

## GTX 光纤通信测试例程

黑金动力社区 2019-04-30

---

# 1 实验简介

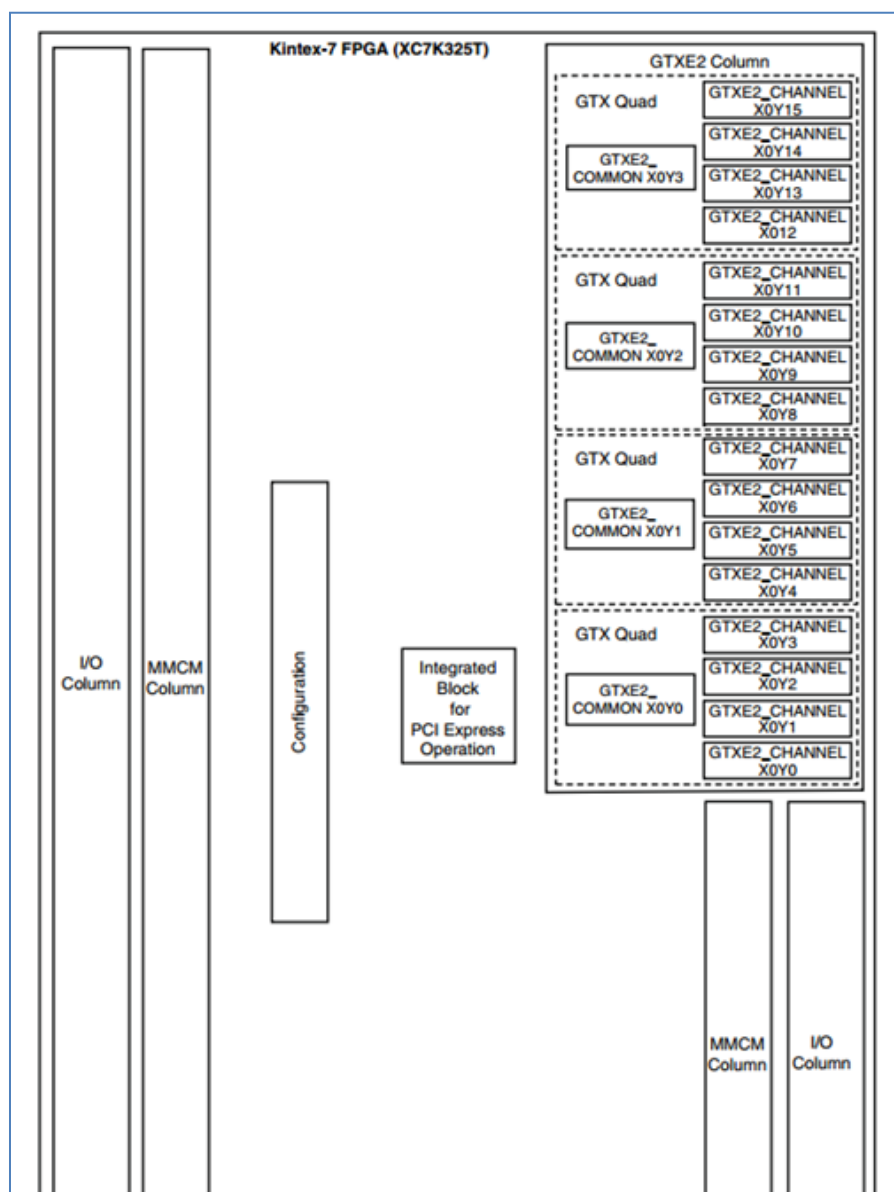
Xilinx 的 Kintex-7 系列 FPGA 集成了 GTX 串行高速收发器, 可以实现高速串行数据通信。在 AX7325 开发板上, FPGA 的 GTX 其中的 8 个收发器通道中 4 路连接到 4 路 SFP 光模块接口, 另外 4 路连接到 QSFP 光模块接口, 用户只需要另外购买 SPF 的光模块就可以实现光纤的数据传输。本实验将介绍通过光纤连接实现光模块之间的数据收发和眼图的测试。

# 2 实验原理

## 2.1 GTX 介绍

AX7325 的 FPGA 芯片(XC7K325TFFG900)自带 16 路高速 GTX 串行高速收发器通道, 每通道的收发速度为高达 12.5 Gb/s。GTX 收发器支持不同的串行传输接口或协议, 比如 PCIE 1.1/2.0/3.0 接口、万兆网 XUI 接口、OC-48、串行 RapidIO 接口、SATA(Serial ATA) 接口、数字分量串行接口(SDI)等等。

Xilinx 以 Quad 来对串行高速收发器进行分组, 四个串行高速收发器和一个 COMMOM (QPLL) 组成一个 Quad, 每一个串行高速收发器称为一个 Channel(通道), 下图为 16 路 GTX 收发器在 Kintex-7 FPGA 芯片中的示意图:



GTX 的具体内部逻辑框图如下所示，它由 4 个如下图的四个收发器通道 GTXE2\_CHANNEL 和一个 GTXE2\_COMMON 组成。每路 GTXE2\_CHANNEL 包含发送电路 TX 和接收电路 RX。

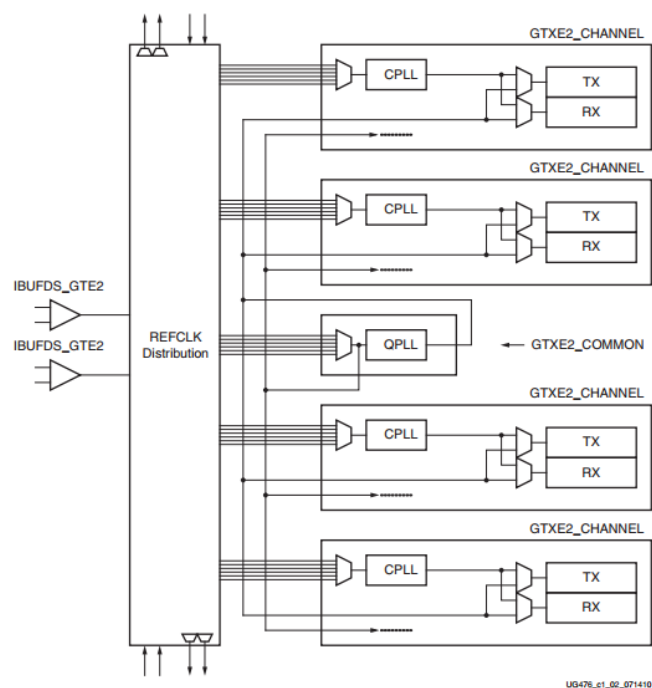
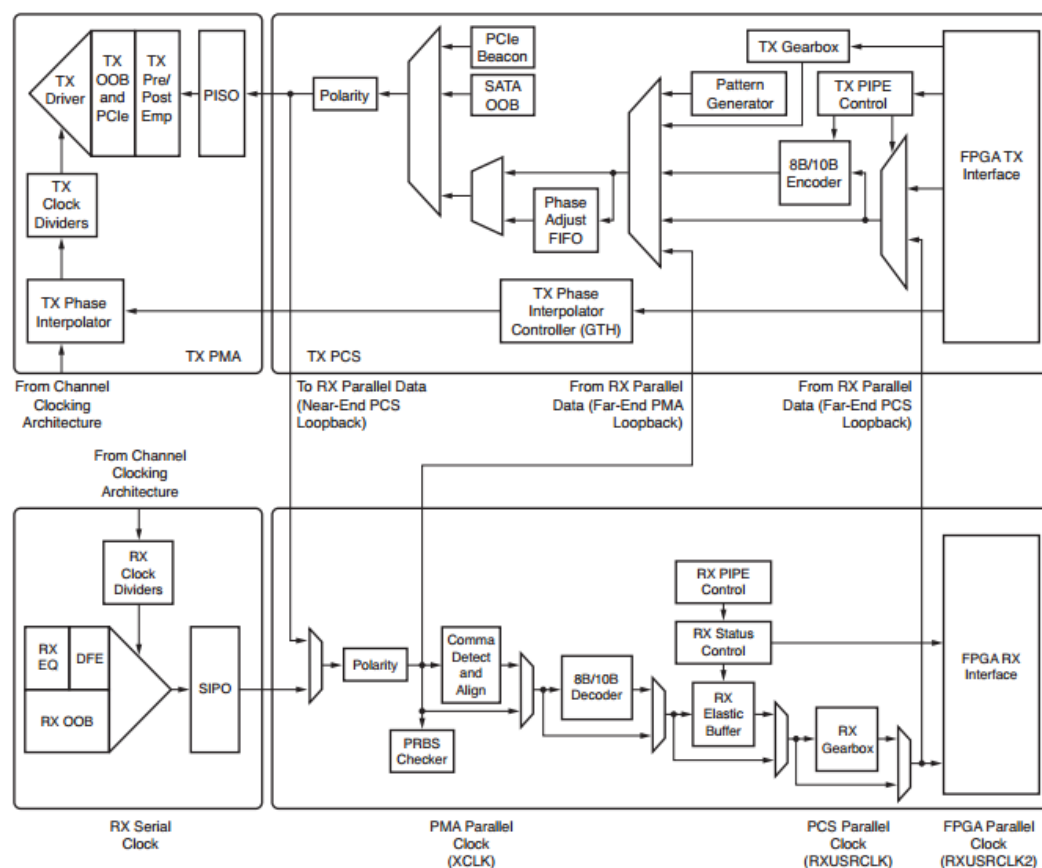


Figure 1-2: GTX Transceiver Quad Configuration

每个 GTXE2\_CHANNEL 的逻辑电路如下图所示：



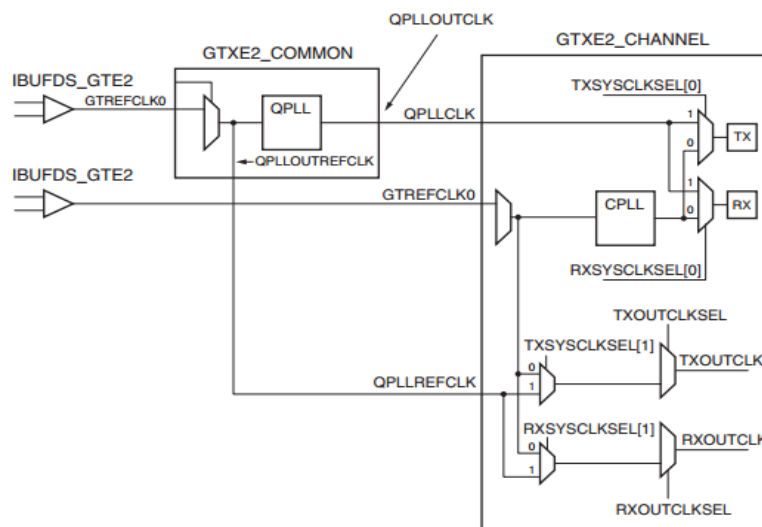
GTXE2\_CHANNEL 的发送端和接收端功能是独立的，均由 PMA(Physical Media Attachment，物理媒介适配层)和 PCS(Physical Coding Sublayer，物理编码子层)两个子层组成。其中 PMA 子层包含高速串并转换(Serdes)、预/后加重、接收均衡、时钟发生器及时钟恢复等电路。PCS 子层包含 8B/10B 编解码、缓冲区、通道绑定和时钟修正等电路。

### GTX 发送和接收处理流程：

首先用户逻辑数据经过 8B/10B 编码后，进入一个发送缓存区（Phase Adjust FIFO），该缓冲区主要是 PMA 子层和 PCS 子层两个时钟域的时钟隔离，解决两者时钟速率匹配和相位差异的问题，最后经过高速 Serdes 进行并串转换(PISO)，有必要的，可以进行预加重(TX Pre-emphasis)、后加重。值得一提的是，如果在 PCB 设计时不慎将 TXP 和 TXN 差分引脚交叉连接，则可以通过极性控制(Polarity)来弥补这个设计错误。接收端和发送端过程相反，相似点较多，这里就不赘述了，需要注意的是 RX 接收端的弹性缓冲区，其具有时钟纠正和通道绑定功能。

### GTX 的参考时钟

GTX 模块有四组参考时钟，每组有两个差分参考时钟输入管脚 (MGTREFCLK0P/N 和 MGTREFCLK1P/N)，作为 GTX 模块的参考时钟源，用户可以自行选择。AX7102 的核心板上，有一路 125Mhz 的 GTX 参考时钟连接到 MGTREFCLK0P/N 上，作为 GTX 的参考时钟。差分参考时钟通过 IBUFDS 模块转换成单端时钟信号进入到 GTXE2\_COMMON 的 QPLL 和 GTXE2\_CHANNEL 的 CPLL 中，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。



## GTX 的 FPGA TX 接口信号

TX 接口信号是 FPGA 的用户数据发往 GTX 的接口信号，该接口信号的名称和说明如下表所示：

Port	Dir	Clock Domain	Description
TXCHARDISPMODE[7:0]	In	TXUSRCLK2	When 8B/10B encoding is disabled, TXCHARDISPMODE is used to extend the data bus for 20-, 40- and 80-bit TX interfaces.
TXCHARDISPVAL[7:0]	In	TXUSRCLK2	When 8B/10B encoding is disabled, TXCHARDISPVAL is used to extend the data bus for 20-, 40- and 80-bit TX interfaces.

Port	Dir	Clock Domain	Description
TXDATA[63:0]	In	TXUSRCLK2	The bus for transmitting data. The width of this port depends on TX_DATA_WIDTH: TX_DATA_WIDTH = 16, 20: TXDATA[15:0] = 16 bits wide TX_DATA_WIDTH = 32, 40: TXDATA[31:0] = 32 bits wide TX_DATA_WIDTH = 64, 80: TXDATA[63:0] = 64 bits wide When a 20-bit, 40-bit, or 80-bit bus is required, the TXCHARDISPVAL and TXCHARDISPMODE ports from the 8B/10B encoder is concatenated with the TXDATA port. See <a href="#">Table 3-2, page 109</a> .
TXUSRCLK	In	Clock	This port is used to provide a clock for the internal TX PCS datapath.
TXUSRCLK2	In	Clock	This port is used to synchronize the FPGA logic with the TX interface. This clock must be positive-edge aligned to TXUSRCLK when TXUSRCLK is provided by the user.

其中 TXCHARDISPMODE[7:0]和 TXCHARDISPVAL[7:0]的功能由 TX8B10BEN 是否使能来决定，当 8B/10B 使能，这些信号作为 TX 的数据信号。

TXDATA[63:0] 发送数据接口信号，数据宽度由 TX\_DATA\_WIDTH 参数决定，当 TX\_DATA\_WIDTH 为 16，或者 20 时，TXDATA 数据宽度为 16；当 TX\_DATA\_WIDTH 为 32，或者 40 时，TXDATA 数据宽度为 32，当 TX\_DATA\_WIDTH 为 64，或者 80 时，TXDATA 数据宽度为 64。通过 TX8B10BEN、TX\_DATA\_WIDTH 参数设置可以配置成不同的 FPGA 接口数据位宽和 GTX 内部数据宽度，如下表所示：

TX8B10BEN	TX_DATA_WIDTH	TX_INT_DATAWIDTH	FPGA Interface Width	Internal Data Width
0	16	0	16	16
	20	0	20	20
	32	0	32	16
	32	1	32	32
	40	0	40	20
	40	1	40	40
	64	1	64	32
	80	1	80	40

当 TX8B10BEN 不使能时，由 TXCHARDISPMODE[7:0]、TXCHARFDPVAL[7:0]信号和 TXDATA 组合成 20 位、40 位和 80 位的数据信号。

		<<< Data Transmission Order is Right to Left (LSB to MSB) <<<																																													
		39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
Data Transmitted	TXCHARDISPMODE[3]	TXCHARDISPVAL[3]		TXDATA[31:24]								TXCHARDISPMODE[2]		TXCHARDISPVAL[2]		TXDATA[23:16]								TXCHARDISPMODE[1]		TXCHARDISPVAL[1]		TXDATA[15:8]								TXCHARDISPMODE[0]		TXCHARDISPVAL[0]		TXDATA[7:0]							
<<< Data Transmission Order is Right to Left (LSB to MSB) <<<																																															
Data Transmitted	TXCHARDISPMODE[7]	TXCHARDISPVAL[7]		TXDATA[63:56]								TXCHARDISPMODE[6]		TXCHARDISPVAL[6]		TXDATA[55:48]								TXCHARDISPMODE[5]		TXCHARDISPVAL[5]		TXDATA[47:40]								TXCHARDISPMODE[4]		TXCHARDISPVAL[4]		TXDATA[39:30]							

