设计及测试简单说明

XGE-PTPv2 原型验证平台

XGE-PTPv2 原型验证平台是作者使用 SystemC TLM 自行开发的电子系统级（ESL）原型验证平台，可以在此平台上进行软硬件协同设计和相关软件及算法的早期开发测试。

原型平台的顶层模块（testbench）的设计文件位于 xge-ptpv2/esl/sc 目录下，即 C++文件 testbench.h/.cpp。
顶层模块 testbench 实例化一个 ptp_instance 模块（loop back 测试），或者两个 ptp_instance 模块（PTPv2 Protocol 测试）。
ptp_instance 等效于一个包含 PTP 硬件引擎功能的 SoC，其中的子模块 m_initiator_top 和 m_target_top 通过抽象总线 m_bus 连接在一起，具体实现可以参考 xge-ptpv2/esl/sc 目录下面的 C++文件 ptp_instance.h/cpp。
m_initiator_top 模块包含子模块 m_controller，可以认为它是一个运行操作系统的 CPU 的简单抽象，具体内容可以参考 xge-ptpv2/esl/sc 目录下的 C++文件 controller.h/.cpp。

PTPv2 的应用软件，包括 loop back 测试和 Protocol 及算法测试，便是运行在 m_controller 上，在 controller 的构造函数中使用 SC_THREAD 宏声明的函数便可以认为是一个个在操作系统中运行的线程。
其中的 controller_thread 是主程序运行的线程，而 isr_thread 线程是中断服务处理例程。

读者如果有兴趣，可以使用某种 RISC 的指令集仿真器来代替 controller，这样会更加贴近真实情况。

软件设计说明

用于在 CPU 上运行的软件代码都放在 xge-ptpv2/esl/sw 目录下，如果读者只对 PTPv2 软件设计感兴趣或者不太熟悉 SystemC/Verilog 等硬件描述语言，阅读 sw 目录下的 C++源文件再加上 controller.h/.cpp 基本就可以了。

协议软件及 clock servo 算法的设计参考了以下开源项目：
https://sourceforge.net/projects/ptpd/

XGE-PTPv2 项目软件对上面的开源软件做了大刀阔斧的修改，比如使用 C++语言对其进行了面向对象化改造，删除大量无关代码，调整软件以适应 XGE-PTPv2 的硬件接口，等等。

编译及运行仿真

设计和测试均基于 Linux 操作系统，需要安装 Verilator，SystemC，以及 CMake。CMake 可以直接使用 Linux 自带的软件包管理工具安装，而 Verilator，SystemC 的具体安装方法，可以参考其官方网站或者本项目开发者的公众号文章。

运行仿真，请执行以下步骤：
cd /path/to/solution
rm -rf build && mkdir build && cd build
cmake ..
make
./ptpv2_tlm

如果需要调试程序，设置断点，单步执行等操作，则可以安装 Visual Studio Code 并打开 ESL 项目所在目录 xge-ptpv2/esl。

软件测试说明

图 1. 是启动./ptpv2_tlm 程序后的界面

```
[[redacted]@CentOS8-[redacted] build]$ ./ptpv2_tlm

        SystemC 2.3.3-Accellera --- Jan 13 2023 21:07:32
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED

Select the Application
0 : Loop Back Test for Local Device
1 : PTPv2 Protocol Test
Please Input the Selection: 1

                Clock ID: 1
=======================================================
            ####  PTPv2 Protocol Test Start!  ####


                Clock ID: 2
=======================================================
            ####  PTPv2 Protocol Test Start!  ####

    Clock ID: 2   (ptpd info)         Clock ID: 1   (ptpd info)     08:00:00.000000  (____)
    Clock ID: 2   (ptpd info)     08:00:00.000000  (____)
    Clock ID: 1   (ptpd info)     08:00:00.000000  (____)  Starting ptpd2 daemon with parameters:      ptpv2d -e -z
    Clock ID: 2   (ptpd debug1)   08:00:00.000000  (____)  Starting ptpd2 daemon with parameters:      ptpv2d -e -z
    Clock ID: 1   (ptpd debug1)   08:00:00.000000  (____)  allocated 1608 bytes for protocol engine data
```

