PG2T70H 开发板实验指导手册

开发手册版本：v1.1

时间：2025-07-31

公司：深圳市小眼睛科技有限公司

客服微信：17665247134

qq 群：808770961

淘宝店铺：小眼睛半导体

邮箱：support@meyesemi.com

公司网址：www.meyesemi.com

微信公众号：

抖音：

视频号：

目录

1. LED 流水灯实验例程

1.1 PG2T70H 开发板简介

PG2T70H 开发板有 8 个用户 LED 灯（LED1～8），FPGA 输出高电平时对应的 LED 灯亮灯（详情请查看“PG2T70H 开发板硬件使用手册”）。

1.2 实验目的

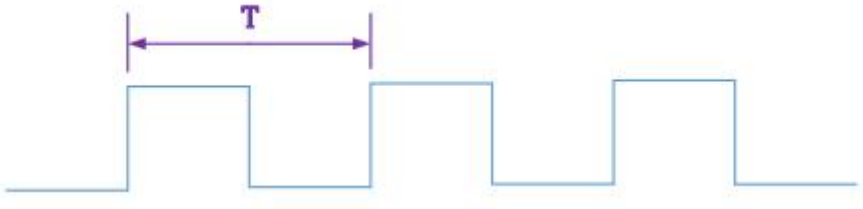
控制 8 个 LED 灯按顺序依次点亮和熄灭。

1.3 实验原理

通常的时，分，秒的计时进位大家应该不陌生；

1 小时=60 分钟=3600 秒，当时针转动 1 小时，秒针跳动 3600 次；

在数字电路中的时钟信号也是有固定的节奏的，这种节奏的开始到结束的时间,我们通常称之为周期（T）。



在数字系统中通常关注到时钟的频率,那频率与周期的关系如下：

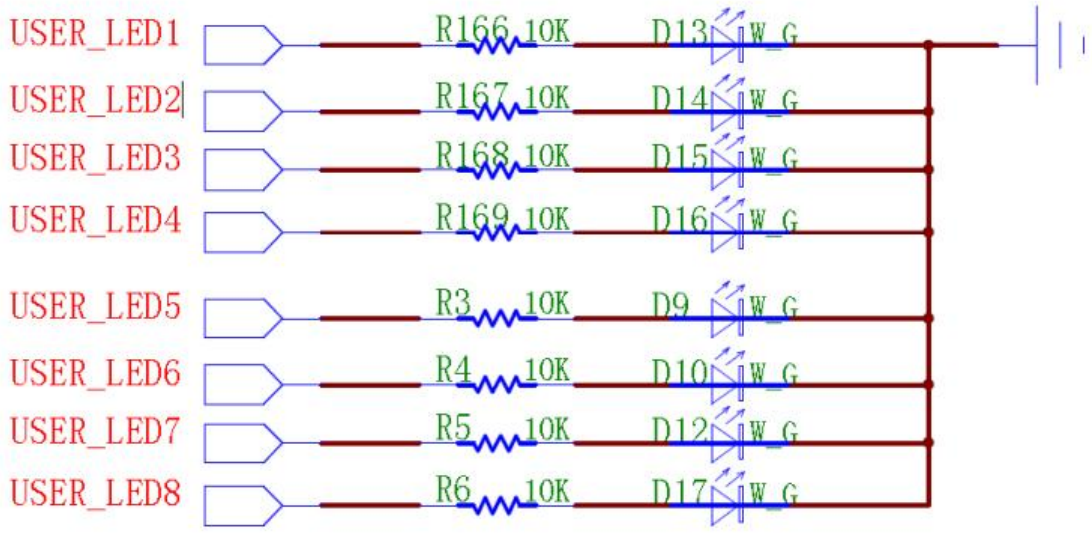
$$f = \frac{1}{T};$$

PG2T70H 板卡上单端时钟有一个 50MHz 和一个 27MHz 的晶振提供时钟给

到 PG2T70H;

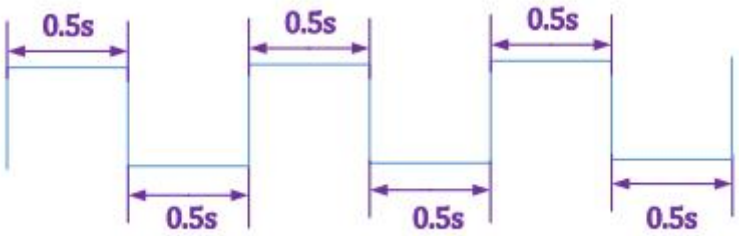
实验分析:

控制 LED 亮灭需要控制 IO 输出的高低电平即可(高电平点亮,低电平熄灭),原理图如下：



控制 LED 依次 0.5s 亮,0.5s 灭,需要控制 IO 依次输出 0.5s 高电平,0.5s 低电平周期变化,

如下图波形:



若使用 50MHz 外部输入时钟，时钟周期为 20ns（在 verilog 设计中的计数器的计时原理基本上是一致的，确认输入时钟周期和目标计时时间后可得到计数器的计数值到达多少后可得到计时宽度）;

$$0.5\text{s} = 25000000*20\text{ns} = 25000000 \times T_{50\text{MHz}};$$

IO 输出状态只有两种：1 或 0；我们可以使用一个计数器，计数满 25000000 个时钟周期时变化不同 LED 点亮。

1.4 实验源码设计

1.4.1 文件头设计

在 module 之前添加文件头，文件头中包含信息有：公司，作者，时间，设计名，工程名，模块名，目标器件，EDA 工具(版本)，模块描述，版本描述（修改描述）等信息；以及仿真时间单位定义；

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:Meyesemi
// Engineer: Will
//
// Create Date: 2023-01-29 20:31
// Design Name:
// Module Name:
// Project Name:
// Target Devices: Pango
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 1.0 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

`define UD #1
```

`timescale1ns/1ps 表示仿真精度是 1ns，显示精度是 1ps；

`defineUD#1 定义 UD 表示#1；#1 仅仿真有效，表示延时一个仿真精度，结合上一条语句表示延时 1ns；

1.4.2 设计 module

```
module led_test(
    input          clk,
```

```
    input           rstn,

    output [7:0]    led
);
```

此段代码是标准的 module 创建的模型，module 创建时需要确认输入输出信号并定义好位宽，之后在对 module 进行具体的逻辑设计；管脚与管脚之间用“,”隔开，最后一个管脚不用间隔符号；

创建 module 时需要定义输入输出信号;本实验输入时钟和复位即可，输出是控制 LED 的亮灭，MES50HP 板卡上共有 8 个 LED，故而输出 8bit 位宽的信号；

单个状态计数 25_000_000，即 24_999_999=25’b1_0111_1101_0111_1000_0011_1111;所以计数器的位宽为 25 位即可，此处请结合数字电路中的同步计数器的工作原理分析；

```
//time counter
    always @(posedge clk)
    begin
        if(!rstn)
            led_light_cnt <= `UD 26'd0;
        else if(led_light_cnt == 26'd24_999_999)
            led_light_cnt <= `UD 26'd0;
        else
            led_light_cnt <= `UD led_light_cnt + 26'd1;
    end
```

当计数器计数到 25'd24_999_999 时，计数过程包含了从 0～26'd2499_9999 的时钟周期，故而总时长时 25’d25_000_000×Tclk；硬件输入时钟为 50MHz，所以此计数器的计数周期是 0.5s；

在指定的时间刻度上对 LED 的状态进行变更，以达到控制 LED 依次亮灭的目的；

led_light_cnt 的计时周期为 0.5s，故在 led_light_cnt 上取一个点来变更 LED 的显示状态即可完成每隔 0.5sLED 显示发生变化；由于 LED 亮和灭只有两个状态，在赋值处理上将寄存器进行移位操作；

```
//Led status change
    always @(posedge clk)
    begin
        if(!rstn)
            led_status <= `UD 8'b0000_0001;
        else if(led_light_cnt == 25'd24_999_999)
```

```
            led_status <= `UD {led_status[6:0],led_status[7]};
    end

    assign led = led_status;
```

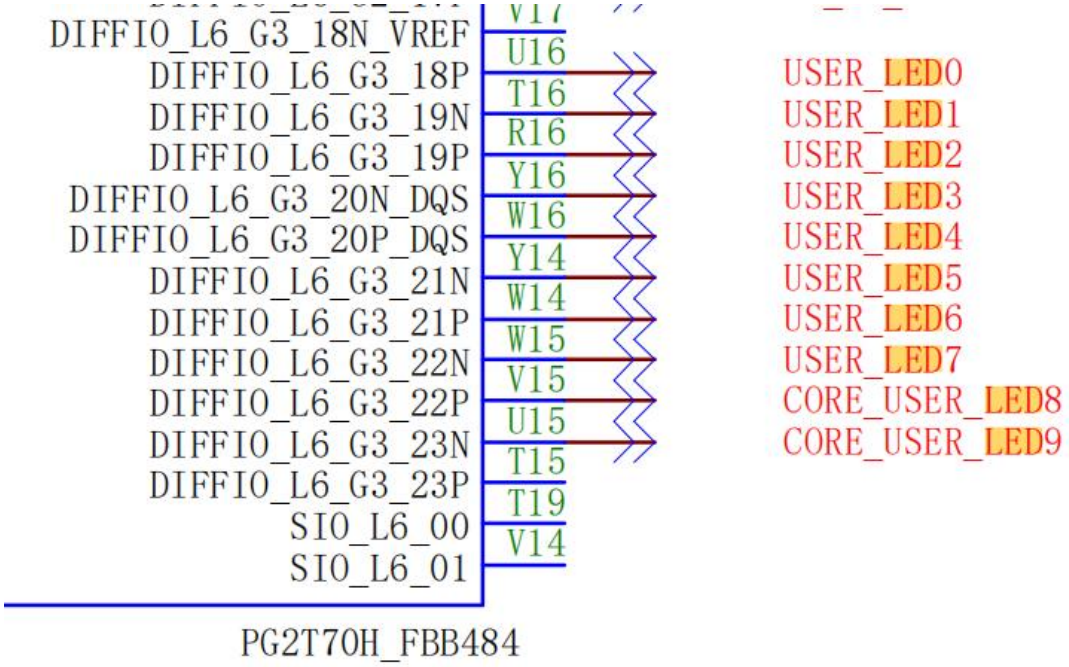
1.4.3 完整的 Module（不含注释）

```
module led_test(
    input           clk,
    input           rstn,

    output [7:0]    led
);


//=============================================================
===============
//reg and wire

    reg [25:0] led_light_cnt    = 26'd0         ;
    reg [ 7:0] led_status       = 8'b0000_0001  ;

//time counter
    always @(posedge clk)
    begin
        if(!rstn)
            led_light_cnt <= `UD 26'd0;
        else if(led_light_cnt == 26'd24_999_999)
            led_light_cnt <= `UD 26'd0;
        else
            led_light_cnt <= `UD led_light_cnt + 26'd1;
    end

//Led status change
    always @(posedge clk)
    begin
        if(!rstn)
            led_status <= `UD 8'b0000_0001;
        else if(led_light_cnt == 25'd24_999_999)
            led_status <= `UD {led_status[6:0],led_status[7]};
    end

    assign led = led_status;

```

```
37. endmodule
```

1.4.4 硬件管脚分配

PG2T70H 的 LED 和 CLK 与 FPGA 的 IO 连接部分的原理图如下，详情可查看硬件使用手册或原理图：



信号	FPGA Pin
LED1	U16
LED2	T16
LED3	R16
LED4	Y16
LED5	W16
LED6	Y14
LED7	W14
LED8	W15

复位设计是低电平有效，PG2T70H 开发板提供了 8 个用户按键（K1～8），按键低电平有效，但按键按下时，IO 上的输入电压为低；当没有按下按键时，IO 上的输入电压为高电平；选择任一个用户按键作为复位输入即可。

1.5 实验现象

8 颗 LED 灯按照设定的顺序和时间依次点亮和熄灭。

2. 键控流水灯实验例程

2.1 PG2T70H 开发板简介

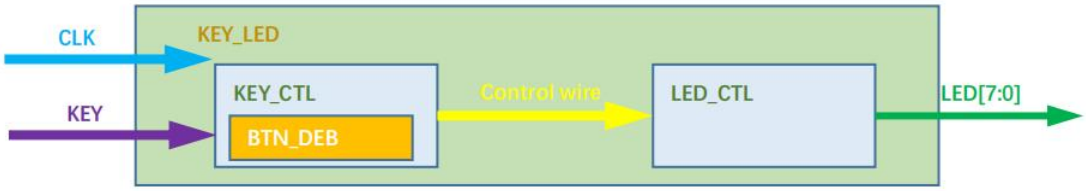
PG2T70H 开发板有 8 个用户 LED 灯（LED1～8），FPGA 输出高电平时对应的 LED 灯亮灯（详情请查看“PG2T70H 开发板硬件使用手册”）。

2.2 实验目的

由 USER_BUTTON1 按键输入，切换 USER_LED1~USER_LED8 的输出效果。

2.3 实验原理

实现框架如下：



（1）顶层实现按键切换 LED 的流水灯状态；

（2）需要设计一个输入控制模块及一个输出控制模块；

这个实验带大家将多个模块整合成为一个工程，涉及到的知识点有子模块设计、模块例化；子模块的设计主要是依据功能定位，确定输入输出，再做具体的设计；

模块例化方式如下：

```
module_name #  (
    .PARAM          (     PARAM_SET )          // PARAM 为例化模块的常量接口；PARAM_SET 为常量赋值内容
) unint_name(                                                  // module_name  为例化 module 名；unint_name 为例化后单元名称
    .port               (    signal           )        // port 为例化模块中的管脚；  signal 为当前模块的信号
);
```

2.3.1 按键控制模块功能

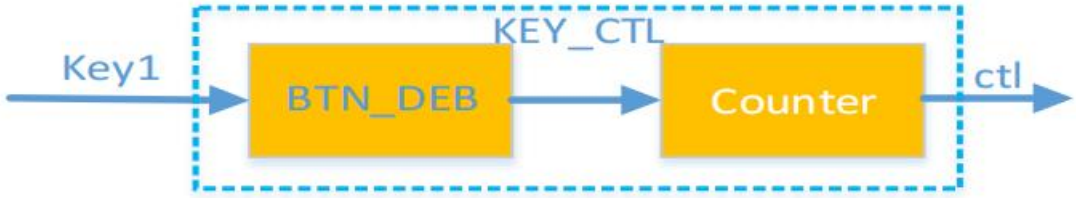
接收按键输入信号。统计按键按下次数，由于流水灯模式是 3 种，计数统计范围是 0～2 循环，将计数结果传递给 LED 控制模块；

根据需求输入信号有：时钟，按键；输出信号有：流水灯控制信号；

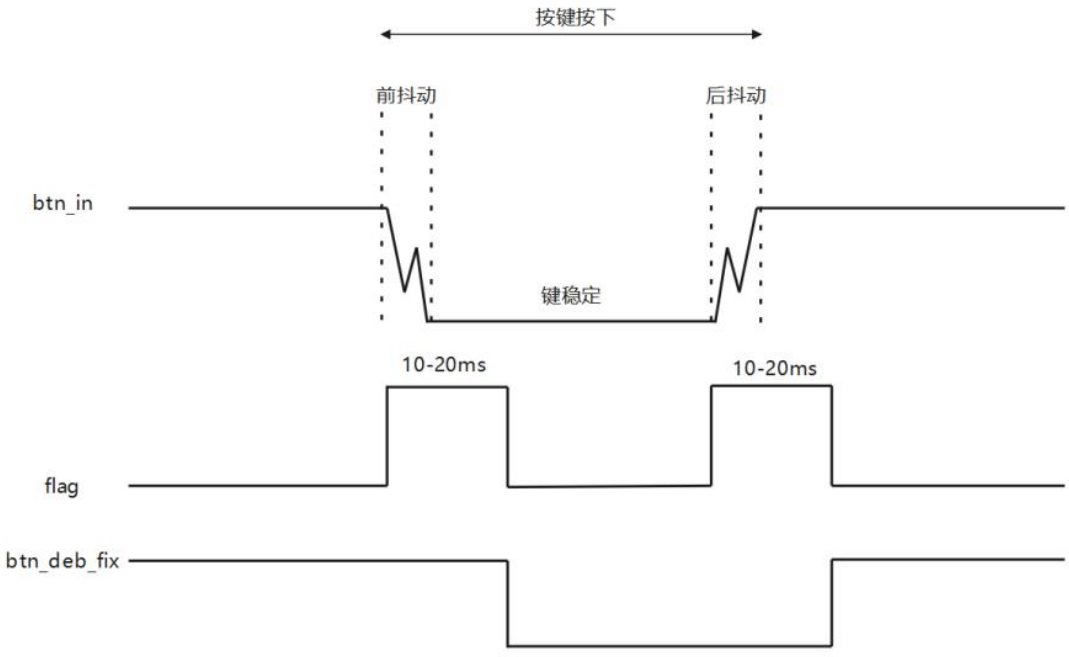
内部功能处理：

<1>内部需要对按键信号做消抖处理；

<2>按键触发计数器（计数值输出）改变继而调整流水灯的状态；



2.3.2 按键消抖模块



前后抖动时间约为 5～10ms，取按键抖动区间开始标识，持续 10-20ms 后标识归零，在抖动区间内输出保持，非消抖区间，按键状态输出。

2.3.3 LED 控制模块功能

3 种流水灯模式有按键传递过来的计数控制切换，每一个 LED 的显示状态完整后进入下一模式初始化。根据需求可得到如下信息：

输入信号：时钟，流水灯模式控制信号；出信号：8bit 位宽的 LED 控制信号；功能处理注意事项：流水灯状态切换点，不同状态的切换时如何初始化；

2.4 实验源码设计

2.4.1 顶层文件源码

```
`timescale 1ns / 1ps
`define UD #1
module key_led_top(
    input           clk,//50MHz
    input           key,

    output [7:0]    led
);

  wire [1:0] ctrl;

  key_ctl key_ctl(
      .clk        (  clk  ),//input           clk,
      .key        (  key  ),//input           key,

      .ctrl       (  ctrl  )//output     [1:0] ctrl
  );

  led u_led(
      .clk   (  clk   ),//input         clk,
      .ctrl  (  ctrl  ),//input  [1:0]  ctrl,

      .led   (  led   ) //output [7:0]  led
  );

endmodule
```

2.4.2 按键控制模块

```
`timescale 1ns / 1ps
`define UD #1
module key_ctl(
    input            clk,
    input            key,

    output     [1:0] ctrl
);

    wire btn_deb;
    // °´¾üÏû¶¶
    btn_deb_fix#(
        .BTN_WIDTH   (  4'd1         ), //parameter BTN_WIDTH = 4'd8
        .BTN_DELAY   (20'h7_ffff     )
    ) u_btn_deb
    (
        .clk         (  clk          ),//input clk,
        .btn_in      (  key          ),//input      [BTN_WIDTH-1:0] btn_in,

        .btn_deb_fix (  btn_deb      ) //output reg [BTN_WIDTH-1:0] btn_deb
    );

    reg btn_deb_1d;
    always @(posedge clk)
    begin
        btn_deb_1d <= `UD btn_deb;
    end

    reg [1:0]  key_push_cnt=2'd0;
    always @(posedge clk)
    begin
        if(~btn_deb & btn_deb_1d)
        begin
            if(key_push_cnt == 2'd2)
                key_push_cnt <= `UD 2'd0;
            else
                key_push_cnt <= `UD key_push_cnt + 2'd1;
```

```
        end
    end

    assign ctrl = key_push_cnt;

endmodule
```

2.4.3 按键消抖模块

```
`timescale 1ns / 1ps
`define UD #1
module btn_deb_fix#(
    parameter        BTN_WIDTH = 4'd8,
    parameter        BTN_DELAY = 20'h7_ffff
)
(
    input                       clk,  //
    input      [BTN_WIDTH-1:0] btn_in,

    output reg [BTN_WIDTH-1:0] btn_deb_fix
);

    //16'h3ad43;
    reg [19:0]          cnt[BTN_WIDTH-1:0];
    reg [BTN_WIDTH-1:0] flag;

    reg [BTN_WIDTH-1:0] btn_in_reg;

    always @(posedge clk)
    begin
     btn_in_reg <= `UD btn_in;
    end

    genvar i;
    generate
        for(i=0;i<BTN_WIDTH;i=i+1)
        begin
            always @(posedge clk)
            begin
             if (btn_in_reg[i] ^ btn_in[i]) //取按键边沿开始抖动区间标识
              flag[i] <= `UD 1'b1;
             else if (cnt[i]==BTN_DELAY)    //持续 10ms-20ms 后归零
              flag[i] <= `UD 1'b0;
```

```
                else
                    flag[i] <= `UD flag[i];
            end

            always @(posedge clk)
            begin
             if(cnt[i]==BTN_DELAY)       //计数 10ms-20ms 时归零
              cnt[i] <= `UD 20'd0;
             else if(flag[i])            //抖动区间有效时计数
              cnt[i] <= `UD cnt[i] + 1'b1;
             else                        //非抖动区间保持 0
              cnt[i] <= `UD 20'd0;
            end

            always @(posedge clk)
            begin
             if(flag[i])                 //抖动区间，消抖输出保持
              btn_deb_fix[i] <= `UD btn_deb_fix[i];
             else                        //非抖动区间，按键状态传递到消抖输出
              btn_deb_fix[i] <= `UD btn_in[i];
            end
        end
    endgenerate

endmodule
```

2.4.4LED 控制模块

```
`timescale 1ns / 1ps
`define UD #1
module led(
    input         clk,//50MHz
    input  [1:0]  ctrl,

    output [7:0]  led
);

    reg [24:0] led_light_cnt = 25'd0;
    reg [ 7:0] led_status = 8'b1000_0000;

    //  time counter
    always @(posedge clk)
    begin
```

```
        if(led_light_cnt == 25'd24_999_999)
            led_light_cnt <= `UD 25'd0;
        else
            led_light_cnt <= `UD led_light_cnt + 25'd1;
    end

    reg [1:0] ctrl_1d=0;    //保存上一个 led 状态周期的 ctrl 值
    always @(posedge clk)
    begin
        if(led_light_cnt == 25'd0)
            ctrl_1d <= ctrl;
    end

    // led status change
    always @(posedge clk)
    begin
        if(led_light_cnt == 25'd24_999_999)//0.5s 周期
        begin
            case(ctrl)
                2'd0 :  //从高位到低位的 led 流水灯
                begin
                    if(ctrl_1d != ctrl)
                        led_status <= `UD 8'b1000_0000;
                    else
                        led_status <= `UD {led_status[0],led_stat
us[7:1]};
                end
                2'd1 :  //隔一亮一交替
                begin
                    if(ctrl_1d != ctrl)
                        led_status <= `UD 8'b1010_1010;
                    else
                        led_status <= `UD ~led_status;
                end
                2'd2 :  //从高位到低位暗灯流水
                begin
                    if(ctrl_1d != ctrl )
                        led_status <= `UD 8'b0111_1111;
                    else
                        led_status <= `UD {led_status[0],led_stat
us[7:1]};
                end
            endcase
        end
```

```
    end

    assign led = led_status;

endmodule
```

2.5 实验现象

每按下一次 KEY1，LED 灯状态切换一次，总共三种 LED 模式供循环切换；

LED 模式一：从高位到低位的 LED 流水灯；LED 模式二：隔一亮一交替点亮；

LED 模式三：从高位到低位暗灯流水；

3.串口收发实验例程

3.1PG2T70H 开发板简介

PG2T70H 开发板集成了一路 USB 转串口模块，采用的 USB-UART 芯片 CP2102,USB 接口采用 USBTypeC 接口，可以用一根 USBTypeC 线连接到 PC 的 USB 口进行串口数据通信（详情请查看“PG2T70H 开发板硬件使用手册”）。

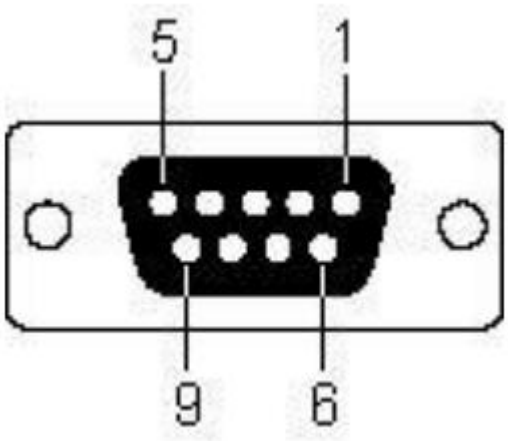
3.2 实验要求

串口通信时波特率设置为 115200bps，数据格式为 1 位起始位、8 位数据位、无校验位、1 位结束位。板子 1s 向串口助手发送一次十进制显示的“www.meyesemi.com”，通过串口助手向板子以十六进制形式发送数字（00~FF），LED 以二进制显示亮起。

3.3 实验原理

3.3.1 串口原理

从下图我们可以看到标准串口接口是 9 根线，具体含义如下：



数据线：

TXD（pin3）：串口数据输出(TransmitData)RXD（pin2）：串口数据输入(ReceiveData)

握手：

RTS（pin7）：发送数据请求(RequesttoSend)CTS（pin8）：清除发送(CleartoSend)

DSR（pin6）：数据发送就绪(DataSendReady)

DCD（pin1）：数据载波检测(DataCarrierDetect)

DTR（pin4）：数据终端就绪(DataTerminalReady)

地线：

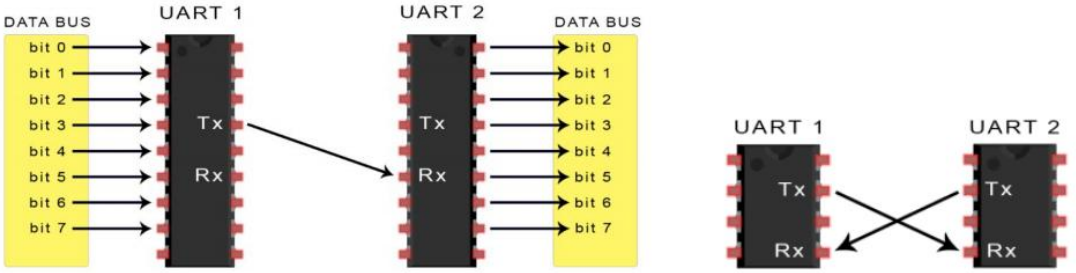
GND（pin5）：地线

其它：

RI（pin9）：铃声指示

通常我们用 RS232 串口仅用到了 9 根传输线中的三根：TXD，RXD，GND。但是对于数据传输，双方必须对数据传输采用使用相同的波特率，约定同样的传输模式（传输架构，握手条件等）。尽管这种方法对于大多数应用已经足够，但是对于接收方过载的情况这种使用受到限制。

RS232 的串口连接方式：



串口传输协议如下：

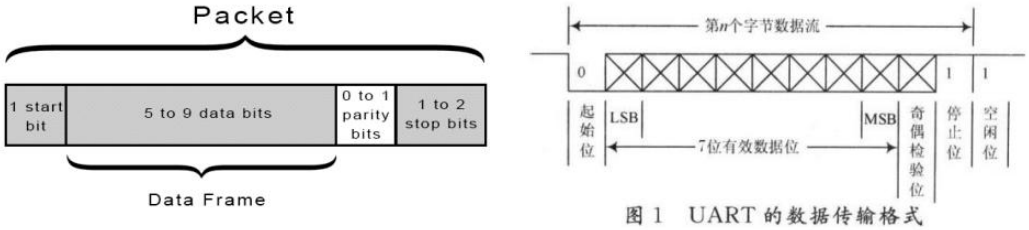


图 1　UART 的数据传输格式

起始位：先发出一个逻辑”0”信号，表示传输字符的开始。

数据位：可以是 5~8 位逻辑”0”或”1”。如 ASCII 码（7 位），扩展 BCD 码（8 位）。

校验位：数据位加上这一位后，使得“1”的位数应为偶数(偶校验)或奇数(奇

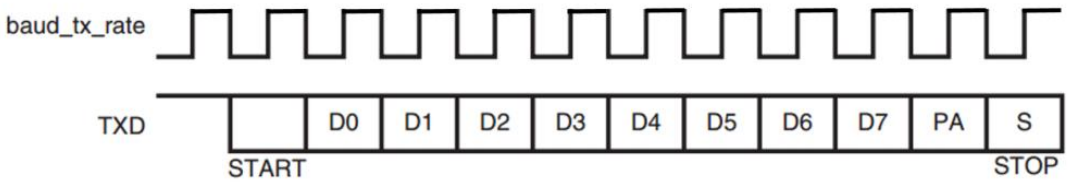
校验)。

停止位：它是一个字符数据的结束标志。可以是 1 位、1.5 位、2 位的高电平。

空闲位：处于逻辑“1”状态，表示当前线路上没有资料传送。

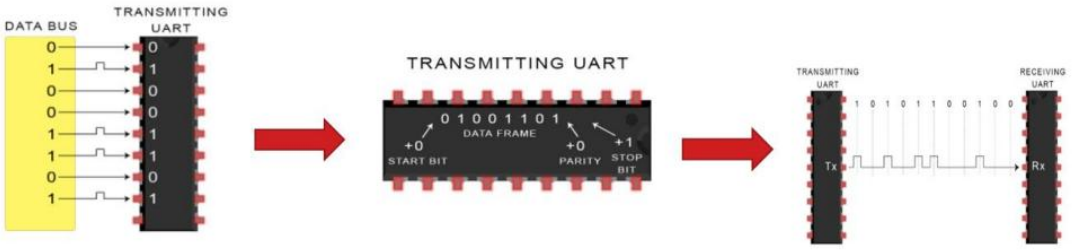
波特率：uart 中的波特率就可以认为是比特率，即每秒传输的位数(bit)。一般选波特率都会有 9600,19200,115200 等选项。其实意思就是每秒传输这么多个比特位数(bit)。

引入波特率的概念后可得到串口的传输节奏如下：

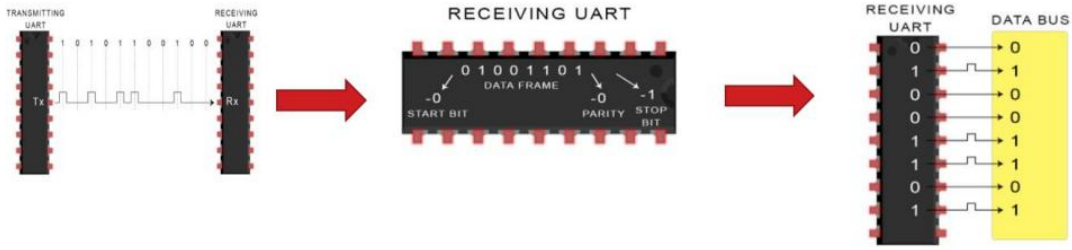


3.3.2 串口传输步骤

串口发送流程：



串口接收流程：



3.3.3 串口发送字符

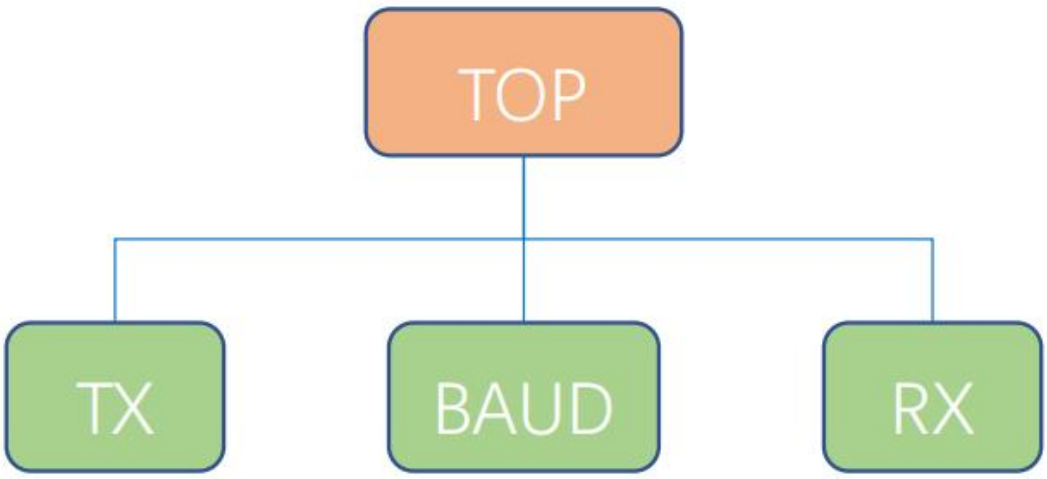
从前面串口协议中可以了解到串口每次传输可以以有 5～8bit 数据，在计算机中字符通常用 ASCII 码（7bit）表示，所以字符的发送可以用 ASCII 码发送。

查询 ASCII 码表格可得到：“www.meyesemi.com”用到的字符对应 ASCII 码；

