Fast-Ant

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。

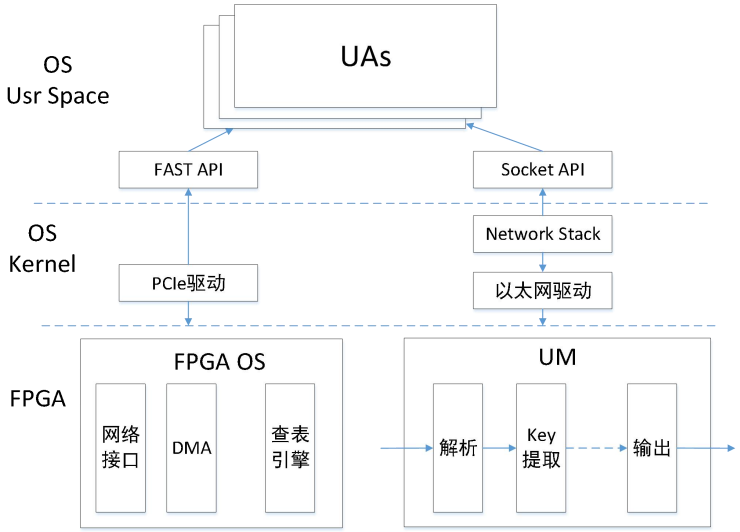
1 引言

1.1 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。

2 测试功能

2.1 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。

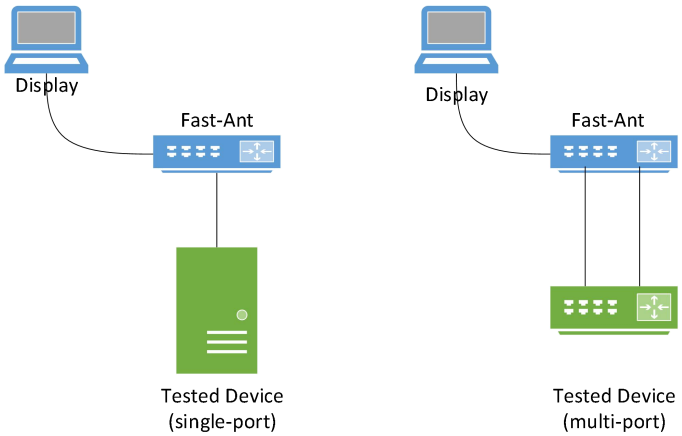


图 2 单/双端口测试模式

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】

2.2 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。

2.3 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。

3 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。

3.1 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)。

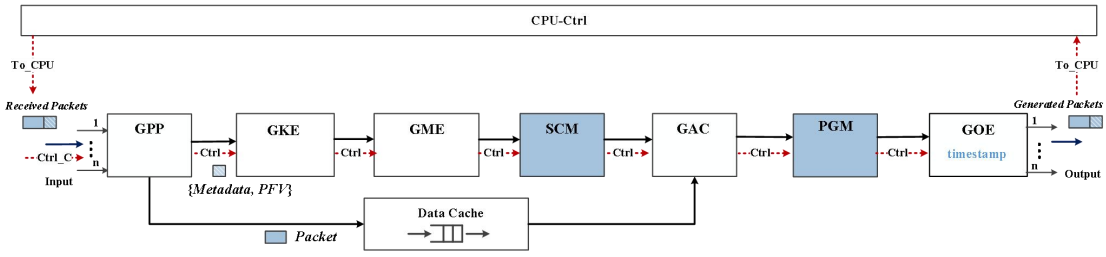


图 3 Fast-Ant 硬件流水线架构

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)。

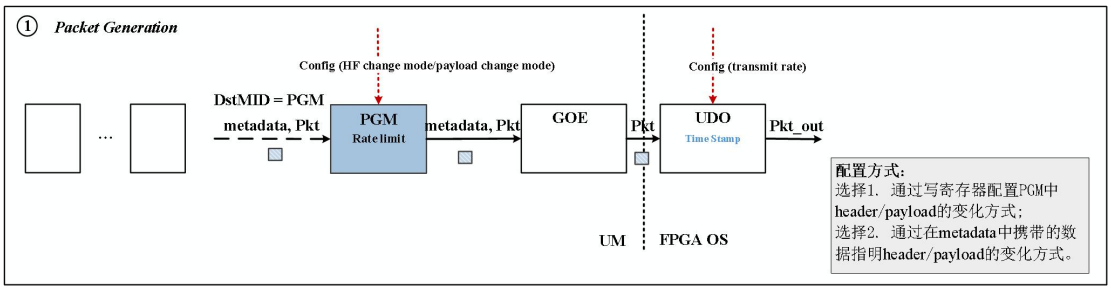


图 4　PGM 工作原理图

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。

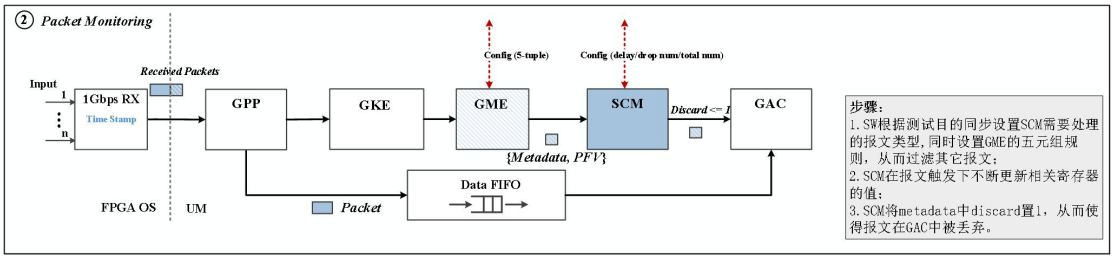


