浙江大学

数字逻辑设计课程设计报告

题目：永不言弃（小球的闯关小游戏）

姓名/学号：	黄炯睿 3170103455
	雷骁 3170105063
	刘非凡 3170102894
	章雨婷 3170102582
指导教师：	洪奇军
参加成员：	黄炯睿 雷骁 刘非凡 章雨婷
专业类别：	信息安全
所在学院：	计算机科学与技术学院

第1章 绪论

1.1 闯关小游戏的设计概述

玩家可以控制一个不断弹跳的球体前进。没有任何指令时，球体每次向前弹跳一格，并且向上弹跳的高度较低。仅当球体碰到地面的格子时对其发出的指令有效。指令只有一种：但当小球所在柱子的下一个柱子高于该柱子时，小球向上跳跃；否则就向下跳跃。玩家要控制球体不断躲过路上的各种障碍，躲过的障碍越多，得分越高。得分显示在七段数码管上，一旦球体碰到障碍，游戏失败。

1.2 主要内容和难点

时钟的设计，因为要保证每一次小球的下落都要恰好落在“板子”上中心的位置，而且随着不同的跳跃或下落，时钟的值要不断的进行改变，这给我们写代码带来了极大的困难。

碰撞信号的处理，因为只用单个键控制小球，所以在不同情况下的碰撞判定有可能不同，需要进行分类判定。并且由于 vga 的扫描实际上小球的位置是离散的点而不是连续的，这给我们的碰撞判定带来很大的麻烦（这个 bug 找了一下午）

由于本实验对于时钟的准确性有着相当高的要求，实际上在写代码时，时钟问题给我们带来了极大的困扰。在不导入背景图等图片时的小球弹跳逻辑完全正确，但是图片导入之后就会产生许多 bug。

