同济大学计算机系

数字逻辑课程综合实验报告

学　　号 2050254

姓　　名 王钧涛

专　　业 计算机科学与技术

授课老师 郭玉臣

目录

一、实验内容

1.1 项目简述

本实验是基于 SD 卡的数字照像机系统综合实验。

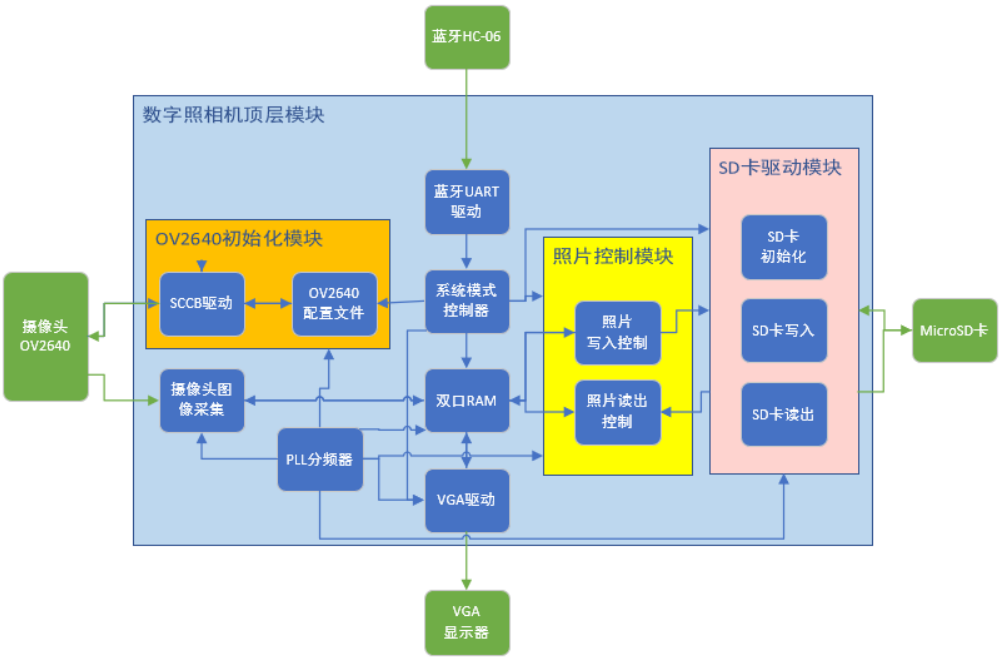
1.2 项目说明

本实验系统的主要功能是通过摄像头采集现实图像，并且能够在蓝牙信号的控制下完成图像的拍摄、存储入 SD 卡和读取显示存储在 SD 卡中的拍摄照片。

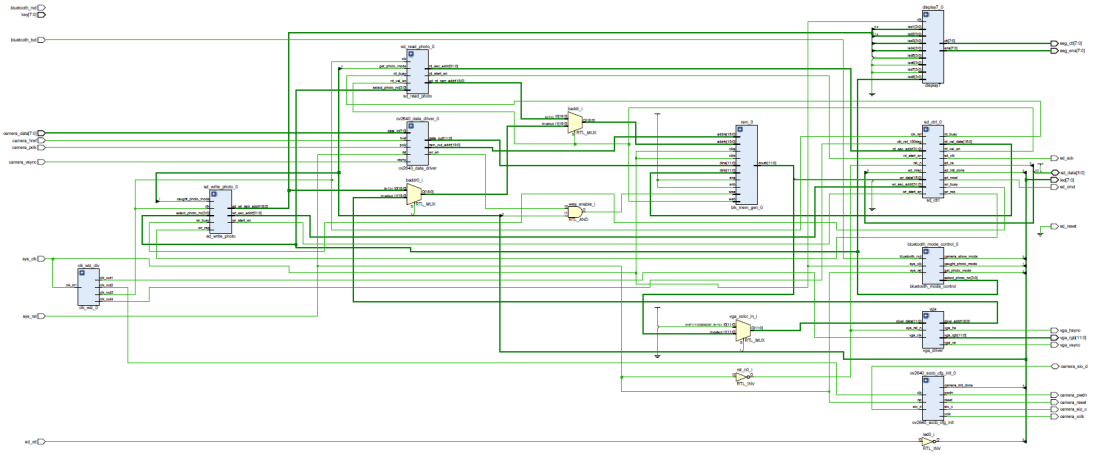
二、系统框图

1、 总系统设计图

总系统框图：



总系统 RTL 设计图：



2、 子系统功能与实现

(1) 系统模式控制子系统：

负责控制系统取景、照相、显示等模式状态和照片存储地址，并且支持通过蓝牙指令切换系统模式。

(2) OV2640 初始化模块子系统：

负责在系统初始化时通过 SCCB 协议将摄像头的寄存器配置文件写入到 OV2640 摄像头模块中。

(3) 双口 RAM 子系统：

负责存储等待显示在 VGA 显示器上的文件，并且负责接受摄像头和 SD 卡的照片输入。

(4) PLL 分频器子系统：

负责将系统的 100Mhz 时钟拆分成不同模块所要求的不同频率的时钟

(5) OV2640 图像采集子系统：

负责将 OV2640 摄像头回传的图像数据打包成 4096 色像素，并且负责发送信号告知 RAM 子系统。

(6) VGA 驱动子系统：

负责将输入的 12 位 4096 色像素写入到屏幕的指定位置中，并且反馈给 RAM 所需要输入的像素地址。

(7) 照片读写控制子系统：

负责控制整体照片写入到 SD 卡中和从 SD 卡中读取照片的位置和时序逻辑,在模式选线的上升沿触发。

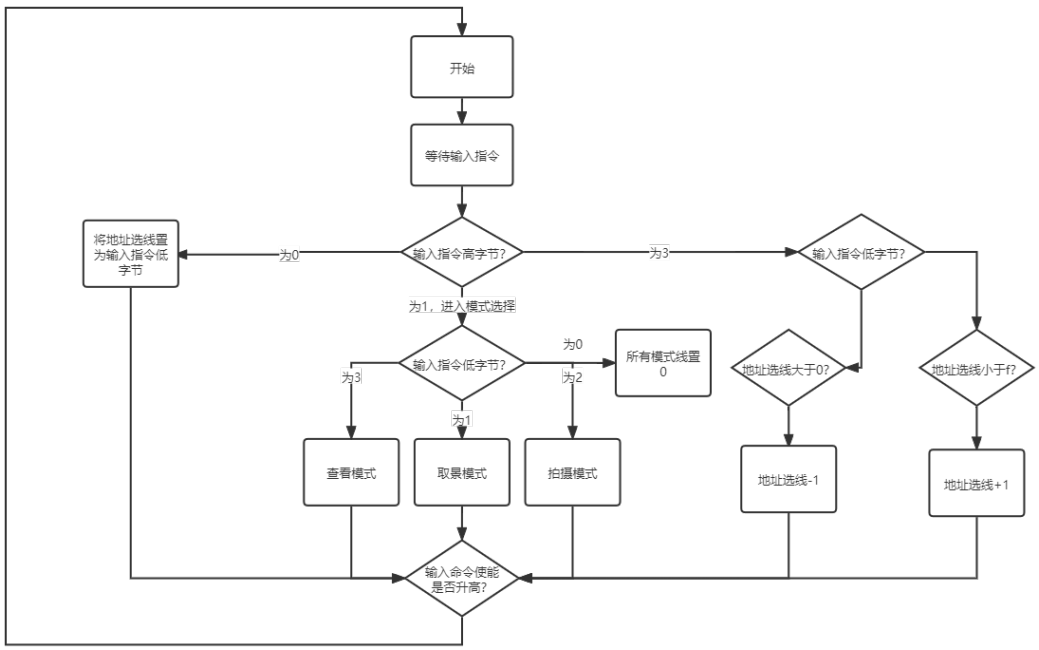
(8) **SD 卡读写控制子系统：**

负责在系统上电时初始化 SD 卡配置，与 SD 卡之间通过 SPI 协议通信，负责将照片读写控制子系统中传入的照片像素信息写入到 SD 卡之中。

三、系统控制器设计

（要求画出所设计数字系统的 ASM 流程图，列出状态转移真值表。由状态转移真值表，求出系统控制器的次态激励函数表达式和控制命令逻辑表达式，并用 Logisim 画出系统控制器逻辑方案图。）

ASM 流程图：



状态转移真值表：

模式选线 MODE[2:0]	次态	输入 CMD[7:0]
3'bxxx	0b000	0x10
3'bxxx	0b001	0x11
3'bxxx	0b010	0x12
3'bxxx	0b100	0x13
3'byyy	0byyy	其余

数据地址选线 NO[3:0]	次态	输入 CMD
4'b0000	4'b0001	0x30
4'b0001	4'b0010	0x30
4'b0010	4'b0011	0x30
4'b0011	4'b0100	0x30
4'b0100	4'b0101	0x30
4'b0101	4'b0110	0x30
4'b0110	4'b0111	0x30
4'b0111	4'b1000	0x30
4'b1000	4'b1001	0x30
4'b1001	4'b1010	0x30
4'b1010	4'b1011	0x30

4'b1011	4'b1100	0x30
4'b1100	4'b1101	0x30
4'b1101	4'b1110	0x30
4'b1110	4'b1111	0x30
4'b1111	4'b1111	0x30
4'b0000	4'b0000	0x31
4'b0001	4'b0000	0x31
4'b0010	4'b0001	0x31
4'b0011	4'b0010	0x31
4'b0100	4'b0011	0x31
4'b0101	4'b0100	0x31
4'b0110	4'b0101	0x31
4'b0111	4'b0110	0x31
4'b1000	4'b0111	0x31
4'b1001	4'b1000	0x31
4'b1010	4'b1001	0x31
4'b1011	4'b1010	0x31
4'b1100	4'b1011	0x31
4'b1101	4'b1100	0x31
4'b1110	4'b1101	0x31
4'b1111	4'b1110	0x31
4'bxxxx	4'b0000	0x10
4'bxxxx	4'b0001	0x11
4'bxxxx	4'b0010	0x12
4'bxxxx	4'b0011	0x13
4'bxxxx	4'b0100	0x14
4'bxxxx	4'b0101	0x15
4'bxxxx	4'b0110	0x16
4'bxxxx	4'b0111	0x17
4'bxxxx	4'b1000	0x18
4'bxxxx	4'b1001	0x19
4'bxxxx	4'b1010	0x1a
4'bxxxx	4'b1011	0x1b
4'bxxxx	4'b1100	0x1c
4'bxxxx	4'b1101	0x1d
4'bxxxx	4'b1110	0x1e
4'bxxxx	4'b1111	0x1f

次态激励表达式：

Ctrl_Mode[0] = CMD[4] & ~CMD[5] & ~CMD[0] & CMD[1]

Ctrl_Mode[1] = CMD[4] & ~CMD[5] & CMD[0] & ~CMD[1]

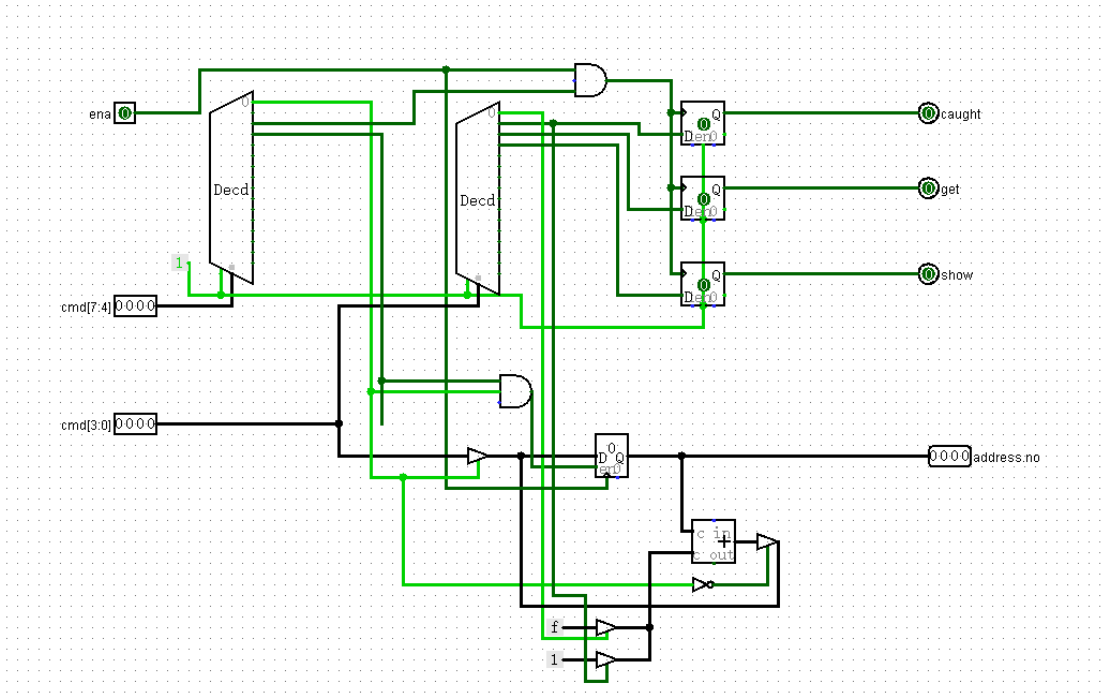
Ctrl_Mode[2] = CMD[4] & ~CMD[5] & CMD[0] & CMD[1]

Address_Mode[3:0] = CMD[4] & CMD[5] & CMD[0] & (Address_Mode[3:0] - 1) | CMD[4] & CMD[5] & ~CMD[0] & (Address_Mode[3:0] + 1) | ~CMD[4] & ~CMD[5] & CMD[3:0]

控制命令表达式：

控制命令采用独热码编码，故命令即为寄存器状态，故没有表达式给出。

Logisim 电路图：



四、子系统模块建模

本系统共划分为 16 个模块，分别是：

VGA 控制模块，蓝牙接受模块，系统控制模块，双口 RAM 模块，时钟分频模块，摄像头寄存器配置模块，SCCB 协议驱动模块，摄像头寄存器初始化模块，摄像头数据处理模块，照片读取模块，照片写入模块，SD 卡控制模块，SD 卡初始化模块，SD 卡读取模块，SD 卡写入模块，七段数码管显示模块。

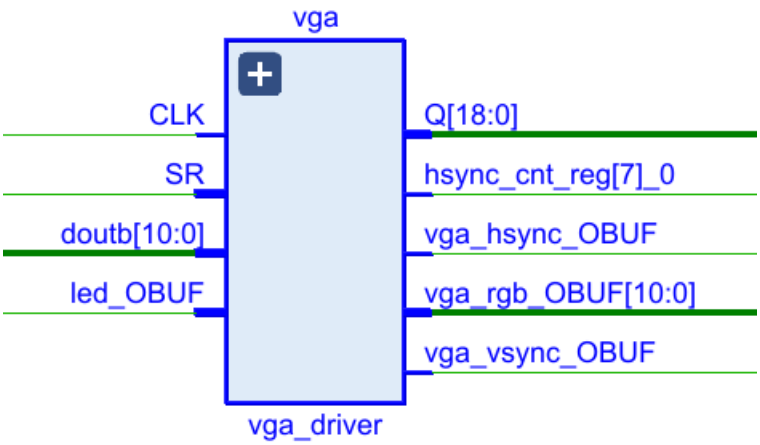
以下分别介绍各模块的建模和功能描述。

VGA 控制模块：

（1） 描述：

VGA 控制模块负责制造场同步信号 VSYNC 和行同步信号 HSYNC 以及将输入像素串入 VGA 显示屏，并且反馈给 RAM 所需要输入的像素地址。

（2） 功能框图：



（3） 接口定义：

```
(4)  module vga_driver(
(5)      input               vga_clk         ,//时钟信号
(6)      input               sys_rst_n       ,//系统复位
(7)      output              vga_hs          ,//行同步信号
(8)      output              vga_vs          ,//场同步信号
(9)      output      [11:0]  vga_rgb         ,//RGA 信号
(10)     output              data_req        ,//数据使能信号
(11)     input       [11:0]  pixel_data      ,//像素数据输入
(12)     output reg  [18:0]  pixel_addr       //像素数据地址
     );
```

