同济大学计算机系

数字逻辑课程综合实验报告

学　　号 **2353900**

姓　　名 汪嘉晨

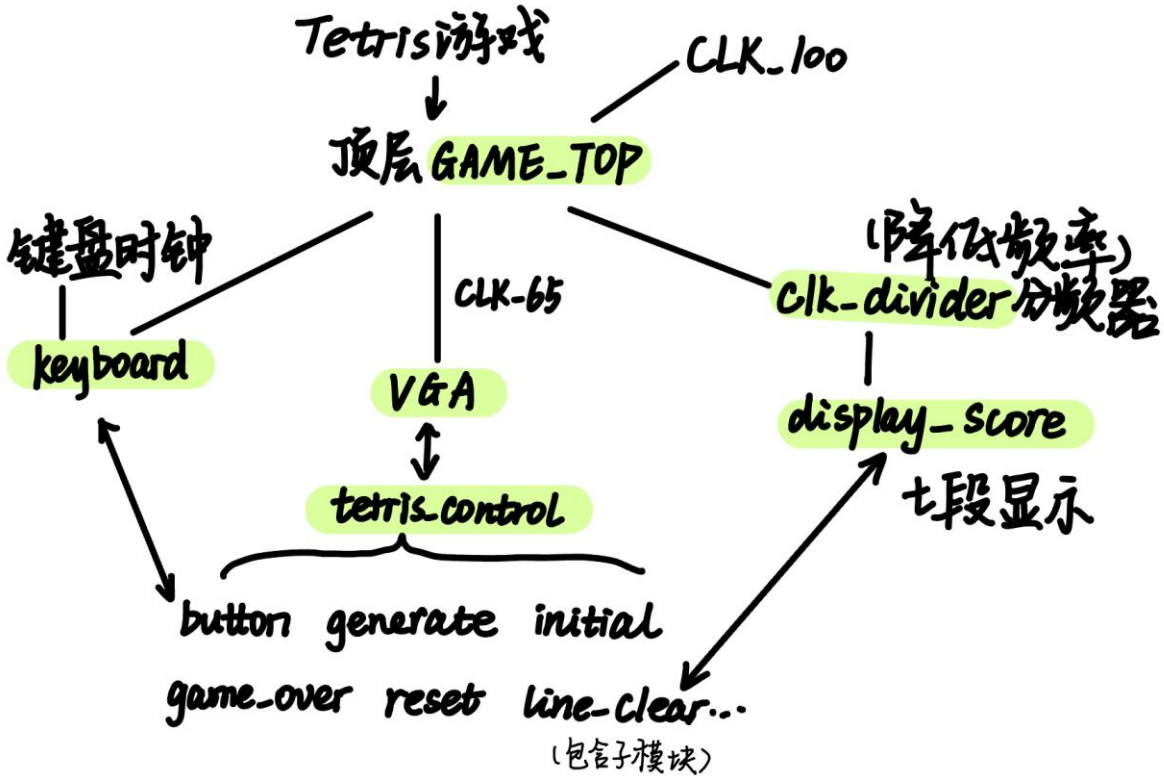
专　　业 计算机科学与技术（精英班）

授课老师 郭玉臣

一、实验内容

基于 FPGA 及 PS2 键盘与 VGA 显示屏实现的简易俄罗斯方块小游戏，VGA 显示，通过键盘实现方块的移动和旋转，并通过七段数字显示管实时显示当前消掉的行数。

二、俄罗斯方块小游戏数字系统总框图



GAME_TOP：控制器，链接各子模块，生成时钟信号和 rst、locked、开始游戏等控制信号

VGA：显示器模块

display_score：七段数码管显示当前分数

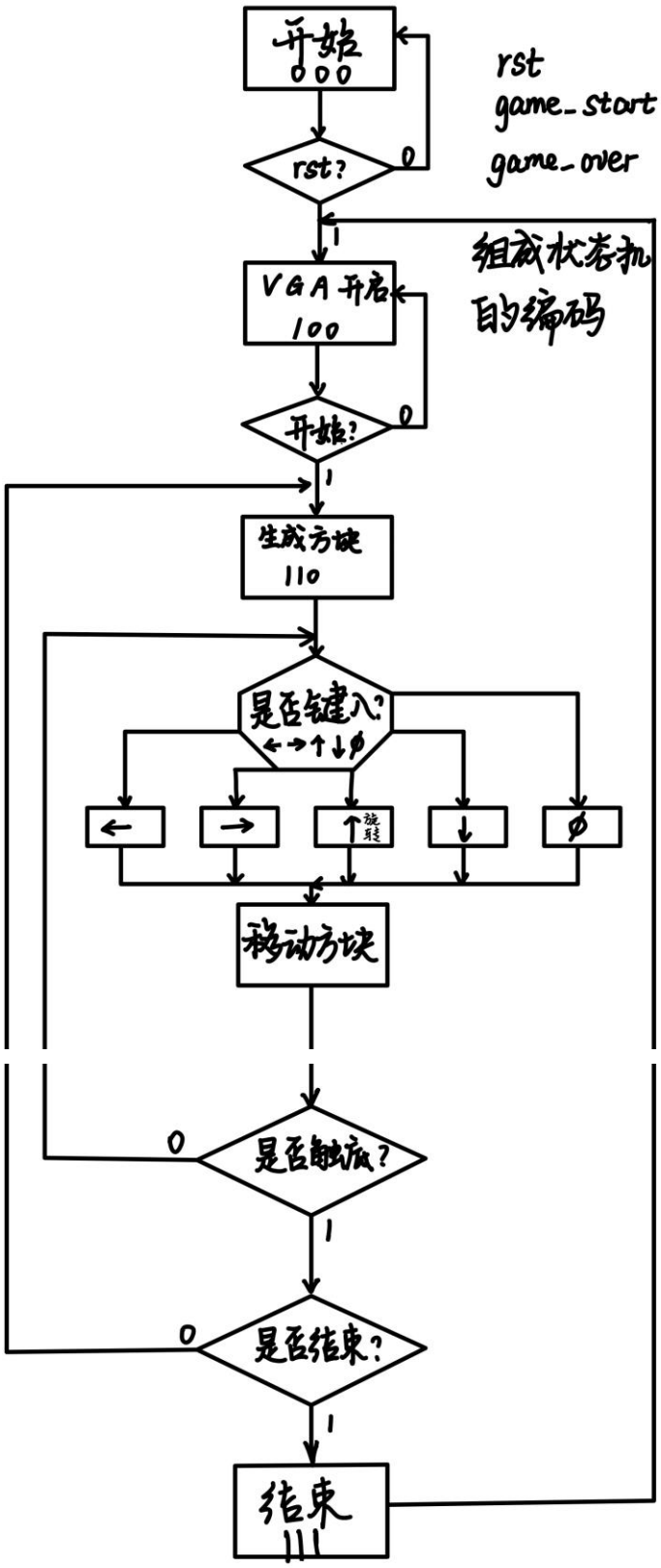
clk_divider：降低频率，为七段数码管显示分数提供相应的频率

keyboard：键盘模块，处理左、右、旋转、快速向下的操作

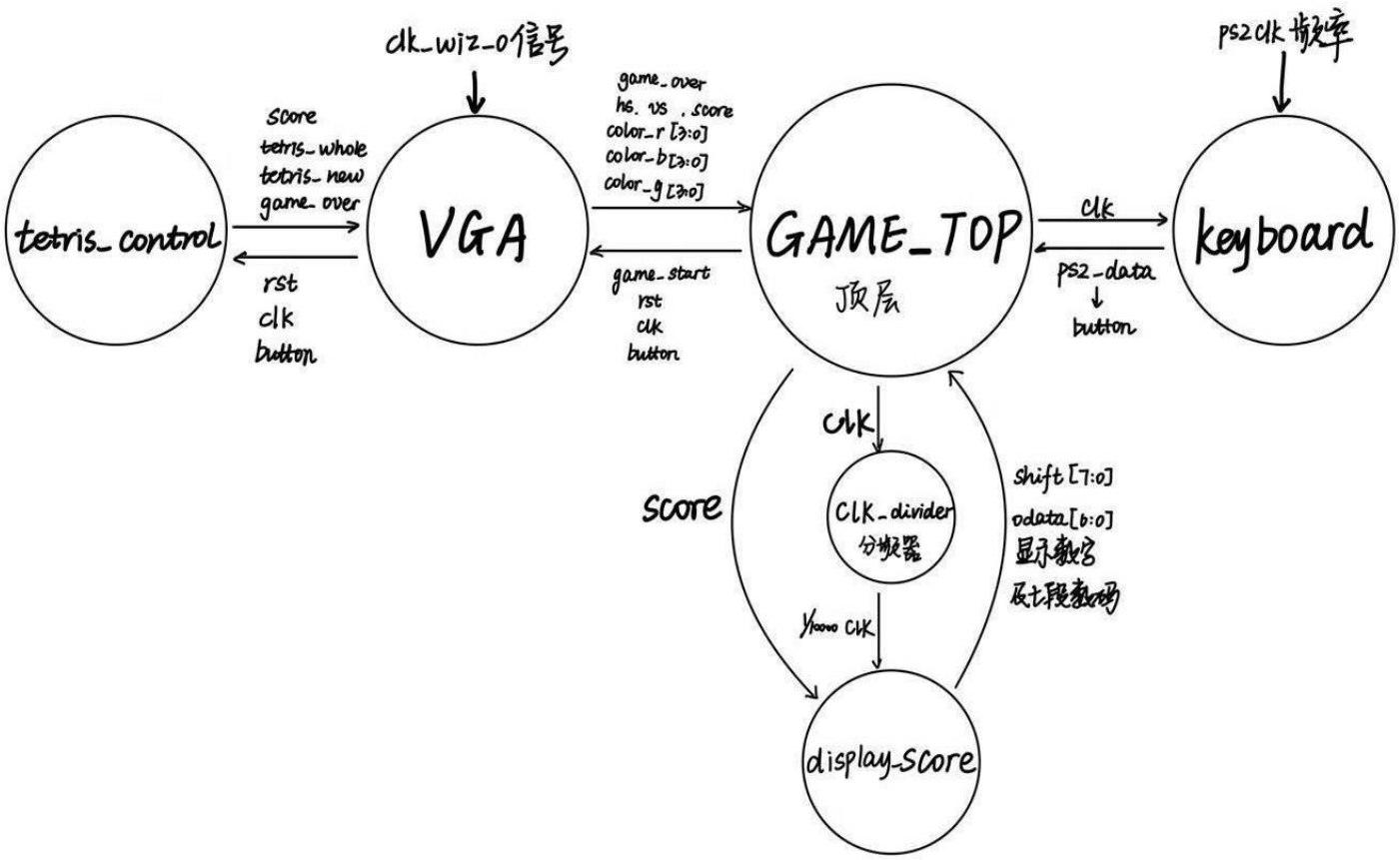
tetris_control：处理按钮链接，生成方块、初始化方块、方块移动、重合判断、游戏结束判断、判断消除行与加分、清空某行、重新开局等逻辑，并将信号输入 VGA 进行显示

三、系统控制器设计

GAME_TOP 顶层状态机如图所示：



四、子系统模块建模



1.VGA

该模块负责图像的输出，通过读取背景、轨道、色块中的信息输出相应图片

module tetris_control 子系统可以用来生成和改变方块

```
module VGA(
    input               game_start,
    input               clk,
    input               rst,                //置位
    input [3:0]         button,             //按键 3210 为左右旋转加速向下
```

```
    output reg [3:0]    color_r,    //R
    output reg [3:0]    color_g,    //G
    output reg [3:0]    color_b,    //B
    output               hs,        //行同步
    output               vs,        //场同步
    output               game_over, //游戏结束
    output      [32:0]   score      //得分
    );
```

2.tetris_control

```
module tetris_control(
    input rst,
    input clk,
    input[3:0] button,                //按键 3210 依次为左 右 旋转 加速向下
    output reg [199:0] tetris_whole,
    output reg [199:0] tetris_new,
    output reg game_over,
    output reg [31:0] score
);
reg over;
reg [15:0]randn;
reg [3:0] button_last;
```

3.keyboard

本部分负责为输入 button 对应的旋转，快速向下，向左与向右：

```
module ps2_keyboard(
    input clk,      // 系统时钟
    input rst,      // 复位信号
    input ps2_clk,  // PS2 时钟信号
    input ps2_data, // PS2 数据信号
```

```
    output reg [3:0] button // 按键控制游戏方块 3210 依次为左、右、旋转、加速向下
);
    // 定义按键扫描码
    parameter LEFT_ARROW  = 8'h6B; // 左箭头键扫描码
    parameter RIGHT_ARROW = 8'h74; // 右箭头键扫描码
    parameter UP_ARROW    = 8'h75; // 上箭头键扫描码（旋转）
    parameter DOWN_ARROW  = 8'h72; // 下箭头键扫描码（加速向下）
```