所有的 FPGA 接口信号的采样时钟是 TXUSRCLK2，在 TXUSRCLK2 的上升沿对 TXDATA 进行采样。TXUSRCLK 是提供给 GTX 模块的 PCS logic 和数据发送，TXUSRCLK 的时钟频率由 GTX 的串行发送速度和内部的数据宽度决定的。计算的公式如下：

$$TXUSRCLK \text{ Rate} = \frac{\text{Line Rate}}{\text{Internal Datapath Width}}$$

TXUSRCLK2 的频率和 TXUSRCLK 有相关性，它是由 TXUSRCLK 的频率和 TX\_DATA\_WIDTH 的值决定的。TXUSRCLK2 的时钟频率计算如下表所示：

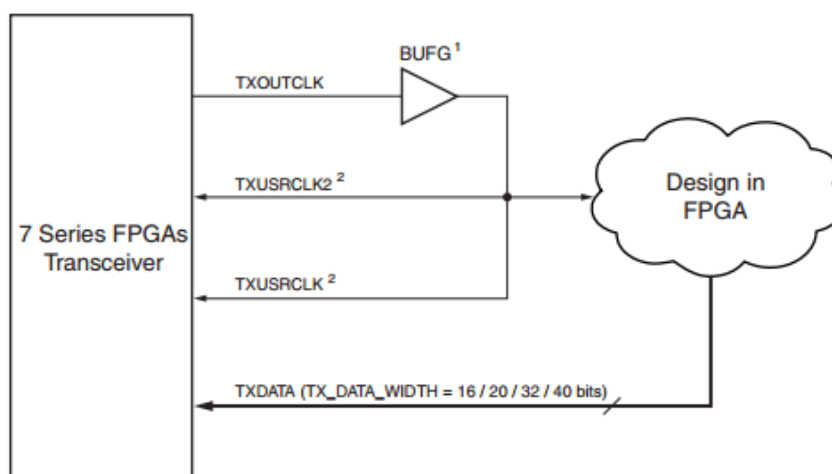
Table 3-3: TXUSRCLK2 Frequency Relationship to TXUSRCLK

FPGA Interface Width	TX_DATA_WIDTH	TX_INT_DATAWIDTH	TXUSRCLK2 Frequency
2-Byte	16, 20	0	$F_{TXUSRCLK2} = F_{TXUSRCLK}$
4-Byte	32, 40	0	$F_{TXUSRCLK2} = F_{TXUSRCLK}/2$
4-Byte	32, 40	1	$F_{TXUSRCLK2} = F_{TXUSRCLK}$
8-Byte	64, 80	1	$F_{TXUSRCLK2} = F_{TXUSRCLK}/2$

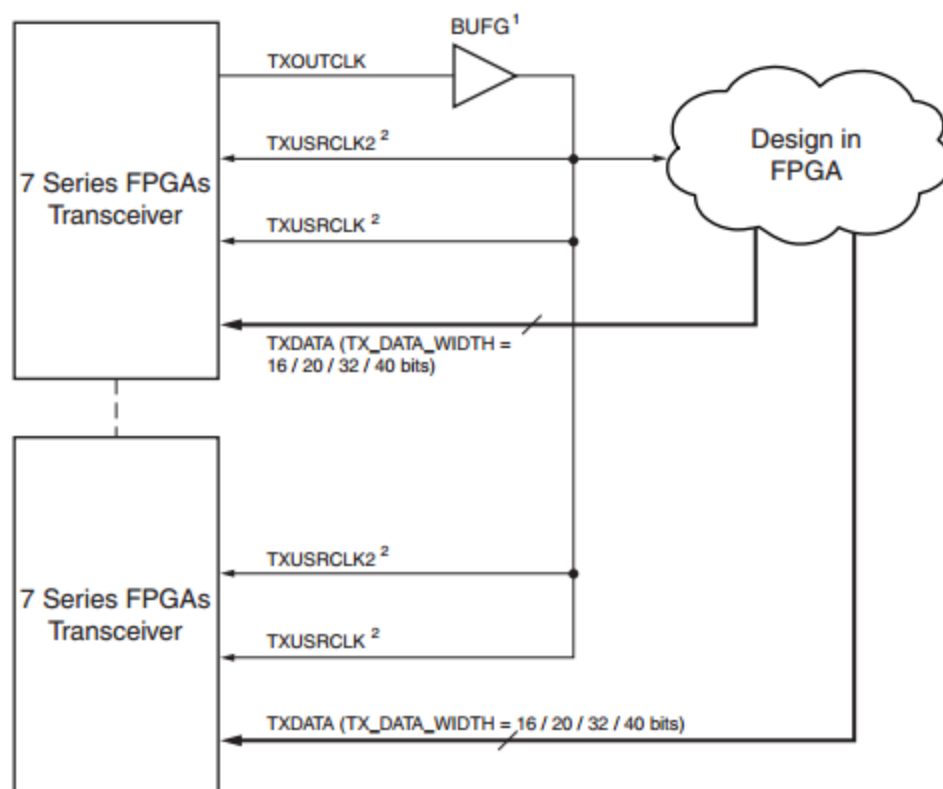
TXUSRCLK2 和 TXUSRCLK 两个时钟时应该遵循下面两个准则：

1. TXUSRCLK 和 TXUSRCLK2 必须是上升沿对齐的，偏差越小越好，因此应该使用 BUFGs 或者 BUFRs 来驱动这两个时钟。
2. 即使 TXUSRCLK、TXUSRCLK2 和 GTX 的参考时钟运行在不同的时钟频率，必须保证三者必须使用同源时钟。所以 TXUSRCLK、TXUSRCLK2 时钟频率必须由 GTX 的参考时钟倍频或者分频得到。

在 XILINX 的官方文档" ug476\_7Series\_Transceivers.pdf"中，建议使用 TXOUTCLK 信号来驱动 TXUSRCLK 和 TXUSRCLK2 时钟，当 TX\_DATA\_WIDTH=16 或者 20 时 (2 字节模式) 或 TX\_DATA\_WIDTH=32 或者 40 时 (4 字节模式且 TX\_INT\_DATAWIDTH=1 时)，TXUSRCLK 和 TXUSRCLK2 的时钟频率相等。TXOUTCLK 通过 BUFG 直接驱动 TXUSRCLK 和 TXUSRCLK2，具体框图如下图所示：



当为其它两种模式时，TXUSRCLK 是 TXUSRCLK2 的时钟频率的 2 倍。TXOUTCLK 通过 MMCM 或者 PLL 来产生 TXUSRCLK 和 TXUSRCLK2 的时钟，具体框图如下图所示：



### GTX 的 FPGA RX 接口信号

RX 接口信号是 GTX 模块发往 FPGA 用户数据的接口信号，该接口信号的名称和说明如下表所示：



Port	Dir	Clock Domain	Description
RXDISPERR[7:0]	Out	RXUSRCLK2	When 8B/10B decoding is disabled, RXDISPERR is used to extend the data bus for 20-bit, 40-bit and 80-bit RX interfaces.
RXCHARISK[7:0]	Out	RXUSRCLK2	When 8B/10B decoding is disabled, RXCHARISK is used to extend the data bus for 20-bit, 40-bit and 80-bit RX interfaces.
RXDATA[63:0]	Out	RXUSRCLK2	The bus for transmitting data. The width of this port depends on RX_DATA_WIDTH: RX_DATA_WIDTH = 16, 20: RXDATA[15:0] = 16 bits wide RX_DATA_WIDTH = 32, 40: RXDATA[31:0] = 32 bits wide RX_DATA_WIDTH = 64, 80: RXDATA[63:0] = 64 bits wide When a 20-bit, 40-bit, or 80-bit bus is required, the RXCHARISK and RXDISPERR ports from the 8B/10B encoder are concatenated with the RXDATA port. See Table 4-52, page 296.
RXUSRCLK	In	Clock	This port is used to provide a clock for the internal RX PCS datapath.
RXUSRCLK2	In	Clock	This port is used to synchronize the FPGA logic with the RX interface. This clock must be positive-edge aligned to RXUSRCLK when RXUSRCLK is provided by the user.

RX 接口信号基本和 TX 接口信号类似, 该用户端口根据不同的 RX8B10BEN 和 RX\_DATA\_WIDTH 两个参考可以设置成 FPGA 的数据接口宽度和 GTX 内部并行数据宽度如下表所示:

Table 4-51: FPGA RX Interface Datapath Configuration

RX8B10BEN	RX_DATA_WIDTH	RX_INT_DATAWIDTH	FPGA Interface Width	Internal Data Width
1	20	0	16	20
	40	0	32	20
	40	1	32	40
	80	1	64	40
0	16	0	16	16
	20	0	20	20
	32	0	32	16
	32	1	32	32
	40	0	40	20
	40	1	40	40
	64	1	64	32
	80	1	80	40

同样 RXUSRCLK 的频率和 RXUSERCLK2 的频率关系也跟 RX\_DATA\_WIDTH 有关。

**Table 4-53: RXUSRCLK2 Frequency Relationship to RXUSRCLK**

FPGA Interface Width	RX_DATA_WIDTH	RX_INT_DATAWIDTH	RXUSRCLK2 Frequency
2-Byte	16, 20	0	$F_{RXUSRCLK2} = F_{RXUSRCLK}$
4-Byte	32, 40	0	$F_{RXUSRCLK2} = F_{RXUSRCLK} / 2$
4-Byte	32, 40	1	$F_{RXUSRCLK2} = F_{RXUSRCLK}$
8-Byte	64, 80	1	$F_{RXUSRCLK2} = F_{RXUSRCLK} / 2$

关于 GTX 的知识就讲到这里，大家想了解更多的 GTX 部分硬件的资料和应用，请参考 Xilinx 提供的文档"ug476\_7Series\_Transceivers.pdf"。

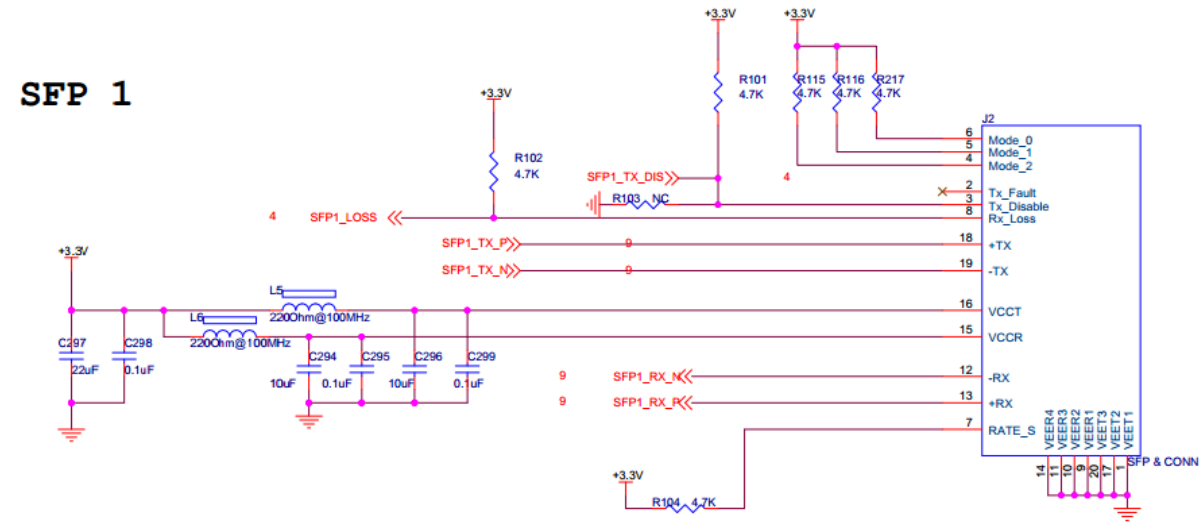
## 2.2 硬件介绍

在 AX7325 开发板上，有 4 路光纤接口 OPT1~OPT4 和 1 路四合 1 的 QSFP 光纤接口，分别连接到 FPGA 芯片的 GTX 的通道上。

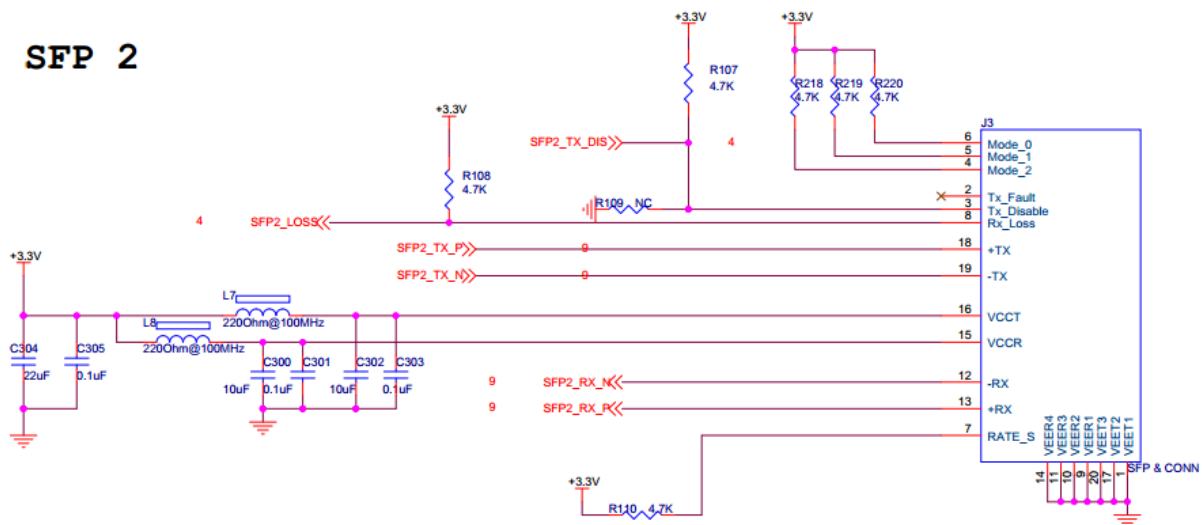
其中 OPT1 光模块接口连接到 GTX 的 Channel0 上，OPT2 跟 GTX 的 Channel1 相连，OPT3 跟 GTX 的 Channel2 相连，OPT4 跟 GTX 的 Channel3 相连。QSFP 光模块连接到 BANK118 的 GTX 的 Channel0~ Channel3，光模块和 FPGA 之间用 0.1uf 的电容器隔开，使用 AC Couple 的模式。

