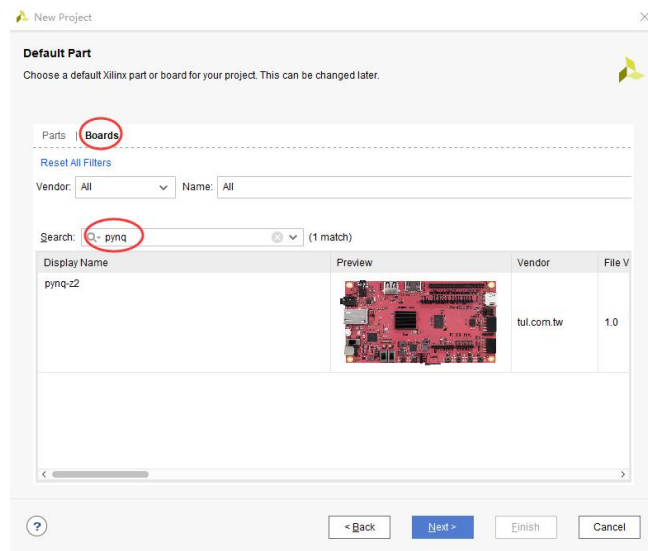


1. 添加板卡文件。由于本次实验采用的是 PYNQ Z2 板子，Vivado 中默认库中是没有此板子的型号的，所以第一次打开时，需要先添加板卡文件。（只要没重置电脑，后续就不需要再添加了）

操作：首先找到 Vivado 的安装目录，将 pynq-z2 文件夹放入到“PATH\data\boards\board_parts\zynq/”，如图

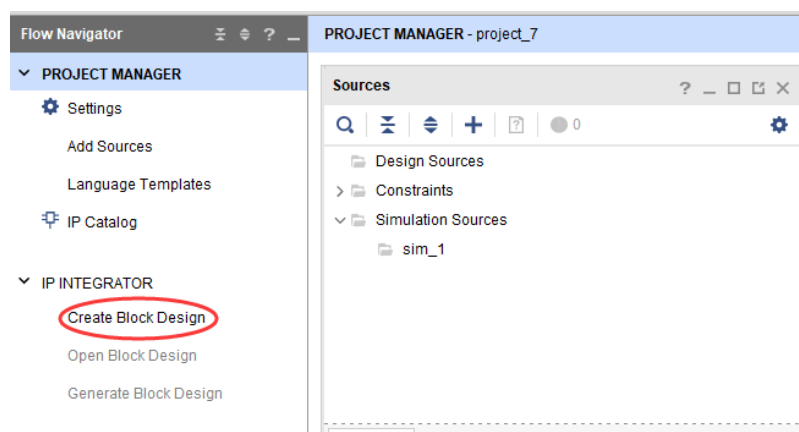


2. 打开 Vivado，依次选择 Create Project > Next，修改项目名称和项目位置，点击 Next，选择 RTL Project，并打勾 Do not specify sources at this time>Next,点击上面的 Boards，如图



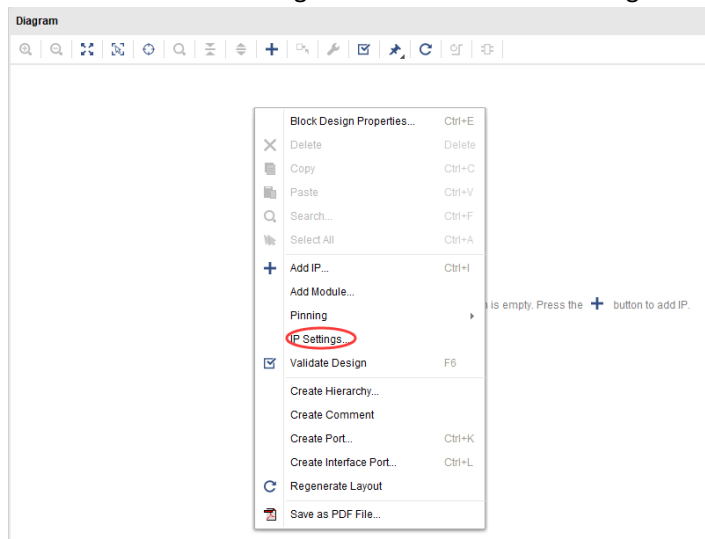
点击 pynq-z2>Next>Finish。

3. 点击左侧 IP INTERATOR/Create Block Design

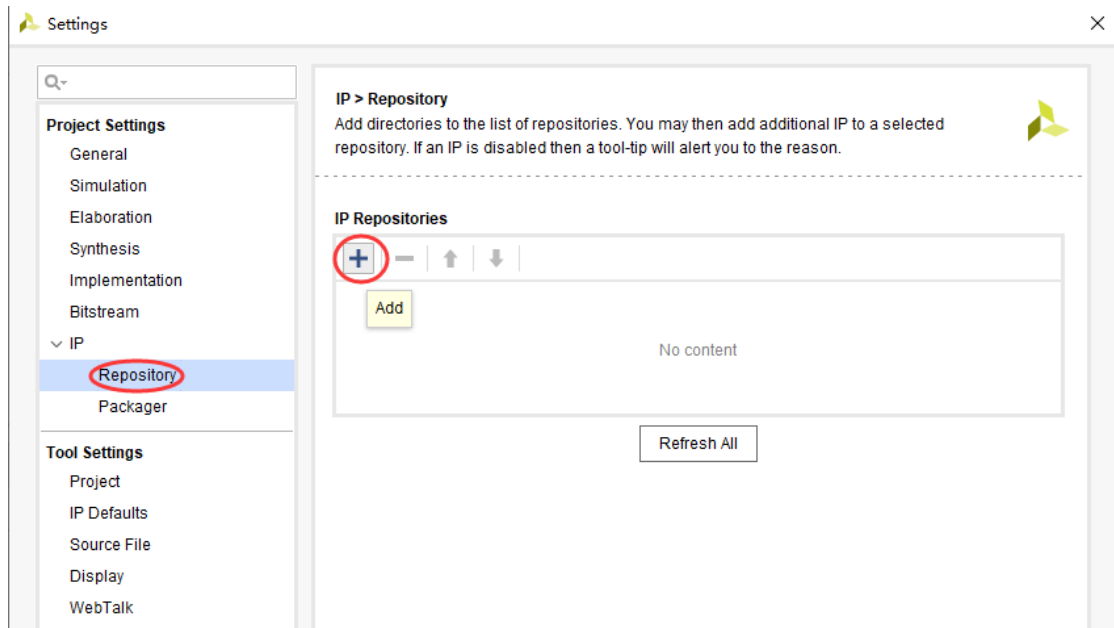


修改模块名称，点击 OK。

4. 添加 IP 核。右键 Diagram 空白处，点击 IP setting。

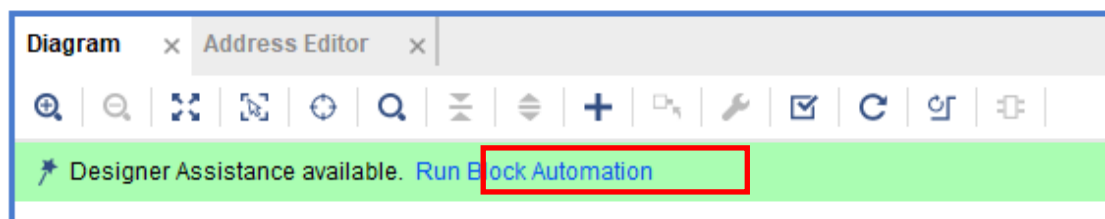


5. 点击 IP/Repository/Add

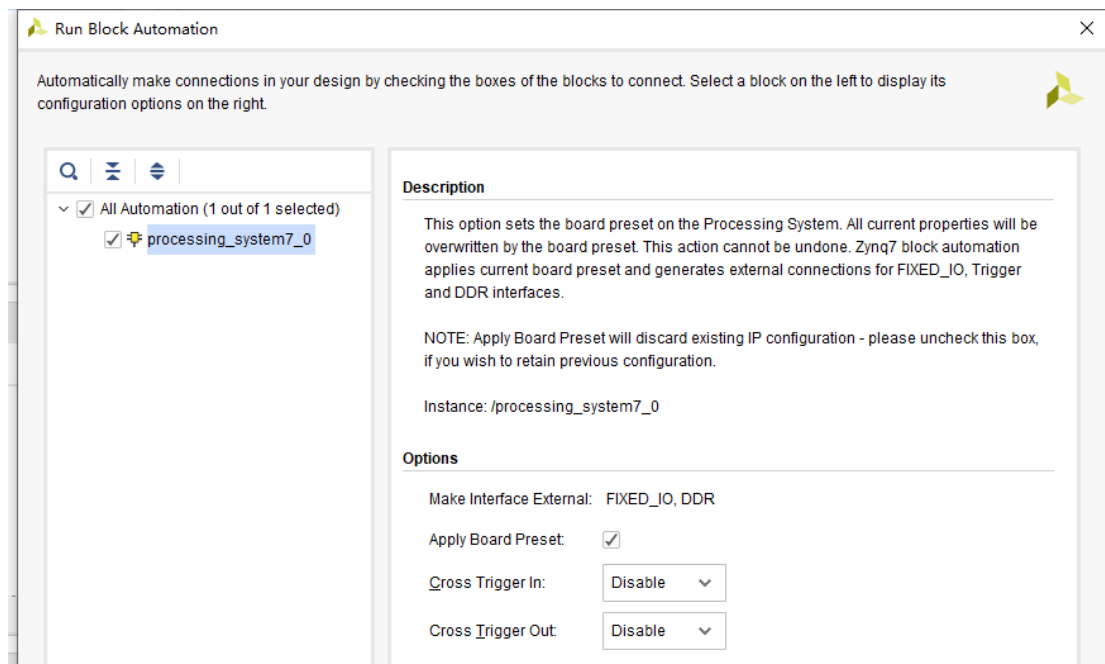


选中 PATH/ vivado-library-master（其中 PATH 为 vivado-library-master 的存储路径），点击 Select> OK>Apply>OK