图 1

选择 0 后输入回车是进行 Loop Back 测试，选择 1 则是进行 PTPv2 Protocol 测试。

Loop back Test 比较简单，就是发送报文并回环得到接收报文，在这个过程中测试控制寄存器的设置和读取是否正常，读取状态寄存器查看是否正常反映硬件状态，中断是否能正常产生等。在此，不做过多描述。

PTPv2 Protocol Test 用以测试协议软件及相应的 Clock Servo 算法能否在 XGE-PTPv2 硬件上正确执行。
入口函数：ptpd::exec()
执行函数：protocol::protocolExec(RunTimeOpts *rtOpts, PtpClock *ptpClock)

本质上，protocolExec 函数是一个无限循环的状态机，驱动状态机的事件可以归结到各种中断信号，其中比较重要的是定时中断信号。设置定时中断产生的事件间隔是 7.8125ms，恰好对应一秒钟 128 个中断，可以方便地按照 PTPv2 标准驱动各种报文的发送。当然，读者在移植程序时也可以用操作系统自带的定时器，只是没有那么精确而已。

文件 xge-ptpv2/esl/sw/dep/constants_dep.h 中的定义的宏常量 PTP_SYNCE, 确定了 Protocol 测试是处于 SyncE 状态还是非 SyncE 状态。如果 PTP_SYNCE 为 1，则是 SYNCE 状态，为 0 则是非 SyncE 状态。

仿真中的所谓 SyncE 状态就是两个 ptp_instance 中的 rtc_clk/tx_clk 所用的时钟频率相同，对应的 INITIAL_TICK 和 LP_INITIAL_TICK 的值也相同。
非 SyncE 状态，理论上应该是时钟频率不一致才是，不过由于 Verilator 仿真的时间精度大约在 100ps 左右，100PPM 的频率差在仿真中看来基本就是一样的，因此我们通过改变 LP_INITIAL_TICK 值，使其和 INITIAL_TICK 不一致来达到等效频差。

图 2. 是将 PTP_SYNCE 设置为 1，编译后进行 SyncE 状态下的 PTPv2 Protocol Test 的结果。

```
    Clock ID: 1   (ptpd debug3)   12:46:17.150006  (slv)  Message Sync :
    Clock ID: 1   (ptpd debug3)   12:46:17.150006  (slv)  Integer 48 :
    Clock ID: 1   (ptpd debug3)   12:46:17.150006  (slv)  LSB : 1687236377
    Clock ID: 1   (ptpd debug3)   12:46:17.150006  (slv)  MSB : 0
    Clock ID: 1   (ptpd debug3)   12:46:17.150006  (slv)  nanoseconds 150001305
    Clock ID: 1   (ptpd debug3)   12:46:17.150006  (slv)
    Clock ID: 1   (ptpd debug3)   12:46:17.150006  (slv)  seconds : 0
    Clock ID: 1   (ptpd debug3)   12:46:17.150007  (slv)  nanoseconds 0
    Clock ID: 1   (ptpd debug3)   12:46:17.150007  (slv)  ==> updateOffset
    Clock ID: 1   (ptpd debug3)   12:46:17.150007  (slv)  offset filter 0
    Clock ID: 1   (ptpd debug3)   12:46:17.150007  (slv)  ==> updateClock
    Clock ID: 1   (ptpd debug1)   12:46:17.150007  (slv)       Observed_drift with AI component: 0
    Clock ID: 1   (ptpd debug1)   12:46:17.150007  (slv)       After PI: Adj: 0   Drift: 0   OFM 0
    Clock ID: 1   (ptpd debug2)   12:46:17.150008  (slv)       adjTickRate2: call adjTickRate to 0.000000 PPM
    Clock ID: 1   (ptpd debug3)   12:46:17.150008  (slv)
--Offset Correction--
    Clock ID: 1   (ptpd debug3)   12:46:17.150008  (slv)  Raw offset from master:           0s         140ns
    Clock ID: 1   (ptpd debug3)   12:46:17.150008  (slv)
--Offset and Delay filtered--
    Clock ID: 1   (ptpd debug3)   12:46:17.150008  (slv)  one-way delay averaged (P2P):           0s         140ns
    Clock ID: 1   (ptpd debug3)   12:46:17.150008  (slv)  offset from master:                0s           0ns
    Clock ID: 1   (ptpd debug3)   12:46:17.150009  (slv)  observed drift:             0,     0.000000 PPM
    Clock ID: 1   (ptpd debug3)   12:46:17.150009  (slv)  activity
 Simulation: 500 milliseconds elapsed!
 PTP Slave, Clock ID 1, reach synchronized convergence state,  Stop!

Info: /OSCI/SystemC: Simulation stopped by user.
```

