

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视频图像的传输，视频图像由黑金双目摄像头模块 AN5642 采集，再通过开发板上的其中两路光模块进行视频信号的光纤发送和接收，然后在 HDMI 显示器上显示出来。

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。关于如何配置 GTX IP 部分，我们已经在“GTX 光纤数据传输例程”文档中介绍过，这里不做介绍，只是这里我们 Line Rate 为参考时钟的 25 倍，所以 Line Rate 设置为 3.125Gbps。如果用户需要配置其它的速度，可以自行修改。

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

Protocol aurora 8b10b single lane 4byte

TX **RX**

Line Rate (Gbps) 3.125 [0.5 - 10.3125] TX off Line Rate (Gbps) 3.125 [0.5 - 10.3125] RX off

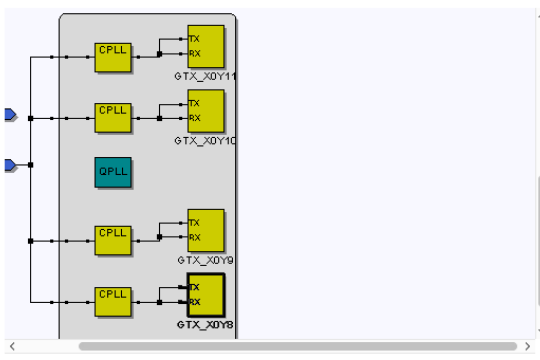
Reference Clock (MHz) 125.000 Range: 60..670 Reference Clock (MHz) 125.000 Range: 60..670