第2章 设计原理

2.1 游戏设计相关内容

此次实验硬件部分为 sword 工具箱，主要使用了七段数码管、VGA 显示屏、SW 输入以及 BTN 按钮输入等部分。

软件使用 ISE Design Suite 了，代码主要使用了 Verilog 语言。

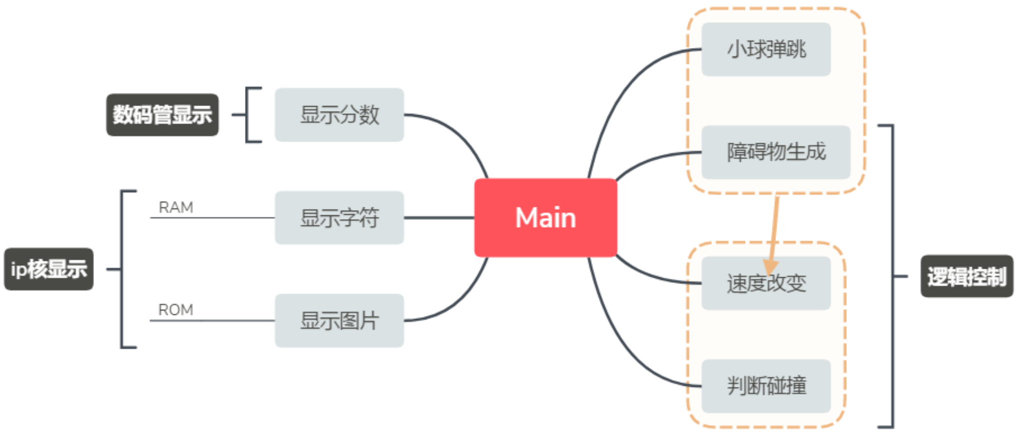
2.2 设计方案

此系统需要调用较多的元件，包括七段数码管、BTN、SW、VGA 等部分。

根据 clk 生成的时钟，在单位时间内实现小球的弹跳状态，并且根据时间每一定周期重

复一次弹跳状态，并始终保持球体处于屏幕正中，只在 y 轴方向上下弹跳。根据前方来的障碍物的不同来判定按下按钮时小球以及背景图的运动。

原理图如下：



图一　“永不言弃”游戏系统原理框图（原理图）

2.3 游戏硬件设计

顶层模块包括 1s 的时钟、VGA 显示模块、七段数码管的显示模块、小球的弹跳模块、障碍物的移动模块、画出物体的模块。

下面依次介绍各模块的详细结构。

1.1 时钟：用 clk 1ns 的时钟生成一个 1s 的时钟。

1.2 字符发生器模块：　用点阵的方式将 10 个数字和 26 个字母以及较常用的符号进行点阵的排列，并且以输入的数字来调出相应的信号进入画图的模块中。

1.3 画图模块：将背景图，字符点阵，小球和障碍物的信号传入这个模块，输出 VGA 当前打印位置的 rgb 数据。

1.4 VGA 模块：用 rgb 数据进行扫描绘图

1.5 障碍物生成及移动模块：利用随机数生成随机高度的新柱子，并且控制障碍物左移的速度，当出发小球向前跳事件时障碍物左移速度加倍。

1.6 小球的弹跳模块：根据小球下一个柱子的高度来判断小球是向上还是向前跳跃。不同的跳跃方式给予不同的初速度。

1.7 七段数码管显示部分：用障碍物每移动 64 个像素点来表示用户当前所跳过的柱子

数，即玩家的得分。

1.8 复位模块：当用户按复位按钮时可以将小球复位

第3章 游戏设计实现

包含图片输入部分和小球运行的逻辑部分以及字符点阵的运用，每个部分含有不同的module，他们的具体实现过程如下：

1. 图片的显示

图片显示利用 VGA 扫描信号以及多张 ip 核的导入，将导入的 ip 核算出扫描位置的 rgb 数据，因为不同位置可能要不同的 ip 核数据所以可以将不同的 rgb 数据在满足特定条件的情况下传入给 VGA 扫描模块。其中待画的有 10 个障碍物、一个小球、游戏运行时的背景图、游戏开始界面、游戏结束界面以及游戏运行时运用字符点阵输出于屏幕下方的句子。十个障碍物是由三种不同高度的柱子以一定顺序的排列。图片显示代码如下：

```
wire [19:0] pow_x, pow_y, pow_radius;

wire direction;

reg collision;

reg [18:0] addr;

wire [11:0] data_1, data_2, data_3, data_4, data_5, data_6, data_7 , data_8;

reg [6:0] height_get;

reg count;

wire  [6:0]  height_1,  height_2,  height_3,  height_4,  height_5,  height_6,  height_7,
height_8, height_9, height_10, left_height;

reg [11:0] vga_data;

wire [7:0] p[90:0];

    //  屏幕中央字符的显示区域

    parameter up_pos =    460;
```

```
	parameter down_pos = 466;
	parameter left_pos = 274;
	parameter right_pos = 365;
	//显示字符
	RAM_set u_ram_1 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b01_0111),//N

	.col0(p[0]),.col1(p[1]),.col2(p[2]),.col3(p[3]),.col4(p[4]),.col5(p[5]),.col6(p[6]));

	RAM_set u_ram_2 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b00_1110),//E

	.col0(p[7]),.col1(p[8]),.col2(p[9]),.col3(p[10]),.col4(p[11]),.col5(p[12]),.col6(p[13]
));
	RAM_set u_ram_3 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b01_1111),//V

	.col0(p[14]),.col1(p[15]),.col2(p[16]),.col3(p[17]),.col4(p[18]),.col5(p[19]),.col6(p[
20]));

	RAM_set u_ram_4 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b00_1110),//E

	.col0(p[21]),.col1(p[22]),.col2(p[23]),.col3(p[24]),.col4(p[25]),.col5(p[26]),.col6(p[
27]));
	RAM_set u_ram_5 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b01_1011),//R

	.col0(p[28]),.col1(p[29]),.col2(p[30]),.col3(p[31]),.col4(p[32]),.col5(p[33]),.col6(p[
34]));

	RAM_set u_ram_6 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b11_1110),//空格

	.col0(p[35]),.col1(p[36]),.col2(p[37]),.col3(p[38]),.col4(p[39]),.col5(p[40]),.col6(p[
41]));
        RAM_set u_ram_7 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b01_0000),//G

        .col0(p[42]),.col1(p[43]),.col2(p[44]),.col3(p[45]),.col4(p[46]),.col5(p[47]),.col6(p[
48]));

        RAM_set u_ram_8 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b01_0010),//I

        .col0(p[49]),.col1(p[50]),.col2(p[51]),.col3(p[52]),.col4(p[53]),.col5(p[54]),.col6(p[
55]));
        RAM_set u_ram_9 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b01_1111),//V

        .col0(p[56]),.col1(p[57]),.col2(p[58]),.col3(p[59]),.col4(p[60]),.col5(p[61]),.col6(p[
62]));
        RAM_set u_ram_10 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b00_1110),//E

        .col0(p[63]),.col1(p[64]),.col2(p[65]),.col3(p[66]),.col4(p[67]),.col5(p[68]),.col6(p[
69]));
        RAM_set u_ram_11 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b11_1110),//" "

        .col0(p[70]),.col1(p[71]),.col2(p[72]),.col3(p[73]),.col4(p[74]),.col5(p[75]),.col6(p[
76]));
        RAM_set u_ram_12 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b01_1110),//U

        .col0(p[77]),.col1(p[78]),.col2(p[79]),.col3(p[80]),.col4(p[81]),.col5(p[82]),.col6(p[
83]));
        RAM_set u_ram_13 (.clk(clkdiv[20]),.rst(SW_OK),.data(6'b01_1001),//P

        .col0(p[84]),.col1(p[85]),.col2(p[86]),.col3(p[87]),.col4(p[88]),.col5(p[89]),.col6(p[
90]));
assign pow_x = (x - col_addr) * (x - col_addr);
assign pow_y = (y - row_addr) * (y - row_addr);
assign pow_radius = radius * radius;
        always @(row_addr or col_addr) begin
                        if(collision == 0)begin
                                if(start == 0)begin
                                        addr = (row_addr - 1) * 640 + col_addr - 1;
                                        height_get <= 2;
                                end
                                // 移动的圆球
                                else if(pow_x + pow_y < pow_radius && count == 0)begin
                                        addr = (row_addr - y + radius) * 40 + col_addr - x + radius;
                                        height_get <= 100;
                                end
                                else if(pow_x + pow_y < pow_radius && count == 1)begin
                                        addr = (row_addr - y + radius) * 40 + col_addr - x + radius;
                                        height_get <= 101;
                                end
                                // 字符
                                else if (row_addr>=up_pos && row_addr<=down_pos
                                && col_addr>=left_pos && col_addr<=right_pos && p[col_addr-
left_pos][row_addr-up_pos]) begin
                                        height_get <= 4;
                                end
                                //10 barriers
                                else if(x_1 >= 24 && col_addr < x_1 - 24 && row_addr > barrier_y
- left_height && row_addr < barrier_y)begin
                                        addr = (row_addr - barrier_y + left_height - 1) * 40 + (col_addr
+ 63 - x_1);
                    height_get <= left_height;
                end
                else if(col_addr > x_1 && col_addr < x_1 + width && row_addr >
barrier_y - height_1 && row_addr < barrier_y)begin
                    addr = (row_addr - barrier_y + height_1 - 1 ) * 40 + (col_addr
- x_1 - 1);
                    height_get <= height_1;
                end
                else if(col_addr > x_2 && col_addr < x_2 + width && row_addr >
barrier_y - height_2 && row_addr < barrier_y)begin
                    addr = (row_addr - barrier_y + height_2 - 1 ) * 40 + (col_addr
- x_2 - 1);
                    height_get <= height_2;
                end
                else if(col_addr > x_3 && col_addr < x_3 + width && row_addr >
barrier_y - height_3 && row_addr < barrier_y)begin
                    addr = (row_addr - barrier_y + height_3 - 1 ) * 40 + (col_addr
- x_3 - 1);
                    height_get <= height_3;
                end
                else if(col_addr > x_4 && col_addr < x_4 + width && row_addr >
barrier_y - height_4 && row_addr < barrier_y)begin
                    addr = (row_addr - barrier_y + height_4 - 1 ) * 40 + (col_addr
- x_4 - 1);
                    height_get <= height_4;
                end
                else if(col_addr > x_5 && col_addr < x_5 + width && row_addr >
barrier_y - height_5 && row_addr < barrier_y)begin
```

```
                    addr = (row_addr - barrier_y + height_5 - 1 ) * 40 + (col_addr
- x_5 - 1);
                    height_get <= height_5;
                end
                else if(col_addr > x_6 && col_addr < x_6 + width && row_addr >
barrier_y - height_6 && row_addr < barrier_y)begin
                    addr = (row_addr - barrier_y + height_6 - 1 ) * 40 + (col_addr
- x_6 - 1);
                    height_get <= height_6;
                end
                else if(col_addr > x_7 && col_addr < x_7 + width && row_addr >
barrier_y - height_7 && row_addr < barrier_y)begin
                    addr = (row_addr - barrier_y + height_7 - 1 ) * 40 + (col_addr
- x_7 - 1);
                    height_get <= height_7;
                end
                else if(col_addr > x_8 && col_addr < x_8 + width && row_addr >
barrier_y - height_8 && row_addr < barrier_y)begin
                    addr = (row_addr - barrier_y + height_8 - 1 ) * 40 + (col_addr
- x_8 - 1);
                    height_get <= height_8;
                end
                else if(col_addr > x_9 && col_addr < x_9 + width && row_addr >
barrier_y - height_9 && row_addr < barrier_y)begin
                    addr = (row_addr - barrier_y + height_9 - 1 ) * 40 + (col_addr
- x_9 - 1);
                    height_get <= height_9;
                end
                else if(col_addr > x_10 && col_addr < x_10 + width && row_addr >
barrier_y - height_10 && row_addr < barrier_y)begin
                    addr = (row_addr - barrier_y + height_10 - 1 ) * 40 + (col_addr
- x_10 - 1);
                    height_get <= height_10;
                end
                else begin
                    addr = (row_addr - 1) * 640 + col_addr - 1;
                    height_get <= 0;
                end
            end
                // 背景
            else begin
                if(start == 0)begin
                    addr = (row_addr - 1) * 640 + col_addr - 1;
                    height_get <= 1;
                end
                else begin
                    addr = (row_addr - 1) * 640 + col_addr - 1;
                    height_get <= 0;
                end
            end
                //Addition
                if(height_get == 10)begin
                    vga_data <= data_1;
                end
                else if(height_get == 40)begin
                    vga_data <= data_2;
                end
                else if(height_get == 70)begin
                vga_data <= data_3;
            end
            else if(height_get == 100)begin
                vga_data <= data_4;
            end
            else if(height_get == 101)begin
                vga_data <= data_5;
            end
            else if(height_get == 0)begin
                vga_data <= data_6;
            end
            else if(height_get == 1)begin
                vga_data <= data_7;
            end
            else if(height_get == 2)begin
                vga_data <= data_8;
            end
            else if (height_get == 4) begin
                vga_data <= 0;
            end

    end

    low_no ROMO1(.a(addr), .spo(data_1));
    mid_no ROMO2(.a(addr), .spo(data_2));
    high_no ROMO3(.a(addr), .spo(data_3));
    pig_0 ROMO4(.a(addr), .spo(data_4));
    pig_4 ROMO5(.a(addr), .spo(data_5));
    back ROMO6(.clka(clkdiv[1]), .addra(addr), .douta(data_6));
    gameover ROMO7(.clka(clkdiv[1]), .addra(addr), .douta(data_7));
    begingame ROMO8(.clka(clkdiv[1]), .addra(addr), .douta(data_8));
```

2.　小球运行的逻辑

小球的运行模拟了自由落体，设置 speed 和 direction，让其根据时钟周期变化，粗略模拟变速运动。小球的速度一次更改一个单位。通过输入小球触底时的初速度来控制小球的运动状态