图 5　SCM 工作原理图

3.2　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。

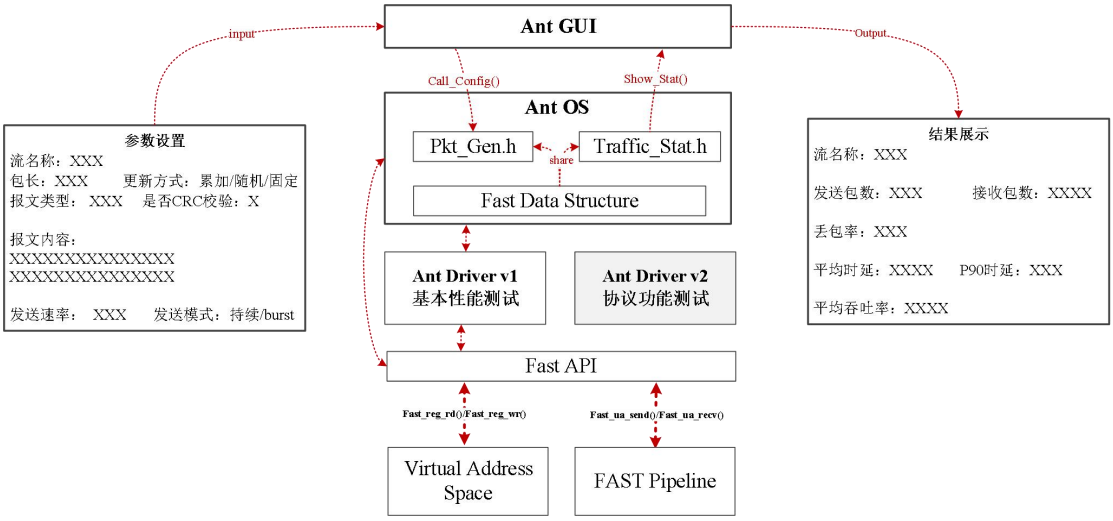


图 6 Fast-Ant 软件架构图

3.3 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。

3.3.1 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。

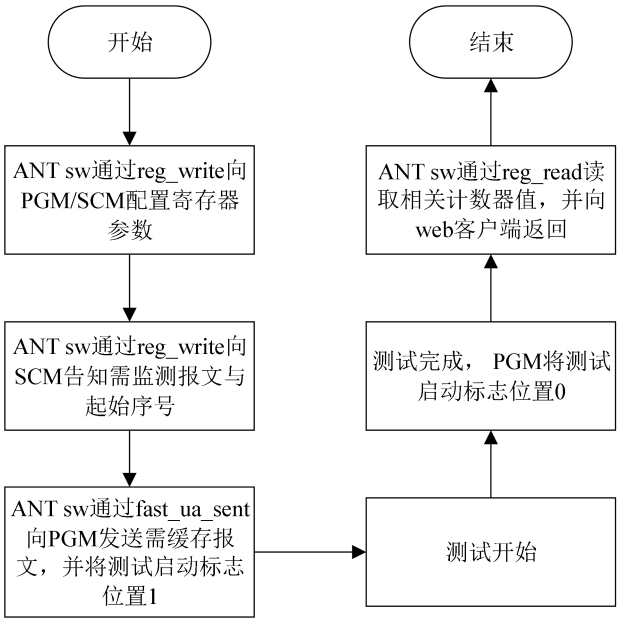


图 7 吞吐率测试流程图

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。

3.3.2 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。

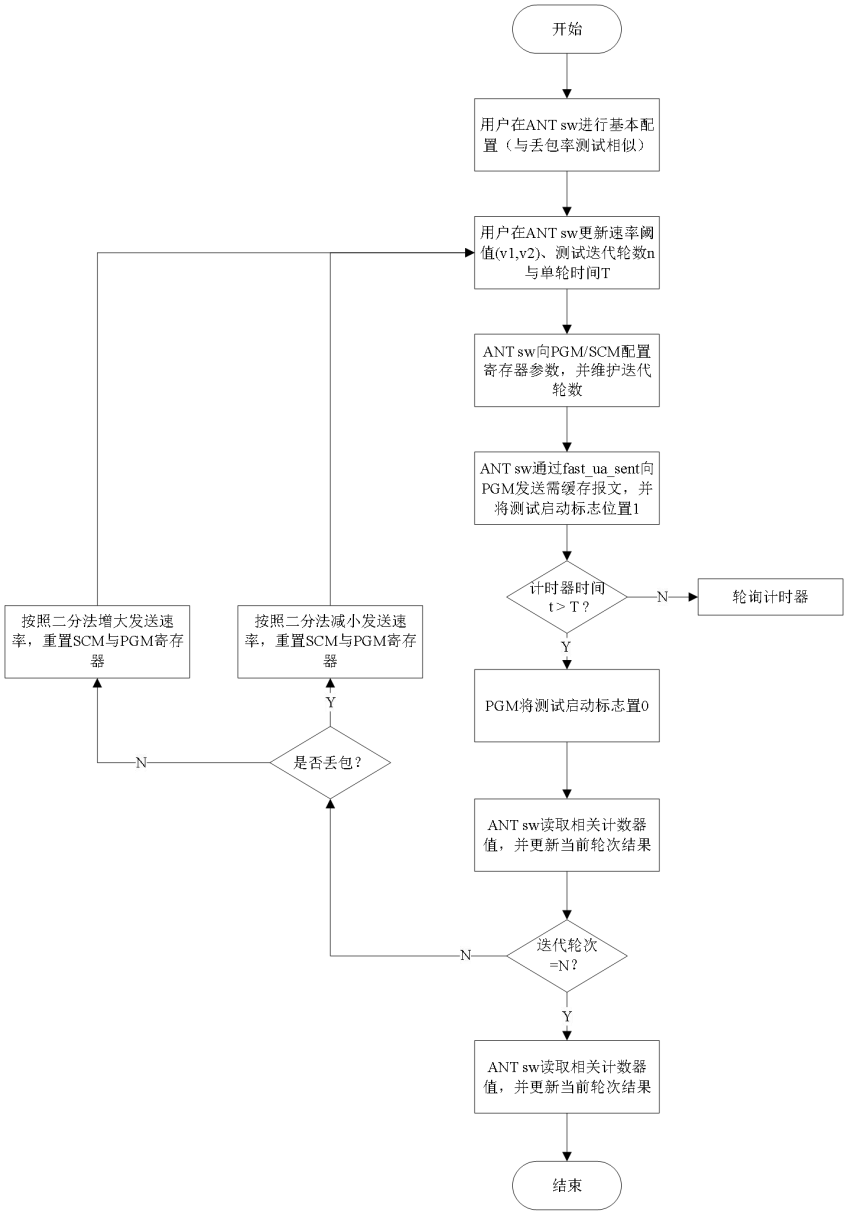


图 8 基于“二分法”吞吐率测试流程图

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。

3.3.3 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。

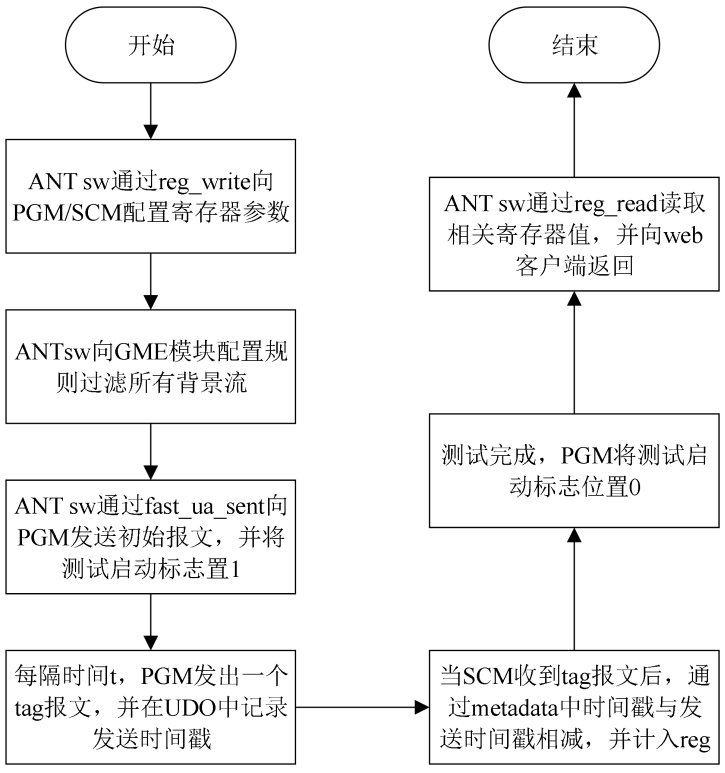


图 9 时延测试流程图

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。

4 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。

4.1 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。

4.2 ANT 中自定义 Metadata 格式