```
8'h1  :  write_data <= `UD 8'h77;// ASCII code is w
8'h2  :  write_data <= `UD 8'h77;// ASCII code is w
8'h3  :  write_data <= `UD 8'h77;// ASCII code is w
8'h4  :  write_data <= `UD 8'h2E;// ASCII code is .
8'h5  :  write_data <= `UD 8'h6D;// ASCII code is m
8'h6  :  write_data <= `UD 8'h65;// ASCII code is e
8'h7  :  write_data <= `UD 8'h79;// ASCII code is y
8'h8  :  write_data <= `UD 8'h65;// ASCII code is e
8'h9  :  write_data <= `UD 8'h73;// ASCII code is s
8'ha  :  write_data <= `UD 8'h65;// ASCII code is e
8'hb  :  write_data <= `UD 8'h6D;// ASCII code is m
8'hc  :  write_data <= `UD 8'h69;// ASCII code is i
8'hd  :  write_data <= `UD 8'h2E;// ASCII code is .
8'he  :  write_data <= `UD 8'h63;// ASCII code is c
8'hf  :  write_data <= `UD 8'h6F;// ASCII code is o
8'h10 :  write_data <= `UD 8'h6D;// ASCII code is m
```

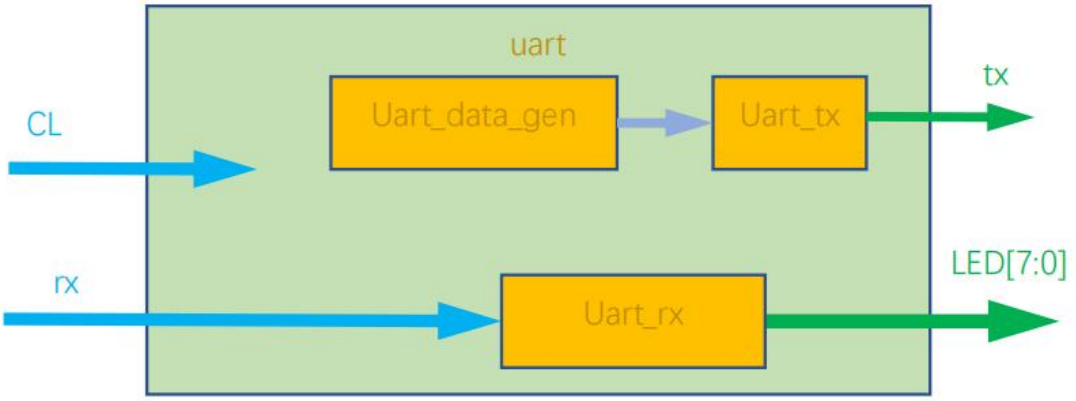
3.4 实验源码设计

从实验目的分析可将实验做如下划分：



从原理上分析波特率的计算是一个计数器，发射和接收可复用，我们在设计时为保持 TX，或 RX 的完整性，故将波特周期计数器集成在各自模块内部；

上述分析仅仅搭建好 MES50HP 的与 PC 通信的桥梁 UART，传输的数据没有体现。故而需要增加发送数据模块，与接收数据模块；



3.4.1 串口发送模块设计

目标：接收到一个发送命令信号时，将 data[7:0]-> 依次发出{start,data[0:7],stop}共 10bit 数据（无校验位，停止位 1bit）；

有两种方法可以将一个并行数据串行化；

方法一：通过 bit 计数与 baud 计数控制移位输出；

```
1.    // transmit bit
2.          always@(posedge clk)
3.          begin
4.             if(!rstn)
5.                 txd <= `UD 1'b1;
6.             else
7.             begin
8.                 if(trans_en)
9.                 Begin
10.   //   将开始标志和停止标志以及传输数据集成放到trans_data 中可用下方语句
11.   //               txd <= `UD trans_data[trans_bit];
12.   // 单 bit 控制用下方语句
13.                    case(trans_bit)
14.                       4'h0    :txd <= `UD 1'b0;
15.                        4'h1    :txd <= `UD tx_data_reg[0];
16.                        4'h2    :txd <= `UD tx_data_reg[1];
17.                        4'h3    :txd <= `UD tx_data_reg[2];
18.                        4'h4    :txd <= `UD tx_data_reg[3];
19.                        4'h5    :txd <= `UD tx_data_reg[4];
20.                        4'h6    :txd <= `UD tx_data_reg[5];
21.                        4'h7    :txd <= `UD tx_data_reg[6];
22.                        4'h8    :txd <= `UD tx_data_reg[7];
23.                       4'h9    :txd <= `UD 1'b1;
24.                        default :txd <= `UD 1'b1;
```

```
                endcase
            end
            else
                txd <= `UD 1'b1;
        end
    end
```

方法二：通过 bit 计数与 baud 计数控制状态跳转，在状态机中输出；

```
//  logical ouput  状态机输出
    always @ (posedge clk)
    begin
      if(tx_en)
      begin
          case(tx_state)
             IDLE      :  uart_tx  <= `UD 1'h1;  //空闲状态输出高电平
             SEND_START :  uart_tx <= `UD 1'h0; //start 状态发送一个波特周期的低电平
             SEND_DATA  :                        //发送状态每个波特周期发送一个 bit;
            begin
               case(tx_bit_cnt)
                   3'h0  :  uart_tx  <= `UD trans_data[0];
                   3'h1  :  uart_tx  <= `UD trans_data[1];
                   3'h2  :  uart_tx  <= `UD trans_data[2];
                   3'h3  :  uart_tx  <= `UD trans_data[3];
                   3'h4  :  uart_tx  <= `UD trans_data[4];
                   3'h5  :  uart_tx  <= `UD trans_data[5];
                   3'h6  :  uart_tx  <= `UD trans_data[6];
                   3'h7  :  uart_tx  <= `UD trans_data[7];
                   default: uart_tx  <= `UD 1'h1;
               endcase
            end
            SEND_STOP  :  uart_tx  <= `UD 1'h1; //发送停止状态 输出 1 个波特周期高电平
            default    :  uart_tx  <= `UD 1'h1; // 其他状态默认与空闲状态一致，保持高电平输出
        endcase
    end
    else
        uart_tx <= `UD 1'h1;
  end 30
```

这里笔者采用方法二，完整 module 设计如下：

```
`timescale 1ns / 1ps
```

```
2.  `define UD #1
3.
4.  module uart_tx #(
5.      parameter            BPS_NUM  =    16'd434
6.  //     设 置 波 特 率 为 4800 时 ， bit 位 宽 时 钟 周 期 个
    数:50MHz set 10417  40MHz set 8333
7.  //     设 置 波 特 率 为 9600 时 ， bit 位 宽 时 钟 周 期 个
    数:50MHz set 5208   40MHz set 4167
8.  //     设 置 波 特 率 为 115200 时 ， bit 位 宽 时 钟 周 期 个
    数:50MHz set 434  40MHz set 347 12M set 104
9.  )
10. (
11.     input           clk,          // clock
            时钟信号
12.     input [7:0]     tx_data,      // uart tx data signal byte;
            等待发送的字节数据
13.     input           tx_pluse,     // uart tx enable signal,rising i
    s active; 发送模块发送触发信号
14.
15.     output reg      uart_tx,      // uart tx transmit data line
            发送模块串口发送信号线
16.     output          tx_busy       // uart tx module work states,hig
    h is busy;发送模块忙状态指示
17. );
18.
19.      //=============================================================
    ===============
20.     //wire and reg in the module
21.      //=============================================================
    ===============
22.     reg              tx_pluse_reg =0;
23.
24.     reg [2:0]      tx_bit_cnt=0; //the bits number has transmited.
25.
26.     reg [2:0]       tx_state=0;  //current state of tx state machin
    e.
27.     reg [2:0]      tx_state_n=0; //next state of tx state machine.
28.
29.     reg [3:0]      pluse_delay_cnt=0;
30.     reg              tx_en = 0;
31.
32.     // uart tx state machine's state
33.      localparam  IDLE     = 4'h0; //tx state machine's state.空闲状
    态
```

```
    localparam  SEND_START = 4'h1; //tx state machine's state.发送start 状态
    localparam  SEND_DATA  = 4'h2; //tx state machine's state.发送数据状态
    localparam  SEND_STOP  = 4'h3; //tx state machine's state.发送stop 状态
    localparam  SEND_END   = 4'h4; //tx state machine's state.发送结束状态

    // uart bps set  the clk's frequency is 50MHz
    reg [15:0]   clk_div_cnt=0; //count for division the clock.

     //=============================================================
    //logic
     //=============================================================
    assign tx_busy = (tx_state != IDLE);
    //some control single.

    always @(posedge clk)
    begin
        tx_pluse_reg <= `UD tx_pluse;
    end

    // uart 模块发送工作使能标志信号
    always @(posedge clk)
    begin
        if(~tx_pluse_reg & tx_pluse)
            tx_en <= 1'b1;
        else if(tx_state == SEND_END)
            tx_en <= 1'b0;
    end

    //division the clock to satisfy baud rate.波特周期计数器
    always @ (posedge clk)
    begin
        if(clk_div_cnt == BPS_NUM || (~tx_pluse_reg & tx_pluse))
            clk_div_cnt   <= `UD 16'h0;
        else
            clk_div_cnt   <= `UD clk_div_cnt + 16'h1;
    end
```

```
    //count the number has transmited.发送数据状态中,发送bit 位计数,以波特周期累加
    always @ (posedge clk)
    begin
        if(!tx_en)
            tx_bit_cnt    <= `UD 3'h0;
        else if((tx_bit_cnt == 3'h7) && (clk_div_cnt == BPS_NUM))
            tx_bit_cnt    <= `UD 3'h0;
         else if((tx_state == SEND_DATA) && (clk_div_cnt == BPS_NUM))
            tx_bit_cnt    <= `UD tx_bit_cnt + 3'h1;
        else
            tx_bit_cnt    <= `UD tx_bit_cnt;
    end

     //============================================================
    //transmitter state machine
     //============================================================

    //   state change 状态跳转
    always @(posedge clk)
    begin
        tx_state <= tx_state_n;
    end

    //   state change condition 状态跳转条件及规律
    always @ (*)
    begin
      case(tx_state)
        IDLE    :
        begin
            if(~tx_pluse_reg & tx_pluse)   //触发发送做16个时钟周期延时后跳转到，发送start状态
                tx_state_n = SEND_START;
          else
            tx_state_n = tx_state;
      end
        SEND_START :
        begin
            if(clk_div_cnt == BPS_NUM)                //发送一个波特周期的低电平后进入，发送数据状态
                tx_state_n = SEND_DATA;
```

```
109.          else
110.           tx_state_n = tx_state;
111.      end
112.         SEND_DATA :
113.         begin
114.             if(tx_bit_cnt == 3'h7 && clk_div_cnt == BPS_NUM)
    //计时 8 个波特周期后（发送了 8bit 数据），跳转到发送 stop 状态
115.              tx_state_n = SEND_STOP;
116.          else
117.           tx_state_n = tx_state;
118.      end
119.         SEND_STOP :
120.         begin
121.             if(clk_div_cnt == BPS_NUM)              //设置停止位
    宽为 1 个波特周期，计数发送一个波特周期的高电平，之后跳转到发送结束状态
122.              tx_state_n = SEND_END;
123.          else
124.           tx_state_n = tx_state;
125.         end
126.         SEND_END :  tx_state_n = IDLE;
127.         default :  tx_state_n = IDLE;
128.       endcase
129.     end
130.
131.     //   logical ouput  状态机输出
132.     always @ (posedge clk)
133.     begin
134.       if(tx_en)
135.       begin
136.           case(tx_state)
137.               IDLE       :  uart_tx  <= `UD 1'h1;           //空
    闲状态输出高电平
138.               SEND_START :  uart_tx  <= `UD 1'h0;           //st
    art 状态发送一个波特周期的低电平
139.               SEND_DATA  :                                  //发
    送状态每个波特周期发送一个 bit；
140.               begin
141.                   case(tx_bit_cnt)
142.                       3'h0  :  uart_tx  <= `UD tx_data[0];
143.                       3'h1  :  uart_tx  <= `UD tx_data[1];
144.                       3'h2  :  uart_tx  <= `UD tx_data[2];
145.                       3'h3  :  uart_tx  <= `UD tx_data[3];
146.                       3'h4  :  uart_tx  <= `UD tx_data[4];
147.                       3'h5  :  uart_tx  <= `UD tx_data[5];
```

```
                    3'h6  :  uart_tx  <= `UD tx_data[6];
                    3'h7  :  uart_tx  <= `UD tx_data[7];
                    default: uart_tx  <= `UD 1'h1;
                endcase
            end
            SEND_STOP  :  uart_tx  <= `UD 1'h1;          //发送停止状态 输出1个波特周期高电平
             default    :  uart_tx  <= `UD 1'h1;          // 其他状态默认与空闲状态一致，保持高电平输出
          endcase
      end
      else
          uart_tx <= `UD 1'h1;
    end

    endmodule
```

3.4.2 串口接收模块设计

串口接收模块是发射模块的逆过程，设计思路区别不大，但是有如下几点需要注意：

1.接收开始信号，当 rx 下降沿到来后保持几个时钟周期的低电平，表明进入接收 start；

2.接收数据提取位置，前面讲发射的时候都是在波特周期开始的位置变更数据，接收数据提取时需要在 rx 稳定时刻取数，去波特周期的中间位置取数；

3.最终输出数据锁存，在最后 1bit 存入寄存器后需要对接收数据锁存，并在之后需要给出数据使能信号，表示输出数据有效；

Module 设计如下：

```
`timescale 1ns / 1ps
`define UD #1

module uart_rx # (
    parameter            BPS_NUM     =    16'd434
//  设置波特率为4800时，  bit位宽时钟周期个数:50MHz set 10417  40MHz set 8333
//  设置波特率为9600时，  bit位宽时钟周期个数:50MHz set 5208   40MHz set 4167
```

```
//  设置波特率为115200时，bit位宽时钟周期个数:50MHz set 434    40MHz set 347
)
(
      //input ports
      input             clk,
      input             uart_rx,

      //output ports
      output reg [7:0]  rx_data,
      output reg        rx_en,
      output            rx_finish
);

    // uart rx state machine's state
    localparam  IDLE         = 4'h0;    //空闲状态，等待开始信号到来.
    localparam  RECEIV_START = 4'h1;    //接收Uart开始信号，低电平一个波特周期.
    localparam  RECEIV_DATA  = 4'h2;    //接收Uart传输数据信号，此工程定义传输8bit，每个波特周期中间位置取值，8个周期后跳转到stop状态.
    localparam  RECEIV_STOP  = 4'h3;    //停止状态数据线是高电平，与空闲状态是一致的按照协议标准需要等待一个停止位周期再做状态跳转.
    localparam  RECEIV_END   = 4'h4;    //结束中转状态.

    //=============================================================
    //wire and reg in the module
    //=============================================================
    reg    [2:0]        rx_state=0;       //current state of tx state machine. 当前状态
    reg    [2:0]        rx_state_n=0;     //next state of tx state machine.    下一个状态
    reg    [7:0]        rx_data_reg;      //
                接收数据缓冲寄存器
    reg                 uart_rx_1d;       //save uart_rx one cycle.            保存uart_rx一个时钟周期
    reg                 uart_rx_2d;       //save uart_rx one cycle.保存uart_rx 前两个时钟周期
    wire                start;            //active when start a byte receive.  检测到start信号标志
    reg    [15:0]       clk_div_cnt;      //count for division the clock.      波特周期计数器
```

```
38.
39.     //============================================================
    ===============
40.     //logic
41.     //============================================================
    ===============
42.
43.     //some control single.
44.     always @ (posedge clk)
45.     begin
46.          uart_rx_1d <= `UD uart_rx;
47.          uart_rx_2d <= `UD  uart_rx_1d;
48.     end
49.
50.     assign start     = (!uart_rx) && (uart_rx_1d || uart_rx_2d);
51.     assign rx_finish = (rx_state == RECEIV_END);
52.
53.
54.     //division the clock to satisfy baud rate.波特周期计数器
55.     always @ (posedge clk)
56.     begin
57.         if(rx_state == IDLE || clk_div_cnt == BPS_NUM)
58.             clk_div_cnt   <= `UD 16'h0;
59.         else
60.             clk_div_cnt   <= `UD clk_div_cnt + 16'h1;
61.     end
62.
63.     // receive bit data numbers
64.     //在接收数据状态中，接收的bit位计数，每一个波特周期计数加1
65.     reg    [2:0]      rx_bit_cnt=0;    //the bits number has tran
    smited.
66.     always @ (posedge clk)
67.     begin
68.         if(rx_state == IDLE)
69.             rx_bit_cnt <= `UD 3'h0;
70.         else if((rx_bit_cnt == 3'h7) && (clk_div_cnt == BPS_NUM))
71.             rx_bit_cnt <= `UD 3'h0;
72.         else if((rx_state == RECEIV_DATA) && (clk_div_cnt == BPS_
    NUM))
73.             rx_bit_cnt <= `UD rx_bit_cnt + 3'h1;
74.         else
75.             rx_bit_cnt <= `UD rx_bit_cnt;
76.     end
77.
```

```
//=============================================================================
//receive state machine
//=============================================================================
    //状态机状态跳转
    always @(posedge clk)
    begin
        rx_state <= rx_state_n;
    end

    //状态机状态跳转条件及跳转规律
    always @ (*)
    begin
      case(rx_state)
          IDLE       :
          begin
              if(start)                                    //监测到start 信号到来，下一状态跳转到start 状态
                  rx_state_n = RECEIV_START;
              else
                  rx_state_n = rx_state;
          end
          RECEIV_START    :
          begin
               if(clk_div_cnt == BPS_NUM)                  //已完成接收start 标志信号
                   rx_state_n = RECEIV_DATA;
               else
                   rx_state_n = rx_state;
           end
          RECEIV_DATA    :
          begin
              if(rx_bit_cnt == 3'h7 && clk_div_cnt == BPS_NUM) //已完成8bit 数据的传输
                  rx_state_n = RECEIV_STOP;
              else
                  rx_state_n = rx_state;
           end
          RECEIV_STOP    :
          begin
              if(clk_div_cnt == BPS_NUM)                   //已完成接收stop 标志信号
                  rx_state_n = RECEIV_END;
```

```
116.                 else
117.                     rx_state_n = rx_state;
118.             end
119.             RECEIV_END    :
120.             begin
121.                 if(!uart_rx_1d)                                    //数据线重新被拉低，表示新数据传输又发送 start 标志信号，需要跳转到 start 状态
122.                     rx_state_n = RECEIV_START;
123.                 else                                               //没有其他状况出现时，回到空闲状态，等待 start 信号的到来
124.                     rx_state_n = IDLE;
125.             end
126.             default    :  rx_state_n = IDLE;
127.         endcase
128.     end
129.
130.     // 状态机输出
131.     always @ (posedge clk)
132.     begin
133.         case(rx_state)
134.             IDLE         ,
135.             RECEIV_START :                                 //在空闲和 start 状态时将接收数据缓冲寄存器和数据使能置位；
136.             begin
137.                 rx_en <= `UD 1'b0;
138.                 rx_data_reg <= `UD 8'h0;
139.             end
140.             RECEIV_DATA  :
141.             begin
142.                 if(clk_div_cnt == BPS_NUM[15:1])          //在一个波特周期的中间位置取数据线上传输的数据；
143.                     rx_data_reg  <= `UD {uart_rx , rx_data_reg[7:1]};  //以循环右移的方式将 uart_rx 数据填入缓冲寄存器的最高位（Uart 传输低位在前，最后一个 bit 刚好是最高位）
144.             end
145.             RECEIV_STOP  :
146.             begin
147.                 rx_en   <= `UD 1'b1;                       // 输出使能信号，表示最新的数据输出有效
148.                 rx_data <= `UD rx_data_reg;                // 将缓冲寄存器的值赋值给输出寄存器
149.             end
150.             RECEIV_END    :
```

```
            begin
                rx_data_reg <= `UD 8'h0;
            end
            default:    rx_en <= `UD 1'b0;
        endcase
    end

endmodule
```

3.4.3 串口发送控制模块设计

目标：产生 1S 间隔的触发信号并输出第一个发送字节，busy 的下降沿时输出下一个字节；

Module 如下：

```
`timescale 1ns / 1ps
`define UD #1
module uart_data_gen(
    input               clk,
    input      [7:0]    read_data,
    input               tx_busy,
    input      [7:0]    write_max_num,
    output reg [7:0]    write_data,
    output reg          write_en
);

    // set every second send a string,"====HELLO WORLD==="
    // 设置约每秒发送一个字符串
    reg [25:0] time_cnt=0;
    reg [ 7:0] data_num;
    always @(posedge clk)
    begin
        time_cnt <= `UD time_cnt + 26'd1;
    end

    // 设置串口发射工作区间
    reg        work_en=0;
    reg        work_en_1d=0;
    always @(posedge clk)
```

```
    begin
        if(time_cnt == 26'd2048)
            work_en <= `UD 1'b1;
        else if(data_num == write_max_num-1'b1)
            work_en <= `UD 1'b0;
    end

    always @(posedge clk)
    begin
        work_en_1d <= `UD work_en;
    end

    // get the tx_busy's falling edge    获取tx_busy的下降沿
    reg             tx_busy_reg=0;
    wire            tx_busy_f;
    always @ (posedge clk) tx_busy_reg <= `UD tx_busy;

    assign tx_busy_f = (!tx_busy) && (tx_busy_reg);

    // 串口发射数据触发信号
    reg write_pluse;
    always @ (posedge clk)
    begin
        if(work_en)
        begin
            if(~work_en_1d || tx_busy_f)
                write_pluse <= `UD 1'b1;
            else
                write_pluse <= `UD 1'b0;
        end
        else
            write_pluse <= `UD 1'b0;
    end

    always @ (posedge clk)
    begin
        if(~work_en & tx_busy_f)
            data_num   <= 7'h0;
        else if(write_pluse)
            data_num   <= data_num + 8'h1;
    end

    always @(posedge clk)
    begin
```

```
        write_en <= `UD write_pluse;
    end

//  字符的对应ASCII码
    always @ (posedge clk)
    begin
        case(data_num)
            8'h0  ,
            8'h1  : write_data <= `UD 8'h77;// ASCII code is w
            8'h2  : write_data <= `UD 8'h77;// ASCII code is w
            8'h3  : write_data <= `UD 8'h77;// ASCII code is w
            8'h4  : write_data <= `UD 8'h2E;// ASCII code is .
            8'h5  : write_data <= `UD 8'h6D;// ASCII code is m
            8'h6  : write_data <= `UD 8'h65;// ASCII code is e
            8'h7  : write_data <= `UD 8'h79;// ASCII code is y
            8'h8  : write_data <= `UD 8'h65;// ASCII code is e
            8'h9  : write_data <= `UD 8'h73;// ASCII code is s
            8'ha  : write_data <= `UD 8'h65;// ASCII code is e
            8'hb  : write_data <= `UD 8'h6D;// ASCII code is m
            8'hc  : write_data <= `UD 8'h69;// ASCII code is i
            8'hd  : write_data <= `UD 8'h2E;// ASCII code is .
             8'he  : write_data <= `UD 8'h63;// ASCII code is c

            8'hf  : write_data <= `UD 8'h6F;// ASCII code is o
            8'h10 : write_data <= `UD 8'h6D;// ASCII code is m
            8'h11 ,
            8'h12 : write_data <= `UD 8'h0d;
            8'h13 : write_data <= `UD 8'h0a;
            default : write_data <= `UD read_data;
        endcase
    end

endmodule
```

3.4.4 串口实验顶层模块设计

目标：板子 1s 向串口助手发送一次十进制显示的“www.meyesemi.com”，通过串口助手向板子以十六进制形式发送数字，LED 以二进制显示亮起。

Uart_data_gen 模块产生一个间隔 1S 钟的触发信号，同时输出第一个发送字节，等待 uart_tx 输出的 busy 下降沿到来，获知 uart_tx 进入空闲状态可发送下一

个 byte 时，再次给出串口发送的触发脉冲，并输出下一个字节；

Uart_rx 模块接收到数据后输出一个 rx_en 信号（接收数据使能信号）、一组接收数据信号；接收的数据信号是锁存的，可直接点亮 LED 灯；

具体的 module 实现如下:

```
1.  `timescale 1ns / 1ps
2.  `define UD #1
3.
4.  module uart_top(
5.      //input ports
6.      input          clk,
7.      input          uart_rx,
8.
9.      //output ports
10.     output  [7:0] led,
11.     output         uart_tx
12. );
13.
14.    parameter       BPS_NUM = 16'd434;
15.        //  设 置 波 特 率 为 4800 时 ， bit 位 宽 时 钟 周 期 个数:50MHz set 10417  40MHz set 8333
16.        //  设 置 波 特 率 为 9600 时 ， bit 位 宽 时 钟 周 期 个数:50MHz set 5208   40MHz set 4167
17.        //  设 置 波 特 率 为 115200 时 ， bit 位 宽 时 钟 周 期 个数:50MHz set 434    40MHz set 347 12M set 104
18.
19.
20. //====================
21. //wire and reg in the module
22. //====================
23.
24.     wire            tx_busy;    //transmitter is free.
25.     wire            rx_finish;   //receiver is free.
26.     wire   [7:0] rx_data;    //the data receive from uart_rx.
27.     wire   [7:0] tx_data;
28.     wire            tx_en;       //enable transmit.
29.
30. //======================
31. //Logic
32. //======================
```

```
    wire rx_en;
//=======================
//instance
//=======================
    reg  [7:0] receive_data;
    always @(posedge clk)  receive_data <= led;
    uart_data_gen uart_data_gen(
        .clk                    (clk      ),//input clk,
        .read_data          (receive_data ),//input [7:0] read_data,
        .tx_busy              (tx_busy      ),   //input tx_busy,
        .write_max_num  (8'h14        ),    //input [7:0]  write_max_num,
        .write_data          (tx_data      ),   //output reg [7:0]  write_data
        .write_en             (tx_en        )     //output reg write_en
    );

    //uart transmit data module.
    uart_tx #(
         .BPS_NUM           (  BPS_NUM       ) //parameter BPS_NUM  =    16'd434
     )
     u_uart_tx(
        .clk                (  clk         ),// input clk,
        .tx_data            (  tx_data       ),// input [7:0] tx_data,
        .tx_pluse           (  tx_en         ),// input tx_pluse,
        .uart_tx            (  uart_tx       ),// output reg uart_tx,
        .tx_busy            (  tx_busy       ) // output tx_busy
    );

    //Uart receive data module.
    uart_rx #(
         .BPS_NUM           (  BPS_NUM       ) //parameter  BPS_NUM  =    16'd434
     )
```

