光纤通信测试例程

1 实验简介

Pango 的 Logos2 系列 FPGA 集成了串行高速收发器 HSSTLP，可以实现高速串行数据通信。在 AXP100 开发板上，FPGA 的 HSSTLP 的 2 个收发器通道已经连接到 2 路 SFP 光模块接口，用户只需要另外购买 SPF 的光模块就可以实现光纤的数据传输。本实验将介绍通过光纤连接实现光模块之间的数据收发测试。

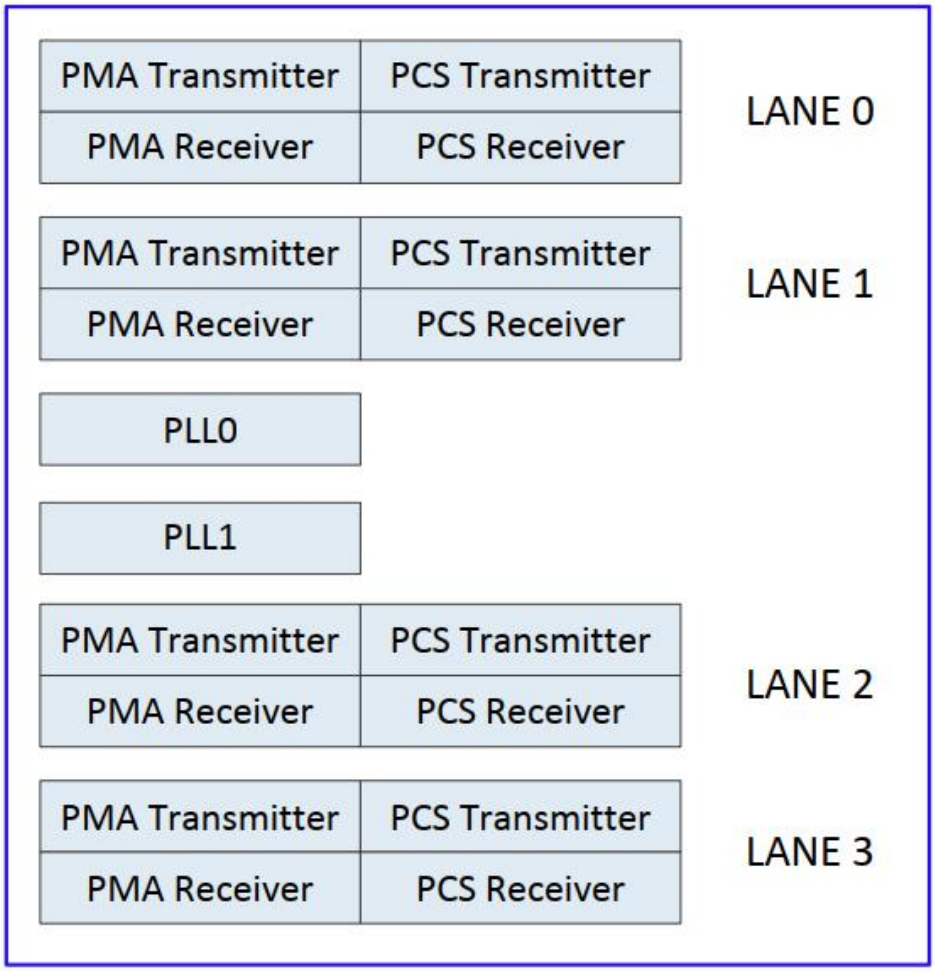
2 实验原理

2.1 HSSTLP 介绍

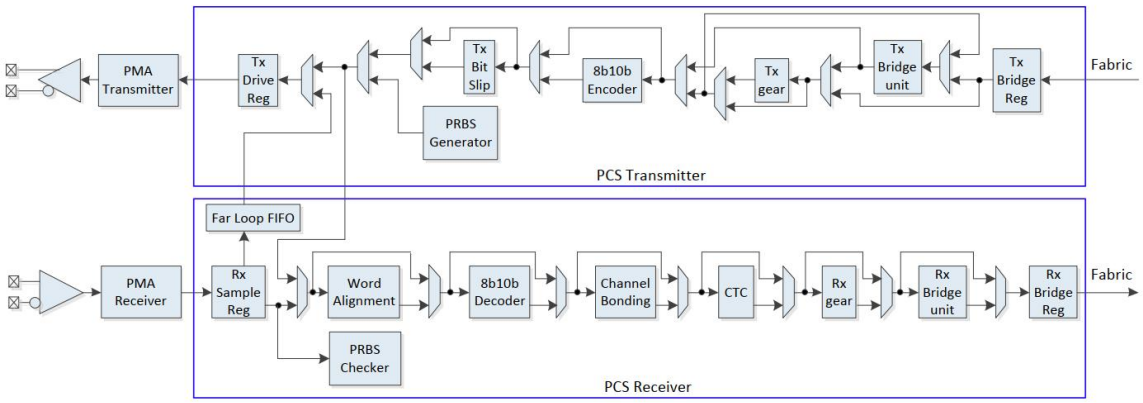
AXP100 开发板 FPGA 芯片(PG2L100HFBG676)自带 8 路高速串行高速收发器通道，每通道的收发速度高达 6.6 Gb/s。收发器支持不同的串行传输接口或协议，支持 PCI Express GEN1, PCI Express GEN2,XAUI,千兆以太网,CPRI,SRIO 等协议。

PG2L100H 包含 2 个 HSSTLP，共可支持 8 个全双工收发 LANE。

每个 HSSTLP 由两个 PLL 和四个收发 LANE 组成，其中每个 LANE 又包括四个组件： PCS Transmitter，PMA Transmitter， PCS Receiver， PMA Receiver。 PCS Transmitter 和 PMA Transmitter 组成发送通路，PCS Receiver 和 PMA Receiver 组成接收通路。 HSSTLP 的结构示意图：



HSSTLP 中的四个收发 LANE 共享 PLL0 和 PLL1，每个发送或者接收 LANE 都可以独立选择 PLL0 或者 PLL1， PLL 工作频率范围为 1.6GHz~6.6GHz。 PLL0 和 PLL1 都各自对应有一对外部差分参考时钟输入，每个 PLL 还可以选择来自另一个 PLL 的参考时钟或者来自 Fabric 的时钟作为参考时钟输入（ Fabric 逻辑时钟做参考时钟，仅用于内部测试）； PLL 输出频率支持动态再分频，以适应 0.6Gbps~6.6Gbps 的 Data Rate 范围。



