

Fast-ANT (Agile Network Tester)网络测试与分析套件

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号 | 修改人 | 修改时间 | 备注 |
| 1.0 | 杨翔瑞 | 2018.08.10 | 第一版 |
| 1.1 | 杨翔瑞 | 2018.08.18 | 增加硬件模块与地址空间的详细设计 |
| 1.2 | 杨翔瑞 | 2018.08.21 | 修改PGM状态机与寄存器地址划分 |
| 1.3 | 杨翔瑞 | 2018.09.16 | 增加时延probe报文格式定义 |
| 1.4 | 杨翔瑞 | 2018.09.18 | 对PGM模块进行重构 |
| 1.4.1 | 蒋越 | 2018.09.19 | 对SCM模块进行细化 |
| 1.4.2 | 杨翔瑞 | 2018.09.20 | 勘误（针对FAST 2.0） |

Fast-Ant是一款基于FPGA/CPU的轻量级网络测试分析器。用户可使用FPGA/CPU的套件对吞吐以及精确时延等信息进行精准测量。Fast-Ant具有功能可定制、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。

Fast-Ant将基于FAST开源平台进行FPGA部件开发，并基于github进行软件部件开发。预计开发周期为2个月，在9月底前完成前期开发任务，并在github发布。用户可下载软件部分首先进行部分功能的使用，或者借助FAST平台使用全部功能。

## 引言

### 背景与思路

目前在很多研究工作中，需要使用网络测试仪对设备的性能、功能进行测试与验证。其中大部分测试与验证工作均基于网络测试仪完成。然而，商用网络测试仪是一种昂贵设备，很多研究机构难以做到大批量的购买与使用，但却需要的网络测试仪对受测设备进行性能与功能测试；另外，目前大多数商用的网络测试仪也难以携带从而对用户的使用造成一定的困难；第三，随着网络领域技术的更新与迭代速度加快，商用网络测试仪很难跟上网络技术（如新的网络协议）的发展速度，这就需要用户相关用户购买新的产品，造成成本高昂。

而同时我们也观察到，在绝大多数情况下，用户仅会使用网络测试仪中几个主要功能对设备性能进行测试（吞吐率、时延、与丢包率等），而这些功能在基于更加廉价的FPGA-CPU平台实现可行性极高；另外，由于FPGA-CPU平台本身具有的良好的可编程性，通过高可扩展的架构，能够在其中方便地添加功能从而支持新的、定制的测试功能，使得基于FPGA-CPU的网络测试平台能够具有很多商用网络测试仪难以具备的灵活性；而最后，FPGA-CPU平台的成本远远低于绝大多数商用网络测试仪，这使其很容易受到很多中小研究机构以及公司的欢迎。我们提出Fast-Ant的初衷就是为了利用FPGA+多核CPU的架构设计一种低成本、可重构、轻量级网络测试平台，从而为研究机构与企业提供一种更加便捷的网络测试解决方案。

如图一所示，FAST[2]是面向多核CPU+FPGA平台，支持互联网创新研究和计算机网络实验教学的开源项目。FAST定义了网络接口加速FPGA与Linux内核以及CPU用户空间数据交互的格式与协议，支持基于多核CPU与FPGA协同的路由交换设备的数据平面高效实现。Fast-Ant网络测试与分析套件即基于FAST平台设计开发，并支持多种网络测试与分析功能。



图一 FAST软硬协同架构图

表1展示了目前商用网络测试仪所支持并常用的几种基本网络测试功能。可以发现，用户在一般情况下需要的网络测试功能较为固定：一般包括吞吐率、时延、丢包率三个方面。对于这三种性能指标，有些机构限于条件与经费限制，使用一些开源网络测试软件进行测量，但是由于软件本身受到操作系统进程调度、加解锁等不可控因素影响，极难保证测量结果的准确性；另外一些机构使用商用网络测试仪进行测量，但是商用网络测试仪由于支持很多并非必要的功能，价格高昂，很多组织难以承受。

表一 商用网络测试仪所支持的常用基本测试功能

|  |  |  |
| --- | --- | --- |
| 功能 | 可调参数 | 实例 |
| 吞吐率测试 | 报文大小、协议类型 | 测量某设备对大小为64B-1460B间满足泊松分布的报文处理的吞吐率。 |
| 时延测试 | 报文大小、协议类型、发送速率 | 测量在10Gbps发送速率下64B报文的P99时延。 |
| 丢包率测试 | 报文大小、协议类型、发送速率 | 测试在10Gbps发送速率下256B大小报文的平均丢包率。 |

结合这三种网络设备基本性能指标，我们认为：使用FPGA能够很好地进行精确测量，结合软件进行管理配置与结果分析展示，既能够节省极高的设备采购成本，又能提供精确测量数据。另外，由于FPGA出色的可编程特性，可以根据使用者特定需求进行定制化的功能开发。

## 测试功能

### 吞吐率测试

Fast-Ant的第一个功能是吞吐率测试。吞吐率是指在没有帧丢失的情况下，设备能够接收的最大速率。其测试思路是：在测试中以一定的速率发送一定数量的帧，并计算待测设备返回的帧，若发送与接收的帧数量相等，则提升发送的帧速率继续测试，直至出现丢包后则减小发送速率。增大和减小发送速率的方式按照二分法的原则，直至到达规定的迭代次数得到受测设备的吞吐率。



图2 单/双端口测试模式

根据测试模式该功能可分为双端口测试与单端口测试，两种模式的示意图2所示。单端口测试模式主要针对端系统，如各类基于NFV技术实现的网络功能等。如果需要测试多端口的转发设备，则可使用双端口测试模式进行吞吐率的测试。需要注意的是，在双端口测试中需要支持双向测试。

是否不同的报文大小测量得到的吞吐率不同？【是的，所以要支持不同大小包测试】

### 时延测试

时延是指从受测设备收到数据包开始到开始向目的端口复制数据包之间的时间间隔。在时延测试中，需要按照一定的持续时间发送帧，每隔固定时间将会有一个打了tag的帧被传输出去，当测试仪收回该帧时，将查看上面的时间戳，从而通过两值相减获取延时。一般情况而言，时延测试的结果应当包含每个帧长度的时延以及每个帧长度流的平均延时。延时测试可以忍受一些帧丢失，因为测试仪使用tag帧来测量延时，没有tag的帧将被丢弃。但是为了获取精确的时延结果，需要通过吞吐率测试获取受测设备支持的最大发送速率。

根据测试模式，时延测试仍然需要分为双端口测试与单端口测试，在双端口测试中将需要支持双向测试。

### 丢包率测试

丢包率测试用以测量被测设备在不同帧速率情况下的丢包数量，测试至少需要两个端口，做一对一的流量互发，也可由单向发送用于测试单项丢包率。在用户设定测试帧速率与测试协议之后，将会在测试时间内显示出各种帧长度的帧丢失情况。

## Fast-Ant总体架构

Fast-Ant基于FAST平台设计，分别在软硬件层面借助了FAST所提供的数据结构、访问硬件寄存器接口函数以及FPGA OS和UM流水线的完整或部分代码，用于提供网络测试功能。为了能够更好发挥软硬件协同的特点与优势，需要将网络测试功能中的子功能在软硬件间进行合理划分。所以，本章首先介绍Fast-Ant的子功能及其在软硬件间的划分，然后将分别介绍Fast-Ant的硬件与软件架构（3.1节与3.2节）。在3.3节我们将分别以测量吞吐率、时延与丢包率为例，详述Fast-Ant的运行流程。在3.4节，我们将介绍Fast-Ant的可扩展性，以及如何在未来支持新型协议功能测试。

