浙江大学

课程设计报告

中文题目：基于数字系统的带 AI 五子棋游戏

英文题目：Verilog Gobang Game with AI

姓名学号：

指导教师：

参加成员：

专业类别：计算机科学与技术

所在学院：计算机学院

论文提交日期 2017 年 1 月 3 日

摘要

本课程设计是一份数字逻辑设计的课程设计，使用硬件描述语言 Verilog HDL 在实验室的 SWORD Kintex-7 开发板上实现了一个基于数字系统的带 AI 五子棋游戏，并对实现中的各种细节包括简单 AI 策略的解析、系统结构设计的详尽分析、AI 计算量的优化方法、硬件模块设计详尽分析过程、各模块的硬件描述、仿真测试问题与解决方案等内容进行了较为详尽的描述。本课程设计综合使用了数字逻辑设计课程中所介绍的各个知识点，包括组合电路与时序电路、状态机、RAM 等，还综合使用了课堂上没有介绍的如 VGA、PS/2 接口等知识点。

关键词： 五子棋，AI，Verilog，数字系统，数字逻辑设计，课程设计

目录

图目录

表目录

第一章 绪论

1.1 五子棋游戏设计背景

五子棋是一种规则简单而富有趣味性的棋类游戏。游戏在一块 15×15 的棋盘上进行，黑白双方轮流行子，先达成五子连珠的一方获胜。若棋盘填满后未出胜负，则双方平局。

五子棋游戏规则简单，仅有若干种不同的基本棋型，初学者易于上手，也很适合简单 AI 的开发。正是因为五子棋简单而趣味的特点，我决定实现一个基于数字系统的带 AI 五子棋游戏，作为数字逻辑设计课程的课程设计。

1.2 国内外现况分析

五子棋作为一种起源于中国的古老游戏，现在已经在世界范围内拥有大批的爱好者，在机器学习等智能领域也有人在进行研究。虽然当前五子棋的 AI 水平还无法与顶尖人类选手匹敌，但是许多研究人员与爱好者也提出了一些简单的五子棋智能算法。

虽然实现五子棋智能算法的方式多种多样，但是使用硬件描述语言，基于数字系统实现的五子棋 AI 却鲜有资料。我搜索了开源社区 Github 的代码库，并没有发现类似设计的先例。

1.3 主要内容和难点

本课程设计希望制作一个基于数字系统的带 AI 五子棋游戏，使用硬件描述语言 Verilog HDL 实现，运行在实验室的 SWORD Kintex7 开发板上。游戏需要实现玩家与 AI、玩家与玩家以及 AI 与 AI 之间的对局功能。简单起见，我们不考虑五子棋中的禁手情况。游戏的画面通过 VGA 接口由显示器输出，显示分辨率为 640×480，画面刷新频率为 60Hz；游戏的指令通过 PS/2 接口由键盘给出。

由于本设计中涉及到简单的五子棋智能，需要一定量的计算，实现过程中可能会消耗较多的电路资源。所以，本设计的难点不仅在于五子棋 AI 的实现，还在于同时尽量降低电路资源的消耗。

第二章 五子棋游戏设计原理

2.1 五子棋游戏设计相关内容

本课程设计主要使用了有限状态自动机、VGA 接口、PS/2 接口以及简单五子棋 AI 等内容。其中，有限状态自动机用于游戏主要状态的转换，VGA 接口用于游戏画面的显示，PS/2 接口用于游戏命令的输入。以下一一进行简单介绍。

2.1.1 有限状态自动机

有限状态自动机，简称状态机，是表示有限多个状态以及在这些状态之间转移和动作的数学模型。状态存储关于过去的信息，它反映从系统开始到现在时刻输入的变化；转移指示状态变更，用必须满足来确使转移发生的条件来描述它；动作是在给定时刻要进行的活动描述。

在时序电路的设计中，我们经常使用两种状态机——Moore 型状态机与 Mealy 型状态机。Moore 状态机的输出只与当前的状态有关，Mealy 状态机的输出与当前状态和输入有关，而两者的下一状态都与当前状态和输入有关。两种状态机的图示见下图。

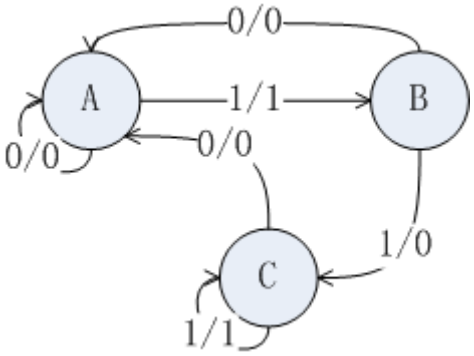

图 1 Mealy 状态机

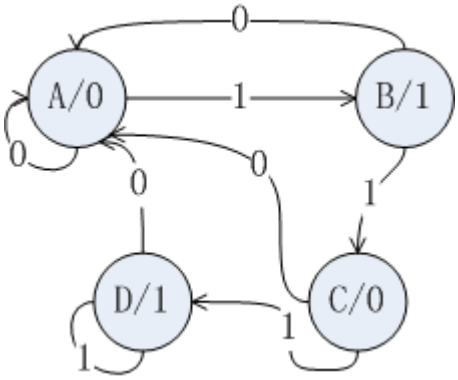

图 2 Moore 状态机

不难看出，Moore 型状态机与 Mealy 型状态机是互相等价，可以互相转换的。但由于 Mealy 型状态机更加简介，本课程设计选择了 Mealy 型状态机的模型作为分析依据。

2.1.2 VGA 接口

VGA 是 IBM 在 1987 年随推出的一种视频传输标准，具有分辨率高、显示速率快、颜色丰富等优点，在彩色显示器领域得到了广泛的应用。

表 1 VGA 常用信号线

信号线	定义
HS	列同步信号（3.3V 电平）
VS	行同步信号（3.3V 电平）
R	红基色（0~0.714V 模拟信号）
G	绿基色（0~0.714V 模拟信号）
B	蓝基色（0~0.714V 模拟信号）

为了在显示器上显示我们需要的图案，我们需要使用标准 VGA 接口中的上述信号线。

HS 与 VS 用于控制显示器的显示频率，R、G 与 B 则用于控制显示器当前像素的颜色。

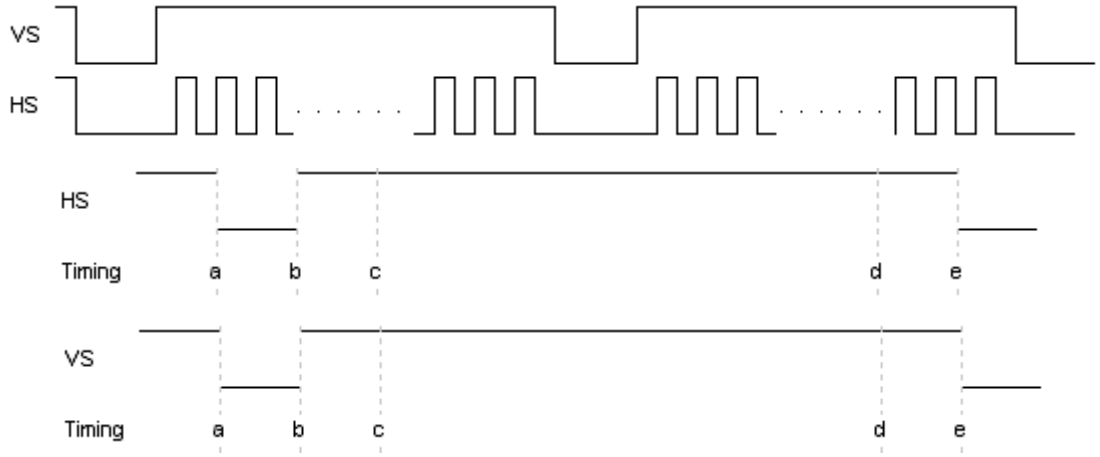

图 3　VGA 时序

上图是 VGA 的标准时序。不难看出，列同步信号与行同步信号的时序都分为 4 段。a-b 段称为同步期，此时同步信号为低电平；b-c 段称为消隐后肩，c-d 段称为显示期，是数据的有效区域，d-e 段称为消隐前肩，这三段的同步信号为高电平。显示器就是根据列同步信号与行同步信号的周期，来决定显示的分辨率、频率等参数。不同分辨率与频率对应的 VS-HS 数值，可以通过查表得到。

我们只需要在一个 VS-HS 完成一个周期的时间内，按从左上角到右下角的顺序输出各个像素点的颜色，就能在显示器上显示出对应的图案。需要注意的是，VGA 颜色的输出必须严格遵循时序，只能在 c-d 段输出非 0 数据，否则会对显示造成干扰。

2.1.3　PS/2 接口

PS/2 原是“personal 2”的意思是 IBM 公司在上个世纪 80 年代推出的一种个人电脑。因为标准不开放，PS/2 电脑在市场中失败了。只有 PS/2 接口一直沿用到今天。

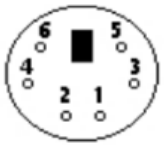

图 4　PS/2 接口

表 2　PS/2 信号线

编号	定义
1	数据
3	接地
4	电源（5V）
5	时钟
2/6	保留接口

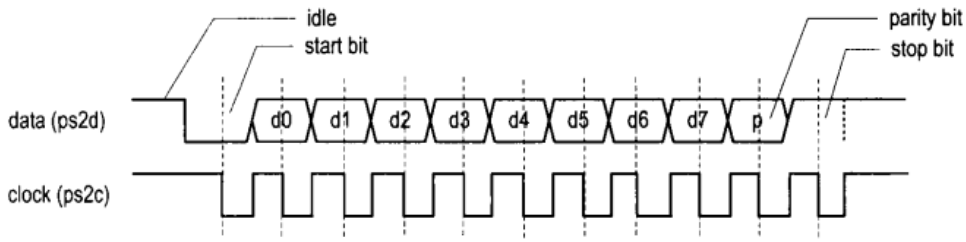

图 5　PS/2 时序

键盘内部自带一个处理器，负责扫描各个按键的状态。当键盘上某一个键被按下或放开后，键盘经过按键信号防抖等处理，会将一串特定的编码通过数据端口串行发送。如 PS/2 时序图所示，接收这些数据的模块应在 PS/2 时钟信号处于下降沿时读取并处理数据。

PS/2 键盘发送的数据每次共有 11 位。其中，第 1 位为起始位，第 2~9 位为 8 位的数据位，第 10 位为奇偶校验位，最后一位为停止位。中间 8 位的数据位就包含按键按下或放开的信息。当有按键被按下时，键盘发送的是对应按键的通码；当有按键被放开时，键盘发送的是对应按键的断码。各个按键的通码与短码可以通过查表得知。

2.1.4 简单的五子棋 AI

本课程设计使用了一种“贪心”的方法来实现一个简单的五子棋 AI。我们知道，五子棋有一些基本棋型，如“冲四”、“活三”等等。根据每种棋型“威胁性”不同，我们可以给每种棋型不同的分数。这样，我们就能算出下一步所有能落子的位置的分数。

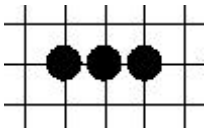

图 6 活三棋型

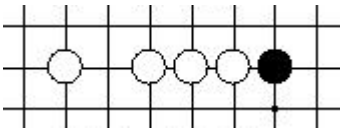

图 7 跳冲四棋型

我们实现的简单 AI 可以采用以下贪心策略：计算下一步自己与对手的可能最高得分。若自己的可能最高得分较高，则下在自己的对应位置，否则就下在对手可能最高得分对应的位置，阻止对手获得可能最高得分。

很显然，这个简单 AI 是一个“只顾眼前”的 AI，不会考虑棋局的“长远发展”。不过，由于这个 AI 的策略比较简单，很适合在课程设计有限的时间内进行实现。

2.2 五子棋游戏设计方案

2.2.1 五子棋游戏结构设计

根据本课程设计希望达成的效果，我们不难将整个工程分成六个模块。其中四个主要模块为 PS/2 输入模块、VGA 显示模块、棋盘数据存储模块以及游戏逻辑模块。另外还有两个辅助模块为随即模块与时钟分频模块。课程设计结构框图如下。

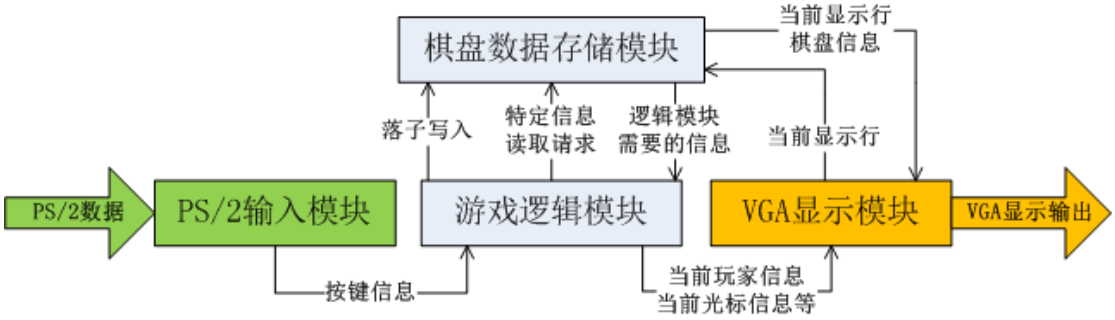

图 8 五子棋游戏结构框图

棋盘数据存储模块存储当前棋盘的棋子信息，有一个写入端口用于写入游戏逻辑模块给

出的下棋信息，还有多个读出端口用于给游戏逻辑模块与 VGA 显示模块送出需要的棋盘信息。将数据存储模块与其它模块独立可以更加方便地管理棋盘数据，并能方便地增加端口送出其它模块需要的数据。

