数字逻辑 project-vga 拼图游戏

1、开发计划

选题：vga 拼图游戏
进度安排&执行记录：

2021 日历年　十二月 日历月

星期一	星期二	星期三	星期四	星期五	星期六	星期日
29	30	01	02	03	04 确认选题	05
06	07	08	09	10	11 分析需求，确定分工和安排	12
13	14	15	16 vga显示部分调试完成	17	18	19
20 游戏控制部分调试完成	21	22 晶体管显示部分调试完成；完成整体测试，拍摄视频和完成	23	24	25	26
27	28	29	30	31	01	02
03	04	05	06	07	08	09

2、设计

1) 需求分析

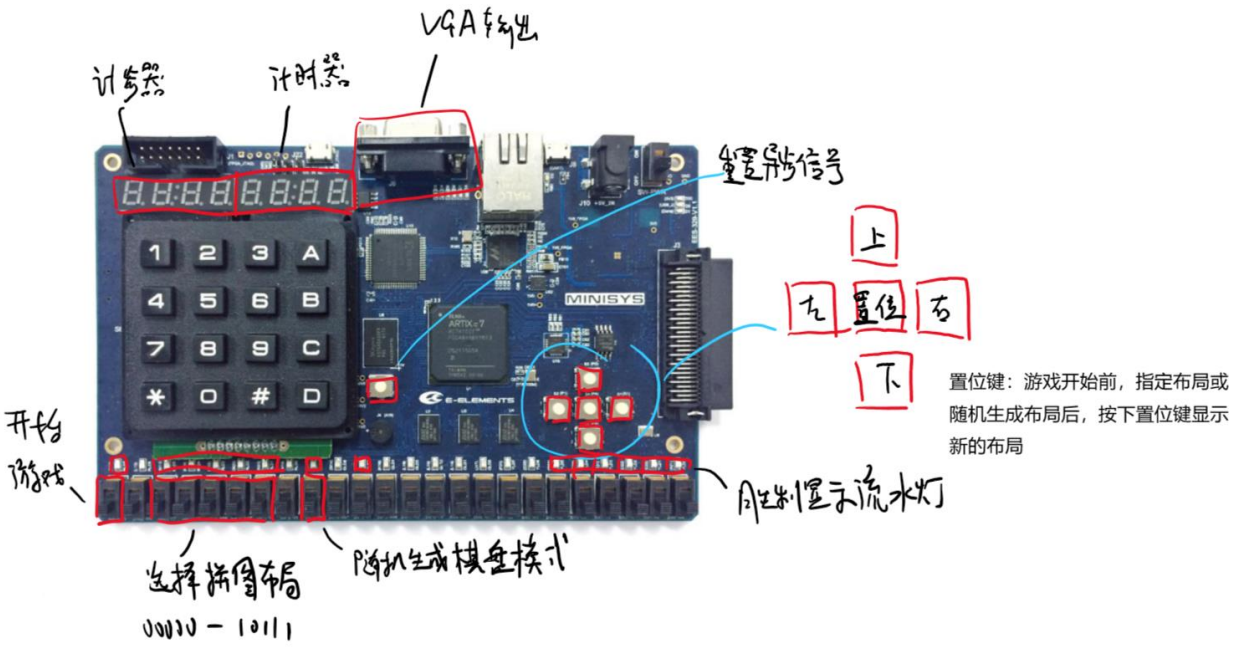
目标利用开发板的 vga 接口和按键等实现一个可操作和显示的拼图游戏。
主要功能：