### Fast-Ant硬件架构

Fast-Ant的硬件部分通过对Fast流水线进行模块扩展实现，其整体架构如图3所示。为了支持报文产生与数据收集的功能，Ant分别在Fast流水线中增加了Statistic Collecting Module (SCM)与Packet Generating Module (PGM)。



图3 Fast-Ant硬件流水线架构

其中，PGM模块位于GAC模块与GOE模块间，用于在软件模块的控制下产生报文并通知报文发送的相关参数。为了充分利用软件的灵活性与硬件时间精确、性能更优的特性，初始报文的产生将由软件完成，并通过FAST API发至PGM模块。PGM存储并控制所收到的报文的发送，这其中涉及到IPD (Inter-Packet-Delay)与相关HF (Header Fields)的更新模式（注：通过修改IPD可对报文的发送速率进行控制；通过修改HF的修改模式可以对如TTL等域字段进行自动更新）。我们通过软件配置PGM相关寄存器定义报文发送的IPD以及HF相关域的更新模式。

其原理如图4所示。首先Fast-Ant的软件端将通过Fast提供的send方法对需要发送的报文（以Fast报文格式）通过PCIe发送至Fast UM，并在报文携带的metadata中指定目的mid为PGM模块；同时，软件端也会通过Fast提供的reg\_write方法的对报文发送的IPD和HF更新模式写入PGM的相关寄存器中。PGM模块根据软件设置的IPD与HF的更新模式，在IPD间隔下产生报文并对相关域进行修改后发送至GOE模块并输出至FPGA OS。另外，若发出的报文需要记录时间戳（为了测试延时等），则软件也通过写UDO模块的相关寄存器说明报文的协议类型，从而支持UDO在报文payload中记录发送的时间戳(但是目前FAST中还没有增加UDO模块)。



图4 PGM工作原理图

SCM模块位于GME模块与GAC模块之间，用于对记录测试结果的寄存器等进行更新。由于Fast-Ant需要支持主动测量，所以将PGM置于GAC之后可以避免报文产生的功能与报文收集与解析功能在流水线中产生冲突；而将SCM模块置于GME与GAC间，即可以借助metadata与PFV对相关域的解析结果，也能够利用GME对所需收取报文进行定制化的过滤。

SCM的工作原理如图5所示。其位于GME模块与GAC模块之间，利用metadata与UDP模块产生的PFV对收到报文的相关信息进行统计记录。具体而言，首先，Fast-Ant的软件端将会调用reg\_write方法为SCM模块指定需要测试的报文类型（特定的协议、流等）以及需要记录的指标（丢包数、收包数、时延等）。而当Fast硬件流水线收到报文时，将首先在FPGA OS中将时间戳记录在报文头前的metadata中，并在进入UM流水线后首先由GAC模块解析并更新metadata与产生PFV，这些内容将可能用于SCM模块对所收到报文进行统计（如收到的总TCP报文数等），并将结果写入相关计数器中，同时会将metadata中discard位置1，从而使得报文在GAC中被丢弃，使得发包与收包可同时在pipeline中完成。在测试完成后，Fast-Ant软件端将通过reg\_read方法获取SCM模块中相关计数器的值并用于计算测试结果。



图5 SCM工作原理图

### Fast-Ant软件架构

Fast-Ant软件端用于接收用户输入的测试需求与测试参数，并在测试完成后以图表和数字相结合的方式向用户展示测试结果。其整体架构如图6所示。Fast-Ant的软件端部分基于FAST库所提供的API与数据结构，分别通过Virtual Address Space与FAST Pipeline两种方式对FPGA的测试参数进行配置与获取测试数据。首先，用户在ANT的GUI界面中对测试需求与参数进行配置，GUI中的输入值将被传到ANT的OS层，并被映射到FAST或ANT的相关数据结构中；其次，ANT将会直接或通过ANT Driver调用FAST API将测试参数与需求配置到FPGA中从而运行测试程序；最后，ANT将同样借助FAST API获取相关测试结果并在经过ANT OS处理后通过GUI返回，形成测试结果。



图6 Fast-Ant软件架构图

### 基本功能设计

Fast-Ant提供RFC2544中规定的针对网络设备吞吐、时延、丢包等性能参数的测试。本节以吞吐率/丢包率测试为例，从运行流程的角度说明Fast-Ant的运行流程。

首先由ANT软件端构造所需测试报文，并通过FAST API将报文写入硬件PGM模块的片上RAM中。然后通过FAST API以配置PGM模块寄存器的方式声明各header-field的更新规则，默认情况下是对IP报文序号进行累加。在软件端同时需要向硬件设置报文的发送速率，在这里根据需求不同可以选择指定IPD或指定发送速率两种控制报文发送速率的方式进行发送。为了支持更多测试场景，ANT同样支持在软件端指定发包模式为突发/持续两种可选模式。

测试开始后，ANT软件端将通过FAST API首先构造基于FAST规范的拟测试报文格式的报文至PGM模块，同时也会通过FAST API将上述内容以配置寄存器的方式写入PGM，其中包括各域的更新规则，报文发送速率，发包模式以及发包持续总时间或发包数量。PGM模块将在相关寄存器控制下完成报文的发送任务。ANT软件端在配置PGM模块的同时，通过写寄存器的方式在SCM中指定需要统计的报文类型以及总计收包个数、系统运行时间、收包总数据量以及累计丢包个数（根据IP报文需要调位进行累加）。

PGM完成发包任务后，将运行标志位寄存器reset从而告知ANT软件端测试已完成，此时，软件端将通过FAST API在SCM与PGM模块中读取与测试结果相关寄存器，并通过计算得到最终结果，展示在web界面中。

#### 丢包率测试流程：

在丢包率测试中，需要设置的发送参数与数据收集参数如表二所示：

表二 丢包率测试支持的报文发送与数据收集参数表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **发送参数** | **pkt-level** | 协议类型 | 包格式 | 包长 | 更新模式 | w/o | w/o |
| **flow-level** | 发送速率 | IPD | 发送模式 | 持续时间 | 总包数 | w/o |
| **收集参数** | **发送** | 实时bps | 实时pps | 平均bps | 平均pps | 总计bit | 总计pkt |
| **接收** | 实时bps | 实时pps | 平均bps | 平均pps | 总计bit | 总计pkt |
| **失败** | 实时bps | 实时pps | 平均bps | 平均pps | 总计bit | 总计pkt |

丢包率测试过程的整体流程如图7所示。首先，需要ANT首先向FPGA对上述发送参数进行配置，并对所有收集参数所指定的寄存器初始化。当fast\_packet到达流水线后，将由PGM模块判断报文的metadata中from\_CPU域是否置位，若置位，则将该报文缓存到RAM中，并在相关寄存器的控制下对报文以指定的速率进行发送，并在每次发送前更新报文header-field中相关域。每发送一个报文，均需对PGM的上述6项计数器进行更新。



