 分组长度，包含metadata字段的分组长度（按字节计算） |
| FAST\_MD[95:88] | smid | 上次处理模块号,最近一次处理分组的模块ID |
| FAST\_MD[87:80] | dmid | 目的模块号，下一次处理分组的模块ID |
| FAST\_MD[79:72] | pst | 标准协议类型 |
| FAST\_MD[71:64] | BID | 分组硬件缓存的缓冲区ID |
| FAST\_MD[63:50] | flowid | 流ID |
| FAST\_MD[49] | pktsrc | 分组的来源，0: 网络接口输入，1: CPU输入 |
| FAST\_MD[48] | reserve | 保留 |
| FAST\_MD[47:0] | ts | 48位接收时间戳 |

BID在IBM模块将报文写入PktBuf时获得，并在EBM模块中根据BID从PktBuf中读出报文。因此，在进行并发流调度时，OSM模块只需要控制BID的传递，来控制EBM模块发送报文。

FAST Metadata 1是为开发者预留的数据空间，可在将来扩展功能时使用。在FAST-ANT 2.0中未使用此数据拍。

接口分组是应用在FPGA OS和UM接口上的134bit的数据格式，其中高6位为控制信息，低128位为报文数据（由Metadata和报文数据组成）。其中，高6位的高2位为头尾表示，低4位为无效字节数，即：[133:132]位为报文数据的头尾标识，01代表报文头部，11代表报文中间数据，10代表报文尾部；[131:128]位为4位的无效字节数，其中0000表示16个字节全部有效，0001表示最低一个字节无效，最高15个字节有效，依次类推，1111表示最低15个字节无效，最高一个字节有效。如图11所示。



图11 接口分组结构示意图

## 2.4 TSN-Metadata定义

在FAST-TSN的实现中，为了减少EOS模块内存储FAST-Metadata的存储开销以及简化调度处理的过程，FAST-TSN根据自身需求设计了TSN-Metadata，具体定义如表3所示。

表3 TSN-Metadata定义表

|  |  |  |
| --- | --- | --- |
| **字段** | **名称** | **备注** |
| TSN\_MD[23:21] | 分组协议类型，pkt\_type | 0：Best Effort，1：预约带宽，  2：PTP，3：TSN，4：并发流量 |
| TSN\_MD[20:9] | 分组长度，pkt\_len | 按字节数计算 |
| TSN\_MD[8] | 输出端口，outport | 0：0号端口输出，1：1号端口输出 |
| TSN\_MD[7:0] | 分组存储ID，bufm\_ID | 对应BufM模块地址编号 |

在FAST-ANT 2.0中，我们对TSN\_MD[23:21]的定义进行了扩充，补充了类型4：并发流量，用于后续在EOS模块中进行流量整形使用。

# 3 FAST-ANT 2.0详细设计

## 3.1 FAST-ANT 2.0软件详细设计

## 3.2 LCM模块详细设计

### 3.2.1 需求与功能分析

为了更好地复用FAST-TSN中的令牌桶调度模块，减少模块之间传递的数据量，最少地修改已有模块代码，在FAST-ANT 2.0中取消了原有的环形控制通路（cin/cout），取而代之，我们设计并实现了本地控制模块（Local Control Module，LCM），用于集中化对各个硬件功能模块内定义的寄存器进行读写操作以及构造FAST-Pkt封装读取数据信息并上送软件端。

因此，在LCM模块中需要支持以下功能：

1. LCM模块接收来自不同端口的报文并区分。对于来自软件端的用于读写寄存器报文，则解析后直接修改对应寄存器；对于来自软件端的用于在PGM模块内构造并发流的报文，则转发至下一硬件功能模块；对于来自其他网络端口的接收到的测试报文，则转发至下一硬件功能模块；
2. LCM模块读取硬件寄存器数据信息，并构造FAST-Pkt报文上送到软件端。

### 3.2.2 LCM模块概要设计

LCM模块是UM流水线入口的第一个硬件功能模块，如图 中黄色模块所示。LCM模块接收所有进入UM流水线的FAST-Pkt，并对进入的报文进行解析分类。LCM模块首先区分报文是来自于软件端还是来自于网络端口。

若报文来自于软件端，则再进一步解析报文（根据FAST-Metadata的inport字段），区分报文是用于写寄存器的参数配置报文还是用于PGM构造并发流量的特定报文。对参数配置报文只需要解析数据后写入相应硬件功能模块的寄存器，报文即可被丢弃；对构造并发流量的特定报文则直接转发到下一个硬件功能模块；

若报文来自于网络端口，则将报文直接转发到下一个硬件功能模块。

对于读取的硬件寄存器的数据信息，则构造成FAST-Pkt并通过与软件端连接的端口上送软件端。



图 LCM模块在UM流水线内的位置

LCM模块的流程示意图如图 所示。



图 LCM模块流程示意图

在表 中定义了LCM所能直接读写的不同硬件功能模块中的寄存器。

表 LCM能够读取的寄存器表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **所属模块** | **寄存器名称** | **读写** | **寄存器大小** | **寄存器初始值** | **寄存器描述** |
| EOS，ESW | direction | 读写 | 1 | 1’b0 | 报文转发端口 |
| ESW | direct\_mac\_addr | 读写 | 48 | 48’b0 | TSN节点直连设备MAC地址。 |
| ESW | esw\_pktin\_cnt | 读 | 64 | 64’b0 | 进入ESW模块的分组计数器。 |
| ESW | esw\_pktout\_cnt | 读 | 64 | 64’b0 | ESW模块输出的分组计数器。 |
| ESW | local\_mac\_addr | 读 | 8 | 8’b0 | TSN本地MAC地址。 |
| ESW | bufm\_ID\_cnt | 读 | 8 | 8’b0 | Bufm中所使用的ID计数器 |
| EOS | token\_bucket\_para | 读写 | 32 | 32’b0 | 令牌桶R |
| EOS | token\_bucket\_depth | 读写 | 32 | 32’b0 | 令牌桶B |
| EOS | time\_slot\_flag | 读 | 1 | 1’b0 | 奇偶时间槽，当前位于奇数时间槽为1；偶数时间槽为0。 |
| EOS | eos\_mdin\_cnt | 读 | 64 | 64’b0 | 进入EOS模块的元数据计数器 |
| EOS | eos\_mdout\_cnt | 读 | 64 | 64’b0 | EOS模块输出的元数据计数器 |
| EOS | eos\_q0\_used\_cnt | 读 | 6 | 6’b0 | EOS模块Q0队列已使用长度计数器 |
| EOS | eos\_q1\_used\_cnt | 读 | 6 | 6’b0 | EOS模块Q1队列已使用长度计数器 |
| EOS | eos\_q2\_used\_cnt | 读 | 6 | 6’b0 | EOS模块Q2队列已使用长度计数器 |
| EOS | eos\_q3\_used\_cnt | 读 | 6 | 6’b0 | EOS模块Q3队列已使用长度计数器 |
| GOE | goe\_pktin\_cnt | 读 | 64 | 64’b0 | 进入GOE模块的分组计数器 |
| GOE | goe\_port0out\_cnt | 读 | 64 | 64’b0 | GOE模块往0口输出的分组计数器 |
| GOE | goe\_port1out\_cnt | 读 | 64 | 64’b0 | GOE模块往1口输出的分组计数器 |
| GOE | goe\_discard\_cnt | 读 | 64 | 64’b0 | GOE模块丢弃的分组计数器 |
| UDC | lcm2udc\_addr | 写 | 20 | 20’b0 | 读取报文的地址 |
| UDC | ram\_rd | 写 | 1 | 1’b0 | 读请求信号 |
| SCM | protocol\_type | 写 | 8 | 8’b0 | 特定报文类型 |
| SCM | statistic\_reset | 写 | 1 | 1’b0 | 重置信号 |
| SCM | scm\_pkt\_num\_cnt | 读 | 64 | 64’b0 | 记录总接收报文数量 |
| SCM | scm\_bit\_num\_cnt | 读 | 64 | 64’b0 | 记录总接收bit数量 |
| SCM | time\_latency | 读 | 32 | 32’b0 | 记录处理延迟 |

### 3.2.3 LCM模块详细设计

#### 3.2.3.1 LCM顶层模块设计

##### 3.2.3.1.1 模块接口设计

UDC模块的接口信号定义图如图 所示。



图 UDC模块接口信号定义图

UDC模块的接口信号定义表如表 所示。

表 UDC模块接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟信号 |
| rst\_n | 1 | Input | 复位信号，低有效 |
| in\_lcm\_data[133:0] | 134 | Input | 进入LCM模块的数据 |
| in\_lcm\_data\_wr | 1 | Input | 进入LCM模块的数据写信号 |
| in\_lcm\_data\_valid | 1 | Input | 进入LCM模块的报文有效信号 |
| in\_lcm\_data\_valid\_wr | 1 | Input | 进入LCM模块的报文有效信号的写信号 |
| in\_lcm\_data\_ready | 1 | Output | LCM准备接收数据的信号 |
| lcm2port\_n\_data[133:0] | 134 | Output | LCM输出到端口n的数据 |
| lcm2port\_n\_data\_wr | 1 | Output | LCM输出到端口n的数据写信号 |
| lcm2port\_n\_data\_valid | 1 | Output | LCM输出到端口n的报文有效信号 |
| lcm2port\_n\_data\_valid\_wr | 1 | Output | LCM输出到端口n的报文有效信号的写信号 |
| out\_lcm\_data[133:0] | 134 | Output | LCM输出的数据 |
| out\_lcm\_data\_wr | 1 | Output | LCM输出的数据写信号 |
| out\_lcm\_data\_valid | 1 | Output | LCM输出的报文有效信号 |
| out\_lcm\_data\_valid\_wr | 1 | Output | LCM输出的报文有效信号的写信号 |
| out\_lcm\_data\_ready | 1 | Input | 下一模块准备接收数据的信号 |
| direction | 1 | Both | 报文转发端口 |
| direct\_mac\_addr | 48 | Both | TSN节点直连设备MAC地址。 |
| esw\_pktin\_cnt | 64 | Input | 进入ESW模块的分组计数器。 |
| esw\_pktout\_cnt | 64 | Input | ESW模块输出的分组计数器。 |
| local\_mac\_addr | 8 | Input | TSN本地MAC地址。 |
| bufm\_ID\_cnt | 8 | Input | Bufm中所使用的ID计数器 |
| token\_bucket\_para | 32 | Both | 令牌桶R |
| token\_bucket\_depth | 32 | Both | 令牌桶B |
| time\_slot\_flag | 1 | Input | 奇偶时间槽，当前位于奇数时间槽为1；偶数时间槽为0。 |
| eos\_mdin\_cnt | 64 | Input | 进入EOS模块的元数据计数器 |
| eos\_mdout\_cnt | 64 | Input | EOS模块输出的元数据计数器 |
| eos\_q0\_used\_cnt | 6 | Input | EOS模块Q0队列已使用长度计数器 |
| eos\_q1\_used\_cnt | 6 | Input | EOS模块Q1队列已使用长度计数器 |
| eos\_q2\_used\_cnt | 6 | Input | EOS模块Q2队列已使用长度计数器 |
| eos\_q3\_used\_cnt | 6 | Input | EOS模块Q3队列已使用长度计数器 |
| goe\_pktin\_cnt | 64 | Input | 进入GOE模块的分组计数器 |
| goe\_port0out\_cnt | 64 | Input | GOE模块往0口输出的分组计数器 |
| goe\_port1out\_cnt | 64 | Input | GOE模块往1口输出的分组计数器 |
| goe\_discard\_cnt | 64 | Input | GOE模块丢弃的分组计数器 |
| lcm2udc\_addr | 20 | Output | 读取报文的地址 |
| ram\_rd | 1 | Output | 读请求信号 |
| protocol\_type | 8 | Output | 特定报文类型 |
| statistic\_reset | 1 | Output | 重置信号 |
| scm\_pkt\_num\_cnt | 64 | Input | 记录总接收报文数量 |
| scm\_bit\_num\_cnt | 64 | Input | 记录总接收bit数量 |
| time\_latency | 32 | Input | 记录处理延迟 |