4 Verilog 代码：

```
(13) module vga_driver(
(14)     input               vga_clk         ,
(15)     input               sys_rst_n       ,
(16)     output              vga_hs          ,
(17)     output              vga_vs          ,
(18)     output      [11:0]  vga_rgb         ,
(19)     output              data_req        ,
(20)     input       [11:0]  pixel_data      ,
(21)     output reg  [18:0]  pixel_addr
(22) );
(23)
(24)     /*************************************************************
     **
(25)         参数定义
(26)     *************************************************************
     **/
(27)     parameter  H_SYNC   =  10'd96   ;   //行同步
```

```
(28)     parameter  H_BACK   =  10'd48   ;   //行显示后沿
(29)     parameter  H_DISP   =  10'd640  ;   //行有效数据
(30)     parameter  H_FRONT  =  10'd16   ;   //行显示前沿
(31)     parameter  H_TOTAL  =  10'd800  ;   //行扫描周期
(32)
(33)     parameter  V_SYNC   =  10'd2    ;   //场同步
(34)     parameter  V_BACK   =  10'd33   ;   //场显示后沿
(35)     parameter  V_DISP   =  10'd480  ;   //场有效数据
(36)     parameter  V_FRONT  =  10'd10   ;   //场显示前沿
(37)     parameter  V_TOTAL  =  10'd525  ;   //场扫描周期
(38)
(39)     /************************************************************
     **
(40)         线网与寄存器定义
(41)     ************************************************************
     **/
(42)     reg  [9:0]          hsync_cnt       ;
(43)     reg  [9:0]          vsync_cnt       ;
(44)     wire                vga_en          ;
(45)     wire                data_req        ;
(46)

(47)     /************************************************************
     **
(48)         线网连接
(49)     ************************************************************
     **/
(50)     assign vga_hs  = (hsync_cnt <= H_SYNC - 1'b1) ? 1'b0 : 1'b1;    //
     行同步信号
(51)     assign vga_vs  = (vsync_cnt <= V_SYNC - 1'b1) ? 1'b0 : 1'b1;    //
     场同步信号
(52)     assign vga_en  = (((hsync_cnt >= H_SYNC+H_BACK) && (hsync_cnt <
     H_SYNC+H_BACK+H_DISP))
(53)                      &&((vsync_cnt >= V_SYNC+V_BACK) && (vsync_cnt <
     V_SYNC+V_BACK+V_DISP)))
(54)                      ?  1'b1 :
     1'b0;                                  //VGA 输出信号
(55)     assign vga_rgb = vga_en ? pixel_data :
     12'd0;                    //RGB444 输出信号
(56)     assign data_req = (((hsync_cnt >= H_SYNC+H_BACK-1'b1) &&
     (hsync_cnt < H_SYNC+H_BACK+H_DISP-1'b1))
(57)                      && ((vsync_cnt >= V_SYNC+V_BACK) && (vsync_cnt <
     V_SYNC+V_BACK+V_DISP)))
(58)                      ?  1'b1 :
     1'b0;                                  //像素点请求信号
```

```
(59)
(60)    /*************************************************************
        **
(61)        像素点地址信号
(62)    *************************************************************
        **/
(63)    always @(posedge vga_clk or negedge sys_rst_n) begin
(64)        if (!sys_rst_n)
(65)            pixel_addr <= 0;
(66)        else if(data_req)
(67)            pixel_addr <= (hsync_cnt - (H_SYNC + H_BACK - 1'b1)) +
        H_DISP * (vsync_cnt - (V_SYNC + V_BACK - 1'b1));
(68)        else
(69)            pixel_addr <= 0;
(70)    end
(71)
(72)    /*************************************************************
        **
(73)        行同步计数器
(74)    *************************************************************
        **/
(75)    always @(posedge vga_clk or negedge sys_rst_n) begin
(76)        if (!sys_rst_n)
(77)            hsync_cnt <= 10'd0;
(78)        else begin
(79)            if(hsync_cnt < H_TOTAL -
        1'b1)
(80)                hsync_cnt <= hsync_cnt +
        1'b1;
(81)            else
(82)                hsync_cnt <= 10'd0;
(83)        end
(84)    end
(85)
(86)    /*************************************************************
        **
(87)        场同步计数器
(88)    *************************************************************
        **/
(89)    always @(posedge vga_clk or negedge sys_rst_n) begin
(90)        if (!sys_rst_n)
(91)            vsync_cnt <= 10'd0;
(92)        else if(hsync_cnt == H_TOTAL - 1'b1) begin
```

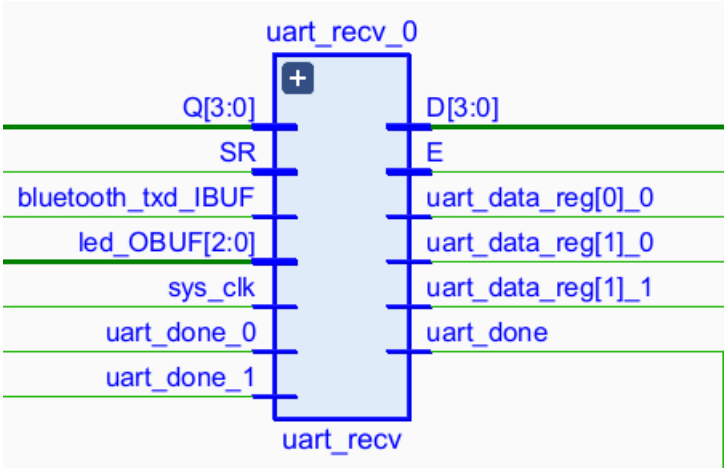
```
(93)            if(vsync_cnt < V_TOTAL -
     1'b1)
(94)                vsync_cnt <= vsync_cnt +
     1'b1;
(95)            else
(96)                vsync_cnt <= 10'd0;
(97)        end
(98)    end
(99)
(100) endmodule
```

蓝牙接受模块：

(1) 描述：

蓝牙接受模块通过 UART 串口协议与蓝牙模块通信，负责将输入的串口数据转成 8 位并行数据，并且通过 uart_done 给出传输完成信号

(2) 功能框图：



(3) 接口定义：

```
(4)module uart_recv(
(5)    input              sys_clk     ,   //系统时钟
(6)    input              sys_rst_n   ,   //系统复位
(7)    input              uart_rxd    ,   //uart 数据 RXD 线
(8)    output reg         uart_done   ,   //接受成功后拉高
(9)    output reg [7:0]   uart_data       //UART 数据输出
);
```

4　Verilog 代码：

```
(10)   module uart_recv(
(11)       input              sys_clk     ,
(12)       input              sys_rst_n   ,
(13)       input              uart_rxd    ,
```

```
(14)        output reg          uart_done   ,   //接受成功后拉高
(15)        output reg [7:0]    uart_data
(16)        );
(17)
(18)        /*************************************************************
   ***
(19)        参数定义
(20)        *************************************************************
   ***/
(21)        parameter  CLK_FREQ = 100000000;            //系统时钟频率
(22)        parameter  UART_BPS = 9600;                 //串口波特率
(23)        localparam BPS_CNT  = CLK_FREQ/UART_BPS;
(24)
(25)        /*************************************************************
   ***
(26)        线网与寄存器定义
(27)        *************************************************************
   ***/
(28)        reg            uart_rxd_0;
(29)        reg            uart_rxd_1;
(30)        reg [15:0]     clk_cnt;
(31)        reg [3:0]      rx_cnt;
(32)        reg            rx_flag;
(33)        reg [7:0]      rx_data;
(34)

(35)        /*************************************************************
   ***
(36)        uart_rxd 打拍、获取传输开始信号
(37)        *************************************************************
   ***/
(38)        wire       start_flag;
(39)        assign  start_flag = uart_rxd_1 & (~uart_rxd_0);
(40)        always @(posedge sys_clk or negedge sys_rst_n) begin
(41)            if (!sys_rst_n) begin
(42)                uart_rxd_0 <= 1'b0;
(43)                uart_rxd_1 <= 1'b0;
(44)            end
(45)            else begin
(46)                uart_rxd_0  <= uart_rxd;
(47)                uart_rxd_1  <= uart_rxd_0;
(48)            end
(49)        end
(50)
```

```
(51)        /*************************************************************
    ***
(52)        开始接受、设置接受状态信号
(53)        *************************************************************
    ***/
(54)        always @(posedge sys_clk or negedge sys_rst_n) begin
(55)            if (!sys_rst_n)
(56)                rx_flag <= 1'b0;
(57)            else begin
(58)                if(start_flag)
(59)                    rx_flag <= 1'b1;
(60)                else if((rx_cnt == 4'd9)&&(clk_cnt == BPS_CNT/2))
(61)                    rx_flag <= 1'b0;
(62)                else
(63)                    rx_flag <= rx_flag;
(64)            end
(65)        end
(66)
(67)        /*************************************************************
    ***
(68)        根据波特率产生对应时钟
(69)        *************************************************************
    ***/
(70)        always @(posedge sys_clk or negedge sys_rst_n) begin
(71)            if (!sys_rst_n) begin
(72)                clk_cnt <= 16'd0;
(73)                rx_cnt  <= 4'd0;
(74)            end else if ( rx_flag ) begin
(75)                    if (clk_cnt < BPS_CNT - 1) begin
(76)                        clk_cnt <= clk_cnt + 1'b1;
(77)                        rx_cnt  <= rx_cnt;
(78)                    end else begin
(79)                        clk_cnt <= 16'd0;
(80)                        rx_cnt  <= rx_cnt + 1'b1;
(81)                    end
(82)                end else begin
(83)                    clk_cnt <= 16'd0;
(84)                    rx_cnt  <= 4'd0;
(85)                end
(86)        end
(87)
(88)        /*************************************************************
    ***
(89)        数据接收过程
```

```
(90)        ************************************************************
   ***/
(91)        always @(posedge sys_clk or negedge sys_rst_n) begin
(92)            if ( !sys_rst_n)
(93)                rx_data <= 8'd0;
(94)            else if(rx_flag)
(95)                if (clk_cnt == BPS_CNT/2) begin
(96)                    case ( rx_cnt )
(97)                    4'd1 : rx_data[0] <= uart_rxd_1;
(98)                    4'd2 : rx_data[1] <= uart_rxd_1;
(99)                    4'd3 : rx_data[2] <= uart_rxd_1;
(100)                   4'd4 : rx_data[3] <= uart_rxd_1;
(101)                   4'd5 : rx_data[4] <= uart_rxd_1;
(102)                   4'd6 : rx_data[5] <= uart_rxd_1;
(103)                   4'd7 : rx_data[6] <= uart_rxd_1;
(104)                   4'd8 : rx_data[7] <= uart_rxd_1;
(105)                   default:;
(106)                   endcase
(107)               end
(108)               else
(109)                   rx_data <= rx_data;
(110)           else
(111)               rx_data <= 8'd0;
(112)       end
(113)
(114)       /************************************************************
   ***
(115)       数据接受完毕，准备收尾
(116)       ************************************************************
   ***/
(117)       always @(posedge sys_clk or negedge sys_rst_n) begin
(118)           if (!sys_rst_n) begin
(119)               uart_data <= 8'd0;
(120)               uart_done <= 1'b0;
(121)           end
(122)           else if(rx_cnt == 4'd9) begin
(123)               uart_data <= rx_data;
(124)               uart_done <= 1'b1;
(125)           end
(126)           else begin
(127)               uart_data <= 8'd0;
(128)               uart_done <= 1'b0;
(129)           end
(130)       end
```

```
(131)  endmodule
(132)
(133)  module uart_send(
(134)      input              sys_clk     ,
(135)      input              sys_rst_n   ,
(136)      input              uart_en     ,
(137)      input  [7:0]       uart_din    ,
(138)      output  reg        uart_txd
(139)      );
(140)
(141)      /*************************************************************
   ***
(142)      参数定义
(143)      *************************************************************
   ***/
(144)      parameter  CLK_FREQ = 100000000;
(145)      parameter  UART_BPS = 9600;
(146)      localparam BPS_CNT  = CLK_FREQ/UART_BPS;
(147)
(148)      /*************************************************************
   ***
(149)      线网与寄存器定义
(150)      *************************************************************
   ***/
(151)
(152)      reg [15:0]      clk_cnt;
(153)      reg [ 3:0]      tx_cnt;//发送数据计数器
(154)      reg             tx_flag;//发送过程标志信号
(155)      reg [ 7:0]      tx_data;
(156)      wire            en_flag;
(157)
(158)      /*************************************************************
   ***
(159)      uart_en 打拍，获取开始使能上升沿
(160)      *************************************************************
   ***/
(161)      reg             uart_en_0;
(162)      reg             uart_en_1;
(163)      assign en_flag = (~uart_en_1) &
   uart_en_0;
(164)      always @(posedge sys_clk or negedge sys_rst_n) begin
(165)          if (!sys_rst_n) begin
(166)              uart_en_0 <= 1'b0;
(167)              uart_en_1 <= 1'b0;
```

```
(168)           end else begin
(169)               uart_en_0 <= uart_en;
(170)               uart_en_1 <= uart_en_0;
(171)           end
(172)       end
(173)
(174)       /*************************************************************
    ***
(175)       发送中相关发送信号控制
(176)       *************************************************************
    ***/
(177)       always @(posedge sys_clk or negedge sys_rst_n) begin
(178)           if (!sys_rst_n) begin
(179)               tx_flag <= 1'b0;
(180)               tx_data <= 8'd0;
(181)           end else if (en_flag) begin
(182)               tx_flag <= 1'b1;
(183)               tx_data <= uart_din;
(184)           end else if ((tx_cnt == 4'd9)&&(clk_cnt == BPS_CNT/2)) begin
(185)               tx_flag <= 1'b0;
(186)               tx_data <= 8'd0;
(187)           end else begin
(188)               tx_flag <= tx_flag;
(189)               tx_data <= tx_data;
(190)           end
(191)       end
(192)
(193)       /*************************************************************
    ***
(194)       进入发送过程，根据波特率设置对应时钟
(195)       *************************************************************
    ***/
(196)       always @(posedge sys_clk or negedge sys_rst_n) begin
(197)           if (!sys_rst_n) begin
(198)               clk_cnt <= 16'd0;
(199)               tx_cnt  <= 4'd0;
(200)           end  else if (tx_flag) begin
(201)               if (clk_cnt < BPS_CNT - 1) begin
(202)                   clk_cnt <= clk_cnt + 1'b1;
(203)                   tx_cnt  <= tx_cnt;
(204)               end else begin
(205)                   clk_cnt <= 16'd0;
(206)                   tx_cnt  <= tx_cnt + 1'b1;
(207)               end
```

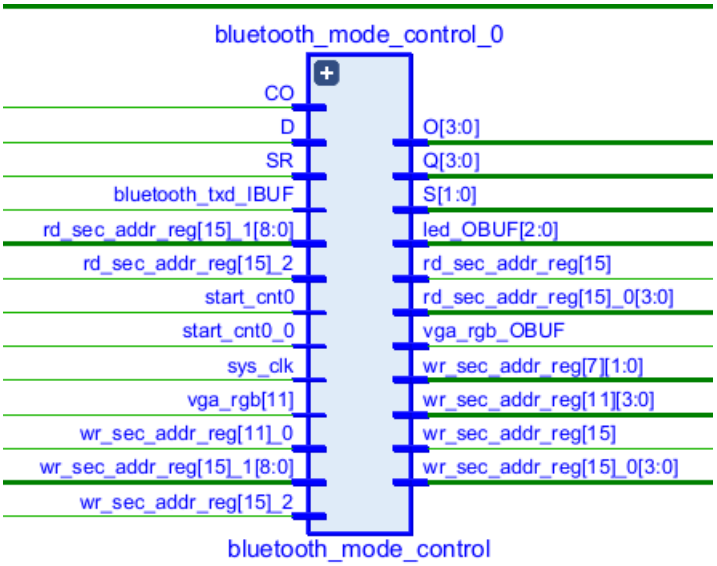
```
(208)           end else begin //发送过程结束
(209)               clk_cnt <= 16'd0;
(210)               tx_cnt  <= 4'd0;
(211)           end
(212)       end
(213)
(214)       /*************************************************************
    ***
(215)       数据发送过程
(216)       *************************************************************
    ***/
(217)       always @(posedge sys_clk or negedge sys_rst_n) begin
(218)           if (!sys_rst_n)
(219)               uart_txd <= 1'b1;
(220)           else if (tx_flag)
(221)               case(tx_cnt)
(222)                   4'd0: uart_txd <= 1'b0;
(223)                   4'd1: uart_txd <= tx_data[0];
(224)                   4'd2: uart_txd <= tx_data[1];
(225)                   4'd3: uart_txd <= tx_data[2];
(226)                   4'd4: uart_txd <= tx_data[3];
(227)                   4'd5: uart_txd <= tx_data[4];
(228)                   4'd6: uart_txd <= tx_data[5];
(229)                   4'd7: uart_txd <= tx_data[6];
(230)                   4'd8: uart_txd <= tx_data[7];
(231)                   4'd9: uart_txd <= 1'b1;
(232)                   default: ;
(233)               endcase
(234)           else
(235)               uart_txd <= 1'b1;
(236)       end
(237)   endmodule
```

系统控制模块：

（1） 描述：

系统控制模块负责将蓝牙接受模块接受到的控制指令进行译码，并且切换相应的系统模式。

（2） 功能框图：



（3） 接口定义：

```
(4)  module bluetooth_mode_control(
(5)      input               sys_clk             ,   //系统时钟
(6)      input               sys_rst             ,   //系统复位
(7)      input               bluetooth_rxd       ,   //蓝牙 RXD
(8)      output reg          get_photo_mode      ,   //读取模式
(9)      output reg          caught_photo_mode   ,   //拍摄模式
(10)     output reg          camera_show_mode    ,   //取景模式
(11)     output reg [3:0]    select_photo_no         //当前照片位选择地址
(12) );
```