利用 vga 接口在屏幕上显示画面；
随着不同的操作显示不同的画面；
基本游戏逻辑功能：

根据输入生成指定拼图布局 / 随机生成拼图布局；
开始游戏，结束游戏；
游戏开始后的合法操作；
游戏胜利；

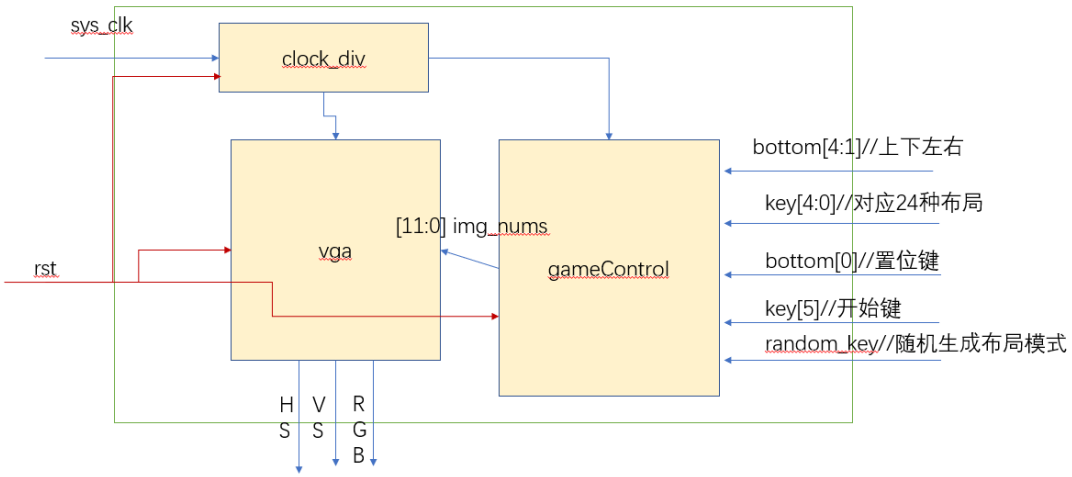
随着游戏状态触发的功能：计步器；计时器；胜利后显示流水灯；

使用端口和输入输出设备：



2) 系统结构设计

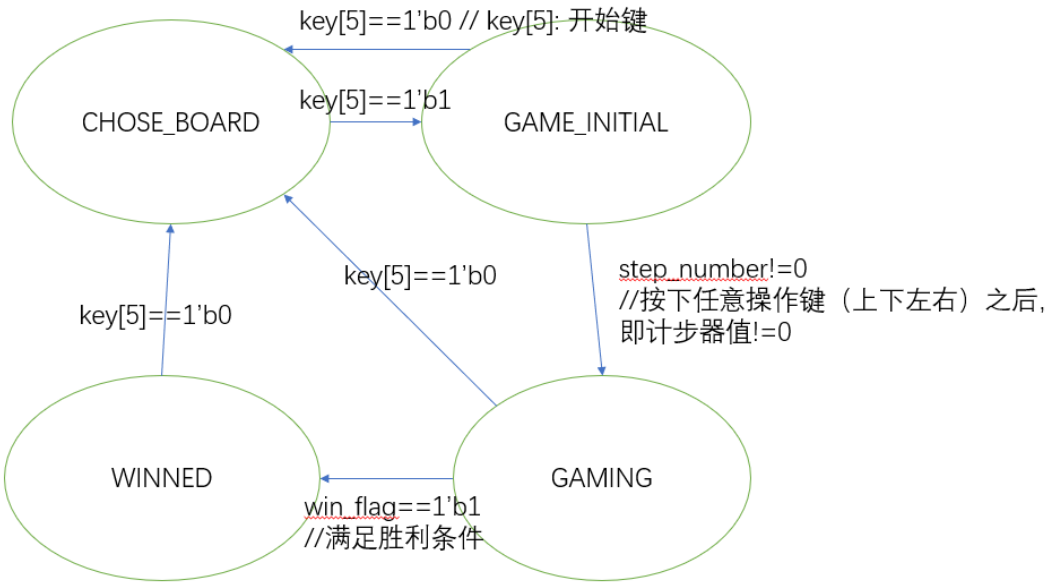
整体结构：



gameControl 模块传给 vga 模块一个 12 位的数据，如 12'b000_001_010_011，每三位（3'b000, 3'b001, 3'b010, 3'b011）表示图的某一块的位置，若为 3'b100，意为该位置为黑块。

游戏控制部分：

有限状态机：



3) 详细设计（部分核心代码）

顶层模块：

top.v

```
module top (input sys_clk,
            input rst,
            input [4:0]bottom,
            input [5:0]key,
            input random_key,
            output possi_led,
            output [7:0]DIG,
            output [7:0]Y,
            output [5:0]flow_led,
            output [3:0] red,
            output [3:0] green,
            output [3:0] blue,
            output hsync,
            vsync,
            output [5:0]key_led,
            output rd_led);
    wire clk;
    wire [11:0] img_nums;
    clock_div clk_d(sys_clk,rst,clk);

    vga vga(clk,rst,img_nums,red,green,blue,hsync,vsync);
    gameControl gc(sys_clk, clk,rst,key[4:0],key[5],bottom[0],b
    assign key_led = key;
    assign rd_led  = random_key;
endmodule
```

