基于FPGA的OV7725摄像头的HDMI显示（含源码）

基于FPGA的HDMI设计导航页面，点击跳转。

1、概述

本文 FPGA 通过SCCB接口初始化OV7725摄像头寄存器，然后采集OV7725的摄像头数据，使用DDR3对数据进行暂存，最后将数据输出到HDMI显示器上进行显示。

该工程对应系统框图如下所示，主要包含OV7725驱动及数据处理模块、DDR3读写控制模块、HDMI控制模块。其中DDR3读写控制模块和 HDMI 显示控制模块在前文设计中已经进行详细讲解过了，本文直接使用即可，不再对其功能的实现进行赘述，不了解的可以参考前文。

由于只需要向OV7725的寄存器中写入数据，所以可以直接使用前文编写的I2C驱动模块代替SCCB驱动模块。iic_drive模块是I2C的驱动模块，兼容SCCB的写时序。init_contrl模块主要功能是控制初始化OV7725的哪些寄存器。Capture_data模块将摄像头输出的8位数据拼接为16位数据，然后传输给DDR3读写模块，将数据存入DDR3中。

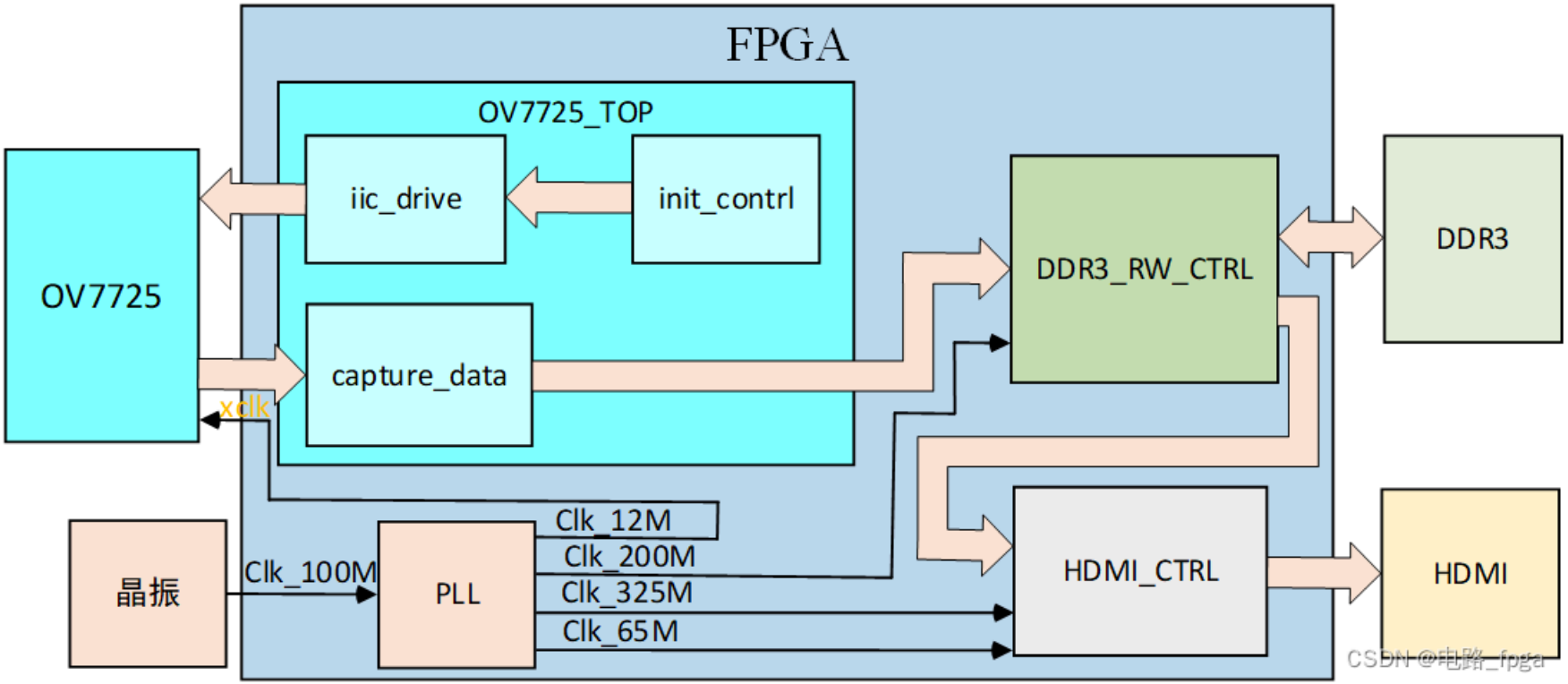


图1 系统框图

后面DDR3控制模块和HDMI显示模块的设计思路与以太网传输图片中两个模块的设计基本一致。

外部的晶振输入100MHz时钟，通过锁相环进行倍频、分频处理，给OV7725提供12MHz的系统时钟XCLK，给DDR3控制模块提供200MHz的参考时钟和MIG IP工作时钟。由于显示器分辨率为1024*768，所以还需要给HDMI模块提供65MHz时钟和325MHz的参考时钟信号。

