同济大学计算机系

数字逻辑课程综合实验报告

一、实验内容

1、项目名称：小球游戏

2、项目描述：

（1） 游戏组成：小球、挡板、障碍物、计分器。

（2） 游戏模式：单人简单，单人困难，双人简单，双人困难。

（3） 游戏规则：

A. 拨动游戏开始开关，小球开始移动，在左、右、上边界以及挡板处发生弹性碰撞，计分器开始计分。

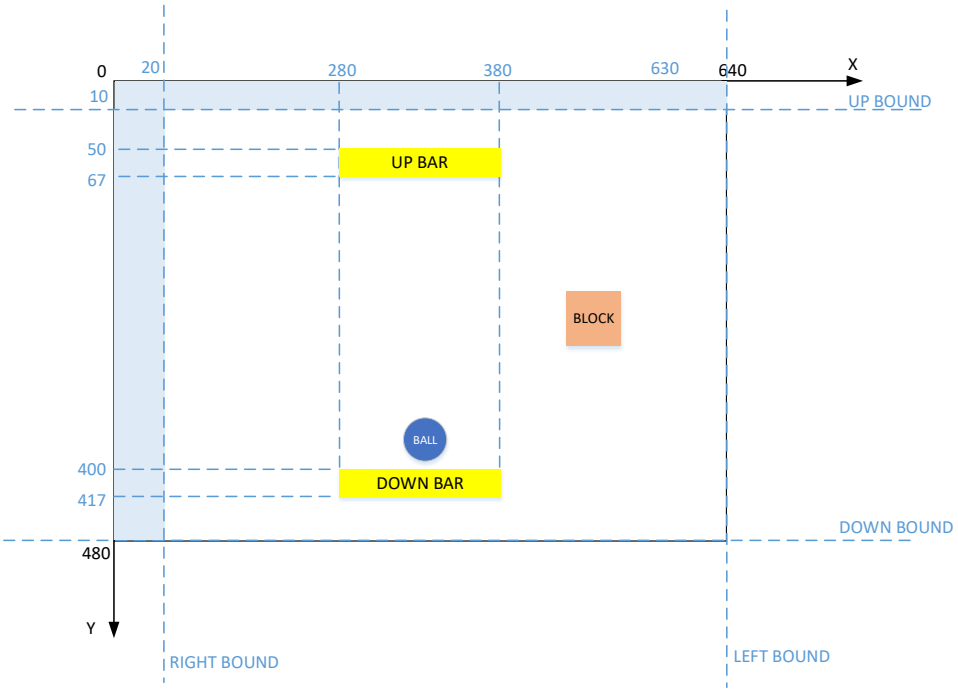
B. 在小球移动碰撞的过程中，挡板接住小球则得分，否则游戏结束。困难模式下，碰到障碍物游戏结束。