图7 吞吐率测试流程图

同时，ANT也会在测试前通过配置寄存器的方式告知SCM模块所需监测的接收报文协议类型及起始序号。SCM接收这些配置后，将会在收到指定报文后首先对上述12个计数器进行更新，并将收到报文的metadata的DISCARD位置1，从而使报文在UDO模块中被丢弃，防止对流水线的发包速率造成影响。整体测试过程的流程图如图7所示。

#### 吞吐率测试流程：

ANT中采用“二分法”机制对设备的丢包率进行测试。吞吐率测试以丢包率测试功能为基础进行，其需要配置的参数如表三所示。

表三 吞吐率测试支持的可配置与返回值参数表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **设置参数** | **pkt-level** | 协议类型 | 包格式 | 包长 | 更新模式 | w/o |
| **flow-level** | 速率阈值 | s | 发送模式 | 单轮时间 | 迭代轮次 |
| **收集参数** | **吞吐率** | bps | pps | w/o | w/o | w/o |
| **丢包率** | 1轮bps | 2轮bps | …… | n轮bps | w/o |
| 1轮pps | 2轮pps | …… | n轮pps | w/o |

其整体测试流程如图8所示。



图8 基于“二分法”吞吐率测试流程图

ANT针对吞吐率测试的原理是首先设置速率阈值测试在对应的速率下的丢包率，利用“二分法”实现针对受测设备真实吞吐率的不断逼近，从而得到其近似的吞吐率。首先ANT sw根据用户配置将丢包率测试相关参数通过FAST API配置到硬件；然后将发送速率初始值与单轮测试时间写入PGM模块，而ANT sw本身维护迭代轮次；ANT sw通过FAST API向PGM发送需缓存报文，并将测试启动标志位置1；当PGM中计时器检查到发包时间已达到单轮测试时间，则将测试启动标志位置0，ANT sw读取相关计数器值，并在web界面中更新当前轮次对应丢包率，并更新当前迭代轮次；若还未达到设置轮次(N)，则通过“二分法”更新下一轮次的发包速率与相关寄存器从而触发新一轮测试，直到迭代次数达到N停止，此时ANT sw将读取相关计数器值并同时将最后一轮丢包率情况与最终吞吐率通过web界面返回给用户。

#### 时延测试流程：

时延测试在获取设备实际吞吐率之后进行。首先在受测设备达到满速率情况下，间隔固定时间发送带有发送时间戳的特殊报文，当测试仪再次收到从受测设备返回的带有时间戳报文时，在接收时再次记录时间戳，从而通过对比两个时间戳的差值获取该报文在受测设备中的转发时延（忽略报文在铜缆中的传输时间）。其需要配置的参数如表四所示。

表四 时延测试支持配置的参数与返回参数表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **设置参数** | **pkt-level** | 协议类型 | 包格式 | 包长 | 更新模式 |
| **flow-level** | 发送速率 | IPD | 发送模式 | w/o |
| **收集参数** | 单个时延 | w/o | w/o | w/o | w/o |
| 平均时延 | w/o | w/o | w/o | w/o |

其整体测试流程如图9所示。



图9 时延测试流程图

ANT在对受测设备转发时延进行测量前，需要获取其真实吞吐率，之后可按照受测设备满速率线速发包，测量该情况下报文的转发时延。首先ANT sw首先对FPGA根据参数表中所列参数进行配置，同时将会向GME模块下发一条过滤规则（我们将tag报文的源端口号置为8888），从而使得所有tag报文能够被SCM模块接收并根据接收/发送时间戳计算得到精确的时延。

上述设置完成后，ANT sw通过FAST API向PGM发送初始报文，并将测试启动标志置1，而在测试开始后，由计时器控制每隔固定时间t发送一个tag报文（用于记录接收/发送时间戳）并在UDO模块中对发送时间进行记录。当SCM收到tag报文后，通过metadata中记录的接收时间戳与报文中的发送时间戳相减，并计入对应的寄存器。当测试完成后，PGM模块将测试启动标志置0。此时ANT sw读取相关寄存器值，并向web客户端返回测试结果。

## Fast-Ant详细设计

本章将对Fast-Ant软硬件接口设计、基本数据结构及各部分功能实现进行详细设计。

### 时延测量Probe报文格式定义

根据ANT对时延进行测量的原理，需要在进行时延测试时，周期性产生一个Probe报文，在报文发送时，由FPGA OS部分在报文的payload中打上发送时间戳，并当该probe报文重新被ANT收到时，再次打上接收时间戳，并将该报文上送至软件，并由相关UA计算时延等参数。为了尽可能保证时延测量时probe报文能够返回ANT，ANT v1中将首先支持IP, UDP, TCP三种格式的probe报文（其中ETH层不含VLAN）。

对于IPv4, IPv6, UDP或TCP报文，其发送时间戳将记录在报文payload的前32位（对于UDP报文，需要偏移4bit），而接收的时间戳将记录在紧接着的32位，共计消耗8个字节。软件端将通过metadata的后128位说明时延测量的probe报文类型，从而告知时间戳标记模块正确的标记位置。

对于每种形式的probe报文，均需要在报文头中进行标记，如下所示：

1. 对于IP报文，复用IP头的标志字段，3bit均为 1表示该报文为IP格式的probe报文；
2. 对于UDP报文，需要借助UDP报文头后payload的前4bit标记是否为probe报文：若为全1，则为probe报文，此时紧接着的前32为则用于标记发送时间戳，而随后的32位用于标记接收时间戳。
3. 对于TCP报文，利用TCP报文头中保留字段（位于头部长度字段与标志位字段之间）的前4比特标记是否为probe报文，若全为1，则为probe报文，此时TCP报文payload的前32位用于标记发送时间戳，而随后的32位用于标记接收时间戳。

对于时延测量probe报文，首先需要在PGM模块进行发送时在metadata字段的后128位上增加标识，从而能够被FAST OS中的UDO模块识别（报文类型、是否为probe报文），若为probe报文，需要在报文的相应位置标记时间戳。

当从接收端收到报文时，需要在FPGA OS中的时间戳模块对probe报文进行识别（报文类型、是否为probe报文），若为时延测量probe报文，则在对应位置打上接收时间戳的标记。当报文进入GPP模块时，需要对probe报文进行识别，并修改metadata，使得报文能够被时延测量的UA接收，并产生最终测试结果。

### ANT中自定义Metadata格式

### Fast-Ant软件详细设计

Fast-Ant将借助OpenBox构建原型系统，OpenBox基于Xlinx的Zynq7000 SoC芯片构架，该芯片不仅包含一块FPGA逻辑，同时带有一块ARM双核嵌入式CPU。在OpenBox中，该ARM处理器通过FAST API可以实现对FPGA片上寄存器的读写与报文的收发。为了能够简化ANT的使用，由于ARM核性能与资源限制，为了便于后期对ANT进行扩展，我们使用在外部PC使用socket透明地调用FAST API与相关数据结构。这使得外联客户端主机可以通过浏览器使用web服务的方式或者更加便捷的方式利用ANT的相关测试功能，避免受限于ARM的资源与性能。

4.1.1节介绍在ANT在配置/获取结果过程中的主要数据结构，4.1.2节对ANT sw根据通过FAST API配置相关寄存器的接口函数进行介绍，4.1.3节对基于web的ANT GUI及与后台CGI的交互过程进行介绍。