对于异步信号，首先DDR3读写控制模块的读、写两侧都是异步FIFO，相应的数据就能够异步FIFO交换数据，不需要做额外处理。

2、OV7725初始化控制模块

根据官方提供的初始化参数 ，对70个寄存器进行初始化，最终输出PCLK时钟频率为24MHz，640*480分辨率的RGB565像素数据。根据前文时序的介绍，首先对8'h12寄存器进行设置，使所有寄存器复位，发出该命令之后，需要等待1ms才能开始配置其余寄存器。

因此通过一个复位计数器和复位标志信号来对复位状态进行指示，为了简化电路，此处使用的计数器依靠溢出清零。

如下代码所示，复位指示信号初始值为0，wdata_cnt计数器用于表示已经初始化的寄存器个数。因为最先初始化的是复位计数器，因此当wdata_cnt等于1并且I2C驱动模块空闲的时候，证明复位寄存器已经被写入数据了，此时复位指示信号拉高，当复位计数器所有位均为高电平时，表示复位的持续时间大于1ms，此时可以配置后续寄存器了。

```
    //生成复位指示信号，初始为低电平，当复位寄存器且等待1ms之后拉低，表示复位完成；
    always@(posedge clk)begin
        if(rst_n==1'b0)begin//初始值为0;
            rst_flag <= 1'b0;
        end//当延时计数器全为1时，表示延时1.3ms，复位完成。
        else if(&delay_cnt)begin
            rst_flag <= 1'b0;
        end//当配置第0个数据之后，如果I2C驱动模块空闲，表示复位寄存器初始化完成，之后需要延时1ms才能继续配置其他寄存器。
        else if(wdata_cnt == 7'd1 && rdy)begin
            rst_flag <= 1'b1;
        end
    end
```

下面代码就是复位的延时计数器，当复位指示信号有效时，对系统时钟进行计数，当溢出时清零，系统时钟周期为10ns，17位计数器从0计数到最大值需要131071个时钟，即1.31ms。

```
    //为了便于计数，直接使用一个17位计数器对100MHz时钟进行计数。
    always@(posedge clk)begin
        if(rst_n==1'b0)begin//初始值为0;
            delay_cnt <= 17'd0;
        end
        else if(rst_flag)begin//处于复位状态下，对系统时钟进行计数。

```

```
            delay_cnt <= delay_cnt + 17'd1;
        end
    end
```

之后就是计数器wdata_cnt，对初始化的寄存器个数进行计数。初始值为0，当初始化复位计数器时，要等延时计数器计数结束才能加一，如果初始化的不是复位寄存器，且已配置寄存器个数小于需要配置的寄存器个数，且I2C驱动模块处于空闲时加1。

```
    //wdata_cnt计数器用于记录配置的寄存器个数；
    always@(posedge clk)begin
        if(rst_n==1'b0)begin//
            wdata_cnt <= 0;
        end
        else if(add_wdata_cnt)begin
            wdata_cnt <= wdata_cnt + 1;
        end
    end

    //记录初始化寄存器的个数，由于第0个寄存器初始化之后需要延时1ms，所以当计数器等于1时，需要等待延时计数器计数结束才能加1。
    //其余时间只需要等待I2C驱动模块空闲，就可以发送数据。
    assign add_wdata_cnt = (((wdata_cnt == 7'd1) && (&delay_cnt)) || ((wdata_cnt != 7'd1) && rdy && (wdata_cnt < REG_NUM)));
```