```
    u_uart_rx (
        .clk                (  clk          ),// input clk,
        .uart_rx            (  uart_rx      ),// input uart_rx,
        .rx_data            (  rx_data      ),// output reg [7:0] rx_data,
        .rx_en              (  rx_en        ),// output reg rx_en,
        .rx_finish          (  rx_finish    ) // output rx_finish
    );

    assign led = rx_data;

endmodule

```

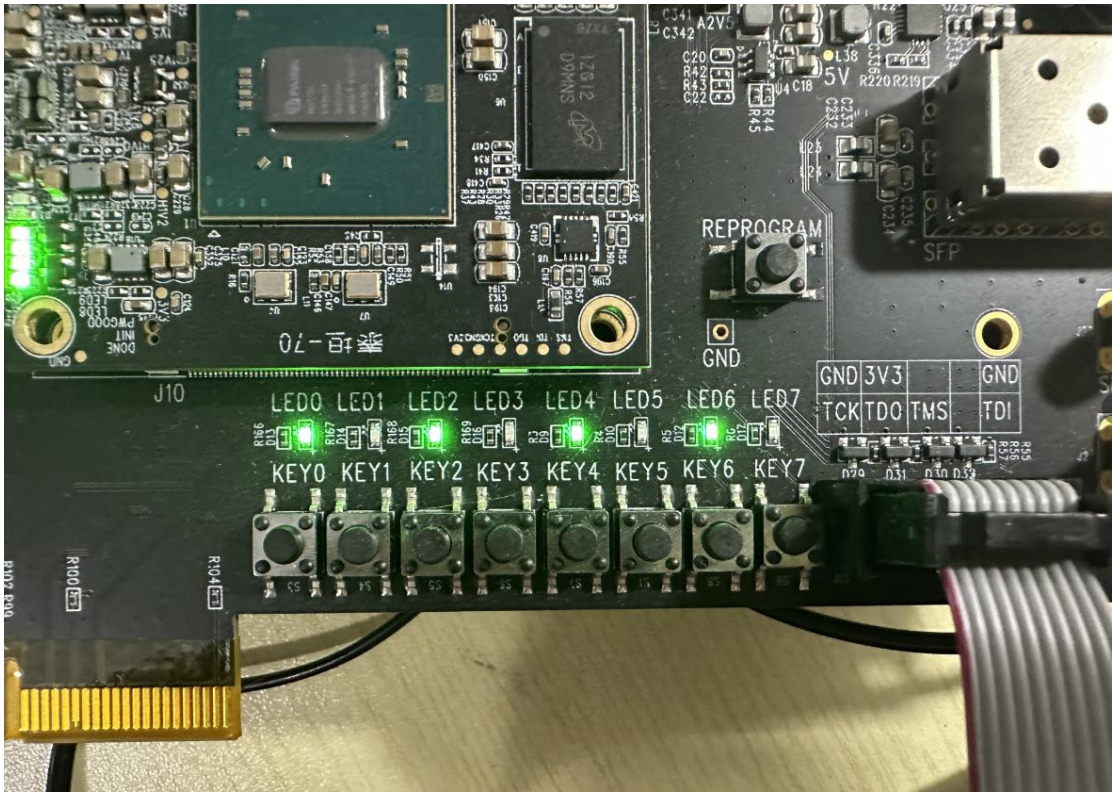
3.5 实验现象

用 SSCOM 串口调试工具，波特率设置为 115200bps，数据格式为 1 位起始位、8 位数据位、无校验位、1 位结束位，用 Type-C 连接开发板与电脑后有如下现象：

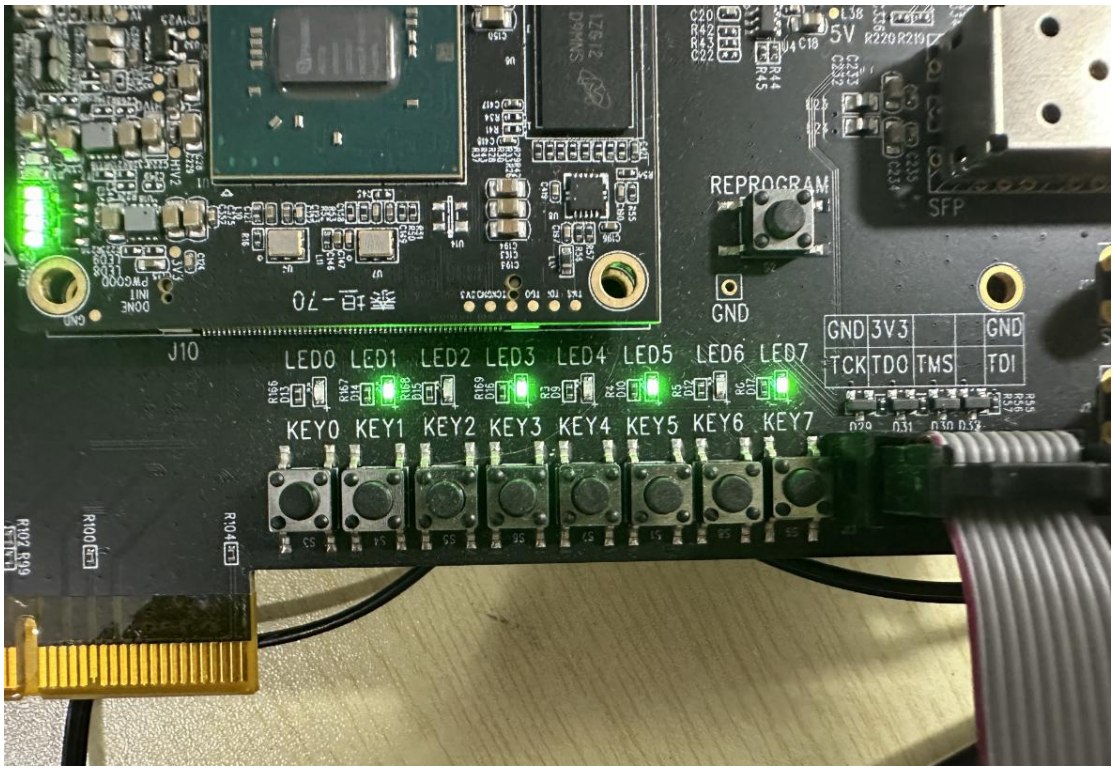
实验现象一：在串口工具中每隔 1S 中打印一次：“www.meyesemi.com”；

实验现象二：

在串口工具上以 Hex 格式发送 55；我们可看到 PG2T70H 板卡上的 LED1,LED3,LED5,LED7 被点亮，LED2,LED4,LED6,LED8 为熄灭状态；



发送 AA；我们可看到 PG2T70H 板卡上的 LED2,LED4,LED6,LED8 被点亮，LED1,LED3,LED5,LED7 为熄灭状态。



也可以试着发送其他数据（00~FF）,看一下 LED 灯的变化；

4_5.HDMI 实验例程说明

4_5.1PG2T70H 开发板简介

HDMI 输入接口采用宏晶微 MS7200HMDI 接收芯片，HDMI 输出接口采用宏晶微 MS7210HMDI 发送芯片。芯片兼容 HDMI1.4b 及以下标准视频的 3D 传输格式，最高分辨率高达 4K@30Hz，最高采样率达到 300MHz，支持 YUV 和 RGB 之间的色彩空间转换，数字接口支持 YUV 及 RGB 格式。

MS7200 和 MS7210 的 IIC 配置接口与 FPGA 的 IO 相连，通过 FPGA 的编程来对芯片进行初始化和配置操作。

PG2T70H 开发板上将 MS7200 的 SA 管脚下拉到地，故 IIC 的 ID 地址为 0x56，将 MS7210 的 SA 管脚上拉到电源电压，故 IIC 的 ID 地址为 0xB2（详情请查看“PG2T70H 开发板硬件使用手册”）。

4_5.2 实验目的

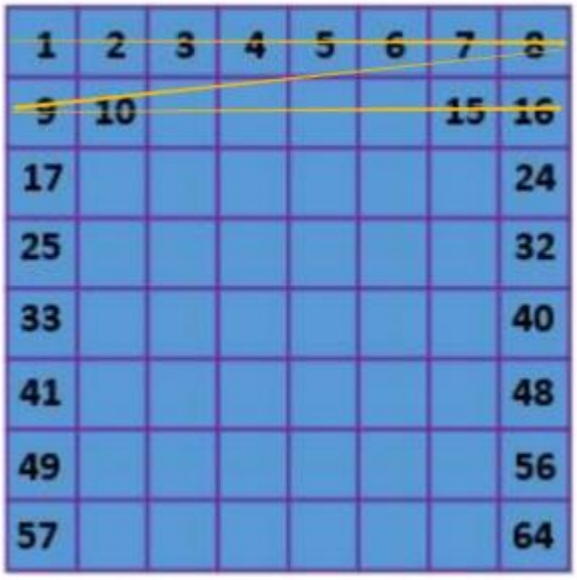
实验 1：MES50HP 开发板通过 HDMI 在屏幕上显示彩条；

实验 2.MES50HP 开发板 HDMIIN 接收，通过 HDMIOUT 实现环路输出；

4_5.3 实验原理

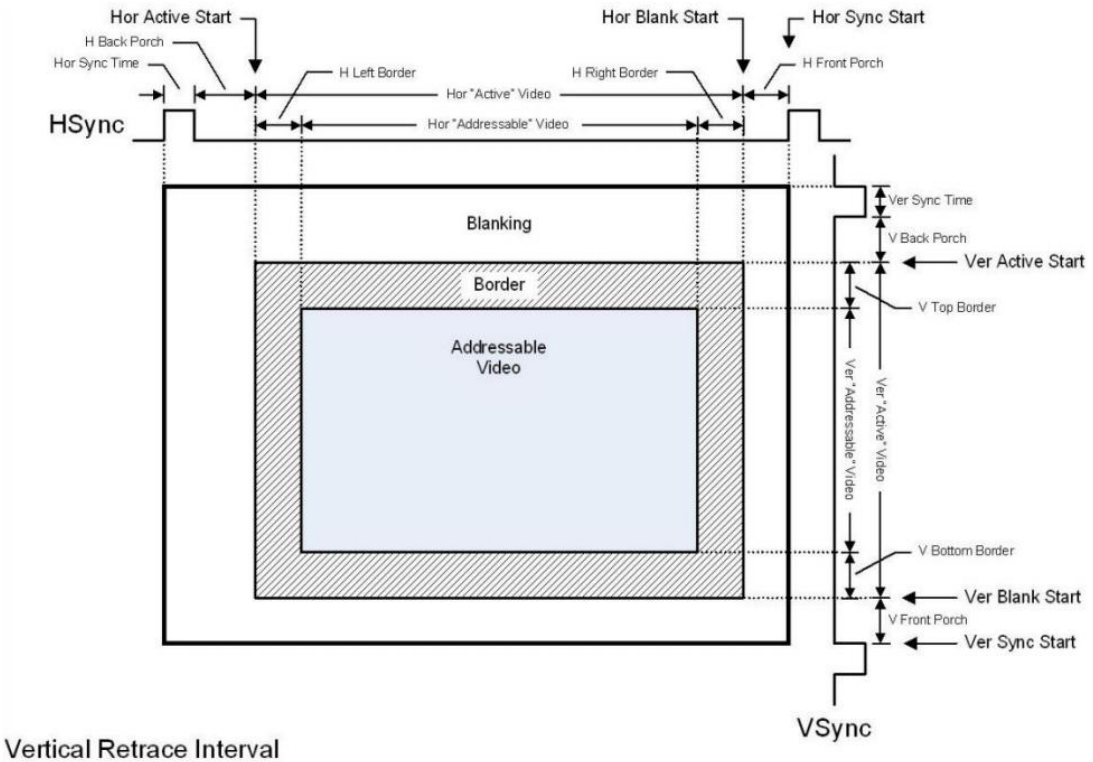
4_5.3.1 显示原理

下图表示一个 8*8 像素的画面，图中每个格子表示一个像素点，显示图像时像素点快速点亮的过程按表格中编号的顺序逐个点亮，从左到右，从上到下，按图中箭头方向的“Z”字形顺序。



以上图为例，每行 8 个像素点，每完成一行信号的传输，会转到下一行信号传输，直到完成第 8 行数据的传输，就完成了一个画面的数据传输了，一个画面也称为一场或一帧，显示每秒中刷新的帧数称为帧率。比如 1920*1080P 像素，就是 1 行有效像素点 1920，一场有效行为 1080 行。

每个像素点的像素值数据，对应每个像素点的颜色。常见的像素值表示格式比如：RGB888，RGB 分别代表：红 R,绿 G，蓝 B，888 是指 R、G、B 分别有 8bit，也就是 R、G、B 每一色光有 28=256 级阶调，通过 RGB 三色光的不同组合，一个像素上最多可显示 24 位的 256*256*256=16,777,216 色。

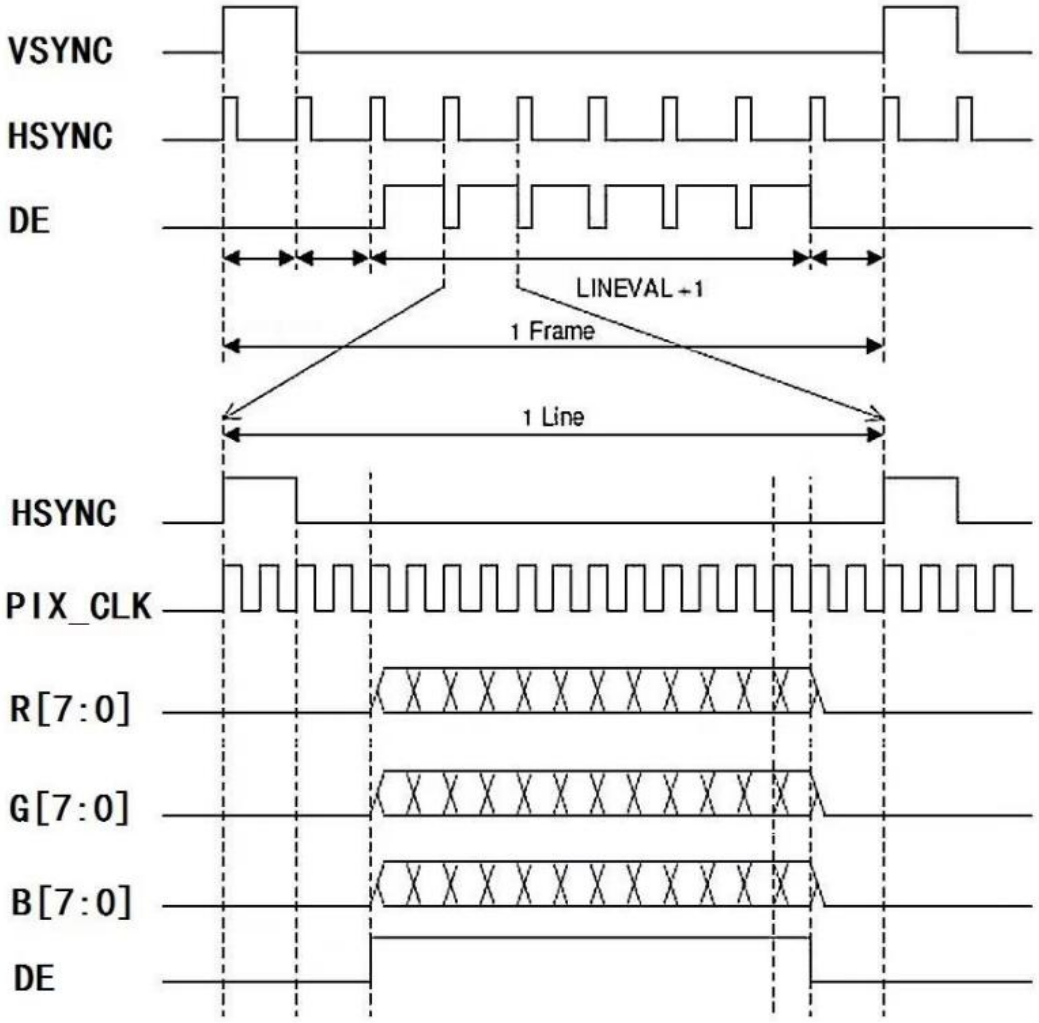


Vertical Retrace Interval

像素数据源源不断输送进来，行、场的切换通过行场同步信号来控制，即 hsync（行同步）和 vsync（场同步信号）。

上图中 Addressable 部分内容是在显示器中可看到的区域，像素点是否有效通过 DE 信号标识；Border 可理解为显示黑边或者显示边框，通常 Border 显示的像素值是 0（黑色）。行、场切换过程都是在用户感受不到的区域进行的，这个区域就是 Blanking 部分，称为消隐区间。同步信号上升沿表示新的一行/一场开始，Hsync 对应行，Vsync 对应场。

彩条产生：



本实验采用 1920*1080@60 的视频规格，详细时序参数如下：

VESA MONITOR TIMING STANDARD

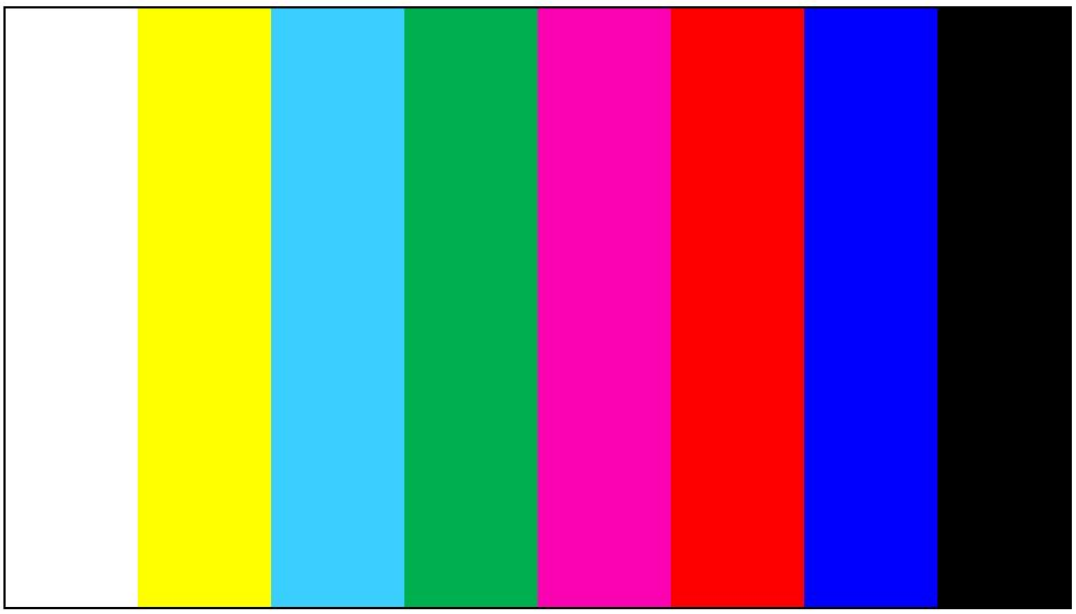
Adopted: 11/17/08
Resolution: 1920 x 1080 at 60 Hz (non-interlaced)
EDID ID: DMT ID: 52h; STD 2 Byte Code: (D1, C0)h; CVT 3 Byte Code: n/a
Method: *** NOT CVT COMPLIANT ***
Per CEA-861 --- 1080p (Code 16) Timing Definition

Detailed Timing Parameters

Timing Name	= **1920 x 1080 @ 60Hz;**						
Hor Pixels	= **1920;**	// Pixels					
Ver Pixels	= **1080;**	// Lines					
Hor Frequency	= 67.500;	// kHz	=	14.8 usec	/	line	
Ver Frequency	= 60.000;	// Hz	=	16.7 msec	/	frame	
Pixel Clock	= **148.500;**	// MHz	=	6.7 nsec	± 0.5%		
Character Width	= **4;**	// Pixels	=	26.9 nsec			
Scan Type	= **NONINTERLACED;**			// H Phase	=	1.4 %	
Hor Sync Polarity	= **POSITIVE**	// HBlank	=	12.7% of HTotal			
Ver Sync Polarity	= **POSITIVE**	// VBlank	=	4.0% of VTotal			
Hor Total Time	= 14.815;	// (usec)	=	550 chars	=	2200 Pixels	
Hor Addr Time	= 12.929;	// (usec)	=	480 chars	=	1920 Pixels	
Hor Blank Start	= 12.929;	// (usec)	=	480 chars	=	1920 Pixels	
Hor Blank Time	= 1.886;	// (usec)	=	70 chars	=	280 Pixels	
Hor Sync Start	= 13.522;	// (usec)	=	502 chars	=	2008 Pixels	
// H Right Border	= 0.000;	// (usec)	=	**0** chars	=	0 Pixels	
// H Front Porch	= 0.593;	// (usec)	=	**22** chars	=	88 Pixels	
Hor Sync Time	= 0.296;	// (usec)	=	**11** chars	=	44 Pixels	
// H Back Porch	= 0.997;	// (usec)	=	**37** chars	=	148 Pixels	
// H Left Border	= 0.000;	// (usec)	=	**0** chars	=	0 Pixels	
Ver Total Time	= 16.667;	// (msec)	=	1125 lines	HT – (1.06xHA)		
Ver Addr Time	= 16.000;	// (msec)	=	1080 lines	= 1.11		
Ver Blank Start	= 16.000;	// (msec)	=	1080 lines			
Ver Blank Time	= 0.667;	// (msec)	=	45 lines			
Ver Sync Start	= 16.059;	// (msec)	=	1084 lines			
// V Bottom Border	= 0.000;	// (msec)	=	**0** lines			
// V Front Porch	= 0.059;	// (msec)	=	**4** lines			
Ver Sync Time	= 0.074;	// (msec)	=	**5** lines			
// V Back Porch	= 0.533;	// (msec)	=	**36** lines			
// V Top Border	= 0.000;	// (msec)	=	**0** lines			

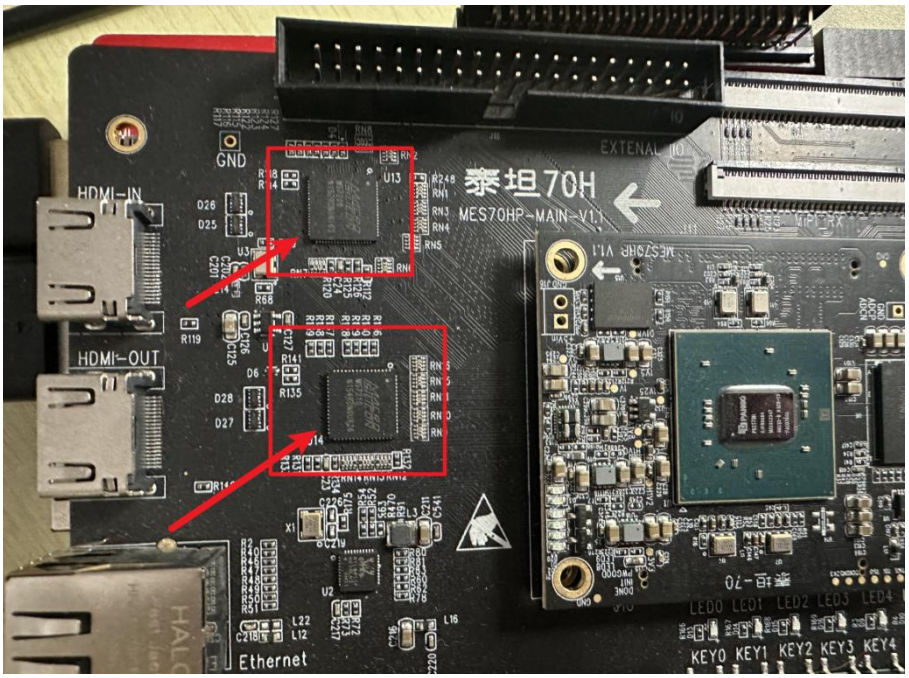
HDMI 显示的数据源采用 verilog 编写的显示时序产生模块 sync_vg 实现上图的时序，彩条生成模块 pattern_vg 根据像素点所在位置，即列数和行数确定像素值，实现彩条图案。

彩条按照每行均匀分成 8 部分，根据每行的像素点数的范围对像素值设置成对应的颜色，实现彩条信号。



4_5.3.2HDMI_PHY 配置

MS7200 为 HMDI 接收芯片，MS7210 为 HMDI 发送芯片，芯片的 IIC 配置接口已与 FPGA 的 IO 相连，芯片正常使用需要通过 FPGA 的对芯片进行初始化和配置操作。

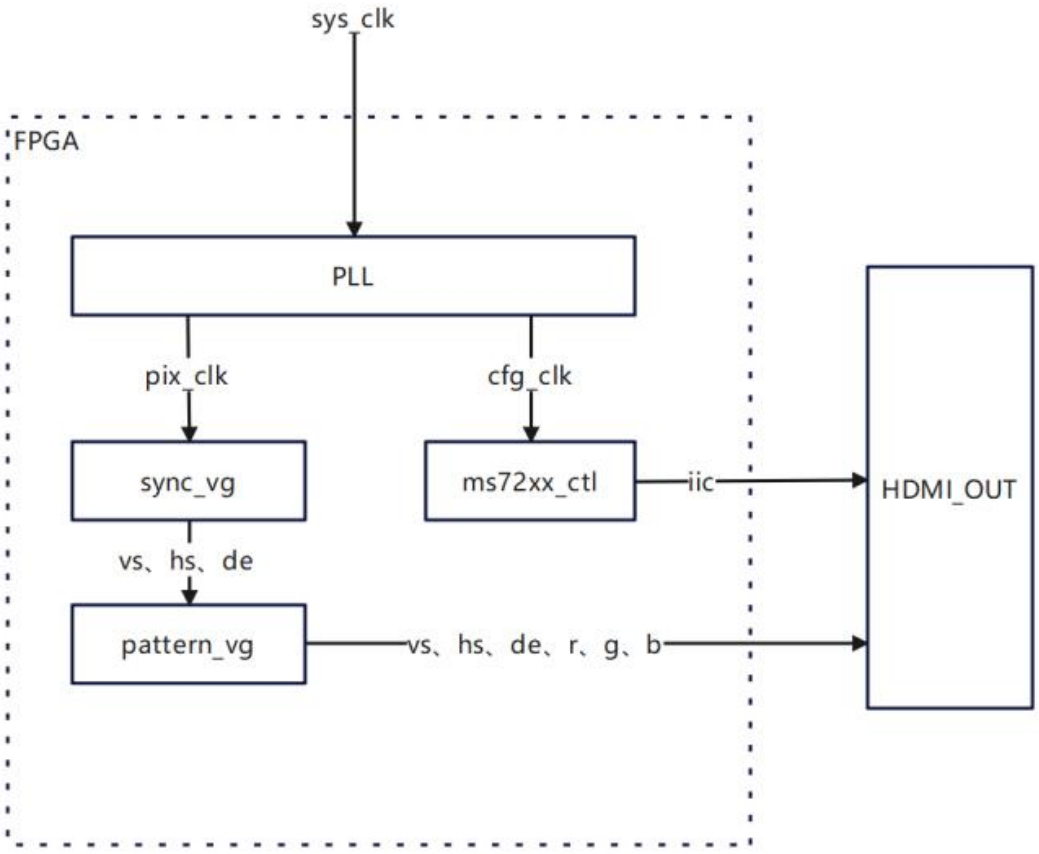


hdmi_loop 例程包含对 MS7200 和 MS7210 的配置模块 ms72xx_ctl，已将 MS7200 和 MS7210 配置成 1920*1080@60RGB888 模式，配置流程参考源码，用户可例化模块 ms72xx_ctl 使用。

4_5.4 实验源码设计

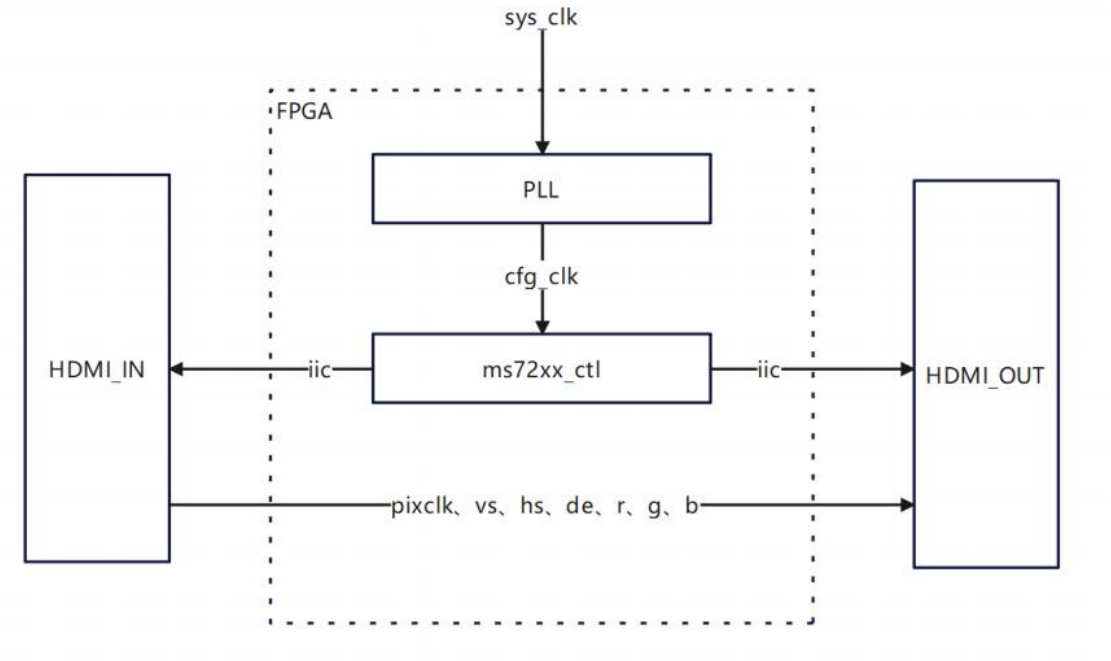
实验 1：hdmi_test

HDMI 输出彩条显示例程，分成 4 个模块，时钟模块 pll、MS7210 配置模块 ms72xx_ctl、显示时序产生模块 sync_vg、彩条生成模块 pattern_vg，以下为模块拓扑图，源码详情请查看 demo。



实验 2：hdmi_loop

HDMI 环路例程，将 MS7200 接收的信号给到 MS7210 即可实现 HDMI 环路输出，以下为模块拓扑图，源码详情请查看 demo。



4_5.5 实验现象

实验 1 现象：hdmi_test

连接好 PG2T70H 开发板和显示器，下载程序，可以看到显示器显示 8 条彩条。

实验 2 现象：hdmi_loop

连接好 PG2T70H 开发板、视频源和显示器，注意视频源必须为 1920*1080P@60，下图为设置分辨率步骤，下载程序，可以看到显示器显示与视频源一致的图像。


系统 › 屏幕 › 高级显示器设置
选择一个显示器以查看或更改其设置
显示器 2: U28H75x
显示器信息
U28H75x
显示器 2: 已连接到 AMD Radeon(TM) Graphics
桌面模式 1920 × 1080，60 Hz
活动信号模式 1920 × 1080，60 Hz
位深度 8 位
颜色格式 YCbCr444
颜色空间 标准动态范围(SDR)
显示器 2 的显示适配器属性
活动信号为实际输出；
可在此处设置分辨率
选择刷新率
较高的速率可提供更流畅的动作，但也使用更多电源 有关刷新率的详细信息
60 Hz

6.DDR3 读写实验例程

6.1PG2T70H 开发板简介

PG2T70H 开发板集成两颗 4Gbit（512MB）DDR3 芯片,型号为 MT41K256M16。DDR3 的总线宽度共为 32bit。DDR3SDRAM 的最高数据速率 800Mbps（详情请查看“PG2T70H 开发板硬件使用手册”）。

6.2 实验要求

生成 DDR3IP 官方例程，实现 DDR3 的读写控制，了解其工作原理和用户接口。

6.3DDR3 控制器简介

PG2T70H 为用户提供一套完整的 DDRmemory 控制器解决方案，配置方式比较灵活，采用软核实现 DDRmemory 的控制，有如下特点：

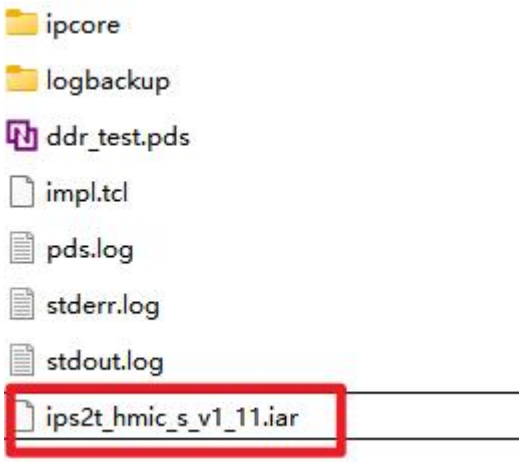
- 支持 DDR3
- 支持 x8、x16MemoryDevice
- 最大位宽支持 32bit
- 支持裁剪的 AXI4 总线协议
- 一个 AXI4256bitHostPort
- 支持 Self_refresh，Powerdown
- 支持 BypassDDRC
- 支持 DDR3WriteLeveling 和 DQSGateTraining
- DDR3 最快速率达 800Mbps

6.4 实验设计

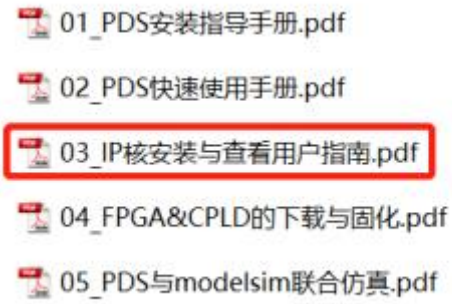
6.4.1 安装 DDR3IP 核

PDS 安装后，需手动添加 DDR3IP，请按以下步骤完成：

（1）DDR3IP 文件：07_ddr_test_1500\ips2t_hmic_s_v1_11.iar

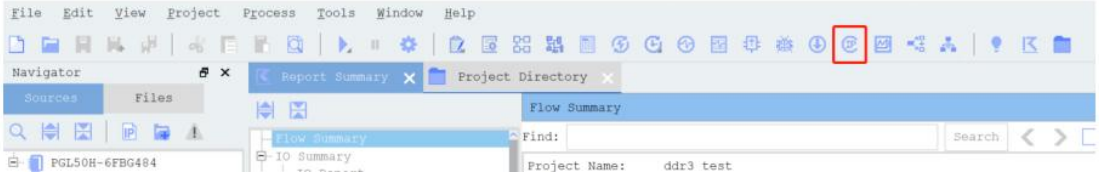


（2）IP 安装步骤：IP 核安装与查看用户指南.pdf

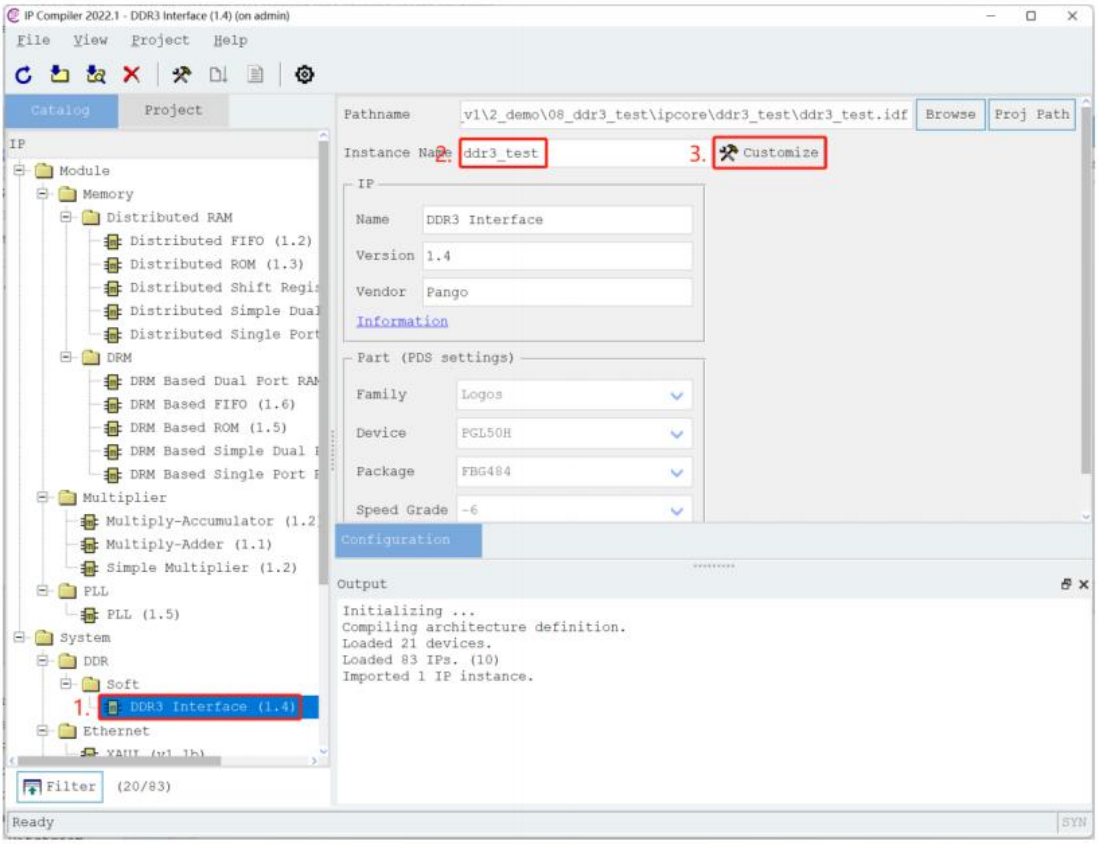


6.4.2DDR3 读写 Example 工程

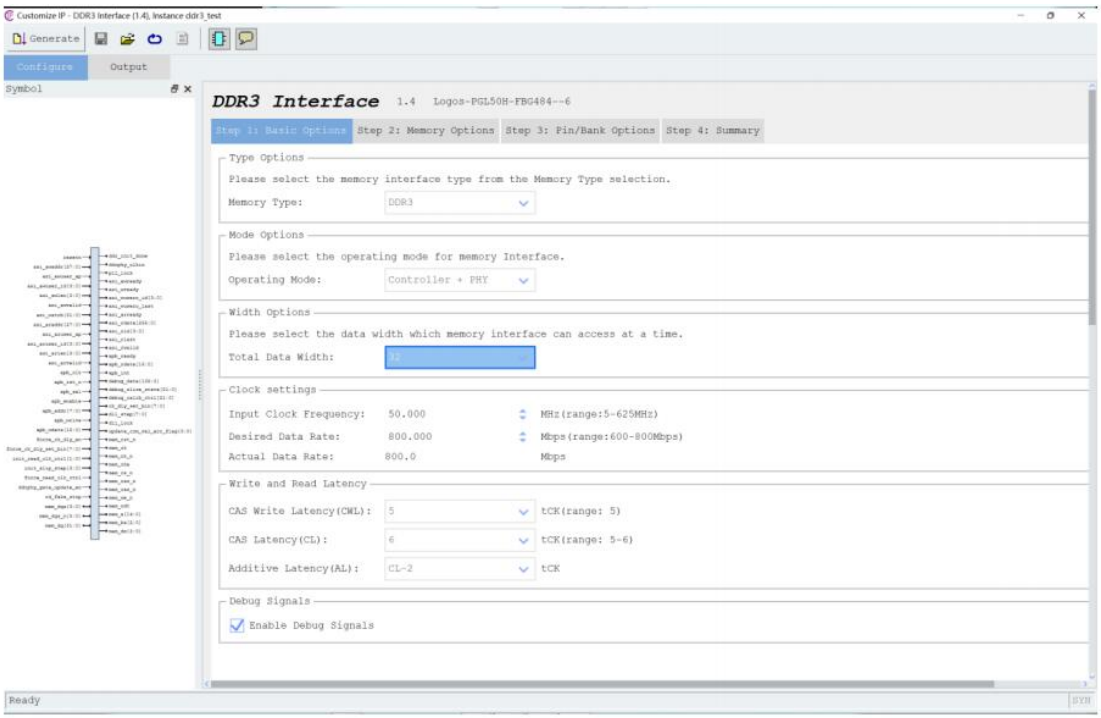
1.打开 PDS 软件，新建工程 ddr3_test，点开如下图标，打开 IPCompiler；



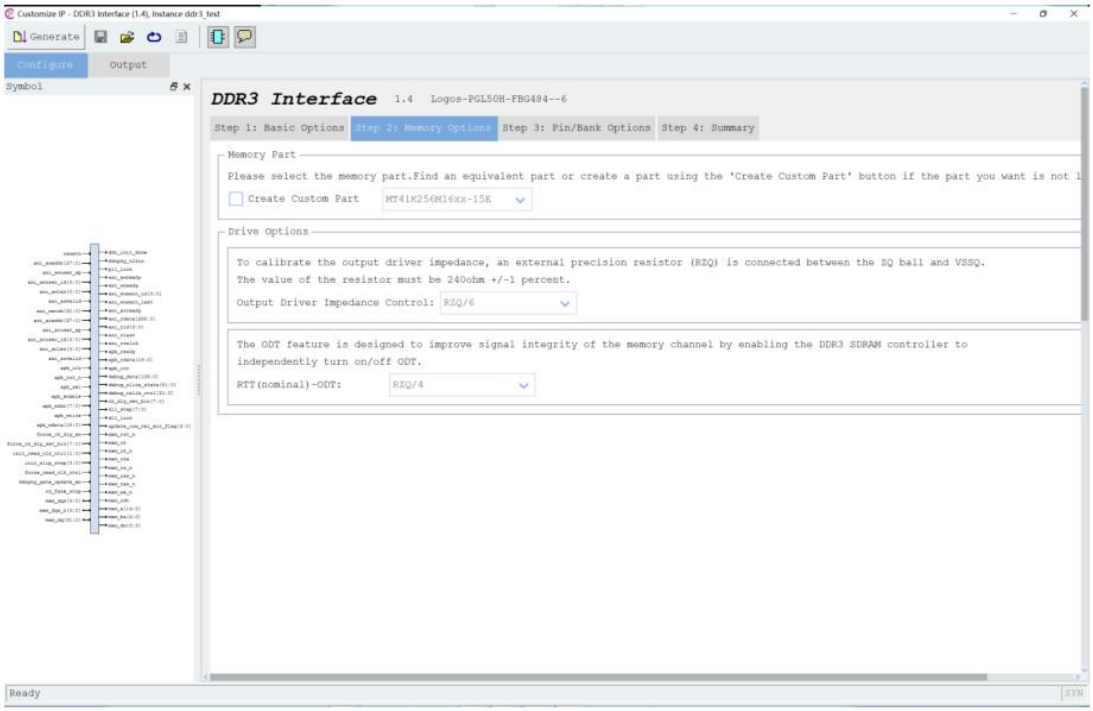
2.选择 DDR3IP，取名，然后点击 Customize；



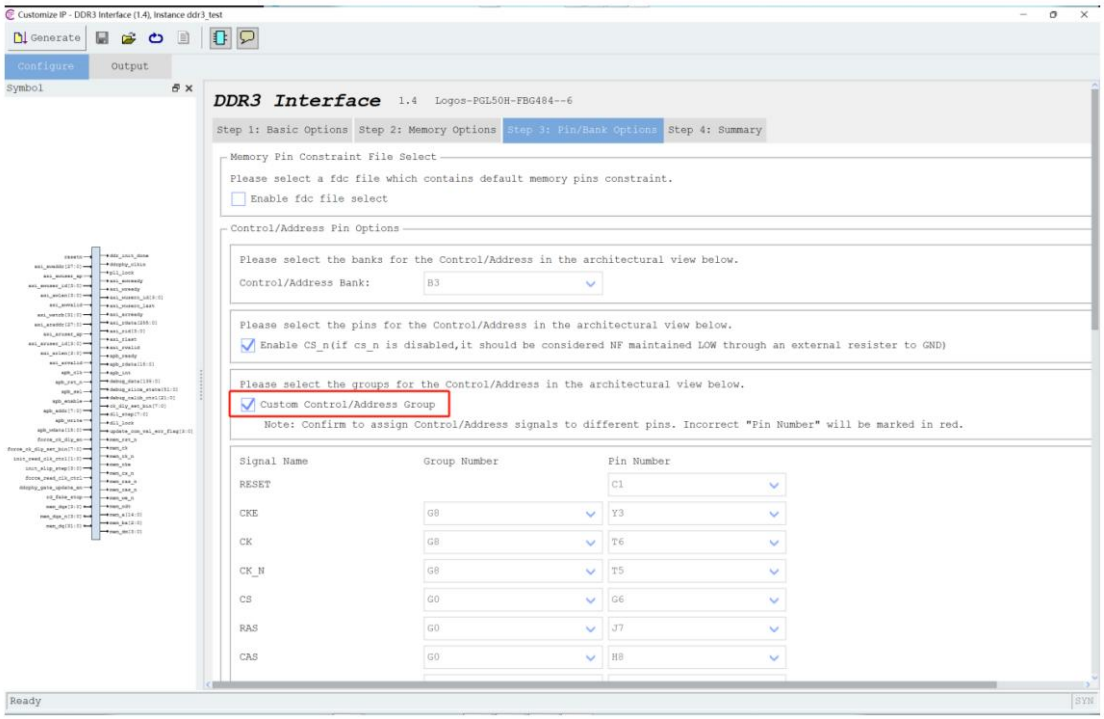
3.在 DDR3 设置界面中 Step1 按照如下设置：

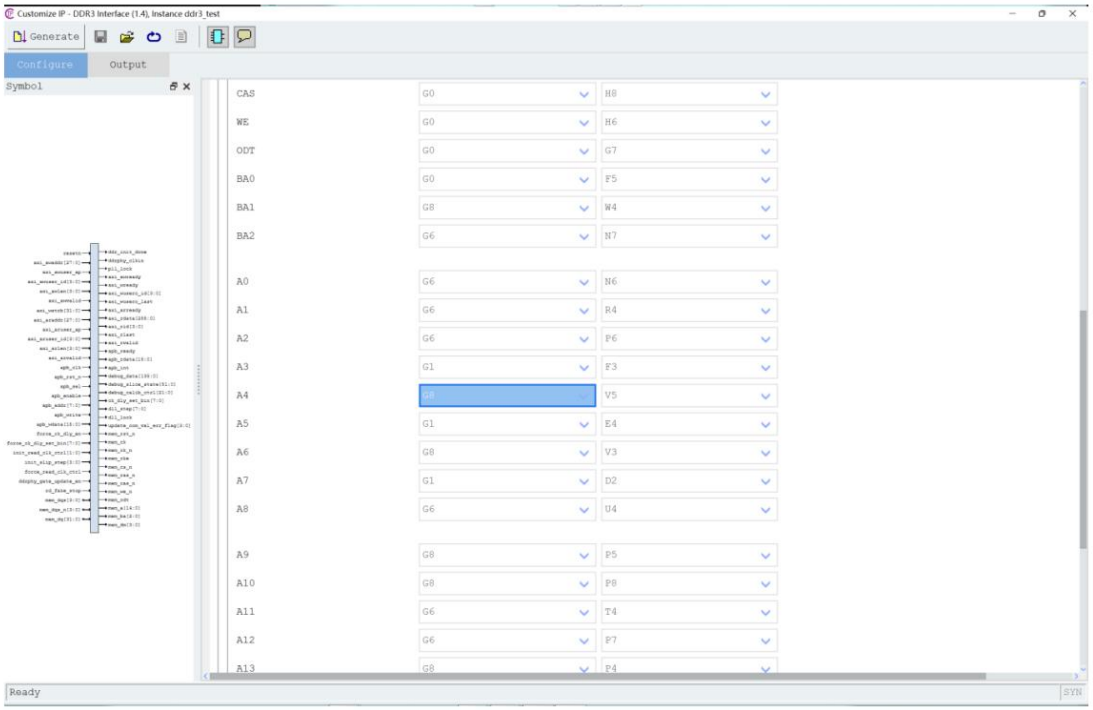


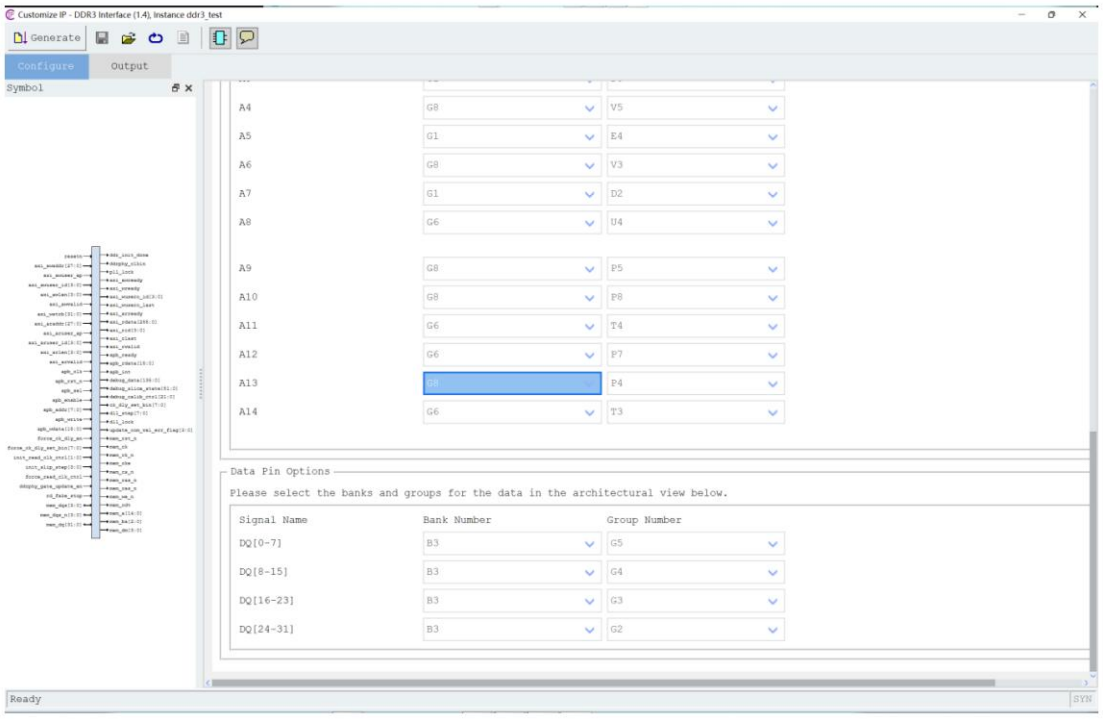
4.Step2 按照如下设置：



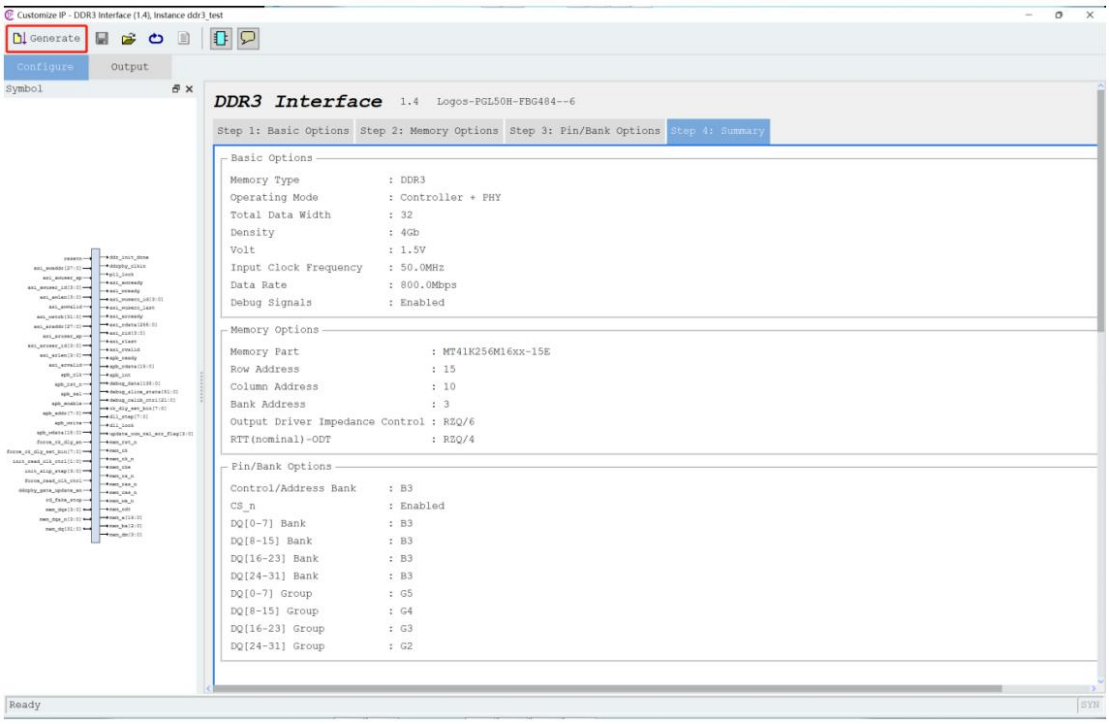
5.Step3 按照如下设置，勾选 CustomControl/AddressGroup，管脚约束参考原理图：







6.Step4 为概要，点击 Generate 可生成 DDR3IP；



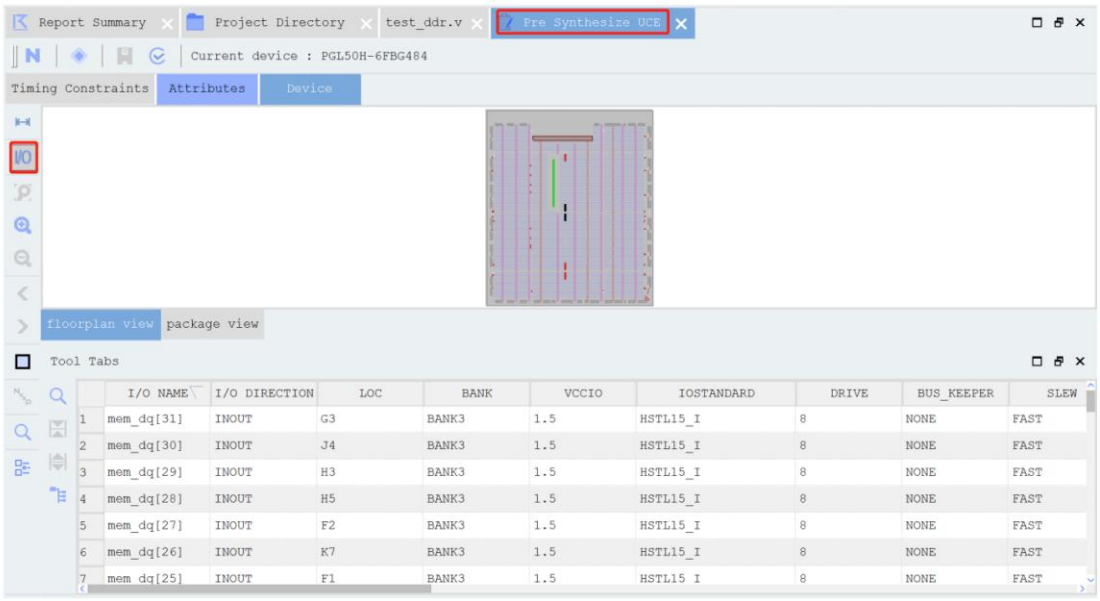
7. 关闭本工程，按此路径打开 Example 工程：2_Demo\07_ddr_test_1500\ipcore\ddr3_test\pnr

8.打开顶层文件 free_clk、ref_clk 可使用同一时钟源：

test ddr.v

