【实验目的】

熟练掌握前面实验中的所有知识点

熟悉几种常用通信接口的工作原理及使用

独立完成具有一定规模的功能电路设计

【实验环境】

PC 一台

Windows 操作系统

Vivado FPGA 实验平台(Nexys4 DDR)

Logisim

vlab.ustc.edu.cn

PS/2 接口键盘

VGA 显示屏

【设计思路】

本次综合实验是《数字逻辑电路实验》课程的最后一次实验，我选择的课题是五子棋游戏（GoBang Game），设计基础是本课程中学习的 Verilog 硬件描述语言和 Nexys 4 开发板，在此之上又使用了两个外接设备：PS/2 接口的键盘以及 VGA 接口的显示屏，用于优化输入与输出。

实验中有一顶层模块：gobang_frame,正如其名，它起到建立起整个游戏框架的作用，将 PS/2 输出模块、游戏逻辑模块、数据存储模

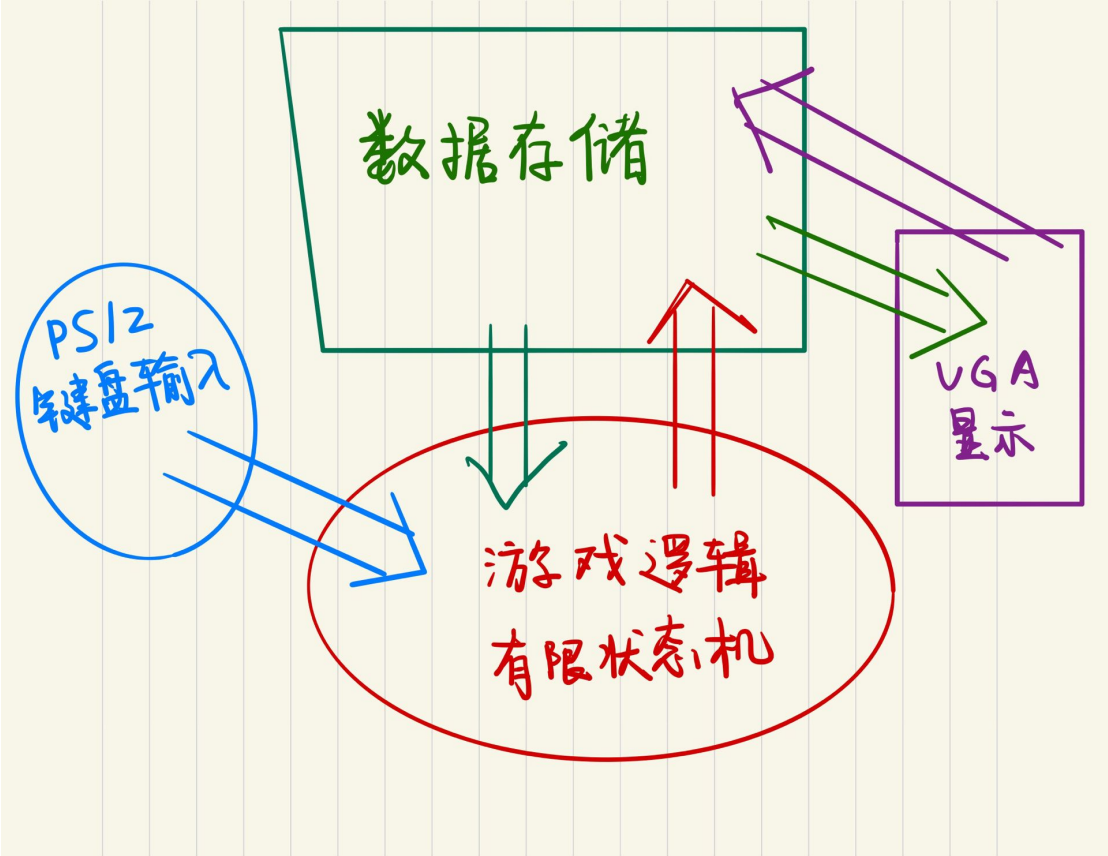
块、VGA 显示模块整个到一起，各个模块之间的数据信息交换都通过 frame 模块来完成，frame 模块的输入、输出通过约束文件与开发板上的管脚相关联。

gobang_frame 模块代码截图如下

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 中科大
// Engineer: 李昱邦
//
// Create Date: 2019/12/09 08:45:06
// Design Name: 五子棋
// Module Name: game_frame
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module game_frame(
    //板载时钟clk信号及复位rst信号
    input wire clk,
    input wire rst,

    //PS-2输入
    input wire PS2_CLK,
    input wire PS2_DATA,

    //输出至VGA显示
    output wire VGA_HS,
    output wire VGA_VS,
    output wire [3:0] VGA_G,
    output wire [3:0] VGA_B

    );
    //产生25.125Mhz的时钟信号，供vga显示使用
    wire clk25Mhz;
    clk_wiz_0  dividor1(.clk_in1(clk),.clk_out1(clk25Mhz));


    //用于游戏逻辑部分的输出（wire）型
    wire cursor_i, cursor_j;
    wire data_clr, data_write;
    wire crt_player, game_running;
    wire [1:0] winner;

    //用于数据通路部分的输出（wire）型
    wire [8:0] black_i, black_j, black_ij, black_ji,
               white_i, white_j, white_ij, white_ji;
    /*代表了相应行、列、对角线的黑、白棋存储信息*/
    wire [14:0] logic_row, display_black, display_white;


    //用于存储PS-2输入的数据（作为PS-2_input模块的输出）
    wire key_up, key_down, key_left, key_right, key_ok;

    wire [3:0] display_i;
    wire [16:0] clk_div;
    wire [3:0] last_i,last_j;
    gobang_logic
        game_logic(
            .clk_slow(clk_div[14]),
            .clk_fast(clk),

            .rst(rst),
            .key_up(key_up),
            .key_down(key_down),
            .key_left(key_left),
            .key_right(key_right),
            .key_ok(key_ok),
            .black_i(black_i),
            .black_j(black_j),
            .black_ij(black_ij),
            .black_ji(black_ji),
            .white_i(white_i),
            .white_j(white_j),
            .white_ij(white_ij),
            .white_ji(white_ji),
            .chess_row(logic_row),
            .data_clr(data_clr),
            .data_write(data_write),
            .cursor_i(cursor_i),
            .cursor_j(cursor_j),
            .crt_player(crt_player),
            .game_running(game_running),
            .winner(winner)
        );

    gobang_data
        data(
            .clk(clk_div[14]),
            .rst(rst),
            .clr(data_clr),
            .write(data_write),
            .write_j(cursor_j),
            .write_color(crt_player),
            .display_i(display_i),
            .logic_row(logic_row),
            .display_black(display_black),
            .display_white(display_white),
            .black_i(black_i),
            .black_j(black_j),
            .black_ij(black_ij),
            .black_ji(black_ji),
            .white_i(white_i),
            .white_j(white_j),
            .white_ij(white_ij)
        );

    ps2_input
        key_input(
            .clk_slow(clk_div[14]),
            .clk_fast(clk),
            .rst(rst),
            .PS2_CLK(PS2_CLK),
            .PS2_DATA(PS2_DATA),
            .key_up(key_up),
            .key_down(key_down),
            .key_left(key_left),
            .key_right(key_right),
            .key_ok(key_ok)
        );

        vga_display
        display(
            .clk(clk25Mhz),
```

```
            .key_ok(key_ok)
        );

        vga_display
        display(
            .clk(clk25Mhz),
            .rst(rst),
            .cursor_i(cursor_i),
            .cursor_j(cursor_j),
            .crt_player(crt_player),
            .game_running(game_running),
            .winner(winner),
            .display_black(display_black),
            .display_white(display_white),
            .display_i(display_i),
            .sync_h(VGA_HS),
            .sync_v(VGA_VS),
            .r(VGA_R),
            .g(VGA_G),
            .b(VGA_B)
        );

    clk_divider
        divider(
            .clk(clk),
            .rst(rst),
            .clk_div(clk_div)
        );