##### 3.2.3.1.2 模块实现

LCM模块内部架构设计图如图 所示。



图 LCM模块内部架构设计图

注：reg\_Rd和reg\_Wr为读写寄存器操作，相关寄存器信号信息请参考3.2.3.1.1小节中的表 。

#### 3.2.3.2 DMUX模块

##### 3.2.3.2.1 模块接口设计

DMUX模块的接口信号定义图如图 所示。



图 DMUX模块接口信号定义图

DMUX模块的接口信号定义表如表 所示。

表 DMUX模块接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟信号 |
| rst\_n | 1 | Input | 复位信号，低有效 |
| in\_lcm\_data[133:0] | 134 | Input | 进入LCM模块的数据 |
| in\_lcm\_data\_wr | 1 | Input | 进入LCM模块的数据写信号 |
| in\_lcm\_data\_valid | 1 | Input | 进入LCM模块的报文有效信号 |
| in\_lcm\_data\_valid\_wr | 1 | Output | 进入LCM模块的报文有效信号的写信号 |
| in\_lcm\_data\_ready | 1 | Output | LCM准备接收数据的信号 |
| dmux2reg\_data[133:0] | 134 | Output | DMUX模块传递至Reg\_W\_R模块的数据 |
| dmux2reg\_data\_wr | 1 | Output | DMUX模块传递至Reg\_W\_R模块的数据写信号 |
| dmux2reg\_data\_valid | 1 | Output | DMUX模块传递至Reg\_W\_R模块的报文有效信号 |
| dmux2reg\_data\_valid\_wr | 1 | Output | DMUX模块传递至Reg\_W\_R模块的报文有效信号的写信号 |
| out\_lcm\_data[133:0] | 134 | Output | LCM输出的数据 |
| out\_lcm\_data\_wr | 1 | Output | LCM输出的数据写信号 |
| out\_lcm\_data\_valid | 1 | Output | LCM输出的报文有效信号 |
| out\_lcm\_data\_valid\_wr | 1 | Output | LCM输出的报文有效信号的写信号 |

##### 3.2.3.2.2 模块实现

DMUX模块需要区分报文是读写寄存器报文（传递至Reg\_W\_R子模块进行解析），还是测试流量和PGM用于构造并发流量的报文（传递至下一个硬件功能模块UDC）。因此，DMUX内实现一个简单的状态机，如图 所示。



图 DMUX模块状态机图

**IDLE\_S：**空闲状态。判断进入DMUX的报文类型。如果是读写寄存器报文，则将pkt\_flag置为1，报文第一拍传递至Reg\_W\_R模块；否则，pkt\_flag置为0，报文第一拍传递至下一个硬件功能模块UDC。跳转到IDLE\_S。

**SEND\_S：**转发状态。根据pkt\_flag来决定报文传递的方向。当报文传递结束，则跳转到IDLE\_S。

#### 3.2.3.3 Reg\_W\_R模块

##### 3.2.3.3.1 模块接口设计

Reg\_W\_R模块的接口信号定义图如图 所示。



图 Reg\_W\_R模块接口信号定义图

Reg\_W\_R模块的接口信号定义表如表 所示。

表 Reg\_W\_R模块接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟信号 |
| rst\_n | 1 | Input | 复位信号，低有效 |
| dmux2reg\_data[133:0] | 134 | Input | Reg\_W\_R模块从DMUX模块接收的数据 |
| dmux2reg\_data\_wr | 1 | Input | Reg\_W\_R模块从DMUX模块接收的数据写信号 |
| dmux2reg\_data\_valid | 1 | Input | Reg\_W\_R模块从DMUX模块接收的报文有效信号 |
| dmux2reg\_data\_valid\_wr | 1 | Input | Reg\_W\_R模块从DMUX模块接收的报文有效信号的写信号 |
| lcm2port\_n\_data[133:0] | 134 | Output | LCM输出到端口n的数据 |
| lcm2port\_n\_data\_wr | 1 | Output | LCM输出到端口n的数据写信号 |
| lcm2port\_n\_data\_valid | 1 | Output | LCM输出到端口n的报文有效信号 |
| lcm2port\_n\_data\_valid\_wr | 1 | Output | LCM输出到端口n的报文有效信号的写信号 |
| reg\_Wr | 统一指代对寄存器进行读写的信号  具体内容可以参考3.2.3.1.1小节中的表 | | |
| reg\_Rd |

##### 3.2.3.3.2 模块实现

在Reg\_W\_R模块中，需要支持对寄存器的读写操作，并能够将读取的寄存器数据信息封装成FAST-Pkt报文并上送软件端。其状态机如图 所示。



图 Reg\_W\_R模块状态机图

**IDLE\_S：**空闲状态。判断进入Reg\_W\_R模块的报文的类型。如果是写报文，则跳转到WR\_S；如果是读报文，则跳转到RD\_S。

**WR\_S：**写寄存器状态。解析报文中携带的寄存器编号和数据，写入指定的寄存器内。在本报文结束时，跳转到IDLE\_S。

**RD\_S：**解析读报文状态。解析读报文内所要读取的寄存器编号，并获取该寄存器值。在解析完此读报文后，跳转到SEND\_S。

**SEND\_S：**读报文上送状态。将获得的寄存器编号和对应的数据封装到上送软件端的报文内并发送。直到整个上送软件端的报文发送完毕，跳转到IDLE\_S。

## 3.3 UDC模块详细设计

### 3.3.1 需求与功能分析

UDC模块需要支持根据用户配置的存储参数（例：存储报文个数、时长）对FAST\_ANT2.0收到的报文按配置参数要求存储到RAM上，完成存储操作后，可将RAM中的报文读出后再送至软件端。

因此，在UDC模块中需要支持以下功能：

1. 接受LCM的配置参数：读请求信号（udc2ram\_rd）、读取报文起始地址（udc2ram \_addr）；
2. 测试流量进入UDC后，写RAM信号（udc2ram\_data\_wr）置为有效，将每个报文写入相应的RAM地址；
3. RAM中报文的存储形式是每个地址存放一拍的报文数据，每个报文固定分配100拍的地址空间；
4. 设置报文计数器(udc2ram\_pkt\_num\_cnt),记录RAM中已经存放报文的个数，每存放一个完整报文，计数器+1：
5. 若读信号(udc2ram\_rd)有效，根据用户给的读取的目的报文号，软件端解析出报文的起始地址，再下发给UDC的地址寄存器（lcm2udc\_addr）；
6. 经过UDC的报文都将发送到SCM。

### 3.3.2 UDC模块概要设计

UDC模块位于LCM模块和SCM模块之间，如图 中黄色模块所示，负责接收来自LCM模块的FAST-Pkt，计数后写入外部RAM中。当测试结束后，支持用户从软件端读取RAM中保存的报文。



图 UDC模块在UM流水线内的位置

UDC模块的流程示意图如图 所示。



图 UDC模块流程示意图

为了更好地支持UDC模块的流量回放功能，在UDC定义了表 所示的寄存器，用于存储统计数据和软件端的配置信息。

表 UDC模块内寄存器定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **寄存器名称** | **寄存器大小** | **寄存器初始值** | **寄存器描述** |
| lcm2udc\_addr | 20 | 20’b0 | 读取报文的地址 |
| ram\_rd | 1 | 1’b0 | 读请求信号 |

### 3.3.3 UDC模块详细设计

#### 3.3.3.1 模块接口设计

UDC模块的接口信号定义图如图 所示。



图 UDC模块的接口信号定义图

UDC模块的接口信号定义表如表 所示。

表 UDC模块的接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟信号 |
| rst\_n | 1 | Input | 复位信号 |
| lcm2udc\_data | 134 | Input | UDC接收的报文分组数据 |
| lcm2udc\_data\_valid\_wr | 1 | Input | 接收报文分组有效 |
| lcm2udc\_data\_valid | 1 | Input | 接收了完整的一个报文 |
| ram2udc\_data | 134 | Input | 从RAM读取的报文分组数据 |
| ram2udc\_data\_valid\_wr | 1 | Input | RAM读取报文分组有效 |
| ram2udc\_data\_valid | 1 | Input | 从RAM中读取了一个完整报文 |
| udc2ram\_pkt\_rd\_num | 20 | Input | 从RAM中读取的报文数 |
| udc2ram\_pkt\_num\_cnt | 20 | Input | RAM中已存放报文个数 |
| lcm2udc\_addr | 20 | Input | 读报文地址 |
| ram\_rd | 1 | Input | 读请求信号 |
| ram\_pkt\_wr | 1 | Input | 往RAM中写报文信号 |
| udc2port0\_data\_wr | 1 | Output | 读取报文往软件端写有效 |
| udc2port0\_data | 134 | Output | 从RAM读取后发往软件端报文 |
| udc2ram\_data\_wr | 1 | Output | 发往RAM的写有效信号 |
| udc2ram\_wdata | 134 | Output | 写入RAM的报文数据 |
| udc2ram\_addr | 20 | Output | 从RAM读取报文的地址 |
| udc2ram\_rd | 1 | Output | 发往RAM的读信号 |
| udc2scm\_data\_wr | 1 | Output | 发往SCM的写有效信号 |
| udc2scm\_data | 134 | Output | 发往SCM的报文数据 |
| udc2scm\_valid\_wr | 1 | Output | 发往SCM报文有效 |
| udc2scm\_valid | 1 | Output | 发出了一完整报文 |
| udc2ram\_pkt\_num\_cnt | 20 | Output | RAM中已存放报文数 |
| ram2udc\_addr | 20 | Output | 从RAM中读取报文的地址 |

#### 3.3.3.2 模块实现

UDC模块的状态机如图 所示。



图 UDC模块的状态机图

**IDLE\_S：**UDC初始状态。测试流量还未进入UDC,写RAM信号（udc2ram\_data\_wr）为0，设置一标志位flag,根据Metadata0[125:120]判断分组输入端口号是0号端口还是非0号端口，若为0号端口，flag置0（非统计流量），若为非0端口，flag置1（统计流量）。若测试流量进入UDC且读信号为0（lcm2udc[133:132]=2'b01 && udc2ram\_rd=1'b0），状态跳转到SEND&STORE\_S状态；若读请求信号有效（udc2ram\_rd=1'b1），状态跳转到READ\_S状态。