将计数器加一条件延时一个时钟周期作为I2C驱动模块的开始信号，由于只需要向寄存器写入数据，所以读写标志信号一直拉低即可。根据计数器的数值生成需要写入寄存器的地址和数据。

```
    //写开始信号，当计数器加一条件有效且没有配置完所有寄存器时拉高；
    always@(posedge clk)begin
        if(rst_n==1'b0)begin//初始值为0;
            start <= 1'b0;
        end
        else begin
            start <= add_wdata_cnt;
        end
    end

    assign rw_flag = 1'b0;//读写控制信号置为低电平，因为只需要对寄存器进行初始化，不会有读操作。

    //设置寄存器的地址和数据；
    always@(posedge clk)begin
        if(rst_n==1'b0)begin//初始值为0;
```

```
            {reg_addr,wdata} <= {8'h1C, 8'h7F}; //MIDH 制造商ID 高8位;
        end
        else if(add_wdata_cnt)begin
            case(wdata_cnt)
                //先对寄存器进行软件复位，使寄存器恢复初始值;
                //寄存器软件复位后，需要延时1ms才能配置其它寄存器;
                7'd0  : {reg_addr,wdata} <= {8'h12, 8'h80}; //COM7 BIT[7]:复位所有的寄存器;
                7'd1  : {reg_addr,wdata} <= {8'h3d, 8'h03}; //COM12 模拟过程直流补偿;
                7'd2  : {reg_addr,wdata} <= {8'h15, 8'h02}; //COM10 href/vsync/pclk/data信号控制;
                7'd3  : {reg_addr,wdata} <= {8'h17, 8'h23}; //HSTART 水平起始位置;
                7'd4  : {reg_addr,wdata} <= {8'h18, 8'ha0}; //HSIZE 水平尺寸;
                7'd5  : {reg_addr,wdata} <= {8'h19, 8'h07}; //VSTRT 垂直起始位置;
                7'd6  : {reg_addr,wdata} <= {8'h1a, 8'hf0}; //VSIZE 垂直尺寸;
                7'd7  : {reg_addr,wdata} <= {8'h32, 8'h00}; //HREF 图像开始和尺寸控制，控制低位;
                7'd8  : {reg_addr,wdata} <= {8'h29, 8'ha0}; //HOutSize 水平输出尺寸;
                7'd9  : {reg_addr,wdata} <= {8'h2a, 8'h00}; //EXHCH 虚拟像素MSB;
                7'd10 : {reg_addr,wdata} <= {8'h2b, 8'h00}; //EXHCL 虚拟像素LSB;
                7'd11 : {reg_addr,wdata} <= {8'h2c, 8'hf0}; //VOutSize 垂直输出尺寸;
                7'd12 : {reg_addr,wdata} <= {8'h0d, 8'h41}; //COM4 PLL倍频设置(multiplier);
                7'd13 : {reg_addr,wdata} <= {8'h11, 8'h00}; //CLKRC 内部时钟配置;
                7'd14 : {reg_addr,wdata} <= {8'h12, 8'h06}; //COM7 输出VGA RGB565格式;
                7'd15 : {reg_addr,wdata} <= {8'h0c, 8'h10}; //COM3 Bit[0]: 0:图像数据 1:彩条测试;
                //DSP 控制
                7'd16 : {reg_addr,wdata} <= {8'h42, 8'h7f}; //TGT_B 黑电平校准蓝色通道目标值;
                7'd17 : {reg_addr,wdata} <= {8'h4d, 8'h09}; //FixGain 模拟增益放大器;
                7'd18 : {reg_addr,wdata} <= {8'h63, 8'hf0}; //AWB_Ctrl0 自动白平衡控制字节0;
                7'd19 : {reg_addr,wdata} <= {8'h64, 8'hff}; //DSP_Ctrl1 DSP控制字节1;
                7'd20 : {reg_addr,wdata} <= {8'h65, 8'h00}; //DSP_Ctrl2 DSP控制字节2;
                7'd21 : {reg_addr,wdata} <= {8'h66, 8'h00}; //DSP_Ctrl3 DSP控制字节3;
                7'd22 : {reg_addr,wdata} <= {8'h67, 8'h00}; //DSP_Ctrl4 DSP控制字节4;
                //AGC AEC AWB
                //COM8 Bit[2]:自动增益使能 Bit[1]:自动白平衡使能 Bit[0]:自动曝光功能;
                7'd23 : {reg_addr,wdata} <= {8'h13, 8'hff}; //COM8;
                7'd24 : {reg_addr,wdata} <= {8'h0f, 8'hc5}; //COM6;
                7'd25 : {reg_addr,wdata} <= {8'h14, 8'h11};
                7'd26 : {reg_addr,wdata} <= {8'h22, 8'h98};
                7'd27 : {reg_addr,wdata} <= {8'h23, 8'h03};
                7'd28 : {reg_addr,wdata} <= {8'h24, 8'h40};
                7'd29 : {reg_addr,wdata} <= {8'h25, 8'h30};
                7'd30 : {reg_addr,wdata} <= {8'h26, 8'ha1};