游戏逻辑模块是游戏的主要逻辑，包含了游戏的主要状态机、简单的 AI、胜利条件的判断等等与逻辑相关的内容。它从 PS/2 输入模块读取按键信息，并根据按键信息代表的指令进行逻辑处理。在下棋操作进行后，它向棋盘数据存储模块写入下棋的信息。在 AI 需要进行计算，或需要判断胜利条件等情况下，它向棋盘数据存储模块发出特定信息的读取信号，并读取送出的棋盘信息。同时，它还会将不在棋盘上的信息（如当前玩家、当前光标等）送至 VGA 显示模块用于显示。

PS/2 输入模块较为简单。它读取 PS/2 端口送入的数据，经过处理后转换为特定按键的按键信息，送至游戏逻辑模块进行处理。

VGA 显示模块也不复杂，它向棋盘数据存储模块发送当前显示需要的棋盘行信息，并从游戏逻辑模块与棋盘数据存储模块读取相应信息用于显示。

两个辅助模块中，时钟分频模块读入 100MHz 的时钟信号，根据不用主要模块的需求，分别送出不同频率的时钟信号；随机模块产生伪随机数，送至游戏逻辑模块用于需要随机的逻辑过程。

2.2.2 五子棋游戏设计思路

根据上述分析，我们不难得出各个模块的设计思路。

对于 VGA 显示模块，由于本课程设计的画面表现较为简单，不需要过多复杂的计算，为了实现简单起见，我们不使用 VRAM 作为中间过渡，而是直接在显示模块中计算当前像素的颜色并输出。查表可知，640×480@60Hz 的显示需要 25.125MHz，即约 25MHz 的时钟信号，将分频后的第 1 位时钟信号送入即可。

对于游戏逻辑模块，我们主要使用一个状态机维护游戏的进程，在不同的状态下进行不同的逻辑处理。由于本课程设计含有 AI、条件判断等内容，需要较为大量的计算，如果直接使用一个频率很高的时钟，可能会导致一个周期内计算无法完成的问题。并且，考虑到玩家的反应速度与电路的执行效率相比是非常缓慢的，所以对于游戏逻辑模块的主要状态机部分，我们只要使用一个约 1000Hz 的时钟信号即可。

不过对于游戏逻辑模块，我们还存在大量计算带来的电路资源消耗的问题。以 AI 的实现为例，如果我们在一个时钟周期内，就将整个棋盘上每个格点的得分全部计算出来，那么就需要 15×15（棋盘大小）个计算电路，而每个计算电路内部也具有一定的电路复杂度。这种做法不仅非常消耗电路资源，在综合过程中还可能会因计算机资源的大量消耗而导致综合失败（我在第一版程序的综合过程中，计算机就发生了死机）。这个问题的解决方法将放在第 3.1 节进行详述。

对于棋盘数据存储模块，不难想到使用一个 RAM 来存储棋盘信息。由于存储模块的写入频率与逻辑模块的频率有关，所以存储模块的时钟信号也只需要与逻辑模块相同，即使用约 1000Hz 的时钟信号即可。然而，存储模块的读取频率与逻辑模块和显示模块都相关，而

显示模块的时钟频率是比较快的。所以，我们不妨将存储模块的写入制作成时序逻辑，而读取用组合逻辑实现。这样既保证了信息写入的稳定性，又保证了信息读取的及时性，也简化了模块实现的方法。

最后，对于 PS/2 输入模块，为了给逻辑模块提供按键上升沿的信息，模块的主要时钟频率需要与逻辑模块相同，即约 1000Hz 的时钟信号。但是，由于 PS/2 接口输入的时钟信号频率约为 10KHz，我们使用较为缓慢的时钟信号可能会导致读入信息的丢失。所以，我们还需要给 PS/2 输入模块提供一个较快的时钟（1MHz 已经完全足够），用于 PS/2 接口输入信息的读取。

2.3 五子棋游戏硬件设计

以下给出各个模块的 Verilog HDL 代码。每个模块详尽的设计分析将在第 3.1 节进行。

2.3.1 游戏逻辑模块

游戏逻辑模块主要包括以下内容。

（1）逻辑顶层模块 gobang_logic：包含了主要状态机，是逻辑部分的主要控制模块；

（2）AI 策略模块 gobang_strategy：简单的五子棋 AI 在此实现。每当激活信号来临时，模块将会运行一次，并送出双方下一步最高可能得分与对应下棋位置；

（3）胜利判断模块 win_checker：每当激活信号来临时，模块将会运行一次，运行结束后送出当前棋局中是否有一方获胜；

（4）分数计算模块 score_calculator：给输入的一行棋子信息算出分数的模块；

（5）棋型判断模块 gobang_patterns：由许多棋型判断模块组成，判断输入的一行棋子信息是否符合某种棋型。

-------- 逻辑顶层模块 gobang_logic --------

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------
// The logic part of the gobang game
//----------------------------------------------------------------------------
module gobang_logic(
    input wire clk_slow,            // Slower clock for main FSM
    input wire clk_fast,            // Faster clock for strategy and checker
    input wire rst,                 // Reset
    input wire random,              // Random signal

    input wire key_up,              // If UP key is pressed
    input wire key_down,            // If DOWN key is pressed
    input wire key_left,            // If LEFT key is pressed
    input wire key_right,           // If RIGHT key is pressed
    input wire key_ok,              // If OK key is pressed
    input wire key_switch,          // If SWITCH key is pressed

    input wire [8:0] black_i,       // Row information for strategy/checker
    input wire [8:0] black_j,       // Column information for s/c
```

```
input wire [8:0] black_ij,       // Main diagonal information for s/c
input wire [8:0] black_ji,       // Counter diagonal information for s/c
input wire [8:0] white_i,
input wire [8:0] white_j,
input wire [8:0] white_ij,
input wire [8:0] white_ji,
input wire [14:0] chess_row,     // Row information for main logic

output wire [3:0] consider_i,    // The row s/c is considering
output wire [3:0] consider_j,    // The column s/c is considering

output reg data_clr,            // Clear the datapath
output reg data_write,          // Writing-enable signal of the datapath

output reg [3:0] cursor_i,       // Row of the cursor
output reg [3:0] cursor_j,       // Column of the cursor
output reg black_is_player,      // If black is not AI
output reg white_is_player,      // If white is not AI
output reg crt_player,          // Current player
output reg game_running,        // If the game is running
output reg [1:0] winner          // Who wins the game
);

// Side parameters
localparam BLACK = 1'b0,
           WHITE = 1'b1;

// Chessboard parameters
localparam BOARD_SIZE = 15;

// State parameters
localparam STATE_IDLE = 3'b000,
           STATE_MOVE = 3'b001,
           STATE_WAIT = 3'b010,
           STATE_DECIDE = 3'b011,
           STATE_PUT_CHESS = 3'b100,
           STATE_PUT_END = 3'b101,
           STATE_CHECK = 3'b110,
           STATE_GAME_END = 3'b111;

// Lasting time for STATE_WAIT
localparam WAIT_TIME = 400;

reg [2:0] state;         // Current game state
reg [8:0] wait_count;
reg [7:0] move_count;    // How many moves have been played

// A simple strategy for the game
reg strategy_clr, strategy_active;
wire [3:0] strategy_i, strategy_j;
wire [12:0] black_best_score;
wire [3:0] black_best_i, black_best_j;
wire [12:0] white_best_score;
wire [3:0] white_best_i, white_best_j;
gobang_strategy
    strategy(
        .clk(clk_fast),
        .rst(rst),
        .clr(strategy_clr),
```

```
        .active(strategy_active),
        .random(random),
        .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),
        .get_i(strategy_i),
        .get_j(strategy_j),
        .black_best_score(black_best_score),
        .black_best_i(black_best_i),
        .black_best_j(black_best_j),
        .white_best_score(white_best_score),
        .white_best_i(white_best_i),
        .white_best_j(white_best_j)
    );

// A checker to check if someone wins the game
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),
        .get_i(win_i),
        .get_j(win_j),
        .is_win(is_win)
    );

// The strategy and the checker uses the same port to get information,
// but they will not use it at the same time
assign consider_i = strategy_i | win_i;
assign consider_j = strategy_j | win_j;

// Main FSM of the game
always @ (posedge clk_slow or negedge rst) begin
    if (!rst) begin
        cursor_i <= BOARD_SIZE;
        cursor_j <= BOARD_SIZE;
        {white_is_player, black_is_player} <= 2'b01;
        crt_player <= BLACK;
        game_running <= 1'b0;
        winner <= 2'b00;

        state <= STATE_IDLE;
```

```
        move_count <= 8'b0;
        data_clr <= 1'b0;
        data_write <= 1'b0;
        strategy_clr <= 1'b0;
        strategy_active <= 1'b0;
        win_clr <= 1'b0;
        win_active <= 1'b0;
    end
    else begin
        case (state)
        STATE_IDLE:
            if (key_ok) begin
                // Press OK key to start the game
                cursor_i <= BOARD_SIZE/2;
                cursor_j <= BOARD_SIZE/2;
                crt_player <= BLACK;
                game_running <= 1'b1;
                winner <= 2'b00;

                state <= STATE_MOVE;
                move_count <= 8'b0;
                data_clr <= 1'b1;
                strategy_clr <= 1'b1;
                win_clr <= 1'b1;
            end
            else if (key_switch)
                // Switch player
                {white_is_player, black_is_player}
                <= {white_is_player, black_is_player} + 2'b1;
            else
                state <= STATE_IDLE;

        STATE_MOVE: begin
            data_clr <= 1'b0;
            strategy_clr <= 1'b0;
            win_clr <= 1'b0;

            if ((crt_player == BLACK && black_is_player) ||
                (crt_player == WHITE && white_is_player)) begin
                // Player's move

                // Move the cursor
                if (key_up && cursor_i > 0)
                    cursor_i <= cursor_i - 1'b1;
                else if (key_down && cursor_i < BOARD_SIZE - 1)
                    cursor_i <= cursor_i + 1'b1;
                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;

                if (key_ok)
                    // Press OK key to put the chess
                    state <= STATE_PUT_CHESS;
            end
            else begin
                // CPU's move
                strategy_active <= 1'b1;
                state <= STATE_WAIT;
```

```
            wait_count <= 0;
        end
    end

    STATE_WAIT:
        // Wait for a while, otherwise the CPU will deicide too fast
        if (wait_count >= WAIT_TIME)
            state <= STATE_DECIDE;
        else
            wait_count <= wait_count + 1'b1;

    STATE_DECIDE: begin
        // Compare the best possible score for my side and the opposite,
        // and choose the best position
        strategy_active <= 1'b0;
        if (black_best_score > white_best_score ||
            (black_best_score == white_best_score &&
            crt_player == BLACK)) begin
            cursor_i <= black_best_i;
            cursor_j <= black_best_j;
        end
        else begin
            cursor_i <= white_best_i;
            cursor_j <= white_best_j;
        end

        state <= STATE_PUT_CHESS;
    end

    STATE_PUT_CHESS:
        // Check if the position is occupied. If not, put the chess
        if (!chess_row[cursor_j]) begin
            move_count <= move_count + 8'b1;
            data_write <= 1'b1;
            state <= STATE_PUT_END;
        end
        else
            state <= STATE_MOVE;

    STATE_PUT_END: begin
        data_write <= 1'b0;
        win_active <= 1'b1;
        state <= STATE_CHECK;
    end

    STATE_CHECK: begin
        // Check if someone wins the game or there is a draw
        win_active <= 1'b0;
        if (is_win || move_count == BOARD_SIZE * BOARD_SIZE)
            state <= STATE_GAME_END;
        else begin
            crt_player <= ~crt_player;
            state <= STATE_MOVE;
        end
    end

    STATE_GAME_END: begin
        if (is_win)
            // Someone wins the game
```

```
                winner <= 2'b01 << crt_player;
            else
                // There is a draw
                winner <= 2'b11;

            state <= STATE_IDLE;
            game_running <= 1'b0;
        end

        endcase
    end
end

endmodule
```

-------- AI 策略模块 gobang_strategy --------

```
`timescale 1ns / 1ps

//------------------------------------------------------------------------------
// A simple strategy for the gobang game
//------------------------------------------------------------------------------
module gobang_strategy(
    input wire clk,                     // Clock
    input wire rst,                     // Reset
    input wire clr,                     // Clear
    input wire active,                  // Active signal
    input wire random,                  // Random signal

    input wire [8:0] black_i,            // Row information
    input wire [8:0] black_j,            // Column information
    input wire [8:0] black_ij,           // Main diagonal information
    input wire [8:0] black_ji,           // Counter diagonal information
    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 [3:0] get_i,              // Current row considered
    output reg [3:0] get_j,              // Current column considered

    output reg [12:0] black_best_score,   // Best possible score
    output reg [3:0] black_best_i,        // Best row
    output reg [3:0] black_best_j,        // Best column
    output reg [12:0] white_best_score,
    output reg [3:0] white_best_i,
    output reg [3:0] white_best_j
    );

    // Chessboard parameters
    localparam BOARD_SIZE = 15;

    // State parameters
    localparam STATE_IDLE = 1'b0,
               STATE_WORKING = 1'b1;

    reg state;    // Current state

    // Scores of the four directions
    wire [12:0] black_score_i, black_score_j, black_score_ij, black_score_ji;
```

```
wire [12:0] white_score_i, white_score_j, white_score_ij, white_score_ji;

// Total scores
wire [12:0] black_score, white_score;
assign black_score = black_score_i + black_score_j +
                     black_score_ij + black_score_ji;
assign white_score = white_score_i + white_score_j +
                     white_score_ij + white_score_ji;

