

GTX 光纤通信测试例程

黑金动力社区 2021-10-13

1 实验简介

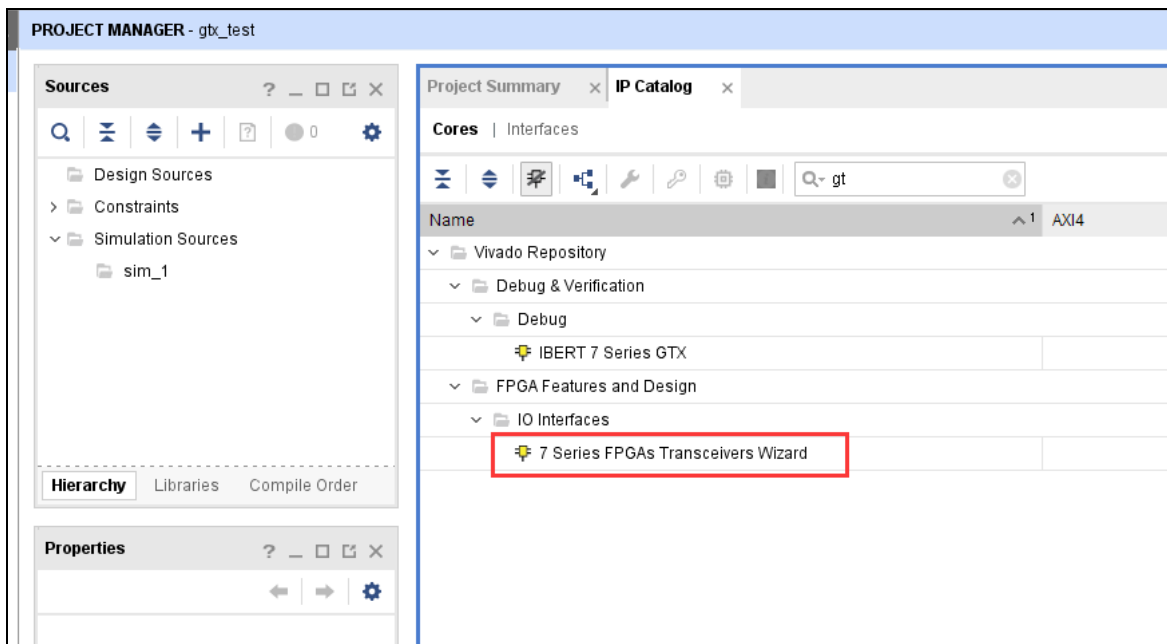
本实验将介绍通过光纤实现数据通信的传输，测试数据由 FPGA 自身产生，由 GTX 发送到第一路光纤口，然后通过光纤线环路到第二路光纤口，通过 GTP/GTX 接收数据进行校验。

2 实验原理

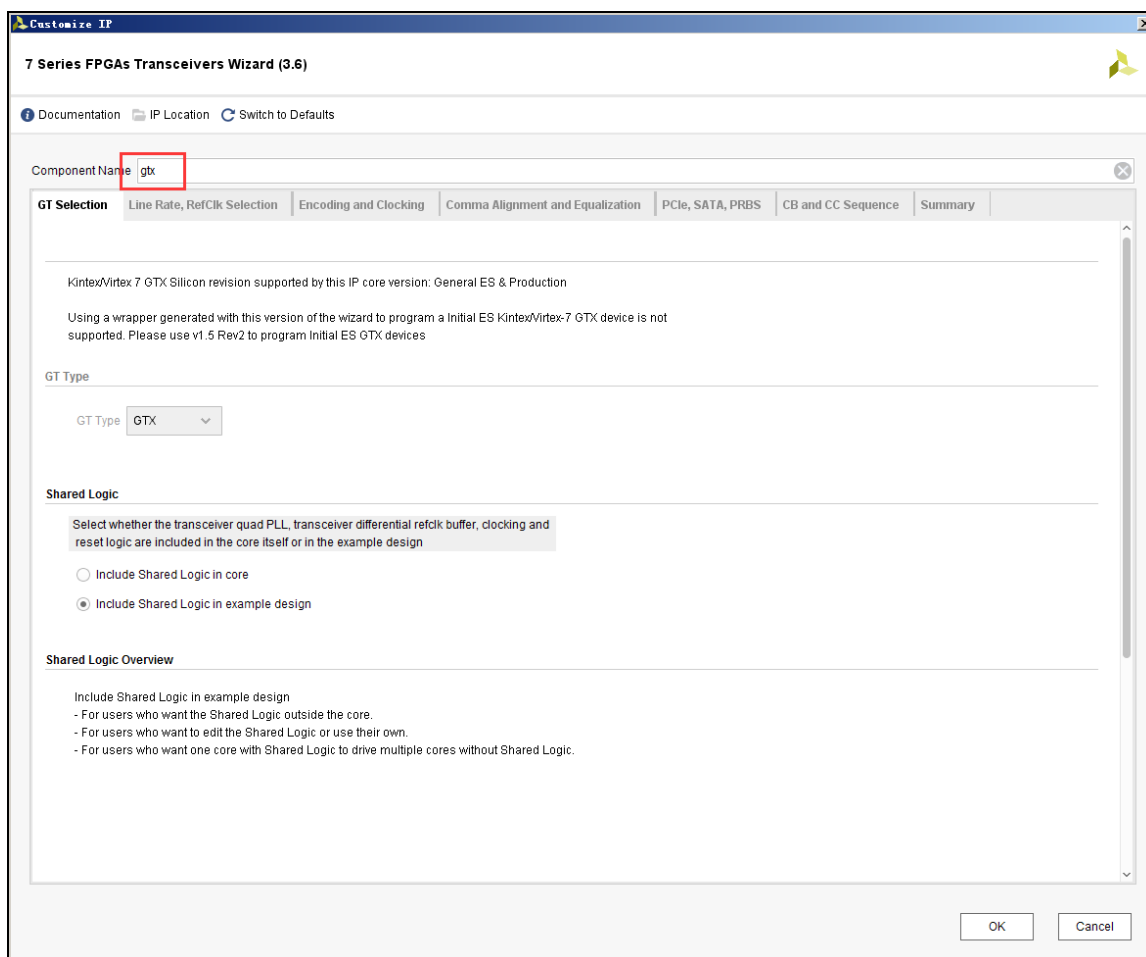
2.1 GTX IP 设计

XILINX 的 Vivado 软件已经为用户设计好了 GTX IP，用户无需关心 GTX 的内部具体工作就可以使用 IP 来实现 GTX 的高速的数据收发。下面我具体的 GTX IP 的生成和配置方法：

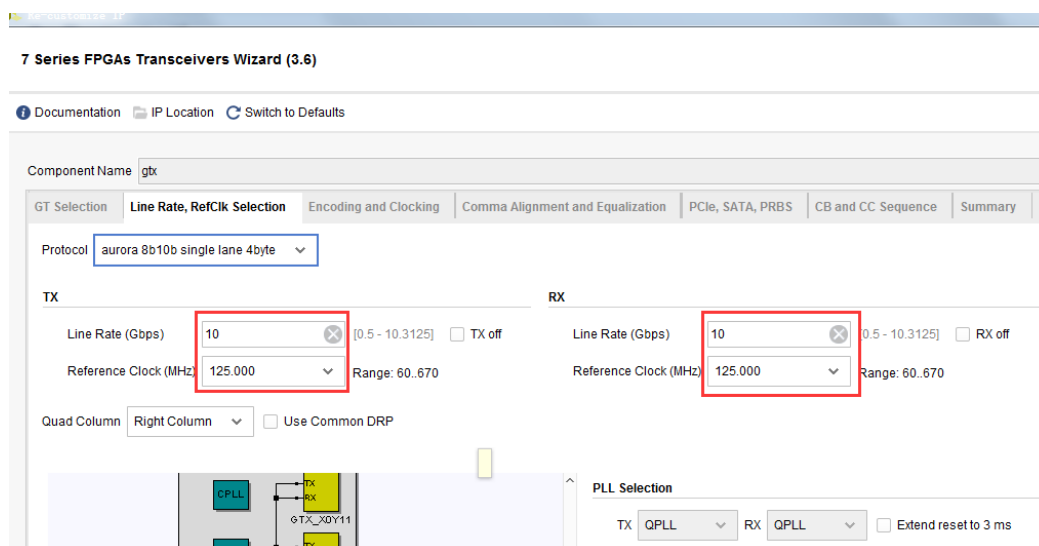
1. 在 IP Catalog 界面中双击 FPGA Features and Design\IO Interface 目录下的"7 Series FPGAs Transceivers Wizard"图标。