```

```
                7'd31 : {reg_addr,wdata} <= {8'h6b, 8'haa};
                7'd32 : {reg_addr,wdata} <= {8'h13, 8'hff};
                //matrix sharpness brightness contrast UV
                7'd33 : {reg_addr,wdata} <= {8'h90, 8'h0a}; //EDGE1 边缘增强控制1;
                //DNSOff 降噪阈值下限,仅在自动模式下有效
                7'd34 : {reg_addr,wdata} <= {8'h91, 8'h01}; //DNSOff;
                7'd35 : {reg_addr,wdata} <= {8'h92, 8'h01}; //EDGE2 锐度(边缘增强)强度上限;
                7'd36 : {reg_addr,wdata} <= {8'h93, 8'h01}; //EDGE3 锐度(边缘增强)强度下限;
                7'd37 : {reg_addr,wdata} <= {8'h94, 8'h5f}; //MTX1 矩阵系数1;
                7'd38 : {reg_addr,wdata} <= {8'h95, 8'h53}; //MTX1 矩阵系数2;
                7'd39 : {reg_addr,wdata} <= {8'h96, 8'h11}; //MTX1 矩阵系数3;
                7'd40 : {reg_addr,wdata} <= {8'h97, 8'h1a}; //MTX1 矩阵系数4;
                7'd41 : {reg_addr,wdata} <= {8'h98, 8'h3d}; //MTX1 矩阵系数5;
                7'd42 : {reg_addr,wdata} <= {8'h99, 8'h5a}; //MTX1 矩阵系数6;
                7'd43 : {reg_addr,wdata} <= {8'h9a, 8'h1e}; //MTX_Ctrl 矩阵控制;
                7'd44 : {reg_addr,wdata} <= {8'h9b, 8'h3f}; //BRIGHT 亮度;
                7'd45 : {reg_addr,wdata} <= {8'h9c, 8'h25}; //CNST 对比度;
                7'd46 : {reg_addr,wdata} <= {8'h9e, 8'h81};
                7'd47 : {reg_addr,wdata} <= {8'ha6, 8'h06}; //SDE 特殊数字效果控制;
                7'd48 : {reg_addr,wdata} <= {8'ha7, 8'h65}; //USAT "U"饱和增益;
                7'd49 : {reg_addr,wdata} <= {8'ha8, 8'h65}; //VSAT "V"饱和增益;
                7'd50 : {reg_addr,wdata} <= {8'ha9, 8'h80}; //VSAT "V"饱和增益;
                7'd51 : {reg_addr,wdata} <= {8'haa, 8'h80}; //VSAT "V"饱和增益;
                //伽马控制
                7'd52 : {reg_addr,wdata} <= {8'h7e, 8'h0c};
                7'd53 : {reg_addr,wdata} <= {8'h7f, 8'h16};
                7'd54 : {reg_addr,wdata} <= {8'h80, 8'h2a};
                7'd55 : {reg_addr,wdata} <= {8'h81, 8'h4e};
                7'd56 : {reg_addr,wdata} <= {8'h82, 8'h61};
                7'd57 : {reg_addr,wdata} <= {8'h83, 8'h6f};
                7'd58 : {reg_addr,wdata} <= {8'h84, 8'h7b};
                7'd59 : {reg_addr,wdata} <= {8'h85, 8'h86};
                7'd60 : {reg_addr,wdata} <= {8'h86, 8'h8e};
                7'd61 : {reg_addr,wdata} <= {8'h87, 8'h97};
                7'd62 : {reg_addr,wdata} <= {8'h88, 8'ha4};
                7'd63 : {reg_addr,wdata} <= {8'h89, 8'haf};
                7'd64 : {reg_addr,wdata} <= {8'h8a, 8'hc5};
                7'd65 : {reg_addr,wdata} <= {8'h8b, 8'hd7};
                7'd66 : {reg_addr,wdata} <= {8'h8c, 8'he8};
                7'd67 : {reg_addr,wdata} <= {8'h8d, 8'h20};
```

```
                7'd68 : {reg_addr,wdata} <= {8'h0e, 8'h65}; //COM5;
                7'd69 : {reg_addr,wdata} <= {8'h09, 8'h00}; //COM2  Bit[1:0] 输出电流驱动能力;
                //只读存储器,防止在case中没有列举的情况，之前的寄存器被重复改写;
                default:{reg_addr,wdata} <= {8'h1C, 8'h7F}; //MIDH 制造商ID 高8位;
            endcase
        end
    end
```

当所有寄存器配置完成后，将配置完成信号拉高。

```
    //初始化完成信号;
    always@(posedge clk)begin
        if(rst_n==1'b0)begin//初始值为0;
            init_done <= 1'b0;
        end//当所有寄存器写入数据，且I2C驱动模块处于空闲状态时，表示初始化完成;
        else if((wdata_cnt == REG_NUM) && rdy)begin
            init_done <= 1'b1;
        end
    end
```

由于该模块的逻辑比较简单，因此没有进行仿真，后续直接上板即可，参考代码如下：

3、摄像头数据处理模块

由于配置摄像头寄存器之后，需要10帧图像数据的时间，才能正常传输数据。当寄存器初始化完成之后，对场同步信号的上升沿进行检测，直到检测到10个为止。

```
    //将场同步信号延迟两个时钟周期，用于检测其上升沿;
    always@(posedge cam_pclk)begin
        cam_vsync_r <= {cam_vsync_r[0],cam_vsync};
        cam_href_r <= {cam_href_r[0],cam_href};
    end

    //检测场同步信号上升沿;
    assign cam_vsync_pos = cam_vsync_r[0] & (~cam_vsync_r[1]);
```

```
    assign cam_href_neg = (~cam_href_r[0]) & cam_href_r[1];

    //延时计数器，用于记录复位后，前几帧数据；
    always@(posedge cam_pclk)begin
        if(rst_n==1'b0)begin//
            delay_cnt <= 0;
        end//当寄存器初始化完成，且检测到场同步上升沿，且小于10帧时加1；
        else if(ov7725_init_done && cam_vsync_pos && (delay_cnt < WAIT_FRAME - 1))begin
            delay_cnt <= delay_cnt + 1;
        end
    end

    //等待完成信号；
    always@(posedge cam_pclk)begin
        if(rst_n==1'b0)begin//初始值为0;
            delay_done <= 1'b0;
        end//
        else if(ov7725_init_done && cam_vsync_pos && (delay_cnt >= WAIT_FRAME - 1))begin
            delay_done <= 1'b1;
        end
    end
```

由于需要把输入的8位像素数据拼接为16位像素数据输出，需要一个标志信号byte_flag，每行结束或者每帧数据开始时清零该信号，当像素有效信号HREF为高电平时，byte_flag翻转。

```
    //相当于一个计数器，用于记录采集数据的个数，当数据有效时翻转，为1时表示采集了两个数据；
    always@(posedge cam_pclk)begin
        if(rst_n==1'b0)begin//初始值为0;
            byte_flag <= 1'b0;
        end
        else if(cam_href_neg | cam_vsync_pos)begin//在一帧数据开始或者一行数据接收结束时清零，便于下次计数的正确；
            byte_flag <= 1'b0;
        end
        else if(cam_href)begin
            byte_flag <= ~byte_flag;