4.3 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。

4.1.1 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。

4.1.2 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。

4.1.3 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，SCM 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。
```

4.1.4 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】

4.4 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。

4.4.1 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。

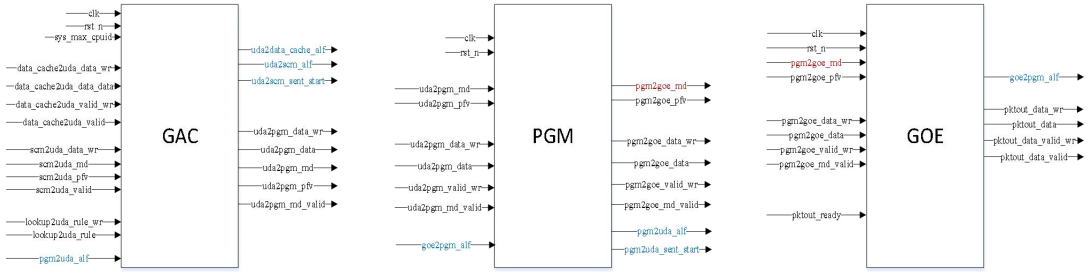


图 10 PGM 模块接口与连接关系图

4.4.2 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

4.4.3 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：

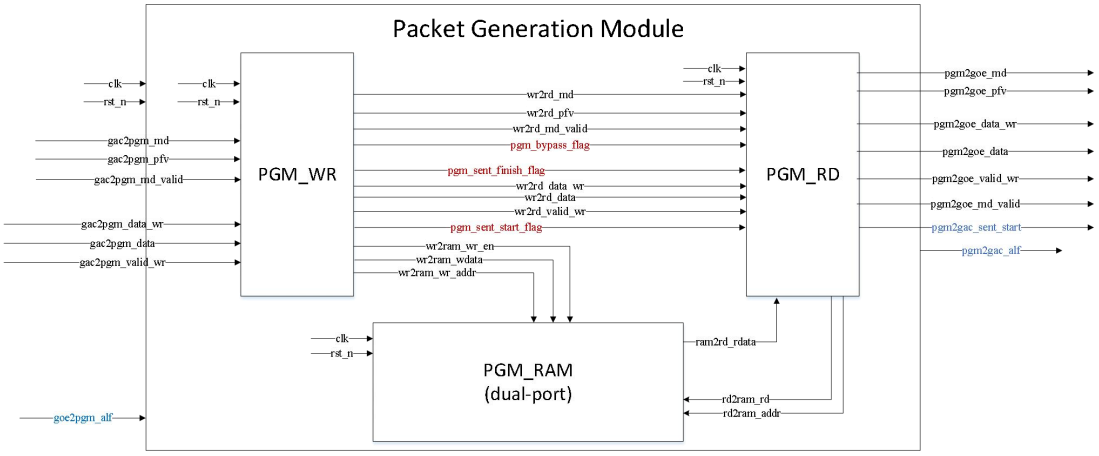


图 11 PGM 模块图

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。

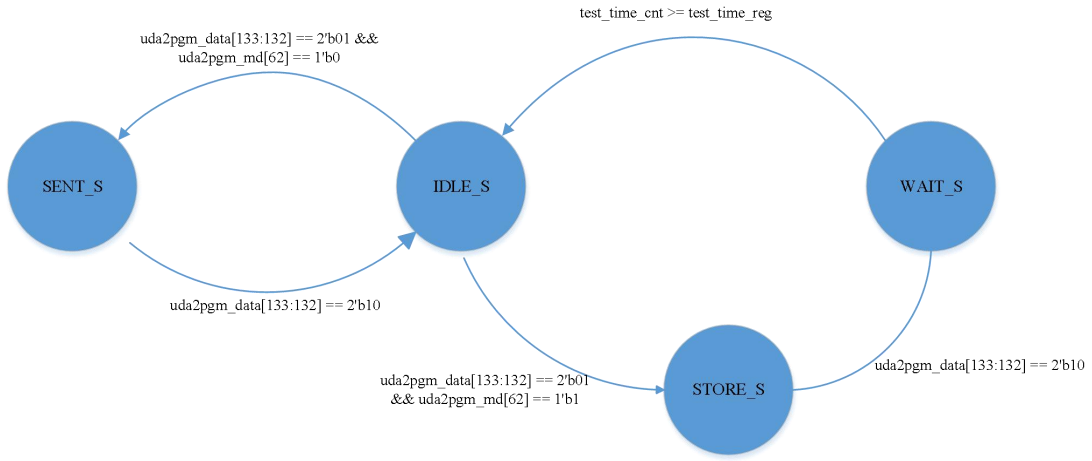


图 12 PGM_WR 状态机