PCS Transmitter 和 Receiver 结构示意图

每个 PCS Transmitter 主要包含以下模块：

Tx Bridge Reg 模块： 用于从 Fabric 到 PCS Transmitter 的数据桥接

Tx Bridge unit 模块： 用于 PCS Transmitter 内部时钟域和 Fabric 时钟域相位补偿

8b10b Encoder 模块：完成符合 IEEE 802.3 1000BASE-X specification 的 8b10b 编码

Tx gear 模块： 完成 64b66b/64b67b 数据适配功能

Tx Bit Slip 模块: 主要功能是根据配置对发送数据实现按位 Slip

PRBS Generator 模块: 产生 PRBS 测试序列

Tx Drive Reg 模块: 用于从 PCS Transmitter 到 PMA Transmitter 数据桥接

每个 PCS Receiver 主要包含以下功能模块：

Rx Sample Reg 模块： 用于从 PMA Receiver 到 PCS Receiver 的数据桥接

PRBS Checker 模块：用于 PRBS 序列的校验

Word Alignment 模块：支持灵活的 Word Alignment 功能

8b10b Decoder 模块：完成符合 IEEE 802.3 1000BASE-X Specification 的 8b10b 解码

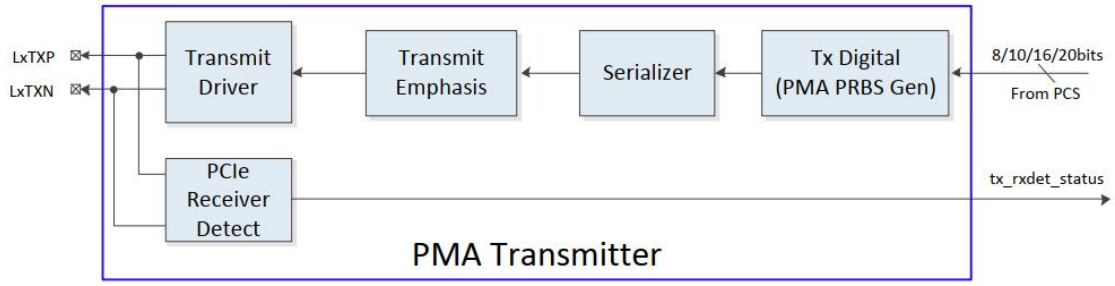
Rx gear 模块： 完成 64b66b/64b67b 数据适配功能

Channel Bonding 模块：用于通道对齐

CTC 模块：用于补偿发送时钟和接收时钟的微小频差

Rx Bridge unit 模块：用于 PCS Receiver 内部时钟域和 Fabric 时钟域相位补偿

Rx Bridge Reg 模块：用于从 PCS Receiver 到 Fabric 的数据桥接。



PMA Transmitter 功能示意图

每个 PMA Transmitter 主要包含以下功能模块：

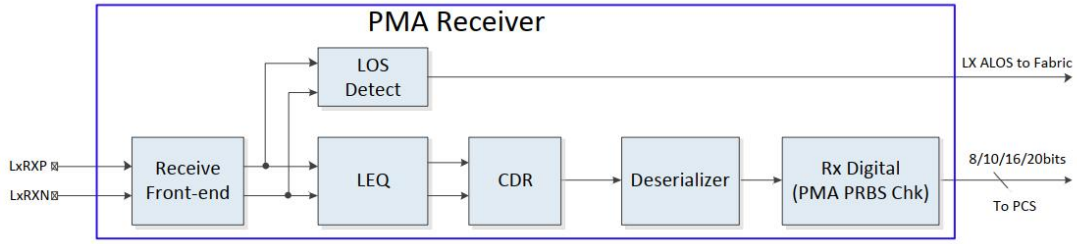
Tx Digital 模块： 完成 PCS Transmitter 到 PMA Transmitter 的数据桥接， 以及 PMA PRBSGenerator。

Serializer 模块： 完成并行数据到串行数据的转换功能

Transmit Emphasis 模块： 支持可调节的去加重功能

Transmit Driver 模块： 支持可调节的发送驱动

PCI Express Receiver Detect 模块：支持基于 PCI Express 的 Receiver Detection 功能



PMA Receiver 功能示意图

每个 PMA Receiver 主要包含以下功能模块：

Receive Front-end 模块：支持多种接收 Termination 模式

LEQ 模块：支持 Linear Equalizer 功能

CDR 模块：数据和时钟恢复功能

LOS Detect 模块：用于检测接收信号是否有效功能

Deserializer 模块： 完成串行数据到 8 bits, 10bits, 16bits 以及 20bits 并行数据的转换功能

Rx Digital 模块： 完成 PMA Receiver 到 PCS Receiver 的数据桥接，以及 PMA PRBS Checker

HSSTLP 的参考时钟

HSSTLP 每个模块有两个差分参考时钟输入管脚(HSSTREFCLK0P/N_QRX 和 HSSTREFCLK1P/N_QRX）作为 HSSTLP 模块的参考时钟源，用户可以自行选择。在核心板上，有 2 路 125Mhz 的 HSSTLP 的参考时钟连接到 Q3 与 Q6HSSTLP 时钟输入管脚上，作为 HSSTLP 的参考时钟。进入到 PLL0 和 PLL1 中后产生 TX 和 RX 电路中所需的时钟频率。TX 和 RX 收发器速度相同的话，TX 电路和 RX 电路可以使用同一个 PLL 产生的时钟，如果 TX 和 RX 收发器速度不相同的话，需要使用不同的 PLL 时钟产生的时钟。

HSSTLP 接口说明

接口信号是 FPGA 的用户数据与 HSSTLP 的接口连接信号，该接口信号的名称和说明如下表所示：

表 1：HSSTLP LANE 时钟相关端口

端口命名	输入/输出	描述
P_RCLK2FABRIC	输出	送到 Fabric 的接收时钟
P_TCLK2FABRIC	输出	送到 Fabric 的发送时钟
P_RX_CLK_FR_CORE	输入	来自 Fabric 的接收时钟
P_RCLK2_FR_CORE	输入	来自 Fabric 的接收时钟，由 P_REFCK2CORE 通过 PLL 倍频生成，是 P_RX_CLK_FR_CORE 的 2 倍频
P_TX_CLK_FR_CORE	输入	来自 Fabric 的发送时钟
P_TCLK2_FR_CORE	输入	来自 Fabric 的接收时钟，由 P_REFCK2CORE 通过 PLL 倍频生成，是 P_TX_CLK_FR_CORE 的 2 倍频
P_CA_ALIGN_RX	输出	接收 LANE CLK Aligner 动态状态输出，0 > 1 的跳变状态表示 aligner 成功，为异步信号
P_CA_ALIGN_TX	输出	发送 LANE CLK Aligner 动态状态输出，0 > 1 的跳变状态表示 aligner 成功，为异步信号
P_CIM_CLK_ALIGNER_RX[7:0]	输入	接收侧的 CLK Aligner delay step 选择，为异步信号
P_CIM_CLK_ALIGNER_TX[7:0]	输入	发送侧的 CLK Aligner delay step 选择，为异步信号
P_CIM_DYN_DLY_SEL_RX	输入	接收 LANE 的 CLK Aligner 功能使能，为异步信号， 1:使能；0:不使能
P_CIM_DYN_DLY_SEL_TX	输入	发送 LANE 的 CLK Aligner 功能使能，为异步信号， 1:使能；0:不使能
P_CIM_START_ALIGN_RX	输入	用于产生接收 LANE CLK Aligner 脉冲的输入源，为异步信号
P_CIM_START_ALIGN_TX	输入	用于产生发送 LANE CLK Aligner 脉冲的输入源，为异步信号