4 Verilog 代码：

```
(13) module bluetooth_mode_control(
(14)     input               sys_clk             ,
(15)     input               sys_rst             ,
(16)     input               bluetooth_rxd       ,   //蓝牙 RXD
(17)     output reg          get_photo_mode      ,   //读取模式
(18)     output reg          caught_photo_mode   ,   //拍摄模式
(19)     output reg          camera_show_mode    ,   //取景模式
(20)     output reg [3:0]    select_photo_no         //当前照片位选择地址
(21) );
(22)     /*************************************************************
     **
(23)     线网与寄存器定义
(24)     *************************************************************
     **/
(25)     wire                uart_done;
(26)     wire [7:0]          uart_data;
(27)
```

```
(28)     /*************************************************************
     **
(29)     寄存器初始化
(30)     *************************************************************
     **/
(31)     initial begin
(32)         get_photo_mode <= 0;
(33)         caught_photo_mode <= 0;
(34)         camera_show_mode <= 1;
(35)         select_photo_no <= 4'b0;
(36)     end
(37)     /*************************************************************
     **
(38)     蓝牙接受器实例化
(39)     *************************************************************
     **/
(40)     uart_recv uart_recv_0(
(41)         .sys_clk        (sys_clk),
(42)         .sys_rst_n      (~sys_rst),
(43)         .uart_rxd       (bluetooth_rxd),
(44)         .uart_done      (uart_done),
(45)         .uart_data      (uart_data)
(46)     );
(47)
(48)     /*************************************************************
     **
(49)     uart_done 打拍获取上升沿
(50)     *************************************************************
     **/
(51)     reg uart_done_0, uart_done_1;
(52)     wire uart_done_ena;
(53)     assign uart_done_ena = uart_done_0 & ~uart_done_1;
(54)     always @(posedge sys_clk) begin
(55)         uart_done_0 <= uart_done;
(56)         uart_done_1 <= uart_done_0;
(57)     end
(58)
(59)     /*************************************************************
     **
(60)     收到蓝牙命令后的模式切换
(61)     *************************************************************
     **/
(62)     always @(posedge sys_clk) begin
(63)         if(uart_done_ena) begin
```

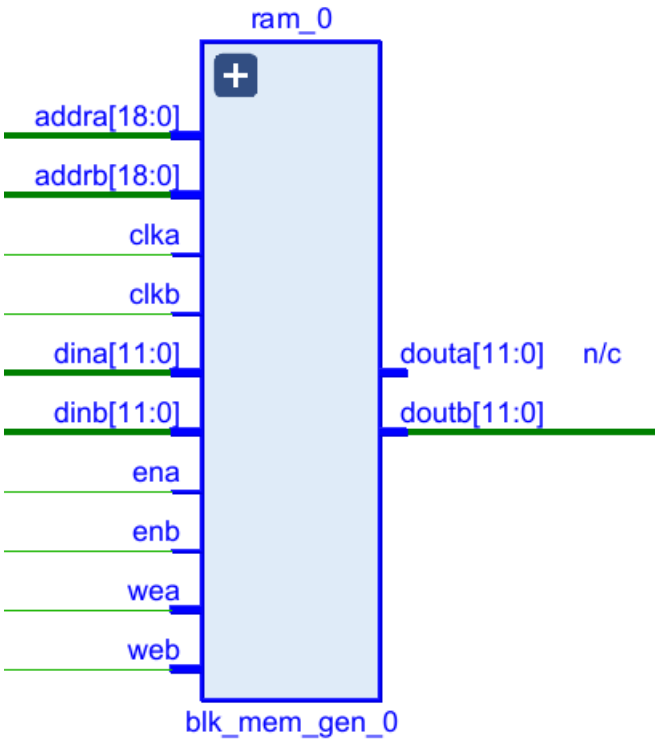
```
(64)            if(uart_data[7:4] == 4'h0) begin                //0x0X,
    代表选择 X 号照片位置
(65)                select_photo_no <= uart_data[3:0];
(66)            end else if(uart_data[7:4] == 4'h1) begin       //0x1X,
    模式选择模式
(67)            case(uart_data[3:0])
(68)                4'h0: begin
(69)                    get_photo_mode <= 0;
(70)                    caught_photo_mode <= 0;
(71)                    camera_show_mode <= 0;
(72)                end
(73)                4'h1: begin
(74)                    get_photo_mode <= 1;
(75)                    caught_photo_mode <= 0;
(76)                    camera_show_mode <= 0;
(77)                end
(78)                4'h2: begin
(79)                    get_photo_mode <= 0;
(80)                    caught_photo_mode <= 1;
(81)                    camera_show_mode <= 0;
(82)                end
(83)                4'h3: begin
(84)                    get_photo_mode <= 0;
(85)                    caught_photo_mode <= 0;
(86)                    camera_show_mode <= 1;
(87)                end
(88)            endcase
(89)            end else if(uart_data[7:4] == 4'h3) begin       //0x3X,
    照片位 1 位 1 位切换
(90)            case(uart_data[3:0])
(91)                4'h0: begin
(92)                    if(select_photo_no < 4'hf)
(93)                        select_photo_no <= select_photo_no + 1;
(94)                end
(95)                4'h1: begin
(96)                    if(select_photo_no > 4'h0)
(97)                        select_photo_no <= select_photo_no - 1;
(98)                end
(99)            endcase
(100)            end
(101)        end
(102)    end
(103)
(104) endmodule
```

双口 RAM 模块：

（1） 描述：

双口 RAM 模块采用 IP 核的形式，A 口和 B 口都同时支持写入和读取操作，用于存储摄像头和 SD 卡写入的图像数据并且提供给 VGA 显示器访问。

（2） 功能框图：



（3） 接口定义：

```
（4）     blk_mem_gen_0 ram_0(
（5）         .clka               (sys_clk)              ,//A 口时钟
（6）         .ena                (1'b1)                 ,//A 口使能
（7）         .wea                (wea_enable)           ,//A 口写入
（8）         .addra              (ram_addr)             ,//A 口地址
（9）         .dina               (ram_data)             ,//A 口数据
（10）        .clkb               (sys_clk)              ,//B 口时钟
（11）        .enb                (1'b1)                 ,//B 口使能
（12）        .addrb              (baddr)                ,//B 口地址
（13）        .dinb               (rd_val_data[11:0])     ,//B 口输入
（14）        .doutb              (doutb)                ,//B 口输出
（15）        .web                (rd_val_en)              //B 口写入控制
（16）    );
```

4　Verilog 代码：

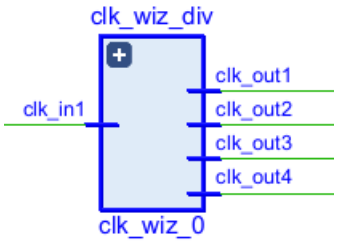
```
（17）     blk_mem_gen_0 ram_0(
（18）         .clka               (sys_clk)              ,
（19）         .ena                (1'b1)                 ,
（20）         .wea                (wea_enable)           ,
（21）         .addra              (ram_addr)             ,
（22）         .dina               (ram_data)             ,
（23）         .clkb               (sys_clk)              ,
（24）         .enb                (1'b1)                 ,
（25）         .addrb              (baddr)                ,
（26）         .dinb               (rd_val_data[11:0])     ,
（27）         .doutb              (doutb)                ,
（28）         .web                (rd_val_en)
（29）     );
```

时钟分频模块：

（1） 描述：

（2） 时钟分频模块采用 PLL 锁相环方式，将输入的 100Mhz 时钟分出供其他模块使用的时钟

（3） 功能框图：



（3） 接口定义：

```
clk_wiz_0 clk_wiz_div(
       .clk_in1           (sys_clk)              ,//系统时钟
       .clk_out1          (clk_vga_24m)          ,//A 输出
       .clk_out2          (clk_sccb_init_25m)     ,//B 输出
       .clk_out3          (clk_sd_50m)           ,//C 输出
       .clk_out4          (clk_sd_50m_180deg)     //D 输出
   );
```

4　Verilog 代码：

```
（4）   clk_wiz_0 clk_wiz_div(
```

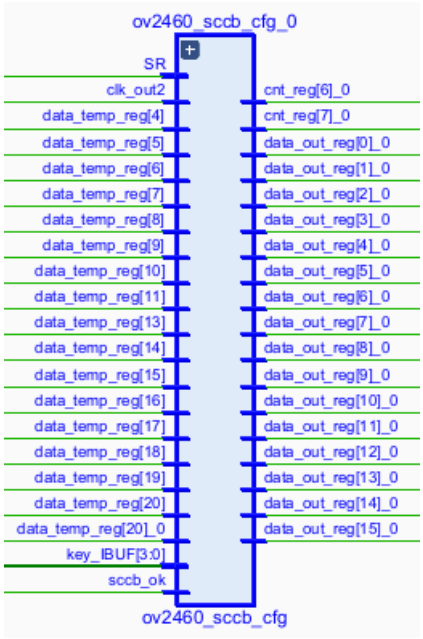
```
(5)          .clk_in1            (sys_clk)               ,
(6)          .clk_out1           (clk_vga_24m)           ,
(7)          .clk_out2           (clk_sccb_init_25m)     ,
(8)          .clk_out3           (clk_sd_50m)            ,
(9)          .clk_out4           (clk_sd_50m_180deg)
(10)     );
```

摄像头寄存器配置模块：

1 描述：

摄像头寄存器配置模块负责维护摄像头寄存器信息，在上电的同时初始化计数器，在 SCCB 空闲的上升沿写入配置信息。

2 功能框图：



3 接口定义：

```
module ov2460_sccb_cfg(
    input               clk             ,//系统时钟
    input               rst             ,//系统复位
    output reg [15:0]   data_out        ,//配置数据输出
    output              cfg_ok          ,//配置写入完毕会拉高
    input               sccb_ok         ,//SCCB 是否能够读写
    input      [3:0]    bright          //亮度调节
);
```

4 Verilog 代码：

```
module ov2460_sccb_cfg(
    input               clk             ,
```

```
    input               rst             ,
    output reg [15:0]   data_out        ,
    output              cfg_ok          ,//配置写入完毕会拉高
    input               sccb_ok         ,
    input      [3:0]    bright
);
    //亮度设置
    //0:(0X00)-2
    //1:(0X10)-1
    //2,(0X20) 0
    //3,(0X30)+1
    //4,(0X40)+2
    /*****************************************************************
        参数配置
    *****************************************************************/
    parameter cfg_number=181;

    /*****************************************************************
        寄存器定义
    *****************************************************************/
    reg [10:0] cnt=0;
    assign cfg_ok=cnt<=cfg_number;

    /*****************************************************************
        配置滚动
    *****************************************************************/
    always @ (posedge clk)
    begin
        if(rst)
            cnt<=0;
        else if(cfg_ok&&sccb_ok)
            cnt<=cnt+1;
    end

    /*****************************************************************
        初始化配置文件
        参考自 OmmVision Software Application Notes VGA 推荐配置
    *****************************************************************/
    always @ (posedge clk)
    case (cnt)
    8'h00:
    data_out <= 16'hFF01;
    8'h01 :
    data_out <= 16'h1280;
```

```
    8'h02 :
    data_out <= 16'hFF00;
    8'h03 :
    data_out <= 16'h2CFF;
    8'h04 :
    data_out <= 16'h2EDF;
    8'h05 :
    data_out <= 16'hFF01;
    8'h06 :
    data_out <= 16'h3C32;
    8'h07 :
    data_out <= 16'h1101;
    8'h08 :
    data_out <= 16'h0902;
    8'h09 :
    data_out <= 16'h0420;
    8'h0A :
    data_out <= 16'h13E5;
    8'h0B :
    data_out <= 16'h1448;
    8'h0C :
    data_out <= 16'h2C0C;
    8'h0D :
    data_out <= 16'h3378;
    8'h0E :
    data_out <= 16'h3A33;
    8'h0F :
    data_out <= 16'h3BFB;
    8'h10 :
    data_out <= 16'h3E00;
    8'h11 :
    data_out <= 16'h4311;
    8'h12 :
    data_out <= 16'h1610;
    8'h13 :
    data_out <= 16'h3992;
    8'h14 :
    data_out <= 16'h35DA;
    8'h15 :
    data_out <= 16'h221A;
    8'h16 :
    data_out <= 16'h37C3;
    8'h17 :
    data_out <= 16'h2300;
```

```
    8'h18 :
    data_out <= 16'h34C0;
    8'h19 :
    data_out <= 16'h361A;
    8'h1A :
    data_out <= 16'h0688;
    8'h1B :
    data_out <= 16'h07C0;
    8'h1C :
    data_out <= 16'h0D87;
    8'h1D :
    data_out <= 16'h0E41;
    8'h1E :
    data_out <= 16'h4C00;
    8'h1F :
    data_out <= 16'h4800;
    8'h20 :
    data_out <= 16'h5B00;
    8'h21 :
    data_out <= 16'h4203;
    8'h22 :
    data_out <= 16'h4A81;
    8'h23 :
    data_out <= 16'h2199;
    8'h24 :
    data_out <= 16'h2440;
    8'h25 :
    data_out <= 16'h2538;
    8'h26 :
    data_out <= 16'h2682;
    8'h27 :
    data_out <= 16'h5C00;
    8'h28 :
    data_out <= 16'h6300;
    8'h29 :
    data_out <= 16'h4600;
    8'h2A :
    data_out <= 16'h0C3C;
    8'h2B :
    data_out <= 16'h6170;
    8'h2C :
    data_out <= 16'h6280;
    8'h2D :
    data_out <= 16'h7C05;
```

```
    8'h2E :
    data_out <= 16'h2080;
    8'h2F :
    data_out <= 16'h2830;
    8'h30 :
    data_out <= 16'h6C00;
    8'h31 :
    data_out <= 16'h6D80;
    8'h32 :
    data_out <= 16'h6E00;
    8'h33 :
    data_out <= 16'h7002;
    8'h34 :
    data_out <= 16'h7194;
    8'h35 :
    data_out <= 16'h73C1;
    8'h36 :
    data_out <= 16'h1240;
    8'h37 :
    data_out <= 16'h1711;
    8'h38 :
    data_out <= 16'h1839;
    8'h39 :
    data_out <= 16'h1900;
    8'h3A :
    data_out <= 16'h1A3C;
    8'h3B :
    data_out <= 16'h3209;
    8'h3C :
    data_out <= 16'h37C0;
    8'h3D :
    data_out <= 16'h4FCA;
    8'h3E :
    data_out <= 16'h50A8;
    8'h3F :
    data_out <= 16'h5A23;
    8'h40 :
    data_out <= 16'h6D00;
    8'h41 :
    data_out <= 16'h3D38;
    8'h42 :
    data_out <= 16'hFF00;
    8'h43 :
    data_out <= 16'hE57F;
```

```
    8'h44 :
    data_out <= 16'hF9C0;
    8'h45 :
    data_out <= 16'h4124;
    8'h46 :
    data_out <= 16'hE014;
    8'h47 :
    data_out <= 16'h76FF;
    8'h48 :
    data_out <= 16'h33A0;
    8'h49 :
    data_out <= 16'h4220;
    8'h4A :
    data_out <= 16'h4318;
    8'h4B :
    data_out <= 16'h4C00;
    8'h4C :
    data_out <= 16'h87D5;
    8'h4D :
    data_out <= 16'h883F;
    8'h4E :
    data_out <= 16'hD703;
    8'h4F :
    data_out <= 16'hD910;
    8'h50 :
    data_out <= 16'hD382;
    8'h51 :
    data_out <= 16'hC808;
    8'h52 :
    data_out <= 16'hC980;
    8'h53 :
    data_out <= 16'h7C00;
    8'h54 :
    data_out <= 16'h7D00;
    8'h55 :
    data_out <= 16'h7C03;
    8'h56 :
    data_out <= 16'h7D48;
    8'h57 :
    data_out <= 16'h7D48;
    8'h58 :
    data_out <= 16'h7C08;
    8'h59 :
    data_out <= 16'h7D20;
```

```
    8'h5A :
    data_out <= 16'h7D10;
    8'h5B :
    data_out <= 16'h7D0E;
    8'h5C :
    data_out <= 16'h9000;
    8'h5D :
    data_out <= 16'h910E;
    8'h5E :
    data_out <= 16'h911A;
    8'h5F :
    data_out <= 16'h9131;
    8'h60 :
    data_out <= 16'h915A;
    8'h61 :
    data_out <= 16'h9169;
    8'h62 :
    data_out <= 16'h9175;
    8'h63 :
    data_out <= 16'h917E;
    8'h64 :
    data_out <= 16'h9188;
    8'h65 :
    data_out <= 16'h918F;
    8'h66 :
    data_out <= 16'h9196;
    8'h67 :
    data_out <= 16'h91A3;
    8'h68 :
    data_out <= 16'h91AF;
    8'h69 :
    data_out <= 16'h91C4;
    8'h6A :
    data_out <= 16'h91D7;
    8'h6B :
    data_out <= 16'h91E8;
    8'h6C :
    data_out <= 16'h9120;
    8'h6D :
    data_out <= 16'h9200;
    8'h6E :
    data_out <= 16'h9306;
    8'h6F :
    data_out <= 16'h93E3;
```

```
    8'h70 :
    data_out <= 16'h9305;
    8'h71 :
    data_out <= 16'h9305;
    8'h72 :
    data_out <= 16'h9300;
    8'h73 :
    data_out <= 16'h9304;
    8'h74 :
    data_out <= 16'h9300;
    8'h75 :
    data_out <= 16'h9300;
    8'h76 :
    data_out <= 16'h9300;
    8'h77 :
    data_out <= 16'h9300;
    8'h78 :
    data_out <= 16'h9300;
    8'h79 :
    data_out <= 16'h9300;
    8'h7A :
    data_out <= 16'h9300;

    8'h7B :
    data_out <= 16'h9600;
    8'h7C :
    data_out <= 16'h9708;
    8'h7D :
    data_out <= 16'h9719;
    8'h7E :
    data_out <= 16'h9702;
    8'h7F :
    data_out <= 16'h970C;
    8'h80 :
    data_out <= 16'h9724;
    8'h81 :
    data_out <= 16'h9730;
    8'h82 :
    data_out <= 16'h9728;
    8'h83 :
    data_out <= 16'h9726;
    8'h84 :
    data_out <= 16'h9702;
    8'h85 :
```

```
    data_out <= 16'h9798;
    8'h86 :
    data_out <= 16'h9780;
    8'h87 :
    data_out <= 16'h9700;
    8'h88 :
    data_out <= 16'h9700;
    8'h89 :
    data_out <= 16'hC3ED;
    8'h8A :
    data_out <= 16'hA400;
    8'h8B :
    data_out <= 16'hA800;
    8'h8C :
    data_out <= 16'hC511;
    8'h8D :
    data_out <= 16'hC651;
    8'h8E :
    data_out <= 16'hBF80;
    8'h8F :
    data_out <= 16'hC710;
    8'h90 :
    data_out <= 16'hB666;
    8'h91 :
    data_out <= 16'hB8A5;
    8'h92 :
    data_out <= 16'hB764;
    8'h93 :
    data_out <= 16'hB97C;
    8'h94 :
    data_out <= 16'hB3AF;
    8'h95 :
    data_out <= 16'hB497;
    8'h96 :
    data_out <= 16'hB5FF;
    8'h97 :
    data_out <= 16'hB0C5;
    8'h98 :
    data_out <= 16'hB194;
    8'h99 :
    data_out <= 16'hB20F;
    8'h9A :
    data_out <= 16'hC45C;
    8'h9B :
```

```
    data_out <= 16'hC050;
    8'h9C :
    data_out <= 16'hC13C;
    8'h9D :
    data_out <= 16'h8C00;
    8'h9E :
    data_out <= 16'h863D;
    8'h9F :
    data_out <= 16'h5000;
    8'hA0 :
    data_out <= 16'h51A0;
    8'hA1 :
    data_out <= 16'h5278;
    8'hA2 :
    data_out <= 16'h5300;
    8'hA3 :
    data_out <= 16'h5400;
    8'hA4 :
    data_out <= 16'h5500;
    8'hA5 :
    data_out <= 16'h5AA0;
    8'hA6 :
    data_out <= 16'h5B78;
    8'hA7 :
    data_out <= 16'h5C00;
    8'hA8 :
    data_out <= 16'hD382;
    8'hA9 :
    data_out <= 16'hC3ED;
    8'hAA :
    data_out <= 16'h7F00;
    8'hAB :
    data_out <= 16'hDA08;
    8'hAC :
    data_out <= 16'hE51F;
    8'hAD :
    data_out <= 16'hE167;
    8'hAE :
    data_out <= 16'hE000;
    8'hAF :
    data_out <= 16'hDD7F;
    8'hB0 :
    data_out <= 16'h0500;
    8'hB1 :
```

