Project

1.开发计划

1.1小组选题

VGA拼图游戏

利用 minisys 开发板上的 VGA 接口做一个开发板与显示屏的交互，达成拼图游戏的效果。首先开发者需要事先加载进开发板一幅图像。在游戏的初始状态时，显示屏可以显示一幅静态图像；一个置位按钮可以使图像分块，打乱图片中各区块的位置，分块规则将根据五个开关确定；一个游戏启动的开关可以开始游戏，产生黑块；游戏开始后，可以通过四个按键分别控制黑块上下左右移动。当游戏进行到图像块等于初始化时的图像顺序时，代表游戏通关结束。

1.2成员分工及百分比

张泽凯	莫砚成	赖建宇
顶层模块设计	流水灯	防抖
VGA显示相关	蜂鸣器	计时器
串口读图	无解判断	计步器
游戏主模块	困难模式逻辑（3*3）	七段数码管显示
困难模式逻辑（3*3）		

1.3进度安排

时间	安排
第八周	开始阅读project，确定选题
第九周	完成初步分工并提前预习时序逻辑
第十周到第十一周	对各自分工内容进行资料查找和学习（VGA显示相关知识、七段数码显示管显示不同数字等）
第十二周	完成顶层模块设计和确定状态迁移流程图
第十三周	开始各自模块的设计
第十四周	代码合并整合，一人调试，剩余两人进行bonus的探究
第十五周	将bonus整合入模块，进行调试
第十六周	答辩

1.4执行记录

时间	执行
第八周	确定project的选题为VGA拼图游戏
第九周	简单预习了时序逻辑，还未进行分工
第十周	完成了简单的初步分工
第十一周	对各自分工内容进行资料查找和学习（VGA显示相关知识、七段数码显示管显示不同数字等）
第十二周	确定了状态迁移流程图，且在实验课与老师交流完善
第十三周	完成了顶层模块设计，开始小模块的书写和测试
第十四周	合并整合了代码，一人调试，剩余两人进行bonus的探究
第十五周	整合bonus加最后的调试，提前答辩
第十六周	起飞

2.设计

2.1需求分析

2.1.1系统功能

利用开发板上的VGA接口与显示屏交互，完成拼图游戏。通过串口读图预先加载图片，利用开发板上的按键进行打乱和游戏过程：在游戏的初始状态时，显示屏可以显示一幅静态图像；一个置位按钮可以使图像分块，打乱图片中各区块的位置，分块规则将根据五个开关确定；一个游戏启动的开关可以开始游戏，产生黑块；游戏开始后，可以通过四个按键分别控制黑块上下左右移动。当游戏进行到图像块

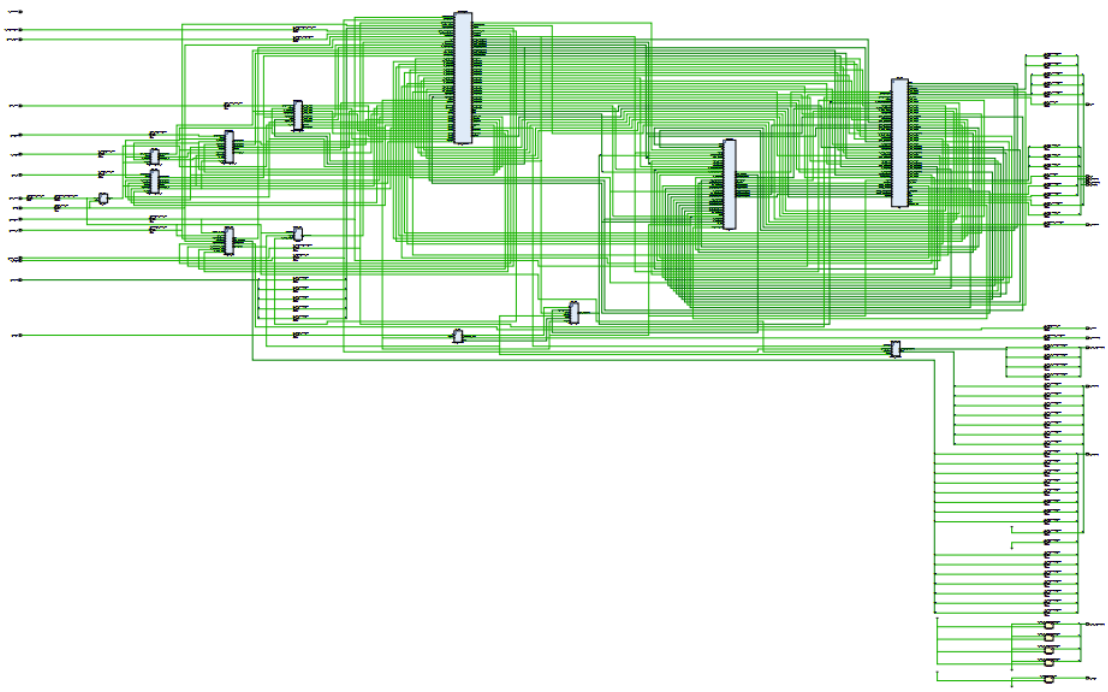
等于初始化时的图像顺序时，代表游戏通关结束。

2.1.2使用的输入输出设备

输入设备	输出设备
拨码开关	VGA显示屏
按键开关	七段数码显示管
（串口读图？）	LED灯
	蜂鸣器

2.2系统结构设计

2.2.0 实现电路总览



2.2.1各模块接口及功能

2.2.1.1 top

该模块为顶层设计模块，统一各模块信号。

其中各个模块具体分析附在后文

```
module top(
    input I_clk_100M, //时钟信号，频率为100MHz
    input I_rst_n,    //游戏统一复位信号
    input I_uart_rx,  //串口输入信号
    input I_black_EN, //拉黑信号
    input [4:0] I_num,//七段数码显示信号
    output O_uart_tx,//串口输出信号
    output O_transmitting,//传输中信号
```

```
    //VGA输出信号
    output [3:0] O_red,
    output [3:0] O_green,
    output [3:0] O_blue,
    output O_hs,,//行信号
    output O_vs, //场信号

    input I_btn_set,//置位信号
    //玩家移动信号
    input I_btn_left,
    input I_btn_right,
    input I_btn_up,
    input I_btn_down,

    output O_completed,   //拼图完成信号
    input I_rand_move_EN, //随机移动使能信号
    input I_rand_set_EN,  //随机置位使能信号
    input I_hard_mode,    //困难模式使能信号

    // ljy
    output [3:0] O_seg_en_time,//计时器使能信号
    output [3:0] O_seg_en_step,//计步器使能信号
    output [7:0] O_num,        //七段数码管要显示的数字

    // myc
    input I_unsolvable, //无解信号
    input I_music_on,   //音乐开启信号
    output wire O_music,//蜂鸣器输出信号
    output wire [15:0] O_led//led灯输出信号
);
    wire W_clk_25M;//时钟信号，频率为25MHz
    //串口传输相关信号
    wire [17:0] read_addr;
    wire W_rx_data_ready;
    wire [7:0] W_rx_data;
    wire W_rx_idle;
    assign O_transmitting = ~W_rx_idle;

    wire [7:0] W_pixel_data;//像素值
    //像素坐标
    wire [9:0] W_pixel_x;
    wire [9:0] W_pixel_y;
    wire W_pixel_valid;//当前像素是否有效
    wire [11:0] pixel_data_rgb444;//屏幕当前扫描到的像素点颜色信息（RGB 444）

    //屏幕各区域显示的图片
    wire [3:0] pos_a;
    wire [3:0] pos_b;
    wire [3:0] pos_c;
    wire [3:0] pos_d;
    wire [3:0] pos_e;
    wire [3:0] pos_f;
    wire [3:0] pos_g;
    wire [3:0] pos_h;
    wire [3:0] pos_i;

    wire W_btn_left;
    wire W_btn_right;
```

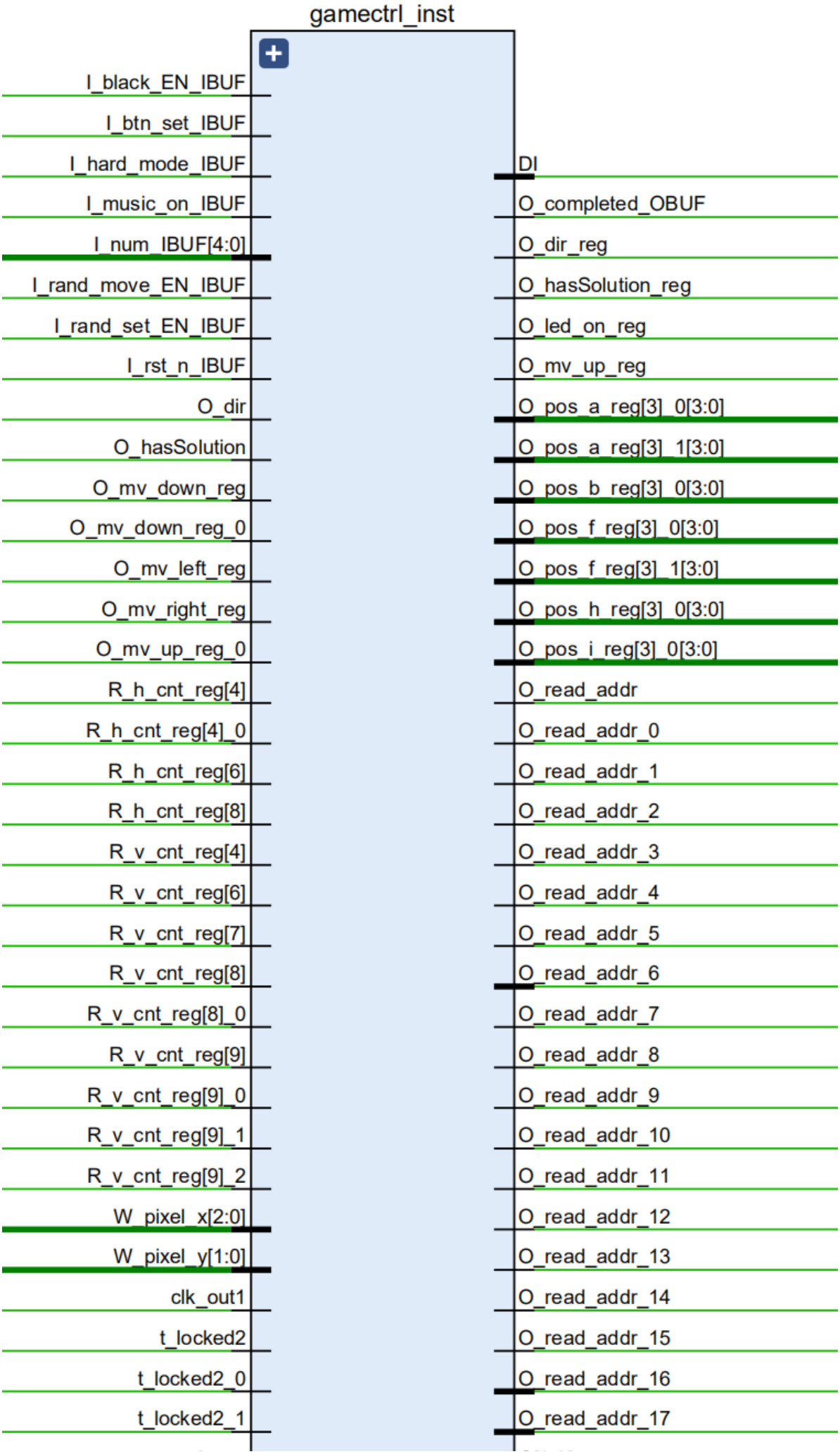
```
    wire W_btn_up;
    wire W_btn_down;
    wire W_hasSolution;//有解信号
    wire W_moving;

    // 生成25MHz的clk
    clk_25m clk_25m(.resetn(I_rst_n), .clk_in1(I_clk_100M),
.clk_out1(W_clk_25M));

    check check_inst(
        .IA(pos_a),
        .IB(pos_b),
        .IC(pos_c),
        .ID(pos_d),
        .I_clk(I_clk_100M),
        .I_rst_n(I_rst_n),
        .hard(I_hard_mode),
        .I_black_EN(I_black_EN),
        .O_hasSolution(W_hasSolution)
    );

    music_led_top ml_top_inst(
        .I_clk(I_clk_100M),
        .I_rst_n(I_rst_n),
        .I_black_EN(I_black_EN),
        .I_completed(O_completed),
        .I_unsolvable(~W_hasSolution),
        .I_music_on(I_music_on),
        .O_music(O_music),
        .O_led(O_led)
    );

    second_counter step_cnt_inst(
        .clk(I_clk_100M),
        .switch(I_black_EN),
        .set(I_btn_set),
        .rst(I_rst_n),
        .num(O_num),
        .seg_en(O_seg_en_time)
    );

    async_receiver rx_inst(
        .clk(W_clk_25M),
        .RxD(I_uart_rx),
        .RxD_data_ready(W_rx_data_ready),
        .RxD_data(W_rx_data),
        .RxD_idle(W_rx_idle)
    );

    vga vga_inst(
        .I_clk_25M(W_clk_25M),
        .I_rst_n(I_rst_n),
        .O_red(O_red),
        .O_green(O_green),
        .O_blue(O_blue),
        .O_hs(O_hs),
        .O_vs(O_vs),
        .I_pixel_data(pixel_data_rgb444),
```