表 2：HSSTLP LANE 复位相关端口

端口命名	输入/输出	描述
P_PCS_TX_RST	输入	对 PCS Transmitter 进行复位，1:复位；0:不复位
P_PCS_RX_RST	输入	对 PCS Receiver 进行复位，1:复位；0:不复位
P_LANE_PD	输入	Lane power down，包括:RX LANE 和 TX LANE； 1: power down；0:not power down
P_LANE_RST	输入	Lane 复位，包括:RX LANE 和 TX LANE； 1:复位；0:不复位
P_RX_LANE_PD	输入	RX LANE power down，包括:RX PMA 和 RX PCS； 1: power down；0:not power down
P_RX_PMA_RST	输入	对 PMA Receiver 进行复位，为异步信号， 1:复位；0:不复位
P_TX_PMA_RST	输入	对 PMA Transmitter 进行复位，为异步信号， 1:复位；0:不复位
P_TX_LANE_PD_CLKPATH	输入	Tx Lane power down，1:复位；0:不复位
P_TX_LANE_PD_PISO	输入	TX 并串转换模块 power down， 1: power down；0:not power down
P_TX_LANE_PD_DRIVER	输入	TX driver power down，1: power down；0:not power down
P_PCS_CB_RST	输入	复位 Channel bonding 之后的的模块，1:复位；0:不复位； 内部测试信号，接固定值 0。
P_CTLE_ADP_RST	输入	PMA 接收端的线性均衡器复位，1:复位；0:不复位； 内部测试信号，接固定值 0。

表 3：HSSTLP LANE 和 Fabric 之间的发送端口

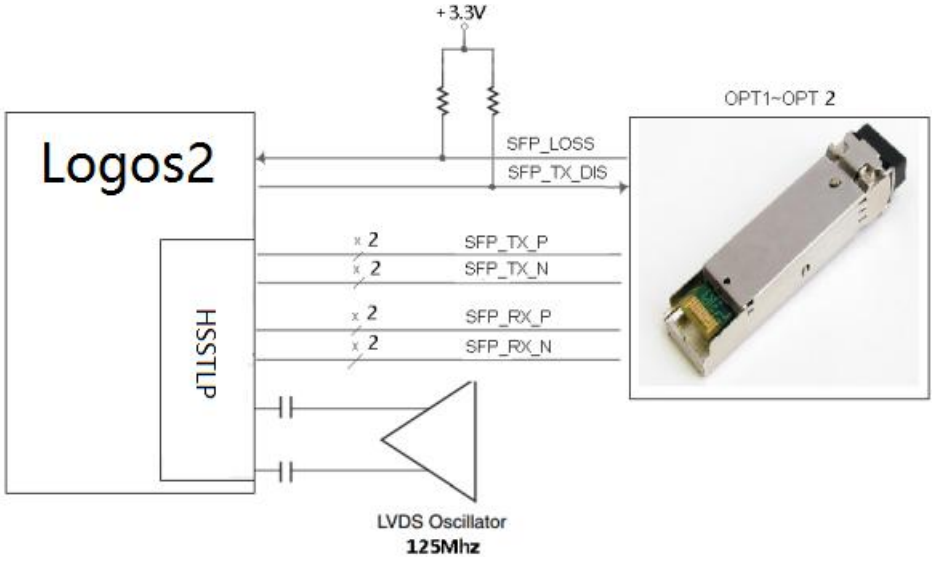
端口命名	输入/输出	描述
P_TDATA[45:0]	输入	发送数据
P_TX_LS_DATA	输入	发送的低频信号
P_TX_BEACON_EN	输入	TX beacon enable 信号, 1:使能；0:不使能
P_TX_DEEMP[1:0]	输入	Transmitter de-emphasis 控制
P_TX_SWING	输入	Transmitter 输出摆幅值进行半摆幅控制 1'b0: 满摆幅(默认值)；1'b1: 半摆幅
P_TX_MARGIN[2:0]	输入	Transmitter 输出摆幅的 DAC 来源选择。默认值 3'b000 3'b000:摆幅来源寄存器 PMA_CH_REG_TX_AMP_DAC0; 3'b001:摆幅来源寄存器 PMA_CH_REG_TX_AMP_DAC1; 3'b010:摆幅来源寄存器 PMA_CH_REG_TX_AMP_DAC2; 3'b011:摆幅来源寄存器 PMA_CH_REG_TX_AMP_DAC3; 其他值：保留
P_TX_RXDET_REQ	输入	Receiver Detection 请求信号
P_TX_RXDET_STATUS	输出	Receiver Detection 结果，为异步信号，1:检测到 Receiver
P_TX_RATE[2:0]	输入	TX 线速率控制信号 2'b00: 线速率是 PLL 时钟频率的 1/4 倍; 2'b01: 线速率和 PLL 时钟频率的 1/2; 2'b10: 线速率和 PLL 时钟频率相等; 2'b11: 线速率是 PLL 时钟频率的 2 倍。 bit[2]: 保留，接固定值 0
P_TX_BUSWIDTH[2:0]	输入	TX PCS 到 TX PMA 的数据位宽选择 3'bX00: 8bit; 3'bX01:10bit; 3'bX10:16bit; 3'bX11:20bit; bit[2]: 保留，接固定值 0
P_TX_SDN	输出	差分输出数据负端，HSSTLP 专用管脚
P_TX_SDP	输出	差分输出数据正端，HSSTLP 专用管脚