6. 右键 Diagram 空白处，Add IP, 搜索 zynq7, 双击 ZYNQ7 Processing System 添加 IP。
点击上方的 Run Block Automation

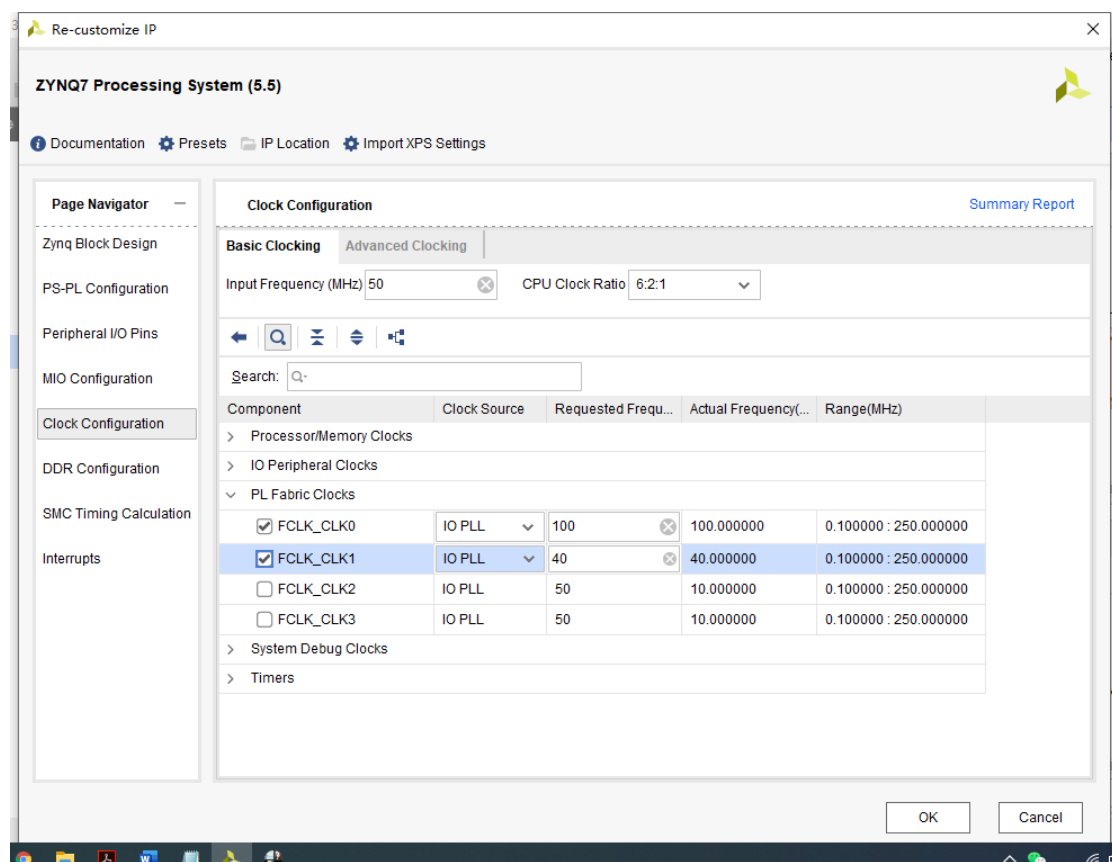


保证 Apply Board Preset 勾选，并点击 OK 确认



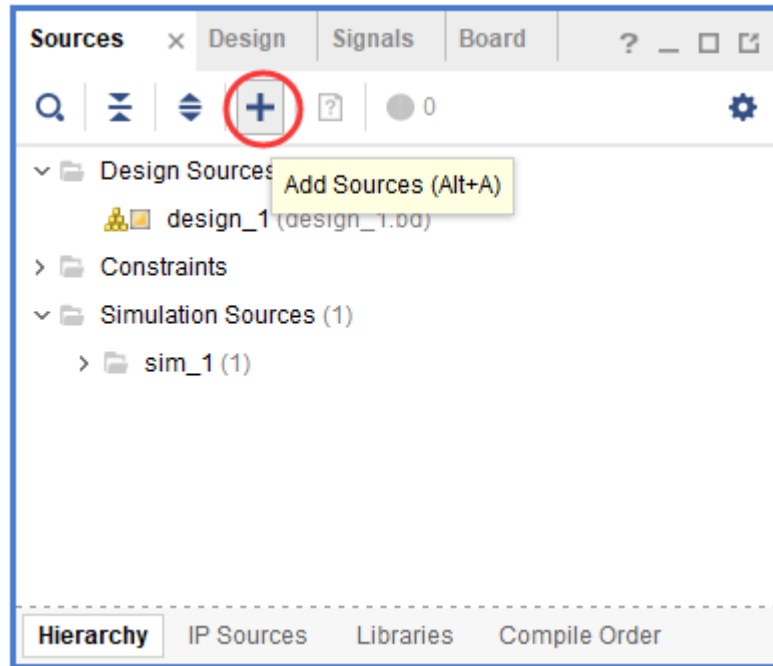
7. 双击添加的 ZYNQ7 Processing System 模块进行配置

7.1 点击左侧 Clock Configuration>PL Fabric Clocks, 勾选 FCLK_CLK1, 将其 Request Frequency 设置为 40MHz

