

Project

Project

1.开发计划

1.1小组选题

VGA拼图游戏

1.2成员分工及百分比

1.3进度安排

1.4执行记录

2.设计

2.1需求分析

2.1.1系统功能

2.1.2使用的输入输出设备

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

2.2.1.2 game_control

2.2.1.3 img_mem_ctrl

2.2.1.4 pixel_ctrl

2.2.1.5 VGA

2.2.1.6 music_led_top

2.2.1.7 check

2.2.1.8 counter_ss

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

2.3.1.2 蜂鸣器音乐 流水灯 控制

2.3.1.2.1 led_music_control

2.3.1.2.2music

2.3.1.3 无解判断

2.3.1.4 计时器

2.3.1.5 game control

2.3.1.6 VGA

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

debug历程

解决方案

3.1.3 音乐模块

问题

- debug经历
 - 解决方案
- 3.1.3 音乐模块
 - 问题
 - debug经历
 - 解决方案
- 3.2系统特色
- 3.3优化方向

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
第十四周	完成了顶层模块设计，开始小模块的书写和测试
第十五周	合并整合了代码，一人调试，剩余两人进行bonus的探究
第十六周	整合bonus加最后的调试，提前答辩
第十七周	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

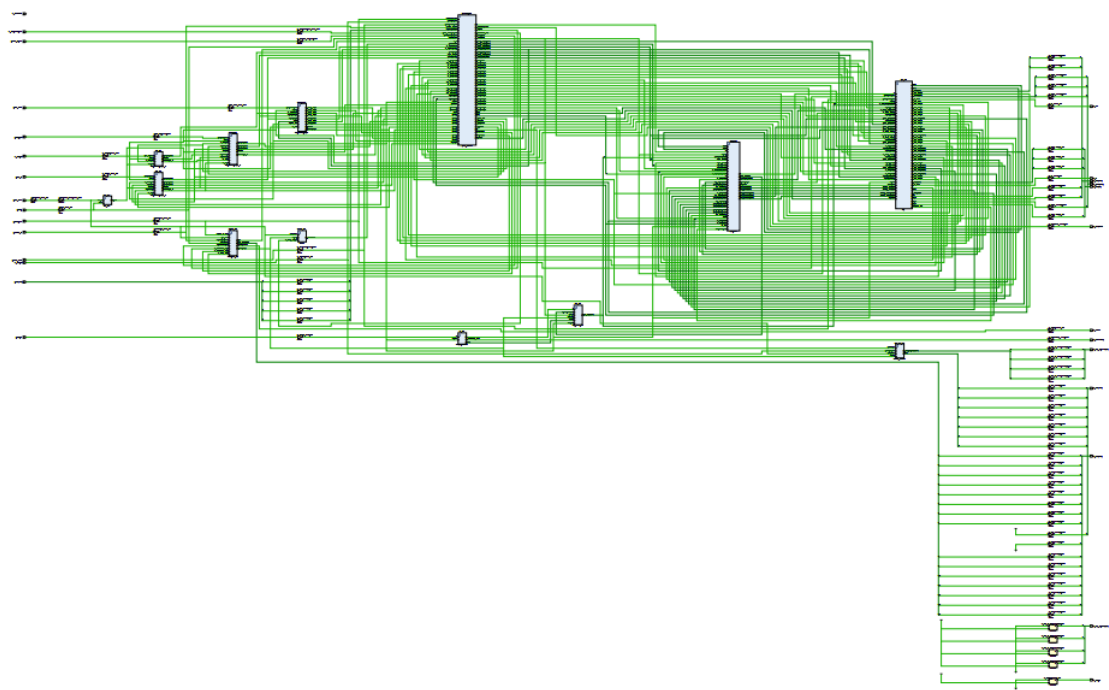
等于初始化时的图像顺序时，代表游戏通关结束。

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

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

```

9      //VGA输出信号
10     output [3:0] o_red,
11     output [3:0] o_green,
12     output [3:0] o_blue,
13     output o_hs, //行信号
14     output o_vs, //场信号
15
16     input i_btn_set, //置位信号
17     //玩家移动信号
18     input i_btn_left,
19     input i_btn_right,
20     input i_btn_up,
21     input i_btn_down,
22
23     output o_completed, //拼图完成信号
24     input i_rand_move_EN, //随机移动使能信号
25     input i_rand_set_EN, //随机置位使能信号
26     input i_hard_mode, //困难模式使能信号
27
28     // ljy
29     output [3:0] o_seg_en_time, //计时器使能信号
30     output [3:0] o_seg_en_step, //计步器使能信号
31     output [7:0] o_num, //七段数码管要显示的数字
32
33     // myc
34     input i_unsolvable, //无解信号
35     input i_music_on, //音乐开启信号
36     output wire o_music, //蜂鸣器输出信号
37     output wire [15:0] o_led //led灯输出信号
38 );
39 wire w_clk_25M; //时钟信号, 频率为25MHz
40 //串口传输相关信号
41 wire [17:0] read_addr;
42 wire w_rx_data_ready;
43 wire [7:0] w_rx_data;
44 wire w_rx_idle;
45 assign o_transmitting = ~w_rx_idle;
46
47 wire [7:0] w_pixel_data; //像素值
48 //像素坐标
49 wire [9:0] w_pixel_x;
50 wire [9:0] w_pixel_y;
51 wire w_pixel_valid; //当前像素是否有效
52 wire [11:0] pixel_data_rgb444; //屏幕当前扫描到的像素点颜色信息 (RGB 444)
53
54 //屏幕各区域显示的图片
55 wire [3:0] pos_a;
56 wire [3:0] pos_b;
57 wire [3:0] pos_c;
58 wire [3:0] pos_d;
59 wire [3:0] pos_e;
60 wire [3:0] pos_f;
61 wire [3:0] pos_g;
62 wire [3:0] pos_h;
63 wire [3:0] pos_i;
64
65 wire w_btn_left;
66 wire w_btn_right;

```

```

67     wire w_btn_up;
68     wire w_btn_down;
69     wire w_hasSolution;//有解信号
70     wire w_moving;
71
72     // 生成25MHz的clk
73     clk_25m clk_25m(.resetn(I_rst_n), .clk_in1(I_clk_100M),
74 .clk_out1(w_clk_25M));
75
76     check check_inst(
77         .IA(pos_a),
78         .IB(pos_b),
79         .IC(pos_c),
80         .ID(pos_d),
81         .I_clk(I_clk_100M),
82         .I_rst_n(I_rst_n),
83         .hard(I_hard_mode),
84         .I_black_EN(I_black_EN),
85         .O_hasSolution(w_hasSolution)
86     );
87
88     music_led_top m1_top_inst(
89         .I_clk(I_clk_100M),
90         .I_rst_n(I_rst_n),
91         .I_black_EN(I_black_EN),
92         .I_completed(O_completed),
93         .I_unsolvable(~w_hasSolution),
94         .I_music_on(I_music_on),
95         .O_music(O_music),
96         .O_led(O_led)
97     );
98
99     second_counter step_cnt_inst(
100         .clk(I_clk_100M),
101         .switch(I_black_EN),
102         .set(I_btn_set),
103         .rst(I_rst_n),
104         .num(O_num),
105         .seg_en(O_seg_en_time)
106     );
107
108     async_receiver rx_inst(
109         .clk(w_clk_25M),
110         .RxD(I_uart_rx),
111         .RxD_data_ready(w_rx_data_ready),
112         .RxD_data(w_rx_data),
113         .RxD_idle(w_rx_idle)
114     );
115
116     vga vga_inst(
117         .I_clk_25M(w_clk_25M),
118         .I_rst_n(I_rst_n),
119         .O_red(O_red),
120         .O_green(O_green),
121         .O_blue(O_blue),
122         .O_hs(O_hs),
123         .O_vs(O_vs),
124         .I_pixel_data(pixel_data_rgb444),

