APRIL 2023, 4 BY ACKYE

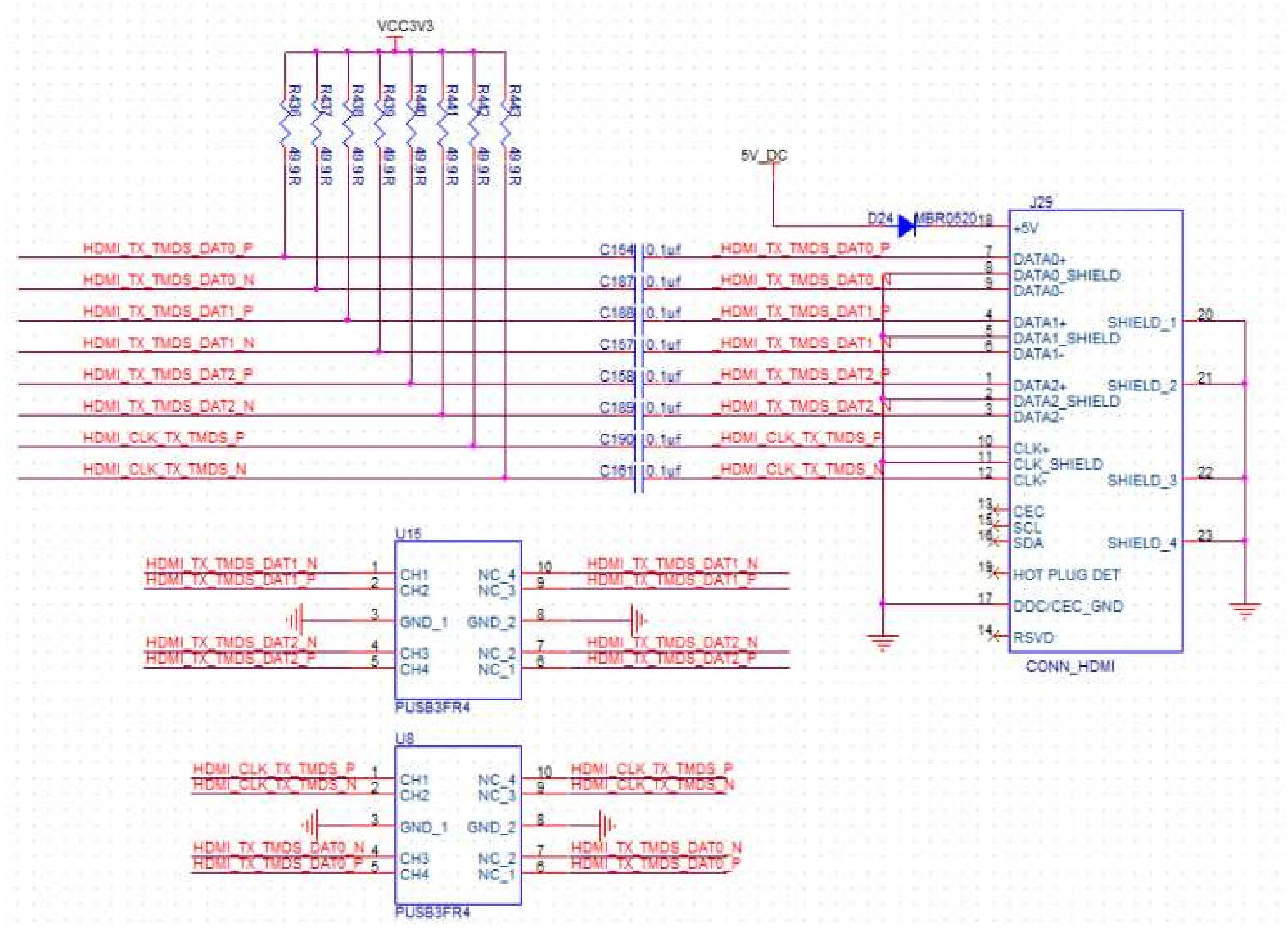
Smart ZYNQ (SP&SL Edition) Project V HDMI function demonstration based on ZYNQ PL resources

This article demonstrates how to use PL logic resources to drive the HDMI part of the circuit on the smallest system board to realize the operation of the point screen

This article is demonstrated on vivado2018.3, please research for other versions

(Note: The content of this section applies to the boards of Smart ZYNQ SP and SL Edition, if it is Smart ZYNQ Standard Edition, please refer to the corresponding board directory)

About the hardware section:



`

The HDMI_TX_TMDS_DATA0-1-2 and HDMI_CLK_TX_TMDS signals are connected to ZYNQ's PL differential signal line.

In addition, the HDMI of the board is connected to the ESD chip (PUSB3FR4), so that the board will not be damaged due to static electricity during the process of plugging and unplugging HDMI

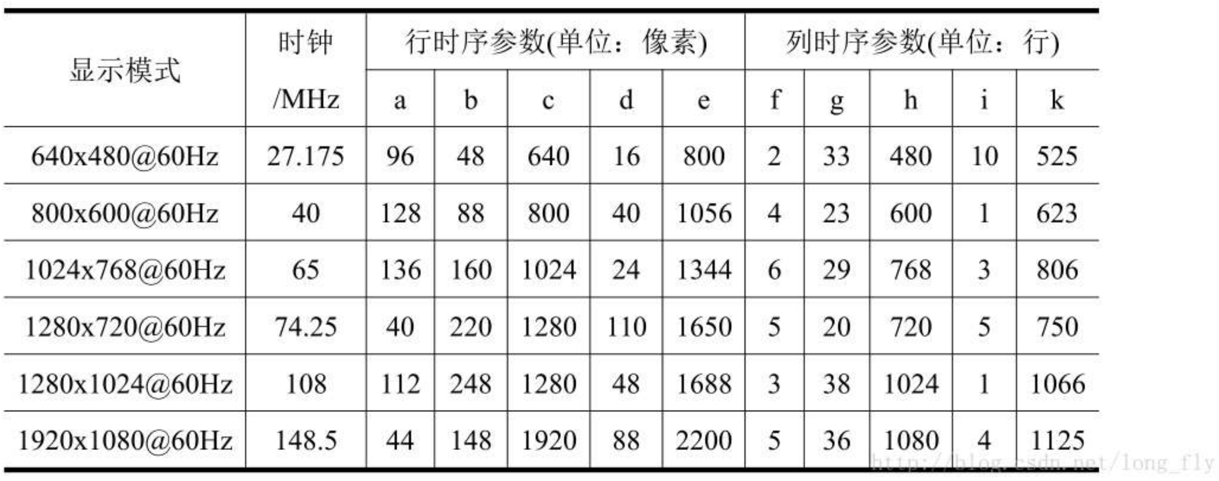
To drive the HDMI module, we also need to use an IP core RGB2DVI officially designed by digilent, and the official download address of this IP core is as follows: https://github.com/Digilent/vivado-library