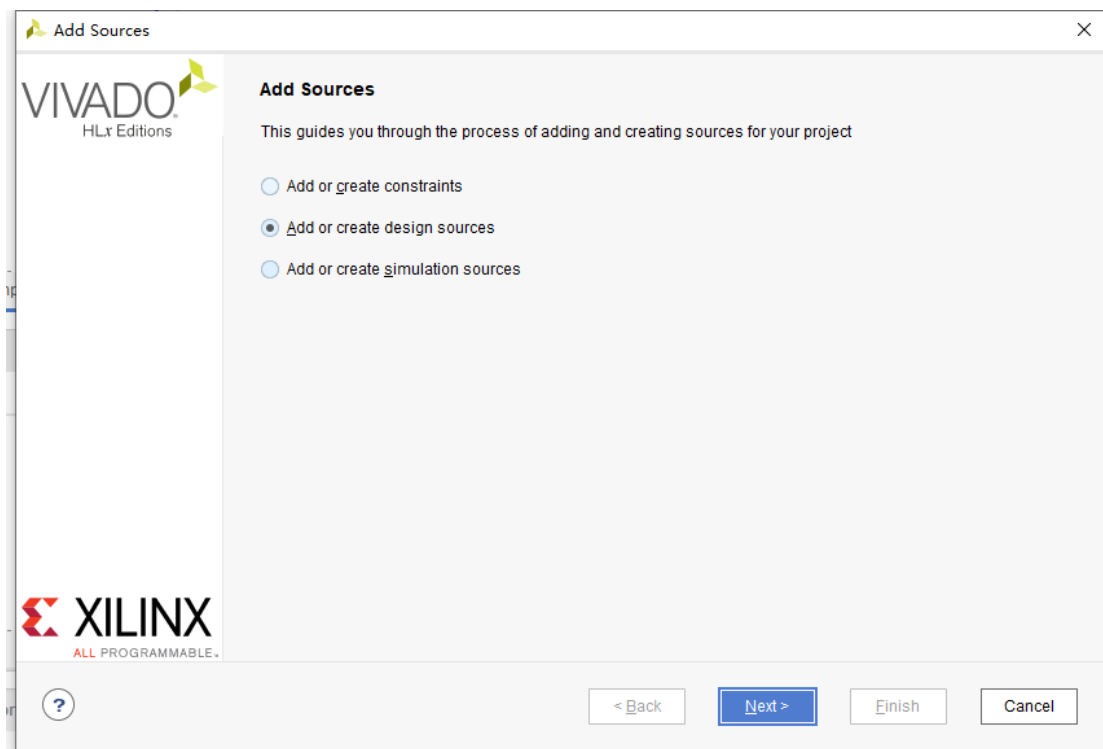


7.2 点击 OK 完成配置。

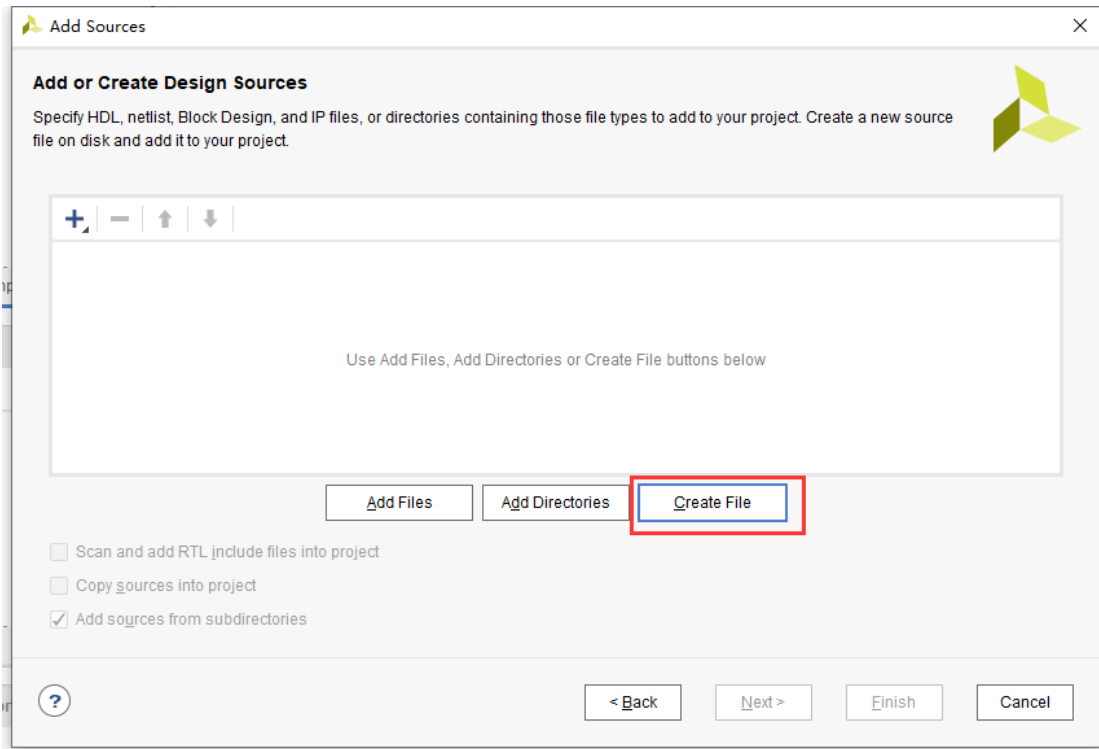
8. 左上角面板选择 Sources 选项卡, 点击 Add Sources



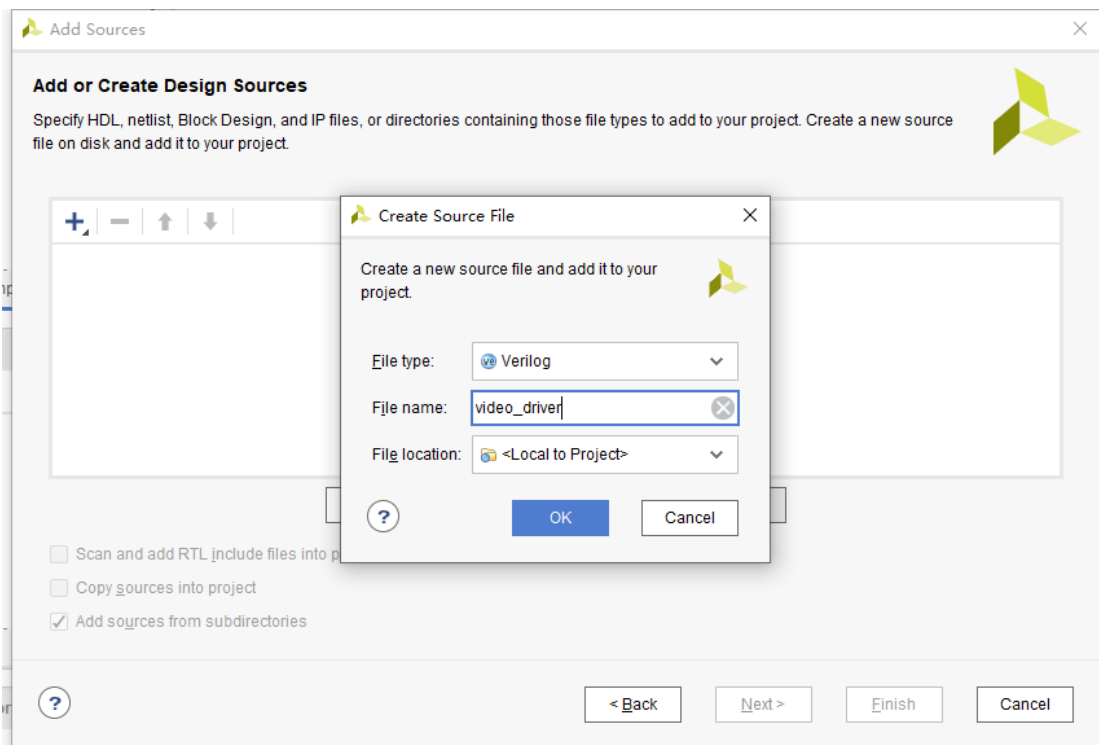
选择 Add or create design sources，点击 Next



点击 Create File



创建 verilog 文件，输入文件名，如“video_driver”



Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: video_driver

I/O Port Definitions

+ - ↑ ↓

Port Name	Direction	Bus	MSB	LSB
	input	<input type="checkbox"/>	0	0

?

OK

Cancel

Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Module name: v

I/O Port Definition

+ - ↑ ↓

Port Name	Di
	input

?

The module definition has not been changed.
Are you sure you want to use these values?

Yes

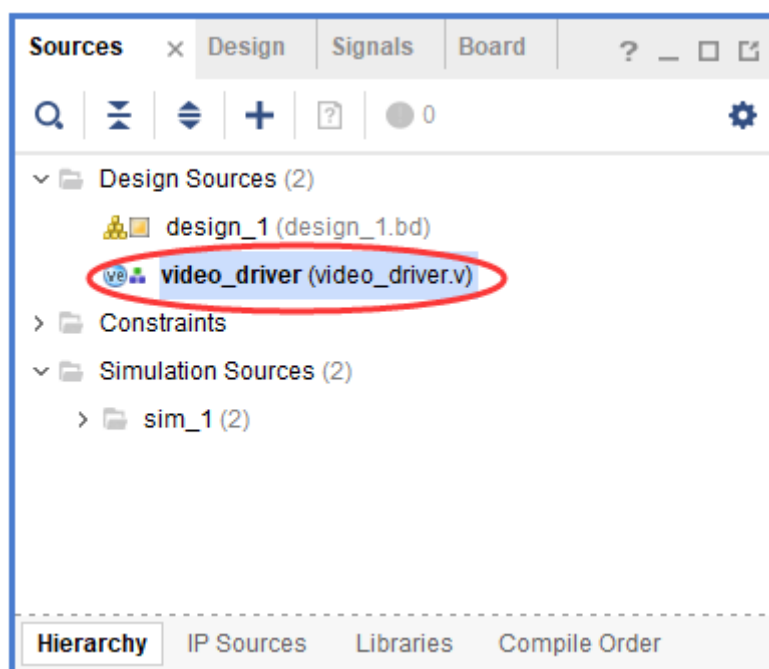
No

?