```

```
        end
    end
```

当输入像素有效（HREF位高电平），如果byte_flag为低电平，表示输入高字节数据，如果byte_flag为高电平，表示输入低字节数据。之后将像素输出有效指示信号拉高。最后可以把场同步信号输出，作为DDR3读、写控制模块写FIFO的复位信号，每帧数据开始时会对写FIFO进行复位，保证下一帧数据正确存储。

```
    //将8位数据转换为16位数据输出，摄像头先传输高字节数据，后传输低字节数据；
    always@(posedge cam_pclk)begin
        if(rst_n==1'b0)begin//初始值为0;
            cmos_frame_data <= 16'd0;
        end
        else if(cam_href)begin
            if(byte_flag)//采集低字节数据；
                cmos_frame_data[7:0] <= cam_data;
            else//采集高字节数据；
                cmos_frame_data[15:8] <= cam_data;
        end
    end

    //输出像素有效指示信号，当采集完两次数据后输出一次有效数据；
    always@(posedge cam_pclk)begin
        if(rst_n==1'b0)begin//初始值为0;
            cmos_frame_valid <= 1'b0;
        end
        else begin//当采集完两次数据且已经过了延时时输出一次有效数据
            cmos_frame_valid <= cam_href & byte_flag & delay_done;
        end
    end

    //输出场同步信号，延时完成后，将行同步信号延时一个时钟输出；
    assign cmos_frame_vsync = delay_done ? cam_vsync_r[0] : 1'b0;
```

该模块的设计就这么简单，起始可以不用关心前10帧图像数据，可以继续简化模块设计。

4、顶层模块

本次设计由于生成的时钟比较多，需要例化两个锁相环模块才能完成。另外摄像头采集的场同步信号作为DDR3读写控制模块的写FIFO复位，由于场同步信号拉高会提前有效像素几千个时钟，这段时间完成FIFO的复位完全没问题。

DDR3读写控制模块使用乒乓模式，因为读速率比较快，所以始终读与写地址相反的地址区域即可。

另外需要对HDMI数据请求信号修改，由于显示器的像素为1024*768，而摄像头采集数据的像素是640*480，需要把图像显示在显示器的中心区域。

则水平像素显示区域为191$_{831}$，垂直像素显示区域为143$_{623}$，这段时间才能把DDR3读写控制模块的读使能信号拉高，一个时钟后得到像素数据。

```
109
110     //请求像素点颜色数据输入，在产生杭长同步信号前两个时钟向上游产生请求信号；
111     always@(posedge clk)begin
112         if(~rst_n)//初始值为0;
113             data_req <= 1'b0;
114         else if((cnt_h >= SHOW_H_B + 192 - 1) && (cnt_h < SHOW_H_E - 192 - 1) && (cnt_v >= SHOW_V_B + 144 - 1) && (cnt_v < SHOW_V_E - 144 - 1))
115             data_req <= 1'b1;
116         else
117             data_req <= 1'b0;
118     end
119
120     always@(posedge clk)begin
121         if(~rst_n)//初始值为0;
122             data_req_r <= 1'b0;
123         else if((cnt_h >= SHOW_H_B - 1) && (cnt_h < SHOW_H_E - 1) && (cnt_v >= SHOW_V_B - 1) && (cnt_v < SHOW_V_E - 1))
124             data_req_r <= 1'b1;
125         else
126             data_req_r <= 1'b0;
127     end
```