**SEND\_STORE\_S：**UDC的转发与存储状态。如果是并发流报文(flag=0)，则直接将报文发往下一模块SCM，跳转到IDLE\_S状态；若为流量回放的统计报文（flag=1），写RAM信号（udc2ram\_data\_wr）置为有效，设一寄存器（udc2ram\_wdata）存放报文，当前报文存放的起始地址为报文计数器（udc2ram\_pkt\_num\_cnt）的值减1再乘以100，令udc2ram\_addr等于这一起始地址，每存放一拍的报文分组，地址udc2ram\_addr +1指向下一拍数据的存放地址，直到存放了一个完整报文（udc2ram\_data[133:132]=2’b10分组写入RAM），此次报文存放结束，报文计数器（udc2ram\_pkt\_num\_cnt）统计+1。同时，报文也发送至SCM，若一完整报文发送完毕（udc2scm\_valid=1’b1），跳转到IDLE\_S状态。

**READ\_S：**将从RAM中读取的报文发送至软件端，当有读请求信号时，读取的报文号(ram2udc\_addr)即为目的报文所在块号，读取报文的起始地址（udc2ram\_addr）为报文号乘以100所得值，每读取一拍分组，地址udc2ram\_addr+1,并将分组送往软件端，直到读取了一个完整的报文（ram2udc\_data\_valid=1’b1），读取报文结束，跳转至IDLE\_S状态。

#### 3.3.3.3 RAM的设计与读写操作

##### 3.3.3.3.1 RAM大小

根据性能需要，RAM设计的大小为1M 134bit = 134Mb

##### 3.3.3.3.2 报文写操作

报文写入RAM的操作分为以下9个步骤：

1. 计算报文存放起始地址，利用报文计数器（udc2ram\_pkt\_num\_cnt）的值，得起始地址udc2ram\_addr = (udc2ram\_pkt\_num\_cnt - 1) × 100；
2. 将报文的第一拍存入起始地址，同时往下一个模块发送；
3. udc2ram\_addr <= udc2ram\_addr + 1；
4. 将报文第二拍存入udc2ram\_addr指向地址，同时往下一个模块发送；
5. udc2ram\_addr <= udc2ram\_addr + 1；
6. 若下一拍不是报文尾，重复上述操作过程；
7. 依此直到存放了报文尾（udc2ram\_data[133:132] == 2’b10），报文存放结束；
8. 报文计数器（udc2ram\_pkt\_num\_cnt）的值加1；
9. 报文计数器（udc2ram\_pkt\_num\_cnt）的值加到10000时，下一个报文计数器更新为1。

##### 3.3.3.3.3 报文读操作

报文从RAM中读出的操作分为以下3个步骤：

1. 根据用户要读取的报文号，软件端计算起始地址下发到ram2udc\_addr，令udc2ram\_addr = ram2udc\_addr；
2. 读取一拍报文，地址udc2ram + 1,并将这一拍报文送软件端；
3. 当读完报文尾时，报文读取结束。

## 3.4 SCM模块详细设计

### 3.4.1 需求与功能分析

SCM模块在FAST-ANT 1.0的基础上，所支持的功能不变，仅改变了控制信息的配置方式。由之前通过环形控制通路配置转变为LCM模块集中配置管理。

SCM模块支持对接收的报文进行计数统计，包括：报文个数、流量大小等。同时，支持测量处理延时，精度可以达到纳秒级。用户也可以设定只统计特定协议的流量。在未来，将支持更多的过滤条件，例如：IP地址、端口号等，供用户进行组合，从而提高对流的过滤统计的粒度。

因此，在SCM内需要支持以下功能：

1. 支持对FAST报文的Metadata和报文数据部分进行解析，提取与统计相关的数据信息，例如：报文长度等；
2. 支持计数统计。根据获取的数据信息，修改报文个数计数器（scm\_pkt\_num\_cnt）和总接收bit计数器（scm\_bit\_num\_cnt）；
3. 支持区分不同来源的报文，对接收的测试报文在计数统计后丢弃，而对其余报文则正常传递至下一硬件功能模块；
4. 支持同LCM通信。接收来自LCM的配置信息，修改相应寄存器。同时，将相关统计技术寄存器的值返回给LCM，进一步上送至软件端处理分析。

### 3.4.2 SCM模块概要设计

SCM模块位于UDC模块和PGM模块之间，如图 所示，负责接收来自UDC模块的FAST-Pkt等数据并统计流量信息（包括：报文个数、流量大小以及延迟等）。同时，将接收到的测试报文丢弃，其余类型的流量的FAST-Pkt等数据则正常送往下一功能模块，即：PGM模块。

同时，SCM模块支持同LCM模块进行通信，允许LCM模块读写寄存器。



图 SCM模块在UM流水线内的位置

SCM模块的流程示意图如图 中黄色模块所示。



图 SCM模块流程示意图

为了更好地支持SCM模块的统计功能，在SCM定义了表 所示的寄存器，用于存储统计数据和软件端的配置信息。

表 SCM模块内寄存器定义表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器名称** | **读写** | **寄存器大小** | **寄存器初始值** | **寄存器描述** |
| protocol\_type | 写 | 8 | 8’b0 | 特定报文类型 |
| statistic\_reset | 写 | 1 | 1’b0 | 重置信号 |
| scm\_pkt\_num\_cnt | 读 | 64 | 64’b0 | 记录总接收报文数量 |
| scm\_bit\_num\_cnt | 读 | 64 | 64’b0 | 记录总接收bit数量 |
| time\_latency | 读 | 32 | 32’b0 | 记录处理延迟 |

### 3.4.3 SCM模块详细设计

#### 3.4.3.1 模块接口设计

SCM模块的接口信号定义图如图 所示。



图 SCM模块接口信号定义图

SCN模块的接口信号定义表如表 所示。

表 SCM模块接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟信号 |
| rst\_n | 1 | Input | 复位信号，低有效 |
| in\_scm\_data | 134 | Input | 进入SCM模块的数据 |
| in\_scm\_data\_wr | 1 | Input | 进入SCM模块的数据写信号 |
| in\_scm\_data\_valid | 1 | Input | 进入SCM模块的有效报文信号 |
| in\_scm\_data\_valid\_wr | 1 | Input | 进入SCM模块的有效报文信号的写信号 |
| protocol\_type | 8 | Input | LCM模块配置的特定报文类型 |
| statistic\_reset | 1 | Input | 重置信号 |
| out\_scm\_data | 134 | Output | 输出的数据 |
| out\_scm\_data\_wr | 1 | Output | 输出的数据的写信号 |
| out\_scm\_data\_valid | 1 | Output | 输出的有效报文信号 |
| out\_scm\_data\_valid\_wr | 1 | Output | 输出的有效报文的写信号 |
| scm\_pkt\_num\_cnt | 64 | Output | 记录总接收报文数量，供LCM模块读取 |
| scm\_bit\_num\_cnt | 64 | Output | 记录总接收bit数量，供LCM模块读取 |
| time\_latency | 32 | Output | 记录处理延迟，供LCM模块读取 |

#### 3.4.3.2 模块实现

SCM的状态机如图 所示。



图 SCM状态机图

**IDLE\_S：**初始状态。接收FAST-Pkt第一拍，并判断该报文来源。如果来源于软件端，即：inport为0，则跳转到SEND\_S进行报文转发；如果来源于其他网络端口，则跳转到STAT\_S状态进行统计操作。

**SEND\_S：**发送状态。负责将报文正常转发到下一硬件功能模块PGM。每发送完一个完整的报文后，跳转到IDLE\_S，等待下一个进入的报文。

**STAT\_S：**统计状态。针对接收的测试流量进行统计计数操作，并丢弃所接收到的测试流量。每丢弃完一个完整的报文后，跳转到IDLE\_S，等待下一个进入的报文。

## 3.5 PGM模块详细设计

### 3.5.1 需求与功能分析

### 3.5.2 PGM模块概要设计

### 3.5.3 PGM模块详细设计

## 3.6 ESW模块详细设计

### 3.6.1需求与功能分析

ESW模块主要的功能需求为：分组类型的解析、比较MAC地址以及TSN-Metadata的构造、分组的转发、流量监管。

分组在进入ESW模块之后需要对分组进行解析，并将分组类型传输给EOS模块以供EOS模块根据分组的不同类型进入不同的队列；若分组的目的是本地直连设备，则直接由ESW模块发往GOE模块做计数；其他的分组则都往下级模块进行传输。往下级模块传输的分组还需根据流水线中的Bufm空闲ID的多少进行分组的丢弃（仅丢弃非TSN分组）。

针对TAP的需求，将所有需要进行转发的分组都复制一份往TAP口进行转发。

### 3.6.2 ESW模块概要设计

#### 3.6.2.1 ESW模块功能实现

##### 3.6.2.1.1 分组类型解析

需要对分组进行解析操作，支持带VLAN头以及标准以太网的解析，两种分组使用不同的解析方法。带VLAN头的分组根据VLAN字段的优先级（PCP）值进行分组类型区分，可区分出TSN分组、预约带宽分组、BestEffort分组。标准以太网的分组则根据协议字段区分出PTP分组或PGM生成的并发流量分组或捕获流量分组。

1. **判断是否携带VLAN头：**

表 VLAN分组的头部格式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [127:80]DMAC | [79:32]SMAC | [31:16]TPID(0x8100) | [15:13]PCP | [12]CFI | [11:0]VLAN ID |

注：TPID（Tag ProtocolIdentifier）:标签协议标识。是IEEE定义的新的类型，表明这是一个加了802.1标签的帧。TPID中包含了一个固定的值0X8100。

***PCP（user priority）：****这三位指明帧的优先级，一共8种优先级，从0到7。*

***CFI（canonical fomat indicator）：****CFI值为0说明是规范格式，1为非规范格式。它被用在令牌环/源路由FDDI介质访问方法中来指示封装帧中所带地址的比特次序信息。*

***VLAN ID（VLAN Identified）：****这是一个12位的域，指明VLAN的ID，取值范围为0～4095，一共4096个，由于0 和4095 为协议保留取值，所以VLAN ID 的取值范围为1～4094每个支持802.1Q协议的交换机发送出来的数据包都会包含这个域，以指明自己属于哪一个VLAN。*

表 标准以太网分组的头部格式

|  |  |  |
| --- | --- | --- |
| [127:80]DMAC | [79:32]SMAC | [31:16]eth type |

表 PTP协议分组的头部格式

|  |  |  |
| --- | --- | --- |
| [127:80]DMAC | [79:32]SMAC | [31:16]eth type（0x88F7） |

可根据首个16字节的低32位（带VLAN头的值是0x8100）判断是否是带有VLAN头的分组。不是带有VLAN头的分组再判断是否是PTP协议分组（PTP协议的值是0x88f7）或是PGM生成的并发流量或是其他的标准以太网分组。

1. **判断分组类型：**

可根据VLAN头中的PCP优先级进行判断，PCP的值的对应关系如表 ：

表 PCP和分组类型对应关系表

|  |  |  |  |
| --- | --- | --- | --- |
| PCP的值 | 6、7 | 3、4、5 | 0、1、2 |
| 分组类型 | TSN分组 | 预约带宽分组 | best effort分组 |

根据分组的协议类型字段信息以及VLAN中的PCP值将分组分为以下几种分组类型：TSN分组、预约带宽分组、BestEffort分组，并在ACTION中的分组协议类型字段记录。

1. **判断并发流量分组和其他分组类型：**

在FAST-TSN的设计中，所有其他类型的分组将被送到BestEffort队列中，而对于PGM生成的并发流量，则需要被送到预约带宽队列（Q2）进行流量整形（采用令牌桶算法）。因此，需要对来自PGM生成的并发流量进行区分并修改FAST-Metadata中的pkttype字段值。