OK

Cancel

9. 在左上角面板双击刚刚创建的 verilog 文件



将以下代码复制进去：

```
module video_driver(
    input          pixel_clk,
    input          sys_rst_n,

    //RGB 接口
    output          video_hs,    //行同步信号
    output          video_vs,    //场同步信号
    output          video_de,    //数据使能
    output [23:0]   video_rgb,   //RGB888 颜色数据

    input  [23:0]   pixel_data,  //像素点数据
    output [10:0]   pixel_xpos,  //像素点横坐标
    output [10:0]   pixel_ypos   //像素点纵坐标
);

//parameter define

//1280*720 分辨率时序参数
//parameter H_SYNC    = 11'd40;  //行同步
//parameter H_BACK    = 11'd220; //行显示后沿
//parameter H_DISP    = 11'd1280; //行有效数据
//parameter H_FRONT   = 11'd110; //行显示前沿
//parameter H_TOTAL   = 11'd1650; //行扫描周期
```

```

//parameter V_SYNC = 11'd5; //场同步
//parameter V_BACK = 11'd20; //场显示后沿
//parameter V_DISP = 11'd720; //场有效数据
//parameter V_FRONT = 11'd5; //场显示前沿
//parameter V_TOTAL = 11'd750; //场扫描周期

//800*600 分辨率时序参数
parameter H_DISP = 11'd800;
parameter H_FRONT = 11'd40;
parameter H_SYNC = 11'd128;
parameter H_BACK = 11'd88;
parameter H_TOTAL = 11'd1056;

parameter V_DISP = 11'd600;
parameter V_FRONT = 11'd1;
parameter V_SYNC = 11'd4;
parameter V_BACK = 11'd23;
parameter V_TOTAL = 11'd628;

//reg define
reg [10:0] cnt_h;
reg [10:0] cnt_v;

//wire define
wire video_en;
wire data_req;

//*****
//** main code
//*****

assign video_de = video_en;

assign video_hs = ( cnt_h < H_SYNC ) ? 1'b0 : 1'b1; //行同步信号赋值
assign video_vs = ( cnt_v < V_SYNC ) ? 1'b0 : 1'b1; //场同步信号赋值

//使能 RGB 数据输出
assign video_en = (((cnt_h >= H_SYNC+H_BACK) && (cnt_h < H_SYNC+H_BACK+H_DISP))
&&((cnt_v >= V_SYNC+V_BACK) && (cnt_v < V_SYNC+V_BACK+V_DISP)))
? 1'b1 : 1'b0;

//RGB888 数据输出
assign video_rgb = video_en ? pixel_data : 24'd0;

```



```

//请求像素点颜色数据输入
assign data_req = (((cnt_h >= H_SYNC+H_BACK-1'b1) &&
                    (cnt_h < H_SYNC+H_BACK+H_DISP-1'b1))
                    && ((cnt_v >= V_SYNC+V_BACK) && (cnt_v < V_SYNC+V_BACK+V_DISP)))
                    ? 1'b1 : 1'b0;

//像素点坐标
assign pixel_xpos = data_req ? (cnt_h - (H_SYNC + H_BACK - 1'b1)) : 11'd0;
assign pixel_ypos = data_req ? (cnt_v - (V_SYNC + V_BACK - 1'b1)) : 11'd0;

//行计数器对像素时钟计数
always @(posedge pixel_clk ) begin
    if (!sys_rst_n)
        cnt_h <= 11'd0;
    else begin
        if(cnt_h < H_TOTAL - 1'b1)
            cnt_h <= cnt_h + 1'b1;
        else
            cnt_h <= 11'd0;
    end
end

//场计数器对行计数
always @(posedge pixel_clk ) begin
    if (!sys_rst_n)
        cnt_v <= 11'd0;
    else if(cnt_h == H_TOTAL - 1'b1) begin
        if(cnt_v < V_TOTAL - 1'b1)
            cnt_v <= cnt_v + 1'b1;
        else
            cnt_v <= 11'd0;
    end
end

endmodule

```

10. 用相同的步骤，创建 video_display.v

```

module video_display(
    input                pixel_clk,
    input                sys_rst_n,

    input                [10:0] pixel_xpos, //像素点横坐标

```

```

        input      [10:0] pixel_ypos, //像素点纵坐标
        output reg [23:0] pixel_data, //像素点数据
        output reg [15:0] rom_addr, //地址
        output     rom_rd_en, //ROM 读使能信号
        input [7:0] R, //R 数据
        input [7:0] G,
        input [7:0] B
    );

    //parameter define
    //parameter H_DISP = 11'd1280; //分辨率——行
    //parameter V_DISP = 11'd720; //分辨率——列

    //parameter define
    localparam PIC_X_START = 11'd100; //图片起始点横坐标
    localparam PIC_Y_START = 11'd100; //图片起始点纵坐标
    localparam PIC_WIDTH = 11'd240; //图片宽度
    localparam PIC_HEIGHT = 11'd240; //图片高度

    localparam BACK_COLOR = 24'hE0FFFF; //背景色，浅蓝色

    //reg define

    //wire define

    /*******
    /**
    main code
    /*******
    assign rom_rd_en = 1'b1; //读使能拉高，即一直读 ROM 数据

    //为 HDMI 不同显示区域绘制图片和背景色
    always @(posedge pixel_clk) begin
        if (!sys_rst_n)
            pixel_data <= BACK_COLOR;
        else if( (pixel_xpos >= PIC_X_START) && (pixel_xpos < PIC_X_START + PIC_WIDTH)
            && (pixel_ypos >= PIC_Y_START) && (pixel_ypos < PIC_Y_START + PIC_HEIGHT))
            pixel_data <= {R,G,B}; //显示图片
        else
            pixel_data <= BACK_COLOR; //屏幕背景色
    end

```

```

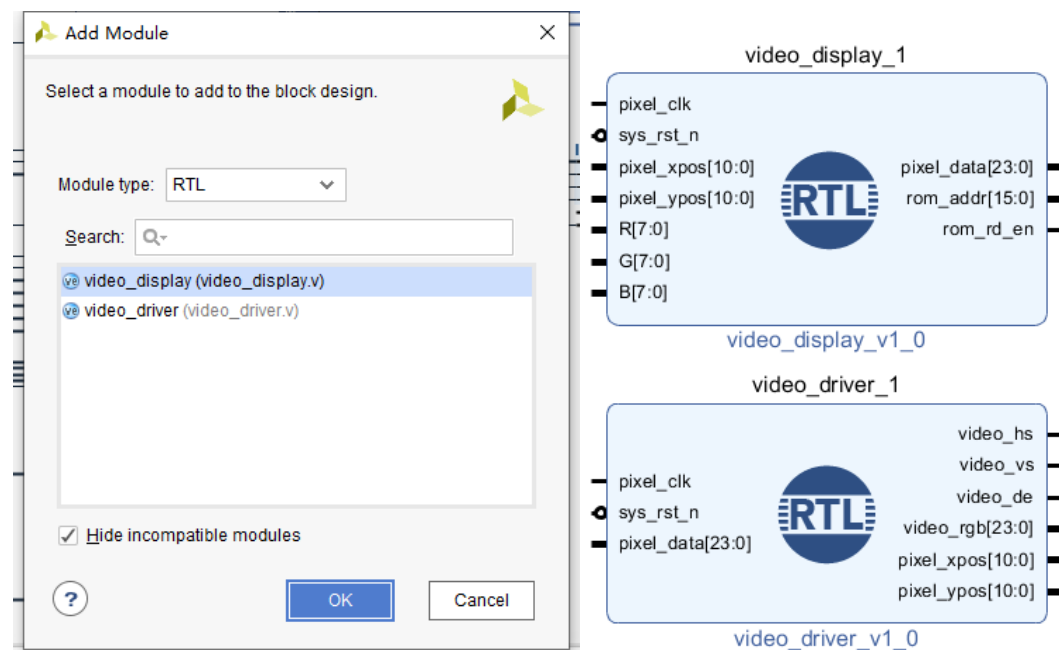
//根据当前扫描点的横纵坐标为 ROM 地址赋值
always @(posedge pixel_clk) begin
    if(!sys_rst_n)
        rom_addr <= 16'd0;
    //当横纵坐标位于图片显示区域时,累加 ROM 地址
    else if((pixel_ypos >= PIC_Y_START) && (pixel_ypos < PIC_Y_START + PIC_HEIGHT)
        && (pixel_xpos >= PIC_X_START) && (pixel_xpos < PIC_X_START + PIC_WIDTH))
        rom_addr <= rom_addr + 1'b1;
    //当横纵坐标位于图片区域最后一个像素点时,ROM 地址清零
    else if((pixel_ypos >= PIC_Y_START + PIC_HEIGHT))
        rom_addr <= 16'd0;
end

//ROM: 存储图片
//blk_mem_gen_0 blk_mem_gen_0 (
// .clka (pixel_clk), // input wire clka
// .ena (rom_rd_en), // input wire ena
// .addra (rom_addr), // input wire [13 : 0] addra
// .douta (rom_rd_data) // output wire [23 : 0] douta
//);

```

Endmodule

11. 添加基于 video_display.v 和 video_driver.v 的模块
 右键 Diagram 的空白处，选择 Add Module，分别选择 video_display.v 与 video_driver.v，生成相应的模块，如右侧