// Score calculators
score_calculator
    calc_black_i(
        .my(black_i),
        .op(white_i),
        .score(black_score_i)
    ),
    calc_black_j(
        .my(black_j),
        .op(white_j),
        .score(black_score_j)
    ),
    calc_black_ij(
        .my(black_ij),
        .op(white_ij),
        .score(black_score_ij)
    ),
    calc_black_ji(
        .my(black_ji),
        .op(white_ji),
        .score(black_score_ji)
    ),
    calc_white_i(
        .my(white_i),
        .op(black_i),
        .score(white_score_i)
    ),
    calc_white_j(
        .my(white_j),
        .op(black_j),
        .score(white_score_j)
    ),
    calc_white_ij(
        .my(white_ij),
        .op(black_ij),
        .score(white_score_ij)
    ),
    calc_white_ji(
        .my(white_ji),
        .op(black_ji),
        .score(white_score_ji)
    );

// FSM of the strategy
// Every time the active signal comes, the strategy will run once
always @ (posedge clk or negedge rst) begin
    if (!rst || clr) begin
        get_i <= 4'b0;
        get_j <= 4'b0;
        black_best_score <= 0;
```

```
            black_best_i <= BOARD_SIZE / 2;
            black_best_j <= BOARD_SIZE / 2;
            white_best_score <= 0;
            white_best_i <= BOARD_SIZE / 2;
            white_best_j <= BOARD_SIZE / 2;

            state <= STATE_IDLE;
        end
        else if (!active && state == STATE_IDLE)
            state <= STATE_WORKING;
        else if (active && state == STATE_WORKING) begin
            // Calculate the best positions
            if ((get_i == 4'b0 && get_j == 4'b0) ||
                black_score > black_best_score ||
                (black_score == black_best_score && random)) begin
                black_best_score <= black_score;
                black_best_i <= get_i;
                black_best_j <= get_j;
            end
            if ((get_i == 4'b0 && get_j == 4'b0) ||
                white_score > white_best_score ||
                (white_score == white_best_score && random)) begin
                white_best_score <= white_score;
                white_best_i <= get_i;
                white_best_j <= get_j;
            end

            // Move to the next position
            if (get_j == BOARD_SIZE - 1) begin
                if (get_i == BOARD_SIZE - 1) begin
                    get_i <= 4'b0;
                    get_j <= 4'b0;
                    state <= STATE_IDLE;
                end
                else begin
                    get_i <= get_i + 1'b1;
                    get_j <= 4'b0;
                end
            end
            else
                get_j <= get_j + 1'b1;
        end
    end

endmodule
```

-------- 胜利判断模块 win_checker --------

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------
// Checker to check if someone wins the game
//----------------------------------------------------------------------------
module win_checker(
    input wire clk,             // Clock
    input wire rst,             // Reset
    input wire clr,             // Clear
    input wire active,          // Active signal

    input wire [8:0] black_i,     // Row information
```

```
input wire [8:0] black_j,     // Column information
input wire [8:0] black_ij,    // Main diagonal information
input wire [8:0] black_ji,    // Counter diagonal information
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 [3:0] get_i,       // Current row considered
output reg [3:0] get_j,       // Current column considered
output reg is_win             // If someone wins the game
);

// Chessboard parameters
localparam BOARD_SIZE = 15;

// State parameters
localparam STATE_IDLE = 1'b0,
           STATE_WORKING = 1'b1;

reg state;    // Current state

// Pattern recognizers
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);

// FSM of the checker
// Every time the active signal comes, the checker will run once
always @ (posedge clk or negedge rst) begin
    if (!rst || clr) begin
        get_i <= 4'b0;
        get_j <= 4'b0;
        is_win <= 0;

        state <= STATE_IDLE;
    end
    else if (!active && state == STATE_IDLE)
        state <= STATE_WORKING;
    else if (active && state == STATE_WORKING) begin
        // Check if someone wins the game
        if (get_i == 4'b0 && get_j == 4'b0)
            is_win <= b0 | b1 | b2 | b3 | w0 | w1 | w2 | w3;
        else
            is_win <= is_win | b0 | b1 | b2 | b3 | w0 | w1 | w2 | w3;

        // Move to the next position
        if (get_j == BOARD_SIZE - 1) begin
            if (get_i == BOARD_SIZE - 1) begin
                get_i <= 4'b0;
                get_j <= 4'b0;
                state <= STATE_IDLE;
            end
```

```
            else begin
                get_i <= get_i + 1'b1;
                get_j <= 4'b0;
            end
        end
        else
            get_j <= get_j + 1'b1;
    end
end

endmodule
```

-------- 分数计算模块 score_calculator --------

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------
// Calculate the score of a given pattern
//----------------------------------------------------------------------------
module score_calculator(
    input wire [8:0] my,       // Pattern of my side
    input wire [8:0] op,       // Pattern of the opposite side
    output reg [12:0] score    // The score of the given pattern
    );

    wire [8:0] my_next;
    assign my_next = my | 9'b000010000;

    wire score2, score4, score5, score8, score15,
         score40, score70, score300, score2000;

    // Pattern recognizers
    pattern_stwo pattern2(my_next, op, score2);
    pattern_ftwo pattern4(my_next, op, score4);
    pattern_sthree pattern5(my_next, op, score5);
    pattern_two pattern8(my_next, op, score8);
    pattern_fthree pattern15(my_next, op, score15);
    pattern_three pattern40(my_next, op, score40);
    pattern_ffour pattern70(my_next, op, score70);
    pattern_four pattern300(my_next, op, score300);
    pattern_five pattern2000(my_next, score2000);

    always @ (*)
        if (my[4] || op[4])
            // Invalid pattern
            score = 0;
        else if (score2000)
            score = 2000;
        else if (score300)
            score = 300;
        else if (score70)
            score = 70;
        else if (score40)
            score = 40;
        else if (score15)
            score = 15;
        else if (score8)
            score = 8;
        else if (score5)
            score = 5;
```

```
        else if (score4)
            score = 4;
        else if (score2)
            score = 2;
        else
            score = 1;

endmodule
```

-------- 棋型判断模块 gobang_patterns --------

```
`timescale 1ns / 1ps
//----------------------------------------------------------------------------
// _ = empty | * = my chess | o = opponent's chess
//----------------------------------------------------------------------------

//----------------------------------------------------------------------------
// Recognize *****
//----------------------------------------------------------------------------
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

// Other recognizers omitted, see src folder for full code
```

2.3.2 棋盘数据存储模块

棋盘数据存储模块 gobang_datapath：存储棋盘的棋子信息，在写入信号为高电平时写入下棋的信息，并以组合逻辑的方式实现数据的送出。

送出的数据包括：读入行号，送出该行对应的棋子信息（用于显示）；读入坐标，送出该坐标八个方向上各四颗棋子的信息（用于棋型判断）。

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------
// Stores the data of the chessboard
//----------------------------------------------------------------------------
module gobang_datapath(
    input wire clk,                     // Clock
    input wire rst,                     // Reset
    input wire clr,                     // Clear

    input wire write,                   // Enable-write signal
```

```
input wire [3:0] write_i,           // The position to write information
input wire [3:0] write_j,
input wire write_color,             // The color to write in

input wire [3:0] logic_i,           // Row needed by logic
input wire [3:0] display_i,         // Row needed by display
input wire [3:0] consider_i,        // Row considered by strategy/checker
input wire [3:0] consider_j,        // Column considered by s/c

output wire [14:0] logic_row,       // Row information for logic
output wire [14:0] display_black,   // Row information for display
output wire [14:0] display_white,
output reg [8:0] black_i,           // Row information for s/c
output reg [8:0] black_j,           // Column information for s/c
output reg [8:0] black_ij,          // Main diagonal information for s/c
output reg [8:0] black_ji,          // Counter diagonal information for s/c
output reg [8:0] white_i,
output reg [8:0] white_j,
output reg [8:0] white_ij,
output reg [8:0] white_ji
);

// Side parameters
localparam BLACK = 1'b0,
           WHITE = 1'b1;

// Chessboard parameters
localparam BOARD_SIZE = 15;

// RAMs, store the information of the black/white chess
reg [14:0] board_black [14:0];
reg [14:0] board_white [14:0];

// Helper registers to fetch information from RAM
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;

// Generate the output
assign logic_row = board_black[logic_i] | board_white[logic_i];
assign display_black = board_black[display_i];
assign display_white = board_white[display_i];

always @ (negedge clk or negedge rst) begin
    if (!rst || clr) 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)
        if (write_color == BLACK)
            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

always @ (*) begin
    i = consider_i;
    j = consider_j;

    // Fetch the data of the rows from i-4 to i+4
    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

    // Write the data of each grid to the output
    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
```

2.3.3 PS/2 输入模块

PS/2 输入模块主要包含以下内容。

（1）PS/2 输入顶层模块 ps2_input：主要用于处理 PS/2 数据接收模块获得的按键数据，输出对应按键是否达到上升沿的信息；

（2）PS/2 数据接收模块 ps2_scan：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短码。

-------- PS/2 输入顶层模块 ps2_input --------

```
`timescale 1ns / 1ps

//-----------------------------------------------------------------------------
// Dealing with the input data of the PS2 keyboard
//-----------------------------------------------------------------------------
module ps2_input(
    input wire clk_slow,      // Slower clock for this module
    input wire clk_fast,      // Faster clock for the PS2 scanner
    input wire rst,           // Reset
    input wire ps2_clk,       // PS2 clock
    input wire ps2_data,      // PS2 data

    output wire key_up,       // If UP key is pressed
    output wire key_down,     // If DOWN key is pressed
    output wire key_left,     // If LEFT key is pressed
    output wire key_right,    // If RIGHT key is pressed
    output wire key_ok,       // If OK key is pressed
    output wire key_switch    // If SWITCH key is pressed
    );

    wire [8:0] crt_data;    // Input data of the PS2 keyboard

    // Key state recorders
    reg [1:0] key_up_state, key_down_state, key_left_state,
              key_right_state, key_ok_state, key_switch_state;
    // Only becomes true at the posedge of each key
    assign key_up = key_up_state[0] & ~key_up_state[1];
    assign key_down = key_down_state[0] & ~key_down_state[1];
    assign key_left = key_left_state[0] & ~key_left_state[1];
```

```
    assign key_right = key_right_state[0] & ~key_right_state[1];
    assign key_ok = key_ok_state[0] & ~key_ok_state[1];
    assign key_switch = key_switch_state[0] & ~key_switch_state[1];

    // PS2 keyboard scanner
    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 negedge rst)
        if (!rst) begin
            key_up_state <= 2'b0;
            key_down_state <= 2'b0;
            key_left_state <= 2'b0;
            key_right_state <= 2'b0;
            key_ok_state <= 2'b0;
            key_switch_state <= 2'b0;
        end
        else begin
            // Record the key state
            key_up_state <= {key_up_state[0], crt_data == 9'h175};
            key_down_state <= {key_down_state[0], crt_data == 9'h172};
            key_left_state <= {key_left_state[0], crt_data == 9'h16b};
            key_right_state <= {key_right_state[0], crt_data == 9'h174};
            key_ok_state <= {key_ok_state[0], crt_data == 9'h029};
            key_switch_state <= {key_switch_state[0], crt_data == 9'h014};
        end

endmodule
```

-------- PS/2 数据接收模块 ps2_scan --------

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------
// PS2 keyboard scanner
//----------------------------------------------------------------------------
module ps2_scan(
    input wire clk,             // Clock
    input wire rst,             // Reset
    input wire ps2_clk,         // PS2 clock
    input wire ps2_data,        // PS2 data

    output reg [8:0] crt_data    // Input data of the keyboard
    );

    reg [1:0] ps2_clk_state;    // PS2 clock recorder
    wire ps2_clk_neg;          // True at the negedge of the PS2 clock
    assign ps2_clk_neg = ~ps2_clk_state[0] & ps2_clk_state[1];

    // Registers for data reading
    reg [3:0] read_state;
    reg [7:0] read_data;

    // Registers for special signals
```

```
    reg is_f0, is_e0;