Here, for the convenience of use, I also put this IP core down on this site for your reference (IP copyright belongs to digilent)

http://www.hellofpga.com/wp-content/uploads/2021/07/rgb2dvi.zip

The input timing of the RGB2DVI meets the VGA standard, and the following are the parameters for VGA resolution

VGA 常用分辨率时序参数



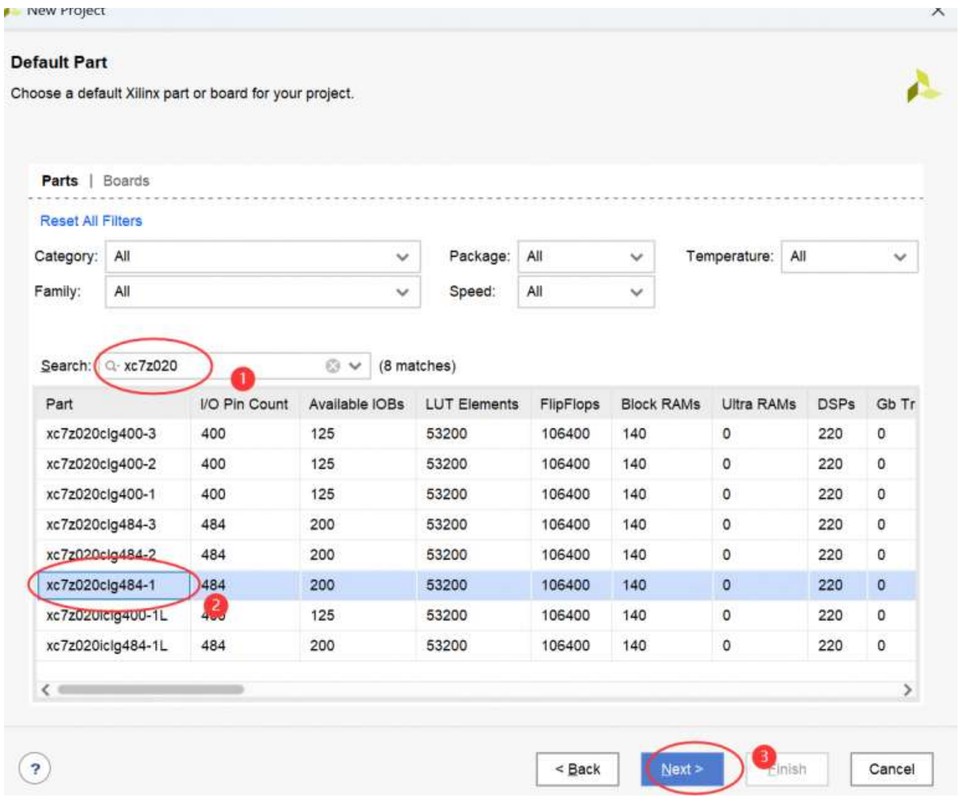
显示模式	时钟	行时序参数(单位：像素)					列时序参数(单位：行)				
	/MHz	a	b	c	d	e	f	g	h	i	k
640x480@60Hz	27.175	96	48	640	16	800	2	33	480	10	525
800x600@60Hz	40	128	88	800	40	1056	4	23	600	1	623
1024x768@60Hz	65	136	160	1024	24	1344	6	29	768	3	806
1280x720@60Hz	74.25	40	220	1280	110	1650	5	20	720	5	750
1280x1024@60Hz	108	112	248	1280	48	1688	3	38	1024	1	1066
1920x1080@60Hz	148.5	44	148	1920	88	2200	5	36	1080	4	1125

Timing diagrams, as well as the concept of blanking, as well as parameters for each region at different resolutions

Module Design:

1. Project Creation:

1) Create a new project, select **XC7Z020CLG484-1** for the chip model


New Project

Default Part

Choose a default Xilinx part or board for your project.

Parts | Boards

Reset All Filters

Category: All   Package: All   Temperature: All

Family: All   Speed: All

Search: xc7z020   (8 matches)

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Tr
xc7z020clg400-3	400	125	53200	106400	140	0	220	0
xc7z020clg400-2	400	125	53200	106400	140	0	220	0
xc7z020clg400-1	400	125	53200	106400	140	0	220	0
xc7z020clg484-3	484	200	53200	106400	140	0	220	0
xc7z020clg484-2	484	200	53200	106400	140	0	220	0
xc7z020clg484-1	484	200	53200	106400	140	0	220	0
xc7z020iclg400-1L	400	125	53200	106400	140	0	220	0
xc7z020iclg484-1L	484	200	53200	106400	140	0	220	0