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。

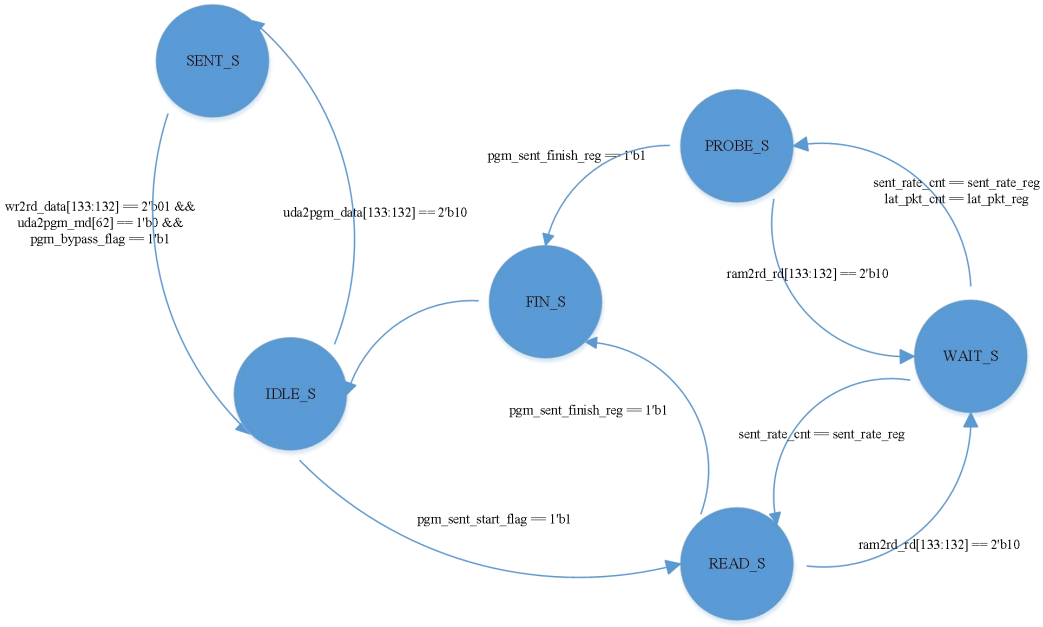


图 13 PGM_RD 状态机

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。

4.4.4 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

4.5 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。

4.5.1 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。

4.5.2 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。

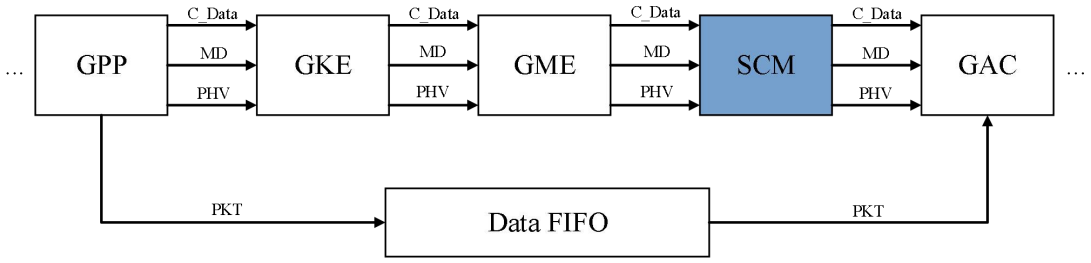


图 14 SCM 位于 FAST UM 五级流水线中的位置

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 PHV 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 15 为 SCM 接口信号关系图。

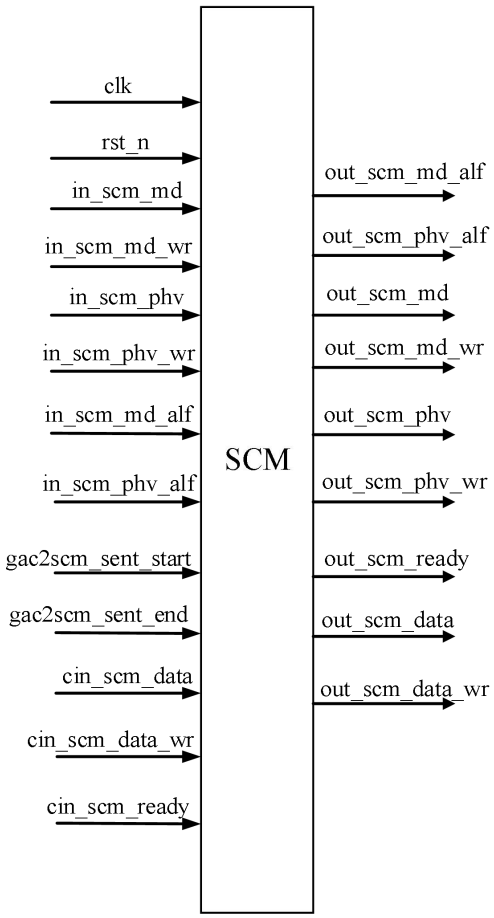


图 15 SCM 接口信号关系图

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
in_scm_md	input	256	GME 传递给 SCM 的 MD
in_scm_md_wr	input	1	GME 传递给 SCM 的 MD 的写使能信号
in_scm_phv	input	1024	GME 传递给 SCM 的 PHV
in_scm_phv_wr	input	1	GME 传递给 SCM 的 PHV 的写使能信号