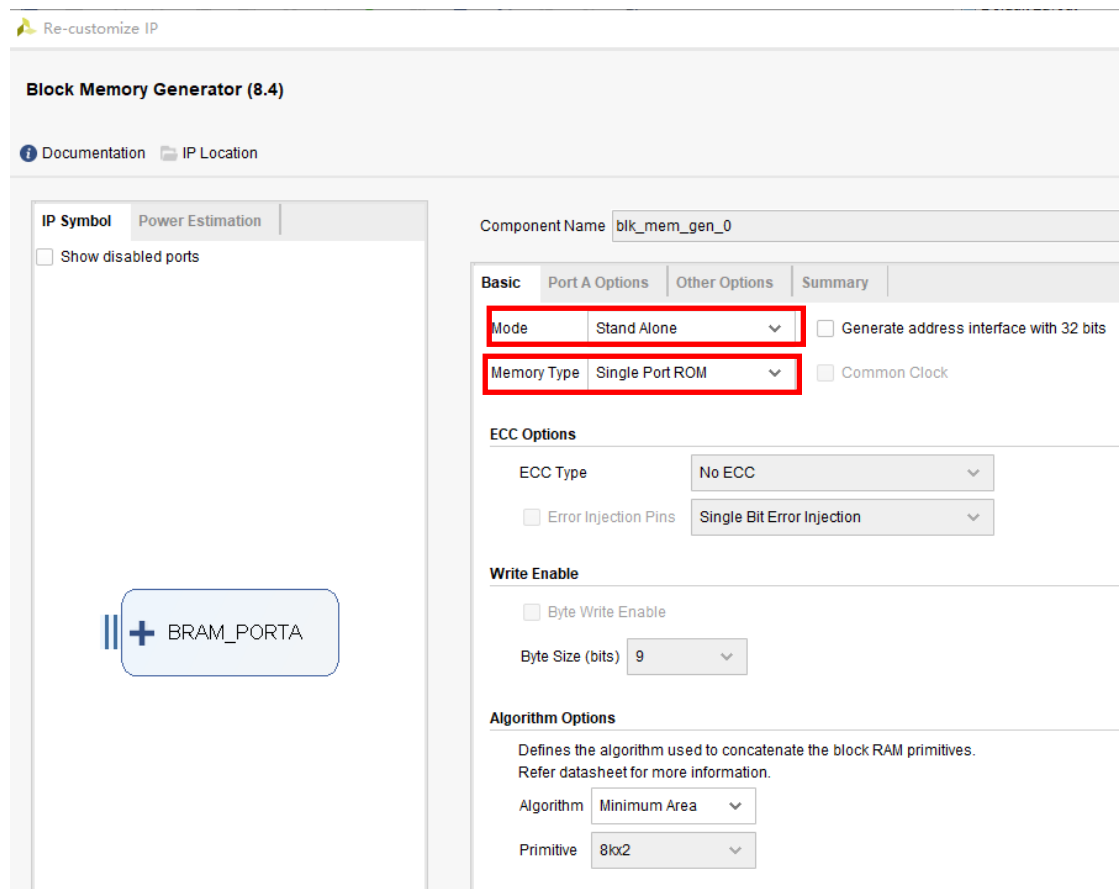


12. 利用 image2coe2.m 获得 coe 文件
 13. 添加 3 个 ROM 存储图片的 R\G\B 值。
 右键 Diagram 的空白处，选择 Add IP，搜索 Block Memory General，双击选择 Block Memory

General。

14. 双击刚刚添加的 Block Memory General 进行配置。

14.1 在 Basic 栏目下，Mode 选择 Stand Alone，Memory Type 选择 Single Port ROM



Re-customize IP

Block Memory Generator (8.4)

Documentation IP Location

IP Symbol Power Estimation

☐ Show disabled ports

Component Name blk_mem_gen_0

Basic Port A Options Other Options Summary

Mode Stand Alone ☐ Generate address interface with 32 bits

Memory Type Single Port ROM ☐ Common Clock

ECC Options

ECC Type No ECC

☐ Error Injection Pins Single Bit Error Injection

Write Enable

☐ Byte Write Enable

Byte Size (bits) 9

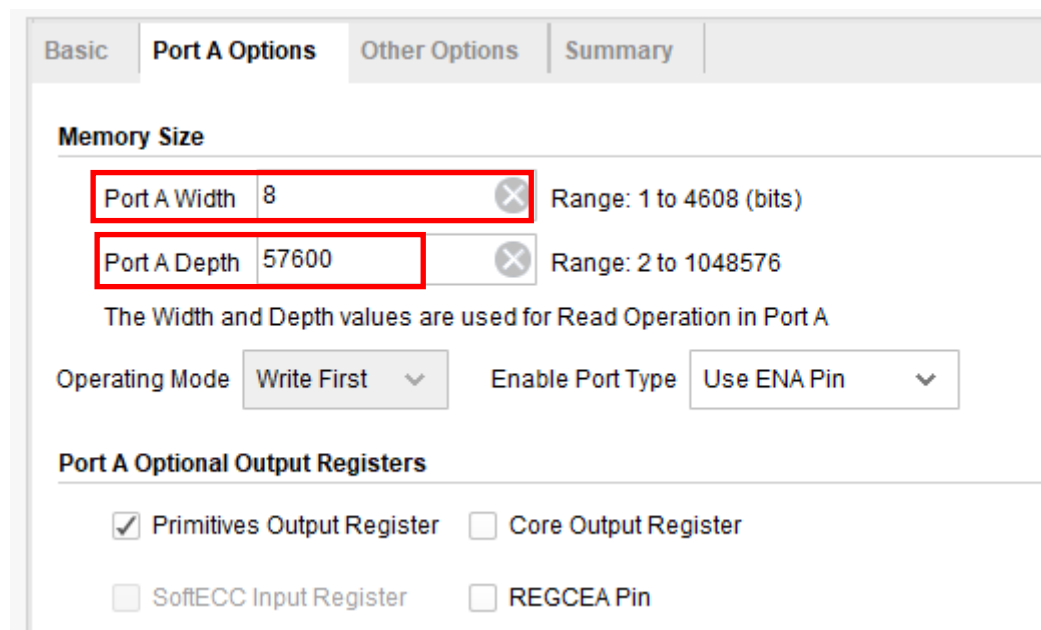
Algorithm Options

Defines the algorithm used to concatenate the block RAM primitives. Refer datasheet for more information.

Algorithm Minimum Area

Primitive 8x2

14.2 在 Port A Options 栏目下，Port A width 设置为 8，Port A Depth 设置为图片大小（宽*高）



Basic Port A Options Other Options Summary

Memory Size

Port A Width 8 Range: 1 to 4608 (bits)

Port A Depth 57600 Range: 2 to 1048576

The Width and Depth values are used for Read Operation in Port A

Operating Mode Write First

Enable Port Type Use ENA Pin

Port A Optional Output Registers

☒ Primitives Output Register ☐ Core Output Register


☐ SoftECC Input Register ☐ REGCEA Pin

14.3 在 Other Options 栏目下，勾选 Load Init File，选择 coe 文件

Memory Initialization

☒ Load Init File

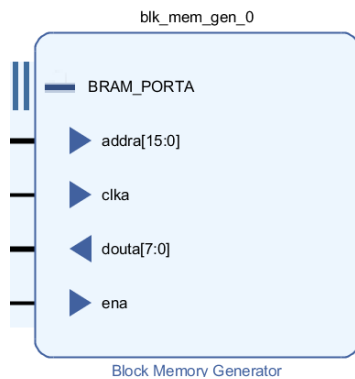
Coe File

 Browse

 Edit

14.4 如此生成 3 个 ROM，分别将之前生成的 R\G\B 对应的 3 个 coe 文件导入。

15. 展开 Block Memory Generator 的 BRAM_PORTA



15.1 将输入端子 `addr[15:0]` 与 `veido_display` 的输出端子 `rom_addr[15:0]` 连接【R/G/B 对应的 3 个 ROM 相同】

15.2 将输入端子 `clka` 与 ZYNQ7 Processing System 的输出端子 `FCLK_CLK1` 连接【R/G/B 对应的 3 个 ROM 相同】

15.3 将输入端子 `ena` 与 `veido_display` 的输出端子 `rom_rd_en` 连接【R/G/B 对应的 3 个 ROM 相同】

15.4 将输出端子 `douta[7:0]` 与 `veido_display` 的端子 R/G/B 相应连接。

16. 将 ZYNQ7 Processing System 的 `M_AXI_GPO_ACLK` 与 `FCLK_CLK0` 连接

17. 对 ZYNQ7 Processing System 的 DDR 与 `FIXED_IO` 生成外部管脚

【方法同之前：点击名称，按住 `Ctrl+T` 生成管脚（注意是仅引脚变橙色才正确，如果是整个模块都变橙色说明你选中的是整个模块，需要重选，如果太小了可以 `Ctrl+鼠标滚轮` 控制缩放）。】

18. 将 `veido_display` 的 `pixel_clk` 与 ZYNQ7 Processing System 的 `FCLK_CLK1` 连接

19. 将 `veido_display` 的 `sys_rst_n` 与 ZYNQ7 Processing System 的 `FCLK_RESET0_N` 连接