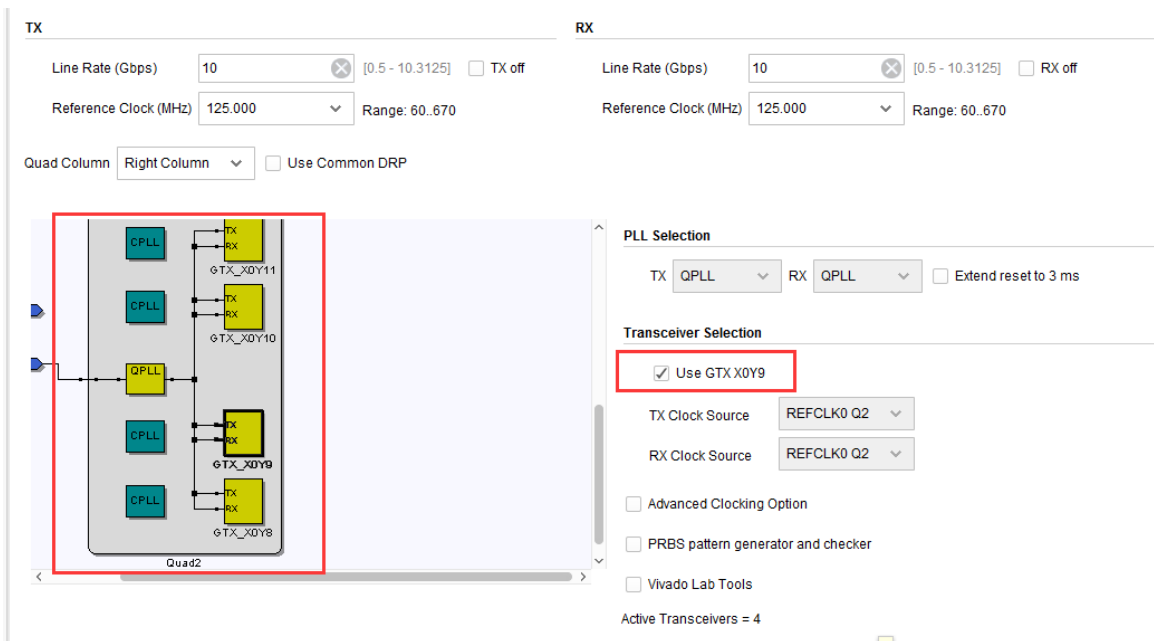
2. Component Name 栏输入"gtx"为取名，在 GT selection 界面里无需修改，保留默认。



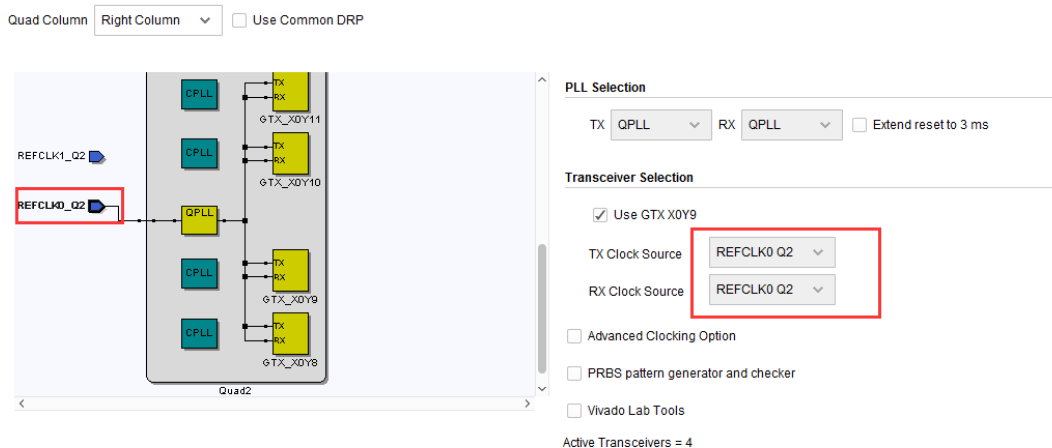
3. 在 Line Rate, Refclk Selection 界面里，首先设置 GTX 的传输协议位 “Aurora 8B10B single lane 4byte”，我们在前面一章讲过，Xilinx 的 GTX 是支持很多种协议的，Aurora 8B/10B 协议是一个可扩展的、轻量级的链路层协议，可以用于通过一条或多条串行链路将数据点到点传输。这里我们用的光模块传输是单路的，所以选择 single lane，数据接口为 4byte，就是 32 位数据。再选择 TX 和 RX 的 Line Rate 速度，这个 Line Rate 速度是需要是 GTX 参考时钟的整数倍，开发板上的 GTX 参考时钟为 125Mhz，这里我们 Line Rate 为参考时钟的 80 倍，所以 Line Rate 设置为 10Gbps，如果用户需要设置其它速率比如 1Gbps，只需要直接修改。Reference Clock 为 125Mhz。



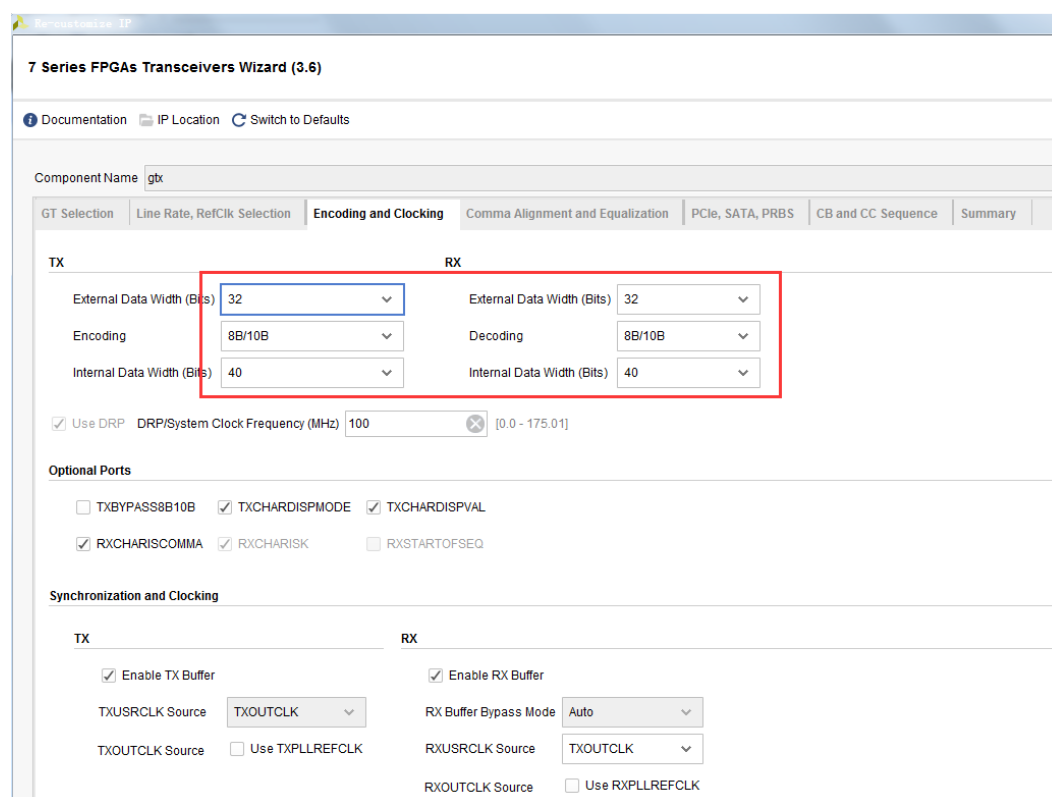
以 AX7Z035 开发板为例, xc7z035ffg676 芯片有 2 个 BANK 的 GTX 收发器, 在 AX7Z035 硬件设计上 BANK111 连接了 4 路 SFP+ 的光模块, 也就是 Quad2。所以这里需要选中 GTX_X0Y8, GTX_X0Y9, GTX_X0Y10, GTX_X0Y11, 然后选择右边的 "Use GTX X0Y8/9/10/11" 前面的钩, 这样 QPLL 都连接到了 4 个 GTX Channel 模块。