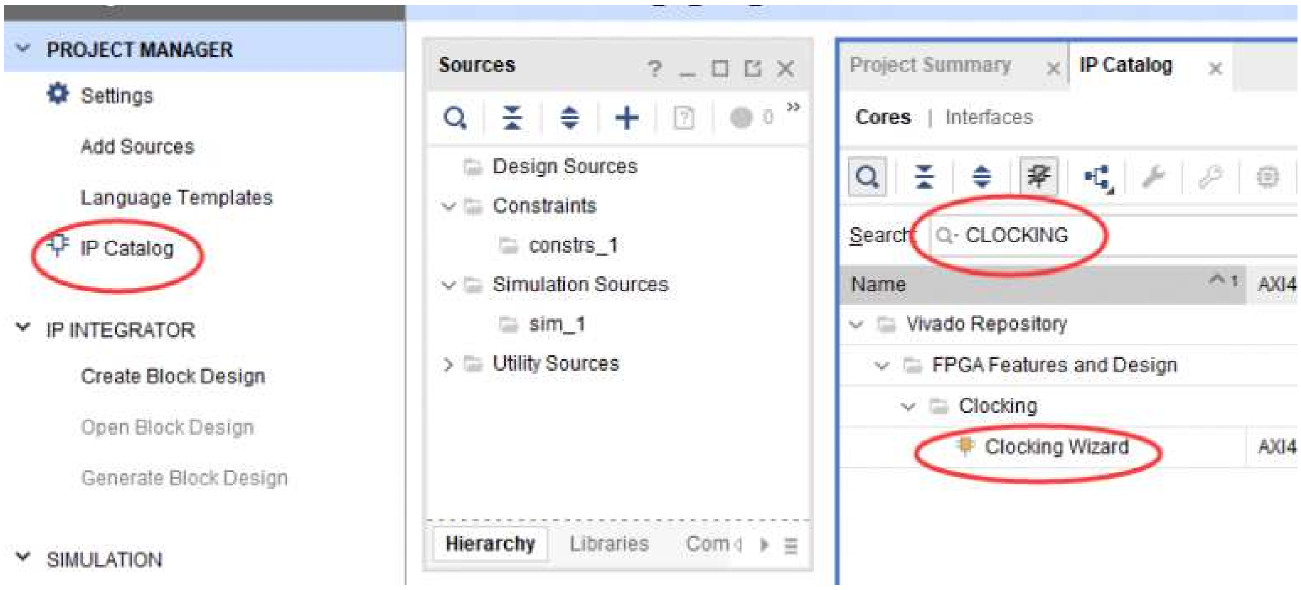
工程创建详细图文教程请参考 Smart ZYNQ（SP&SL 版) 工程一 用ZYNQ的PL资源点亮一个LED（完整图文）

2.时钟模块设计

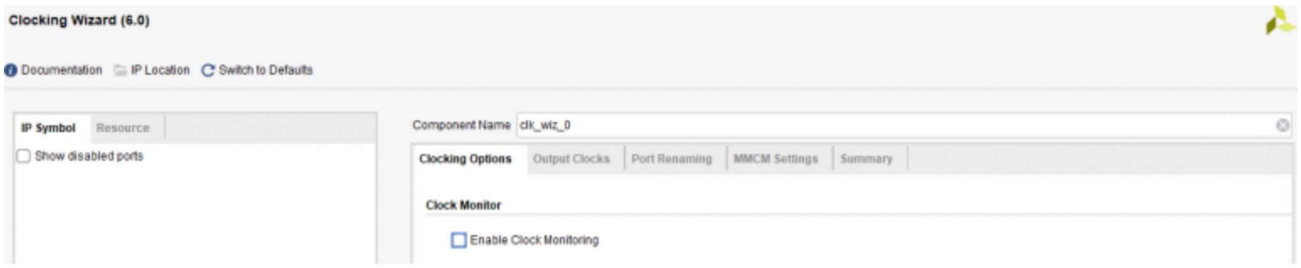
我们尝试点亮800X600的屏幕分辨率，如VGA的分辨率参数表中标注的，800X600分辨率需要的工作的像素时钟为40MH，而我们板子上焊接的有源晶体是50MHZ，这里就需要用时钟管理模块MMCM来生成我们要的40MHZ频率和 5倍像素时钟的200Mhz频率 **（也可以通过always块来实现5倍关系的时钟，具体看本文最下方增加的部分内容，这种方式适用的分辨率更广）**

VIVADO系统给我们内置了很多功能强大的模块，包括我们要的时钟模块，这里我们只需要调用时钟模块输入我们要的参数就好

1)第一步，点击IP Catalog 打开模块选择器， 在里面的搜索栏输入 CLOCKING ，系统会自动跳出符合的 Clocking Wizard选项，双击它

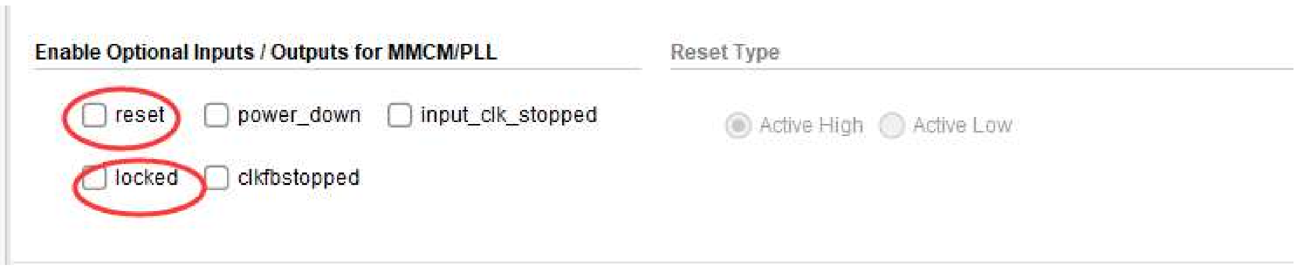


2）在弹出的窗口中我们将input Frequence 输入频率修改为板子上焊接的50M时钟， 右边改为单端输入



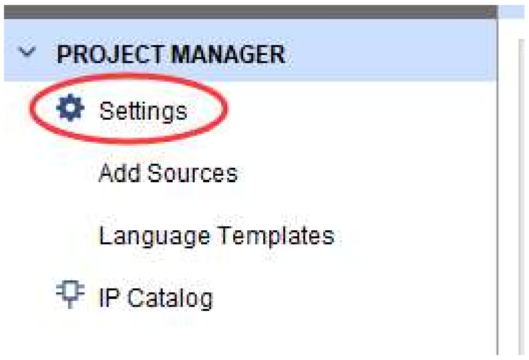
3）在output Clocks选项中 将clk_out1改成40m，将clk_out2改成200m