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：

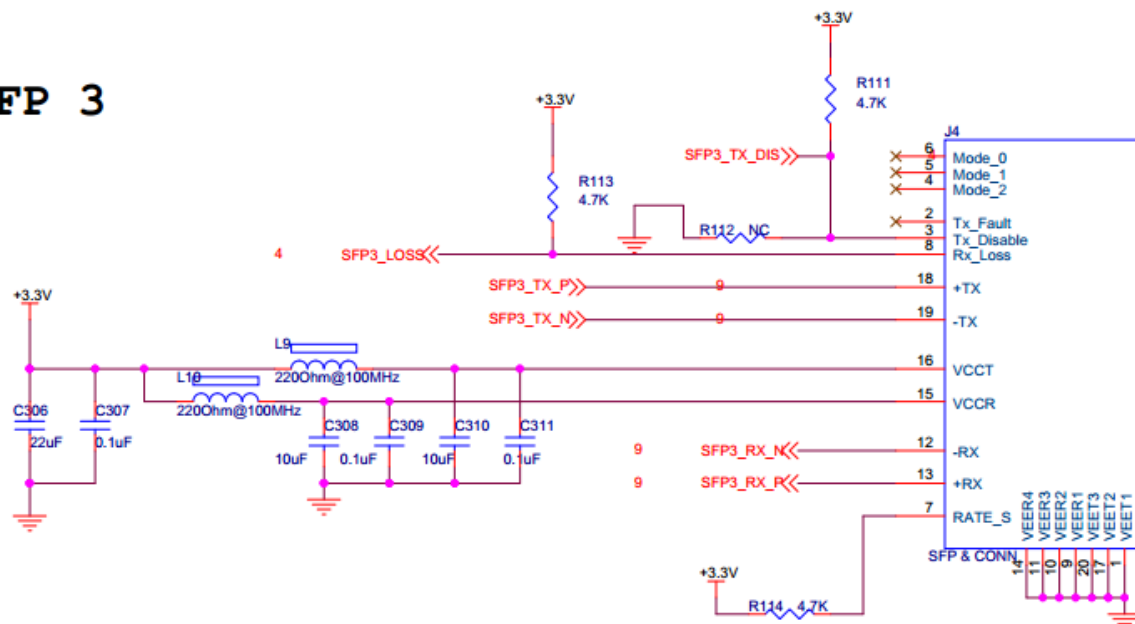
## SFP 1



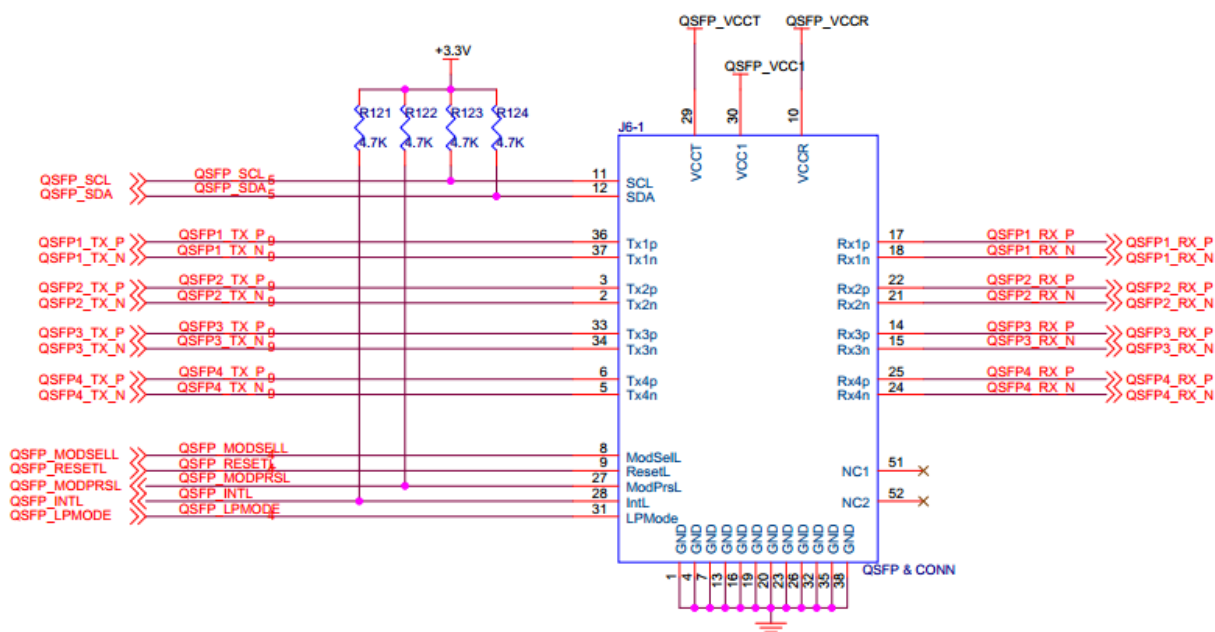
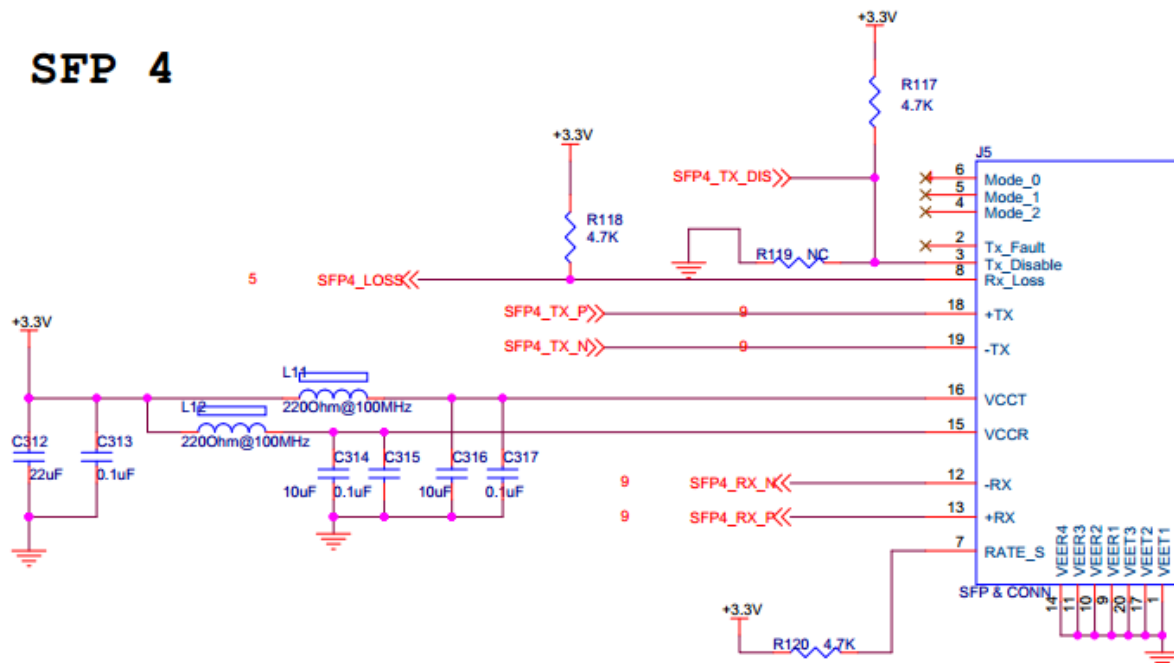
## SFP 2



## SFP 3



## SFP 4



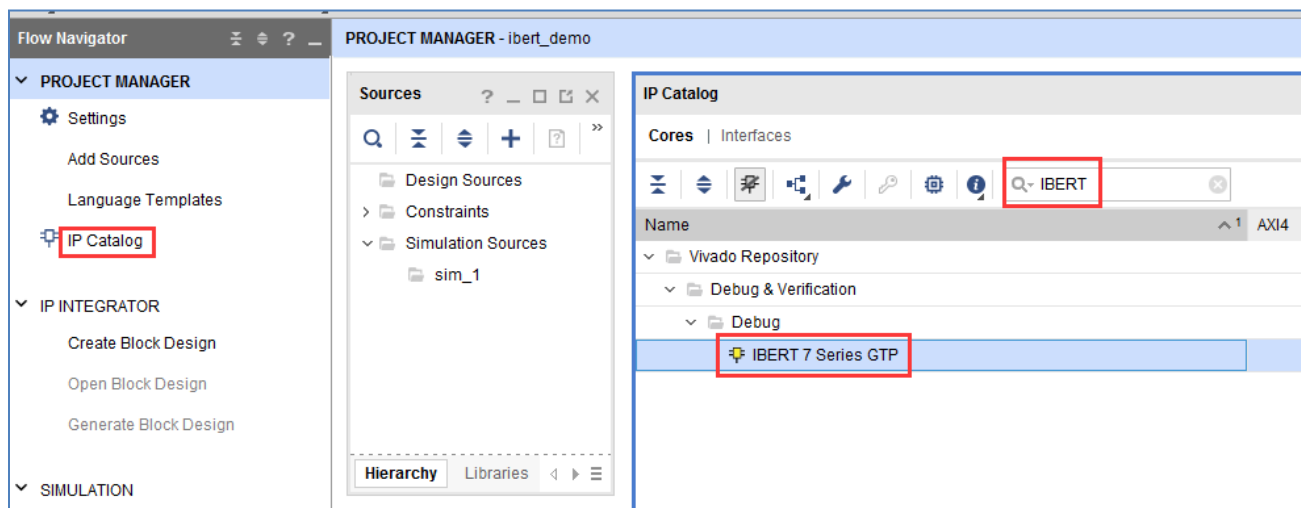
### 3 程序设计

这里有两个程序：一个是 4 路 SFP 光纤模块通信测试程序，另一个是 QSFP 光纤模块与 4 路 SFP 通信测试程序。

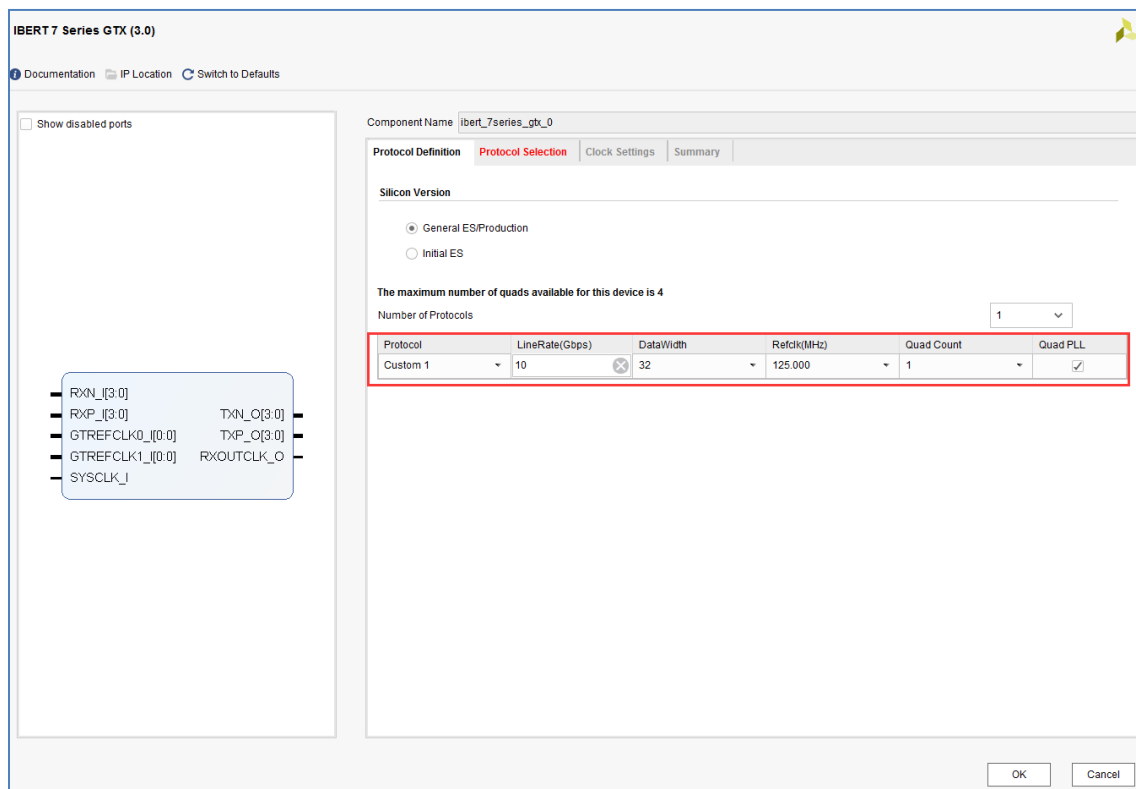
### 3.1 4 路 SFP 光纤模块通信测试程序

在使用 GTX 之前，我们先来测试一下开发板上的 GTX 模块工作是否正常。以下为 GTX 数据通信的具体测试步骤：

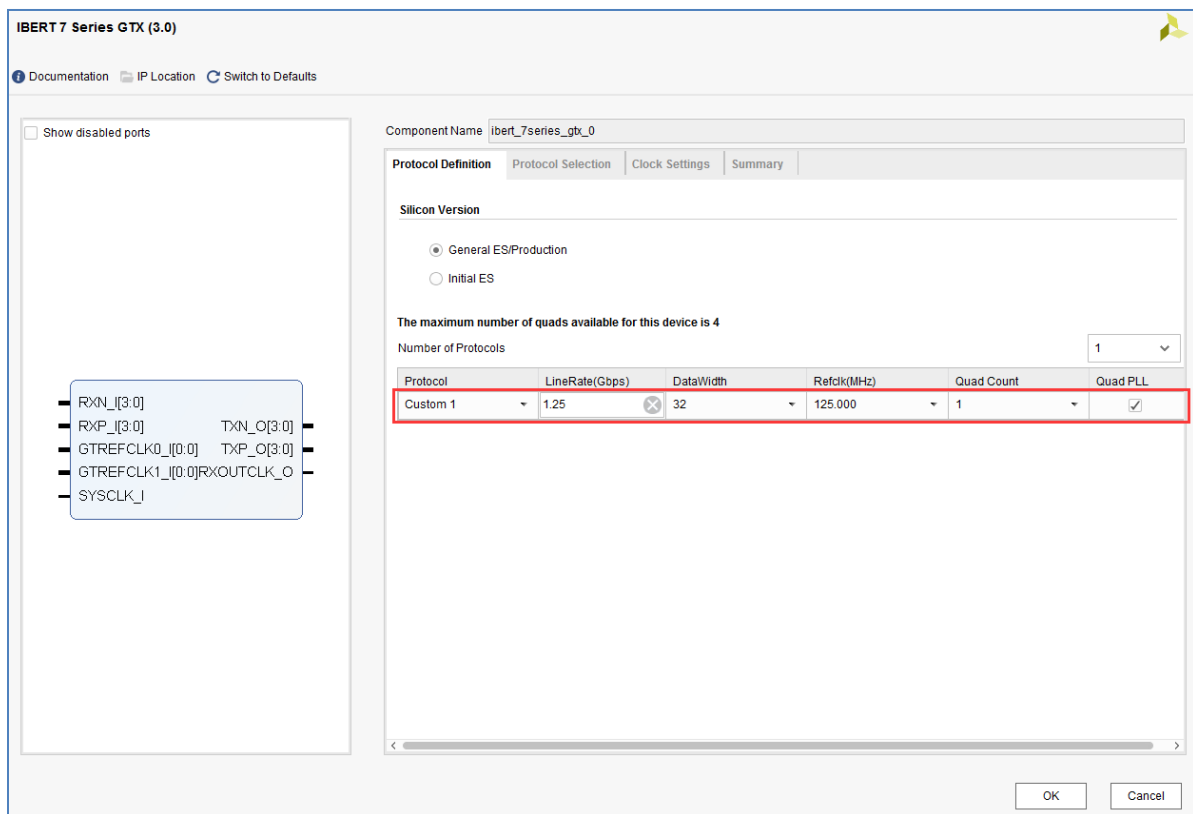
1. 新建一个工程 ibert\_demo，在 IP Catalog 界面中双击 Debug 目录下的 IBERT 7 Series GTX IP。



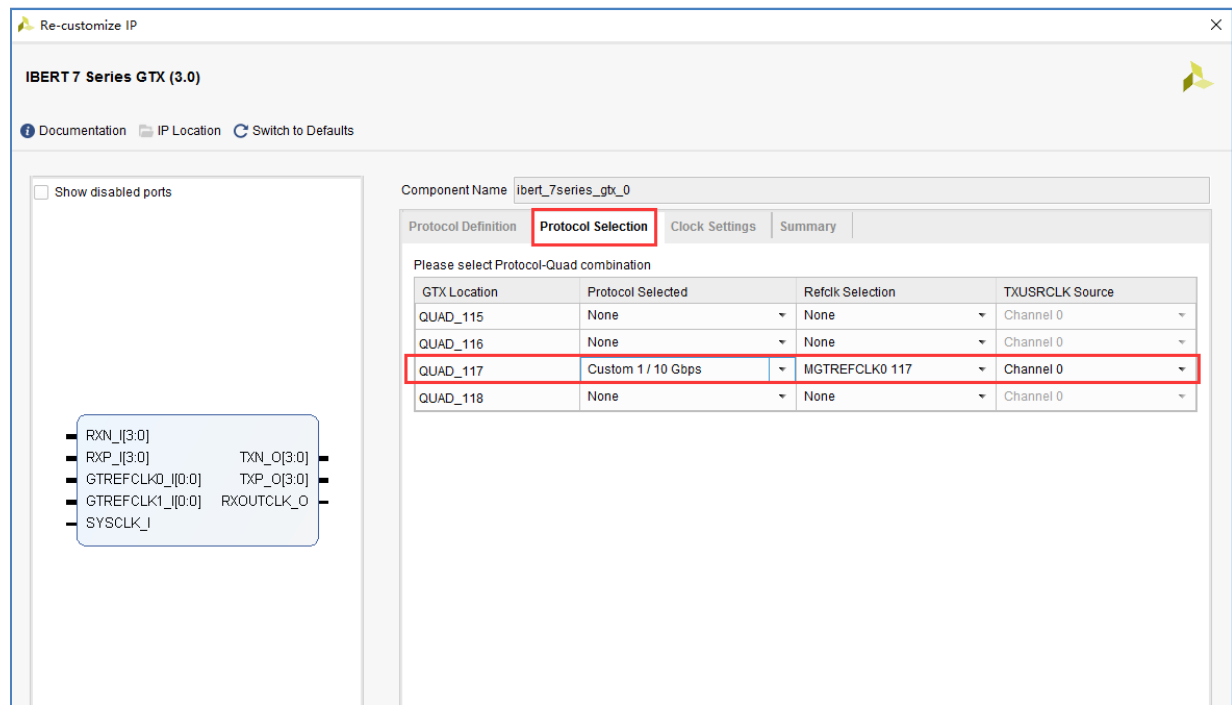
2. 在 Protocol Definition 页面中输入 LineRate 的速度,如果用户使用的是 10G 的光模块，这里可以选择最高的速率 10Gbps, 参考时钟为 125Mhz，LineRate 的频率为参考时钟的整数 80 倍。