##### 3.6.2.1.2 MAC地址比较

根据分组的源MAC、目的MAC分别进行比较。源MAC与本地直连设备MAC进行比较，若源MAC等于本地直连设备MAC再看输入端口是否为2（即本地直连设备），若输入端口不是2则将分组丢弃，具体原因参考后面的注；若源MAC不等于本地直连设备MAC或源MAC等于本地直连设备且输入端口是2则进行目的MAC比较。目的MAC本地直连设备的MAC地址比较。根据比较结果进行接下来的处理。

*[注：考虑到分组的目的MAC可能不存在于整个网络，因此在网络中传输了一个轮回之后还未找到目的，这样就可能导致分组一直在网络中循环，为了解决这一问题增加一个检测机制。若分组的源MAC是本地直连设备，而输入端口不为2则表示该分组是循环一次的，需要进行丢弃。]。*

1. **源MAC地址比较**

将分组的源MAC与本地直连设备的MAC地址进行比较：若源MAC等于本地直连设备MAC且分组输入端口不为2，则将分组进行丢弃；若源MAC不等于本地直连设备MAC或者源MAC等于本地直连设备MAC且分组输入端口为2则进行目的MAC比较。

1. **目的MAC地址比较**

将分组的目的MAC与本地直连设备MAC地址进行比较。得出如表 所示的以下结果：

表 目的MAC比较结果表

|  |  |
| --- | --- |
| **比较** | **结果** |
| 目的MAC等于本地直连设备地址 | outtype = 2’b00  outport = 6’h2 |
| 目的MAC不等于本地直连设备地址  源MAC不等于本地LCM或本地直连设备地址 | outtype = 2’b00  outport = ~inport |
| 目的MAC不等于本地直连设备地址  源MAC等于本地LCM或本地直连设备地址 | outtype = 2’b00  outport = direction |

1. **ACTION的构造**

根据比较结果对ACTION进行构造，ACTION的格式以及含义如表 ：

表 ACTION格式和含义表

|  |  |  |
| --- | --- | --- |
| **名称** | 分组协议类型（pkt\_type） | 分组输出端口/输出ID（outport） |
| **位宽** | 3 | 6 |
| **位置** | [8:6] | [5:0] |
| **含义** | 0：best effort 1：预约带宽  2：PTP 3：TSN 4：PGM并发 |  |

##### 3.6.2.1.3 分组转发

根据比较结果进行分组的转发。若目的MAC是本地直连设备的MAC地址则往GOE模块传输。若目的MAC未比较对上则执行本地LCM和本地直连设备进的分组由direction信号决定由哪个端口出、0号网口进的分组由1号网口出、1号网口进的分组由0号网口出的规则进行处理。同时将所有需要进行转发的分组都复制一份转发给TAP口。

##### 3.6.2.1.4 元数据修改和TSN-Metadata构造

分组在传输时都需要进行元数据的修改；同时若分组需要向下级模块传输则还需要构造TSN-Metadata，用于EOS模块内调度使用。

1. FAST-Metadata的修改

ACTION表的表项中已经给出可判断分组去向的三个重要信息：分组输出类型、分组输出目的、分组输出端口。元数据可在查表结束之后根据ACTION表对自身的这三个重要信息进行修改，以保证在之后的模块中可正确地对分组进行传输。

1. TSN-Metadata的构造

在EOS中需要对分组的类型进行判断，以便存入对应的队列。因此ESW根据需求构造仅包含这些信息的TSN\_MD数据传输给EOS模块，同时IBM模块需要8位的数据进行储存分组存入Bufm的ID号。因此在ESW定义的TSN-Metadata数据的内容如表 ：

表 TSN-Metadata数据格式定义表

|  |  |  |
| --- | --- | --- |
| **字段** | **名称** | **备注** |
| TSN\_MD[23:21] | 分组协议类型，pkt\_type | 0：Best Effort，1：预约带宽，  2：PTP，3：TSN，4：并发流量 |
| TSN\_MD[20:9] | 分组长度，pkt\_len | 按字节数计算 |
| TSN\_MD[8] | 输出端口，outport | 0：0号端口输出，1：1号端口输出 |
| TSN\_MD[7:0] | 分组存储ID，bufm\_ID | 对应BufM模块地址编号 |

##### 3.6.2.1.5 流量监管

分组是往下级模传输的，则需要根据Bufm的空闲ID数量进行判断分组是否应该丢弃。为了避免在非TSN分组过多的情况下造成Bufm资源不够用而导致TSN分组无法正常地在流水线中进行传输，有必要在ESW模块对分组进行流量监管。可根据Bufm中的空闲ID块对分组进行管理，当Bufm中的空闲ID小于某值1时，可控制ESW模块对best effort分组进行丢弃；当Bufm中的空闲ID大于某值1但小与某值2时，可控制ESW模块对预约带宽分组进行丢弃。

#### 3.6.2.2 ESW模块设计

##### 3.6.2.2.1模块划分

设计中将ESW模块设计为3个模块：ESW顶层模块、PKE模块、PAC模块。具体框架图以及子模块实现如图 所示。



图 ESW模块架构设计图

图中关键数据的格式和定义如下所示：

表 key格式定义表

|  |  |  |
| --- | --- | --- |
| **Key** | | |
| DMAC[101:54] | SMAC[53:6] | inport[5:0] |

表 ACTION格式定义表

|  |  |  |
| --- | --- | --- |
| **名称** | 分组协议类型（pkt\_type） | 分组输出端口/输出ID（outport） |
| **位宽** | 3 | 6 |
| **位置** | [8:6] | [5:0] |

表 TSN-Metadata格式定义表

|  |  |  |  |
| --- | --- | --- | --- |
| 分组协议类型[23:21]（pkt\_type） | 分组长度[20:9]（pkt\_len） | 输出端口[8]（outport） | 分组存储ID[7:0]（bufm\_ID） |

根据ESW模块的功能分析，给出ESW模块处理流程图如图 所示：



图 PKE模块流程图



图 PFW模块流程图



图 PAC模块流程图

**PKE模块：**首先判断FAST-Metadata中的inport字段来区分是否为PGM产生的并发流量。再判断是带VLAN头的分组还是标准以太网分组。若是带VLAN头的分组则根据PCP值进行分组类型的区分：TSN分组、预约带宽分组、best effort分组；若是标准以太网分组则根据协议字段进行分组类型区分：PTP分组、best effort分组。并将解析的结果写到pkttype中；同时提取出分组的三元组{目的MAC、源MAC、输入端口}与分组同时传输给PFW模块。

**PFW模块：**根据pkttype字段确定是否为PGM生成的并发流量。若是，则由软件端配置转发端口；否则，根据三元组的信息与本地直连设备MAC进行比较：源MAC比较：若源MAC等于本地直连设备MAC再看输入端口是否为2，若输入端口不是2则将分组丢弃；若源MAC不等于本地直连设备MAC或源MAC等于本地直连设备且输入端口是2则进行目的MAC比较。目的MAC比较：目的MAC等于本地直连设备MAC、目的MAC不等于本地直连设备MAC两种结果。根据匹配结果对ACTION中的其他字段信息进行修改；若匹配不命中，输出端口则执行本地LCM和本地直连设备进的分组由direction信号决定由哪个端口出、0号网口进的分组由1号网口出、1号网口进的分组由0号网口出的规则进行处理。将PKE模块传输给来的pkttype信息填到ACTION中再和分组发往PAC模块。

**PAC模块**：根据ACTION中的内容进行分组的转发。若ACTION的输出端口非0或1则将分组元数据根据ACTION进行修改之后将分组往本地直连设备传输；反之则根据Bufm的空闲ID进行分组的丢弃或转发。当Bufm大于值2，分组都可直接进行元数据修改之后传输分组；当Bufm大于值1小于值2时，分组类型为best effort的需要进行丢弃，其他分组进行正常传输；当Bufm小于值1时，分组类型为预约带宽或best effort的都需要进行丢弃，其他分组进行正常传输。不丢弃时，在分组传输时需要构造TSN-Metadata。在所有需要进行转发的分组都复制一份转发给TAP口。

##### 3.6.2.2.2 接口定义

ESW模块内部接口定义见表 。

表 ESW模块内部接口定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **模块名** | **信号名** | **位宽** | **方向** | **备注** |
| LCM | in\_esw\_data | 134 | Input | 分组数据输入 |
| in\_esw\_data\_wr | 1 | Input | 分组数据输入有效，高有效 |
| in\_esw\_valid | 1 | Input | 分组数据输入有效位 |
| in\_esw\_valid\_wr | 1 | Input | 分组数据输入有效位有效，高有效 |
| direction | 1 | Input | 交换机的数据流向 |
| local\_mac\_addr | 48 | Input | TSN本地LCM的MAC地址 |
| direct\_mac\_addr | 48 | Input | TSN本地外连设备的MAC地址 |
| esw\_pktin\_cnt | 64 | Output | 进入ESW模块的分组计数器 |
| esw\_pktout\_cnt | 64 | Output | ESW模块输出的分组计数器 |
| bufm\_ID\_cnt | 8 | Input | Bufm中空闲的ID计数器 |
| IBM | out\_esw\_data | 134 | Output | 分组数据往下级模块输出 |
| out\_esw\_data\_wr | 1 | Output | 分组数据往下级模块输出有效，高有效 |
| out\_esw\_valid | 1 | Output | 分组数据往下级模块输出有效位 |
| out\_esw\_valid\_wr | 1 | Output | 分组数据往下级模块输出有效位有效，高有效 |
| out\_esw\_tsn\_md | 24 | Output | TSN\_MD数据输出 |
| out\_esw\_tsn\_md\_wr | 1 | Output | TSN\_MD数据输出有效信号，高有效 |
| bufm\_ID\_count | 8 | Input | Bufm使用的ID数量 |
| GOE | out\_esw2port\_data\_N | 134 | Output | 分组数据往GOE输出（N=2/3） |
| out\_esw2port \_data\_wr\_N | 1 | Output | 分组数据往GOE输出有效，高有效（N=2/3） |
| out\_esw2port \_valid\_N | 1 | Output | 分组数据往GOE输出有效位（N=2/3） |
| out\_esw2port \_valid\_wr\_N | 1 | Output | 分组数据往GOE输出有效位有效，高有效（N=2/3） |

### 3.6.3 ESW模块详细实现

#### 3.6.3.1 ESW顶层模块

##### 3.6.3.1.1 功能分析

顶层模块只负责完成一些接口转接操作。

##### 3.6.3.1.2 接口定义

ESW模块的接口信号定义图如图 所示。



图 ESW模块的接口信号定义图

ESW模块的接口信号定义表如表 所示。

表 ESW模块的接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| From LCM | | | |
| in\_esw\_data | 134 | Input | 分组数据输入 |
| in\_esw\_data\_wr | 1 | Input | 分组数据输入有效，高有效 |
| in\_esw\_valid | 1 | Input | 分组数据输入有效位 |
| in\_esw\_valid\_wr | 1 | Input | 分组数据输入有效位有效，高有效 |
| direction | 1 | Input | 交换机的数据流向 |
| local\_mac\_addr | 48 | Input | TSN本地LCM的MAC地址 |
| direct\_mac\_addr | 48 | Input | TSN本地外连设备的MAC地址 |
| esw\_pktin\_cnt | 64 | Output | 进入ESW模块的分组计数器 |
| esw\_pktout\_cnt | 64 | Output | ESW模块输出的分组计数器 |
| bufm\_ID\_cnt | 8 | Output | Bufm中空闲的ID计数器 |
| To IBM | | | |
| out\_esw\_data | 134 | Output | 分组数据往下级模块输出 |
| out\_esw\_data\_wr | 1 | Output | 分组数据往下级模块输出有效，高有效 |
| out\_esw\_valid | 1 | Output | 分组数据往下级模块输出有效位 |
| out\_esw\_valid\_wr | 1 | Output | 分组数据往下级模块输出有效位有效，高有效 |
| out\_esw\_tsn\_md | 24 | Output | TSN\_MD数据输出 |
| out\_esw\_tsn\_md\_wr | 1 | Output | TSN\_MD数据输出有效信号，高有效 |
| bufm\_ID\_count | 5 | Input | Bufm空闲的ID数量 |
| To GOE | | | |
| out\_esw2port\_data\_N | 134 | Output | 分组数据往网口输出（N=2/3） |
| out\_esw2port \_data\_wr\_N | 1 | Output | 分组数据往网口输出有效，高有效（N=2/3） |
| out\_esw2port \_valid\_N | 1 | Output | 分组数据往网口输出有效位（N=2/3） |
| out\_esw2port \_valid\_wr\_N | 1 | Output | 分组数据往网口输出有效位有效，高有效（N=2/3） |