4.clk_divider

本部分负责为七段数码管显示分数分频

```
input clk_in,       // 输入时钟
    output reg clk_out  // 输出分频后的时钟
);
    reg [15:0] clk_div; // 分频器计数器
```

5.display_score

本部分负责七段数码管显示分数，将 32 位 score 转换成 BCD 码再 clk_divider 转换的频率下进行显示：

```
// BCD 转换
module Bin2BCD(
    input   [31:0] number,     // 数字
    output [3:0] bcd0,
    output [3:0] bcd1,
    output [3:0] bcd2,
    output [3:0] bcd3,
    output [3:0] bcd4,
    output [3:0] bcd5,
    output [3:0] bcd6,
    output [3:0] bcd7
);
```

```
    reg [31:0] bin;
    reg [31:0] result;
    reg [31:0] bcd;
```

```
input clk_65hz,
    input [31:0] score,
    output reg [7:0] shift, // 第几个数码管(片选)
    output reg [6:0] oData
);
wire [3:0] Data[7:0];
reg [3:0] cnt = 0; // 计数器
wire slow_clk; // 分频后的慢时钟
// 实例化分频器
clk_divider clk_div_inst(
    .clk_in(clk_65hz),
    .clk_out(slow_clk)
);
```

五、测试模块建模

1.VGA 模块：

验证 VGA 模块的功能，测试文件模拟了时钟信号、复位信号、游戏开始信号和按键信号，并观察 VGA 模块的输出。以下是测试文件的逻辑步骤：

逻辑步骤

（1）初始化输入信号：初始化 game_start、clk、rst 和 button 信号为 0。

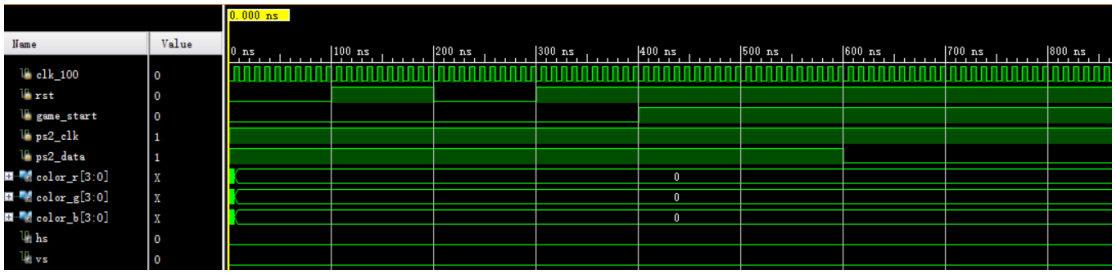
（2）复位过程：通过设置 rst 信号为 1，然后为 0，再次为 1，模拟复位过程。这个过程确保 VGA 模块在复位后处于已知状态。

（3）开始游戏：设置 game_start 信号为 1，表示游戏开始。

（4）模拟按键按下和松开：模拟按下和松开左箭头键（button = 4'b1000），右箭头键（button = 4'b0100），上箭头键（旋转，button = 4'b0010），下箭头键（加速向下，button = 4'b0001））每次按键按下和松开之间有 200ns 的延迟。

（5）结束仿真：在仿真运行 1000ns 后，停止仿真。

结果如图所示：



2.键盘模块：

验证 ps2_keyboard 模块的功能。它模拟了时钟信号、复位信号和 PS2 键盘信号，并观察 ps2_keyboard 模块的输出。以下是测试文件的逻辑步骤：

逻辑步骤

（1）初始化输入信号：初始化 clk、rst、ps2_clk 和 ps2_data 信号。

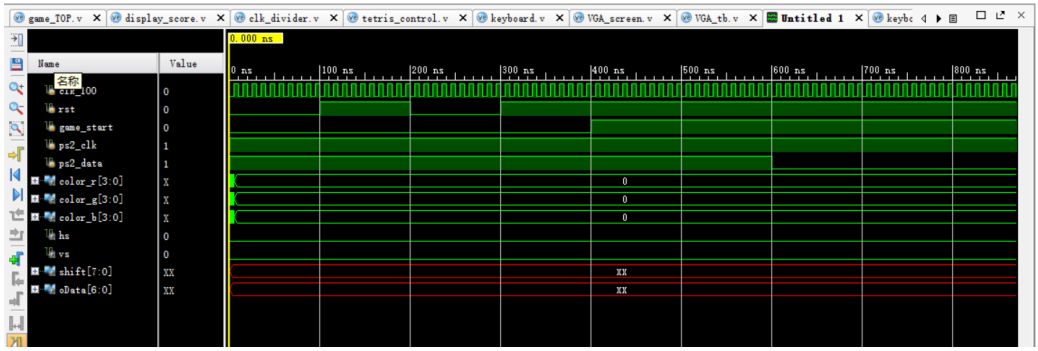
（2）复位过程：通过设置 rst 信号为 1，然后为 0，再次为 1，模拟复位过程。这个过程确保 ps2_keyboard 模块在复位后处于已知状态。

（3）模拟按键按下和松开：使用 send_ps2_data 任务模拟 PS2 键盘信号，按下和松开左箭头键、右箭头键、上箭头键（旋转）和下箭头键（加速向下）。每次按键按下和松开之间有 200ns 的延迟。