分频器把 100MHz 的频率降到 25MHz。
gameControl 模块负责处理游戏的逻辑，输出一个 12 位的数字 img_nums，表示该时刻屏幕上 4 个方格的图像；img_nums 输入到 vga 模块，然后把正确的图像输出到屏幕上。

VGA 显示部分：

vga.v：vga 显示控制模块

```
module vga (input clk,
            rst,
            input [11:0] img_nums,
            output reg [3:0] red,
            output reg [3:0] green,
            output reg [3:0] blue,
            output hsync,
            vsync);

    wire [11:0] h_cnt;
    wire [11:0] v_cnt;

    wire active_flag;
    wire [11:0] data;
    reg [17:0] addr;

    //sync
    vga_sync sync(clk, rst, h_cnt, v_cnt, hsync, vsync, active_flag);

    //img
    image_0 im_0(.addra(addr),.clka(clk), .douta(data));

    always @(posedge clk or posedge rst)
    begin
        if (rst) begin
            red   <= 4'b0000;
            green <= 4'b0000;
            blue  <= 4'b0000;
        end
        else if (active_flag && h_cnt - 144 <= 480) begin
            // a, c
            if (h_cnt - 144 <= 240) begin
            // a
                if (v_cnt - 35 <= 240) begin
                    case(img_nums[11:9])
                        3'b000: addr  <= (v_cnt - 35) * 480 + (h_cnt - 144);
                        3'b001: addr  <= (v_cnt - 35) * 480 + (h_cnt - 144) + 240;
                        3'b010: addr  <= (v_cnt - 35) * 480 + (h_cnt - 144) + 115200;
                        3'b011: addr  <= (v_cnt - 35) * 480 + (h_cnt - 144) + 115440;
                        default: addr <= 230400;
                    endcase
                end
                // c
                else begin…
```

//篇幅原因省略了其他几种 case 的代码

```
            if (data != 230400) begin
                red   <= data[11:8];
                green <= data[7:4];
                blue  <= data[3:0];
            end
            else begin
                red   <= 4'b0000;
                green <= 4'b0000;
                blue  <= 4'b0000;
            end
        end
        else begin
            red   <= 4'b0000;
            green <= 4'b0000;
            blue  <= 4'b0000;
        end
    end

endmodule
```

VGA 模块可以 640*480 分辨率显示一张 480*480 的图片。
图片读取部分内置在 vga 模块中。模块接受之前提到的 12 位数据 img_nums，以此计算每一格包含的像素的地址偏移量，从而读取图片数据。

根据 img_nums 每 3 位的值，计算地址偏移量

vga_sync.v：根据 vga 显示原理生成同步信号

```
module vga_sync
(
    input clk, rst,
    output reg [11:0] h_cnt,
    output reg [11:0] v_cnt,
    output hsync, vsync, active_flag
);

//hsync
always @(posedge clk or posedge rst)
begin
    if(rst)
        h_cnt <= 12'd0;
    else if(h_cnt == 799)
        h_cnt <= 12'd0;
    else
        h_cnt <= h_cnt + 1;
end
assign hsync = (h_cnt < 96) ? 0 : 1;

//vsync
always @(posedge clk or posedge rst)
begin
    if(rst)
        v_cnt <= 12'd0;
    else if(v_cnt == 524)
        v_cnt <= 12'd0;
    else if(h_cnt == 799)
        v_cnt <= v_cnt + 1;
    else
        v_cnt <= v_cnt;
end
assign vsync = (v_cnt < 2) ? 1'b0 : 1'b1;

assign active_flag = (h_cnt >= 144) && (h_cnt <= 624) && (v_cnt >= 35) && (v_cnt <= 515);
endmodule
```