endmodule
```

各个模块之间的关系如下图所示：



ps2_input 模块将接收到的键盘信号：上、下、左、右移动以及空格五个信号传给逻辑模块 gobang_logic；gobang_logic 模块将要

放置棋子的位置、颜色等信息传递给存储模块 gobang_data；gobang_data 模块将要下棋位置附近的棋子放置情况发送给逻辑模块，以便逻辑模块进行该位置能否下棋、下完棋后是否获胜等判断，data 模块还将棋盘相关信息（哪里下了黑白子、光标在哪里等）传递给 VGA 显示模块，以便将图像展示在显示屏上；vga_display 模块接收 gobang_data 模块传来的信息，结合自己产生的各种时序信号对显示屏进行扫描，显示图像。

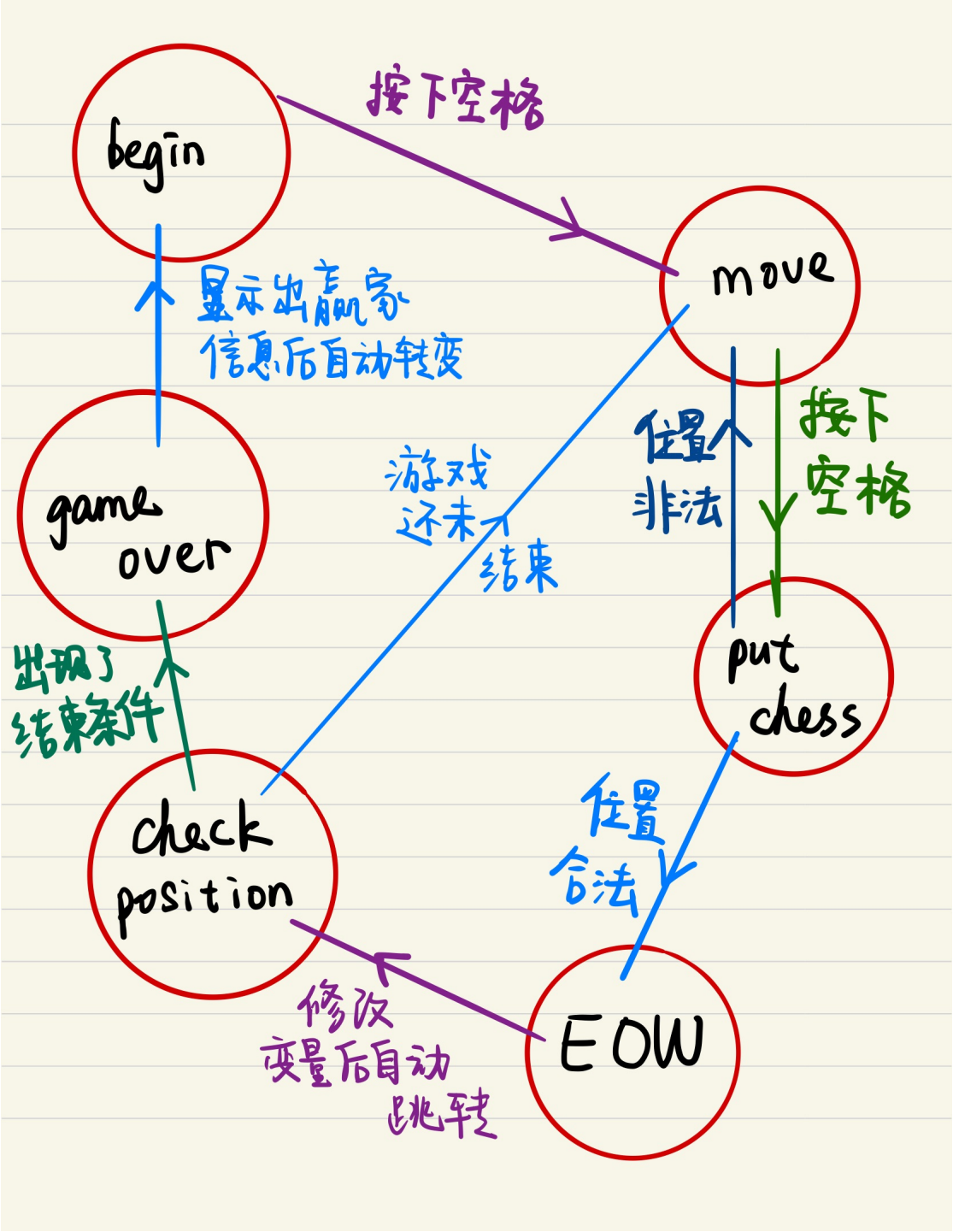
【模块详解】

（具体解释以注释的形式写在代码旁边，概要性的解释写在外部）

1.游戏逻辑模块：

画出有限状态机如下：

注：其中 EOW 状态是“End of Write”的缩写，由于每次放置棋子(put_chess)后，向 gobang_data 模块写入数据的激活信号往往未能及时修改，所以又额外设计了一个状态来完成这项工作



源代码如下：

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
//////
//游戏逻辑部分
//主要实现了有限状态机
```

```
//////////////////////////////////////////////////////////////////////////////////
//////

module gobang_logic(
    input wire clk_slow,
    input wire clk_fast,
    input wire rst,

    //键盘控制选择以及移动
    input wire key_up,                  // 按键向上移动
    input wire key_down,                // 按键向下移动
    input wire key_left,                // 按键向左移动
    input wire key_right,               // 按键向右移动
    input wire key_ok,                  // 按下回车键

    //以当前位置为中心的9 行、9 列
    input wire [8:0] black_i,
    input wire [8:0] black_j,

    //当前位置所在的两条9 位对角线
    input wire [8:0] black_ij,
    input wire [8:0] black_ji,

    //同理
    input wire [8:0] white_i,
    input wire [8:0] white_j,
    input wire [8:0] white_ij,
    input wire [8:0] white_ji,

    //当前光标所在行的棋子信息
    input wire [14:0] chess_row,

    // 清空数据通路
    output reg data_clr,
    //下棋
    output reg data_write,

    // 光标所在的行、列
    output reg [3:0] cursor_i,
    output reg [3:0] cursor_j,

    // 当前玩家
    output reg crt_player,
    //游戏是否正在进行中
```

```
    output reg game_running,
    //赢家
    output reg [1:0] winner
    );

    // 0 表示黑，1 表示白
    localparam BLACK = 1'b0,
               WHITE = 1'b1;

    // 标准五子棋盘大小为 15*15
    localparam BOARD_SIZE = 15;

    // 六种状态
    localparam state_begin = 3'b000,
               state_move = 3'b001,
               state_put_chess = 3'b010,
               state_EOW = 3'b011,
               state_check_position = 3'b100,
               state_gameover = 3'b101;

    reg [2:0] state;          // 当前状态，三位表示
    reg [7:0] move_count;     // 记录已经走了多少步

    // 判断是否某一方获胜
    reg win_clr, win_active;
    wire [3:0] win_i, win_j;
    wire is_win;
    win_checker
        checker(
            .clk(clk_fast),
            .rst(rst),
            .clr(win_clr),
            .active(win_active),
            .black_i(black_i),
            .black_j(black_j),
            .black_ij(black_ij),
            .black_ji(black_ji),
            .white_i(white_i),
            .white_j(white_j),
            .white_ij(white_ij),
            .white_ji(white_ji),
            .is_win(is_win)
        );
```

```
// 有限状态机的状态跳转：
always @(posedge clk_slow or posedge rst)
begin
    if (rst) //复位
    begin
        cursor_i <= BOARD_SIZE;
        cursor_j <= BOARD_SIZE;
        crt_player <= BLACK;
        game_running <= 1'b0;
        winner <= 2'b00;
        state <= state_begin;
        move_count <= 8'b0;
        data_clr <= 1'b0;
        data_write <= 1'b0;
        win_clr <= 1'b0;
        win_active <= 1'b0;
    end
    else
    begin
        case (state)
        state_begin:
            if (key_ok) // 按下空格键开始游戏
            begin
            //游戏开始时进行各项初始化
                cursor_i <= BOARD_SIZE/2;
                cursor_j <= BOARD_SIZE/2;
                crt_player <= BLACK;
                game_running <= 1'b1;
                winner <= 2'b00;
                move_count <= 8'b0;
                data_clr <= 1'b1;
                win_clr <= 1'b1;
                state <= state_move;
            end
            else
                state <= state_begin;    //若一直没有按下选择按钮，则保持该状态

        state_move:
        begin
            data_clr <= 1'b0;
            win_clr <= 1'b0;
            begin
               //玩家控制移动光标
```

```
                    if (key_up && cursor_i > 0)             //注意行坐标是低地址在上
                        cursor_i <= cursor_i - 1'b1;
                    else if (key_down && cursor_i < BOARD_SIZE - 1)
                        cursor_i <= cursor_i + 1'b1;
                    else if (key_left && cursor_j > 0)
                        cursor_j <= cursor_j - 1'b1;
                    else if (key_right && cursor_j < BOARD_SIZE - 1)
                        cursor_j <= cursor_j + 1'b1;
                    else if (key_ok)         // 按下了“确定”键（空格键）
                        state <= state_put_chess;
                    else
                    begin
                        cursor_i <= cursor_i;
                        cursor_j <= cursor_j;
                    end
                end
            end

            state_put_chess:
                // 检查该位置是否已被放置上棋子
                if (!chess_row[cursor_j])   //chess_row 是来源于数据模块的输入，判断该位置是否已经下过棋子
                                            //
                begin
                    move_count <= move_count + 8'b1;
                    data_write <= 1'b1;
                    state <= state_EOW;

                end
                else
                //若已被放置，重新回到之前的移动光标状态
                    state <= state_move;

            state_EOW:
            begin
                data_write <= 1'b0;
                win_active <= 1'b1;
                state <= state_check_position;
            end

            state_check_position:
            begin
                // 检查是否有玩家已经赢得比赛，或者全屏幕已被占满（平局）
```

```
                win_active <= 1'b0;
                if (is_win || move_count == BOARD_SIZE * BOARD_SIZE)
                    state <= state_gameover;
                else
                begin
                    crt_player <= ~crt_player;//玩家转换
                    state <= state_move;
                end
            end

            state_gameover:
            begin
                if (is_win)
                    // 有一方获胜
                    if(crt_player == 1'b0)
                        winner <= 2'b01;
                    else
                        winner <= 2'b10;
                else
                    // 平局
                    winner <= 2'b11;
                state <= state_begin;       //回到初始状态
                game_running <= 1'b0;
            end
            endcase
        end
    end
endmodule
```