#### ANT主要数据结构

ANT使用fast\_packet向硬件流水线发送报文，fast\_packet数据结构如下：

struct fast\_packet{

struct um\_metadata um; //具体格式如下所示

u16 flag; //两字节的对齐标志；

u8 data[0]; //完整以太网报文数据；

}\_\_attribute\_\_((packet))

而在fast\_packet报文中, um\_metadata使用256bit标识报文的特征（如From\_CPU = 1用于标识该报文来自本地CPU），um\_metadata的数据结构如下图所示：

struct um\_metadata{

u64 ts:32, /\*\*< @brief 时间戳\*/

reserve:17, /\*\*< @brief 保留\*/

pktsrc:1, /\*\*< @brief 分组的来源，0为网络接口输入，1为CPU输入\*/

flowID:14; /\*\*< @brief 流ID\*/

u64 seq:8, /\*\*< @brief 分组接收序列号\*/

pst:8, /\*\*< @brief 标准协议类型（参考硬件定义）\*/

dstmid:8, /\*\*< @brief 下一个处理分组的模块ID\*/

srcmid:8, /\*\*< @brief 最近一次处理分组的模块ID\*/

len:12, /\*\*< @brief 报文长度\*/

discard:1, /\*\*< @brief 丢弃位\*/

priority:3, /\*\*< @brief 分组优先级\*/

outport:6, /\*\*< @brief 单播：分组输出端口ID，组播/泛洪：组播/泛洪表地址索引\*/

outtype:2, /\*\*< @brief 输出类型，00：单播，01：组播，10：泛洪，11：从输入接口输出\*/

inport:6, /\*\*< @brief 分组的输入端口号\*/

pktdst:1, /\*\*< @brief 分组目的，0为网络接口输出，1为送CPU\*/

pkttype:1; /\*\*< @brief 报文类型，0：数据报文，1：控制报文\*/};

综上，所有软件可读/写的寄存器/计数器可分为四类：

表五 ANT sw可读写寄存器汇总表

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PGM | 参数reg | 协议  类型 | 更新  模式 | 发送速率 | 发送  模式 | 运行状态标识 | 测试时间 | 间隔时间 | 时间戳标识 |
| **计数reg** | 平均  比特率 | 平均报文速率 | 总比特数 | 总报  文数 | 运行  时间 | 时延测量间隔时间 | w/o | w/o |
| SCM | **参数reg** | 协议  类型 | 运行状态标识 | 时间戳标识 | w/o | w/o | w/o | w/o | w/o |
| **计数reg** | 平均  比特率 | 平均报文速率 | 总比特数 | 总报  文数 | 时间差  (n个) | 运行时间 | w/o | w/o |

为了能够最大限度节省硬件寄存器资源同时便于硬件的配置过程，ANT所采用的数据结构如下所示，而ant\_para\_reg成员中的pad(共39位)将用于后续功能扩展。

struct ant\_para\_reg{

u64 proto\_type:8, //协议类型，位宽为8，最多支持256种协议类型；

update\_mode:6, //报文header-field的更新模式，位宽为6；

send\_mode:2, //报文发送模式， 0为持续，1为burst；

time\_stamp:9, //时间戳标识，高位1为代表需要时间戳，低8位表示协议类型；

pad:39; //目前作为保留位，用于后续功能扩展；

};

除上述数据结构以外，PGM与SCM模块还共需要5个64位寄存器用于配置参数，分别为：

u32 sent\_rate\_reg; //报文发送速率，以两个报文的间隔时间/10ns得到间隔cycle数;

u8 pgm\_status\_reg; //PGM模块运行状态位;

u8 scm\_status\_reg; //SCM模块运行状态位;

u64 test\_time\_reg; //测试时间;

u64 block\_time\_reg; //间隔时间; 用于定时发送时延测量报文;

上述数据结构用于定义ANT的相关配置参数，而ANT还需要定义一些计数器从而读取测试结果，这些计数器分别在ANT sw中定义为：

u64 sent\_bit\_num\_cnt; //发送总bit数；

u64 sent\_pkt\_num\_cnt; //发送总报文数；

u64 sent\_time\_cnt; //PGM共计发包时间；

u64 recv\_bit\_num\_cnt; //接收总bit数；

u64 recv\_pkt\_num\_cnt; //接收总pkt数；

u64 recv\_time\_cnt; //SCM共计收包时间；

u64 recv\_latency\_cnt; //时延测试报文所记录的时间差，间隔一定时间读取从而获取在不//增加寄存器数量的情况下获取多次测量结果。

ANT将不同测试功能针对的参数配置封装为一个函数进行处理，若用户针对某些参数未进行配置，则使用函数默认参数写入对应硬件寄存器。

#### ANT虚拟地址空间定义

由于ANT中测试参数配置与读取测试结果本质是读写硬件寄存器，所以需要对上述用到的配置参数及计数器划分硬件地址空间。根据FAST 2.0的设计规范，每个UM中模块将自己维护地址空间，通过指定dmid从而将消息定向到特定的模块，并读取或写入数据，具体软件端可读写的硬件寄存器详见4.4与4.5节的虚拟地址空间定义部分。

#### ANT软件端接口函数

首先，为了支持对丢包率的测量，软件除具体报文内容外，还需要对协议类型、header-field更新模式、发送速率、IPD、发送模式等参数在SCM与PGM中进行配置。另外，还需要设置用于统计报文发送、接收的总比特数、报文总数、实时bps/pps等多个计数器。最后，还需要提供运行状态标识位（见4.1.1节）用于修改SCM与PGM模块的状态机（注：在NM08中采用32位的虚拟地址空间，读写寄存器的范围为0~4GB（读写位宽为32bit，通过两次读写可以读取/写入64bit数据），可满足ANT的需求；

为了支持对吞吐率的测量，除上述寄存器、计数器外，还需要设置当前运行时间与单轮迭代时间等计数器用于支持使用“二分法”对吞吐率进行测量。需要说明的是，在ANT中，计时主要是通过时钟周期计数器与单时间片（在125M时钟频率的FPGA中，单时间片大小约为8ns，100M时钟频率的FPGA上，单时间片大小约为10ns）相乘实现。同时还需注意，在吞吐率测试中迭代轮次记录以及速率采用“二分法”进行调整是通过ANT sw实现的，从而简化硬件设计；

最后，为了支持对时延的测量，需要PGM模块每隔固定报文数发送一个payload中带有时间戳的时延测量报文。该功能需要在支持丢包率测量的基础上，增加一个寄存器用于需间隔的报文数，一个计数器用于比对当前间隔报文数是否达到n。同时，需要在UDP模块中识别probe报文，并依据probe报文修改metadata第二拍中的某一位用于标记probe报文，GACSCM模块根据metadata识别出时延测量的probe报文，并修改action字段，使其被上传至UA模块，UA模块根据报文体中的时间戳对时延进行计算。