表 4：HSSTLP LANE 和 Fabric 之间的接收端口

端口命名	输入/输出	描述
P_RDATA[46:0]	输出	接收数据
P_RX_SIGDET_STATUS	输出	端口有效信号检测，为异步信号： 0:从端口 P_RX_SDP/P_RX_SDN 没有检测到有效信号 1:从端口 P_RX_SDP/P_RX_SDN 检测到了有效信号
P_RX_SATA_COMINIT	输出	SATA COMINIT 状态，1:检测到；0:未检测到
P_RX_SATA_COMWAKE	输出	SATA COMWAKE 状态，1:检测到；0:未检测到
P_RX_LS_DATA	输出	输出到 Fabric 的低频信号
P_RX_READY	输出	CDR 已成功锁定标志信号，为异步信号， 1:锁定;0:未锁定
P_TEST_STATUS[19:0]	输出	RX 输出测试状态寄存器，内部测试信号
P_PCS_WORD_ALIGN_EN	输入	当配置为 RX CLK Slip 端口控制方式有效后控制有效，则作为 RX CLK Slip 控制信号，为异步信号， 0->1 的一次上升沿，PMA RX 的 Deserializer 模块的数据 slip 一个比特
		当配置为外部状态机时，作为 Word Alignment 使能信号，为异步信号，1:使能；0:不使能
		当配置为外部状态机时，作为 Word Alignment 使能信号，为异步信号，1:使能；0:不使能
P_PCS_LSM_SYNCED	输出	Word Alignment 成功，状态机锁定标志，为异步信号， 1:Word Alignment 成功;0:Word Alignment 未成功；
P_PCS_MCB_EXT_EN	输入	外置状态机模式下 Channel bonding 使能，为异步信号， 1:使能；0:不使能
P_PCS_RX_MCB_STATUS	输出	Channel Bonding 控制状态机指示信号，为异步信号， 1:处于绑定状态;0:未绑定状态
P_RXGEAR_SLIP	输入	slip indication to rx gear box with 64b66b/67b decoder mode ，边沿触发，检测到上升沿或者下降沿都会延时 1bit
P_RX_POLARITY_INVERT	输入	Rx Sample Reg 的极性反转使能，为异步信号， 1:极性反转;0:极性不反转
P_CEB_ADETECT_EN[3:0]	输入	测试信号，正常模式下需在外部置 4'b1111
P_RX_RATE[2:0]	输入	RX 线速率控制信号 2'b00: 线速率是 PLL 时钟频率的 1/4; 2'b01: 线速率是 PLL 时钟频率的 1/2; 2'b10: 线速率和 PLL 时钟频率相等; 2'b11: 线速率是 PLL 时钟频率的 2 倍. bit[2]：保留，接固定值 0
P_RX_BUSWIDTH[2:0]	输入	Rx PMA 到 RX PCS 的数据位宽选择 3'bX00: 8bit; 3'bX01:10bit; 3'bX10:16bit; 3'bX11:20bit; bit[2]：保留，接固定值 0
P_RX_HIGHZ	输入	Rx 输入高阻控制信号，0:不高阻；1:高阻
P_RX_SDN	输入	差分输入数据负端，HSSTLP 专用管脚
P_RX_SDP	输入	差分输入数据正端，HSSTLP 专用管脚

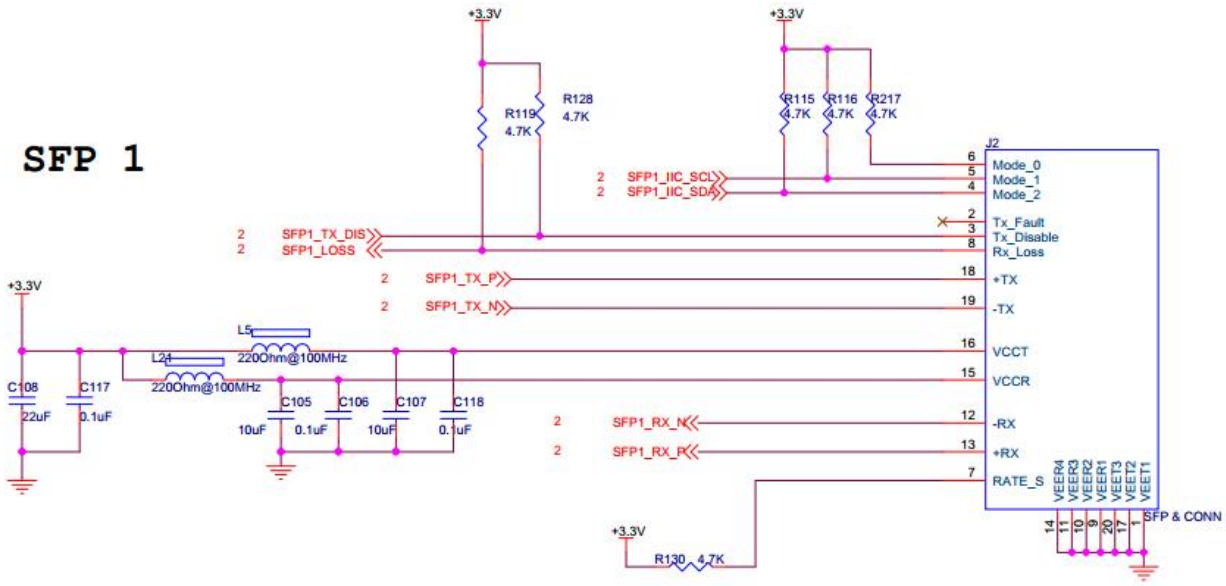
2.2 硬件介绍

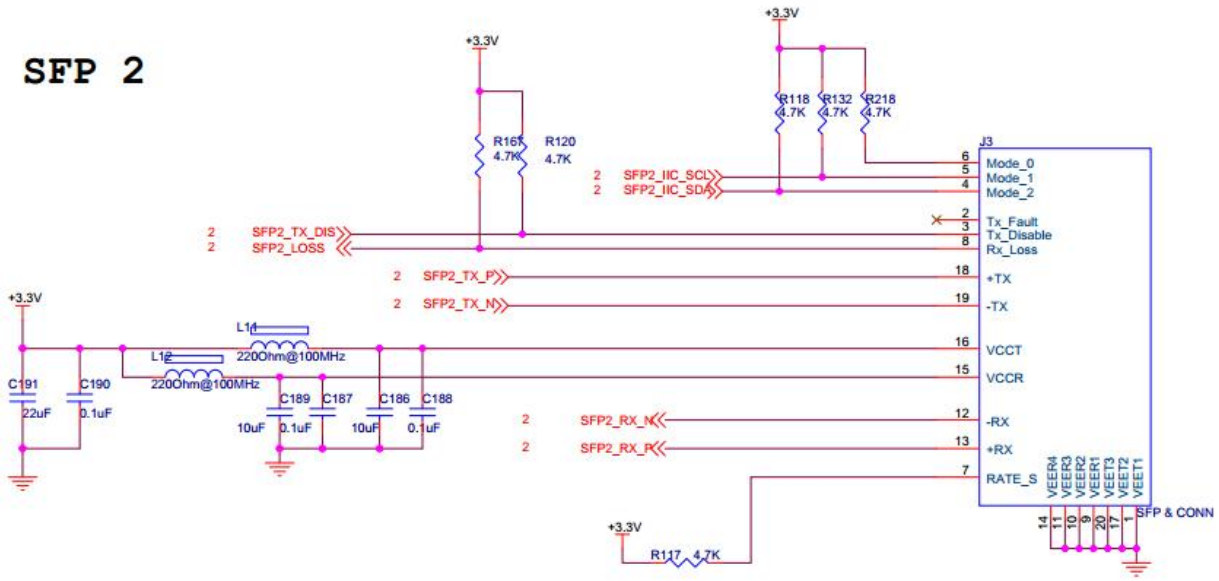
在开发板上，有 2 路光纤接口 SFP1~SFP2, 分别连接到 FPGA 芯片的 HSSTLP 的通道上。FPGA 和光纤连接的设计示意图如下图所示:



其中 SFP1 光模块接口连接到 Q3 HSSTLP 的 Channel3 上，SFP2 跟 Q3 HSSTLP 的的 Channel2 相连。光模块和 FPGA 之间用 0.1uf 的电容隔开，使用 AC Couple 的模式。

光模块的 LOSS 信号和 TX_Disable 信号连接到 FPGA 的普通 IO 上。LOSS 信号用来检测光模块的光接收是否丢失，如果没有插入光纤或者 Link 上，LOSS 信号为高，否则为低。TX_Disable 信号用来使能或者不使能光模块的光发射，如果 TX_Disable 信号为高，光发射关闭，否则光发送使能，正常使用的时候需要拉低此信号。硬件原理图如下：

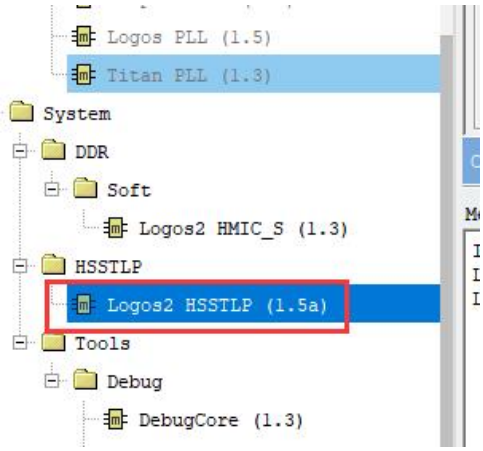




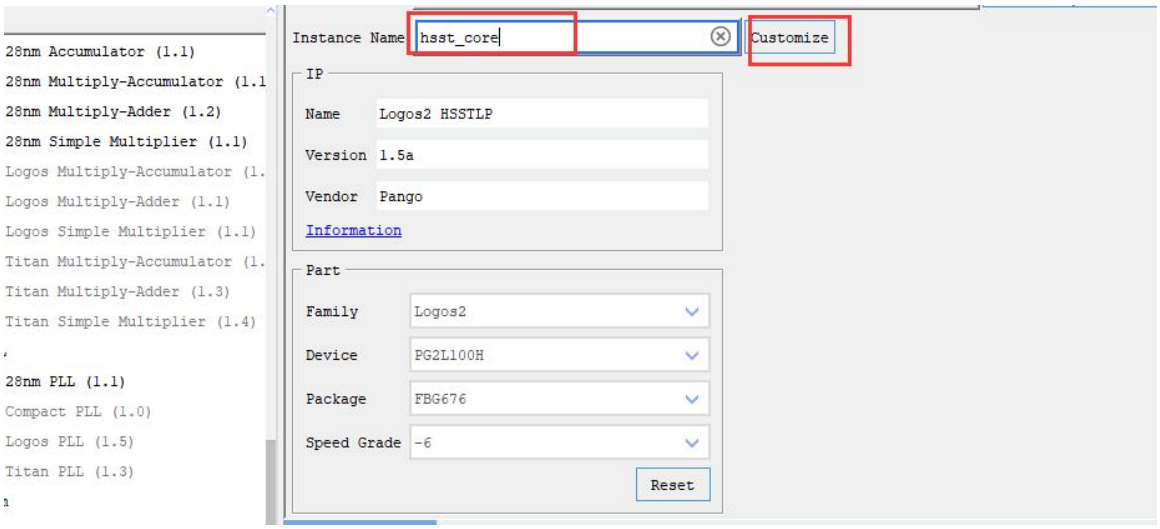
3 程序设计

我们先来测试一下开发板上的 HSSTLP 模块工作是否正常。以下是具体测试步骤：

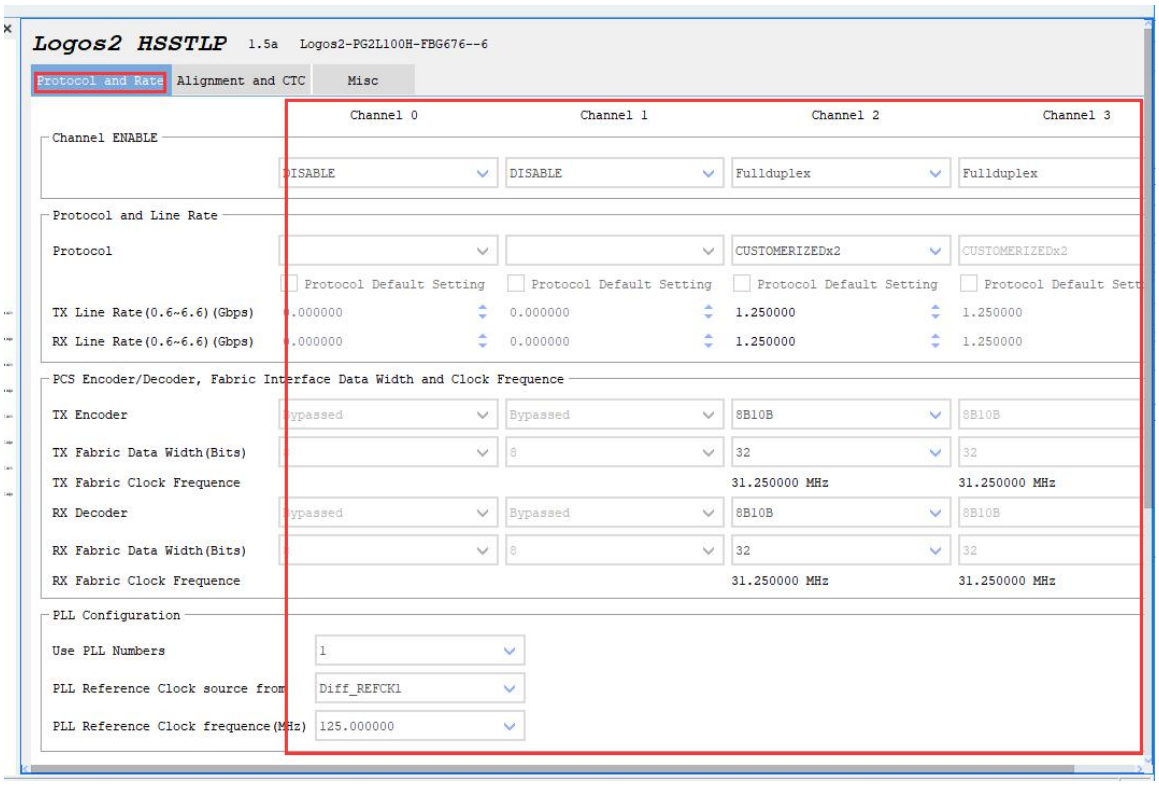
1. 新建一个工程 hsst_core，在 tools 菜单栏下 IP Compiler 添加 HSST IP，ipm2l_hsstlp_v1_5a.iar IP 在 hsst_test 文件夹下，添加 IP 在《DDR3 读写测试实验》教程中已讲过，这里不再重复。