其中用到的胜利判断模块：

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
//检查是否有一方获胜
//////////////////////////////////////////////////////////////////////////////////
//////
module win_checker(
    input wire clk,
    input wire rst,
    input wire clr,
    // 激活信号
    input wire active,
```

```
    //行、列、对角线信息
    input wire [8:0] black_i,
    input wire [8:0] black_j,
    input wire [8:0] black_ij,
    input wire [8:0] black_ji,
    input wire [8:0] white_i,
    input wire [8:0] white_j,
    input wire [8:0] white_ij,
    input wire [8:0] white_ji,

    output reg is_win
    );
    // 尺寸信息
    localparam BOARD_SIZE = 15;
    // 状态常量
    localparam STATE_BEGIN = 1'b0,
               STATE_WORKING = 1'b1;
    reg state;    // 当前状态
    // 检查是否已经连成 5 子
    wire b0, b1, b2, b3, w0, w1, w2, w3;
    pattern_five pattern_b0(black_i, b0),
                 pattern_b1(black_j, b1),
                 pattern_b2(black_ij, b2),
                 pattern_b3(black_ji, b3),
                 pattern_w0(white_i, w0),
                 pattern_w1(white_j, w1),
                 pattern_w2(white_ij, w2),
                 pattern_w3(white_ji, w3);
    //每当激活信号来临，checker 会对棋盘进行一次检查
    always @(posedge clk or posedge rst)
    begin
        if (rst || clr)
         begin
            is_win <= 0;
            state <= STATE_BEGIN;
        end
        else if ( !active && state == STATE_BEGIN)    //未收到激活信号且处于开始
状态
            state <= STATE_WORKING;
        else if (active && state == STATE_WORKING)
                is_win <= b0 | b1 | b2 | b3 | w0 | w1 | w2 | w3;
     end
endmodule
```

pattern_five 模块用于判断这 9 个在一条线上的棋子是否连成了五子，他的实现方式很简单：

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
//判断是否连成了五子
//////////////////////////////////////////////////////////////////////////////////
//////
module pattern_five(
    input wire [8:0] my,
    output reg ret
    );
    always @(*)
        if ((my[0] && my[1] && my[2] && my[3] && my[4]) ||
            (my[1] && my[2] && my[3] && my[4] && my[5]) ||
            (my[2] && my[3] && my[4] && my[5] && my[6]) ||
            (my[3] && my[4] && my[5] && my[6] && my[7]) ||
            (my[4] && my[5] && my[6] && my[7] && my[8]))
            ret = 1'b1;
        else
            ret = 1'b0;
endmodule
```

2. 数据存储模块：

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
//记录当前棋盘的数据
//为方便逻辑部分输入与 VGA 部分输出都能顺利完成，输入部分采用时序逻辑，输出部分采用组合逻辑
assign
//同时将最近写入位置附近棋盘的有关信息输出，作为判断是否连成五子的条件
//////////////////////////////////////////////////////////////////////////////////
//////

module gobang_data(
    input wire clk,
    input wire rst,
    input wire clr,      //当逻辑部分按下空格或复位后，clr 为 1，清空棋盘上的数据
    input wire write,    //当前玩家确定的光标位置没有棋子时，该信号为 1，控制将该位置标记
为有棋
```

```
    //待下棋的位置，棋盘为15*15，坐标为4位*4位即可
    input wire [3:0] write_i,
    input wire [3:0] write_j,
    //下黑棋或者白棋
    input wire write_color,

    input wire [3:0] display_i,
    output wire [14:0] logic_row,
    output wire [14:0] display_black,
    output wire [14:0] display_white,

    //输出当前放置棋子的位置的上、下、左、右9行9列、2组对角线数据
    //用于检查是否已经有一方拼出5子
    output reg [8:0] black_i,
    output reg [8:0] black_j,
    output reg [8:0] black_ij,
    output reg [8:0] black_ji,
    output reg [8:0] white_i,
    output reg [8:0] white_j,
    output reg [8:0] white_ij,
    output reg [8:0] white_ji
    );

    // 1黑 0白
    localparam BLACK = 1'b0,
               WHITE = 1'b1;
    //棋盘共15格 （15*15）
    localparam BOARD_SIZE = 15;
    //  存储黑白棋的信息
    //这两个寄存器变量都存储了15*15的信息，即整个棋盘上黑棋、白棋的位置
    reg [14:0] board_black [14:0];
    reg [14:0] board_white [14:0];
    //记录最近写入的棋子的坐标，以便于判断是否一方获胜
    reg [3:0] last_i;
    reg [3:0] last_j;

    integer i, j;
    reg [14:0] row_4b, row_3b, row_2b, row_1b, row0b,
               row1b, row2b, row3b, row4b;
    reg [14:0] row_4w, row_3w, row_2w, row_1w, row0w,
               row1w, row2w, row3w, row4w;

    //输出采用组合逻辑
```

```
    assign logic_row = board_black[write_i] | board_white[write_i]; //只要有黑棋子或白棋子，这里就被标记为有棋子

    //该位置是否显示黑白由
    assign display_black = board_black[display_i];
    assign display_white = board_white[display_i];

    always @(negedge clk or posedge rst)
    begin
        if (rst || clr) //复位时或清零时 （清零即state_begin状态按下空格时）
        begin
            board_black[0] <= 15'b0;
            board_black[1] <= 15'b0;
            board_black[2] <= 15'b0;
            board_black[3] <= 15'b0;
            board_black[4] <= 15'b0;
            board_black[5] <= 15'b0;
            board_black[6] <= 15'b0;
            board_black[7] <= 15'b0;
            board_black[8] <= 15'b0;
            board_black[9] <= 15'b0;
            board_black[10] <= 15'b0;
            board_black[11] <= 15'b0;
            board_black[12] <= 15'b0;
            board_black[13] <= 15'b0;
            board_black[14] <= 15'b0;

            board_white[0] <= 15'b0;
            board_white[1] <= 15'b0;
            board_white[2] <= 15'b0;
            board_white[3] <= 15'b0;
            board_white[4] <= 15'b0;
            board_white[5] <= 15'b0;
            board_white[6] <= 15'b0;
            board_white[7] <= 15'b0;
            board_white[8] <= 15'b0;
            board_white[9] <= 15'b0;
            board_white[10] <= 15'b0;
            board_white[11] <= 15'b0;
            board_white[12] <= 15'b0;
            board_white[13] <= 15'b0;
            board_white[14] <= 15'b0;
        end
        else if (write)
```