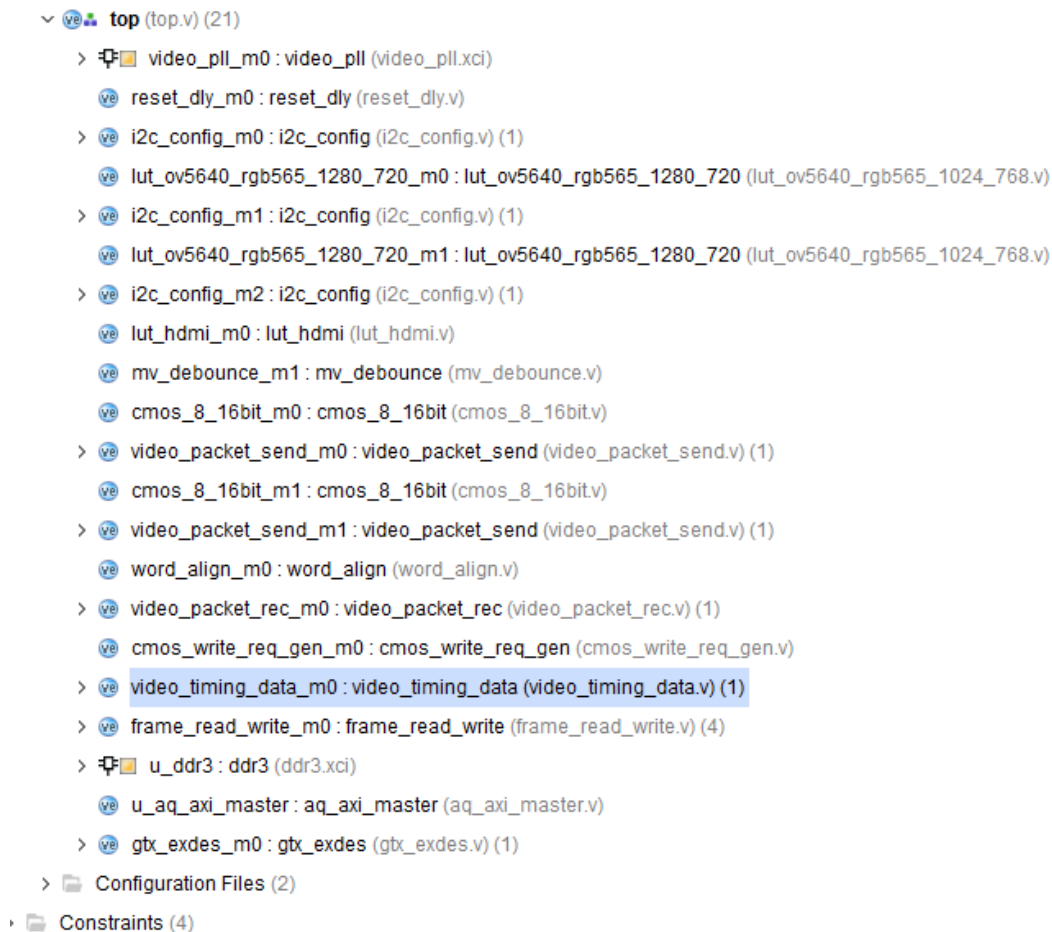
Quad Column Right Column Use Common DRP

PLL Selection TX CPLL RX CPLL Extend reset to 3 ms

Transceiver Selection Use GTX X0Y8 TX Clock Source REFCLK0 Q2 RX Clock Source REFCLK0 Q2 Advanced Clocking Option PRBS pattern generator and checker Vivado Lab Tools

Active Transceivers = 4

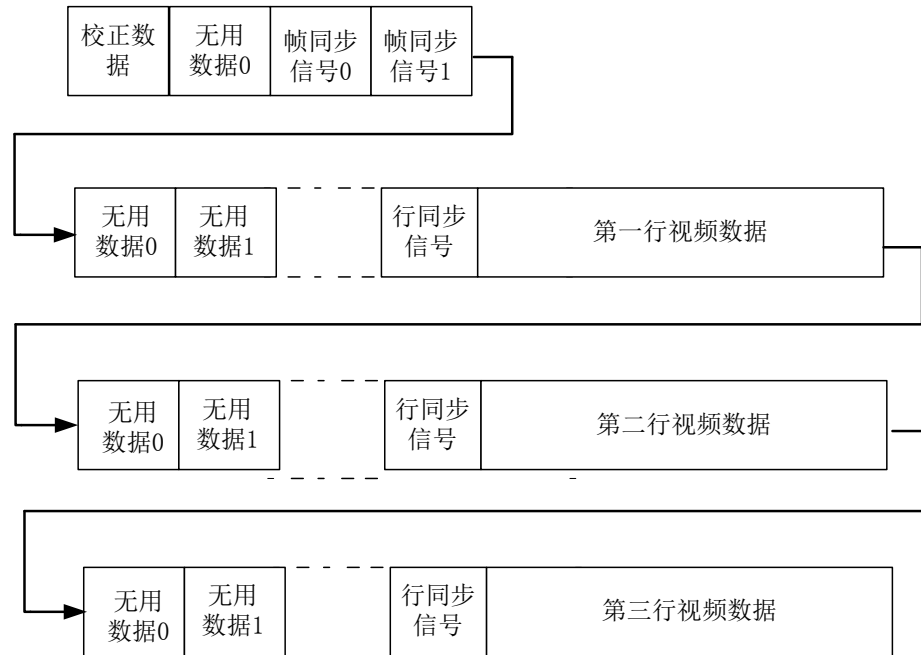




因为大部分程序跟前面的双目采集 HDMI 切换显示实验和光纤数据传输实验一样，我们这里只对增加部分的模块做一下功能说明：

1) GTX 数据包准备模块 video_packet_send.v

接收到的视频图像数据存放在 16 位进，32 位出的 FIFO 中，在 video_packet_send.v 中会由一个状态机来发送视频图像的数据，首先一帧图像开始传输前，GTX 会发送校正数据和帧同步信号。再判断这个 FIFO 中的数据量，如果 FIFO 内的数据还没有一行的视频数据，GTX 则发送无用的数据，当 FIFO 内已经有一行视频的数据时，GTX 会先发送行同步信号，然后再把这一行视频的数据通过 GTX 发送出去。一行数据发送完成，再重新判断 FIFO 内的数据量，FIFO 数据量达到一行视频数据时，接着发送第二行视频图像。GTX 数据发送一帧图像的流程如下图：



所有的 GTX 发送的数据位数为 32 位，帧同步信号、行同步信号、无用数据定义如下：

校正数据： 定义为 32 位的 “f7_f7_f7_f7”