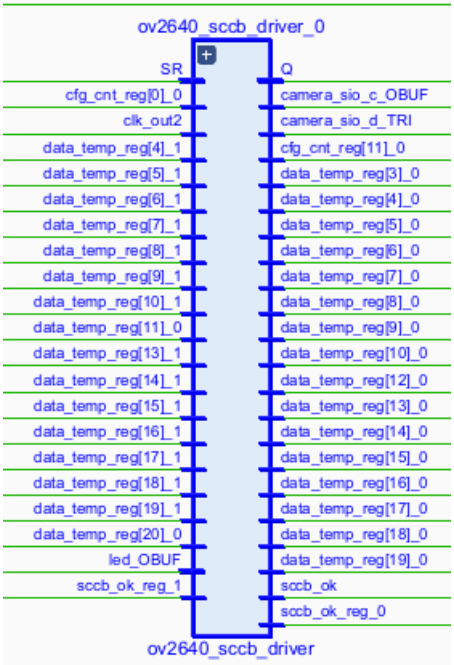
```
    data_out <= 16'hFF00;
    8'hB2 :
    data_out <= 16'h7C00;
    8'hBA3 :
    data_out <= 16'h7D04;
    8'hB4 :
    data_out <= 16'h7C09;
    8'hB5 :
    data_out <= {8'h7D, bright ,4'h0};
    endcase
endmodule
```

SCCB 协议驱动模块：

1 描述：

SCCB 协议驱动模块负责维护与摄像头之间的 SCCB 通信的基础连接，由于本实验系统并没有用到 SCCB 的反馈协议，因此没有写入读取 SCCB 返回数据的模块，也即该模块只能定向发送寄存器配置信息。

2 功能框图：



3 接口定义：

```
module ov2640_sccb_driver(
    input               clk         ,//系统时钟
    input               rst         ,//系统复位
    inout               sio_d       ,//通信数据线
    output reg          sio_c       ,//通信时序线
    input               cfg_ok      ,//寄存器配置写入使能
```

```
    output reg          sccb_ok     ,//返回的 SCCB 通信空闲使能
    input [7:0]         slave_id    ,//通信的 ID 号
    input [7:0]         cfg_addr    ,//配置文件地址
    input [7:0]         value        //等待写入的寄存器值
);
```

4 Verilog 代码：

```
module ov2640_sccb_driver(
    input               clk         ,
    input               rst         ,
    inout               sio_d       ,
    output reg          sio_c       ,
    input               cfg_ok      ,
    output reg          sccb_ok     ,
    input [7:0]         slave_id    ,
    input [7:0]         cfg_addr    ,
    input [7:0]         value
);

    /*****************************************************************
        寄存器定义与初始化
    *****************************************************************/
    reg  [20:0]         cfg_cnt = 0 ;
    reg                 sio_d_tmp   ;
    reg  [31:0]         data_temp   ;
    initial sccb_ok <= 0;

    /*****************************************************************
        配置文件计数器
    *****************************************************************/
    always @ (posedge clk)
    begin
        if(cfg_cnt == 0)
            cfg_cnt <= cfg_ok;
        else
            if(cfg_cnt[20:11] == 31)
                cfg_cnt <= 0;
            else
                cfg_cnt <= cfg_cnt + 1;
    end

    /*****************************************************************
        SCCB 初始化完成信号配置
    *****************************************************************/
```

```
    always @ (posedge clk)
        sccb_ok <= (cfg_cnt == 0) && (cfg_ok==1);

    /*************************************************************
        SIOC 串口通信信号配置
    *************************************************************/
    always @ (posedge clk) begin
        if(cfg_cnt[20:11] == 0)
            sio_c <= 1;
        else if(cfg_cnt[20:11]==1) begin
            if(cfg_cnt[10:9] == 2'b11)
                sio_c <= 0;
            else
                sio_c <= 1;
        end else if(cfg_cnt[20:11] == 29) begin
            if(cfg_cnt[10:9] == 2'b00)
                sio_c <= 0;
            else
                sio_c <= 1;
        end else if(cfg_cnt[20:11] == 30 || cfg_cnt[20:11] == 31)
            sio_c <= 1;
        else begin
            if(cfg_cnt[10:9] == 2'b00)
                sio_c <= 0;
            else if(cfg_cnt[10:9] == 2'b01)
                sio_c <= 1;
            else if(cfg_cnt[10:9] == 2'b01)
                sio_c <= 1;
            else if(cfg_cnt[10:9] == 2'b11)
                sio_c <= 0;
        end
    end

    /*************************************************************
        SIOD 串口通信信号输出
    *************************************************************/
    always @ (posedge clk) begin
        if(cfg_cnt[20:11] == 10 || cfg_cnt[20:11] == 19 || cfg_cnt[20:11]
== 28)
            sio_d_tmp <= 0;
        else
            sio_d_tmp <= 1;
    end
```

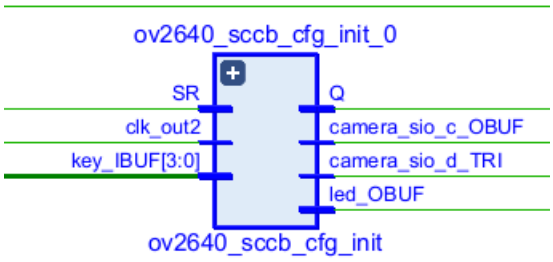
```
    /*************************************************************
        配置文件数据输出配置
    *************************************************************/
    always @ (posedge clk) begin
        if(rst)
            data_temp<=32'hffffffff;
        else
        begin
            if(cfg_cnt==0&&cfg_ok==1)
                data_temp<={2'b10,slave_id,1'bx,cfg_addr,1'bx,value,1'bx
,3'b011};
            else if(cfg_cnt[10:0]==0)
                data_temp<={data_temp[30:0],1'b1};
        end
    end
    assign sio_d=sio_d_tmp?data_temp[31]:'bz;//三态门控制

endmodule
```

摄像头寄存器初始化模块：

1 描述：

该模块负责组合摄像头寄存器配置文件模块和 SCCB 通信模块，在系统上电之后自动初始化寄存器配置，如果能初始化完成就拉高摄像头初始化使能，使得摄像头能够输出正常的信息。

2 功能框图：



3 接口定义：

```
module ov2640_sccb_cfg_init(
    input               clk         ,//摄像头输入时钟
    input               rst         ,//复位信号
    output              sio_c       ,//时序通信线
    inout               sio_d       ,//数据通信线
    output              reset       ,//摄像头初始化失败线，高电平有效
    output              pwdn        ,//pwdn 选线
```

```
    output              xclk        ,//外置晶振
    input       [3:0]   bright      ,//亮度选线
    output              camera_init_done//摄像头初始化完成线，高电平有效
);
```

4 Verilog 代码：

```
module ov2640_sccb_cfg_init(
    input               clk         ,
    input               rst         ,
    output              sio_c       ,
    inout               sio_d       ,
    output              reset       ,
    output              pwdn        ,
    output              xclk        ,
    input       [3:0]   bright      ,
    output              camera_init_done
);

    /*****************************************************************
        线网定义
    *****************************************************************/
    wire [15:0]         data_send   ;
    wire                cfg_ok      ;
    wire                sccb_ok     ;

    /*****************************************************************
        线网连接
    *****************************************************************/
    assign reset = 1;
    assign pwdn = 0;
    assign xclk = clk;
    pullup up (sio_d);

    /*****************************************************************
        配置文件模块实例化
    *****************************************************************/
    ov2460_sccb_cfg ov2460_sccb_cfg_0(
        .clk            (clk)           ,
        .rst            (rst)           ,
        .data_out       (data_send)     ,
        .cfg_ok         (cfg_ok)        ,
        .sccb_ok        (sccb_ok)       ,
        .bright         (bright)
    );
```

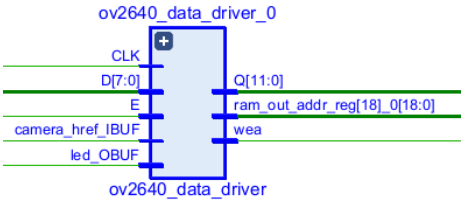
```
    /*************************************************************
        SCCB 总线驱动模块实例化
    *************************************************************/
    ov2640_sccb_driver ov2640_sccb_driver_0(
        .clk            (clk)           ,
        .rst            (rst)           ,
        .sio_d          (sio_d)         ,
        .sio_c          (sio_c)         ,
        .cfg_ok         (cfg_ok)        ,
        .sccb_ok        (sccb_ok)       ,
        .slave_id       (8'h60)         ,
        .cfg_addr       (data_send[15:8]),
        .value          (data_send[7:0])
    );

    /*************************************************************
        摄像头初始化完成信号打拍
    *************************************************************/
    reg camera_init_done_0;
    reg camera_init_done_1;
    assign camera_init_done = camera_init_done_0 & ~camera_init_done_1;
    always @(posedge clk) begin
        if(rst) begin
            camera_init_done_0 <= 0;
            camera_init_done_1 <= 0;
        end else if(~camera_init_done) begin
            camera_init_done_0 <= sccb_ok;
            camera_init_done_1 <= camera_init_done_0;
        end
    end

endmodule
```

摄像头数据处理模块：

1 描述：

摄像头数据处理模块负责接受摄像头传回的图像信息，并且将 8 位串行数据合并为 16 位的 RGB 像素数据，并且内置 RGB565 转 RGB444 功能。

2 功能框图：



3 接口定义：

```
module ov2640_data_driver(
    input               rst                 ,//重置，高电平有效
    input               pclk                ,//摄像头输入时钟
    input               href                ,//行同步线
    input               vsync               ,//场同步线
    input       [7:0]   data_in             ,//摄像头数据输入线
    output reg  [11:0]  data_out            ,//RGB 像素数据输出线
    output reg          wr_en               ,//数据完成写入使能
    output reg  [18:0]  ram_out_addr         //输出的地址
);
```

4 Verilog 代码：

```
module ov2640_data_driver(
    input               rst                 ,
    input               pclk                ,
    input               href                ,
    input               vsync               ,
    input       [7:0]   data_in             ,
    output reg  [11:0]  data_out            ,
    output reg          wr_en               ,
    output reg  [18:0]  ram_out_addr
);

    /*************************************************************
        寄存器定义
    *************************************************************/
    reg  [15:0]         ori_color = 0       ;
    reg  [18:0]         ram_next_addr = 0   ;
    reg  [1:0]          bit_status = 0      ;   //用于合并颜色块
    initial ram_out_addr <= 0;

    /*************************************************************
        数据合并+ori_color 转 RGB444
    *************************************************************/
    always@ (posedge pclk) begin
        if(vsync == 0) begin
            ram_out_addr <= 0;
```

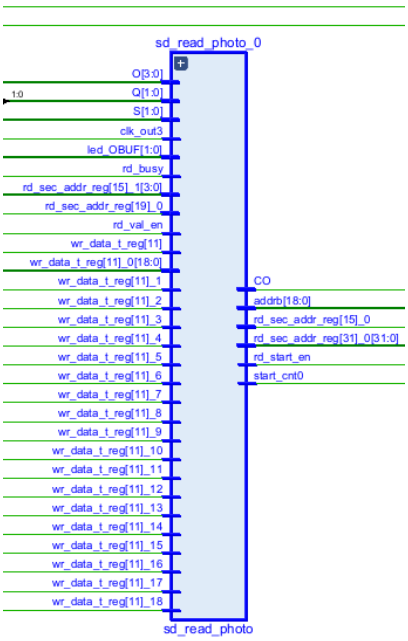
```
            ram_next_addr <= 0;
            bit_status <= 0;
        end else begin
            data_out <= { ori_color[15:12], ori_color[10:7],
ori_color[4:1] };
            ram_out_addr <= ram_next_addr;
            wr_en <= bit_status[1];
            bit_status <= {bit_status[0], (href && !bit_status[0])};//都
是高电平有效
            ori_color <= {ori_color[7:0], data_in};
            if(bit_status[1] == 1)
                ram_next_addr <= ram_next_addr+1;
        end
    end

endmodule
```

照片读取模块：

1 描述：

在照片读取模式控制线上升沿触发，产生 SD 卡读取信号并且负责将 SD 卡读取的数据写入到 RAM 特定位置中。

2 功能框图：



3 接口定义：

```
module sd_read_photo(
```

```
    input               clk               ,//系统输入时钟
    input               get_photo_mode    ,//获取照片模式选线
    input       [3:0]   select_photo_no   ,//照片地址选线
    input               rd_busy           ,//读取忙信号
    input               rd_val_en         ,//读取数据输出使能
    output reg          rd_start_en       ,//读取启动使能
    output      [18:0]  sd_rd_ram_addr    ,//写入 RAM 地址
    output reg  [31:0]  rd_sec_addr       //扇区选择地址
    );
```

4 Verilog 代码：

```
module sd_read_photo(
    input               clk               ,
    input               get_photo_mode    ,
    input       [3:0]   select_photo_no   ,
    input               rd_busy           ,
    input               rd_val_en         ,
    output reg          rd_start_en       ,
    output      [18:0]  sd_rd_ram_addr    ,
    output reg  [31:0]  rd_sec_addr
    );
    /*************************************************************
        参数定义
    *************************************************************/
    parameter READ_TIME = 1200;
    parameter DIVIDED_CLK = 50000;

    /*************************************************************
        线网定义
    *************************************************************/
    reg                 send_clk          ;
    reg                 get_photo_mode_0  ;
    reg                 get_photo_mode_1  ;
    reg         [18:0]  sd_rd_ram_addr_t  ;
    wire                get_photo_mode_ena;
    integer             cnt = 0           ;
    initial rd_sec_addr <= 0;

    /*************************************************************
        读取时钟产生
    *************************************************************/
    always @(posedge clk) begin
        cnt <= cnt + 1;
        if(cnt == DIVIDED_CLK) begin
```

```
            cnt <= 0;
            send_clk <= 1;
        end
        else
            send_clk <= 0;
    end

    /*************************************************************
        模式进入打拍
    *************************************************************/
    assign get_photo_mode_ena = get_photo_mode_0 & ~get_photo_mode_1;
    always @(posedge clk) begin
        get_photo_mode_0 <= get_photo_mode;
        get_photo_mode_1 <= get_photo_mode_0;
    end

    /*************************************************************
        生成读取信号
    *************************************************************/
    reg [10:0] start_cnt;
    initial start_cnt <= 0;
    always @(posedge clk) begin
        if(get_photo_mode_ena) begin
            start_cnt <= 1;
            case (select_photo_no)
                4'h0: rd_sec_addr <= 32'd73743;
                4'h1: rd_sec_addr <= 32'd74943;
                4'h2: rd_sec_addr <= 32'd76143;
                4'h3: rd_sec_addr <= 32'd77343;
                4'h4: rd_sec_addr <= 32'd78543;
                4'h5: rd_sec_addr <= 32'd79743;
                4'h6: rd_sec_addr <= 32'd80943;
                4'h7: rd_sec_addr <= 32'd82143;
                4'h8: rd_sec_addr <= 32'd83343;
                4'h9: rd_sec_addr <= 32'd84543;
                4'ha: rd_sec_addr <= 32'd85743;
                4'hb: rd_sec_addr <= 32'd86943;
                4'hc: rd_sec_addr <= 32'd88143;
                4'hd: rd_sec_addr <= 32'd89343;
                4'he: rd_sec_addr <= 32'd90543;
                4'hf: rd_sec_addr <= 32'd91743;
                default: rd_sec_addr <= 32'd73743;
            endcase
        end
```