```

```

124         .O_pixel_x(w_pixel_x),
125         .O_pixel_y(w_pixel_y),
126         .O_pixel_valid(w_pixel_valid)
127     );
128
129     anti_shake_single ass_left(
130         .I_key(I_btn_left),
131         .I_rst_n(I_rst_n),
132         .I_clk(w_clk_25M),
133         .O_key(w_btn_left)
134     );
135
136     anti_shake_single ass_right(
137         .I_key(I_btn_right),
138         .I_rst_n(I_rst_n),
139         .I_clk(w_clk_25M),
140         .O_key(w_btn_right)
141     );
142
143     anti_shake_single ass_up(
144         .I_key(I_btn_up),
145         .I_rst_n(I_rst_n),
146         .I_clk(w_clk_25M),
147         .O_key(w_btn_up)
148     );
149
150     anti_shake_single ass_down(
151         .I_key(I_btn_down),
152         .I_rst_n(I_rst_n),
153         .I_clk(w_clk_25M),
154         .O_key(w_btn_down)
155     );
156
157
158     pixel_ctrl pctrl_inst(
159         .I_pixel_x(w_pixel_x),
160         .I_pixel_y(w_pixel_y),
161         .I_pixel_data(w_pixel_data),
162         .I_black_EN(I_black_EN),
163         .O_pixel_data(pixel_data_rgb444),
164
165         .I_pos_a(pos_a),
166         .I_pos_b(pos_b),
167         .I_pos_c(pos_c),
168         .I_pos_d(pos_d),
169         .I_pos_e(pos_e),
170         .I_pos_f(pos_f),
171         .I_pos_g(pos_g),
172         .I_pos_h(pos_h),
173         .I_pos_i(pos_i),
174
175         .O_read_addr(read_addr),
176         .I_rst_n(I_rst_n),
177         .I_clk_25M(w_clk_25M),
178         .I_hard_mode(I_hard_mode)
179     );
180
181     game_control gamectrl_inst(

```

```

182         .I_num(I_num),
183         .I_rst_n(I_rst_n),
184         .O_pos_a(pos_a),
185         .O_pos_b(pos_b),
186         .O_pos_c(pos_c),
187         .O_pos_d(pos_d),
188         .O_pos_e(pos_e),
189         .O_pos_f(pos_f),
190         .O_pos_g(pos_g),
191         .O_pos_h(pos_h),
192         .O_pos_i(pos_i),
193
194         .O_completed(O_completed),
195
196         .I_btn_set(I_btn_set),
197         .I_btn_left(W_btn_left),
198         .I_btn_right(W_btn_right),
199         .I_btn_up(W_btn_up),
200         .I_btn_down(W_btn_down),
201         .I_black_EN(I_black_EN),
202         .I_clk(W_clk_25M),
203         .I_rand_move_EN(I_rand_move_EN),
204         .I_rand_set_EN(I_rand_set_EN),
205         .I_hard_mode(I_hard_mode),
206         .O_moving(W_moving)
207     );
208
209     img_mem_ctrl imc_inst(
210         .I_clk_25M(W_clk_25M),
211         .I_rst_n(I_rst_n),
212         .I_write_en(W_rx_data_ready),
213         .I_write_data(W_rx_data),
214         .O_pixel_data(W_pixel_data),
215         .I_read_addr(read_addr)
216     );
217 endmodule

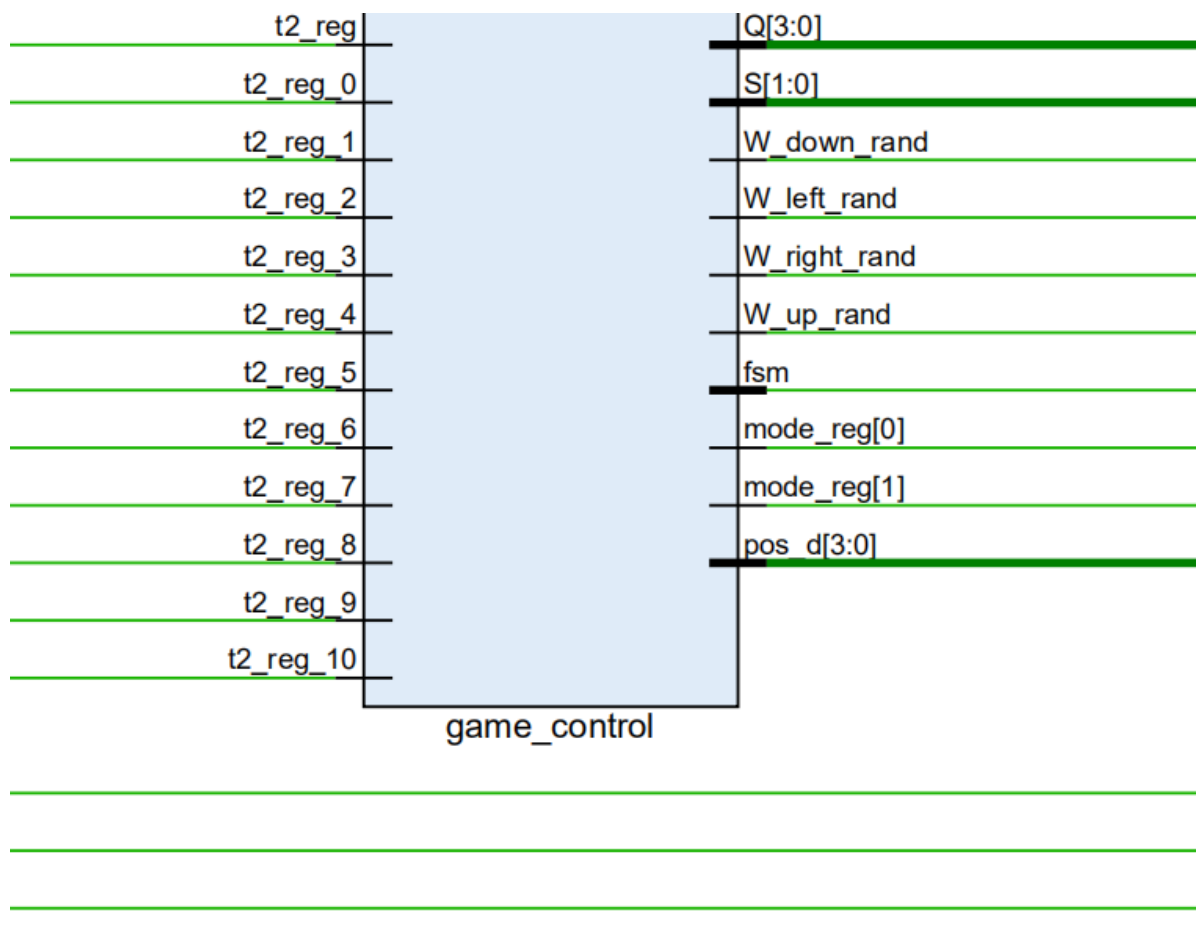
```

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		DI
I_music_on_IBUF		O_completed_OBUF
I_num_IBUF[4:0]		O_dir_reg
I_rand_move_EN_IBUF		O_hasSolution_reg
I_rand_set_EN_IBUF		O_led_on_reg
I_rst_n_IBUF		O_mv_up_reg
O_dir		O_pos a reg[3] 0[3:0]
O_hasSolution		O_pos a reg[3] 1[3:0]
O_mv_down_reg		O_pos b reg[3] 0[3:0]
O_mv_down_reg_0		O_pos f reg[3] 0[3:0]
O_mv_left_reg		O_pos f reg[3] 1[3:0]
O_mv_right_reg		O_pos h reg[3] 0[3:0]
O_mv_up_reg_0		O_pos i reg[3] 0[3:0]
R_h_cnt_reg[4]		O_read_addr
R_h_cnt_reg[4]_0		O_read_addr_0
R_h_cnt_reg[6]		O_read_addr_1
R_h_cnt_reg[8]		O_read_addr_2
R_v_cnt_reg[4]		O_read_addr_3
R_v_cnt_reg[6]		O_read_addr_4
R_v_cnt_reg[7]		O_read_addr_5
R_v_cnt_reg[8]		O_read_addr_6
R_v_cnt_reg[8]_0		O_read_addr_7
R_v_cnt_reg[9]		O_read_addr_8
R_v_cnt_reg[9]_0		O_read_addr_9
R_v_cnt_reg[9]_1		O_read_addr_10
R_v_cnt_reg[9]_2		O_read_addr_11
W_pixel_x[2:0]		O_read_addr_12
W_pixel_y[1:0]		O_read_addr_13
clk_out1		O_read_addr_14
t_locked2		O_read_addr_15
t_locked2_0		O_read_addr_16
t_locked2_1		O_read_addr_17