ANT通过调用FAST API实现对FAST硬件流水线各模块寄存器的配置。所有对FPGA寄存器/计数器的操作，最终都通过调用fast\_reg\_wr()与fast\_reg\_rd()实现。这两个函数的原型如下：

void fast\_reg\_rd(u64 regaddr); //读硬件64位寄存器

void fast\_reg\_wr(u64 regaddr, u64 regvalue); //写硬件64位寄存器

为了在ANT中更加方便调用上述函数进行参数配置，同时便于后续功能的扩展，我们对上再提供一层逻辑封装。

int config\_ant\_parameter(struct\* ant\_para\_reg); /\*\*<配置参数，返回值0代表成功，-1为失败\*/

int config\_sent\_rate(u64 sent\_rate); /\*\*<设置发送时间，返回值含义同上\*/

int set\_pgm\_status(int flag); /\*\*<设置PGM模块标志位，返回值含义同上\*/

int set\_scm\_status(int flag); /\*\*<设置SCM模块标志位，返回值含义同上\*/

int set\_test\_time(u64 test\_time); /\*\*<设置PGM模块运行时间，返回值含义同上\*/

int set\_block\_time(u64 block\_time); /\*\*<设置时延测试报文间隔时间，返回值含义同上\*/

int reset\_pgm(); /\*\*<将PGM模块中参数与计数器值全部重置，返回值含义同上\*/

int reset\_scm(); /\*\*<将SCM模块中参数与计数器值重置，返回值含义同上\*/

另外，还需要提供一组API用于读取相应计数器并处理得到测试结果返回用户，如下所示：

int get\_drop\_bits(u64 \* drop\_bits\_num); /\*\*<获取总miss bits数\*/

int get\_drop\_pkts(u64 \* drop\_pkts\_num); /\*\*<获取总miss pkts数\*/

int get\_drop\_rate(double \* bits\_rate, double \* pkts\_rate); /\*\*<获取平均丢包率，单位分别为bps,pps\*/

int get\_average\_latency(u64 \* ave\_latency); /\*\*<获取平均时延，单位为ns\*/

int get\_all\_latency(u64 \* ave\_latency[N]); /\*\*<获取所有测试报文时延，单位为ns\*/

最后，在吞吐率测试过程中，需要ANT从软件端实现“二分法”对吞吐率的测量，为了方便调用，我们采用标准API对该功能进行封装：

int run\_ dich\_throughput (u64 low\_rate, u64 high\_rate, int round, u64 time, u64\* throughput);

**Summary**

调用二分法机制对吞吐率进行测量

**Parameters**

low\_rate (u64) 速率下界，单位为bit;

high\_rate (u64) 速率上界，单位为bit;

round (int) 迭代次数；

time (u64) 单轮运行时间；

throughput (u64 \*) 测得吞吐率，返回参数

**Description**

该函数以阻塞方式运行，完成“二分法”测量吞吐率的测试完整过程中与FPGA

数据交互。成功返回0，失败返回-1。

#### 基于web的ANT 软件端设计

为了避免繁琐的软件安装过程，便于用户使用，ANT采用基于B/S的机制构建前端，从而使得用户可以直接通过浏览器使用ANT的测试功能。OpenBox采用的是ARM处理器，目前采用文件系统预加载的方式运行，最大支持文件系统为32MB，所以目前无法搭建大型web服务器。ANT计划采用thttpd[1]搭建轻量级web服务后台，并在第一版本中通过html构建静态页面作为ANT对用户提供的GUI。

同时为了尽可能实现减小ARM处理器中的资源消耗，采用C语言作为CGI，实现后端与FPGA交互以及相关数据处理逻辑。

【界面设计将参考“小兵以太网测试仪”，本节略】

### PGM模块详细设计

PGM (Packet Generation Module)位于FAST流水线的GAC模块后与GOE模块前，用于在ANT软件控制下产生报文并进行发送。4.2.1节将对PGM模块的整体架构与输入输出信号进行介绍，4.2.2节将讨论PGM的状态机变化，4.2.3节对PGM模块中数据结构进行说明。

#### 整体框架

PGM所处位置与前后模块的连接关系如图10所示。PGM位于GAC与GOE之间，其接口信号完全与GAC的输出信号与GOE的输入信号匹配，从而减少对前后两个模块的更改。从逻辑上来看，ANT在运行开始时，来自CPU的触发报文经GAC模块通过FIFO到达PGM模块，PGM将报文连同带外控制信号写入本地RAM中，并在间隔N拍后读将报文读出并在相关信号控制下对报文的部分位进行修改（主要为序号等信息）。PGM在软件配置的相关寄存器控制下，复制并将报文、PFV值以及metadata值(目前是否打时间戳等控制信息由metadata携带)告知GOE模块，GOE模块将执行报文发送的动作。



图10 PGM模块接口与连接关系图

#### 模块接口

详细来看，PGM模块接口定义如表六所示。由于ANT中的PGM模块位于GAC与GOE之间，所以我们将GAC模块原先送往GOE模块的信号均修改为PGM模块的输入，另外PGM模块根据ANT的配置产生的报文将以GOE输入信号的格式在每一拍送至GOE模块。

表六 PGM模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst\_n | input | 1 | 复位信号 |
| gac2pgm\_md | input | 256 | 来自GAC的metadata |
| gac2pgm\_pfv | input | 1040 | 来自GAC的分组特征向量 |
| gac2pgm\_data\_wr | input | 1 | 来自GAC的数据写使能信号 |
| gac2pgm\_data | input | 134 | 来自GAC的报文数据 |
| gac2pgm\_valid\_wr | input | 1 | 来自GAC的metadata使能信号 |
| gac2pgm\_md\_valid | input | 1 | 来自GAC的metadata有效信号 |
| goe2pgm\_alf | input | 1 | PGM模块对GOE的反压信号 |
| pgm2goe\_md | output | 256 | PGM发往GOE的metadata |
| pgm2goe\_pfv | output | 1040 | PGM发往GOE的分组特征向量 |
| pgm2goe\_data\_wr | output | 1 | PGM发往GOE的数据写使能信号 |
| pgm2goe\_data | output | 134 | PGM发往GOE的报文数据 |
| pgm2goe\_valid\_wr | output | 1 | PGM发往GOE的写有效信号 |
| pgm2goe\_md\_valid | output | 1 | PGM发往GOE的metadata有效信号 |
| pgm2gac\_alf | output | 1 | PGM对GAC的反压信号 |
| pgm2gac\_sent\_start | output | 1 | PGM对SCM告知测量开始的信号 |

在此情况下，FAST UM中仅有GAC模块的输出信号名与GOE模块的输入信号名作出对应修改即可适配新加入的PGM模块。PGM作为顶层模块，其中包含PGM\_WR, PGM\_RD, PGM\_RAM三个子模块，这三个子模块的输入输出信号定义表如下所示：

表七 PGM\_WR模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst\_n | input | 1 | 复位信号 |
| gac2pgm\_md | input | 256 | 来自GAC的metadata |
| gac2pgm\_pfv | input | 1040 | 来自GAC的分组特征向量 |
| gac2pgm\_data\_wr | input | 1 | 来自GAC的数据写使能信号 |
| gac2pgm\_data | input | 134 | 来自GAC的报文数据 |
| gac2pgm\_valid\_wr | input | 1 | 来自GAC的data结束信号 |
| gac2pgm\_md\_valid | input | 1 | 来自GAC的metadata有效信号 |
| wr2rd\_md | output | 256 | 发往PGM\_RD的md信号 |
| wr2rd\_pfv | output | 1040 | 发往PGM\_RD的pfv信号 |
| wr2rd\_md\_valid | output | 1 | 发往PGM\_RD的md有效信号 |
| pgm\_bypass\_flag | output | 1 | 发往PGM\_RD的PGM旁路信号 |
| pgm\_sent\_finish\_flag | output | 1 | 发往PGM\_RD的发包结束信号 |
| wr2rd\_data\_wr | output | 1 | 发往PGM\_RD的数据有效信号 |
| wr2rd\_data | output | 134 | 发往PGM\_RD的data信号 |
| wr2rd\_valid\_wr | output | 1 | 发往PGM\_RD的data结束信号 |
| pgm\_sent\_start\_flag | output | 1 | 发往PGM\_RD的发包开始信号 |
| wr2ram\_wr | output | 1 | 发往PGM\_RAM的写有效信号 |
| wr2ram\_wdata | output | 144 | 发往PGM\_RAM的写数据 |
| wr2ram\_wr\_addr | output | 7 | 发往PGM\_RAM的写地址信号 |