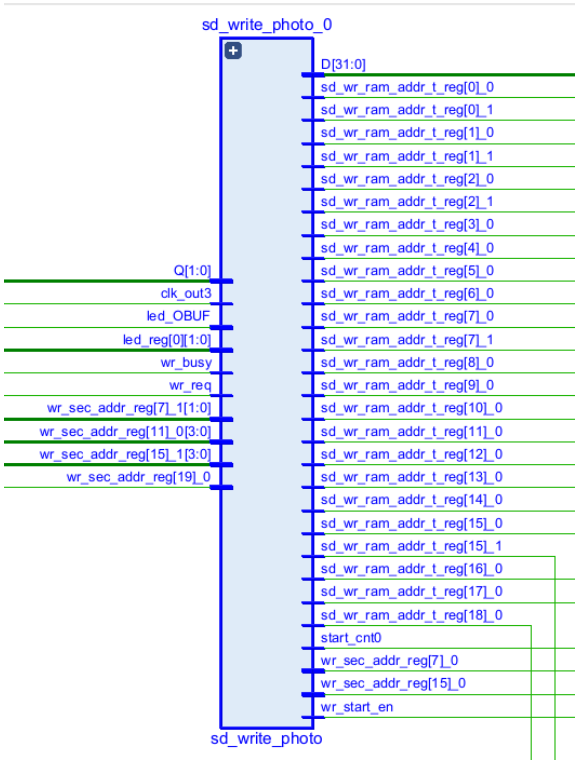
```
        rd_start_en <= 0;
        if(send_clk && ~rd_busy && start_cnt > 0) begin
            rd_start_en <= 1;
            rd_sec_addr <= rd_sec_addr + 1;
            start_cnt <= start_cnt + 1;
            if(start_cnt == READ_TIME) begin
                start_cnt <= 0;
            end
        end
    end

    /*************************************************************
        获取照片写入 RAM 地址
    *************************************************************/
    assign sd_rd_ram_addr = (sd_rd_ram_addr_t > 0) ? sd_rd_ram_addr_t -
19'd1 : 19'd0;
    always@(posedge clk) begin
        if(!get_photo_mode) begin
            sd_rd_ram_addr_t <= 19'b0;
        end else if(rd_val_en) begin
            if(sd_rd_ram_addr_t < 19'd307219) begin
                sd_rd_ram_addr_t <= sd_rd_ram_addr_t +19'd1;
            end
            else begin
                sd_rd_ram_addr_t <= 19'd0;
            end
        end
    end

endmodule
```

照片写入模块：

1 描述：

照片写入模块与照片读取模块类似，在照片写入信号的上升沿触发，负责产生照片写入使能，并且将等待写入的照片的像素地址和像素信息连同等待写入的扇区地址一起写入到 SD 卡之中。

2 功能框图：



3 接口定义：

```
module sd_write_photo(
    input               clk                 , //输入时钟
    input       [3:0]   select_photo_no     ,//照片编号选线
    input               caught_photo_mode   ,//照片写入模式选线
    input               wr_req              ,//照片写入请求线
    input               wr_busy             ,//SD 卡写入繁忙线
    output reg          wr_start_en         ,//SD 卡写入开始线
    output reg  [31:0]  wr_sec_addr         ,//写入扇区地址
    output      [18:0]  sd_wr_ram_addr       //写入像素的 RAM 地址产生
    );
```

4 Verilog 代码：

```
module sd_write_photo(
    input               clk                 ,
    input       [3:0]   select_photo_no     ,
    input               caught_photo_mode   ,
    input               wr_req              ,
    input               wr_busy             ,
    output reg          wr_start_en         ,
    output reg  [31:0]  wr_sec_addr         ,
    output      [18:0]  sd_wr_ram_addr
    );

    /*************************************************************
```

```
    参数定义
*****************************************************************/
parameter READ_TIME = 1200;
parameter DIVIDED_CLK = 50000;

/*****************************************************************
    线网定义
*****************************************************************/
reg [18:0]          sd_wr_ram_addr_t;
reg                 send_clk;
initial             wr_sec_addr <= 0;
integer             cnt = 0;

/*****************************************************************
    发送时钟产生
*****************************************************************/
always @(posedge clk) begin
    cnt <= cnt + 1;
    if(cnt == DIVIDED_CLK) begin
        cnt <= 0;
        send_clk <= 1;
    end
    else
        send_clk <= 0;
end

/*****************************************************************
    进入模式打拍
*****************************************************************/
reg caught_photo_mode_0, caught_photo_mode_1;
wire caught_photo_mode_ena;
assign caught_photo_mode_ena = caught_photo_mode_0 &
~caught_photo_mode_1;
always @(posedge clk) begin
    caught_photo_mode_0 <= caught_photo_mode;
    caught_photo_mode_1 <= caught_photo_mode_0;
end

/*****************************************************************
    发送信号产生
*****************************************************************/
reg [10:0] start_cnt;
initial start_cnt <= 0;
always @(posedge clk) begin
```

```
        if(caught_photo_mode_ena) begin
            start_cnt <= 1;
            case (select_photo_no)
                4'h0: wr_sec_addr <= 32'd73743;
                4'h1: wr_sec_addr <= 32'd74943;
                4'h2: wr_sec_addr <= 32'd76143;
                4'h3: wr_sec_addr <= 32'd77343;
                4'h4: wr_sec_addr <= 32'd78543;
                4'h5: wr_sec_addr <= 32'd79743;
                4'h6: wr_sec_addr <= 32'd80943;
                4'h7: wr_sec_addr <= 32'd82143;
                4'h8: wr_sec_addr <= 32'd83343;
                4'h9: wr_sec_addr <= 32'd84543;
                4'ha: wr_sec_addr <= 32'd85743;
                4'hb: wr_sec_addr <= 32'd86943;
                4'hc: wr_sec_addr <= 32'd88143;
                4'hd: wr_sec_addr <= 32'd89343;
                4'he: wr_sec_addr <= 32'd90543;
                4'hf: wr_sec_addr <= 32'd91743;
                default: wr_sec_addr <= 32'd73743;
            endcase
        end
        wr_start_en <= 0;
        if(send_clk && ~wr_busy && start_cnt > 0) begin
            wr_start_en <= 1;
            wr_sec_addr <= wr_sec_addr + 1;
            start_cnt <= start_cnt + 1;
            if(start_cnt == READ_TIME) begin
                start_cnt <= 0;
            end
        end
    end

    /************************************************************
        获取照片读取 RAM 地址
    ************************************************************/
    assign sd_wr_ram_addr = (sd_wr_ram_addr_t[18] & sd_wr_ram_addr_t[17]) ?
(19'b0) : sd_wr_ram_addr_t;
    always@(posedge clk) begin
        if(!caught_photo_mode) begin
            sd_wr_ram_addr_t <= 19'b0;
        end else if(wr_start_en) begin
            sd_wr_ram_addr_t <= sd_wr_ram_addr_t - 19'b1;
        end else if(wr_req) begin
```

```
            if(sd_wr_ram_addr_t < 19'd307219) begin
                sd_wr_ram_addr_t <= sd_wr_ram_addr_t +19'd1;
            end
            else begin
                sd_wr_ram_addr_t <= 19'd0;
            end
        end
    end
endmodule
```

SD 卡控制模块：

1 描述：

该模块为 SD 卡控制的顶层模块，负责实例化 SD 卡初始化模块、SD 卡写入模块和 SD 卡读取模块，并且控制三个模块之间的时序信息，使得 SD 卡初始化模块先于 SD 卡读写模块工作，并且在 SD 卡初始化完成之后，输出 SD 卡初始化完成信号，提供统一接口给外部照片写入和读取模块。

2 功能框图：

3 接口定义：

```
module sd_ctrl(
    input               clk_ref        ,
    input               clk_ref_180deg ,
    input               rst_n          ,

    input               sd_miso        ,
    output              sd_clk         ,
    output  reg         sd_cs          ,
    output  reg         sd_mosi        ,

    input               wr_start_en    ,
    input  [31:0]       wr_sec_addr    ,
    input  [15:0]       wr_data        ,
    output              wr_busy        ,
    output              wr_req         ,

    input               rd_start_en    ,
    input  [31:0]       rd_sec_addr    ,
    output              rd_busy        ,
    output              rd_val_en      ,
    output [15:0]       rd_val_data    ,
```

```
    output              sd_init_done
    );
```

4 Verilog 代码：

```
module sd_ctrl(
    input               clk_ref        ,
    input               clk_ref_180deg ,
    input               rst_n          ,

    input               sd_miso        ,
    output              sd_clk         ,
    output  reg         sd_cs          ,
    output  reg         sd_mosi        ,

    input               wr_start_en    ,
    input  [31:0]       wr_sec_addr    ,
    input  [15:0]       wr_data        ,
    output              wr_busy        ,
    output              wr_req         ,

    input               rd_start_en    ,
    input  [31:0]       rd_sec_addr    ,
    output              rd_busy        ,
    output              rd_val_en      ,
    output [15:0]       rd_val_data    ,

    output              sd_init_done
    );

    /*************************************************************
        线网与定义
    *************************************************************/
    wire               init_sd_clk    ;
    wire               init_sd_cs     ;
    wire               init_sd_mosi   ;
    wire               wr_sd_cs       ;
    wire               wr_sd_mosi     ;
    wire               rd_sd_cs       ;
    wire               rd_sd_mosi     ;
    assign  sd_clk =
(sd_init_done==1'b0)  ?  init_sd_clk  :  clk_ref_180deg; //SD 卡的 SPI_CLK
设置

    /*************************************************************
```

```
        SD 卡信号接口选择
    ***************************************************************/
    always @(*) begin
        if(sd_init_done == 1'b0) begin //SD 卡初始化完成之前,端口信号和初始
化模块信号相连
            sd_cs = init_sd_cs;
            sd_mosi = init_sd_mosi;
        end else if(wr_busy) begin
            sd_cs = wr_sd_cs;
            sd_mosi = wr_sd_mosi;
        end else if(rd_busy) begin
            sd_cs = rd_sd_cs;
            sd_mosi = rd_sd_mosi;
        end else begin
            sd_cs = 1'b1;
            sd_mosi = 1'b1;
        end
    end

    /***************************************************************
        SD 卡初始化模块实例化
    ***************************************************************/
    sd_init sd_init_0(
        .clk_ref            (clk_ref)           ,
        .rst_n              (rst_n)             ,
        .sd_miso            (sd_miso)           ,
        .sd_clk             (init_sd_clk)       ,
        .sd_cs              (init_sd_cs)        ,
        .sd_mosi            (init_sd_mosi)      ,
        .sd_init_done       (sd_init_done)
    );
    /***************************************************************
        SD 卡写数据模块实例化
    ***************************************************************/
    sd_write sd_write_0(
        .clk_ref            (clk_ref)           ,
        .clk_ref_180deg     (clk_ref_180deg)    ,
        .rst_n              (rst_n)             ,
        .sd_miso            (sd_miso)           ,
        .sd_cs              (wr_sd_cs)          ,
        .sd_mosi            (wr_sd_mosi)        ,
        .wr_start_en        (wr_start_en & sd_init_done),
        .wr_sec_addr        (wr_sec_addr)       ,
        .wr_data            (wr_data)           ,
```

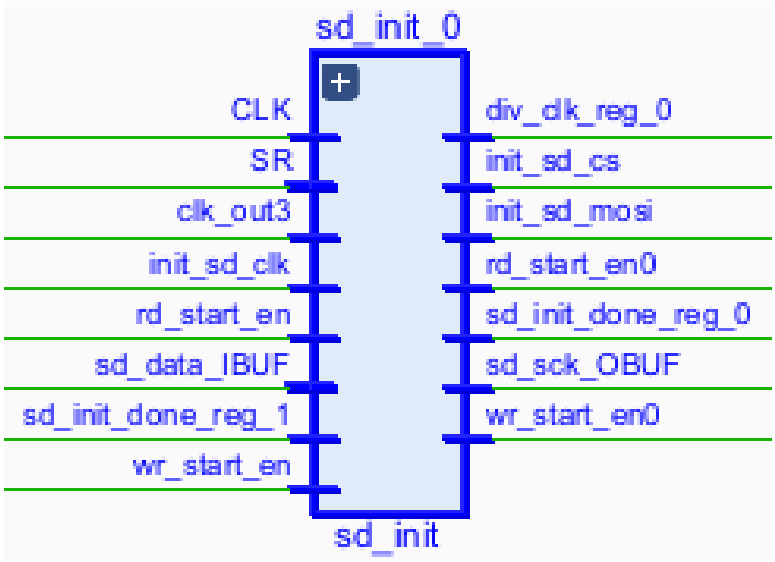
```
        .wr_busy            (wr_busy)           ,
        .wr_req             (wr_req)
    );
    /*************************************************************
        SD 卡读数据模块实例化
    *************************************************************/
    sd_read sd_read_0(
        .clk_ref            (clk_ref)           ,
        .clk_ref_180deg     (clk_ref_180deg)    ,
        .rst_n              (rst_n)             ,
        .sd_miso            (sd_miso)           ,
        .sd_cs              (rd_sd_cs)          ,
        .sd_mosi            (rd_sd_mosi)        ,
        .rd_start_en        (rd_start_en & sd_init_done),
        .rd_sec_addr        (rd_sec_addr)       ,
        .rd_busy            (rd_busy)           ,
        .rd_val_en          (rd_val_en)         ,
        .rd_val_data        (rd_val_data)
    );

endmodule
```

SD 卡初始化模块：

1 描述：

SD 卡初始化模块在系统上电的同时写入 SD 卡初始化信息，将 SD 卡设置为应用信号控制模式。

2 功能框图：



3 接口定义：

```
module sd_init(
    input           clk_ref       ,//反相 SD 卡时钟
    input           rst_n         ,//低电平有效

    input           sd_miso       ,//SPI 输入线
    output          sd_clk        ,//SPI 时钟
    output  reg     sd_cs         ,//SPI 片选信号线
    output  reg     sd_mosi       ,//SPI 输出线
    output  reg     sd_init_done   //SD 卡初始化完成使能
    );
```