    // Record the PS2 clock
    always @ (posedge clk or negedge rst)
        if (!rst)
            ps2_clk_state <= 2'b0;
        else
            ps2_clk_state <= {ps2_clk_state[0], ps2_clk};

    always @ (posedge clk or negedge rst) begin
        if (!rst) begin
            read_state <= 4'b0;
            read_data <= 8'b0;

            is_f0 <= 1'b0;
            is_e0 <= 1'b0;
            crt_data <= 9'b0;
        end
        else if (ps2_clk_neg) begin
            // Reads in the data
            if (read_state > 4'b1001)
                read_state <= 4'b0;
            else begin
                if (read_state > 4'b0 && read_state < 4'b1001)
                    read_data[read_state - 1] <= ps2_data;
                read_state <= read_state + 1'b1;
            end
        end
        else if (read_state == 4'b1010 && |read_data) begin
            if (read_data == 8'hf0)
                is_f0 <= 1'b1;
            else if (read_data == 8'he0)
                is_e0 <= 1'b1;
            else
                if (is_f0) begin
                    // A key is released
                    is_f0 <= 1'b0;
                    is_e0 <= 1'b0;
                    crt_data <= 9'b0;
                end
                else if (is_e0) begin
                    is_e0 <= 1'b0;
                    crt_data <= {1'b1, read_data};
                end
                else
                    crt_data <= {1'b0, read_data};

            read_data <= 8'b0;
        end
    end

endmodule
```

2.3.4 VGA 显示模块

VGA 显示模块主要包含以下内容。

（1）VGA 显示顶层模块 vga_display：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；

（2）VGA 时序生成模块 vga_sync：生成 640×480@60Hz 的 VGA 显示时序；

（3）显示图案模块 display_pics：一些 ROM，储存着需要显示的一些图像信息。

-------- VGA 显示顶层模块 vga_display --------

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------
// Display the color for each pixel
//----------------------------------------------------------------------------
module vga_display(
    input wire clk,                    // Clock (25MHz)
    input wire rst,                    // Reset

    input wire [3:0] cursor_i,          // Cursor position
    input wire [3:0] cursor_j,
    input wire black_is_player,         // If black is not AI
    input wire white_is_player,         // If white is not AI
    input wire crt_player,              // Current player
    input wire game_running,            // If the game is running
    input wire [1:0] winner,            // Who wins the game

    input wire [14:0] display_black,    // Row information for display
    input wire [14:0] display_white,

    output wire [3:0] display_i,        // Row needed by display
    output wire sync_h,                 // VGA horizontal sync
    output wire sync_v,                 // VGA vertical sync
    output wire [3:0] r,                // VGA red component
    output wire [3:0] g,                // VGA green component
    output wire [3:0] b                 // VGA blue component
    );

    // Side parameters
    localparam BLACK = 1'b0,
               WHITE = 1'b1;

    // Chessboard display parameters
    localparam BOARD_SIZE = 15,
               GRID_SIZE = 23,
               GRID_X_BEGIN = 148,
               GRID_X_END = 492,
               GRID_Y_BEGIN = 68,
               GRID_Y_END = 412;

    // Side information display parameters
    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,
               SIDE_WHITE_Y_END = 296;
```

```
// Current player pointer display parameters
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;

// Title display parameters
localparam TITLE_X_BEGIN = 0,
           TITLE_X_END = 140,
           TITLE_Y_BEGIN = 62,
           TITLE_Y_END = 418;

// Instruction display parameters
localparam INS_X_BEGIN = 145,
           INS_X_END = 494,
           INS_Y_BEGIN = 424,
           INS_Y_END = 467;

// Result display parameters
localparam RES_X_BEGIN = 187,
           RES_X_END = 452,
           RES_Y_BEGIN = 20,
           RES_Y_END = 47;

// Author info display parameters
localparam AUTHOR_X_BEGIN = 510,
           AUTHOR_X_END = 634,
           AUTHOR_Y_BEGIN = 455,
           AUTHOR_Y_END = 476;



// Current display color
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;

// VGA control signal generator
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)
    );

// Chessboard display registers
reg [3:0] row, col;
```

```
integer delta_x, delta_y;
assign display_i = row < BOARD_SIZE ? row : 4'b0;

// Patterns needed to be displayed
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, black_ai_data,
            white_player_data, white_ai_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);
pic_side_ai black_ai(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_ai_data),
            white_ai(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_ai_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, ins_ai_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);
pic_ins_ai ins_ai(x >= INS_X_BEGIN && x <= INS_X_END &&
                  y >= INS_Y_BEGIN && y <= INS_Y_END,
                  y - INS_Y_BEGIN, ins_ai_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);

// Calculate the current row and column
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;

    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

// Calculate the color
always @ (posedge clk) begin
    if (x >= GRID_X_BEGIN && x <= GRID_X_END &&
        y >= GRID_Y_BEGIN && y <= GRID_Y_END) begin
        // Draw the chessboard
        if (display_black[col] &&
            chess_piece_data[delta_x + GRID_SIZE/2])
```

```
            // Draw a black chess
            rgb <= 12'h000;
        else if (display_white[col] &&
                 chess_piece_data[delta_x + GRID_SIZE/2])
            // Draw a white chess
            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)))
            // Draw a red square as a cursor
            rgb <= 12'hf00;
        else if (delta_x == 0 || delta_y == 0)
            // Draw the light border of a grid
            rgb <= 12'hda6;
        else if (delta_x == 1 || delta_y == 1)
            // Draw the dark border of a grid
            rgb <= 12'h751;
        else
            rgb <= 12'hc81;
    end
    else if (x >= CRT_BLACK_X_BEGIN && x <= CRT_BLACK_X_END &&
             y >= CRT_BLACK_Y_BEGIN && y <= CRT_BLACK_Y_END) begin
        // Draw the current player pointer for black side
        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
        // Draw the current player pointer for white side
        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
        // Draw who plays the black side
        if (black_is_player)
            rgb <= black_player_data[SIDE_BLACK_X_END - x] ?
                   12'h000 : 12'hc81;
        else
            rgb <= black_ai_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
        // Draw who plays the white side
        if (white_is_player)
            rgb <= white_player_data[SIDE_WHITE_X_END - x] ?
                   12'hfff : 12'hc81;
        else
            rgb <= white_ai_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
        // Draw instructions
        if (!game_running)
            rgb <= ins_start_data[INS_X_END - x] ? 12'h000 : 12'hc81;
        else if ((black_is_player && crt_player == BLACK) ||
                 (white_is_player && crt_player == WHITE))
            rgb <= ins_player_data[INS_X_END - x] ? 12'h000 : 12'hc81;
        else
```

```
                rgb <= ins_ai_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
            // Draw the result
            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) begin
            // Draw the title
            rgb <= title_data[TITLE_X_END - x] ? 12'hfff : 12'hc81;
        end
        else if (x >= AUTHOR_X_BEGIN && x <= AUTHOR_X_END &&
                 y >= AUTHOR_Y_BEGIN && y <= AUTHOR_Y_END) begin
            // Draw the author info
            rgb <= author_info_data[AUTHOR_X_END - x] ? 12'h000 : 12'hc81;
        end
        else
            // Draw the background
            rgb <= 12'hc81;
    end

endmodule
```

-------- VGA 时序生成模块 vga_sync --------

```
`timescale 1ns / 1ps

//------------------------------------------------------------------------------
// Generate the VGA control signal
//------------------------------------------------------------------------------
module vga_sync(
    input wire clk,          // Clock (25MHz)
    input wire rst,          // Reset
    output wire sync_h,      // VGA horizontal sync
    output wire sync_v,      // VGA vertical sync
    output wire video_on,    // True when in the display area
    output reg [9:0] x,      // X coordinate
    output reg [9:0] y       // Y coordinate
    );

    assign sync_h = ~(x > 655 && x < 752);
    assign sync_v = ~(y > 489 && y < 492);
    assign video_on = (x < 640 && y < 480);

    always @ (posedge clk or negedge 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
```

-------- 显示图案模块 display_pics --------

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------------
// ROMs which record some patterns needed to be displayed
//----------------------------------------------------------------------------------

//----------------------------------------------------------------------------------
// Picture of a chess piece
//----------------------------------------------------------------------------------
module pic_chess_piece(
    input wire en,
    input wire [31:0] i,
    output reg [22:0] ret
    );

    always @ (en or i)
        if (!en)
            ret = 23'b0;
        else
            case (i)
            32'd00: ret = 23'b00000000000000000000000;
            32'd01: ret = 23'b00000000000000000000000;
            32'd02: ret = 23'b00000000011111110000000;
            32'd03: ret = 23'b00000011111111111000000;
            32'd04: ret = 23'b00000111111111111100000;
            32'd05: ret = 23'b00001111111111111110000;
            32'd06: ret = 23'b00011111111111111111000;
            32'd07: ret = 23'b00011111111111111111000;
            32'd08: ret = 23'b00111111111111111111100;
            32'd09: ret = 23'b00111111111111111111100;
            32'd10: ret = 23'b00111111111111111111100;
            32'd11: ret = 23'b00111111111111111111100;
            32'd12: ret = 23'b00111111111111111111100;
            32'd13: ret = 23'b00111111111111111111100;
            32'd14: ret = 23'b00111111111111111111100;
            32'd15: ret = 23'b00011111111111111111000;
            32'd16: ret = 23'b00011111111111111111000;
            32'd17: ret = 23'b00001111111111111110000;
            32'd18: ret = 23'b00000111111111111100000;
            32'd19: ret = 23'b00000011111111111000000;
            32'd20: ret = 23'b00000000011111110000000;
            32'd21: ret = 23'b00000000000000000000000;
            32'd22: ret = 23'b00000000000000000000000;
            default: ret = 23'b0;
            endcase

endmodule

// Other pictures omitted, see src folder for full code
```

2.3.5 辅助模块

辅助模块主要包含以下内容。

（1）时钟分频模块 clk_divider：将 100MHz 的时钟进行分频；

（2）随机数模块 random_generator：产生随机数。

-------- 时钟分频模块 clk_divider --------

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------
// Clock division module
//----------------------------------------------------------------------------
module clk_divider(
    input wire clk,             // Clock
    input wire rst,             // Reset
    output reg [31:0] clk_div    // Division result
    );

    always @ (posedge clk or negedge rst)
        if (!rst)
            clk_div <= 32'b0;
        else
            clk_div <= clk_div + 1;

endmodule
```

-------- 随机数模块 random_generator --------

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------
// Generate a random number
//----------------------------------------------------------------------------
module random_generator(
    input wire clk,              // Clock
    input wire rst,              // Reset
    input wire load,             // Load signal
    input wire [31:0] seed,       // Random seed
    output reg [31:0] rand_num    // A random number
    );

    always @ (posedge clk or negedge rst)
        if (!rst)
            rand_num <= 32'b0;
        else if (load)
            // Load the random seed
            rand_num <= seed;
        else
            rand_num <= {rand_num[30:0],
                         rand_num[31] ^ rand_num[29] ^ rand_num[28] ^
                         rand_num[27] ^ rand_num[23] ^ rand_num[20] ^
                         rand_num[19] ^ rand_num[17] ^ rand_num[15] ^
                         rand_num[14] ^ rand_num[12] ^ rand_num[11] ^
                         rand_num[9]  ^ rand_num[4]  ^ rand_num[3]  ^
                         rand_num[2]};
 endmodule
```

2.3.6 顶层模块

顶层模块 gobang_top 主要起着胶合各个模块的作用，并传递外界的输入输出信号。

```
`timescale 1ns / 1ps

//----------------------------------------------------------------------------
// Top module
//----------------------------------------------------------------------------
module gobang_top(
    input wire clk,         // Clock (100MHz)
    input wire rst,         // Reset button, 0 = pressed, 1 = released
    input wire ps2_clk,     // PS2 clock
    input wire ps2_data,    // PS2 data
    output wire sync_h,     // VGA horizontal sync
    output wire sync_v,     // VGA vertical sync
    output wire [3:0] r,    // VGA red component
    output wire [3:0] g,    // VGA green component
    output wire [3:0] b,    // VGA blue component
    output wire buz         // Arduino buzzer
    );

    wire [3:0] consider_i, consider_j, cursor_i, cursor_j;
    wire data_clr, data_write;
    wire black_is_player, white_is_player, crt_player, game_running;
    wire [1:0] winner;

    wire [8:0] black_i, black_j, black_ij, black_ji,
               white_i, white_j, white_ij, white_ji;
    wire [14:0] logic_row, display_black, display_white;

    wire key_up, key_down, key_left, key_right, key_ok, key_switch;

    wire [3:0] display_i;

    wire [31:0] rand_num;
    wire [31:0] clk_div;

    // Turn off the arduino buzzer
    assign buz = 1'b1;

    gobang_logic
        game_logic(
            .clk_slow(clk_div[16]),
            .clk_fast(clk_div[6]),
            .rst(rst),
            .random(rand_num[0]),
            .key_up(key_up),
            .key_down(key_down),
            .key_left(key_left),
            .key_right(key_right),
            .key_ok(key_ok),
            .key_switch(key_switch),
            .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),
        .consider_i(consider_i),
        .consider_j(consider_j),
        .data_clr(data_clr),
        .data_write(data_write),
        .cursor_i(cursor_i),
        .cursor_j(cursor_j),
        .black_is_player(black_is_player),
        .white_is_player(white_is_player),
        .crt_player(crt_player),
        .game_running(game_running),
        .winner(winner)
    );

gobang_datapath
    data(
        .clk(clk_div[16]),
        .rst(rst),
        .clr(data_clr),
        .write(data_write),
        .write_i(cursor_i),
        .write_j(cursor_j),
        .write_color(crt_player),
        .logic_i(cursor_i),
        .display_i(display_i),
        .consider_i(consider_i),
        .consider_j(consider_j),
        .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),
        .white_ji(white_ji)
    );

ps2_input
    keyboard(
        .clk_slow(clk_div[16]),
        .clk_fast(clk_div[6]),
        .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),
        .key_switch(key_switch)
    );

vga_display
    display(
```

```
            .clk(clk_div[1]),
            .rst(rst),
            .cursor_i(cursor_i),
            .cursor_j(cursor_j),
            .black_is_player(black_is_player),
            .white_is_player(white_is_player),
            .crt_player(crt_player),
            .game_running(game_running),
            .winner(winner),
            .display_black(display_black),
            .display_white(display_white),
            .display_i(display_i),
            .sync_h(sync_h),
            .sync_v(sync_v),
            .r(r),
            .g(g),
            .b(b)
        );

    random_generator
        rand(
            .clk(clk_div[6]),
            .rst(rst),
            .load(key_ok),
            .seed(clk_div),
            .rand_num(rand_num)
        );

