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；  完善架构对比分析。 |
| 2.3.1 | 蒋越 | 2019.06.07 | 完善PGM模块设计。 |
| 2.3.2 | 吴尚明 | 2019.06.08 | 修改UDC接口信号设计以及RAM读取操作。 |
| 2.3.3 | 蒋越、吴尚明 | 2019.06.09 | 修改UDC状态机设计；  完善PGM\_RD模块的详细设计与实现。 |
| 2.4 | 蒋越、吴尚明 | 2019.06.10 | 根据修改意见，重新设计硬件架构部分；  合并UDC模块和SCM模块为SSM模块；  重新设计并发流量生成部分，去掉FAST-TSN中令牌桶调度部分。 |
| 2.5 | 蒋越、吴尚明 | 2019.06.12 | 根据修改意见，保持现有功能模块不变，重新设计硬件架构；  细化SSM模块、LCM模块以及PGM模块设计，包括子模块划分，状态机简化等；  对并发流量分为八大类别，并给出每一类别五元组变化的方式。 |
| 2.5.1 | 蒋越、吴尚明 | 2019.06.16 | 修改RAM设计；  订正状态机图。 |
| 2.5.2 | 蒋越、吴尚明 | 2019.06.17 | 调整设计文档架构，明确需求分析与概要设计；  重构硬件架构设计图。 |
| 2.6 | 蒋越、吴尚明 | 2019.06.20 | 订正文档。 |
| 2.6.1 | 蒋越 | 2019.06.24 | 修改PGM模块的SCHEDULE子模块中的时刻调度表实现部分。 |
| 2.6.2 | 蒋越 | 2019.06.25 | 修改部分模块接口信号定义。 |
| 2.6.3 | 吴尚明 | 2019.06.28 | 修改SSM模块状态机设计。 |
| 2.6.4 | 蒋越 | 2019.07.01 | 修改DMUX模块设计；  修改PGM模块部分子模块状态机描述。 |
| 2.7 | 蒋越 | 2019.07.08 | 完成软件端架构设计。 |
| 2.7.1 | 蒋越 | 2019.07.09 | 完善软件端设计；  订正文档图表编号 |

# 1 FAST-ANT 2.0总体概述

## FAST-ANT 2.0基本介绍

FAST-ANT（Agile Network Tester）2.0是一款基于FPGA/CPU的轻量级网络测试分析仪，支持多类型流量的并发生成、测试流量统计回放以及针对吞吐率、丢包率和延迟等网络相关指标的测试。

FAST-ANT 2.0采用软硬件协同的设计思想，充分发挥软件在灵活扩展和硬件在性能精度方面的优势。软件部分支持图形化界面，方便用户自定义测试参数并且能够直观清晰地获取测试结果。同时，开放配置测试参数和获取测试结果的API，支持用户自定义设计开发测试功能。硬件部分支持高速率的流量发送，提供精确时间戳标记以及对硬件寄存器的直接读写操作，为测试提供了性能和精度的保障。软硬件协同的设计思想可以向用户屏蔽底层硬件的具体实现，取而代之的是提供读写硬件存储单元的API，将测试的灵活性和可定制性在软件端进行充分展示。这样，用户无需过多关注具体的实现细节，能够迅速展开测试工作，快速开发测试功能。

FAST-ANT 2.0将在OpenBox-S4上实现原型系统。后续将移植到NUDT-NSG团队自主开发的40G NP上，更进一步提高测试仪的性能。

## FAST-ANT 2.0功能分析

### 1.2.1 多类型流量并发功能

FAST-ANT 2.0支持多类型流量并发功能。由于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、多类型的要求。

### 1.2.2 流量统计回放功能

FAST-ANT 2.0支持测试流量统计回放功能。对于捕获到的测试流量，我们不仅支持统计收到的报文个数、总比特数以及记录报文接收时间戳，还支持存储一定数量的报文用于后续读取分析。流量统计回放功能能够满足用户对特定时间段或特定数量报文的处理分析，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或特定数量的所捕获到的报文并再次使用；或是在测试结束后，对某一时段产生的或特定数量的流量进行进一步分析来获取网络的某一时段状态。

该功能的思路是：FPGA捕获特定数量的报文并存储到相应的硬件存储空间内。用户根据自身需要给出所要读取的报文编号并从硬件存储空间内读出。读出后的报文可以供用户在软件端进行处理分析，生成可视化的数据结果。

### 1.2.3 基本网络测试功能

FAST-ANT 2.0支持对吞吐量、丢包率以及延迟等相关网络指标进行测试。

对于吞吐量的测试，FAST-ANT 2.0在硬件中准确计数收到的测试报文个数，修改相应寄存器值。在测试结束后由软件端读取并计算待测设备的吞吐量；

对于丢包率的测试，FAST-ANT 2.0既可以在特定的时间段内统计发送的报文个数，也可以按照指定数目发送报文，并在接收端统计收到的报文个数，从而计算出待测设备或网络的丢包率；

对于延迟的测试，FAST-ANT 2.0在发送报文的时候在报文体指定位置记录发送时间戳，并在收到报文后在报文体指定位置记录接收时间戳，同时存入硬件的存储空间内。接收与发送时间戳由特定的硬件功能模块统一提供。在测试结束后，软件端从硬件存储空间内读出报文，并解析报文体获取发送和接收时间戳，从而计算出待测设备的处理延迟或网络的转发延迟。

# 2 FAST-ANT 2.0概要设计

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。原型系统在OpenBox-S4上进行开发实现。

本章首先在2.1小节介绍FAST-ANT 2.0的软件概要设计，包括软件需求与功能分析以及软件架构设计等。接着在2.2小节介绍FAST-ANT 2.0的硬件概要设计，包括硬件架构设计以及各个功能模块的需求与功能分析等。最后在2.3小节介绍FAST-ANT 2.0所使用的数据分组结构。

## 2.1 FAST-ANT 2.0软件概要设计

FAST-ANT 2.0软件端负责接收用户配置的参数，执行用户选择的功能，并以可视化的界面展示测试结果。用户可以自定义配置报文内容、报文发送速率、报文发送个数以及报文发送时间等参数，支持并发流生成、流量回放以及相关基本网络性能测试。为了更加直观高效地展示测试结果，提升用户体验，向用户提供图形可视化界面，方便用户进行测试。同时，开放FAST-ANT 2.0 API供开发者基于FAST-ANT 2.0扩展新的测试功能。



图3 FAST-ANT 2.0软件架构图

FAST-ANT 2.0的软件结构设计如图3所示。整个软件端全部实现在远端Linux主机上，通过和OpenBox-S4内的FPGA建立通信，收发报文来实现交互。软件端的核心在于在FAST-ANT Lib的基础上开发了读写寄存器的编程接口（regWR和regRD）。FAST-ANT Lib主要由libnet和libpcap两个库组成，来实现在Linux主机上发送和接收报文。读写寄存器的操作则是通过构造发送以及接收解析相关的报文来实现的。基于读写寄存器编程接口，进一步扩展了FAST-ANT 2.0所支持的网络测试功能，并以函数的形式展现。最后，在FAST-ANT Func提供的函数接口基础上封装一层图形可视化界面供用户使用，从而屏蔽底层实现的细节。

在测试开始前，用户首先在提供的FAST-ANT GUI界面中配置相对应的测试参数并调用相应的功能函数。GUI中的输入值将传递到FAST-ANT Func层，并映射到对应的FAST-ANT数据结构中。接着，将调用相应的FAST-ANT API，以报文的形式将相关测试参数配置到FPGA中的硬件功能模块并运行相应的测试功能。

在测试结束后，FAST-ANT将同样通过调用相应的FAST-ANT API读取FPGA内相关寄存器的值和存储空间内的报文，并在FAST-ANT Func层经过处理后以图表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硬件概要设计

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

### 2.2.1 硬件总体架构

FAST-ANT 2.0的硬件架构设计是在传统FAST架构基础上进行扩展补充后得到的。架构设计将单流水线转化成多条并行的功能流水线，如图4所示。在图4所示的硬件架构设计中，功能划分更加明确，每一个功能模块专注于实现预先设计功能，而不用进行额外的处理操作，例如：转发与本模块无关的报文。并且，LCM模块作为本地控制模块，负责集中化管理所有硬件功能模块中寄存器的读写操作，以及同软件端的通信交互。相比于FAST-ANT 1.0中的环形控制通路设计，LCM模块的集中化管理减少了每个硬件功能模块对控制信息报文的额外处理，减少了硬件资源消耗，并突出了集中控制的设计思想。

PGM模块和GOE模块分别在FAST-ANT 1.0的基础上进行重构和复用；LCM模块是为了支持FAST-ANT 2.0的新功能而专门设计开发的；同时，FAST-ANT 2.0继承了FAST ANT 1.0中SCM模块的功能，并结合新的功能需求，设计并实现了SSM模块。DMUX模块和MUX模块则用于决定报文的具体传递方向，需要专门设计开发。

其中，核心功能模块包含以下几部分：



图4 FAST-ANT 2.0硬件架构设计图

***DMUX模块：***四选一模块，用于区分不同类型的报文（包括：来自软件端的读写寄存器报文、来自软件端用于构造并发流的报文以及网络端口进入的测试流量报文），并将报文传递至相应的硬件功能模块。

***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。

***SSM（Storage Statistics Module）模块：***存储统计模块，用于支持流量回放功能和统计计数功能。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，即：已存储多少报文。支持用户通过提供报文序号来从硬件中读取相应的报文。同时，SSM模块内维护接收报文个数计数器和接收总bit数计数器，软件端能够读取这些计数器，从而支持对丢包率和吞吐率的测试。

***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，对预配置的时刻调度表进行查表轮询，判断当前时刻满足发送条件的报文，从而从RAM中读出并发送。

***MUX模块：***二选一模块，用于决定是来自LCM模块的报文还是来自SSM模块的报文通过端口0上送至软件端处理。

### 2.2.2 UM接口信号定义

FPGA OS与UM的接口信号图如图5所示。



图5 UM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 |
| 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 |

### 2.2.3 DMUX模块

#### 2.2.3.1 DMUX模块需求与功能分析

DMUX模块负责对进入UM流水线内的报文进行分类识别，并分别转发到PGM模块、LCM模块以及SSM模块。

因此，DMUX模块需要支持以下功能：

1) 支持区分不同类型的报文；

2) 支持将区分后的报文转发到正确的硬件功能模块。

#### 2.2.3.2 DMUX模块概要设计

DMUX模块是UM流水线的第一个模块，如图6所示，接收来自FPGA OS的FAST-Pkt报文，并区分不同类型的报文后，分发至相应的硬件功能模块。



图6 DMUX在UM流水线内的位置

进入DMUX模块的报文分为以下三类：

1. 经过端口0，来自软件端的控制报文，包括对寄存器的读写操作等；
2. 经过端口0，来自软件端构造的生成并发流量的报文；
3. 经过其他端口，测试流量报文。

由于软件端实现在不同终端设备上，导致区分c类报文和a、b类报文的方法也不同。软件端的实现可分为实现在Xilinx Zynq 7000开发板的ARM芯片上和远端Linux主机上。具体实现如下：