平动：Speed：7->0->7　　Direction:1->0->1

高跳：Speed：9->0->5　　Direction:1->0->1

掉落：Speed：5->0->9　　Direction:1->0->1

代码如下：

```
`timescale 1ns / 1ps
module bounce(
    input wire clk,
    input wire start,
    input wire reset,
    input wire [3:0] max_speed,
    output reg [9:0] x,
    output reg [8:0] y,
    output reg [3:0] speed,
    output reg direction
    );

    initial begin
        x <= 10'd276;
        y <= 9'd320;
        speed <= 7;
        direction <= 1;
    end
```

```
always @(posedge clk or negedge reset) begin
    if(reset == 0)begin
        if(start == 0)begin
            x <= 10'd276;
            y <= 9'd320;
            speed <= 7;
            direction <= 1;
        end
    end
    else begin
        if(start == 1) begin
            if(direction == 1)
                y <= y - speed;
            else
                y <= y + speed;
            if(speed == 0)
                direction <= ~direction;
            if(direction == 1 && speed != 0 || speed == 0 && direction == 0)begin
                if(speed == 0)
                    speed <= max_speed;
                else
                    speed <= speed - 1;
            end
            else begin
                if(speed == 14 - max_speed)
                    speed <= 0;
                else
                    speed <= speed + 1;
                end
            end
        end
    end

endmodule
```

3.　障碍物的移动

用坐标记录 10 个障碍物柱子的横坐标，高度，统一的纵坐标和宽度

简单地根据时钟和输入 barrier_speed 统一改变柱子的坐标信息，

当出现第一个被记录的柱子部分移动到显示屏幕边沿，依此向左传递位置信息，即 x_1<=x_2, x_2<=x_3…如此，并生成第十个柱子的坐标以及形状信息（柱子的高度随机生成）。这样就能实现在判断是否有柱子移动到屏幕外面时，只需要判断 x_1，而且便于管理这些数据。此外由于即使柱子部分越过边界，也需要显示，故将部分越过边界的柱子的高度记录在 left_height 里面。

因为障碍物的移动是恒速的，可以用它来计算玩家所得分数，柱子每移动 64 个像素单位计分板加 1。

```
`timescale 1ns / 1ps
module barrier(
    input wire [31:0] clk,
    input wire start,
    input wire reset,
    input wire [3:0] barrier_speed,
    output reg [15:0] score,
    output reg [5:0] width,
    output reg [8:0] y,
    output reg [9:0] x_1,
    output reg [9:0] x_2,
    output reg [9:0] x_3,
output reg [9:0] x_4,
output reg [9:0] x_5,
output reg [9:0] x_6,
output reg [9:0] x_7,
output reg [9:0] x_8,
output reg [9:0] x_9,
output reg [9:0] x_10,
output reg [6:0] height_1,
output reg [6:0] height_2,
output reg [6:0] height_3,
output reg [6:0] height_4,
output reg [6:0] height_5,
output reg [6:0] height_6,
output reg [6:0] height_7,
output reg [6:0] height_8,
output reg [6:0] height_9,
output reg [6:0] height_10,
output reg [6:0] left_height
);

reg [5:0] random_num;
reg [19:0] cnt;

initial begin
    cnt <= 0;
    random_num <= 53;
    score <= 0;
    width <= 40;
    y <= 350;
```

```
        x_1 <= 0;
        x_2 <= 64;
        x_3 <= 128;
        x_4 <= 192;
        x_5 <= 256;
        x_6 <= 320;
        x_7 <= 384;
        x_8 <= 448;
        x_9 <= 512;
        x_10 <= 576;
        height_1 <= 10;
        height_2 <= 10;
        height_3 <= 10;
        height_4 <= 10;
        height_5 <= 10;
        height_6 <= 10;
        height_7 <= 10;
        height_8 <= 40;
        height_9 <= 40;
        height_10 <= 40;
        left_height <= 0;
    end

    always @(posedge clk[0])begin
        cnt <= cnt + 1;
        if(cnt == 1000000)begin
            random_num <= random_num + 53;
            cnt <= 0;
        end
```

```
end


always @(posedge clk[20] or negedge reset) begin
    if(reset == 0) begin
        if(start == 0)begin
            score <= 0;
            width <= 40;
            y <= 350;
            x_1 <= 0;
            x_2 <= 64;
            x_3 <= 128;
            x_4 <= 192;
            x_5 <= 256;
            x_6 <= 320;
            x_7 <= 384;
            x_8 <= 448;
            x_9 <= 512;
            x_10 <= 576;
            height_1 <= 10;
            height_2 <= 10;
            height_3 <= 10;
            height_4 <= 10;
            height_5 <= 10;
            height_6 <= 10;
            height_7 <= 10;
            height_8 <= 40;
            height_9 <= 40;
            height_10 <= 40;
            left_height <= 0;
        end
    end
    else begin
        if(start == 1)begin
            if(x_1 >= barrier_speed) begin
                x_1 <= x_1 - barrier_speed;
                x_2 <= x_2 - barrier_speed;
                x_3 <= x_3 - barrier_speed;
                x_4 <= x_4 - barrier_speed;
                x_5 <= x_5 - barrier_speed;
                x_6 <= x_6 - barrier_speed;
                x_7 <= x_7 - barrier_speed;
                x_8 <= x_8 - barrier_speed;
                x_9 <= x_9 - barrier_speed;
                x_10 <= x_10 - barrier_speed;
            end
            else begin
                x_1 <= x_2 - barrier_speed;
                x_2 <= x_3 - barrier_speed;
                x_3 <= x_4 - barrier_speed;
                x_4 <= x_5 - barrier_speed;
                x_5 <= x_6 - barrier_speed;
                x_6 <= x_7 - barrier_speed;
                x_7 <= x_8 - barrier_speed;
                x_8 <= x_9 - barrier_speed;
                x_9 <= x_10 - barrier_speed;
                x_10 <= x_10 + 64 - barrier_speed;

                left_height <= height_1;
```

```
                    height_1 <= height_2;
                    height_2 <= height_3;
                    height_3 <= height_4;
                    height_4 <= height_5;
                    height_5 <= height_6;
                    height_6 <= height_7;
                    height_7 <= height_8;
                    height_8 <= height_9;
                    height_9 <= height_10;

                    if(random_num <= 32 && height_10 == 40 &&
height_9 != 70)
                        height_10 <= 10;
                    else if(random_num <= 32 && height_10 == 70)
                        height_10 <= 40;
                    else if(random_num > 32 && height_10 == 10)
                        height_10 <= 40;
                    else if(random_num > 32 && height_10 == 40)
                        height_10 <= 70;
                    else if(height_10 == 10)
                        height_10 <= 10;
                    else if(height_10 == 40)
                        height_10 <= 40;
                    else if(height_10 == 70)
                        height_10 <= 70;
                    else
                        height_10 <= 10;
                    if(barrier_speed == 4)
                        score <= score + 1;
                    else if (barrier_speed == 8)
                        score <= score + 2;
                end
            end
        end
    end
endmodule
```

4.　小球与障碍物的速度控制

柱子移动速度：通过输入 barrier_speed 简单控制一个时钟上升沿，所有柱子都左移速度决定的单位像素，这里速度只有两个值即 4 和 8，即小球平跳跳远时，速度变为 8，效果上正好跨过一个柱子，缺省状态下，只在相邻柱子变化， 即速度为 4。

小球跳跃速度：由于小球的跳动只通过一个按键控制，而需要同时视情况决定是平跳还是高跳还是低跳，这取决于当前柱子的高度以及后面相邻两个柱子的高度。这里根据小球跳跃的模块，只需要设置 max_speed 输出即可相应地实现小球下一个周期地运动状态。

周期性响应：要保证一个周期——即小球跳离柱子后再次回到柱子平面，这段时间内，小球和柱子的移动不受影响，只在小球处于柱子平面时响应 change 按钮的下降沿，并且在无 change 下降沿的情况下，一个周期自动复位小球和柱子的速度。

```
`timescale 1ns / 1ps
module change_speed(
    input wire clk,
    input wire direction,
    input wire change,
    input wire start,
    input wire reset,
    input wire collision,
input wire [3:0] ball_speed,
input wire [6:0] height_5,
input wire [6:0] height_6,
input wire [6:0] height_7,
output reg [3:0] barrier_speed,
output reg [3:0] max_speed
);

initial begin
        max_speed <= 7;
        barrier_speed <= 4;
end

always @(posedge clk or negedge change or negedge reset)begin
        if(reset == 0)begin
                if(start == 0 || collision == 1)begin
                        max_speed <= 7;
                        barrier_speed <= 4;
                end
        end
        else begin
                if(change == 0)begin
                        if(start == 1)begin
                                if(ball_speed == 0 && direction == 0 && height_5 >= height_6)
                                        barrier_speed <= 8;
                                else if(ball_speed == 0 && direction == 0)
                                        barrier_speed <= 4;

                                if(ball_speed == 0 && direction == 0 && height_5 >= height_6 &&
height_5 > height_7)
                                    max_speed <= 5;
                              else if(ball_speed == 0 && direction == 0 && height_6 > height_5)
                                    max_speed <= 9;
                              else if(ball_speed == 0 && direction == 0)
                                    max_speed <= 7;
                        end
                  end
                  else begin
                        if(start == 1)begin
                              if(ball_speed    ==    0    &&    direction    ==    0    &&    height_6    <
height_5)begin
                                    max_speed <= 5;
                                    barrier_speed <= 4;
                              end
                              else begin
                                    if(ball_speed == 0 && direction == 0)begin
                                          max_speed <= 7;
                                          barrier_speed <= 4;
                                    end
                              end
                        end
                  end
            end
      end

endmodule
```