4）将界面托到最下面，因为我们这里的要求并不高，所以把时钟的复位reset，和locked选项去除，最后点击ok生成模块

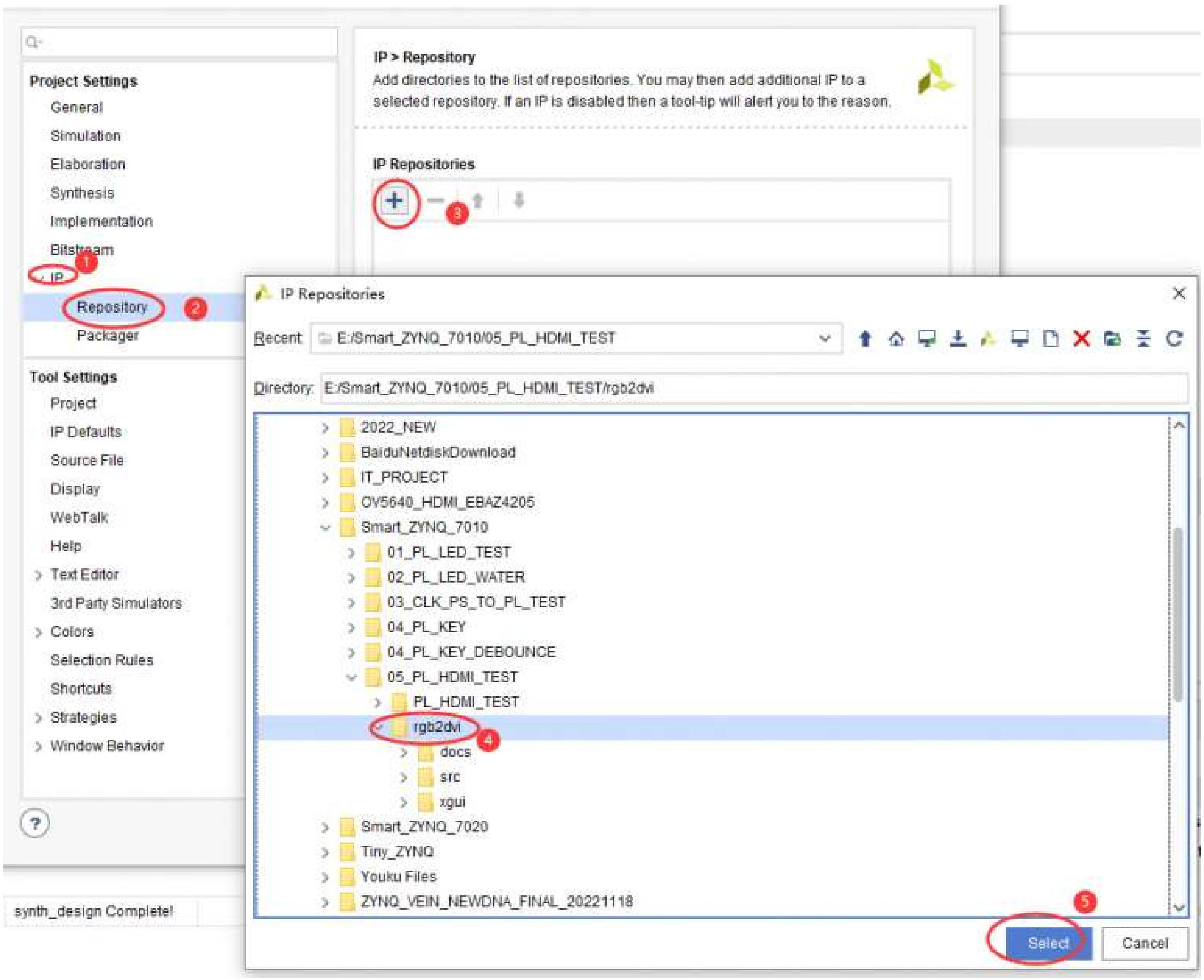


3.导入下下来的DVI2RGB模块

1）点击Setting



2）在弹出的设置窗口，如下图，展开IP选项，选中Packager，在点击里面的加号增加目录，选中RGB2DVI的目录**(这里已提前将RGB2DVI的目录复制到工程目录下)**，点击ok



IP > Repository

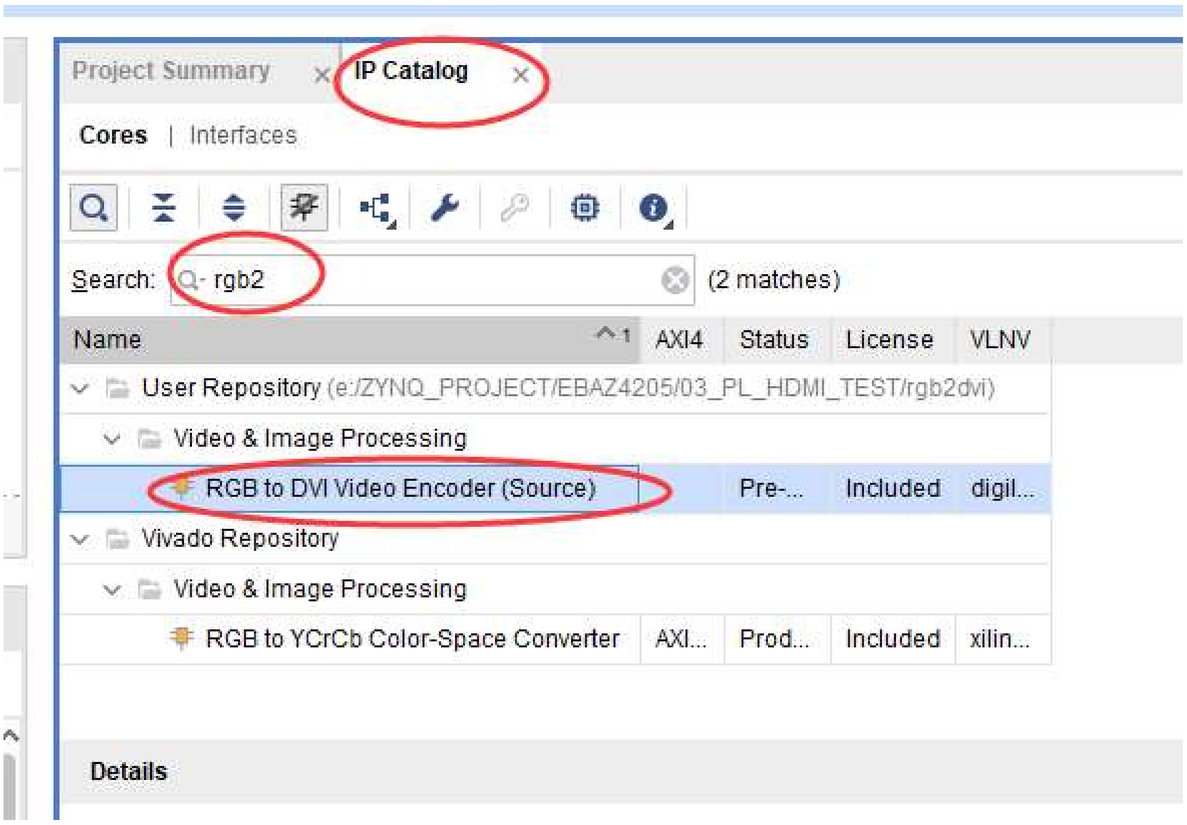
Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