```
        begin
            last_i <= write_i;
            last_j <= write_j;
            if (write_color == BLACK)   //写入数据
                board_black[write_i] <= board_black[write_i] | (15'b1 <<
write_j);
            else
                board_white[write_i] <= board_white[write_i] | (15'b1 <<
write_j);
        end
    end


    always @(*)
    if(write == 0)
    begin
        i <= last_i;
        j <= last_j;
        //取第 i-4 到 i+4 行的数据
        if (i - 4 >= 0)
        begin
            row_4b = board_black[i - 4];
            row_4w = board_white[i - 4];
        end
        else
        begin
            row_4b = 15'b0;
            row_4w = 15'b0;
        end

        if (i - 3 >= 0)
        begin
            row_3b = board_black[i - 3];
            row_3w = board_white[i - 3];
        end
        else
        begin
            row_3b = 15'b0;
            row_3w = 15'b0;
        end

        if (i - 2 >= 0)
        begin
            row_2b = board_black[i - 2];
```

```
            row_2w = board_white[i - 2];
        end
        else
        begin
            row_2b = 15'b0;
            row_2w = 15'b0;
        end

        if (i - 1 >= 0)
        begin
            row_1b = board_black[i - 1];
            row_1w = board_white[i - 1];
        end
        else
        begin
            row_1b = 15'b0;
            row_1w = 15'b0;
        end

        if (i >= 0 && i < BOARD_SIZE)
        begin
            row0b = board_black[i];
            row0w = board_white[i];
        end
        else
        begin
            row0b = 15'b0;
            row0w = 15'b0;
        end

        if (i + 1 < BOARD_SIZE)
        begin
            row1b = board_black[i + 1];
            row1w = board_white[i + 1];
        end
        else
        begin
            row1b = 15'b0;
            row1w = 15'b0;
        end

        if (i + 2 < BOARD_SIZE)
        begin
            row2b = board_black[i + 2];
```

```
            row2w = board_white[i + 2];
        end
        else
        begin
            row2b = 15'b0;
            row2w = 15'b0;
        end

        if (i + 3 < BOARD_SIZE)
        begin
            row3b = board_black[i + 3];
            row3w = board_white[i + 3];
        end
        else
        begin
            row3b = 15'b0;
            row3w = 15'b0;
        end

        if (i + 4 < BOARD_SIZE)
        begin
            row4b = board_black[i + 4];
            row4w = board_white[i + 4];
        end
        else
        begin
            row4b = 15'b0;
            row4w = 15'b0;
        end

        // 更新最近下棋位置附近的信息
        if (j - 4 >= 0)
        begin
            black_i[0] = row0b[j - 4];
            white_i[0] = row0w[j - 4];
            black_ij[0] = row_4b[j - 4];
            white_ij[0] = row_4w[j - 4];
            black_ji[0] = row4b[j - 4];
            white_ji[0] = row4w[j - 4];
        end
        else
        begin
            black_i[0] = 1'b0;
            white_i[0] = 1'b0;
```

```
            black_ij[0] = 1'b0;
            white_ij[0] = 1'b0;
            black_ji[0] = 1'b0;
            white_ji[0] = 1'b0;
        end

        if (j - 3 >= 0)
        begin
            black_i[1] = row0b[j - 3];
            white_i[1] = row0w[j - 3];
            black_ij[1] = row_3b[j - 3];
            white_ij[1] = row_3w[j - 3];
            black_ji[1] = row3b[j - 3];
            white_ji[1] = row3w[j - 3];
        end
        else
        begin
            black_i[1] = 1'b0;
            white_i[1] = 1'b0;
            black_ij[1] = 1'b0;
            white_ij[1] = 1'b0;
            black_ji[1] = 1'b0;
            white_ji[1] = 1'b0;
        end

        if (j - 2 >= 0)
        begin
            black_i[2] = row0b[j - 2];
            white_i[2] = row0w[j - 2];
            black_ij[2] = row_2b[j - 2];
            white_ij[2] = row_2w[j - 2];
            black_ji[2] = row2b[j - 2];
            white_ji[2] = row2w[j - 2];
        end
        else
        begin
            black_i[2] = 1'b0;
            white_i[2] = 1'b0;
            black_ij[2] = 1'b0;
            white_ij[2] = 1'b0;
            black_ji[2] = 1'b0;
            white_ji[2] = 1'b0;
        end
```

```
        if (j - 1 >= 0)
        begin
            black_i[3] = row0b[j - 1];
            white_i[3] = row0w[j - 1];
            black_ij[3] = row_1b[j - 1];
            white_ij[3] = row_1w[j - 1];
            black_ji[3] = row1b[j - 1];
            white_ji[3] = row1w[j - 1];
        end
        else
        begin
            black_i[3] = 1'b0;
            white_i[3] = 1'b0;
            black_ij[3] = 1'b0;
            white_ij[3] = 1'b0;
            black_ji[3] = 1'b0;
            white_ji[3] = 1'b0;
        end

        if (j >= 0 && j < BOARD_SIZE)
        begin
            black_i[4] = row0b[j];
            white_i[4] = row0w[j];
            black_ij[4] = row0b[j];
            white_ij[4] = row0w[j];
            black_ji[4] = row0b[j];
            white_ji[4] = row0w[j];

            black_j[0] = row_4b[j];
            black_j[1] = row_3b[j];
            black_j[2] = row_2b[j];
            black_j[3] = row_1b[j];
            black_j[4] = row0b[j];
            black_j[5] = row1b[j];
            black_j[6] = row2b[j];
            black_j[7] = row3b[j];
            black_j[8] = row4b[j];
            white_j[0] = row_4w[j];
            white_j[1] = row_3w[j];
            white_j[2] = row_2w[j];
            white_j[3] = row_1w[j];
            white_j[4] = row0w[j];
            white_j[5] = row1w[j];
            white_j[6] = row2w[j];
```

```
            white_j[7] = row3w[j];
            white_j[8] = row4w[j];
        end
        else
        begin
            black_i[4] = 1'b0;
            white_i[4] = 1'b0;
            black_ij[4] = 1'b0;
            white_ij[4] = 1'b0;
            black_ji[4] = 1'b0;
            white_ji[4] = 1'b0;
            black_j[0] = 1'b0;
            black_j[1] = 1'b0;
            black_j[2] = 1'b0;
            black_j[3] = 1'b0;
            black_j[4] = 1'b0;
            black_j[5] = 1'b0;
            black_j[6] = 1'b0;
            black_j[7] = 1'b0;
            black_j[8] = 1'b0;
            white_j[0] = 1'b0;
            white_j[1] = 1'b0;
            white_j[2] = 1'b0;
            white_j[3] = 1'b0;
            white_j[4] = 1'b0;
            white_j[5] = 1'b0;
            white_j[6] = 1'b0;
            white_j[7] = 1'b0;
            white_j[8] = 1'b0;
        end

        if (j + 1 < BOARD_SIZE)
        begin
            black_i[5] = row0b[j + 1];
            white_i[5] = row0w[j + 1];
            black_ij[5] = row1b[j + 1];
            white_ij[5] = row1w[j + 1];
            black_ji[5] = row_1b[j + 1];
            white_ji[5] = row_1w[j + 1];
        end
        else
        begin
            black_i[5] = 1'b0;
            white_i[5] = 1'b0;
```