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

endmodule
```

第三章 五子棋游戏设计实现

3.1 实现方法

以下将较为详细地介绍各个模块的实现方法，以及实现过程中难点的解决方案。

3.1.1 游戏逻辑模块

3.1.1.1 逻辑顶层模块

逻辑顶层模块（gobang_logic）主要由一个状态机构成，用于维护游戏当前的状态。作为顶层模块，它还包含了 AI 策略模块与胜利判断模块。

逻辑顶层模块的状态机共有 8 个状态。状态机初始化后，首先进入游戏开始前的状态（STATE_IDLE）。此时若键盘上的左 Ctrl 键被按下，则会切换黑白方的玩家与 AI；若键盘上的空格键被按下，则激活送至棋盘数据存储模块的棋盘清空信号，将游戏运行信号 game_running 置为高电平，并进入一方行动状态（STATE_MOVE）。

进入一方行动状态（STATE_MOVE）后，棋盘清空信号关闭。若当前一方是玩家行动，则根据键盘的上下左右按键移动光标，按下空格键后进入尝试下棋状态（STATE_PUT_CHESS）；若当前一方是 AI 行动，则激活 AI 判断信号，并进入等待状态（STATE_WAIT）。进入等待状态，不仅是为了防止 AI 行动过快造成玩家体验不佳，也能给 AI 的计算以充足的时间。

进入等待状态（STATE_WAIT）后，等待计数器 wait_count 开始计数，每一个时钟周期完成后计数器+1。若计数器达到了等待时间 WAIT_TIME，则转入 AI 决定状态（STATE_DECIDE）。

进入 AI 决定状态（STATE_DECIDE）后，AI 激活信号关闭。该状态对 AI 算出的双方最高可能得分进行判断，若己方得分较高，则将光标移动到己方对应的位置尝试得分；否则将光标移动到对方对应的位置，阻止对方得分。光标移动结束后，同样进入尝试下棋状态（STATE_PUT_CHESS）。

进入尝试下棋状态（STATE_PUT_CHESS）后，模块通过棋盘数据存储模块传来的数据，判断当前光标指示的格点是否已经有棋子占据。如果无棋子占据，则将数据写入信号置高电平，步数计数器+1，并进入下棋完成状态（STATE_PUT_END），否则退回一方行动状态（STATE_MOVE）。由于 AI 的判断中已经过滤了已有棋子的格点，所以退回一方行动状态时只可能是玩家在行动。

进入下棋完成状态（STATE_PUT_END）后，数据写入信号被置为低电平，并激活胜利判断信号，进入棋局检查状态（STATE_CHECK）。

进入棋局检查状态（STATE_CHECK）后，胜利判断信号关闭。若胜利条件检查模块输出已有一方胜利的信号，或当前步数计数器已达到棋盘大小（15×15），则进入游戏结束状态（STATE_GAME_END）。否则交换当前玩家，回到一方行动状态（STATE_MOVE）。

进入游戏结束状态（STATE_GAME_END）后，若已知当前有一方玩家胜利，则将胜利玩

家信号 winner 表示为当前玩家。否则说明当前棋局平局，将胜利玩家信号 winner 置为平局状态，回到游戏开始前状态（STATE_IDLE）。

3.1.1.2 AI 策略模块

AI 策略模块（gobang_strategy）负责在激活信号到来时，枚举棋盘上所有格点，并计算对于每个格点己方与对方的分数，得出双方的最高可能得分与对应位置。模块内部主要包含了用于计算格点分数的分数计算模块（score_calculator）。

为了实现每次激活信号到来时，模块运行一次的效果，我在模块里使用了一个只有两个状态的简单状态机。状态机初始化后，首先进入空白状态（STATE_IDLE）。此时若激活信号为低电平，则进入计算状态（STATE_WORKING）。不过，进入计算状态后，只有当激活信号为高电平时，模块才会执行计算电路，并在计算完成后回到空白状态（STATE_IDLE）。这样就能实现“一次激活，一次运行”了。

但是，正如在第 2.2 节中提出的那样，如果我们在一个时钟周期内，就将整个棋盘上每个格点的得分全部计算出来，那么就需要大量的计算模块，将会大量消耗电路资源与综合时的计算机资源。考虑到本课程设计对时序的要求并不高，逻辑顶层模块的时钟频率非常低，我在实现过程中就将 15×15 个格点的计算拆分至 15×15 个时钟周期完成。这样，每个周期就只需要计算一个格点的分数，也就只需要一个计算模块，大大节省了电路资源。不过，这种方法降低了 AI 策略模块的计算速度。为了能在逻辑顶层模块的时钟完成一个周期的时间内将各个格点的分数全部计算完成，送至 AI 策略模块的时钟频率就应该比逻辑顶层模块的时钟频率高很多。经过平衡，我将 AI 策略模块的时钟频率设为约 1MHz，既能让模块计算及时结束，又不会因时钟频率过高而出现电路来不及计算的情况。

3.1.1.3 胜利判断模块

胜利判断模块（win_checker）负责在激活信号到来时，枚举棋盘上所有格点，并判断格点周围的棋型是否满足五子相连。

可以看到，胜利判断模块的功能与 AI 策略模块的功能有些相似，两个模块的实现方法也基本相同。胜利判断模块也使用了一个只有两个状态的简单状态机实现“一次激活，一次运行”的效果，同样也将所有格点的计算拆分至多个时钟周期的完成，模块时钟频率同样也为约 1MHz。只不过，由于两个模块计算目的不同，所以胜利判断模块内部包含的不是分数计算模块，而是用于判断五子相连的棋型判断模块。

3.1.1.4 分数计算模块

分数计算模块（score_calculator）的实现就比较简单了。这个模块是一个组合逻辑电路，内部包含了所有棋型的棋型判断模块（gobang_patterns）。每个棋型判断模块具有不同的分数与优先级，分数计算模块就根据判断模块送出的结果，输出不同的分数。

3.1.1.5 棋型判断模块

棋型判断模块（gobang_patterns）的实现也非常简单。这个模块同样是一个组合逻辑电路，根据自己需要判断的棋型与输入的棋子信息输出判断结果。我在写代码的时候使用了较

为麻烦的判断方法，目的是便于综合器的综合。

3.1.2 棋盘数据存储模块

棋盘数据存储模块（gobang_datapath）使用了两个数据深度与宽度均为 15 的 RAM 分别存储棋盘上黑子和白子的信息。模块的时钟与逻辑顶层模块同步，但为了保证数据写入时信号的稳定性，数据存储模块在时钟下降沿时才写入数据（而逻辑顶层模块在时钟上升沿时进行逻辑处理）。

为了满足逻辑顶层模块与 VGA 显示顶层模块的数据读取需要，棋盘数据存储模块以组合逻辑的形式送出数据。从数据存储模块读取的数据类型有两种：给出棋盘的行号，存储模块输出这一行所有黑子与白子的信息；给出棋盘上的坐标，存储模块输出这个坐标周期八个方向各四个格点的所有黑子与白子信息。这两种数据的读取在不同的端口进行，便于各个模块同时对数据存储模块发送读取信号而不互相冲突。

同样，在实现数据读取时，我使用了较为麻烦的方式写代码，也是为了便于综合器对硬件描述语言进行综合。

3.1.3 PS/2 输入模块

3.1.3.1 PS/2 输入顶层模块

PS/2 输入顶层模块（ps2_input）中包含了 PS/2 数据接收模块（ps2_scan），接收其传入的按键状态代码。由于本课程设计不希望在键盘按键按下不放时进行连续操作，所以需要在 PS/2 输入顶层模块中进行按键上升沿信息的采集。

获取按键上升沿信息的方法非常容易。对于需要采集信息的按键，我使用了一个位宽为 2 的寄存器。其中，第 0 位表示当前按键的状态，第 1 位表示上一个时钟周期按键的状态。显然，如果寄存器的值与 2'b01 相等，说明按键上升沿到来了。显然，模块的时钟频率应与逻辑顶层模块同步。

3.1.3.2 PS/2 数据接收模块

PS/2 输入顶层模块（ps2_scan）根据 PS/2 协议读入 PS/2 接口传入的数据，并转换为按键状态代码。由于 PS/2 接口的数据是串行传入的，所以该模块内部也包含了一个简单的状态机，表明当前接收到了数据的第几位。简单起见，本模块没有处理数据中的奇偶校验位，也不考虑键盘上的两个特殊键 Pause 与 Print Screen。

模块中使用了与 PS/2 输入顶层模块类似的方法读取 PS/2 时钟的下降沿。在模块的每个时钟周期内，若 PS/2 时钟到达的下降沿，则将数据存入一个寄存器中，并使状态机的状态向前推进。在数据的奇偶校验位到来后，模块会对寄存器内储存的数据进行判断并清空寄存器。如果存储的是关于断码的特殊数据 8'hf0 或关于第二类按键的特殊数据 8'he0，则将对应的标记寄存器设为 1，否则根据存储的数据与特殊数据标记输出相应的按键状态代码。在数据的停止位到来后，状态机重置。

需要注意的是，由于 PS/2 时钟的频率约为 10KHz，而 PS/2 输入顶层模块的时钟约为 1000Hz，所以还需要给 PS/2 数据接收模块另外传入一个频率较高的时钟。在实现过程中，

我选择了时钟分频后一个约 1MHz 的信号。

3.1.4 VGA 显示模块

3.1.4.1 VGA 显示顶层模块

VGA 显示顶层模块（vga_display）内部包含了 VGA 时序生成模块（vga_sync）与显示图案模块（display_pics）。该模块读取时序生成模块输出的坐标信息，计算当前需要输出的颜色值。

在实现的过程中，我将屏幕划分成了几个不同的矩形区域，在坐标进入每个矩形区域的范围后进行不同图形的显示工作。在坐标进入棋盘区域的范围后，模块还以组合逻辑的方式算出当前屏幕坐标对应的棋盘坐标，并根据逻辑顶层模块与棋盘数据存储模块传来的信息进行显示。在坐标进入其它图形的显示范围后，模块也会根据从显示图案模块的各个 ROM 中读取的图像信息进行显示。

同样，为了便于综合，我实现屏幕坐标转棋盘坐标的方式也较为麻烦，是通过较多判断语句进行的实现。

3.1.4.2 VGA 时序生成模块

VGA 时序生成模块（vga_sync）根据 VGA 协议生成列同步信号与行同步信号，同时输出当前显示的坐标。640×480@60Hz 的显示模式所需的各种数据不难通过查表获得。

3.1.4.3 显示图案模块

显示图案模块（display_pics）以 ROM 的形式存储了屏幕上需要显示的各种图案。在实现过程中，图案的 01 信息是通过 C++程序从图片转换而来的。

3.2 实现过程

在实现课程设计的过程中，我的实现步骤如下。

（1）首先实现了不含玩家控制部分的逻辑模块、棋盘数据存储模块以及时钟分频模块，可以在模拟软件中观察两个 AI 对局的过程。不过由于没有完成随机模块，所以只能观察固定的对局；

（2）实现了较为简陋的 VGA 显示模块，可以用简陋的图形在屏幕上表示两个 AI 固定的对局的过程。

（3）实现了 PS/2 输入模块，完成了逻辑模块的玩家控制部分，同时完成了随机数生成模块，可以用简陋的图形显示，完成课程设计的所有功能（包括玩家与 AI、玩家之间、AI 之间随机的对局）；

（4）给 VGA 显示模块添加了各种图形，可以用较为友好的界面显示，完成课程设计的所有功能。

由于本课程设计使用硬件描述语言实现，在实现的过程中需要随时思考当前功能的硬件实现方式以及电路优化方式，不可用过于行为化的软件思维进行实现，否则可能会使得电路资源大量消耗，或者综合失败。我在第一版程序的实现过程中就犯了这样的错误，最后导致

程序综合失败。

综合完成后，顶层模块的 RTL 逻辑图如下。

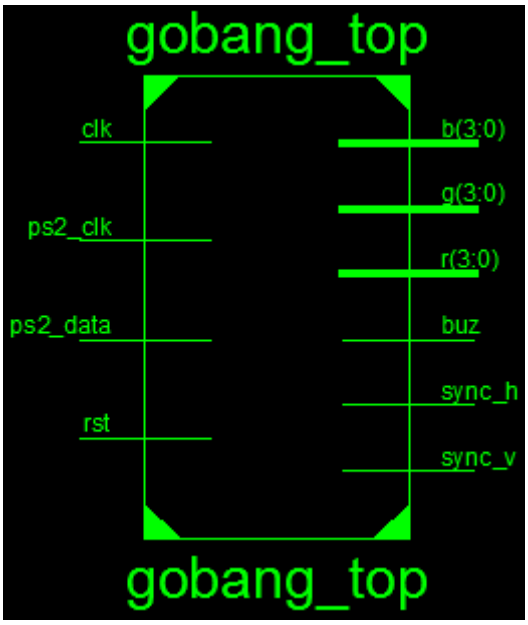

图 9　顶层模块 RTL 逻辑图

其中，clk 表示输入的 100MHz 时钟信号，rst 表示重置按钮信号，ps2_clk 与 ps2_data 分别是 PS/2 键盘的时钟与数据；sync_h 与 sync_v 是 VGA 接口的列同步信号与行同步信号，r、g、b 是颜色信号，buz 用于关闭 Arduino 子板上的蜂鸣器。

完成各个模块的设计与实现之后，需要考虑模块的仿真验证问题。由于时钟分频模块与随机数生成模块代码简单，无需验证。VGA 显示模块由于数据验算复杂，但物理验证较为容易，所以直接进行物理验证。剩下的棋盘数据存储模块、逻辑模块与 PS/2 输入模块则进行仿真验证。

3.3　仿真与调试

3.3.1　棋盘数据存储模块的仿真与调试

棋盘数据存储模块调试的重点主要在于棋盘边界数据的读写，测试在读取的部分数据可能超出棋盘边界的情况下，模块能否输出正确数据。仿真关键代码与仿真结果如下。

```
initial begin
    // Initialize Inputs
    clk = 0;
    rst = 0;
    clr = 0;
    write = 0;
    write_i = 0;
    write_j = 0;
    write_color = 0;
    logic_i = 0;
    display_i = 0;
    consider_i = 0;
    consider_j = 0;
```