#### 3.6.3.2 PKE模块

##### 3.6.3.2.1 功能分析

判断是带VLAN头的分组还是标准以太网分组还是PGM模块生成的并发流量。若是带VLAN头的分组则根据PCP值进行分组类型的区分：TSN分组、预约带宽分组、best effort分组；若是标准以太网分组则根据协议字段进行分组类型区分：PTP分组、best effort分组。并将解析的结果写到pkttype中；同时提取出分组的三元组{目的MAC、源MAC、输入端口}与分组同时传输给PFW模块。

##### 3.6.3.2.2 接口定义

PKE模块的接口信号定义图如图 所示。



图 PKE模块的接口信号定义图

PKE模块的接口信号定义表如表 所示。

表 PKE模块的接口信号定义图

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| From LCM | | | |
| in\_pke\_data | 134 | Input | 分组数据输入 |
| in\_pke\_data\_wr | 1 | Input | 分组数据输入有效，高有效 |
| in\_pke\_valid | 1 | Input | 分组数据输入有效位 |
| in\_pke\_valid\_wr | 1 | Input | 分组数据输入有效位有效，高有效 |
| esw\_pktin\_cnt | 64 | Output | 进入ESW模块的分组计数器 |
| To PFW | | | |
| out\_pke\_data | 134 | Output | 分组数据输出 |
| out\_pke\_data\_wr | 1 | Output | 分组数据输出有效，高有效 |
| out\_pke\_valid | 1 | Output | 分组数据输出有效位 |
| out\_pke\_valid\_wr | 1 | Output | 分组数据输出效位有效，高有效 |
| out\_pke\_pkttype | 3 | Output | 分组协议类型信息输出 |
| out\_pke\_key | 102 | Output | 分组三元组信息输出 |

##### 3.6.3.2.3 模块实现

根据对PKE模块的功能分析，可直接用一个状态机实现。PKE模块的状态机如图 所示。



图 PKE模块的状态机图

**IDLE\_S：**空闲状态机。等待分组进入子模块；确保分组是第一拍，将元数据中的输入端口字段提取出来填到key[5:0]并将分组数据写入寄存器中进行缓存，跳转到MD1\_S。

**MD1\_S：**元数据传输状态机。将分组的第二拍数据继续用寄存器进行缓存，跳到KEY\_S。

**PARS\_S：**分组解析状态机。根据分组的该拍（以太网头）数据进行区分协议类型：若缓存的第一拍的[125:120]为0，则将pkttype置为4；若该拍的pkt[32:16]字段值为0x88f7，则将pkttype置为2；若值为0x8100，再根据pkt[15:13]的值将pkttype分别置为0、1、3（具体可参照3.2.1.1）；若值为其他则将pkttype置为0。将分组的DMAC、SMAC提取出来分别填到key[101:54]、key[53:6]。将第一拍分组数据从寄存器中读出并传输给PFW模块，再将第三拍数据进行缓存，跳转到TRANS\_S。

**TRANS\_S：**传输状态机。将分组数据传输给PAC模块，确保分组是正常顺序传输的，直到分组全部传输完成，可根据分组的头尾标志进行判断，跳转到IDLE\_S。

#### 3.6.3.3 PFW模块

##### 3.6.3.3.1 功能分析

首先，判断pkttype是否为4。若为4，则为PGM模块生成的并发流量，则将软件端配置的direction信号赋值给output；否则， 根据三元组的信息与本地直连设备MAC进行比较：源MAC比较：若源MAC等于本地直连设备MAC再看输入端口是否为2，若输入端口不是2则将分组丢弃；若源MAC不等于本地直连设备MAC或源MAC等于本地直连设备且输入端口是2则进行目的MAC比较。目的MAC比较：目的MAC等于本地直连设备MAC、目的MAC不等于本地直连设备MAC两种结果。根据匹配结果对ACTION中的其他字段信息进行修改；若匹配未命中，输出端口则执行本地LCM和本地直连设备进的分组由direction信号决定由哪个端口出、0号网口进的分组由1号网口出、1号网口进的分组由0号网口出的规则进行处理。将PKE模块传输给来的pkttype信息填到ACTION中再和分组发往PAC模块。

##### 3.6.3.3.2 接口定义

PFW模块的接口信号定义图如图 所示。



图 PFW模块的接口信号定义图

PFW模块的接口信号定义表如表 所示。

表 PFW模块的接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| From PKE | | | |
| in\_pfw\_data | 134 | Output | 分组数据输入 |
| in\_pfw\_data\_wr | 1 | Output | 分组数据输入效，高有效 |
| in\_pfw\_valid | 1 | Output | 分组数据输入效位 |
| in\_pfw\_valid\_wr | 1 | Output | 分组数据输入位有效，高有效 |
| in\_pfw\_pkttype | 3 | Output | 分组协议类型信息输入 |
| in\_pfw\_key | 102 | Output | 分组三元组信息输入 |
| To PAC | | | |
| out\_pfw\_data | 134 | Output | 分组数据输出 |
| out\_pfw\_data\_wr | 1 | Output | 分组数据输出有效，高有效 |
| out\_pfw\_valid | 1 | Output | 分组数据输出有效位 |
| out\_pfw\_valid\_wr | 1 | Output | 分组数据输出效位有效，高有效 |
| out\_pfw\_action | 9 | Output | action信息输出 |
| out\_pfw\_action\_wr | 1 | Output | action信息输出有效，高有效 |
| From LCM | | | |
| direction | 1 | Input | 交换机的数据流向 |
| local\_mac\_addr | 48 | Input | TSN本地LCM的MAC地址 |
| direct\_mac\_addr | 48 | Input | TSN本地外连设备的MAC地址 |

##### 3.6.3.3.3 模块实现

根据对PKE模块的功能分析，可直接使用一个状态机实现。PKE模块的状态机图如图 所示。



图 PKE模块的状态机图

**IDLE\_S：**空闲状态机。等待分组进入子模块；首先，判断pkttype是否为4。若为4，则跳转到D\_COM\_S，pgm\_flag置为高；否则，再根据分组第一拍元数据的smid字段信息[95:88]，若值为128则将用一个寄存器flag置高进行标识；然后跳转到S\_COM\_S。同时将分组第一拍数据用寄存器进行缓存。

**S\_COM\_S：**分组源MAC比较状态机。将PKE传输过来的key[53:6]与direct\_mac进行比较，若{(key[53:6]==direct\_mac)&&(key[5:0]==2)}| (key[53:6]!=direct\_mac)则跳转到D\_COM\_S；若{(key[53:6]==direct\_mac)&(key[5:0]！=2)}则跳转到DICS\_S。若源MAC与本地直连设备地址相等，将寄存器flag置高。同时将分组第二拍数据用寄存器进行缓存。

**D\_COM\_S：**分组目的MAC比较状态机。将PKE传输过来的key[101:54]与direct\_mac进行比较，若key[101:54]==direct\_mac则将action[5:0]填为2；若key[101:54]！=direct\_mac则再根据flag位：若flag为低，则将action[5:0]填为~key[0]；若flag为高（代表分组是来自本地直连设备或本地LCM），则将action[5:0]填为direction。跳转到TRANS\_S。将分组第一拍数据进行传输并将第三拍数据进行缓存。若pgm\_flag为高，则转发第一拍数据并缓存第二拍，跳转到TRANS\_S。

**TRANS\_S：**传输状态机。将分组数据传输给PAC模块，直到分组全部传输完成，可根据分组的头尾标志进行判断，跳转到IDLE\_S。

**DICS\_S：**丢弃状态机。将分组进行丢弃，直到分组全部丢弃完成，跳转到IDLE\_S。

#### 3.6.3.4 PAC模块

##### 3.6.3.4.1 功能分析

根据ACTION中的内容进行分组的转发。ACTION中的分组输出目的若为1则将元数据根据ACTION进行修改之后将分组往本地LCM传输，若是PTP分组还需往TAP复制一份转发；若ACTION的分组输出目的为0且输出端口为2，则元数据根据ACTION进行修改之后将分组往本地直连设备传输，同时复制一份分组发往TAP口；若ACTION的分组输出目的为0且输出端口不为2，则根据Bufm的空闲ID进行分组的丢弃或转发。当Bufm大于2，分组都可直接进行元数据修改之后传输分组；当Bufm大于值1小于值2时，分组类型为Best Effort的需要进行丢弃，其他分组进行正常传输；当Bufm小于值1时，分组类型为预约带宽或Best Effort的都需要进行丢弃，其他分组进行正常传输。不丢弃时，在分组传输时需要构造TSN-Metadata。在所有需要进行转发的分组都复制一份转发给TAP口。

##### 3.6.3.4.2 接口定义

PAC模块的接口信号定义图如图 所示。



图 PAC模块的接口信号定义图

PAC模块的接口信号定义表如表 所示。

表 PAC模块的接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| From PFW | | | |
| in\_pac\_data | 134 | input | 分组数据输入 |
| in\_pac\_data\_wr | 1 | input | 分组数据输入有效，高有效 |
| in\_pac\_valid | 1 | input | 分组数据输入有效位 |
| in\_pac\_valid\_wr | 1 | input | 分组数据输入效位有效，高有效 |
| in\_pac\_action | 9 | input | action数据输入 |
| in\_pac\_action\_wr | 1 | input | action数据输入有效，高有效 |
| To IBM | | | |
| out\_pac\_data | 134 | output | 分组数据往下级模块输出 |
| out\_pac\_data\_wr | 1 | output | 分组数据往下级模块输出有效，高有效 |
| out\_pac\_valid | 1 | output | 分组数据往下级模块输出有效位 |
| out\_pac\_valid\_wr | 1 | output | 分组数据往下级模块输出效位有效，高有效 |
| out\_pac\_tsn\_md | 24 | output | TSN\_MD数据输出 |
| out\_pac\_tsn\_md\_wr | 1 | output | TSN\_MD数据输出有效，高有效 |
| bufm\_ID\_count | 5 | input | Bufm中空闲的ID计数器 |
| To LCM | | | |
| bufm\_ID\_cnt | 8 | output | ESW模块输出的Bufm空闲ID计数器 |
| esw\_pktout\_cnt | 64 | output | ESW模块输出分组计数器 |
| To GOE | | | |
| out\_pac2port\_data\_N | 134 | output | 分组数据往网口输出（N=2/3） |
| out\_pac2 port\_data\_wr\_N | 1 | output | 分组数据往网口输出有效，高有效（N=2/3） |
| out\_pac2 port\_valid\_N | 1 | output | 分组数据往网口输出有效位（N=2/3） |
| out\_pac2 port\_valid\_wr\_N | 1 | output | 分组数据往网口输出效位有效，高有效（N=2/3） |