另外 TX Clock Source 和 RX Clock Source 需要都选择 REFCLK0 Q2, 因为电路设计上 125Mhz 的参考时钟是连接到 BANK117 的 REFCLK0 的管脚上的。



4. 在 Encoding and Clocking 界面里，设置 TX 和 RX 的外部数据宽度，8B/10B 使能，内部数据宽度等信息，这里设置内部数据宽度为 40。因为外部数据宽度是 32，通过 8B/1B 转换后为 40。



4. 在 Comma Alignment and Equalization 界面里，选择默认设置。这里选择 Comma 值为 K28.5, K28.5 是一种用以表示 Fibre Channel 操作开始的特殊 10 比特字符。8B/10B 编码中将 K28.5 作为 K 码的控制字符，称为“comma”，所以可以用 comma 字符指示帧的开始和结束标志，或始终修正和数据流对齐的控制字符。

Customize IP

7 Series FPGAs Transceivers Wizard (3.6)

Documentation IP Location Switch to Defaults

Component Name gtx

GT Selection Line Rate, RefClk Selection Encoding and Clocking **Comma Alignment and Equalization** PCIe, SATA, PRBS CB and CC Sequence Summary

RX COMMA Alignment

RX COMMA detection

☒ Use comma detection Comma Value K28.5 Comma Mask 1111111111

☐ Decode valid comma only Plus Comma 0101111100 Align to Two Byte Boundaries

☐ Combine plus/minus commas (double-length comma) Minus Comma 1010000011

Optional Ports

☒ ENPCOMMAALIGN (Enables positive Comma Alignment) ☒ ENMCOMMAALIGN (Enables negative Comma Alignment)

☐ RXSLIDE ☒ RXBYTEISALIGN ☒ RXBYTEREALIGN ☒ RXCOMMADET

Termination and Equalization

Differential Swing and Emphasis Mode Custom

RX Equalization **RX Termination**

Mode LPM-Auto Voltage Programmable

Automatic Gain Control Auto Trim Value (mV) 800

Optional Ports

☒ TXPOLARITY ☐ TXINHIBIT ☒ TXDIFFCTRL ☒ TXPOSTCURSOR ☒ TXPRECURSOR ☒ TXMAINCURSOR

☐ TXQPISENN ☐ TXQPISENP ☐ TXQPIBIASEN ☐ TXQPIWEAKPUP ☐ TXQPISTRONGPDOWN

OK Cancel

5. PCIe,SATA, PRBS 页面无需修改，保持默认设置。

Customize IP

7 Series FPGAs Transceivers Wizard (3.6)

Documentation IP Location Switch to Defaults

Component Name: gtx

GT Selection Line Rate, RefClk Selection Encoding and Clocking Comma Alignment and Equalization **PCle, SATA, PRBS** CB and CC Sequence Summary

PCle Express and SATA

☐ Enable PCI Express

SATA COM sequence

Bursts: 4 [0 - 7] Idles: 4 [0 - 7]

PCI Express Parameters

Transition Time

To P2: 100 [0 - 255] From P2: 60 [0 - 4095] To/From Non P2: 25 [0 - 255]

Optional Ports

☒ LOOPBACK ☐ RXCOMWAKEDET ☐ TXDETECTRX ☐ RXSTATUS

☐ TXCOMINIT ☐ TXELECIDLE ☐ RXVALID ☐ TXCOMSAS

☐ PHYSTATUS ☐ RXCOMINITDET ☐ TXCOMWAKE ☐ RXCOMSASDET

☐ TXCOMFINISH ☒ TXPOWERDOWN ☒ RXPOWERDOWN

OOB signalling and PRBS

☐ Use RX OOB Signal Detection

PRBS

☒ Use PRBS Detector ☒ Use Port TXPRBSSEL ☒ Use Port TXPRBSFORCEERR ☐ RXPRBS_LOOPBACK

OK Cancel

6. 在 CB and Sequence 页面中不用修改。

Customize IP

7 Series FPGAs Transceivers Wizard (3.6)

Documentation IP Location Switch to Defaults

Component Name:

GT Selection Line Rate, RefClk Selection Encoding and Clocking Comma Alignment and Equalization PCIe, SATA, PRBS **CB and CC Sequence** Summary

Channel Bonding

☐ Use Channel Bonding ☐ Use Two Channel Bonding Sequences

Sequence Max Skew: Sequence length:

Clock correction

☒ Use Clock Correction PPM Offset +/-: [-1250 - 1250]

☐ Use Two Clock Correction Sequences Periodicity of the CC sequence (bytes):

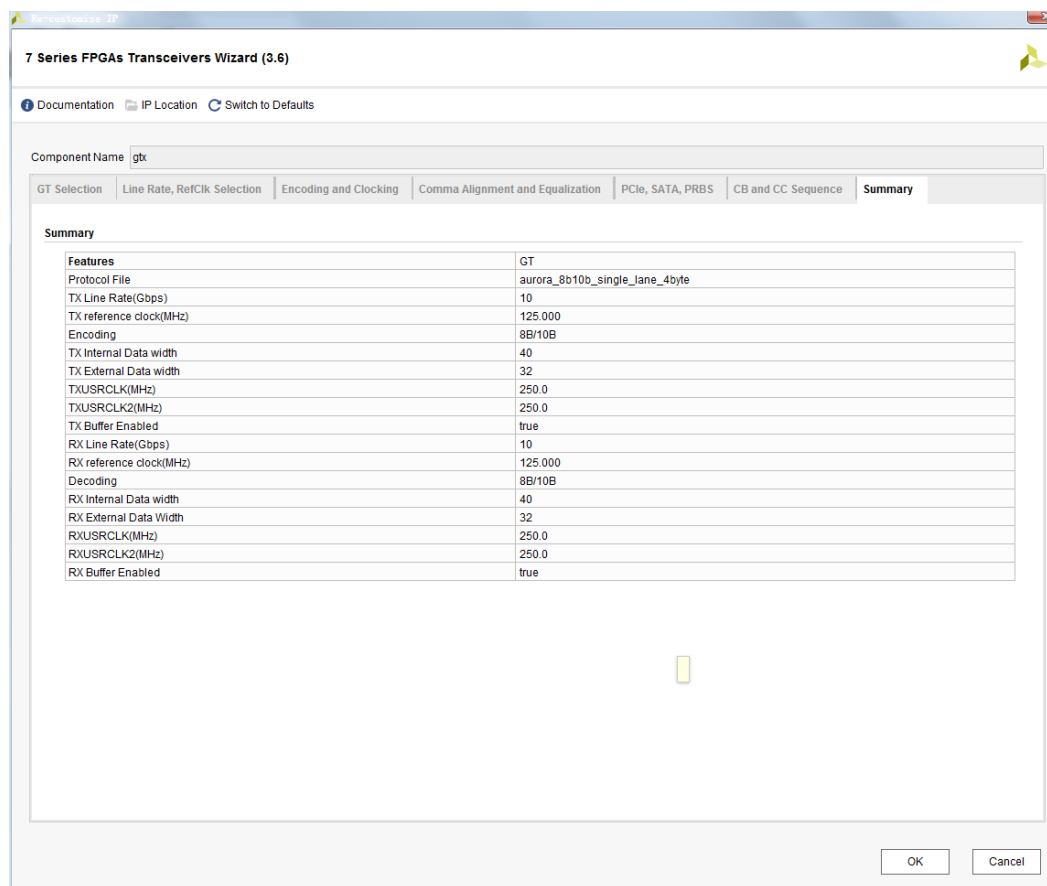
Sequence length:

Sequence Definition

	Sequence	K Character	Inverted Disparity	Don't Care
Sequence1, Byte1	11110111	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence1, Byte2	11110111	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence1, Byte3	11110111	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence1, Byte4	11110111	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence2, Byte1	00000000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence2, Byte2	00000000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence2, Byte3	00000000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence2, Byte4	00000000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

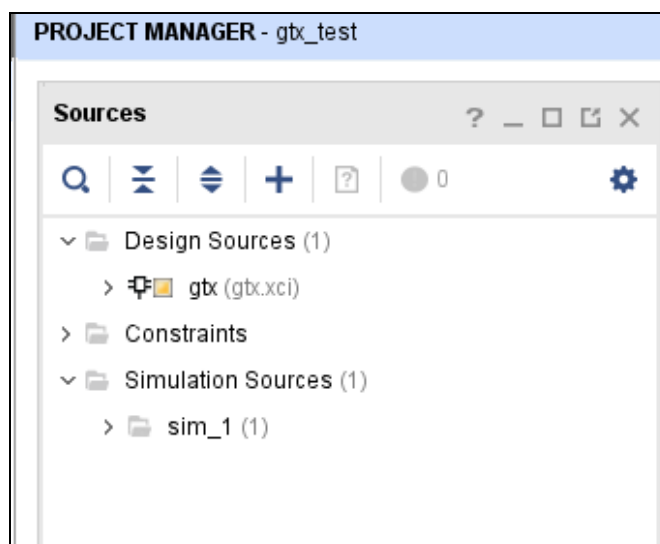
OK Cancel

7. 在 Summary 界面中检查一下配置情况，点击 OK 完成。

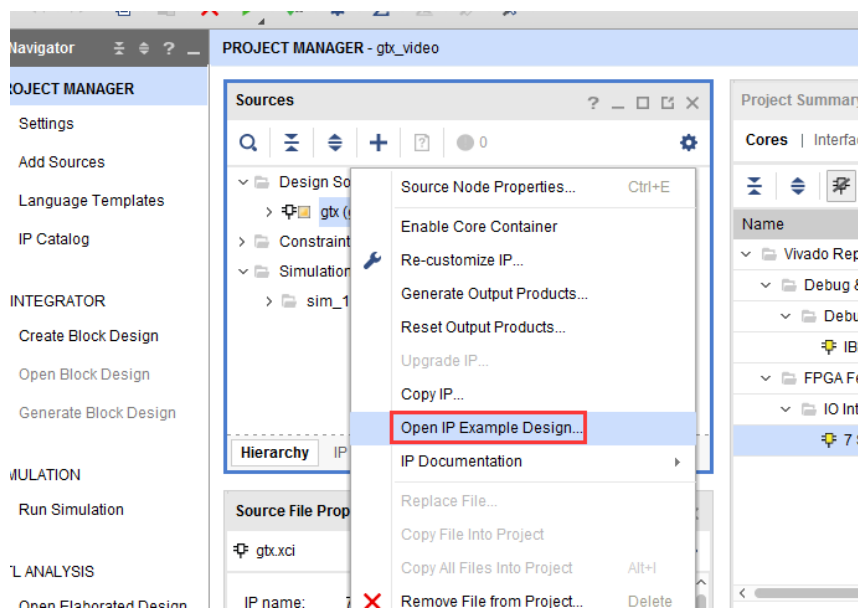


关于更多 GTX IP 的配置信息，大家请参考 Xilinx 提供的文档“pg168-gtwizard.pdf”和“UG476 7 Series FPGAs GTX_GTH Transceivers User Guide.pdf”，这两个文档上有对 GTX IP 各个配置参数做了详细的介绍。

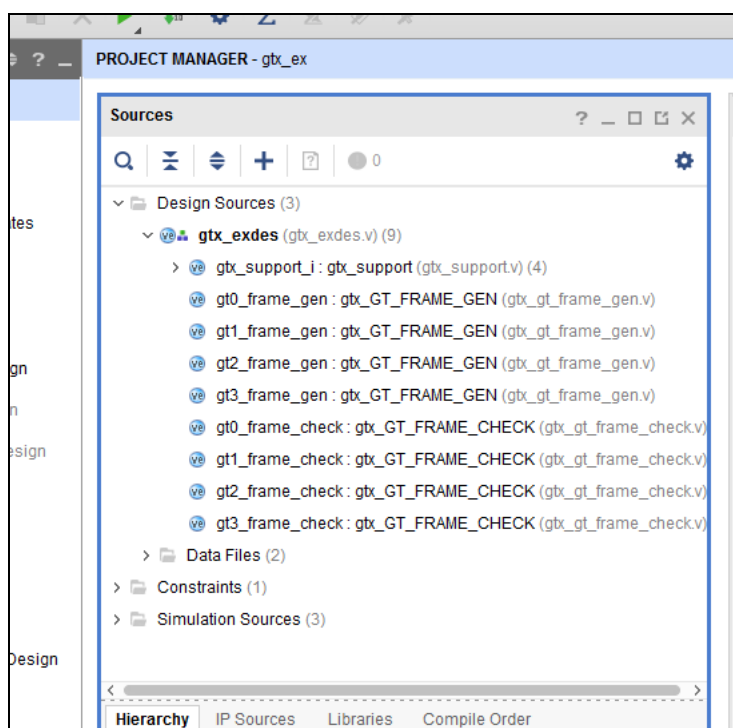
8. 配置完成在工程中自动添加刚刚生成 gtx 的 IP。



9. 我们来生成 GTX IP 的 example 工程，右键选择 gtx,在下拉菜单里选择"Open IP Example Design..."。



10. 生成的 example 工程如下图所示，在这个例子工程中，程序会在 gt0_frame_gen 模块中产生测试数据进行 GTX 的数据传输，在 gt0_frame_check 模块接收并检查是否正确，如果不正确，错误统计值增加。



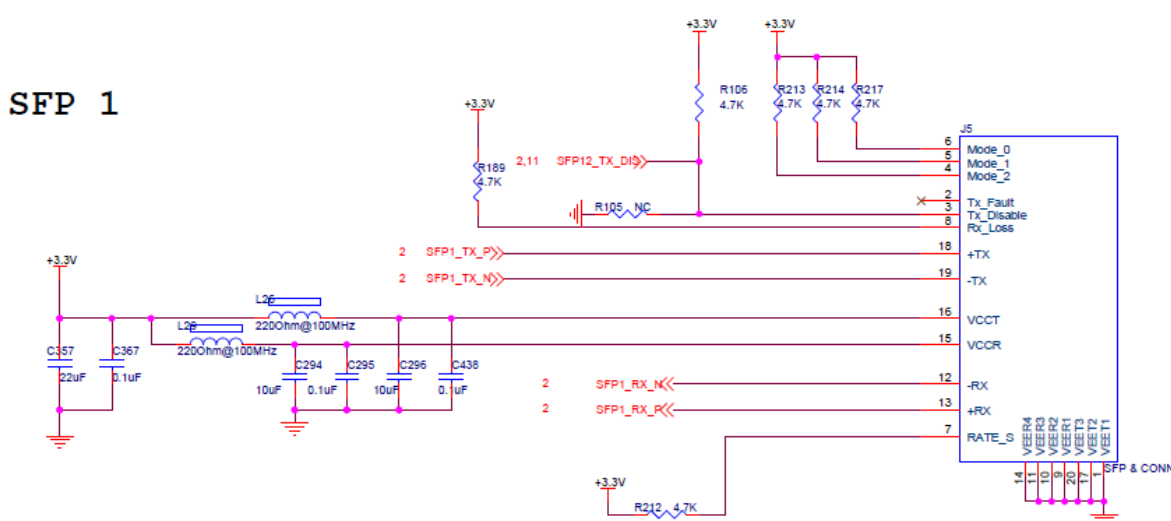
关于 example 的工程代码和测试我们这里不做介绍，大家自己去看去测试就可以了。在后面的 GTX 数据传输的软件开发中我们会用到这里 example 工程中的一些文件和 IP。