游戏控制部分：

gameControl.v：游戏逻辑总控制模块

```
module gameControl (input sys_clk, clk_d,
                    input rst,
                    input [4:0] board_num_sw,
                    input start_sw,
                    input set_bt,
                    input [3:0] act_bt,
                    input random_sw,
                    output possi_led,
                    output [7:0]DIG,
                    output [7:0]Y,
                    output [5:0]flow_led,
                    output [11:0]out);

    localparam CHOSE_BOARD  = 2'b00;
    localparam GAMING       = 2'b01;
    localparam GAME_INITIAL = 2'b10;
    localparam WINNED       = 2'b11;


    wire set_flag;
    wire[3:0] act_flag;
    bottomFlag stF(set_bt,clk_d,rst,set_flag);
    bottomFlag actF0(act_bt[0],clk_d,rst,act_flag[0]);
    bottomFlag actF1(act_bt[1],clk_d,rst,act_flag[1]);
    bottomFlag actF2(act_bt[2],clk_d,rst,act_flag[2]);
    bottomFlag actF3(act_bt[3],clk_d,rst,act_flag[3]);

    wire active;
    assign active = act_flag[0]|act_flag[1]|act_flag[2]|act_flag[3];

    wire [1:0] game_status;
    wire win_flag;
    wire [13:0]step_number;
    wire timer_en;
    wire ini_flag;
    fsm fsm(clk_d,rst,start_sw,win_flag,active,ini_flag,game_status,step_number,timer_en);
    play pC(game_status,clk_d,rst,act_flag,random_sw,set_flag,board_num_sw,ini_flag,possi_l


    flowLED fled(clk_d,rst,win_flag,flow_led);
    seg_functions sf(sys_clk, rst,timer_en, step_number,game_status, Y, DIG);
endmodule
```

把各个按键输入转化成防抖后的信号，fsm 模块输出游戏现在的状态，play 模块输出此时画面的 4 个方格的信息。

fsm.v：有限状态机

```
localparam CHOSE_BOARD  = 2'b00;
localparam GAMING       = 2'b01;
localparam GAME_INITIAL = 2'b10;
localparam WINNED       = 2'b11;
```

```
//game status update
always @(posedge clk_d,posedge rst) begin
    if (rst)
        game_status <= CHOSE_BOARD;
    else if (~start_sw)
        game_status <= CHOSE_BOARD;
    else if (start_sw&win_flag)
        game_status <= WINNED;
    else if (start_sw&!ini_flag&!win_flag)
        game_status <= GAME_INITIAL;
    else
        game_status <= GAMING;
end
```

CHOSE_BOARD 状态：游戏未开始，可以通过置位键换棋盘，屏幕无黑块
GAME_INITIAL 状态：初始化黑块，只持续一个时钟周期
GAMING：开始游戏，黑块可移动，开始计步，计时
WINNED：游戏停止，黑块不可移动，定格画面，记步计时停止，显示流水灯

play.v：核心游戏逻辑控制模块

```
//game_status
localparam CHOSE_BOARD  = 2'b00;
localparam GAMING       = 2'b01;
localparam GAME_INITIAL = 2'b10;
localparam WINNED       = 2'b11;

//black_position
localparam LEFT_UP    = 2'b00;
localparam LEFT_DOWN  = 2'b10;
localparam RIGHT_UP   = 2'b01;
localparam RIGHT_DOWN = 2'b11;

reg[11:0]out;       //board without black block
reg [11:0]out_game;//board of output, with or without balck block depending on game_status
reg [1:0]black_pos;//position of black block

wire[11:0]rand_gen;//board generated from random
random rd(clk_d,rst,rand_gen);//random board generator

always @(posedge clk_d,posedge rst) begin
    if (rst)begin
        black_pos <= LEFT_DOWN;
        out       <= 12'b000_001_010_011;
    end
    else  begin
        case(game_status)
            CHOSE_BOARD:begin
            if (set&~random)begin
            case(num)
                5'b00000:out <= 12'b000_001_011_010; //0132
                5'b00001:out <= 12'b000_010_011_001; //0231
                5'b00010:out <= 12'b001_000_011_010; //1032
                5'b00011:out <= 12'b001_010_011_000; //1230
```