图像帧同步信号 0： 定义为 32 位的 “ff_00_00_bc”

图像帧同步信号 1： 定义为 32 位的 “ff_00_01_bc”

GTX 插入的无用数据 0： 定义为 32 位的 “ff_55_55_bc”

GTX 插入的无用数据 1： 定义为 32 位的 “ff_aa_aa_bc”

图像行同步信号： 定义为 32 位的 “ff_00_02_bc”

这些同步信号和无用数据的高 24 位数据是用户自己定义的，低 8 位“bc”是 K28.5 码控制字符。K 码特征字定义在 Xilinx 的 “ug482_7Series_GTX_Transceivers.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 发送同步信号和无用数据的时候，gt_tx_ctrl 信号设置为 0001，发送视频数据的时候则置为 0000。

```
case(state)
SEND_CORRECTION:
begin
    gt_tx_data <= 32'hF7_F7_F7_F7;
    gt_tx_ctrl <= 4'b1111;
    state <= SEND_DUMMY;
end

SEND_FRAME_SYNC0: //发送视频的帧同步信号0
begin
    state <= SEND_FRAME_SYNC1;
    gt_tx_data <= 32'hff_00_00_bc;
    gt_tx_ctrl <= 4'b0001;
end

ena
SEND_LINE_START: //发送一行数据开始传输信号
begin
    state <= SEND_LINE_DATA;
    gt_tx_data <= 32'hff_00_02_bc;
    gt_tx_ctrl <= 4'b0001;
end
```

2) 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 20 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 GTX 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。

```
always@(posedge rx_clk)
begin
    case(align_bit)
        4'b0001:
            rx_data_align <= gt_rx_data;
        4'b0100:
            rx_data_align <= {gt_rx_data[15:0], gt_rx_data_d0[31:16]};
        default:
            rx_data_align <= 32'd0;
    endcase
end
```

3) GTX 视频数据解析模块 `video_packet_rec.v`

因为接收到的 32 位数据中只有一部分是视频图像的数据，其它的是帧同步，行同步和无用的数据，在 `video_packet_rec.v` 模块里需要把视频图像的数据解析出来存入到一个 32 位进，8 位出的 FIFO 中。

程序的一个功能是检测 GTX 数据中的行同步信号（数据为 `ff_00_02_bc`），如果接收到行同步信号，就把后面接收的一行视频数据存放到 FIFO 中。

```

//解析行同步信号
always@(posedge rx_clk or posedge rst)
begin
    if(rst)
        wr_en <= 1'b0;
    else if(gt_rx_ctrl == 4'b0001 && gt_rx_data == 32'hff_00_02_bc)
        wr_en <= 1'b1;
    else if(wr_cnt == ({1'b0,vout_width[15:1]} - 16'd1))
        wr_en <= 1'b0;
end

```

程序的另一个功能是恢复视频图像的帧同步信号（数据为 ff_00_00_bc），如果接收到帧同步信号，则置位视频图像的帧信号 vs。

```

24 //恢复帧同步信号
25 always@(posedge rx_clk or posedge rst)
26 begin
27     if(rst)
28         vs_r <= 1'b0;
29     else if(gt_rx_ctrl == 4'b0001 && gt_rx_data == 32'hff_00_00_bc)
30         vs_r <= 1'b1;
31     else if(vs_cnt > 16'd100)
32         vs_r <= 1'b0;
33 end

```

另外模块中会判断 FIFO 中存入的视频数据，如果 FIFO 内的数据量大于一行视频的数据，则产生 FIFO 的读使能信号，把 FIFO 的一行视频数据输出给外部接口模块。