1. 软件端实现在Xilinx Zynq 7000开发板的ARM芯片上

可以根据输入端口号（Metadata[125:120]）的不同进行区分数据报文（c类）和控制报文（a类和b类）；区分a类报文和b类报文，可以根据优先级（Metadata[111:109]）的不同进行区分，其中设置b类报文的Metadata[111:109] == 3’b111。

针对区分后的报文，DMUX模块将这些报文转发至对应的硬件功能模块。

DMUX模块的流程图如图7所示。



图7 DMUX模块流程图（Xilinx Zynq 7000开发板的ARM芯片上实现）

1. 软件端实现在远端Linux主机上

可以根据输入端口号（Metadata[125:120]）的不同进行区分数据报文（c类）和控制报文（a类和b类）；对于a类报文，我们指定为IP报文，其协议字段（8位宽）为c8。而其余报文则均认为是b类报文。

针对区分后的报文，DMUX模块将这些报文转发至对应的硬件功能模块。

DMUX模块的流程图如图8所示。



图8 DMUX模块流程图（远端Linux主机上实现）

### 2.2.4 LCM模块

#### 2.2.4.1 LCM模块需求与功能分析

为了突出集中化管理控制信息的思想，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读取数据信息后构造FAST-Pkt封装，通过端口0上送软件端。

因此，在LCM模块中需要支持以下功能：

1. LCM模块接收来自软件端的控制报文，对相关硬件功能模块内的寄存器执行读写操作；
2. LCM模块构造FAST-Pkt报文来封装读取的寄存器值，并通过端口0上送到软件端。

#### 2.2.4.2 LCM模块概要设计

LCM模块是FAST-ANT 2.0硬件架构中UM流水线部分的本地控制模块，如图9所示，负责集中化管理对不同硬件功能模块内寄存器的读写操作，并构造FAST-Pkt报文来封装读取的寄存器数据，通过端口0上送软件端处理分析。

LCM模块接收来自DMUX已经完成分类的FAST-Pkt报文，即来自软件端的控制报文（包括读写寄存器操作）。解析报文内容，区分控制报文是写寄存器报文，还是读寄存器报文。如果是写寄存器报文，则找到相应的寄存器，并写入新的寄存器值；如果是读寄存器报文，则获取该寄存器值后，将数据封装到新的FAST-Pkt报文内，并送往MUX模块，最终通过端口0上送至软件端进行处理和分析。



图9 LCM模块在UM流水线内的位置

LCM模块的流程示意图如图10所示。



图10 LCM模块流程示意图

在表2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2 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** |
| SSM | lcm2ssm\_addr | 写 | 20 | 20’b0 | 读取报文的首地址 |
| protocol\_type | 写 | 8 | 8’b0 | 特定报文类型 |
| ssm\_pkt\_num | 读 | 64 | 64’b0 | 记录总接收报文数量 |
| ssm\_bit | 读 | 64 | 64’b0 | 记录总接收bit数量 |
| PGM | sent\_start\_time\_n\_reg | 读写 | 64 | 64’b0 | 第n个报文的起始发送时刻 |
| sent\_rate\_n\_reg | 读写 | 64 | 64’b0 | 第n个报文的发送速率，即发送间隔的拍数 |
| sent\_bit\_cnt | 读 | 64 | 64’b0 | 总发送bit数 |
| sent\_pkt\_n\_cnt | 读 | 64 | 64’b0 | 第n个报文的总发送报文个数 |
| sent\_time\_cnt | 读 | 64 | 64’b0 | 当前测试时间 |
| sent\_time\_reg | 读写 | 64 | 64’b0 | 总测试时间 |
| sent\_num\_cnt | 读 | 64 | 64’b0 | 已发送报文个数 |
| sent\_num\_reg | 读写 | 64 | 64’b0 | 总发送报文个数 |
| GOE | goe\_port\_n\_pkt\_cnt | 读 | 64 | 64’b0 | 端口n输出的报文个数 |
| goe\_port\_n\_bit\_cnt | 读 | 64 | 64’b0 | 端口n输出的bit数 |

### 2.2.5 SSM模块

#### 2.2.5.1 SSM模块需求与功能分析

SSM(Store Statistics Mudule)模块为存储与统计模块，需统计接收报文的个数，接收报文的总bit数，将lcm记录的时间戳数据存入报文体中指定位置，再将报文存入RAM中，若用户有读请求，则根据读取报文的地址将报文从RAM中读取并发送至软件端。

为了实现统计丢包率和吞吐率的功能，SSM模块需实现：

1. 对SSM模块接收的测试流量的报文个数（ssm\_pkt\_num\_cnt）、接收报文的bit数（ssm\_bit\_num\_cnt）进行统计；
2. 将统计的接收报文个数（ssm\_pkt\_num\_cnt）、接收报文的bit数（ssm\_bit\_num\_cnt）发送至LCM模块，由LCM模块送至软件端。

为了实现将报文存入RAM,同时记录时间戳数据，SSM模块中需实现：

1. 根据RAM中已经存放报文个数（ram\_pkt\_num\_cnt）计算当前报文在RAM中存放的起始地址；
2. 当指定要存放时间戳的数据进入SSM模块中时，将LCM模块的时间戳数据赋给指定的位置，再将报文存入RAM中相应的的地址空间，每存放一拍报文数据，地址+1指向下一地址，直到整个报文存放完毕，每存放一完整报文，已经存放报文个数ram\_pkt\_num\_cnt <= ram\_pkt\_num\_cnt+1;

为了实现读取用户指定的报文数据，需在SSM模块中实现：

1. 当有读取报文的需求时，软件端根据用户想读取的报文号，解析出读取报文的起始地址，由LCM模块下发到SSM模块，SSM模块将地址发送至RAM，读信号置为有效，RAM读出的报文直接送往软件端而不再经过SSM模块。

#### 2.2.5.2 SSM模块概要设计

SSM模块位于DMUX模块和MUX模块之间，并同LCM模块通信，如图11中黄色模块所示，负责接收来自DMUX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。同时，同LCM模块进行通信，能够支持对硬件寄存器的读写操作。

负责接收来自LCM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）。同时，将接收到的测试报文存储至RAM中。

同时，SSM模块支持同LCM模块进行通信，允许LCM模块读取SSM中寄存器的值(总接收报文数量ssm\_pkt\_num\_cnt、总接收bit数量ssm\_bit\_num\_cnt)。并支持将RAM中保存的报文上送软件端。



图11 SSM模块在UM流水线内的位置

为了更好地支持SSM模块的流量回放功能，在SSM定义了表3所示的寄存器，用于存储统计数据和软件端的配置信息。

表3 UDC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** |
| lcm2ssm\_addr | 写 | 20 | 20’b0 | 读取报文的首地址 |
| protocol\_type | 写 | 8 | 8’b0 | 特定报文类型 |
| ssm\_pkt\_num | 读 | 64 | 64’b0 | 记录总接收报文数量 |
| ssm\_bit | 读 | 64 | 64’b0 | 记录总接收bit数量 |

### 2.2.6 PGM模块

#### 2.2.6.1 PGM模块需求与功能分析

PGM（Packet Generation Module）模块在FAST-ANT 1.0的基础上，由支持发送单一流转变为支持同时发送多条流。同时，配置发送参数的方式也由FAST-ANT 1.0中通过环形控制通路配置转变为FAST-ANT 2.0中通过LCM模块统一集中配置。

因此，在PGM模块中需要支持以下功能：

1. 支持接收并存储软件端为生成并发流量所构造的不同类型的报文；
2. 支持不同的流量发送模式（按时间发送或按报文数发送）；
3. 支持软件端定义配置不同流量的发送参数，包括：报文的发送时刻、报文的发送速率（发送间隔）；
4. 支持衍变构造新的子流；
5. 支持存储时刻调度表，并按照时刻调度表生成并发流量；
6. 支持同LCM模块进行通信，从而实现不同参数寄存器的读写操作。

#### 2.2.6.2 PGM模块概要设计

PGM模块位于DMUX模块之后，如图12所示，负责接收来自DMUX模块转发的软件端用于构造并发流量的FAST-Pkt，并存入PGM模块内的PGM\_RAM中。同时，接收来自LCM模块的配置信息，读写相应寄存器。为了支持PGM生成多条并发流量，在PGM\_RD内部保存软件端配置的时刻调度表，用于决定当前时刻应该发送哪一类型的报文，从而满足不同类型的流量以不同速率产生，并能够实现并发。



图12 PGM模块在UM流水线内的位置

PGM模块的流程图如图13所示。



图13 PGM模块流程示意图

PGM模块由三个子模块PGM\_WR、PGM\_RD以及PGM\_RAM共同组成，详细设计部分见3.5.3小节。在这里给出PGM模块内所定义的寄存器信息，如表4所示。

表4 PGM模块内寄存器定义表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **子模块** | **寄存器名称** | **读写** | **寄存器大小** | **寄存器初始值** | **寄存器描述** |
| PGM\_WR | sent\_start\_time\_n\_reg | 读写 | 64 | 64’b0 | 第n个报文的起始发送时刻 |
| sent\_rate\_n\_reg | 读写 | 64 | 64’b0 | 第n个报文的发送速率，即发送间隔的拍数 |
| PGM\_RD | sent\_bit\_cnt | 读 | 64 | 64’b0 | 总发送bit数 |
| sent\_pkt\_n\_cnt | 读 | 64 | 64’b0 | 第n个报文的总发送报文个数 |
| sent\_time\_cnt | 读 | 64 | 64’b0 | 当前测试时间 |
| sent\_time\_reg | 读写 | 64 | 64’b0 | 总测试时间 |
| sent\_num\_cnt | 读 | 64 | 64’b0 | 已发送报文个数 |
| sent\_num\_reg | 读写 | 64 | 64’b0 | 总发送报文个数 |

### 2.2.7 MUX模块

#### 2.2.7.1 MUX模块需求与功能分析

MUX模块负责接收来LCM模块和SSM模块的FAST-Pkt报文，并决定哪个模块的报文通过端口0上送软件端。

因此，MUX模块需要支持以下功能：

1. 支持接收来自LCM模块和SSM模块的FAST-Pkt报文；
2. 支持根据选择信号进行二选一输出至端口0。

#### 2.2.7.2 MUX模块概要设计

MUX模块位于UM流水线的输出部分，如图14所示。接收来自SSM模块和LCM模块的FAST-Pkt报文，并根据LCM模块传递的软件端的选择信号，来确定是将来自SSM模块的FAST-Pkt报文，还是来自LCM模块的FAST-Pkt报文通过端口0上送到软件端进行处理。