```
19    parameter MEM_DM_WIDTH          = MEM_DQ_WIDTH/8,
20    parameter MEM_DQS_WIDTH         = MEM_DQ_WIDTH/8,
21    parameter CTRL_ADDR_WIDTH       = MEM_ROW_ADDR_WIDTH + MEM_BADDR_WIDTH + MEM_COL_
22  )(
23    input                                   ref_clk         ,
24    //input                                   free_clk        ,
25    input                                   rst_board       ,
26    output                                  pll_lock        ,
27    output                                  ddr_init_done   ,
28
29    //uart
30    input                                   uart_rxd        ,
31    output                                  uart_txd        ,
32
33    output                                  mem_rst_n       ,
34    output                                  mem_ck          ,
35    output                                  mem_ck_n        ,
36    output                                  mem_cke         ,
37
38    output                                  mem_cs_n        ,
39
40    output                                  mem_ras_n       ,
41    output                                  mem_cas_n       ,
42    output                                  mem_we_n        ,
43    output                                  mem_odt         ,
44    output      [MEM_ROW_ADDR_WIDTH-1:0]    mem_a           ,
45    output      [MEM_BADDR_WIDTH-1:0]       mem_ba          ,
46    inout       [MEM_DQS_WIDTH-1:0]         mem_dqs         ,
47    inout       [MEM_DQS_WIDTH-1:0]         mem_dqs_n       ,
48    inout       [MEM_DQ_WIDTH-1:0]          mem_dq          ,
49    output      [MEM_DM_WIDTH-1:0]          mem_dm          ,
50    output reg                              heart_beat_led  ,
51    output                                  err_flag_led
52
53
54  );
55  assign free_clk = ref_clk  ;
56
```

9.对“Step3 已做管脚约束”外的其他管脚，对照原理图使用 UCE 工具进行修改：

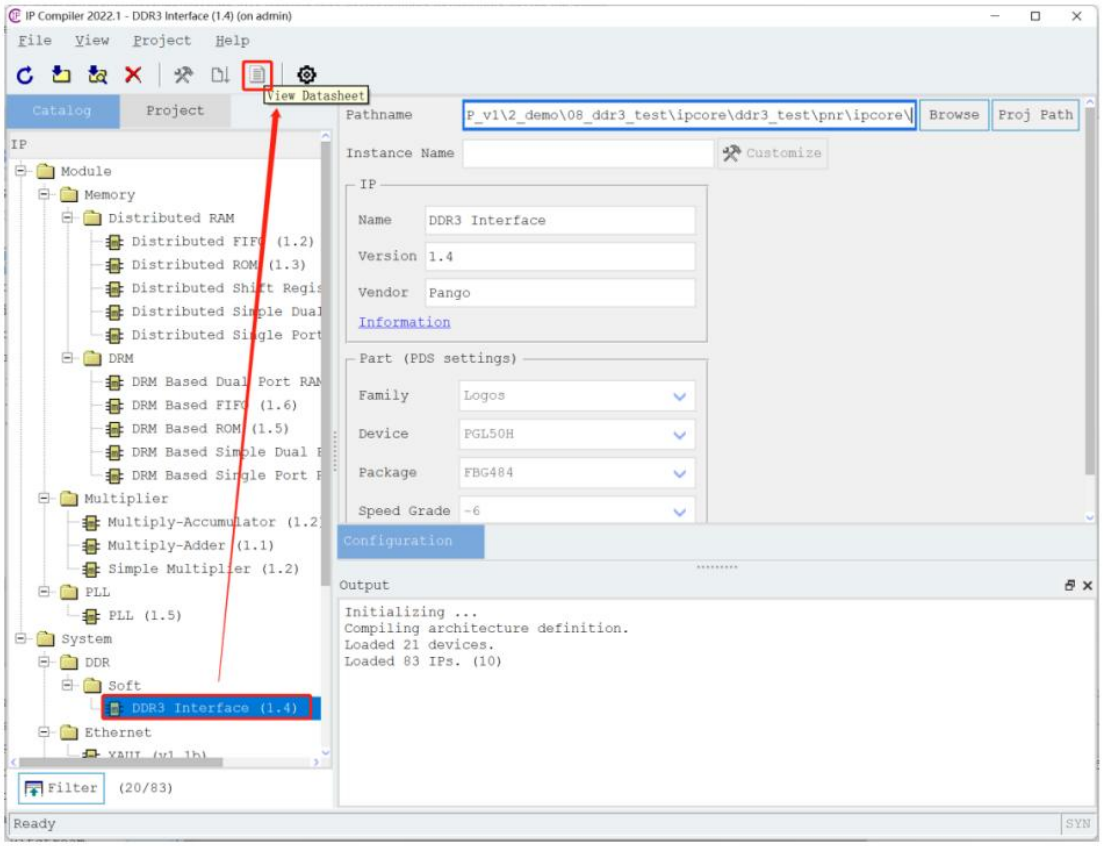


	I/O NAME	I/O DIRECTION	LOC	BANK	VCCIO	IOSTANDARD	DRIVE	BUS_KEEPER	SLEW
1	mem_dq[31]	INOUT	G3	BANK3	1.5	HSTL15_I	8	NONE	FAST
2	mem_dq[30]	INOUT	J4	BANK3	1.5	HSTL15_I	8	NONE	FAST
3	mem_dq[29]	INOUT	H3	BANK3	1.5	HSTL15_I	8	NONE	FAST
4	mem_dq[28]	INOUT	H5	BANK3	1.5	HSTL15_I	8	NONE	FAST
5	mem_dq[27]	INOUT	F2	BANK3	1.5	HSTL15_I	8	NONE	FAST
6	mem_dq[26]	INOUT	K7	BANK3	1.5	HSTL15_I	8	NONE	FAST
7	mem_dq[25]	INOUT	F1	BANK3	1.5	HSTL15_I	8	NONE	FAST

10.以下管脚可约束在 LED，方便观察实验现象；

ddr_init_done	OUTPUT	B2
err_flag_led	OUTPUT	A2
heart_beat_led	OUTPUT	B3
pll_lock	OUTPUT	A3

11.可按以下方式查看 IP 核的用户指南，了解 Example 模块组成；



6.5 实验现象

注：例程位置：2_Demo\07_ddr_test_1500\ipcore\ddr3_test\pnr

下载程序，可以看到 LED1 常亮，LED2 常灭，LED3 闪烁，LED4 常亮；

信号名称	参考说明	LED 编号
ddr_init_done	初始化标志	1
err_flag_led	数据检测错误信号	2
heart_beat_led	心跳信号	3
pll_lock	Pll 锁定指示	4

7_8.光纤通信测试实验例程

7_8.1PG2T70H 开发板简介

PG2T70H 内置了线速率高达 6.375Gbps 高速串行接口模块，即 HSST。开发板 MES50HP 有 2 路 SFP 光纤接口，用户需购买光模块(市场上 6.375G 光模块以下均可）插入到这 2 个光纤接口中进行光纤数据通信（详情请查看“PG2T70H 开发板硬件使用手册”）。

7_8.2 实验要求

通过光纤连接实现光模块之间的数据收发。

7_8.3HSST 简介

PGL50H 内置了线速率高达 6.375Gbps 高速串行接口模块，即 HSST，包含 1 个 HSST，共 4 个全双工收发 LANE，除了 PMA，HSST 还集成了丰富的 PCS 功能，可灵活应用于各种串行协议标准。在产品内部，每个 HSST 支持 1~4 个全双工收发 LANE。HSST 主要特性包括：

- 支持线速率：0.6bps-6.375Gbps
- 灵活的参考时钟选择方式
- 可编程输出摆幅和去加重
- 接收端自适应线性均衡器
- 数据通道支持 8bitonly,10bitonly,8b10b8bit,16bitonly,20bitonly,8b10b16bit,32bitonly,40bitonly,8b10b32bit,64b66b/64b67b16bit,64b66b/64b67b32bit 模式
- 可灵活配置的 PCS，可支持 PCIExpressGEN1,PCIExpressGEN2,XAUI,千兆以太网,CPRI,SRIO 等协议
- 灵活的字节对齐功能
- 支持 RxClockSlip 功能以保证固定的接收延时
- 支持协议标准 8b10b 编码解码

- 支持协议标准 64b66b/64b67b 数据适配功能
- 灵活的 CTC 方案
- 支持 x2 和 x4 的通道绑定
- HSST 的配置支持动态修改
- 近端环回和远端环回模式
- 内置 PRBS 功能

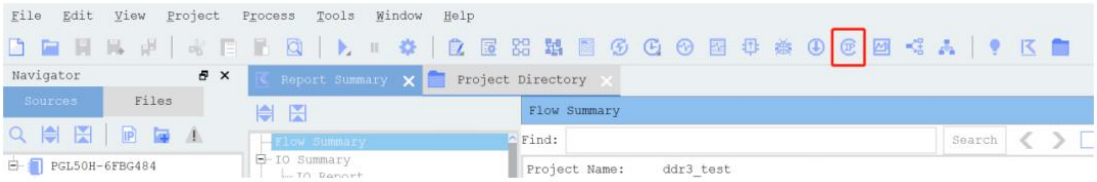
7_8.4 实验设计

7_8.4.1 安装 HSSTIP 核

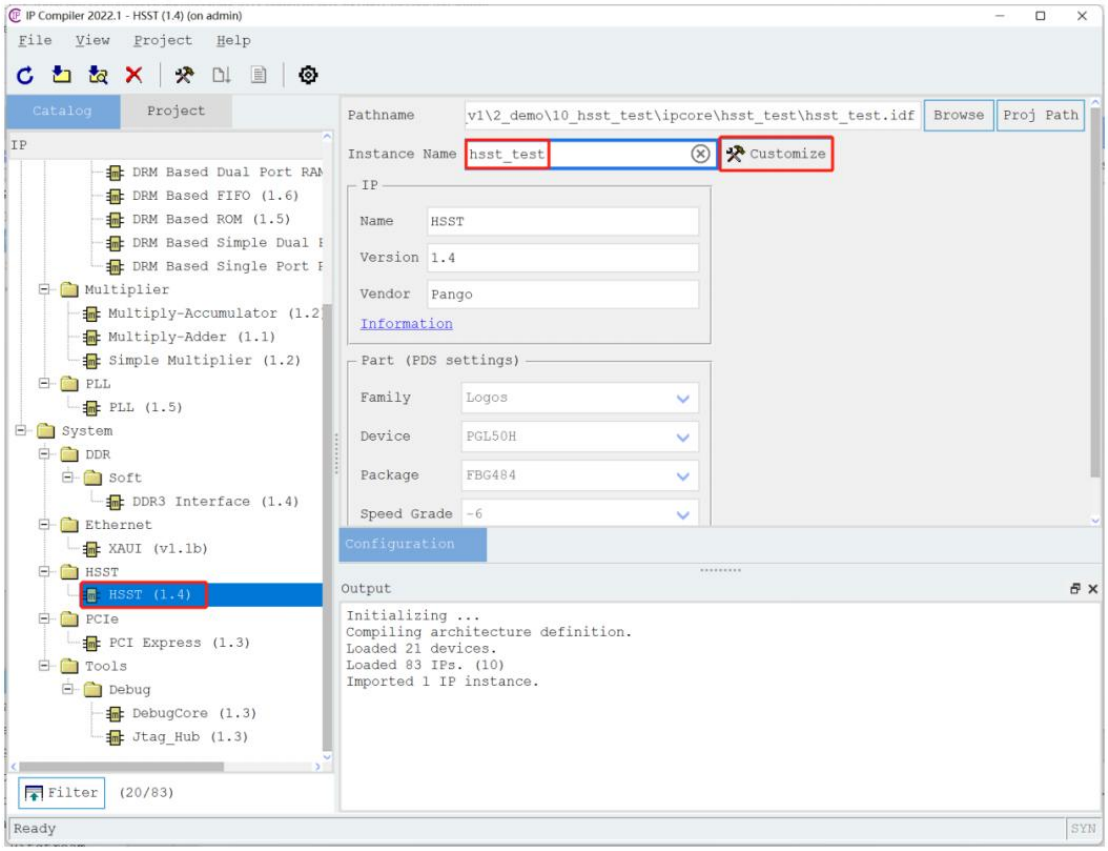
PDS 安装后，需手动添加 HSSTIP。

7_8.4.2 光纤通信测试例程

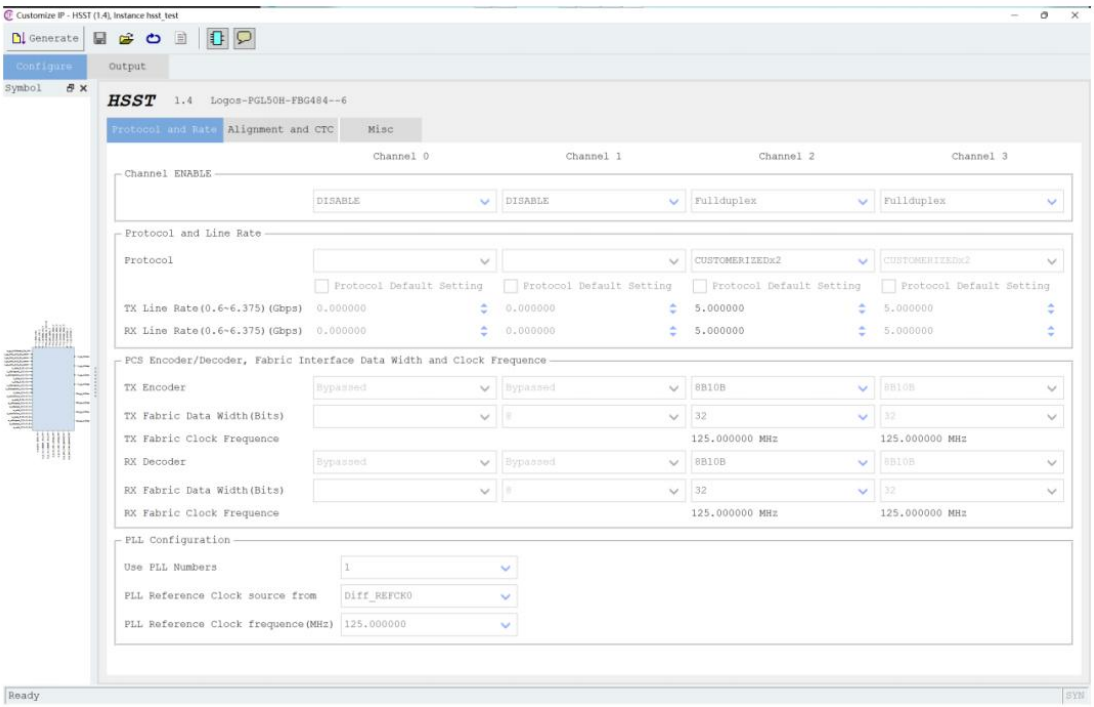
1.打开 PDS 软件，新建工程 hsst_test，点开如下图标，打开 IPCompiler；



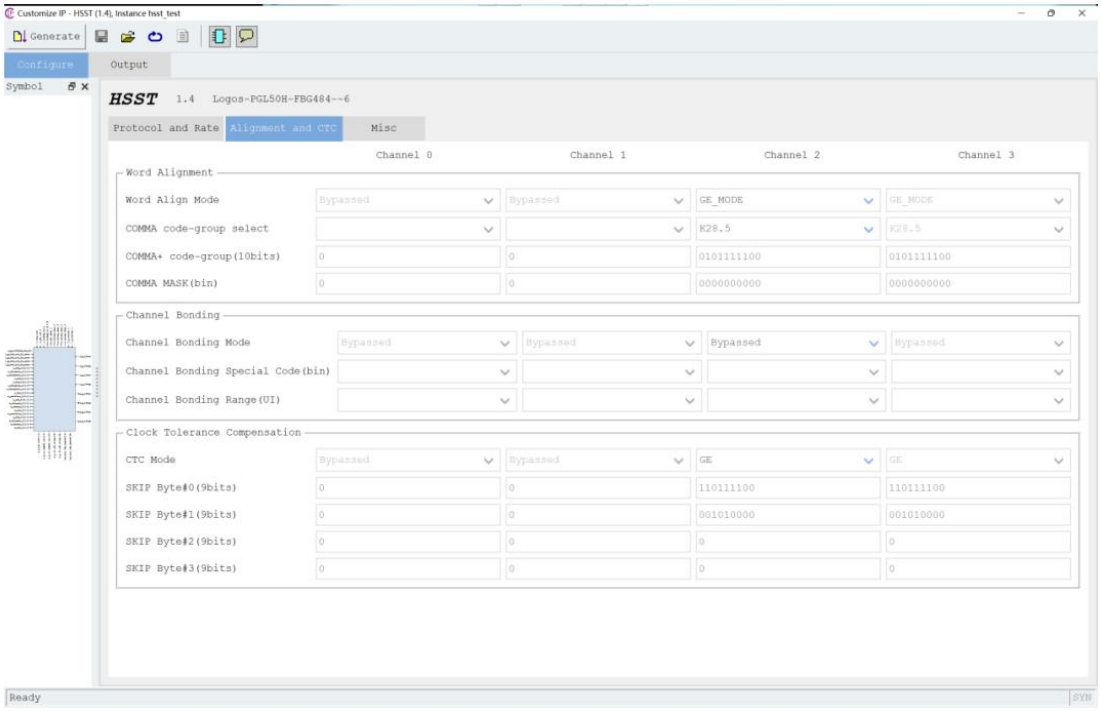
2.选择 HSSTIP，取名，然后点击 Customize；



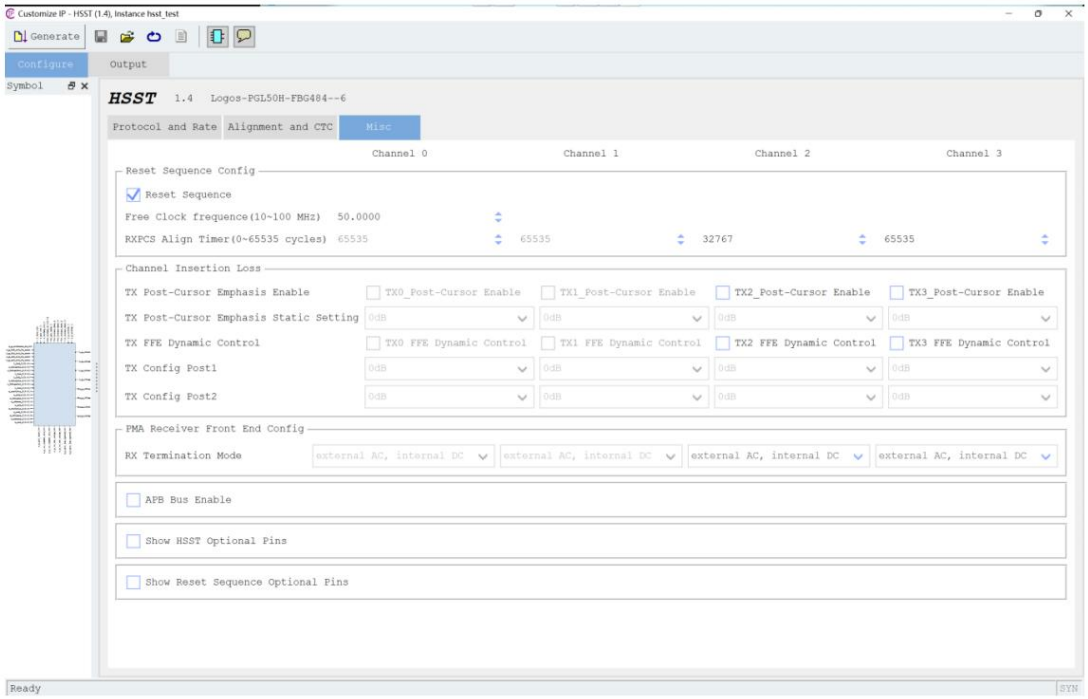
3.在 HSST 设置界面中 ProtocolandRate 按照如下设置，Channel0Channel1 为 DISABLE，Channel2Channel3 为 Fullduplex：



4.AlignmentandCTC 按照如下设置：



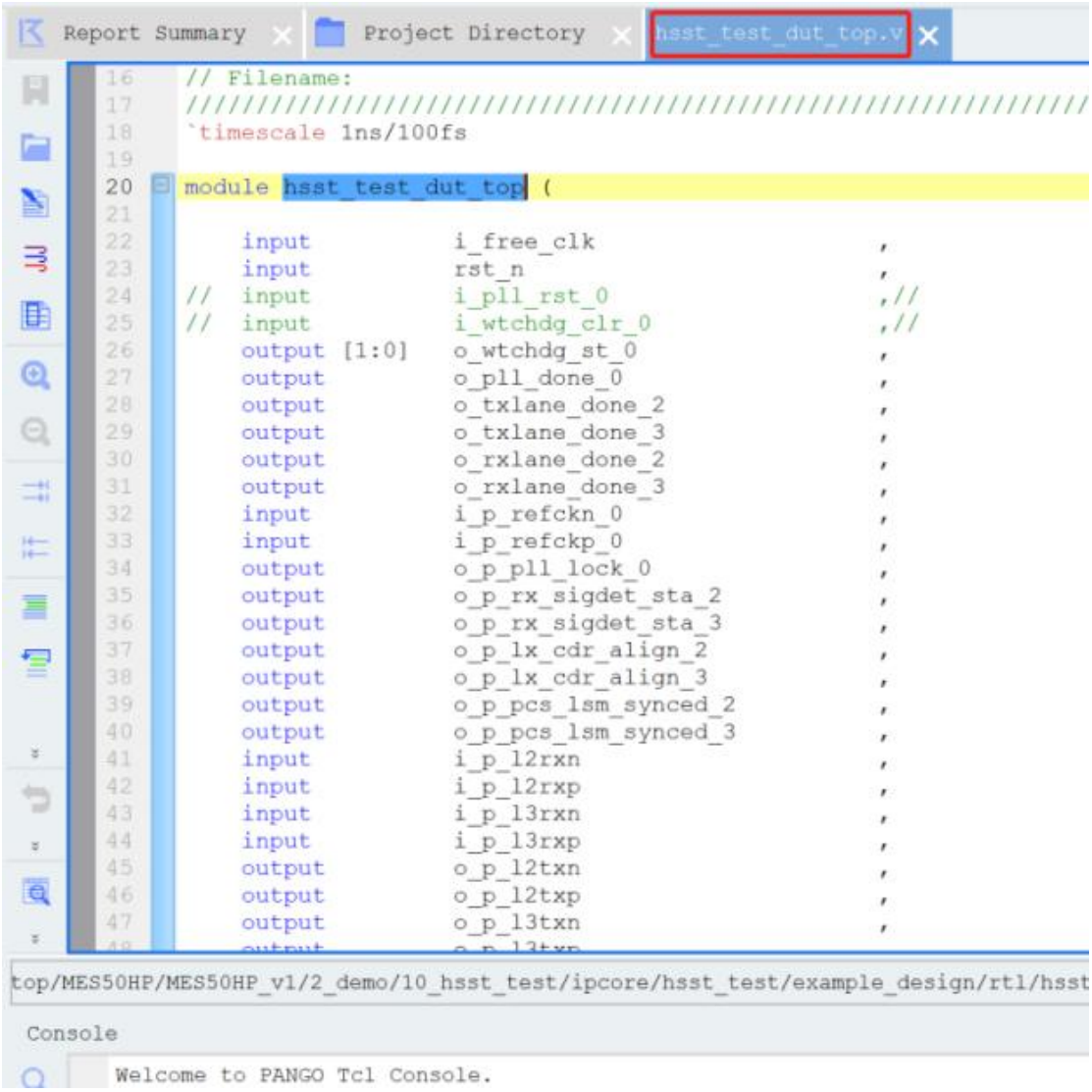
5.Misc 按照如下设置，点击 Generate 可生成 HSSTIP：



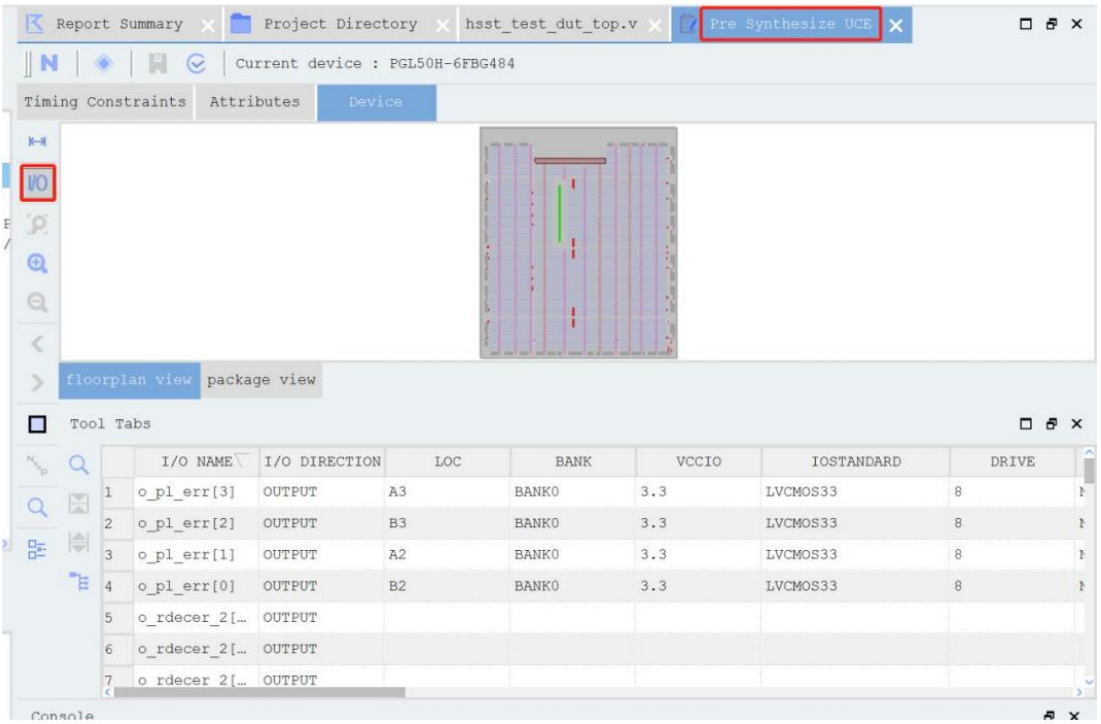
6.关闭本工程，按此路径打开 Example 工程：

2_Demo\8_hsst_test\ipcore\hsst_test\pnr\example_design

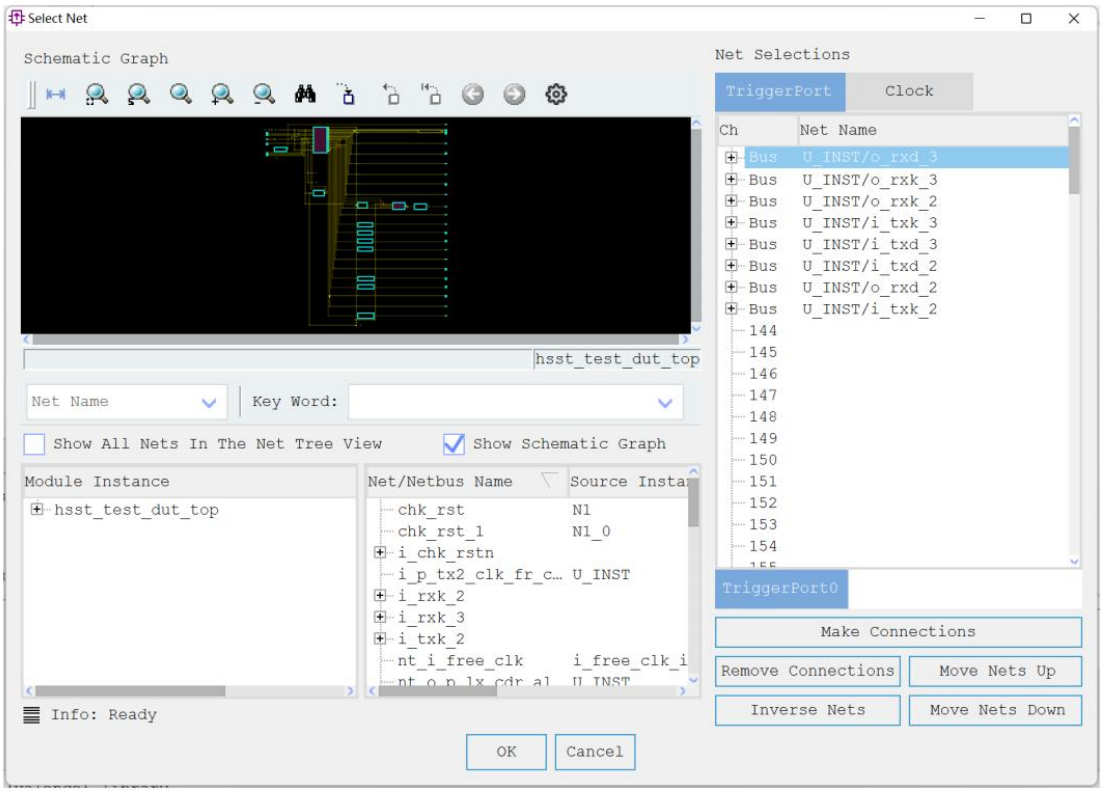
7.为了能在开发板上运行，需对顶层文件 hsst_test_dut_top 的复位进行修改，详情请查看 10_hsst_test 例程顶层文件：



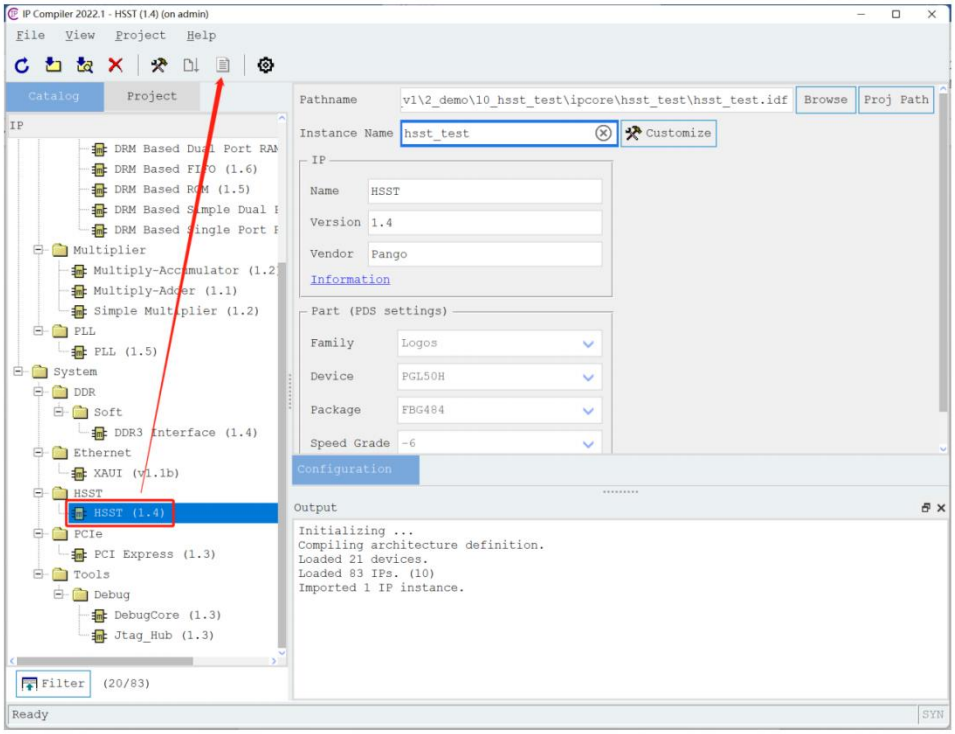
8.修改管脚分配，详情请查看原理图或 10_hsst_test 例程；



9.进行 Debugger 插核操作，操作步骤请查看“PDS 快速使用手册.pdf”；

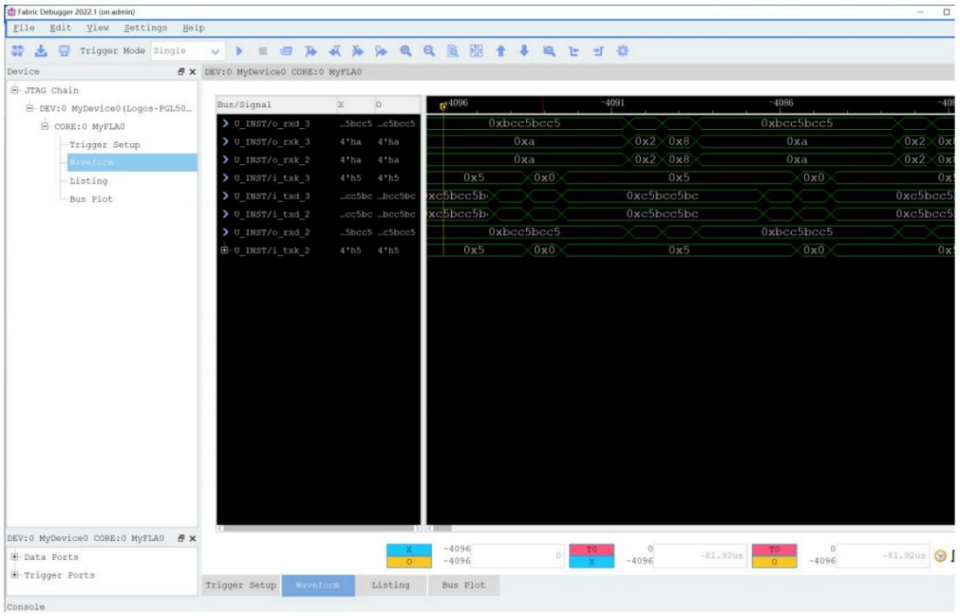


10.可按以下方式查看 IP 核的用户指南，了解 Example 模块组成；



7_8.5 实验现象

因为 PG2T70H 开发板只有一路 SFP 接口，因此用户需购买两块板卡，一块板卡发送，一块板卡接收，即可运行此例程。把光纤两端接入分别接入两块板卡的 SFP 接口（用户需购买光模块），进行 Debugger 在线调试，可以在窗口中看到发送和接收的数据一致的。



9.以太网传输实验例程

9.1 实验目的

学习如何在 FPGA 上使用 RGMII（简化千兆媒体独立接口）协议实现以太网通信接口。以及理解并实践以太网 PHY 芯片的上电复位和延时复位过程，确保 PHY 能够正确启动和工作。同时学习如何使用 util_gmii_to_rgmii 模块，实现 GMII（千兆媒体独立接口）信号与 RGMII 信号之间的转换，解决 FPGA 内部使用 GMII 而 PHY 使用 RGMII 的问题。

9.2 实验原理

GMII 是用于千兆以太网的标准接口，使用 8 位数据总线，独立的发送和接收时钟，数据在时钟的上升沿传输。RGMII 通过使用双边沿触发和 4 位数据总线，将接口引脚数量减少一半，实现更高的集成度。数据在时钟的上升沿和下降沿传输。由于 FPGA 内部通常使用 GMII 信号，而外部 PHY 芯片可能采用 RGMII 接口，因此需要通过 util_gmii_to_rgmii 模块，实现 GMII 与 RGMII 信号的双向转换，确保 FPGA 与 PHY 之间的数据通信顺畅。

PHY 芯片在上电后需要一定的时间进行内部寄存器的初始化和自检。通过 power_on_rst 模块，利用系统时钟 sys_clk，产生一个延时的复位信号 phy_rst_n。该模块在上电后计时指定的毫秒数（如 50ms），然后释放复位信号，确保 PHY 芯片有足够的时间完成初始化。

9.3 工程说明

9.3.1 顶层代码讲解

此例程顶层代码如下所示：