4 Verilog 代码：

```
module sd_init(
    input           clk_ref       ,
    input           rst_n         ,

    input           sd_miso       ,
    output          sd_clk        ,
    output  reg     sd_cs         ,
    output  reg     sd_mosi       ,
    output  reg     sd_init_done
    );

    /*************************************************************
        参数定义
    *************************************************************/
    parameter  CMD0  = {8'h40,8'h00,8'h00,8'h00,8'h00,8'h95};   //复位命
令
    parameter  CMD8  = {8'h48,8'h00,8'h00,8'h01,8'haa,8'h87};   //接口状
态查询命令
    parameter  CMD55 = {8'h77,8'h00,8'h00,8'h00,8'h00,8'hff};   //应用指令
    parameter  ACMD41= {8'h69,8'h40,8'h00,8'h00,8'h00,8'hff};   //发送操
作寄存器指令

    parameter  DIV_FREQ = 200;                                  //初始化时
钟分频系数
    parameter  POWER_ON_NUM = 5000;                             //上电等待
时间
    parameter  OVER_TIME_NUM = 25000;                           //响应超时
时间

    parameter  st_idle        = 7'b000_0001;                    //上电等待
SD 卡稳定
```

```
    parameter  st_send_cmd0   = 7'b000_0010;                    //发送软件
复位命令
    parameter  st_wait_cmd0   = 7'b000_0100;                    //等待 SD
卡响应
    parameter  st_send_cmd8   = 7'b000_1000;                    //检查 SD
卡版本
    parameter  st_send_cmd55  = 7'b001_0000;                    //应用指令
切换
    parameter  st_send_acmd41 = 7'b010_0000;                    //发送寄存
器配置
    parameter  st_init_done   = 7'b100_0000;                   //SD 卡初始
化完成

    /*************************************************************
        寄存器与线网配置
    *************************************************************/
    reg    [7:0]   cur_state      ;
    reg    [7:0]   next_state     ;
    reg    [7:0]   div_cnt        ;
    reg            div_clk        ;
    reg    [12:0]  poweron_cnt    ;
    reg            res_en         ;
    reg    [47:0]  res_data       ;
    reg            res_flag       ;
    reg    [5:0]   res_bit_cnt    ;
    reg    [5:0]   cmd_bit_cnt    ;
    reg    [15:0]  over_time_cnt  ;
    reg            over_time_en   ;
    wire           div_clk_180deg ;


    assign  sd_clk = ~div_clk;
    assign  div_clk_180deg = ~div_clk;
    /*************************************************************
        时钟分频
    *************************************************************/
    always @(posedge clk_ref or negedge rst_n) begin
        if(!rst_n) begin
            div_clk <= 1'b0;
            div_cnt <= 8'd0;
        end
        else begin
            if(div_cnt == DIV_FREQ/2-1'b1) begin
                div_clk <= ~div_clk;
```

```
                div_cnt <= 8'd0;
            end
            else
                div_cnt <= div_cnt + 1'b1;
        end
    end

    /*************************************************************
        商店等待稳定
    *************************************************************/
    always @(posedge div_clk or negedge rst_n) begin
        if(!rst_n)
            poweron_cnt <= 13'd0;
        else if(cur_state == st_idle) begin
            if(poweron_cnt < POWER_ON_NUM)
                poweron_cnt <= poweron_cnt + 1'b1;
        end
        else
            poweron_cnt <= 13'd0;
    end

    /*************************************************************
        接收 sd 卡返回的响应数据
    *************************************************************/
    always @(posedge div_clk_180deg or negedge rst_n) begin
        if(!rst_n) begin
            res_en <= 1'b0;
            res_data <= 48'd0;
            res_flag <= 1'b0;
            res_bit_cnt <= 6'd0;
        end else begin
            //sd_miso = 0 开始接收响应数据
            if(sd_miso == 1'b0 && res_flag == 1'b0) begin
                res_flag <= 1'b1;
                res_data <= {res_data[46:0],sd_miso};
                res_bit_cnt <= res_bit_cnt + 6'd1;
                res_en <= 1'b0;
            end else if(res_flag) begin
                //R1 返回 1 个字节,R3 R7 返回 5 个字节
                //在这里统一按照 6 个字节来接收,多出的 1 个字节为 NOP(8 个时钟周
期的延时)
                res_data <= {res_data[46:0],sd_miso};
                res_bit_cnt <= res_bit_cnt + 6'd1;
                if(res_bit_cnt == 6'd47) begin
```

```
                    res_flag <= 1'b0;
                    res_bit_cnt <= 6'd0;
                    res_en <= 1'b1;
                end
            end else
                res_en <= 1'b0;
        end
    end

    /*************************************************************
       状态依据分频时钟推进
    *************************************************************/
    always @(posedge div_clk or negedge rst_n) begin
        if(!rst_n)
            cur_state <= st_idle;
        else
            cur_state <= next_state;
    end

    /*************************************************************
        状态机，主 SD 卡初始化配置
    *************************************************************/
    always @(*) begin
        next_state = st_idle;
        case(cur_state)
            st_idle : begin
                //上电至少等待 74 个同步时钟周期
                if(poweron_cnt == POWER_ON_NUM)          //默认状态,上电等
待 SD 卡稳定
                    next_state = st_send_cmd0;
                else
                    next_state = st_idle;
            end
            st_send_cmd0 : begin                         //发送软件复位命令
                if(cmd_bit_cnt == 6'd47)
                    next_state = st_wait_cmd0;
                else
                    next_state = st_send_cmd0;
            end
            st_wait_cmd0 : begin                         //等待 SD 卡响应
                if(res_en) begin
                    if(res_data[47:40] == 8'h01)
                        next_state = st_send_cmd8;
                    else
```

```
                    next_state = st_idle;
                end
                else if(over_time_en)
                    next_state = st_idle;
                else
                    next_state =
st_wait_cmd0;
            end
            st_send_cmd8 : begin                        //CMD8,检测 SD 卡是
否适配
                if(res_en) begin
                    if(res_data[19:16] == 4'b0001)
                        next_state = st_send_cmd55;
                    else
                        next_state = st_idle;
                end
                else
                    next_state = st_send_cmd8;
            end
            st_send_cmd55 : begin  //切换应用相关命令
                if(res_en) begin
                    if(res_data[47:40] == 8'h01)
                        next_state = st_send_acmd41;
                    else
                        next_state = st_send_cmd55;
                end
                else
                    next_state = st_send_cmd55;
            end
            st_send_acmd41 : begin                      //发送操作寄存器
                if(res_en) begin
                    if(res_data[47:40] == 8'h00)
                        next_state = st_init_done;
                    else
                        next_state = st_send_cmd55;     //初始化未完成,重
新发起
                end
                else
                    next_state = st_send_acmd41;
            end
            st_init_done : next_state = st_init_done;    //初始化完成
            default : next_state = st_idle;
        endcase
    end
```

```
    always @(posedge div_clk or negedge rst_n) begin
        if(!rst_n) begin
            sd_cs <= 1'b1;
            sd_mosi <= 1'b1;
            sd_init_done <= 1'b0;
            cmd_bit_cnt <= 6'd0;
            over_time_cnt <= 16'd0;
            over_time_en <= 1'b0;
        end else begin
            over_time_en <= 1'b0;
            case(cur_state)
                st_idle : begin                                 //上电等待 SD
卡稳定
                    sd_cs <= 1'b1;
                    sd_mosi <= 1'b1;
                end
                st_send_cmd0 : begin                            //发送 CMD0 软
件复位命令
                    cmd_bit_cnt <= cmd_bit_cnt + 6'd1;
                    sd_cs <= 1'b0;
                    sd_mosi <= CMD0[6'd47 - cmd_bit_cnt];
                    if(cmd_bit_cnt == 6'd47)
                        cmd_bit_cnt <= 6'd0;
                end

                st_wait_cmd0 : begin
                    sd_mosi <= 1'b1;
                    if(res_en)                                  //SD 卡返回响
应信号
                        sd_cs <=
1'b1;
                    over_time_cnt <= over_time_cnt + 1'b1;
                    if(over_time_cnt == OVER_TIME_NUM - 1'b1) //SD 卡响应超
时,重新发送软件复位命令
                        over_time_en <= 1'b1;
                    if(over_time_en)
                        over_time_cnt <=
16'd0;
                end
                st_send_cmd8 : begin                            //发送 CMD8
                    if(cmd_bit_cnt<=6'd47) begin
                        cmd_bit_cnt <= cmd_bit_cnt + 6'd1;
                        sd_cs <= 1'b0;
```

```
                sd_mosi <= CMD8[6'd47 - cmd_bit_cnt];
            end
            else begin
                sd_mosi <= 1'b1;
                if(res_en) begin
                    sd_cs <= 1'b1;
                    cmd_bit_cnt <= 6'd0;
                end
            end

        end
        st_send_cmd55 : begin                          //发送 CMD55
            if(cmd_bit_cnt<=6'd47) begin
                cmd_bit_cnt <= cmd_bit_cnt + 6'd1;
                sd_cs <= 1'b0;
                sd_mosi <= CMD55[6'd47 - cmd_bit_cnt];
            end
            else begin
                sd_mosi <= 1'b1;
                if(res_en) begin
                    sd_cs <= 1'b1;
                    cmd_bit_cnt <= 6'd0;
                end
            end

        end
        st_send_acmd41 : begin                         //发送 ACMD41
            if(cmd_bit_cnt <= 6'd47) begin
                cmd_bit_cnt <= cmd_bit_cnt + 6'd1;
                sd_cs <= 1'b0;
                sd_mosi <= ACMD41[6'd47 - cmd_bit_cnt];
            end
            else begin
                sd_mosi <= 1'b1;
                if(res_en) begin
                    sd_cs <= 1'b1;
                    cmd_bit_cnt <= 6'd0;
                end
            end
        end
        st_init_done : begin                           //初始化完成
            sd_init_done <= 1'b1;
            sd_cs <= 1'b1;
            sd_mosi <= 1'b1;
```

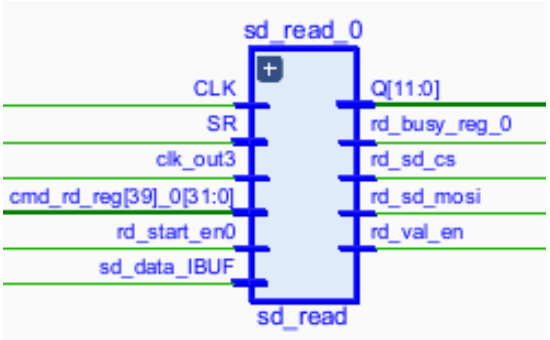
```
                end
                default : begin
                    sd_cs <= 1'b1;
                    sd_mosi <= 1'b1;
                end
            endcase
        end
    end

endmodule
```

SD 卡读取模块：

1 描述：

SD 卡读取模块在收到 SD 卡读取信号上升沿时，会按照给定的扇区信息将读取命令发送给 SD 卡，并且按照接受时序将 SD 卡输出的串口信息合并成 16 位的像素信息，并且在合并完成之后拉高输出数据有效信号线提供给外界访问。此模块每次读取以 512 字节（一个扇区）为基本单位，在读取过程中会拉高读忙信号。

2 功能框图：



3 接口定义：

```
module sd_read(
    input               clk_ref       ,//读取时钟
    input               clk_ref_180deg,//反相读取时钟
    input               rst_n         ,//重置，低电平有效
    input               sd_miso       ,//SPI 输入线
    output  reg         sd_cs         ,//SPI 片选信号线
    output  reg         sd_mosi       ,//SPI 输出线
    input               rd_start_en   ,//读取开始信号
    input       [31:0]  rd_sec_addr   ,//读取扇区地址
    output  reg         rd_busy       ,//读忙信号
    output  reg         rd_val_en     ,//读取值可用信号
    output  reg  [15:0] rd_val_data    //读取数据
    );
```

4 Verilog 代码：

```
module sd_read(
    input               clk_ref       ,
    input               clk_ref_180deg,
    input               rst_n         ,
    input               sd_miso       ,
    output  reg         sd_cs         ,
    output  reg         sd_mosi       ,
    input               rd_start_en   ,
    input       [31:0]  rd_sec_addr   ,
    output  reg         rd_busy       ,
    output  reg         rd_val_en     ,
    output  reg  [15:0]  rd_val_data
    );

    /*************************************************************
        线网与寄存器定义
    *************************************************************/
    reg           rd_en_d0      ;
    reg           rd_en_d1      ;
    reg           res_en        ;
    reg    [7:0]   res_data      ;
    reg           res_flag      ;
    reg    [5:0]   res_bit_cnt   ;

    reg           rx_en_t       ;
    reg    [15:0]  rx_data_t     ;
    reg           rx_flag       ;
    reg    [3:0]   rx_bit_cnt    ;
    reg    [8:0]   rx_data_cnt   ;
    reg           rx_finish_en  ;

    reg    [3:0]   rd_ctrl_cnt   ;
    reg    [47:0]  cmd_rd        ;
    reg    [5:0]   cmd_bit_cnt   ;
    reg           rd_data_flag  ;
    wire          pos_rd_en     ;
    assign  pos_rd_en = (~rd_en_d1) & rd_en_d0;

    /*************************************************************
        rd_start_en 信号延时打拍
    *************************************************************/
    always @(posedge clk_ref or negedge rst_n) begin
```

```
    if(!rst_n) begin
        rd_en_d0 <= 1'b0;
        rd_en_d1 <= 1'b0;
    end else begin
        rd_en_d0 <= rd_start_en;
        rd_en_d1 <= rd_en_d0;
    end
end

/*************************************************************
    接收 sd 卡返回的响应数据
*************************************************************/
always @(posedge clk_ref_180deg or negedge rst_n) begin
    if(!rst_n) begin
        res_en <= 1'b0;
        res_data <= 8'd0;
        res_flag <= 1'b0;
        res_bit_cnt <= 6'd0;
    end else begin
        if(sd_miso == 1'b0 && res_flag == 1'b0) begin
            res_flag <= 1'b1;
            res_data <= {res_data[6:0],sd_miso};
            res_bit_cnt <= res_bit_cnt + 6'd1;
            res_en <= 1'b0;
        end else if(res_flag) begin
            res_data <= {res_data[6:0],sd_miso};
            res_bit_cnt <= res_bit_cnt + 6'd1;
            if(res_bit_cnt == 6'd7) begin
                res_flag <= 1'b0;
                res_bit_cnt <= 6'd0;
                res_en <= 1'b1;
            end
        end
        else
            res_en <= 1'b0;
    end
end

/*************************************************************
    接收 SD 卡有效数据
*************************************************************/
always @(posedge clk_ref_180deg or negedge rst_n) begin
    if(!rst_n) begin
        rx_en_t <= 1'b0;
```

```
            rx_data_t <= 16'd0;
            rx_flag <= 1'b0;
            rx_bit_cnt <= 4'd0;
            rx_data_cnt <= 9'd0;
            rx_finish_en <= 1'b0;
        end else begin
            rx_en_t <= 1'b0;
            rx_finish_en <= 1'b0;
            if(rd_data_flag && sd_miso == 1'b0 && rx_flag == 1'b0)
                rx_flag <= 1'b1;
            else if(rx_flag) begin
                rx_bit_cnt <= rx_bit_cnt + 4'd1;
                rx_data_t <= {rx_data_t[14:0],sd_miso};
                if(rx_bit_cnt == 4'd15) begin
                    rx_data_cnt <= rx_data_cnt + 9'd1;
                    if(rx_data_cnt <= 9'd255)
                        rx_en_t <= 1'b1;
                    else if(rx_data_cnt == 9'd257) begin
                        rx_flag <= 1'b0;
                        rx_finish_en <= 1'b1;
                        rx_data_cnt <= 9'd0;
                        rx_bit_cnt <= 4'd0;
                    end
                end
            end
            else
                rx_data_t <= 16'd0;
        end
    end

    /*************************************************************
        寄存输出数据有效信号和数据
    *************************************************************/
    always @(posedge clk_ref or negedge rst_n) begin
        if(!rst_n) begin
            rd_val_en <= 1'b0;
            rd_val_data <= 16'd0;
        end
        else begin
            if(rx_en_t) begin
                rd_val_en <= 1'b1;
                rd_val_data <= rx_data_t;
            end
            else
```

```
                rd_val_en <= 1'b0;
        end
    end

    /*************************************************************
        读命令产生
    *************************************************************/
    always @(posedge clk_ref or negedge rst_n) begin
        if(!rst_n) begin
            sd_cs <= 1'b1;
            sd_mosi <= 1'b1;
            rd_ctrl_cnt <= 4'd0;
            cmd_rd <= 48'd0;
            cmd_bit_cnt <= 6'd0;
            rd_busy <= 1'b0;
            rd_data_flag <= 1'b0;
        end else begin
            case(rd_ctrl_cnt)
                4'd0 : begin
                    rd_busy <= 1'b0;
                    sd_cs <= 1'b1;
                    sd_mosi <= 1'b1;
                    if(pos_rd_en) begin
                        cmd_rd <= {8'h51,rd_sec_addr,8'hff};    //写入单个
命令块 CMD17
                        rd_ctrl_cnt <= rd_ctrl_cnt + 4'd1;
                        rd_busy <= 1'b1;
                    end
                end
                4'd1 : begin
                    if(cmd_bit_cnt <= 6'd47) begin              //开始按位
发送读命令
                        cmd_bit_cnt <= cmd_bit_cnt + 6'd1;
                        sd_cs <= 1'b0;
                        sd_mosi <= cmd_rd[6'd47 - cmd_bit_cnt];
                    end
                    else begin
                        sd_mosi <= 1'b1;
                        if(res_en) begin                        //SD 卡响应
                            rd_ctrl_cnt <= rd_ctrl_cnt + 4'd1;
                            cmd_bit_cnt <= 6'd0;
                        end
                    end
                end
```

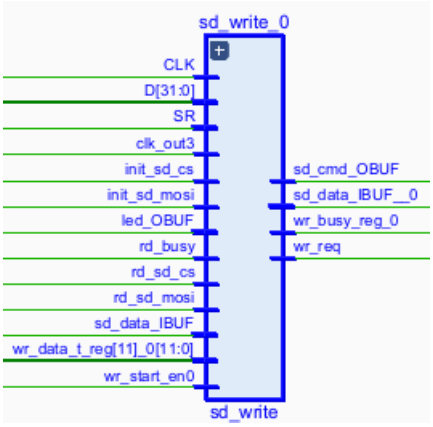
```
                4'd2 : begin
                    rd_data_flag <= 1'b1;
                    if(rx_finish_en) begin                    //数据接收
完成
                        rd_ctrl_cnt <= rd_ctrl_cnt + 4'd1;
                        rd_data_flag <= 1'b0;
                        sd_cs <= 1'b1;
                    end
                end
                default : begin
                    sd_cs <= 1'b1;
                    rd_ctrl_cnt <= rd_ctrl_cnt + 4'd1;
                end
            endcase
        end
    end

endmodule
```

SD 卡写入模块：

1 描述：

SD 卡写入模块与 SD 卡读取模块类似，在收到写入信号上升沿时进入写入模式，发送写入命令给 SD 卡，并且在每次 SD 卡写入数据发送前发出 SD 卡写入数据请求信号，此时必须在一个时钟周期内向 SD 卡写入模块发送 16 位写入数据，否则会导致时序混乱。在 SD 卡写入完成后，也即读取忙信号拉低后，必须等待至少 8 个时钟周期以供该模块与 SD 卡交流写入是否成功。

2 功能框图：



3 接口定义：

```
module sd_write(
    input                clk_ref        ,//输入时钟
    input                clk_ref_180deg ,//反相输入时钟
```

```
    input               rst_n          ,//重置，低电平有效
    input               sd_miso        ,//SD 卡写入数据线
    output  reg         sd_cs          ,//SD 卡片选数据线
    output  reg         sd_mosi        , //SD 卡输出数据线
    input               wr_start_en    ,//写入使能
    input  [31:0]       wr_sec_addr    ,//写入扇区地址
    input  [15:0]       wr_data        ,//写入数据
    output  reg         wr_busy        ,//写入忙信号
    output  reg         wr_req          //写入数据请求信号
    );
```