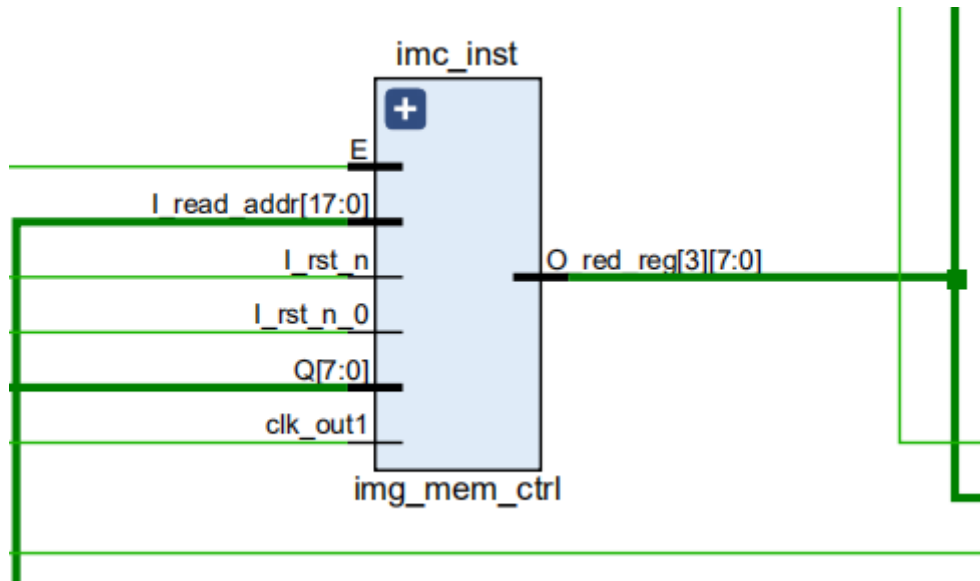
```

1  module game_control(
2      input [4:0] I_num,
3      input I_clk,
4      input I_rst_n,
5      input I_btn_set,
6      input I_btn_left,
7      input I_btn_right,
8      input I_btn_up,
9      input I_btn_down,
10     output reg [3:0] O_pos_a,
11     output reg [3:0] O_pos_b,
12     output reg [3:0] O_pos_c,
13     output reg [3:0] O_pos_d,
14     output reg [3:0] O_pos_e,
15     output reg [3:0] O_pos_f,
16     output reg [3:0] O_pos_g,
17     output reg [3:0] O_pos_h,
18     output reg [3:0] O_pos_i,
19     input I_black_EN,
20     input I_rand_move_EN,
21     input I_rand_set_EN,
22     input I_hard_mode,
23     output O_completed
24 );

```

2.2.1.3 img_mem_ctrl

该模块用于管理内存的写入和读取

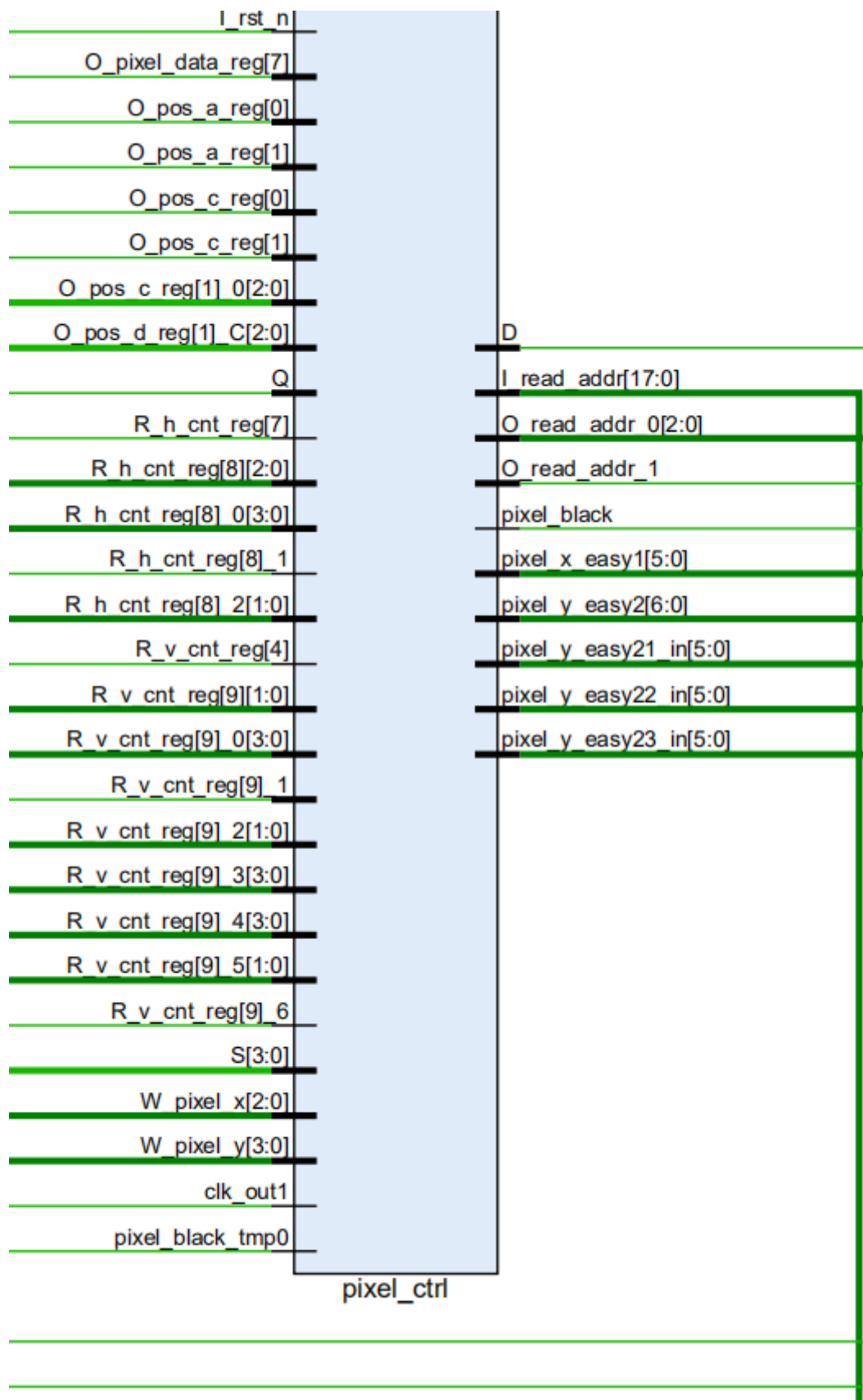


```
1 module img_mem_ctrl(  
2     input I_clk_25M, //时钟信号, 频率为25MHz  
3     input I_rst_n, //复位信号  
4     input I_write_en, //写入数据的使能信号  
5     input [7:0] I_write_data, //要写入的数据  
6     input [17:0] I_read_addr, //读取地址  
7     output reg [7:0] O_pixel_data //读取的数据  
8 );
```

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



```

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

```

```

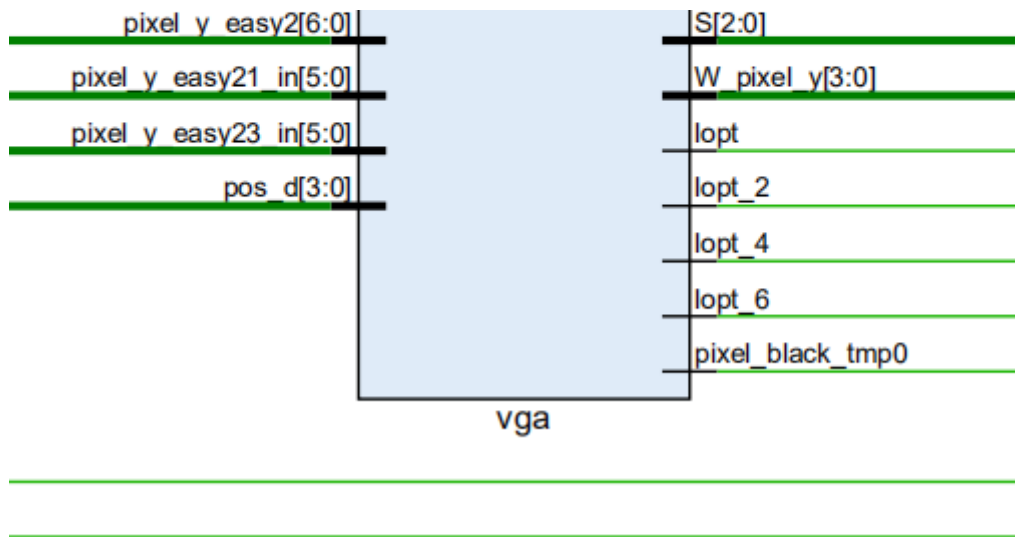
10      //相应屏幕区域显示的图片区域
11      input  [3:0]  I_pos_a,
12      input  [3:0]  I_pos_b,
13      input  [3:0]  I_pos_c,
14      input  [3:0]  I_pos_d,
15      input  [3:0]  I_pos_e,
16      input  [3:0]  I_pos_f,
17      input  [3:0]  I_pos_g,
18      input  [3:0]  I_pos_h,
19      input  [3:0]  I_pos_i,
20      //内存读取地址
21      output [17:0] o_read_addr,
22      //困难模式
23      input  I_hard_mode
24  );