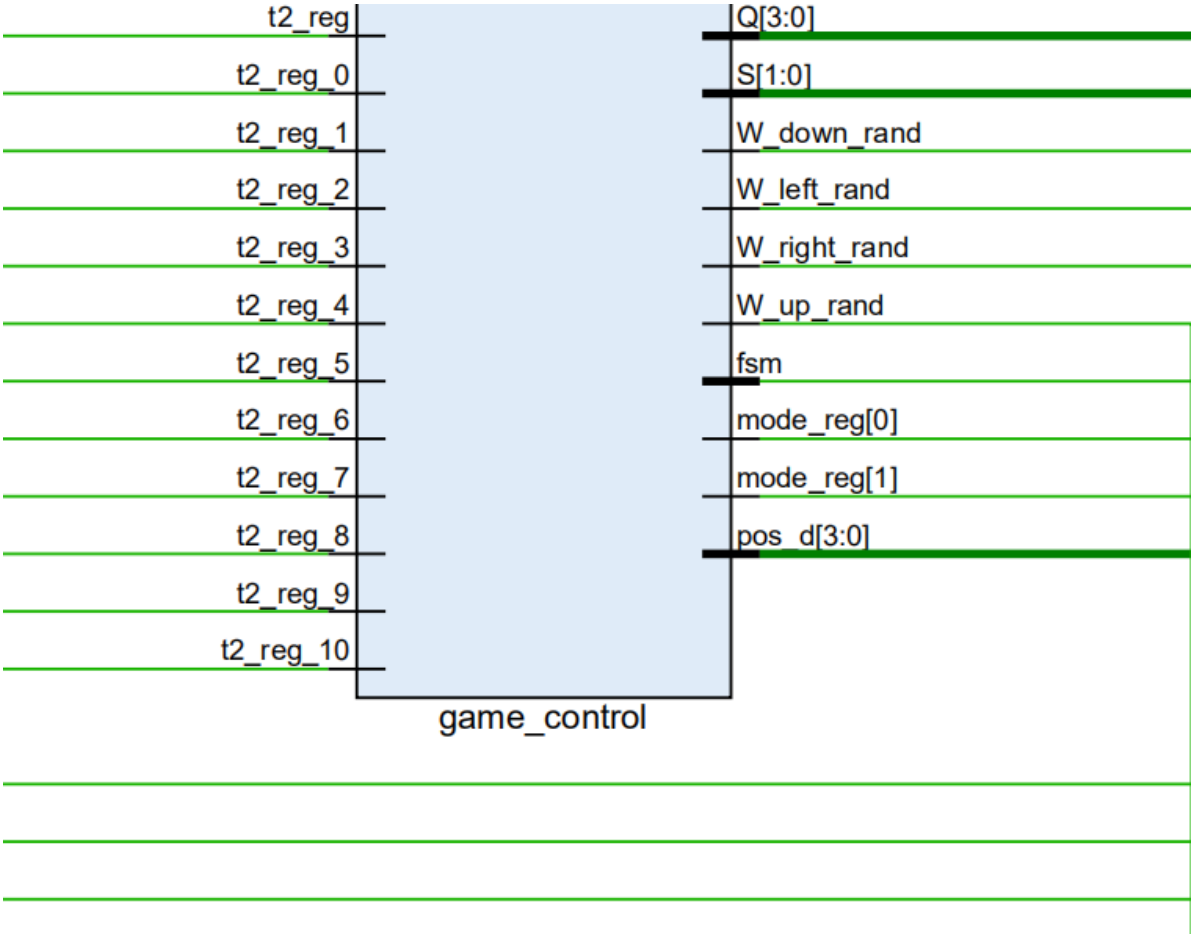
```
        .O_pixel_x(W_pixel_x),
        .O_pixel_y(W_pixel_y),
        .O_pixel_valid(W_pixel_valid)
    );

    anti_shake_single ass_left(
        .I_key(I_btn_left),
        .I_rst_n(I_rst_n),
        .I_clk(W_clk_25M),
        .O_key(W_btn_left)
    );

    anti_shake_single ass_right(
        .I_key(I_btn_right),
        .I_rst_n(I_rst_n),
        .I_clk(W_clk_25M),
        .O_key(W_btn_right)
    );

    anti_shake_single ass_up(
        .I_key(I_btn_up),
        .I_rst_n(I_rst_n),
        .I_clk(W_clk_25M),
        .O_key(W_btn_up)
    );

    anti_shake_single ass_down(
        .I_key(I_btn_down),
        .I_rst_n(I_rst_n),
        .I_clk(W_clk_25M),
        .O_key(W_btn_down)
    );


    pixel_ctrl pctrl_inst(
        .I_pixel_x(W_pixel_x),
        .I_pixel_y(W_pixel_y),
        .I_pixel_data(W_pixel_data),
        .I_black_EN(I_black_EN),
        .O_pixel_data(pixel_data_rgb444),

        .I_pos_a(pos_a),
        .I_pos_b(pos_b),
        .I_pos_c(pos_c),
        .I_pos_d(pos_d),
        .I_pos_e(pos_e),
        .I_pos_f(pos_f),
        .I_pos_g(pos_g),
        .I_pos_h(pos_h),
        .I_pos_i(pos_i),

        .O_read_addr(read_addr),
        .I_rst_n(I_rst_n),
        .I_clk_25M(W_clk_25M),
        .I_hard_mode(I_hard_mode)
    );

    game_control gamectrl_inst(
```

```
        .I_num(I_num),
        .I_rst_n(I_rst_n),
        .O_pos_a(pos_a),
        .O_pos_b(pos_b),
        .O_pos_c(pos_c),
        .O_pos_d(pos_d),
        .O_pos_e(pos_e),
        .O_pos_f(pos_f),
        .O_pos_g(pos_g),
        .O_pos_h(pos_h),
        .O_pos_i(pos_i),

        .O_completed(O_completed),

        .I_btn_set(I_btn_set),
        .I_btn_left(W_btn_left),
        .I_btn_right(W_btn_right),
        .I_btn_up(W_btn_up),
        .I_btn_down(W_btn_down),
        .I_black_EN(I_black_EN),
        .I_clk(W_clk_25M),
        .I_rand_move_EN(I_rand_move_EN),
        .I_rand_set_EN(I_rand_set_EN),
        .I_hard_mode(I_hard_mode),
        .O_moving(W_moving)
    );

    img_mem_ctrl imc_inst(
        .I_clk_25M(W_clk_25M),
        .I_rst_n(I_rst_n),
        .I_write_en(W_rx_data_ready),
        .I_write_data(W_rx_data),
        .O_pixel_data(W_pixel_data),
        .I_read_addr(read_addr)
    );
endmodule
```