in_scm_md_alf	input	1	GAC 能够接收 MD 的信号
in_scm_phv_alf	input	1	GAC 能够接收 PHV 的信号
gac2scm_sent_start	input	1	GAC 传递给 SCM 的测试开始信号
gac2scm_sent_end	input	1	GAC 传递给 SCM 的测试结束信号
cin_scm_data	input	134	GME 传递给 SCM 的控制通路数据
cin_scm_data_wr	input	1	GME 传递给 SCM 的控制通路数据的写使能信号
cin_scm_ready	input	1	GAC 准备接收 SCM 传递的控制通路数据的信号
out_scm_md_alf	output	1	SCM 能够接收 MD 的信号

out_scm_phv_alf	output	1	SCM 能够接收 PHV 的信号
out_scm_md	output	256	SCM 传递给 GAC 的 MD
out_scm_md_wr	output	1	SCM 传递给 GAC 的 MD 的写使能信号
out_scm_phv	output	1024	SCM 传递给 GAC 的 PHV
out_scm_phv_wr	output	1	SCM 传递给 GAC 的 PHV 的写使能信号
out_scm_ready	output	1	SCM 准备接收 GME 传递的控制通路数据的信号
out_scm_data	output	134	SCM 传递给 GAC 的控制通路数据
out_scm_data_wr	output	1	SCM 传递给 GAC 的控制通路数据的写使能信号

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 16 所示。

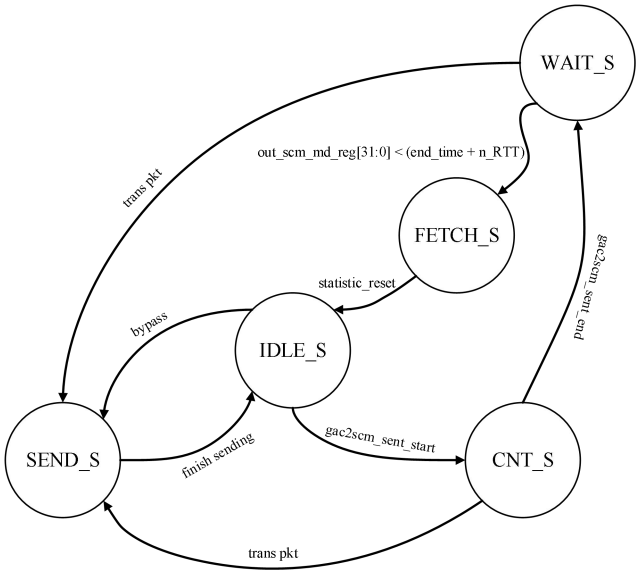


图 16 SCM 状态转换

a) IDLE_S：