```
            black_ij[5] = 1'b0;
            white_ij[5] = 1'b0;
            black_ji[5] = 1'b0;
            white_ji[5] = 1'b0;
        end

        if (j + 2 < BOARD_SIZE)
        begin
            black_i[6] = row0b[j + 2];
            white_i[6] = row0w[j + 2];
            black_ij[6] = row2b[j + 2];
            white_ij[6] = row2w[j + 2];
            black_ji[6] = row_2b[j + 2];
            white_ji[6] = row_2w[j + 2];
        end
        else
        begin
            black_i[6] = 1'b0;
            white_i[6] = 1'b0;
            black_ij[6] = 1'b0;
            white_ij[6] = 1'b0;
            black_ji[6] = 1'b0;
            white_ji[6] = 1'b0;
        end

        if (j + 3 < BOARD_SIZE)
        begin
            black_i[7] = row0b[j + 3];
            white_i[7] = row0w[j + 3];
            black_ij[7] = row3b[j + 3];
            white_ij[7] = row3w[j + 3];
            black_ji[7] = row_3b[j + 3];
            white_ji[7] = row_3w[j + 3];
        end
        else
        begin
            black_i[7] = 1'b0;
            white_i[7] = 1'b0;
            black_ij[7] = 1'b0;
            white_ij[7] = 1'b0;
            black_ji[7] = 1'b0;
            white_ji[7] = 1'b0;
        end
        if (j + 4 < BOARD_SIZE)
```

```
        begin
            black_i[8] = row0b[j + 4];
            white_i[8] = row0w[j + 4];
            black_ij[8] = row4b[j + 4];
            white_ij[8] = row4w[j + 4];
            black_ji[8] = row_4b[j + 4];
            white_ji[8] = row_4w[j + 4];
        end
        else
        begin
            black_i[8] = 1'b0;
            white_i[8] = 1'b0;
            black_ij[8] = 1'b0;
            white_ij[8] = 1'b0;
            black_ji[8] = 1'b0;
            white_ji[8] = 1'b0;
        end
    end
endmodule
```

该模块需要向 gobang_logic 模块传送两个信息：

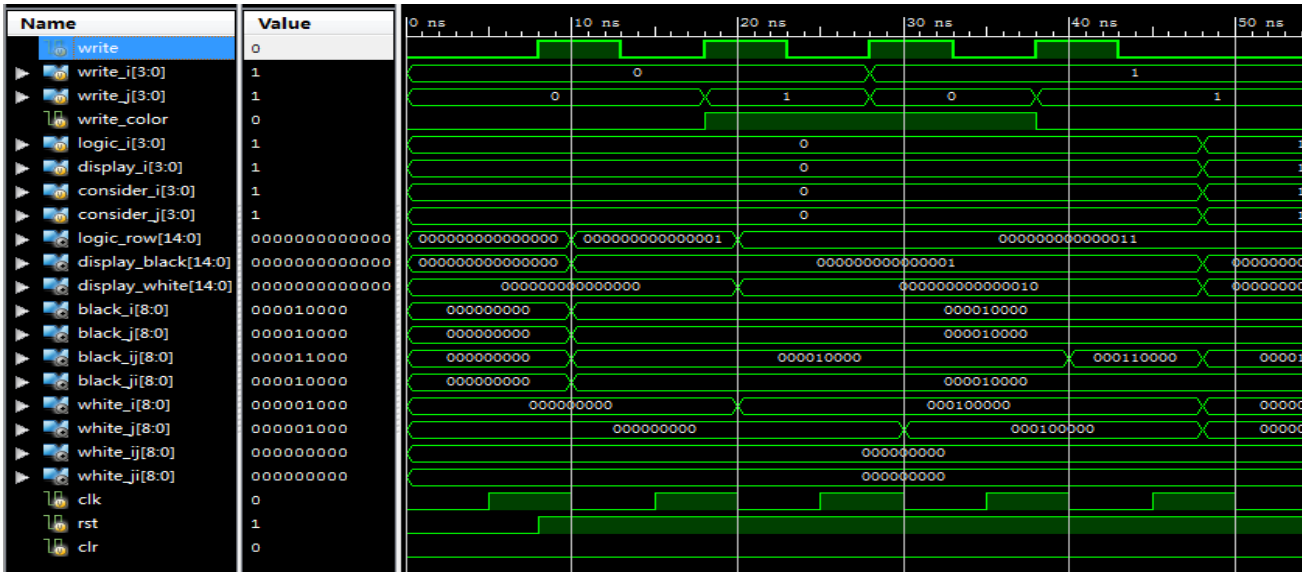
（1） 当前光标所在行的棋子放置信息。之后逻辑模块再根据光标的列坐标确定出光标对应棋盘网格交界处有、无放置棋子，以此控制有限状态机状态的跳转情况。

（2） 最近一次下棋位置附近的棋子放置信息。逻辑模块据此判断是否有一方获得了胜利。具体来讲，我们要传递出：列上以该位置为中心的 9 个位置、行上以该位置为中心的 9 个位置、左上—右下、左下—右上两条对角线上以该位置为中心的 9 个位置　共 4 组信息；　　而每个位置存在三种状态：未放棋子、放白棋子、放黑棋子，所以我们共传递出 8 组数据，有无白棋子 4 组、有无黑棋子 4 组，这样也便于逻辑模块根据放置棋子的颜色进行判断

设计过程中还对该模块进行了仿真测试，（仿真代码不在此赘述）

仿真代码在对应位置写入棋子，并且专门检验了对边界信息的处理能力。逐个检查输出信息，不难看出模块对棋盘边界外的数据正确输出了“0”，对棋盘内部的数据也按棋子情况正确输出了“0”或“1”。

波形图像截图如下：



3. PS/2 输入模块：

使用该模块进行输入的原因：Nexys 4 开发板上的自带按键，去抖动效果并不完美；而 PS/2 接口的设计特点可以消除这样的不良影响。并且使用键盘输入的用户体验更佳。

根据讲义中的介绍，截取了信号边沿（上升沿）

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
//这是 PS/2 模块的顶层模块，将底层接收到的通码转变为程序内部信号，传输至逻辑模块中
//////////////////////////////////////////////////////////////////////////////////
//////

module ps2_input(
    input wire clk_slow,
    input wire clk_fast,
```

```
    input wire rst,
    input wire PS2_CLK,
    input wire PS2_DATA,

    //通过W、A、S、D来控制光标上下左右移动
    //将信号传输至逻辑模块
    output wire key_up,
    output wire key_down,
    output wire key_left,
    output wire key_right,
    output wire key_ok
    );
    //用于接收scanner传来的通码
    wire [7:0] crt_data;


    //取信号边缘
    reg up_r1,up_r2,
        down_r1,down_r2,
        right_r1,right_r2,
        left_r1,left_r2,
        space_r1,space_r2;

    //截取信号的上升沿
    assign key_up = (up_r1 == 1) && (up_r2 == 0);
    assign key_down = (down_r1 == 1) && (down_r2 == 0);
    assign key_left = (left_r1 == 1) && (left_r2 == 0);
    assign key_right = (right_r1 == 1) && (right_r2 == 0);
    assign key_ok = (space_r1 == 1) && (space_r2 == 0);

    //调用 PS2_scanner 模块
    ps2_scan
        scanner(
            .clk(clk_fast),
            .rst(rst),
            .ps2_clk(PS2_CLK),
            .ps2_data(PS2_DATA),
            .crt_data(crt_data)
        );

    always @(posedge clk_slow or posedge rst)
        if(rst)
        begin
            up_r1 <= 1'b0;
```

```
            down_r1 <= 1'b0;
            left_r1 <= 1'b0;
            right_r1 <= 1'b0;
            space_r1 <= 1'b0;
        end
        else
        begin
        //查资料得对应通码
            up_r1 <= (crt_data == 8'h1D);
            down_r1 <= (crt_data == 8'h1B);
            left_r1 <= (crt_data == 8'h1C);
            right_r1 <= (crt_data == 8'h23);
            space_r1 <= (crt_data == 8'h29);
        end
    always @(posedge clk_slow or posedge rst)
        if (rst)
        begin
            up_r2 <= 1'b0;
            down_r2 <= 1'b0;
            left_r2 <= 1'b0;
            right_r2 <= 1'b0;
            space_r2 <= 1'b0;
        end
        else
        begin
            up_r2 <= up_r1;
            down_r2 <= down_r1;
            right_r2 <= right_r1;
            left_r2 <= left_r1;
            space_r2 <= space_r1;
        end
endmodule
```