选择布局阶段（CHOSE_BOARD），根据 5 位的输入来指定输出某一种布局，只有按下置位键才生效

//篇幅原因省略了其他 20 种 case 的代码

```
                    default:out <= out;
                endcase
            end
            else if (set&random)  begin
                out <= rand_gen;
            end
            else begin
                out <= out;
            end
        end
        GAME_INITIAL:begin
            ini_flag <= 1'b1;
            out      <= out;
            casex(out)
                12'b011_xxx_xxx_xxx:black_pos <= LEFT_UP;
                12'bxxx_011_xxx_xxx:black_pos <= RIGHT_UP;
                12'bxxx_xxx_011_xxx:black_pos <= LEFT_DOWN;
                12'bxxx_xxx_xxx_011:black_pos <= RIGHT_DOWN;
                default: black_pos            <= LEFT_DOWN;
            endcase
        end
```

如果随机模式启动，则按下置位键时生成输出随机生成的布局

游戏初始化阶段（GAME_INITIAL）：把黑块位置定位到 3'b011 表示的那一块，后续所有的操作都与 black_pos 有关

```
            GAMING:begin
                ini_flag <= 1'b1;
                case(black_pos)
                    LEFT_UP:begin
                        case(act)
                            4'b0010:begin
                                out       <= {out[8:6],out[11:9],out[5:0]};
                                black_pos <= RIGHT_UP;
                            end
                            4'b0100:begin
                                out       <= {out[5:3],out[8:6],out[11:9],out[2:0]};
                                black_pos <= LEFT_DOWN;
                            end
                            default:begin
                                out       <= out;
                                black_pos <= black_pos;
                            end
                        endcase
                    end
```

游戏阶段（GAMING）：对于黑块目前的每种位置，只有某两种移动是有效的，对应两种交换操作，同时更新 back_pos 的位置

//篇幅原因省略了其他 3 种 case 的代码

```
                    default:begin
                        out       <= out;
                        black_pos <= black_pos;
                    end
                endcase
            end
            WINNED:begin
                ini_flag  <= 1'b1;
                black_pos <= black_pos;
                out       <= out;
            end
            default:begin
                ini_flag  <= 1'b1;
                black_pos <= black_pos;
                out       <= out;
            end
        endcase
    end
end
```

胜利阶段（WINNED）：如果不把开始键关掉，图像不再变化

```
 //winning detection
 always @(posedge clk_d) begin
    case(out_game)
        12'b000_001_010_100:win_flag <= 1'b1;
        default:            win_flag <= 1'b0;
    endcase
 end

 //print black block
 always @(posedge clk_d,posedge rst) begin
    if (rst)out_game <= 12'b000_001_010_011;
    else begin
    if (game_status == GAME_INITIAL|game_status == GAMING)begin
        case(black_pos)
            LEFT_UP:out_game    <= {3'b100,(out[8:0])};
            LEFT_DOWN:out_game  <= {(out[11:6]),3'b100,(out[2:0])};
            RIGHT_DOWN:out_game <= {(out[11:3]),3'b100};
            RIGHT_UP:out_game   <= {(out[11:9]),3'b100,(out[5:0])};
            default:out_game    <= out_game;
        endcase
    end
    else if (game_status == WINNED)
        out_game <= out_game;
    else
        out_game <= out;
 end
```

胜利条件检测：检查输出时候对应一共四种状态（黑块在四个角落）

out 里储存的是没有黑块的位置信息，如果是在游戏阶段，对应 black_pos 的值加上黑块（out_game 变量）

random.v：随机棋盘生成器

```
reg[4:0] regi;

always @(posedge clk_d,posedge rst) begin
    if (rst)
        regi <= 0;
    else if (regi>=5'b11000)
        regi <= 0;
    else
        regi <= regi+1'b1;
end

always @(posedge clk_d,posedge rst) begin
    if (rst)
        rand_out <= 12'b000_001_010_011;
    else begin
        case(regi)
            5'b00000:rand_out <= 12'b000_001_011_010; //0132
            5'b00001:rand_out <= 12'b000_010_011_001; //0231
            5'b00010:rand_out <= 12'b001_000_011_010; //1032
//篇幅原因省略了其他几种 case 的代码
            default:rand_out <= rand_out;
        endcase
    end
end
```