2.2 硬件介绍

在 AX7Z035 开发板上，有 4 路光纤接口 OPT1~OPT4，分别连接到 FPGA 芯片的 GTX 的通道上。

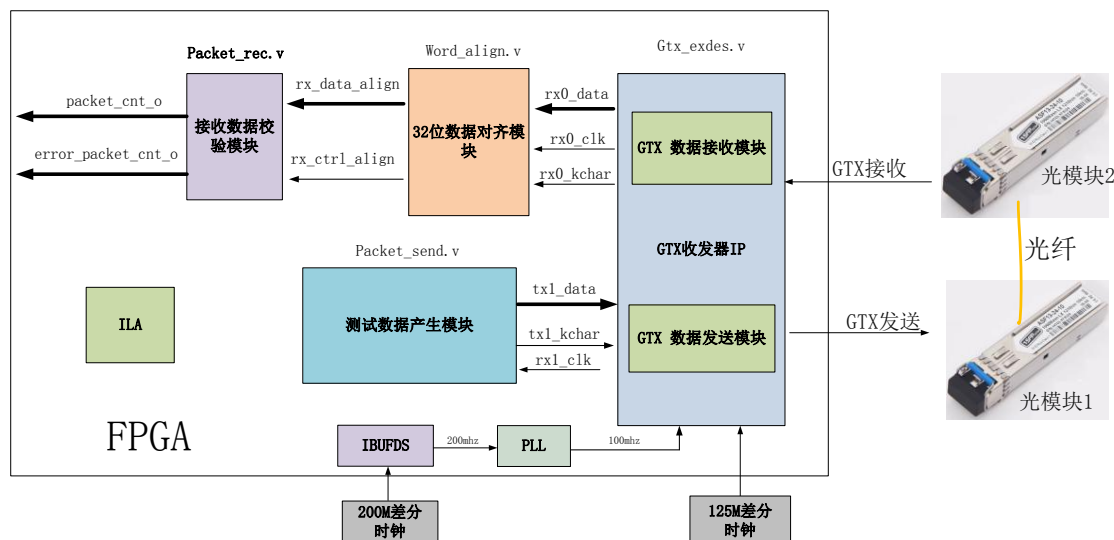
其中 SFP1 光模块接口连接到 GTX 的 Channel0 上，SFP2 跟 GTX 的 Channel1 相连，SFP3 跟 GTX 的 Channel2 相连，SFP4 跟 GTX 的 Channel3 相连。

光模块的 TX_Disable 信号连接到 FPGA 的普通 IO 上，SFP1 和 SFP2 用同一个信号控制，SFP3 和 SFP4 用同一个信号控制。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。LOSS 信号这里不使用。TSFP1 光模块硬件原理图如下：



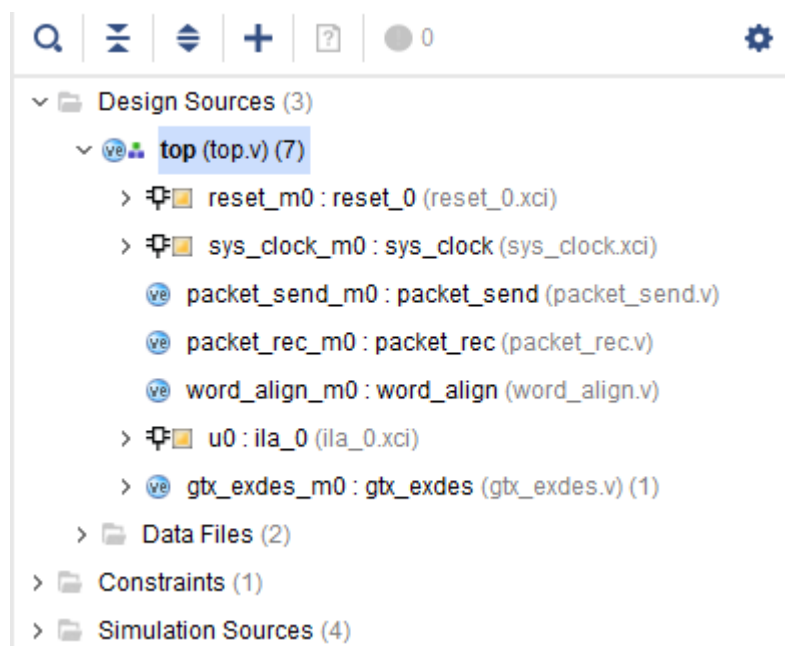
3 程序设计

光纤数据传输的 FPGA 程序设计是在 example 的工程代码的基础上添加了一个 TOP 文件和 3 个.v 文件，工程的逻辑框图如下图所示：



程序产生数据使用 GTX IP 发送给外部的光模块 1，光模块 1 把电信号转换成光信号通过光纤传输到光模块 2，光模块 2 又把光信号转换成电信号输入到 FPGA 的 GTX 接收，GTX 接收到的数据需要做一个 32 位数据对齐之后，并解析出数据信号和控制数据信号，校验模块对这些数据信号和控制信号进行校验计算判断接收数据和发送数据是否一致。

光纤数据传输设计好的工程如下图所示:

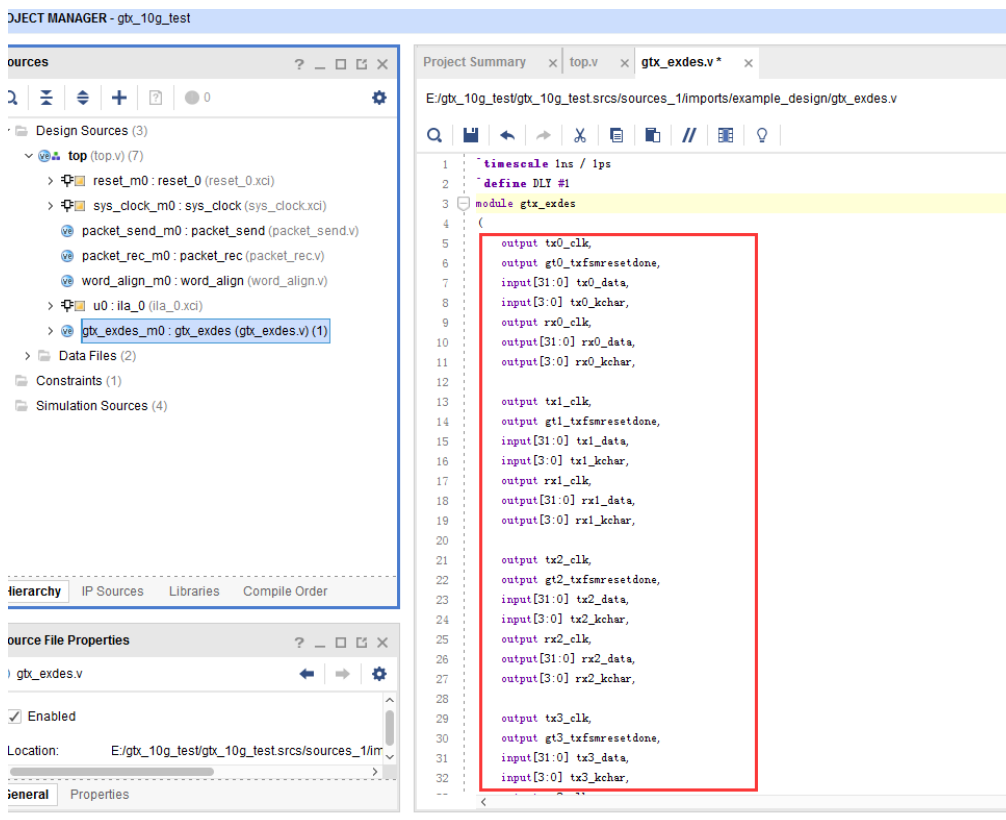


这里工程添加了 ILA 工具可以用来查看接收到的数据。

1) gtx 数据通信模块

在前面我们已经生成过 `gtx IP` 的 `example` 工程，这里删除了 `gt0_frame_gen.v` 模块和 `gt0_frame_check.v` 模块。因为这两个是测试数据的产生和检查模块，本例程用不上。另外对 `gtp_exdes.v` 文件进行修改，主要是删除 `gt0 frame_gen.v` 模块和 `gt0 frame check.v` 模块的例化，再

在模块的 Port 处添加以下的 4 个 channel 的用户接口信号，添加后的四个通道的信号如下图所示：



再把端口定义的四 Channel 的信号和 gtx_support 子模块里的信号进行连接。

```

650
651     assign tx0_clk = gt0_txusrclk2_i;
652     assign rx0_clk = gt0_rxusrclk2_i;
653     assign rx0_data = gt0_rxdata_i;
654     assign rx0_kchar = gt0_rxcharisk_i;
655     assign gt0_txfsmresetdone = gt0_txfsmresetdone_i;
656
657
658     assign tx1_clk = gt1_txusrclk2_i;
659     assign rx1_clk = gt1_rxusrclk2_i;
660     assign rx1_data = gt1_rxdata_i;
661     assign rx1_kchar = gt1_rxcharisk_i;
662     assign gt1_txfsmresetdone = gt1_txfsmresetdone_i;
663
664     assign tx2_clk = gt2_txusrclk2_i;
665     assign rx2_clk = gt2_rxusrclk2_i;
666     assign rx2_data = gt2_rxdata_i;
667     assign rx2_kchar = gt2_rxcharisk_i;
668     assign gt2_txfsmresetdone = gt2_txfsmresetdone_i;
669
670     assign tx3_clk = gt3_txusrclk2_i;
671     assign rx3_clk = gt3_rxusrclk2_i;

```

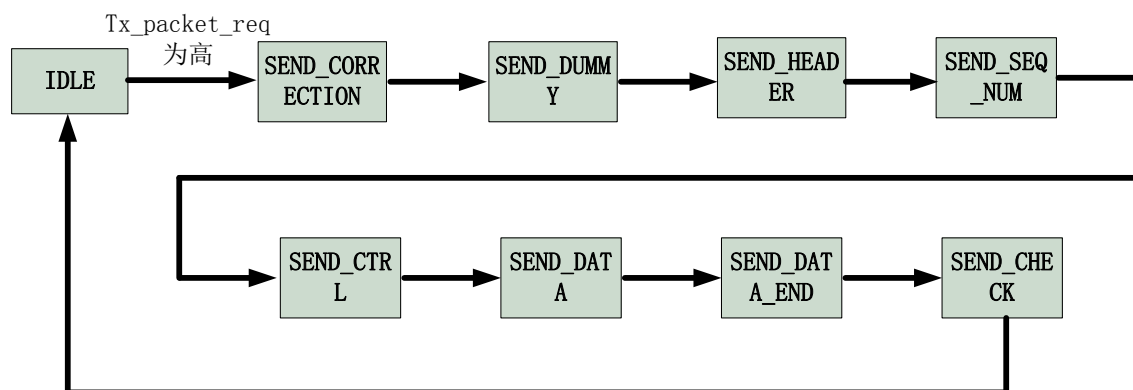
下面对在 gtx_exdes.v 端口中添加的几个 GTX 用户接口信号做一下介绍，以下以 channel0 的 GTX 接口为例：

信号名	位数	输入输出	说明
tx0_clk	1	输出	数据发送时钟，也就是第十四部分介绍的 TXUSRCLK2，频率为 GTX 的参考时钟 250Mhz。数据在上升沿有效。
tx0_data	32	输入	GTX 发送数据。
tx0_kchar	4	输入	<p>GTX 发送的 K 控制字，用来指示发送的数据是 K 码控制字符还是正常传输数据。高电平表明是 K 码控制字符，4 位对应发送数据的 4 个 Byte。</p> <p>tx0_kchar [3] 对应 tx0_data [31:24]</p> <p>tx0_kchar [2] 对应 tx0_data [23:16]</p> <p>tx0_kchar [1] 对应 tx0_data [15:8]</p> <p>tx0_kchar [0] 对应 tx0_data [7:0]</p>
rx0_clk	1	输出	数据接收时钟，也就是第十四部分介绍的 RXUSRCLK2，频率为 GTX 的参考时钟 250Mhz。数据在上升沿有效。
rx0_data	32	输出	GTX 接收数据。
rx0_kchar	4	输出	<p>GTX 接收 K 控制字，用来指示接收的数据是 K 码控制字符还是正常传输数据。高电平表明是 K 码控制字符，4 位对应接收数据 32 位的 4 个 Byte。</p> <p>rx0_kchar [3] 对应 rx0_data [31:24]</p> <p>rx0_kchar [2] 对应 rx0_data [23:16]</p> <p>rx0_kchar [1] 对应 rx0_data [15:8]</p> <p>rx0_kchar [0] 对应 rx0_data [7:0]</p>
gt0_tx fsmresetdone	1	输出	GTX 初始化完成信号。

知道了 GTX 用户接口信号的含义，我们就能通过这些用户接口实现光纤数据的收发。

2) gtx 数据包发送模块 packet_send.v

在 packet_send.v 中会由一个状态机来发送测试数据，首先一包数据开始传输前，会先发送同步包头信号，再发送数据包的序号和控制信号。然后把这测试数据通过 GTX 发送出去。发送流程如下图：



所有的 GTX 发送的数据位数为 32 位，在发送数据包之前需要发送 32 位的数据校正控制字，校正控制字用于消除不同的系统时钟之间造成的频率误差。gt_tx_ctrl 信号设置为 1111，表明这 32 位数据都是控制字。

```

    }
    SEND_CORRECTION:
    begin
        gt_tx_data <= 32'hF7_F7_F7_F7;
        gt_tx_ctrl <= 4'b1111;
        state <= SEND_DUMMY;
    end
  }
end

```

这个 32 位的校正字（F7F7F7F7）在 GTX 的 IP 里已经配置好了。所以 GTX 接收数据包的时候会自动做校正

	Sequence	K Character	Inverted Disparity	Don't Care
Sequence1, Byte1	11110111	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence1, Byte2	11110111	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence1, Byte3	11110111	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence1, Byte4	11110111	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence2, Byte1	00000000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence2, Byte2	00000000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence2, Byte3	00000000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Sequence2, Byte4	00000000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

同步信号包头信号定义为 32 位的 “ff_00_00_bc”，低 8 位“bc”是 K28.5 码控制字符。K 码特征字定义在 Xilinx 的 “7 Series FPGAs GTX_GTH Transceivers User Guide.pdf” 文档里有描述。