2.2.1.2 game_control

该模块用于管理移动操作，拉黑操作，并且实现了随机打乱和随机置位的功能

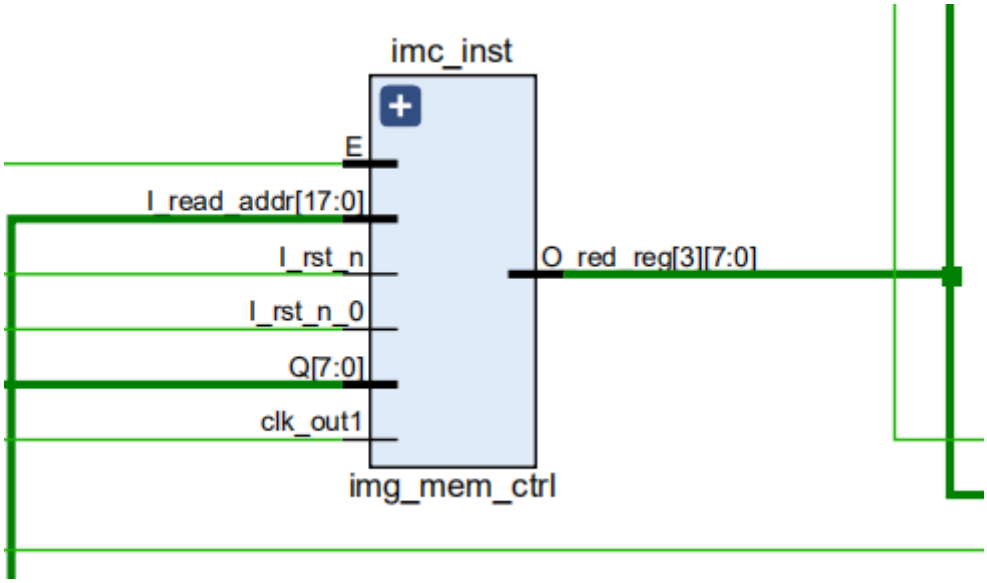

gamectrl_inst
I_black_EN_IBUF
I_btn_set_IBUF
I_hard_mode_IBUF
I_music_on_IBUF
I_num_IBUF[4:0]
I_rand_move_EN_IBUF
I_rand_set_EN_IBUF
I_rst_n_IBUF
O_dir
O_hasSolution
O_mv_down_reg
O_mv_down_reg_0
O_mv_left_reg
O_mv_right_reg
O_mv_up_reg_0
R_h_cnt_reg[4]
R_h_cnt_reg[4]_0
R_h_cnt_reg[6]
R_h_cnt_reg[8]
R_v_cnt_reg[4]
R_v_cnt_reg[6]
R_v_cnt_reg[7]
R_v_cnt_reg[8]
R_v_cnt_reg[8]_0
R_v_cnt_reg[9]
R_v_cnt_reg[9]_0
R_v_cnt_reg[9]_1
R_v_cnt_reg[9]_2
W_pixel_x[2:0]
W_pixel_y[1:0]
clk_out1
t_locked2
t_locked2_0
t_locked2_1
DI
O_completed_OBUF
O_dir_reg
O_hasSolution_reg
O_led_on_reg
O_mv_up_reg
O_pos_a_reg[3]_0[3:0]
O_pos_a_reg[3]_1[3:0]
O_pos_b_reg[3]_0[3:0]
O_pos_f_reg[3]_0[3:0]
O_pos_f_reg[3]_1[3:0]
O_pos_h_reg[3]_0[3:0]
O_pos_i_reg[3]_0[3:0]
O_read_addr
O_read_addr_0
O_read_addr_1
O_read_addr_2
O_read_addr_3
O_read_addr_4
O_read_addr_5
O_read_addr_6
O_read_addr_7
O_read_addr_8
O_read_addr_9
O_read_addr_10
O_read_addr_11
O_read_addr_12
O_read_addr_13
O_read_addr_14
O_read_addr_15
O_read_addr_16
O_read_addr_17




```
module game_control(
    input [4:0] I_num,
    input I_clk,
    input I_rst_n,
    input I_btn_set,
    input I_btn_left,
    input I_btn_right,
    input I_btn_up,
    input I_btn_down,
    output reg [3:0] O_pos_a,
    output reg [3:0] O_pos_b,
    output reg [3:0] O_pos_c,
    output reg [3:0] O_pos_d,
    output reg [3:0] O_pos_e,
    output reg [3:0] O_pos_f,
    output reg [3:0] O_pos_g,
    output reg [3:0] O_pos_h,
    output reg [3:0] O_pos_i,
    input I_black_EN,
    input I_rand_move_EN,
    input I_rand_set_EN,
    input I_hard_mode,
    output O_completed
);
```

2.2.1.3 img_mem_ctrl

该模块用于管理内存的写入和读取

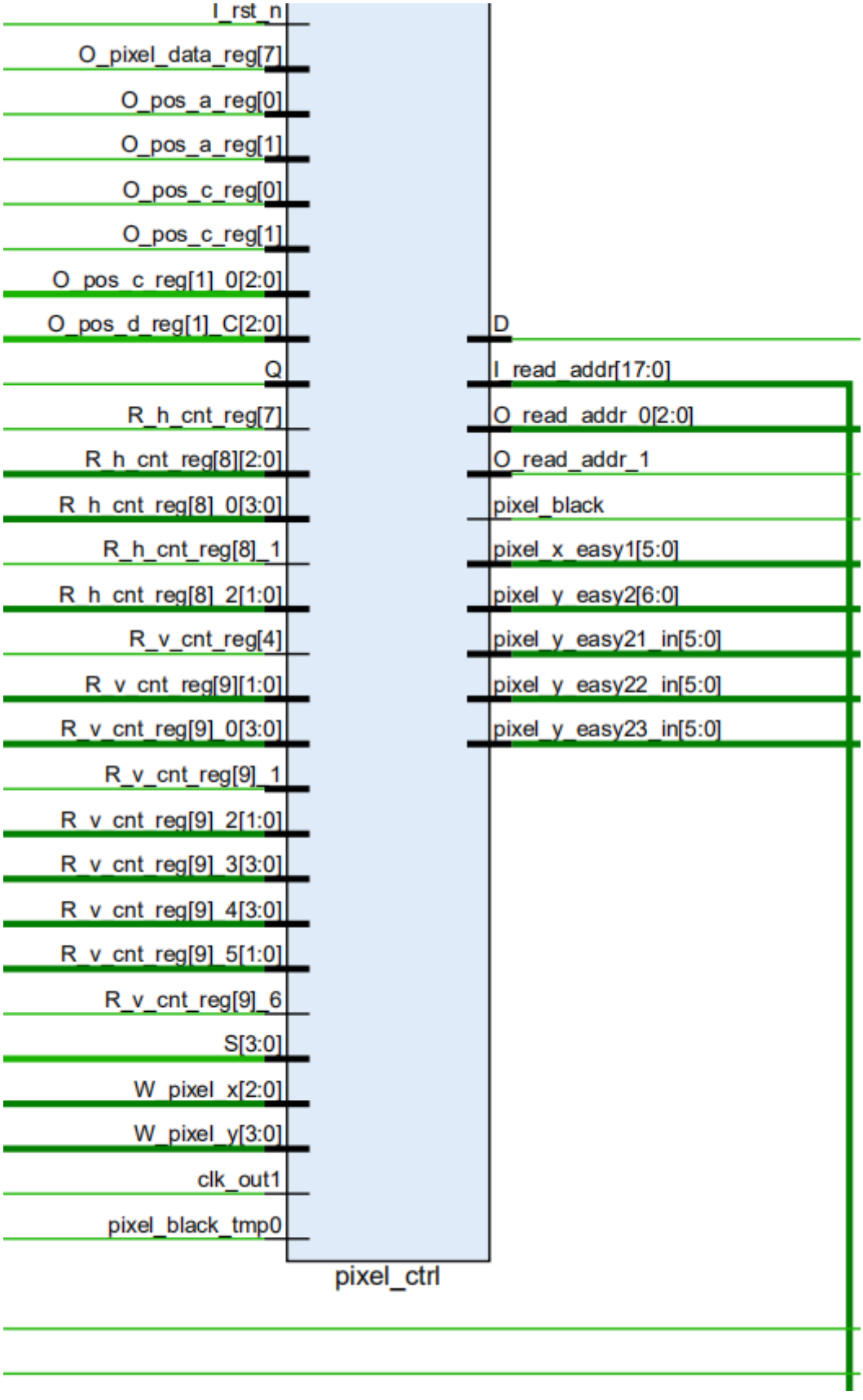



```
module img_mem_ctrl(
    input I_clk_25M,//时钟信号，频率为25MHz
    input I_rst_n,//复位信号
    input I_write_en, //写入数据的使能信号
    input [7:0] I_write_data, //要写入的数据
    input [17:0] I_read_addr,//读取地址
    output reg [7:0] O_pixel_data//读取的数据
);
```