SCM 初始状态，此时 SCM 内数据信号和使能信号置为 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。此时状态跳转到 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。

b) 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。
当转发结束时，跳转到 IDLE_S 状态。

c) 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。转发时跳转到 SEND_S 状态。
当接收到 gac2scm_sent_end 信号时，跳转至 WAIT_S 状态。

d) 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，即超时，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，并负责跳转到 SEND_S 状态进行转发。

e) FETCH_S：

软件获取相关寄存器的值，并在确定新的测试开始时，接收 statistic_reset 信号，相关寄存器清零，并跳转至 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	8	测试报文类型
0x7000 0001	statistic_reset	1	测试重置信号
0x7000 0002	n_RTT	32	超时间隔
0x7000 0008 0x7000 0009	scm_bit_num_cnt	64	记录总接收 bit 数量
0x7000 000A 0x7000 000B	scm_pkt_num_cnt	64	记录总接收报文数量
0x7000 000C 0x7000 000D	scm_time_cnt	64	记录总接收时间

5 相关研究

5.1 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。

5.2 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。

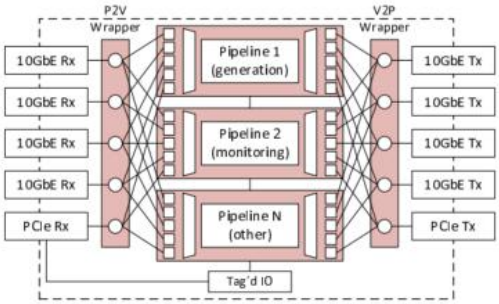
测量时延应当是测量一个报文的时延还是所有报文的平均时延？

5.3 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

6 参考文献

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

7 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。
2. 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
3. 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。
4. 对于同一个 reg 类型变量，不能在超过一个的 always 语句中赋值。
5. 在时延测量中需要在 payload 中打时间戳，如何打，打在什么位置？
   1) 对于 IP 报文，从第 35 字节开始，共占用四个字节
   2) 对于 ARP 报文，无法在报文体上打时间戳
   3) 对于 UDP 报文，需要在第 43 字节开始，共占用四个字节
6. 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。
7. 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。
8. 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。
9. 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。