C. 单人模式一人操控，仅有下方一块挡板：双人模式两人操控，有上下两块挡板，只有两个配合得当才能取得高分。

D. 简单模式无障碍物，得分 = 游戏速度。
困难模式有障碍物，得分 =2*游戏速度。

E. 可以通过拨码开关调节游戏速度，16 级调速，其中 0000 代表小球、挡板、障碍物（若有）速度为 0，1111 代表速度达到最大值。

（4） 屏幕布局示意图

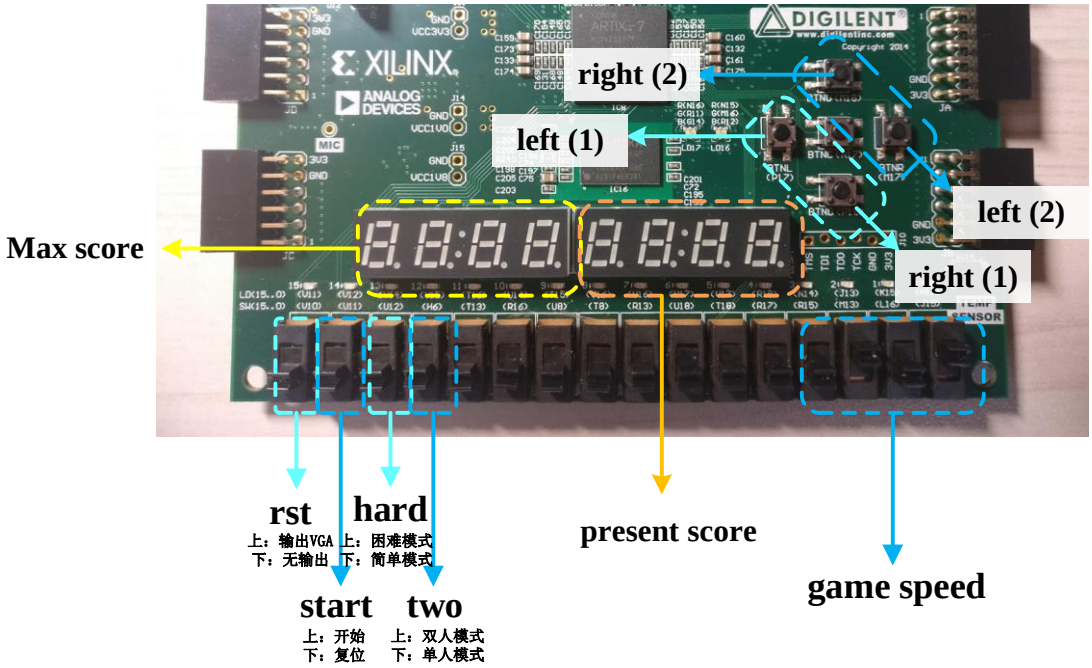


- 屏幕所需时钟：50Mhz。
- 小球圆心初始位置：x=330，y= 390。
- 小球半径：r= 10。
- 下挡板、上挡板（若有）初始位置如图。板长=100。
- 挡板移动速度：
  简单模式：$V_{板}$=游戏速度，困难模式：$V_{板}$=2*游戏速度。
- 障碍物（若有）初始位置：x=100，y=100。
- 障碍物（若有）边长：30。
- 为了能使小球边界处回弹有更好的效果，避免由于时序问题引起的小