表八 PGM\_RD模块输入输出信号定义表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名称** | **方向** | **宽度** | **信号描述** |
| clk | input | 1 | 时钟信号 |
| rst\_n | input | 1 | 复位信号 |
| wr2rd\_md | input | 256 | 来自PGM\_WR的md信号 |
| wr2rd\_pfv | input | 1040 | 来自PGM\_WR的pfv信号 |
| wr2rd\_md\_valid | input | 1 | 来自PGM\_WR的md有效信号 |
| pgm\_bypass\_flag | input | 1 | 来自PGM\_WR的PGM旁路信号 |
| pgm\_sent\_finish\_flag | input | 1 | 来自PGM\_WR的发包结束信号 |
| wr2rd\_data\_wr | input | 1 | 来自PGM\_WR的数据有效信号 |
| wr2rd\_data | input | 134 | 来自PGM\_WR的data信号 |
| wr2rd\_valid\_wr | input | 1 | 来自PGM\_WR的data结束信号 |
| pgm\_sent\_start\_flag | input | 1 | 来自PGM\_WR的发包开始信号 |
| pgm2goe\_md | output | 256 | PGM\_RD发往GOE的metadata |
| pgm2goe\_pfv | output | 1040 | PGM\_RD发往GOE的分组特征向量 |
| pgm2goe\_data\_wr | output | 1 | PGM\_RD发往GOE的数据写使能信号 |
| pgm2goe\_data | output | 134 | PGM\_RD发往GOE的报文数据 |
| pgm2goe\_valid\_wr | output | 1 | PGM\_RD发往GOE的data结束信号 |
| pgm2goe\_md\_valid | output | 1 | PGM\_RD发往GOE的md有效信号 |
| pgm2gac\_alf | output | 1 | PGM\_RD对GAC的反压信号 |
| pgm2gac\_sent\_start | output | 1 | PGM\_RD对SCM告知测量开始的信号 |
|  |  |  |  |

#### 模块设计

PGM的发包原理如下： 首先ANT软件端将下发一条模板报文，PGM收到该报文后将会把报文存储在片上RAM；当测试开始时，PGM不断从该RAM中读出报文，并按照一定的规则和速率利用FAST流水线进行发送。所以，我们将PGM的功能拆分为三部分，分别为：PGM\_RD, PGM\_WR与PGM\_RAM，分别用于接收模板报文，发送报文与存储报文。这三个子模块的连接关系如下图所示：



图11 PGM模块图

**PGM\_WR:** PGM\_WR模块用于解析来自CPU的模板报文，并将其存储在PGM\_RAM中，其状态图如下所示。PGM\_WR初始处在idle\_s状态，如果收到非来自ANT软件端的报文，则跳转至sent\_s状态，并执行正常的forward操作，将报文以及metadata与pfv送至GOE模块，并在执行完毕后返回idle\_s状态。而当收到来自ANT软件端的报文时，将会跳转至store\_s状态，并将所收到的data存储至PGM\_RAM中，当读取到报文尾时，PGM\_WR将跳转至wait\_s状态，在此状态下，PRM\_WR模块将会进行计时，并在计时器到达test\_time\_cnt >= test\_time\_reg时将pgm\_sent\_finish\_flag置位，并跳转至idle\_s状态，从而通知PGM\_RD停止发送报文。



图12 PGM\_WR状态机

**PGM\_RD:** PGM\_RD为PGM模块中的核心子模块，通过按照一定速率从PGM\_RAM中读取报文，并按照一定的规则修改报文中特定字段，实现发包的功能，其状态图如下所示。PGM\_RD首先处于idle\_s状态，当收到来自PGM\_WR的旁路信号，同时md, pfv, data等信号有效时，跳转至sent\_s状态，并直接转发所收到的流水线中的报文并不做任何处理，当处理到报文尾时，跳转回idle\_s状态。而当收到发包开始的消息时，将跳转至read\_s状态，并从PGM\_RAM中读取一个完整报文进行发送，当至报文尾时，将lat\_pkt\_cnt加1并跳转至wait\_s状态，并触发sent\_rate\_cnt开始计数，当到达sent\_rate\_cnt == sent\_rate\_reg的条件时，再次检查lat\_pkt\_cnt是否与lat\_pkt\_reg相等。若相等，则跳转至probe\_s状态，发送一个时延测量probe报文, 完成后根据pgm\_sent\_finish\_reg决定是否跳转到fin\_s状态还是wait\_s状态；若不相等，则跳转回read\_s状态进行下一个报文的发送，等待pgm\_sent\_finish\_reg被置位后跳转回fin\_s状态。

当模块处于fin\_s时，将会向ANT软件端发送一条通告报文（toCPU），ANT软件端收到后将会从PGM模块中读取结果数据，并将模块中的soft\_reset置位，从而触发所有寄存器清零，并回到idle\_s状态。



图13 PGM\_RD状态机

**PGM\_RAM:**

PGM\_RAM模块为标准IP核生成的144\*128的RAM模块，PGM模块接收来自CPU的报文格式，并以一定的规则、速率产生相似报文并送GOE模块。因此，首先PGM模块需要片内RAM对收到的来自软件的报文进行缓存。如下所示：

ram\_144\_128 pgm\_ram(

.aclr(~rst\_n),

.clock(clk),

.wraddress(pgm\_ram\_waddr),

.rdaddress(pgm\_rm\_raddr),

.data(pgm\_ram\_wdata),

.rden(pgm\_ram\_rd),

.wren(pgm\_ram\_wr),

.q(pgm\_ram\_rdata)

);

该ram能够存储一个最大为1500B的以太网报文，即来自CPU的最大的报文大小。当PGM收到来自GAC的报文后，将会报文存储在pgm\_ram中，并不断将报文读出并通过GOE发送。需要注意的是，报文从pgm\_ram中读取速率由PGM模块中相关寄存器控制，同时PGM中的一些寄存器将会控制如何对报文进行修改。

#### 虚拟地址空间设计

在PGM模块中，需要设置一些ANT软件可读/写的计数器与寄存器，用于软件对硬件功能进行配置并且获取测试结果。根据fast 2.0设计规范，每个模块可单独进行计数器与寄存器的编址，而PGM模块共包含3个子模块，在PGM\_WR与PGM\_RD模块中均包含ANT软件端可读写的计数器与寄存器，定义如下表所示：