```

2.2.1.5 VGA

用于输出VGA显示所需要的相关信号

	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
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]	



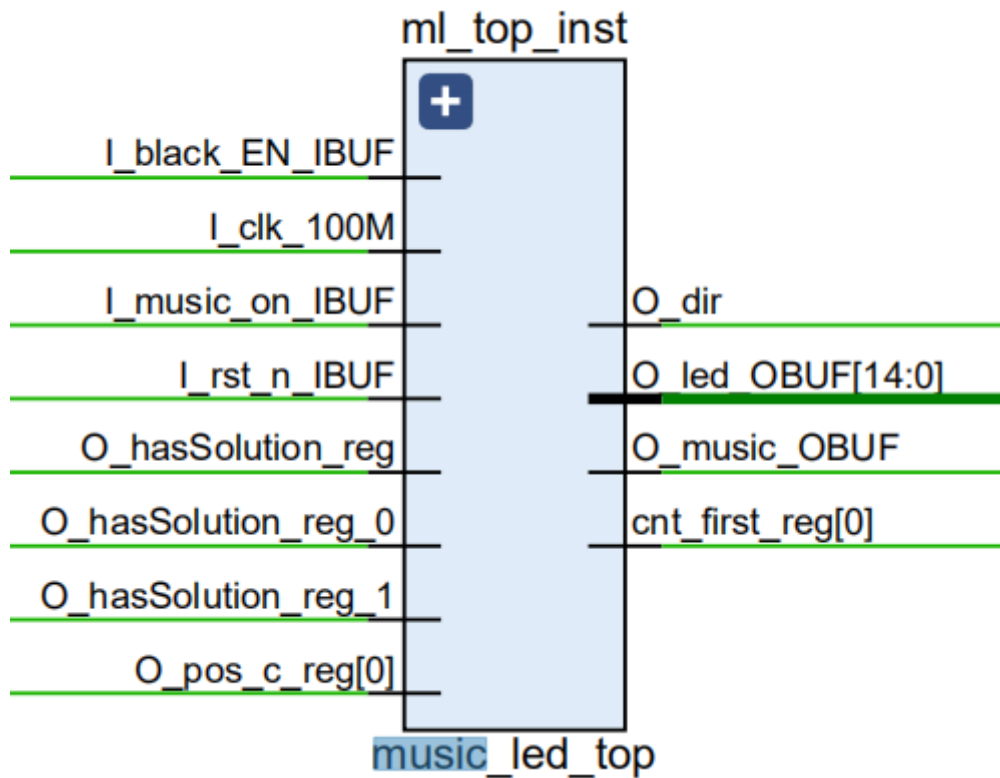
```

1 module vga(
2     input I_clk_25M, // 时钟信号，频率为25MHz
3     input I_rst_n, // 复位信号
4
5     // 像素点颜色信息 (RGB 444)
6     output reg [3:0] O_red,
7     output reg [3:0] O_green,
8     output reg [3:0] O_blue,
9
10    output O_hs, // 行信号
11    output O_vs, // 场信号
12
13
14    input [11:0] I_pixel_data, // 屏幕当前扫描到的像素点颜色信息 (RGB 444)
15    output [9:0] O_pixel_x, // 屏幕当前扫描到的像素点的横坐标
16    output [9:0] O_pixel_y, // 屏幕当前扫描到的像素点的纵坐标
17    output O_pixel_valid // 当前扫描位置是否在显示区域内
18
19 );

```

2.2.1.6 music_led_top

根据游戏状态控制led灯和蜂鸣器状态



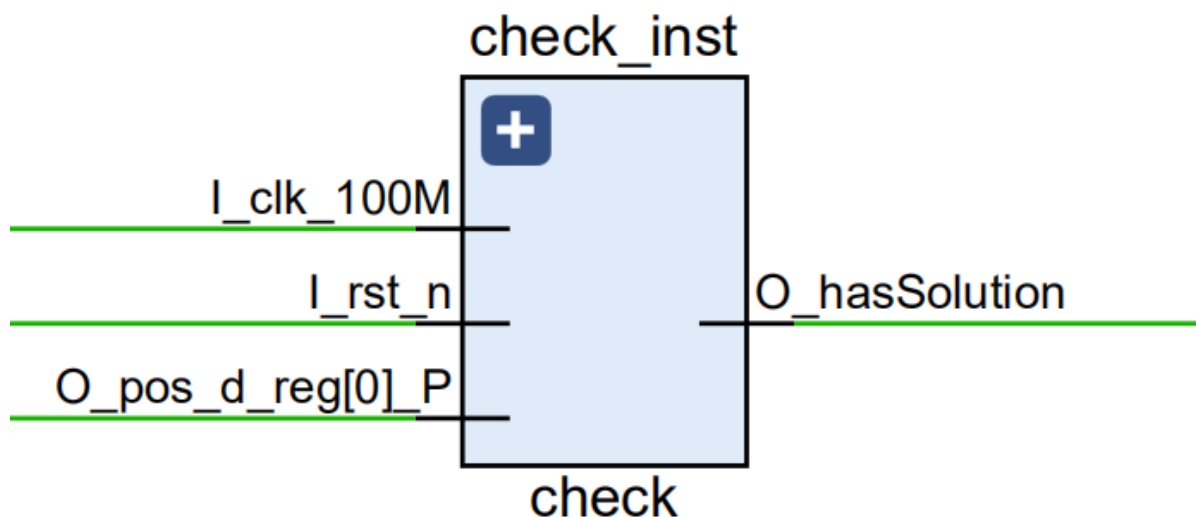
```

1 module music_led_top(
2     input I_clk, // 时钟信号
3     input I_rst_n, // 复位信号
4     input I_black_EN, // 拉黑信号
5     input I_completed, // 拼图完成信号
6     input I_unsolvable, // 拼图无解信号
7     input I_music_on, // 音乐开关
8     output wire O_music, // 蜂鸣器输出
9     output wire [15:0] O_led // led灯输出
10 );

```

2.2.1.7 check

根据游戏状态判断是否有解



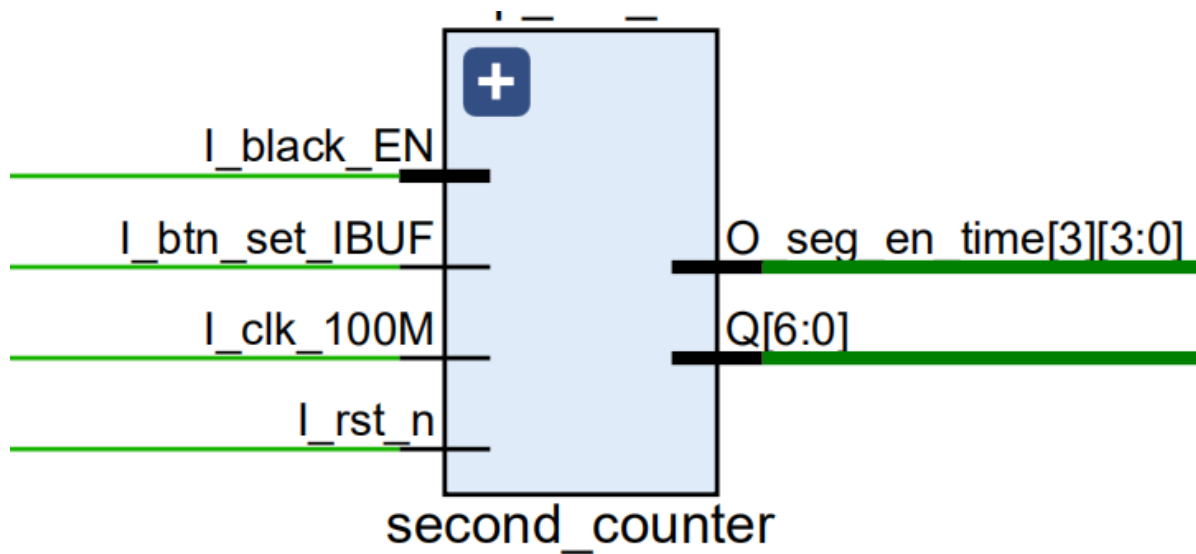

```

1  module check (
2      //2*2游戏信息
3      input [1:0] IA,
4      input [1:0] IB,
5      input [1:0] IC,
6      input [1:0] ID,
7      input I_clk, // 时钟信号
8      input I_black_EN, //拉黑信号
9      input I_rst_n, //复位信号
10     output reg o_hasSolution //有解信号
11 );