4 Verilog 代码：

```
module sd_write(
    input               clk_ref        ,
    input               clk_ref_180deg ,
    input               rst_n          ,
    input               sd_miso        ,
    output  reg         sd_cs          ,
    output  reg         sd_mosi        ,
    input               wr_start_en    ,
    input  [31:0]       wr_sec_addr    ,
    input  [15:0]       wr_data        ,
    output  reg         wr_busy        ,
    output  reg         wr_req
    );

    /*************************************************************
        寄存器与参数定义
    *************************************************************/
    parameter  HEAD_BYTE = 8'hfe    ;      //写命令命令
头
    reg           wr_en_d0         ;
    reg           wr_en_d1         ;
    reg           res_en           ;
    reg    [7:0]  res_data         ;
    reg           res_flag         ;
    reg    [5:0]  res_bit_cnt      ;

    reg    [3:0]  wr_ctrl_cnt      ;
    reg    [47:0] cmd_wr           ;
    reg    [5:0]  cmd_bit_cnt      ;
    reg    [3:0]  bit_cnt          ;
    reg    [8:0]  data_cnt         ;
    reg    [15:0] wr_data_t        ;
```

```
reg            detect_done_flag ;
reg    [7:0]   detect_data      ;
wire           pos_wr_en        ;
assign  pos_wr_en = (~wr_en_d1) & wr_en_d0;

/*************************************************************
    wr_start_en 信号延时打拍
*************************************************************/
always @(posedge clk_ref or negedge rst_n) begin
    if(!rst_n) begin
        wr_en_d0 <= 1'b0;
        wr_en_d1 <= 1'b0;
    end else begin
        wr_en_d0 <= wr_start_en;
        wr_en_d1 <= wr_en_d0;
    end
end

/*************************************************************
    接收 sd 卡返回的响应数据
*************************************************************/
always @(posedge clk_ref_180deg or negedge rst_n) begin
    if(!rst_n) begin
        res_en <= 1'b0;
        res_data <= 8'd0;
        res_flag <= 1'b0;
        res_bit_cnt <= 6'd0;
    end else begin
        if(sd_miso == 1'b0 && res_flag == 1'b0) begin
            res_flag <= 1'b1;
            res_data <= {res_data[6:0],sd_miso};
            res_bit_cnt <= res_bit_cnt + 6'd1;
            res_en <= 1'b0;
        end
        else if(res_flag) begin
            res_data <= {res_data[6:0],sd_miso};
            res_bit_cnt <= res_bit_cnt + 6'd1;
            if(res_bit_cnt == 6'd7) begin
                res_flag <= 1'b0;
                res_bit_cnt <= 6'd0;
                res_en <= 1'b1;
            end
        end
        else
```

```
                res_en <= 1'b0;
        end
    end

    /*************************************************************
        检测 SD 卡是否空闲
    *************************************************************/
    always @(posedge clk_ref or negedge rst_n) begin
        if(!rst_n)
            detect_data <= 8'd0;
        else if(detect_done_flag)
            detect_data <= {detect_data[6:0],sd_miso};
        else
            detect_data <= 8'd0;
    end

    /*************************************************************
        SD 卡写入数据
    *************************************************************/
    always @(posedge clk_ref or negedge rst_n) begin
        if(!rst_n) begin
            sd_cs <= 1'b1;
            sd_mosi <= 1'b1;
            wr_ctrl_cnt <= 4'd0;
            wr_busy <= 1'b0;
            cmd_wr <= 48'd0;
            cmd_bit_cnt <= 6'd0;
            bit_cnt <= 4'd0;
            wr_data_t <= 16'd0;
            data_cnt <= 9'd0;
            wr_req <= 1'b0;
            detect_done_flag <= 1'b0;
        end
        else begin
            wr_req <= 1'b0;
            case(wr_ctrl_cnt)
                4'd0 : begin
                    wr_busy <= 1'b0;                        //写空闲
                    sd_cs <= 1'b1;
                    sd_mosi <= 1'b1;
                    if(pos_wr_en) begin
                        cmd_wr <= {8'h58,wr_sec_addr,8'hff};    //CMD24
                        wr_ctrl_cnt <= wr_ctrl_cnt + 4'd1;
                        wr_busy <= 1'b1;
```

```
                    end
                end
                4'd1 : begin
                    if(cmd_bit_cnt <= 6'd47) begin              //发送写命
令
                        cmd_bit_cnt <= cmd_bit_cnt + 6'd1;
                        sd_cs <= 1'b0;
                        sd_mosi <= cmd_wr[6'd47 - cmd_bit_cnt];
                    end
                    else begin
                        sd_mosi <= 1'b1;
                        if(res_en) begin
                            wr_ctrl_cnt <= wr_ctrl_cnt + 4'd1;
                            cmd_bit_cnt <= 6'd0;
                            bit_cnt <= 4'd1;
                        end
                    end
                end

                4'd2 : begin
                    bit_cnt <= bit_cnt + 4'd1;
                    if(bit_cnt>=4'd8 && bit_cnt <= 4'd15) begin
                        sd_mosi <= HEAD_BYTE[4'd15-bit_cnt];
                        if(bit_cnt == 4'd14)
                            wr_req <= 1'b1;
                        else if(bit_cnt == 4'd15)
                            wr_ctrl_cnt <= wr_ctrl_cnt + 4'd1;
                    end
                end
                4'd3 : begin                                    //写入数据
                    bit_cnt <= bit_cnt + 4'd1;
                    if(bit_cnt == 4'd0) begin
                        sd_mosi <= wr_data[4'd15-bit_cnt];
                        wr_data_t <= wr_data;
                    end
                    else
                        sd_mosi <= wr_data_t[4'd15-bit_cnt];
                    if((bit_cnt == 4'd14) && (data_cnt <= 9'd255))
                        wr_req <= 1'b1;
                    if(bit_cnt == 4'd15) begin
                        data_cnt <= data_cnt + 9'd1;
                        if(data_cnt == 9'd255) begin
                            data_cnt <= 9'd0;
                            wr_ctrl_cnt <= wr_ctrl_cnt + 4'd1;
```

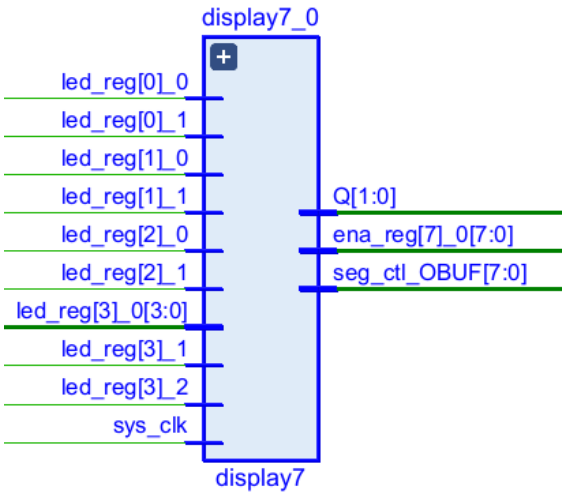
```
                    end
                end
            end
            4'd4 : begin                                    //CRC 校验
                bit_cnt <= bit_cnt + 4'd1;
                sd_mosi <= 1'b1;
                //crc 写入完成,控制计数器加 1
                if(bit_cnt == 4'd15)
                    wr_ctrl_cnt <= wr_ctrl_cnt + 4'd1;
            end
            4'd5 : begin
                if(res_en)
                    wr_ctrl_cnt <= wr_ctrl_cnt + 4'd1;
            end
            4'd6 : begin
                detect_done_flag <= 1'b1;                   //写入完成
                if(detect_data == 8'hff) begin
                    wr_ctrl_cnt <= wr_ctrl_cnt + 4'd1;
                    detect_done_flag <= 1'b0;
                end
            end
            default : begin
                sd_cs <= 1'b1;
                wr_ctrl_cnt <= wr_ctrl_cnt + 4'd1;
            end
        endcase
    end
end

endmodule
```

七段数码管显示模块：

1 描述：

该模块负责刷新七段数码管显示内容，具有 8 个 4 位 8421 码输入信号，提供 32 位数据的 16 进制形式输出。刷新频率位 1000Hz。

2 功能框图：



接口定义：

```
module display7(
    input               clk     ,//时钟信号
    input [3:0]         led1    ,//第一位
    input [3:0]         led2    ,//第二位
    input [3:0]         led3    ,//第三位
    input [3:0]         led4    ,//第四位
    input [3:0]         led5    ,//第五位
    input [3:0]         led6    ,//第六位
    input [3:0]         led7    ,//第七位
    input [3:0]         led8    ,//第八位
    output reg [7:0]    ena     ,//七段数码管输出使能
    output [7:0]        ctl       //七段数码管控制使能
);
```

Verilog 代码：

```
module display7(
    input               clk     ,
    input [3:0]         led1    ,
    input [3:0]         led2    ,
    input [3:0]         led3    ,
    input [3:0]         led4    ,
    input [3:0]         led5    ,
    input [3:0]         led6    ,
    input [3:0]         led7    ,
    input [3:0]         led8    ,
    output reg [7:0]    ena     ,
    output [7:0]        ctl
);
    wire display_clk;
    Divider display_div(.I_CLK(clk), .O_CLK(display_clk));
```

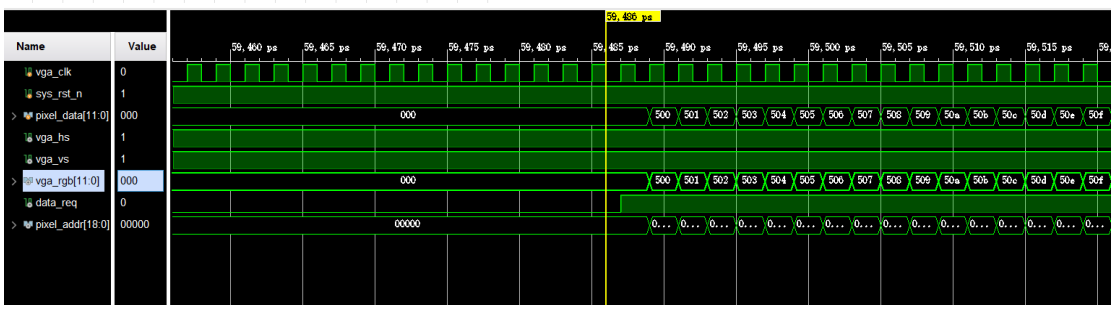
```
    reg [3:0] led;
    trans7 trans7_0(led, ctl);
    reg [2:0] cnt;
    initial begin
      cnt <= 0;
      ena <= 0;
    end
    always @(posedge display_clk) begin
        cnt <= cnt + 1;
    end
    always @(posedge display_clk)
        case(cnt)
            default: led <= led1;
            3'b000: led <= led1;
            3'b001: led <= led2;
            3'b010: led <= led3;
            3'b011: led <= led4;
            3'b100: led <= led5;
            3'b101: led <= led6;
            3'b110: led <= led7;
            3'b111: led <= led8;
        endcase
    always @(posedge display_clk)
        case(cnt)
            default: ena <= 8'b11111110;
            3'b000: ena <= 8'b11111110;
            3'b001: ena <= 8'b11111101;
            3'b010: ena <= 8'b11111011;
            3'b011: ena <= 8'b11110111;
            3'b100: ena <= 8'b11101111;
            3'b101: ena <= 8'b11011111;
            3'b110: ena <= 8'b10111111;
            3'b111: ena <= 8'b01111111;
        endcase
endmodule
```

五、测试模块建模

VGA 控制模块：

```
`timescale 1ps/1ps

module vga_driver_tb();
    reg vga_clk, sys_rst_n;
```

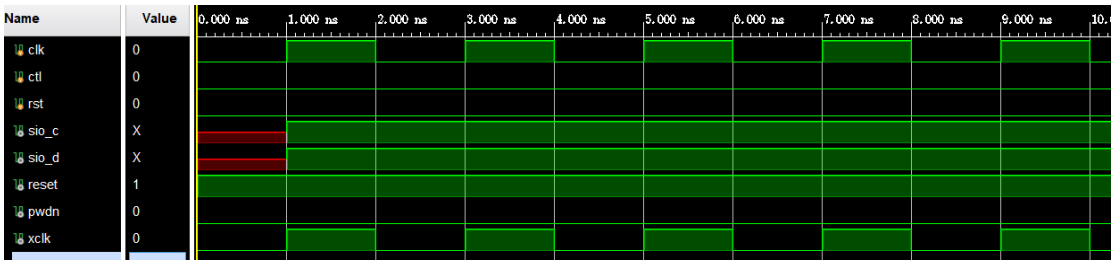
```
    reg [11:0] pixel_data;
    wire vga_hs, vga_vs;
    wire [11:0] vga_rgb;
    wire data_req;
    wire [18:0] pixel_addr;
    vga_driver vga_driver_0(
        vga_clk,
        sys_rst_n,
        vga_hs,
        vga_vs,
        vga_rgb,
        data_req,
        pixel_data,
        pixel_addr
    );

    initial begin
        vga_clk = 0;sys_rst_n =1;
        pixel_data = 0;
    end
    always @(pixel_addr)
        pixel_data <= pixel_addr[11:0];

    always #1 vga_clk = ~vga_clk;

endmodule
```



摄像头寄存器初始化模块

```
module ov2640_sccb_cfg_init_tb();
    reg clk;
    reg ctl;
    reg rst;
    wire sio_c;
    wire sio_d;
    wire reset,pwdn,xclk;
    wire camera_init_done;
```

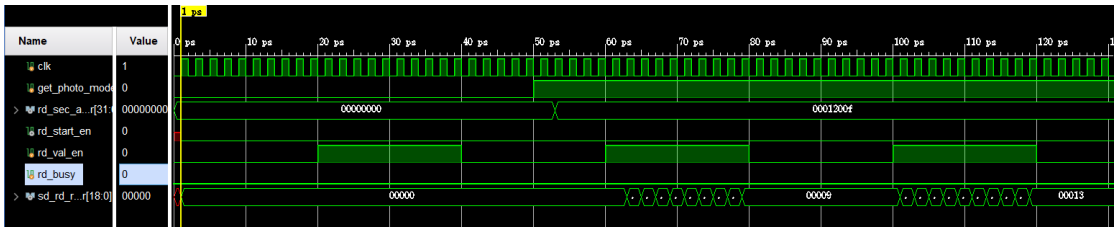
```
    ov2640_sccb_cfg_init ov2640_sccb_cfg_init_0(
        clk,
        rst,
        sio_c,
        sio_d,
        reset,
        pwdn,
        xclk,
        4'b0,
        camera_init_done
    );
    initial begin
        clk = 0; rst = 0;ctl = 0;
    end
    always #1 clk = ~clk;
endmodule
```



照片读取模块：

```
module sd_read_photo_tb();
    reg clk;
    reg get_photo_mode;
    wire [31:0] rd_sec_addr;
    wire rd_start_en;
    reg rd_val_en ;
    reg rd_busy;
    wire [18:0] sd_rd_ram_addr;
    sd_read_photo sd_read_photo_0(
        .clk            (clk),
        .get_photo_mode (get_photo_mode),
        .rd_sec_addr    (rd_sec_addr),
        .rd_busy        (rd_busy),
        .rd_start_en    (rd_start_en),
        .rd_val_en      (rd_val_en),
        .sd_rd_ram_addr (sd_rd_ram_addr)
    );

    initial begin
```

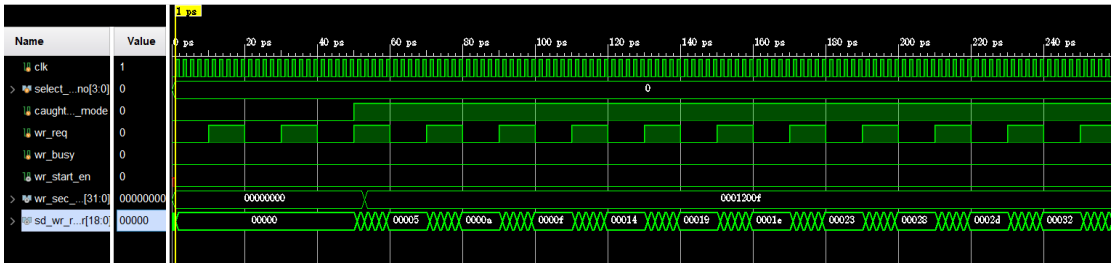
```
        clk = 0;
        rd_busy = 0;
        rd_val_en = 0;
        get_photo_mode = 0;
        #50 get_photo_mode = 1;
        #500 get_photo_mode = 0;
        #50 get_photo_mode = 1;
    end
    always #20 rd_val_en = ~rd_val_en;
    always #1 clk = ~clk;
endmodule
```