其它模块在这里我们不做介绍了。

5) 管脚约束

这里的管脚约束有 4 个 xdc 文件，分别定义了系统，AN5642，HDMI，GTX 的管脚定义。

```

system.xdc (target)
an5642.xdc
hdmixdc
gtxdc

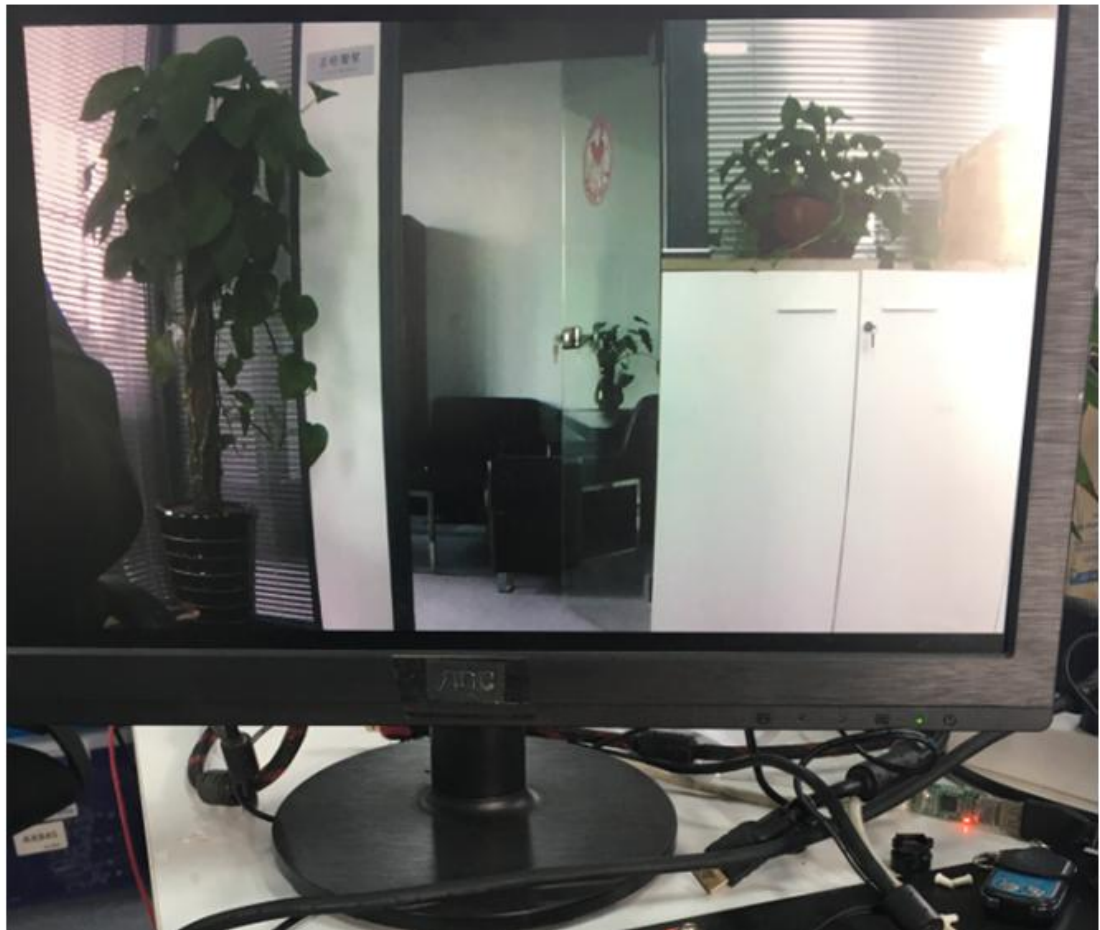
```

4 光纤视频传输现象

编译项目通过后我们就可以开始 OV5642 双目光纤视频传输的实验了。开发板的扩展口上插上摄像头 AN5642 模块，HDMI 口连接 HDMI 显示器，光模块插入到 SFP1 和 SFP2 的接口上，再连接光纤。硬件连接后如下图所示：



再下载 bit 文件到 FPGA, 我们就可以在 HDMI 显示器上看到视频图像从 AN5642 采集后 FPGA 先通过光纤传输，再环路回 FPGA，然后在 HDMI 显示器上显示的视频图像了。



程序中设计的是 4 个 GTX Channel (Channel0~3) 同时发送视频数据, 但只有接收 GTX Channel1 的视频图像数据显示, 因为 GTX Channel1 对应的是 SFP1, 所以只要当光纤连接 SFP1 光模块的接收和其它 4 路里的任意一路光模块的发送, 就能进行视频的传输了。