其中处理接口处时钟信号的 ps2_scanner 模块源码如下：

为了方便处理，本次设计采用“W、A、S、D”四个键来控制光标上下左右的移动。这样的好处是：其通/断码信号与空格（space）建基本一致，不用处理来自四个箭头信号特殊的“E0”信号。

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
//////
```

```
//这是一个底层模块，主要用于接收 PS-2 接口传来的按键信息，输出接收到的通码或断码。
//////////////////////////////////////////////////////////////////////////////////
//////

module ps2_scan(
    input wire clk,
    input wire rst,
    input wire ps2_clk,
    input wire ps2_data,

    //传给上层模块的通码信号
    output reg [7:0] crt_data
    );
    reg [3:0]ps2_clk_neg_cnt;
    wire ps2_clk_neg;
    reg [7:0] read_data;
    reg get_f0;
    reg ps2_clk_r1,ps2_clk_r2;

    //获取 ps2_clk 的下降沿
    always @(posedge clk or posedge rst)
        if (rst)
            ps2_clk_r1 <= 1'b1;
        else
            ps2_clk_r1 <= ps2_clk;

    always @(posedge clk or posedge rst)
        if (rst)
            ps2_clk_r2 <= 1'b1;
        else
            ps2_clk_r2 <= ps2_clk_r1;

    assign ps2_clk_neg = (ps2_clk_r1 == 1'b0) && (ps2_clk_r2 == 1'b1);

    //利用 ps2 时钟信号读取数据
    always @ (posedge clk or posedge rst)
    begin
        if (rst)
        begin
            ps2_clk_neg_cnt <= 4'b0;
            read_data <= 8'b0;
            get_f0 <= 1'b0;
            crt_data <= 9'b0;
        end
```

```
        else if (ps2_clk_neg)
        begin
            if (ps2_clk_neg_cnt > 4'b1001)
                ps2_clk_neg_cnt <= 4'b0;
            else
            begin
                if (ps2_clk_neg_cnt > 4'b0 && ps2_clk_neg_cnt < 4'b1001)
                    read_data[ps2_clk_neg_cnt - 1] <= ps2_data;
                ps2_clk_neg_cnt <= ps2_clk_neg_cnt + 1'b1;
            end
        end
        else if (ps2_clk_neg_cnt == 4'b1010 && |read_data)
        begin
            if (read_data == 8'hf0)
                get_f0 <= 1'b1;
            else    //读入的数据不是f0
                if (get_f0) //上一个信号是f0，说明这是断码的后半段
                begin
                    get_f0 <= 1'b0;
                    crt_data <= 8'b0;
                end
                else
                    crt_data <= read_data;
            read_data <= 8'b0;
        end
    end
endmodule
```

4. VGA 显示模块：

首先根据显示屏的参数，设计一个模块，利用 IP 核生成的 25.125MHZ 频率时钟信号，产生我们扫描显示屏所需要的时序逻辑：

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// 640*480 @ 60HZ
//////////////////////////////////////////////////////////////////////////////////
//////
module vga_sync(
```

```
    input wire clk,     //25.125MHZ 的扫描信号
    input wire rst,
    output wire sync_h,
    output wire sync_v,
    //是否在显示区域
    output wire video_on,
    //X、Y 坐标
    output reg [9:0] x,
    output reg [9:0] y
    );
    assign sync_h = ~(x > 655 && x < 752);
    assign sync_v = ~(y > 489 && y < 492);

    //vedio_on 是一个控制信号
    //当 H 处于 c 段且 V 处于 h 段时，rgb 才可以输出有效视频信息
    assign video_on = (x < 640 && y < 480);
    //800 像素，525 行
    always @(posedge clk or posedge rst)
    begin
        if (rst)
        begin
            x <= 0;
            y <= 0;
        end
        else
        begin
            if (x == 799)
                x <= 0;
            else
                x <= x + 1'b1;
            if (y == 524)
                y <= 0;
            else if (x == 799)
                y <= y + 1'b1;
        end
    end
endmodule
```

之后 VGA 显示的主模块代码如下：

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
//这是 VGA 部分的顶层模块
//主要根据 VGA 时序生成模块传出的“坐标信息”与当前“棋盘数据”，计算需要显示的颜色;
```

```
//并将所需要的数据传输至主模块，连接到对应的约束端口上
//////////////////////////////////////////////////////////////////////////////////
//////
module vga_display(
    input wire clk,
    input wire rst,
    //光标位置
    input wire [3:0] cursor_i,
    input wire [3:0] cursor_j,

    //当前玩家
    input wire crt_player,

    //是否在游戏中
    input wire game_running,
    //赢家
    input wire [1:0] winner,

    //vga 显示所需的行信息
    input wire [14:0] display_black,
    input wire [14:0] display_white,

    //传递给 vga 模块的行信息
    output wire [3:0] display_i,

    //vga 行、列信息
    output wire sync_h,
    output wire sync_v,
    //  红绿蓝三色
    output wire [3:0] r,
    output wire [3:0] g,
    output wire [3:0] b
    );

    // 1 表示黑，0 表示白
    localparam BLACK = 1'b0,
               WHITE = 1'b1;

    //棋盘相关常量
    localparam BOARD_SIZE = 15,
               GRID_SIZE = 23,              //每格所占的宽度
               GRID_X_BEGIN = 148,
               GRID_X_END = 492,
               GRID_Y_BEGIN = 68,
```

```
            GRID_Y_END = 412;

    // 显示当前正在走棋的是 黑方 还是 白方 玩家
    localparam SIDE_BLACK_X_BEGIN = 545,
               SIDE_BLACK_X_END = 616,
               SIDE_BLACK_Y_BEGIN = 182,
               SIDE_BLACK_Y_END = 200,
               SIDE_WHITE_X_BEGIN = 545,
               SIDE_WHITE_X_END = 616,
               SIDE_WHITE_Y_BEGIN = 278,        //白色玩家在黑色玩家的下方
               SIDE_WHITE_Y_END = 296;

    // 当前玩家指示
    localparam CRT_BLACK_X_BEGIN = 510,
               CRT_BLACK_X_END = 541,
               CRT_BLACK_Y_BEGIN = 185,
               CRT_BLACK_Y_END = 198,
               CRT_WHITE_X_BEGIN = 510,
               CRT_WHITE_X_END = 541,
               CRT_WHITE_Y_BEGIN = 281,
               CRT_WHITE_Y_END = 294;

    // 标题展示
    localparam TITLE_X_BEGIN = 0,
               TITLE_X_END = 140,
               TITLE_Y_BEGIN = 62,
               TITLE_Y_END = 418;

    // 指令展示
    localparam INS_X_BEGIN = 145,
               INS_X_END = 494,
               INS_Y_BEGIN = 424,
               INS_Y_END = 467;

    // 展示最终赢家获胜信息的区域
    localparam RES_X_BEGIN = 187,
               RES_X_END = 452,
               RES_Y_BEGIN = 20,
               RES_Y_END = 47;
   // 作者信息
    localparam AUTHOR_X_BEGIN = 510,
               AUTHOR_X_END = 634,
               AUTHOR_Y_BEGIN = 455,
               AUTHOR_Y_END = 476;
```

```
    // 生成VGA 控制信号
    wire video_on;
    wire [9:0] x, y;
    vga_sync
        sync(
            .clk(clk),
            .rst(rst),
            .sync_h(sync_h),
            .sync_v(sync_v),
            .video_on(video_on),
            .x(x),
            .y(y)
        );

    // 当前显示的颜色，通过vedio_on 控制在合适的时间段，其余时间RGB 信号为0
    reg [11:0] rgb;
    assign r = video_on ? rgb[11:8] : 4'b0;
    assign g = video_on ? rgb[7:4]  : 4'b0;
    assign b = video_on ? rgb[3:0]  : 4'b0;