##### 3.6.3.4.3 模块实现

根据对PAC模块的功能分析，可直接用一个状态机实现。PAC模块的状态机如图 所示。



图 PAC模块的状态机图

**IDLE\_S：**空闲状态机。等待分组进行，根据action的内容对分组的元数据进行修改。再根据action[5:0]的值：action[5:0]==2则跳转到DIR\_S，并将分组数据直接传输给GOE模块；若action[5:0]！=2则跳转到TRA\_S，并将分组第一拍数据用寄存器进行缓存。

**DIR\_S：**本地直连设备传输状态机。本地直连设备传输状态机。将分组传输给GOE，直到分组全部传输完成，跳转到IDLE\_S。

**TRA\_S：**流量监管状态机。若满足{(action[8:6]==0)&&Bufm\_ID\_cnt==2}|(action[8:6]==0/1)&&Bufm\_ID\_cnt=1则跳转到DICS\_S；否则跳转到TRANS\_S。将分组第二拍数据用寄存器进行缓存，并构造TSN\_MD。

**TRANS\_S：**网口传输状态机。将分组往下级模块进行转发直到传输完成，同时TSN\_MD数据传输给下级模块。跳转到IDLE\_S。

**DICS\_S：**丢弃状态机。将分组进行丢弃，知道分组全部丢弃完成，跳转到IDLE\_S。

由于还需将所有的分组复制一份发往TAP口，因此添加一个always块来实现：根据两组输出信号的“数据有效”信号来判断当前传输的分组是什么流向的，则相应地将这组输出信号复制一份给TAP口。

## 3.7 EOS模块详细设计

### 3.7.1 需求与功能分析

为了支持基于FAST的TSN交换，需要在EOS模块内实现：

1. 分队列缓存不同类型报文的元数据（TSN-Metadata），即：使用Q0、Q1队列缓存TSN报文的元数据；使用Q0队列缓存PTP和预约带宽报文的元数据，并支持对PGM模块产生的并发流的缓存；使用Q3队列缓存Best Effort的以太网报文的元数据；
2. 根据报文优先级调度队列，Q0= Q1> Q2> Q3；
3. 实现CQF调度功能，即：使用Q0、Q1实现乒乓队列；
4. 使用令牌桶对带宽预约流量（即：队列Q2的元数据）进行整形；
5. 配置令牌桶参数，目前包括两个参数，即：令牌桶的桶深B和向令牌桶内添加令牌的速率R；
6. 允许LCM模块读取EOS模块内相关寄存器的状态信息。

### 3.7.2 EOS模块概要设计

本节主要介绍EOS模块的整体架构设计，EOS模块的整体架构设计如图 所示。



图 EOS模块整体架构设计图

其中，EOS模块的接口信号定义如表 所示。

表 EOS模块接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| QS - LCM接口信号定义 | | | |
| time\_slot\_flag | 1 | LCM -> QS | 时间槽切换信号。当前位于奇数时间槽为1，位于偶数时间槽为0 |
| MB - LCM接口信号定义 | | | |
| q0\_used\_cnt | 6 | MB -> LCM | MB子模块Q0队列的已使用长度的计数器 |
| q1\_used\_cnt | 6 | MB -> LCM | MB子模块Q1队列的已使用长度的计数器 |
| q2\_used\_cnt | 6 | MB -> LCM | MB子模块Q2队列的已使用长度的计数器 |
| q3\_used\_cnt | 6 | MB -> LCM | MB子模块Q3队列的已使用长度的计数器 |
| GC - LCM接口信号定义 | | | |
| time\_slot\_flag | 1 | LCM -> GC | 时间槽切换信号。当前位于奇数时间槽为1，位于偶数时间槽为0 |
| tb\_depth | 32 | LCM -> GC | Q2的令牌桶深度 |
| rate\_limit | 32 | LCM -> GC | Q2的令牌桶速率（每800ns向令牌桶内添加令牌的数量；支持的最小带宽为10Mbps） |
| EOS to LCM接口信号定义 | | | |
| mdin\_cnt | 64 | EOS -> LCM | 进入EOS模块的元数据计数器 |
| mdout\_cnt | 64 | EOS -> LCM | EOS模块输出的元数据计数器 |
| QS - IMB接口信号定义 | | | |
| md | 24 | IBM -> QS | 元数据输入 |
| md\_wr | 1 | IBM -> QS | 元数据输入有效信号 |
| QS – MB接口信号定义 | | | |
| md0 | 9 | QS -> MB | Q0队列元数据 |
| md1 | 9 | QS -> MB | Q1队列元数据 |
| md2 | 21 | QS -> MB | Q2队列元数据 |
| md3 | 9 | QS -> MB | Q3队列元数据 |
| md\_wr | 4 | QS -> MB | Q0、Q1、Q2、Q3队列元数据有效写信号 |
| MB – GC接口信号定义 | | | |
| md\_outport | 4 | MB -> GC | Q0、Q1、Q2、Q3队列元数据输出端口 |
| fifo\_empty | 4 | MB -> GC | Q0、Q1、Q2、Q3队列空信号 |
| pkt\_len | 12 | MB -> GC | Q2队列元数据对应的报文需要消耗的令牌数（1字节消耗1个令牌） |
| GC – EBM接口信号定义 | | | |
| pkt\_valid | 1 | EBM -> GC | EBM中分组最后一拍输出信号 |
| bandwidth\_discard | 1 | GC -> EBM | 带宽预约分组丢弃信号 |
| GC – FPGA OS Egress接口信号定义 | | | |
| pktout\_usedw\_0 | 8 | GC -> FPGA OS | 0号端口FIFO已经存储的报文深度 |
| pktout\_usedw\_1 | 8 | GC -> FPGA OS | 1号端口FIFO已经存储的报文深度 |
| GC – TS接口信号定义 | | | |
| schedule\_valid | 4 | GC -> TS | MB模块的4个队列可调度信号 |
| q2\_rden | 1 | TS -> GC | Q2队列元数据的读信号 |
| MB - TS接口信号定义 | | | |
| md | 8 | MB -> TS | 队列元数据 |
| md\_wr | 1 | MB -> TS | 队列元数据有效信号 |
| rden | 4 | TS -> MB | Q0、Q1、Q2、Q3队列元数据读信号 |
| TS – EBM接口信号定义 | | | |
| md | 8 | TS -> EBM | 元数据 |
| md\_wr | 1 | TS -> EBM | 元数据有效信号 |

如图 所示，EOS包含了4个处理流程，即：队列选择（Queue Selecting，QS）模块负责元数据的队列选择；元数据缓存（Metadata Buffer）模块负责缓存元数据；门控（Gate Control，GC）模块负责判断4个队列是否可以被调度，同时支持令牌桶算法对Q2队列进行流量整形；转发调度（Transmitting and Scheduling，TS）模块负责根据优先级进行调度操作。



（a）QS模块处理流程图



（b）MB模块处理流程图



（c）GC模块处理流程图



（d）TS模块处理流程图

图 EOS子模块处理流程图

每一个功能子模块的具体功能如下所示：

**QS模块：**队列选择（Queue Selecting）模块，负责将不同的元数据写入相对应的队列中。根据当前时间槽的奇偶和元数据TSN-Metadata[23:21]类型决定将元数据传输到哪个队列：

1. 若TSN-Metadata[23:21]为0，则将TSN-Metadata[8:0]传输给Q3队列；
2. 若TSN-Metadata[23:21]为1，则将TSN-Metadata[20:9]对应的pkt\_len减去两拍的FAST-Metadata长度（32字节），并连同TSN-Metadata[8:0]传输给Q2队列；
3. 若TSN-Metadata[23:21]为2，则将TSN-Metadata[20:9]置0（PTP分组无需进行流量整形，不需要消耗令牌），并连同TSN-Metadata[8:0]传输给Q2队列；
4. 若TSN-Metadata[23:21]为3且当前为奇数时间槽，则将TSN-Metadata[8:0]传输到Q1队列，当前为偶数时间槽，则将TSN-Metadata[8:0]传输到Q0队列；
5. 若TSN-Metadata[23:21]为4，则将TSN-Metadata[20:9]对应的pkt\_len减去两拍的FAST-Metadata长度（32字节），并连同TSN-Metadata[8:0]传输给Q2队列；

**MB模块：**元数据缓存（Metadata Buffer）模块。其中，Q0队列缓存TSN分组的元数据的偶数时钟队列；Q1队列缓存TSN分组的元数据的奇数时钟队列；Q2队列缓存带宽预约分组、PTP分组以及PGM模块生成的并发流的元数据的队列；Q3缓存Best Effort流量的元数据的队列。同时，等待TS模块的读信号，将相应队列的元数据读出。

**GC模块：**门控（Gate Control）模块，负责令牌桶中的令牌计数；同时，判断带宽预约流量是否应该被丢弃，并将判断结果传给EBM模块，以及判断MB中的4个队列是否可以被调度，并将决策结果传递给TS模块。

**TS模块：**转发调度（Transmitting and Scheduling）模块。TS模块根据GC模块传来的判断结果，对可进行调度的队列采用绝对优先级的调度策略来调度MB模块中的元数据。

### 3.7.3 EOS模块详细设计

#### 3.7.3.1 EOS顶层模块设计

##### 3.7.3.1.1模块接口设计

EOS顶层模块接口信号定义图如图 所示。



图 EOS模块外部接口信号定义图

接口信号定义表如表 所示。

表 EOS模块外部接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟信号 |
| rst\_n | 1 | Input | 复位信号，低有效 |
| EOS – LCM接口信号定义 | | | |
| in\_eos\_time\_slot\_flag | 1 | Input | 时间槽切换信号，当前位于奇数时间槽为1，偶数时间槽为0 |
| in\_eos\_tb\_rate | 32 | Input | 每800ns往令牌桶中添加令牌的数量；支持的最小带宽为10Mbps |
| out\_eos\_q0\_used\_cnt | 6 | Output | MB模块Q0队列的已使用长度计数器 |
| out\_eos\_q1\_used\_cnt | 6 | Output | MB模块Q1队列的已使用长度计数器 |
| out\_eos\_q2\_used\_cnt | 6 | Output | MB模块Q2队列的已使用长度计数器 |
| out\_eos\_q3\_used\_cnt | 6 | Output | MB模块Q3队列的已使用长度计数器 |
| out\_eos\_mdin\_cnt | 64 | Output | 进入EOS模块的元数据计数器 |
| out\_eos\_mdout\_cnt | 64 | Output | EOS模块输出的元数据计数器 |
| EOS – IBM接口信号定义 | | | |
| in\_eos\_md | 24 | Input | 元数据输入 |
| in\_eos\_md\_wr | 1 | Input | 元数据输入有效信号 |
| EOS - FPGA OS Egress接口信号定义 | | | |
| pktout\_usedw\_0 | 8 | Input | 0号端口fifo已存储的报文深度 |
| pktout\_usedw\_1 | 8 | Input | 1号端口fifo已存储的报文深度 |
| EOS – EBM接口信号定义 | | | |
| out\_eos\_md | 8 | Output | 元数据输出 |
| out\_eos\_md\_wr | 1 | Output | 元数据输出有效信号 |
| in\_eos\_pkt\_valid | 1 | Input | EBM中分组最后一拍输出信号 |
| out\_eos\_bandwidth\_discard | 1 | Output | 带宽预约分组丢弃信号 |