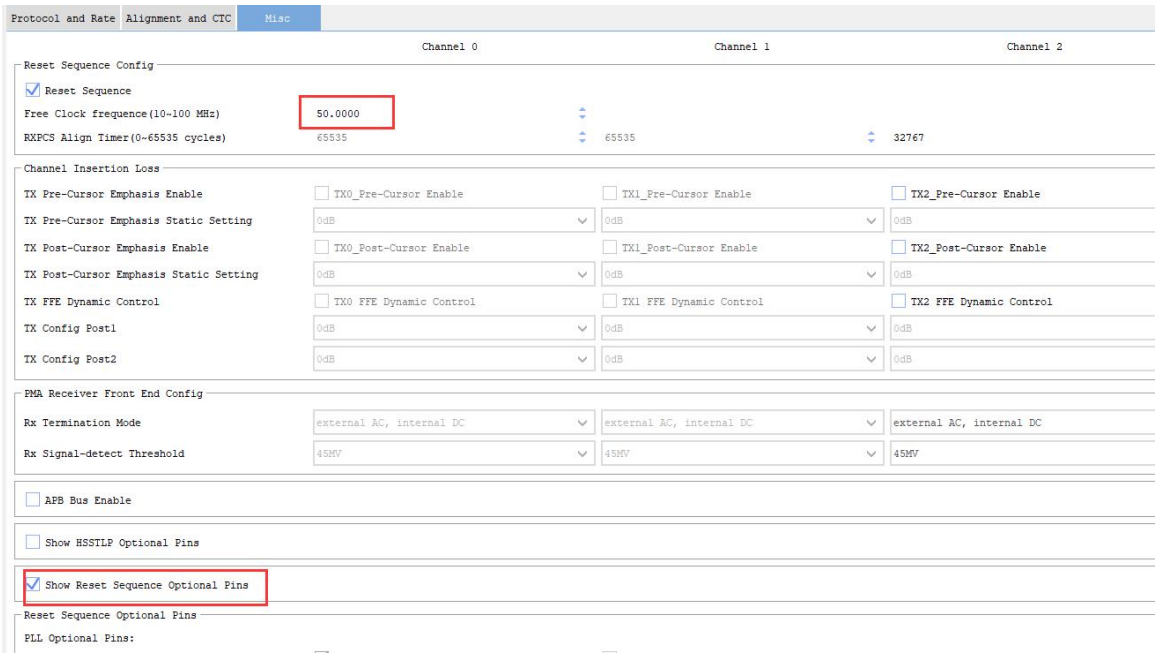
2.新建 HSSTLP IP 并命名为 hsst_core，再单击 Customize；



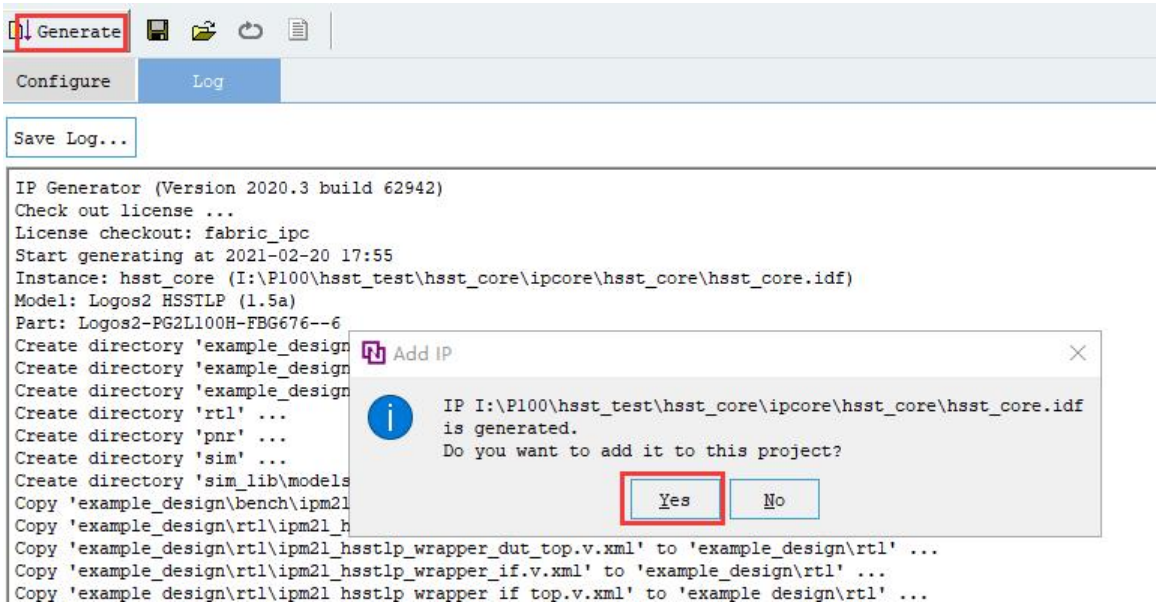
3.在弹出的界面中 Protocol and Rate 栏中按如下设置，Channel0 Channel1 为 DISABLE，Channel2 Channel3 为 Fullduplex 由硬件决定，速率：1.25G，数据位宽：32 位，选择 8/10 编解码，参考时钟为 Diff_REFCK1 ，频率 125M，Protocol 为 CUSTOMERIZEDX2;如下图所示：