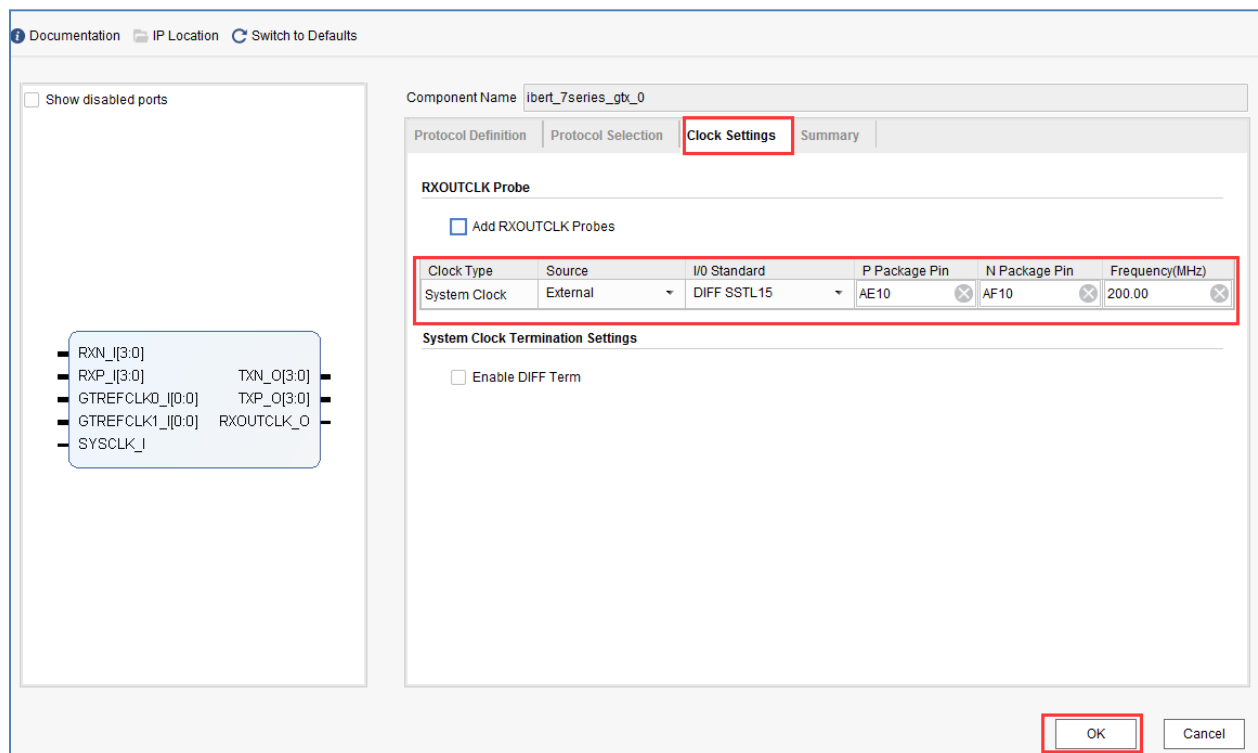
如果用户使用的是 1.25G 的光模块，这里的 LineRate 的速度选择为 1.25Gbps，LineRate 的频率为参考时钟的整数 10 倍。



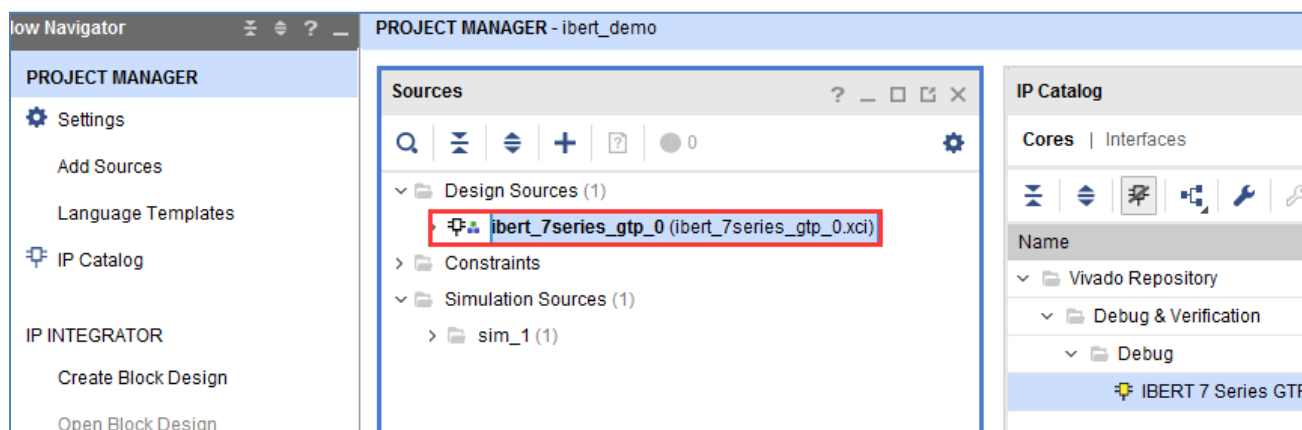
3. 在 Protocol Selection 界面里，选择 Protocol Selection 项为 Custom 1/10Gbps( 1.25G 光模块的时候则选择 Custom 1/1.25Gbps)。



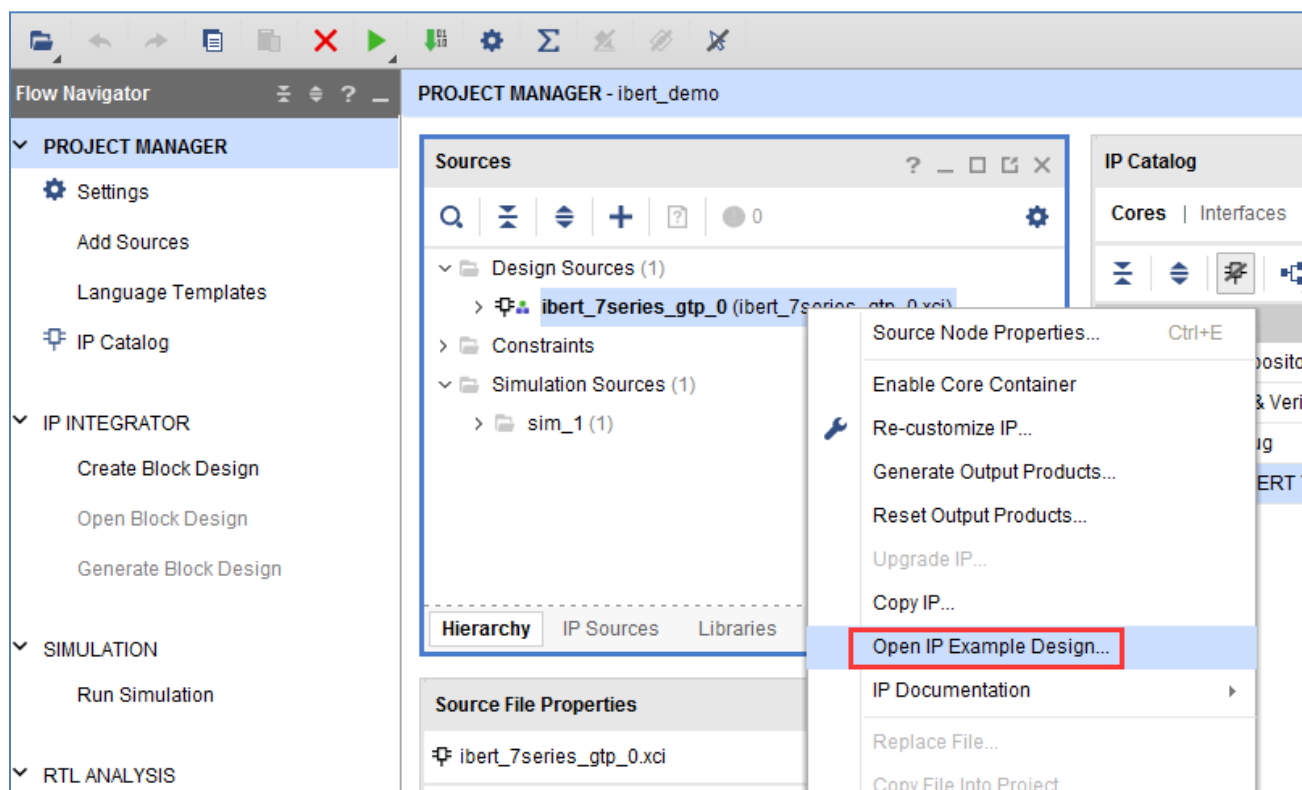
4. 在 Clock Settings 界面里，选择系统时钟的管脚。这里的管脚设定需要跟开发板一致。



5. 生成后的 ibert\_7series\_GTX IP 自动添加到项目中。

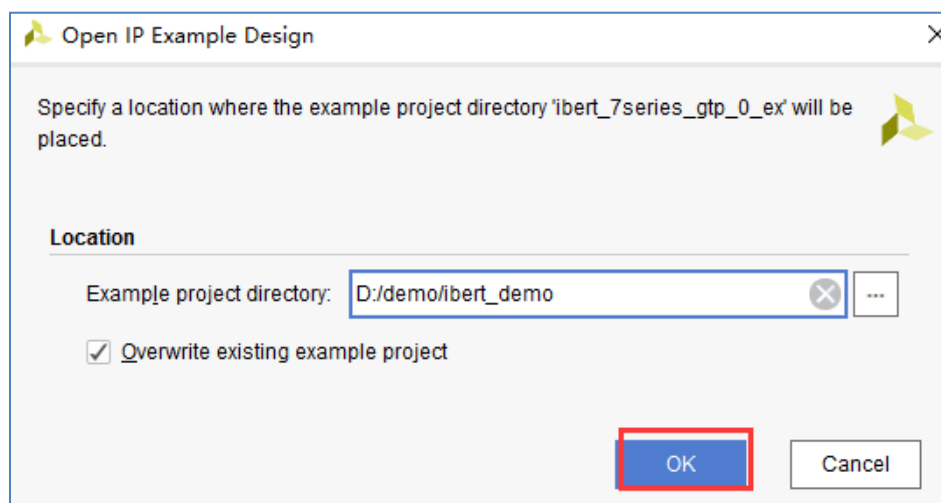


6. 右键选择 ibert\_7series\_GTX IP，在弹出的下拉框中选择 Open IP Example Design。

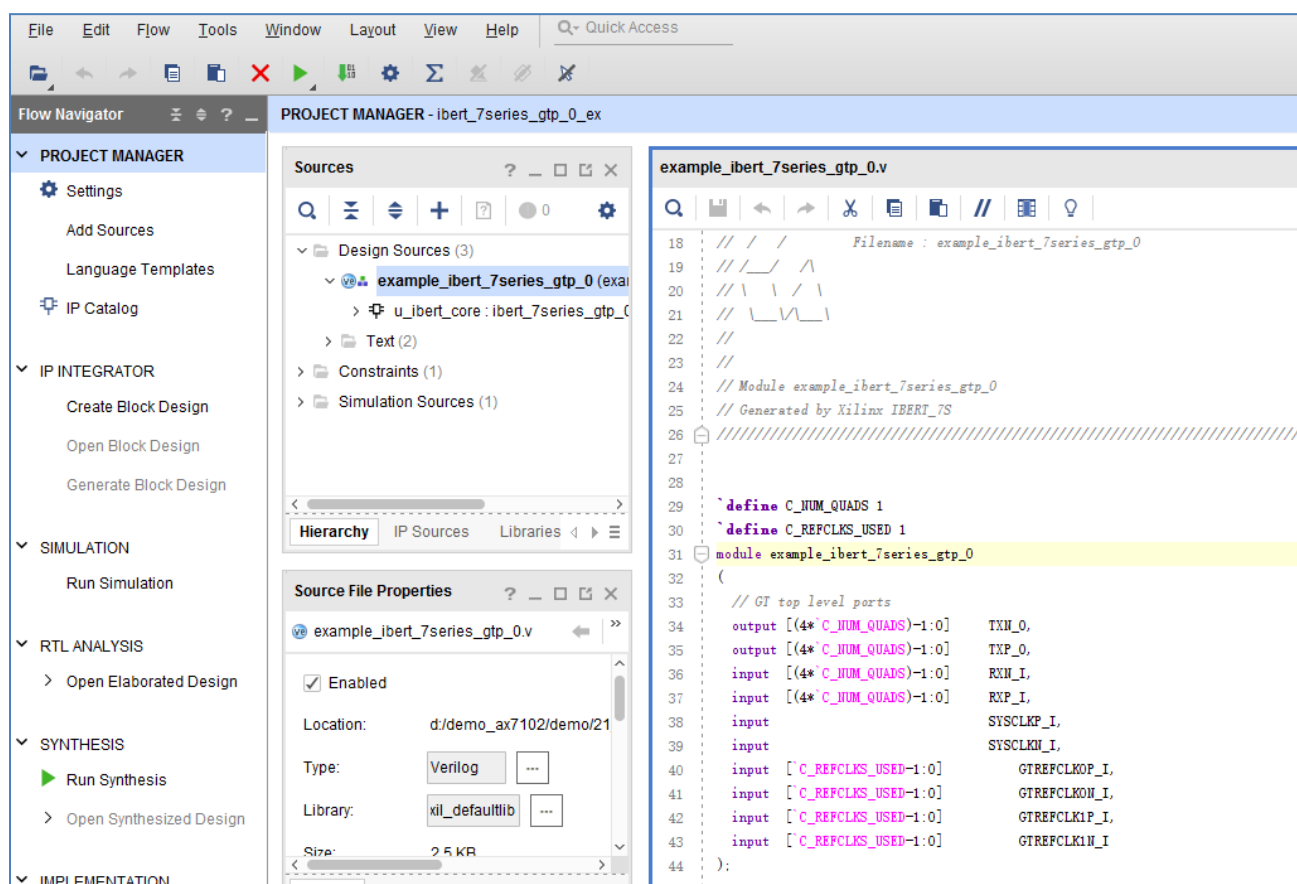


再选择 Example project 的放置目录。





7. 软件自动生成 example\_ibert\_7series\_GTX\_0 的新项目，在这个项目中 verilog 程序和管脚约束文件都已经配置好了。

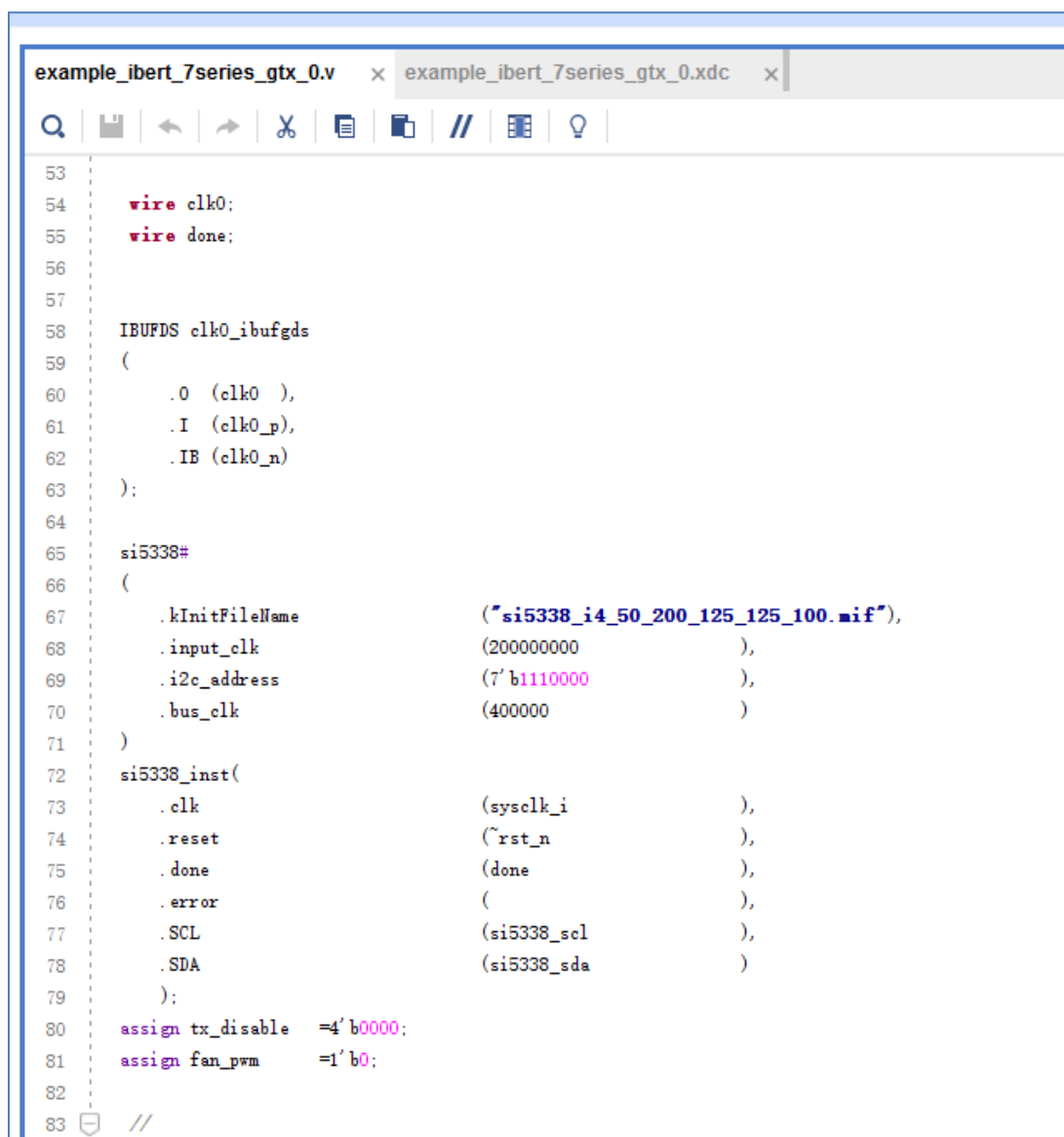


8. 因为在 AX7325 的硬件电路中，我们使用了 tx\_disable 信号来允许/禁止 SFP 光模块的发送，所以这里需要在 TOP 程序中定义 4 个 tx\_disable 信号并赋值为 0，一直使能 SFP 光模块的发送，同时由于 GTX 的参考时钟是由 SI5338 可编程时钟芯片产生的，需把相关接口和程序也添加进来。

```

17
18
19 `define C_NUM_QUADS 1
20 `define C_REFCLKS_USED 1
21 module example_ibert_7series_gtx_0
22 (
23     output [3:0]      tx_disable,
24     output            fan_pwm,
25     input             clk0_p,
26     input             clk0_n,
27     inout             si5338_scl, //i2c clock
28     inout             si5338_sda, //i2c data
29     input             rst_n,
30     // GT top level ports
31     output [(4*C_NUM_QUADS)-1:0] TXN_0,
32     output [(4*C_NUM_QUADS)-1:0] TXP_0,
33     input  [(4*C_NUM_QUADS)-1:0] RXN_I,
34     input  [(4*C_NUM_QUADS)-1:0] RXP_I,
35     input             SYSCLKP_I,
36     input             SYSCLKN_I,
37     input  [C_REFCLKS_USED-1:0] GTREFCLKOP_I,
38     input  [C_REFCLKS_USED-1:0] GTREFCLKON_I,
39     input  [C_REFCLKS_USED-1:0] GTREFCLK1P_I,
40     input  [C_REFCLKS_USED-1:0] GTREFCLK1N_I
41 );
42
43 //
44 // Ibert refclk internal signals