Table C-2: Valid Control K Characters

Special Code Name	Bits HGF EDCBA	Current RD - abcdei fghj	Current RD + abcdei fghj
K28.0	000 11100	001111 0100	110000 1011
K28.1	001 11100	001111 1001	110000 0110
K28.2	010 11100	001111 0101	110000 1010
K28.3	011 11100	001111 0011	110000 1100
K28.4	100 11100	001111 0010	110000 1101
K28.5	101 11100	001111 1010	110000 0101
K28.6	110 11100	001111 0110	110000 1001
K28.7(0)	111 11100	001111 1000	110000 0111
K23.7	111 10111	111010 1000	000101 0111
K27.7	111 11011	110110 1000	001001 0111
K29.7	111 11101	101110 1000	010001 0111
K30.7	111 11110	011110 1000	100001 0111

向 GTX 发送 K28.5 码控制字符时，需要拉高 gt_tx_ctrl 信号的对应位，标示发送数据里的某个字节位为 K 码控制字。所以这里在向 GTX 发送同步信号时，gt_tx_ctrl 信号设置为 0001，发送其它数据的时候则置为 0000。

```
48      SEND_HEADER:
49      begin
50          gt_tx_data <= 32'hff_00_00_bc;
51          gt_tx_ctrl <= 4'b0001;
52          state <= SEND_SEQ_NUM;
53          check_sum <= 32'd0;
54      end
55      SEND_SEQ_NUM:
56      begin
57          gt_tx_data <= sequence_number;
58          gt_tx_ctrl <= 4'b0000;
59          state <= SEND_CTRL;
```

3) 位数据对齐模块 word_align.v

GTX 收发器外部用户数据接口的宽度为 32 位，内部数据宽度为 40 位(8b/10b 转换)。在实际测试过程中发现，发送的 32 位数据会有可能出现 16 位的数据的移位，就是说发送的数据和接收到的数据会有 16 位的错位，下表演示 GTX 发送数据和接收数据移位的情况：

GTX 发送的数据		GTX 接收的数据	
数据 1	11111111	数据 1	11112222
数据 2	22222222	数据 2	22223333
数据 3	33333333	数据 3	33334444
数据 4	44444444	数据 4	44445555

数据 5	55555555	数据 5	5555.....
.....

因为我们在 GTX 发送同步信号和无用数据的时候加入了 K 码控制字，并且设置 `gt_tx_ctrl` 信号为 0001, 如果出现 16 位数据移位的情况，接收到的同步信号和无用数据时，K 码控制字也会跟着移位，`gt_tx_ctrl` 的信号就会变成 0100。所以我们在程序可以通过判断 `gt_tx_ctrl` 信号的值来判断接收到的 GTP 数据是否移位，如果接收到的 `gt_tx_ctrl` 为 0001，跟我们发送的时候一样，说明数据没有移位；如果接收到的 `gt_tx_ctrl` 为 0100，接收到的数据移位，需要重新组合，在 `word_align.v` 模块里完成。

```

24 always@(posedge rx_clk)
25 begin
26     case (align_bit)
27     4'b0001:
28         rx_data_align <= gt_rx_data;
29     4'b0100:
30         rx_data_align <= {gt_rx_data[15:0], gt_rx_data_d0[31:16]};
31     default:
32         rx_data_align <= 32'd0;
33     endcase
34 end

```

4) GTX 数据解析模块 `packet_rec.v`

因为接收到的 32 位数据中只有一部分是有效的数据，其它的是同步包头，序列数据，控制数据和 Checksum，在 `packet_rec.v` 模块里会计算数据的 checksum, 然后跟接收到的 checksum 值进行对比，如果不正确，会产生数据错误信号。

程序的一个功能是检测 GTX 数据中的同步包头信号（数据为 `ff_00_02_bc`），如果接收到同步包头信号，开始一包数据的接收。

```

WAIT_HEADER:
begin
    check_sum <= 32'd0;
    if(gt_rx_ctrl[0] == 1'b1 && gt_rx_data[7:0] == 8'hbc)
        state <= SEQ_NUM;
end
SEQ_NUM:

```

程序的另一个功能是判断统计数据的 checksum 并和接收到的 checksum 判断。


```

3   else if(state == CHECK)
4   begin
5       packet_cnt <= packet_cnt + 1;
6       if(check_sum != gt_rx_data || sequence_number != (last_sequence_number + 1))
7           error_packet_cnt <= error_packet_cnt + 1;
8   end
9 end

```

5) 管脚约束

这里的管脚约束是在 gtx IP 的 example 工程中的 gtx_exdes.xdc 文件中修改而来，比如 GTX 的参考时钟输入管脚，这里需要跟开发板上的管脚对应。另外还需要添加 SFP 光模块的发送控制管脚的定义如

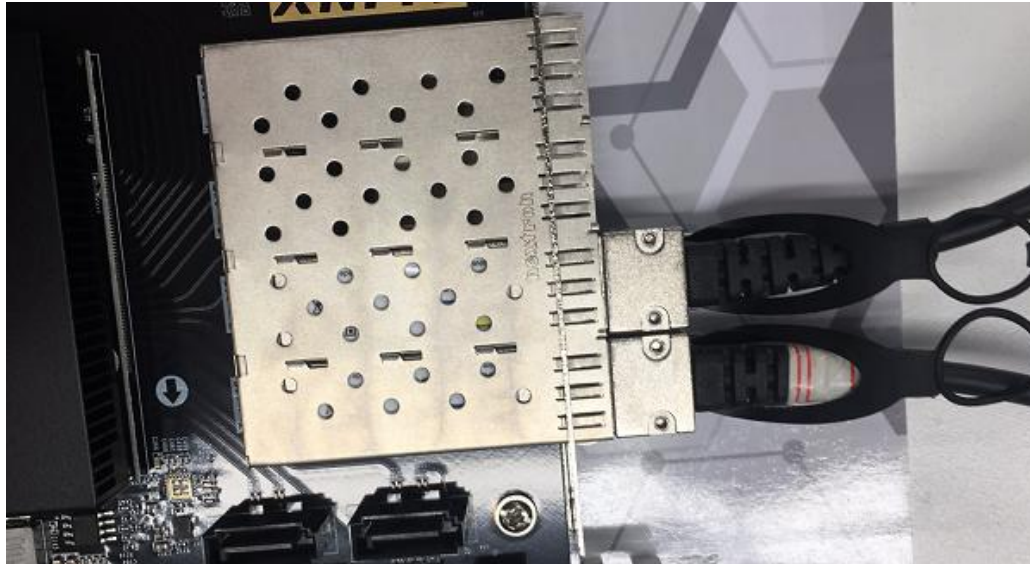
```

7  ##### key define#####
8  set_property PACKAGE_PIN AF15 [get_ports rst_n]
9  set_property IOSTANDARD LVCMOS33 [get_ports rst_n]
10 ##### fan define#####
11 set_property IOSTANDARD LVCMOS33 [get_ports fan_pwm]
12 set_property PACKAGE_PIN Y15 [get_ports fan_pwm]
13
14 ##### GI reference clock constraints #####
15 create_clock -period 8.000 [get_ports Q2_CLK0_GIREFCLK_PAD_P_IN]
16
17 ##### RefClk Location constraints #####
18 set_property PACKAGE_PIN AA5 [get_ports Q2_CLK0_GIREFCLK_PAD_N_IN]
19 set_property PACKAGE_PIN AA6 [get_ports Q2_CLK0_GIREFCLK_PAD_P_IN]
20
21
22 set_property PACKAGE_PIN AF17 [get_ports {tx_disable[0]}]
23 set_property IOSTANDARD LVCMOS33 [get_ports {tx_disable[0]}]
24
25 set_property PACKAGE_PIN AE17 [get_ports {tx_disable[1]}]
26 set_property IOSTANDARD LVCMOS33 [get_ports {tx_disable[1]}]
27
28
29
30 ##### mgt wrapper constraints #####
31 set_property LOC GTXE2_CHANNEL_X0Y8 [get_cells gtx_exdes_m0/gtx_support_i/gtx_init_i/inst/gtx_i/gt0_gtx_i/gtxe2_i]
32 ##----- Set placement for gt1_gtx_wrapper_i/GTXE2_CHANNEL -----
33 set_property LOC GTXE2_CHANNEL_X0Y9 [get_cells gtx_exdes_m0/gtx_support_i/gtx_init_i/inst/gtx_i/gt1_gtx_i/gtxe2_i]
34 ##----- Set placement for gt2_gtx_wrapper_i/GTXE2_CHANNEL -----
35 set_property LOC GTXE2_CHANNEL_X0Y10 [get_cells gtx_exdes_m0/gtx_support_i/gtx_init_i/inst/gtx_i/gt2_gtx_i/gtxe2_i]
36 ##----- Set placement for gt3_gtx_wrapper_i/GTXE2_CHANNEL -----
37 set_property LOC GTXE2_CHANNEL_X0Y11 [get_cells gtx_exdes_m0/gtx_support_i/gtx_init_i/inst/gtx_i/gt3_gtx_i/gtxe2_i]
38

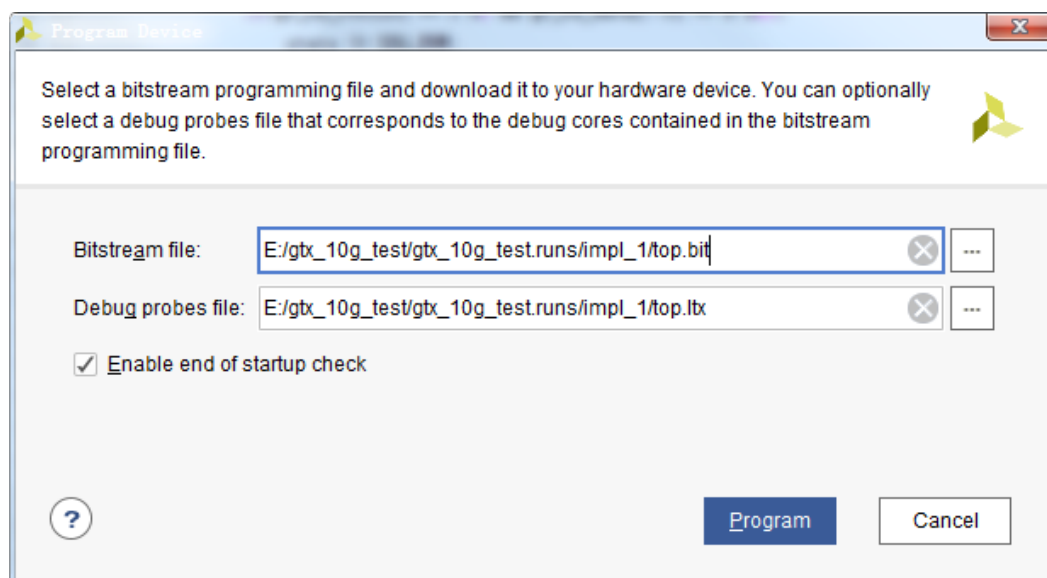
```