20. 将 `video_driver` 的 `pixel_clk` 与 ZYNQ7 Processing System 的 `FCLK_CLK1` 连接

21. 将 `video_driver` 的 `sys_rst_n` 与 ZYNQ7 Processing System 的 `FCLK_RESET0_N` 连接

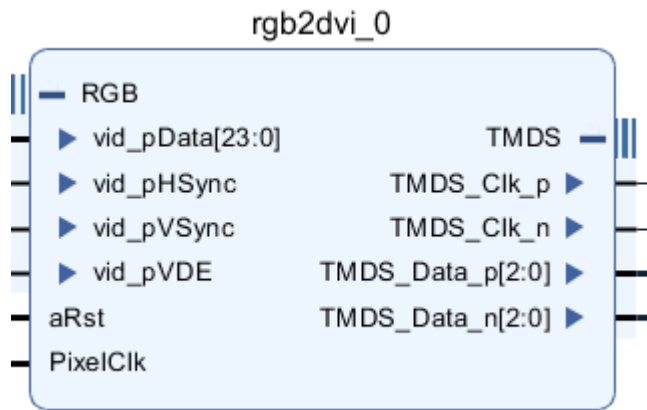
22. 将 `veido_display` 的 `pixel_xpos[10:0]`、`pixel_ypos[10:0]` 与 `video_driver` 的 `pixel_xpos[10:0]`、`pixel_ypos[10:0]` 相应连接

23. 将 `veido_display` 的 `pixel_data[23:0]` 与 `video_driver` 的 `pixel_data[23:0]` 连接

24. 右键 Diagram 的空白处，选择 Add IP，搜索 RGB to DVI Video Encoder，双击添加 RGB to DVI Video Encoder。

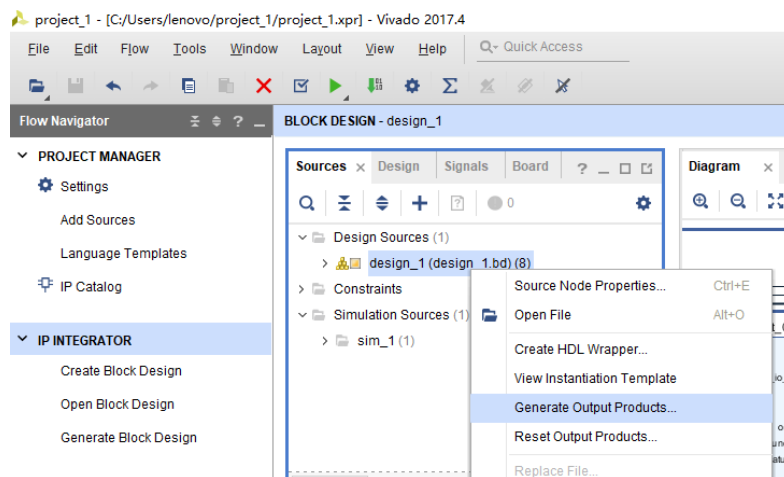
25. 将 RGB to DVI Video Encoder 的 `PixelClk` 与 ZYNQ7 Processing System 的 `FCLK_CLK1` 连接

26. 展开 RGB to DVI Video Encoder 的 RGB

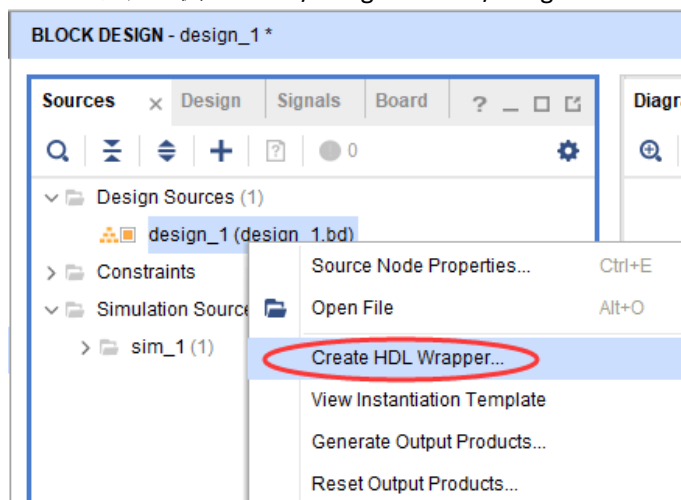


将 vid_pData[23:0]、vid_pHSync、vid_pVSync、vid_pVDE
与 video_driver 的 video_rgb[23:0]、video_hs、video_vs、video_de 连接

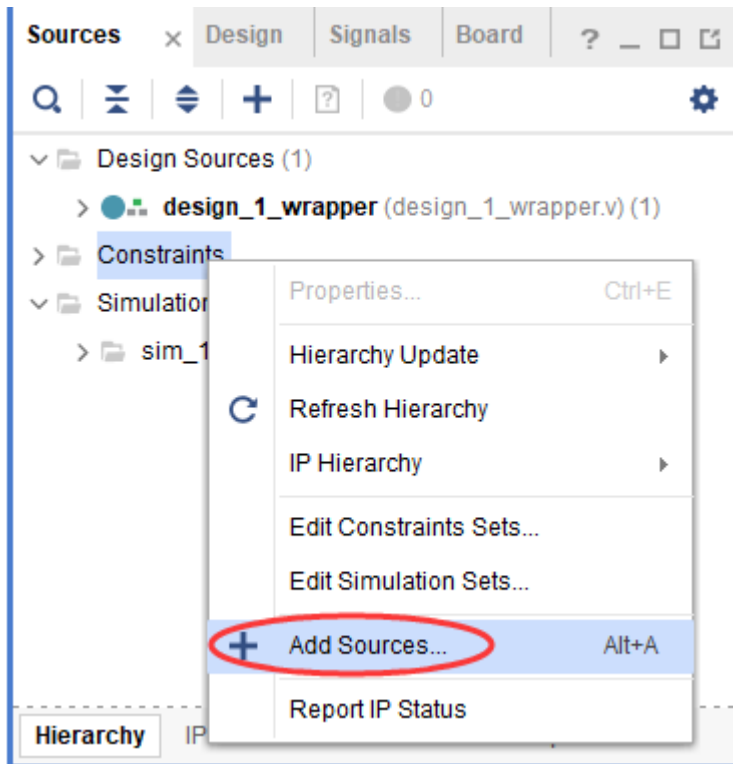
27. 展开 RGB to DVI Video Encoder 的 TMDS
生成 TMDS_Clk_p、TMDS_Clk_n、TMDS_Data_p[2:0]、TMDS_Data_n[2:0]的外部管脚。
28. 点击上方验证按钮，若出现成功，则完成验证
29. 右键左侧的 Sources>Design Sources>design_1，选择 Generate Output Products



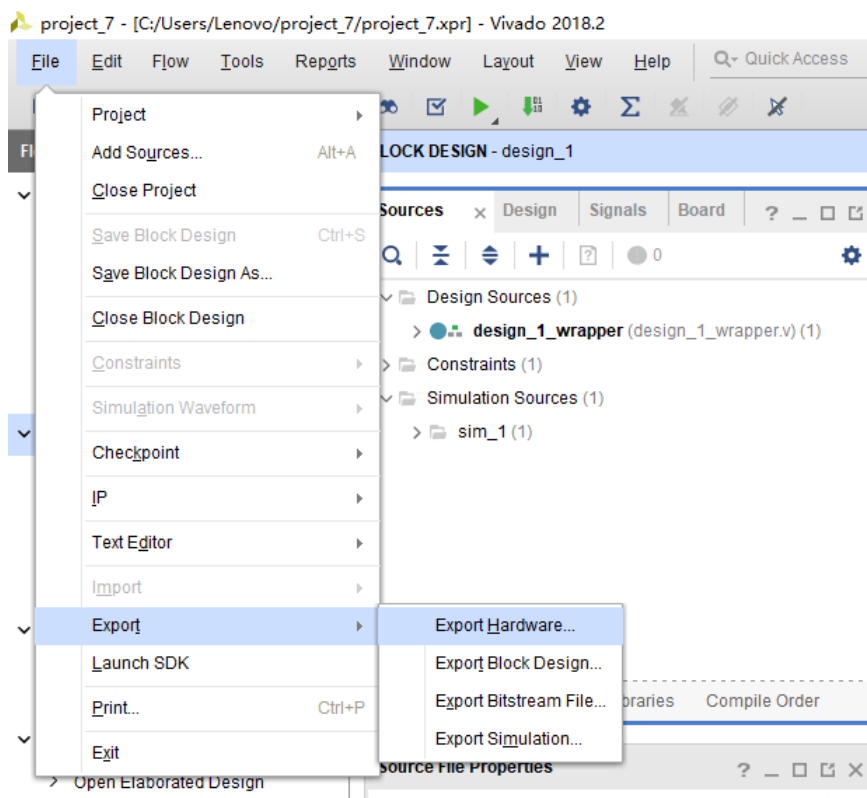
30. 右键左侧 Sources/Design Sources/design>Create HDL Wrapper