```
1. `timescale 1ns / 1ps
2. ////////////////////////////////////////////////////////////////////////////////
3. // Module Name:    ethernet_test
```

```
////////////////////////////////////////////////////////////////////////
module ethernet_test(
    input       sys_clk,
    input       rst_n,
    output      phy_rst_n,
  output      e_mdc,
  inout       e_mdio,
  output[3:0] rgmii_txd/*synthesis PAP_MARK_DEBUG = "ture"*/,
  output      rgmii_txctl/*synthesis PAP_MARK_DEBUG = "ture"*/,
  output      rgmii_txc/*synthesis PAP_MARK_DEBUG = "ture"*/,
  input[3:0]  rgmii_rxd/*synthesis PAP_MARK_DEBUG = "ture"*/,
  input       rgmii_rxctl/*synthesis PAP_MARK_DEBUG = "ture"*/,
  input       rgmii_rxc/*synthesis PAP_MARK_DEBUG = "ture"*/,
    output[3:0]      led
    );
wire            reset_n;
wire   [ 7:0]   gmii_txd;
wire            gmii_tx_en;
wire            gmii_tx_er;
wire            gmii_tx_clk;
wire            gmii_crs;
wire            gmii_col;
wire   [ 7:0]   gmii_rxd/*synthesis PAP_MARK_DEBUG = "ture"*/;
wire            gmii_rx_dv/*synthesis PAP_MARK_DEBUG = "ture"*/;
wire            gmii_rx_er/*synthesis PAP_MARK_DEBUG = "ture"*/;
wire            gmii_rx_clk;
wire  [ 1:0]    speed_selection; // 1x gigabit, 01 100Mbps, 00 10
mbps
wire            duplex_mode;     // 1 full, 0 half
wire            rgmii_rxcpll;
assign speed_selection = 2'b10;
assign duplex_mode = 1'b1;
wire                sys_clk;
wire                sys_clk_w;
wire                led_r;

wire            e_rx_dv    ;
wire [7:0]      e_rxd      ;
wire            e_tx_en    ;
wire [7:0]      e_txd      ;
```

```
wire            e_rst_n    ;

//assign led =~led_r;
/*************************************************************
*********
generate single end clock
*************************************************************
*********/
power_on_rst #
  (
 .CLK_FRE(50),
 .DELAY_MS(50)
  )
reset_power_on_m0
(
 .clk                       (sys_clk                  ),

 .rst_n                     (rst_n                   ),
      //user reset high active
 .power_on_rstn             (phy_rst_n                )
      //power on reset low active
);

//assign phy_rst_n = 1'b1;
util_gmii_to_rgmii util_gmii_to_rgmii_m0(
 .reset          (1'b0),

 .rgmii_td       (rgmii_txd),
 .rgmii_tx_ctl   (rgmii_txctl),
 .rgmii_txc      (rgmii_txc),
 .rgmii_rd       (rgmii_rxd),
 .rgmii_rx_ctl   (rgmii_rxctl),
 .gmii_rx_clk    (gmii_rx_clk),

// .gmii_txd       (gmii_txd),
// .gmii_tx_en     (gmii_tx_en),
 .gmii_txd       (e_txd),
 .gmii_tx_en     (e_tx_en),

 .gmii_tx_er     (1'b0),
 .gmii_tx_clk    (gmii_tx_clk),
 .gmii_crs       (gmii_crs),
 .gmii_col       (gmii_col),
 .gmii_rxd       (gmii_rxd),
```

```
    .rgmii_rxc      (rgmii_rxc),//add
 .gmii_rx_dv     (gmii_rx_dv),
 .gmii_rx_er     (gmii_rx_er),
 .speed_selection(speed_selection),
 .duplex_mode    (duplex_mode),
    .led            (led[0]),
    .sys_clk        (sys_clk)
 );


wire [31:0] pack_total_len;

gmii_arbi arbi_inst
(
.clk                (gmii_tx_clk      ),
.rst_n              (rst_n            ),
.speed              (speed_selection  ),
.link               (1'b1             ),
.pack_total_len     (pack_total_len   ),

.e_rst_n            (e_rst_n          ),
.gmii_rx_dv         (gmii_rx_dv       ),
.gmii_rxd           (gmii_rxd         ),
.gmii_tx_en         (gmii_tx_en       ),
.gmii_txd           (gmii_txd         ),
.e_rx_dv            (e_rx_dv          ),
.e_rxd              (e_rxd            ),
.e_tx_en            (e_tx_en          ),
.e_txd              (e_txd            )
);


mac_test mac_test0
(
    .gmii_tx_clk            (gmii_tx_clk),
    .gmii_rx_clk            (gmii_rx_clk) ,
    .rst_n                  (rst_n),