图14 MUX模块在UM流水线内的位置

MUX模块的流程图如图15所示。



图15 MUX模块流程图

## 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16所示。

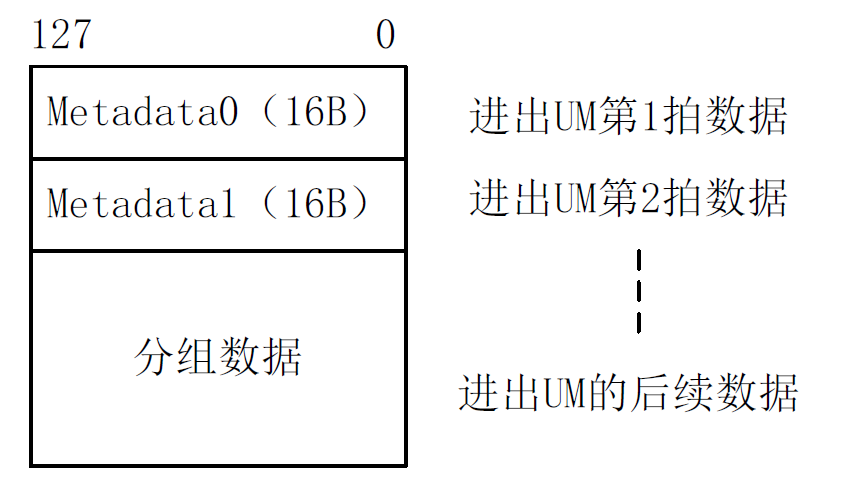


图16 分组数据结构示意图

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5 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 |

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17所示。



图17 接口分组结构示意图

# 3 FAST-ANT 2.0详细设计

## 3.1 FAST-ANT 2.0软件详细设计

FAST-ANT 2.0的软件部分实现在远端Linux主机上，通过和OpenBox-S4内的FPGA建立通信，以收发报文的形式进行操作交互。基于libnet和libpcap这两个Linux操作系统上收发报文的库，FAST-ANT 2.0提供了对寄存器的读写操作函数，并在读写寄存器函数的基础上进一步封装成功能函数，支持FAST-ANT 2.0所计划扩展的功能。最后，向用户提供图形可视化界面，方便用户操作使用。

下面将详细介绍软件端各功能层及相关的关键技术。

### 3.1.1 FAST-ANT Lib

FAST-ANT Lib的核心库是Linux操作系统所提供的收发报文的libnet库和libpcap库。在这两个库的基础上，通过构造发送以及接收解析报文，设计并实现了对FPGA中寄存器的读写操作函数。

#### 3.1.1.1 libnet

libnet是一个小型的接口函数库，主要用C语言编写完成，提供了低层网络数据包的构造、处理以及发送功能。libnet的开发目的是建立一个简单统一的网络编程接口以屏蔽不同操作系统底层网络编程的差别，使得程序员将精力集中在解决关键问题上。

libnet库提供的接口函数包含了15种数据包生成器和两种数据包发送器（IP层和数据链路层）。

在FAST-ANT 2.0中，通过使用libnet库来构造并发送报文给FPGA，用于实现对FPGA中寄存器的操作。其中，所用到的libnet库中的关键函数如下：

表6 libnet\_init函数

|  |
| --- |
| **函数名：**libnet\_t \*libnet\_init(int injection\_type, char \*device, char \*err\_buf); |
| **功能：**数据包内存初始化及建立 |
| **参数：** |
| injection\_type：构造的类型 |
| *LIBNET\_LINK* 链路层 |
| *LIBNET\_RAW4* 网络接口层（网络层） |
| *LIBNET\_LINK\_ADV* 链路层高级版本 |
| *LIBNET\_RAW4\_ADV* 网络层高级版本 |
| device：网络接口，如“eth0”，或IP地址，亦可为NULL（自动查询搜索） |
| err\_buf：存放出错的信息 |
| **返回值：** |
| 成功：一个libnet \*类型的指针，后面的操作均需要使用这个指针 |
| 失败：NULL |

表7 libnet\_build\_ipv4函数

|  |
| --- |
| **函数名：**libnet\_ptag\_t libnet\_build\_ipv4(u\_int16\_t ip\_len, u\_int8\_t tos,u\_int16\_t id, u\_int16\_t flag,u\_int8\_t ttl, u\_int8\_t prot,u\_int16 sum, u\_int32\_t src,u\_int32\_t dst, u\_int8\_t \*payload,u\_int32\_t payload\_s,libnet\_t \*l,libnet\_ptag\_t ptag ); |
| **功能：**构造一个IPv4的数据包 |
| **参数：** |
| ip\_len：IP包总长度 |
| tos：服务类型 |
| id：IP标识 |
| flag：片偏移 |
| ttl：生存时间 |
| prot：上层协议 |
| sum：校验和。设为0则libnet自动填充 |
| src：源IP地址 |
| dst：目的IP地址 |
| payload：负载内容，可设置为NULL（通常写为NULL，由上层包填充具体内容。当然也可以自定义填充内容。本项目中便使用自填充的方式） |
| payload\_s：负载长度，或为0（通常为0） |
| l：libnet句柄，libnet\_init()返回的libnet\*指针 |
| ptag：协议标记，第一次构造新的发送包时，这里写0。同一个应用程序，下一次再组包时，这个位置的值填写此函数的返回值 |
| **返回值：** |
| 成功：协议标记 |
| 失败：-1 |

表8 libnet\_build\_ethernet函数

|  |
| --- |
| **函数名：**libnet\_ptag\_t libnet\_build\_ethernet(u\_int8\_t\*dst, u\_int8\_t \*src,u\_int16\_ttype, u\_int8\_t\*payload,u\_int32\_tpayload\_s, libnet\_t\*l,libnet\_ptag\_t ptag ); |
| **功能：**构造一个以太网数据包 |
| **参数：** |
| dst：目的MAC地址 |
| src：源MAC地址 |
| type：上层协议类型 |
| payload：负载内容，即附带的数据，可设置为NULL（通常写NULL，当然也可以自填充相关数据内容。本项目中便自定义payload内容） |
| l：libnet句柄，libnet\_init()返回的libnet\*指针 |
| ptag：协议标记，第一次构造新的发送包时，这里写0。同一个应用程序，下一次再组包时，这个位置的值写此函数的返回值 |
| **返回值：** |
| 成功：协议标记 |
| 失败：-1 |

表9 libnet\_write函数

|  |
| --- |
| **函数名：**int libnet\_write(libnet\_t \* l); |
| **功能：**发送数据包 |
| **参数：** |
| l：libnet句柄，libnet\_init()返回的libnet \*指针 |
| **返回值：** |
| 成功：发送数据包的长度 |
| 失败：返回-1 |

#### 3.1.1.2 libpcap

libpcap（Packet Capture Library）即数据包捕获函数库，是Unix/Linux平台下的网络数据包捕获函数库。它是一个独立于系统的用户层包捕获的API接口，为底层网络监测提供了一个可移植的框架。著名的软件TCPDUMP就是在libpcap的基础上开发而成的。

libpcap可以实现以下功能：

* 数据包捕获：捕获流经网卡的原始数据包；
* 流量采集与统计：网络采集的流量信息；
* 规则过滤：提供自带规则过滤功能，按需要选择过滤规则。

在FAST-ANT 2.0中，通过使用libpcap库来捕获来自FPGA的报文，包括了读寄存器报文和流量重放报文，并解析报文内容，提取相应的参数值和统计数据。其中，所用到的关键函数如下：

表10 pcap\_loop函数

|  |
| --- |
| **函数名：**int pcap\_loop(pcap\_t \*p,int cnt,pcap\_handler callback,u\_char \*user); |
| **功能：**循环捕获网络数据包，直到遇到错误或者满足退出条件，每次捕获一个数据包就会调用callback指定的回调函数。所以，可以在回调函数中进行数据包的处理操作。 |
| **参数：** |
| p：pcap\_open\_live()返回的pcap\_t类型的指针 |
| cnt：指定捕获数据包的个数，一旦抓到cnt个数据包，pcap\_loop立即返回。如果是-1，则会一直捕获直到出现错误 |
| callback：回调函数，名字任意，根据需要自行取名 |
| user：向回调函数中传递参数，一般为NULL |

### 3.1.2 FAST-ANT API

在FAST-ANT Lib的基础上，开发了软件端读写寄存器的功能函数regWR和regRD以及捕获FPGA上送报文的报文捕获功能函数capPkt。通过使用这两个功能函数所提供的编程接口，可以直接对硬件中的寄存器进行读写操作，从而实现对测试参数的配置以及读取硬件计数器统计结果。

下面，将详细介绍regWR、regRD和capPkt函数。

#### 3.1.2.1 regWR

写寄存器功能函数是在libnet库基础上，通过构造相应的操作命令报文，将写寄存器编号和值下发到FPGA中相应的硬件功能模块中，最终由硬件功能模块解析后修改相应寄存器值。函数具体内容如下：

表11 regWR函数

|  |
| --- |
| **函数名：**void regWR(int regNum, unsigned char \*regWRVal); |
| **功能：**写硬件寄存器值。 |
| **参数：** |
| regNum：寄存器编号值 |
| regWRVal：写入寄存器的值 |

写寄存器函数是基于libnet库基础，将写入的写寄存器编号和值放入构造报文的数据字段的指定位，并通过网卡发送到硬件FPGA内。

#### 3.1.2.2 regRD

读寄存器功能函数是在libnet库基础上，通过构造相应的操作命令报文，将读寄存器编号下发到FPGA中相应的硬件功能模块中，最终由硬件功能模块解析出读寄存器编号，并将相应寄存器值放入构造报文上送回软件端。函数具体内容如下：

表12 regRD函数

|  |
| --- |
| **函数名：**void regRD(int regNum); |
| **功能：**读硬件寄存器值。 |
| **参数：** |
| regNum：寄存器编号值 |

读寄存器函数是基于libnet库基础，将读寄存器编号放入构造的报文并通过网卡下发到FPGA内。

#### 3.1.2.3 capPkt

报文捕获函数是在libpcap库基础上，接收硬件上送的报文并解析，提取读寄存器值或流量重放功能所保存的报文。本函数是libpcap库中pcap\_loop函数的callback函数，函数具体内容如下：

表13 capPkt函数

|  |
| --- |
| **函数名：**void callback(u\_char \*userarg,const struct pcap\_pkthdr \*pkthdr,const u\_char \*packet); |
| **功能：**捕获报文并解析。 |
| **参数：** |
| userarg：pcap\_loop()的最后一个参数，当收到足够数量的包后pcap\_loop会调用callback回调函数，同时将pcap\_loop()的user参数传递给它 |
| pkthdr：是收到数据包的pcap\_pkthdr类型的指针，和pcap\_next()第二个参数是一样的 |
| packet：收到的数据包数据 |
| **返回值：** |
| 成功：返回0 |
| 失败：返回负数 |