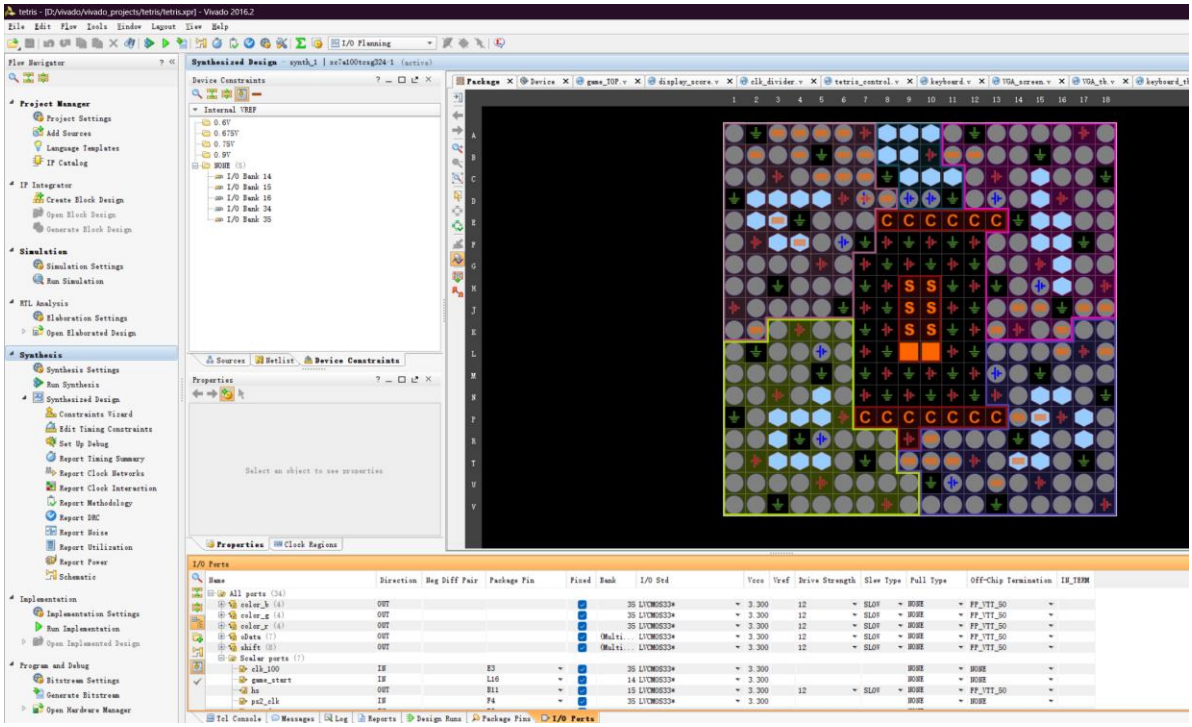
（4）结束仿真：在仿真运行 1000ns 后，停止仿真。

结果如图所示：

六、实验结果

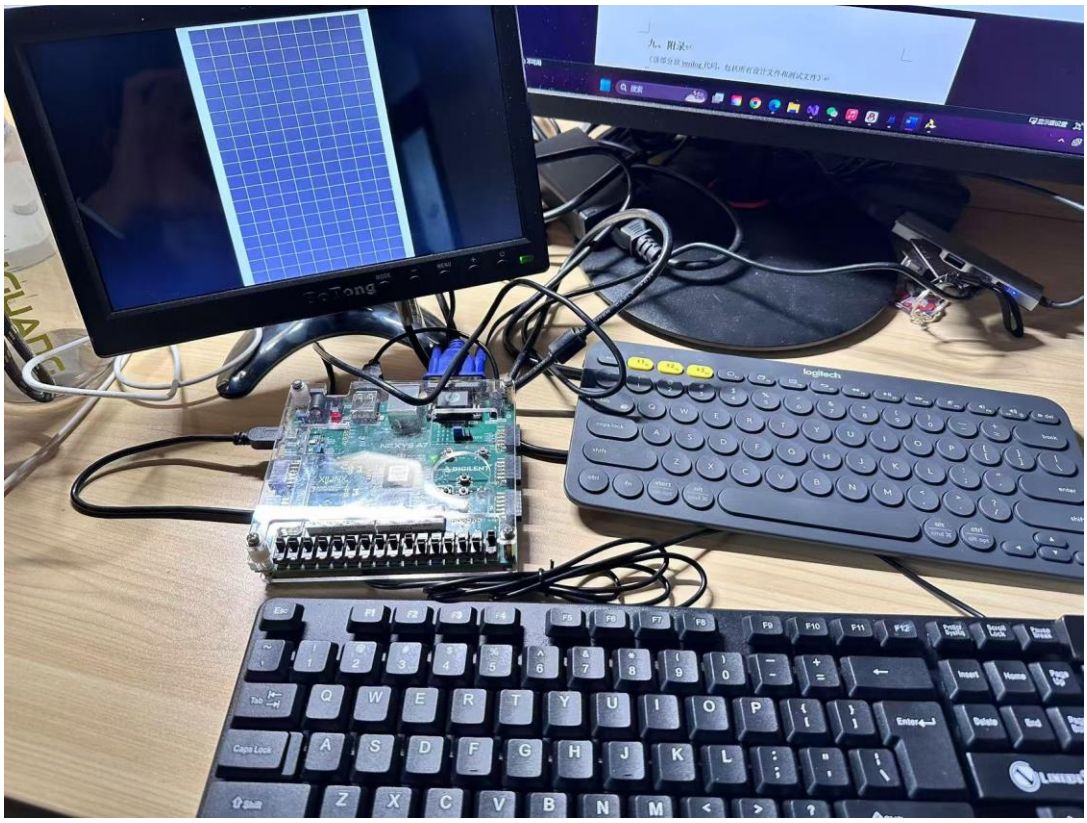


通过模拟：

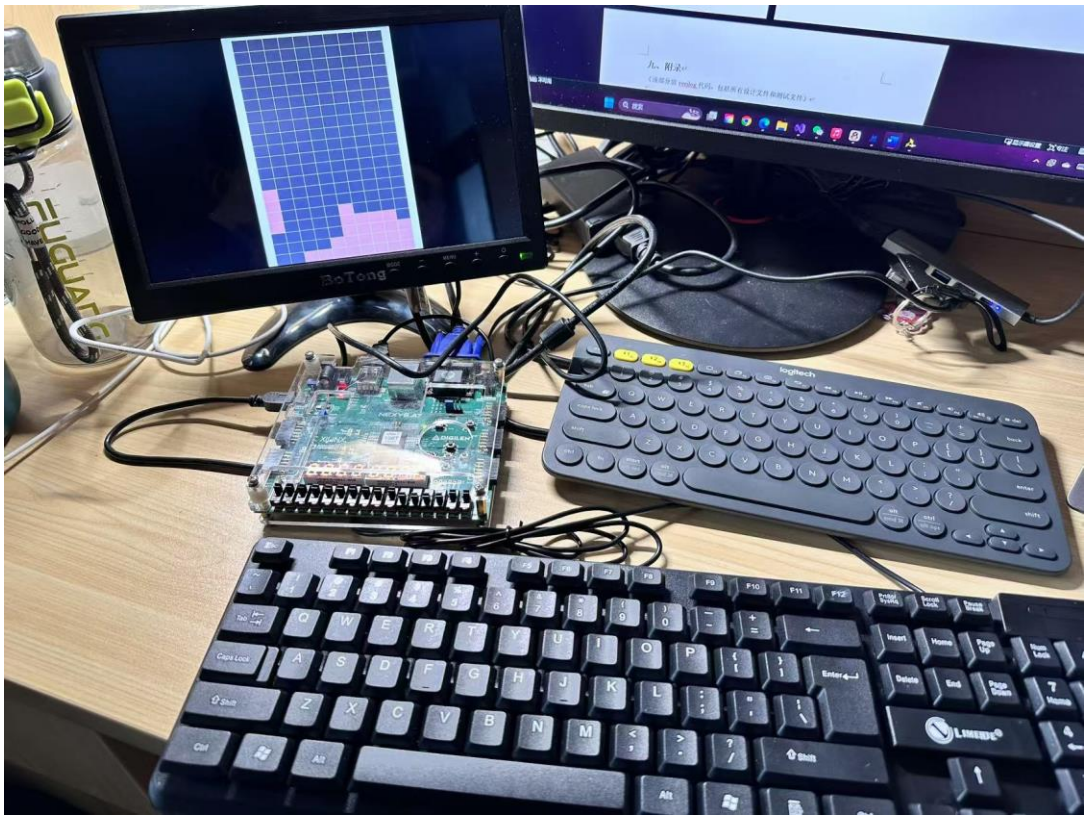


将开发板链接 VGA 与键盘：

打开 J15（启动程序）和 L16（开始游戏）即可开始游戏（如图）：

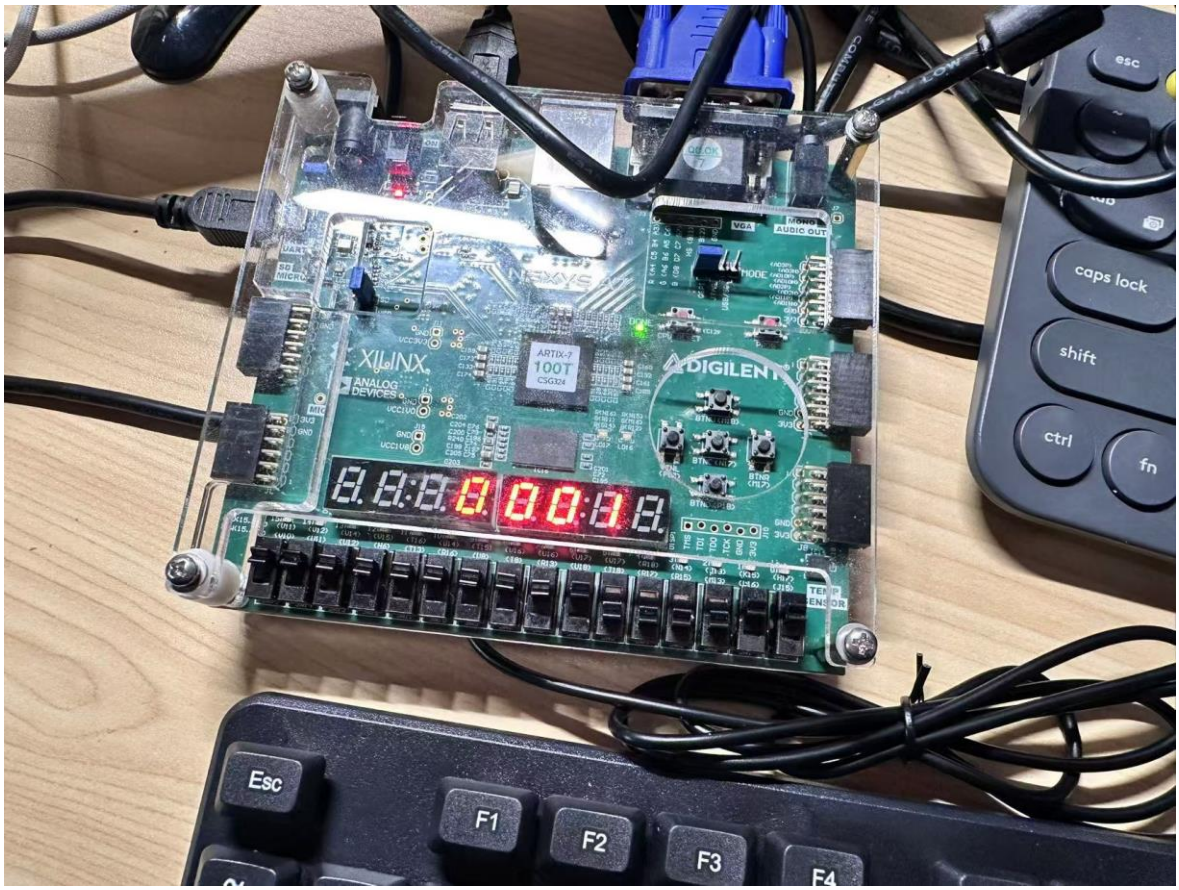


游戏进行中：



消除每行，分数+10：

注意到由于频闪问题只拍到了 0001，实际上是 00000010：



七、结论

本项目可以随机生成俄罗斯方块，根据键盘输入进行旋转平移等操作，可以消除已经满了的行，同时七段发光二极管可以显示分数。

八、心得体会及建议

对于**心得**：本次大作业让我学习了制作时钟和使用分频器，理解了如何控制时序、同步不同模块的操作，对于 verilog 语言、状态机等知识的运用也越发熟练。如何确保分频器正确工作、如何调节时钟频率以适应不同模块的需求，每解决一个问题，我都会有一种成就感。

硬件设计并不是单纯的拼接电路和模块，它还需要深入理解每个模块的时序需求以及如何高效地将这些需求融合在一起。将近一周断断续续地搓代码，我对我的进步感到自豪，当然我也承认这条路上并非风顺：

当下存在一些尚未解决的**问题：**

①键盘仍有循环的发生，不能够一次按动只记录一次有效结果，而是一直按照这个方向进行下去，如一直旋转、一直向左向右，当然人工解决方法也比较简单：点击向下就可以了（目前看来最为简单的方式）；

②在方块即将落地的时候按左移或者右移键可能会和旁边的方块重叠……

对于建议：诚然，数字逻辑这门计算机类课程需要我们有自学的能力，但是我感觉 1 个学分自学压力实在是太大了……verilog 语言的学习，平常做实验的一堆思路以及本次大作业“手搓火箭”感觉对我来说挑战性不小，当然我也从学长 github 开源的键盘引脚设置之类的小知识点不断学习，提升自己。也希望能够为学弟学妹们提供 verilog 自学课程（？比如文档什么的）

九、附录

1.game_top

```
`timescale 1ns / 1ps
module GAME_TOP(
    // 时钟
    input           clk_100,                // 100MHz 时钟
    // 复位控制
    input           rst,                    // 复位
    input           game_start,             // 开始
    // VGA
    output [3:0]    color_r,
    output [3:0]    color_g,
    output [3:0]    color_b,
    output          hs,
    output          vs,
    // 七段数码管
    output [7:0]    shift,
    output [6:0]    oData,
```

```
    // PS/2 键盘引脚
    input           ps2_clk,                // PS/2 键盘时钟
    input           ps2_data                // PS/2 键盘数据
);
    // 时钟
    wire clk_65, locked;
    // 游戏是否结束
    wire game_over;
    // 按键控制信号
    wire [3:0] button;
    // 得分
    wire [31:0] score;
    // 分频
    clk_wiz_0 uut(
        .clk_in1(clk_100),
        .clk_out1(clk_65),
        .reset(~rst),   // 1 有效
        .locked(locked)
    );
    // 实例化 PS/2 键盘模块
    ps2_keyboard uut_ps2_keyboard(
        .clk(clk_100),
        .rst(~rst),
        .ps2_clk(ps2_clk),
        .ps2_data(ps2_data),
        .button(button)
    );
    // VGA 模块实例化
    VGA uut_vga(
        .clk(clk_65),
```

```
        .rst(locked),
        .color_r(color_r),
        .color_g(color_g),
        .color_b(color_b),
        .hs(hs),
        .vs(vs),
        .game_over(game_over),
        .game_start(game_start),
        .button(button),
        .score(score)
    );
    // 数码管显示分数
    display_score uut_score(
        .clk_65hz(clk_65),
        .score(score),
        .shift(shift),
        .oData(oData)
    );
endmodule
```

2.clk_divider

```
`timescale 1ns / 1ps
module clk_divider(
    input clk_in,       // 输入时钟
    output reg clk_out  // 输出分频后的时钟
);
    reg [15:0] clk_div; // 分频器计数器
    always @(posedge clk_in) begin
        if (clk_div == 16'd49999) begin
            clk_div <= 0;
```

```
            clk_out <= ~clk_out;
        end else begin
            clk_div <= clk_div + 1;
        end
    end
endmodule
```

3.display_score

```
`timescale 1ns / 1ps
// BCD 转换
module Bin2BCD(
    input   [31:0] number,     // 数字
    output [3:0] bcd0,
    output [3:0] bcd1,
    output [3:0] bcd2,
    output [3:0] bcd3,
    output [3:0] bcd4,
    output [3:0] bcd5,
    output [3:0] bcd6,
    output [3:0] bcd7
);
    reg [31:0] bin;
    reg [31:0] result;
    reg [31:0] bcd;
    // 转换为BCD 码
    always @(number) begin
        bin = number[31:0];
        result = 32'd0;
        repeat (31) begin
            result[0] = bin[31];
```

```
            if (result[3:0] > 4)
                result[3:0] = result[3:0] + 4'd3;
            if (result[7:4] > 4)
                result[7:4] = result[7:4] + 4'd3;
            if (result[11:8] > 4)
                result[11:8] = result[11:8] + 4'd3;
            if (result[15:12] > 4)
                result[15:12] = result[15:12] + 4'd3;
            if (result[19:16] > 4)
                result[19:16] = result[19:16] + 4'd3;
            if (result[23:20] > 4)
                result[23:20] = result[23:20] + 4'd3;
            if (result[27:24] > 4)
                result[27:24] = result[27:24] + 4'd3;
            if (result[31:28] > 4)
                result[31:28] = result[31:28] + 4'd3;
            result = result << 1;
            bin = bin << 1;
        end
        result[0] = bin[31];
        bcd = result;
    end
    assign bcd0 = bcd[3:0];
    assign bcd1 = bcd[7:4];
    assign bcd2 = bcd[11:8];
    assign bcd3 = bcd[15:12];
    assign bcd4 = bcd[19:16];
    assign bcd5 = bcd[23:20];
    assign bcd6 = bcd[27:24];
    assign bcd7 = bcd[31:28];
```

```
endmodule
module display_score(
    input clk_65hz,
    input [31:0] score,
    output reg [7:0] shift, // 第几个数码管(片选)
    output reg [6:0] oData
);
wire [3:0] Data[7:0];
reg [3:0] cnt = 0; // 计数器
wire slow_clk; // 分频后的慢时钟
// 实例化分频器
clk_divider clk_div_inst(
    .clk_in(clk_65hz),
    .clk_out(slow_clk)
);
// 转换为BCD
Bin2BCD uut_bin2bcd(
    .number(score),
    .bcd0(Data[0]),
    .bcd1(Data[1]),
    .bcd2(Data[2]),
    .bcd3(Data[3]),
    .bcd4(Data[4]),
    .bcd5(Data[5]),
    .bcd6(Data[6]),
    .bcd7(Data[7])
);
// 片选输出
always @(posedge slow_clk) begin
    if (cnt == 4'd7)
```

```
        cnt <= 0;
    else
        cnt <= cnt + 1;
    shift <= ~(8'b00000001 << cnt); // 选择一个数码管进行输出
    case (Data[cnt])
        4'b0000: oData <= 7'b1000000; // 0
        4'b0001: oData <= 7'b1111001; // 1
        4'b0010: oData <= 7'b0100100; // 2
        4'b0011: oData <= 7'b0110000; // 3
        4'b0100: oData <= 7'b0011001; // 4
        4'b0101: oData <= 7'b0010010; // 5
        4'b0110: oData <= 7'b0000010; // 6
        4'b0111: oData <= 7'b1111000; // 7
        4'b1000: oData <= 7'b0000000; // 8
        4'b1001: oData <= 7'b0010000; // 9
        default: oData <= 7'b1111111; // 默认值
    endcase
end
endmodule
```