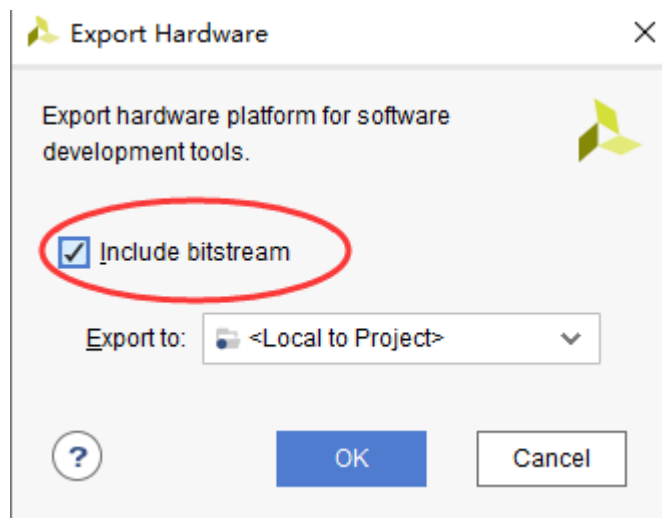
31. 右键 Sources/Constraints>Add Sources，添加约束文件。



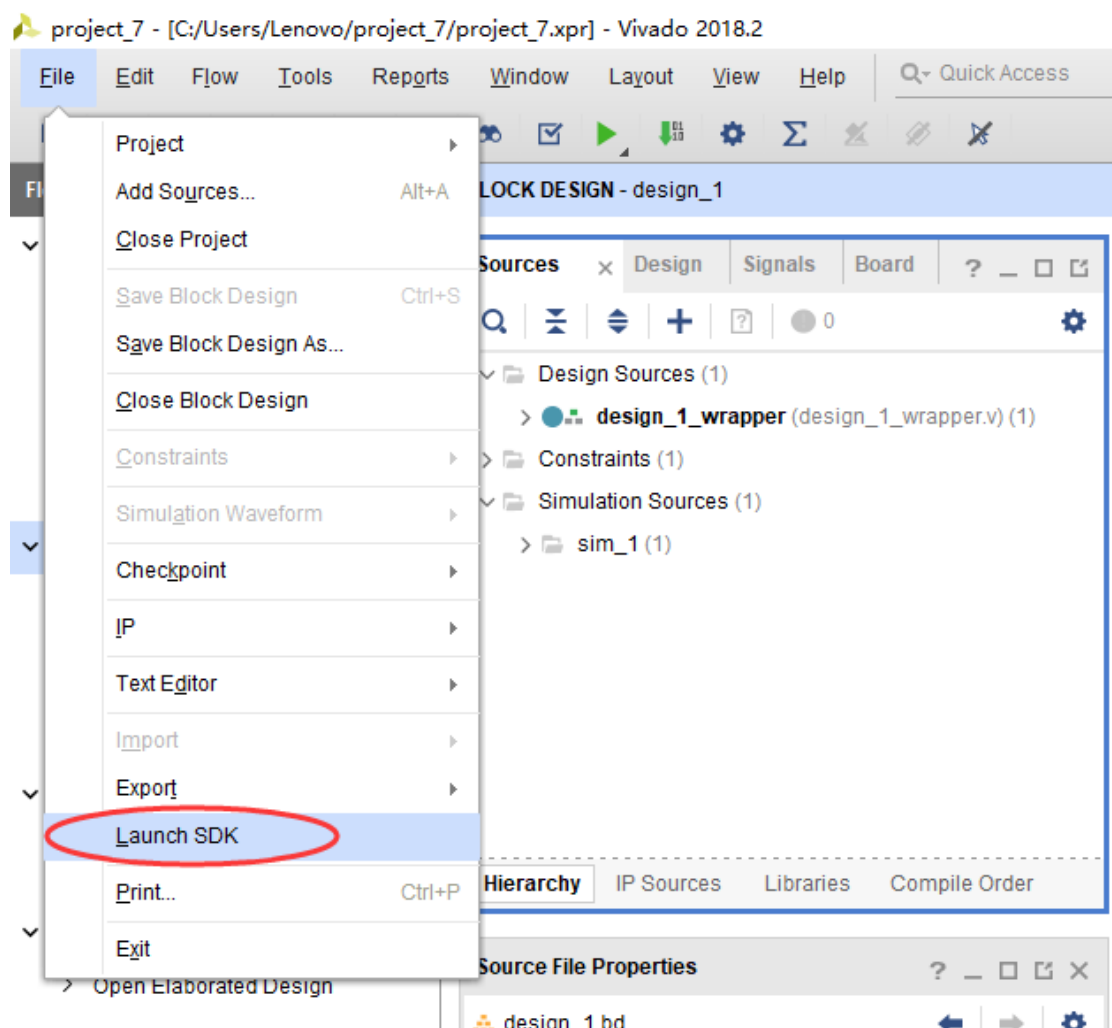
32. 选择 Add or create constraints, 点击 Next, 点击 Add Files, 选择 pynq-z2_v1.0.xdc (之前彩条实验使用的), 点击 OK, 点击 Finish, 完成添加约束文件。
33. 点击左侧的 PROGRAM AND DEBUG/Generate Bitstream, 点击 OK>OK, 进行生成比特流。
(这一步耗费的时间比较长, 可以看右上角完全 ready 了, 再进行下一步)
34. 跳出生成比特流成功的对话框后, 点击 Cancel。点击左上角的 File/Export/Export Hardware



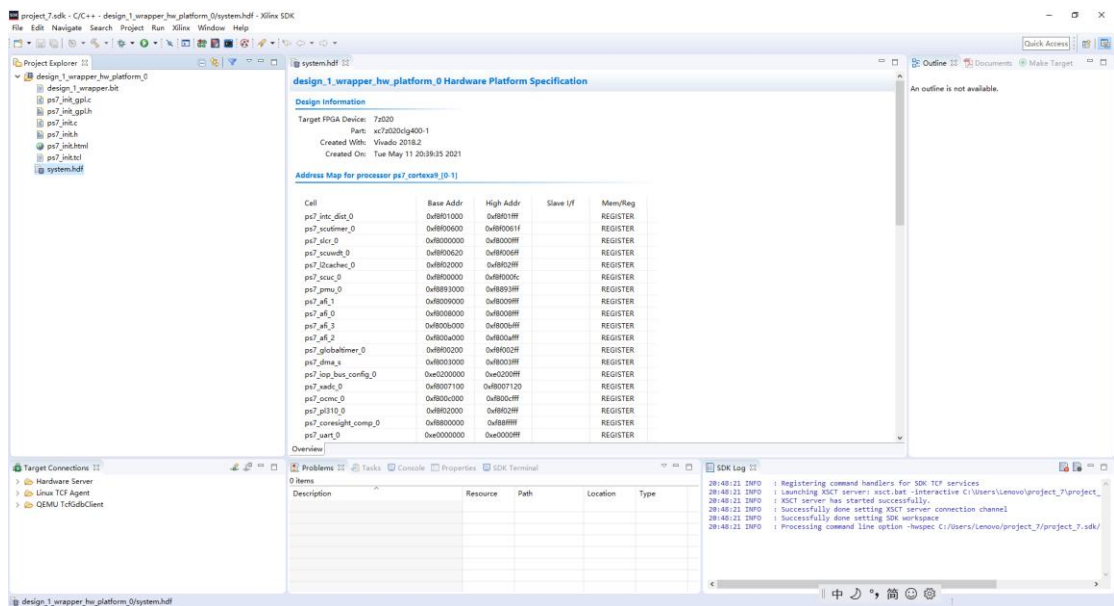
35. 勾选 Include bitstream，点击 OK



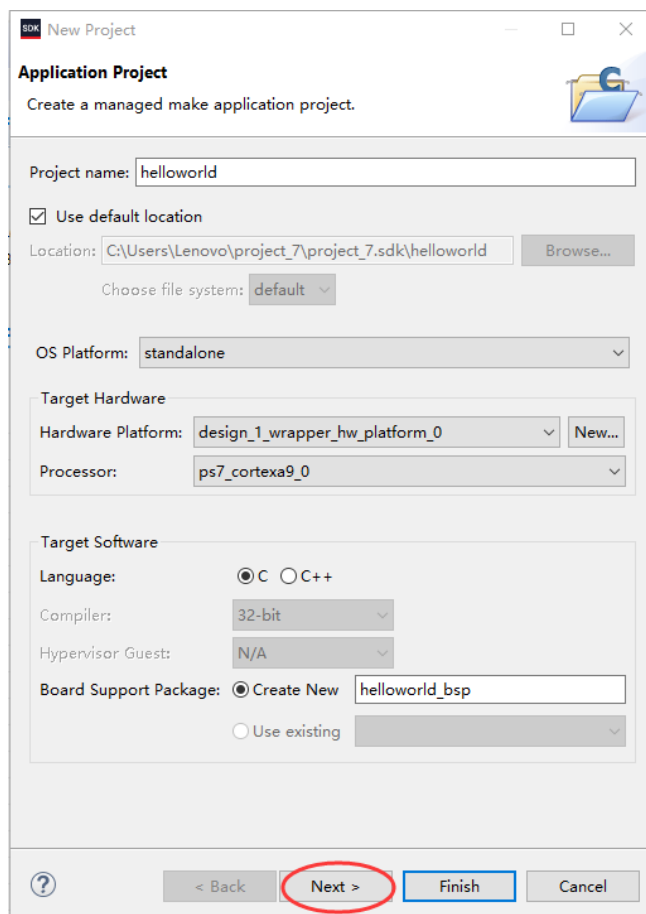
36. 点击 File/Launch SDK，对话框选择 OK。（这一步需要启动另一个软件 SDK，需要一点时间）