5.　碰撞的判定

在小球运动到障碍柱子的左边沿，判断低端是否低于柱子最高处，若低于显然无法到达柱子最高处，此时判断是有碰撞，置 collision 为 1 表示碰撞状态，置 start 为 0 表示此时游戏状态为停止。

此外 SW_OK[1]控制了游戏状态，三种：即初始，进行，结束。同时由 collision 和 start 的值决定。

```
    always@(*) begin
        if(SW_OK[1] == 1)begin
            if(x + radius >= x_5 && x + radius <= x_5 + 4 && y + 
radius >= barrier_y - height_5)begin
                start <= 0;
                collision <= 1;
            end
            else begin
                start <= 1;
                collision <= 0;
            end
        end
        else begin
            start <= 0;
            collision <= 0;
        end
    end
```

6.　障碍物的随机生成

利用 6 位的数字 random（最大 64）每一定周期后加上一个在 32-64 靠中间位置的素数（这里取 53），使 random 数字以一定概率落在 32 数字的两边。利用该数字和即将成为第 9

个柱子的第十个柱子的高，来随机生成符合小球跳跃的新的第十个柱子。

```
initial begin
        cnt <= 0;
        random_num <= 53;
end


    always @(posedge clk[0])begin
        cnt <= cnt + 1;
        if(cnt == 1000000)begin
            random_num <= random_num + 53;
            cnt <= 0;
        end
    end
always @(posedge clk[20] or negedge reset) begin
    if(reset == 1) begin
            if(random_num <= 32 && height_10 == 40 && height_9 != 70)
                height_10 <= 10;
            else if(random_num <= 32 && height_10 == 70)
                height_10 <= 40;
            else if(random_num > 32 && height_10 == 10)
                height_10 <= 40;
            else if(random_num > 32 && height_10 == 40)
                height_10 <= 70;
            else if(height_10 == 10)
                height_10 <= 10;
            else if(height_10 == 40)
                height_10 <= 40;
            else if(height_10 == 70)
                height_10 <= 70;
                else
                    height_10 <= 10;
    end
end
```

7.　　字符点阵

将每一个字符看作由 7*8 的像素块，并在 RAM_set 的模块中将 26 个字符和 10 个数字进行编码，输入一个给定数字能够输出一个 7*8 的点阵。 用存储器将输出的数个输出字符保存下来，再用 VGA 将点阵输出，可以在显示屏上看到清晰的字符。用数个字符可以组成有意义的句子显示出来。

```
`timescale 1ns / 1ps
module RAM_set(
    input clk,
    input rst,
    input [5:0] data,
    output reg [7:0] col0,
    output reg [7:0] col1,
    output reg [7:0] col2,
    output reg [7:0] col3,
    output reg [7:0] col4,
    output reg [7:0] col5,
    output reg [7:0] col6
    );

    always @(posedge clk or negedge rst)
        begin
            if (!rst)
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0000_0000;
                    col2 <= 8'b0000_0000;

                    col3 <= 8'b0000_0000;

                    col4 <= 8'b0000_0000;

                    col5 <= 8'b0000_0000;

                    col6 <= 8'b0000_0000;

                end

            else

                begin

                    case (data)

                        6'b00_0000: // "0"

                            begin

                                col0 <= 8'b0000_0000;

                                col1 <= 8'b0011_1110;

                                col2 <= 8'b0101_0001;

                                col3 <= 8'b0100_1001;

                                col4 <= 8'b0100_0101;

                                col5 <= 8'b0011_1110;

                                col6 <= 8'b0000_0000;

                            end

                        6'b00_0001: // "1"

                            begin

                                col0 <= 8'b0000_0000;

                                col1 <= 8'b0000_0000;;

                                col2 <= 8'b0100_0010;

                                col3 <= 8'b0111_1111;

                                col4 <= 8'b0100_0000;

                                col5 <= 8'b0000_0000;;

                                col6 <= 8'b0000_0000;

                            end
            6'b00_0010: // "2"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0100_0010;
                    col2 <= 8'b0110_0001;
                    col3 <= 8'b0101_0001;
                    col4 <= 8'b0100_1001;
                    col5 <= 8'b0100_0110;
                    col6 <= 8'b0000_0000;
                end
            6'b00_0011: // "3"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0010_0010;
                    col2 <= 8'b0100_0001;
                    col3 <= 8'b0100_1001;
                    col4 <= 8'b0100_1001;
                    col5 <= 8'b0011_0110;
                    col6 <= 8'b0000_0000;
                end
            6'b00_0100: // "4"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0001_1000;
                    col2 <= 8'b0001_0100;
                    col3 <= 8'b0001_0010;
                    col4 <= 8'b0111_1111;
                    col5 <= 8'b0001_0000;
                    col6 <= 8'b0000_0000;
                    end
                6'b00_0101: // "5"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0010_0111;
                        col2 <= 8'b0100_0101;
                        col3 <= 8'b0100_0101;
                        col4 <= 8'b0100_0101;
                        col5 <= 8'b0011_1001;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_0110: // "6"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0011_1110;
                        col2 <= 8'b0100_1001;
                        col3 <= 8'b0100_1001;
                        col4 <= 8'b0100_1001;
                        col5 <= 8'b0011_0010;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_0111: // "7"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0110_0001;
                        col2 <= 8'b0001_0001;
                        col3 <= 8'b0000_1001;
                        col4 <= 8'b0000_0101;
                        col5 <= 8'b0000_0011;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_1000: // "8"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0011_0110;
                        col2 <= 8'b0100_1001;
                        col3 <= 8'b0100_1001;
                        col4 <= 8'b0100_1001;
                        col5 <= 8'b0011_0110;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_1001: // "9"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0010_0110;
                        col2 <= 8'b0100_1001;
                        col3 <= 8'b0100_1001;
                        col4 <= 8'b0100_1001;
                        col5 <= 8'b0011_1110;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_1010: // "A"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0111_1100;
                        col2 <= 8'b0001_0010;
                        col3 <= 8'b0001_0001;
                        col4 <= 8'b0001_0010;
                        col5 <= 8'b0111_1100;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_1011: // "B"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0111_1111;
                        col2 <= 8'b0100_1001;
                        col3 <= 8'b0100_1001;
                        col4 <= 8'b0100_1001;
                        col5 <= 8'b0011_0110;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_1100: // "C"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0011_1110;
                        col2 <= 8'b0100_0001;
                        col3 <= 8'b0100_0001;
                        col4 <= 8'b0100_0001;
                        col5 <= 8'b0010_0010;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_1101: // "D"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0111_1111;
                        col2 <= 8'b0100_0001;
                        col3 <= 8'b0100_0001;
                        col4 <= 8'b0100_0001;
                        col5 <= 8'b0011_1110;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_1110: // "E"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0111_1111;
                        col2 <= 8'b0100_1001;
                        col3 <= 8'b0100_1001;
                        col4 <= 8'b0100_1001;
                        col5 <= 8'b0100_0001;
                        col6 <= 8'b0000_0000;
                    end
                6'b00_1111: // "F"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0111_1111;
                        col2 <= 8'b0000_1001;
                        col3 <= 8'b0000_1001;
                        col4 <= 8'b0000_1001;
                        col5 <= 8'b0000_0001;
                        col6 <= 8'b0000_0000;
                    end
                6'b01_0000: // "G"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0011_1110;
                        col2 <= 8'b0100_0001;
                        col3 <= 8'b0100_1001;
                        col4 <= 8'b0100_1001;
                        col5 <= 8'b0011_1010;
                        col6 <= 8'b0000_0000;
                    end
                6'b01_0001: // "H"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0111_1111;
                        col2 <= 8'b0000_1000;
                        col3 <= 8'b0000_1000;
                        col4 <= 8'b0000_1000;
                        col5 <= 8'b0111_1111;
                        col6 <= 8'b0000_0000;
                    end
                6'b01_0010: // "I"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0000_0000;
                        col2 <= 8'b0100_0001;
                        col3 <= 8'b0111_1111;
                        col4 <= 8'b0100_0001;
                        col5 <= 8'b0000_0000;
                        col6 <= 8'b0000_0000;
                    end
                6'b01_0011: // "J"
                    begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0010_0000;
```

```
                    col2 <= 8'b0100_0001;
                    col3 <= 8'b0100_0001;
                    col4 <= 8'b0011_1111;
                    col5 <= 8'b0000_0001;
                    col6 <= 8'b0000_0000;
                end
            6'b01_0100: // "K"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0111_1111;
                    col2 <= 8'b0000_1000;
                    col3 <= 8'b0001_0100;
                    col4 <= 8'b0010_0010;
                    col5 <= 8'b0100_0001;
                    col6 <= 8'b0000_0000;
                end
            6'b01_0101: // "L"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0111_1111;
                    col2 <= 8'b0100_0000;
                    col3 <= 8'b0100_0000;
                    col4 <= 8'b0100_0000;
                    col5 <= 8'b0100_0000;
                    col6 <= 8'b0000_0000;
                end
            6'b01_0110: // "M"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0111_1111;
                    col2 <= 8'b0000_0010;
                    col3 <= 8'b0000_1100;
                    col4 <= 8'b0000_0010;
                    col5 <= 8'b0111_1111;
                    col6 <= 8'b0000_0000;
                end
            6'b01_0111: // "N"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0111_1111;
                    col2 <= 8'b0000_0010;
                    col3 <= 8'b0000_0100;
                    col4 <= 8'b0000_1000;
                    col5 <= 8'b0111_1111;
                    col6 <= 8'b0000_0000;
                end
            6'b01_1000: // "O"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0011_1110;
                    col2 <= 8'b0100_0001;
                    col3 <= 8'b0100_0001;
                    col4 <= 8'b0100_0001;
                    col5 <= 8'b0011_1110;
                    col6 <= 8'b0000_0000;
                end
            6'b01_1001: // "P"
                begin
```

```
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0111_1111;
                    col2 <= 8'b0000_1001;
                    col3 <= 8'b0000_1001;
                    col4 <= 8'b0000_1001;
                    col5 <= 8'b0000_0110;
                    col6 <= 8'b0000_0000;
                end
            6'b01_1010: // "Q"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0011_1110;
                    col2 <= 8'b0100_0001;
                    col3 <= 8'b0101_0001;
                    col4 <= 8'b0110_0001;
                    col5 <= 8'b0111_1110;
                    col6 <= 8'b0000_0000;
                end
            6'b01_1011: // "R"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0111_1111;
                    col2 <= 8'b0000_1001;
                    col3 <= 8'b0001_1001;
                    col4 <= 8'b0010_1001;
                    col5 <= 8'b0100_0110;
                    col6 <= 8'b0000_0000;
                end
            6'b01_1100: // "S"
```

```
                begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0010_0110;
                        col2 <= 8'b0100_1001;
                        col3 <= 8'b0100_1001;
                        col4 <= 8'b0100_1001;
                        col5 <= 8'b0011_0010;
                        col6 <= 8'b0000_0000;
                end
        6'b01_1101: // "T"
                begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0000_0001;
                        col2 <= 8'b0000_0001;
                        col3 <= 8'b0111_1111;
                        col4 <= 8'b0000_0001;
                        col5 <= 8'b0000_0001;
                        col6 <= 8'b0000_0000;
                end
        6'b01_1110: // "U"
                begin
                        col0 <= 8'b0000_0000;
                        col1 <= 8'b0011_1111;
                        col2 <= 8'b0100_0000;
                        col3 <= 8'b0100_0000;
                        col4 <= 8'b0100_0000;
                        col5 <= 8'b0011_1111;
                        col6 <= 8'b0000_0000;
                end
```