Recent: E:/Smart_ZYNQ_7010/05_PL_HDMI_TEST

Directory: E:/Smart_ZYNQ_7010/05_PL_HDMI_TEST/rgb2dvi

```
2022_NEW
BaiduNetdiskDownload
IT_PROJECT
OV5640_HDMI_EBAZ4205
Smart_ZYNQ_7010
  01_PL_LED_TEST
  02_PL_LED_WATER
  03_CLK_PS_TO_PL_TEST
  04_PL_KEY
  04_PL_KEY_DEBOUNCE
  05_PL_HDMI_TEST
    PL_HDMI_TEST
    rgb2dvi
      docs
      src
      xgui
Smart_ZYNQ_7020
Tiny_ZYNQ
Youku Files
ZYNQ_VEIN_NEWDNA_FINAL_20221118
```

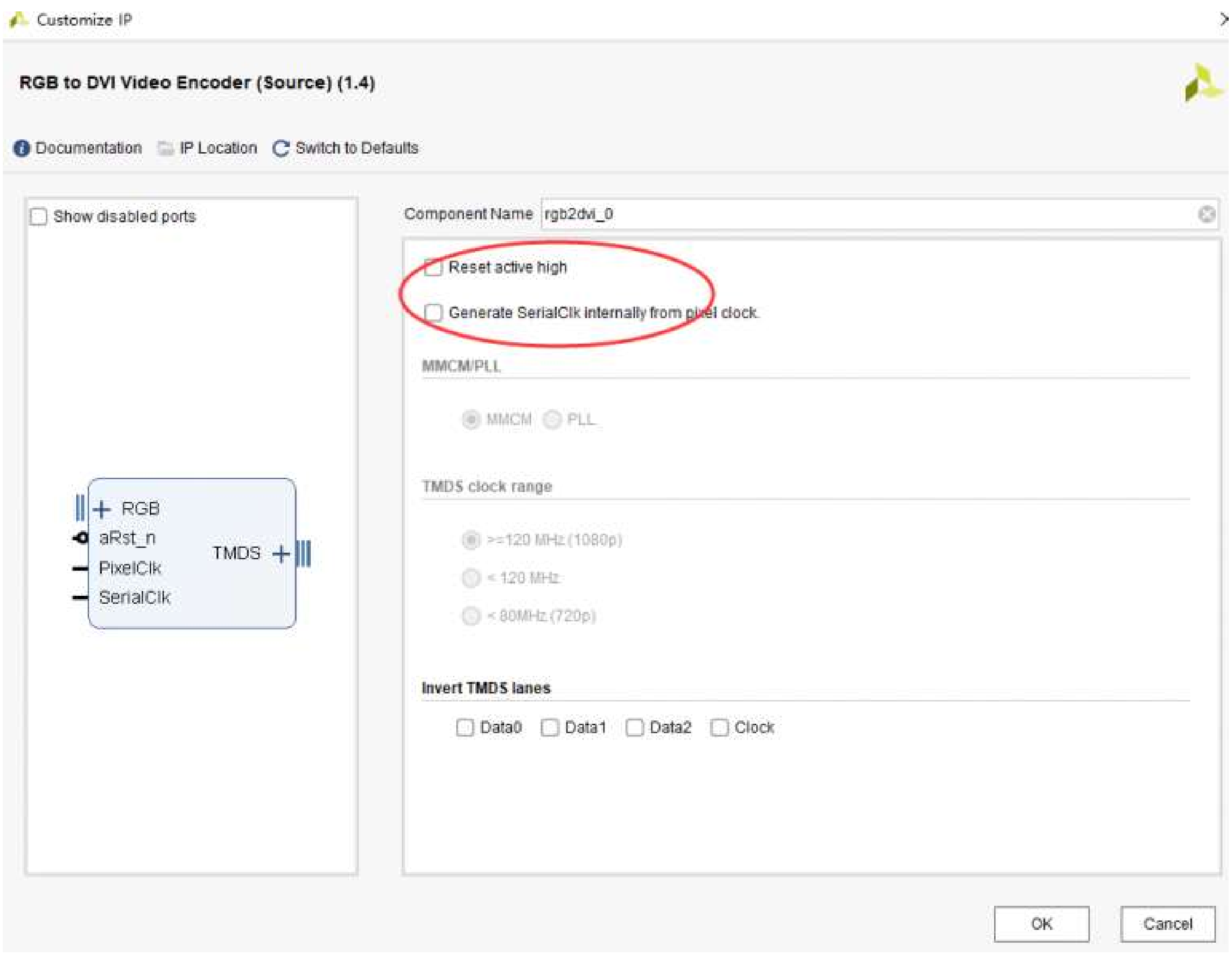
3）导入RGB2DVI IP，如下图所示，在IP管理器里搜索RGB 双击并打开RGB to DVI Video Encoder 选项



Search: rgb2 (2 matches)

Name	AXI4	Status	License	VLNV
User Repository (e:/ZYNQ_PROJECT/EBAZ4205/03_PL_HDMI_TEST/rgb2dvi)				
Video & Image Processing				
RGB to DVI Video Encoder (Source)		Pre-...	Included	digil...
Vivado Repository				
Video & Image Processing				
RGB to YCrCb Color-Space Converter	AXI...	Prod...	Included	xilin...