照片写入模块：

```
module sd_write_photo_tb();
    reg clk;
    reg [3:0] select_photo_no;
    reg caught_photo_mode;
    reg wr_req;
    reg wr_busy;
    wire wr_start_en;
    wire [31:0] wr_sec_addr;
    wire [18:0] sd_wr_ram_addr;

    sd_write_photo sd_write_photo_0(
        .clk                (clk),
        .select_photo_no    (select_photo_no),
        .caught_photo_mode  (caught_photo_mode),
        .wr_req             (wr_req),
        .wr_busy            (wr_busy),
        .wr_start_en        (wr_start_en),
        .wr_sec_addr        (wr_sec_addr),
        .sd_wr_ram_addr     (sd_wr_ram_addr)
    );
    initial begin
        caught_photo_mode = 0;
        select_photo_no = 0;
        clk = 0;
```

```
        wr_busy = 0;
        wr_req = 0;
        #50;
        caught_photo_mode = 1;
        #500;
        caught_photo_mode = 0;
    end
    always #10 wr_req = ~wr_req;
    always #1 clk = ~clk;
endmodule
```



SD 卡控制模块：

```
`timescale 1ps/1ps

module sd_ctrl_tb();
    reg clk_ref, clk_ref_180deg, rst_n;
    reg sd_miso;
    wire sd_clk;
    wire sd_cs, sd_mosi;
    reg wr_start_en;
    reg [31:0] wr_sec_addr;
    reg [15:0] wr_data;
    wire wr_busy, wr_req,rd_busy,rd_val_en;
    reg rd_start_en;
    reg [31:0] rd_sec_addr;
    wire [15:0] rd_val_data;
    wire sd_init_done;
    sd_ctrl sd_ctrl_0(
        .clk_ref        (clk_ref),
        .clk_ref_180deg (clk_ref_180deg),
        .rst_n          (rst_n),
        .sd_miso        (sd_miso),
        .sd_clk         (sd_clk),
        .sd_cs          (sd_cs),
        .sd_mosi        (sd_mosi),
        .wr_start_en    (wr_start_en),
```

```
        .wr_sec_addr    (wr_sec_addr),
        .wr_data        (wr_data),
        .wr_busy        (wr_busy),
        .wr_req         (wr_req),
        .rd_start_en    (rd_start_en),
        .rd_sec_addr    (rd_sec_addr),
        .rd_busy        (rd_busy),
        .rd_val_en      (rd_val_en),
        .rd_val_data    (rd_val_data),
        .sd_init_done   (sd_init_done)
    );
    initial begin
        clk_ref = 0;
        clk_ref_180deg= 1;
        rst_n = 0 ;
        sd_miso = 1;
        wr_start_en = 0;
        wr_sec_addr = 0;
        wr_data = 0;
        rd_start_en = 0;
        rd_sec_addr = 0;
        #10 rst_n = 1;
    end

    always #1 clk_ref = ~clk_ref;
    always #1 clk_ref_180deg = ~clk_ref_180deg;

    always @(posedge clk_ref or negedge rst_n) begin
    if(!rst_n) begin
        sd_init_done_d0 <= 1'b0;
        sd_init_done_d1 <= 1'b0;
    end
    else begin
        sd_init_done_d0 <= sd_init_done;
        sd_init_done_d1 <= sd_init_done_d0;
    end
end

always @(posedge clk_ref or negedge rst_n) begin
    if(!rst_n) begin
        wr_start_en <= 1'b0;
        wr_sec_addr <= 32'd0;
    end
    else begin
```

```
        if(pos_init_done) begin
            wr_start_en <= 1'b1;
            wr_sec_addr <= 32'd2000;
        end
        else
            wr_start_en <= 1'b0;
    end
end

always @(posedge clk_ref or negedge rst_n) begin
    if(!rst_n)
        wr_data_t <= 16'b0;
    else if(wr_req)
        wr_data_t <= wr_data_t + 16'b1;
end

always @(posedge clk_ref or negedge rst_n) begin
    if(!rst_n) begin
        wr_busy_d0 <= 1'b0;
        wr_busy_d1 <= 1'b0;
    end
    else begin
        wr_busy_d0 <= wr_busy;
        wr_busy_d1 <= wr_busy_d0;
    end
end

always @(posedge clk_ref or negedge rst_n) begin
    if(!rst_n) begin
        rd_start_en <= 1'b0;
        rd_sec_addr <= 32'd0;
    end
    else begin
        if(neg_wr_busy) begin
            rd_start_en <= 1'b1;
            rd_sec_addr <= 32'd2000;
        end
        else
            rd_start_en <= 1'b0;
    end
end

always @(posedge clk_ref or negedge rst_n) begin
    if(!rst_n) begin
```

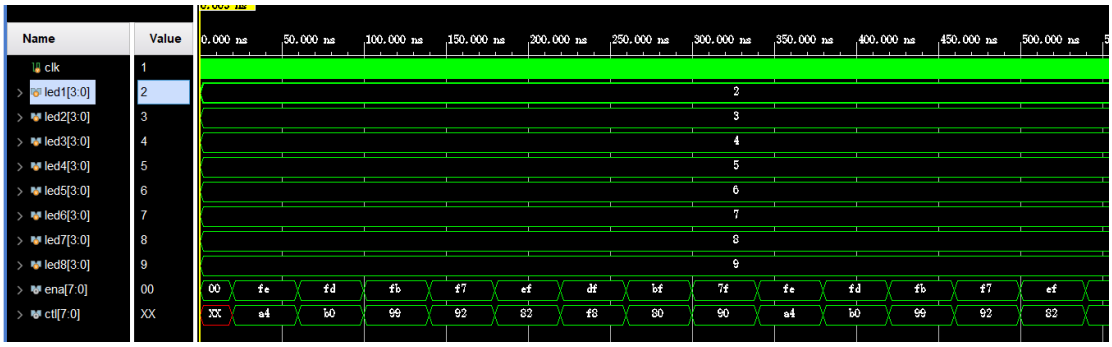
```
        rd_comp_data <= 16'd0;
        rd_right_cnt <= 9'd0;
    end
    else begin
        if(rd_val_en) begin
            rd_comp_data <= rd_comp_data + 16'b1;
            if(rd_val_data == rd_comp_data)
                rd_right_cnt <= rd_right_cnt + 9'd1;
        end
    end
end

endmodule
```

由于模拟信号太长，长达 8 万个时钟周期，此处无法放下，故未给出 Modisim 模拟信号贴图

七段数码管显示模块：

```
`timescale 1ps/1ps

module display7_tb();
    reg clk;
    reg [3:0] led1;
    reg [3:0] led2;
    reg [3:0] led3;
    reg [3:0] led4;
    reg [3:0] led5;
    reg [3:0] led6;
    reg [3:0] led7;
    reg [3:0] led8;

    wire [7:0] ena;
    wire [7:0] ctl;
    display7 display7_0(
        clk,
        led1,
        led2,
        led3,
        led4,
        led5,
        led6,
        led7,
        led8,
        ena,
```

```
        ctl
    );
    initial begin
        clk = 0;
        led1 = 2;
        led2 = 3;
        led3 = 4;
        led4 = 5;
        led5 = 6;
        led6 = 7;
        led7 = 8;
        led8 = 9;
    end
    always #1 clk = ~clk;
endmodule
```



注：

1、双口 RAM 模块与时钟分频模块为 IP 核，未进行 testbench 建模

2、SD 卡初始化模块、SD 卡读取模块和 SD 卡写入模块统一在 SD 卡控制模块中进行测试。

3、摄像头寄存器配置模块与 SCCB 协议驱动模块统一在摄像头寄存器初始化模块中进行测试。

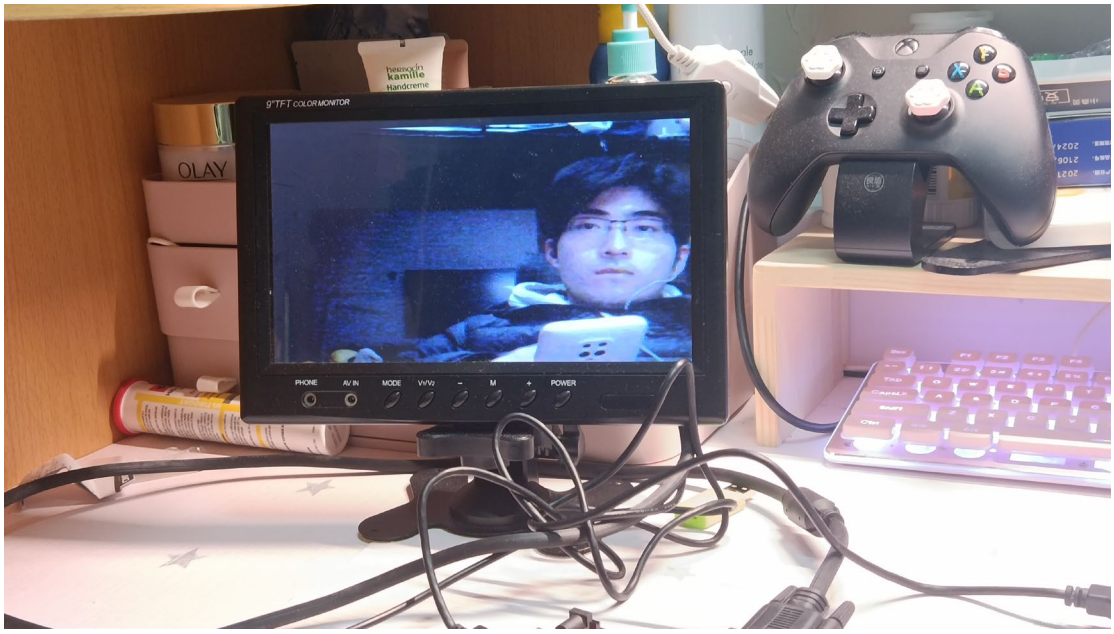
4、顶层模块只包含所有模块的线网连接，而建模难度又太大，故未进行 testbench 建模

六、实验结果与结论

实验结果如图：

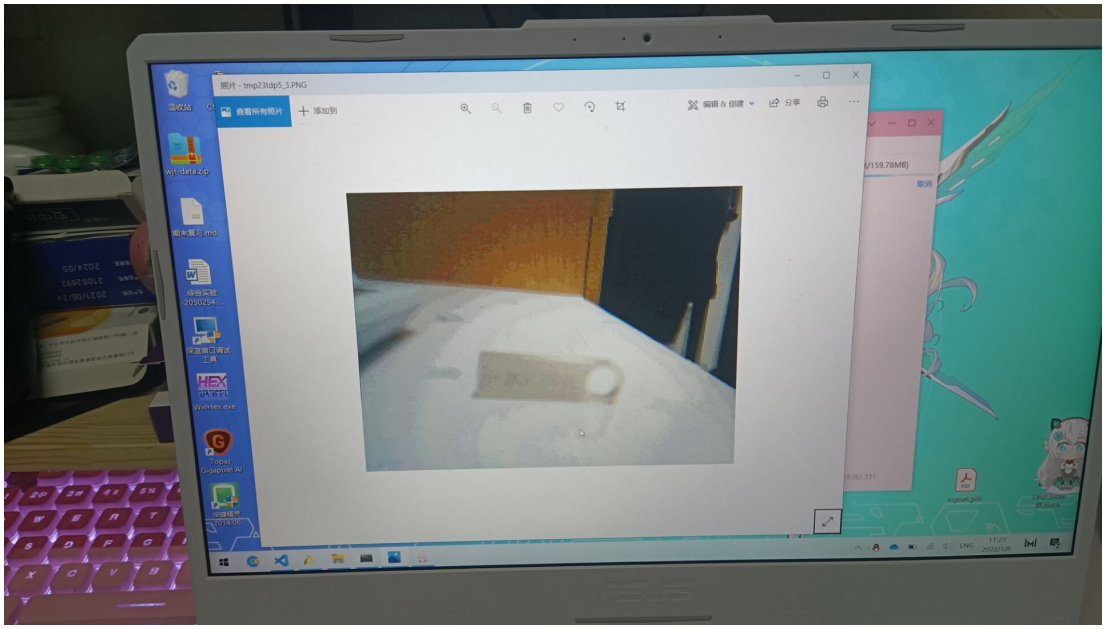
摄像头拍摄照片：

查看之前已经拍摄的照片：



查看电脑导入的照片：

在电脑上查看拍摄的照片：



本次实验使用 OV2640 摄像头、VGA 显示器、SD 卡和蓝牙模块，实现了一个比较简单和基础的照相机系统，能够查看、拍摄照片并保存到 SD 卡上，私以为是一个比较复杂、比较完整也比较实用的系统了。

以下介绍本次实验主要的难点、困难和一些考虑不周的地方。

首先讲一下**蓝牙模块**的问题。学校选用了 HC-06 主从一体机模块作为蓝牙模块提供给我们，虽然是主从一体机，但是这个模块的主机模式却十分鸡肋。该模块的主机模式不支持通过设备 MAC 号绑定其他蓝牙从机设备，而是通过定向寻

找与自身 PIN 码相匹配的设备进行绑定。这种模式也就导致了对于大量的常见的商用的、不提供蓝牙 PIN 码或者不提供更改 PIN 码的设备无法进行绑定。询问淘宝相关店家后得知，这种模块基本只用于两个 HC-06 模块之间的互相连接，我只能放弃最初作为蓝牙主机的设计了。

其次是**摄像头模块**。学校采用了微雪电子的 OV9925-2640 的模块，却没有提供相对应的设计手册和使用指南。2640 其实是一个已经被主流单片机 DIY 市场抛弃的一个存在，他没有 OV7725 的廉价与简洁，也没有 OV5640 的高清晰度和丰富功能，因此相对于其他这些主流摄像头而言，OV2640 的相关设计手册和配置方法非常稀少，即使找到也尽是些不知版本、不带注释和说明和 C 语言设计代码。

故而，该模块的主要难点非但不在于 SCCB 协议的驱动拟写，反而出现在了摄像头的寄存器配置上。而对于初学者而言，调试中不仅仅要面对驱动的故障，还要考虑是否是软件层面上的寄存器配置失误，**严重加大了调试摄像头的难度**。此外，微雪官网所提供的 OV2640_DS 寄存器说明文件，先不谈其**只有英文版**以及许多同学抱怨的**错误寄存器地址**，其完全没有任何对于明度、亮度以及对比度等调节方式的说明。查阅网上的一些用于单片机的 C 语言库函数，其明度亮度和焦距等基本配置的调节方式居然是非直接式的、通过写入间接访问寄存器地址和数据来控制调节，而这部分的配置在手册中完全没有一点参考资料。我连续尝试了几款不同的库函数配置文件，对明度、亮度和焦距的调节均以失败告终，不得已只能将该部分调节功能从设计中移除。

其余 VGA 模块和 SD 卡模块的拟写相对简单，产生出协议中指定的信号波形就能成功运行，网上的参考资料也较为丰富和详尽，这部分调试较为顺利。

受限于制作时间的匆忙，本次实验对照片的写入并没有实现真正的新建文件，而是朝着已有文件中写入，也不能支持真正删除一个文件，这是本次实验的一些不是很周全的地方。

七、心得体会和建议

参加完本课程后我感觉收获很多，我感觉最重要的是转变了传统的软件设计思路。软件的设计思路是串行的，一条一条地执行下来，而硬件的设计思路是并行的，所有模块都在时钟的驱动下同步的工作，这就给多模块的耦合造成了许多的不便。我刚开始接触硬件的时候，一直用软件的设计思路去设计硬件，产生了许多的问题，之后随着课程的发展，我才慢慢的逐渐意识到，无论是一个简单的数字钟还是我们现在使用的超级计算机，所有的无论大小的硬件设备，其本质都是一个有限的状态机，设计硬件的过程就是根据给定的输出要求去产生相应的输出的过程。

写大作业调 bug 的时候我发现了软件和硬件设计的另外一个至关重要的差距，也就是调试的区别。软件的调试是运行时的，是一遍运行着软件，一边一步一步的走下去，然后在这个走的过程中观察程序的控制是否出现了错误的地方，而这种调试方法可以执行根本上是由于软件的执行是很快的，基本上不需要什么时间。而硬件不一样，硬件从仿真到综合再到下板，即使是像本次试验这样比较简单的一个实验，一套流程的时间都要以分钟为单位计数。到本次实验的最终版本的时候，我的代码进行一个完整的仿真综合下板需要差不多 10 分钟的时间，如果还是采用往常下板观察实验现象的方式，调试的效率实在是太低了。

因此，对于硬件而言，其设计理念的关键在于模拟现象而非真实现象，硬件的开发不应当先写代码再做测试，而是应当先写测试代码再做实际的硬件设计。对于 SD 卡和摄像头这些对时序约束要求非常高的器件而言更是如此，故而我对于本课程的建议是，在学习 Verilog 语法的时候，不能仅仅把重点放在实际能够综合的那些硬件设计部分的语法上面，同时也应当介绍甚至是着重强调那些虽然不能仿真综合，但是却可以更加方便快捷的写出模拟测试代码的那些语法，也即那些高级语法。模拟是硬件设计的基础，如果连模拟都不会，一个硬件开发者就完全失去了明确自己代码是否有问题的能力，更谈不上设计一个复杂的硬件了。

放眼来看，目前中国对于美国的高端芯片控制禁令没有一丝一毫的解决办法，关键芯片，乃至我们的教学用的 FPGA 芯片中国都没有能力生产，受到其他国家的制约，这和国内高校学生更加向往软件设计、而嫌弃硬件设计也有些许的关系。我认为，要想造出中国芯，让祖国不受到其他人的制约，关键要从计算机本科教育抓起。我希望学校能够开设更多的硬件设计的讲堂，能够不从课程的角度，而是从这门工艺设计本身的角度去让学生了解硬件设计，抛开那些外界的流言蜚语，引导更多有才华的学生投入到芯片设计的领域来。