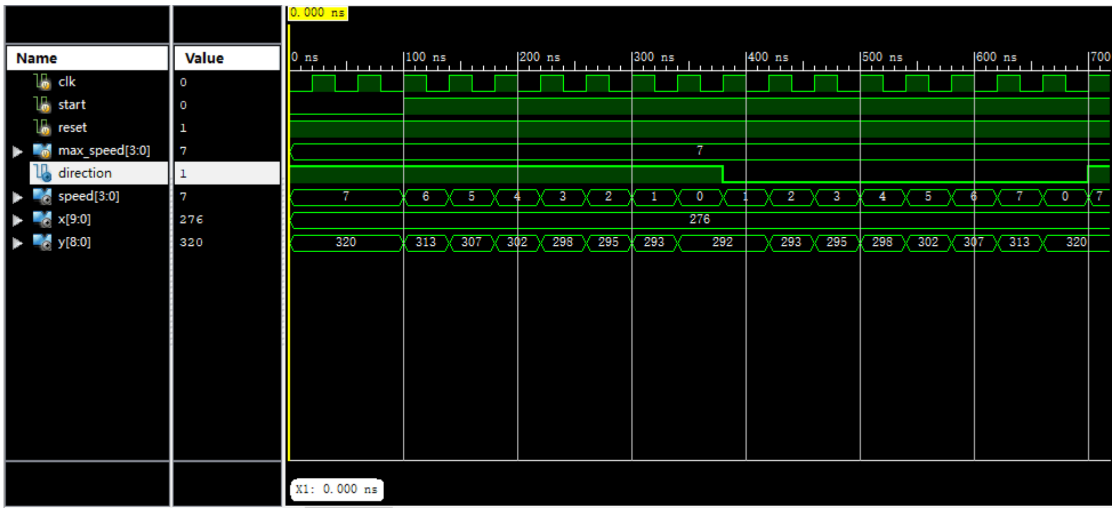
```
6'b01_1111: // "V"
        begin
                col0 <= 8'b0000_0000;
                col1 <= 8'b0001_1111;
                col2 <= 8'b0010_0000;
                col3 <= 8'b0100_0000;
                col4 <= 8'b0010_0000;
                col5 <= 8'b0001_1111;
                col6 <= 8'b0000_0000;
        end
6'b10_0000: // "W"
        begin
                col0 <= 8'b0000_0000;
                col1 <= 8'b0011_1111;
                col2 <= 8'b0100_0000;
                col3 <= 8'b0011_0000;
                col4 <= 8'b0100_0000;
                col5 <= 8'b0011_1111;
                col6 <= 8'b0000_0000;
        end
6'b10_0001: // "X"
        begin
                col0 <= 8'b0000_0000;
                col1 <= 8'b0110_0011;
                col2 <= 8'b0001_0100;
                col3 <= 8'b0000_1000;
                col4 <= 8'b0001_0100;
                col5 <= 8'b0110_0011;
                col6 <= 8'b0000_0000;
                end
            6'b10_0010: // "Y"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0000_0011;
                    col2 <= 8'b0000_0100;
                    col3 <= 8'b0111_1000;
                    col4 <= 8'b0000_0100;
                    col5 <= 8'b0000_0011;
                    col6 <= 8'b0000_0000;
                end
            6'b10_0011: // "Z"
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0110_0001;
                    col2 <= 8'b0101_0001;
                    col3 <= 8'b0100_1001;
                    col4 <= 8'b0100_0101;
                    col5 <= 8'b0100_0011;
                    col6 <= 8'b0000_0000;
                end
            6'b11_1110: // " "
                begin
                    col0 <= 8'b0000_0000;
                    col1 <= 8'b0000_0000;
                    col2 <= 8'b0000_0000;
                    col3 <= 8'b0000_0000;
                    col4 <= 8'b0000_0000;
                    col5 <= 8'b0000_0000;
```