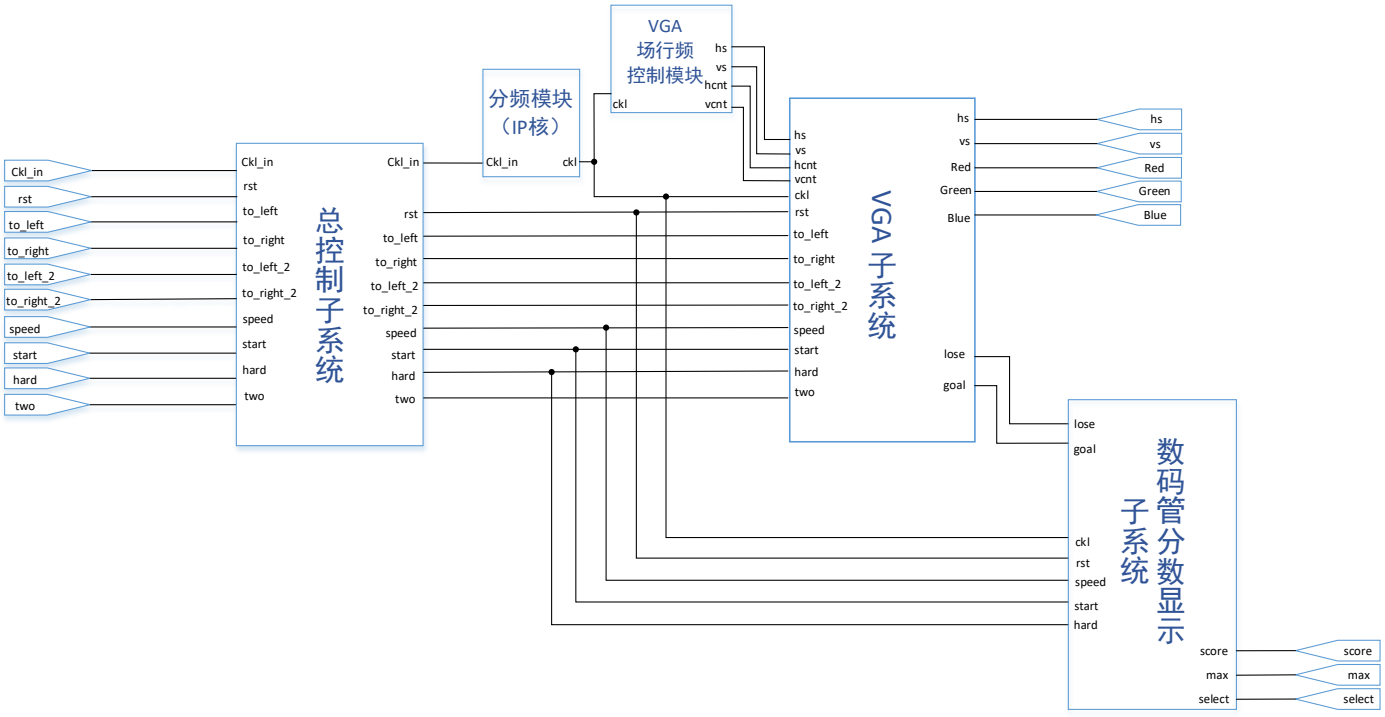
球在边界处变形，游戏界面留出了上边界和左边界。

- 游戏模式不同小球颜色不同。

（5） 开关、按钮、数码管说明



二、小球游戏数字系统总框图

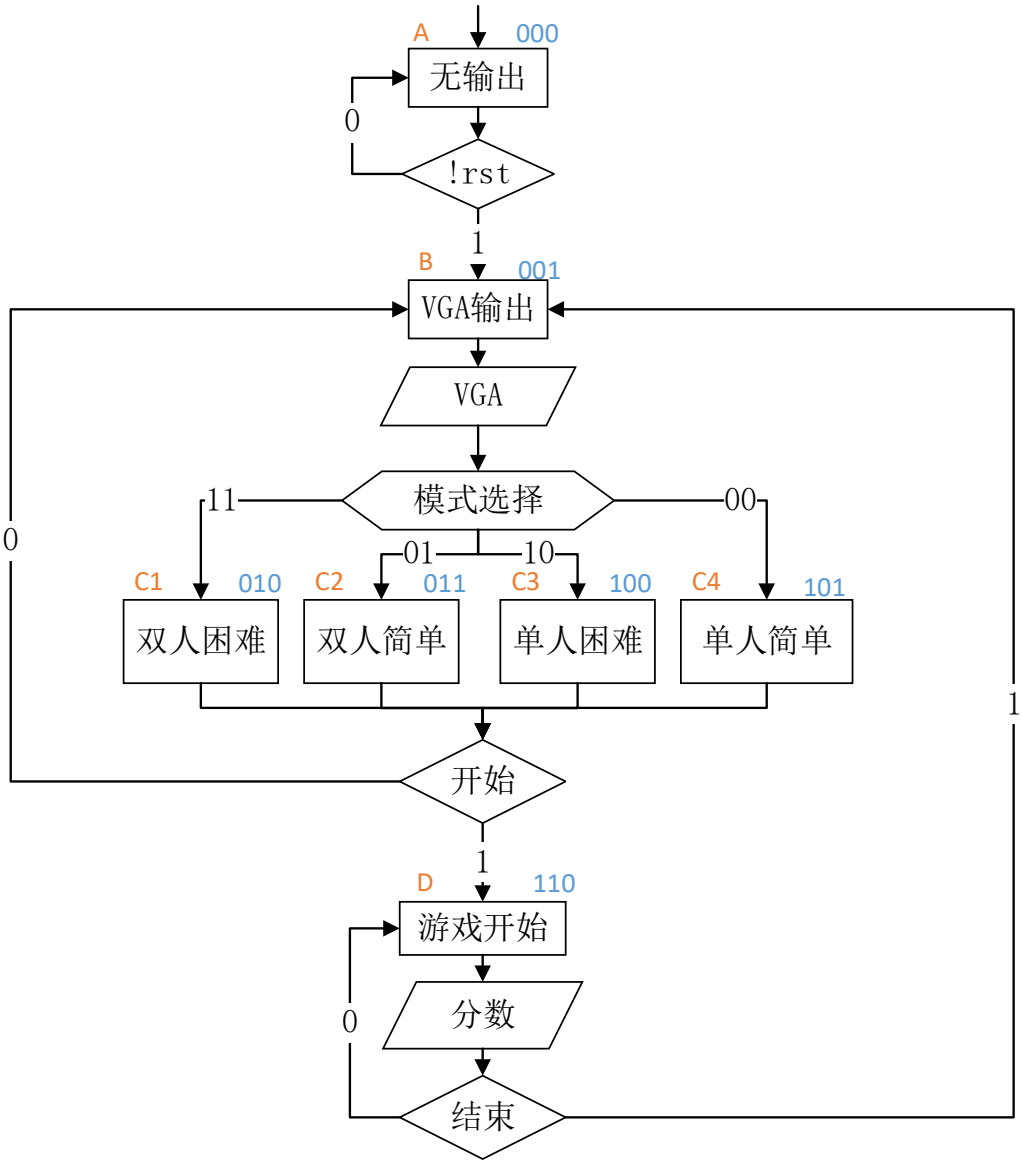


1、总控制子系统：负责调用其他系统，传输数据

2、分频模块：时钟分频
3、VGA 行场频模块：控制 VGA 显示屏的行场频，使其能正常显示
4、VGA 子系统：控制小球、挡板、障碍物的移动和颜色，产生得分、游戏结束信号。
5、LED 分数显示子系统：显示当前分数和最高分。

三、系统控制器设计

1、游戏总控制
（1） ASM 流程图



游戏总控制ASM

（2）状态转移表（多路选择器型控制器）

PS		NS				转换条件			
编码	状态名	状态名	Q_2	Q_1	Q_0				
0 (000)	A	A	0	0	0	$\overline{rst}$	Q_2= 0	Q_1= 0	Q_0= 0
		B	0	0	1	$\overline{!\ rst}$	Q_2= 0	Q_1= 0	Q_0= $\overline{!\ rst}$
1 (001)	B	C1	0	1	0	mn	Q_2= 0	Q_1=mn	Q_0= 0
		C2	0	1	1	$\bar{m}n$	Q_2= 0	Q_1=$\bar{m}n$	Q_0=$\bar{m}n$
		C3	1	0	0	$m\bar{n}$	Q_2=$m\bar{n}$	Q_1= 0	Q_0= 0
		C4	1	0	1	$\overline{mn}$	Q_2=$\overline{mn}$	Q_1= 0	Q_0=$\overline{mn}$
2 (010)	C1	B	0	0	1	$\overline{start}$	Q_2= 0	Q_1= 0	Q_0=$\overline{start}$
		D	1	1	0	start	Q_2= start	Q_1= start	Q_0= 0
3 (011)	C2	B	0	0	1	$\overline{start}$	Q_2= 0	Q_1= 0	Q_0=$\overline{start}$
		D	1	1	0	start	Q_2= start	Q_1= start	Q_0= 0
4 (100)	C3	B	0	0	1	$\overline{start}$	Q_2= 0	Q_1= 0	Q_0=$\overline{start}$
		D	1	1	0	start	Q_2= start	Q_1= start	Q_0= 0
5 (101)	C4	B	0	0	1	$\overline{start}$	Q_2= 0	Q_1= 0	Q_0=$\overline{start}$
		D	1	1	0	start	Q_2= start	Q_1= start	Q_0= 0
6 (110)	D	B	0	0	1	$\overline{end}$	Q_2=$\overline{end}$	Q_1=$\overline{end}$	Q_0= 0
		D	1	1	0	end	Q_2= 0	Q_1= 0	Q_0=end