4.VGA_screen

```
`timescale 1ns / 1ps
module VGA(
    input               game_start,
    input               clk,
    input               rst,                //置位
    input [3:0]         button,             //按键  3210 依次为左 右 旋转 加速向下
    output reg [3:0]    color_r,    //R
    output reg [3:0]    color_g,    //G
```

```
output reg [3:0]    color_b,    //B
output               hs,        //行同步
output               vs,        //场同步
output               game_over, //游戏结束
output      [32:0]   score      //得分
);
//1024*768/60Hz
//行时序常数
parameter  HS_SYNC     = 136,   //同步脉冲
           HS_BACK     = 160,   //显示后沿
           HS_ACTIVE   = 1024,  //分辨率/显示区域
           HS_FRONT    = 24;    //显示前沿
//场时序常数
parameter  VS_SYNC     = 6,     //同步脉冲
           VS_BACK     = 29,    //显示后沿
           VS_ACTIVE   = 768,   //分辨率
           VS_FRONT    = 3;     //显示前沿
//帧长
parameter   COL = 1344,ROW = 806,FRAM = 60;
//计数器
reg [11:0]  h_cnt = 12'd0;      //行计数器
reg [11:0]  v_cnt = 12'd0;      //场计数器
reg [11:0]  f_cnt = 12'b0;      //帧计数器，用来控制方块下落
reg clk1s = 0;
//计数器转坐标
wire [11:0] x;
wire [11:0] y;//坐标
wire active;//显示是否可用
wire [199:0] tetris_whole;     //记录已经有的方块 10*20
wire [199:0] tetris_new;      //记录下落的方块
```

```
    //记录每一格子的像素
    parameter Tetris_Size = 38;
    //记录格子位置
    reg[4:0] tetris_x;
    reg[4:0] tetris_y;

    //开始显示
    always @(posedge clk) begin
        if(active) begin
            color_r = 4'd0; color_g = 4'd0; color_b = 4'd0; // 默认黑色
            // 最外围边框
            if(x >= 300 && x < 724) begin
                color_r <= 4'd15; color_g <= 4'd15; color_b <= 4'd15; /
/ 白色
            end
            // 内部区域
            if(x >= 322 && x < 702 && y >= 8 && y < 768) begin
                tetris_x = (x - 322) / Tetris_Size; // 计算方块的 x 位置
                tetris_y = (y - 8) / Tetris_Size;   // 计算方块的 y 位置

                // 背景颜色改为深灰色
                color_r <= 4'd5; color_g <= 4'd5; color_b <= 4'd5; // 深
灰色背景

                // 边线
                if(!((x - 322) % Tetris_Size) || !((y - 8) % Tetris_Siz
e)) begin
                    color_r <= 4'd15; color_g <= 4'd15; color_b <= 4'd1
5; // 白色边线
```

```
                end
                // 方块颜色
                else begin
                    if(tetris_whole[tetris_x * 20 + tetris_y] | tetris_
new[tetris_x * 20 + tetris_y]) begin // 方块
                        color_r <= 4'd15;
                        color_g <= 4'd10;
                        color_b <= 4'd10; // 红色方块
                    end
                end
            end
        end
        else begin
            color_r = 4'd0; color_g = 4'd0; color_b = 4'd0; // 游戏未激
活时为黑色
        end
    end
    //轨道
    tetris_control uut_tetris(
        .rst(game_start),
        .clk(clk1s),              //clk 为1，改变一次数组
        .tetris_whole(tetris_whole),
        .tetris_new(tetris_new),
        .game_over(game_over),
        .button(button),
        .score(score)
    );
    //行同步循环
    always @(posedge clk or negedge rst) begin
        if(!rst)                                        //复位信号置零
```

```
            h_cnt <= 12'd0;
        else if(h_cnt == COL-1)                         //一行结束置零
            h_cnt <= 12'd0;
        else
            h_cnt <= h_cnt + 1'b1;                      //计数
    end
    //场同步循环
    always @(posedge clk or negedge rst) begin
        if(!rst)
            v_cnt <= 12'd0;
        else if(v_cnt == ROW-1)
            v_cnt <= 12'd0;
        else if(h_cnt == COL-1)                         //计数，以h_cnt 计到
满的时候 v_cnt+1
            v_cnt <= v_cnt + 1'b1;
        else
            v_cnt <= v_cnt;
    end
    //帧计数，一秒一次循环
    always @(posedge clk) begin
        if(f_cnt == FRAM/4 - 1) begin                   //60 帧就是
1s  0.5s
            f_cnt <= 12'd0;
            clk1s <= ~clk1s;
        end
        else if(v_cnt == ROW - 1)                       //计数，以 v_cnt 计
到满的时候 f_cnt+1
            f_cnt <= f_cnt + 1'b1;
        else
            f_cnt <= f_cnt;
```

```
    end
    //输出场同步与行同步
    assign hs = (h_cnt < HS_SYNC)? 1'b0 : 1'b1;
    assign vs = (v_cnt < VS_SYNC)? 1'b0 : 1'b1;
    assign active =  (h_cnt >= (HS_SYNC + HS_BACK))  &&
//是否是图片显示有效时间
                     (h_cnt <= (HS_SYNC + HS_BACK + HS_ACTIVE))  &&
//行显示时序
                     (v_cnt >= (VS_SYNC + VS_BACK))  &&
                     (v_cnt <= (VS_SYNC + VS_BACK + VS_ACTIVE))  ;
//列显示时序
    assign x = (active) ? h_cnt - (HS_SYNC + HS_BACK):0;
//x，y 坐标
    assign y = (active) ? v_cnt - (VS_SYNC + VS_BACK):0;
endmodule
```