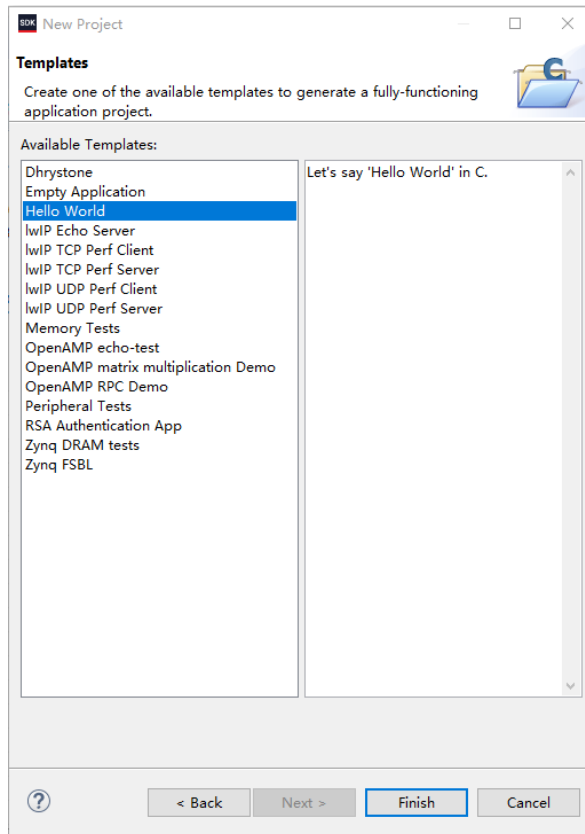
37. 成功打开 SDK 软件界面如图



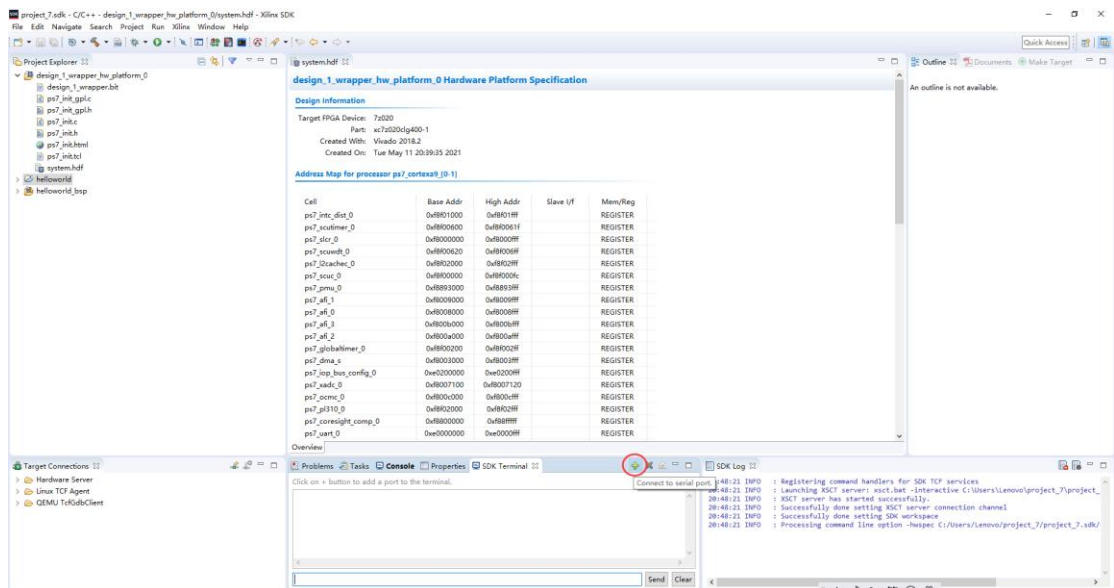
38. 点击左上 File/New/Application Project，对话框中 Project name 写入 Hello world，点击 Next。



39. 选择 Hello World，点击 Finish。



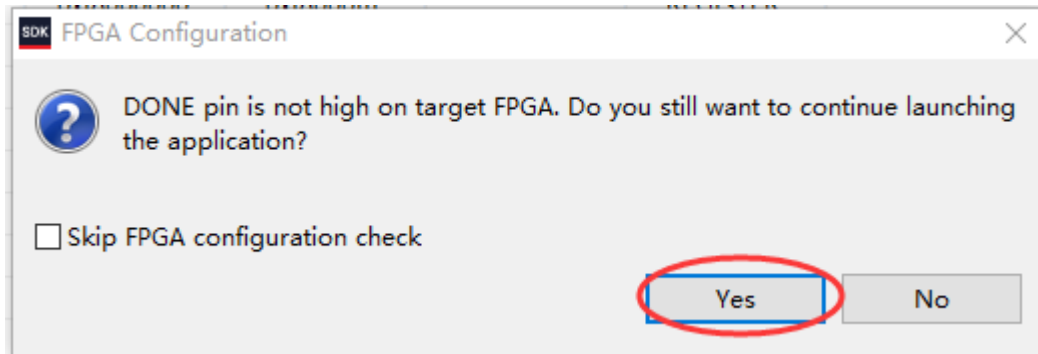
40. 返回 SDK 软件中，点击下方的 ‘+’ 按钮



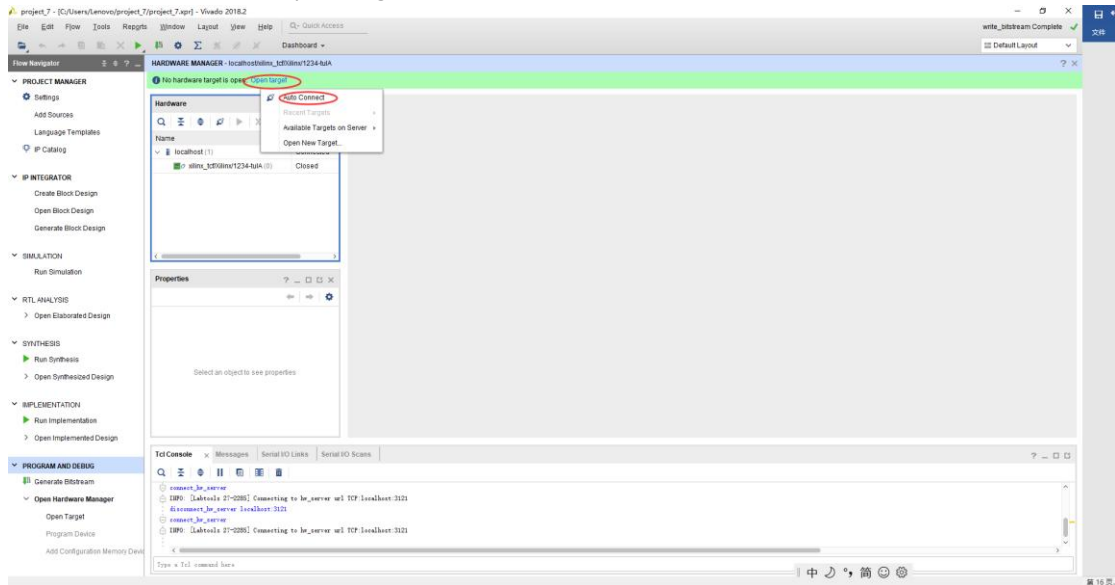
41. 对话框中 Port 选择为设备管理器中的 USB 串行接口端口，点击 OK。

42. 右键左侧的刚刚你建立的 helloworld 工程文件夹，Run As/Launch on Hardware (System Debugger)

43. 跳出的对话框选 Yes



44. 返回 Vivado 中，点击 Open target



45. 点击 Program device，点击 OK