//    .pack_total_len         (32'd125000000  ),//d25000000
    d125000000 pack_total_len
//    .gmii_rx_dv             (gmii_rx_dv),
//    .gmii_rxd               (gmii_rxd ),
```

```
125.
126.     .pack_total_len         (pack_total_len  ),//d25000000
   d125000000 pack_total_len
127.     .gmii_rx_dv             (e_rx_dv),
128.     .gmii_rxd               (e_rxd ),
129.
130.     .gmii_tx_en             (gmii_tx_en),
131.     .gmii_txd               (gmii_txd )
132.
133. );
134. endmodule
```

首先，模块的端口定义包括了系统时钟 sys_clk、复位信号 rst_n 以及与 PHY 芯片交互的信号。其中，phy_rst_n 是 PHY 芯片的复位信号，e_mdc 和 e_mdio 是用于 MDIO 接口的管理数据时钟和数据线，尽管在代码中并未进一步使用。rgmii_txd、rgmii_txctl、rgmii_txc 是 RGMII 接口的发送端信号，用于发送数据、发送控制和发送时钟。rgmii_rxd、rgmii_rxctl、rgmii_rxc 是 RGMII 接口的接收端信号，用于接收数据、接收控制和接收时钟。led 用于连接 LED 指示灯，显示模块的运行状态。

在模块内部，定义了一系列的信号来处理 GMII 接口的数据传输。gmii_txd、gmii_tx_en、gmii_rxd、gmii_rx_dv 等信号用于 GMII 接口的数据发送和接收。speed_selection 和 duplex_mode 被设定为常数，分别为 2'b10 和 1'b1，表示将 PHY 芯片配置为千兆位速率和全双工模式。

接下来，代码实例化了一个 power_on_rst 模块，用于在上电时对 PHY 芯片进行延时复位。这个模块通过系统时钟 sys_clk 和用户复位信号 rst_n，在上电后延时 50 毫秒（由参数 DELAY_MS(50)指定），然后释放对 PHY 的复位信号 phy_rst_n。这确保了 PHY 芯片有足够的时间进行内部初始化。

然后，代码实例化了 util_gmii_to_rgmii 模块，这个模块的作用是实现 GMII 和 RGMII 接口之间的信号转换。由于 FPGA 内部通常使用 GMII 接口，而 PHY 芯片可能使用 RGMII 接口，因此需要进行信号的转换。该模块连接了 GMII 的发送和接收信号以及对应的 RGMII 信号，同时配置了速率和双工模式。值得注意的是，在连接发送数据时，gmii_txd 和 gmii_tx_en 连接的是 e_txd 和 e_tx_en，这意味着发送的数据来自于其他模块的处理，而不是直接从 GMII 接口发送。

接下来，代码实例化了一个 gmii_arbi 模块，起到仲裁器的作用。这个模块

负责协调 GMII 接口与 MAC 测试模块之间的数据传输。它接收来自 PHY 的 GMII 接收信号 gmii_rx_dv 和 gmii_rxd，并将其传递给 MAC 测试模块（通过 e_rx_dv 和 e_rxd）。同时，它接收来自 MAC 测试模块的发送信号 e_tx_en 和 e_txd，并通过 GMII 发送信号 gmii_tx_en 和 gmii_txd 将数据发送到 PHY。这种设计使得 MAC 测试模块可以专注于数据的生成和处理，而不必处理底层的 GMII 接口细节。

最后，代码实例化了一个 mac_test 模块，用于生成测试数据包并验证接收到的数据。该模块使用发送和接收时钟 gmii_tx_clk 和 gmii_rx_clk，以及复位信号 rst_n。pack_total_len 是一个 32 位的信号，用于指定要发送的数据包总长度。mac_test 模块接收来自 gmii_arbi 模块的接收数据 e_rx_dv 和 e_rxd，并通过 gmii_tx_en 和 gmii_txd 发送数据。

总体而言，这个模块实现了一个完整的以太网通信测试平台。通过 power_on_rst 模块，确保 PHY 芯片在正确的时间复位。util_gmii_to_rgmii 模块解决了 GMII 与 RGMII 接口的兼容性问题。gmii_arbi 模块负责数据的仲裁和传递，使得 MAC 层的测试模块可以顺利地进行数据的发送和接收。mac_test 模块则专注于生成和验证数据包，实现对以太网通信的功能性测试。

9.3.2util_gmii_to_rgmii 模块讲解

util_gmii_to_rgmii 模块的源码如下所示：

```
1.
2.  module util_gmii_to_rgmii (
3.    reset,
4.    rgmii_td,
5.    rgmii_tx_ctl,
6.    rgmii_txc,
7.    rgmii_rd,
8.    rgmii_rx_ctl,
9.    gmii_rx_clk,
10.   rgmii_rxc,
11.   gmii_txd,
12.   gmii_tx_en,
13.   gmii_tx_er,
14.   gmii_tx_clk,
```

```
    gmii_crs,
    gmii_col,
    gmii_rxd,
    gmii_rx_dv,
    gmii_rx_er,
    speed_selection,
    duplex_mode,
    led,
    pll_phase_shft_lock,
    clk,
    sys_clk
    );
input sys_clk;
output pll_phase_shft_lock;
output clk;
 output reg            led;
    input           rgmii_rxc;//add
    input           reset;
    output  [ 3:0]  rgmii_td;
    output          rgmii_tx_ctl;
    output          rgmii_txc;
    input   [ 3:0]  rgmii_rd;
    input           rgmii_rx_ctl;
    output           gmii_rx_clk;
    input   [ 7:0]  gmii_txd;
    input           gmii_tx_en;
    input           gmii_tx_er;
    output          gmii_tx_clk;
    output          gmii_crs;
    output          gmii_col;
    output  [ 7:0]  gmii_rxd;
    output          gmii_rx_dv;
    output          gmii_rx_er;
    input  [ 1:0]   speed_selection; // 1x gigabit, 01 100Mbps, 00
  10mbps
    input           duplex_mode;     // 1 full, 0 half

    wire gigabit;
    wire gmii_tx_clk_s;
    wire gmii_rx_dv_s;

    wire  [ 7:0]    gmii_rxd_s;
    wire            rgmii_rx_ctl_delay;
    wire            rgmii_rx_ctl_s;
```

```
  // registers
  reg             tx_reset_d1;
  reg             tx_reset_sync;
  reg             rx_reset_d1;
  reg   [ 7:0]    gmii_txd_r;
  reg             gmii_tx_en_r;
  reg             gmii_tx_er_r;
  reg   [ 7:0]    gmii_txd_r_d1;
  reg             gmii_tx_en_r_d1;
  reg             gmii_tx_er_r_d1;

  reg             rgmii_tx_ctl_r;
  reg   [ 3:0]    gmii_txd_low;
  reg             gmii_col;
  reg             gmii_crs;

  reg  [ 7:0]     gmii_rxd;
  reg             gmii_rx_dv;
  reg             gmii_rx_er;
  wire         padt1     ;
  wire         padt2     ;
  wire         padt3     ;
  wire         padt4     ;
  wire         padt5     ;
  wire         padt6    ;
  wire         stx_txc   ;
  wire         stx_ctr   ;
  wire  [3:0]  stxd_rgm  ;
  assign gigabit        = speed_selection [1];
 assign gmii_tx_clk    = gmii_tx_clk_s;
  assign gmii_tx_clk_s  = gmii_rx_clk;

//test led
reg[28:0] cnt_timer;
  always @(posedge gmii_tx_clk_s)
  begin
  cnt_timer<=cnt_timer+1'b1;
if( cnt_timer==29'h3ffffff)
begin
   led=~led;
    cnt_timer<=29'h0;
end
  end

```

```
102. wire gmii_rx_clk;
103. 
104. 
105. //GTP_CLKBUFG GTP_CLKBUFG_RXSHFT(
106. //    .CLKIN     (rgmii_rxc),
107. //    .CLKOUT    (gmii_rx_clk)
108. //);
109. 
110. wire rx_clki_shft;
111. pll_sft U_pll_phase_shift(
112.     .clkout0   (rx_clki_shft       ),    //125MHz
113.     .clkin1    (rgmii_rxc          ),
114.     .clkfb     (gmii_rx_clk        ),
115.     .rst   (1'b0               ),
116.     .lock  ()
117.     );
118. 
119. GTP_CLKBUFG GTP_CLKBUFG_RXSHFT(
120.     .CLKIN     (~rx_clki_shft),
121.     .CLKOUT    (gmii_rx_clk )
122. );
123. //assign gmii_rx_clk=rgmii_rxc;
124.   always @(posedge gmii_rx_clk)
125.   begin
126.     gmii_rxd       = gmii_rxd_s;
127.     gmii_rx_dv     = gmii_rx_dv_s;
128.     gmii_rx_er     = gmii_rx_dv_s ^ rgmii_rx_ctl_s;
129.   end
130. 
131.   always @(posedge gmii_tx_clk_s) begin
132.     tx_reset_d1    <= reset;
133.     tx_reset_sync  <= tx_reset_d1;
134.   end
135. 
136.   always @(posedge gmii_tx_clk_s)
137.   begin
138.     rgmii_tx_ctl_r = gmii_tx_en_r ^ gmii_tx_er_r;
139.     gmii_txd_low   = gigabit ? gmii_txd_r[7:4] :  gmii_txd_r[3:0]
    ;
140.     gmii_col       = duplex_mode ? 1'b0 : (gmii_tx_en_r| gmii_tx
    _er_r) & ( gmii_rx_dv | gmii_rx_er) ;
141.     gmii_crs       = duplex_mode ? 1'b0 : (gmii_tx_en_r| gmii_tx
    _er_r| gmii_rx_dv | gmii_rx_er);
142.   end
```

```
143.
144.   always @(posedge gmii_tx_clk_s) begin
145.     if (tx_reset_sync == 1'b1) begin
146.       gmii_txd_r   <= 8'h0;
147.       gmii_tx_en_r <= 1'b0;
148.       gmii_tx_er_r <= 1'b0;
149.     end
150.     else
151.     begin
152.       gmii_txd_r   <= gmii_txd;
153.       gmii_tx_en_r <= gmii_tx_en;
154.       gmii_tx_er_r <= gmii_tx_er;
155.       gmii_txd_r_d1   <= gmii_txd_r;
156.       gmii_tx_en_r_d1 <= gmii_tx_en_r;
157.       gmii_tx_er_r_d1 <= gmii_tx_er_r;
158.     end
159.   end
160.
161.
162.
163.
164. //------------------------------------
165. GTP_OSERDES_E2 #
166. (
167. . GRS_EN ("TRUE"),
168. . OSERDES_MODE ("DDR2TO1_SAME_EDGE"),
169. . TSERDES_EN ("FALSE"),
170. . UPD0_SHIFT_EN ("FALSE"),
171. . UPD1_SHIFT_EN ("FALSE"),
172. . INIT_SET (2'b00),
173. . GRS_TYPE_DQ ("RESET"),
174. . LRS_TYPE_DQ0 ("ASYNC_RESET"),
175. . LRS_TYPE_DQ1 ("ASYNC_RESET"),
176. . LRS_TYPE_DQ2 ("ASYNC_RESET"),
177. . LRS_TYPE_DQ3 ("ASYNC_RESET"),
178. . GRS_TYPE_TQ ("RESET"),
179. . LRS_TYPE_TQ0 ("ASYNC_RESET"),
180. . LRS_TYPE_TQ1 ("ASYNC_RESET"),
181. . LRS_TYPE_TQ2 ("ASYNC_RESET"),
182. . LRS_TYPE_TQ3 ("ASYNC_RESET"),
183. . TRI_EN  ("FALSE"),
184. . TBYTE_EN ("FALSE"),
185. . MIPI_EN ("FALSE"),
186. . OCASCADE_EN ("FALSE")
```

```
) GTP_OSERDES_E2_INST1 (
. RST (tx_reset_sync),
. OCE (1'b1),
. TCE (1'b0),
. OCLKDIV (gmii_tx_clk_s),
. SERCLK (gmii_tx_clk_s),
. OCLK (gmii_tx_clk_s),
. MIPI_CTRL (),
. UPD0_SHIFT (1'b0),
. UPD1_SHIFT (1'b0),
. OSHIFTIN0 (),
. OSHIFTIN1 (),
. DI ({6'd0,gmii_txd_low[3],gmii_txd_r_d1[3]}),    // DDR captur
  e data
. TI (),
. TBYTE_IN (),
. OSHIFTOUT0 (),
. OSHIFTOUT1 (),
. DO (stxd_rgm[3]),
. TQ (padt2)
);

GTP_OUTBUF  gtp_outbuf2
(

    .I(stxd_rgm[3]),
    .O(rgmii_td[3])
);
//---------------------------------------
GTP_OSERDES_E2 #
(
. GRS_EN ("TRUE"),
. OSERDES_MODE ("DDR2TO1_SAME_EDGE"),
. TSERDES_EN ("FALSE"),
. UPD0_SHIFT_EN ("FALSE"),
. UPD1_SHIFT_EN ("FALSE"),
. INIT_SET (2'b00),
. GRS_TYPE_DQ ("RESET"),
. LRS_TYPE_DQ0 ("ASYNC_RESET"),
. LRS_TYPE_DQ1 ("ASYNC_RESET"),
. LRS_TYPE_DQ2 ("ASYNC_RESET"),
. LRS_TYPE_DQ3 ("ASYNC_RESET"),
. GRS_TYPE_TQ ("RESET"),
. LRS_TYPE_TQ0 ("ASYNC_RESET"),
```

```
. LRS_TYPE_TQ1 ("ASYNC_RESET"),
. LRS_TYPE_TQ2 ("ASYNC_RESET"),
. LRS_TYPE_TQ3 ("ASYNC_RESET"),
. TRI_EN  ("FALSE"),
. TBYTE_EN ("FALSE"),
. MIPI_EN ("FALSE"),
. OCASCADE_EN ("FALSE")
) GTP_OSERDES_E2_INST2 (
. RST (tx_reset_sync),
. OCE (1'b1),
. TCE (1'b0),
. OCLKDIV (gmii_tx_clk_s),
. SERCLK (gmii_tx_clk_s),
. OCLK (gmii_tx_clk_s),
. MIPI_CTRL (),
. UPD0_SHIFT (1'b0),
. UPD1_SHIFT (1'b0),
. OSHIFTIN0 (),
. OSHIFTIN1 (),
. DI ({6'd0,gmii_txd_low[2],gmii_txd_r_d1[2]}),
. TI (),
. TBYTE_IN (),
. OSHIFTOUT0 (),
. OSHIFTOUT1 (),
. DO (stxd_rgm[2]),
. TQ (padt3)
);


GTP_OUTBUF  gtp_outbuf3
(
    .I(stxd_rgm[2]),
    .O(rgmii_td[2])
);
//---------------------------------
GTP_OSERDES_E2 #
(
. GRS_EN ("TRUE"),
. OSERDES_MODE ("DDR2TO1_SAME_EDGE"),
. TSERDES_EN ("FALSE"),
. UPD0_SHIFT_EN ("FALSE"),
. UPD1_SHIFT_EN ("FALSE"),
. INIT_SET (2'b00),
. GRS_TYPE_DQ ("RESET"),
```

```
. LRS_TYPE_DQ0 ("ASYNC_RESET"),
. LRS_TYPE_DQ1 ("ASYNC_RESET"),
. LRS_TYPE_DQ2 ("ASYNC_RESET"),
. LRS_TYPE_DQ3 ("ASYNC_RESET"),
. GRS_TYPE_TQ ("RESET"),
. LRS_TYPE_TQ0 ("ASYNC_RESET"),
. LRS_TYPE_TQ1 ("ASYNC_RESET"),
. LRS_TYPE_TQ2 ("ASYNC_RESET"),
. LRS_TYPE_TQ3 ("ASYNC_RESET"),
. TRI_EN  ("FALSE"),
. TBYTE_EN ("FALSE"),
. MIPI_EN ("FALSE"),
. OCASCADE_EN ("FALSE")
) GTP_OSERDES_E2_INST3 (
. RST (tx_reset_sync),
. OCE (1'b1),
. TCE (1'b0),
. OCLKDIV (gmii_tx_clk_s),
. SERCLK (gmii_tx_clk_s),
. OCLK (gmii_tx_clk_s),
. MIPI_CTRL (),
. UPD0_SHIFT (1'b0),
. UPD1_SHIFT (1'b0),
. OSHIFTIN0 (),
. OSHIFTIN1 (),
. DI ({6'd0,gmii_txd_low[1],gmii_txd_r_d1[1]}),
. TI (),
. TBYTE_IN (),
. OSHIFTOUT0 (),
. OSHIFTOUT1 (),
. DO (stxd_rgm[1]),
. TQ (padt4)
);


GTP_OUTBUF  gtp_outbuf4
(
    .I(stxd_rgm[1]),
    .O(rgmii_td[1])
);
//--------------------------------------
GTP_OSERDES_E2 #
(
```

```
. GRS_EN ("TRUE"),
. OSERDES_MODE ("DDR2TO1_SAME_EDGE"),
. TSERDES_EN ("FALSE"),
. UPD0_SHIFT_EN ("FALSE"),
. UPD1_SHIFT_EN ("FALSE"),
. INIT_SET (2'b00),
. GRS_TYPE_DQ ("RESET"),
. LRS_TYPE_DQ0 ("ASYNC_RESET"),
. LRS_TYPE_DQ1 ("ASYNC_RESET"),
. LRS_TYPE_DQ2 ("ASYNC_RESET"),
. LRS_TYPE_DQ3 ("ASYNC_RESET"),
. GRS_TYPE_TQ ("RESET"),
. LRS_TYPE_TQ0 ("ASYNC_RESET"),
. LRS_TYPE_TQ1 ("ASYNC_RESET"),
. LRS_TYPE_TQ2 ("ASYNC_RESET"),
. LRS_TYPE_TQ3 ("ASYNC_RESET"),
. TRI_EN  ("FALSE"),
. TBYTE_EN ("FALSE"),
. MIPI_EN ("FALSE"),
. OCASCADE_EN ("FALSE")
) GTP_OSERDES_E2_INST4 (
. RST (tx_reset_sync),
. OCE (1'b1),
. TCE (1'b0),
. OCLKDIV (gmii_tx_clk_s),
. SERCLK (gmii_tx_clk_s),
. OCLK (gmii_tx_clk_s),
. MIPI_CTRL (),
. UPD0_SHIFT (1'b0),
. UPD1_SHIFT (1'b0),
. OSHIFTIN0 (),
. OSHIFTIN1 (),
. DI ({6'd0,gmii_txd_low[0],gmii_txd_r_d1[0]}),
. TI (),
. TBYTE_IN (),
. OSHIFTOUT0 (),
. OSHIFTOUT1 (),
. DO (stxd_rgm[0]),
. TQ (padt5)
);


GTP_OUTBUF  gtp_outbuf5
(
```

```
362.
363.     .I(stxd_rgm[0]),
364.
365.     .O(rgmii_td[0])
366. );
367. //-------------------------------------
368. GTP_OSERDES_E2 #
369. (
370. . GRS_EN ("TRUE"),
371. . OSERDES_MODE ("DDR2TO1_SAME_EDGE"),
372. . TSERDES_EN ("FALSE"),
373. . UPD0_SHIFT_EN ("FALSE"),
374. . UPD1_SHIFT_EN ("FALSE"),
375. . INIT_SET (2'b00),
376. . GRS_TYPE_DQ ("RESET"),
377. . LRS_TYPE_DQ0 ("ASYNC_RESET"),
378. . LRS_TYPE_DQ1 ("ASYNC_RESET"),
379. . LRS_TYPE_DQ2 ("ASYNC_RESET"),
380. . LRS_TYPE_DQ3 ("ASYNC_RESET"),
381. . GRS_TYPE_TQ ("RESET"),
382. . LRS_TYPE_TQ0 ("ASYNC_RESET"),
383. . LRS_TYPE_TQ1 ("ASYNC_RESET"),
384. . LRS_TYPE_TQ2 ("ASYNC_RESET"),
385. . LRS_TYPE_TQ3 ("ASYNC_RESET"),
386. . TRI_EN  ("FALSE"),
387. . TBYTE_EN ("FALSE"),
388. . MIPI_EN ("FALSE"),
389. . OCASCADE_EN ("FALSE")
390. ) GTP_OSERDES_E2_INST0 (
391. . RST (tx_reset_sync),
392. . OCE (1'b1),
393. . TCE (1'b0),
394. . OCLKDIV (gmii_tx_clk_s),
395. . SERCLK (gmii_tx_clk_s),
396. . OCLK (gmii_tx_clk_s),
397. . MIPI_CTRL (),
398. . UPD0_SHIFT (1'b0),
399. . UPD1_SHIFT (1'b0),
400. . OSHIFTIN0 (),
401. . OSHIFTIN1 (),
402. . DI ({6'd0,rgmii_tx_ctl_r,gmii_tx_en_r_d1}),
403. . TI (),
404. . TBYTE_IN (),
405. . OSHIFTOUT0 (),
```

```
. OSHIFTOUT1 (),
. DO (stx_ctr),
. TQ (padt1)
);


GTP_OUTBUF  gtp_outbuf1
(

    .I(stx_ctr),
    .O(rgmii_tx_ctl)
);
//-----------------------------
GTP_OSERDES_E2 #
(
. GRS_EN ("TRUE"),
. OSERDES_MODE ("DDR2TO1_SAME_EDGE"),
. TSERDES_EN ("FALSE"),
. UPD0_SHIFT_EN ("FALSE"),
. UPD1_SHIFT_EN ("FALSE"),
. INIT_SET (2'b00),
. GRS_TYPE_DQ ("RESET"),
. LRS_TYPE_DQ0 ("ASYNC_RESET"),
. LRS_TYPE_DQ1 ("ASYNC_RESET"),
. LRS_TYPE_DQ2 ("ASYNC_RESET"),
. LRS_TYPE_DQ3 ("ASYNC_RESET"),
. GRS_TYPE_TQ ("RESET"),
. LRS_TYPE_TQ0 ("ASYNC_RESET"),
. LRS_TYPE_TQ1 ("ASYNC_RESET"),
. LRS_TYPE_TQ2 ("ASYNC_RESET"),
. LRS_TYPE_TQ3 ("ASYNC_RESET"),
. TRI_EN  ("FALSE"),
. TBYTE_EN ("FALSE"),
. MIPI_EN ("FALSE"),
. OCASCADE_EN ("FALSE")
) GTP_OSERDES_E2_INST5 (
. RST (tx_reset_sync),
. OCE (1'b1),
. TCE (1'b0),
. OCLKDIV (gmii_tx_clk_s),
. SERCLK (gmii_tx_clk_s),
. OCLK (gmii_tx_clk_s),
. MIPI_CTRL (),
. UPD0_SHIFT (1'b0),
```

```
. UPD1_SHIFT (1'b0),
. OSHIFTIN0 (),
. OSHIFTIN1 (),
. DI (8'b00000001),
. TI (),
. TBYTE_IN (),
. OSHIFTOUT0 (),
. OSHIFTOUT1 (),
. DO (rgmii_txc),
. TQ (padt6)
);


//wire [7:0] delay_step_b ;
//wire [7:0] delay_step_gray ;
//
//assign delay_step_b = 8'd128;   // 0~247 , 10ps/step
//
//assign delay_step_gray=((delay_step_b>>1)^delay_step_b);  // o
   nly support gray code
//
//GTP_IODELAY_E2 #
//(
//.DELAY_STEP_SEL ("PORT"),//PORT PARAMETER
//.DELAY_STEP_VALUE( )
//)
// GTP_IODELAY_E2_inst0 (
//.DI (stx_txc),                      // rx clk input
//.DELAY_SEL (1'b1),
//.DELAY_STEP (delay_step_gray),
//.DO (rgmii_txc)             // rx clk output
//);

//---------------------------------------
wire [5:0] nc1;
GTP_ISERDES_E2 #
(
.ISERDES_MODE ("DDR1TO2_SAME_PIPELINED"),
.CASCADE_MODE("MASTER"),
.BITSLIP_EN("FALSE"),
.GRS_EN ("TRUE"),
.NUM_ICE(1'b0),
.GRS_TYPE_Q0("RESET"),
.GRS_TYPE_Q1("RESET"),
```

```
.GRS_TYPE_Q2("RESET"),
.GRS_TYPE_Q3("RESET"),
.LRS_TYPE_Q0("ASYNC_RESET"),
.LRS_TYPE_Q1("ASYNC_RESET"),
.LRS_TYPE_Q2("ASYNC_RESET"),
.LRS_TYPE_Q3("ASYNC_RESET")
) gtp_iserdes_inst0 (
.RST(1'b0),
.ICE0(1'b1),
.ICE1(1'b0),
.DESCLK (gmii_rx_clk),
.ICLK (gmii_rx_clk),
.ICLKDIV(gmii_rx_clk),
.DI (rgmii_rd[0]),
.BITSLIP(),
.ISHIFTIN0(),
.ISHIFTIN1(),
.IFIFO_WADDR(),
.IFIFO_RADDR(),
.DO({nc1,gmii_rxd_s[4],gmii_rxd_s[0]}),
.ISHIFTOUT0(),
.ISHIFTOUT1()
);
//-----------------------------------------
wire [5:0] nc2;
GTP_ISERDES_E2 #
(
.ISERDES_MODE ("DDR1TO2_SAME_PIPELINED"),
.CASCADE_MODE("MASTER"),
.BITSLIP_EN("FALSE"),
.GRS_EN ("TRUE"),
.NUM_ICE(1'b0),
.GRS_TYPE_Q0("RESET"),
.GRS_TYPE_Q1("RESET"),
.GRS_TYPE_Q2("RESET"),
.GRS_TYPE_Q3("RESET"),
.LRS_TYPE_Q0("ASYNC_RESET"),
.LRS_TYPE_Q1("ASYNC_RESET"),
.LRS_TYPE_Q2("ASYNC_RESET"),
.LRS_TYPE_Q3("ASYNC_RESET")
) gtp_iserdes_inst1 (
.RST(1'b0),
.ICE0(1'b1),
.ICE1(1'b0),
```

```
.DESCLK (gmii_rx_clk),
.ICLK (gmii_rx_clk),
.ICLKDIV(gmii_rx_clk),
.DI (rgmii_rd[1]),
.BITSLIP(),
.ISHIFTIN0(),
.ISHIFTIN1(),
.IFIFO_WADDR(),
.IFIFO_RADDR(),
.DO({nc2,gmii_rxd_s[5],gmii_rxd_s[1]}),
.ISHIFTOUT0(),
.ISHIFTOUT1()
);
//-------------------------------------
wire [5:0] nc3;
GTP_ISERDES_E2 #
(
.ISERDES_MODE ("DDR1TO2_SAME_PIPELINED"),
.CASCADE_MODE("MASTER"),
.BITSLIP_EN("FALSE"),
.GRS_EN ("TRUE"),
.NUM_ICE(1'b0),
.GRS_TYPE_Q0("RESET"),
.GRS_TYPE_Q1("RESET"),
.GRS_TYPE_Q2("RESET"),
.GRS_TYPE_Q3("RESET"),
.LRS_TYPE_Q0("ASYNC_RESET"),
.LRS_TYPE_Q1("ASYNC_RESET"),
.LRS_TYPE_Q2("ASYNC_RESET"),
.LRS_TYPE_Q3("ASYNC_RESET")
) gtp_iserdes_inst2 (
.RST(1'b0),
.ICE0(1'b1),
.ICE1(1'b0),
.DESCLK (gmii_rx_clk),
.ICLK (gmii_rx_clk),
.ICLKDIV(gmii_rx_clk),
.DI (rgmii_rd[2]),
.BITSLIP(),
.ISHIFTIN0(),
.ISHIFTIN1(),
.IFIFO_WADDR(),
.IFIFO_RADDR(),
.DO ({nc3,gmii_rxd_s[6],gmii_rxd_s[2]}),
```

```
.ISHIFTOUT0(),
.ISHIFTOUT1()
);
//---------------------------------------
wire [5:0] nc4;
GTP_ISERDES_E2 #
(
.ISERDES_MODE ("DDR1TO2_SAME_PIPELINED"),
.CASCADE_MODE("MASTER"),
.BITSLIP_EN("FALSE"),
.GRS_EN ("TRUE"),
.NUM_ICE(1'b0),
.GRS_TYPE_Q0("RESET"),
.GRS_TYPE_Q1("RESET"),
.GRS_TYPE_Q2("RESET"),
.GRS_TYPE_Q3("RESET"),
.LRS_TYPE_Q0("ASYNC_RESET"),
.LRS_TYPE_Q1("ASYNC_RESET"),
.LRS_TYPE_Q2("ASYNC_RESET"),
.LRS_TYPE_Q3("ASYNC_RESET")
) gtp_iserdes_inst3 (
.RST(1'b0),
.ICE0(1'b1),
.ICE1(1'b0),
.DESCLK (gmii_rx_clk),
.ICLK (gmii_rx_clk),
.ICLKDIV(gmii_rx_clk),
.DI (rgmii_rd[3]),
.BITSLIP(),
.ISHIFTIN0(),
.ISHIFTIN1(),
.IFIFO_WADDR(),
.IFIFO_RADDR(),
.DO ({nc4,gmii_rxd_s[7],gmii_rxd_s[3]}),
.ISHIFTOUT0(),
.ISHIFTOUT1()
);
//-----------------------------------------
wire [5:0] nc5;
GTP_ISERDES_E2 #
(
.ISERDES_MODE ("DDR1TO2_SAME_PIPELINED"),
.CASCADE_MODE("MASTER"),
.BITSLIP_EN("FALSE"),
```