图 2

在 SyncE 状态下，仿真结果显示，Protocol 及 Clock Servo 算法运行不到 1 秒就可以让 PTP Slave 和 PTP Master 到达同步状态，并且同步精度非常高，offsetFromMaster 在 0 ns 左右，观察到的频率差在 0.0 PPM。
在作者的电脑上，整个仿真从开始到结束，大致运行了二十几分钟。

图 3. 是将 PTP_SYNCE 设置为 0，编译后进行非 SyncE 状态下的 PTPv2 Protocol Test 的结果，在此状态下设置的 Master-Slave 频率差为 10.0 PPM。

```
    Clock ID: 1   (ptpd debug3)   14:05:11.840005  (slv)  Sync message received :
    Clock ID: 1   (ptpd debug3)   14:05:11.840005  (slv)  Message Sync :
    Clock ID: 1   (ptpd debug3)   14:05:11.840005  (slv)  Integer 48 :
    Clock ID: 1   (ptpd debug3)   14:05:11.840006  (slv)  LSB : 1687241111
    Clock ID: 1   (ptpd debug3)   14:05:11.840006  (slv)  MSB : 0
    Clock ID: 1   (ptpd debug3)   14:05:11.840006  (slv)  nanoseconds 840001216
    Clock ID: 1   (ptpd debug3)   14:05:11.840006  (slv)
    Clock ID: 1   (ptpd debug3)   14:05:11.840006  (slv)  seconds : 0
    Clock ID: 1   (ptpd debug3)   14:05:11.840006  (slv)  nanoseconds 0
    Clock ID: 1   (ptpd debug3)   14:05:11.840006  (slv)  ==> updateOffset
    Clock ID: 1   (ptpd debug3)   14:05:11.840007  (slv)  offset filter -24
    Clock ID: 1   (ptpd debug3)   14:05:11.840007  (slv)  ==> updateClock
    Clock ID: 1   (ptpd debug1)   14:05:11.840007  (slv)       Observed_drift with AI component: -4311
    Clock ID: 1   (ptpd debug1)   14:05:11.840007  (slv)       After PI: Adj: -4319   Drift: -4311   OFM -24
    Clock ID: 1   (ptpd debug2)   14:05:11.840007  (slv)       adjTickRate2: call adjTickRate to 10.055955 PPM
    Clock ID: 1   (ptpd debug3)   14:05:11.840007  (slv)
--Offset Correction--
    Clock ID: 1   (ptpd debug3)   14:05:11.840008  (slv)  Raw offset from master:           0s         -25ns
    Clock ID: 1   (ptpd debug3)   14:05:11.840008  (slv)
--Offset and Delay filtered--
    Clock ID: 1   (ptpd debug3)   14:05:11.840008  (slv)  one-way delay averaged (P2P):           0s         120ns
    Clock ID: 1   (ptpd debug3)   14:05:11.840008  (slv)  offset from master:               0s          -24ns
    Clock ID: 1   (ptpd debug3)   14:05:11.840008  (slv)  observed drift:          -4311,     -10.037329 PPM
    Clock ID: 1   (ptpd debug3)   14:05:11.840008  (slv)  activity
 PTP Slave, Clock ID 1, reach synchronized convergence state,  Stop!

Info: /OSCI/SystemC: Simulation stopped by user.
```

图 3

在非 SyncE 状态下，仿真结果显示，Protocol 及 Clock Servo 算法运行需要 15 秒以上才可以让 PTP Slave 和 PTP Master 到达同步状态，offsetFromMaster 在 20 ns 左右，观察到的 Slave-Master 频率差在-10.03 PPM 左右，不管收敛速度还是同步精度都不如 SyncE 状态。
在作者的电脑上，整个非 SyncE 状态的 PTPv2 Protocol 及算法仿真从开始到结束，运行时间超过了六个小时。时间相当长，跑这种仿真需要有耐心。