```

2.2.1.8 counter_ss

计算显示的步数和时间



```

1  module counter_ss(
2      input clk, //时钟信号
3      input wire switch, //暂停信号
4      input set, //置位信号
5      input rst, //复位信号
6      output reg [7:0] num, //显示的数字
7      output reg [3:0] seg_en_s //步数显示的位置
8  );

```

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

```
1 module anti_shake_single(  
2     input I_key,I_clk,I_rst_n,  
3     output O_key  
4 );  
5     wire key_changed1;  
6     reg [20:0] count;  
7     // reg [2:0] count; // for simulation  
8     reg t1, t_locked1, t2, t_locked2;  
9  
10    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  
11        if(~I_rst_n) t1 <= 0;  
12        else t1 <= I_key;  
13  
14    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  
15        if(~I_rst_n) t_locked1 <= 0;  
16        else t_locked1 <= t1;  
17  
18    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  
19  
20    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  
21        if(~I_rst_n) count <= 0;  
22        else if(key_changed1) count <= 0;  
23        else count <= count + 1'b1;  
24  
25    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  
26        if(~I_rst_n) t2 <= 0;  
27        else if(count == 2000000)//20ms左右  
28    // else if(count == 2) // for simulation  
29        t2 <= I_key;  
30  
31    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  
32        if(~I_rst_n) t_locked2 <= 0;  
33        else t_locked2 <= t2;  
34  
35    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  
36  
37 endmodule  
38
```

仿真波形图：

```
1 module anti_shake_single_sim();  
2     reg I_key,I_clk,I_rst_n;  
3     wire O_key;  
4     anti_shake_single u(I_key,I_clk,I_rst_n,O_key);  
5     initial begin
```

```

6   I_clk = 1'b0;
7   I_rst_n = 1'b0;
8   I_key = 1'b0;
9   forever #5 I_clk =~I_clk;
10  end
11  initial fork
12  #3 I_rst_n = 1'b1;
13  #5 I_key = 1'b1;
14  #7 I_key = 1'b0;
15  #16 I_key = 1'b1;
16  #200 I_key = 1'b0;
17  #310 I_key = 1'b1;
18  #313 I_key = 1'b0;
19  #326 I_key = 1'b1;
20  #339 I_key = 1'b0;
21  #430 I_key = 1'b1;
22  #480 I_key = 1'b0;
23  #700 I_key = 1'b1;
24  #750 I_key = 1'b0;
25  join
26  endmodule
27

```

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

```

1  //in led_music_control
2  `timescale 1ns / 1ps
3  module check (
4      input [1:0] IA,
5      input [1:0] IB,
6      input [1:0] IC,
7      input [1:0] ID,
8      //是否开启3*3
9      input hard,
10     input I_clk,
11     input I_black_EN,
12     input I_rst_n,
13     output reg o_hasSolution
14 );
15
16 always @(posedge I_clk, negedge I_rst_n) begin
17     if(~I_rst_n) begin
18         o_hasSolution <= 1'b0;
19     end
20     else if(hard) begin
21         o_hasSolution <= 1'b1;
22     end
23     else if(I_black_EN) begin
24         case ({IA,IB,IC,ID})
25             8'b00_01_01_11 : o_hasSolution <= 1'b1;//1234
26             8'b01_01_11_01 : o_hasSolution <= 1'b1;//3142
27             8'b11_00_10_01 : o_hasSolution <= 1'b1;//4132

```

```

28         8'b00_01_11_01 : O_hasSolution <= 1'b1;//1243
29         8'b00_10_11_01 : O_hasSolution <= 1'b1;//1342
30         8'b11_00_01_10 : O_hasSolution <= 1'b1;//4123
31         8'b01_11_10_00 : O_hasSolution <= 1'b1;//2431
32         8'b10_11_01_00 : O_hasSolution <= 1'b1;//3421
33         8'b11_10_01_00 : O_hasSolution <= 1'b1;//4321
34         8'b01_11_00_10 : O_hasSolution <= 1'b1;//2413
35         8'b01_10_00_11 : O_hasSolution <= 1'b1;//2314
36         8'b10_01_00_11 : O_hasSolution <= 1'b1;//3214
37
38         default: O_hasSolution <= 1'b0;
39     endcase
40
41     end
42     else O_hasSolution <= 1'b0;
43 end
44
45 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

```

1  //in music.v
2  `timescale 1ns / 1ps
3
4  module Music(input clk, input wire[4:0] mode, output reg[0:0] music = 0,
5  output wire[6:0] o_led);
6
6  parameter do_low = 191110;
7  parameter re_low = 170259;
8  parameter me_low = 151685;
9  parameter fa_low = 143172;
10 parameter so_low = 127554;
11 parameter la_low = 113636;
12 parameter si_low = 101239;
13
14 parameter do = 93941;
15 parameter re = 85136;
16 parameter me = 75838;
17 parameter fa = 71582;
18 parameter so = 63776;
19 parameter la = 56818;
20 parameter si = 50618;
21
22 parameter do_high = 47778;
23 parameter re_high = 42567;
24 parameter me_high = 37921;
25 parameter fa_high = 36498;
26 parameter so_high = 31888;
27 parameter la_high = 28409;
28 parameter si_high = 25309;
29
30 parameter beat = 40 * 500000;
31 parameter gap = 10 * 500000;
32 parameter index_period = beat + gap;
33

```

```

34 parameter silence = beat<<9;
35
36 /*
37 0 - silence
38 1 - 7 low
39 8 - 14 meidum
40 15 - 21 high
41 */
42 parameter Star =
43 210'b0000001000010010100101010010100101101011000000110001101011010110001100
44 010000100000000010010101001010010110101101100011000000001001010100101001011
45 010110110001100000000100001001010010101001010010110101100000011000110101101
46 01100011000100001000;
47 parameter HappyBirthday =
48 130'b0111110000011111000110010000000000001101011100111110001100110110001111
49 011111000001100011010110001110011100111101100011010010100101;
50 parameter MerryChristmas =
51 335'b0111101111011110111010000011010110001100100110111110000100011000000000
52 011000110001101011100111101110000000111001111011110111101100000000111101111
53 011101000001101011000110000000011010111110000100011001010001100010000001100
54 01100000000110001110011111000010001100001000000000011010000000000110101101
55 0111001111100000111101111000000110001100;
56
57 parameter ST_length = 42;
58 parameter HB_length = 26;
59 parameter MC_length = 67;
60
61 reg[29:0] freq = beat;
62
63 reg[2000:0] melody = 0;
64 integer melody_length = 0;
65
66 integer frequency_count = 0; // count1 control frequency
67 integer index_count = 0; // count2 control beat;
68
69 integer index = 0; // index control the location music playing
70
71 reg [0:0] isSilence = 0;
72 reg [0:0] isEnd = 0;
73 reg [0:0] isPeriodic = 0;
74
75 reg[4:0] last_mode = 0;
76
77 //歌曲选择
78 always @(posedge clk) begin
79
80     if(mode != last_mode) begin
81         last_mode = mode;
82         isEnd = 0;
83         index = 0;
84         index_count = 0;
85
86         if(mode >= 17) isPeriodic = 1;
87         else begin isPeriodic = 0; melody_length = 1; end
88
89         if(mode == 17) melody_length = MC_length;
90         if(mode == 18) melody_length = HB_length;
91         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;

```