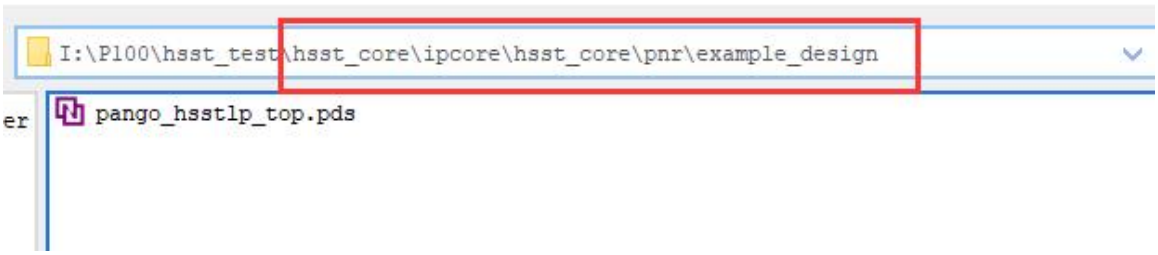
4.在 Alignment and CTC 栏中采用默认设置， Misc 栏中采用设置如下图所示：

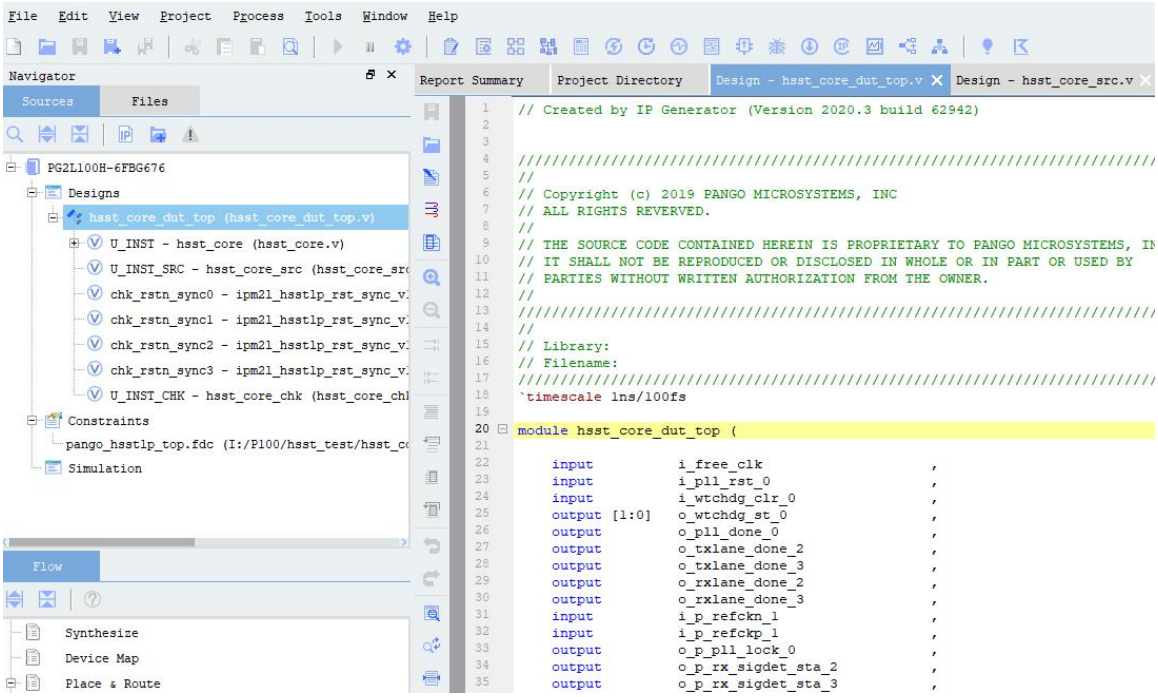


5. 单击 Generate 产生 IP 即可，并单击”Yes”，软件会自动生成 demo。



6. 打开软件生成的 demo,工程在如下位置：





7. 为了适合开发板硬件电路设计，主要对 hsst_core_dut_top.v 的复位进行修改，进行统一复位，增加调试接口测试，修改管脚分配，其它不变，然后程序编译综合产生位流文件。

```
assign          SFP1_IIC_SCL          = 1'b1 ;
assign          SFP2_IIC_SCL          = 1'b1 ;
assign         SFP_TX_DISABLE1        = 1'b0 ;
assign         SFP_TX_DISABLE2        = 1'b0 ;
wire          i_wtchdg_clr_0 ;
wire          src_rst                  /* synthesis PAP_MARK_DEB
wire          chk_rst                 /* synthesis PAP_MARK_DEBU
wire          pll_rst_d;
wire          i_pll_rst_0;
wire          i_p_cfg_rst;
wire          rst_n_dly_d0;
assign        i_p_cfg_rst=~rst_n;
assign        i_pll_rst_0= pll_rst_d;
assign        pll_rst_d=i_p_cfg_rst;
assign        i_wtchdg_clr_0=pll_rst_d;

assign        src_rst=pll_rst_d;
assign        chk_rst=pll_rst_d;
```