```
    // Put a black chess on (0, 0)
    #8; rst = 1;
    write_i = 0; write_j = 0; write_color = 0;
    write = 1;
    #5; write = 0;

    // Put a white chess on (0, 1)
    #5; write_i = 0; write_j = 1; write_color = 1;
    write = 1;
    #5; write = 0;

    // Put a white chess on (1, 0)
    #5; write_i = 1; write_j = 0; write_color = 1;
    write = 1;
    #5; write = 0;

    // Put a black chess on (1, 1)
    #5; write_i = 1; write_j = 1; write_color = 0;
    write = 1;
    #5; write = 0;

    // Check row 1 and position (1, 1)
    #5; logic_i = 1; display_i = 1;
    consider_i = 1; consider_j = 1;
end

always begin
    #5;
    clk = ~clk;
end
```

图 10　棋盘数据存储模块仿真结果图

图中 write 是写入信号，write_i 与 write_j 是写入的坐标，write_color 是写入的棋子颜色；logic_i 是逻辑顶层模块当前需要的行号（用于判断某个位置是否已经有棋子），display_i 是 VGA 显示模块当前需要的行号（用于棋子的显示），consider_i 与 consider_j 是 AI 策略模块与胜利判断模块当前需要的坐标（用于获得坐标周围八个方向各四个格点的信息）；logic_row 表示 logic_i 对应的一行各个格点是否有棋子，display_black 与 display_white 表示 display_i 对应的一行各个格点是否有黑子或白子，剩下的 black_i 与 black_j 等等是 consider_i 与 consider_j 周围八个方向各四个格点的黑子或白子信息。

仿真代码在(0, 0)写入黑子，(0, 1)写入白子，(1, 0)写入白子，(1, 1)写入黑子。之所以在棋盘的左上角写入棋子，就是为了检验模块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。

3.3.2 游戏主要逻辑的仿真与调试

游戏主要逻辑的仿真与调试是对逻辑顶层模块、AI 策略模块、胜利判断模块等逻辑相关模块的综合仿真。由于本课程设计包含 AI 对局的功能，所以只需要在仿真软件中观察两个 AI 下棋位置的记录，就能很方便地检验逻辑部分是否正常运行。

由于逻辑部分需要与棋盘数据存储模块一起运行，所以在调试过程中，我制作了一个临时的顶层模块进行连接。以下仿真代码的关键部分针对的是临时的顶层模块，对整个课程设计实现完成后的顶层模块无法适用。

```
initial begin
    clk_slow = 0;
    clk_fast = 0;
    rst = 0;
    random = 0;
    key_ok = 0;

    // Press OK key to start game
    #700; rst = 1;
    key_ok = 1;

    #500; key_ok = 0;
end

always begin
    #1000;
    clk_slow = ~clk_slow;
end

always begin
    #1;
    clk_fast = ~clk_fast;
end
```

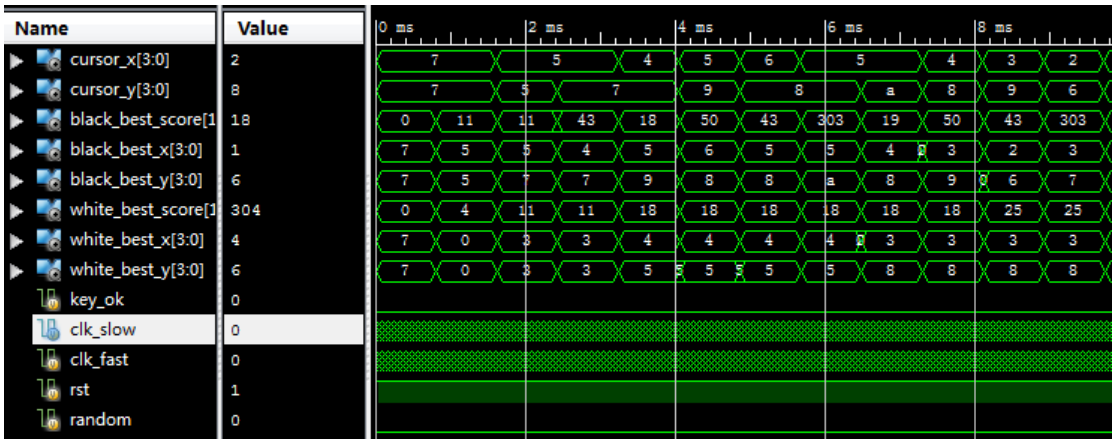

图 11　游戏主要逻辑仿真结果图 1

图中 cursor_x 与 cursor_y 代表光标位置，也就是 AI 下棋的位置；black_best_score 代表黑方下一步最高可能得分，black_best_x 与 black_best_y 代表最高得分对应的下棋位置。白方类似数据同理。由于这张仿真结果是课程设计早期的仿真结果，所以图中仍以 x 和 y 代表行坐标和列坐标，与最终代码里用 i 和 j 代表行坐标与列坐标略有不同，但不影响仿真结果。

进行仿真时，我使用了简单的五子棋软件观察两个 AI 的棋局，并验算双方的最高可能得分。

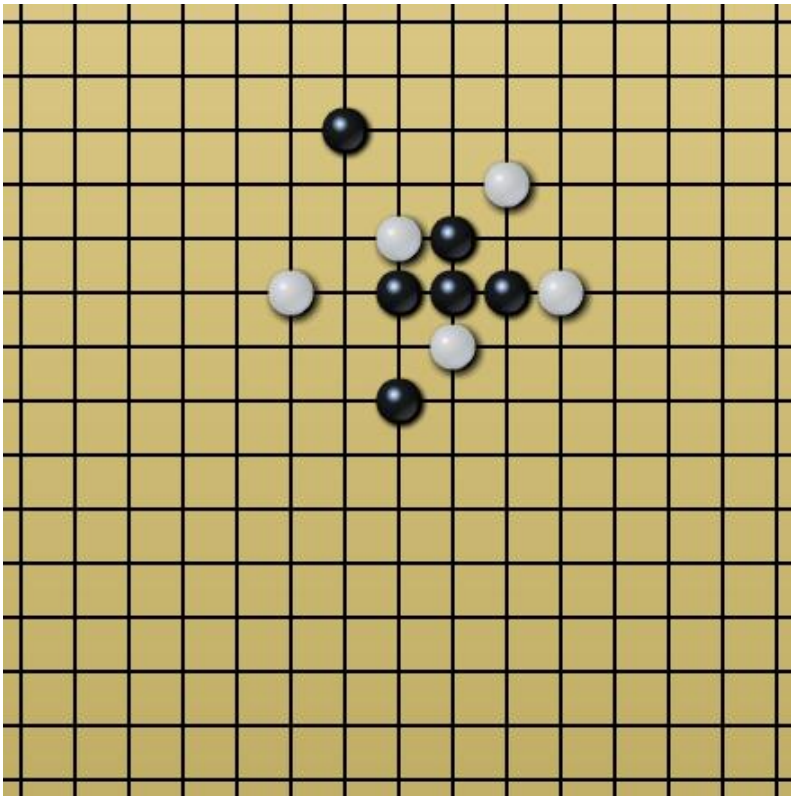

图 12　游戏主要逻辑仿真棋局前期

例如，上图就是图 11 代表的棋局，当前为白方行棋。从图 11 的最后可以看到，如果下一步黑方将棋下在(3, 7)的位置，就能获得最高可能得分 303 分（活四 300 分 + 其它三个方向无棋型 3 分），而白方将棋下在(3, 8)的位置，只能获得最高可能得分 25 分（三方向活二 24 分 + 一个方向无棋型 1 分），可以看到仿真的数值均运算正确。

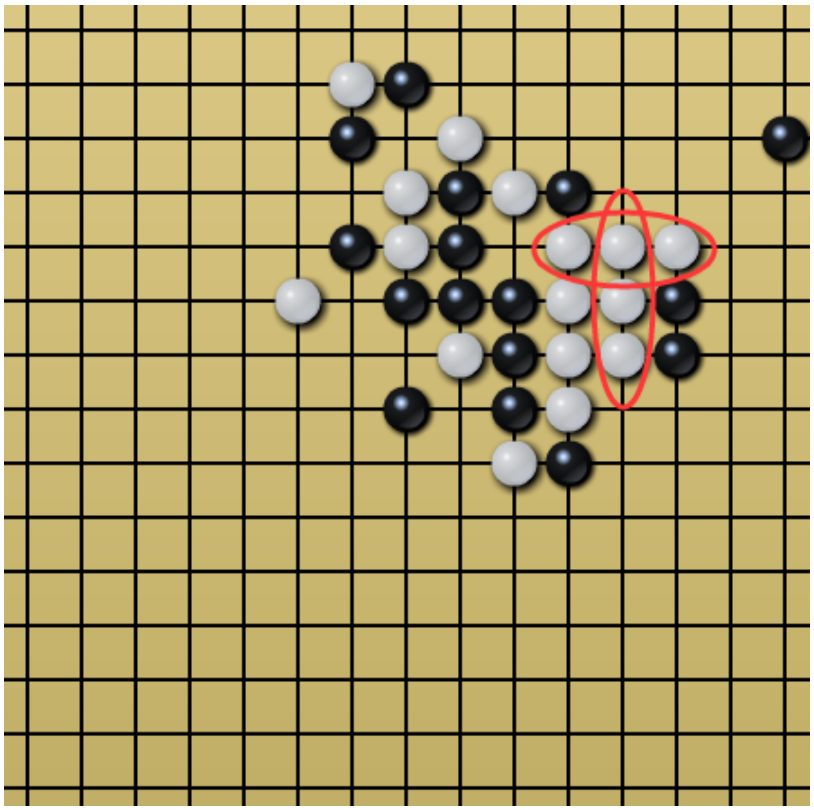

图 13　游戏主要逻辑仿真棋局后期

仿真过程中也能看到这个 AI 策略的缺陷。由于我们使用的是一个“只顾眼前”的贪心策略，很容易被对手形成两个活三棋型（图中红圈）而输棋。

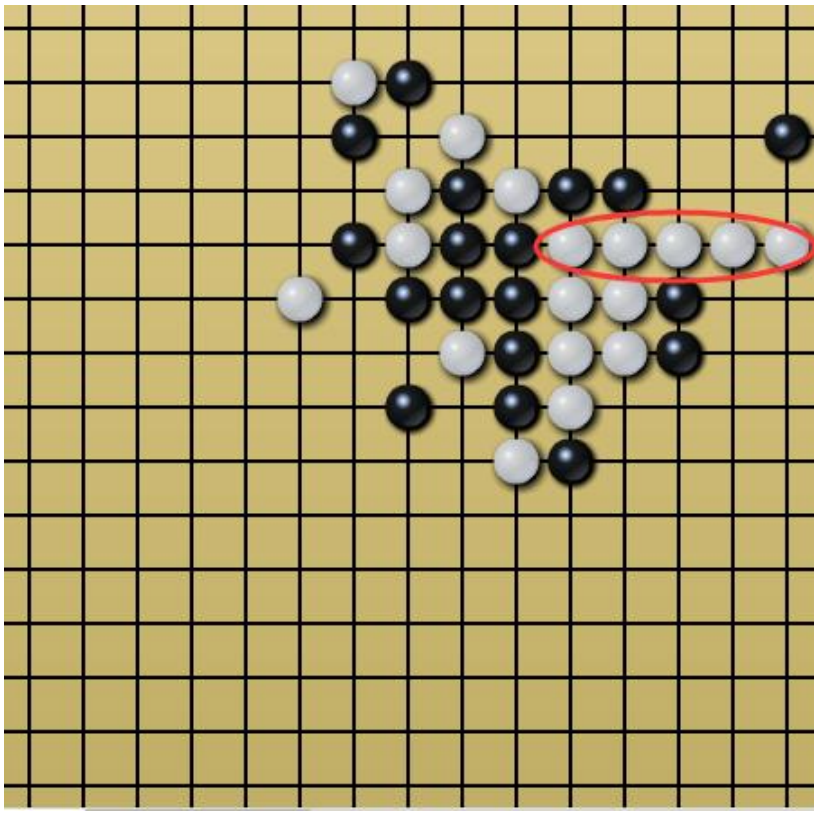

图 14　游戏主要逻辑仿真棋局结束

在仿真过程中，我也发现了代码里的小错误。仿真棋局本应像上图一样以白方在(4, e)处落子胜利而结束，但仿真图形里的落子坐标仍然在继续变化。经过检查，我发现了胜利判断模块中的代码错误，经过修改后，棋局可以正确结束，如下图所示。

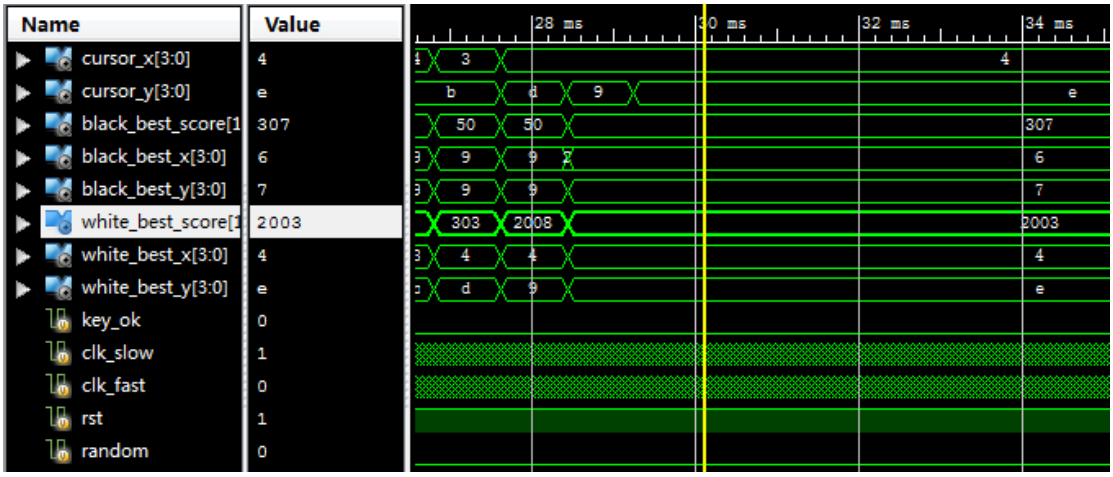

图 15　游戏主要逻辑仿真结果图 2

3.3.3　PS/2 输入模块的仿真与调试

PS/2 输入模块的仿真与调试重点在于串行输入数据的识别，特别是特殊数据 8'hf0 与 8'he0 的识别，以及观察模块能否正确识别多次按下又放开相同按键。仿真代码关键部分与仿真结果如下。