2.2.1.4 pixel_ctrl

该模块用于

- 转换颜色信息（RGB332->RGB444)
- 根据当前屏幕像素坐标输出相应的内存地址

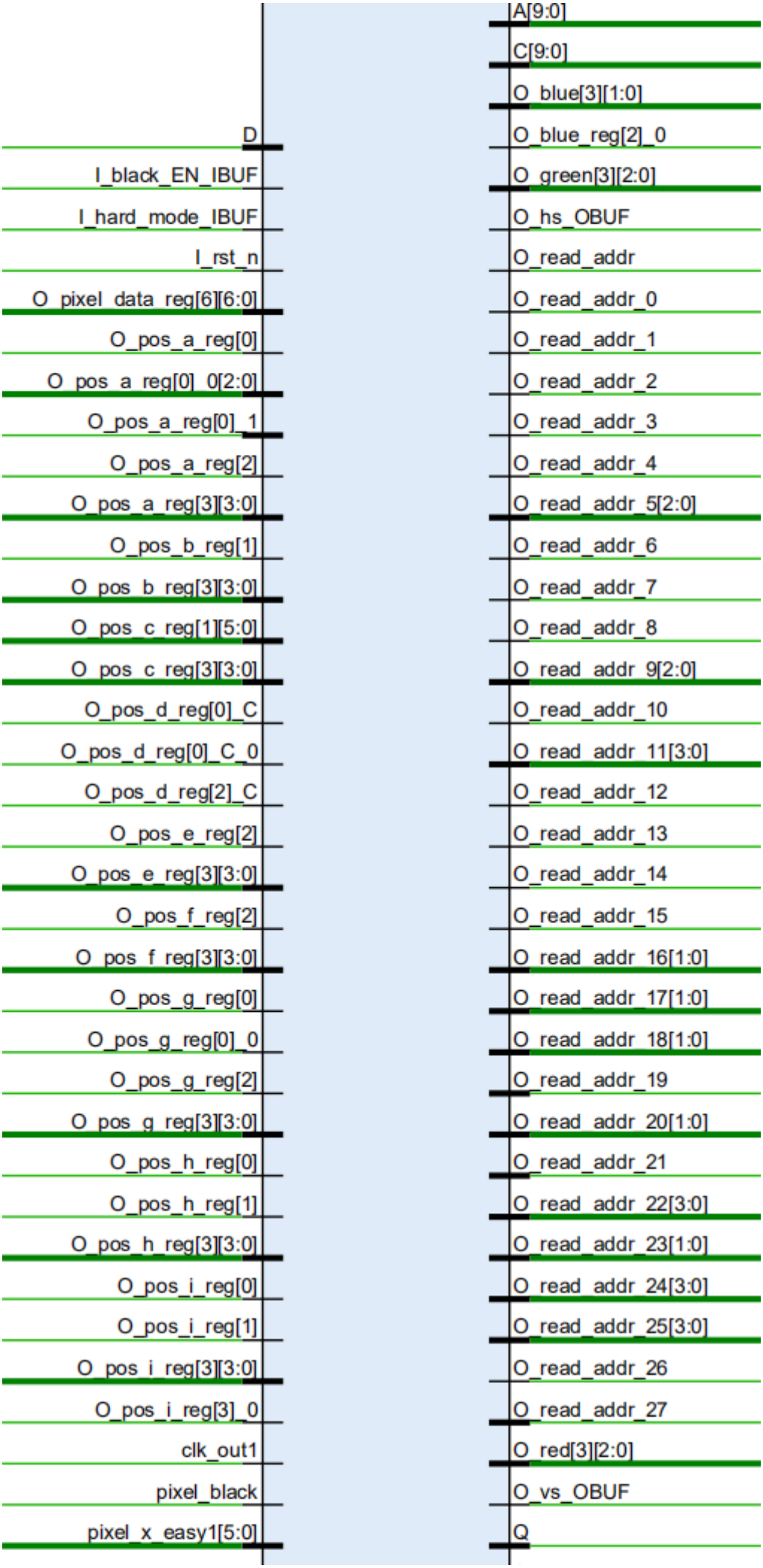


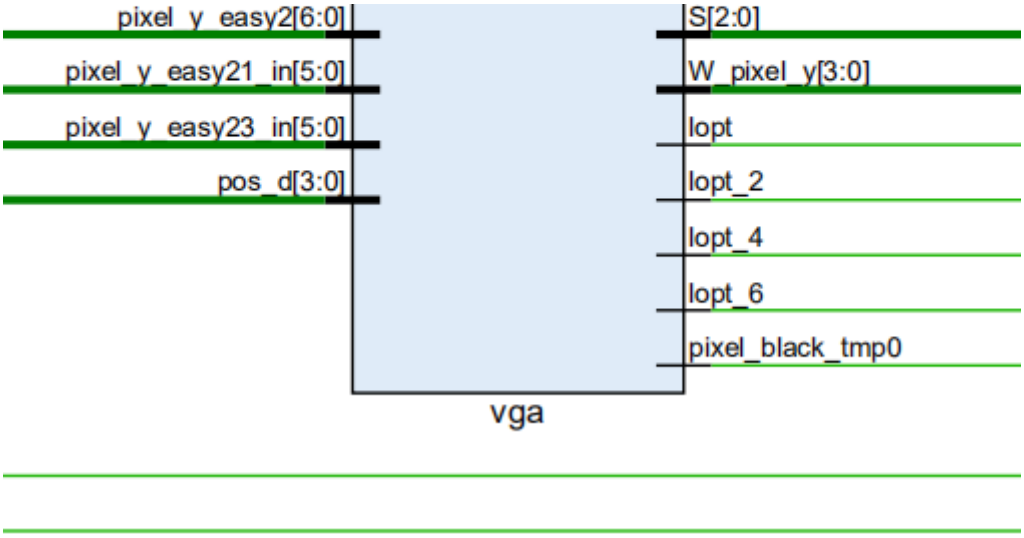

```
module pixel_ctrl(
    input I_clk_25M, // 时钟信号，频率为25MHz
    input I_rst_n,//复位信号
    input [9:0] I_pixel_x,//屏幕当前扫描到的像素点的横坐标
    input [9:0] I_pixel_y,//屏幕当前扫描到的像素点的纵坐标
    input [7:0] I_pixel_data,//屏幕当前扫描到的像素点颜色信息（RGB 332）
    input I_black_EN,//拉黑信号
    output [11:0] O_pixel_data,//屏幕当前扫描到的像素点颜色信息（RGB 444）

```

```
    //相应屏幕区域显示的图片区域
    input [3:0] I_pos_a,
    input [3:0] I_pos_b,
    input [3:0] I_pos_c,
    input [3:0] I_pos_d,
    input [3:0] I_pos_e,
    input [3:0] I_pos_f,
    input [3:0] I_pos_g,
    input [3:0] I_pos_h,
    input [3:0] I_pos_i,
    //内存读取地址
    output [17:0] O_read_addr,
    //困难模式
    input I_hard_mode
);
```

2.2.1.5 VGA

用于输出VGA显示所需要的相关信号

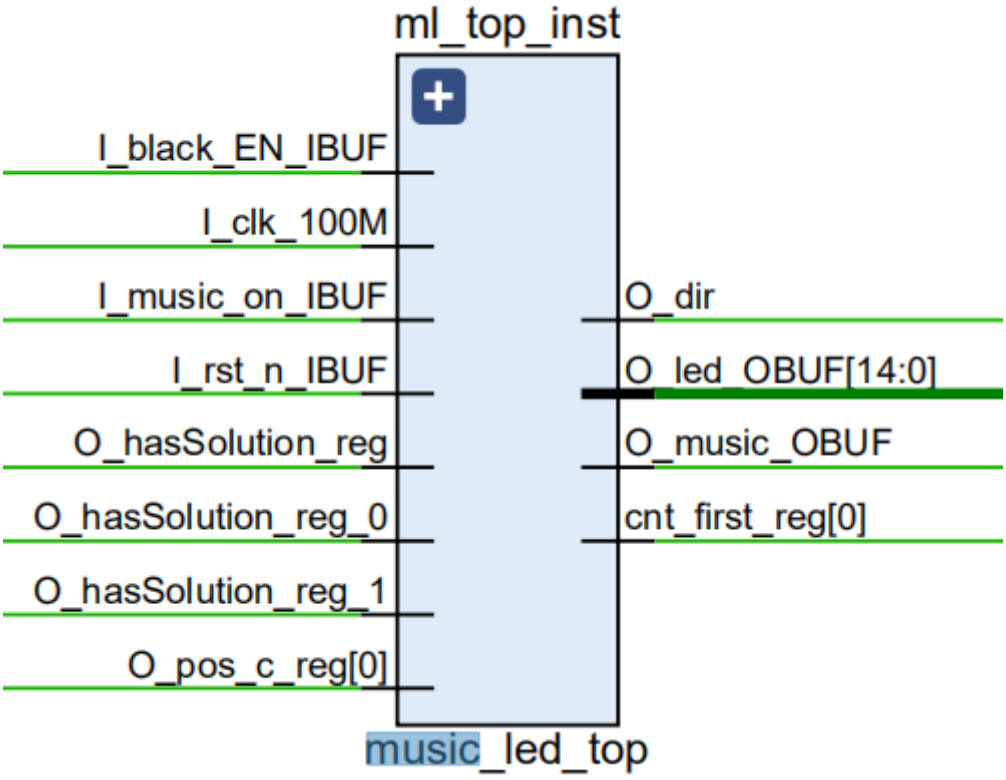

D
I_black_EN_IBUF
I_hard_mode_IBUF
I_rst_n
O_pixel_data_reg[6][6:0]
O_pos_a_reg[0]
O_pos_a_reg[0]_0[2:0]
O_pos_a_reg[0]_1
O_pos_a_reg[2]
O_pos_a_reg[3][3:0]
O_pos_b_reg[1]
O_pos_b_reg[3][3:0]
O_pos_c_reg[1][5:0]
O_pos_c_reg[3][3:0]
O_pos_d_reg[0]_C
O_pos_d_reg[0]_C_0
O_pos_d_reg[2]_C
O_pos_e_reg[2]
O_pos_e_reg[3][3:0]
O_pos_f_reg[2]
O_pos_f_reg[3][3:0]
O_pos_g_reg[0]
O_pos_g_reg[0]_0
O_pos_g_reg[2]
O_pos_g_reg[3][3:0]
O_pos_h_reg[0]
O_pos_h_reg[1]
O_pos_h_reg[3][3:0]
O_pos_i_reg[0]
O_pos_i_reg[1]
O_pos_i_reg[3][3:0]
O_pos_i_reg[3]_0
clk_out1
pixel_black
pixel_x_easy1[5:0]
A[9:0]
C[9:0]
O_blue[3][1:0]
O_blue_reg[2]_0
O_green[3][2:0]
O_hs_OBUF
O_read_addr
O_read_addr_0
O_read_addr_1
O_read_addr_2
O_read_addr_3
O_read_addr_4
O_read_addr_5[2:0]
O_read_addr_6
O_read_addr_7
O_read_addr_8
O_read_addr_9[2:0]
O_read_addr_10
O_read_addr_11[3:0]
O_read_addr_12
O_read_addr_13
O_read_addr_14
O_read_addr_15
O_read_addr_16[1:0]
O_read_addr_17[1:0]
O_read_addr_18[1:0]
O_read_addr_19
O_read_addr_20[1:0]
O_read_addr_21
O_read_addr_22[3:0]
O_read_addr_23[1:0]
O_read_addr_24[3:0]
O_read_addr_25[3:0]
O_read_addr_26
O_read_addr_27
O_red[3][2:0]
O_vs_OBUF
Q




```
module vga(
    input I_clk_25M,// 时钟信号，频率为25MHz
    input I_rst_n,//复位信号

    //像素点颜色信息（RGB 444）
    output reg [3:0] O_red,
    output reg [3:0] O_green,
    output reg [3:0] O_blue,

    output O_hs,//行信号
    output O_vs,//场信号


    input [11:0] I_pixel_data,//屏幕当前扫描到的像素点颜色信息（RGB 444）
    output [9:0] O_pixel_x,//屏幕当前扫描到的像素点的横坐标
    output [9:0] O_pixel_y,//屏幕当前扫描到的像素点的纵坐标
    output O_pixel_valid//当前扫描位置是否在显示区域内

);
```