### 3.1.3 FAST-ANT Func

在FAST-ANT API的基础上，我们进一步封装提供了三个功能函数用来实现FAST-ANT 2.0所支持的测试功能。具体内容如下：

#### 3.1.3.1 GenMF

生成并发流的函数通过调用FAST-ANT API中所提供的regWR函数来对与生成并发流量相关的硬件寄存器进行配置。

报文的内容配置在TXT文件，并在构造报文时读取。相关测试参数（如：发送速率、发送时长、发送报文个数等）均通过regWR函数写入到硬件FPGA内的寄存器中。

在配置结束后，通过regWR函数触发硬件开始发包，从而实现并发流量的生成操作。

#### 3.1.3.2 TraRP

流量回放功能的函数通过调用FAST-ANT API中所提供的regRD函数来写入需要读取的报文编号，硬件FPGA收到包含报文编号的报文后提取编号并转换成读RAM地址，最终从RAM中读取相应的报文并上送回软件端。

通过调用capPkt函数，捕获所接收到的报文，并解析报文内容，提取相关数据信息。

#### 3.1.3.3 CollDI

收集数据信息的函数通过综合调用FAST-ANT API中的写寄存器函数regWR、读寄存器函数regRD以及捕获报文函数capPkt，实现测试流量的发送、接收以及数据信息的读取等，从而实现对延迟、吞吐率、丢包率等基本网络指标的测试。

### 3.1.4 FAST-ANT GUI

FAST-ANT 2.0的软件界面设计参考小兵发包仪的界面设计，直观高效地向用户展示数据测试功能和测试结果。

FAST-ANT 2.0使用QT进行软件可视化界面的开发。

## 3.2 FAST-ANT 2.0硬件详细设计

### 3.2.1 DMUX模块

#### 3.2.1.1 模块接口设计

DMUX模块的接口信号定义图如图18所示。



图18 DMUX模块接口信号定义图

DMUX模块的接口信号定义表如表14所示。

表14 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 |
| pktin\_data | 134 | Input | 输入DMUX模块的数据 |
| pktin\_data\_wr | 1 | Input | 输入DMUX模块的数据的写信号 |
| pktin\_data\_valid | 1 | Input | 输入DMUX模块的报文的有效信号 |
| pktin\_data\_valid\_wr | 1 | Input | 输入DMUX模块的报文的有效信号的写信号 |
| pktin\_ready | 1 | Output | DMUX模块准备接收数据信号 |
| dmux2pgm\_data | 134 | Output | DMUX模块转发给PGM模块的数据 |
| dmux2pgm\_data\_wr | 1 | Output | DMUX模块转发给PGM模块的数据的写信号 |
| dmux2pgm\_data\_valid | 1 | Output | DMUX模块转发给PGM模块的报文的有效信号 |
| dmux2pgm\_data\_valid\_wr | 1 | Output | DMUX模块转发给PGM模块的报文的有效信号的写信号 |
| pgm2dmux\_data\_ready | 1 | Input | PGM模块准备接收数据信号 |
| dmux2lcm\_data | 134 | Output | DMUX模块转发给LCM模块的数据 |
| dmux2lcm\_data\_wr | 1 | Output | DMUX模块转发给LCM模块的数据的写信号 |
| dmux2lcm\_data\_valid | 1 | Output | DMUX模块转发给LCM模块的报文的有效信号 |
| dmux2lcm\_data\_valid\_wr | 1 | Output | DMUX模块转发给LCM模块的报文的有效信号的写信号 |
| lcm2dmux\_data\_ready | 1 | Input | LCM模块准备接收数据信号 |
| dmux2ssm\_data | 134 | Output | DMUX模块转发给SSM模块的数据 |
| dmux2ssm\_data\_wr | 1 | Output | DMUX模块转发给SSM模块的数据的写信号 |
| dmux2ssm\_data\_valid | 1 | Output | DMUX模块转发给SSM模块的报文的有效信号 |
| dmux2ssm\_data\_valid\_wr | 1 | Output | DMUX模块转发给SSM模块的报文的有效信号的写信号 |
| ssm2dmux\_data\_ready | 1 | Input | SSM模块准备接收数据信号 |

#### 3.2.3.2 模块实现

DMUX模块负责区分收到的FAST-Pkt，并转发至相应的硬件功能模块。其中，分类依据请参考3.2.2小节，此处不再赘述。

因此，只需要在DMUX模块中实现一个简单的状态机便可以完成区分转发功能。根据软件端在不同终端上的实现，我们给出了相对应的状态机图。

1. 软件端实现在Xilinx Zynq 7000开发板的ARM芯片上

DMUX模块的状态机图如图19所示。



图19 DMUX模块状态机图

**IDLE\_S：**空闲状态。对收到的FAST-Pkt报文的第一拍（即Metadata）进行解析判断。如果pktin\_data[133:132] == 2'b01 && pktin\_data[125:120] != 6'b0，则跳转到SEND\_SSM\_S；如果pktin\_data[133:132] == 2'b01 && pktin\_data[125:120] == 6'b0 && pktin\_data[111:109] == 3'b111，则跳转到SEND\_PGM\_S；如果pktin\_data[133:132] == 2'b01 && pktin\_data[125:120] == 6'b0 && pktin\_data[111:109] != 3'b111，则跳转到SEND\_LCM\_S。

**SEND\_SSM\_S：**转发SSM模块状态。将收到的报文转发至SSM模块。如果pktin\_data[133:132] == 2'b10，则跳转到IDLE\_S；如果pktin\_data[133:132] == 2'b11，则跳转到SEND\_SSM\_S。

**SEND\_PGM\_S：**转发PGM模块状态。转发PGM模块状态。将收到的报文转发至PGM模块。如果pktin\_data[133:132] == 2'b10，则跳转到IDLE\_S；如果pktin\_data[133:132] == 2'b11，则跳转到SEND\_PGM\_S。

**SEND\_LCM\_S：**转发LCM模块状态。转发LCM模块状态。将收到的报文转发至LCM模块。如果pktin\_data[133:132] == 2'b10，则跳转到IDLE\_S；如果pktin\_data[133:132] == 2'b11，则跳转到SEND\_LCM\_S。

1. 软件端实现在远端Linux主机上

DMUX模块的状态机图如图20所示。



图20 DMUX模块状态机图

**IDLE\_S**：空闲状态。对收到的FAST-Pkt报文的第一拍（即Metadata）进行解析判断。如果pktin\_data[133:132] == 2'b01 && pktin\_data[125:120] != 6'b0，则跳转至SEND\_SSM\_S；否则跳转至STORE\_S。

**SEND\_SSM\_S：**转发SSM模块状态。将收到的报文转发至SSM模块。如果pktin\_data[133:132] == 2'b10，则跳转至IDLE\_S；如果pktin\_data[133:132] == 2'b11，则跳转至SEND\_SSM\_S。

**STORE\_S：**缓存状态。由于需要判断是送往PGM模块的报文还是送往SSM模块的报文，先将收到的报文缓存到FIFO中，并根据是否为IP报文且协议号为c8或c9进行区分。若为IP报文且协议号为c8，则代表是送LCM模块的写报文，将select\_flag和w\_r\_flag信号置为无效；若为IP报文且协议号为c9，则代表是送LCM模块的读报文，将select\_flag信号置为无效，w\_r\_flag信号置为有效。否则，将select\_flag信号置为有效。

如果pktin\_data[133:132] == 2'b11，则跳转至STORE\_S；如果pktin\_data[133:132] == 2'b10且select\_flag有效，则跳转至SEND\_PGM\_S；如果pktin\_data[133:132] == 2'b10且select\_flag无效，则跳转至SEND\_LCM\_S。

**SEND\_PGM\_S：**转发PGM模块状态。从FIFO中读出报文并转发至PGM模块。如果fifo\_data\_out[133:132] != 2’b10，则跳转至SEND\_PGM\_S；否则，跳转至IDLE\_S。

**SEND\_LCM\_S：**转发LCM模块状态。从FIFO中读出报文并转发至LCM模块。如果fifo\_data\_out[133:132] != 2’b10，则跳转至SEND\_LCM\_S；否则，跳转至IDLE\_S。

### 3.2.2 LCM模块

#### 3.2.2.1 LCM顶层模块设计

##### 3.2.2.1.1 模块接口设计

LCM模块的接口信号定义图如图21所示。



图21 LCM模块接口信号定义图

LCM模块的接口信号定义表如表15所示。

表15 LCM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模块准备接收数据的信号 |
| out\_lcm\_data[133:0] | 134 | Output | LCM模块输出的数据 |
| out\_lcm\_data\_wr | 1 | Output | LCM模块输出的数据写信号 |
| out\_lcm\_data\_valid | 1 | Output | LCM模块输出的报文有效信号 |
| out\_lcm\_data\_valid\_wr | 1 | Output | LCM模块输出的报文有效信号的写信号 |
| out\_lcm\_data\_ready | 1 | Input | MUX模块准备接收数据的信号 |
| lcm2ssm\_reset | 1 | Output | SSM模块重置 |
| lcm2ssm\_rd | 1 | Output | 用户读请求信号 |
| lcm2ssm\_addr | 11 | Output | 用户读报文首地址 |
| protocol\_type | 8 | Output | 筛选报文协议类型 |
| ssm\_bit | 64 | Input | SSM接收bit数 |
| ssm\_pkt\_num | 64 | Input | SSM接收报文个数 |
| pgm\_config\_reset | 1 | Output | 配置参数重置信号 |
| sent\_start\_time\_n\_reg | 64 | Both | 第n个报文的起始发送时刻 |
| sent\_rate\_n\_reg | 64 | Both | 第n个报文的发送速率，即发送间隔的拍数 |
| table\_entry\_wr | 1 | Output | 时刻调度表表项有效信号 |
| sent\_start | 1 | Output | 测试开始信号 |
| sent\_ finish | 1 | Input | 测试结束信号 |
| sent\_model | 1 | Output | 测试模式选择信号 |
| sent\_bit\_cnt | 64 | Input | 总发送bit数 |
| sent\_pkt\_0\_cnt | 64 | Input | 1号流的总发送报文个数 |
| sent\_pkt\_1\_cnt | 64 | Input | 2号流的总发送报文个数 |
| sent\_pkt\_2\_cnt | 64 | Input | 3号流的总发送报文个数 |
| sent\_pkt\_3\_cnt | 64 | Input | 4号流的总发送报文个数 |
| sent\_time\_cnt | 64 | Input | 当前测试时间 |
| sent\_time\_reg | 64 | Both | 总测试时间 |
| sent\_num\_cnt | 64 | Input | 已发送报文个数 |
| sent\_num\_reg | 64 | Both | 总发送报文个数 |
| sent\_ready | 1 | Input | 测试准备就绪信号 |
| mux2port\_0\_rd | 1 | Output | MUX模块的转发选择信号 |
| lcm\_timestamp | 64 | Output | LCM模块提供的公共时钟计数器 |