注：1、rst 为复位信号，低电平有效

2、mn 为模式选择信号，对应四种游戏模式 C1,C2,C3,C4

（3）MUX 数据输入表达式

$MUXQ_2(0)$= 0　　$MUXQ_1(0)$= 0　　$MUXQ_0(0)$= $\overline{!\ rst}$

$MUXQ_2(1)$= $\bar{n}$　　$MUXQ_1(1)$= n　　$MUXQ_0(1)$= $\bar{m}$

$MUXQ_2(2)$= start　　$MUXQ_1(2)$= start　　$MUXQ_0(2)$=$\overline{start}$

$MUXQ_2(3)$= start　　$MUXQ_1(3)$= start　　$MUXQ_0(3)$=$\overline{start}$

$MUXQ_2(4)$= start　　$MUXQ_1(4)$= start　　$MUXQ_0(4)$=$\overline{start}$

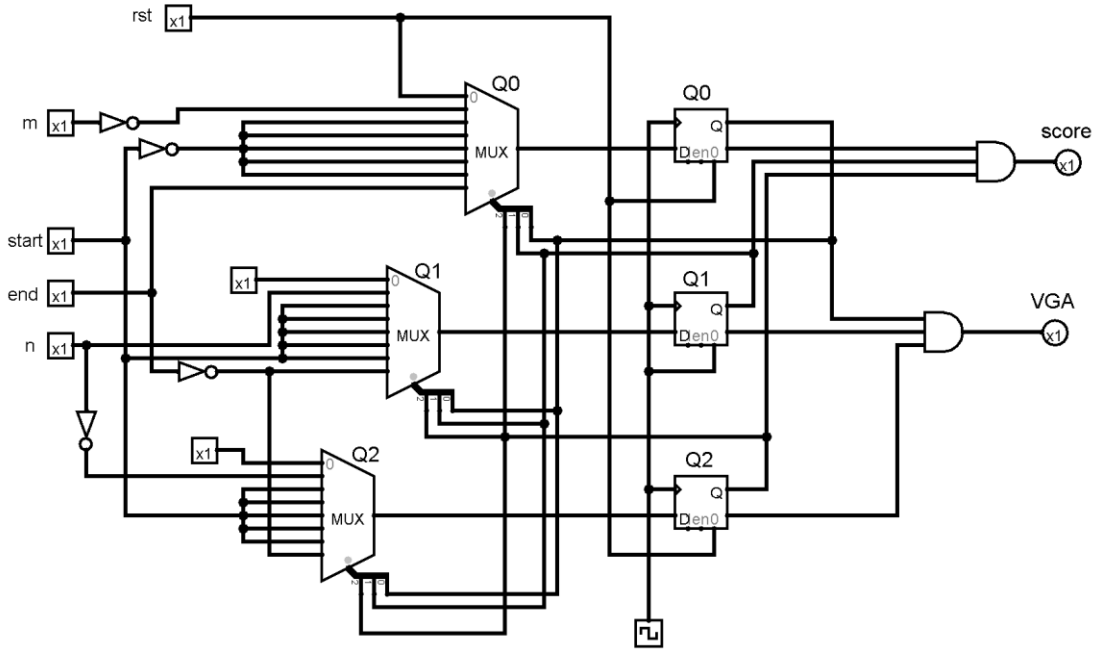
$MUXQ_2(5)$= start　　$MUXQ_1(5)$= start　　$MUXQ_0(5)$=$\overline{start}$