2.2.1.6 music_led_top

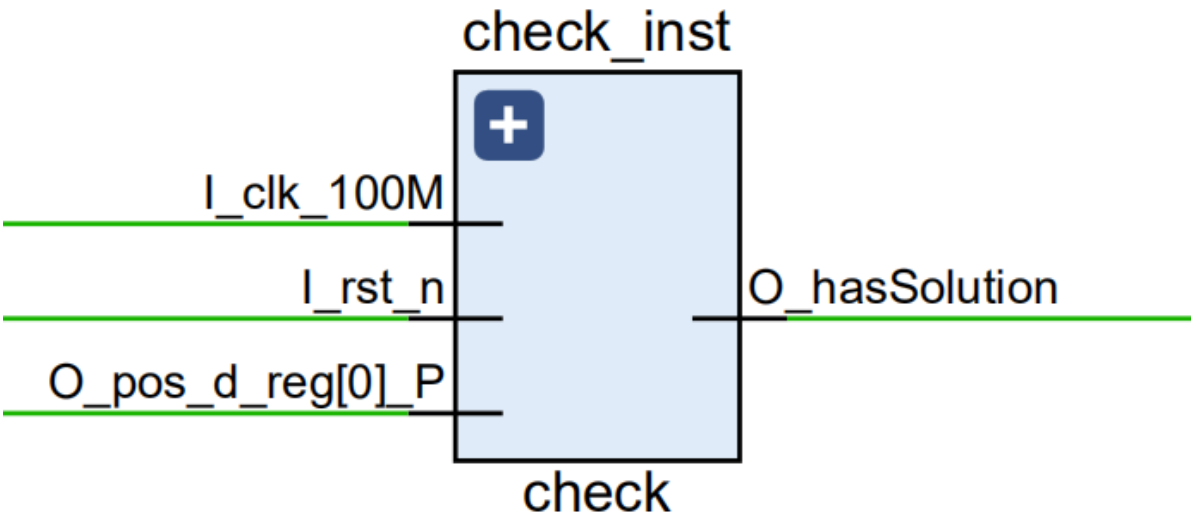
根据游戏状态控制led灯和蜂鸣器状态




```
module music_led_top(
    input I_clk,// 时钟信号
    input I_rst_n,//复位信号
    input I_black_EN,//拉黑信号
    input I_completed,//拼图完成信号
    input I_unsolvable,//拼图无解信号
    input I_music_on,//音乐开关
    output wire  O_music, //蜂鸣器输出
    output wire [15:0] O_led //led灯输出
);
```

2.2.1.7 check

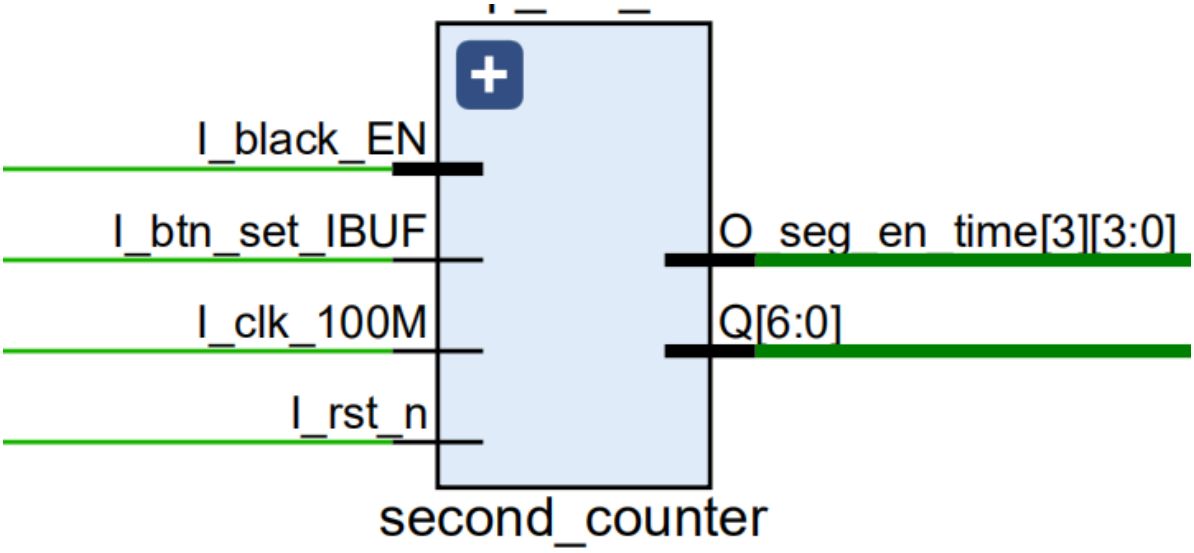
根据游戏状态判断是否有解



```
module check (
    //2*2游戏信息
    input [1:0] IA,
    input [1:0] IB,
    input [1:0] IC,
    input [1:0] ID,
    input I_clk,// 时钟信号
    input I_black_EN,//拉黑信号
    input I_rst_n,//复位信号
    output reg O_hasSolution//有解信号
);
```

2.2.1.8 counter_ss

计算显示的步数和时间




```
module counter_ss(
input clk, //时钟信号
input wire switch, //暂停信号
input set,//置位信号
input rst, //复位信号
output reg [7:0] num,//显示的数字
output reg [3:0] seg_en_s//步数显示的位置
    );
```

2.2.2状态迁移、事务处理流程图

2.3详细设计

2.3.1核心代码及说明

2.3.1.1 防抖

```
module anti_shake_single(
    input I_key,I_clk,I_rst_n,
    output O_key
    );
    wire key_changed1;
    reg [20:0] count;
//  reg [2:0] count; // for simulation
    reg t1, t_locked1, t2, t_locked2;

    always @(posedge I_clk or negedge I_rst_n)//每当时钟上升沿到来时，读取输入的值赋给t1
        if(~I_rst_n) t1 <= 0;
        else t1 <= I_key;

    always @(posedge I_clk or negedge I_rst_n)//延后一个时间周期，将t1的值赋给t_locked1
        if(~I_rst_n) t_locked1 <= 0;
        else t_locked1 <= t1;

    assign key_changed1 = ~t_locked1 & t1;//取t1和~t_locked1的交集，记录是否出现了输入值的变化

    always @(posedge I_clk or negedge I_rst_n)//在每个时钟上升沿计数，若有输入值的变化则清空计数
        if(~I_rst_n) count <= 0;
        else if(key_changed1) count <= 0;
        else count <= count + 1'b1;

    always @(posedge I_clk or negedge I_rst_n)//计数累计达到一定时间就将输入值赋给t2
        if(~I_rst_n) t2 <= 0;
        else if(count == 2000000)//20ms左右
//   else if(count == 2) // for simulation
            t2 <= I_key;

    always @(posedge I_clk or negedge I_rst_n)//延后一个时间周期，将t2的值赋给t_locked2
        if(~I_rst_n) t_locked2 <= 0;
        else t_locked2 <= t2;

    assign O_key = ~t_locked2 & t2;//取t2和~t_locked2的交集，记录是否出现了预期的信号变化

endmodule

```

仿真波形图：

```
module anti_shake_single_sim();
reg I_key,I_clk,I_rst_n;
wire O_key;
anti_shake_single u(I_key,I_clk,I_rst_n,O_key);
initial begin

```

```
I_clk = 1'b0;
I_rst_n = 1'b0;
I_key = 1'b0;
forever #5 I_clk =~I_clk;
end
initial fork
#3 I_rst_n = 1'b1;
#5 I_key = 1'b1;
#7 I_key = 1'b0;
#16 I_key = 1'b1;
#200 I_key = 1'b0;
#310 I_key = 1'b1;
#313 I_key = 1'b0;
#326 I_key = 1'b1;
#339 I_key = 1'b0;
#430 I_key = 1'b1;
#480 I_key = 1'b0;
#700 I_key = 1'b1;
#750 I_key = 1'b0;
join
endmodule

```

2.3.1.2 蜂鸣器音乐 流水灯 控制

2.3.1.2.1 led_music_control

该部分通过读取游戏状态来统一协调流水灯和蜂鸣器状态

```
//in led_music_control
`timescale 1ns / 1ps
module check (
    input [1:0] IA,
    input [1:0] IB,
    input [1:0] IC,
    input [1:0] ID,
    //是否开启3*3
    input hard,
    input I_clk,
    input I_black_EN,
    input I_rst_n,
    output reg O_hasSolution
);

always @(posedge I_clk,negedge I_rst_n) begin
    if(~I_rst_n) begin
        O_hasSolution <= 1'b0;
    end
    else if(hard) begin
        O_hasSolution <= 1'b1;
    end
    else if(I_black_EN) begin
        case ({IA,IB,IC,ID})
            8'b00_01_01_11 : O_hasSolution <= 1'b1;//1234
            8'b01_01_11_01 : O_hasSolution <= 1'b1;//3142
            8'b11_00_10_01 : O_hasSolution <= 1'b1;//4132
```

```
            8'b00_01_11_01 : o_hasSolution <= 1'b1;//1243
            8'b00_10_11_01 : o_hasSolution <= 1'b1;//1342
            8'b11_00_01_10 : o_hasSolution <= 1'b1;//4123
            8'b01_11_10_00 : o_hasSolution <= 1'b1;//2431
            8'b10_11_01_00 : o_hasSolution <= 1'b1;//3421
            8'b11_10_01_00 : o_hasSolution <= 1'b1;//4321
            8'b01_11_00_10 : o_hasSolution <= 1'b1;//2413
            8'b01_10_00_11 : o_hasSolution <= 1'b1;//2314
            8'b10_01_00_11 : o_hasSolution <= 1'b1;//3214

            default: o_hasSolution <= 1'b0;
        endcase

    end
    else  o_hasSolution <= 1'b0;
end

endmodule
```

2.3.1.2.2music

利用两个计数器控制节拍和频率，实现了可以自己播放想要的音乐的功能，利用led灯实现音乐律动

```
//in music.v
`timescale 1ns / 1ps

module Music(input clk, input wire[4:0] mode, output reg[0:0] music = 0,
output wire[6:0] o_led);

parameter do_low = 191110;
parameter re_low = 170259;
parameter me_low = 151685;
parameter fa_low = 143172;
parameter so_low = 127554;
parameter la_low = 113636;
parameter si_low = 101239;

parameter do = 93941;
parameter re = 85136;
parameter me = 75838;
parameter fa = 71582;
parameter so = 63776;
parameter la = 56818;
parameter si = 50618;

parameter do_high = 47778;
parameter re_high = 42567;
parameter me_high = 37921;
parameter fa_high = 36498;
parameter so_high = 31888;
parameter la_high = 28409;
parameter si_high = 25309;

parameter beat = 40 * 500000;
parameter gap =  10 * 500000;
parameter index_period = beat + gap;