4）设置RGB2DVI模块，因为我们已经在时钟模块中设置了 5倍的编码时钟，所以 模块里不需要再生成模块时钟，如下图去掉复位和内部串行时钟前面的勾，点击OK


Customize IP

RGB to DVI Video Encoder (Source) (1.4)

Documentation IP Location Switch to Defaults

Show disabled ports

Component Name rgb2dvi_0

☐ Reset active high

☐ Generate SerialClk internally from pixel clock

MMCM/PLL

● MMCM ○ PLL

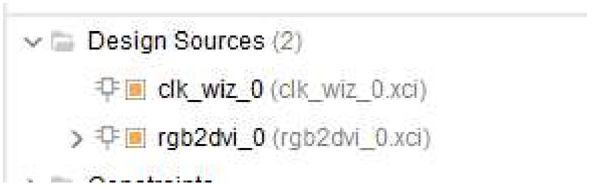
TMDS clock range

● >=120 MHz (1080p)
○ < 120 MHz
○ < 80MHz (720p)

Invert TMDS lanes

☐ Data0 ☐ Data1 ☐ Data2 ☐ Clock

完成之后 我们便得到了两个模块


Design Sources (2)
clk_wiz_0 (clk_wiz_0.xci)
rgb2dvi_0 (rgb2dvi_0.xci)