```

139 5'd21 : freq = si_high;
140 default : freq = silence;
141 endcase
142 end
143
144 //利用当前频率判断律动灯状态判断
145 always @(freq) begin
146     case(freq)
147         silence : o_led = 7'b0000_000;
148         do_low  : o_led = 7'b0000_001;
149         re_low  : o_led = 7'b0000_010;
150         me_low  : o_led = 7'b0000_100;
151         fa_low  : o_led = 7'b0001_000;
152         so_low  : o_led = 7'b0010_000;
153         la_low  : o_led = 7'b0100_000;
154         si_low  : o_led = 7'b1000_000;
155
156         do      : o_led = 7'b0000_001;
157         re      : o_led = 7'b0000_010;
158         me      : o_led = 7'b0000_100;
159         fa      : o_led = 7'b0001_000;
160         so      : o_led = 7'b0010_000;
161         la      : o_led = 7'b0100_000;
162         si      : o_led = 7'b1000_000;
163
164         do_high : o_led = 7'b0000_001;
165         re_high : o_led = 7'b0000_010;
166         me_high : o_led = 7'b0000_100;
167         fa_high : o_led = 7'b0001_000;
168         so_high : o_led = 7'b0010_000;
169         la_high : o_led = 7'b0100_000;
170         si_high : o_led = 7'b1000_000;
171
172         default : o_led = 7'b0000_000;
173     endcase
174 end
175
176
177
178 endmodule
179
180

```

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

```

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

```

```

5         input [3:0] IC,
6         input [3:0] ID,
7         //是否开启3*3
8         input hard,
9         input I_clk,
10        input I_black_EN,
11        input I_rst_n,
12        output reg O_hasSolution
13    );
14
15    always @(posedge I_clk, negedge I_rst_n) begin
16        if(~I_rst_n) begin
17            O_hasSolution <= 1'b0;
18        end
19        else if(hard) begin
20            O_hasSolution <= 1'b1;
21        end
22        else if(I_black_EN) begin
23            case ({IA[1:0], IB[1:0], IC[1:0], ID[1:0]})
24                16'b00_01_10_11 : O_hasSolution <= 1'b1; //1234
25                16'b10_00_11_01 : O_hasSolution <= 1'b1; //3142
26                16'b11_00_10_01 : O_hasSolution <= 1'b1; //4132
27                16'b00_01_11_10 : O_hasSolution <= 1'b1; //1243
28                16'b00_10_11_01 : O_hasSolution <= 1'b1; //1342
29                16'b11_00_01_10 : O_hasSolution <= 1'b1; //4123
30                16'b01_11_10_00 : O_hasSolution <= 1'b1; //2431
31                16'b10_11_01_00 : O_hasSolution <= 1'b1; //3421
32                16'b11_10_01_00 : O_hasSolution <= 1'b1; //4321
33                16'b01_11_00_10 : O_hasSolution <= 1'b1; //2413
34                16'b01_10_00_11 : O_hasSolution <= 1'b1; //2314
35                16'b10_01_00_11 : O_hasSolution <= 1'b1; //3214
36
37                default: O_hasSolution <= 1'b0;
38            endcase
39        end
40        else O_hasSolution <= 1'b0;
41    end
42
43
44    endmodule

```

2.3.1.4 计时器

计时器：

1. 输入输出端口

```

1 module second_counter(
2     input clk, //时钟信号
3     input wire switch, //暂停信号
4     input set, //置位信号
5     input rst, //复位信号
6     output reg [7:0] num, //显示的数字
7     output reg [3:0] seg_en //显示的位置
8 );

```

2. 参数


```

1  parameter      SEG_NUM0 = 8'b1100_0000,
2                  SEG_NUM1 = 8'b1111_1001,
3                  SEG_NUM2 = 8'b1010_0100,
4                  SEG_NUM3 = 8'b1011_0000,
5                  SEG_NUM4 = 8'b1001_1001,
6                  SEG_NUM5 = 8'b1001_0010,
7                  SEG_NUM6 = 8'b1000_0010,
8                  SEG_NUM7 = 8'b1111_1000,
9                  SEG_NUM8 = 8'b1000_0000,
10                 SEG_NUM9 = 8'b1001_0000;
11  parameter      DUAN_3 = 4'b0111,
12                 DUAN_2 = 4'b1011,
13                 DUAN_1 = 4'b1101,
14                 DUAN_0 = 4'b1110;

```

3.分频 (产生周期为1ms的时钟线)

```

1  always @(posedge clk) begin
2      if(fenpin == 100000)
3          fenpin <= 0;
4      else
5          fenpin <= fenpin + 1'b1;
6  end
7
8  always @ (posedge clk) begin
9      if(fenpin < 50000)
10         clk_1ms <= 0;
11     else
12         clk_1ms <= 1;
13 end

```

4.状态机:

```

1  always @ (posedge clk or negedge rst) begin
2      if (~rst) begin
3          state <= 1;
4          cen <= 0;
5      end
6      else if (set) begin
7          state <= 1;
8          cen <= 0;
9      end
10     else begin
11         case(state)
12             0: begin
13                 cen <= 1;
14                 if(!switch)
15                     state <= 1;
16                 else
17                     state <= 0;
18             end
19             1: begin
20                 cen <= 0;
21                 if(switch)
22                     state <= 0;
23                 else

```

```

24     state <= 1;
25     end
26 endcase
27 end
28 end

```

5.时间计算

```

1  always @ (posedge clk_1ms or negedge rst) begin
2  if(~rst)begin
3  ms1 <= 0;
4  ms100 <= 0;
5  ms10 <= 0;
6  s1 <= 0;
7  s2 <= 0;
8  s3 <= 0;
9  s4 <= 0;
10 end
11 else if(set)begin
12 ms1 <= 0;
13 ms100 <= 0;
14 ms10 <= 0;
15 s1 <= 0;
16 s2 <= 0;
17 s3 <= 0;
18 s4 <= 0;
19 end
20 else begin
21 if(cen) begin
22     case(ms1)
23     4'b1001: ms1 <= 4'b0000;
24     default: ms1 <= ms1 + 1'b1;
25     endcase
26     case(ms10)
27     4'b1001: begin
28         if(ms1 == 4'b1001)
29             ms10 <= 4'b0000;
30         end
31     default: begin
32         if(ms1 == 4'b1001)
33             ms10 <= ms10 + 1'b1;
34         end
35     endcase
36     case(ms100)
37     4'b1001: begin
38         if(ms1 == 4'b1001 && ms10 == 4'b1001)
39             ms100 <= 4'b0000;
40         end
41     default: begin
42         if(ms1 == 4'b1001 && ms10 == 4'b1001)
43             ms100 <= ms100 + 1'b1;
44         end
45     endcase
46     case(s1)
47     4'b1001: begin
48         if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 == 4'b1001)
49             s1 <= 4'b0000;

```

```

50         end
51     default: begin
52         if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 == 4'b1001)
53             s1 <= s1 + 1'b1;
54         end
55     endcase
56     case(s2)
57         4'b1001: begin
58             if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 == 4'b1001
&& s1==4'b1001)
59                 s2 <= 4'b0000;
60             end
61         default: begin
62             if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 == 4'b1001
&& s1==4'b1001)
63                 s2 <= s2 + 1'b1;
64             end
65         endcase
66     case(s3)
67         4'b1001: begin
68             if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 ==
4'b1001 && s1==4'b1001 && s2==4'b1001)
69                 s3 <= 4'b0000;
70             end
71         default: begin
72             if(ms1 == 4'b1001 && ms10 == 4'b1001 && ms100 ==
4'b1001 && s1==4'b1001 && s2==4'b1001)
73                 s3 <= s3 + 1'b1;
74             end
75         endcase
76     case(s4)
77         4'b1001: begin
78             if(ms1 == 4'b1001 && ms10 == 4'b1001 &&
ms100 == 4'b1001 && s1==4'b1001 && s2==4'b1001 && s3==4'b1001)
79                 s4 <= 4'b0000;
80             end
81         default: begin
82             if(ms1 == 4'b1001 && ms10 == 4'b1001 &&
ms100 == 4'b1001 && s1==4'b1001 && s2==4'b1001 && s3==4'b1001)
83                 s4 <= s4 + 1'b1;
84             end
85         endcase
86     end
87 end
88 end

```

6.七段数码管显示

```

1  always @ (posedge clk) begin
2      case(count[19:18])
3          2'b00: begin
4              seg_en_time <= DUAN_3;
5              case(s4)
6                  4'b0000: num <= SEG_NUM0;
7                  4'b0001: num <= SEG_NUM1;
8                  4'b0010: num <= SEG_NUM2;
9                  4'b0011: num <= SEG_NUM3;

```