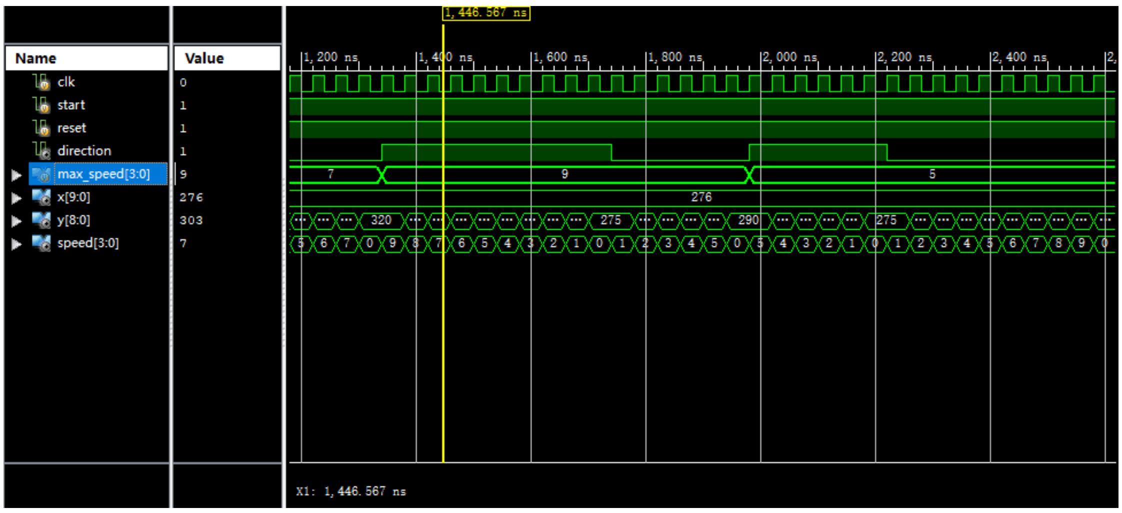
```
                                        col6 <= 8'b0000_0000;
                                    end
                                6'b11_1111: // ":"
                                    begin
                                        col0 <= 8'b0000_0000;
                                        col1 <= 8'b0000_0000;
                                        col2 <= 8'b0011_0110;
                                        col3 <= 8'b0011_0110;
                                        col4 <= 8'b0000_0000;
                                        col5 <= 8'b0000_0000;
                                        col6 <= 8'b0000_0000;
                                    end
                                default: // "*"
                                    begin
                                        col0 <= 8'b0000_0000;
                                        col1 <= 8'b0010_0010;
                                        col2 <= 8'b0001_0100;
                                        col3 <= 8'b0000_1000;
                                        col4 <= 8'b0001_0100;
                                        col5 <= 8'b0010_0010;
                                        col6 <= 8'b0000_0000;
                                    end
                            endcase
                        end
                end
endmodule
```

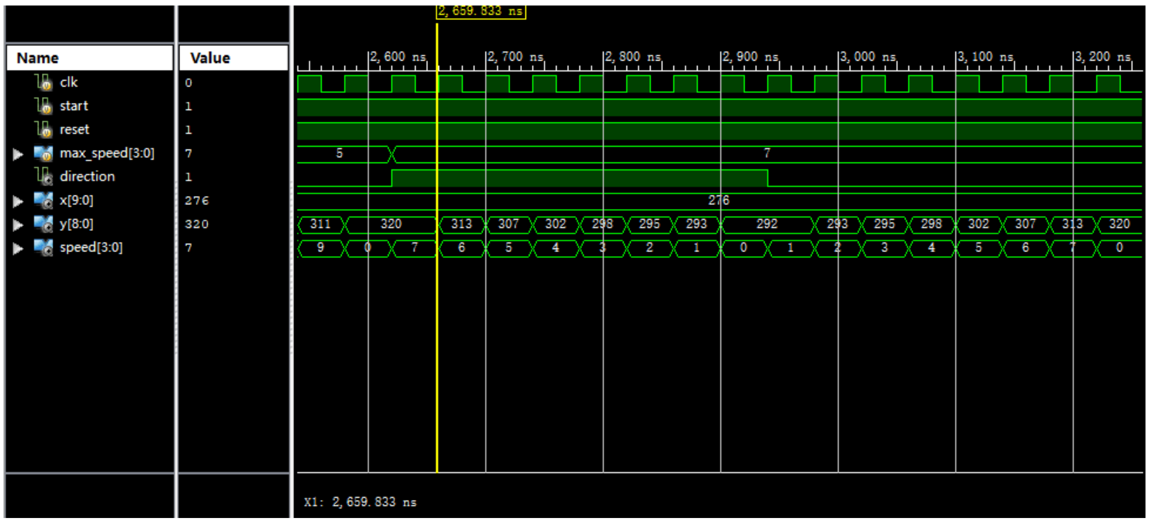
第4章 系统测试验证与结果分析（拍照和介绍输入输出）

4.1 仿真与调试

1. bounce 模块的仿真







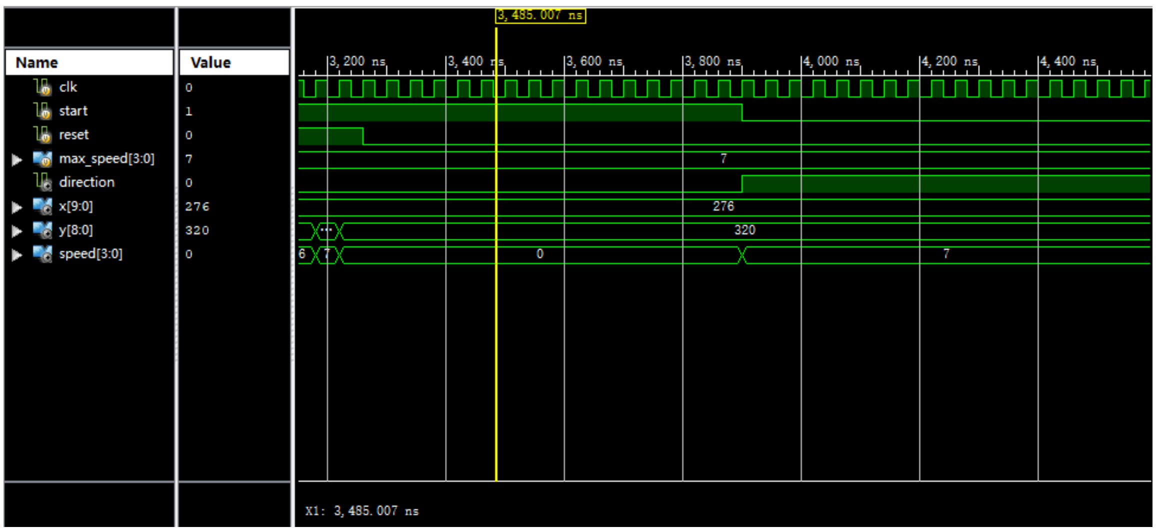


图 2 bounce 的仿真波形图

前 100ms，start=0 ，游戏为静止状态，可以看到小球的 x，y 坐标是不变的。

从 100ms 开始，start =1 ，max_speed = 7 游戏进行，可以看到小球的速度是 7-0-7 的变化，方向标志位 1-0-1 的变化，y 的变化也是先减小后增大。 过程中小球模拟了自由落体的运动。

第一张：这个情况下，max_speed = 9，从而可以看见小球的速度变化为 9-0-5，方向 direction 变化为 1-0-1，可以看见一个跳跃周期后，小球的 y 坐标变大了，从而实现了上跳

第二张：这个情况下，max_speed = 5，从而可以看见小球的变化速度为 5-0-9，方向 direction 变化为 1-0-1，可以看见一个跳跃周期后，小球的 y 坐标变小了，从而实现了下坠

第三张：这个情况下，max_speed 又被设置为 7，相当于回到无按键响应的正常状态，效果同第一张图。

第四章：这个情况下，reset = 0，start = 0，此刻为重置功能，可以看见小球的 x，y 坐标等各种标志值均恢复为初始值，表示游戏重置。

2. barrier 模块的仿真

简单地根据时钟周期和 barrier_speed 改变十个柱子的 x 坐标，以及 height 高度，可以从仿真中看到，y 是不会改变的，而 x 一直在以一定的（barrier_speed)的速度减少。