```
localparam [7:0] SPACE = 8'h29, UP = 8'h75;
localparam [7:0] E0 = 8'he0, F0 = 8'hf0;

task send_data;
    input [7:0] data;
    reg [3:0] i;

    begin
        for (i=0;i<11;i=i+1) begin
            #30; ps2_clk = 1;
            if (i > 0 && i < 9)
                ps2_data = data[i-1];
            #30; ps2_clk = 0;
        end
        #30; ps2_clk = 1;
    end
endtask

initial begin
    clk_slow = 0;
    clk_fast = 0;
    rst = 1;
    ps2_clk = 1;
    ps2_data = 0;

    #50; rst = 0;
    #30; rst = 1;

    // Press UP ARROW (e0 + 75)
    #100; send_data(E0); send_data(UP);
```

```
    // Release UP ARROW (e0 + f0 + 75)
    #100; send_data(E0); send_data(F0); send_data(UP);

    // Press SPACE (29)
    #100; send_data(SPACE);

    // Release SPACE (f0 + 29)
    #100; send_data(F0); send_data(SPACE);

    // Press SPACE (29)
    #100; send_data(SPACE);

    // Release SPACE (f0 + 29)
    #100; send_data(F0); send_data(SPACE);
end

always begin
    #70;
    clk_slow = ~clk_slow;
end

always begin
    #5;
    clk_fast = ~clk_fast;
end
```

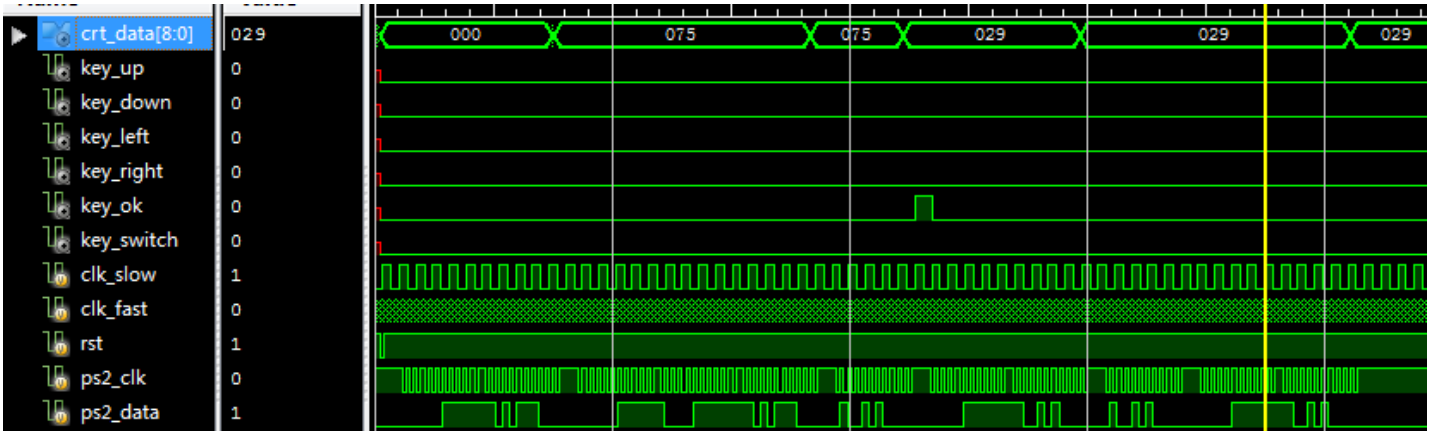

图 16　PS/2 输入模块初次仿真结果图 1

在仿真代码里，我进行了以下模拟：按下方向键上键（通码 e0 + 75），放开方向键上键（断码 e0 + f0 + 75），按下空格键（通码 29），放开空格键（断码 f0 + 29），再次按下空格键，再次放开空格键。key_up 代表方向键上键的按键上升沿，key_ok 代表空格键的按键上升沿，crt_data 代表 PS/2 数据接收模块接收到的串行数据。

从图 16 可以看到，初次仿真的结果并不符合我们的预期。在按下方向键上键之后，PS/2 数据接收模块只接收到了通码的后半部分 75，前半部分 e0 似乎没有在 crt_data 内体现，key_up 信号也没有变为高电平。放开方向键上键后，crt_data 也没有变回 0。之后第一次按下空格键虽然使得 key_ok 信号正确地变成了高电平，但是放开空格键和第二次按下空格键的结果都不符合我们的预期。

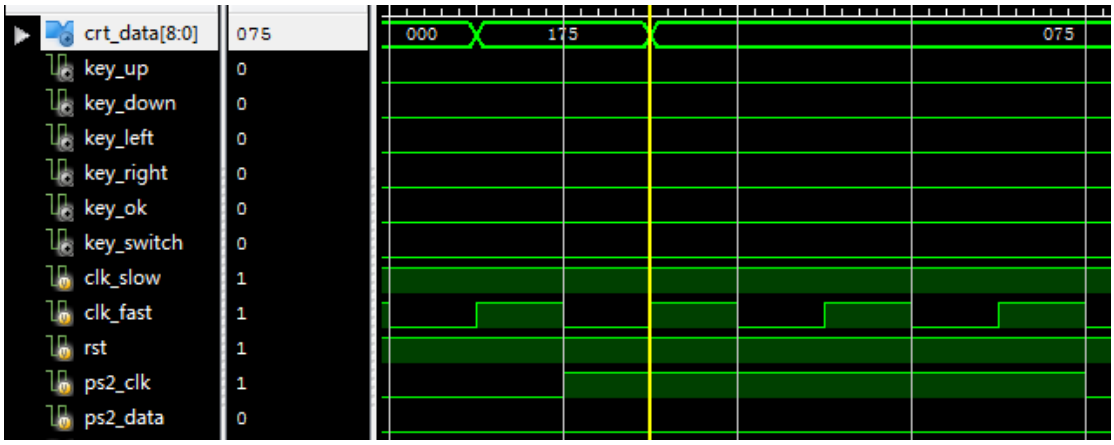

图 17　PS/2 输入模块初次仿真结果图 2

不过，如果我们将仿真结果图放大就会发现，在一段很短的时间内（一个 clk_fast 周期内），PS/2 数据接收模块的确正确接收到了 e0 + 75 的信息，可是为什么它又很快地丢弃了 e0，只保留了 75 的信息呢？我们来对比一下 PS/2 输入模块的第一版代码（下方左侧）与最终代码（下方右侧）的代码片段。

```
else if (read_state == 4'b1010) begin
    if (read_data == 8'hf0)
        is_f0 <= 1'b1;
    else if (read_data == 8'he0)
        is_e0 <= 1'b1;
    else
        if (is_f0) begin
            // A key is released
            is_f0 <= 1'b0;
            is_e0 <= 1'b0;
            crt_data <= 9'b0;
        end
        else if (is_e0) begin
            is_e0 <= 1'b0;
            crt_data <= {1'b1,
read_data};
        end
        else
            crt_data <= {1'b0,
read_data};
end
```

```
else if (read_state == 4'b1010 &&
|read_data) begin
    if (read_data == 8'hf0)
        is_f0 <= 1'b1;
    else if (read_data == 8'he0)
        is_e0 <= 1'b1;
    else
        if (is_f0) begin
            // A key is released
            is_f0 <= 1'b0;
            is_e0 <= 1'b0;
            crt_data <= 9'b0;
        end
        else if (is_e0) begin
            is_e0 <= 1'b0;
            crt_data <= {1'b1,
read_data};
        end
        else
            crt_data <= {1'b0,
read_data};