##### 3.7.3.1.2 模块实现

EOS模块内部所有接口信号以及架构设计如图 所示。



注：图中MB传给LCM的N\_used\_cnt[5:0]中的N为q0、q1、q2、q3。

需要一个always块来实现对进入EOS模块的元数据和EOS模块输出的元数据进行计数。

#### 3.7.3.2 QS模块设计

QS模块功能是选择将元数据写入的队列。根据当前时间槽的奇偶和元数据 [23:21]类型决定将元数据传输给MB模块4个队列中的哪一个。

##### 3.7.3.2.1 模块接口设计

QS模块的接口信号定义图如图 所示。



图 QS模块接口信号定义图

信号接口定义表如表 所示。

表 QS模块接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟 |
| rst\_n | 1 | Input | 复位，低有效 |
| QS – EOS接口信号定义 | | | |
| in\_qs\_time\_slot\_flag | 1 | Input | 时间槽切换信号，当前位于奇数时间槽为1，偶数时间槽为0 |
| in\_qs\_md | 24 | Input | 元数据输入 |
| in\_qs\_md\_wr | 1 | Input | 元数据输入有效信号 |
| QS – MB接口信号定义 | | | |
| out\_qs\_md0 | 9 | Output | Q0队列元数据输出 |
| out\_qs\_md1 | 9 | Output | Q1队列元数据输出 |
| out\_qs\_md2 | 16 | Output | Q2队列元数据输出 |
| out\_qs\_md3 | 9 | Output | Q3队列元数据输出 |
| out\_qs\_md\_wr | 4 | Output | Q0、Q1、Q2、Q3队列元数据输出有效信号 |

##### 3.7.3.2.2 模块实现

根据上述的QS子模块元数据的队列选择功能，可使用一个always块来实现。具体功能实现如下。

QS模块等待元数据到来，判断该元数据[23:21]类型：

1. 若为4，则将输出到Q2队列的写信号置1，同时将TSN-Metadata [20:9]pkt\_len减去两拍FAST-Metadata长度（32字节），并连同元数据[8:0]输出到Q2队列；
2. 若为3且当前处于奇数时间槽，则将输出到Q1队列的写信号置1，TSN-Metadata[8:0] 输出到Q1队列；
3. 若为3且当前处于偶数时间槽，则将输出到Q0队列的写信号置1，TSN-Metadata [8:0] 输出到Q0队列；
4. 若为2，则将输出到Q2队列的写信号置1，同时将TSN-Metadata [20:9]pkt\_len置0（PTP分组不需要进行流量整形，不需要消耗令牌），并连同TSN-Metadata [8:0]输出到Q2队列；
5. 若为1，则将输出到Q2队列的写信号置1，同时将TSN-Metadata [20:9]pkt\_len减去两拍FAST-Metadata长度（32字节），并连同元数据[8:0]输出到Q2队列；
6. 若TSN-Metadata [23:21]为0，则将输出到Q3队列的写信号置1，TSN-Metadata [8:0]输出到Q3队列。

#### 3.7.3.3 MB模块设计

MB模块进行元数据缓存，Q0为缓存TSN元数据的偶数时钟队列，Q1为缓存TSN元数据的奇数时钟队列，Q2为缓存带宽预约元数据、PTP元数据以及PGM产生并发流元数据的队列，Q3为缓存best effort元数据的队列。等待TS的元数据读信号，将相应队列的元数据读出。

##### 3.7.3.3.1 模块接口设计

MB模块的接口信号定义图如图 所示。



图 MB模块接口信号定义图

MB模块的接口信号定义表如表 所示。

表 MB模块接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟 |
| rst\_n | 1 | Input | 复位，低有效 |
| MB – EOS接口信号定义 | | | |
| out\_mb\_q0\_used\_cnt | 6 | Output | MB模块Q0队列的已使用长度计数器 |
| out\_mb\_q1\_used\_cnt | 6 | Output | MB模块Q1队列的已使用长度计数器 |
| out\_mb\_q2\_used\_cnt | 6 | Output | MB模块Q2队列的已使用长度计数器 |
| out\_mb\_q3\_used\_cnt | 6 | Output | MB模块Q3队列的已使用长度计数器 |
| MB – QS接口信号定义 | | | |
| in\_mb\_md0 | 9 | Input | Q0队列元数据输入 |
| in\_mb\_md1 | 9 | Input | Q1队列元数据输入 |
| in\_mb\_md2 | 21 | Input | Q2队列元数据输入 |
| in\_mb\_md3 | 9 | Input | Q3队列元数据输入 |
| in\_mb\_md\_wr | 4 | Input | Q0、Q1、Q2、Q3队列元数据输入有效信号 |
| MB – TS接口信号定义 | | | |
| out\_mb\_md | 8 | Output | 队列元数据输出 |
| out\_mb\_md\_wr | 1 | Output | 队列元数据输出有效信号 |
| in\_mb\_md\_rden | 4 | Input | Q0、Q1、Q2、Q3队列元数据读使能信号 |
| MB – GC接口信号定义 | | | |
| out\_mb\_md\_outport | 4 | Output | Q0、Q1、Q2、Q3队列元数据输出端口 |
| out\_mb\_fifo\_empty | 4 | Output | Q0、Q1、Q2、Q3队列空信号 |
| out\_mb\_pkt\_len | 12 | Output | Q2队列元数据对应的报文需消耗的令牌数（1字节消耗1个令牌） |

##### 3.7.3.3.2 模块实现

MB模块只是进行元数据的缓存，实现较简单，不需要使用状态机；关键是给4个队列的FIFO设置合适的位宽和深度，分析计算过程如下:

MB模块任意时刻的元数据最大缓存量的分析计算：为了给Q0和Q1队列设置合适的队列长度和给BufM缓存区分配合适的ID数量， 需要计算MB模块任意时刻的元数据最大缓存量，因为BufM的一个ID对应MB的一个元数据，并且一个ID对应2Kb的存储空间，足够缓存一个最长分组。根据乒乓队列的读写特点，只要保证在一个时间槽内BufM能缓存全部的TSN分组便可，所以只需计算MB模块在一个时间槽内TSN元数据的最大缓存量；考虑极端情况，在一个时间槽内，当从端口进入的TSN报文全为最短报文（64字节），以太网最小帧间距是12字节，且TSN流量达到链路负载的最大比例5%（50Mbps）时，MB模块的TSN元数据的缓存量最大。假设时间敏感流量乒乓队列切换的时间槽为125us（802.1Qch中给出的典型切换时间），因此Q0和Q1每个队列缓存深度（ID数量）最大需要

(125us \* 5% \* 1Gbps) / ((64B + 12B) \* 8) = 11

由上述分析知，Q0和Q1队列长度最大需要11，FIFO深度可设置为16；Q2、Q3队列的FIFO深度不应低于Bufm中ID数量，可设置为16。

综上分析：4个队列的FIFO位宽、深度设置如表 所示。

表 4个队列的FIFO参数表

|  |  |  |
| --- | --- | --- |
| **队列FIFO** | **位宽** | **深度** |
| Q0 | 9 | 16 |
| Q1 | 9 | 16 |
| Q2 | 21 | 16 |
| Q3 | 9 | 16 |

#### 3.7.3.4 GC模块设计

GC模块负责令牌桶中的令牌计数以及判断MB的4个队列是否可被调度，把判断结果输给TS模块。

##### 3.7.3.4.1 模块接口设计

GC模块的接口信号定义图如图 所示。



图 GC模块的接口信号定义图

具体接口信号定义表如表 所示。

表 GC模块的接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟信号 |
| rst\_n | 1 | Input | 复位信号，低有效 |
| GC – EOS接口信号定义 | | | |
| in\_gc\_time\_slot\_flag | 1 | Input | 时间槽切换信号，当前位于奇数时间槽为1，偶数时间槽为0 |
| in\_gc\_rate\_limit | 32 | Input | 每800ns往令牌桶中添加令牌的数量；支持的最小带宽为10Mbps |
| in\_gc\_pkt\_valid | 1 | Input | EBM中分组最后一拍输出信号 |
| out\_gc\_bandwidth\_discard | 1 | Output | 带宽预约分组丢弃信号 |
| pktout\_usedw\_0 | 8 | Input | 0号端口fifo已存储的报文深度 |
| pktout\_usedw\_1 | 8 | Input | 1号端口fifo已存储的报文深度 |
| GC – MB接口信号定义 | | | |
| in\_gc\_md\_outport | 4 | Input | Q0、Q1、Q2、Q3队列元数据输出端口 |
| in\_gc\_fifo\_empty | 4 | Input | Q0、Q1、Q2、Q3队列空信号 |
| in\_gc\_pkt\_len | 12 | Input | Q2队列元数据对应的报文需消耗的令牌数（1字节消耗1个令牌） |
| GC – TS接口信号定义 | | | |
| in\_gc\_q2\_rden | 1 | Input | Q2元数据的读信号，用于判断TS是否调度了Q2的元数据，令牌桶中是否应减去相应的令牌 |
| out\_gc\_schedule\_valid | 4 | Output | Q0、Q1、Q2、Q3队列元数据可调度信号，高有效 |

##### 3.7.3.4.2 模块实现

门控模块负责令牌桶中的令牌计数，并根据令牌桶中的剩余令牌、pkt\_valid、time\_slot\_flag、fifo\_empty、md\_outport、pkt\_len和pktout\_usedw\_N(N=0、1)来判断MB的4个队列是否可被调度，把判断结果输给TS模块。

为了实现上述功能，需要3个always块。每个always块的具体功能实现如下：

1. 一个always块用来判断Q0、Q1、Q2、Q3是否可被调度，可使用一个状态机进行控制。其状态机如图 所示。



图 判断4个队列是否可被调度的状态机

**IDLE\_S：**起始状态。当pkt\_valid==1’b1 || init\_flag==1’b1时，跳转到JUDGE\_Q0\_S。（init\_flag作用：为了使FPGA开始工作时，队列能被调度，设置寄存器init\_flag，复位时置为1。）

JUDGE\_QUEUE\_S：判断状态。若schedule\_valid[3:0]==1’b1，将schedule\_valid[3:0]置0，init\_flag置0，跳转到IDLE\_S；否则留在JUDGE\_QUEUE\_S，并判断Q0、Q1、Q2、Q3队列是否可被调度，具体判断情况如下。