仿真图如下：

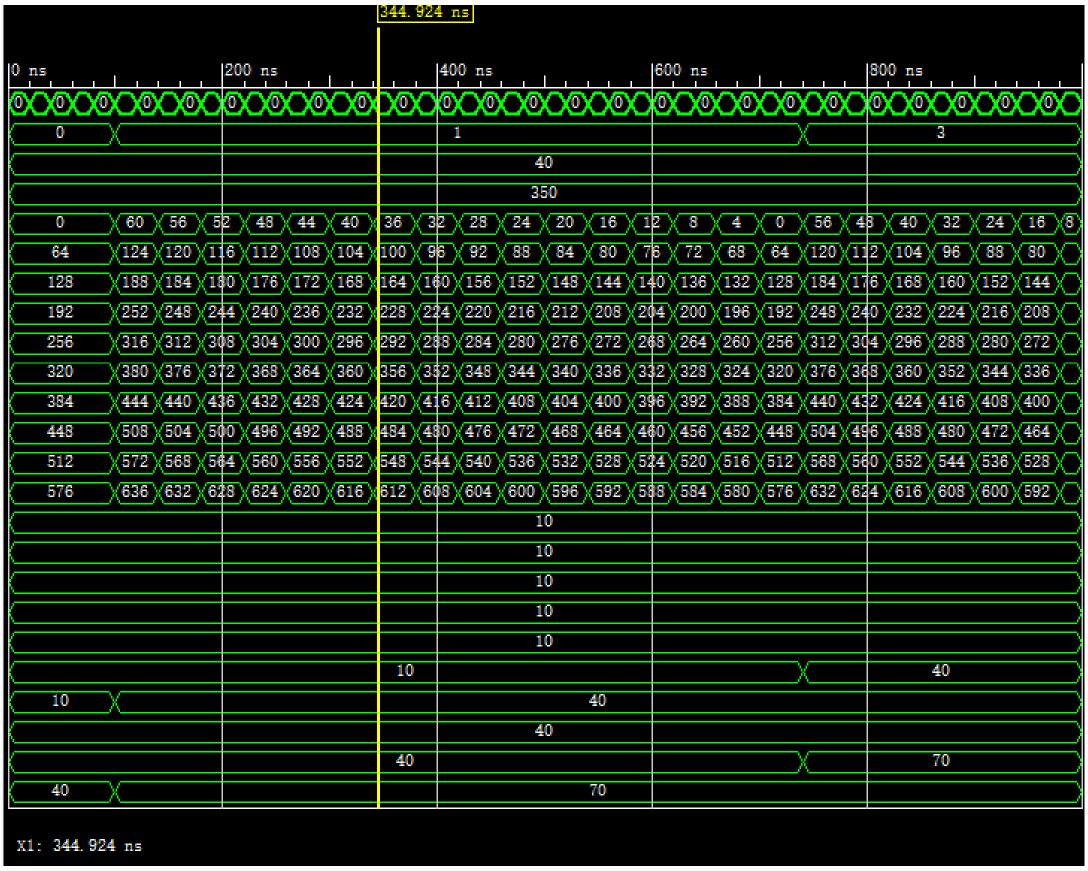


图 3 barrier 的仿真波形图

仿真里设置了一直响应 change 按钮（实际上只在一定的环境下——direction=0，ball_speed=0），根据 height_5，height_6，height_7 的值判断，需要将 max_speed 和 barrier_speed 相应地改变，从而输出到另外两个模块中实现简介控制小球和障碍物地运动状态，这里可以从仿真中输出的值对应三个 height 值发生的变化。