    // 作为棋盘展示用的寄存器
    reg [3:0] row, col;
    integer delta_x, delta_y;
    assign display_i = row < BOARD_SIZE ? row : 4'b0;

    // 需要被展示的图案
    wire [22:0] chess_piece_data;
    pic_chess_piece chess_piece(x >= GRID_X_BEGIN && x <= GRID_X_END && y >=
GRID_Y_BEGIN && y <= GRID_Y_END,
                                delta_y + GRID_SIZE/2, chess_piece_data);

    wire [71:0] black_player_data,
                white_player_data;
    pic_side_player black_player(x >= SIDE_BLACK_X_BEGIN &&
                                 x <= SIDE_BLACK_X_END &&
                                 y >= SIDE_BLACK_Y_BEGIN &&
                                 y <= SIDE_BLACK_Y_END,
                                 y - SIDE_BLACK_Y_BEGIN, black_player_data),
                    white_player(x >= SIDE_WHITE_X_BEGIN &&
                                 x <= SIDE_WHITE_X_END &&
                                 y >= SIDE_WHITE_Y_BEGIN &&
                                 y <= SIDE_WHITE_Y_END,
                                 y - SIDE_WHITE_Y_BEGIN, white_player_data);
```

```
wire [31:0] black_ptr_data, white_ptr_data;
pic_crt_ptr black_ptr(x >= CRT_BLACK_X_BEGIN && x <= CRT_BLACK_X_END &&
                      y >= CRT_BLACK_Y_BEGIN && y <= CRT_BLACK_Y_END,
                      y - CRT_BLACK_Y_BEGIN, black_ptr_data),
            white_ptr(x >= CRT_WHITE_X_BEGIN && x <= CRT_WHITE_X_END &&
                      y >= CRT_WHITE_Y_BEGIN && y <= CRT_WHITE_Y_END,
                      y - CRT_WHITE_Y_BEGIN, white_ptr_data);

wire [140:0] title_data;
pic_title title(x >= TITLE_X_BEGIN && x <= TITLE_X_END &&
                y >= TITLE_Y_BEGIN && y <= TITLE_Y_END,
                y - TITLE_Y_BEGIN, title_data);

wire [349:0] ins_start_data, ins_player_data;
pic_ins_start ins_start(x >= INS_X_BEGIN && x <= INS_X_END &&
                        y >= INS_Y_BEGIN && y <= INS_Y_END,
                        y - INS_Y_BEGIN, ins_start_data);
pic_ins_player ins_player(x >= INS_X_BEGIN && x <= INS_X_END &&
                          y >= INS_Y_BEGIN && y <= INS_Y_END,
                          y - INS_Y_BEGIN, ins_player_data);


wire [265:0] black_wins_data, white_wins_data, res_draw_data;
pic_black_wins black_wins(x >= RES_X_BEGIN && x <= RES_X_END &&
                          y >= RES_Y_BEGIN && y <= RES_Y_END,
                          y - RES_Y_BEGIN, black_wins_data);
pic_white_wins white_wins(x >= RES_X_BEGIN && x <= RES_X_END &&
                          y >= RES_Y_BEGIN && y <= RES_Y_END,
                          y - RES_Y_BEGIN, white_wins_data);
pic_res_draw res_draw(x >= RES_X_BEGIN && x <= RES_X_END &&
                      y >= RES_Y_BEGIN && y <= RES_Y_END,
                      y - RES_Y_BEGIN, res_draw_data);

wire [124:0] author_info_data;
pic_author_info author_info(x >= AUTHOR_X_BEGIN && x <= AUTHOR_X_END &&
                            y >= AUTHOR_Y_BEGIN && y <= AUTHOR_Y_END,
                            y - AUTHOR_Y_BEGIN, author_info_data);
// 计算当前的行与列
always @(x or y)
begin
    if (y >= GRID_Y_BEGIN &&
        y < GRID_Y_BEGIN + GRID_SIZE)
```

```
            row = 4'b0000;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE &&
                 y < GRID_Y_BEGIN + GRID_SIZE*2)
            row = 4'b0001;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*2 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*3)
            row = 4'b0010;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*3 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*4)
            row = 4'b0011;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*4 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*5)
            row = 4'b0100;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*5 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*6)
            row = 4'b0101;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*6 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*7)
            row = 4'b0110;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*7 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*8)
            row = 4'b0111;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*8 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*9)
            row = 4'b1000;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*9 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*10)
            row = 4'b1001;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*10 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*11)
            row = 4'b1010;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*11 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*12)
            row = 4'b1011;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*12 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*13)
            row = 4'b1100;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*13 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*14)
            row = 4'b1101;
        else if (y >= GRID_Y_BEGIN + GRID_SIZE*14 &&
                 y < GRID_Y_BEGIN + GRID_SIZE*15)
            row = 4'b1110;
        else
```

```
            row = 4'b1111;

        if (x >= GRID_X_BEGIN &&
            x < GRID_X_BEGIN + GRID_SIZE)
            col = 4'b0000;
        else if (x >= GRID_X_BEGIN + GRID_SIZE &&
                 x < GRID_X_BEGIN + GRID_SIZE*2)
            col = 4'b0001;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*2 &&
                 x < GRID_X_BEGIN + GRID_SIZE*3)
            col = 4'b0010;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*3 &&
                 x < GRID_X_BEGIN + GRID_SIZE*4)
            col = 4'b0011;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*4 &&
                 x < GRID_X_BEGIN + GRID_SIZE*5)
            col = 4'b0100;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*5 &&
                 x < GRID_X_BEGIN + GRID_SIZE*6)
            col = 4'b0101;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*6 &&
                 x < GRID_X_BEGIN + GRID_SIZE*7)
            col = 4'b0110;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*7 &&
                 x < GRID_X_BEGIN + GRID_SIZE*8)
            col = 4'b0111;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*8 &&
                 x < GRID_X_BEGIN + GRID_SIZE*9)
            col = 4'b1000;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*9 &&
                 x < GRID_X_BEGIN + GRID_SIZE*10)
            col = 4'b1001;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*10 &&
                 x < GRID_X_BEGIN + GRID_SIZE*11)
            col = 4'b1010;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*11 &&
                 x < GRID_X_BEGIN + GRID_SIZE*12)
            col = 4'b1011;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*12 &&
                 x < GRID_X_BEGIN + GRID_SIZE*13)
            col = 4'b1100;
        else if (x >= GRID_X_BEGIN + GRID_SIZE*13 &&
                 x < GRID_X_BEGIN + GRID_SIZE*14)
            col = 4'b1101;
```

```
        else if (x >= GRID_X_BEGIN + GRID_SIZE*14 &&
                 x < GRID_X_BEGIN + GRID_SIZE*15)
            col = 4'b1110;
        else
            col = 4'b1111;


        //计算对应棋子的显色坐标
        //棋子在棋盘格中央处放置，GRID_SIZE/2
        //delta_x/y 表示当前扫描的（x，y）坐标距离对应的(col , row)棋盘格处有多远
        delta_x = GRID_X_BEGIN + col*GRID_SIZE + GRID_SIZE/2 - x;
        delta_y = GRID_Y_BEGIN + row*GRID_SIZE + GRID_SIZE/2 - y;
    end