* 判断Q0是否可被调度，若满足以下3个条件：

1） time\_slot\_flag = =1；

2） Q0的empty == 0；

3） 输出端口为N且pktout\_usedw\_N < a；

则将schedule\_valid[0]置1；否则将schedule\_valid[0]置0。

* 判断Q1是否可被调度，若满足以下3个条件：

1） time\_slot\_flag == 0；

2） Q1的empty == 0；

3） 输出端口为N且pktout\_usedw\_N < a；

则将schedule\_valid[1]置1；否则将schedule\_valid[1]置0。

* 判断Q2是否可被调度，若满足以下2个条件：

1） Q2的empty == 0；

2） 输出端口为N且pktout\_usedw\_N<a；

则将schedule\_valid[2]置1，；否则将schedule\_valid[2]置0。

* 判断Q3是否可被调度，若满足以下2个条件：

1） Q3的empty == 0；

2） 输出端口为N且pktout\_usedw\_N<a；

则将schedule\_valid[3]置1；否则将schedule\_valid[3]置0。

1. 一个always块用来对令牌桶中令牌进行计数(为了避免在两个always块的上升沿同时对CT进行操作，在此always块设置下降沿有效)。设置令牌桶中一个令牌代表 1Byte；令牌桶的容量为BC（可容纳4095个令牌），令牌桶中剩余令牌为RT，在调度Q2队列元数据时，需消耗的令牌为CT，令牌桶的计时器为TB\_cnt。每经100\*8ns(TB\_cnt计100个时钟下降沿)，往令牌桶中添加

AT=100\*8\* (token\_bucket\_rate/8)/ (10^9)= token\_bucket\_rate /(10Mbps)

个令牌(支持的最小带宽为10Mpbs)；若RT+AT-CT<BC，添加令牌后令牌桶中的剩余令牌为RT+AT-CT；若RT+AT-CT≥BC(令牌数量超过桶的容量)，添加令牌后令牌桶中的剩余令牌为BC。

1. 一个always块(上升沿有效)用来判断令牌是否要被消耗，带宽预约流量是否要被丢弃。若TS输出给Q2的读信号q2\_rden为1且RT≥报文字节数，则将该报文字节数赋给CT，不丢弃该带宽预约流量；若q2\_rden为1且RT<报文字节数，则将CT置0，丢弃该带宽预约流量。

#### 3.7.3.5 TS模块设计

TS模块根据GC模块传来的判断结果，对可进行调度的队列采用绝对优先级调度策略选择发送元数据，优先级为TSN元数据>带宽预约元数据=PTP元数据=PGM生成并发流量元数据>best effort元数据；若Q0或Q1队列是可被调度的，则将该队列的元数据读出；若Q0或Q1队列是不可被调度的，Q2队列是可被调度的，则将Q2的元数据读出；若Q0或Q1队列、Q2队列是不可被调度的，Q3队列是可被调度的，则将Q3的元数据读出。

##### 3.7.3.5.1 模块接口设计

TS模块的接口信号定义图如图 所示。



图 TS模块接口信号定义图

TS模块的接口信号定义表如表 所示。

表 TS模块接口信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **位宽** | **方向** | **备注** |
| clk | 1 | Input | 时钟信号 |
| rst\_n | 1 | Input | 复位信号，低有效 |
| TS – MB接口信号定义 | | | |
| in\_ts\_md | 8 | Input | 元数据输入 |
| in\_ts\_md\_wr | 1 | Input | 元数据输入有效信号 |
| out\_ts\_md\_rden | 4 | Input | Q0、Q1、Q2、Q3队列元数据读信号 |
| TS – GC接口信号定义 | | | |
| in\_ts\_schedule\_valid | 4 | Input | Q0、Q1、Q2、Q3队列元数据可被调度信号 |
| out\_ts\_q2\_rden | 1 | Output | TS输出给Q2的读信号，用于判断TS是否调度了Q2的元数据，在GC模块中令牌桶中是否应减去相应的令牌 |
| TS – EOS接口信号定义 | | | |
| out\_ts\_md | 8 | Output | 元数据输出 |
| out\_ts\_md\_wr | 1 | Output | 元数据输出有效信号 |

##### 3.7.3.5.2 模块实现

根据上述的TS子模块队列元数据调度发送的功能，可使用2个always块实现。每个always块的具体功能实现如下。

1. 一个always块用于实现队列调度功能。若Q0的schedule\_valid为1，则将输出给Q0队列的读信号置1；若Q1的schedule\_valid为1，则将输出给Q1队列的读信号置1；若Q0、Q1的schedule\_valid都为0，Q2的schedule\_valid为1，则将输出给Q2队列的读信号置1；若Q0、Q1、Q2的schedule\_valid都为0，Q3的schedule\_valid为1，则将输出给Q3队列的读信号置1。
2. 一个always块用于输出元数据。等待输入的元数据有效信号为1，将元数据的[7:0]输出。

##### 3.7.3.5.3 UDO端口a的计算

为了使报文到达UDO相应端口时，该端口有足够的空闲空间来存储报文，需计算UDO中的fifo剩余报文拍数a最大为多少时，TS才能调度元数据。在UDO中端口每个时钟周期输出8bit，传128bit需要16个时钟周期，从TS发出读信号给元数据缓存队列到报文写入相应端口的fifo需要16个时钟周期左右，这段时间内端口发完一个128bit；GOE到UDO至多2拍；而fifo深度128-最长报文拍数95=33，所以a为33+1-2=32。

# 4 可读写寄存器设计

表 FAST-ANT 2.0可读写寄存器列表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **所属模块** | **寄存器名称** | **读写** | **寄存器大小** | **寄存器初始值** | **寄存器描述** |
| EOS，ESW | direction | 读写 | 1 | 1’b0 | 报文转发端口 |
| ESW | direct\_mac\_addr | 读写 | 48 | 48’b0 | TSN节点直连设备MAC地址。 |
| ESW | esw\_pktin\_cnt | 读 | 64 | 64’b0 | 进入ESW模块的分组计数器。 |
| ESW | esw\_pktout\_cnt | 读 | 64 | 64’b0 | ESW模块输出的分组计数器。 |
| ESW | local\_mac\_addr | 读 | 8 | 8’b0 | TSN本地MAC地址。 |
| ESW | bufm\_ID\_cnt | 读 | 8 | 8’b0 | Bufm中所使用的ID计数器 |
| EOS | token\_bucket\_para | 读写 | 32 | 32’b0 | 令牌桶R |
| EOS | token\_bucket\_depth | 读写 | 32 | 32’b0 | 令牌桶B |
| EOS | time\_slot\_flag | 读 | 1 | 1’b0 | 奇偶时间槽，当前位于奇数时间槽为1；偶数时间槽为0。 |
| EOS | eos\_mdin\_cnt | 读 | 64 | 64’b0 | 进入EOS模块的元数据计数器 |
| EOS | eos\_mdout\_cnt | 读 | 64 | 64’b0 | EOS模块输出的元数据计数器 |
| EOS | eos\_q0\_used\_cnt | 读 | 6 | 6’b0 | EOS模块Q0队列已使用长度计数器 |
| EOS | eos\_q1\_used\_cnt | 读 | 6 | 6’b0 | EOS模块Q1队列已使用长度计数器 |
| EOS | eos\_q2\_used\_cnt | 读 | 6 | 6’b0 | EOS模块Q2队列已使用长度计数器 |
| EOS | eos\_q3\_used\_cnt | 读 | 6 | 6’b0 | EOS模块Q3队列已使用长度计数器 |
| GOE | goe\_pktin\_cnt | 读 | 64 | 64’b0 | 进入GOE模块的分组计数器 |
| GOE | goe\_port0out\_cnt | 读 | 64 | 64’b0 | GOE模块往0口输出的分组计数器 |
| GOE | goe\_port1out\_cnt | 读 | 64 | 64’b0 | GOE模块往1口输出的分组计数器 |
| GOE | goe\_discard\_cnt | 读 | 64 | 64’b0 | GOE模块丢弃的分组计数器 |
| UDC | lcm2udc\_addr | 写 | 20 | 20’b0 | 读取报文的地址 |
| UDC | ram\_rd | 写 | 1 | 1’b0 | 读请求信号 |
| SCM | protocol\_type | 写 | 8 | 8’b0 | 特定报文类型 |
| SCM | statistic\_reset | 写 | 1 | 1’b0 | 重置信号 |
| SCM | scm\_pkt\_num\_cnt | 读 | 64 | 64’b0 | 记录总接收报文数量 |
| SCM | scm\_bit\_num\_cnt | 读 | 64 | 64’b0 | 记录总接收bit数量 |
| SCM | time\_latency | 读 | 32 | 32’b0 | 记录处理延迟 |

# 5 进一步工作与开发计划

## 5.1 进一步工作

1. 在OpenBox-S4上实现FAST-ANT 2.0原型系统后，移植至40G NP并测试相关功能是否正常、相关指标是否满足；
2. 在40G NP上进行硬件功能模块的细化，充分发挥不同FPGA的优势，将高性能和精度相关的功能模块移植到Altera FPGA上，将发送控制模块移植到Xilinx FPGA上；
3. 关于FAST架构的思考：在FPGA OS提供类似于OSNT的Wrapper，支持端口和流水线之间的自定义映射（多对多映射关系）。UM部分只给出标准的数据帧的位宽，而数据帧的内容格式是如何定义的则完全交给开发者自定义。这样，FAST将UM部分进一步开放，开发者可以在UM内实现更多自定义的网络功能，无需遵循之前传统的五级流水线设计，简化了UM内部的结构。而多对多映射的实现可以支持多条流水线并行处理，也可以支持开发者对硬件调度的相关研究和开发；
4. 关于FAST-ANT的思考：随着测试场景和需求的不断复杂化和多样化，测试功能也在不断丰富扩充。部分功能为了充分利用硬件的特性，例如：偏向于最大化发挥精度或性能，而通用的架构则不再适用，定制化的内部架构更是一种趋势。硬件FPGA可重构化则是一种发展趋势。针对不同的硬件功能生成对应的硬件烧录温江，软件端则根据用户测试需求，编写加载脚本，支持动态重构FPGA，真正实现软件配置、硬件可重构的测试架构。

## 5.2 开发计划

### 5.2.1 项目人员

孙志刚，全巍，徐东来，杨翔瑞，蒋越，吴尚明。

### 5.2.2 项目分工

1. ***设计文档指导***

孙志刚，全巍，杨翔瑞；

1. ***项目开发指导***

徐东来，杨翔瑞；

1. ***软件开发***

蒋越：

1. Linux端和硬件FPGA的LCM模块之间的通信；
2. 图形化界面输入和输出的数据结构。

华芯通：

1. FAST-ANT 2.0软件端图形化界面。
2. ***硬件开发***

蒋越：

1. LCM模块设计实现，定义数据帧格式；
2. PGM模块重写，复用FAST-TSN中令牌桶调度部分。

吴尚明：

1. SCM模块重新设计实现，并增加新的统计功能；
2. UDC模块设计实现；

### 5.2.3 进度安排

|  |  |  |
| --- | --- | --- |
| 日期 | 工作内容 | 参与人员 |
| 2019.05.27 | 针对设计文档2.0进行讨论修改。 | 孙志刚，全巍，徐东来，杨翔瑞，蒋越，吴尚明 |
| 2019.05.28  ~  2019.05.30 | 完成详细设计部分。  软件部分完成核心数据结构、函数设计；  硬件部分完成核心功能模块接口信号和状态机设计。 | 蒋越，吴尚明 |
| 2019.05.31 | 针对设计文档2.1进行讨论修改。 | 孙志刚，全巍，徐东来，杨翔瑞，蒋越，吴尚明 |
| 2019.06.01  ~  2019.06.02 | 完善软件和硬件详细设计部分。 | 蒋越，吴尚明 |
| 2019.06.03  ~  2019.06.16 | 开发硬件功能模块并进行仿真测试。 | 蒋越，吴尚明 |
| 2019.06.17  ~  2019.07.05 | 开发软件部分并测试；  上板联合调试软硬件代码。 | 蒋越，吴尚明，杨翔瑞，徐东来 |