5.tetris_control

```
`timescale 1ns / 1ps
module tetris_control(
    input rst,
    input clk,
    input[3:0] button,
    output reg [199:0] tetris_whole,
    output reg [199:0] tetris_new,
    output reg game_over,
    output reg [31:0] score
);
reg over;
reg [15:0]randn;
reg [3:0] button_last;
```

```
//记录 6 种方块初始值
parameter TTS0_4 = 20'b0000_0000_0000_0000_0011 , TTS0_5 = 20'b0000_000
0_0000_0000_0001, TTS0_6 = 20'b0000_0000_0000_0000_0001;
parameter TTS1_4 = 20'b0000_0000_0000_0000_0000 , TTS1_5 = 20'b0000_000
0_0000_0000_0011, TTS1_6 = 20'b0000_0000_0000_0000_0011;
parameter TTS2_4 = 20'b0000_0000_0000_0000_0000 , TTS2_5 = 20'b0000_000
0_0000_0000_1111, TTS2_6 = 20'b0000_0000_0000_0000_0000;
parameter TTS3_4 = 20'b0000_0000_0000_0000_0000 , TTS3_5 = 20'b0000_000
0_0000_0000_0100, TTS3_6 = 20'b0000_0000_0000_0000_0111;
parameter TTS4_4 = 20'b0000_0000_0000_0000_0001 , TTS4_5 = 20'b0000_000
0_0000_0000_0011, TTS4_6 = 20'b0000_0000_0000_0000_0001;
parameter TTS5_4 = 20'b0000_0000_0000_0000_0000 , TTS5_5 = 20'b0000_000
0_0000_0000_0011, TTS5_6 = 20'b0000_0000_0000_0000_0110;
initial begin
    over <= 2'b1;
    randn <= 0;
    game_over <= 0;
    tetris_whole <= 200'b0;
    tetris_new <= 200'b0;
    score <= 32'b0;
    button_last <= 4'b0000;
end
reg [199:0] tmp;
reg [4:0]i,j;
always @ (posedge clk) begin
    if(button[3] && !button_last[3] && !over && !tetris_new[19:0]) begi
n                                       //按键左移
        tmp[19:0] = tetris_new[1 * 20 + 19:1 * 20];
        tmp[1 * 20 + 19:1 * 20] = tetris_new[2 * 20 + 19:2 * 20];
        tmp[2 * 20 + 19:2 * 20] = tetris_new[3 * 20 + 19:3 * 20];
```

```
        tmp[3 * 20 + 19:3 * 20] = tetris_new[4 * 20 + 19:4 * 20];
        tmp[4 * 20 + 19:4 * 20] = tetris_new[5 * 20 + 19:5 * 20];
        tmp[5 * 20 + 19:5 * 20] = tetris_new[6 * 20 + 19:6 * 20];
        tmp[6 * 20 + 19:6 * 20] = tetris_new[7 * 20 + 19:7 * 20];
        tmp[7 * 20 + 19:7 * 20] = tetris_new[8 * 20 + 19:8 * 20];
        tmp[8 * 20 + 19:8 * 20] = tetris_new[9 * 20 + 19:9 * 20];
        tmp[8 * 20 + 19:8 * 20] = tetris_new[9 * 20 + 19:9 * 20];
        tmp[9 * 20 + 19:9 * 20] = 20'b0;
        if(tmp && !(tmp & tetris_whole))
            tetris_new = tmp;
    end
    else if(button[2] && !button_last[2] && !over && !tetris_new[9 * 20
+ 19:9 * 20]) begin                    //按键右移
        tmp[19:0] = 0;
        tmp[1 * 20 + 19:1 * 20] = tetris_new[0 * 20 + 19:0 * 20];
        tmp[2 * 20 + 19:2 * 20] = tetris_new[1 * 20 + 19:1 * 20];
        tmp[3 * 20 + 19:3 * 20] = tetris_new[2 * 20 + 19:2 * 20];
        tmp[4 * 20 + 19:4 * 20] = tetris_new[3 * 20 + 19:3 * 20];
        tmp[5 * 20 + 19:5 * 20] = tetris_new[4 * 20 + 19:4 * 20];
        tmp[6 * 20 + 19:6 * 20] = tetris_new[5 * 20 + 19:5 * 20];
        tmp[7 * 20 + 19:7 * 20] = tetris_new[6 * 20 + 19:6 * 20];
        tmp[8 * 20 + 19:8 * 20] = tetris_new[7 * 20 + 19:7 * 20];
        tmp[9 * 20 + 19:9 * 20] = tetris_new[8 * 20 + 19:8 * 20];
        if(tmp && !(tmp & tetris_whole))
            tetris_new = tmp;
    end
    else if(button[1] && !button_last[1] && !over) begin
                                    //按键旋转
        i = 0;
```

```
        while((i < 10) && (!tetris_new[i*20] & !tetris_new[i*20+1] & !t
etris_new[i*20+2] & !tetris_new[i*20+3] & !tetris_new[i*20+4] & !tetris
_new[i*20+5] & !tetris_new[i*20+6] & !tetris_new[i*20+7] & !tetris_new[
i*20+8] & !tetris_new[i*20+9]
            & !tetris_new[i*20+10] & !tetris_new[i*20+11] & !tetris_new
[i*20+12] & !tetris_new[i*20+13] & !tetris_new[i*20+14] & !tetris_new[i
*20+15] & !tetris_new[i*20+16] & !tetris_new[i*20+17] & !tetris_new[i*2
0+18] & !tetris_new[i*20+19]))
            i = i + 1;
        j = 0;
        while((j < 20) && (!tetris_new[j] & !tetris_new[j+20] & !tetris
_new[j+20*2] & !tetris_new[j+20*3] & !tetris_new[j+20*4] & !tetris_new[
j+20*5] & !tetris_new[j+20*6] & !tetris_new[j+20*7] & !tetris_new[j+20*
8] & !tetris_new[j+20*9]))
            j = j + 1;
        //i j  i*20+j
        case ((j + 3<=19 && tetris_new[i*20+j+3])||(i + 3<=9 && tetris_
new[(i+3)*20+j]))
            1'b0: begin
                tmp = 0;
                if(j + 2 <= 19 && i + 2 <= 9 && (tetris_new[i * 20 + j
+ 2] | tetris_new[(i + 1) * 20 + j + 2] | tetris_new[(i + 2) * 20 + j +
 2] | tetris_new[(i + 2) * 20 + j] | tetris_new[(i + 2) * 20 + j + 1]))
 begin//3*3
                    tmp[i * 20 + j] = tetris_new[i * 20 + j + 2];//0 0
 0 2
                    tmp[i * 20 + j + 1] = tetris_new[(i + 1) * 20 + j +
 2];//0 1  1 2
                    tmp[i * 20 + j + 2] = tetris_new[(i + 2) * 20 + j +
 2];//0 2  2 2
```

```
                    tmp[(i + 1) * 20 + j] = tetris_new[i * 20 + j + 1];
//1 0  0 1
                    tmp[(i + 1) * 20 + j + 1] = tetris_new[(i + 1) * 20
 + j + 1];//1 1  1 1
                    tmp[(i + 1) * 20 + j + 2] = tetris_new[(i + 2) * 20
 + j + 1];//1 2  2 1
                    tmp[(i + 2) * 20 + j] = tetris_new[i * 20 + j];//2
0  0 0
                    tmp[(i + 2) * 20 + j + 1] = tetris_new[(i + 1) * 20
 + j];//2 1  1 0
                    tmp[(i + 2) * 20 + j + 2] = tetris_new[(i + 2) * 20
 + j];//2 2  2 0
                end
            end
            1'b1: begin//四块长条的
                tmp = 0;
                if(j + 3 <= 19 && i + 3 <= 9) begin
                    tmp[i * 20 + j] = tetris_new[i * 20 + j + 3];//0 0
 0 3
                    tmp[i * 20 + j + 1] = tetris_new[(i + 1) * 20 + j +
 3];//0 1  1 3
                    tmp[i * 20 + j + 2] = tetris_new[(i + 2) * 20 + j +
 3];//0 2  2 3
                    tmp[i * 20 + j + 3] = tetris_new[(i + 3) * 20 + j +
 3];//0 3  3 3
                    tmp[(i + 1) * 20 + j] = tetris_new[i * 20 + j + 2];
//1 0  0 2
                    tmp[(i + 1) * 20 + j + 1] = tetris_new[(i + 1) * 20
 + j + 2];//1 1  1 2
```

```
                    tmp[(i + 1) * 20 + j + 2] = tetris_new[(i + 2) * 20
 + j + 2];//1 2  2 2
                    tmp[(i + 1) * 20 + j + 3] = tetris_new[(i + 3) * 20
 + j + 2];//1 3  3 2
                    tmp[(i + 2) * 20 + j] = tetris_new[i * 20 + j + 1];
//2 0  0 1
                    tmp[(i + 2) * 20 + j + 1] = tetris_new[(i + 1) * 20
 + j + 1];//2 1  1 1
                    tmp[(i + 2) * 20 + j + 2] = tetris_new[(i + 2) * 20
 + j + 1];//2 2  2 1
                    tmp[(i + 2) * 20 + j + 3] = tetris_new[(i + 3) * 20
 + j + 1];//2 3  3 1
                    tmp[(i + 3) * 20 + j] = tetris_new[i * 20 + j];//3
0  0 0
                    tmp[(i + 3) * 20 + j + 1] = tetris_new[(i + 1) * 20
 + j];//3 1  1 0
                    tmp[(i + 3) * 20 + j + 2] = tetris_new[(i + 2) * 20
 + j];//3 2  2 0
                    tmp[(i + 3) * 20 + j + 3] = tetris_new[(i + 3) * 20
 + j];//3 3  3 0
                end
            end
        endcase
        if(tmp && !(tmp & tetris_whole))
            tetris_new = tmp;
    end
    else if(rst && button[0] && !over) begin
                     //下落
        tmp[0 * 20 + 19:0 * 20] = tetris_new[0 * 20 + 19:0 * 20] << 1;
        tmp[1 * 20 + 19:1 * 20] = tetris_new[1 * 20 + 19:1 * 20] << 1;
```

```
        tmp[2 * 20 + 19:2 * 20] = tetris_new[2 * 20 + 19:2 * 20] << 1;
        tmp[3 * 20 + 19:3 * 20] = tetris_new[3 * 20 + 19:3 * 20] << 1;
        tmp[4 * 20 + 19:4 * 20] = tetris_new[4 * 20 + 19:4 * 20] << 1;
        tmp[5 * 20 + 19:5 * 20] = tetris_new[5 * 20 + 19:5 * 20] << 1;
        tmp[6 * 20 + 19:6 * 20] = tetris_new[6 * 20 + 19:6 * 20] << 1;
        tmp[7 * 20 + 19:7 * 20] = tetris_new[7 * 20 + 19:7 * 20] << 1;
        tmp[8 * 20 + 19:8 * 20] = tetris_new[8 * 20 + 19:8 * 20] << 1;
        tmp[9 * 20 + 19:9 * 20] = tetris_new[9 * 20 + 19:9 * 20] << 1;
        if(tmp && !(tmp & tetris_whole)
        && !((tmp[0 * 20 + 19:0 * 20] << 1) & tetris_whole[0 * 20 + 19:
0 * 20]) && !tmp[0 * 20 + 18]
        && !((tmp[1 * 20 + 19:1 * 20] << 1) & tetris_whole[1 * 20 + 19:
1 * 20]) && !tmp[1 * 20 + 18]
        && !((tmp[2 * 20 + 19:2 * 20] << 1) & tetris_whole[2 * 20 + 19:
2 * 20]) && !tmp[2 * 20 + 19]
        && !((tmp[3 * 20 + 19:3 * 20] << 1) & tetris_whole[3 * 20 + 19:
3 * 20]) && !tmp[3 * 20 + 19]
        && !((tmp[4 * 20 + 19:4 * 20] << 1) & tetris_whole[4 * 20 + 19:
4 * 20]) && !tmp[4 * 20 + 19]
        && !((tmp[5 * 20 + 19:5 * 20] << 1) & tetris_whole[5 * 20 + 19:
5 * 20]) && !tmp[5 * 20 + 19]
        && !((tmp[6 * 20 + 19:6 * 20] << 1) & tetris_whole[6 * 20 + 19:
6 * 20]) && !tmp[6 * 20 + 19]
        && !((tmp[7 * 20 + 19:7 * 20] << 1) & tetris_whole[7 * 20 + 19:
7 * 20]) && !tmp[7 * 20 + 19]
        && !((tmp[8 * 20 + 19:8 * 20] << 1) & tetris_whole[8 * 20 + 19:
8 * 20]) && !tmp[8 * 20 + 19]
        && !((tmp[9 * 20 + 19:9 * 20] << 1) & tetris_whole[9 * 20 + 19:
9 * 20]) && !tmp[9 * 20 + 19])
            tetris_new = tmp;
```

```
    end
    if(!rst) begin//低电平重置
        tetris_whole<=200'b0;
        tetris_new<=200'b0;
        over <= 2'b1;
        game_over <= 0;
        tetris_whole <= 200'b0;
        tetris_new <= 200'b0;
        score <= 32'b0;
    end
    else if(game_over) begin//游戏结束
        tetris_whole<=200'b0;
        tetris_new<=200'b0;
        score <= 32'b0;
    end//game_over end 游戏结束

    else begin//游戏没有结束
        if(over) begin                              //如果上一块已经落下出现新方块
            //把 tetris_new 和 tetris_whole 合并
            tetris_whole = tetris_whole | tetris_new;
            tetris_new = 200'b0;
            //消除
            if(tetris_whole[0 * 20 + 19] & tetris_whole[1 * 20 + 19] & tetris_whole[2 * 20 + 19] & tetris_whole[3 * 20 + 19] &
            tetris_whole[4 * 20 + 19] & tetris_whole[5 * 20 + 19] & tetris_whole[6 * 20 + 19] & tetris_whole[7 * 20 + 19] &
            tetris_whole[8 * 20 + 19] & tetris_whole[9 * 20 + 19]) begin
```

```
                tetris_whole[0 * 20 + 19 : 0 * 20] = tetris_whole[0 * 2
0 + 19:0 * 20] << 1;
                tetris_whole[1 * 20 + 19 : 1 * 20] = tetris_whole[1 * 2
0 + 19:1 * 20] << 1;
                tetris_whole[2 * 20 + 19 : 2 * 20] = tetris_whole[2 * 2
0 + 19:2 * 20] << 1;
                tetris_whole[3 * 20 + 19 : 3 * 20] = tetris_whole[3 * 2
0 + 19:3 * 20] << 1;
                tetris_whole[4 * 20 + 19 : 4 * 20] = tetris_whole[4 * 2
0 + 19:4 * 20] << 1;



                tetris_whole[5 * 20 + 19 : 5 * 20] = tetris_whole[5 * 2
0 + 19:5 * 20] << 1;
                tetris_whole[6 * 20 + 19 : 6 * 20] = tetris_whole[6 * 2
0 + 19:6 * 20] << 1;
                tetris_whole[7 * 20 + 19 : 7 * 20] = tetris_whole[7 * 2
0 + 19:7 * 20] << 1;
                tetris_whole[8 * 20 + 19 : 8 * 20] = tetris_whole[8 * 2
0 + 19:8 * 20] << 1;
                tetris_whole[9 * 20 + 19 : 9 * 20] = tetris_whole[9 * 2
0 + 19:9 * 20] << 1;
                score = score + 32'd10;
            end
            if(tetris_whole[0 * 20 + 18] & tetris_whole[1 * 20 + 18] &
tetris_whole[2 * 20 + 18] & tetris_whole[3 * 20 + 18] &
            tetris_whole[4 * 20 + 18] & tetris_whole[5 * 20 + 18] & tet
ris_whole[6 * 20 + 18] & tetris_whole[7 * 20 + 18] &
            tetris_whole[8 * 20 + 18] & tetris_whole[9 * 20 + 18]) begi
n
```

```
                tetris_whole[0 * 20 + 18 : 0 * 20] = tetris_whole[0 * 2
0 + 18:0 * 20] << 1;
                tetris_whole[1 * 20 + 18 : 1 * 20] = tetris_whole[1 * 2
0 + 18:1 * 20] << 1;
                tetris_whole[2 * 20 + 18 : 2 * 20] = tetris_whole[2 * 2
0 + 18:2 * 20] << 1;
                tetris_whole[3 * 20 + 18 : 3 * 20] = tetris_whole[3 * 2
0 + 18:3 * 20] << 1;
                tetris_whole[4 * 20 + 18 : 4 * 20] = tetris_whole[4 * 2
0 + 18:4 * 20] << 1;
                tetris_whole[5 * 20 + 18 : 5 * 20] = tetris_whole[5 * 2
0 + 18:5 * 20] << 1;
                tetris_whole[6 * 20 + 18 : 6 * 20] = tetris_whole[6 * 2
0 + 18:6 * 20] << 1;
                tetris_whole[7 * 20 + 18 : 7 * 20] = tetris_whole[7 * 2
0 + 18:7 * 20] << 1;
                tetris_whole[8 * 20 + 18 : 8 * 20] = tetris_whole[8 * 2
0 + 18:8 * 20] << 1;
                tetris_whole[9 * 20 + 18 : 9 * 20] = tetris_whole[9 * 2
0 + 18:9 * 20] << 1;
                score = score + 32'd10;
            end
            if(tetris_whole[0 * 20 + 17] & tetris_whole[1 * 20 + 17] &
tetris_whole[2 * 20 + 17] & tetris_whole[3 * 20 + 17] &
            tetris_whole[4 * 20 + 17] & tetris_whole[5 * 20 + 17] & tet
ris_whole[6 * 20 + 17] & tetris_whole[7 * 20 + 17] &
            tetris_whole[8 * 20 + 17] & tetris_whole[9 * 20 + 17]) begi
n
                tetris_whole[0 * 20 + 17 : 0 * 20] = tetris_whole[0 * 2
0 + 17:0 * 20] << 1;
```

```
                tetris_whole[1 * 20 + 17 : 1 * 20] = tetris_whole[1 * 2
0 + 17:1 * 20] << 1;
                tetris_whole[2 * 20 + 17 : 2 * 20] = tetris_whole[2 * 2
0 + 17:2 * 20] << 1;
                tetris_whole[3 * 20 + 17 : 3 * 20] = tetris_whole[3 * 2
0 + 17:3 * 20] << 1;
                tetris_whole[4 * 20 + 17 : 4 * 20] = tetris_whole[4 * 2
0 + 17:4 * 20] << 1;
                tetris_whole[5 * 20 + 17 : 5 * 20] = tetris_whole[5 * 2
0 + 17:5 * 20] << 1;
                tetris_whole[6 * 20 + 17 : 6 * 20] = tetris_whole[6 * 2
0 + 17:6 * 20] << 1;
                tetris_whole[7 * 20 + 17 : 7 * 20] = tetris_whole[7 * 2
0 + 17:7 * 20] << 1;
                tetris_whole[8 * 20 + 17 : 8 * 20] = tetris_whole[8 * 2
0 + 17:8 * 20] << 1;
                tetris_whole[9 * 20 + 17 : 9 * 20] = tetris_whole[9 * 2
0 + 17:9 * 20] << 1;
                score = score + 32'd10;
            end
            if(tetris_whole[0 * 20 + 16] & tetris_whole[1 * 20 + 16] &
tetris_whole[2 * 20 + 16] & tetris_whole[3 * 20 + 16] &
            tetris_whole[4 * 20 + 16] & tetris_whole[5 * 20 + 16] & tet
ris_whole[6 * 20 + 16] & tetris_whole[7 * 20 + 16] &
            tetris_whole[8 * 20 + 16] & tetris_whole[9 * 20 + 16]) begi
n
                tetris_whole[0 * 20 + 16 : 0 * 20] = tetris_whole[0 * 2
0 + 16:0 * 20] << 1;
                tetris_whole[1 * 20 + 16 : 1 * 20] = tetris_whole[1 * 2
0 + 16:1 * 20] << 1;
```

```
                tetris_whole[2 * 20 + 16 : 2 * 20] = tetris_whole[2 * 2
0 + 16:2 * 20] << 1;
                tetris_whole[3 * 20 + 16 : 3 * 20] = tetris_whole[3 * 2
0 + 16:3 * 20] << 1;
                tetris_whole[4 * 20 + 16 : 4 * 20] = tetris_whole[4 * 2
0 + 16:4 * 20] << 1;
                tetris_whole[5 * 20 + 16 : 5 * 20] = tetris_whole[5 * 2
0 + 16:5 * 20] << 1;
                tetris_whole[6 * 20 + 16 : 6 * 20] = tetris_whole[6 * 2
0 + 16:6 * 20] << 1;
                tetris_whole[7 * 20 + 16 : 7 * 20] = tetris_whole[7 * 2
0 + 16:7 * 20] << 1;
                tetris_whole[8 * 20 + 16 : 8 * 20] = tetris_whole[8 * 2
0 + 16:8 * 20] << 1;
                tetris_whole[9 * 20 + 16 : 9 * 20] = tetris_whole[9 * 2
0 + 16:9 * 20] << 1;
                score = score + 32'd10;
            end
            if(tetris_whole[0 * 20 + 15] & tetris_whole[1 * 20 + 15] &
tetris_whole[2 * 20 + 15] & tetris_whole[3 * 20 + 15] &
            tetris_whole[4 * 20 + 15] & tetris_whole[5 * 20 + 15] & tet
ris_whole[6 * 20 + 15] & tetris_whole[7 * 20 + 15] &
            tetris_whole[8 * 20 + 15] & tetris_whole[9 * 20 + 15]) begi
n
                tetris_whole[0 * 20 + 15 : 0 * 20] = tetris_whole[0 * 2
0 + 15:0 * 20] << 1;
                tetris_whole[1 * 20 + 15 : 1 * 20] = tetris_whole[1 * 2
0 + 15:1 * 20] << 1;
                tetris_whole[2 * 20 + 15 : 2 * 20] = tetris_whole[2 * 2
0 + 15:2 * 20] << 1;
```

```
                tetris_whole[3 * 20 + 15 : 3 * 20] = tetris_whole[3 * 2
0 + 15:3 * 20] << 1;
                tetris_whole[4 * 20 + 15 : 4 * 20] = tetris_whole[4 * 2
0 + 15:4 * 20] << 1;
                tetris_whole[5 * 20 + 15 : 5 * 20] = tetris_whole[5 * 2
0 + 15:5 * 20] << 1;
                tetris_whole[6 * 20 + 15 : 6 * 20] = tetris_whole[6 * 2
0 + 15:6 * 20] << 1;
                tetris_whole[7 * 20 + 15 : 7 * 20] = tetris_whole[7 * 2
0 + 15:7 * 20] << 1;
                tetris_whole[8 * 20 + 15 : 8 * 20] = tetris_whole[8 * 2
0 + 15:8 * 20] << 1;
                tetris_whole[9 * 20 + 15 : 9 * 20] = tetris_whole[9 * 2
0 + 15:9 * 20] << 1;
                score = score + 32'd10;
            end
            if(tetris_whole[0 * 20 + 14] & tetris_whole[1 * 20 + 14] &
tetris_whole[2 * 20 + 14] & tetris_whole[3 * 20 + 14] &
            tetris_whole[4 * 20 + 14] & tetris_whole[5 * 20 + 14] & tet
ris_whole[6 * 20 + 14] & tetris_whole[7 * 20 + 14] &
            tetris_whole[8 * 20 + 14] & tetris_whole[9 * 20 + 14]) begi
n
                tetris_whole[0 * 20 + 14 : 0 * 20] = tetris_whole[0 * 2
0 + 14:0 * 20] << 1;
                tetris_whole[1 * 20 + 14 : 1 * 20] = tetris_whole[1 * 2
0 + 14:1 * 20] << 1;
                tetris_whole[2 * 20 + 14 : 2 * 20] = tetris_whole[2 * 2
0 + 14:2 * 20] << 1;
                tetris_whole[3 * 20 + 14 : 3 * 20] = tetris_whole[3 * 2
0 + 14:3 * 20] << 1;
```

```
                tetris_whole[4 * 20 + 14 : 4 * 20] = tetris_whole[4 * 2
0 + 14:4 * 20] << 1;
                tetris_whole[5 * 20 + 14 : 5 * 20] = tetris_whole[5 * 2
0 + 14:5 * 20] << 1;
                tetris_whole[6 * 20 + 14 : 6 * 20] = tetris_whole[6 * 2
0 + 14:6 * 20] << 1;
                tetris_whole[7 * 20 + 14 : 7 * 20] = tetris_whole[7 * 2
0 + 14:7 * 20] << 1;
                tetris_whole[8 * 20 + 14 : 8 * 20] = tetris_whole[8 * 2
0 + 14:8 * 20] << 1;
                tetris_whole[9 * 20 + 14 : 9 * 20] = tetris_whole[9 * 2
0 + 14:9 * 20] << 1;
                score = score + 32'd10;
            end
            if(tetris_whole[0 * 20 + 13] & tetris_whole[1 * 20 + 13] &
tetris_whole[2 * 20 + 13] & tetris_whole[3 * 20 + 13] &
            tetris_whole[4 * 20 + 13] & tetris_whole[5 * 20 + 13] & tet
ris_whole[6 * 20 + 13] & tetris_whole[7 * 20 + 13] &
            tetris_whole[8 * 20 + 13] & tetris_whole[9 * 20 + 13]) begi
n
                tetris_whole[0 * 20 + 13 : 0 * 20] = tetris_whole[0 * 2
0 + 13:0 * 20] << 1;
                tetris_whole[1 * 20 + 13 : 1 * 20] = tetris_whole[1 * 2
0 + 13:1 * 20] << 1;
                tetris_whole[2 * 20 + 13 : 2 * 20] = tetris_whole[2 * 2
0 + 13:2 * 20] << 1;
                tetris_whole[3 * 20 + 13 : 3 * 20] = tetris_whole[3 * 2
0 + 13:3 * 20] << 1;
                tetris_whole[4 * 20 + 13 : 4 * 20] = tetris_whole[4 * 2
0 + 13:4 * 20] << 1;
```

```
                tetris_whole[5 * 20 + 13 : 5 * 20] = tetris_whole[5 * 2
0 + 13:5 * 20] << 1;
                tetris_whole[6 * 20 + 13 : 6 * 20] = tetris_whole[6 * 2
0 + 13:6 * 20] << 1;
                tetris_whole[7 * 20 + 13 : 7 * 20] = tetris_whole[7 * 2
0 + 13:7 * 20] << 1;
                tetris_whole[8 * 20 + 13 : 8 * 20] = tetris_whole[8 * 2
0 + 13:8 * 20] << 1;
                tetris_whole[9 * 20 + 13 : 9 * 20] = tetris_whole[9 * 2
0 + 13:9 * 20] << 1;
                score = score + 32'd10;
            end
            if(tetris_whole[0 * 20 + 12] & tetris_whole[1 * 20 + 12] &
tetris_whole[2 * 20 + 12] & tetris_whole[3 * 20 + 12] &
            tetris_whole[4 * 20 + 12] & tetris_whole[5 * 20 + 12] & tet
ris_whole[6 * 20 + 12] & tetris_whole[7 * 20 + 12] &
            tetris_whole[8 * 20 + 12] & tetris_whole[9 * 20 + 12]) begi
n
                tetris_whole[0 * 20 + 12 : 0 * 20] = tetris_whole[0 * 2
0 + 12:0 * 20] << 1;
                tetris_whole[1 * 20 + 12 : 1 * 20] = tetris_whole[1 * 2
0 + 12:1 * 20] << 1;
                tetris_whole[2 * 20 + 12 : 2 * 20] = tetris_whole[2 * 2
0 + 12:2 * 20] << 1;
                tetris_whole[3 * 20 + 12 : 3 * 20] = tetris_whole[3 * 2
0 + 12:3 * 20] << 1;
                tetris_whole[4 * 20 + 12 : 4 * 20] = tetris_whole[4 * 2
0 + 12:4 * 20] << 1;
                tetris_whole[5 * 20 + 12 : 5 * 20] = tetris_whole[5 * 2
0 + 12:5 * 20] << 1;
```

```
                tetris_whole[6 * 20 + 12 : 6 * 20] = tetris_whole[6 * 2
0 + 12:6 * 20] << 1;
                tetris_whole[7 * 20 + 12 : 7 * 20] = tetris_whole[7 * 2
0 + 12:7 * 20] << 1;
                tetris_whole[8 * 20 + 12 : 8 * 20] = tetris_whole[8 * 2
0 + 12:8 * 20] << 1;
                tetris_whole[9 * 20 + 12 : 9 * 20] = tetris_whole[9 * 2
0 + 12:9 * 20] << 1;
                score = score + 32'd10;
            end
            if(tetris_whole[0 * 20 + 11] & tetris_whole[1 * 20 + 11] &
tetris_whole[2 * 20 + 11] & tetris_whole[3 * 20 + 11] &
            tetris_whole[4 * 20 + 11] & tetris_whole[5 * 20 + 11] & tet
ris_whole[6 * 20 + 11] & tetris_whole[7 * 20 + 11] &
            tetris_whole[8 * 20 + 11] & tetris_whole[9 * 20 + 11]) begi
n
                tetris_whole[0 * 20 + 11 : 0 * 20] = tetris_whole[0 * 2
0 + 11:0 * 20] << 1;
                tetris_whole[1 * 20 + 11 : 1 * 20] = tetris_whole[1 * 2
0 + 11:1 * 20] << 1;
                tetris_whole[2 * 20 + 11 : 2 * 20] = tetris_whole[2 * 2
0 + 11:2 * 20] << 1;
                tetris_whole[3 * 20 + 11 : 3 * 20] = tetris_whole[3 * 2
0 + 11:3 * 20] << 1;
                tetris_whole[4 * 20 + 11 : 4 * 20] = tetris_whole[4 * 2
0 + 11:4 * 20] << 1;
                tetris_whole[5 * 20 + 11 : 5 * 20] = tetris_whole[5 * 2
0 + 11:5 * 20] << 1;
                tetris_whole[6 * 20 + 11 : 6 * 20] = tetris_whole[6 * 2
0 + 11:6 * 20] << 1;
```

```
                tetris_whole[7 * 20 + 11 : 7 * 20] = tetris_whole[7 * 2
0 + 11:7 * 20] << 1;
                tetris_whole[8 * 20 + 11 : 8 * 20] = tetris_whole[8 * 2
0 + 11:8 * 20] << 1;
                tetris_whole[9 * 20 + 11 : 9 * 20] = tetris_whole[9 * 2
0 + 11:9 * 20] << 1;
                score = score + 32'd10;
            end
            if (tetris_whole[0 * 20 + 10] & tetris_whole[1 * 20 + 10] &
 tetris_whole[2 * 20 + 10] & tetris_whole[3 * 20 + 10] &
                tetris_whole[4 * 20 + 10] & tetris_whole[5 * 20 + 10] &
 tetris_whole[6 * 20 + 10] & tetris_whole[7 * 20 + 10] &
                tetris_whole[8 * 20 + 10] & tetris_whole[9 * 20 + 10])
begin
                tetris_whole[0 * 20 + 10 : 0 * 20] = tetris_whole[0 * 2
0 + 10:0 * 20] << 1;
                tetris_whole[1 * 20 + 10 : 1 * 20] = tetris_whole[1 * 2
0 + 10:1 * 20] << 1;
                tetris_whole[2 * 20 + 10 : 2 * 20] = tetris_whole[2 * 2
0 + 10:2 * 20] << 1;
                tetris_whole[3 * 20 + 10 : 3 * 20] = tetris_whole[3 * 2
0 + 10:3 * 20] << 1;
                tetris_whole[4 * 20 + 10 : 4 * 20] = tetris_whole[4 * 2
0 + 10:4 * 20] << 1;
                tetris_whole[5 * 20 + 10 : 5 * 20] = tetris_whole[5 * 2
0 + 10:5 * 20] << 1;
                tetris_whole[6 * 20 + 10 : 6 * 20] = tetris_whole[6 * 2
0 + 10:6 * 20] << 1;
                tetris_whole[7 * 20 + 10 : 7 * 20] = tetris_whole[7 * 2
0 + 10:7 * 20] << 1;
```

```
                tetris_whole[8 * 20 + 10 : 8 * 20] = tetris_whole[8 * 2
0 + 10:8 * 20] << 1;
                tetris_whole[9 * 20 + 10 : 9 * 20] = tetris_whole[9 * 2
0 + 10:9 * 20] << 1;
                score = score + 32'd10;
            end
            if (tetris_whole[0 * 20 + 9] & tetris_whole[1 * 20 + 9] & t
etris_whole[2 * 20 + 9] & tetris_whole[3 * 20 + 9] &
                tetris_whole[4 * 20 + 9] & tetris_whole[5 * 20 + 9] & t
etris_whole[6 * 20 + 9] & tetris_whole[7 * 20 + 9] &
                tetris_whole[8 * 20 + 9] & tetris_whole[9 * 20 + 9]) be
gin
                tetris_whole[0 * 20 + 9 : 0 * 20] = tetris_whole[0 * 20
 + 9:0 * 20] << 1;
                tetris_whole[1 * 20 + 9 : 1 * 20] = tetris_whole[1 * 20
 + 9:1 * 20] << 1;
                tetris_whole[2 * 20 + 9 : 2 * 20] = tetris_whole[2 * 20
 + 9:2 * 20] << 1;
                tetris_whole[3 * 20 + 9 : 3 * 20] = tetris_whole[3 * 20
 + 9:3 * 20] << 1;
                tetris_whole[4 * 20 + 9 : 4 * 20] = tetris_whole[4 * 20
 + 9:4 * 20] << 1;
                tetris_whole[5 * 20 + 9 : 5 * 20] = tetris_whole[5 * 20
 + 9:5 * 20] << 1;
                tetris_whole[6 * 20 + 9 : 6 * 20] = tetris_whole[6 * 20
 + 9:6 * 20] << 1;
                tetris_whole[7 * 20 + 9 : 7 * 20] = tetris_whole[7 * 20
 + 9:7 * 20] << 1;
                tetris_whole[8 * 20 + 9 : 8 * 20] = tetris_whole[8 * 20
 + 9:8 * 20] << 1;
```

```
                tetris_whole[9 * 20 + 9 : 9 * 20] = tetris_whole[9 * 20
 + 9:9 * 20] << 1;
                score = score + 32'd10;
            end
            if (tetris_whole[0 * 20 + 8] & tetris_whole[1 * 20 + 8] & t
etris_whole[2 * 20 + 8] & tetris_whole[3 * 20 + 8] &
                tetris_whole[4 * 20 + 8] & tetris_whole[5 * 20 + 8] & t
etris_whole[6 * 20 + 8] & tetris_whole[7 * 20 + 8] &
                tetris_whole[8 * 20 + 8] & tetris_whole[9 * 20 + 8]) be
gin
                tetris_whole[0 * 20 + 8 : 0 * 20] = tetris_whole[0 * 20
 + 8:0 * 20] << 1;
                tetris_whole[1 * 20 + 8 : 1 * 20] = tetris_whole[1 * 20
 + 8:1 * 20] << 1;
                tetris_whole[2 * 20 + 8 : 2 * 20] = tetris_whole[2 * 20
 + 8:2 * 20] << 1;
                tetris_whole[3 * 20 + 8 : 3 * 20] = tetris_whole[3 * 20
 + 8:3 * 20] << 1;
                tetris_whole[4 * 20 + 8 : 4 * 20] = tetris_whole[4 * 20
 + 8:4 * 20] << 1;
                tetris_whole[5 * 20 + 8 : 5 * 20] = tetris_whole[5 * 20
 + 8:5 * 20] << 1;
                tetris_whole[6 * 20 + 8 : 6 * 20] = tetris_whole[6 * 20
 + 8:6 * 20] << 1;
                tetris_whole[7 * 20 + 8 : 7 * 20] = tetris_whole[7 * 20
 + 8:7 * 20] << 1;
                tetris_whole[8 * 20 + 8 : 8 * 20] = tetris_whole[8 * 20
 + 8:8 * 20] << 1;
                tetris_whole[9 * 20 + 8 : 9 * 20] = tetris_whole[9 * 20
 + 8:9 * 20] << 1;
```

```
                score = score + 32'd10;
            end
            if (tetris_whole[0 * 20 + 7] & tetris_whole[1 * 20 + 7] & t
etris_whole[2 * 20 + 7] & tetris_whole[3 * 20 + 7] &
                tetris_whole[4 * 20 + 7] & tetris_whole[5 * 20 + 7] & t
etris_whole[6 * 20 + 7] & tetris_whole[7 * 20 + 7] &
                tetris_whole[8 * 20 + 7] & tetris_whole[9 * 20 + 7]) be
gin
                tetris_whole[0 * 20 + 7 : 0 * 20] = tetris_whole[0 * 20
 + 7:0 * 20] << 1;
                tetris_whole[1 * 20 + 7 : 1 * 20] = tetris_whole[1 * 20
 + 7:1 * 20] << 1;
                tetris_whole[2 * 20 + 7 : 2 * 20] = tetris_whole[2 * 20
 + 7:2 * 20] << 1;
                tetris_whole[3 * 20 + 7 : 3 * 20] = tetris_whole[3 * 20
 + 7:3 * 20] << 1;
                tetris_whole[4 * 20 + 7 : 4 * 20] = tetris_whole[4 * 20
 + 7:4 * 20] << 1;
                tetris_whole[5 * 20 + 7 : 5 * 20] = tetris_whole[5 * 20
 + 7:5 * 20] << 1;
                tetris_whole[6 * 20 + 7 : 6 * 20] = tetris_whole[6 * 20
 + 7:6 * 20] << 1;
                tetris_whole[7 * 20 + 7 : 7 * 20] = tetris_whole[7 * 20
 + 7:7 * 20] << 1;
                tetris_whole[8 * 20 + 7 : 8 * 20] = tetris_whole[8 * 20
 + 7:8 * 20] << 1;
                tetris_whole[9 * 20 + 7 : 9 * 20] = tetris_whole[9 * 20
 + 7:9 * 20] << 1;
                score = score + 32'd10;
            end
```

```
            if (tetris_whole[0 * 20 + 6] & tetris_whole[1 * 20 + 6] & t
etris_whole[2 * 20 + 6] & tetris_whole[3 * 20 + 6] &
                tetris_whole[4 * 20 + 6] & tetris_whole[5 * 20 + 6] & t
etris_whole[6 * 20 + 6] & tetris_whole[7 * 20 + 6] &
                tetris_whole[8 * 20 + 6] & tetris_whole[9 * 20 + 6]) be
gin
                tetris_whole[0 * 20 + 6 : 0 * 20] = tetris_whole[0 * 20
 + 6:0 * 20] << 1;
                tetris_whole[1 * 20 + 6 : 1 * 20] = tetris_whole[1 * 20
 + 6:1 * 20] << 1;
                tetris_whole[2 * 20 + 6 : 2 * 20] = tetris_whole[2 * 20
 + 6:2 * 20] << 1;
                tetris_whole[3 * 20 + 6 : 3 * 20] = tetris_whole[3 * 20
 + 6:3 * 20] << 1;
                tetris_whole[4 * 20 + 6 : 4 * 20] = tetris_whole[4 * 20
 + 6:4 * 20] << 1;
                tetris_whole[5 * 20 + 6 : 5 * 20] = tetris_whole[5 * 20
 + 6:5 * 20] << 1;
                tetris_whole[6 * 20 + 6 : 6 * 20] = tetris_whole[6 * 20
 + 6:6 * 20] << 1;
                tetris_whole[7 * 20 + 6 : 7 * 20] = tetris_whole[7 * 20
 + 6:7 * 20] << 1;
                tetris_whole[8 * 20 + 6 : 8 * 20] = tetris_whole[8 * 20
 + 6:8 * 20] << 1;
                tetris_whole[9 * 20 + 6 : 9 * 20] = tetris_whole[9 * 20
 + 6:9 * 20] << 1;
                score = score + 32'd10;
            end
            if (tetris_whole[0 * 20 + 5] & tetris_whole[1 * 20 + 5] & t
etris_whole[2 * 20 + 5] & tetris_whole[3 * 20 + 5] &
```

```
                tetris_whole[4 * 20 + 5] & tetris_whole[5 * 20 + 5] & t
etris_whole[6 * 20 + 5] & tetris_whole[7 * 20 + 5] &
                tetris_whole[8 * 20 + 5] & tetris_whole[9 * 20 + 5]) be
gin
                tetris_whole[0 * 20 + 5 : 0 * 20] = tetris_whole[0 * 20
 + 5:0 * 20] << 1;
                tetris_whole[1 * 20 + 5 : 1 * 20] = tetris_whole[1 * 20
 + 5:1 * 20] << 1;
                tetris_whole[2 * 20 + 5 : 2 * 20] = tetris_whole[2 * 20
 + 5:2 * 20] << 1;
                tetris_whole[3 * 20 + 5 : 3 * 20] = tetris_whole[3 * 20
 + 5:3 * 20] << 1;
                tetris_whole[4 * 20 + 5 : 4 * 20] = tetris_whole[4 * 20
 + 5:4 * 20] << 1;
                tetris_whole[5 * 20 + 5 : 5 * 20] = tetris_whole[5 * 20
 + 5:5 * 20] << 1;
                tetris_whole[6 * 20 + 5 : 6 * 20] = tetris_whole[6 * 20
 + 5:6 * 20] << 1;
                tetris_whole[7 * 20 + 5 : 7 * 20] = tetris_whole[7 * 20
 + 5:7 * 20] << 1;
                tetris_whole[8 * 20 + 5 : 8 * 20] = tetris_whole[8 * 20
 + 5:8 * 20] << 1;
                tetris_whole[9 * 20 + 5 : 9 * 20] = tetris_whole[9 * 20
 + 5:9 * 20] << 1;
            end
            if (tetris_whole[0 * 20 + 4] & tetris_whole[1 * 20 + 4] & t
etris_whole[2 * 20 + 4] & tetris_whole[3 * 20 + 4] &
                tetris_whole[4 * 20 + 4] & tetris_whole[5 * 20 + 4] & t
etris_whole[6 * 20 + 4] & tetris_whole[7 * 20 + 4] &
```

```
                tetris_whole[8 * 20 + 4] & tetris_whole[9 * 20 + 4]) be
gin
                tetris_whole[0 * 20 + 4 : 0 * 20] = tetris_whole[0 * 20
 + 4:0 * 20] << 1;
                tetris_whole[1 * 20 + 4 : 1 * 20] = tetris_whole[1 * 20
 + 4:1 * 20] << 1;
                tetris_whole[2 * 20 + 4 : 2 * 20] = tetris_whole[2 * 20
 + 4:2 * 20] << 1;
                tetris_whole[3 * 20 + 4 : 3 * 20] = tetris_whole[3 * 20
 + 4:3 * 20] << 1;
                tetris_whole[4 * 20 + 4 : 4 * 20] = tetris_whole[4 * 20
 + 4:4 * 20] << 1;
                tetris_whole[5 * 20 + 4 : 5 * 20] = tetris_whole[5 * 20
 + 4:5 * 20] << 1;
                tetris_whole[6 * 20 + 4 : 6 * 20] = tetris_whole[6 * 20
 + 4:6 * 20] << 1;
                tetris_whole[7 * 20 + 4 : 7 * 20] = tetris_whole[7 * 20
 + 4:7 * 20] << 1;
                tetris_whole[8 * 20 + 4 : 8 * 20] = tetris_whole[8 * 20
 + 4:8 * 20] << 1;
                tetris_whole[9 * 20 + 4 : 9 * 20] = tetris_whole[9 * 20
 + 4:9 * 20] << 1;
            end
            //产生新方块
            case(randn % 6)
            3'd0:begin
                tetris_new[4 * 20 + 19:4 * 20] = TTS0_4;tetris_new[5 *
20 + 19:5 * 20] = TTS0_5;tetris_new[6 * 20 + 19:6 * 20] = TTS0_6;
            end
            3'd1:begin
```

```
                tetris_new[4 * 20 + 19:4 * 20] = TTS1_4;tetris_new[5 *
20 + 19:5 * 20] = TTS1_5;tetris_new[6 * 20 + 19:6 * 20] = TTS1_6;
            end
            3'd2:begin
                tetris_new[4 * 20 + 19:4 * 20] = TTS2_4;tetris_new[5 *
20 + 19:5 * 20] = TTS2_5;tetris_new[6 * 20 + 19:6 * 20] = TTS2_6;
            end
            3'd3:begin
                tetris_new[4 * 20 + 19:4 * 20] = TTS3_4;tetris_new[5 *
20 + 19:5 * 20] = TTS3_5;tetris_new[6 * 20 + 19:6 * 20] = TTS3_6;
            end
            3'd4:begin
                tetris_new[4 * 20 + 19:4 * 20] = TTS4_4;tetris_new[5 *
20 + 19:5 * 20] = TTS4_5;tetris_new[6 * 20 + 19:6 * 20] = TTS4_6;
            end
            3'd5:begin
                tetris_new[4 * 20 + 19:4 * 20] = TTS5_4;tetris_new[5 *
20 + 19:5 * 20] = TTS5_5;tetris_new[6 * 20 + 19:6 * 20] = TTS5_6;
            end
            endcase
            //判断有没有死亡
            if(tetris_new & tetris_whole)
                game_over = 1'b1;
            over <= 1'b0;
            randn <= (randn + 1) * 11;
        end
        else begin                                  //如果上一块还没有落下
            //上一块落下
            tetris_new[0 * 20 + 19:0 * 20] <= tetris_new[0 * 20 + 19:0
* 20] << 1;
```

```
            tetris_new[1 * 20 + 19:1 * 20] <= tetris_new[1 * 20 + 19:1
* 20] << 1;
            tetris_new[2 * 20 + 19:2 * 20] <= tetris_new[2 * 20 + 19:2
* 20] << 1;
            tetris_new[3 * 20 + 19:3 * 20] <= tetris_new[3 * 20 + 19:3
* 20] << 1;
            tetris_new[4 * 20 + 19:4 * 20] <= tetris_new[4 * 20 + 19:4
* 20] << 1;
            tetris_new[5 * 20 + 19:5 * 20] <= tetris_new[5 * 20 + 19:5
* 20] << 1;
            tetris_new[6 * 20 + 19:6 * 20] <= tetris_new[6 * 20 + 19:6
* 20] << 1;
            tetris_new[7 * 20 + 19:7 * 20] <= tetris_new[7 * 20 + 19:7
* 20] << 1;
            tetris_new[8 * 20 + 19:8 * 20] <= tetris_new[8 * 20 + 19:8
* 20] << 1;
            tetris_new[9 * 20 + 19:9 * 20] <= tetris_new[9 * 20 + 19:9
* 20] << 1;
            //判断是否到底了
            if(((tetris_new[0 * 20 + 19:0 * 20] << 2) & tetris_whole[0
* 20 + 19:0 * 20]) || tetris_new[0 * 20 + 18])
                over <= 1'b1;
            else if(((tetris_new[1 * 20 + 19:1 * 20] << 2) & tetris_who
le[1 * 20 + 19:1 * 20]) || tetris_new[1 * 20 + 18])
                over <= 1'b1;
            else if(((tetris_new[2 * 20 + 19:2 * 20] << 2) & tetris_who
le[2 * 20 + 19:2 * 20]) || tetris_new[2 * 20 + 18])
                over <= 1'b1;
            else if(((tetris_new[3 * 20 + 19:3 * 20] << 2) & tetris_who
le[3 * 20 + 19:3 * 20]) || tetris_new[3 * 20 + 18])
```

```
                over <= 1'b1;
            else if(((tetris_new[4 * 20 + 19:4 * 20] << 2) & tetris_who
le[4 * 20 + 19:4 * 20]) || tetris_new[4 * 20 + 18])
                over <= 1'b1;
            else if(((tetris_new[5 * 20 + 19:5 * 20] << 2) & tetris_who
le[5 * 20 + 19:5 * 20]) || tetris_new[5 * 20 + 18])
                over <= 1'b1;
            else if(((tetris_new[6 * 20 + 19:6 * 20] << 2) & tetris_who
le[6 * 20 + 19:6 * 20]) || tetris_new[6 * 20 + 18])
                over <= 1'b1;
            else if(((tetris_new[7 * 20 + 19:7 * 20] << 2) & tetris_who
le[7 * 20 + 19:7 * 20]) || tetris_new[7 * 20 + 18])
                over <= 1'b1;
            else if(((tetris_new[8 * 20 + 19:8 * 20] << 2) & tetris_who
le[8 * 20 + 19:8 * 20]) || tetris_new[8 * 20 + 18])
                over <= 1'b1;
            else if(((tetris_new[9 * 20 + 19:9 * 20] << 2) & tetris_who
le[9 * 20 + 19:9 * 20]) || tetris_new[9 * 20 + 18]|| !tetris_new)
                over <= 1'b1;
        end
    end //游戏没有结束 end
end
endmodule
```