4.增加我们的代码内容

1）这里创建一个显示模块显示模块负责输出标准的RGB时序，这里调用了一个网上的参考代码，具体如下(color_bar.v)

```
module color_bar(
          input                 clk,           //pixel clock
          input                 rst,           //reset signal high
active
        output                hs,           //horizontal
synchronization
        output                vs,           //vertical
synchronization
        output                de,           //video valid
        output[7:0]           rgb_r,        //video red data
        output[7:0]           rgb_g,        //video green data
        output[7:0]           rgb_b         //video blue data
);

`define VIDEO_800_600
//video timing parameter definition
`ifdef  VIDEO_1280_720
parameter H_ACTIVE = 16'd1280;           //horizontal active time
(pixels)
parameter H_FP = 16'd110;                //horizontal front porch
(pixels)
parameter H_SYNC = 16'd40;               //horizontal sync
time(pixels)
parameter H_BP = 16'd220;                //horizontal back porch
(pixels)
parameter V_ACTIVE = 16'd720;            //vertical active Time
(lines)
parameter V_FP  = 16'd5;                 //vertical front porch
(lines)
parameter V_SYNC  = 16'd5;               //vertical sync time
(lines)
parameter V_BP  = 16'd20;                //vertical back porch
(lines)
parameter HS_POL = 1'b1;                 //horizontal sync
polarity, 1 : POSITIVE,0 : NEGATIVE;
parameter VS_POL = 1'b1;                 //vertical sync polarity,
1 : POSITIVE,0 : NEGATIVE;
`endif

//480x272 9Mhz
`ifdef  VIDEO_480_272
parameter H_ACTIVE = 16'd480;
parameter H_FP = 16'd2;
parameter H_SYNC = 16'd41;
parameter H_BP = 16'd2;
parameter V_ACTIVE = 16'd272;
parameter V_FP  = 16'd2;
parameter V_SYNC  = 16'd10;
parameter V_BP  = 16'd2;
parameter HS_POL = 1'b0;
parameter VS_POL = 1'b0;
`endif

//640x480 25.175Mhz
`ifdef  VIDEO_640_480
parameter H_ACTIVE = 16'd640;
parameter H_FP = 16'd16;
parameter H_SYNC = 16'd96;
parameter H_BP = 16'd48;
parameter V_ACTIVE = 16'd480;
parameter V_FP  = 16'd10;
parameter V_SYNC  = 16'd2;
parameter V_BP  = 16'd33;
parameter HS_POL = 1'b0;
parameter VS_POL = 1'b0;
`endif

//800x480 33Mhz
`ifdef  VIDEO_800_480
parameter H_ACTIVE = 16'd800;
parameter H_FP = 16'd40;
parameter H_SYNC = 16'd128;
parameter H_BP = 16'd88;
parameter V_ACTIVE = 16'd480;
parameter V_FP  = 16'd1;
parameter V_SYNC  = 16'd3;
parameter V_BP  = 16'd21;
parameter HS_POL = 1'b0;
parameter VS_POL = 1'b0;
`endif

//800x600 40Mhz
`ifdef  VIDEO_800_600
parameter H_ACTIVE = 16'd800;
parameter H_FP = 16'd40;
parameter H_SYNC = 16'd128;
parameter H_BP = 16'd88;
parameter V_ACTIVE = 16'd600;
parameter V_FP  = 16'd1;
parameter V_SYNC  = 16'd4;
parameter V_BP  = 16'd23;
parameter HS_POL = 1'b1;
parameter VS_POL = 1'b1;
`endif

//1024x768 65Mhz
`ifdef  VIDEO_1024_768
parameter H_ACTIVE = 16'd1024;
parameter H_FP = 16'd24;
parameter H_SYNC = 16'd136;
parameter H_BP = 16'd160;
parameter V_ACTIVE = 16'd768;
parameter V_FP  = 16'd3;
parameter V_SYNC  = 16'd6;
parameter V_BP  = 16'd29;
parameter HS_POL = 1'b0;
parameter VS_POL = 1'b0;
`endif

//1920x1080 148.5Mhz
`ifdef  VIDEO_1920_1080
parameter H_ACTIVE = 16'd1920;
parameter H_FP = 16'd88;
parameter H_SYNC = 16'd44;
parameter H_BP = 16'd148;
parameter V_ACTIVE = 16'd1080;
parameter V_FP  = 16'd4;
parameter V_SYNC  = 16'd5;
parameter V_BP  = 16'd36;
parameter HS_POL = 1'b1;
parameter VS_POL = 1'b1;
`endif
parameter H_TOTAL = H_ACTIVE + H_FP + H_SYNC + H_BP;//horizontal
total time (pixels)
parameter V_TOTAL = V_ACTIVE + V_FP + V_SYNC + V_BP;//vertical
total time (lines)
//define the RGB values for 8 colors
parameter WHITE_R       = 8'hff;
parameter WHITE_G       = 8'hff;
parameter WHITE_B       = 8'hff;
parameter YELLOW_R      = 8'hff;
parameter YELLOW_G      = 8'hff;
parameter YELLOW_B      = 8'h00;
parameter CYAN_R        = 8'h00;
parameter CYAN_G        = 8'hff;
parameter CYAN_B        = 8'hff;
parameter GREEN_R       = 8'h00;
parameter GREEN_G       = 8'hff;
parameter GREEN_B       = 8'h00;
parameter MAGENTA_R     = 8'hff;
parameter MAGENTA_G     = 8'h00;
parameter MAGENTA_B     = 8'hff;
parameter RED_R         = 8'hff;
parameter RED_G         = 8'h00;
parameter RED_B         = 8'h00;
parameter BLUE_R        = 8'h00;
parameter BLUE_G        = 8'h00;
parameter BLUE_B        = 8'hff;
parameter BLACK_R       = 8'h00;
parameter BLACK_G       = 8'h00;
parameter BLACK_B       = 8'h00;
reg hs_reg;                      //horizontal sync register
reg vs_reg;                      //vertical sync register
reg hs_reg_d0;                   //delay 1 clock of 'hs_reg'
reg vs_reg_d0;                   //delay 1 clock of 'vs_reg'
reg[11:0] h_cnt;                 //horizontal counter
reg[11:0] v_cnt;                 //vertical counter
reg[11:0] active_x;              //video x position
reg[11:0] active_y;              //video y position
reg[7:0] rgb_r_reg;              //video red data register
reg[7:0] rgb_g_reg;              //video green data register
reg[7:0] rgb_b_reg;              //video blue data register
reg h_active;                    //horizontal video active
reg v_active;                    //vertical video active
wire video_active;               //video active(horizontal active
and vertical active)
reg video_active_d0;             //delay 1 clock of video_active
assign hs = hs_reg_d0;
assign vs = vs_reg_d0;
assign video_active = h_active & v_active;
assign de = video_active_d0;
assign rgb_r = rgb_r_reg;
assign rgb_g = rgb_g_reg;
assign rgb_b = rgb_b_reg;
always@(posedge clk or posedge rst)
```

```
begin
        if(rst == 1'b1)
                begin
                        hs_reg_d0 <= 1'b0;
                        vs_reg_d0 <= 1'b0;
                        video_active_d0 <= 1'b0;
                end
        else
                begin
                        hs_reg_d0 <= hs_reg;
                        vs_reg_d0 <= vs_reg;
                        video_active_d0 <= video_active;
                end
end

always@(posedge clk or posedge rst)
begin
        if(rst == 1'b1)
                h_cnt <= 12'd0;
        else if(h_cnt == H_TOTAL - 1)//horizontal counter maximum
value
                h_cnt <= 12'd0;
        else
                h_cnt <= h_cnt + 12'd1;
end

always@(posedge clk or posedge rst)
begin
        if(rst == 1'b1)
                active_x <= 12'd0;
        else if(h_cnt >= H_FP + H_SYNC + H_BP - 1)//horizontal
video active
                active_x <= h_cnt - (H_FP[11:0] + H_SYNC[11:0] +
H_BP[11:0] - 12'd1);
        else
                active_x <= active_x;
end

always@(posedge clk or posedge rst)
begin
        if(rst == 1'b1)
                v_cnt <= 12'd0;
        else if(h_cnt == H_FP  - 1)//horizontal sync time
```

```
                if(v_cnt == V_TOTAL - 1)//vertical counter maximum
value
                        v_cnt <= 12'd0;
                else
                        v_cnt <= v_cnt + 12'd1;
        else
                v_cnt <= v_cnt;
end

always@(posedge clk or posedge rst)
begin
        if(rst == 1'b1)
                hs_reg <= 1'b0;
        else if(h_cnt == H_FP - 1)//horizontal sync begin
                hs_reg <= HS_POL;
        else if(h_cnt == H_FP + H_SYNC - 1)//horizontal sync end
                hs_reg <= ~hs_reg;
        else
                hs_reg <= hs_reg;
end

always@(posedge clk or posedge rst)
begin
        if(rst == 1'b1)
                h_active <= 1'b0;
        else if(h_cnt == H_FP + H_SYNC + H_BP - 1)//horizontal
active begin
                h_active <= 1'b1;
        else if(h_cnt == H_TOTAL - 1)//horizontal active end
                h_active <= 1'b0;
        else
                h_active <= h_active;
end

always@(posedge clk or posedge rst)
begin
        if(rst == 1'b1)
                vs_reg <= 1'd0;
        else if((v_cnt == V_FP - 1) && (h_cnt == H_FP -
1))//vertical sync begin
                vs_reg <= HS_POL;
        else if((v_cnt == V_FP + V_SYNC - 1) && (h_cnt == H_FP -
1))//vertical sync end
```

```
                vs_reg <= ~vs_reg;
        else
                vs_reg <= vs_reg;
end

always@(posedge clk or posedge rst)
begin
        if(rst == 1'b1)
                v_active <= 1'd0;
        else if((v_cnt == V_FP + V_SYNC + V_BP - 1) && (h_cnt ==
H_FP - 1))//vertical active begin
                v_active <= 1'b1;
        else if((v_cnt == V_TOTAL - 1) && (h_cnt == H_FP - 1))
//vertical active end
                v_active <= 1'b0;
        else
                v_active <= v_active;
end

always@(posedge clk or posedge rst)
begin
        if(rst == 1'b1)
                begin
                        rgb_r_reg <= 8'h00;
                        rgb_g_reg <= 8'h00;
                        rgb_b_reg <= 8'h00;
                end
        else if(video_active)
                if(active_x == 12'd0)
                        begin
                                rgb_r_reg <= WHITE_R;
                                rgb_g_reg <= WHITE_G;
                                rgb_b_reg <= WHITE_B;
                        end
                else if(active_x == (H_ACTIVE/8) * 1)
                        begin
                                rgb_r_reg <= YELLOW_R;
                                rgb_g_reg <= YELLOW_G;
                                rgb_b_reg <= YELLOW_B;
                        end
                else if(active_x == (H_ACTIVE/8) * 2)
                        begin
                                rgb_r_reg <= CYAN_R;
                                rgb_g_reg <= CYAN_G;
                                rgb_b_reg <= CYAN_B;
                        end
                else if(active_x == (H_ACTIVE/8) * 3)
                        begin
                                rgb_r_reg <= GREEN_R;
                                rgb_g_reg <= GREEN_G;
                                rgb_b_reg <= GREEN_B;
                        end
                else if(active_x == (H_ACTIVE/8) * 4)
                        begin
                                rgb_r_reg <= MAGENTA_R;
                                rgb_g_reg <= MAGENTA_G;
                                rgb_b_reg <= MAGENTA_B;
                        end
                else if(active_x == (H_ACTIVE/8) * 5)
                        begin
                                rgb_r_reg <= RED_R;
                                rgb_g_reg <= RED_G;
                                rgb_b_reg <= RED_B;
                        end
                else if(active_x == (H_ACTIVE/8) * 6)
                        begin
                                rgb_r_reg <= BLUE_R;
                                rgb_g_reg <= BLUE_G;
                                rgb_b_reg <= BLUE_B;
                        end
                else if(active_x == (H_ACTIVE/8) * 7)
                        begin
                                rgb_r_reg <= BLACK_R;
                                rgb_g_reg <= BLACK_G;
                                rgb_b_reg <= BLACK_B;
                        end
                else
                        begin
                                rgb_r_reg <= rgb_r_reg;
                                rgb_g_reg <= rgb_g_reg;
                                rgb_b_reg <= rgb_b_reg;
                        end
        else
                begin
                        rgb_r_reg <= 8'h00;
                        rgb_g_reg <= 8'h00;
```

```
                                rgb_b_reg <= 8'h00;
                        end
        end

        endmodule
```