```

10         4'b0100: num <= SEG_NUM4;
11         4'b0101: num <= SEG_NUM5;
12         4'b0110: num <= SEG_NUM6;
13         4'b0111: num <= SEG_NUM7;
14         4'b1000: num <= SEG_NUM8;
15         4'b1001: num <= SEG_NUM9;
16     endcase
17 end
18
19 2'b01: begin
20     seg_en_time <= DUAN_2;
21     case(s3)
22         4'b0000: num <= SEG_NUM0;
23         4'b0001: num <= SEG_NUM1;
24         4'b0010: num <= SEG_NUM2;
25         4'b0011: num <= SEG_NUM3;
26         4'b0100: num <= SEG_NUM4;
27         4'b0101: num <= SEG_NUM5;
28         4'b0110: num <= SEG_NUM6;
29         4'b0111: num <= SEG_NUM7;
30         4'b1000: num <= SEG_NUM8;
31         4'b1001: num <= SEG_NUM9;
32     endcase
33 end
34 2'b10: begin
35     seg_en_time <= DUAN_1;
36     case(s2)
37         4'b0000: num <= SEG_NUM0;
38         4'b0001: num <= SEG_NUM1;
39         4'b0010: num <= SEG_NUM2;
40         4'b0011: num <= SEG_NUM3;
41         4'b0100: num <= SEG_NUM4;
42         4'b0101: num <= SEG_NUM5;
43         4'b0110: num <= SEG_NUM6;
44         4'b0111: num <= SEG_NUM7;
45         4'b1000: num <= SEG_NUM8;
46         4'b1001: num <= SEG_NUM9;
47     endcase
48 end
49 2'b11: begin
50     seg_en_time <= DUAN_0;
51     case(s1)
52         4'b0000: num <= SEG_NUM0;
53         4'b0001: num <= SEG_NUM1;
54         4'b0010: num <= SEG_NUM2;
55         4'b0011: num <= SEG_NUM3;
56         4'b0100: num <= SEG_NUM4;
57         4'b0101: num <= SEG_NUM5;
58         4'b0110: num <= SEG_NUM6;
59         4'b0111: num <= SEG_NUM7;
60         4'b1000: num <= SEG_NUM8;
61         4'b1001: num <= SEG_NUM9;
62     endcase
63 end
64 endcase
65
66 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。

```
1  module game_control(  
2      input  [4:0] I_num,  
3      input  I_clk,  
4      input  I_rst_n,  
5      input  I_btn_set,  
6      input  I_btn_left,  
7      input  I_btn_right,  
8      input  I_btn_up,  
9      input  I_btn_down,  
10     output reg [3:0] O_pos_a,  
11     output reg [3:0] O_pos_b,  
12     output reg [3:0] O_pos_c,  
13     output reg [3:0] O_pos_d,  
14     output reg [3:0] O_pos_e,  
15     output reg [3:0] O_pos_f,  
16     output reg [3:0] O_pos_g,  
17     output reg [3:0] O_pos_h,  
18     output reg [3:0] O_pos_i,  
19     input  I_black_EN,  
20     input  I_rand_move_EN,  
21     input  I_rand_set_EN,  
22     input  I_hard_mode,  
23     output O_completed,  
24     output reg O_moving  
25 );  
26  
27 wire num_valid;  
28 wire [1:0] pos_a_native;  
29 wire [1:0] pos_b_native;  
30 wire [1:0] pos_c_native;  
31 wire [1:0] pos_d_native;  
32 wire [1:0] pos_a_rand;  
33 wire [1:0] pos_b_rand;  
34 wire [1:0] pos_c_rand;  
35 wire [1:0] pos_d_rand;  
36 wire game_start = I_black_EN;  
37  
38 assign O_completed = I_hard_mode
```

```

39     ? ((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))
40     : ((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));
41
42 wire w_left, w_right, w_up, w_down;
43 wire w_left_rand, w_right_rand, w_up_rand, w_down_rand;
44
45 random_move rand_mv_inst(
46     .I_clk(I_clk),
47     .I_rst_n(I_rst_n),
48     .I_en(I_rand_move_EN),
49     .O_mv_left(w_left_rand),
50     .O_mv_right(w_right_rand),
51     .O_mv_up(w_up_rand),
52     .O_mv_down(w_down_rand)
53 );
54
55 rand_pos rand_pos_inst(
56     .I_clk(I_clk),
57     .O_pos_a(pos_a_rand),
58     .O_pos_b(pos_b_rand),
59     .O_pos_c(pos_c_rand),
60     .O_pos_d(pos_d_rand)
61 );
62
63 assign w_left = I_btn_left | w_left_rand;
64 assign w_right = I_btn_right | w_right_rand;
65 assign w_up = I_btn_up | w_up_rand;
66 assign w_down = I_btn_down | w_down_rand;
67
68 permutation perm_inst(
69     .I_num(I_num),
70     .O_pos_a(pos_a_native),
71     .O_pos_b(pos_b_native),
72     .O_pos_c(pos_c_native),
73     .O_pos_d(pos_d_native),
74     .O_num_valid(num_valid)
75 );
76
77 always @(posedge I_clk, negedge I_rst_n) begin
78     if(~I_rst_n) begin
79         O_moving <= 0;
80         if(I_hard_mode) begin
81             O_pos_a <= 4'b0000;
82             O_pos_b <= 4'b0001;
83             O_pos_c <= 4'b0010;
84
85             O_pos_d <= 4'b0100;
86             O_pos_e <= 4'b0101;
87             O_pos_f <= 4'b0110;
88
89             O_pos_g <= 4'b1000;
90             O_pos_h <= 4'b1001;
91             O_pos_i <= 4'b1010;
92         end
93         else begin

```

```

94         O_pos_a <= 4'b0000;
95         O_pos_b <= 4'b0001;
96         O_pos_c <= 4'b0010;
97         O_pos_d <= 4'b0011;
98     end
99 end
100 else if(I_btn_set) begin
101     O_moving <= 0;
102     if(I_hard_mode) begin
103         O_pos_a <= 4'b0000;
104         O_pos_b <= 4'b0001;
105         O_pos_c <= 4'b0010;
106
107         O_pos_d <= 4'b0100;
108         O_pos_e <= 4'b0101;
109         O_pos_f <= 4'b0110;
110
111         O_pos_g <= 4'b1000;
112         O_pos_h <= 4'b1001;
113         O_pos_i <= 4'b1010;
114     end
115     else if(I_rand_set_EN) begin
116         O_pos_a <= pos_a_rand;
117         O_pos_b <= pos_b_rand;
118         O_pos_c <= pos_c_rand;
119         O_pos_d <= pos_d_rand;
120     end
121     else if(num_valid) begin
122         O_pos_a <= pos_a_native;
123         O_pos_b <= pos_b_native;
124         O_pos_c <= pos_c_native;
125         O_pos_d <= pos_d_native;
126     end
127 end
128 else if(game_start) begin
129     if(I_hard_mode) begin
130         casex ({w_left, w_right, w_up, w_down})
131         4'b1xxx: begin
132             if(O_pos_i == 4'b1000) begin
133                 O_pos_h <= O_pos_i;
134                 O_pos_i <= O_pos_h;
135                 O_moving <= 1;
136             end
137             else if(O_pos_h == 4'b1000) begin
138                 O_pos_g <= O_pos_h;
139                 O_pos_h <= O_pos_g;
140                 O_moving <= 1;
141             end
142             else if(O_pos_f == 4'b1000) begin
143                 O_pos_f <= O_pos_e;
144                 O_pos_e <= O_pos_f;
145                 O_moving <= 1;
146             end
147             else if(O_pos_e == 4'b1000) begin
148                 O_pos_e <= O_pos_d;
149                 O_pos_d <= O_pos_e;
150                 O_moving <= 1;
151             end

```

```

152         else if(O_pos_c == 4'b1000) begin
153             O_pos_c <= O_pos_b;
154             O_pos_b <= O_pos_c;
155             O_moving <= 1;
156         end
157         else if(O_pos_b == 4'b1000) begin
158             O_pos_b <= O_pos_a;
159             O_pos_a <= O_pos_b;
160             O_moving <= 1;
161         end
162         else O_moving <= 0;
163     end
164     4'b01xx: begin
165         if(O_pos_a == 4'b1000) begin
166             O_pos_b <= O_pos_a;
167             O_pos_a <= O_pos_b;
168             O_moving <= 1;
169         end
170         else if(O_pos_b == 4'b1000) begin
171             O_pos_c <= O_pos_b;
172             O_pos_b <= O_pos_c;
173             O_moving <= 1;
174         end
175         else if(O_pos_d == 4'b1000) begin
176             O_pos_d <= O_pos_e;
177             O_pos_e <= O_pos_d;
178             O_moving <= 1;
179         end
180         else if(O_pos_e == 4'b1000) begin
181             O_pos_e <= O_pos_f;
182             O_pos_f <= O_pos_e;
183             O_moving <= 1;
184         end
185         else if(O_pos_g == 4'b1000) begin
186             O_pos_g <= O_pos_h;
187             O_pos_h <= O_pos_g;
188             O_moving <= 1;
189         end
190         else if(O_pos_h == 4'b1000) begin
191             O_pos_i <= O_pos_h;
192             O_pos_h <= O_pos_i;
193             O_moving <= 1;
194         end
195         else O_moving <= 0;
196     end
197     4'b001x: begin
198         if(O_pos_d == 4'b1000) begin
199             O_pos_a <= O_pos_d;
200             O_pos_d <= O_pos_a;
201             O_moving <= 1;
202         end
203         else if(O_pos_e == 4'b1000) begin
204             O_pos_e <= O_pos_b;
205             O_pos_b <= O_pos_e;
206             O_moving <= 1;
207         end
208         else if(O_pos_f == 4'b1000) begin
209             O_pos_f <= O_pos_c;