```



```
53
54     wire clk0;
55     wire done;
56
57
58     IBUFDS clk0_ibufgds
59     (
60         .O (clk0 ),
61         .I (clk0_p),
62         .IB (clk0_n)
63     );
64
65     si5338#
66     (
67         .kInitFileName      ("si5338_i4_50_200_125_125_100.mif"),
68         .input_clk          (200000000 ),
69         .i2c_address        (7'b1110000 ),
70         .bus_clk            (400000 )
71     )
72     si5338_inst(
73         .clk                (sysclk_i ),
74         .reset              (~rst_n ),
75         .done               (done ),
76         .error              ( ),
77         .SCL                (si5338_scl ),
78         .SDA                (si5338_sda )
79     );
80     assign tx_disable      =4'b0000;
81     assign fan_pwm        =1'b0;
82
83     //
```

9. 再在 xdc 文件中添加如下管脚的约束。

```

example_ibert_7series_gtx_0.v  x  example_ibert_7series_gtx_0.xdc *  x
🔍  📁  ⏪  ⏩  ✂  📄  📁  //  📄  💡

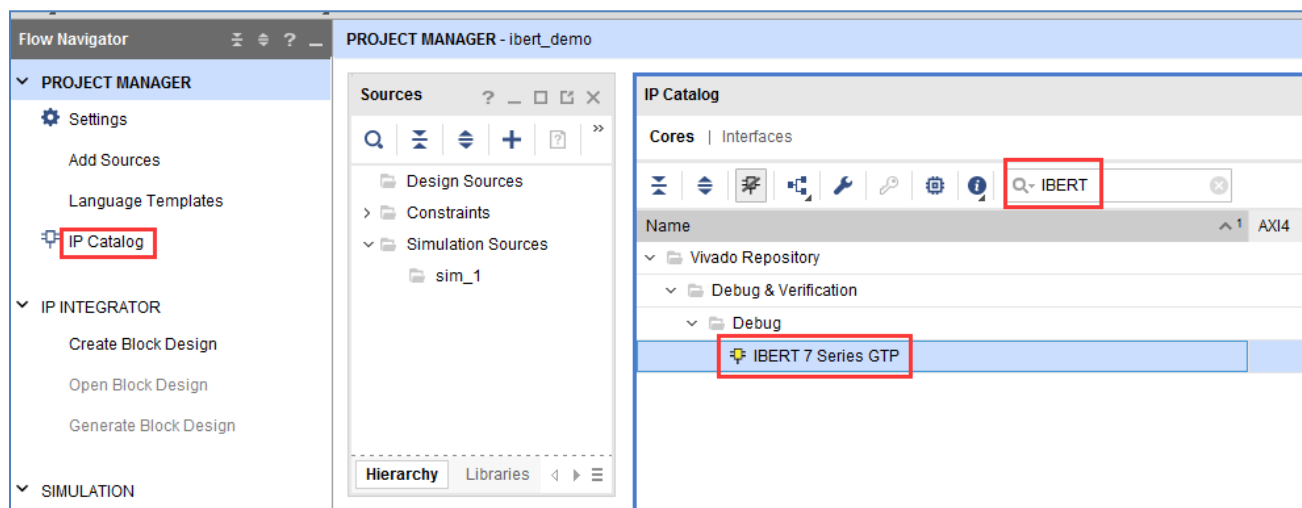
17  set_property PACKAGE_PIN AE10 [get_ports SYSCLKP_I]
18  set_property IOSTANDARD DIFF_SSTL15 [get_ports SYSCLKP_I]
19  set_property PACKAGE_PIN AF10 [get_ports SYSCLKN_I]
20  set_property IOSTANDARD DIFF_SSTL15 [get_ports SYSCLKN_I]
21  #####BPI Configure Setting#####
22  set_property BITSTREAM.CONFIG.CONFIGRATE 9 [current_design]
23  set_property CONFIG_VOLTAGE 3.3 [current_design]
24  set_property CFGBVS VCC0 [current_design]
25  set_property CONFIG_MODE BPI16 [current_design]
26  #####si5338 Setting#####
27  set_property IOSTANDARD DIFF_SSTL15 [get_ports clk0_p]
28  set_property IOSTANDARD DIFF_SSTL15 [get_ports clk0_n]
29  set_property PACKAGE_PIN F20 [get_ports clk0_p]
30  set_property PACKAGE_PIN E20 [get_ports clk0_n]
31  set_property IOSTANDARD LVCMOS33 [get_ports si5338_scl]
32  set_property PACKAGE_PIN P23 [get_ports si5338_scl]
33  set_property IOSTANDARD LVCMOS33 [get_ports si5338_sda]
34  set_property PACKAGE_PIN M25 [get_ports si5338_sda]
35  ##### key define#####
36  set_property PACKAGE_PIN AG27 [get_ports rst_n]
37  set_property IOSTANDARD LVCMOS33 [get_ports rst_n]
38  ##### fan define#####
39  set_property IOSTANDARD LVCMOS33 [get_ports fan_pwm]
40  set_property PACKAGE_PIN AE26 [get_ports fan_pwm]
41  #####
42  set_property IOSTANDARD LVCMOS33 [get_ports {tx_disable[3]}]
43  set_property IOSTANDARD LVCMOS33 [get_ports {tx_disable[2]}]
44  set_property IOSTANDARD LVCMOS33 [get_ports {tx_disable[1]}]
45  set_property IOSTANDARD LVCMOS33 [get_ports {tx_disable[0]}]
46  set_property PACKAGE_PIN T28 [get_ports {tx_disable[0]}]
47  set_property PACKAGE_PIN T27 [get_ports {tx_disable[1]}]
48  set_property PACKAGE_PIN U28 [get_ports {tx_disable[2]}]
49  set_property PACKAGE_PIN U25 [get_ports {tx_disable[3]}]

```

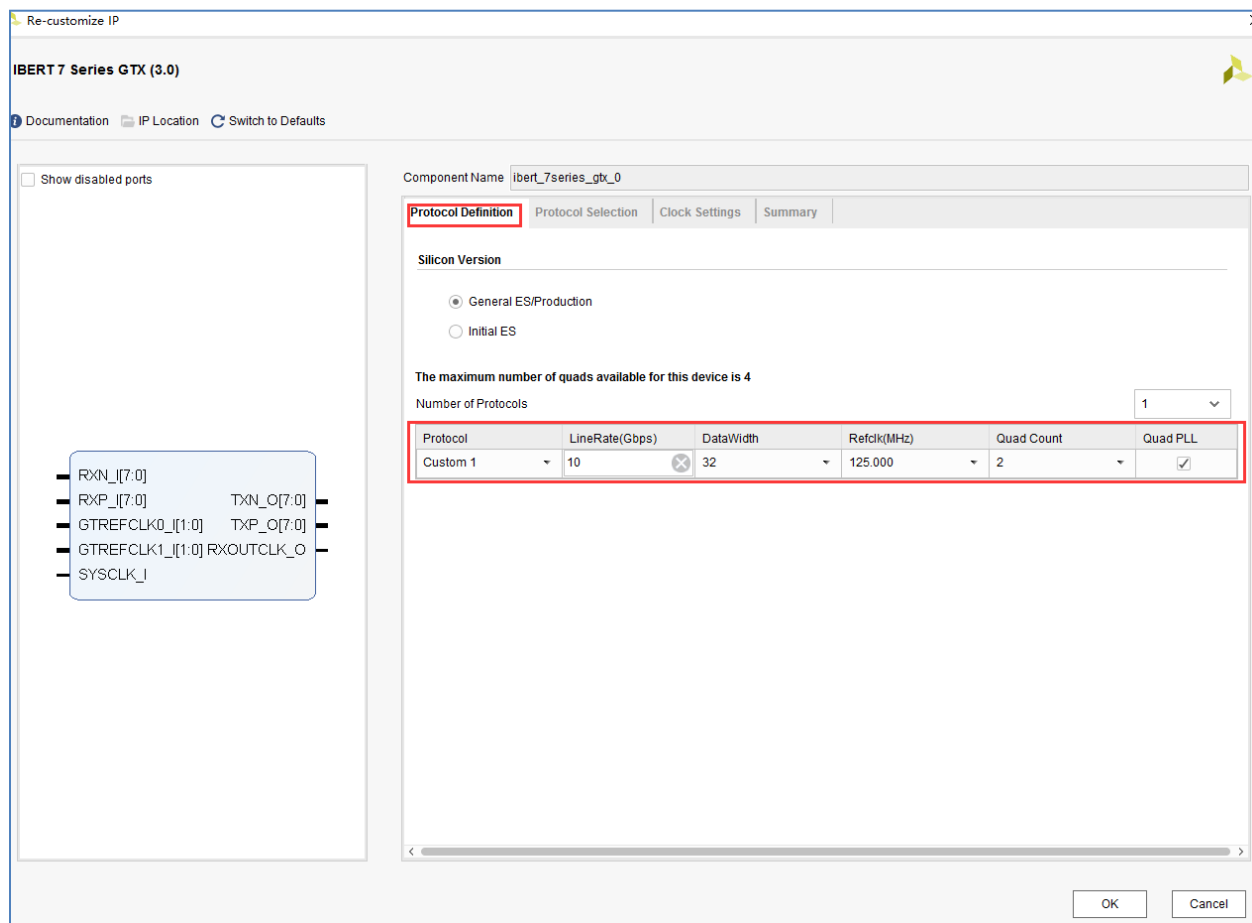
GTX 测试的项目设计完成，保存工程并编译工程生成测试的.bit 文件。

## 3.2 QSFP 与 4 路 SFP 通信测试程序

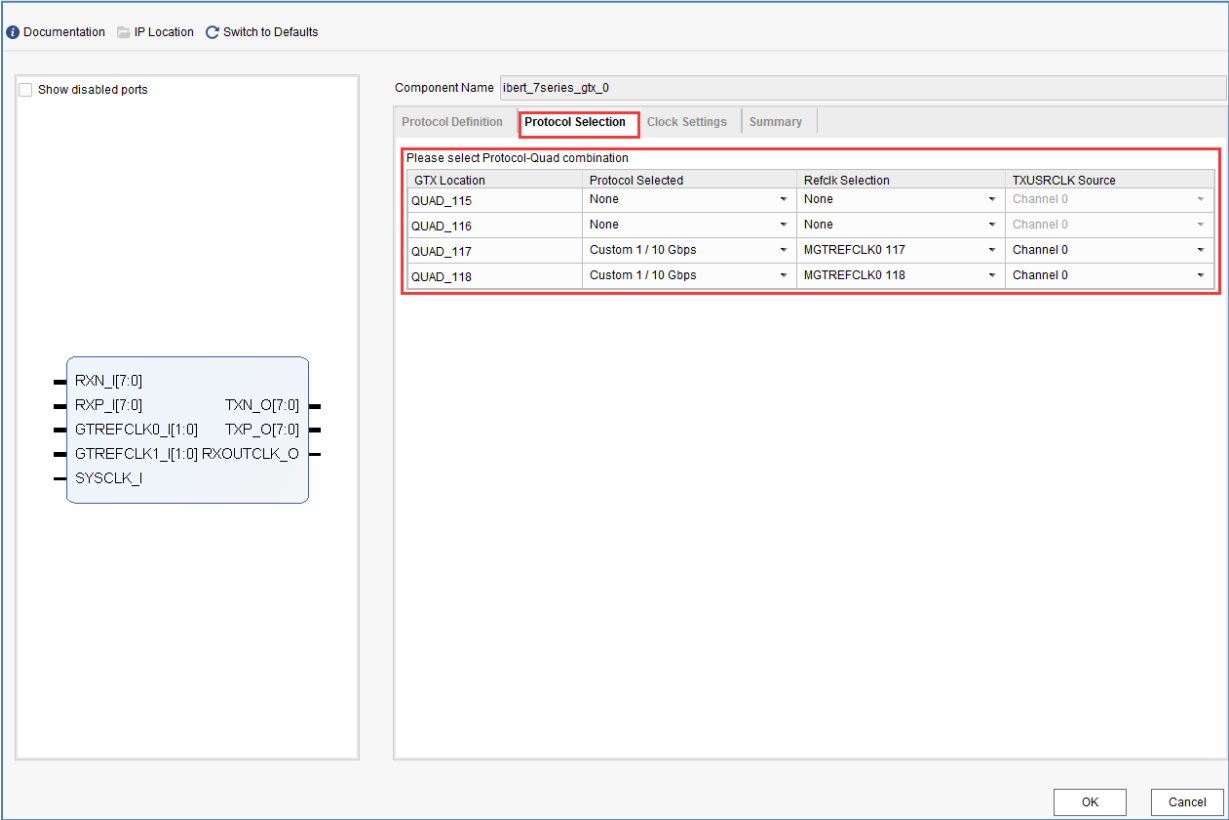
1. 新建一个工程 ibert\_demo , 在 IP Catalog 界面中双击 Debug 目录下的 IBERT 7 Series GTX IP。