6.keyboard

```
`timescale 1ns / 1ps
module ps2_keyboard_tb;
    // 输入信号
    reg clk;
    reg rst;
```

```
reg ps2_clk;
reg ps2_data;
// 输出信号
wire [3:0] button;
// 实例化被测试模块
ps2_keyboard uut (
    .clk(clk),
    .rst(rst),
    .ps2_clk(ps2_clk),
    .ps2_data(ps2_data),
    .button(button)
);
// 时钟生成
always #10 clk = ~clk; // 50MHz 时钟周期为 20ns
// 模拟 PS2 键盘信号
task send_ps2_data;
    input [7:0] data;
    integer i;
    begin
        // 起始位
        ps2_data = 0;
        #20000; // 20us
        ps2_clk = 0;
        #5000; // 5us
        ps2_clk = 1;
        #5000; // 5us
        // 数据位
        for (i = 0; i < 8; i = i + 1) begin
            ps2_data = data[i];
            #20000; // 20us
```

```
            ps2_clk = 0;
            #5000; // 5us
            ps2_clk = 1;
            #5000; // 5us
        end
        // 奇偶校验位
        ps2_data = ~^data;
        #20000; // 20us
        ps2_clk = 0;
        #5000; // 5us
        ps2_clk = 1;
        #5000; // 5us
        // 停止位
        ps2_data = 1;
        #20000; // 20us
        ps2_clk = 0;
        #5000; // 5us
        ps2_clk = 1;
        #5000; // 5us
    end
endtask
initial begin
    // 初始化输入信号
    clk = 0;
    rst = 0;
    ps2_clk = 1;
    ps2_data = 1;
    // 复位
    #100;
    rst = 1;
```

```
        #100;
        rst = 0;
        #100;
        rst = 1;
        // 模拟按键按下和松开
        #200;
        send_ps2_data(8'h6B); // 按下左箭头键
        #200;
        send_ps2_data(8'hF0); // 断码
        send_ps2_data(8'h6B); // 松开左箭头键
        #200;
        send_ps2_data(8'h74); // 按下右箭头键
        #200;
        send_ps2_data(8'hF0); // 断码
        send_ps2_data(8'h74); // 松开右箭头键
        #200;
        send_ps2_data(8'h75); // 按下上箭头键（旋转）
        #200;
        send_ps2_data(8'hF0); // 断码
        send_ps2_data(8'h75); // 松开上箭头键
        #200;
        send_ps2_data(8'h72); // 按下下箭头键（加速向下）
        #200;
        send_ps2_data(8'hF0); // 断码
        send_ps2_data(8'h72); // 松开下箭头键
        // 结束仿真
        #1000;
        $stop;
    end
endmodule
```