##### 3.2.2.1.2 模块实现

LCM模块内部架构设计图如图22所示（注：reg\_Rd和reg\_Wr为读写寄存器操作，相关寄存器信息请参考3.3.3.1.1小节中的表）。



图22 LCM模块内部架构设计图

LCM模块由Parser子模块、Reg\_WR子模块以及Reg\_RD子模块共同组成。

其中，Parser模块负责解析来自DMUX模块的报文内容，从而确定是写寄存器报文，还是读寄存器报文。如果是写寄存器报文，则将写寄存器编号（wr\_reg\_n）和写寄存器值（wr\_reg\_n）传递给Reg\_WR模块；如果是读寄存器报文，则将读寄存器编号（rd\_reg\_n）传递给Reg\_RD模块。

Reg\_WR模块负责根据Parser模块传递来的写寄存器编号（wr\_reg\_n）和写寄存器值（wr\_reg\_n）对相应的寄存器进行修改。

Reg\_RD模块负责根据Parser模块传递来的读寄存器编号（rd\_reg\_n）来读取相应寄存器的值，并放入到构造的FAST-Pkt报文的指定位置后，发送至MUX模块

#### 3.2.2.2 Parser模块

##### 3.2.2.2.1 模块接口设计

Parser模块的接口信号定义图如图23所示。



图23 Parser模块接口信号定义图

Parser模块的接口信号定义表如表 所示。

表16 Parser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 |
| wr\_reg\_n | 8 | Output | 写寄存器编号 |
| wr\_reg\_n\_value | 64 | Output | 写寄存器值 |
| rd\_reg\_n | 8 | Output | 读寄存器编号 |

##### 3.2.2.2.2 模块实现

Parser模块区分接收的控制报文是写寄存器报文还是读寄存器报文。若为写寄存器报文，则解析出写寄存器编号和相应的写寄存器值；若为读寄存器报文，则解析出读寄存器编号。Parser模块通过实现图24所示的状态机来支持上述功能的实现。



图24 Parser模块状态机图

**IDLE\_S：**空闲状态。根据FAST-Pkt报文的第一拍（in\_lcm\_data[133:132] == 2’b01）的第48位（in\_lcm\_data[48]，即Metadata[48]）来判断控制报文类型。若in\_lcm\_data[48] == 1’b1，则控制报文为读寄存器报文，跳转至RD\_S；若in\_lcm\_data[48] == 1’b0，则控制报文为写寄存器报文，跳转至WR\_S。

**RD\_S：**读状态。读寄存器编号存放在FAST-Pkt报文的第五拍的第111位到第104位（in\_lcm\_data[111:104]）。因此，在RD\_S内维护一个报文拍数计数器（in\_lcm\_pkt\_cnt）。若in\_lcm\_pkt\_cnt != 11’d5，则跳转至RD\_S本身；若in\_lcm\_pkt\_cnt == 11’d5，则rd\_reg\_n = in\_lcm\_data[111:104]，并跳转至IDLE\_S。

**WR\_S：**写状态。写寄存器编号和写寄存器值分别放在FAST-Pkt报文的第五拍的第111位到第104位（in\_lcm\_data[111:104]）和第103位到第40位（in\_lcm\_data[103:40]）。同RD\_S，WR\_S内维护一个报文拍数计数器（in\_lcm\_pkt\_cnt）。若in\_lcm\_pkt\_cnt != 11’d5，则跳转至WR\_S本身；若in\_lcm\_pkt\_cnt == 11’d5，则wr\_reg\_n = in\_lcm\_data[111:104]并且wr\_reg\_n\_value = in\_lcm\_data[103:40]，跳转至IDLE\_S。

#### 3.2.2.3 Reg\_WR模块

##### 3.2.2.3.1 模块接口设计

Reg\_WR模块的接口信号定义图如图27所示。



图27 Reg\_W\_R模块接口信号定义图

Reg\_WR模块的接口信号定义表如表18所示。

表18 Reg\_WR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 |
| wr\_reg\_n | 8 | Input | 写寄存器编号 |
| wr\_reg\_n\_value | 64 | Input | 写寄存器值 |
| reg\_wr | 统一指代对寄存器进行写的信号  具体内容可以参考3.2.3.1.1小节中的表 | | |

##### 3.2.2.3.2 模块实现

在Reg\_WR模块中，需要根据Parser模块传递的写寄存器编号（wr\_reg\_n）和写寄存器值（wr\_reg\_n\_value）来修改对应寄存器的值。

因此，只需要使用case语句对寄存器编号进行轮询查找，根据寄存器编号来修改相应寄存器的值即可。

#### 3.2.2.4 Reg\_RD模块

##### 3.2.2.4.1 模块接口设计

Reg\_RD模块的接口信号定义图如图29所示。



图29 Reg\_RD模块接口信号定义图

Reg\_RD模块的接口信号定义表如表19所示。

表19 Reg\_RD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 |
| rd\_reg\_n | 8 | Output | 读寄存器编号 |
| out\_lcm\_data[133:0] | 134 | Output | LCM模块输出的数据 |
| out\_lcm\_data\_wr | 1 | Output | LCM模块输出的数据写信号 |
| out\_lcm\_data\_valid | 1 | Output | LCM模块输出的报文有效信号 |
| out\_lcm\_data\_valid\_wr | 1 | Output | LCM模块输出的报文有效信号的写信号 |
| out\_lcm\_data\_ready | 1 | Input | MUX模块准备接收数据的信号 |
| reg\_rd | 统一指代对寄存器进行读的信号  具体内容可以参考3.2.3.1.1小节中的表 | | |

##### 3.2.2.4.2 模块实现

在Reg\_RD模块中，需要根据Parser模块传递的读寄存器编号（rd\_reg\_n）来查询相应寄存器的值，并通过构造FAST-Pkt报文，将读寄存器编号和读取的寄存器值封装在报文的第六拍的第127位到第120位（out\_lcm\_data[127:120]）和第120位到第56位（out\_lcm\_data[120:56]），并将报文发送至MUX模块，进一步通过端口0上送至软件端进行处理分析。因此，Reg\_RD模块可以通过图30所示的状态机来实现上述功能。



图30 Reg\_RD模块状态机图

**IDLE\_S：**空闲状态。当收到来自Parser模块传递的读寄存器编号（rd\_reg\_n）后，匹配到对应的读寄存器编号（rd\_reg\_n），则跳转至RD\_S。否则，跳转到IDLE\_S。

**SEND\_S：**构造发送状态。将读寄存器编号和读寄存器值封装在构造的FAST-Pkt报文中，并传递至MUX模块。构造的FAST-Pkt报文为6拍（最小的64字节IP报文）。因此，当报文拍数计数器计数到5时（upload\_pkt\_cnt == 11’d5），发送报文尾并跳转到IDLE\_S。否则依次发送报文头和报文体。并且，在报文拍数计数器计数到5时（upload\_pkt\_cnt == 11’d5），将读寄存器编号和读寄存器值分别存入第六拍的第127位到第120位（out\_lcm\_data[127:120]）和第120位到第56位（out\_lcm\_data[120:56]），并发送至MUX模块。

### 3.2.3 SSM模块

#### 3.2.3.1 SSM顶层模块

##### 3.2.3.1.1 模块接口设计

SSM模块的接口信号定义图如图31所示。



图31 SSM模块的接口信号定义图

SSM模块的接口信号定义表如表20所示。

表20 SSM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 |
| lcm2ssm\_reset | 1 | Input | SSM模块重置 |
| lcm2ssm\_time | 64 | Input | lcm记录的时间戳 |
| lcm2ssm\_rd | 1 | Input | 用户读请求信号 |
| lcm2ssm\_addr | 11 | Input | 用户读报文首地址 |
| ssm\_bit | 64 | Output | SSM接收bit数 |
| ssm\_pkt\_num | 64 | Output | SSM接收报文个数 |
| dmux2ssm\_data | 134 | Input | 接收测试报文数据 |
| dmux2ssm\_data\_wr | 1 | Input | 测试报文输入有效 |
| dmux2ssm\_valid | 1 | Input | 报文最后一拍输入 |
| dmux2ssm\_valid\_wr | 1 | Input | 报文最后一拍输入有效 |
| ssm2mux\_data | 134 | Output | RAM中读取后输出报文 |
| ssm2mux\_data\_wr | 1 | Output | 读取后输出报文有效 |
| ssm2mux\_valid | 1 | Output | 输出报文最后一拍 |
| ssm2mux\_valid\_wr | 1 | Output | 输出报文最后一拍有效 |
| ssm2ram\_wr\_addr | 11 | Output | RAM存放报文的地址 |
| ssm2ram\_data | 134 | Output | 存放的报文数据 |
| ssm2ram\_data\_wr | 1 | Output | 往RAM存放报文有效 |
| ram2ssm\_data | 134 | Input | 从RAM读取的报文 |
| ssm2ram\_addr | 11 | Output | 读取报文的地址 |
| ssm2ram\_rd | 1 | Output | 读请求 |

##### 3.2.3.1.2 模块实现

SSM模块由SSM\_REG、SSM\_WR、SSM\_RD三个子模块组成，如图32所示。



图32 SSM模块内部架构设计图

其中，子模块的介绍如下：

SSM\_REG模块：统计模块。每来一个报文，此报文字节数为FAST\_MD0[107:96]-12’d32,接收报文的bit数ssm\_bit\_num\_cnt <= (ssm\_bit\_num\_cnt + (FAST\_MD0[107:96]-12’d32)\*8),每个报文接收结束时，接收报文个数ssm\_pkt\_num\_cnt <= ssm\_pkt\_num\_cnt+1。

SSM\_WR模块：报文存储模块。根据RAM中已经存放报文个数（ram\_pkt\_num\_cnt）计算当前报文在RAM中存放的起始地址；当指定要存放时间戳的数据进入SSM中时，将LCM模块的时间戳数据赋给指定的位置，再将报文存入RAM中相应的的地址空间，每存放一拍报文数据，地址+1指向下一地址，直到整个报文存放完毕，每存放一完整报文，已经存放报文个数ram\_pkt\_num\_cnt <= ram\_pkt\_num\_cnt+1;

SSM\_RD模块：报文读取模块。当有读取报文的需求时，软件端根据用户想读取的报文号，解析出读取报文的起始地址，由LCM模块下发到SSM模块，SSM模块将地址发送至RAM,读信号置为有效，RAM读出的报文直接送往软件端。

#### 3.2.3.2 SSM\_REG模块

##### 3.2.3.2.1 模块接口设计

SSM\_REG模块的接口信号定义图如图33所示。



图33 SSM\_REG模块接口信号定义图

SSM\_REG模块的接口信号定义表如表21所示。

表21 SSM\_REG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\_ssm\_reg\_data | 134 | Input | 接收测试报文 |
| in\_ssm\_reg\_data\_wr | 1 | Input | 接收测试报文有效 |
| in\_ssm\_reg\_valid | 1 | Input | 接收了完整的报文 |
| in\_ssm\_reg\_valid\_wr | 1 | Input | 接收了完整的报文有效 |
| reset\_reg | 1 | Input | 重置信号（高有效） |
| ssm\_bit\_reg2lcm | 64 | Output | 接收报文bit数 |
| ssm\_pkt\_num2lcm | 64 | Output | 接收报文个数 |