```



```

210         O_pos_c <= O_pos_f;
211         O_moving <= 1;
212     end
213     else if(O_pos_g == 4'b1000) begin
214         O_pos_g <= O_pos_d;
215         O_pos_d <= O_pos_g;
216         O_moving <= 1;
217     end
218     else if(O_pos_h == 4'b1000) begin
219         O_pos_h <= O_pos_e;
220         O_pos_e <= O_pos_h;
221         O_moving <= 1;
222     end
223     else if(O_pos_i == 4'b1000) begin
224         O_pos_i <= O_pos_f;
225         O_pos_f <= O_pos_i;
226         O_moving <= 1;
227     end
228     else O_moving <= 0;
229 end
230 4'b0001: begin
231     if(O_pos_a == 4'b1000) begin
232         O_pos_a <= O_pos_d;
233         O_pos_d <= O_pos_a;
234         O_moving <= 1;
235     end
236     else if(O_pos_b == 4'b1000) begin
237         O_pos_b <= O_pos_e;
238         O_pos_e <= O_pos_b;
239         O_moving <= 1;
240     end
241     else if(O_pos_c == 4'b1000) begin
242         O_pos_c <= O_pos_f;
243         O_pos_f <= O_pos_c;
244         O_moving <= 1;
245     end
246     else if(O_pos_d == 4'b1000) begin
247         O_pos_d <= O_pos_g;
248         O_pos_g <= O_pos_d;
249         O_moving <= 1;
250     end
251     else if(O_pos_e == 4'b1000) begin
252         O_pos_h <= O_pos_e;
253         O_pos_e <= O_pos_h;
254         O_moving <= 1;
255     end
256     else if(O_pos_f == 4'b1000) begin
257         O_pos_f <= O_pos_i;
258         O_pos_i <= O_pos_f;
259         O_moving <= 1;
260     end
261     else O_moving <= 0;
262 end
263 default: O_moving <= 0;
264 endcase
265 end
266 else begin
267     casex ({w_left, w_right, w_up, w_down})

```

```

268     4'b1xxx: begin
269         if(O_pos_b[1:0] == 2'b10) begin
270             O_pos_b <= O_pos_a;
271             O_pos_a <= O_pos_b;
272             O_moving <= 1;
273         end
274         else if(O_pos_d[1:0] == 2'b10) begin
275             O_pos_d <= O_pos_c;
276             O_pos_c <= O_pos_d;
277             O_moving <= 1;
278         end
279         else O_moving <= 0;
280     end
281     4'b01xx: begin
282         if(O_pos_a[1:0] == 2'b10) begin
283             O_pos_b <= O_pos_a;
284             O_pos_a <= O_pos_b;
285             O_moving <= 1;
286         end
287         else if(O_pos_c[1:0] == 2'b10) begin
288             O_pos_c <= O_pos_d;
289             O_pos_d <= O_pos_c;
290             O_moving <= 1;
291         end
292         else O_moving <= 0;
293     end
294     4'b001x: begin
295         if(O_pos_c[1:0] == 2'b10) begin
296             O_pos_a <= O_pos_c;
297             O_pos_c <= O_pos_a;
298             O_moving <= 1;
299         end
300         else if(O_pos_d[1:0] == 2'b10) begin
301             O_pos_b <= O_pos_d;
302             O_pos_d <= O_pos_b;
303             O_moving <= 1;
304         end
305         else O_moving <= 0;
306     end
307     4'b0001: begin
308         if(O_pos_a[1:0] == 2'b10) begin
309             O_pos_c <= O_pos_a;
310             O_pos_a <= O_pos_c;
311             O_moving <= 1;
312         end
313         else if(O_pos_b[1:0] == 2'b10) begin
314             O_pos_d <= O_pos_b;
315             O_pos_b <= O_pos_d;
316             O_moving <= 1;
317         end
318         else O_moving <= 0;
319     end
320     default: O_moving <= 0;
321     endcase
322
323 end
324
325 end

```

```
326
327 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。

```
1  module vga(
2      input I_clk_25M,
3      input I_rst_n,
4      output reg [3:0] O_red,
5      output reg [3:0] O_green,
6      output reg [3:0] O_blue,
7      output O_hs,
8      output O_vs,
9
10     input [11:0] I_pixel_data,
11     output [9:0] O_pixel_x,
12     output [9:0] O_pixel_y,
13     output O_pixel_valid
14 );
15
16     parameter C_H_SYNC_PULSE      = 10'd96 ,
17               C_H_BACK_PORCH      = 10'd48 ,
18               C_H_ACTIVE_TIME     = 10'd640 ,
19               C_H_FRONT_PORCH     = 10'd16 ,
20               C_H_LINE_PERIOD     = 10'd800 ;
21
22     parameter C_V_SYNC_PULSE      = 10'd2 ,
23               C_V_BACK_PORCH      = 10'd33 ,
24               C_V_ACTIVE_TIME     = 10'd480 ,
25               C_V_FRONT_PORCH     = 10'd10 ,
26               C_V_FRAME_PERIOD    = 10'd525 ;
27
28     reg [9:0] R_h_cnt;
29     reg [9:0] R_v_cnt;
30
31     wire w_active_flag;
32
33     //行时序
34     always @(posedge I_clk_25M or negedge I_rst_n) begin
35         if(!I_rst_n) R_h_cnt <= 10'd0;
36         else if(R_h_cnt == C_H_LINE_PERIOD - 1) R_h_cnt <= 10'd0;
37         else R_h_cnt <= R_h_cnt + 1;
38     end
39
40     //场时序
41     always @(posedge I_clk_25M or negedge I_rst_n) begin
42         if(!I_rst_n) R_v_cnt <= 10'd0;
43         else if (R_v_cnt == C_V_FRAME_PERIOD - 1) R_v_cnt <= 10'd0;
44         else if (R_h_cnt == C_H_LINE_PERIOD - 1) R_v_cnt <= R_v_cnt + 1;
45         else R_v_cnt <= R_v_cnt;
46     end
47
48     //行同步信号 & 场同步信号 低电平有效
49     assign O_hs = (R_h_cnt < C_H_SYNC_PULSE) ? 1'b0 : 1'b1;
```

```

50     assign O_vs = (R_v_cnt < C_V_SYNC_PULSE) ? 1'b0 : 1'b1;
51
52     assign w_active_flag = (R_h_cnt >= (C_H_SYNC_PULSE + C_H_BACK_PORCH)) &&
53                             (R_h_cnt < (C_H_SYNC_PULSE + C_H_BACK_PORCH +
C_H_ACTIVE_TIME)) &&
54                             (R_v_cnt >= (C_V_SYNC_PULSE + C_V_BACK_PORCH)) &&
55                             (R_v_cnt < (C_V_SYNC_PULSE + C_V_BACK_PORCH +
C_V_ACTIVE_TIME));
56
57
58     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;
59     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;
60     assign O_pixel_valid = w_active_flag;
61
62
63     always @(posedge I_clk_25M, negedge I_rst_n) begin
64         if(!I_rst_n) begin
65             O_red <= 4'b0;
66             O_green <= 4'b0;
67             O_blue <= 4'b0;
68         end
69         else if(w_active_flag) begin
70             O_blue <= I_pixel_data[3:0];
71             O_green <= I_pixel_data[7:4];
72             O_red <= I_pixel_data[11:8];
73         end
74         else begin
75             O_red <= 4'b0;
76             O_green <= 4'b0;
77             O_blue <= 4'b0;
78         end
79     end
80
81 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， $20000000\text{ns}=20\text{ms}$ ）

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