```

```
parameter silence = beat<<9;

/*
0 - silence
1 - 7 low
8 - 14 meidum
15 - 21 high
*/
parameter Star = 210'b0000001000010010100101010010100101101011000000110001101011010110001100010000100000000010010101001010010110101101100011000000001001010100101001011010110110001100000000100001001010010101001010010110101100000011000110101101011000110001000010000;
parameter HappyBirthday = 130'b011111000000111110001100100000000000001101011100111110001100110110001111011111000000110001101011000111001110011110110001101001010010 1;
parameter MerryChristmas = 335'b011110111101111011101000001101011000110010011011111000010001100000000001100011000110101110011110111000000001110011110111101111011000000001111011110111010000011010110001100000000110101111100001000110010100011000100000011000110000000001100011100111110000100011000010000000000110100000000000110101101011100111110000011110111110000000110001100;

parameter ST_length = 42;
parameter HB_length = 26;
parameter MC_length = 67;

reg[29:0] freq =  beat;

reg[2000:0] melody = 0;
integer melody_length = 0;

integer frequency_count = 0;      // count1 control frequency
integer index_count = 0;      // count2 control beat;

integer index = 0;       // index control the location music playing

reg [0:0] isSilence = 0;
reg [0:0] isEnd = 0;
reg [0:0] isPeriodic = 0;

reg[4:0] last_mode = 0;

//歌曲选择
always @(posedge clk) begin

    if(mode != last_mode) begin
        last_mode = mode;
        isEnd = 0;
        index = 0;
        index_count = 0;

        if(mode >= 17) isPeriodic = 1;
        else begin isPeriodic = 0; melody_length = 1; end

        if(mode == 17) melody_length = MC_length;
        if(mode == 18) melody_length = HB_length;
        if(mode == 19) melody_length = ST_length;
```

```
81
82          case(mode)
83              17: melody = MerryChristmas;
84              18: melody = HappyBirthday;
85              19: melody = Star;
86              default : melody = mode;
87          endcase
88
89      end
90
91  //频率控制（播放对应音符）
92      if(frequency_count >= freq) begin
93          frequency_count = 0;
94          music = ~music;
95      end
96      else frequency_count = frequency_count + 1;
97
98  //节拍控制和循环播放
99
100     if(index_count > index_period) begin
101         index_count = 0;
102         index = index + 1;
103         if(index > melody_length && isPeriodic) begin
104             isEnd = 0;
105             index = 0;
106         end
107     end
108
109     index_count = index_count + 1;
110 end
111
112 //选取对于音符频率
113 always @ * begin
114 if(isEnd)
115 freq = silence;
116 else
117 case(melody[index * 5 +4 -:5])
118 5'd0 : freq = silence;
119 5'd1 : freq = do_low;
120 5'd2 : freq = re_low;
121 5'd3 : freq = me_low;
122 5'd4 : freq = fa_low;
123 5'd5 : freq = so_low;
124 5'd6 : freq = la_low;
125 5'd7 : freq = si_low;
126 5'd8 : freq = do;
127 5'd9 : freq = re;
128 5'd10 : freq = me;
129 5'd11: freq = fa;
130 5'd12 : freq = so;
131 5'd13 : freq = la;
132 5'd14 : freq = si;
133 5'd15 : freq = do_high;
134 5'd16 : freq = re_high;
135 5'd17 : freq = me_high;
136 5'd18 : freq = fa_high;
137 5'd19 : freq = so_high;
138 5'd20 : freq = la_high;
```

```
5'd21 : freq = si_high;
default : freq = silence;
endcase
end

//利用当前频率判断律动灯状态判断
always @(freq) begin
    case(freq)
    silence : o_led = 7'b0000_000;
    do_low  : o_led = 7'b0000_001;
    re_low  : o_led = 7'b0000_010;
    me_low  : o_led = 7'b0000_100;
    fa_low  : o_led = 7'b0001_000;
    so_low  : o_led = 7'b0010_000;
    la_low  : o_led = 7'b0100_000;
    si_low  : o_led = 7'b1000_000;

    do      : o_led = 7'b0000_001;
    re      : o_led = 7'b0000_010;
    me      : o_led = 7'b0000_100;
    fa      : o_led = 7'b0001_000;
    so      : o_led = 7'b0010_000;
    la      : o_led = 7'b0100_000;
    si      : o_led = 7'b1000_000;

    do_high : o_led = 7'b0000_001;
    re_high : o_led = 7'b0000_010;
    me_high : o_led = 7'b0000_100;
    fa_high : o_led = 7'b0001_000;
    so_high : o_led = 7'b0010_000;
    la_high : o_led = 7'b0100_000;
    si_high : o_led = 7'b1000_000;

    default : o_led = 7'b0000_000;
    endcase
end



endmodule


```

2.3.1.3 无解判断

2*2的拼图游戏一共有 4! (24) 种，在外部利用算法 (逆序对奇偶) 判断后以打表的形式在project中实现。

算法核心思路：

将矩阵从左到右，从上到下排成一个一维数组，设其逆序对的个数加上空白格在原矩阵所在的行列号之和P。若P(A)与P(B)的奇偶性相同，则两个矩阵可以通过拼图游戏进行转换

```
    `timescale 1ns / 1ps
    module check (
        input [3:0] IA,
        input [3:0] IB,
```

```
        input [3:0] IC,
        input [3:0] ID,
        //是否开启3*3
        input hard,
        input I_clk,
        input I_black_EN,
        input I_rst_n,
        output reg O_hasSolution
    );

    always @(posedge I_clk,negedge I_rst_n) begin
        if(~I_rst_n) begin
            O_hasSolution <= 1'b0;
        end
        else if(hard) begin
            O_hasSolution <= 1'b1;
        end
        else if(I_black_EN) begin
            case ({IA[1:0],IB[1:0],IC[1:0],ID[1:0]})
                16'b00_01_10_11 : O_hasSolution <= 1'b1;//1234
                16'b10_00_11_01 : O_hasSolution <= 1'b1;//3142
                16'b11_00_10_01 : O_hasSolution <= 1'b1;//4132
                16'b00_01_11_10 : O_hasSolution <= 1'b1;//1243
                16'b00_10_11_01 : O_hasSolution <= 1'b1;//1342
                16'b11_00_01_10 : O_hasSolution <= 1'b1;//4123
                16'b01_11_10_00 : O_hasSolution <= 1'b1;//2431
                16'b10_11_01_00 : O_hasSolution <= 1'b1;//3421
                16'b11_10_01_00 : O_hasSolution <= 1'b1;//4321
                16'b01_11_00_10 : O_hasSolution <= 1'b1;//2413
                16'b01_10_00_11 : O_hasSolution <= 1'b1;//2314
                16'b10_01_00_11 : O_hasSolution <= 1'b1;//3214

                default: O_hasSolution <= 1'b0;
            endcase

        end
        else  O_hasSolution <= 1'b0;
    end

    endmodule
```

2.3.1.4 计时器

计时器：

1. 输入输出端口

```
module second_counter(
input clk, //时钟信号
input wire switch, //暂停信号
input set,//置位信号
input rst, //复位信号
output reg [7:0] num,//显示的数字
output reg [3:0] seg_en//显示的位置
    );
```

2. 参数

```
parameter     SEG_NUM0 = 8'b1100_0000,
              SEG_NUM1 = 8'b1111_1001,
              SEG_NUM2 = 8'b1010_0100,
              SEG_NUM3 = 8'b1011_0000,
              SEG_NUM4 = 8'b1001_1001,
              SEG_NUM5 = 8'b1001_0010,
              SEG_NUM6 = 8'b1000_0010,
              SEG_NUM7 = 8'b1111_1000,
              SEG_NUM8 = 8'b1000_0000,
              SEG_NUM9 = 8'b1001_0000;
parameter     DUAN_3 = 4'b0111,
              DUAN_2 = 4'b1011,
              DUAN_1 = 4'b1101,
              DUAN_0 = 4'b1110;
```

3.分频（产生周期为1ms的时钟线）

```
always @(posedge clk) begin
       if(fenpin == 100000)
           fenpin <= 0;
        else
           fenpin <= fenpin + 1'b1;
end

always @ (posedge clk) begin
   if(fenpin < 50000)
        clk_1ms  <= 0;
    else
        clk_1ms  <= 1;
end
```

4.状态机：

```
always @ (posedge clk or negedge rst) begin
if (~rst) begin
state <= 1;
cen <= 0;
end
else if (set) begin
state <= 1;
cen <= 0;
end
else begin
 case(state)
    0: begin
        cen <= 1;
        if(!switch)
         state <= 1;
         else
          state <= 0;
        end
     1: begin
        cen <= 0;
        if(switch)
         state <= 0;
         else
```

```
          state <= 1;
        end
 endcase
    end
end
```

5.时间计算

```
always @ (posedge clk_1ms or negedge rst) begin
if(~rst)begin
ms1 <= 0;
ms100 <= 0;
ms10 <= 0;
s1 <= 0;
s2 <= 0;
s3 <= 0;
s4 <= 0;
end
else if(set)begin
ms1 <= 0;
ms100 <= 0;
ms10 <= 0;
s1 <= 0;
s2 <= 0;
s3 <= 0;
s4 <= 0;
end
else begin
if(cen) begin
     case(ms1)
     4'b1001: ms1 <= 4'b0000;
     default:  ms1 <= ms1 + 1'b1;
     endcase
     case(ms10)
     4'b1001: begin
               if(ms1 == 4'b1001)
                 ms10 <= 4'b0000;
                 end
     default: begin
               if(ms1 == 4'b1001)
                 ms10 <= ms10 + 1'b1;
                 end
     endcase
     case(ms100)
     4'b1001: begin
               if(ms1 == 4'b1001 && ms10 == 4'b1001)
                 ms100 <= 4'b0000;
                 end
     default: begin
               if(ms1 == 4'b1001 && ms10 == 4'b1001)
                 ms100 <= ms100 + 1'b1;
                 end
     endcase
     case(s1)
     4'b1001: begin
               if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 == 4'b1001)
                 s1 <= 4'b0000;
```

```
                end
     default: begin
               if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 == 4'b1001)
                 s1 <= s1 + 1'b1;
                 end
     endcase
     case(s2)
          4'b1001: begin
                     if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 == 4'b1001
&& s1==4'b1001)
                       s2 <= 4'b0000;
                       end
          default: begin
                     if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 == 4'b1001
&& s1==4'b1001)
                       s2 <= s2 + 1'b1;
                       end
          endcase
     case(s3)
               4'b1001: begin
                          if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 ==
4'b1001 && s1==4'b1001 && s2==4'b1001)
                            s3 <= 4'b0000;
                            end
               default: begin
                          if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 ==
4'b1001 && s1==4'b1001 && s2==4'b1001)
                            s3 <= s3 + 1'b1;
                            end
               endcase
     case(s4)
                          4'b1001: begin
                                     if(ms1 == 4'b1001 && ms10 == 4'b1001 &&
ms100 == 4'b1001 && s1==4'b1001 && s2==4'b1001 && s3==4'b1001)
                                       s4 <= 4'b0000;
                                       end
                          default: begin
                                     if(ms1 == 4'b1001 && ms10 == 4'b1001 &&
ms100 == 4'b1001 && s1==4'b1001 && s2==4'b1001 && s3==4'b1001)
                                       s4 <= s4 + 1'b1;
                                       end
                          endcase
end
end
end
```