SSM\_REG模块的流程图如图34所示。



图34 SSM\_REG模块流程图

##### 3.2.3.2.2 模块实现

SSM\_REG模块的状态机图如图35所示。



图35 SSM\_REG模块状态机图

**IDLE\_S：**初始状态。若重置信号有效（reset \_reg<= 1'b1），跳转至CLEAR\_S状态；若测试报文开始进入SSM\_REG（in\_ssm\_reg\_data[133:132] == 2'b01 && in\_ssm\_reg\_data\_wr == 1'b1 && reset\_reg == 1'b0），跳转至STAT\_S状态；

**CLEAR\_S：**清零状态。将接收报文个数计数器（ssm\_pkt\_num\_cnt）和总接收bit计数器（ssm\_bit\_num\_cnt）清零后（reset\_reg == 1'b0），跳转至IDLE\_S状态；

**STAT\_S：**统计状态。根据FAST\_MD0[107:96]计算报文的总bit数，并更新总接收bit计数器（ssm\_bit\_num\_cnt），当接收到报文尾时（in\_ssm\_reg\_data[133:132] == 2'b10 && in\_ssm\_reg\_data\_wr == 1'b1），接收报文个数计数器（ssm\_pkt\_num\_cnt）统计+1，跳转至IDLE\_S状态。

#### 3.2.3.3 SSM\_WR模块

##### 3.2.3.3.1 模块接口设计

SSM\_WR模块的接口信号定义图如图36所示。



图36 SSM\_WR模块接口信号定义图

SSM\_WR模块的接口信号定义表如表22所示。

表22 SSM\_WR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\_ssm\_wr\_data | 134 | Input | 接收的测试报文 |
| in\_ssm\_wr\_data\_wr | 1 | Input | 接收的测试报文有效 |
| in\_ssm\_wr\_valid | 1 | Input | 接收了完整的测试报文 |
| in\_ssm\_wr\_valid\_wr | 1 | Input | 接收的完整的测试报文有效 |
| lcm2ssm\_wr\_time | 64 | Input | lcm记录的时间戳 |
| reset\_wr | 1 | Input | RAM重置信号 |
| ssm\_wr\_addr | 11 | Output | 报文存放的地址 |
| out\_ssm\_wdata | 134 | Output | 存入RAM的报文 |
| out\_ssm\_wdata\_wr | 1 | Output | 存入RAM的报文有效 |

SSM\_WR模块的流程图如图37所示。



图37 SSM\_WR模块流程图

##### 3.2.3.3.2 模块实现

SSM\_WR模块的状态机图如图38所示。



图38 SSM\_WR模块状态机图

**IDLE\_S：**初始状态，若测试流量进入SSM\_WR模块（in\_ssm\_wr\_data[133:132] == 2'b01 && in\_ssm\_wr\_data\_wr == 1'b1 && reset\_wr == 1'b0），根据RAM已存放报文个数（ram\_pkt\_num\_cnt）算得报文存放的起始地址，将报文存入RAM，跳转到WRITE\_S状态，若重置信号有效（reset\_wr == 1’b0），跳转到CLEAR\_S状态;

**WRITE\_S：**写状态。判断当前一拍报文是否为记录时间戳的报文，若是则将LCM中时间戳数据赋给指定位置再将报文存入RAM；若不是则直接存入RAM，当报文存放结束时（in\_ssm\_wr\_data[133:132]==2’b10），RAM已经存放报文个数（ram\_pkt\_num\_cnt）统计加1，跳转到IDLE\_S状态。

**CLEAR\_S：**清零状态。将RAM清零，并跳转到IDLE\_S。

#### 3.2.3.4 SSM\_RD模块

##### 3.2.3.4.1 模块接口设计

SSM\_RD模块的接口信号定义图如图39所示。



图39 SSM\_RD模块接口信号定义图

SSM\_RD模块的接口信号定义表如表23所示。

表23 SSM\_RD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 |
| lcm2ram\_rd | 1 | Input | 读请求信号 |
| lcm2ram\_rd\_addr | 11 | Input | 读报文首地址 |
| ram2ssm\_rd\_data | 134 | Input | 从RAM中读取的报文 |
| ram\_rd\_addr | 11 | Output | 从RAM中读取的报文的地址 |
| ram\_rd | 1 | Output | 读使能 |
| out\_ssm\_rd\_data | 134 | Output | RAM中读取后输出的报文 |
| out\_ssm\_rd\_data\_wr | 1 | Output | RAM中读取后输出的报文有效 |
| out\_ssm\_rd\_valid | 1 | Output | 输出了完整的报文 |
| out\_ssm\_rd\_valid\_wr | 1 | Output | 输出的完整报文有效 |

SSM\_RD模块的流程图如图40所示。



图40 SSM\_RD模块流程图

##### 3.2.3.4.2 模块实现

SSM\_RD模块的状态机图如图41所示。



图41 SSM\_RD模块状态机图

**IDLE\_S：**SSM模块初始状态。若读请求信号有效（lcm2ssm\_rd == 1'b1），将读取报文的起始地址写入RAM读地址端口，读使能端置为有效（ram\_rd <=1’b1）状态跳转到HAUNT1\_S状态。

**HAUNT1\_S：**间隔一拍状态。因为从RAM中读数据需要经过两拍才能生效，所以在IDLE\_S写入的读地址和读信号需要间隔两拍后才能有效。在HAUNT1\_S中，给出下一读地址和读信号，并无条件跳转到HAUNT2\_S。

**HAUNT2\_S：**间隔两拍状态。给出再下一拍的读地址和读信号，并无条件跳转到READ\_S。

**READ\_S：**读取报文状态。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(ram\_rd\_addr)即为目的报文所在块号，读取报文的起始地址（ram\_rd\_addr）为报文号乘以128所得值，每读取一拍分组，地址ram\_rd\_addr + 1，并将分组送往软件端，直到读取了一个完整的报文（ram2ssm\_data[133:132] == 2’b10），读取报文结束，读使能信号置为0（ram\_rd <= 1’b0），跳转至RD\_FIN\_S状态。否则跳转到READ\_S继续从RAM中读取报文。

**RD\_FIN\_S：**读完成状态，防止重复的读取同一个报文。若读信号有效（lcm2ram\_rd==1’b1），输出信号置0（out\_ssm\_rd\_data<=134’b0）,并维持在RD\_FIN\_S状态；若读信号无效（lcm2ram\_rd==1’b0），输出信号置0（out\_ssm\_rd\_data<=134’b0）,跳转到IDLE\_S状态。

#### 3.2.3.5 RAM设计

每个报文分配128拍的地址空间，设计一个可以存放16个报文的RAM，需要2K×134位的地址空间。

Block RAM 基本单元为18Kb Block RAM和2个18Kb Block RAM构成的36Kb Block RAM，其他配置均以这两种基本单元展开。

RAM选择2K×134，将由7个2K×18 的36Kb Block RAM和1个2K×9的18Kb Block RAM构成2K×135位RAM，利用率达134/135=99%。

### 3.2.4 PGM模块

#### 3.2.4.1 PGM顶层模块

##### 3.2.4.1.1 模块接口设计

PGM模块的接口信号定义图如图42所示。



图42 PGM模块接口信号定义图

PGM模块的接口信号定义表如表22所示。

表22 PGM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\_pgm\_data | 134 | Input | 输入PGM模块的数据 |
| in\_pgm\_data\_wr | 1 | Input | 输入PGM模块的数据的写信号 |
| in\_pgm\_data\_valid | 1 | Input | 输入PGM模块的报文的有效信号 |
| in\_pgm\_data\_valid\_wr | 1 | Input | 输入PGM模块的报文的有效信号的写信号 |
| in\_pgm\_data\_ready | 1 | Output | PGM模块准备接收数据的信号 |
| pgm\_config\_reset | 1 | Input | 配置参数重置信号 |
| sent\_start\_time\_n\_reg | 64 | Both | 第n个报文的起始发送时刻 |
| sent\_rate\_n\_reg | 64 | Both | 第n个报文的发送速率，即发送间隔的拍数 |
| sent\_start | 1 | Input | 测试开始信号 |
| sent\_finish | 1 | Output | 测试结束信号 |
| sent\_model | 1 | Input | 测试模式选择信号 |
| sent\_bit\_cnt | 64 | Output | 总发送bit数 |
| sent\_pkt\_0\_cnt | 64 | Input | 1号流的总发送报文个数 |
| sent\_pkt\_1\_cnt | 64 | Input | 2号流的总发送报文个数 |
| sent\_pkt\_2\_cnt | 64 | Input | 3号流的总发送报文个数 |
| sent\_pkt\_3\_cnt | 64 | Input | 4号流的总发送报文个数 |
| sent\_time\_cnt | 64 | Output | 当前测试时间 |
| sent\_time\_reg | 64 | Both | 总测试时间 |
| sent\_num\_cnt | 64 | Output | 已发送报文个数 |
| sent\_num\_reg | 64 | Both | 总发送报文个数 |
| sent\_ready | 1 | Output | 测试准备就绪信号 |
| lcm2pgm\_time | 64 | Input | LCM模块提供给PGM模块的时间戳 |
| out\_pgm\_data | 134 | Output | PGM模块转发的数据 |
| out\_pgm\_data\_wr | 1 | Output | PGM模块转发的数据的写信号 |
| out\_pgm\_data\_valid | 1 | Output | PGM模块转发的报文的有效信号 |
| out\_pgm\_data\_valid\_wr | 1 | Output | PGM模块转发的报文的有效信号的写信号 |
| out\_pgm\_data\_ready | 1 | Input | FPGA OS准备接收数据的信号 |

##### 3.2.4.1.2 模块实现

PGM模块由三个子模块共同组成，包括：PGM\_WR子模块、PGM\_RD子模块以及PGM\_RAM子模块。其中，PGM\_WR子模块用于将收到的构造报文写入PGM\_RAM并构造时刻调度表表项传递至PGM\_RD模块。PGM\_RD子模块用于根据时刻调度表的调度结果生成并发流量。两者将在3.5.3.2和3.5.3.3小节中详细介绍。PGM\_RAM子模块用于存储软件端所构造的用于生成不同类型并发流量的报文。PGM模块的信号接口定义见图43所示。