CSDN @电路_fpga

图2 数据请求信号的范围进行更改

将HDMI的场同步信号作为DDR3读写控制模块的读FIFO的复位信号，每读取一帧数据就对读FIFO的数据清零，确保下一帧图像正确显示。

由于摄像头采集的数据一行包含640个像素点，为了DDR3读侧FIFO不溢出，FIFO深度设置为2048。

整个模块的RTL视图如下所示：

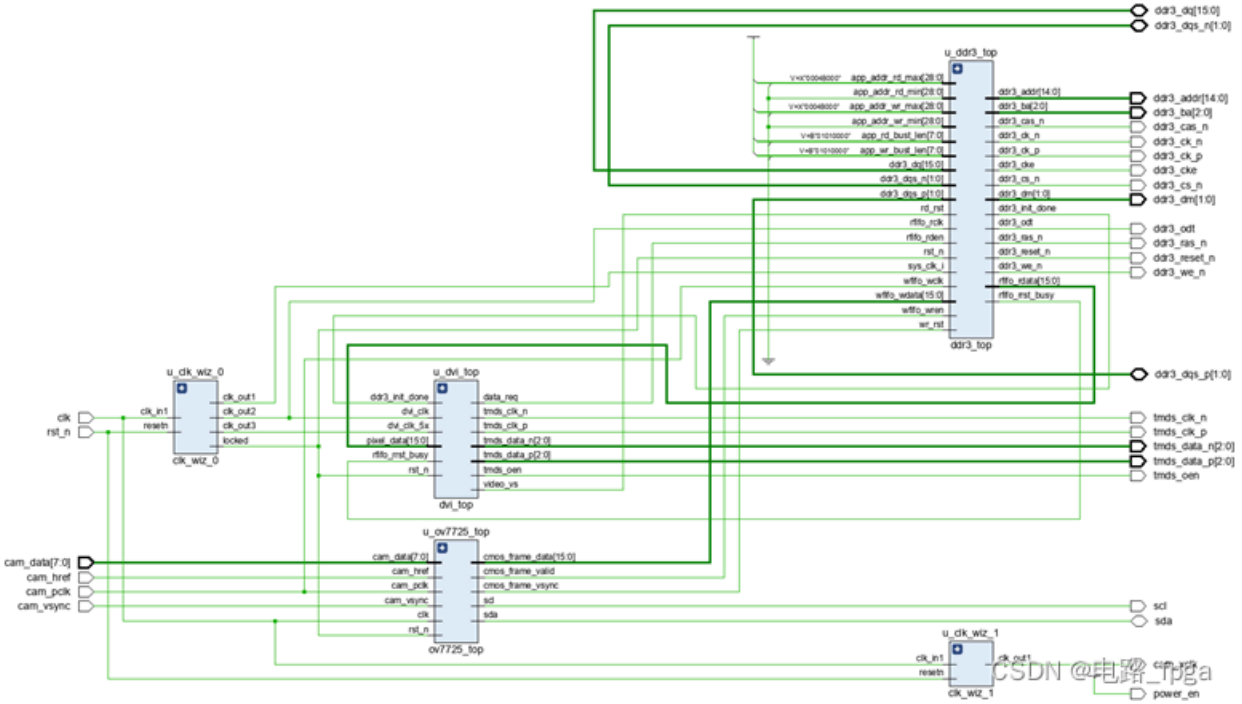


图3 RTL系统视图

参考代码如下：

```
    assign power_en = 1'b1;//使能模块电源，仅对此模块有用；

    //例化锁相环，输出200MHZ时钟，作为DDR的参考时钟；
    //生成65MHz时钟作为HDMI的参考时钟信号；
    clk_wiz_0 u_clk_wiz_0(
        .clk_out1   ( clk_200m      ),//output clk_out1
        .clk_out2   ( dvi_clk       ),//output clk_out2
        .clk_out3   ( dvi_clk_5x    ),//output clk_out3
        .resetn     ( rst_n         ),//input resetn
        .locked     ( sys_rst_n     ),//output locked
        .clk_in1    ( clk           ) //input clk_in1
    );

    //还要生成一路频率位12MHz的时钟输出给摄像头模块；
    clk_wiz_1 u_clk_wiz_1(
        .clk_out1   ( cam_xclk      ),//output clk_out1
```

```
        .resetn     ( rst_n         ),//input resetn
        .clk_in1    ( clk           ) //input clk_in1
    );

    //例化OV7725摄像头采集模块;
    ov7725_top u_ov7725_top (
        .clk                ( clk               ),//系统时钟信号,100MHz;
        .rst_n              ( sys_rst_n         ),//系统复位信号，低电平有效;
        .cam_pclk           ( cam_pclk          ),//摄像头数据像素时钟;
        .cam_vsync          ( cam_vsync         ),//摄像头场同步信号;
        .cam_href           ( cam_href          ),//摄像头行同步信号;
        .cam_data           ( cam_data          ),//摄像头输入数据信号;
        .cmos_frame_vsync   ( cmos_frame_vsync  ),//帧有效信号;
        .cmos_frame_valid   ( cmos_frame_valid  ),//数据有效使能信号;
        .cmos_frame_data    ( cmos_frame_data   ),//有效数据;
        .scl                ( scl               ),//SCCB串行时钟信号;
        .sda                ( sda               ) //SCCB双向串行数据信号;
    );

    //例化DDR3顶层模块
    ddr3_top u_ddr3_top (
        .sys_clk_i          ( clk_200m          ),//MIG IP核输入时钟，200MHz;
        .rst_n              ( sys_rst_n         ),//复位,低有效;
        .ddr3_init_done     ( ddr3_init_done    ),//ddr3初始化完成信号;
        //DDR3接口信号
        .ddr3_addr          ( ddr3_addr         ),//ddr3 地址;
        .ddr3_ba            ( ddr3_ba           ),//ddr3 banck地址;
        .ddr3_ras_n         ( ddr3_ras_n        ),//ddr3 行选择;
        .ddr3_cas_n         ( ddr3_cas_n        ),//ddr3 列选择;
        .ddr3_we_n          ( ddr3_we_n         ),//ddr3 读写选择;
        .ddr3_reset_n       ( ddr3_reset_n      ),//ddr3 复位;
        .ddr3_ck_p          ( ddr3_ck_p         ),//ddr3 时钟正;
        .ddr3_ck_n          ( ddr3_ck_n         ),//ddr3 时钟负;
        .ddr3_cke           ( ddr3_cke          ),//ddr3 时钟使能;
        .ddr3_cs_n          ( ddr3_cs_n         ),//ddr3 片选;
        .ddr3_dm            ( ddr3_dm           ),//ddr3_dm;
        .ddr3_odt           ( ddr3_odt          ),//ddr3_odt;
        .ddr3_dq            ( ddr3_dq           ),//ddr3 数据;
        .ddr3_dqs_n         ( ddr3_dqs_n        ),//ddr3 dqs负;
        .ddr3_dqs_p         ( ddr3_dqs_p        ),//ddr3 dqs正;