    // 计算颜色信息
    always @(posedge clk)
    begin
        if (x >= GRID_X_BEGIN && x <= GRID_X_END &&
            y >= GRID_Y_BEGIN && y <= GRID_Y_END)
            // 画出棋盘（网格、棋子、光标）
            begin

            if (display_black[col] &&
                chess_piece_data[delta_x + GRID_SIZE/2])
                //黑色
                rgb <= 12'h000;
            else if (display_white[col] &&
                     chess_piece_data[delta_x + GRID_SIZE/2])
                // 白色
                rgb <= 12'hfff;
            else if (row == cursor_i && col == cursor_j &&
                    (delta_x == GRID_SIZE/2 || delta_x == -(GRID_SIZE/2) ||
                     delta_y == GRID_SIZE/2 || delta_y == -(GRID_SIZE/2)))
                // 红色光标，注意这时候上面判断中的条件都是取等
                rgb <= 12'hf00;
            else if (delta_x == 0 || delta_y == 0)
                // 浅色边
                rgb <= 12'hda6;
            else if (delta_x == 1 || delta_y == 1)
                // 深色边
                rgb <= 12'h751;
            else    //棋盘部分背景色，金色 fd0
                rgb <= 12'hfd0;
        end
```

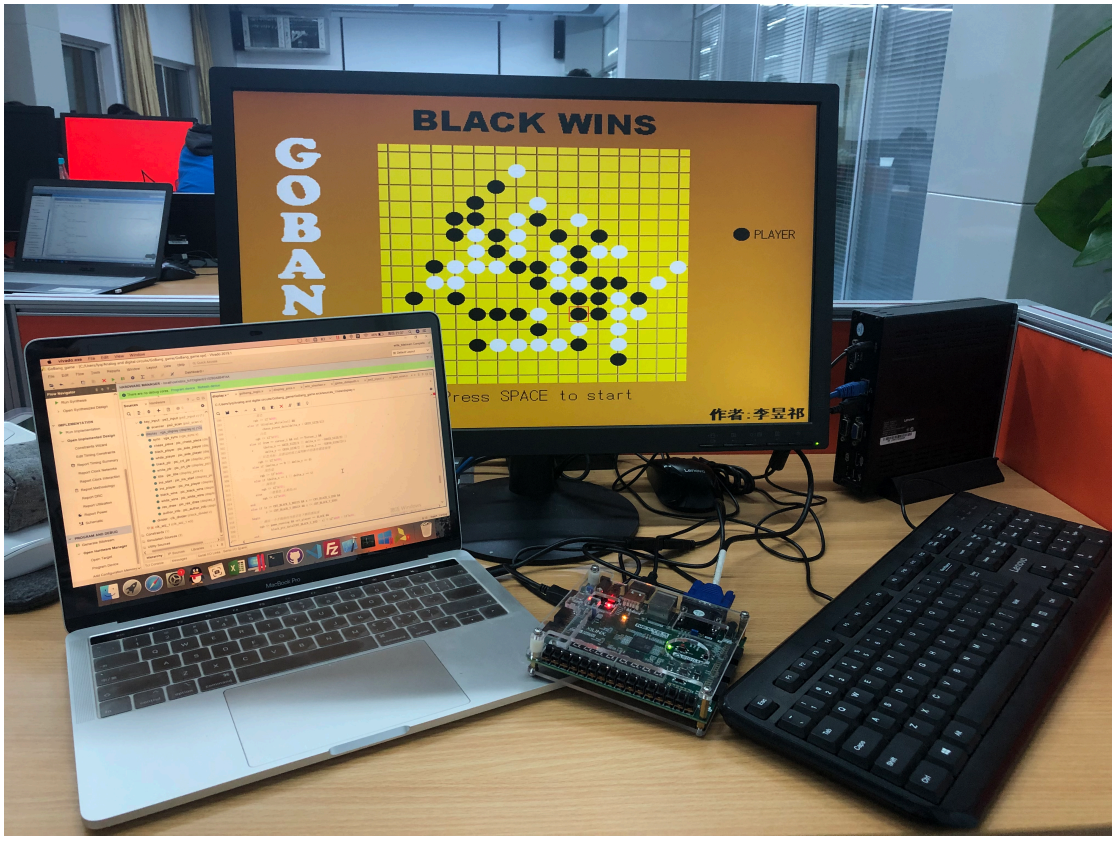
```
        else if (x >= CRT_BLACK_X_BEGIN && x <= CRT_BLACK_X_END &&
                 y >= CRT_BLACK_Y_BEGIN && y <= CRT_BLACK_Y_END)
        begin
            // 画出一个手指指向当前正在下棋的黑玩家
            //间隙部分仍用背景色  c81
            rgb <= game_running && crt_player == BLACK &&
                   black_ptr_data[CRT_BLACK_X_END - x] ? 12'h000 : 12'hc81;
        end
        else if (x >= CRT_WHITE_X_BEGIN && x <= CRT_WHITE_X_END &&
                 y >= CRT_WHITE_Y_BEGIN && y <= CRT_WHITE_Y_END)
        begin
            // 画出一个手指指向当前正在下棋的白玩家
            rgb <= game_running && crt_player == WHITE &&
                   white_ptr_data[CRT_WHITE_X_END - x] ? 12'hfff : 12'hc81;
        end
        else if (x >= SIDE_BLACK_X_BEGIN && x <= SIDE_BLACK_X_END &&
                 y >= SIDE_BLACK_Y_BEGIN && y <= SIDE_BLACK_Y_END)
        begin

            if (crt_player == BLACK)
            // 显示出黑色玩家信息
                rgb <= black_player_data[SIDE_BLACK_X_END - x] ?
                       12'h000 : 12'hc81;
        end
        else if (x >= SIDE_WHITE_X_BEGIN && x <= SIDE_WHITE_X_END &&
                 y >= SIDE_WHITE_Y_BEGIN && y <= SIDE_WHITE_Y_END)
        begin
            // 显示出白色玩家信息
            if (crt_player == WHITE)
                rgb <= white_player_data[SIDE_WHITE_X_END - x] ?
                       12'hfff : 12'hc81;
        end
        else if (x >= INS_X_BEGIN && x <= INS_X_END &&
                 y >= INS_Y_BEGIN && y <= INS_Y_END)
        begin
            // 在非游戏进行时提醒玩家按下空格开始游戏
            if (!game_running)
                rgb <= ins_start_data[INS_X_END - x] ? 12'h000 : 12'hc81;
        end
        else if (x >= RES_X_BEGIN && x <= RES_X_END &&
                 y >= RES_Y_BEGIN && y <= RES_Y_END)
        begin
            // 显示出赢家信息或平局，该区域可能有三个不同的显示
            case (winner)
```

```
            2'b00: rgb <= 12'hc81;
            2'b01: rgb <= black_wins_data[RES_X_END - x] ? 12'h000 : 12'hc81;
            2'b10: rgb <= white_wins_data[RES_X_END - x] ? 12'hfff : 12'hc81;
            2'b11: rgb <= res_draw_data[RES_X_END - x] ? 12'hfff : 12'hc81;
            endcase
        end
        else if (x >= TITLE_X_BEGIN && x <= TITLE_X_END &&
                 y >= TITLE_Y_BEGIN && y <= TITLE_Y_END)
            // 写出标题
            rgb <= title_data[TITLE_X_END - x] ? 12'hfff : 12'hc81;
        else if (x >= AUTHOR_X_BEGIN && x <= AUTHOR_X_END &&
                 y >= AUTHOR_Y_BEGIN && y <= AUTHOR_Y_END)
        begin
            // 显示出我的名字
            rgb <= author_info_data[AUTHOR_X_END - x] ? 12'h000 : 12'hc81;
        end
        else
            // 画出背景，c81 土黄色
            rgb <= 12'hc81;
    end
endmodule
```

其中画图调用的模块，实际上就是一些简单的 ROM，存储了一些我们提前设置好的文字图像信息，在 x、y 信号扫描至某一特定位置后，先根据 y 值（y 值距离图像区域边界的偏移量），从 ROM 中读取一列信息，再根据 x 值（x 代表行信息），从这一列中读取出对应一个像素点的 bool 值，若为真“1”则给 RGB 赋值，显示我们需要的颜色；否则就显示背景颜色等。

【效果展示】

（于 2019 年 12 月 12 日晚，摄于中国科学技术大学西校区电三楼 410 教室）



【总结与思考】

1. （1）对整个学期的实验进行了系统性的回顾，进行了一次极具综合性、功能性、复杂性、创新性的电路设计。

（2）学习使用了有关 PS/2 和 VGA 接口的相关知识，自己编写了处理接口处信息交换的模块，丰富了自己硬件相关的知识

（3）熟练了使用 Testbench 对 verilog 进行仿真的过程，在模拟过程中对模块设计的正确性进行了检验，省去了烧写至板子上再实际检验的不便。

2.任务量：　略大

3.难度：　适中

4.改进建议：

（1）在实验室多配备一些PS/2兼容性更好的键盘；

（2）大作业检查ddl前一周尽量将406大教室空闲时间开放给学生使用