7.clk_wiz_0

```
// file: clk_wiz_0.v
//
// (c) Copyright 2008 - 2013 Xilinx, Inc. All rights reserved.
//
// This file contains confidential and proprietary information
// of Xilinx, Inc. and is protected under U.S. and
// international copyright and other intellectual property
// laws.
//
// DISCLAIMER
// This disclaimer is not a license and does not grant any
// rights to the materials distributed herewith. Except as
// otherwise provided in a valid license issued to you by
// Xilinx, and to the maximum extent permitted by applicable
// law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
// WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
// AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
// BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
// INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
// (2) Xilinx shall not be liable (whether in contract or tort,
// including negligence, or under any other theory of
// liability) for any loss or damage of any kind or nature
// related to, arising under or in connection with these
// materials, including for any direct, or any indirect,
// special, incidental, or consequential loss or damage
// (including loss of data, profits, goodwill, or any type of
// loss or damage suffered as a result of any action brought
// by a third party) even if such damage or loss was
```

```
// reasonably foreseeable or Xilinx had been advised of the
// possibility of the same.
//
// CRITICAL APPLICATIONS
// Xilinx products are not designed or intended to be fail-
// safe, or for use in any application requiring fail-safe
// performance, such as life-support or safety devices or
// systems, Class III medical devices, nuclear facilities,
// applications related to the deployment of airbags, or any
// other applications that could lead to death, personal
// injury, or severe property or environmental damage
// (individually and collectively, "Critical
// Applications"). Customer assumes the sole risk and
// liability of any use of Xilinx products in Critical
// Applications, subject only to applicable laws and
// regulations governing limitations on product liability.
//
// THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
// PART OF THIS FILE AT ALL TIMES.
//
//------------------------------------------------------------------
-------
// User entered comments
//------------------------------------------------------------------
-------
// None
//
//------------------------------------------------------------------
-------
//  Output     Output      Phase    Duty Cycle   Pk-to-Pk     Phase
```

```
//   Clock     Freq (MHz)  (degrees)    (%)     Jitter (ps)  Error (ps)
//------------------------------------------------------------------------
-------
// clk_out1____65.000______0.000______50.0______254.866____297.890
//
//------------------------------------------------------------------------
-------
// Input Clock   Freq (MHz)    Input Jitter (UI)
//------------------------------------------------------------------------
-------
// __primary_________100.000____________0.010
`timescale 1ps/1ps
(* CORE_GENERATION_INFO = "clk_wiz_0,clk_wiz_v5_3_1,{component_name=clk
_wiz_0,use_phase_alignment=true,use_min_o_jitter=false,use_max_i_jitter
=false,use_dyn_phase_shift=false,use_inclk_switchover=false,use_dyn_rec
onfig=false,enable_axi=0,feedback_source=FDBK_AUTO,PRIMITIVE=MMCM,num_o
ut_clk=1,clkin1_period=10.0,clkin2_period=10.0,use_power_down=false,use
_reset=true,use_locked=true,use_inclk_stopped=false,feedback_type=SINGL
E,CLOCK_MGR_TYPE=NA,manual_override=false}" *)
module clk_wiz_0
 (
 // Clock in ports
  input         clk_in1,
  // Clock out ports
  output        clk_out1,
  // Status and control signals
  input         reset,
  output        locked
 );
  clk_wiz_0_clk_wiz inst
```

```
  (
 // Clock in ports
  .clk_in1(clk_in1),
  // Clock out ports
  .clk_out1(clk_out1),
  // Status and control signals
  .reset(reset),
  .locked(locked)
  );
Endmodule
```