```

```
        //复位及突发读写长度设置信号;
        .app_addr_wr_min    ( 29'd0             ),//读ddr3的起始地址;
        .app_addr_wr_max    ( 29'd307200        ),//读ddr3的结束地址;
        .app_wr_bust_len    ( 8'd80             ),//从ddr3中读数据时的突发长度;
        .app_addr_rd_min    ( 29'd0             ),//读ddr3的起始地址;
        .app_addr_rd_max    ( 29'd307200        ),//读ddr3的结束地址;
        .app_rd_bust_len    ( 8'd80             ),//从ddr3中读数据时的突发长度;
        .wr_rst             ( cmos_frame_vsync  ),//写复位信号，上升沿有效，持续时间必须大于ui_clk的周期;
        .rd_rst             ( video_vs          ),//读复位信号，上升沿有效，持续时间必须大于ui_clk周期;
        //写数据相关信号;
        .wfifo_wclk         ( cam_pclk          ),//写FIFO写时钟信号;
        .wfifo_wren         ( cmos_frame_valid  ),//写FIFO写使能信号;
        .wfifo_wdata        ( cmos_frame_data   ),//写FIFO写数据信号;
        .wfifo_wcount       (                   ),//写FIFO中的数据个数;
        .wfifo_full         ( wfifo_full        ),//写FIFO满指示信号;
        .wfifo_wrst_busy    ( wfifo_wrst_busy   ),//写FIFO复位完成指示信号，低电平表示复位完成;
        //读数据相关信号
        .rfifo_rclk         ( dvi_clk           ),//读FIFO读时钟;
        .rfifo_rden         ( data_req          ),//读FIFO读使能信号;
        .rfifo_rdata        ( pixel_data        ),//读FIFO读数据;
        .rfifo_rcount       ( rfifo_rcount      ),//读FIFO中的数据个数;
        .rfifo_empty        (                   ),//读FIFO空指示信号;
        .rfifo_rrst_busy    ( rfifo_rrst_busy   ) //读FIFO复位状态指示信号，高电平表示处于复位过程中。
    );

    //例化DVI接口驱动模块
    dvi_top  u_dvi_top (
        .dvi_clk            ( dvi_clk           ),//DVI时钟信号，1024*768分辨率时为65MHz
        .dvi_clk_5x         ( dvi_clk_5x        ),//DVI的5倍参考时钟信号,325MHz.
        .rst_n              ( sys_rst_n         ),//复位信号，低电平有效。
        .ddr3_init_done     ( ddr3_init_done    ),//DDR3初始化完成;
        .rfifo_rrst_busy    ( rfifo_rrst_busy   ),//读FIFO的复位状态指示信号;
        .pixel_data         ( pixel_data        ),
        .data_req           ( data_req          ),
        .video_vs           ( video_vs          ),
        //HDMI接口信号
        .tmds_oen           ( tmds_oen          ),
        .tmds_clk_p         ( tmds_clk_p        ),
        .tmds_clk_n         ( tmds_clk_n        ),
        .tmds_data_p        ( tmds_data_p       ),

```

```
        .tmds_data_n        ( tmds_data_n       )
    );
```

5、上板测试

综合工程之后，下载到开发板实测结果如下图所示；

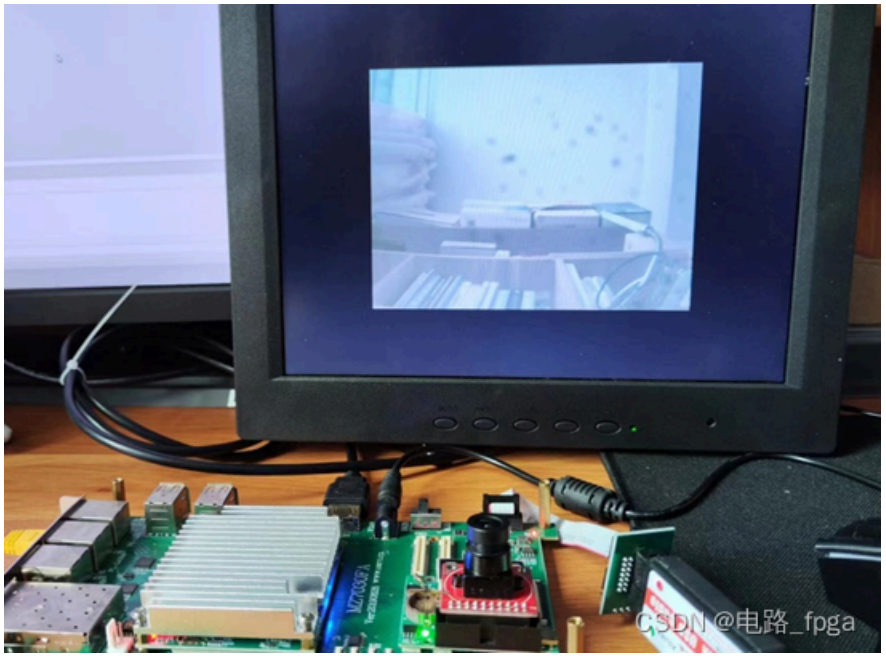
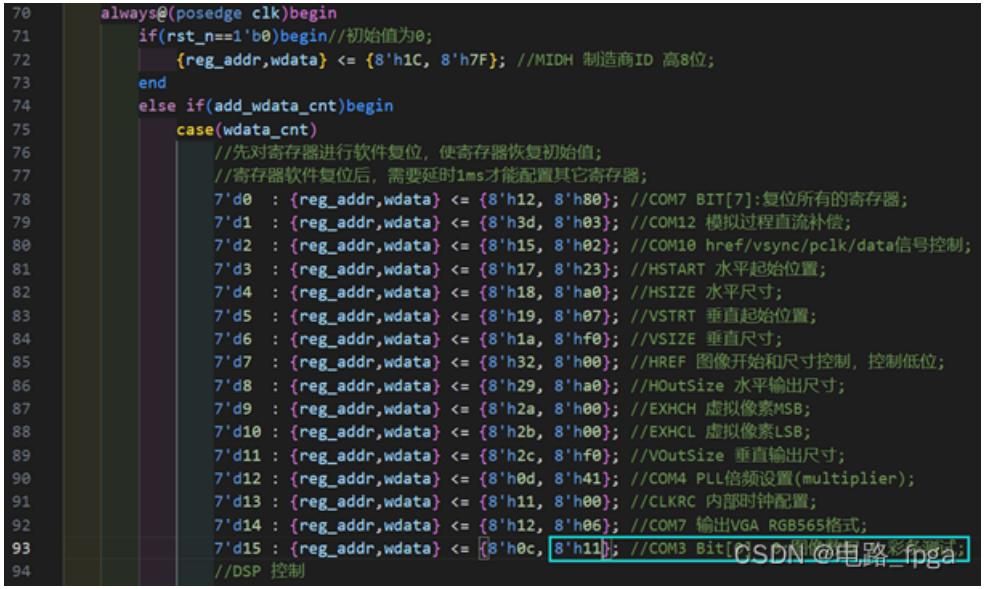