图43 PGM模块内部结构设计图

#### 3.2.4.2 PGM\_WR模块

##### 3.2.4.2.1 模块接口设计

PGM\_WR模块的接口信号定义图如图44所示。



图44 PGM\_WR模块接口信号定义图

PGM\_WR模块的接口信号定义表如表23所示。

表23 PGM\_WR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\_pgm\_data | 134 | Input | 输入PGM模块的数据 |
| in\_pgm\_data\_wr | 1 | Input | 输入PGM模块的数据的写信号 |
| in\_pgm\_data\_valid | 1 | Input | 输入PGM模块的报文的有效信号 |
| in\_pgm\_data\_valid\_wr | 1 | Input | 输入PGM模块的报文的有效信号的写信号 |
| in\_pgm\_data\_ready | 1 | Output | PGM模块准备接收数据的信号 |
| pgm\_config\_reset | 1 | Input | 配置参数重置信号 |
| sent\_start\_time\_n\_reg | 64 | Both | 第n个报文的起始发送时刻 |
| sent\_rate\_n\_reg | 64 | Both | 第n个报文的发送速率，即发送间隔的拍数 |
| table\_entry\_wr | 1 | Input | 时刻调度表表项是否有效 |
| wr2ram\_n\_wr\_en | 1 | Output | PGM\_WR模块写PGM\_RAM的数据的写信号 |
| wr2ram\_n\_wr\_addr | 10 | Output | PGM\_WR模块写PGM\_RAM的数据的写地址 |
| wr2ram\_n\_wdata | 134 | Output | PGM\_WR模块写PGM\_RAM的数据 |
| {sent\_start\_time\_n\_reg,  sent\_rate\_n\_reg,  pkt\_n\_addr} | 138 | Output | PGM\_WR模块传递给PGM\_RD模块的时刻调度表表项 |
| table\_entry\_flag | 1 | Output | PGM\_WR模块转发时刻调度表表项的开始结束信号 |

##### 3.2.4.2.2 模块实现

PGM\_WR模块负责将软件端配置的用于构造并发流量的报文存储到PGM\_RAM中，并将写入RAM的地址同报文发送时刻、报文发送间隔共同转发到PGM\_RD模块中，填入时刻调度表内。

在PGM\_RAM中，我们采用报文存储块的形式存储报文，即为每一个构造的用于生成并发流量的报文分配一个128拍的报文存储块。同时，在PGM\_WR模块内部维护一个pkt\_count的计数器，用于计数报文的个数，同时也是报文存储块的首地址。因此，每收到一个构造报文，则将首地址（pkt\_count）和报文发送时刻（sent\_start\_time\_n\_reg）、报文发送间隔（sent\_rate\_n\_reg）共同转发到PGM\_RD模块，写入时刻调度表内。

具体实现的状态机如图45所示。



图45 PGM\_WR模块状态机图

**IDLE\_S：**空闲状态。接收来自DMUX模块的数据，当收到FAST-Pkt报文头（in\_pgm\_data[133:132] == 2'b01）并且写数据信号有效（in\_pgm\_data\_wr == 1'b1）时，跳转至STORE\_S，并将构造好的时刻调度表表项转发到PGM\_RD模块。当收到软件端的重置信号时，跳转至IDLE\_S状态，清空PGM\_WR模块内的寄存器。

**STORE\_S：**存储状态。将软件端构造的报文体存储到PGM\_RAM中，跳转至STORE\_S。并在存储到FAST-Pkt报文尾部时，将存储报文的首地址传给PGM\_RD模块，同时跳转至IDLE\_S。

#### 3.2.4.3 PGM\_RD模块

##### 3.2.4.3.1 PGM\_RD顶层模块

###### 3.2.4.3.1.1 模块接口设计

PGM\_RD模块的接口信号定义图如图46所示。



图46 PGM\_RD模块接口信号定义图

PGM\_RD模块的接口信号定义表如表24所示。

表24 PGM\_RD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 |
| pgm\_config\_reset | 1 | Input | 配置参数重置信号 |
| sent\_start | 1 | Input | 测试开始信号 |
| sent\_finish | 1 | Output | 测试结束信号 |
| sent\_model | 1 | Input | 测试模式选择信号 |
| ram2rd\_data | 134 | Input | 从PGM\_RAM中读出的数据 |
| {sent\_start\_time\_n\_reg,  sent\_rate\_n\_reg,  pkt\_n\_addr} | 138 | Output | PGM\_WR模块传递给PGM\_RD模块的时刻调度表表项 |
| table\_entry\_flag | 1 | Output | PGM\_WR模块转发时刻调度表表项的开始结束信号 |
| sent\_bit\_cnt | 64 | Output | 总发送bit数 |
| sent\_pkt\_0\_cnt | 64 | Input | 1号流的总发送报文个数 |
| sent\_pkt\_1\_cnt | 64 | Input | 2号流的总发送报文个数 |
| sent\_pkt\_2\_cnt | 64 | Input | 3号流的总发送报文个数 |
| sent\_pkt\_3\_cnt | 64 | Input | 4号流的总发送报文个数 |
| sent\_time\_cnt | 64 | Output | 当前测试时间 |
| sent\_time\_reg | 64 | Both | 总测试时间 |
| sent\_num\_cnt | 64 | Output | 已发送报文个数 |
| sent\_num\_reg | 64 | Both | 总发送报文个数 |
| sent\_ready | 1 | Output | 测试准备就绪信号 |
| rd2ram\_rd | 1 | Output | 读PGM\_RAM的读信号 |
| rd2ram\_addr | 10 | Output | 读PGM\_RAM的读地址 |
| out\_pgm\_data | 134 | Output | PGM模块转发的数据 |
| out\_pgm\_data\_wr | 1 | Output | PGM模块转发的数据的写信号 |
| out\_pgm\_data\_valid | 1 | Output | PGM模块转发的报文的有效信号 |
| out\_pgm\_data\_valid\_wr | 1 | Output | PGM模块转发的报文的有效信号的写信号 |
| out\_pgm\_data\_ready | 1 | Input | GOE模块准备接收数据的信号 |

###### 3.2.4.3.1.2 模块实现

PGM\_RD模块由SCHEDULE模块和PKT\_SEND模块共同组成，如图47所示。



图47 PGM\_RD模块内部架构设计图

SCHEDULE模块负责接收来自PGM\_WR模块转发的时刻调度表表项，并写入内部RAM中。内部调度模块调度输出符合发送条件的报文在PGM\_RAM中的地址，并转发给PKT\_SEND模块。同时，更新对应的时刻调度表表项。

PKT\_SEND模块负责从PGM\_RAM中读取发送的报文，并进行发送。

##### 3.2.4.3.2 SCHEDULE模块

###### 3.2.4.3.2.1 模块接口设计

SCHEDULE模块的接口信号定义图如图48所示。



图48 SCHEDULE模块接口信号定义图

SCHEDULE模块的接口信号定义表如表25所示。

表25 SCHEDULE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 |
| {sent\_start\_time\_n\_reg,  sent\_rate\_n\_reg,  pkt\_n\_addr} | 138 | Output | PGM\_WR模块传递给PGM\_RD模块的时刻调度表表项 |
| table\_entry\_flag | 1 | Output | PGM\_WR模块转发时刻调度表表项的开始结束信号 |
| pgm\_config\_reset | 1 | Input | 配置参数重置信号 |
| sent\_start | 1 | Input | 测试开始信号 |
| sent\_finish | 1 | Output | 测试结束信号 |
| sent\_model | 1 | Input | 测试模式选择信号 |
| sent\_pkt\_addr | 10 | Input | 读PGM\_RAM中的数据的读地址 |
| sent\_pkt\_rd | 1 | Input | 读PGM\_RAM中的数据的读信号 |
| sent\_time\_cnt | 64 | Output | 当前测试时间 |
| sent\_time\_reg | 64 | Both | 总测试时间 |
| sent\_num\_cnt | 64 | Output | 已发送报文个数 |
| sent\_num\_reg | 64 | Both | 总发送报文个数 |
| sent\_ready | 1 | Output | 测试准备就绪信号 |

###### 3.2.4.3.2.2 模块实现

SCHEDULE模块负责存储用于生成多类型并发流的时刻调度表，并在当前时刻同每一条流的发送时刻进行比较，从而确定PKT\_SEND模块在当前时刻所应在PGM\_RAM中所读取的报文。

3.2.4.3.2.2.1 时刻调度表设计

时刻调度表用于记录报文的发送时刻、报文发送间隔以及报文在PGM\_RAM中的存储地址，如表 所示。在当前时刻等于表项中报文的发送时刻时，将报文在PGM\_RAM中的存储地址转发给PKT\_SEND模块，用于从PGM\_RAM中读取报文并发送。并且修改报文的发送时刻，重新将表项写入对应的寄存器中。

表26 时刻调度表

|  |  |  |
| --- | --- | --- |
| **报文发送时刻Tsend** | **报文发送间隔Tinternal** | **报文存储地址Addr** |
| 0 | 13432 | 0x00 |
| 100 | 23668 | 0x80 |

时刻调度表中报文的发送时刻和报文发送间隔均由软件端配置。现对时刻调度表的具体操作流程进行举例说明。

报文1的起始发送时刻是0，当sent\_time\_cnt为0时刻时，报文1满足发送条件，读取对应的报文存储地址0x00送PKT\_SEND模块，并将报文发送时刻修改为13432，重新写入到时刻调度表中，等待时刻到达13432再次发送报文。当时刻到达100时，便读出报文2的地址0x80并转发给PKT\_SEND模块，同时修改报文发送时刻为100 + 23668 = 23768，重新写入时刻调度表中，等待时刻到达23768再次发送报文2。

在FAST-ANT 2.0中，每一拍数据是134位，即：6位+128位，也就是说一拍能够携带16字节的报文数据。对于最小报文64字节，需要6拍发送完成；对于最大报文1518字节，需要97拍（包括了前两拍的Metadata）。

报文的发送速率是通过控制报文之间的发送间隔来实现的。在这里，我们给出速率跟拍数的关系表达式，如式（1）所示。

(1)

其中，sent\_rate\_reg是报文发送间隔拍数，pkt\_length是报文长度，speed\_precent是线速的百分比（为整数，例如：10代表百分之10）。

对于最小报文64字节，以百分百的线速发送报文，所需要的间隔是67拍；以百分之五十的线速发送报文，所需要的间隔是134拍；对于最大报文1518字节，以百分百的线速发送报文，所需要的间隔是1229拍；以百分之五十的线速发送报文，所需要的间隔是2459拍；从以上的数据可以看出，报文自身的发送时间和报文的发送间隔相比较，数量级上的差距很大。因此，理论上，只要软件端合理配置报文的起始发送时间，同一时刻有多个报文满足发送条件的概率极小，甚至在某段时间内，概率为零。这也为时刻调度表精确调度报文的发送提供了理论基础。

在FAST-ANT 2.0中，我们支持用户构造四大类不同类型的流量（可参考3.2.4.4.2小节），并在PKT\_SEND模块中将这四大类不同类型的流衍生出更多的子流，从而实现多条并发流的生成。因此，在时刻调度表中只需要存储四条表项。每一个表项中有三个子表项，用三个寄存器分别存储报文发送时刻、报文发送间隔以及报文存储地址。总共需要十二个寄存器来保存整个时刻调度表即可。