表九 PGM虚拟地址空间定义

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 地址 | 寄存器名 | 位宽 | 属性R/W | 初始值 | 说明 |
| PGM\_WR | 0x0000 0001 | sent\_time\_cnt | 64 | R | 64’b0 | 当前测试时间 |
| 0x0001 0001 | sent\_time\_reg | 64 | R/W | 64’b0 | 总测试时间 |
| 0x0000 0000 | soft\_reset | 1 | W | 1’b0 | reset开关，置1后将跳转至idle状态 |
| PGM\_RD | 0x0000 0000 | soft\_reset | 1 | W | 1’b0 | reset开关，置1后将跳转至idle状态 |
| 0x0000 0001 | sent\_rate\_cnt | 32 | R | 32’b0 | 连续两pkt间拍数 |
| 0x0001 0001 | sent\_rate\_reg | 32 | R/W | 32’b0 | 连续两pkt间拍数 |
| 0x0000 0002 | lat\_pkt\_cnt | 32 | R | 32’b0 | Probe报文间报文数 |
| 0x0001 0002 | lat\_pkt\_reg | 32 | R/W | 32’b0 | Probe报文间报文数 |
| 0x0000 0003 | sent\_bit\_ cnt | 64 | R | 64’b0 | 总发送bit数 |
| 0x0000 0005 | sent\_pkt\_ cnt | 64 | R | 64’b0 | 总发送pkt数 |
| 0x0001 0010 | lat\_flag | 1 | R/W | 1’b0 | 进行时延测试标记 |

### SCM模块详细设计

SCM(Statistical Collecting Module)位于GME与GAC之间，依靠PFV与metadata对收到的报文的感兴趣数据（数量、数据量与时延等）进行统计。并在收集后将metadata中的DISCARD置位，从而使得已完成统计的报文在GAC模块中被丢弃。

#### 设计概述

##### 4.5.1.1需求分析

支持对FAST-ANT收到的报文进行相关的统计操作，从而获取测试方所感兴趣的的数据，如：报文数量、时延等。并且根据具体的需求来决定对收到的报文是进行丢弃操作，还是上送软件端处理。

##### 4.5.1.2功能描述

1) 支持对报文数据字段或对FAST提供的metadata进行解析，提取与数据统计有关的信息，如：时间戳等；

2) 支持计数操作，包括对总接收bit数计数、总接收报文数计数等；

3) 支持将统计结果写入指定寄存器，并允许软件端进行读取；

4) 支持根据不同需求对报文进行丢弃或上送软件的操作。

#### SCM详细设计

##### 4.5.2.1SCM整体框架

SCM位于GME和GAC之间，如图14所示。



图13 SCM位于FAST UM五级流水线中的位置

SCM负责获取相关统计信息，对保存测试结果的寄存器（计数器）进行更新，以及对报文进行丢弃或上送软件操作。

SCM相关的计数器如表十所示。

表十 SCM中相关计数器信息

|  |  |  |  |
| --- | --- | --- | --- |
| 计数器名称 | 计数器大小 | 计数器初始值 | 计数器描述 |
| scm\_bit\_num\_cnt | 64 | 64’b0 | 记录总接收bit数量 |
| scm\_pkt\_num\_cnt | 64 | 64’b0 | 记录总接收报文数量 |
| scm\_time\_cnt | 64 | 64’b0 | 记录总接收时间 |

FAST-ANT将SCM置于GME和GAC之间，以借助FAST所提供的Metadata和PFV数据消息来获取对相关字段（域）的解析结果。并通过软件端写寄存器（protocol\_type）操作来确定测试所需获取的报文类型，减少不必要的解析工作，从而进一步提升测试性能。

由于FAST-ANT提供主动测量，即：自定义测试报文并发送，因此需要在SCM中确定对报文是进行直接丢弃操作，还是上送软件端。这样，便可以确保主动测量和测试统计在FAST-ANT中同时进行。

##### 4.5.2.2SCM接口信号关系图

SCM位于GME与GAC之间，其信号接口与FAST流水线中原GME对GAC的输出信号与GAC来自GME的输入信号相匹配，从而减少对前后两个模块的更新与修改，同时也符合FAST UM流水线中模块间的接口定义规范。图14为SCM接口信号关系图。



GME缺输出信号（md, pfv…）

图14 SCM接口信号关系图

##### 4.5.2.3SCM接口信号定义

SCM接口信号定义如表十一所示。

表十一 SCM接口信号定义

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名称 | 方向 | 宽度 | 信号描述 |
| clk | input | 1 | 时钟信号 |
| rst\_n | input | 1 | 复位信号 |
| gme2scm\_md | input | 256 | 来自GME的Metadata数据 |
| gme2scm\_pfv | input | 1040 | 来自GME的PFV数据 |
| gme2scm\_valid | input | 1 | 来自GME的有效信号 |
| scm2gme\_alf | output | 1 | 为低，表示SCM能够接收数据 |
| gac2scm\_alf | input | 1 | 为低，表示GAC能够接收数据 |
| gac2scm\_sent\_start | input | 1 | 为高，表示测试开始 |
| scm2gac\_data\_wr | output | 1 | 来自SCM的写数据信号 |
| scm2gac\_valid | output | 1 | 来自SCM的有效信号 |
| scm2gac\_pfv | output | 位数不对 | 来自SCM的PFV数据 |
| scm2gac\_md | output | 256 | 来自SCM的Metadata数据 |

##### 4.5.2.4SCM状态转换

SCM状态机包含五种状态，分别为IDLE\_S、SEND\_S、CNT\_S、WAIT\_S和FETCH\_S。其具体状态转换如图15所示。



图15 SCM状态转换

1. IDLE\_S：

SCM初始状态，此时SCM内的所有计数器值均置0。

当FAST-ANT未启动时，SCM作为FAST流水线的一个模块，将收到的数据正常转发到下一模块，接收到gme2scm\_valid信号跳转至SEND\_S状态。

当FAST-ANT启动时，接收到GAC2scm\_sent\_start信号跳转至CNT\_S状态。

1. SEND\_S：

SCM作为FAST流水线的一个模块时的转发状态，此时将收到的数据正常转发到下一模块。

当接收到reset信号时，跳转到IDLE\_S状态。

1. CNT\_S：

SCM测试统计状态，此时SCM根据对收到的测试报文进行解析和统计来修改相应计数器。对于非测试报文则正常转发。

当接收到GAC2scm\_sent\_end信号时，跳转至WAIT\_S状态。

1. WAIT\_S：

SCM等待接收统计终止状态。在此状态中，SCM在接收到GAC2scm\_sent\_end信号后n个RTT（n由探针报文确定）后停止接收测试报文，即timeout信号，跳转到FETCH\_S状态。

如果在n个RTT内仍有测试报文，那么在WAIT\_S状态中仍然进行解析和统计操作。

1. FETCH\_S：

软件获取相关寄存器的值，并在确定新的测试开始时，发送statistic\_reset信号，跳转至IDLE\_S状态。

##### 4.5.2.5地址空间划分

SCM内部地址空间划分如表十二所示。

表十二 SCM内部地址空间划分

|  |  |  |  |
| --- | --- | --- | --- |
| 地址 | 寄存器名称 | 寄存器大小 | 寄存器描述 |
| 0x7000 0000 | protocol\_type | 3 | 测试报文类型 |
| 0x7000 0008 | scm\_bit\_num\_cnt | 64 | 记录总接收bit数量 |
| 0x7000 0048 | scm\_pkt\_num\_cnt | 64 | 记录总接收报文数量 |
| 0x7000 0088 | scm\_time\_cnt | 64 | 记录总接收时间 |