图4 开发板测试平台

由于我手里这颗摄像头年代太过久远，且镜头上有一些杂质，导致摄像头成像比较差。

图5 显示器显示摄像头数据

可以通过修改配置寄存器的数值，如下图所示，将摄像头改成输出彩条图像。即寄存器8'h0c的最低位改为1，摄像头被设置为输出彩条测试模式。



CSDN @电路_fpga

图6 修改测试程序

最终得到的彩条显示结果如下图所示，证明FPGA整个工程的数据处理是没有问题的，后续更换一个摄像头就好了。

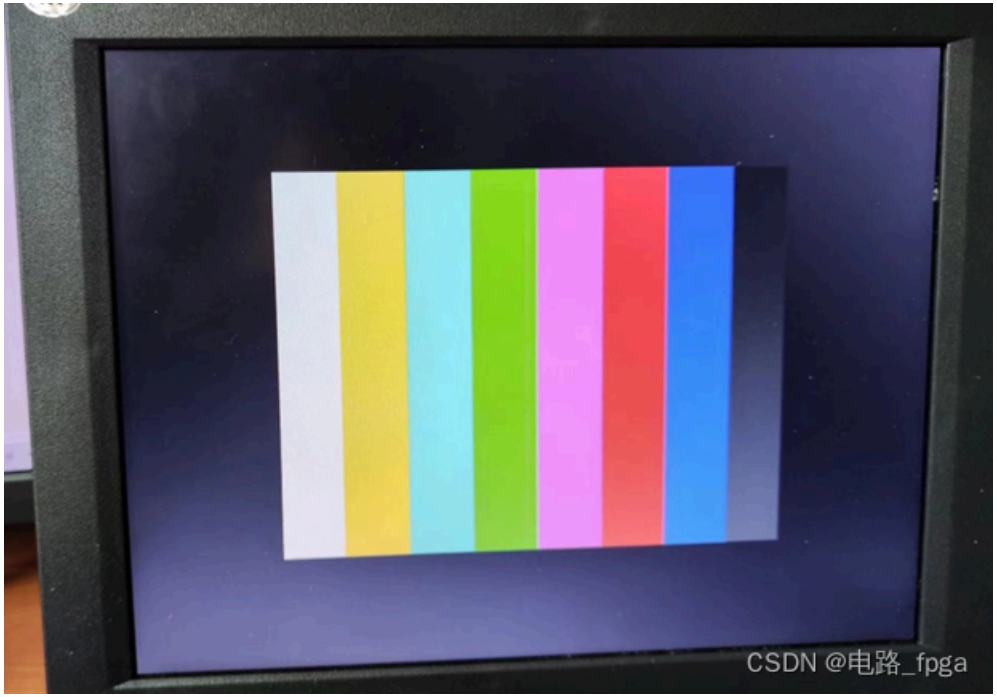


图7 显示器显示摄像头输出的彩条测试图像

本次工程设计到此结束了，有前文的DDR3设计和HDMI设计作为保障，摄像头采集数据通过HDMI显示在显示器上的工程其实并不复杂。

积少成多，量变终究引发质变，对于每个细节都要找到问题原因，一起加油！！！

本工程可以在公众号后台回复“**基于OV7725摄像头的HDMI显示**”（不包括引号）获取，工程项目使用vivado2021.1在zynq7030上进行开发。

如果对文章内容理解有疑惑或者对代码不理解，可以在评论区或者后台留言，看到后均会回复！

如果本文对您有帮助，还请多多点赞👍、评论💬和收藏⭐！您的支持是我更新的最大动力！将持续更新工程！