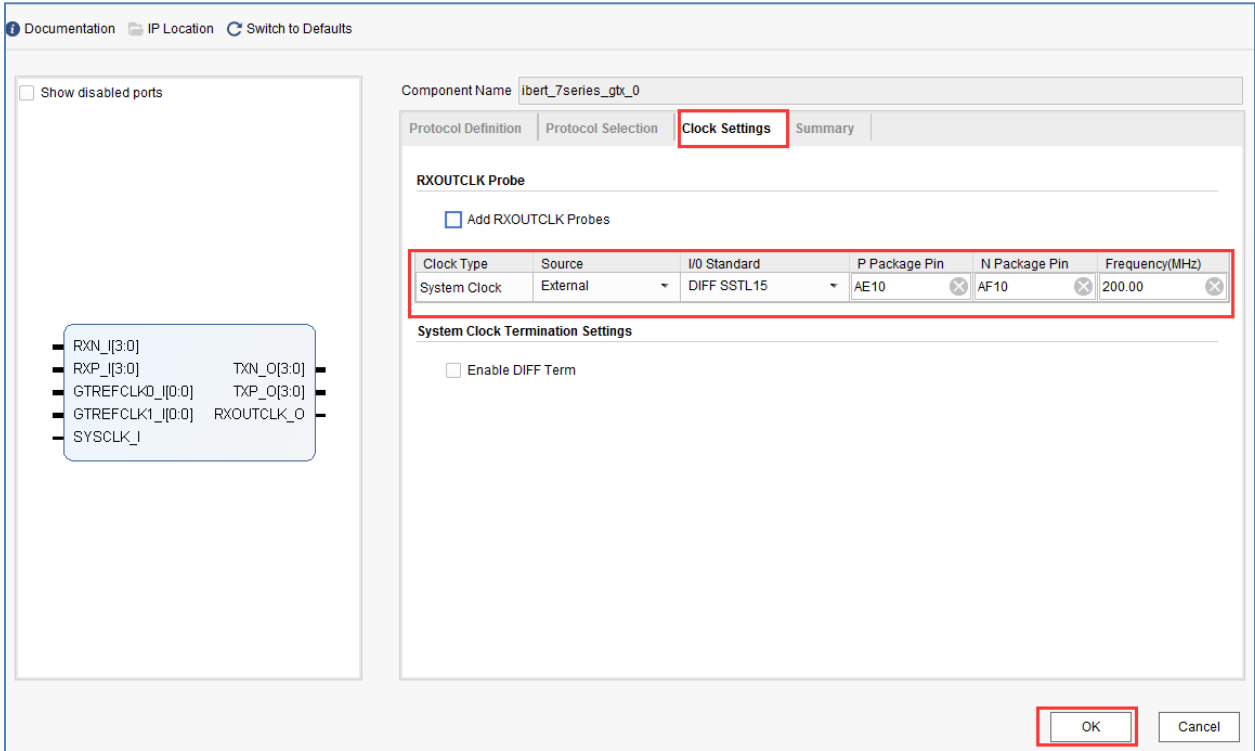
2. 在 Protocol Definition 页面中输入 LineRate 的速度,如果用户使用的是 10G 的光模块, 这里可以选择最高的速率 10Gbps, 参考时钟为 125Mhz, LineRate 的频率为参考时钟的整数 80 倍。



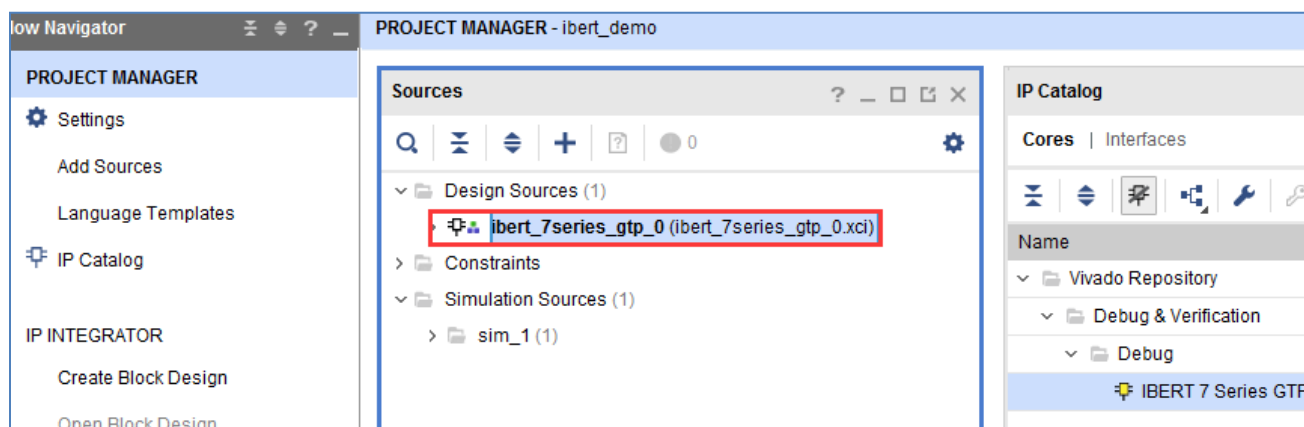
3. 在 Protocol Selection 界面里, 选择 Protocol Selection 项为 Custom 1/10Gbps。



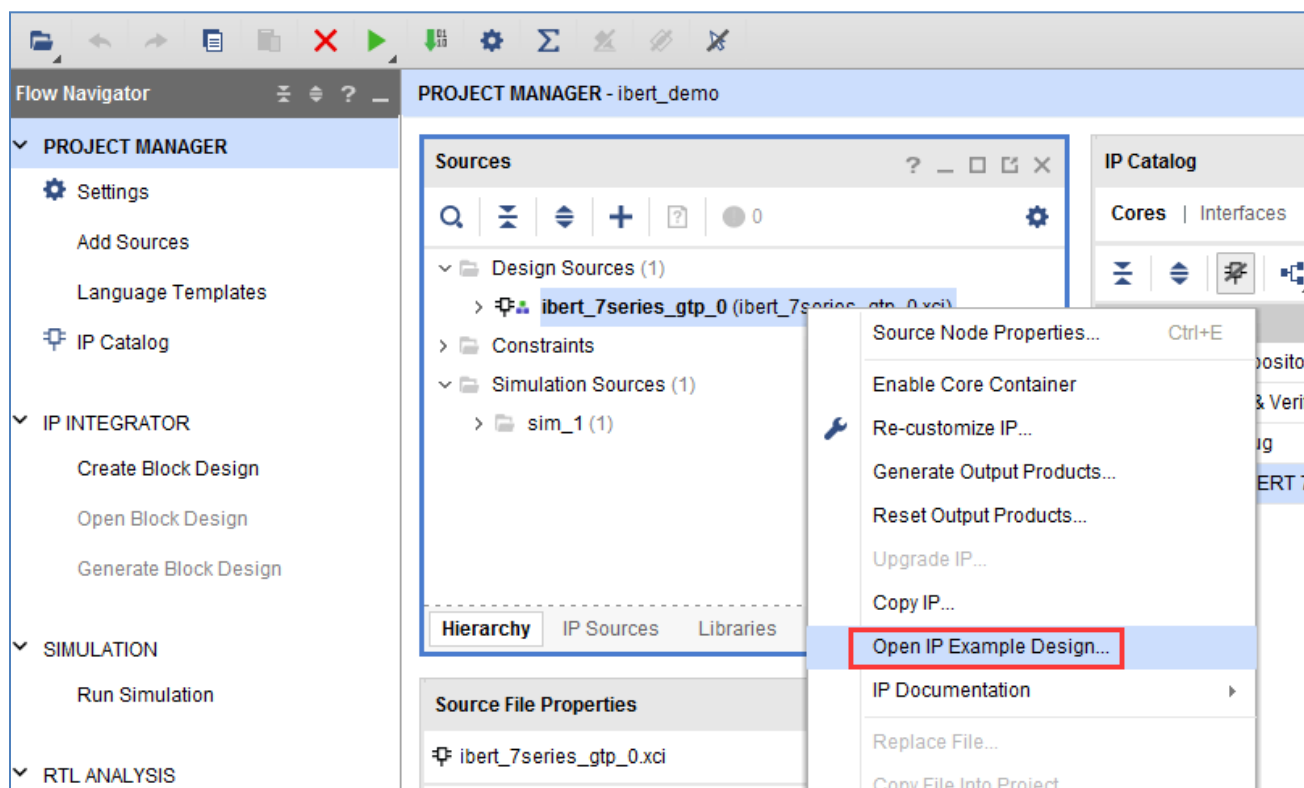
4. 在 Clock Settings 界面里，选择系统时钟的管脚。这里的管脚设定需要跟开发板一致。



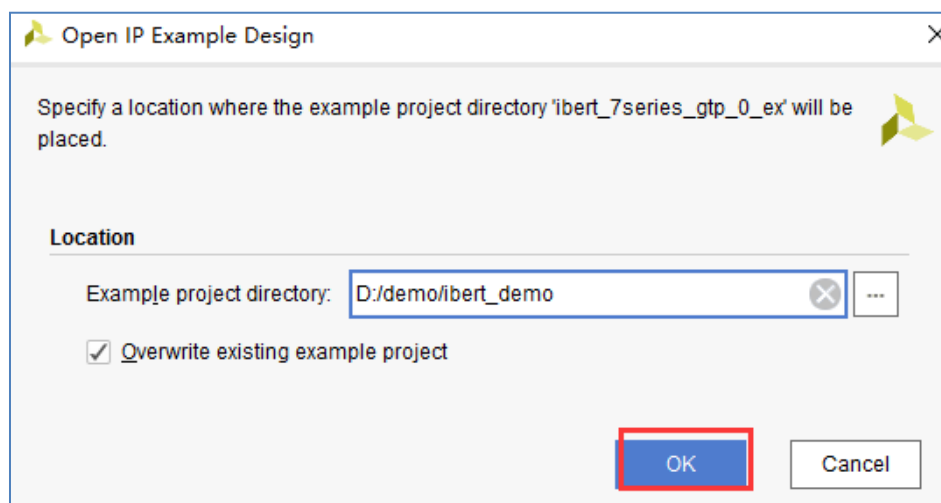
5. 生成后的 `ibert_7series_GTX` IP 自动添加到项目中。



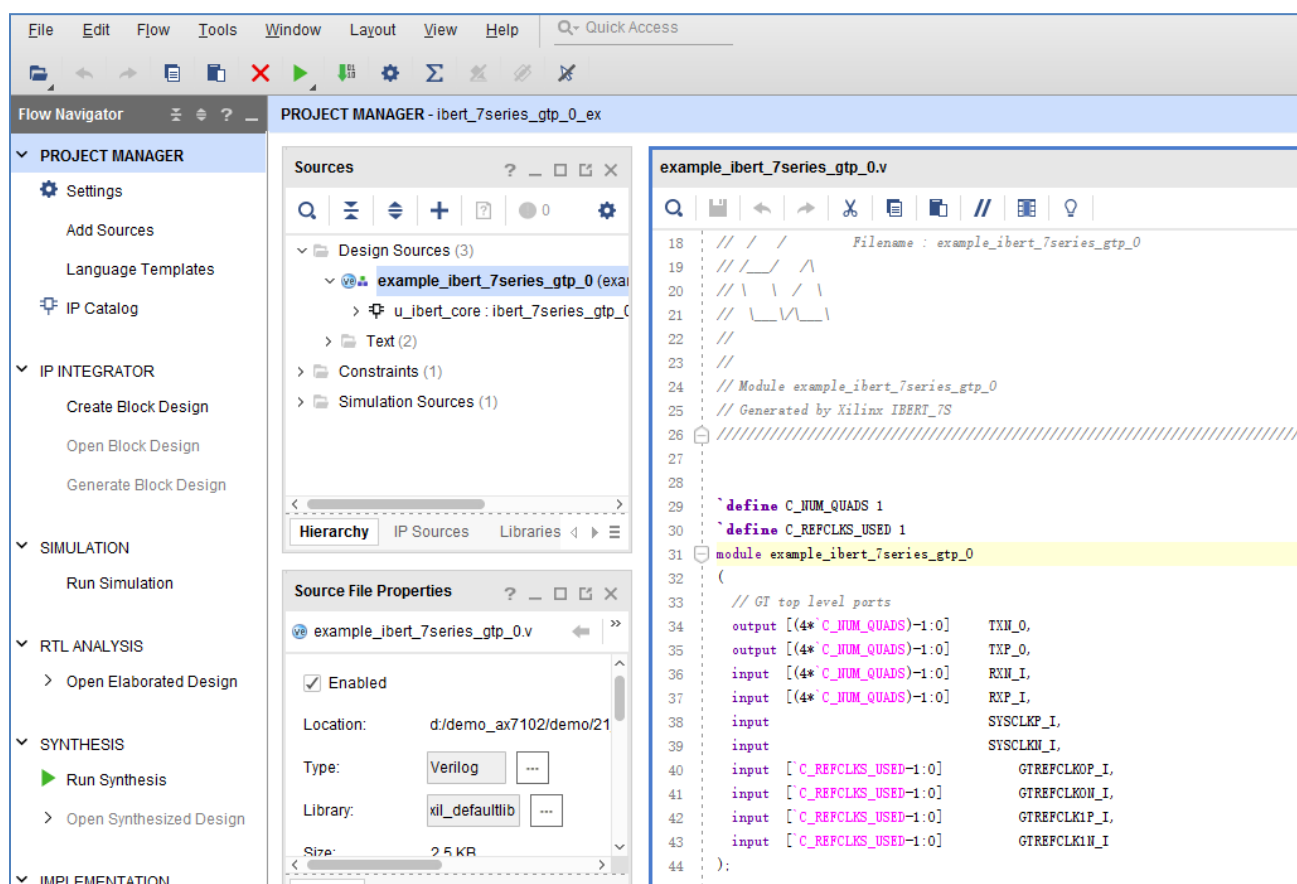
6. 右键选择 ibert\_7series\_GTX IP，在弹出的下拉框中选择 Open IP Example Design。