8.VGA_tb

```
`timescale 1ns / 1ps
module VGA_tb;
    // 输入信号
    reg game_start;
    reg clk;
    reg rst;
    reg [3:0] button;
    // 输出信号
    wire [3:0] color_r;
    wire [3:0] color_g;
    wire [3:0] color_b;
    wire hs;
    wire vs;
    wire game_over;
    wire [32:0] score;
    // 实例化被测试模块
    VGA uut (
```

```
        .game_start(game_start),
        .clk(clk),
        .rst(rst),
        .button(button),
        .color_r(color_r),
        .color_g(color_g),
        .color_b(color_b),
        .hs(hs),
        .vs(vs),
        .game_over(game_over),
        .score(score)
    );
    // 时钟生成
    always #20 clk = ~clk; // 25MHz 时钟周期为 40ns
    initial begin
        // 初始化输入信号
        game_start = 0;
        clk = 0;
        rst = 0;
        button = 4'b0000;
        // 复位
        #100;
        rst = 1;
        #100;
        rst = 0;
        #100;
        rst = 1;
        // 开始游戏
        #100;
        game_start = 1;
```

```
        // 模拟按键按下
        #200;
        button = 4'b1000; // 按下左箭头键
        #200;
        button = 4'b0000; // 松开按键
        #200;
        button = 4'b0100; // 按下右箭头键
        #200;
        button = 4'b0000; // 松开按键
        #200;
        button = 4'b0010; // 按下上箭头键（旋转）
        #200;
        button = 4'b0000; // 松开按键
        #200;
        button = 4'b0001; // 按下下箭头键（加速向下）
        #200;
        button = 4'b0000; // 松开按键
        // 结束仿真
        #1000;
        $stop;
    end
endmodule
```

9.keyboard_tb

```
`timescale 1ns / 1ps
module VGA_tb;
    // 输入信号
    reg game_start;
    reg clk;
```

```
reg rst;
reg [3:0] button;
// 输出信号
wire [3:0] color_r;
wire [3:0] color_g;
wire [3:0] color_b;
wire hs;
wire vs;
wire game_over;
wire [32:0] score;
// 实例化被测试模块
VGA uut (
    .game_start(game_start),
    .clk(clk),
    .rst(rst),
    .button(button),
    .color_r(color_r),
    .color_g(color_g),
    .color_b(color_b),
    .hs(hs),
    .vs(vs),
    .game_over(game_over),
    .score(score)
);
// 时钟生成
always #20 clk = ~clk; // 25MHz 时钟周期为 40ns
initial begin
    // 初始化输入信号
    game_start = 0;
    clk = 0;
```

```
        rst = 0;
        button = 4'b0000;
        // 复位
        #100;
        rst = 1;
        #100;
        rst = 0;
        #100;
        rst = 1;
        // 开始游戏
        #100;
        game_start = 1;
        // 模拟按键按下
        #200;
        button = 4'b1000; // 按下左箭头键
        #200;
        button = 4'b0000; // 松开按键
        #200;
        button = 4'b0100; // 按下右箭头键
        #200;
        button = 4'b0000; // 松开按键
        #200;
        button = 4'b0010; // 按下上箭头键（旋转）
        #200;
        button = 4'b0000; // 松开按键
        #200;
        button = 4'b0001; // 按下下箭头键（加速向下）
        #200;
        button = 4'b0000; // 松开按键
        // 结束仿真
```

```
        #1000;
        $stop;
    end
endmodule
```