用 counter 的值来生成棋盘

bottomFlag.v：按键防抖

```
module bottomFlag(input key,
                  clk,
                  clr,
                  output key_changed2);
    wire key_changed1;
    reg [20:0] count;
    reg sample1, sample_locked1, sample2, sample_locked2;

    always @(posedge clk or posedge clr)
        if (clr) sample1 <= 0;
        else sample1     <= key;

    always @(posedge clk or posedge clr)
        if (clr) sample_locked1 <= 0;
        else sample_locked1     <= sample1;

    assign key_changed1 = ~sample_locked1 & sample1; //key pressed

    always @(posedge clk or posedge clr)
        if (clr) count              <= 0;
        else if (key_changed1) count <= 0;
        else count                  <= count + 1;

    always @(posedge clk or posedge clr)
        if (clr) sample2 <= 0;
        else if (count == 2000000)
        sample2 <= key;

    always @(posedge clk or posedge clr)
        if (clr) sample_locked2 <= 0;
        else sample_locked2     <= sample2;

    assign key_changed2 = ~sample_locked2 & sample2; //key pressed for 2000000 clk

endmodule
```

输出的 key_changed 信号只持续一个时钟周期，表示按键已经按下并且持续 2000000 个时钟周期

计步器，计时器，分频器，流水灯，晶体管显示等代码较简单，具体见代码文件，篇幅原因这里不展示。

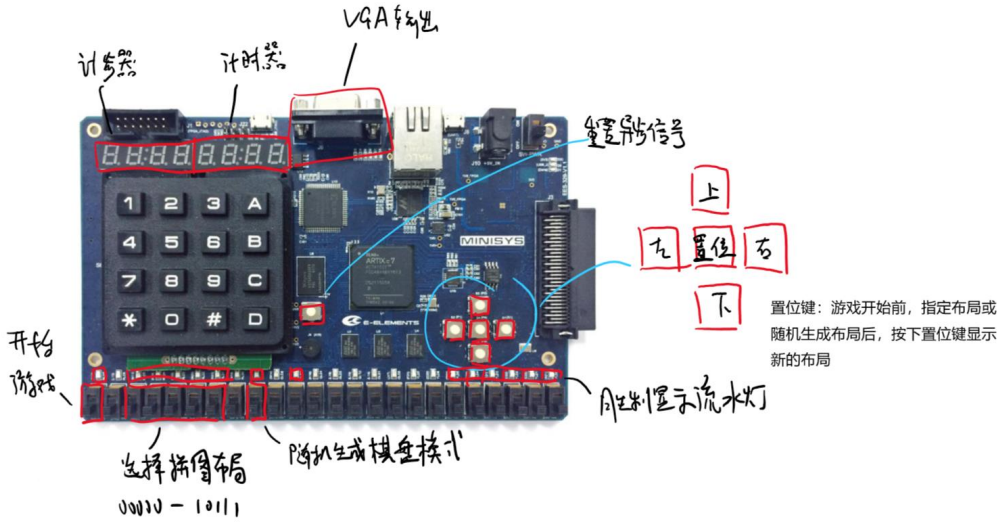
算法判断是否有解的功能采用了穷举法比对的方式，2*2 的布局有 24 种情况，其中 12 种有解，穷举部分代码较简单，见代码文件，篇幅原因这里不展示。