4 光纤数据传输测试

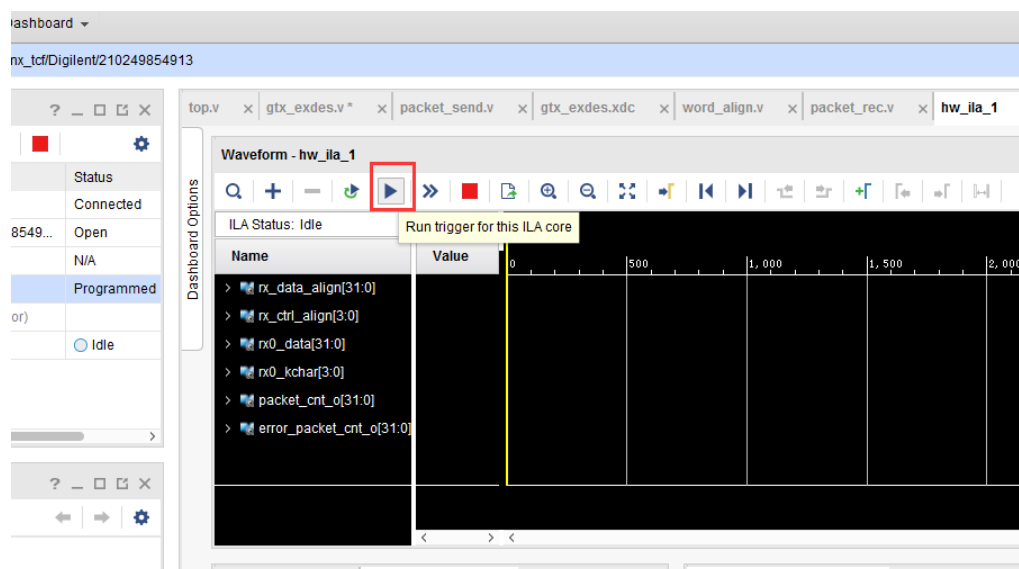
编译项目通过后我们就可以开始光纤数据传输的实验了。光模块插入到 AX7Z035 开发板的 OPT1~OPT2 接口上，再连接光纤。AX7Z035 硬件连接后如下图所示：



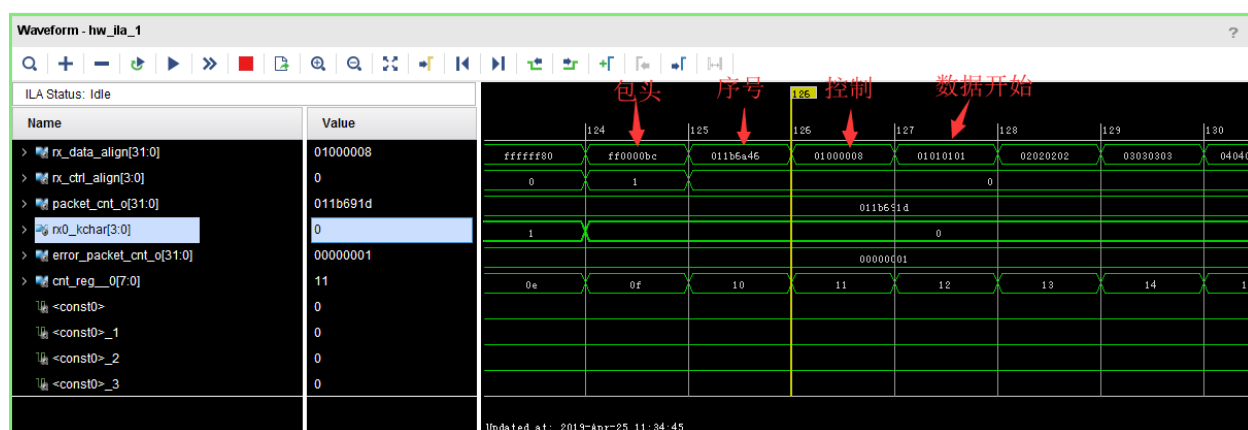
再下载 bit 文件到 FPGA,



点击 Run Trigger for this ILA core 按钮。



我们就可以在 ILA 里看到接收的测试数据了。







如果有接收的数据包错误，error_packet_cnt_o 的值会加 1。

这里为止 GTX 光纤数据传输例程就介绍完了。如果用户需要对工程中的 gtx IP 重新配置的话，如果只是修改工程里的 gtx IP 配置的话，编译的时候会报错。用户需要在修改工程中的 gtx IP 配置的同时，还是需要重新生成 gtx IP 的 example 文件，然后用 example 工程中生成的文件去替换工程中的以下文件。

gtx_10g_test.srcs ▸ sources_1 ▸ imports ▸ example_design ▸ support

新建文件夹

名称	修改日期	类型	大小
 gtx_common.v	2019/5/5 19:16	V 文件	9 KB
 gtx_common_reset.v	2019/5/5 19:16	V 文件	5 KB
 gtx_gt_usrclk_source.v	2019/5/5 19:16	V 文件	6 KB
 gtx_support.v	2019/5/6 8:56	V 文件	96 KB