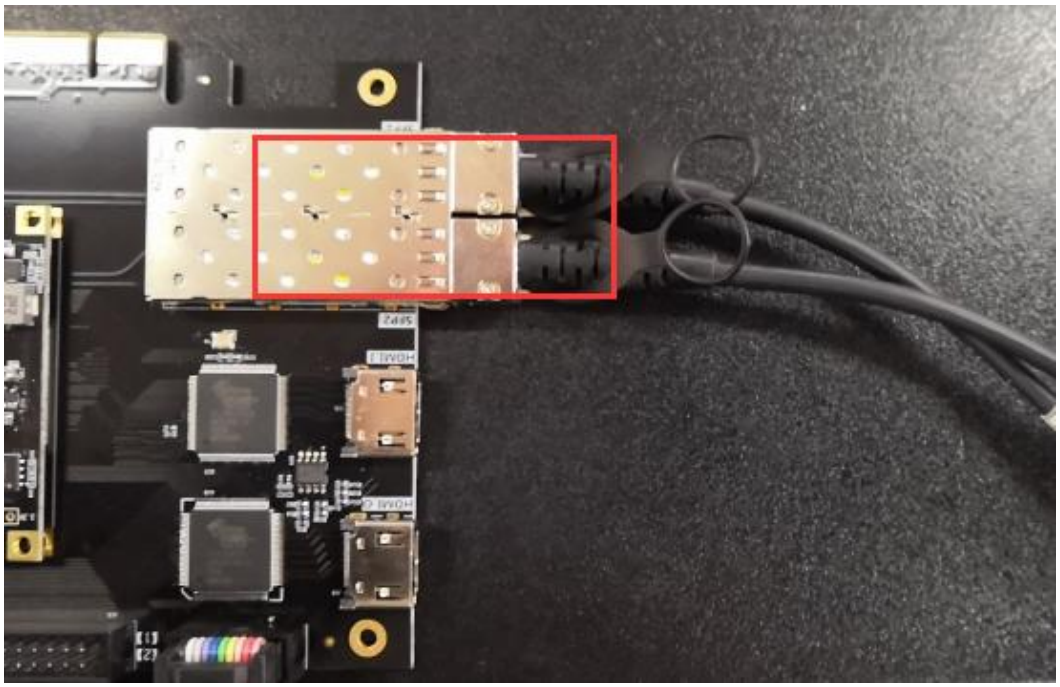
```
create_clock -name {free_clk} [get_ports {i_free_clk}] -period {20} -waveform {0.000 10.000}
create_clock -name {p_clk2core_tx_2} [get_pins {U_INST.o_p_clk2core_tx_2}] -period {6.4} -waveform {0 3.2}
#create_clock -name {p_clk2core_rx_2} [get_pins {U_INST.o_p_clk2core_rx_2}] -period {6.4} -waveform {0 3.2}
##==== END Clocks
##==== BEGIN "set_clock_groups"
set_clock_groups -name free_clk        -asynchronous -group [get_clocks {free_clk}]
set_clock_groups -name p_clk2core_tx_0 -asynchronous -group [get_clocks {p_clk2core_tx_0}]
set_clock_groups -name p_clk2core_tx_1 -asynchronous -group [get_clocks {p_clk2core_tx_1}]
set_clock_groups -name p_clk2core_tx_2 -asynchronous -group [get_clocks {p_clk2core_tx_2}]
set_clock_groups -name p_clk2core_tx_3 -asynchronous -group [get_clocks {p_clk2core_tx_3}]
set_clock_groups -name p_clk2core_rx_0 -asynchronous -group [get_clocks {p_clk2core_rx_0}]
set_clock_groups -name p_clk2core_rx_1 -asynchronous -group [get_clocks {p_clk2core_rx_1}]
set_clock_groups -name p_clk2core_rx_2 -asynchronous -group [get_clocks {p_clk2core_rx_2}]
set_clock_groups -name p_clk2core_rx_3 -asynchronous -group [get_clocks {p_clk2core_rx_3}]
##==== END "set_clock_groups"
##==== BEGIN "Inputs/Outpusts"
##==== END "Inputs/Outpusts"
##==== BEGIN "Delay Paths"
##==== END "Delay Paths"
##==== BEGIN Attributes - (Populated from tab in SCOPE, do not edit)
## The Q3 Location Constraint
define_attribute {i:U_INST.U_GTP_HSSTLP_WRAPPER.CHANNEL2_ENABLE.U_GTP_HSSTLP_LANE2} {PAP_LOC} {HSSTLP_364_918:U2_HSSTLP_LANE}
define_attribute {i:U_INST.U_GTP_HSSTLP_WRAPPER.CHANNEL3_ENABLE.U_GTP_HSSTLP_LANE3} {PAP_LOC} {HSSTLP_364_918:U3_HSSTLP_LANE}
define_attribute {i:U_INST.U_GTP_HSSTLP_WRAPPER.PLL0_ENABLE.U_GTP_HSSTLP_PLL0} {PAP_LOC} {HSSTLP_364_918:U0_HSSTLP_PLL}
## The Q6 Location Constraint
#define_attribute {i:U_INST.U_GTP_HSSTLP_WRAPPER.CHANNEL0_ENABLE.U_GTP_HSSTLP_LANE0} {PAP_LOC} {HSSTLP_364_0:U0_HSSTLP_LANE}
#define_attribute {i:U_INST.U_GTP_HSSTLP_WRAPPER.CHANNEL1_ENABLE.U_GTP_HSSTLP_LANE1} {PAP_LOC} {HSSTLP_364_0:U1_HSSTLP_LANE}
#define_attribute {i:U_INST.U_GTP_HSSTLP_WRAPPER.CHANNEL2_ENABLE.U_GTP_HSSTLP_LANE2} {PAP_LOC} {HSSTLP_364_0:U2_HSSTLP_LANE}
#define_attribute {i:U_INST.U_GTP_HSSTLP_WRAPPER.CHANNEL3_ENABLE.U_GTP_HSSTLP_LANE3} {PAP_LOC} {HSSTLP_364_0:U3_HSSTLP_LANE}
#define_attribute {i:U_INST.U_GTP_HSSTLP_WRAPPER.PLL0_ENABLE.U_GTP_HSSTLP_PLL0} {PAP_LOC} {HSSTLP_364_0:U0_HSSTLP_PLL}
#define_attribute {i:U_INST.U_GTP_HSSTLP_WRAPPER.PLL1_ENABLE.U_GTP_HSSTLP_PLL1} {PAP_LOC} {HSSTLP_364_0:U1_HSSTLP_PLL}
##==== END Attributes
define_attribute {t:U_INST.o_p_clk2core_tx_2} {PAP_CLOCK_ASSIGN} {GTP_CLKBUFG}
#define_attribute {t:U_INST.o_p_clk2core_rx_2} {PAP_CLOCK_ASSIGN} {GTP_CLKBUFG}
##==== BEGIN Attributes - (Populated from tab in SCOPE, do not edit)
##==== END Attributes
##==== BEGIN Attributes - IO table (Populated from tab in SCOPE, do not edit)
####==== END Attributes(IO table)
```

8. 把光纤两端接入接口 SFP1 和 SFP2 进行环路测试，下载程序到开发板中进行在线调试，可看到窗口中发送和接收的数据一致的。

4 实验现象

开发板自身是不带 SFP 的光模块和光纤的，所以测试之前需要自己准备 SFP 的光模块和光纤。因为光纤传输至少需要 2 个光模块，用户需要准备 2 个 SFP 光模块才是做光纤通信实验。10G 或者 1.25G SFP 的光模块和光纤再淘宝上都能购买到，在购买 SFP 光模块的同时，同时让商家提供配套的光纤就可以了。

测试之前我们把 SFP 的光模块分别插入到光模块的接口上，再用光纤把光模块 SFP1 和 SFP2 对连起来。因为这里我们用的光模块及光纤是 TX 和 RX 是分开的，这样 SFP1 光模块 RX 需要跟 SFP2 光模块的 TX 相连，SFP1 光模块的 TX 需要连接到 SFP2 光模块的 RX。连接后如下图所示：



下载程序到开发板中进行在线调试，可看到窗口中发送和接收的数据一致的。