$MUXQ_2(6)$= $\overline{end}$　　$MUXQ_1(6)$= $\overline{end}$　　$MUXQ_0(6)$= end

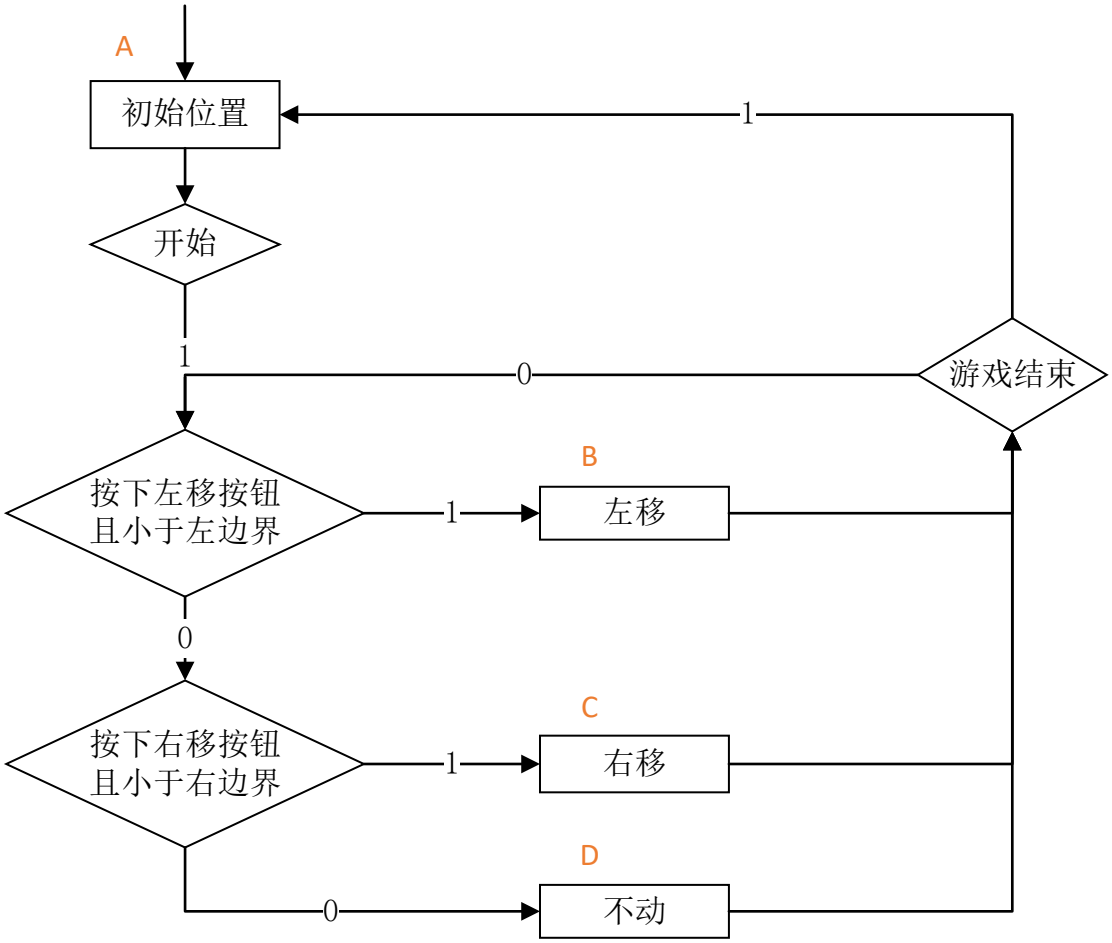
（4）控制信号表达式

VGA=$\overline{Q2Q1}Q0$　　score=$Q2Q1\overline{Q0}$

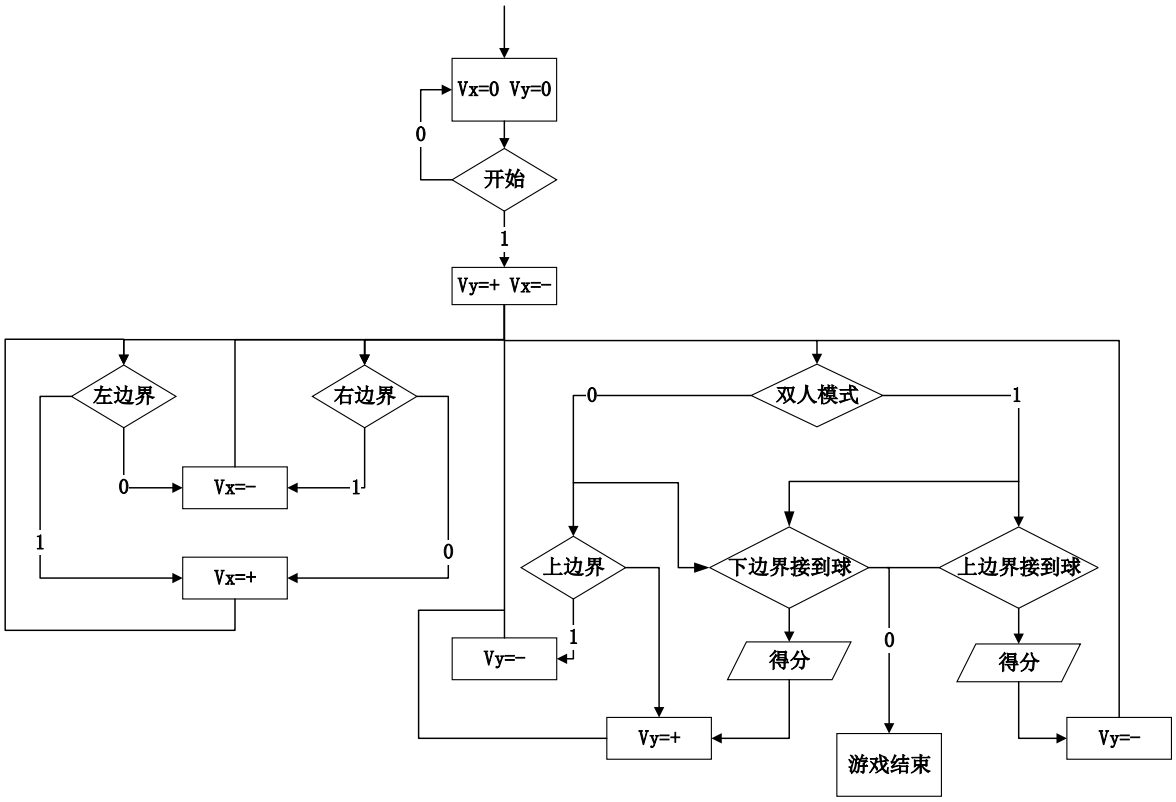
(5)Logisim



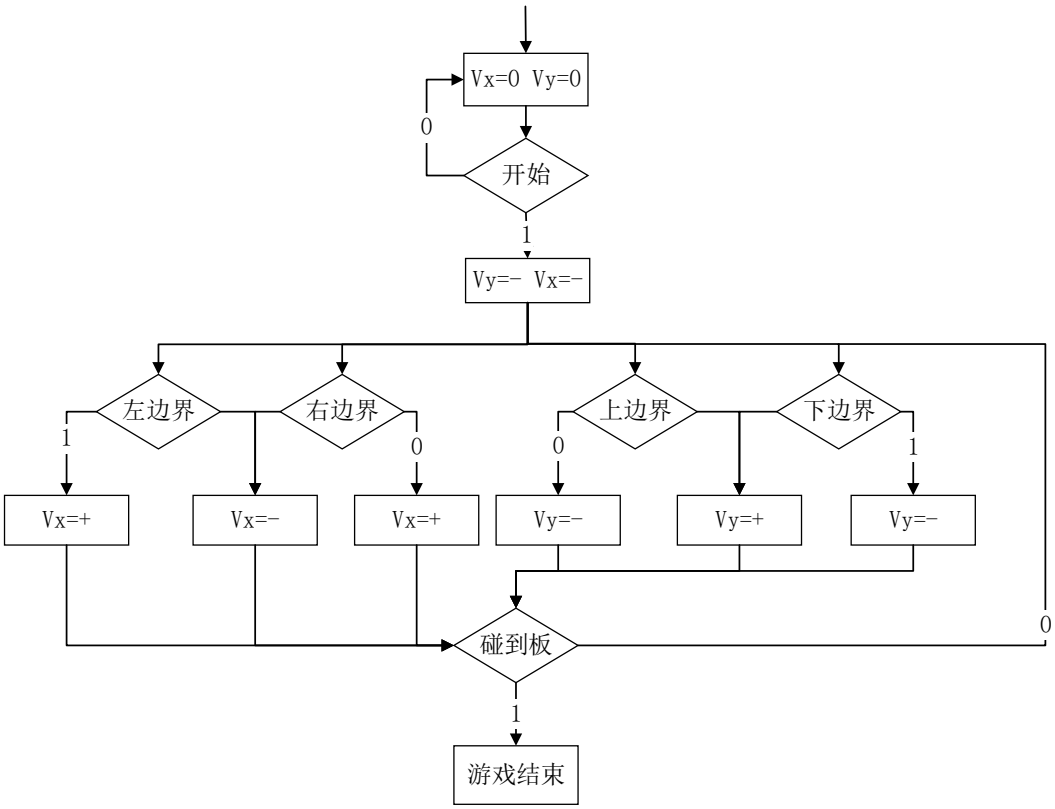
2、小球位置控制


A
初始位置
1
开始
1
0
游戏结束
B
按下左移按钮
且小于左边界
1
左移
0
C
按下右移按钮
且小于右边界
1
右移
D
0
不动

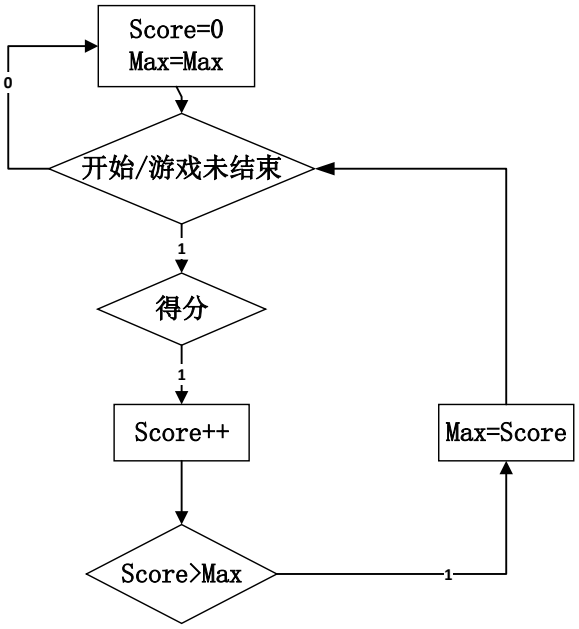
3、小球速度控制



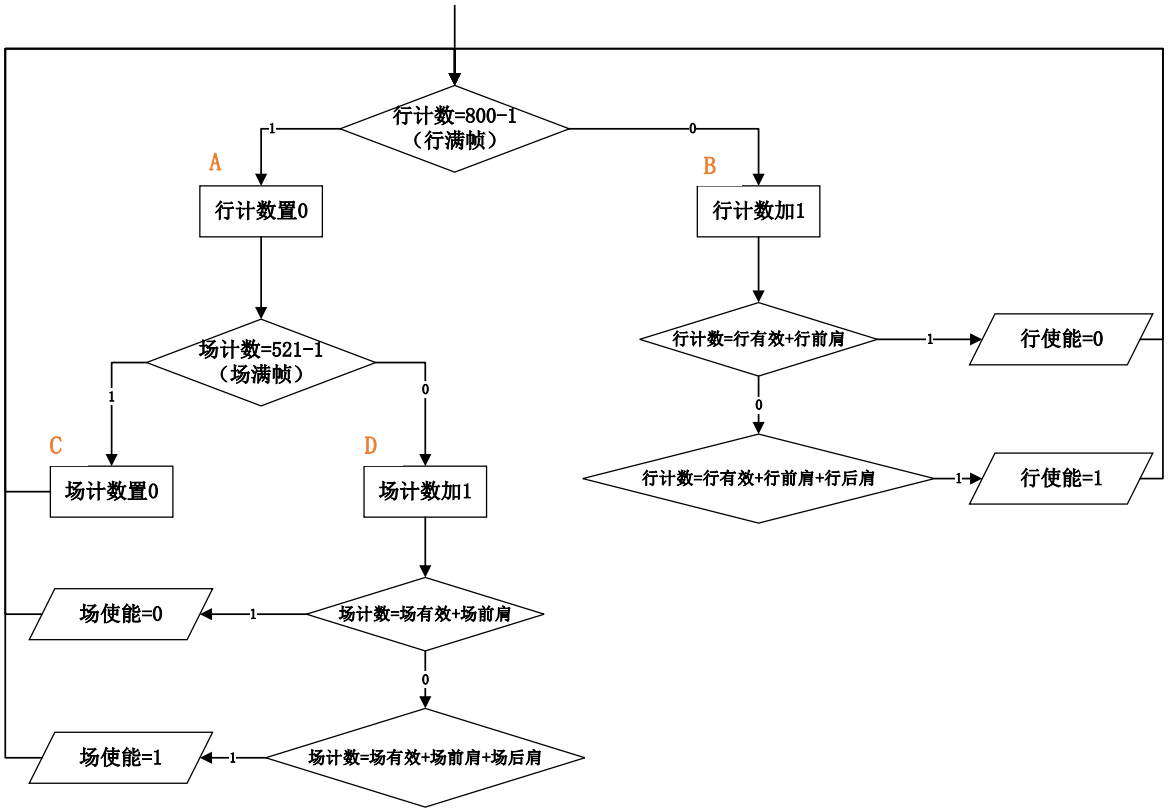
4、障碍物控制



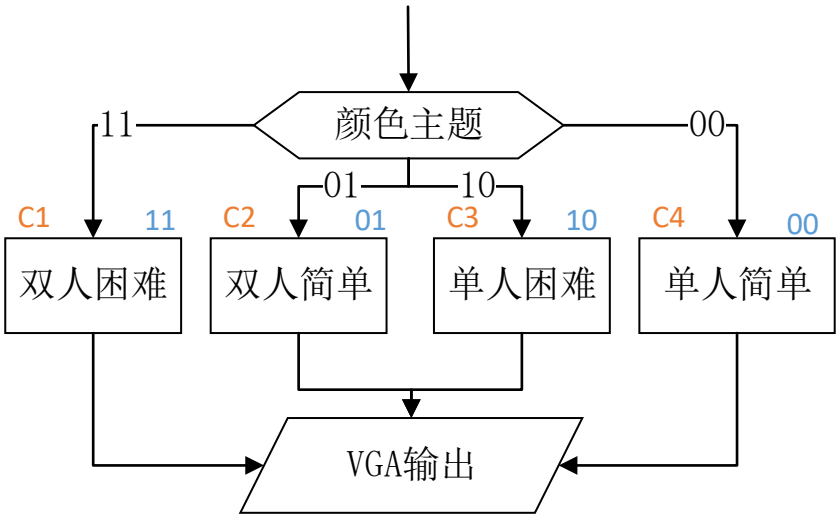
5、分数控制



6、VGA 场行频控制



7、游戏主题颜色控制

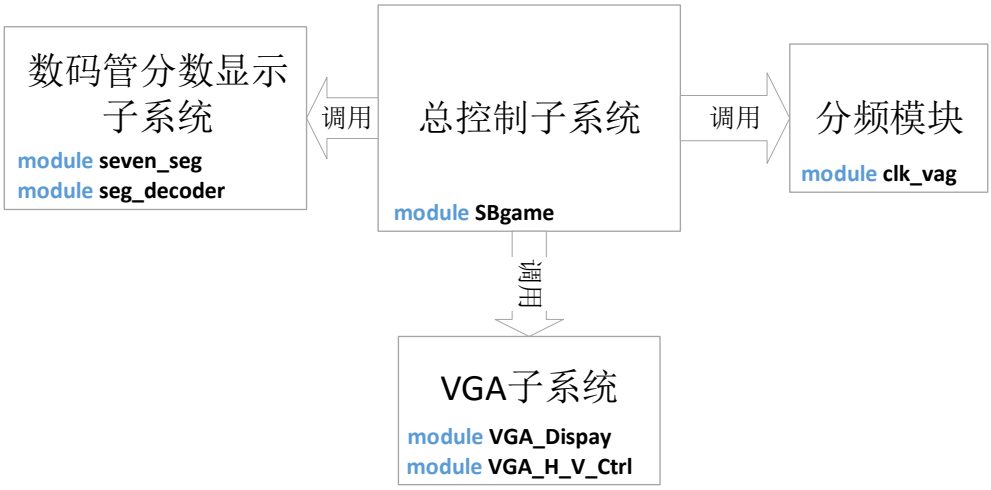


四、子系统模块建模

（该部分要求对实验中的所有子系统模块进行描述，给出各子系统的功能框图及接口信号定义，并列出各模块建模的 verilog 代码）

1、总控制子系统

（1） 功能框图：



（2） 接口定义

```
module SBgame(
        input clk_in,                   //输入时钟
        input rst,                      //复位
        input to_left,                  //下挡板左移
        input to_right,                 //下挡板右移
        input [3:0] bar_move_speed,     //游戏速度
        input start,                    //游戏开始