3. change_speed 模块的仿真

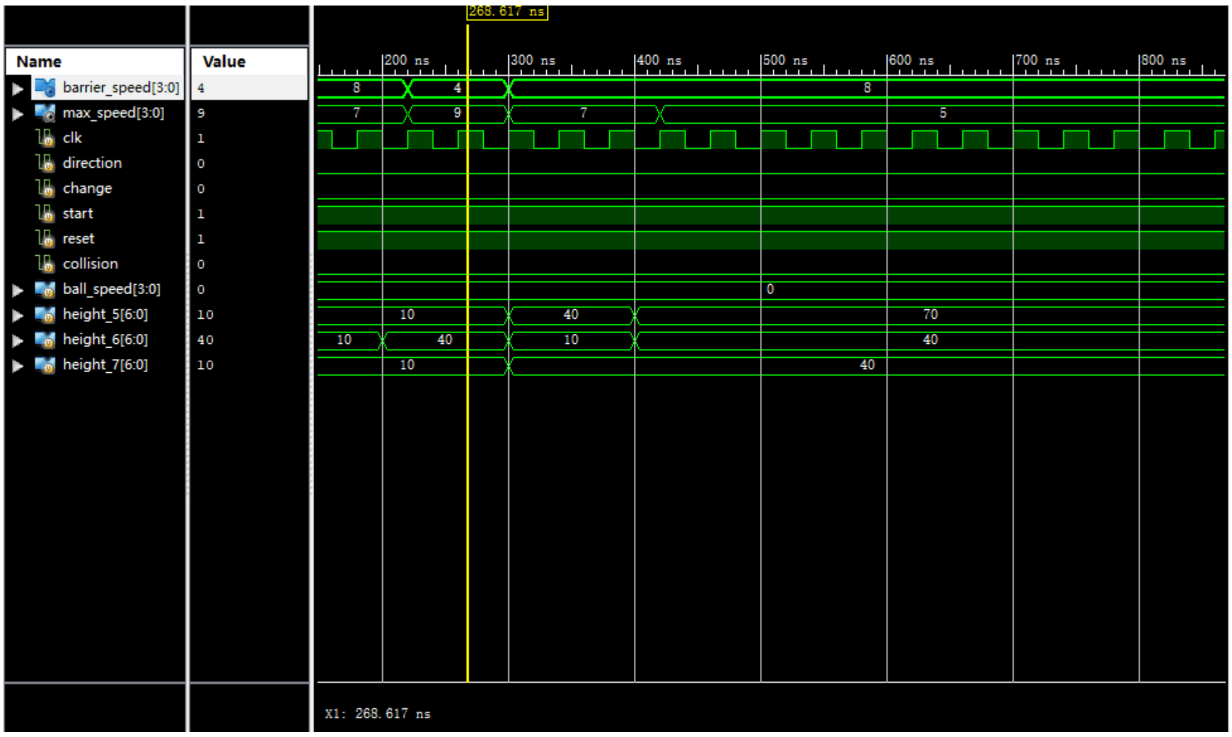


图 4 change_speed 的仿真波形图

4.2 成果展示

4.3.1 开始游戏界面

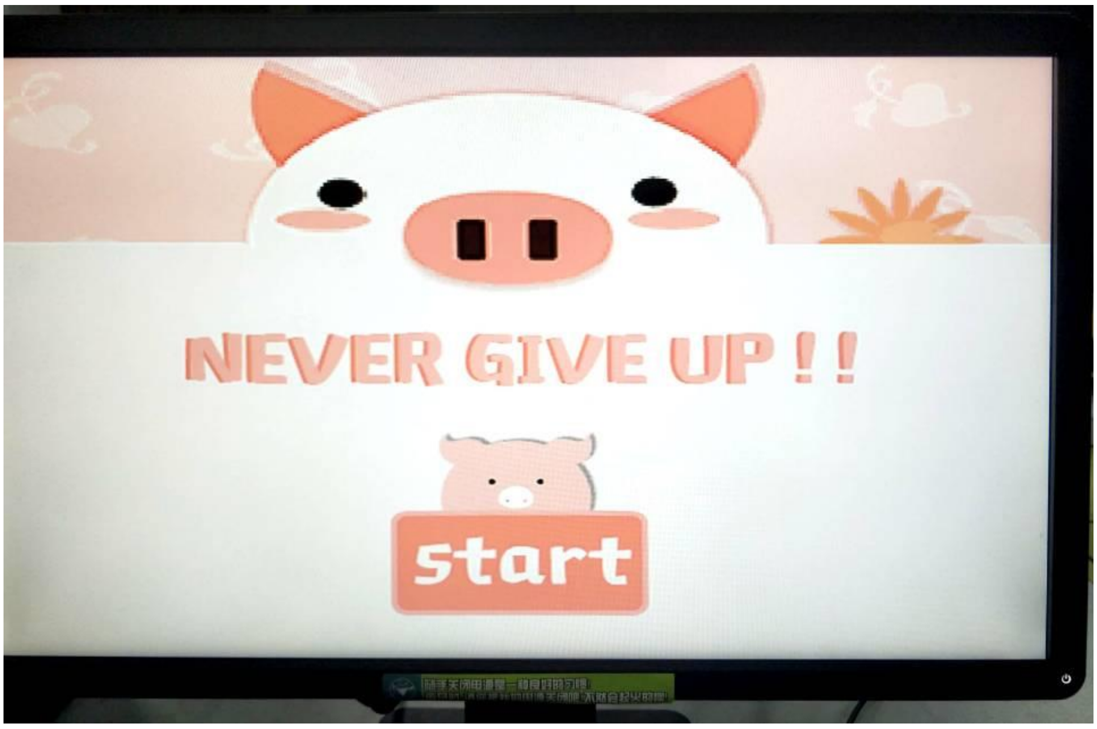


图 5 游戏开始界面

4.3.2 结束游戏界面

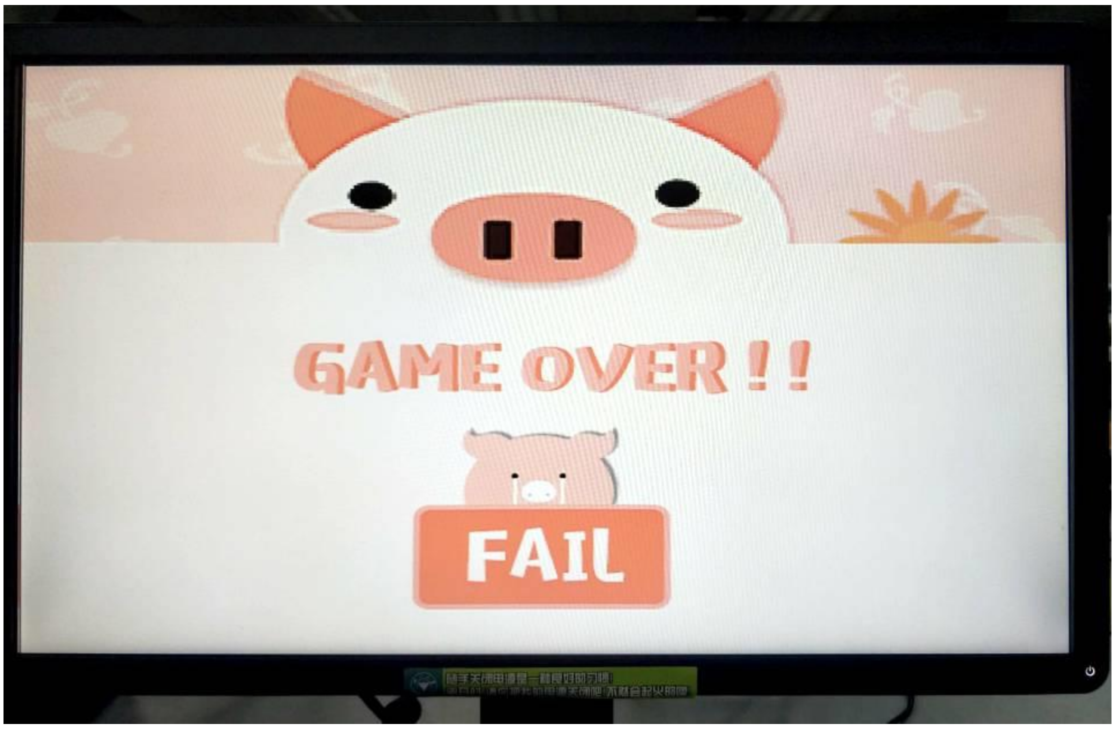


图 6 游戏结束界面

4.3.3 游戏过程中小猪的跳跃

当小猪处于前一个柱子时，如果不按下 button，小猪的跳跃高度不会发生变化，按下 button 时，小猪跳跃高度发生变化，如图：

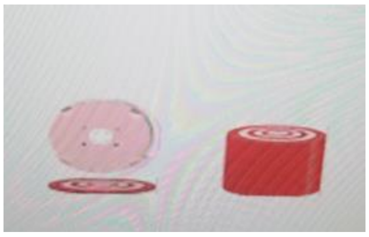
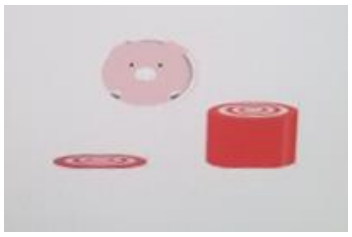
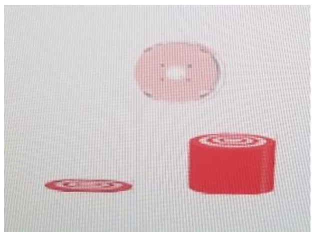


图 7 原始位置



不按下 button



按下 button

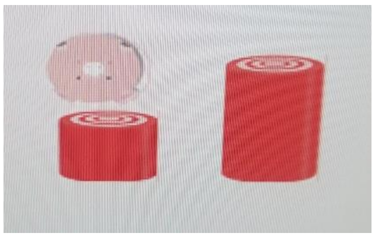
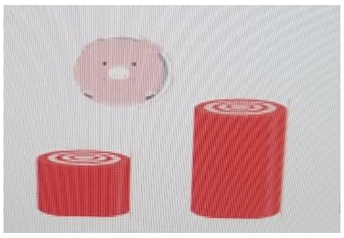
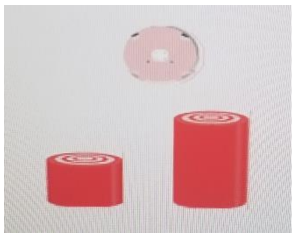


图 8 原始位置



不按下 button



按下 button

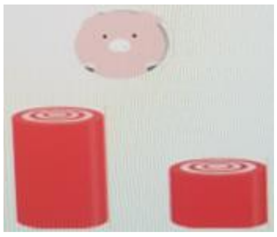
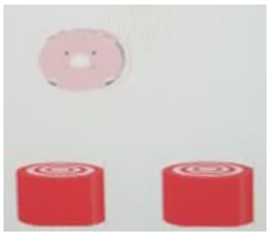
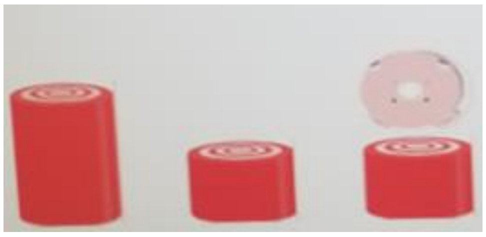




图 9 不按下 button　　　按下 button　　　　　　　按下 button 后降落地点

4.3.4 游戏结束时的七段数码管显示十进制分数

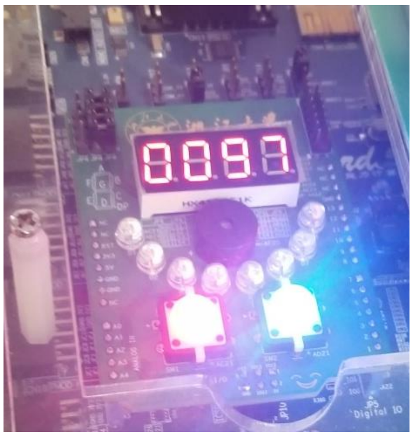


图 10 数码管显示分数

第5章 结论与展望

本次实验之中我们也遇到了很多的困难，最开始的就是对于 verilog 语言的不熟悉。Always 语句内才可以使用的 if-else， reg 和 wire 不一样的特性，都让我们叫苦不迭。印象尤深的是要写一个在不同条件下导入不同的 ip 核这个功能，因为软件语言可以直接在 if-else 下调用函数，带着这个思路，我们一度无法完成该功能。可见软件语言与硬件与语言的巨大差距。最后还是“曲线救国”，将 ip 核算出的 rgb 都存起来，再在 if-else 语句中赋值。

尽管这些有着这些巨大的困难，最终我们还是尽自己所能写完了这个大作业，确实和我们最初提交的报告里所想完成的游戏仍有着不小的差距，在写大作业的过程中，我们收获到的是更加开阔的思想，敢于质疑每一个部分，更敢于质疑自己的成果。做完这份大作业，不仅是对自己的挑战，还有深深的反思，每一次思路的淤塞，都换来了对知识的新的思考，成长很多。