    read_data <= 8'b0;
end
```

对 read_data 的清空似乎是保证代码正确性重要的一环。事实上，由于 clk_fast 的频率比 ps2_clk 的频率高很多，如果在处理完已经获得的输出数据 read_data 后没有进行清空，那么这一段代码片段会在下一个 ps2_clk 下降沿到来之前重复执行。然而，is_f0 和 is_e0 两个标记寄存器在代码片段第一次执行时早都被置为 0 了，之后重复的执行自然会导致数据记录错误。为了防止代码片段重复执行，我们需要清空 read_data，并且只能在 read_data 非 0 时执行代码片段。

经过这样的修改，以下是 PS/2 输入模块的最终仿真结果。

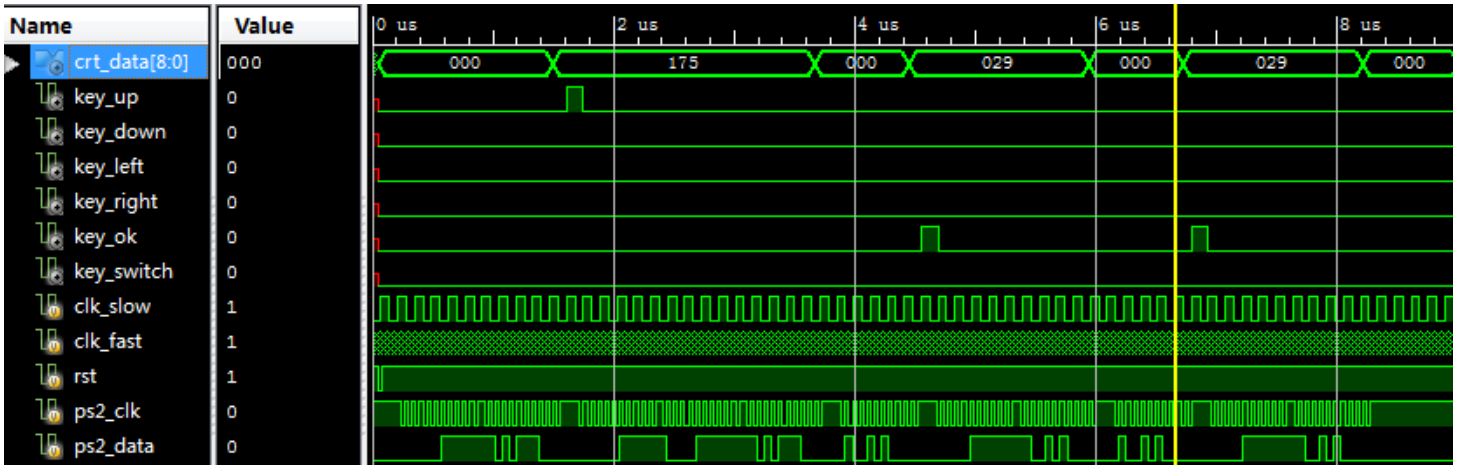

图 18　PS/2 输入模块最终仿真结果图

可以看到，crt_data 的数据都被正确记录了，key_up 也能被正确置为高电平，key_ok 也能被正确置为高电平两次。模块仿真表现符合预期。

第四章　系统测试验证与结果分析

4.1　功能测试

根据本课程设计希望达成的功能，我对最终成果进行了测试。以下为测试项目与结果。

表 3　功能测试项目与结果

测试项目	结果
游戏开始前按下左 Ctrl 键可以切换玩家与 AI	通过
游戏开始后按下左 Ctrl 键不能切换玩家与 AI	通过
游戏开始前按下空格键可以开始游戏	通过
游戏开始后按下四个方向键可以移动光标， 且光标不会移动到棋盘外	通过
游戏开始后若为玩家行棋， 光标指向无棋子格点时按下空格键可以下棋	通过
游戏开始后若为玩家行棋， 光标指向有棋子格点时按下空格键不能下棋	通过
游戏开始后若为 AI 行棋， 按下空格键玩家不能下棋	通过
黑方获胜时可以显示黑方获胜提示	通过
白方获胜时可以显示白方获胜提示	通过
平局时可以显示平局提示	通过
可以正确完成黑方玩家与白方 AI 的对局	通过
可以正确完成白方玩家与黑方 AI 的对局	通过
可以正确完成两个 AI 的对局	通过
可以正确完成两个玩家的对局	通过

4.2　技术参数测试

由于本课程设计没有与技术参数有关的设计，所以不进行技术参数测试。

4.3　结果分析

由以上测试结果可以得出，最终的设计成果成功实现了一个支持玩家与 AI、玩家与玩家以及 AI 与 AI 对局的基于数字系统的五子棋游戏。当然，我们肯定不能说，这个基于数字系统的带 AI 五子棋游戏并不是一个完美的五子棋游戏。至少从 AI 方面来说，它还存在着策略较为简单，AI 水平不够高的问题。当然，作为一份数字逻辑课程的课程设计，能够在有限的时间内取得这些成果，至少我对自己的作品是非常满意的。

4.4 系统演示与操作说明

4.4.1 操作说明

在游戏开始之前，按键盘的左 Ctrl 键可以切换黑白方的玩家与 AI，按空格键开始游戏。在游戏开始后，轮到玩家操作时，按键盘的四个方向键可以移动光标，按空格键可以在光标指示的位置下棋。

操作说明会时刻显示在屏幕上，也可以在屏幕上进行查看。

4.4.2 系统演示

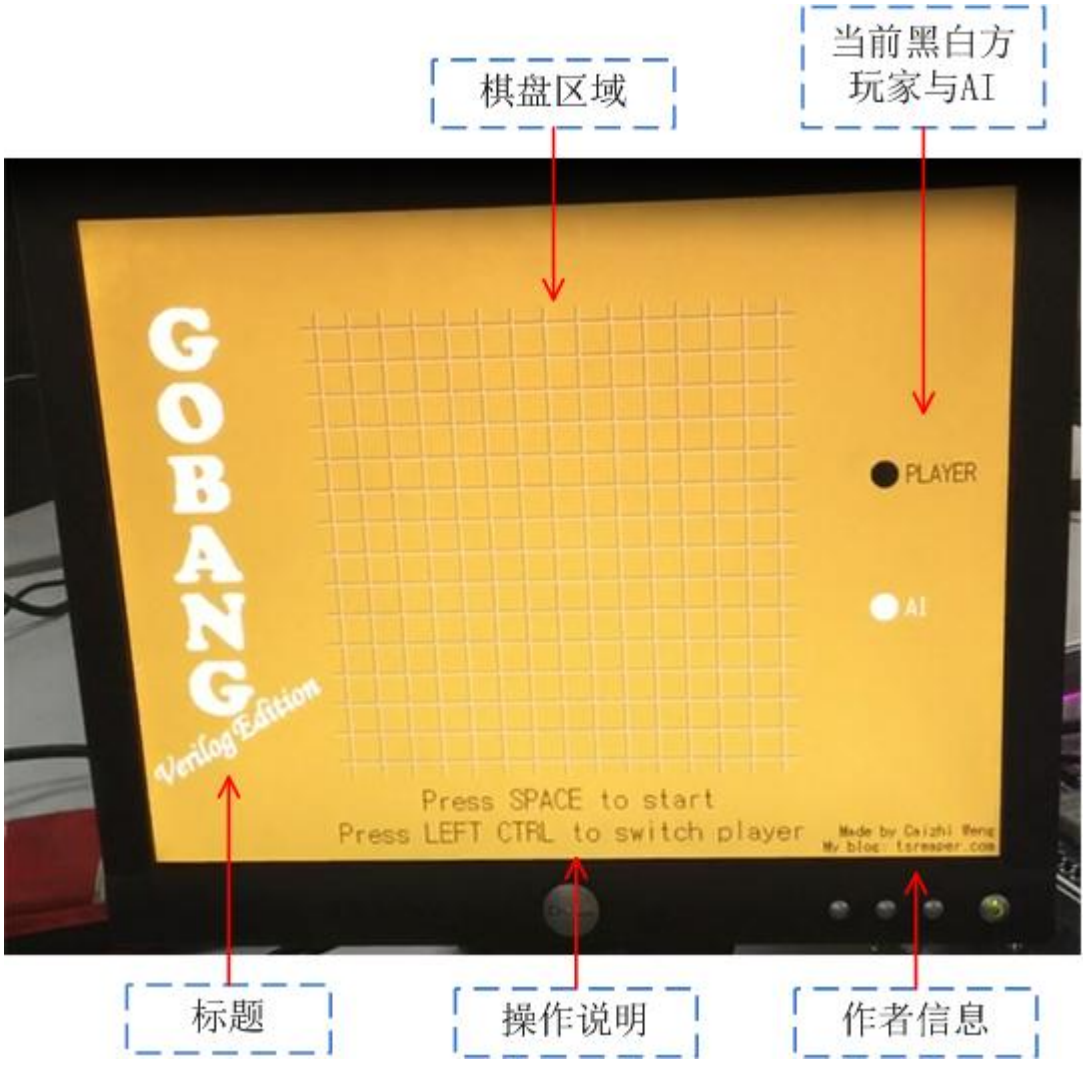

图 19 游戏开始前画面

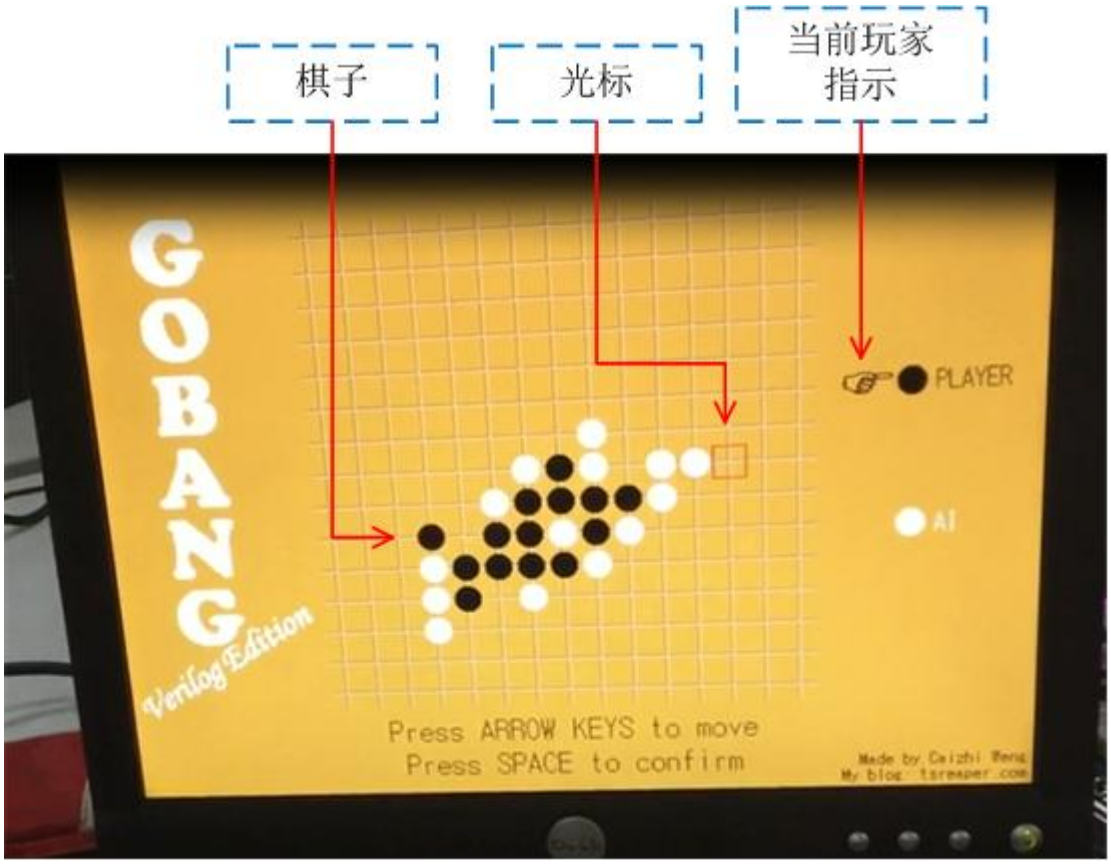

图 20　游戏进行画面

图 21　黑方胜利画面

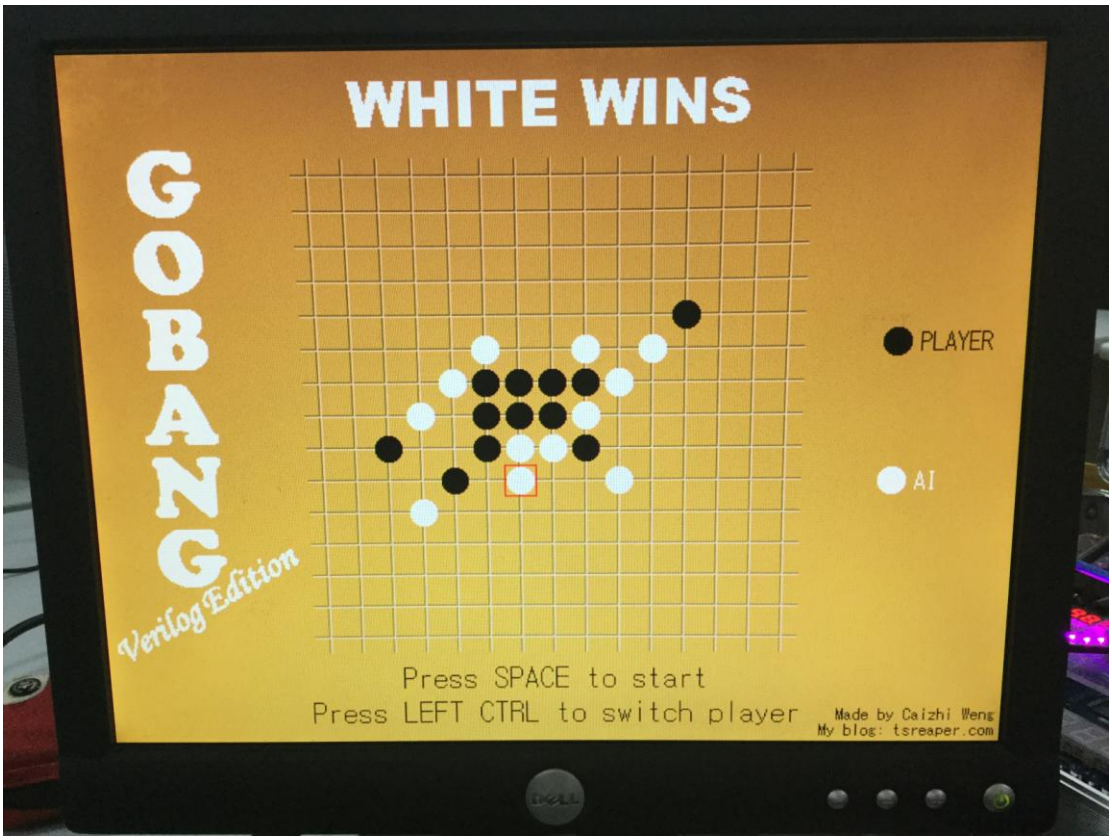

图 22　白方胜利画面

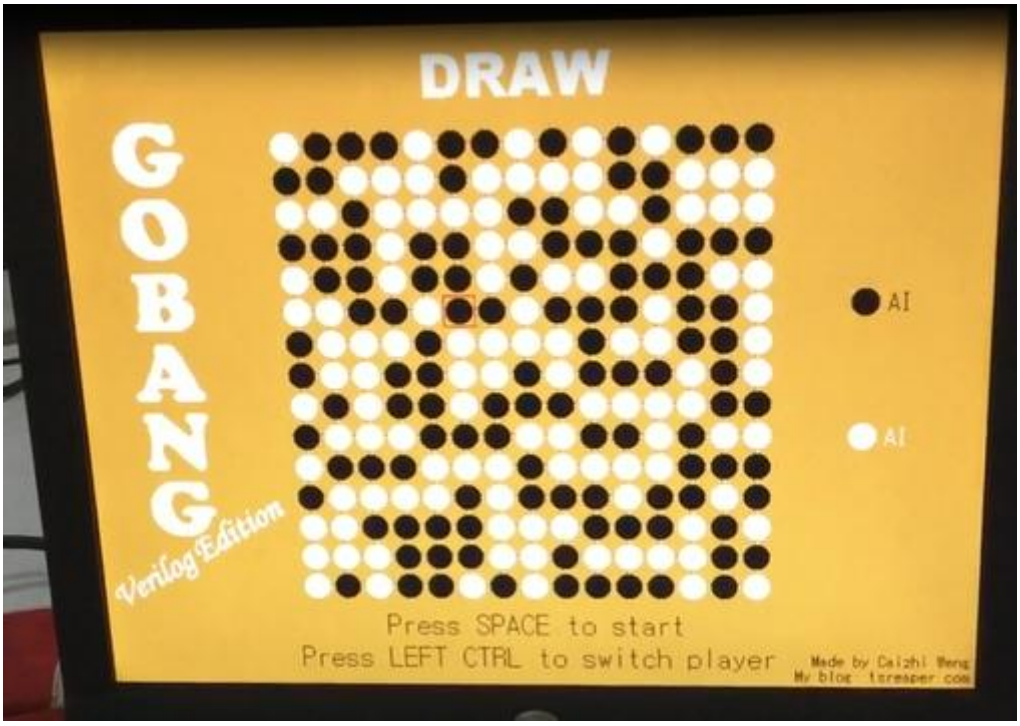

图 23　平局画面

第五章　结论与展望

“实践是检验真理的唯一标准”，这份课程设计正是对课堂所讲理论知识很好的补充。在数字逻辑设计的课堂上，老师经常强调硬件描述语言以及硬件思想与软件思想的不同。然而，只有当我在制作课程设计的过程中，真正碰到了代码无法综合、电路开销过大等问题的时候，我才能真正体会两者的区别。也正是在我为优化电路，学习课堂上没有提及的 VGA、PS/2 等模块而四处搜索资料时，我才渐渐体会到了一些硬件设计的思想，对硬件设计有了自己初步的认识。

这次课程设计的制作，给了我一次与之前所有实验课完全不同的体验。这种自己提出想法，遇到问题，解决问题，然后制作出最终成果所带来的顿悟感与满足感，不是上上理论课或者做做普通实验写写报告可以带来的。这次课程设计可以说是数字逻辑设计课程最大的亮点之一，希望之后的各类课程都能加入更多这样的实践元素。

对于这次课程设计，我唯一不满意的就是这份报告模版。报告模版中提到的内容要点存在一些前后颠倒或大量重复的问题，希望今后能够对报告模版进行修正。