6.七段数码管显示

```
always @ (posedge clk) begin
  case(count[19:18])
  2'b00: begin
          seg_en_time <= DUAN_3;
             case(s4)
             4'b0000: num <= SEG_NUM0;
             4'b0001: num <= SEG_NUM1;
             4'b0010: num <= SEG_NUM2;
             4'b0011: num <= SEG_NUM3;
```

```
            4'b0100: num <= SEG_NUM4;
            4'b0101: num <= SEG_NUM5;
            4'b0110: num <= SEG_NUM6;
            4'b0111: num <= SEG_NUM7;
            4'b1000: num <= SEG_NUM8;
            4'b1001: num <= SEG_NUM9;
            endcase
          end

  2'b01: begin
          seg_en_time <= DUAN_2;
            case(s3)
            4'b0000: num <= SEG_NUM0;
            4'b0001: num <= SEG_NUM1;
            4'b0010: num <= SEG_NUM2;
            4'b0011: num <= SEG_NUM3;
            4'b0100: num <= SEG_NUM4;
            4'b0101: num <= SEG_NUM5;
            4'b0110: num <= SEG_NUM6;
            4'b0111: num <= SEG_NUM7;
            4'b1000: num <= SEG_NUM8;
            4'b1001: num <= SEG_NUM9;
            endcase
          end
  2'b10: begin
          seg_en_time <= DUAN_1;
            case(s2)
            4'b0000: num <= SEG_NUM0;
            4'b0001: num <= SEG_NUM1;
            4'b0010: num <= SEG_NUM2;
            4'b0011: num <= SEG_NUM3;
            4'b0100: num <= SEG_NUM4;
            4'b0101: num <= SEG_NUM5;
            4'b0110: num <= SEG_NUM6;
            4'b0111: num <= SEG_NUM7;
            4'b1000: num <= SEG_NUM8;
            4'b1001: num <= SEG_NUM9;
            endcase
          end
  2'b11: begin
          seg_en_time <= DUAN_0;
            case(s1)
            4'b0000: num <= SEG_NUM0;
            4'b0001: num <= SEG_NUM1;
            4'b0010: num <= SEG_NUM2;
            4'b0011: num <= SEG_NUM3;
            4'b0100: num <= SEG_NUM4;
            4'b0101: num <= SEG_NUM5;
            4'b0110: num <= SEG_NUM6;
            4'b0111: num <= SEG_NUM7;
            4'b1000: num <= SEG_NUM8;
            4'b1001: num <= SEG_NUM9;
            endcase
          end
  endcase

end
```

2.3.1.5 game control

玩家可能进行非法的移动操作，该模块阻止了该情况的发生。若移动后会超出边界的话，将不进行移动。

同时，该模块会输出移动后各区块位置显示的图片区域，以及拼图游戏是否完成。

此外，该模块还实现了随机置位和随机移动的功能。

当拼图为2x2，随机置位开关（`I_rand_set_EN`）关闭时，按下置位按钮会根据五个拨码开关控制不同图像区块的排列组合；

当随机置位开关（`I_rand_set_EN`）开启时，按下置位按钮会进行随机置位操作，即随机置位到 24 种状态的任意一种；

当拼图为3x3，按下置位按钮只会恢复初始状态，即拼图完成状态。3x3拼图的打乱需要通过开启随机移动开关实现。

```
module game_control(
    input [4:0] I_num,
    input I_clk,
    input I_rst_n,
    input I_btn_set,
    input I_btn_left,
    input I_btn_right,
    input I_btn_up,
    input I_btn_down,
    output reg [3:0] O_pos_a,
    output reg [3:0] O_pos_b,
    output reg [3:0] O_pos_c,
    output reg [3:0] O_pos_d,
    output reg [3:0] O_pos_e,
    output reg [3:0] O_pos_f,
    output reg [3:0] O_pos_g,
    output reg [3:0] O_pos_h,
    output reg [3:0] O_pos_i,
    input I_black_EN,
    input I_rand_move_EN,
    input I_rand_set_EN,
    input I_hard_mode,
    output O_completed,
    output reg O_moving
);

wire num_valid;
wire [1:0] pos_a_native;
wire [1:0] pos_b_native;
wire [1:0] pos_c_native;
wire [1:0] pos_d_native;
wire [1:0] pos_a_rand;
wire [1:0] pos_b_rand;
wire [1:0] pos_c_rand;
wire [1:0] pos_d_rand;
wire game_start = I_black_EN;

assign O_completed = I_hard_mode
```

```
    ? ((O_pos_a == 4'b0000) && (O_pos_b == 4'b0001) && (O_pos_c == 4'b0010) && (O_pos_d == 4'b0100) && (O_pos_e == 4'b0101) && (O_pos_f == 4'b0110) && (O_pos_g == 4'b1000) && (O_pos_h == 4'b1001) && (O_pos_h == 4'b1010))
    : ((O_pos_a[1:0] == 2'b00) && (O_pos_b[1:0] == 2'b01) && (O_pos_c[1:0] == 2'b10) && (O_pos_d[1:0] == 2'b11));

wire W_left, W_right, W_up, W_down;
wire W_left_rand, W_right_rand, W_up_rand, W_down_rand;

random_move rand_mv_inst(
    .I_clk(I_clk),
    .I_rst_n(I_rst_n),
    .I_en(I_rand_move_EN),
    .O_mv_left(W_left_rand),
    .O_mv_right(W_right_rand),
    .O_mv_up(W_up_rand),
    .O_mv_down(W_down_rand)
);

rand_pos rand_pos_inst(
    .I_clk(I_clk),
    .O_pos_a(pos_a_rand),
    .O_pos_b(pos_b_rand),
    .O_pos_c(pos_c_rand),
    .O_pos_d(pos_d_rand)
);

assign W_left = I_btn_left | W_left_rand;
assign W_right = I_btn_right | W_right_rand;
assign W_up = I_btn_up | W_up_rand;
assign W_down = I_btn_down | W_down_rand;

permutation perm_inst(
    .I_num(I_num),
    .O_pos_a(pos_a_native),
    .O_pos_b(pos_b_native),
    .O_pos_c(pos_c_native),
    .O_pos_d(pos_d_native),
    .O_num_valid(num_valid)
);

always @(posedge I_clk, negedge I_rst_n) begin
    if(~I_rst_n) begin
        O_moving <= 0;
        if(I_hard_mode) begin
            O_pos_a <= 4'b0000;
            O_pos_b <= 4'b0001;
            O_pos_c <= 4'b0010;

            O_pos_d <= 4'b0100;
            O_pos_e <= 4'b0101;
            O_pos_f <= 4'b0110;

            O_pos_g <= 4'b1000;
            O_pos_h <= 4'b1001;
            O_pos_i <= 4'b1010;
        end
        else begin
```

```
            O_pos_a <= 4'b0000;
            O_pos_b <= 4'b0001;
            O_pos_c <= 4'b0010;
            O_pos_d <= 4'b0011;
        end
    end
    else if(I_btn_set) begin
        O_moving <= 0;
        if(I_hard_mode) begin
            O_pos_a <= 4'b0000;
            O_pos_b <= 4'b0001;
            O_pos_c <= 4'b0010;

            O_pos_d <= 4'b0100;
            O_pos_e <= 4'b0101;
            O_pos_f <= 4'b0110;

            O_pos_g <= 4'b1000;
            O_pos_h <= 4'b1001;
            O_pos_i <= 4'b1010;
        end
        else if(I_rand_set_EN) begin
            O_pos_a <= pos_a_rand;
            O_pos_b <= pos_b_rand;
            O_pos_c <= pos_c_rand;
            O_pos_d <= pos_d_rand;
        end
        else if(num_valid) begin
            O_pos_a <= pos_a_native;
            O_pos_b <= pos_b_native;
            O_pos_c <= pos_c_native;
            O_pos_d <= pos_d_native;
        end
    end
    else if(game_start) begin
        if(I_hard_mode) begin
            casex ({W_left, W_right, W_up, W_down})
            4'b1xxx: begin
                if(O_pos_i == 4'b1000) begin
                    O_pos_h <= O_pos_i;
                    O_pos_i <= O_pos_h;
                    O_moving <= 1;
                end
                else if(O_pos_h == 4'b1000) begin
                    O_pos_g <= O_pos_h;
                    O_pos_h <= O_pos_g;
                    O_moving <= 1;
                end
                else if(O_pos_f == 4'b1000) begin
                    O_pos_f <= O_pos_e;
                    O_pos_e <= O_pos_f;
                    O_moving <= 1;
                end
                else if(O_pos_e == 4'b1000) begin
                    O_pos_e <= O_pos_d;
                    O_pos_d <= O_pos_e;
                    O_moving <= 1;
                end
```

```
                else if(O_pos_c == 4'b1000) begin
                    O_pos_c <= O_pos_b;
                    O_pos_b <= O_pos_c;
                    O_moving <= 1;
                end
                else if(O_pos_b == 4'b1000) begin
                    O_pos_b <= O_pos_a;
                    O_pos_a <= O_pos_b;
                    O_moving <= 1;
                end
                else O_moving <= 0;
            end
            4'b01xx: begin
                if(O_pos_a == 4'b1000) begin
                    O_pos_b <= O_pos_a;
                    O_pos_a <= O_pos_b;
                    O_moving <= 1;
                end
                else if(O_pos_b == 4'b1000) begin
                    O_pos_c <= O_pos_b;
                    O_pos_b <= O_pos_c;
                    O_moving <= 1;
                end
                else if(O_pos_d == 4'b1000) begin
                    O_pos_d <= O_pos_e;
                    O_pos_e <= O_pos_d;
                    O_moving <= 1;
                end
                else if(O_pos_e == 4'b1000) begin
                    O_pos_e <= O_pos_f;
                    O_pos_f <= O_pos_e;
                    O_moving <= 1;
                end
                else if(O_pos_g == 4'b1000) begin
                    O_pos_g <= O_pos_h;
                    O_pos_h <= O_pos_g;
                    O_moving <= 1;
                end
                else if(O_pos_h == 4'b1000) begin
                    O_pos_i <= O_pos_h;
                    O_pos_h <= O_pos_i;
                    O_moving <= 1;
                end
                else O_moving <= 0;
            end
            4'b001x: begin
                if(O_pos_d == 4'b1000) begin
                    O_pos_a <= O_pos_d;
                    O_pos_d <= O_pos_a;
                    O_moving <= 1;
                end
                else if(O_pos_e == 4'b1000) begin
                    O_pos_e <= O_pos_b;
                    O_pos_b <= O_pos_e;
                    O_moving <= 1;
                end
                else if(O_pos_f == 4'b1000) begin
                    O_pos_f <= O_pos_c;
```

```
                    O_pos_c <= O_pos_f;
                    O_moving <= 1;
                end
                else if(O_pos_g == 4'b1000) begin
                    O_pos_g <= O_pos_d;
                    O_pos_d <= O_pos_g;
                    O_moving <= 1;
                end
                else if(O_pos_h == 4'b1000) begin
                    O_pos_h <= O_pos_e;
                    O_pos_e <= O_pos_h;
                    O_moving <= 1;
                end
                else if(O_pos_i == 4'b1000) begin
                    O_pos_i <= O_pos_f;
                    O_pos_f <= O_pos_i;
                    O_moving <= 1;
                end
                else O_moving <= 0;
            end
            4'b0001: begin
                if(O_pos_a == 4'b1000) begin
                    O_pos_a <= O_pos_d;
                    O_pos_d <= O_pos_a;
                    O_moving <= 1;
                end
                else if(O_pos_b == 4'b1000) begin
                    O_pos_b <= O_pos_e;
                    O_pos_e <= O_pos_b;
                    O_moving <= 1;
                end
                else if(O_pos_c == 4'b1000) begin
                    O_pos_c <= O_pos_f;
                    O_pos_f <= O_pos_c;
                    O_moving <= 1;
                end
                else if(O_pos_d == 4'b1000) begin
                    O_pos_d <= O_pos_g;
                    O_pos_g <= O_pos_d;
                    O_moving <= 1;
                end
                else if(O_pos_e == 4'b1000) begin
                    O_pos_h <= O_pos_e;
                    O_pos_e <= O_pos_h;
                    O_moving <= 1;
                end
                else if(O_pos_f == 4'b1000) begin
                    O_pos_f <= O_pos_i;
                    O_pos_i <= O_pos_f;
                    O_moving <= 1;
                end
                else O_moving <= 0;
            end
            default: O_moving <= 0;
            endcase
        end
        else begin
            casex ({W_left, W_right, W_up, W_down})
```

```
            4'b1xxx: begin
                if(O_pos_b[1:0] == 2'b10) begin
                    O_pos_b <= O_pos_a;
                    O_pos_a <= O_pos_b;
                    O_moving <= 1;
                end
                else if(O_pos_d[1:0] == 2'b10) begin
                    O_pos_d <= O_pos_c;
                    O_pos_c <= O_pos_d;
                    O_moving <= 1;
                end
                else O_moving <= 0;
            end
            4'b01xx: begin
                    if(O_pos_a[1:0] == 2'b10) begin
                    O_pos_b <= O_pos_a;
                    O_pos_a <= O_pos_b;
                    O_moving <= 1;
                end
                else if(O_pos_c[1:0] == 2'b10) begin
                    O_pos_c <= O_pos_d;
                    O_pos_d <= O_pos_c;
                    O_moving <= 1;
                end
                else O_moving <= 0;
            end
            4'b001x: begin
                if(O_pos_c[1:0] == 2'b10) begin
                    O_pos_a <= O_pos_c;
                    O_pos_c <= O_pos_a;
                    O_moving <= 1;
                end
                else if(O_pos_d[1:0] == 2'b10) begin
                    O_pos_b <= O_pos_d;
                    O_pos_d <= O_pos_b;
                    O_moving <= 1;
                end
                else O_moving <= 0;
            end
            4'b0001: begin
                if(O_pos_a[1:0] == 2'b10) begin
                    O_pos_c <= O_pos_a;
                    O_pos_a <= O_pos_c;
                    O_moving <= 1;
                end
                else if(O_pos_b[1:0] == 2'b10) begin
                    O_pos_d <= O_pos_b;
                    O_pos_b <= O_pos_d;
                    O_moving <= 1;
                end
                else O_moving <= 0;
            end
            default: O_moving <= 0;
            endcase

        end
    end
end
```

```
endmodule
```