再选择 Example project 的放置目录。

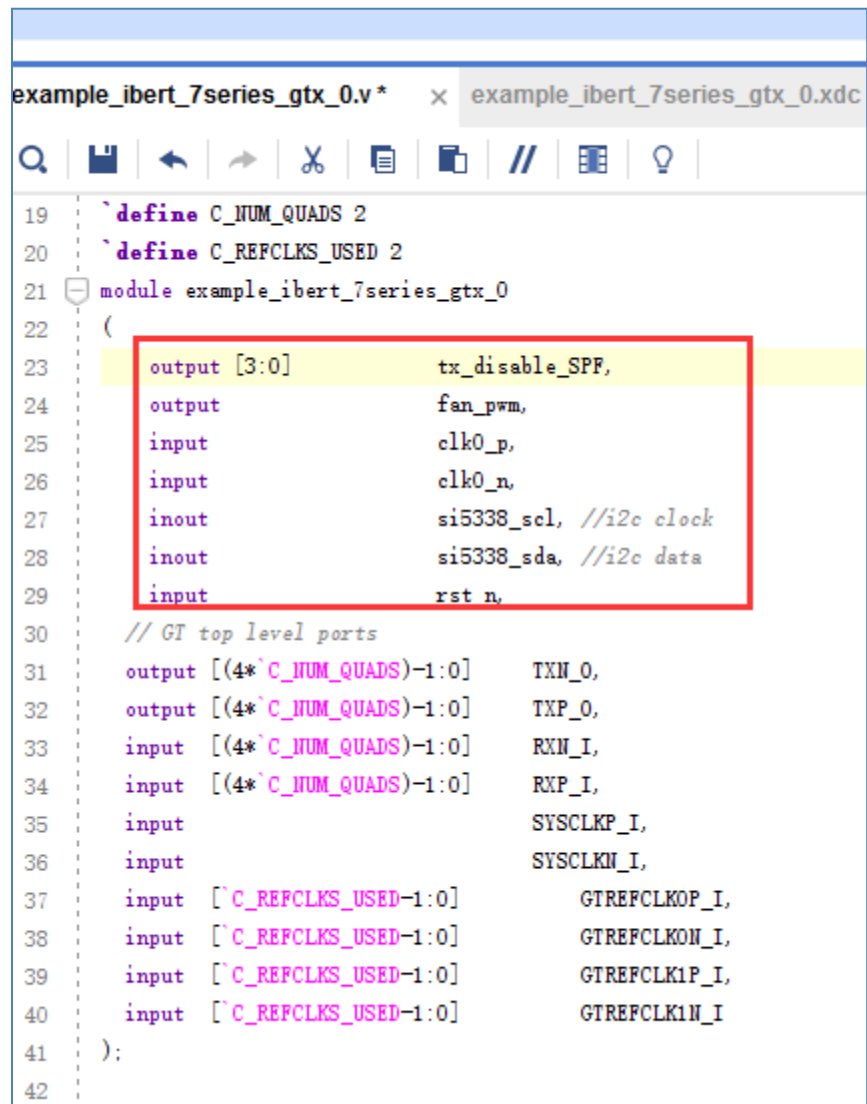


7. 软件自动生成 example\_ibert\_7series\_GTX\_0 的新项目，在这个项目中 verilog 程序和管脚约束文件都已经配置好了。

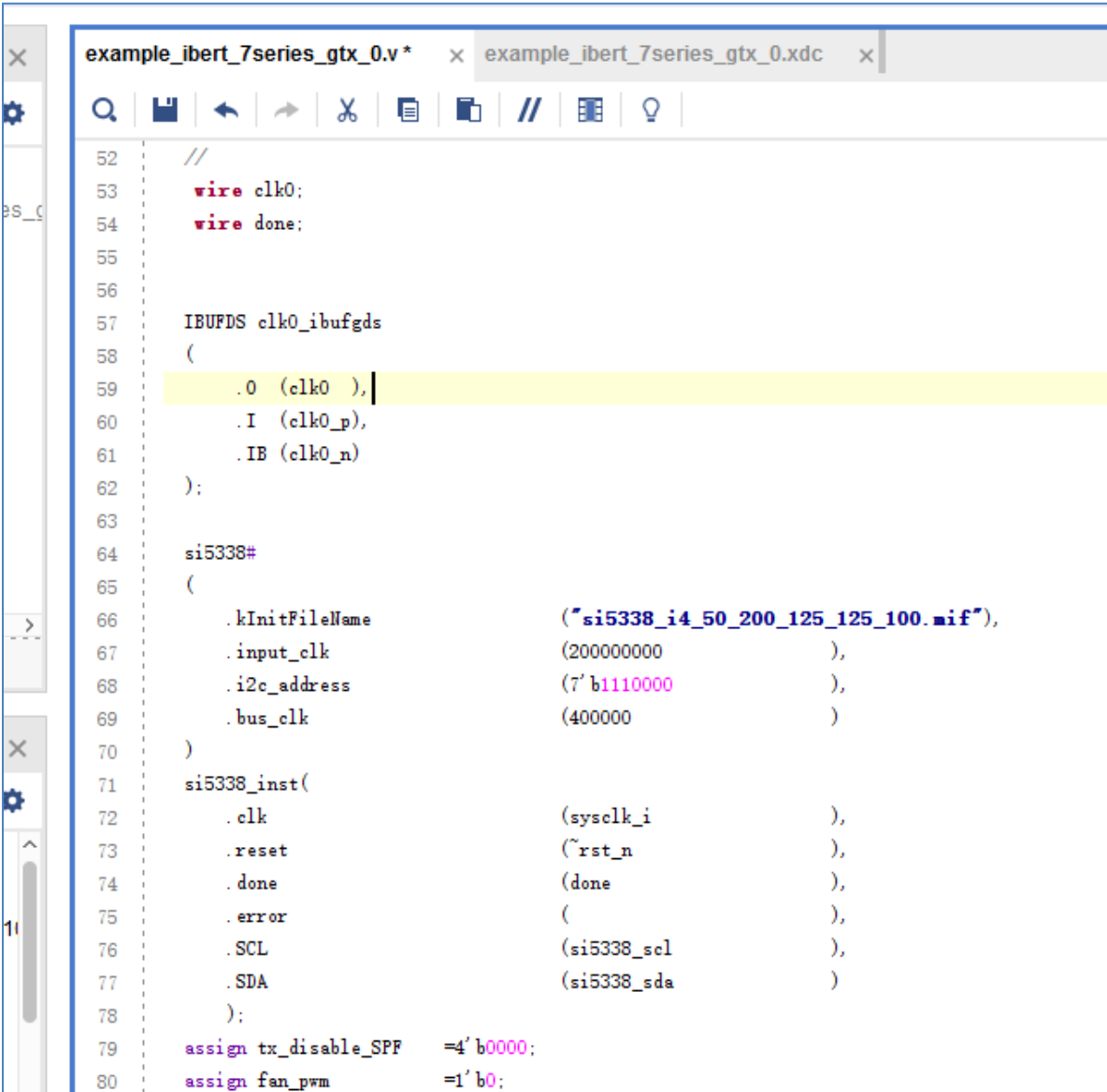


8. 因为在 AX7325 的硬件电路中，我们使用了 tx\_disable 信号来允许/禁止 SFP 光模块的发送，所以这里需要在 TOP 程序中定义 4 个 tx\_disable 信号并赋值为 0，一直使能 SFP 光模块的发送，同时由于 GTX 的参考时钟是由 SI5338 可编程时钟芯片产生的，需把相关接口和程序也添加进来。



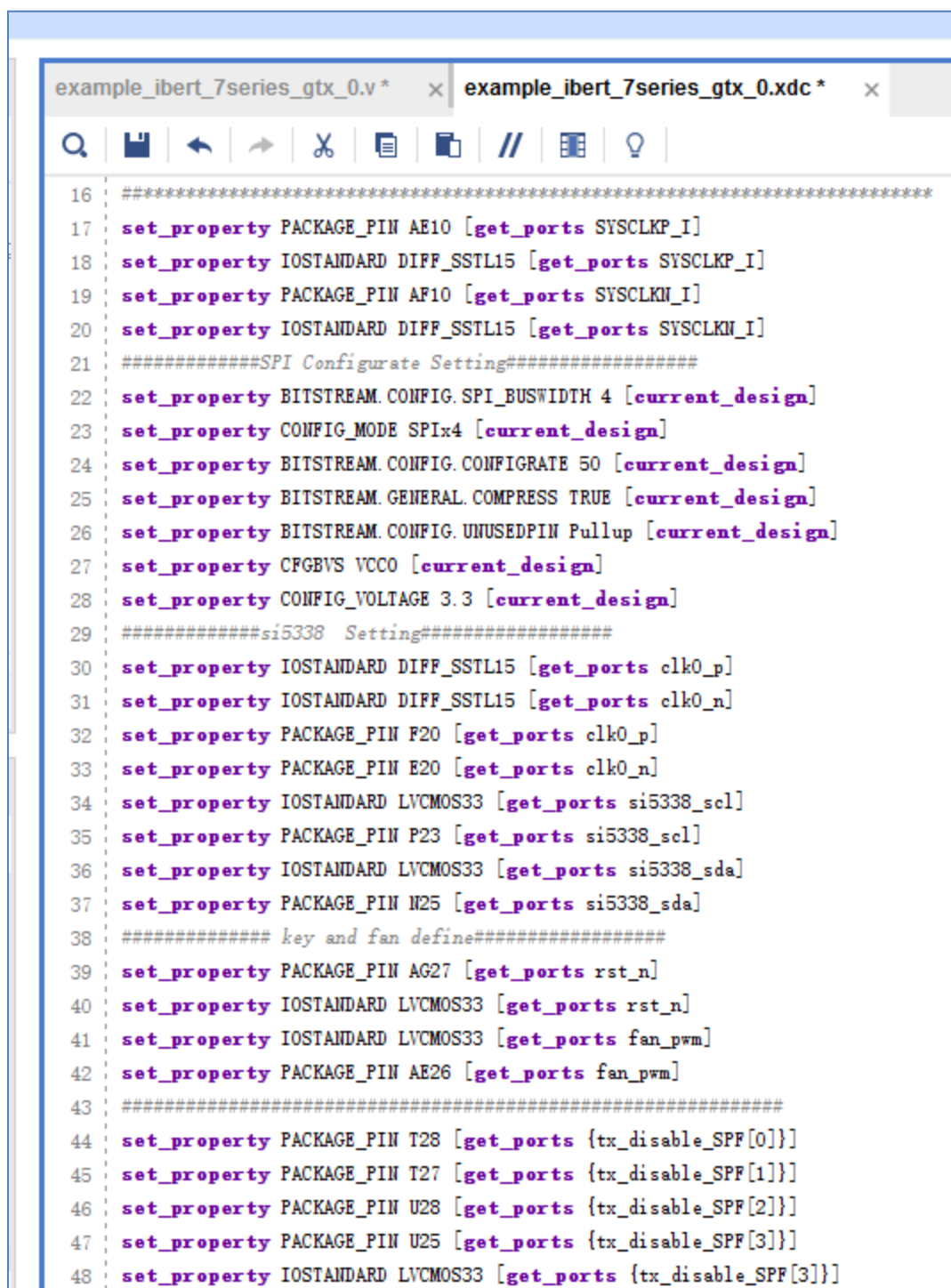


```
example_ibert_7series_gtx_0.v* x example_ibert_7series_gtx_0.xdc
Q [icon] [icon] [icon] [icon] [icon] [icon] [icon] [icon] [icon]
19 `define C_NUM_QUADS 2
20 `define C_REFCLKS_USED 2
21 module example_ibert_7series_gtx_0
22 (
23     output [3:0]      tx_disable_SPF,
24     output            fan_pwm,
25     input             clk0_p,
26     input             clk0_n,
27     inout             si5338_scl, //i2c clock
28     inout             si5338_sda, //i2c data
29     input             rst_n,
30     // GT top level ports
31     output [(4*C_NUM_QUADS)-1:0] TXN_0,
32     output [(4*C_NUM_QUADS)-1:0] TXP_0,
33     input  [(4*C_NUM_QUADS)-1:0] RXN_I,
34     input  [(4*C_NUM_QUADS)-1:0] RXP_I,
35     input                      SYSCLKP_I,
36     input                      SYSCLKN_I,
37     input  [C_REFCLKS_USED-1:0] GTREFCLKOP_I,
38     input  [C_REFCLKS_USED-1:0] GTREFCLKON_I,
39     input  [C_REFCLKS_USED-1:0] GTREFCLK1P_I,
40     input  [C_REFCLKS_USED-1:0] GTREFCLK1N_I
41 );
42
```



```
52 //
53 wire clk0;
54 wire done;
55
56
57 IBUFDS clk0_ibufgds
58 (
59     .O (clk0 ),
60     .I (clk0_p),
61     .IB (clk0_n)
62 );
63
64 si5338#
65 (
66     .kInitFileName      ("si5338_i4_50_200_125_125_100.mif"),
67     .input_clk          (200000000),
68     .i2c_address        (7'b1110000),
69     .bus_clk            (400000)
70 )
71 si5338_inst(
72     .clk                (sysclk_i),
73     .reset              (~rst_n),
74     .done               (done),
75     .error              ( ),
76     .SCL                (si5338_scl),
77     .SDA                (si5338_sda)
78 );
79 assign tx_disable_SPF  =4'b0000;
80 assign fan_pwm        =1'b0;
```

9. 再在 xdc 文件中添加如下管脚的约束，见工程：



```

16 #####
17 set_property PACKAGE_PIN AE10 [get_ports SYSCLKP_I]
18 set_property IOSTANDARD DIFF_SSTL15 [get_ports SYSCLKP_I]
19 set_property PACKAGE_PIN AF10 [get_ports SYSCLKN_I]
20 set_property IOSTANDARD DIFF_SSTL15 [get_ports SYSCLKN_I]
21 #####SPI Configure Setting#####
22 set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
23 set_property CONFIG_MODE SPIx4 [current_design]
24 set_property BITSTREAM.CONFIG.CONFIGRATE 50 [current_design]
25 set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
26 set_property BITSTREAM.CONFIG.UNUSEDPIN Pullup [current_design]
27 set_property CFGBVS VCC0 [current_design]
28 set_property CONFIG_VOLTAGE 3.3 [current_design]
29 #####si5338 Setting#####
30 set_property IOSTANDARD DIFF_SSTL15 [get_ports clk0_p]
31 set_property IOSTANDARD DIFF_SSTL15 [get_ports clk0_n]
32 set_property PACKAGE_PIN F20 [get_ports clk0_p]
33 set_property PACKAGE_PIN E20 [get_ports clk0_n]
34 set_property IOSTANDARD LVCMOS33 [get_ports si5338_scl]
35 set_property PACKAGE_PIN P23 [get_ports si5338_scl]
36 set_property IOSTANDARD LVCMOS33 [get_ports si5338_sda]
37 set_property PACKAGE_PIN M25 [get_ports si5338_sda]
38 ##### key and fan define#####
39 set_property PACKAGE_PIN AG27 [get_ports rst_n]
40 set_property IOSTANDARD LVCMOS33 [get_ports rst_n]
41 set_property IOSTANDARD LVCMOS33 [get_ports fan_pwm]
42 set_property PACKAGE_PIN AE26 [get_ports fan_pwm]
43 #####
44 set_property PACKAGE_PIN T28 [get_ports {tx_disable_SPF[0]}]
45 set_property PACKAGE_PIN T27 [get_ports {tx_disable_SPF[1]}]
46 set_property PACKAGE_PIN U28 [get_ports {tx_disable_SPF[2]}]
47 set_property PACKAGE_PIN U25 [get_ports {tx_disable_SPF[3]}]
48 set_property IOSTANDARD LVCMOS33 [get_ports {tx_disable_SPF[3]}]

```

GTX 测试的项目设计完成，保存工程并编译工程生成测试的.bit 文件。

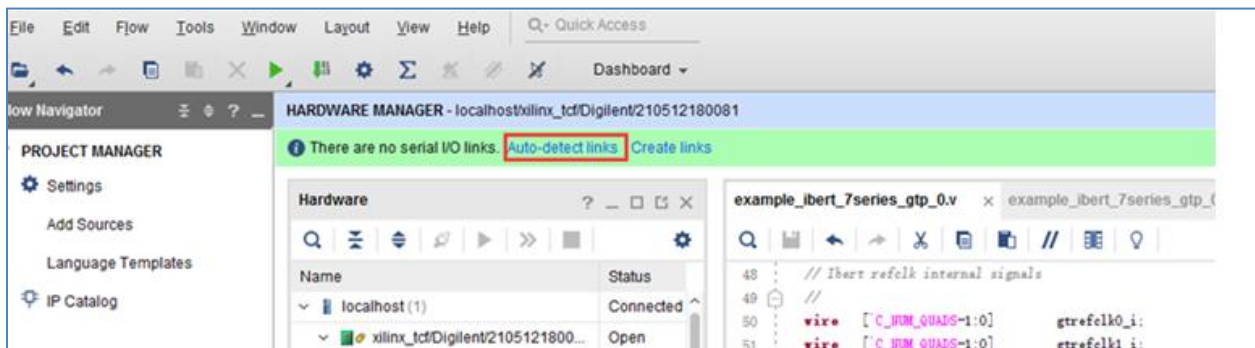
## 4 GTX 的眼图测试现象

因为 AX7325 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。

### 4.1 4 路 SFP 眼图测试现象

测试之前我们把 SFP 的光模块分别插到四个光模块的接口上，再用光纤把光模块 OPT1 和 OPT2 对连起来，把光模块 OPT3 和光模块 OPT4 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，这样这个光模块 RX 需要跟另一个光模块的 TX 相连，TX 需要连接到另一个光模块的 RX。

在 Vivado 软件里下载 .bit 文件到 FPGA，下载后选择 Auto-detect links 软件会自动检测 Serial I/O Links。



在 Serial I/O Links 界面会出现 4 路数据通信的情况，下图为 5 Gbps 连接速度的界面。

Serial I/O Links										
Name	TX	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	RX Pattern	TX Pre-Cursor
Ungrouped Links (4)										
Found 3	MGT_X0Y10/TX	MGT_X0Y11/RX	10,000 Gbps	1.21E12	0E0	8.264E...	Reset	PRBS 7-bit	PRBS 7-bit	1.67 dB (00111)
Found 2	MGT_X0Y11/TX	MGT_X0Y10/RX	10,000 Gbps	1.21E12	0E0	8.263E...	Reset	PRBS 7-bit	PRBS 7-bit	1.67 dB (00111)
Found 1	MGT_X0Y8/TX	MGT_X0Y9/RX	10,000 Gbps	1.21E12	0E0	8.263E...	Reset	PRBS 7-bit	PRBS 7-bit	1.67 dB (00111)
Found 0	MGT_X0Y9/TX	MGT_X0Y8/RX	10,000 Gbps	1.21E12	0E0	8.263E...	Reset	PRBS 7-bit	PRBS 7-bit	1.67 dB (00111)

对这个界面，我们这里简单介绍一下，比如第一个 Found 0 Link，它由 MGT\_X0Y0 通道的 TX 发送数据，由 MGT\_X0Y8 通道的 RX 接收，数据 Link 的速度为 10Gbps。Bits 这列为发送的数据量

(比如 1.21E-12 就是发送了  $1.21 \times 10^{12}$  的 12 次方个数据)，这个值随着时间的增加会不断增加。Error 项为错误的数据，这里我们看到的是 0，说明没有数据接收错误。BER 为误码率，Errors 的数量除以传输的数据数量就等于 BER 误码率。BERT Reset 是复位统计的数据，重新计数。TX Pattern 为发送的测试数据，默认为 PRBS 7bit，这里我们也可以选择其它的测试数据来测试光纤数据的传输。

可能大家还不太清楚 MGT\_X0Y8~MGT\_X0Y11 各自代表那个光纤通道，在 .xdc 文件里，有如下的定义：

```

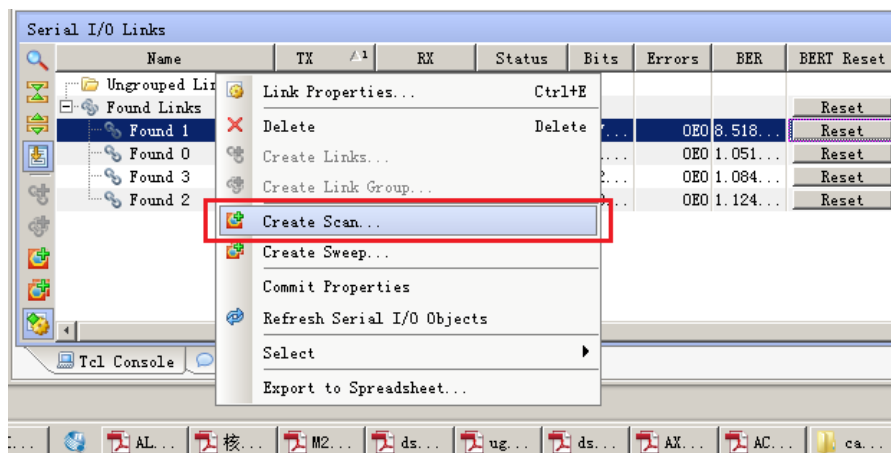
104 ##
105 ## GTXE2 Channel and Common Loc constraints
106 ##
107 set_property LOC GTXE2_CHANNEL_X0Y8 [get_cells u_ibert_core/inst/QUAD[0].u_q/CH[0].u_ch/u_gtxe2_channel]
108 set_property LOC GTXE2_CHANNEL_X0Y9 [get_cells u_ibert_core/inst/QUAD[0].u_q/CH[1].u_ch/u_gtxe2_channel]
109 set_property LOC GTXE2_CHANNEL_X0Y10 [get_cells u_ibert_core/inst/QUAD[0].u_q/CH[2].u_ch/u_gtxe2_channel]
110 set_property LOC GTXE2_CHANNEL_X0Y11 [get_cells u_ibert_core/inst/QUAD[0].u_q/CH[3].u_ch/u_gtxe2_channel]
111 set_property LOC GTXE2_COMMON_X0Y2 [get_cells u_ibert_core/inst/QUAD[0].u_q/u_common/u_gtxe2_common]