约束文件：

```
set_property PACKAGE_PIN Y18 [get_ports sys_clk]
set_property PACKAGE_PIN P20 [get_ports rst]
set_property PACKAGE_PIN G17 [get_ports {red[0]}]
set_property PACKAGE_PIN G18 [get_ports {red[1]}]
set_property PACKAGE_PIN J15 [get_ports {red[2]}]
set_property PACKAGE_PIN H15 [get_ports {red[3]}]
set_property PACKAGE_PIN H17 [get_ports {green[0]}]
set_property PACKAGE_PIN H18 [get_ports {green[1]}]
set_property PACKAGE_PIN J22 [get_ports {green[2]}]
set_property PACKAGE_PIN H22 [get_ports {green[3]}]
set_property PACKAGE_PIN H20 [get_ports {blue[0]}]
set_property PACKAGE_PIN G20 [get_ports {blue[1]}]
set_property PACKAGE_PIN K21 [get_ports {blue[2]}]
set_property PACKAGE_PIN K22 [get_ports {blue[3]}]
set_property PACKAGE_PIN M21 [get_ports hsync]
set_property PACKAGE_PIN L21 [get_ports vsync]
set_property PACKAGE_PIN P1 [get_ports {bottom[4]}]
set_property PACKAGE_PIN P2 [get_ports {bottom[3]}]
set_property PACKAGE_PIN R1 [get_ports {bottom[2]}]
set_property PACKAGE_PIN P5 [get_ports {bottom[1]}]
set_property PACKAGE_PIN P4 [get_ports {bottom[0]}]
set_property PACKAGE_PIN K17 [get_ports {key_led[5]}]
set_property PACKAGE_PIN M13 [get_ports {key_led[4]}]
set_property PACKAGE_PIN F15 [get_ports {Y[0]}]
set_property PACKAGE_PIN F13 [get_ports {Y[1]}]
set_property PACKAGE_PIN F14 [get_ports {Y[2]}]
set_property PACKAGE_PIN F16 [get_ports {Y[3]}]
set_property PACKAGE_PIN E17 [get_ports {Y[4]}]
set_property PACKAGE_PIN C14 [get_ports {Y[5]}]
set_property PACKAGE_PIN C15 [get_ports {Y[6]}]
set_property PACKAGE_PIN E13 [get_ports {Y[7]}]
set_property PACKAGE_PIN A18 [get_ports {DIG[7]}]
set_property PACKAGE_PIN A20 [get_ports {DIG[6]}]
set_property PACKAGE_PIN B20 [get_ports {DIG[5]}]
set_property PACKAGE_PIN E18 [get_ports {DIG[4]}]
set_property PACKAGE_PIN F18 [get_ports {DIG[3]}]
set_property PACKAGE_PIN D19 [get_ports {DIG[2]}]
set_property PACKAGE_PIN E19 [get_ports {DIG[1]}]
set_property PACKAGE_PIN C19 [get_ports {DIG[0]}]
set_property PACKAGE_PIN D21 [get_ports {flow_led[4]}]
set_property PACKAGE_PIN D22 [get_ports {flow_led[2]}]
set_property PACKAGE_PIN E22 [get_ports {flow_led[1]}]
set_property PACKAGE_PIN A21 [get_ports {flow_led[0]}]
set_property PACKAGE_PIN M17 [get_ports rd_led]
```

```
set_property PACKAGE_PIN K14 [get_ports {key_led[3]}]
set_property PACKAGE_PIN K13 [get_ports {key_led[2]}]
set_property PACKAGE_PIN M20 [get_ports {key_led[1]}]
set_property PACKAGE_PIN N20 [get_ports {key_led[0]}]
set_property PACKAGE_PIN Y9 [get_ports {key[5]}]
set_property PACKAGE_PIN Y7 [get_ports {key[4]}]
set_property PACKAGE_PIN Y8 [get_ports {key[3]}]
set_property PACKAGE_PIN AB8 [get_ports {key[2]}]
set_property PACKAGE_PIN AA8 [get_ports {key[1]}]
set_property PACKAGE_PIN V8 [get_ports {key[0]}]
set_property PACKAGE_PIN AB6 [get_ports random_key]
set_property PACKAGE_PIN G21 [get_ports {flow_led[5]}]
set_property PACKAGE_PIN E21 [get_ports {flow_led[3]}]
set_property PACKAGE_PIN M15 [get_ports possi_led]

set_property PACKAGE_PIN Y9 [get_ports {key[5]}]
set_property PACKAGE_PIN Y7 [get_ports {key[4]}]
set_property PACKAGE_PIN Y8 [get_ports {key[3]}]
set_property PACKAGE_PIN AB8 [get_ports {key[2]}]
set_property PACKAGE_PIN AA8 [get_ports {key[1]}]
set_property PACKAGE_PIN V8 [get_ports {key[0]}]
set_property PACKAGE_PIN AB6 [get_ports random_key]
```



置位键：游戏开始前，指定布局或随机生成布局后，按下置位键显示新的布局

3、总结及优化

1) 总结:

实现了题目要求的所有基本功能，bonus 功能中除了串口读图和区块拓展以外的功能也全部实现了。
部分算法使用了穷举法来判断，尽量减少逻辑判断，以简化综合出来的电路。
游戏逻辑和模块间的关系较清晰和简单。

2) 优化方向

Play 模块中的两层 case 嵌套可以进一步减少到一层，电路应该会更加简化，但是会损失一点代码可读性。
Vga 显示图片的分辨率可以进一步拓展。
未实现的 bonus 功能：窗口读图，区块拓展。