```
.GRS_EN ("TRUE"),
.NUM_ICE(1'b0),
.GRS_TYPE_Q0("RESET"),
.GRS_TYPE_Q1("RESET"),
.GRS_TYPE_Q2("RESET"),
.GRS_TYPE_Q3("RESET"),
.LRS_TYPE_Q0("ASYNC_RESET"),
.LRS_TYPE_Q1("ASYNC_RESET"),
.LRS_TYPE_Q2("ASYNC_RESET"),
.LRS_TYPE_Q3("ASYNC_RESET")
) gtp_iserdes_inst4 (
.RST(1'b0),
.ICE0(1'b1),
.ICE1(1'b0),
.DESCLK (gmii_rx_clk),
.ICLK (gmii_rx_clk),
.ICLKDIV(gmii_rx_clk),
.DI (rgmii_rx_ctl),
.BITSLIP(),
.ISHIFTIN0(),
.ISHIFTIN1(),
.IFIFO_WADDR(),
.IFIFO_RADDR(),
.DO ({nc5,rgmii_rx_ctl_s,gmii_rx_dv_s}),
.ISHIFTOUT0(),
.ISHIFTOUT1()
);
endmodule
```

模块的端口定义了输入和输出信号，包括复位信号 reset、系统时钟 sys_clk、GMII 和 RGMII 接口的相关信号，以及速度和双工模式的选择信号 speed_selection 和 duplex_mode。其中，rgmii_td、rgmii_tx_ctl、rgmii_txc 是 RGMII 发送端口信号，rgmii_rd、rgmii_rx_ctl、rgmii_rxc 是 RGMII 接收端口信号。gmii_txd、gmii_tx_en、gmii_tx_er、gmii_tx_clk 是 GMII 发送信号，gmii_rxd、gmii_rx_dv、gmii_rx_er、gmii_rx_clk 是 GMII 接收信号。led 用于状态指示。

在模块内部，首先定义了一些中间信号和寄存器，用于数据和状态的暂存和同步。gigabit 信号通过 speed_selection 的高位确定，表示是否处于千兆模式。gmii_tx_clk_s 被分配为 gmii_rx_clk，用于同步发送和接收时钟。

在发送路径上，模块使用了 GTP_OSERDES_E2（输出串并转换器）来将并行的 GMII 数据转换为 RGMII 所需的 DDR（双倍数据速率）串行数据。为了实

现这一点，代码实例化了多个 GTP_OSERDES_E2 模块，每个模块负责一个数据位或控制信号的转换。

具体来说，发送数据部分首先对 GMII 发送数据和控制信号进行寄存器暂存，以消除亚稳态和同步问题。gmii_txd_r、gmii_tx_en_r、gmii_tx_er_r 用于存储当前的发送数据和控制信号，gmii_txd_r_d1、gmii_tx_en_r_d1、gmii_tx_er_r_d1 用于存储前一个时钟周期的数据，这样可以在 DDR 传输中同时获得上升沿和下降沿的数据。

然后，通过实例化 GTP_OSERDES_E2 模块，将 8 位的 GMII 数据分成高 4 位和低 4 位，分别在时钟的上升沿和下降沿传输，形成 RGMII 所需的 4 位 DDR 数据。每个 GTP_OSERDES_E2 实例对应 RGMII 的一个数据位或控制信号，配置为“DDR2TO1_SAME_EDGE”模式，实现从双倍速率到单倍速率的转换。

在接收路径上，模块使用了 GTP_ISERDES_E2（输入串并转换器）来将 RGMII 的 DDR 串行数据转换为并行的 GMII 数据。每个 GTP_ISERDES_E2 实例接收 RGMII 的一个数据位或控制信号，配置为“DDR1TO2_SAME_PIPELINED”模式，实现从单倍速率到双倍速率的转换。接收到的 RGMII 数据在时钟的上升沿和下降沿被采样，然后组合成 8 位的 GMII 数据 gmii_rxd_s。

为了保证接收数据的同步性，代码还使用了一个 PLL（锁相环）模块 pll_sft 对接收时钟 rgmii_rxc 进行相位偏移调整，生成了 gmii_rx_clk，用于驱动接收数据的采样。这个处理是为了补偿 RGMII 接口中的时钟和数据之间的相位偏移，确保在正确的时刻采样数据。

在主逻辑中，always 块用于在 gmii_rx_clk 的时钟域下，将采样到的 GMII 接收数据和控制信号更新到寄存器 gmii_rxd、gmii_rx_dv 和 gmii_rx_er 中。gmii_rx_er 通过 gmii_rx_dv_s 和 rgmii_rx_ctl_s 的异或得到，表示接收错误信号。

在发送时钟域 gmii_tx_clk_s 下，模块通过一系列寄存器和逻辑，生成 RGMII 的发送控制信号 rgmii_tx_ctl_r 和数据 gmii_txd_low。rgmii_tx_ctl_r 是 gmii_tx_en_r 和 gmii_tx_er_r 的异或，用于指示发送数据的有效性和错误状态。gmii_txd_low 根据当前的工作速率 gigabit，在千兆模式下取 gmii_txd_r 的高 4 位，在百兆或十兆模式下取低 4 位。

代码还处理了碰撞检测信号 gmii_col 和载波检测信号 gmii_crs，在半双工模

式下，根据发送和接收的活动状态生成对应的信号。在全双工模式下，这些信号被置为 0。

为了实现对时钟信号的输出，模块还实例化了一个 GTP_OSERDES_E2 模块，用于产生 RGMII 的发送时钟 rgmii_txc。发送时钟以 DDR 模式输出，在 RGMII 接口中用于同步发送的数据和控制信号。

模块还包含一个简单的 LED 闪烁逻辑，用于指示模块的运行状态。在 gmii_tx_clk_s 时钟下，一个计数器 cnt_timer 累加，当达到一定值时翻转 led 信号，实现 LED 的周期性闪烁。

总的来说，这个模块通过使用 FPGA 的高速串并转换器和相位调整技术，实现了 GMII 和 RGMII 接口之间的信号转换，满足了千兆以太网通信的要求。模块精细地处理了发送和接收路径上的数据和控制信号的时序和同步，确保了数据传输的可靠性。同时，通过对速率和双工模式的配置，模块能够适应不同的工作模式，为以太网通信的实现提供了灵活性。

9.3.3gmii_arbi 模块讲解

gmii_arbi 模块的源码如下：

```
1.  `timescale 1ns / 1ps
2.  //////////////////////////////////////////////////////////////////////////////////
3.  // Module Name:    ethernet_test
4.  //////////////////////////////////////////////////////////////////////////////////
5.  module gmii_arbi
6.  (
7.   input                 clk,
8.   input                 rst_n,
9.   input  [1:0]          speed,  //以太网速度
10.  input                 link,       //Link 信号
11. (* MARK_DEBUG="true" *)input                 gmii_rx_dv,//外部的gmii 接收有效
12. (* MARK_DEBUG="true" *)input  [7:0]          gmii_rxd,//外部的gmii 接收数据
13. (* MARK_DEBUG="true" *)input                 gmii_tx_en,//外部的gmii 发送使能
```

```
(* MARK_DEBUG="true" *)input  [7:0]          gmii_txd,      //外部的gmii 发送数据
output reg [31:0]     pack_total_len,     //delay time 1s
output                e_rst_n,
(* MARK_DEBUG="true" *)output reg           e_rx_dv,//仲裁后接收有效
(* MARK_DEBUG="true" *)output reg [7:0]     e_rxd,//仲裁后接收数据
(* MARK_DEBUG="true" *)output reg           e_tx_en,//仲裁后发送使能
(* MARK_DEBUG="true" *)output reg [7:0]     e_txd//仲裁后发送数据
);

reg                   eth_1000m_en    ;
wire                  eth_10_100m_en  ;
reg                   eth_100m_en     ;
reg                   eth_10m_en      ;
reg [1:0]             speed_d0        ;
reg [1:0]             speed_d1        ;
reg [1:0]             speed_d2        ;
reg                   link_d0         ;
reg                   link_d1         ;
reg                   link_d2         ;

wire                  e10_100_tx_en   ;
wire [7:0]            e10_100_txd     ;
wire                  e10_100_rx_dv   ;
wire [7:0]            e10_100_rxd     ;

reg                   e_rst_en        ;
reg [7:0]             e_rst_cnt       ;

assign e_rst_n = link_d2 & e_rst_en ;

always @(posedge clk or negedge rst_n)
  begin
    if (~rst_n)
 begin
   speed_d0 <= 2'b00 ;
   speed_d1 <= 2'b00 ;
   speed_d2 <= 2'b00 ;
   link_d0  <= 1'b0 ;
   link_d1  <= 1'b0 ;
   link_d2  <= 1'b0 ;
```

```
54.  end
55.  else
56.  begin
57.    speed_d0 <= speed    ;
58.    speed_d1 <= speed_d0 ;
59.    speed_d2 <= speed_d1 ;
60.    link_d0  <= link     ;
61.    link_d1  <= link_d0  ;
62.    link_d2  <= link_d1  ;
63.  end
64.   end
65.
66.
67.
68. always @(posedge clk or negedge rst_n)
69.   begin
70.     if (~rst_n)
71.  begin
72.    eth_1000m_en   <= 1'b0  ;
73.    eth_100m_en    <= 1'b0 ;
74.    eth_10m_en     <= 1'b0 ;
75.    pack_total_len <= 32'd2500000 ;
76.  end
77.  else if (speed_d2 == 2'b10)           //1000M
78.  begin
79.    eth_1000m_en   <= 1'b1 ;
80.    eth_100m_en    <= 1'b0 ;
81.    eth_10m_en     <= 1'b0 ;
82.    pack_total_len <= 32'd125000000 ;   //1s
83.  end
84.  else if (speed_d2 == 2'b01)           //100M
85.  begin
86.    eth_1000m_en   <= 1'b0  ;
87.    eth_100m_en    <= 1'b1  ;
88.    eth_10m_en     <= 1'b0  ;
89.    pack_total_len <= 32'd25000000 ;    //1s
90.  end
91.  else if (speed_d2 == 2'b00)           //10M
92.  begin
93.    eth_1000m_en   <= 1'b0  ;
94.    eth_100m_en    <= 1'b0  ;
95.    eth_10m_en     <= 1'b1  ;
96.    pack_total_len <= 32'd2500000 ;     //1s
97.  end
```

```
98.
99.   end
100.
101. always @(posedge clk or negedge rst_n)
102.   begin
103.     if (~rst_n)
104.  begin
105.    e_rx_dv   <= 1'b0 ;
106.    e_rxd     <= 8'd0 ;
107.    e_tx_en   <= 1'b0 ;
108.    e_txd     <= 8'd0 ;
109.  end
110.  else if (eth_1000m_en)
111.  begin
112.    e_rx_dv   <= gmii_rx_dv  ;
113.    e_rxd     <= gmii_rxd    ;
114.    e_tx_en   <= gmii_tx_en  ;
115.    e_txd     <= gmii_txd    ;
116.  end
117.  else if (eth_100m_en | eth_10m_en)
118.  begin
119.    e_rx_dv   <= e10_100_rx_dv ;
120.    e_rxd     <= e10_100_rxd   ;
121.    e_tx_en   <= e10_100_tx_en ;
122.    e_txd     <= e10_100_txd   ;
123.  end
124.   end
125.
126.
127.
128. always @(posedge clk or negedge rst_n)
129.   begin
130.     if (~rst_n)
131.       e_rst_en <= 1'b1 ;
132.     else if (speed_d2 != speed_d1)
133.       e_rst_en <= 1'b0 ;
134.     else if (e_rst_cnt == 8'd200)
135.     e_rst_en <= 1'b1 ;
136.   end
137.
138. always @(posedge clk or negedge rst_n)
139.   begin
140.     if (~rst_n)
141.       e_rst_cnt <= 8'd0 ;
```

```
    else if (~e_rst_en)
      e_rst_cnt <= e_rst_cnt + 1'b1 ;
    else
   e_rst_cnt <= 8'd0 ;
  end

assign eth_10_100m_en = eth_100m_en | eth_10m_en ;

gmii_tx_buffer tx_buffer_inst
(
 .clk                    (clk                 ),
 .rst_n                  (e_rst_n             ),
 .eth_10_100m_en         (eth_10_100m_en      ),
 .link                   (e_rst_n             ),
 .gmii_tx_en             (gmii_tx_en          ),
 .gmii_txd               (gmii_txd            ),
 .e10_100_tx_en          (e10_100_tx_en       ),
 .e10_100_txd            (e10_100_txd         )
);


gmii_rx_buffer  rx_buffer_inst
(
 .clk                    (clk                 ),
 .rst_n                  (e_rst_n             ),
 .link                   (e_rst_n             ),
 .eth_100m_en            (eth_100m_en         ),
 .eth_10m_en             (eth_10m_en          ),
 .gmii_rx_dv             (gmii_rx_dv          ),
 .gmii_rxd               (gmii_rxd            ),
 .e10_100_rx_dv          (e10_100_rx_dv       ),
 .e10_100_rxd            (e10_100_rxd         )

);

endmodule
```

模块的输入包括系统时钟 clk、复位信号 rst_n、以太网速度选择信号 speed（两位，00 表示 10Mbps，01 表示 100Mbps，10 表示 1000Mbps），链路状态信号 link，以及 GMII 接口的发送和接收信号 gmii_tx_en、gmii_txd、gmii_rx_dv、gmii_rxd。输出包括一个 32 位的 pack_total_len，用于指示数据包的总长度，e_rst_n 信号，用于对 MAC 层的复位控制，以及仲裁后的发送和接收信号 e_tx_en、e_txd、

e_rx_dv、e_rxd，供 MAC 层使用。

在模块内部，首先对 speed 和 link 信号进行多级寄存器同步，防止由于信号的异步变化导致的亚稳态问题。通过寄存器 speed_d0、speed_d1、speed_d2 和 link_d0、link_d1、link_d2，将 speed 和 link 信号同步到系统时钟域中。

接下来，根据同步后的速度信号 speed_d2，确定当前的以太网速度模式。若 speed_d2 为 2'b10，表示处于 1000Mbps（千兆）模式，设置 eth_1000m_en 为高电平，eth_100m_en 和 eth_10m_en 为低电平，同时将 pack_total_len 设为 32'd125000000，用于后续的计时或数据长度控制。若 speed_d2 为 2'b01，表示处于 100Mbps 模式，设置 eth_100m_en 为高电平，其他为低电平，pack_total_len 设为 32'd25000000。若 speed_d2 为 2'b00，表示处于 10Mbps 模式，设置 eth_10m_en 为高电平，其他为低电平，pack_total_len 设为 32'd2500000。

模块还通过 e_rst_en 和 e_rst_cnt 实现对 MAC 层的复位控制。当速度信号 speed_d2 发生变化时，e_rst_en 被置为低电平，开始对 MAC 层进行复位。e_rst_cnt 开始计数，当计数达到 8'd200 时，e_rst_en 重新置为高电平，结束复位过程。e_rst_n 信号由 link_d2 和 e_rst_en 的逻辑与得到，只有当链路连接且复位结束时，e_rst_n 为高电平，MAC 层才能正常工作。

在数据传输方面，当 eth_1000m_en 为高电平，即处于千兆模式时，模块直接将 GMII 接口的发送和接收信号 gmii_tx_en、gmii_txd、gmii_rx_dv、gmii_rxd 传递给 MAC 层的接口 e_tx_en、e_txd、e_rx_dv、e_rxd。当处于 100Mbps 或 10Mbps 模式，即 eth_100m_en 或 eth_10m_en 为高电平时，数据需要经过速率适配模块进行处理。模块实例化了 gmii_tx_buffer 和 gmii_rx_buffer，用于在较低速率下对发送和接收数据进行缓存和速率匹配。

gmii_tx_buffer 模块负责在 10Mbps 和 100Mbps 模式下，对发送数据进行适配。它根据时钟和复位信号，以及速度模式，缓存来自 GMII 接口的发送数据 gmii_tx_en 和 gmii_txd，并输出适配后的数据 e10_100_tx_en 和 e10_100_txd。gmii_rx_buffer 模块则负责接收方向的速率适配，处理 GMII 接口的接收数据 gmii_rx_dv 和 gmii_rxd，输出适配后的数据 e10_100_rx_dv 和 e10_100_rxd。

此外，模块通过信号 eth_10_100m_en（由 eth_100m_en 和 eth_10m_en 的或运算得到）来指示当前是否处于 10Mbps 或 100Mbps 模式，用于控制数据的仲裁

逻辑。在这种模式下，仲裁后的发送和接收信号从速率适配模块获取，否则直接从 GMII 接口获取。

通过上述逻辑，gmii_arbi 模块实现了对不同速率的以太网数据进行仲裁和适配，确保在各种速率下，MAC 层都能正确地发送和接收数据。模块根据速度和链路状态动态调整数据路径，并在速度变化时对 MAC 层进行复位，保证系统的稳定性和可靠性。

这个模块对于以太网通信的实现非常重要，它能够根据不同的网络速度，灵活地调整数据传输方式，确保数据的完整性和有效性。同时，通过对复位信号的控制，模块能够在链路状态变化时，及时地对 MAC 层进行复位，防止错误数据的产生。这些特性使得 gmii_arbi 模块在以太网系统中扮演了关键的角色，为上层的 MAC 层提供了稳定可靠的通信接口。

9.3.4mac_test 模块讲解

mac_test 模块的源码如下：

```
1.  ////////////////////////////////////////////////////////////////////
    //////////////////
2.  //Module Name : mac_top
3.  //Description :
4.  //
5.  ////////////////////////////////////////////////////////////////////
    //////////////////
6.  //`define TEST_SPEED
7.  `timescale 1 ns/ 1 ns
8.  module mac_test
9.  (
10.  input                rst_n  ,
11.  input [31:0]         pack_total_len,
12.  input                gmii_tx_clk ,
13.  input                gmii_rx_clk ,
14.  input                gmii_rx_dv,
15.  input  [7:0]         gmii_rxd,
16.  output reg           gmii_tx_en,
17.  output reg [7:0]     gmii_txd,
18. output reg [15:0]     udp_send_data_length,
19. output                write_sel,
```

```
output                udp_rec_data_valid,
    output  al_full,
    output  emp_sum,
output  checksum_wr,
    output [4:0] use_rd,
 output [7:0]           udp_rec_ram_rdata ,
 output [10:0]          udp_rec_ram_read_addr ,
 output [15:0]          udp_rec_data_length
);

localparam UDP_WIDTH = 32 ;
localparam UDP_DEPTH = 5 ;


reg                   gmii_rx_dv_d0 ;
reg   [7:0]           gmii_rxd_d0 ;
wire                  gmii_tx_en_tmp ;
wire   [7:0]          gmii_txd_tmp ;

reg   [7:0]           ram_wr_data ;
reg                   ram_wr_en ;
wire                  udp_ram_data_req ;
reg   [15:0]          udp_send_data_length ;

wire  [7:0]           tx_ram_wr_data ;
wire                  tx_ram_wr_en ;
wire                  udp_tx_req ;
wire                  arp_request_req ;
wire                  mac_send_end ;
reg                   write_end ;

wire [7:0]            udp_rec_ram_rdata ;
reg  [10:0]           udp_rec_ram_read_addr ;
wire [15:0]           udp_rec_data_length ;
wire                  udp_rec_data_valid ;

wire                  udp_tx_end  ;
wire                  almost_full ;

reg                   udp_ram_wr_en ;
reg                   udp_write_end ;
wire                  write_ram_end ;
reg  [31:0]           wait_cnt ;
reg  [UDP_WIDTH-1:0]  udp_data [UDP_DEPTH-1:0];
```

```
64.
65. reg  [4:0] i;
66. reg  [1:0] j ;
67.
68. reg  write_sel ;
69.
70. wire button_negedge ;
71.
72. wire mac_not_exist ;
73. wire arp_found ;
74.
75. parameter IDLE          = 9'b000_000_001 ;
76. parameter ARP_REQ       = 9'b000_000_010 ;
77. parameter ARP_SEND      = 9'b000_000_100 ;
78. parameter ARP_WAIT      = 9'b000_001_000 ;
79. parameter GEN_REQ       = 9'b000_010_000 ;
80. parameter WRITE_RAM     = 9'b000_100_000 ;
81. parameter SEND          = 9'b001_000_000 ;
82. parameter WAIT          = 9'b010_000_000 ;
83. parameter CHECK_ARP     = 9'b100_000_000 ;
84.
85.
86. reg [8:0]    state  ;
87. reg [8:0]    next_state ;
88. reg  [15:0]  ram_cnt ;
89. reg    almost_full_d0 ;
90. reg    almost_full_d1 ;
91. always @(posedge gmii_tx_clk or negedge rst_n)
92.   begin
93.     if (~rst_n)
94.       state  <=  IDLE  ;
95.     else
96.       state  <= next_state ;
97.   end
98.
99. always @(*)
100.   begin
101.     case(state)
102.       IDLE        :
103.         begin
```

```
104.         if (wait_cnt == pack_total_len)    //1s
105.             next_state <= ARP_REQ ;
106.           else
107.             next_state <= IDLE ;
108.         end
109.
110.       ARP_REQ     :
111.         next_state <= ARP_SEND ;
112.       ARP_SEND    :
113.         begin
114.           if (mac_send_end)
115.             next_state <= ARP_WAIT ;
116.           else
117.             next_state <= ARP_SEND ;
118.         end
119.       ARP_WAIT    :
120.         begin
121.           if (arp_found)
122.             next_state <= WAIT ;
123.           else if (wait_cnt == pack_total_len)
124.             next_state <= ARP_REQ ;
125.           else
126.             next_state <= ARP_WAIT ;
127.         end
128.       GEN_REQ     :
129.         begin
130.           if (udp_ram_data_req)
131.             next_state <= WRITE_RAM ;
132.           else
133.             next_state <= GEN_REQ ;
134.         end
135.       WRITE_RAM   :
136.         begin
137.     `ifdef TEST_SPEED
138.     if (ram_cnt == udp_send_data_length - 1)
139.     `else
140.           if (write_ram_end)
141.     `endif
142.             next_state <= WAIT     ;
143.           else
144.             next_state <= WRITE_RAM ;
145.         end
146.
147.       SEND        :
```

```
148.         begin
149.           if (udp_tx_end)
150.             next_state <= WAIT ;
151.           else
152.             next_state <= SEND ;
153.         end
154.
155.       WAIT        :
156.         begin
157.     `ifdef TEST_SPEED
158.           if (wait_cnt == 32'd90)              //frame gap
159.     `else
160.     if (wait_cnt == pack_total_len)    //1s
161.     `endif
162.             next_state <= CHECK_ARP ;
163.           else
164.             next_state <= WAIT ;
165.         end
166.       CHECK_ARP   :
167.         begin
168.           if (mac_not_exist)
169.             next_state <= ARP_REQ ;
170.           else if (almost_full_d1)
171.             next_state <= CHECK_ARP ;
172.           else
173.             next_state <= GEN_REQ ;
174.         end
175.       default     :
176.         next_state <= IDLE ;
177.     endcase
178.   end
179.
180.
181. assign write_ram_end        = (write_sel)? udp_write_end : write
   _end ;
182. assign tx_ram_wr_data       = (write_sel)? udp_rec_ram_rdata : r
   am_wr_data ;
183. assign tx_ram_wr_en         = (write_sel)? udp_ram_wr_en : ram_w
   r_en ;
184.
185.
186. always@(posedge gmii_rx_clk or negedge rst_n)
```

```
187.   begin
188.     if(rst_n == 1'b0)
189.       begin
190.         gmii_rx_dv_d0 <= 1'b0 ;
191.         gmii_rxd_d0   <= 8'd0 ;
192.       end
193.     else
194.       begin
195.         gmii_rx_dv_d0 <= gmii_rx_dv ;
196.         gmii_rxd_d0   <= gmii_rxd ;
197.       end
198.   end
199.
200. always@(posedge gmii_tx_clk or negedge rst_n)
201.   begin
202.     if(rst_n == 1'b0)
203.       begin
204.         gmii_tx_en <= 1'b0 ;
205.         gmii_txd   <= 8'd0 ;
206.       end
207.     else
208.       begin
209.         gmii_tx_en <= gmii_tx_en_tmp ;
210.         gmii_txd   <= gmii_txd_tmp ;
211.       end
212.   end
213.
214.
215. mac_top mac_top0
216. (
217.  .gmii_tx_clk                (gmii_tx_clk)                  ,
218.  .gmii_rx_clk                (gmii_rx_clk)                  ,
219.  .rst_n                      (rst_n)  ,
220.
221.  .source_mac_addr            (48'h00_0a_35_01_fe_c0)   ,
   //source mac address
222.  .TTL                        (8'h80),
223.  .source_ip_addr             (32'hc0a80002),
224.  .destination_ip_addr        (32'hc0a80003),
```

```
.udp_send_source_port        (16'h1f90),
.udp_send_destination_port   (16'h1f90),

.ram_wr_data                 (tx_ram_wr_data) ,
.ram_wr_en                   (tx_ram_wr_en),
.udp_ram_data_req            (udp_ram_data_req),
.udp_send_data_length        (udp_send_data_length),
.udp_tx_end                  (udp_tx_end                  ),
.almost_full                 (almost_full                 ),

.udp_tx_req                  (udp_tx_req),
.arp_request_req             (arp_request_req ),

.mac_send_end                (mac_send_end),
.mac_data_valid              (gmii_tx_en_tmp),
.mac_tx_data                 (gmii_txd_tmp),
.rx_dv                       (gmii_rx_dv_d0   ),
.mac_rx_datain               (gmii_rxd_d0 ),

.udp_rec_ram_rdata           (udp_rec_ram_rdata),
.udp_rec_ram_read_addr       (udp_rec_ram_read_addr),
.udp_rec_data_length         (udp_rec_data_length ),

.udp_rec_data_valid          (udp_rec_data_valid),
.arp_found                   (arp_found ),
.mac_not_exist               (mac_not_exist ),
        .al_full (al_full),
        .emp_sum (emp_sum),
.checksum_wr(checksum_wr),
        .use_rd  (use_rd)
) ;


always @(*)
  begin
    udp_data[0] <={"H","E","L","L"};
    udp_data[1] <={"O"," ","A","L"};
    udp_data[2] <={"I","N","X"," "};
    udp_data[3] <={"H","E","I","J"};
    udp_data[4] <={"I","N"," ","\n"};

  end
```

```
268.
269. //reg    almost_full_d0 ;
270. //reg    almost_full_d1 ;
271.
272. always@(posedge gmii_rx_clk or negedge rst_n)
273.   begin
274.     if(rst_n == 1'b0)
275.       begin
276.         almost_full_d0   <= 1'b0 ;
277.         almost_full_d1   <= 1'b0 ;
278.       end
279.     else
280.       begin
281.         almost_full_d0   <= almost_full ;
282.         almost_full_d1   <= almost_full_d0 ;
283.       end
284.   end
285.
286. always@(posedge gmii_rx_clk or negedge rst_n)
287.   begin
288.     if(rst_n == 1'b0)
289.    udp_send_data_length <= 16'd0 ;
290.     else if (write_sel)
291.   udp_send_data_length <= udp_rec_data_length - 8  ;
292.   else
293.   `ifdef TEST_SPEED
294.        udp_send_data_length <= 16'd1000 ;
295.   `else
296.     udp_send_data_length <= 4*UDP_DEPTH ;
297.         //udp_send_data_length <=16'd20 ;
298.   `endif
299.   end
300.
301.
302. always@(posedge gmii_tx_clk or negedge rst_n)
303.   begin
304.     if(rst_n == 1'b0)
305.       write_sel <= 1'b0 ;
306.     else if (state == WAIT)
```

```
307.       begin
308.         if (udp_rec_data_valid)
309.           write_sel <= 1'b1 ;
310.         else
311.           write_sel <= 1'b0 ;
312.       end
313.   end
314.
315. assign udp_tx_req    = (state == GEN_REQ) ;
316. assign arp_request_req  = (state == ARP_REQ) ;
317.
318. always@(posedge gmii_tx_clk or negedge rst_n)
319.   begin
320.     if(rst_n == 1'b0)
321.       wait_cnt <= 0 ;
322.     else if ((state==IDLE||state == WAIT || state == ARP_WAIT) &
    & state != next_state)
323.       wait_cnt <= 0 ;
324.     else if (state==IDLE||state == WAIT || state == ARP_WAIT)
325.       wait_cnt <= wait_cnt + 1'b1 ;
326.   else
327.     wait_cnt <= 0 ;
328.   end
329.
330.
331. `ifdef TEST_SPEED
332. /************************************************************/
333. //Test ethernet speed
334. //reg  [15:0]     ram_cnt ;
335. always@(posedge gmii_tx_clk or negedge rst_n)
336.  begin
337.    if(rst_n == 1'b0)
338.      ram_cnt <= 11'd0 ;
339.    else if (state == WRITE_RAM)
340.      ram_cnt <= ram_cnt + 1'b1 ;
341.    else
342.      ram_cnt <= 11'd0 ;
343.  end
344.
345. always@(posedge gmii_tx_clk or negedge rst_n)
346.  begin
347.    if(rst_n == 1'b0)
```

```
      ram_wr_en <= 1'b0 ;
    else if (state == WRITE_RAM)
      ram_wr_en <= 1'b1 ;
    else
      ram_wr_en <= 1'b0 ;
  end


always@(posedge gmii_tx_clk or negedge rst_n)
  begin
    if(rst_n == 1'b0)
      ram_wr_data <= 8'd0 ;
    else if (state == WRITE_RAM)
      ram_wr_data <= ram_cnt[7:0] ;
    else
      ram_wr_data <= 8'd0 ;
  end
/*************************************************************/
`else
always@(posedge gmii_tx_clk or negedge rst_n)
   begin
     if(rst_n == 1'b0)
       begin
         write_end  <= 1'b0;
         ram_wr_data <= 0;
         ram_wr_en  <= 0 ;
         i <= 0 ;
         j <= 0 ;
       end
     else if (state == WRITE_RAM)
       begin
         if(i == 5)
           begin
             ram_wr_en <=1'b0;
             write_end <= 1'b1;
           end
         else
           begin
             ram_wr_en <= 1'b1 ;
             write_end <= 1'b0 ;
             j <= j + 1'b1 ;
```

```
            case(j)
              2'd0    :  ram_wr_data <= udp_data[i][31:24] ;
              2'd1    :  ram_wr_data <= udp_data[i][23:16] ;
              2'd2    :  ram_wr_data <= udp_data[i][15:8] ;
              2'd3    :  ram_wr_data <= udp_data[i][7:0] ;
              default :  ram_wr_data <= 8'h00 ;
            endcase

            if (j == 3)
              begin
                j <= 0 ;
                i <= i + 1'b1;
              end
          end
      end
    else
      begin
        write_end  <= 1'b0;
        ram_wr_data <= 0;
        ram_wr_en  <= 0 ;
        i <= 0 ;
        j <= 0 ;
      end
  end
`endif