        input hard,                     //困难模式选择
       input two,                       //双人模式选择
       input to_left_2,                 //上挡板左移
       input to_right_2,                //上挡板右移
```

```
output HSync,                    //VGA hs 输出
output [3:0] OutBlue,            //VGA  蓝色输出
output [3:0] OutGreen,           //VGA  绿色输出
output [3:0] OutRed,             //VGA  红色输出
output VSync,                    //VGA vs 输出
output [7:0] seg_select,         //数码管位选信号
output [6:0] seg_LED             //数码管段选信号
 );
```

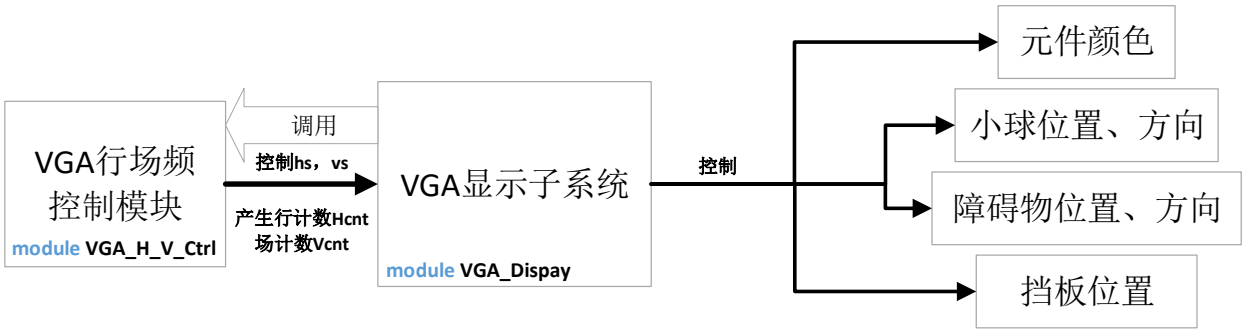
（3） 模块代码

```
module SBgame(
                input clk_in,
                input rst,
                input to_left,
                input to_right,
                input [3:0] bar_move_speed,
                input start,
                input hard,
                  input two,
                  input to_left_2,
                  input to_right_2,
                  output HSync,
                  output [3:0] OutBlue,