## 相关研究

### Packet Generator on NetFPGA

可以生成pcap文件，同时也可以利用生成的pcap文件在packet generator进行流量重放。正是因为有了基于pcap的报文产生机制，所以packet generator没有设计类似小兵测试仪一样的发包（设置报文内容、协议、发包速率等）功能。个人认为从实用的角度而言，应当提供RFC2544所规定的测试功能，以及类似小兵一样的用户接口才能适用大多数的应用场景。但是小兵不提供测试时延的功能，这点将在Fast-Ant中得到解决。

同时，软件通过写硬件寄存器可以控制的报文发送的延迟、速率以及发送次数，并且起到了的打印结果的作用。

该发包器的作用与软件发包器tcpreply的作用相同。Tcpreply的作用是将pcap文件中的流量重新复现，一般用于对网络入侵检测系统的功能进行测试。

### Network Monitoring on NetFPGA

本工作主要借助NetFPGA与PC实现。作者列举了tcpdump/wireshark/ntop等基于PC的工具，emulex的产品Endace DAG Card可以在万兆以太网链路中实现100%的数据包捕获，但是价格是NetFPGA的三倍。NetFPGA与NetMagic相比，增加了CPU与各端口交换数据的功能，而NetMagic只可以通过以太网与CPU交换数据。

作者使用NetFPGA进行报文过滤已经对感兴趣的报文打时间戳，但是流选择与选择流后相关的报文会被软件进行处理（被过滤的报文仅会在硬件进行处理）。作者使用DDS产生时间戳，能够实现以秒为单位（而不是10ns），同时解决频率偏移的问题。

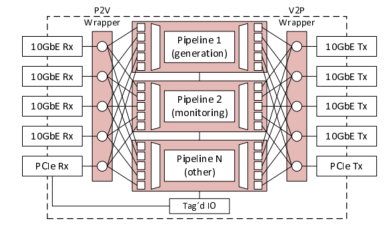
测量时延应当是测量一个报文的时延还是所有报文的平均时延？

### OSNT on NetFPGA

OSNT是在前述两个工作的基础上，实现的基于NetFPGA的开源网络测试仪。OSNT具有四种工作模式，分别为：

1. OSNT流量产生器：能够发送、收集报文，并在的所发送和收集的报文中打时间戳，能够精确测量交换机、路由器设备的时延与丢包率；
2. OSNT流量监测器：将FPGA收到的感兴趣的报文以pcap文件的形式存储到软件，并利用一些技术（packet batching, ring-receivers和pre-allocated memory）解决了PCIe的瓶颈问题，为用户提供带有高精度时间戳与丢包率可控的流量信息；
3. 混合OSNT系统：通过结合流量产生器与流量监测器的功能，混合OSNT系统能够对被测网络或设备进行全线速、单条流的特性分析。
4. 可扩展的OSNT系统：（不属于研究范围，略）

为了能够在OSNT中整合流量产生器与流量监测器等功能，作者设计了NetV架构，NetV在用户定义模块中实现了多条流水线，从而能够支持不同的功能在同一NetFPGA上实现（同时实现代码的重用）。每条流水线拥有各自的Tag ID, 这样能够保证各自索访问寄存器在不同流水线间的隔离。在这种情况下，各条流水线的处理速度不会受到影响，但是FPGA资源的消耗将会增大。



相比较FAST使用单流水线，但用metadata来区分某个模块是否应当处理当前数据来说，FAST使用的硬件资源开销更小，将新功能的开发转变为新模块的开发，而不是整条流水线的开发，个人认为各有利弊，但是对虚拟地址空间的分配应当更为复杂（我不能确定，想看看NetMagic的设计文档）。

作者在设计中用了一半的篇幅介绍如何将经过FPGA导入到host端，生成PCAP文件。对于Fast-Ant的初始设计，认为暂时无需考虑向软件导出PCAP文件的功能。作者使用NetFPGA达到的时间戳精度是6.25ns（也就是160MHz），使用Fast-Ant能达到的精度为10ns，时间精度已经可以接受。

主动测量： 测试吞吐率、时延与丢包率、openflow交换机/控制器压力测试

被动测量： PTP协议测试

## 参考文献

[1]. thttpd: Light-weight Web server on ARM.

[2]. Fast Project

## 问题描述

* + - 1. GAC模块需要将来自CPU的packet的next dmid修改为6，从而将其指定到PGM模块中进行处理，而将来自端口的packet的dmid依旧保留为5，从而直接在GOE中处理。

答： 方法一是修改GME、GAC两个模块代码，使经过GME的下一个dmid为7，经过GAC的下一个dmid为6。

方法二是重新分配硬件模块的mid号，从1到7重新编写。

* + - 1. 在PGM发送报文时，是否需要每次从RAM中获取报文，还是直接读出后直接每拍进行发送。

答：目前是连续从RAM中读出，因为寄存器资源难以满足1500B的要求

* + - 1. 在进行重复测量时，需要reset后进行，不行，因为在测量吞吐率时，需要定时改变测量参数并重新启动。所以当状态恢复到IDLE时就应当将所有状态清空（寄存器与计数器，但是RAM可以不清空，因为会继续发送同一种类型的报文）。但是在清空前需要将需要软件可读的信息写入拥有虚拟地址空间的寄存器中。
      2. 对于同一个reg类型变量，不能在超过一个的always语句中赋值。
      3. 在时延测量中需要在payload中打时间戳，如何打，打在什么位置？

对于IP报文，从第35字节开始，共占用四个字节

对于ARP报文，无法在报文体上打时间戳

对于UDP报文，需要在第43字节开始，共占用四个字节

* + - 1. 多数计数器与寄存器使用64位大小，但是目前总线的localbus\_data\_out与localbus\_data\_in为32位，使用fast\_reg\_read()与fast\_reg\_write()一次只能读出/写入32位数据。需要调用两次接口才能操作一个寄存器/计数器。
      2. 目前虚拟地址空间的地址为4GB，能够适配NetMagic，但是无法适配OpenBox。
      3. 在测量吞吐时，需要首先由软件读取相关计数器的值，之后才能重置，并迭代进入下一轮；这就涉及到状态机跳转到CLEAN\_S的时候不能再修改counter的值，然而这可能会影响硬件状态的复位。

答：方法一： CLEAN状态不对counter/reg进行复位，直接回到IDLE状态，但是下次进行测试前需要进行rst，用户是否可接受？

方法二：CLEAN状态下将用户所需的几个寄存器值写入另一组fast接口可读的寄存器，并清空原始寄存器的值，这样比较合适。

* + - 1. 首先仅实现IP报文的解析与序号的修改，能够调试通过后再添加UDP等。

0917上午需要从最精细粒度过一遍三种不同的业务应当如何实现，对PGM/SCM模块以及其它已有模块应当实现的功能以及需要添加的寄存器进行整理，并添加进设计文档中。吞吐的测量对PGM模块以及SCM模块的协同有了连续性的要求，当第一轮的测试结果检测到丢包时，需要软件读出相应数据，并触发重置硬件寄存器，然后启动第二轮测量。关键点在于由软件触发第二次测试，这应能够保证硬件的简单易修改特性。

而在时延测量时，需要在报文发送的过程中，周期性触发另一状态，从而允许插入时延测量probe报文的发送，并跳转回正常的发送状态。