```

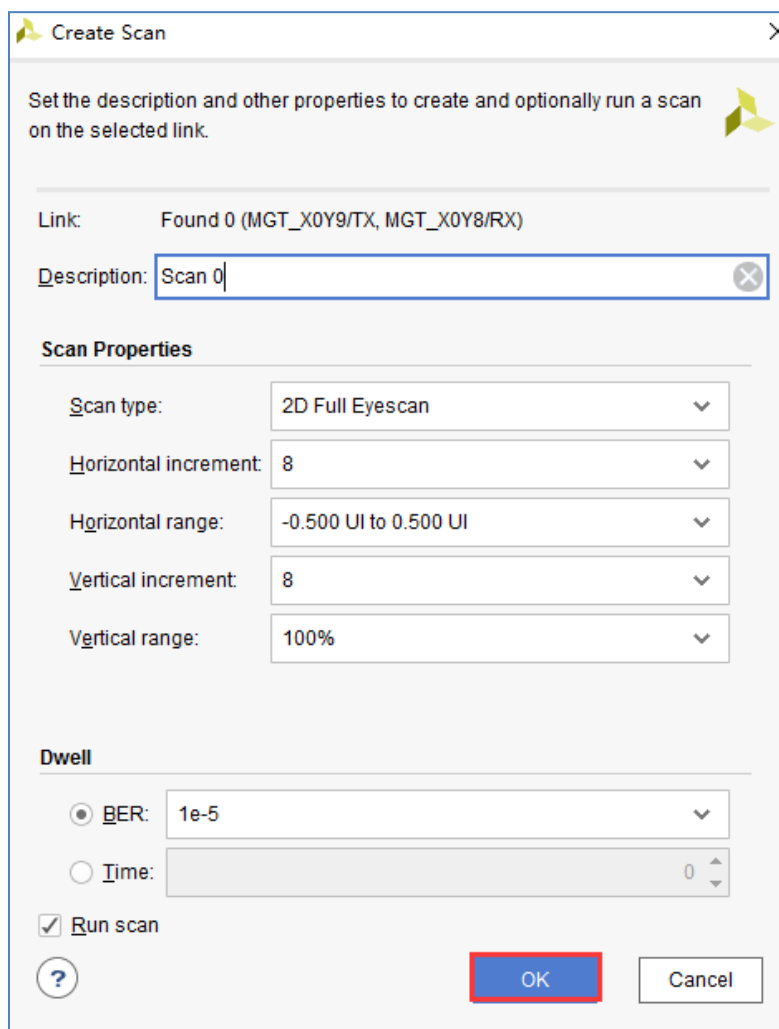
说明 MGT\_X0Y8~11 分别对应的是 GTX 的 Channel0~3，在 AX7325 开发板上，GTX 的 Channel0 是连接到了 OPT1，Channel1 是连接到 OPT2，Channel2 是连接到 OPT3，Channel3 是连接到 OPT4。所以 MGT\_X0Y8~11 和开发板上的 OPT 光模块之间对应关系如下表所示：

MGT_X0Yx	Channel	OPT 光模块
MGT_X0Y8	Channel0	OPT1
MGT_X0Y9	Channel1	OPT2
MGT_X0Y10	Channel2	OPT3
MGT_X0Y11	Channel3	OPT4

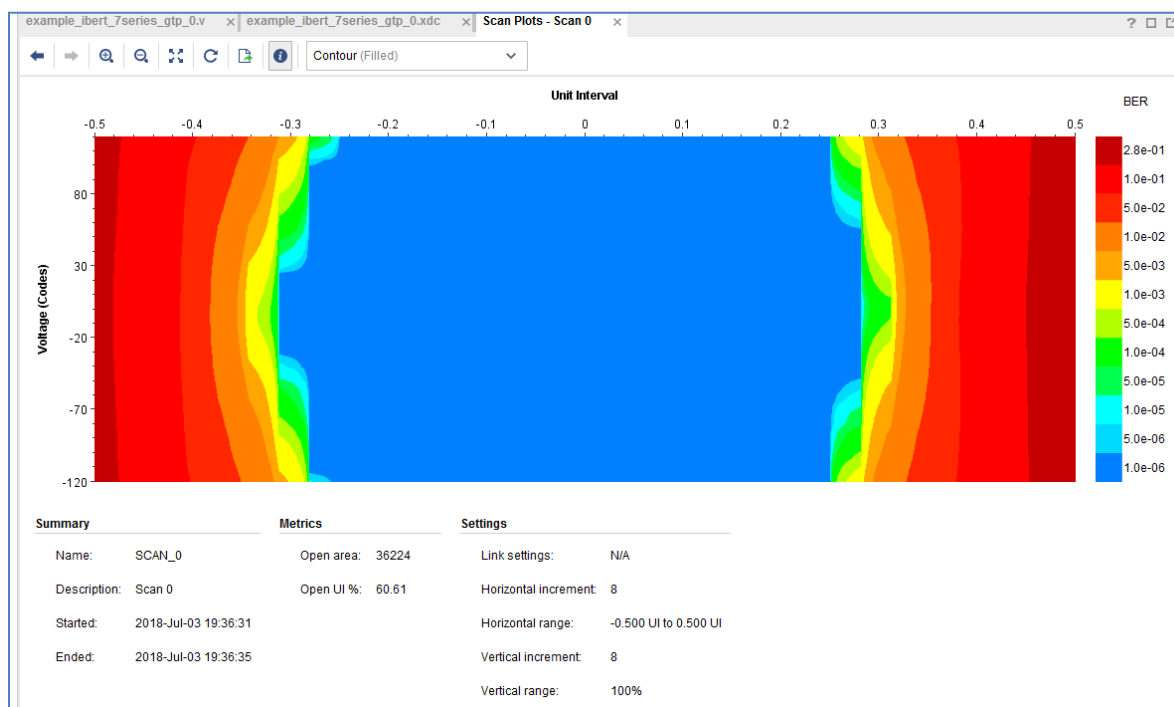
接下去我们来测试光纤通信的电眼图，一般情况，测试板上的电眼图需要配合高端示波器和差分探头才能测量。但这里我们不需要外接任何设备或仪器，就可以测量光纤数据通信的眼图情况，极大的方便了 FPGA 高速串行通信的软硬件调试。右键点击我们想看的通道，比如我们这里选择第一路 Found 1(MGT\_X0Y9 发，MGT\_X0Y8 接收)，再在下拉菜单里选择 Create Scan...



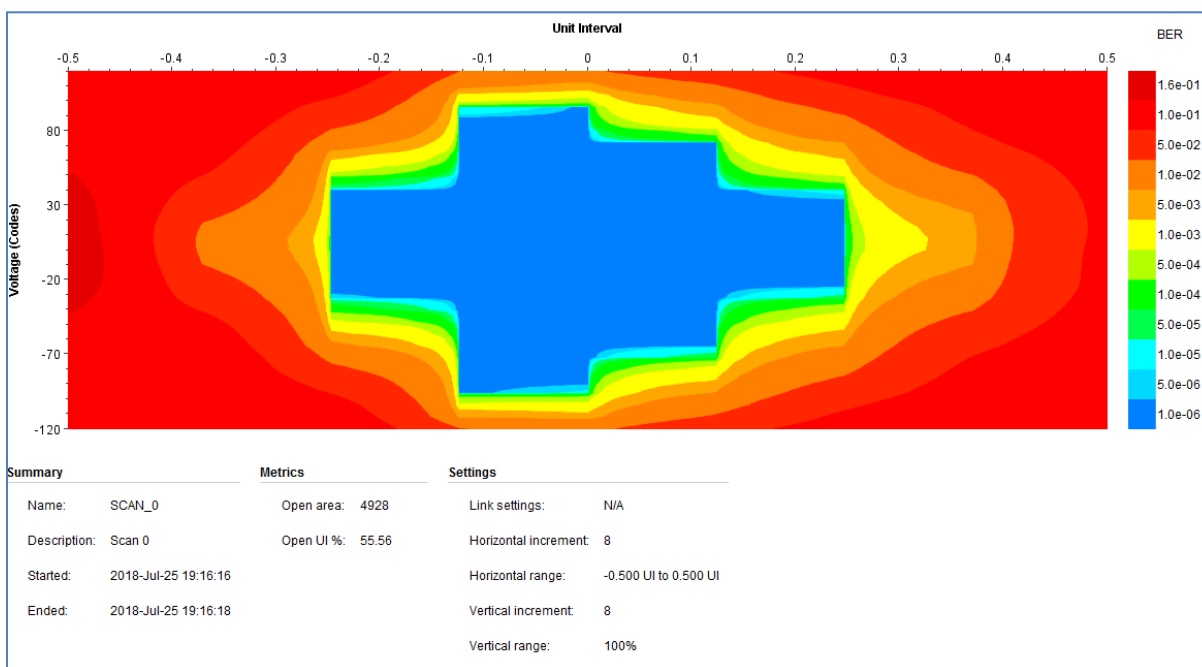
在 Create\_scan 界面中无需修改，点击 OK。



这时会出现这个 Link 的眼图的测试情况，Link 速度为 1.25Gbps 的眼图如下图所示：



Link 速度为 10.0Gbps 的眼图如下图所示：



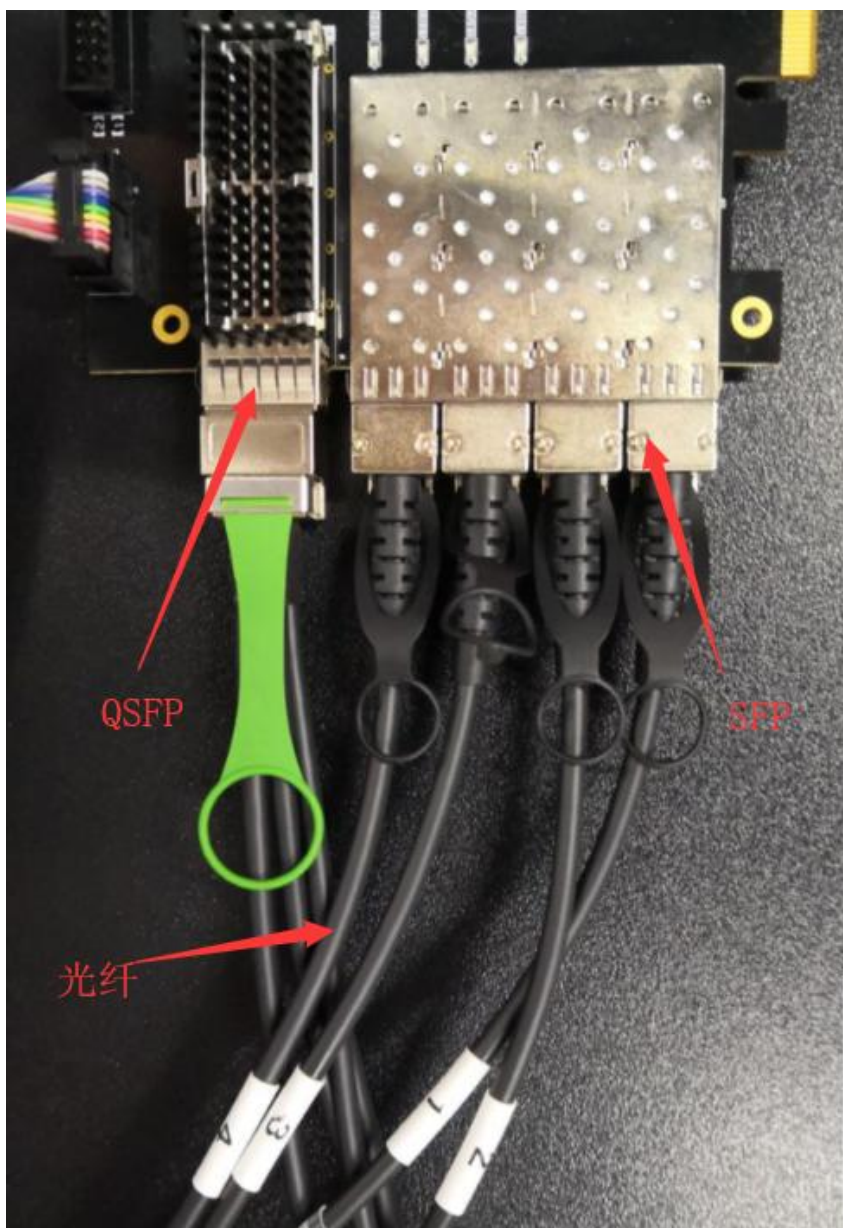
眼图中颜色越蓝的地方，BER 值越小，说明这个区域误码率越低，或者几乎没有误码率。颜色越红，表示这个区域误码率越高。一般来讲，这个眼图的眼睛张的越开，说明数据传输信号越好。从上面两张图片可以看出，Link 的速度越低，对应的眼图也会更好，Link 的速度越高，对应的眼图会下降，这对硬件设计和 PCB 设计提出了更高的要求。对眼图的介绍我们这里不做多讲，



大家自己百度搜索一下查找相关的资料。用同样的方法大家也可以分别看一下其它几路 Link 的眼图情况。

## 4.2 QSFP 与 4 路 SFP 眼图测试现象

测试之前我们把用光纤把 QSFP 和 4 路 SFP 的光模块分别分别连接起来（顺序可以任意），实验中我们的连接方式如下图所示。



连接好后，接下来上电下载 QSFP 与 4 路 SFP 的程序，测试效果如下：



Tcl ConsoleMessagesSerial I/O LinksSerial I/O Scans?

Q

Name	TX	RX	Status	Bits	Errors	BER	BERT Reset	TX Pattern	RX Pattern	T
Ungrouped Links (0)										
Found Links (8)										
Found 0	MGT_X0Y14/TX	MGT_X0Y8/RX	10.000 Gbps	1.823E11	0E0	5.485E-12	Reset	PRBS 7-bit	PRBS 7-bit	1
Found 1	MGT_X0Y12/TX	MGT_X0Y9/RX	10.000 Gbps	1.822E11	0E0	5.487E-12	Reset	PRBS 7-bit	PRBS 7-bit	1
Found 2	MGT_X0Y15/TX	MGT_X0Y10/RX	10.000 Gbps	1.823E11	0E0	5.484E-12	Reset	PRBS 7-bit	PRBS 7-bit	1
Found 3	MGT_X0Y13/TX	MGT_X0Y11/RX	10.000 Gbps	1.824E11	0E0	5.483E-12	Reset	PRBS 7-bit	PRBS 7-bit	1
Found 4	MGT_X0Y9/TX	MGT_X0Y12/RX	10.000 Gbps	1.823E11	0E0	5.486E-12	Reset	PRBS 7-bit	PRBS 7-bit	1
Found 5	MGT_X0Y11/TX	MGT_X0Y13/RX	10.000 Gbps	1.824E11	1E0	5.482E-12	Reset	PRBS 7-bit	PRBS 7-bit	1
Found 6	MGT_X0Y8/TX	MGT_X0Y14/RX	10.000 Gbps	1.823E11	0E0	5.485E-12	Reset	PRBS 7-bit	PRBS 7-bit	1
Found 7	MGT_X0Y10/TX	MGT_X0Y15/RX	10.000 Gbps	1.824E11	0E0	5.484E-12	Reset	PRBS 7-bit	PRBS 7-bit	1

到此为止，本章的实验到此就结束了，GTX 的光纤数据通信测试是通过 lbert\_7series\_GTX\_0 的 example 例子来测试开发板上的光纤数据传输，检测光纤通信是否存在丢包的情况及统计传输的误码率，另外也可以用软件直接观察每路 Link 的眼图状况。通过观察眼图来确认光纤通信的硬件设计和数据传输是否可靠。