3.2.4.3.2.2.2 状态机设计

SCHEDULE模块的状态机图如图49所示。



图49 SCHEDULE模块状态机图

**IDLE\_S：**空闲状态。当收到来自软件端配置的测试开始信号有效（sent\_start == 1'b1 && sent\_finish == 1'b0）时，跳转到POLL\_S。当收到来自软件端的配置信息清空信号（pgm\_config\_reset == 1'b1）时，跳转到RST\_S。

**POLL\_S：**轮询状态。对比当前时刻和时刻调度表内报文发送时刻是否相等。如果相等，则将对应报文的存储地址发送至PKT\_SEND模块，并继续轮询比较；如果不相等，则继续轮询比较。直到测试时间超时（sent\_time\_cnt >= sent\_time\_reg）或发送了指定个数的报文（sent\_num\_cnt >= sent\_num\_reg）后，跳转到IDLE\_S。如果当前时刻大于等于报文的发送时刻（sent\_time\_cnt >= sent\_start\_time\_n\_reg），则跳转至SEND\_S。

**SEND\_S：**发送报文状态。如果发送至报文尾部（ram2rd\_data[133:132] == 2'b10），则跳转至POLL\_S；否则，跳转至SEND\_S。

**CLEAR\_S：**重置状态。清空时刻调度表时刻调度表表项以及所有配置信息寄存器的值，并在重置信号无效（pgm\_config\_reset == 1'b0）后跳转到IDLE\_S。

##### 3.2.4.3.3 PKT\_SEND模块

###### 3.2.4.3.3.1 模块接口设计

PKT\_SEND模块的接口信号定义图如图50所示。



图50 PKT\_SEND模块接口信号定义图

PKT\_SEND模块的接口信号定义表如表27所示。

表27 PKT\_SEND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 |
| st\_n | 1 | Input | 复位信号，低有效 |
| sent\_pkt\_addr | 10 | Input | 读PGM\_RAM中的数据的读地址 |
| sent\_pkt\_rd | 1 | Input | 读PGM\_RAM中的数据的读信号 |
| sent\_pkt\_0\_cnt | 64 | Input | 1号流的总发送报文个数 |
| sent\_pkt\_1\_cnt | 64 | Input | 2号流的总发送报文个数 |
| sent\_pkt\_2\_cnt | 64 | Input | 3号流的总发送报文个数 |
| sent\_pkt\_3\_cnt | 64 | Input | 4号流的总发送报文个数 |
| sent\_time\_cnt | 64 | Output | 当前测试时间 |
| ram2rd\_data | 134 | Input | 读PGM\_RAM中的数据 |
| rd2ram\_rd | 1 | Output | 读PGM\_RAM的读信号 |
| rd2ram\_addr | 10 | Output | 读PGM\_RAM的读地址 |
| out\_pgm\_data | 134 | Output | PGM模块转发的数据 |
| out\_pgm\_data\_wr | 1 | Output | PGM模块转发的数据的写信号 |
| out\_pgm\_data\_valid | 1 | Output | PGM模块转发的报文的有效信号 |
| out\_pgm\_data\_valid\_wr | 1 | Output | PGM模块转发的报文的有效信号的写信号 |
| out\_pgm\_data\_ready | 1 | Input | GOE模块准备接收数据的信号 |

###### 3.2.4.3.3.2 模块实现

PKT\_SEND模块负责接收来自SCHEDULE模块转发的读PGM\_RAM中数据的读信号和读地址，并从PGM\_RAM中读出数据并发送。具体实现的状态机如图51所示。



图51 PKT\_SEND模块状态机图

**IDLE\_S：**空闲状态。当收到来自SCHEDULE模块的读信号有效时（sent\_pkt\_rd == 1'b1），将rd2ram\_rd置为1，rd2ram\_addr = sent\_pkt\_addr，跳转到HAUNT1\_S。

**HAUNT1\_S：**等待一拍状态。读PGM\_RAM数据生效需要在两拍后，因此，在HAUNT1\_S等待一拍，rd2ram\_addr = rd2ram\_addr + 1，无条件跳转到HAUNT2\_S。

**HAUNT2\_S：**等待两拍状态。等待两拍状态，rd2ram\_addr = rd2ram\_addr + 1，无条件跳转到READ\_S。

**READ\_S：**读并发送状态。读出PGM\_RAM中的数据并生成对应子流后发送，修改计数器值，并跳转到READ\_S。直到读到报文尾（ram2rd\_data[133:132] == 2'b10），发送报文尾并跳转到IDLE\_S。

#### 3.2.4.4 PGM\_RAM

##### 3.2.4.4.1 PGM\_RAM设计

PGM\_RAM用于存储生成并发流量的构造报文。软件端构造用于生成并发流量的特定报文，并下发至硬件PGM\_WR模块，PGM\_WR模块将报文依次写入PGM\_RAM内。在测试初始化结束后，PGM\_RD模块再根据时刻调度表从PGM\_RAM中依次将报文读出并发送。

基于以上流程，PGM\_RAM可以考虑采用简单双端口RAM（Simple Dual-Port RAM），即一个端口用于写入数据，另一个端口用于读取数据。

软件端所构造的FAST-Pkt在硬件中采用的是数据拍的形式传递的，每一拍数据宽度为134位（6 + 128，参考2.3小节）。对于每一个报文，在PGM\_RAM中分配固定大小的存储块，数据宽度为134位，深度为128。在PGM\_RAM中共存储4类报文。

综上所述，PGM\_RAM的数据位宽为134位，地址位宽为10位，共占用4个36Kb Block RAM。

##### 3.2.4.4.2 生成并发流量的构造报文

为了更好地支持FAST-ANT 2.0所提供的并发流量功能，同时简化功能实现，我们允许用户在软件端定义构造四大类流，并在硬件端对这四大类流进行修改，衍生出不同的子流，最终实现100000条流并发。

由源IP地址、目的IP地址、源端口号、目的端口号以及协议共同组成的五元组可以唯一确定一条流。因此，我们可以允许用户构造四类报文，并针对五元组中源IP地址、目的IP地址、源端口号以及目的端口号这四项进行变化，从而衍生出100000条不同子流。

假设现在用户在软件端构造了四种不同类型的报文1、报文2、报文3和报文4，那么在硬件中，针对报文1，我们修改源IP地址，比如对地址自加一；针对报文3，我们修改源端口号，比如进行累加随机数。通过这样的操作，便可以在这四大类流的基础上衍生出多条子流，从而生成不同的并发流量。

目前，在FAST-ANT 2.0中，我们对这四项累加以当前时刻作为随机种子，生成的16位随机数，从而实现随机改变源IP地址、目的IP地址、源端口号以及目的端口号的目的。

### 3.2.5 MUX模块

#### 3.2.5.1 模块接口设计

MUX模块的接口信号定义图如图52所示。



图52 MUX模块接口信号定义图

MUX模块的接口信号定义表如表28所示。

表28 MUX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 |
| lcm2mux\_data | 134 | Input | LCM模块转发给MUX模块的数据 |
| lcm2mux\_data\_wr | 1 | Input | LCM模块转发给MUX模块的数据的写信号 |
| lcm2mux\_data\_valid | 1 | Input | LCM模块转发给MUX模块的报文的有效信号 |
| lcm2mux\_data\_valid\_wr | 1 | Input | LCM模块转发给MUX模块的报文的有效信号的写信号 |
| ssm2mux\_data | 134 | Input | SSM模块转发给MUX模块的数据 |
| ssm2mux\_data\_wr | 1 | Input | SSM模块转发给MUX模块的数据的写信号 |
| ssm2mux\_data\_valid | 1 | Input | SSM模块转发给MUX模块的报文的有效信号 |
| ssm2mux\_data\_valid\_wr | 1 | Input | SSM模块转发给MUX模块的报文的有效信号的写信号 |
| mux2port\_0\_rd | 1 | Input | MUX模块的转发选择信号 |
| mux2port\_0\_data | 134 | Output | MUX模块转发给端口0的数据 |
| mux2port\_0\_data\_wr | 1 | Output | MUX模块转发给端口0的数据的写信号 |
| mux2port\_0\_data\_valid | 1 | Output | MUX模块转发给端口0的报文的有效信号 |
| mux2port\_0\_data\_valid\_wr | 1 | Output | MUX模块转发给端口0的报文的有效信号的写信号 |

#### 3.2.5.2 模块实现

MUX模块负责根据输出选择来决定将来自SSM模块的FAST-Pkt报文，还是来自LCM模块的FAST-Pkt报文通过端口0上送到软件端处理分析。通过一个简单的状态机就可以实现上述功能，状态机图如图53所示。



图53 MUX模块状态机图

**IDLE\_S：**空闲状态。如果输出选择信号为高（mux2port\_0\_rd == 1'b1），且SSM模块转发报文给MUX模块（ssm2mux\_data[133:132] == 2'b01），则跳转至SSM\_SEND\_S；如果输出选择信号为低（mux2port\_0\_rd == 1'b0），且LCM模块转发报文给MUX模块（lcm2mux\_data[133:132] == 2'b01），则跳转至LCM\_SEND\_S

**SSM\_SEND\_S：**转发SSM状态。转发SSM模块的报文，直到报文尾部（ssm2mux\_data[133:132] == 2'b10），跳转至IDLE\_S。

**LCM\_SEND\_S：**转发LCM状态。转发LCM模块的报文，直到报文尾部（lcm2mux\_data[133:132] == 2'b10），跳转至IDLE\_S。

# 4 可读写寄存器汇总

表29 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** |
| SSM | lcm2ssm\_addr | 写 | 20 | 20’b0 | 读取报文的首地址 |
| protocol\_type | 写 | 8 | 8’b0 | 特定报文类型 |
| ssm\_pkt\_num | 读 | 64 | 64’b0 | 记录总接收报文数量 |
| ssm\_bit | 读 | 64 | 64’b0 | 记录总接收bit数量 |
| PGM | sent\_start\_time\_n\_reg | 读写 | 64 | 64’b0 | 第n个报文的起始发送时刻 |
| sent\_rate\_n\_reg | 读写 | 64 | 64’b0 | 第n个报文的发送速率，即发送间隔的拍数 |
| sent\_bit\_cnt | 读 | 64 | 64’b0 | 总发送bit数 |
| sent\_pkt\_n\_cnt | 读 | 64 | 64’b0 | 第n个报文的总发送报文个数 |
| sent\_time\_cnt | 读 | 64 | 64’b0 | 当前测试时间 |
| sent\_time\_reg | 读写 | 64 | 64’b0 | 总测试时间 |
| sent\_num\_cnt | 读 | 64 | 64’b0 | 已发送报文个数 |
| sent\_num\_reg | 读写 | 64 | 64’b0 | 总发送报文个数 |

# 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，并设计时刻调度表。

吴尚明：

1. 设计并实现SSM模块；
2. 设计并实现DMUX模块和MUX模块。

### 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  ~  2019.07.12 | 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，卜天宇，彭锦涛 |