2）创建一个顶层模块，分别调用时钟、RGB2DVI、以及彩条纹显示模块（top.v）

```
module top(
    input clk,
    output[2:0] TMDS_DATA_p,
    output[2:0] TMDS_DATA_n,
    output      TMDS_CLK_p,
    output      TMDS_CLK_n
    );

    wire clk_40m;
    wire clk_200m;
    clk_wiz_0 u2(
        .clk_in1(clk),
        .clk_out1(clk_40m),
        .clk_out2(clk_200m)
    );

    wire VGA_HS,VGA_VS,VGA_DE;
    wire[7:0] R,G,B;
    color_bar u4(
            .clk(clk_40m),            //pixel clock
            .rst(1'b0),            //reset signal high active
            .hs(VGA_HS),             //horizontal synchronization
            .vs(VGA_VS),             //vertical synchronization
            .de(VGA_DE),             //video valid
        .rgb_r(R),          //video red data
        .rgb_g(G),          //video green data
        .rgb_b(B)           //video blue data
);


    rgb2dvi_0 u1(
        .aRst_n(1'b1),
        .SerialClk(clk_200m),
        .PixelClk(clk_40m),
        .TMDS_Clk_p(TMDS_CLK_p),
        .TMDS_Clk_n(TMDS_CLK_n),
        .TMDS_Data_p(TMDS_DATA_p),
        .TMDS_Data_n(TMDS_DATA_n),
        .vid_pData({R,B,G}),
        .vid_pHSync(VGA_HS),
        .vid_pVSync(VGA_VS),
        .vid_pVDE(VGA_DE)
    );
endmodule
```

最后再增加约束文件 （HDMI_TEST.XDC）

```
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN M21 [get_ports {TMDS_DATA_p[0]}]
set_property PACKAGE_PIN L21 [get_ports {TMDS_DATA_p[1]}]
set_property PACKAGE_PIN J21 [get_ports {TMDS_DATA_p[2]}]
set_property PACKAGE_PIN N22 [get_ports TMDS_CLK_p]
set_property PACKAGE_PIN M19 [get_ports clk]
```

6.编译综合，并运行代码

最终显示效果如图

完整工程下载:**(板子有多个版本，请按照对应板子的芯片型号进行下载测试)**

05_PL_HDMI_TEST_XC7Z020　　**下载**

以上代码实现 800X600的 60hz HDMI 稳定输出

章节内容补， 增加always方式来实现分频，将分辨率提高到1080P 60HZ （只测试了部分显示器，这边测试稳定输出，不代表全部显示器都能兼容，请自行尝试）

经过测试本例程通过修改时钟可以 实现低到640X480 60HZ 高到 1080P 60HZ **（60HZ 在我自己的三星曲面屏上可以实可以稳定输出）**

为了实现不同分辨率，用PLL 来输出两个5倍关系的时钟并不一定高度同步，这边可以换个思路，用PLL 生成5倍频时钟，然后用always块来分频生成像素时钟

具体如下

```
reg PIX_CLK;
wire PIX_CLK_5X;
clk_wiz_0 u2(
    .clk_in1(clk),
    .clk_out1(PIX_CLK_5X)
);

reg [2:0]count;
always@(posedge PIX_CLK_5X)begin
```

```
        if(count>=3'd4)begin PIX_CLK<=1'b1;count<=3'd0;end
        else begin  PIX_CLK<=1'b0; count<=count+1'b1;end
    end
```

通过上述方式 PLL 生成一个5倍的像素时钟 PIX_CLK_5X的时钟，而经过always块的5拍分频，生成了个单倍的时钟PIX_CLK，通过这种方式，受PLL的限制比较小。

主分辨率设置成742.5M（148.5M的5倍）实际PLL倍频出的结果是742.188M 稍有区别，这里可能会导致部分显示器不兼容

下面是 1080P 60HZ 的例子，用了上述时钟生成的方法（备注，**因为1080P情况下mmcm的时钟已经超过mmcm IP设置的最大推荐值了**，增加不同的显示功能时**对逻辑的时序约束要求比较高**，所以增加自己的功能可能会出现不可预料的结果，建议大家自行尝试，**如果是学习调试，可以先从720P的代码入手**）

Here is the complete code for the 1080P 60HZ :

05_PL_HDMI_TEST_1080P_60HZ_XC7Z020 **Download**

SMART ZYNQ SP & SL