2.3.1.6 VGA

VGA用到的分辨率为640*480@60Hz, 因此VGA使用的时钟频率为25MHz。该模块输出当前扫描像素点坐标，输入当前像素点颜色信息，并将之转换成RGB通道输出。

```
module vga(
    input I_clk_25M,
    input I_rst_n,
    output reg [3:0] O_red,
    output reg [3:0] O_green,
    output reg [3:0] O_blue,
    output O_hs,
    output O_vs,

    input [11:0] I_pixel_data,
    output [9:0] O_pixel_x,
    output [9:0] O_pixel_y,
    output O_pixel_valid

);
    parameter   C_H_SYNC_PULSE      =   10'd96  ,
                C_H_BACK_PORCH      =   10'd48  ,
                C_H_ACTIVE_TIME     =   10'd640 ,
                C_H_FRONT_PORCH     =   10'd16  ,
                C_H_LINE_PERIOD     =   10'd800 ;

    parameter   C_V_SYNC_PULSE      =   10'd2   ,
                C_V_BACK_PORCH      =   10'd33  ,
                C_V_ACTIVE_TIME     =   10'd480 ,
                C_V_FRONT_PORCH     =   10'd10  ,
                C_V_FRAME_PERIOD    =   10'd525 ;

    reg [9:0] R_h_cnt;
    reg [9:0] R_v_cnt;

    wire W_active_flag;

    //行时序
    always @(posedge I_clk_25M or negedge I_rst_n) begin
        if(!I_rst_n) R_h_cnt <= 10'd0;
        else if(R_h_cnt == C_H_LINE_PERIOD - 1) R_h_cnt <= 10'd0;
        else R_h_cnt <= R_h_cnt + 1;
    end

    //场时序
    always @(posedge I_clk_25M or negedge I_rst_n) begin
        if(!I_rst_n) R_v_cnt <= 10'd0;
        else if (R_v_cnt == C_V_FRAME_PERIOD - 1) R_v_cnt <= 10'd0;
        else if (R_h_cnt == C_H_LINE_PERIOD - 1) R_v_cnt <= R_v_cnt + 1;
        else R_v_cnt <= R_v_cnt;
    end

    //行同步信号 & 场同步信号 低电平有效
    assign O_hs = (R_h_cnt < C_H_SYNC_PULSE) ? 1'b0 : 1'b1;
```

```
    assign O_vs = (R_v_cnt < C_V_SYNC_PULSE) ? 1'b0 : 1'b1;

    assign W_active_flag = (R_h_cnt >= (C_H_SYNC_PULSE + C_H_BACK_PORCH)) &&
                           (R_h_cnt < (C_H_SYNC_PULSE + C_H_BACK_PORCH + C_H_ACTIVE_TIME)) &&
                           (R_v_cnt >= (C_V_SYNC_PULSE + C_V_BACK_PORCH)) &&
                           (R_v_cnt < (C_V_SYNC_PULSE + C_V_BACK_PORCH + C_V_ACTIVE_TIME));


    assign O_pixel_x = (R_h_cnt >= C_H_SYNC_PULSE + C_H_BACK_PORCH) ? (R_h_cnt - C_H_SYNC_PULSE - C_H_BACK_PORCH) : 0;
    assign O_pixel_y = (R_v_cnt >= C_V_SYNC_PULSE + C_V_BACK_PORCH) ? (R_v_cnt - C_V_SYNC_PULSE - C_V_BACK_PORCH) : 0;
    assign O_pixel_valid = W_active_flag;


    always @(posedge I_clk_25M, negedge I_rst_n) begin
        if(!I_rst_n) begin
            O_red <= 4'b0;
            O_green <= 4'b0;
            O_blue <= 4'b0;
        end
        else if(W_active_flag) begin
            O_blue <= I_pixel_data[3:0];
            O_green <= I_pixel_data[7:4];
            O_red <= I_pixel_data[11:8];
        end
        else begin
            O_red <= 4'b0;
            O_green <= 4'b0;
            O_blue <= 4'b0;
        end
    end

endmodule
```

2.4约束文件

2.4.1主要输入端口

2.4.1.1 时钟信号和复位信号

2.4.1.2 移动、置位按钮

2.4.1.3 黑块拉高拨码开关

2.4.1.4 置位拨码开关

2.4.2主要输出端口

2.4.2.1 VGA相关输出

2.4.2.2 LED灯

2.4.2.3 七段数码管显示

2.4.2.4 蜂鸣器

3.总结及优化

3.1问题及解决方案

3.1.1 防抖模块

问题

加入防抖模块后，游戏运行过程中，黑块的移动会出现卡顿的现象，甚至多次点击都没有反应。

debug历程

由于防抖模块已经完成了仿真验证，初步排除是防抖模块本身的错误。

接下来怀疑是计算错误，防抖预期是对于20ms左右的信号扰动进行消除，反复验算后基本排除。（在10ns为周期的时间线中，计数2000000次，20000000ns=20ms）

由于游戏控制模块接收的是经过防抖模块处理后的信号，思考防抖模块和游戏控制之间连接的关系。

经过检查，游戏控制模块中的时钟信号为25M，而经过防抖模块输出的信号只会输出100M的一个周期时长，因此相对于游戏控制模块检查输入信号的频率来说，防抖模块输出的信号过于“稀疏”了。

具体来说，对于游戏模块来说，每40ns才会去读取一次输入信号，而防抖模块处理后的信号持续时长只有10ns，因此，游戏模块很容易错过这样的信息。

解决方案

统一了两个模块的时钟信号，防抖模块中的时间线也改为频率25M，将counter除以4，保证仍然消除的是不到20ms的信号抖动，输出的信号持续时长也变为了40ns，重新上板测试后，问题解决。

3.1.3 音乐模块

问题

在加入自己的编曲的音乐，蜂鸣器的播放出现混乱状态

debug经历

仔细研究编曲规则编写c++程序，完成简谱到二进制编码的转换，初步排除是编曲规则理解错误

后来发现是因为Verilog中读入的二进制编码的最低位在最右侧，而c++输出的字符串结果将第一位音符放在了最左侧，因此整个曲谱被颠倒过来了，呈现出混乱状态。

解决方案

修改c++代码，将第一位音符最后输出。最后进行上板测试后音乐可以正常播放了

3.1.3 音乐模块

问题

在加入自己的编曲的音乐，蜂鸣器的播放出现混乱状态

debug经历

仔细研究编曲规则编写c++程序，完成简谱到二进制编码的转换，初步排除是编曲规则理解错误

后来发现是因为Verilog中读入的二进制编码的最低位在最右侧，而c++输出的字符串结果将第一位音符放在了最左侧，因此整个曲谱被颠倒过来了，呈现出混乱状态。

解决方案

修改c++代码，将第一位音符最后输出。最后进行上板测试后音乐可以正常播放了

3.2系统特色

- 根据游戏状态
- 拥有多种随机方式：1.随机置位 2.随机移动
- 通过游戏状态显示特定流水灯和播放音乐并显示音乐律动

3.3优化方向

- 代码中有很多的`if...else if...`，可以将其用`case`进行替代。
- 图片画质升级
- 游戏逻辑升级，方便实现mxn游戏模式