//send udp received data to udp tx ram
always@(posedge gmii_tx_clk or negedge rst_n)
  begin
    if(rst_n == 1'b0)
      udp_rec_ram_read_addr <= 11'd0 ;
    else if (state == WRITE_RAM)
      udp_rec_ram_read_addr <= udp_rec_ram_read_addr + 1'b1 ;
    else
      udp_rec_ram_read_addr <= 11'd0 ;
  end

always@(posedge gmii_tx_clk or negedge rst_n)
  begin
    if(rst_n == 1'b0)
```

```
429.       udp_ram_wr_en <= 1'b0 ;
430.     else if (state == WRITE_RAM && udp_rec_ram_read_addr < udp_r
   ec_data_length - 8)
431.       udp_ram_wr_en <= 1'b1 ;
432.     else
433.       udp_ram_wr_en <= 1'b0 ;
434.   end
435.
436. always@(posedge gmii_tx_clk or negedge rst_n)
437.   begin
438.     if(rst_n == 1'b0)
439.       udp_write_end <= 1'b0 ;
440.     else if (state == WRITE_RAM && udp_rec_ram_read_addr == udp_
   rec_data_length - 8)
441.       udp_write_end <= 1'b1 ;
442.     else
443.       udp_write_end <= 1'b0 ;
444.   end
445.
446.
447. endmodule
448.
449.
```

首先，模块的输入和输出端口定义了与 GMII 接口的交互信号，以及一些用于控制和数据传输的信号。输入端口包括：

rst_n：复位信号，低电平有效。

pack_total_len：用于设定等待时间或数据包总长度的 32 位信号。

gmii_tx_clk 和 gmii_rx_clk：GMII 接口的发送和接收时钟信号。

gmii_rx_dv 和 gmii_rxd：GMII 接口的接收数据有效信号和接收数据总线。

输出端口包括：

gmii_tx_en 和 gmii_txd：GMII 接口的发送使能信号和发送数据总线。

udp_send_data_length：UDP 发送数据的长度。

write_sel：写入选择信号，用于选择数据源。

udp_rec_data_valid：UDP 接收数据有效信号。

其他信号如 al_full、emp_sum、checksum_wr、use_rd、udp_rec_ram_rdata、udp_rec_ram_read_addr、udp_rec_data_length，用于内部数据处理和状态指示。

模块内部首先定义了一些局部参数和寄存器，包括 UDP 数据的宽度和深度，

用于存储发送的数据内容。接下来，定义了一些用于数据暂存和控制的寄存器，如 gmii_rx_dv_d0、gmii_rxd_d0、ram_wr_data、ram_wr_en 等。

模块通过实例化一个 mac_top 子模块，完成 MAC 层的主要功能。mac_top 模块负责处理 ARP 请求、UDP 数据的发送和接收，以及与 GMII 接口的具体交互。

在主模块中，定义了一个状态机，用于控制数据发送和接收的流程。状态机的各个状态包括：

IDLE：空闲状态，等待计时器达到预设的 pack_total_len，然后进入 ARP 请求状态。

ARP_REQ：发送 ARP 请求的状态，触发 ARP 请求的发送。

ARP_SEND：ARP 请求发送中，等待发送完成后进入 ARP 等待状态。

ARP_WAIT：等待 ARP 响应，如果在等待时间内收到 ARP 响应，则进入等待状态，否则重新发送 ARP 请求。

GEN_REQ：生成 UDP 发送请求的状态，等待 UDP 模块准备好数据传输。

WRITE_RAM：将要发送的数据写入 RAM 的状态，根据不同的测试模式，可能是生成测试数据或发送接收到的 UDP 数据。

SEND：数据发送状态，等待 UDP 数据发送完成。

WAIT：发送完成后的等待状态，等待一段时间后检查 ARP 表。

CHECK_ARP：检查 ARP 表中是否存在目标 MAC 地址，如果不存在，则重新发送 ARP 请求。

在状态机的控制下，模块完成了数据的发送和接收流程。首先，在空闲状态下，模块等待计时器达到设定值，然后发送 ARP 请求，尝试获取目标 IP 地址对应的 MAC 地址。在成功获取 MAC 地址后，模块生成 UDP 发送请求，将数据写入发送缓存。如果处于测试模式，模块会生成固定的测试数据；如果有接收到的 UDP 数据需要转发，模块会将接收到的数据写入发送缓存。

在数据写入完成后，模块进入发送状态，等待 UDP 数据发送完成。发送完成后，进入等待状态，等待一段时间后再次检查 ARP 表，以确保目标 MAC 地址仍然有效。如果目标 MAC 地址失效，模块会重新发送 ARP 请求。

模块还处理了 GMII 接口的发送和接收信号的同步和暂存。通过对

gmii_rx_dv、gmii_rxd、gmii_tx_en、gmii_txd 信号进行寄存器同步，确保数据在不同时钟域之间的正确传输。

在 UDP 数据处理方面，模块定义了一个 UDP 数据数组 udp_data，用于存储要发送的测试数据。在非测试模式下，模块会根据接收到的 UDP 数据长度，动态调整发送的数据长度 udp_send_data_length。

模块还通过一些控制信号，如 udp_tx_req、arp_request_req，与 mac_top 子模块进行通信，触发 ARP 请求和 UDP 数据发送。

计时器 wait_cnt 用于在不同状态之间进行时间控制，如在空闲状态下等待一定时间再发送 ARP 请求，在等待状态下等待一定时间再检查 ARP 表。

模块还定义了一些条件编译指令，如 ifdefTEST_SPEED，用于控制测试模式下的行为。在测试速度时，模块会生成大量的数据用于发送，以测试以太网的传输速度。

在发送数据时，模块根据当前的状态，控制数据写入发送缓存的流程。在 WRITE_RAM 状态下，模块会根据数据计数器 ram_cnt，将数据写入发送缓存 ram_wr_data，并控制写使能信号 ram_wr_en。

对于接收到的 UDP 数据，模块在需要转发时，会将数据从接收缓存读出，写入发送缓存，以实现数据的转发功能。

总体来说，这个 mac_test 模块通过状态机的控制，实现了以太网 MAC 层的 ARP 请求、UDP 数据的发送和接收、数据缓存的读写，以及与 GMII 接口的交互。模块的设计考虑了测试模式和正常工作模式下的不同需求，具有一定的灵活性和可扩展性。

9.3.5power_on_rst 模块讲解

power_on_rst 模块源码如下：

```
1.  `timescale 1ns / 1ps
2.
3.  module power_on_rst
4.          #(
5.            parameter CLK_FRE  = 50,
6.            parameter DELAY_MS = 50
7.          )
8.          (
```

```
          input clk,
          input rst_n,

          output power_on_rstn
        );

reg [31:0] rst_cnt ;
localparam RESET_DELAY = CLK_FRE*1000*DELAY_MS ;  //

always @(posedge clk or negedge rst_n)
  begin
    if (!rst_n)
      rst_cnt <= 32'd0 ;
    else if (rst_cnt < RESET_DELAY)  //50ms
      rst_cnt <= rst_cnt + 1'b1 ;
    else
      rst_cnt <= rst_cnt ;
  end

assign power_on_rstn = (rst_cnt < RESET_DELAY)? 1'b0 : 1'b1 ;

endmodule
```

模块采用参数化设计，具有灵活性，能够适应不同的时钟频率和延时需求。参数 CLK_FRE 代表时钟频率，默认值为 50MHz，即输入时钟 clk 的频率。参数 DELAY_MS 表示延时时间，默认值为 50 毫秒，即复位信号保持低电平的持续时间。通过调整这两个参数，可以根据系统需求设置不同的复位延时。

在模块内部，定义了一个 32 位的寄存器 rst_cnt 作为计数器，用于计数时钟周期数。同时，计算出一个局部参数 RESET_DELAY，表示需要延时的总时钟周期数，计算方式为 RESET_DELAY=CLK_FRE*1000*DELAY_MS。这里，CLK_FRE 以 MHz 为单位，DELAY_MS 以毫秒为单位，乘以 1000 是将毫秒转换为微秒，再乘以时钟频率，得到总的时钟周期数。

计数器的工作机制是在每个时钟上升沿，根据复位信号 rst_n 的状态，决定计数器的行为。当 rst_n 为低电平时，表示外部复位激活，计数器 rst_cnt 被清零。当 rst_n 为高电平且 rst_cnt 小于 RESET_DELAY 时，计数器在每个时钟周期递增 1。当计数器达到或超过 RESET_DELAY 时，计数器保持当前值不变，不再递增。

输出的复位信号 power_on_rstn 根据计数器的值确定。当 rst_cnt 小于 RESET_DELAY 时，power_on_rstn 为低电平，表示系统仍处于复位状态。当 rst_cnt

达到或超过 RESET_DELAY 时，power_on_rstn 变为高电平，表示复位结束，系统开始正常运行。通过这种方式，模块实现了在上电或复位后，延时一段指定的时间再释放复位信号的功能。

这个 power_on_rst 模块对于需要在上电后等待电源和时钟稳定的系统非常有用，可以避免因电源抖动或时钟不稳定导致的系统异常行为。通过参数化的设计，模块可以灵活地适应不同的系统要求，确保系统在合适的时机进入工作状态，提升整体的可靠性。

9.5 实验现象

用网线连接 PT2T70H 开发板网口和 PC 端口；

设置接收端（PC 端）IP 地址为 192.168.0.3，开发板的 IP 地址为 192.168.0.2；

通过命令提示符，输入 arp -a，可以查到 IP：192.168.0.2，MAC：00_0a_35_01_fe_c0; 并且可通过 ping 验证数据链路是否正常连接以及数据传输是否正常。

10.PCIE 通信测试实验例程

10.1 实验目的

完成 PCIE 通信测试。

10.2 实验原理

PG2L100H 集成内置了线速率高达 6.6Gbps 高速串行接口模块，即 HSSTLP。PG2L100H 开发板提供一个 PCIe x4 接口，PCIE 卡的外形尺寸符合标准 PCIe 卡电气规范要求，可直接在普通 PC 的 x4 PCIe 插槽上使用。

10.2.1 PCIE 简介

PCIE IP 符合 PCI Express® Base Specification Revision 2.1[8]协议和 PHY Interface for the PCI ExpressTM Architecture Version 2.00[12]（数据通路扩展为 32 bits）协议。

功能特性	特性说明
支持配置 Device Type	PCI Express Endpoint
	Legacy PCI Express Endpoint
	Root Port of PCI Express Root Complex
支持配置 Max Link Width	x1
	x2
	x4
支持配置 Max Link Speed	2.5GT/s
	5GT/s
支持 Max Link Width 设置为 x1 时可选 LPLL	
支持 100MHz Reference Clk	-
支持 Upconfigure Capable	-
支持 AXI-Stream Slave 个数选择	1
	2
	3
支持 Debug 接口	-
支持通过 Apb 动态配置 PCIe Configuration Space	
支持 Receive Queue Management	-
支持 Lane Reversal	-
支持 Force No Scrambling	-

支持配置 ID	支持配置 Vendor ID
	支持配置 Device ID
	支持配置 Revision ID
	PCI Express Endpoint、Legacy PCI Express Endpoint 支持配置 Subsystem Vendor ID
	PCI Express Endpoint、Legacy PCI Express Endpoint 支持配置 Subsystem ID
	支持配置 Classcode
支持 BAR 配置	PCI Express Endpoint、Legacy PCI Express Endpoint 支持配置 6 个 BAR
	Root Port of PCI Express Root Complex 仅支持配置 BAR0、BAR1
	PCI Express Endpoint 支持配置为 Memory BAR
	Legacy PCI Express Endpoint、Root Port of PCI Express Root Complex 支持配置为 Memory、IO BAR
	支持 32bit BAR
	32bit BAR 支持配置大小为 256 Byte -2G Byte
	BAR0、BAR2、BAR4 支持 64bit BAR
	64bit BAR 支持 Prefetchable
	64bit BAR 支持配置大小为 256 Byte -8E Byte
	支持 Expansion ROM BAR
	Expansion ROM BAR 支持配置大小为 2K Byte -16M Byte
支持配置 Max Payload Size	128 Byte
	256 Byte
	512 Byte
	1024 Byte
支持配置 Extended Tag Field 与 Extended Tag Default	_
支持 Atomic 事务	-
RC 时支持设置 Read Completion Boundary	_
支持配置 Target Link Speed	-
RC 时支持设置 CRS Software Visibility	_
支持设置 ECRC Generation Capable	_
默认使能 ECRC Check Capable	-
支持 INIT 中断	PCI Express Endpoint、Legacy PCI Express Endpoint 只支持 INTA

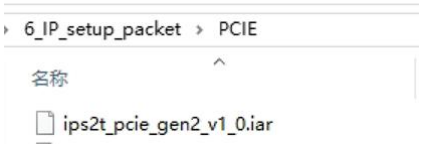
	Root Port of PCI Express Root Complex 支持 INTA、INTB、INTC、INTD
支持 MSI 中断	支持 64-bit Address MSI 中断
	支持 Multiple Message Capable：1、2、4、8、16、32 个 Vectors
	支持 Per Vector Masking Capable
支持 MSIx 中断	支持配置 Table Size 、Offset 与 BIR
	支持配置 PBA Offset 与 BIR

注：“-”表示无该项说明。

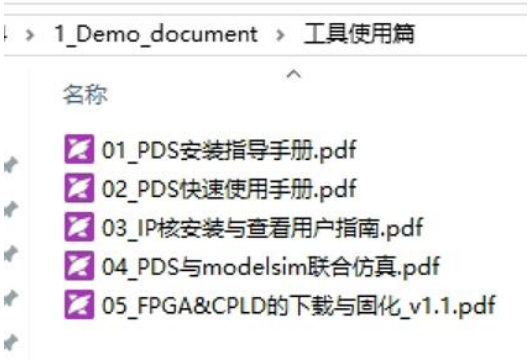
10.3 工程说明

10.3.1 安装 PCIE IP 核

PDS 安装后，需手动添加 PCIE IP，请按以下步骤完成：
PCIE IP 文件：6_IP_setup_packet\ips2tl_pcie_gen2_v1_0c.iar

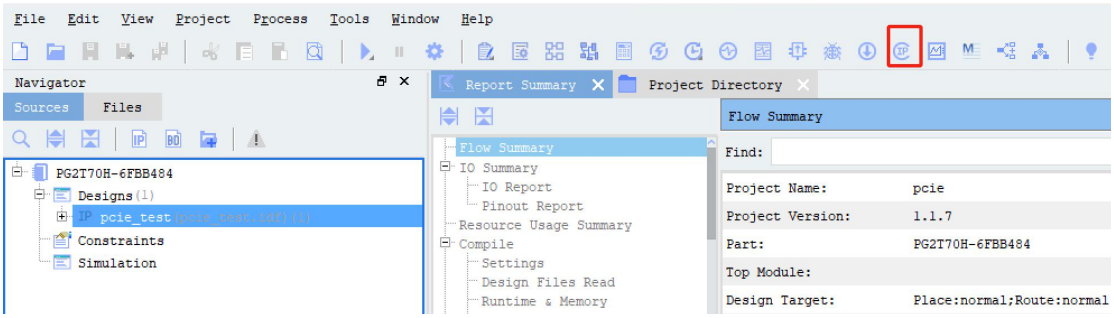


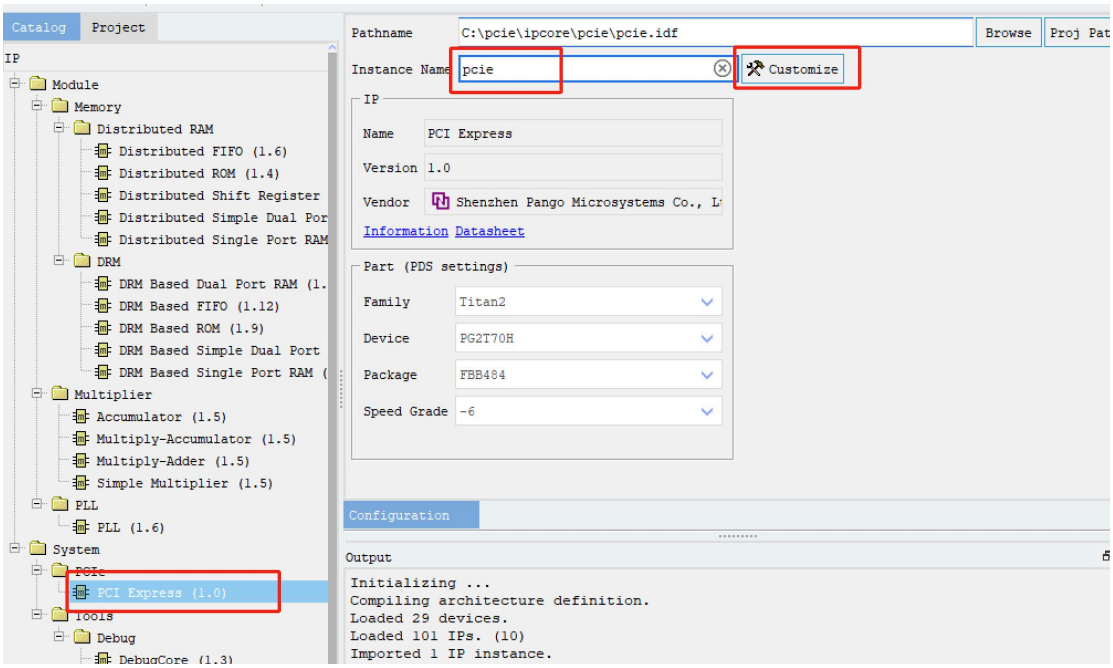
IP 安装步骤：请查看 工具使用篇\03_IP 核安装与查看用户指南



10.3.2 PCIE 参考设计例程

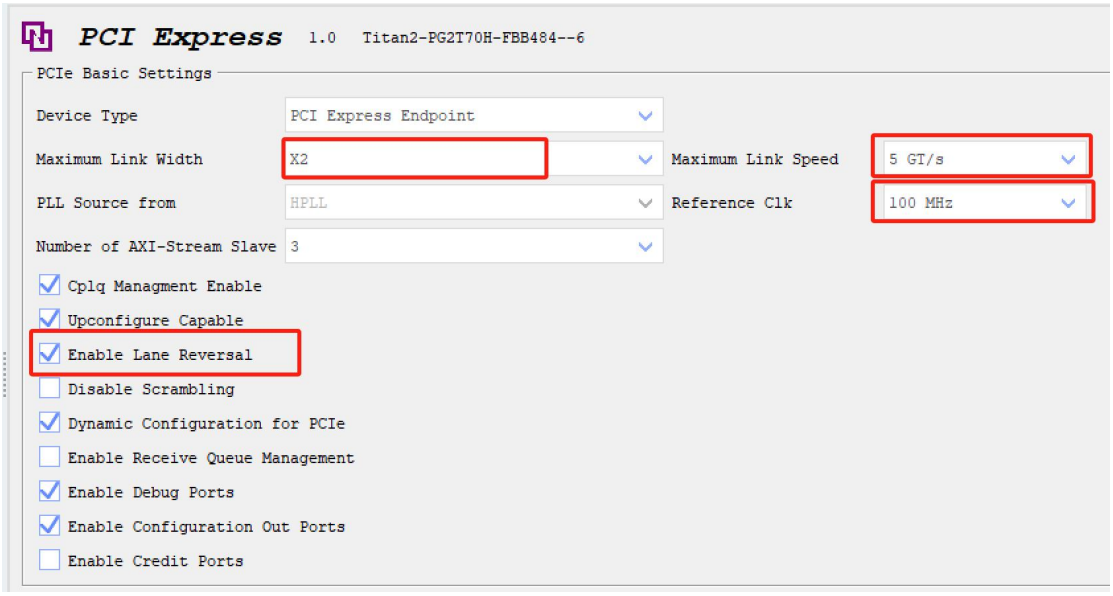
打开 PDS 软件，新建工程 pcie_test，点开如下图标，打开 IP Compiler；



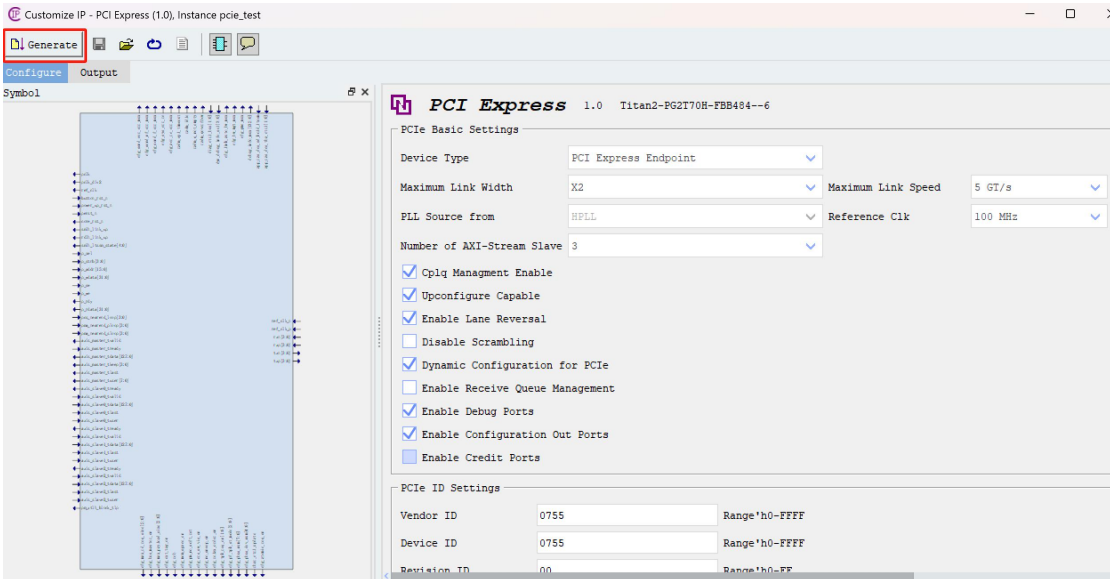


选择 PCIE IP，取名，然后点击 Customize；

在 PCIE 设置界面中：根据开发板配置 lane 数，可选择 X2，配置参考时钟，可参考下图：



需要注意的是，需要勾选上 Enable Lane Reversal，否则会导致 PCIE 实验失败。

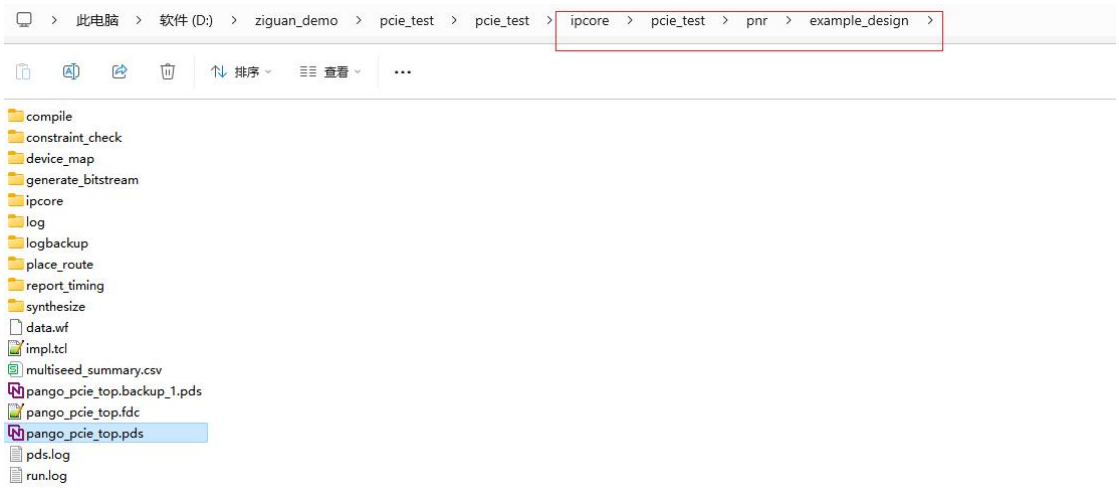


其他设置可保持默认，点击 Generate 生成 PCIE IP。

关闭本工程，按此路径打开 Example 工程：

Xxxxx\pcie_test\ipcore\pcie_test\pnr\example_design

主要:xxxx 是自己电脑的路径，后面的 pcie_test 及其后面的路径是固定的。

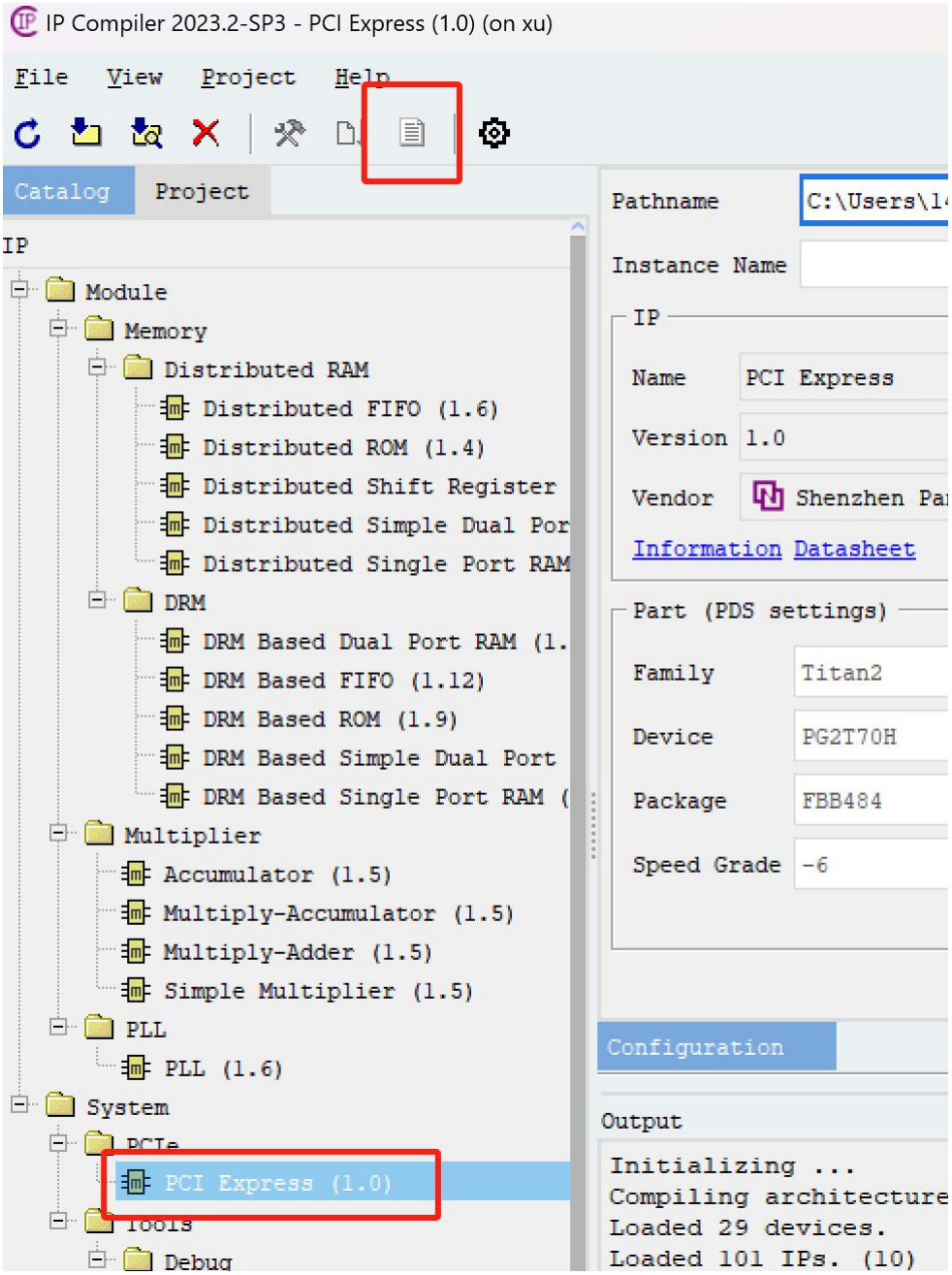


按照开发板管脚，修改相关管脚约束：

	I/O NAME	I/O DIRECTION	LOC	BANK	VCCIO	IOSTANDARD
9	pclk_led	OUTPUT	U16	BANKL6	3.3	LVCMOS33
10	rdlh_link_up	OUTPUT	T16	BANKL6	3.3	LVCMOS33
11	ref_led	OUTPUT	Y16	BANKL6	3.3	LVCMOS33
12	smlh_link_up	OUTPUT	R16	BANKL6	3.3	LVCMOS33
13	txd	OUTPUT	E21	BANKL5	3.3	LVCMOS33
14	button_rst_n	INPUT	K21	BANKL5	3.3	LVCMOS33
15	perst_n	INPUT	A14	BANKL4	3.3	LVCMOS33
16	ref_clk_n	INPUT	D5	NOBANK		
17	ref_clk_p	INPUT	D6	NOBANK		
18	rxd	INPUT	E22	BANKL5	3.3	LVCMOS33

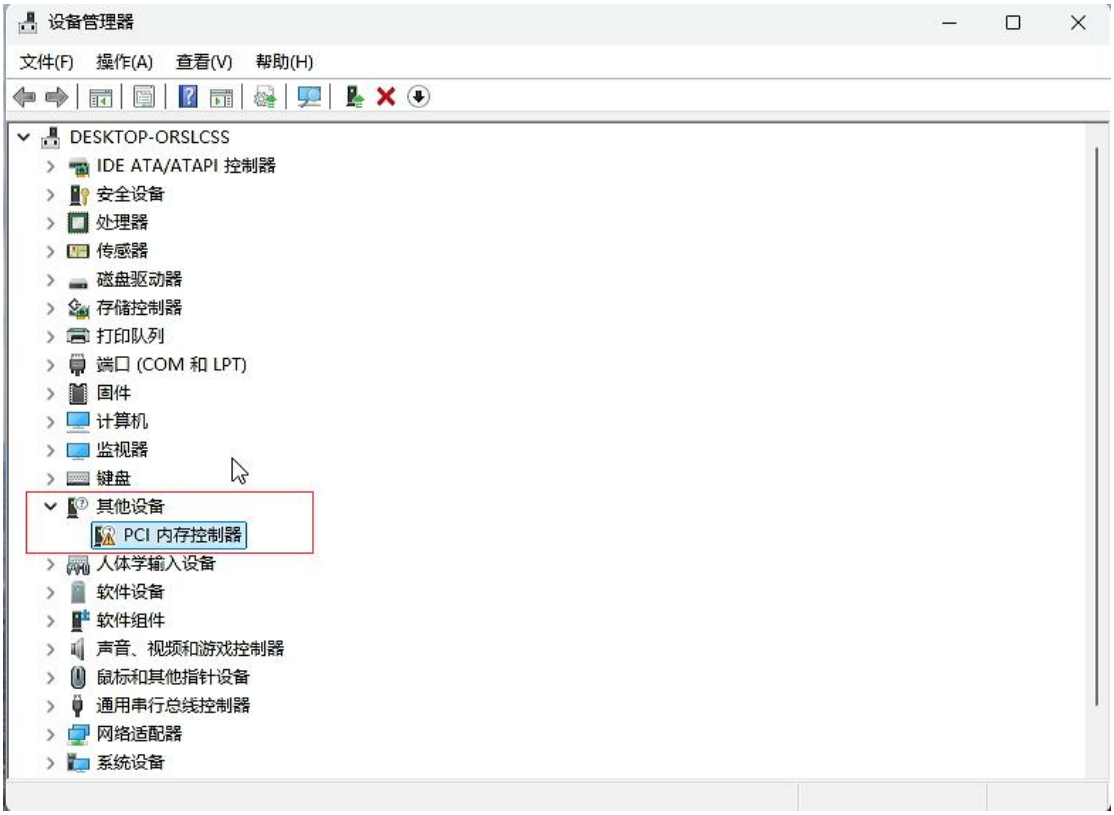
注意，txd 和 rxd 是串口。txp[0]，txp[1]，rxp[0]，rxp[1]等差分信号需要在.fdc 文件进行约束，具体约束请参考《UG050008_Titan2 系列 FPGA 高速串行收发器（HSSTHP）用户指南_V1.4》。

可按以下方式查看 IP 核的用户指南，了解 Example 模块组成；



10.4 实验现象

将程序固化到 flash 内，把开发板插入电脑 PCIE 卡槽，开机。打开设备管理器，可识别到 PCIE 设备。



Win 下能弹出该设备即可。