                  output [3:0] OutGreen,
                  output [3:0] OutRed,
                  output VSync,
                output [7:0] seg_select,
                output [6:0] seg_LED
      );


wire mclk;
wire lose;
wire goal;

clk_vag clk
 (
 // Clock in ports
  .clk_in(clk_in),
  // Clock out ports
  .clk_out(mclk),
  // Status and control signals
  .resetn(rst)
```

```
  );

VGA_Dispay u_VGA_Disp(
      .clk(mclk),
      .to_left(to_left),
      .to_right(to_right),
      .bar_move_speed(bar_move_speed),
      .start(start),
      .hard(hard),
      .hs(HSync),
      .Blue(OutBlue),
      .Green(OutGreen),
      .Red(OutRed),
      .vs(VSync),
      .lose(lose),
      .goal(goal),
      .rst(rst),
      .two(two),
      .to_left_2(to_left_2),
      .to_right_2(to_right_2)
      );

seven_seg score_board(
      .clk(mclk),
      .rst(rst),
      .goal(goal),
      .hard(hard),
      .lose(lose),
      .start(start),
      .select(seg_select),
      .seg(seg_LED),
       .bar_move_speed(bar_move_speed)
      );


endmodule
```

2、VAG 子系统

（1） 系统框图



（2） 接口定义

```
module VGA_Dispay(
    input clk,                          //输入变频时钟
    input to_left,                      //下挡板左移
    input to_right,                     //下挡板右移
    input [3:0] bar_move_speed,         //游戏速度
    input start,                        //游戏开始
    input hard,                         //困难模式选择
    input rst,                          //复位
    input two,                          //双人模式选择
    input to_left_2,                    //上挡板左移
    input to_right_2,                   //上挡板右移
    output hs,                          //VGA hs
    output vs,                          //VGA vs
    output reg [3:0] Red,               //VGA  红色输出
    output reg [3:0] Green,             //VGA  绿色输出
    output reg [3:0] Blue，             //VGA  蓝色输出
    output lose,                        //游戏结束信号
    output reg goal                     //游戏得分信号
    );

module VGA_H_V_Ctrl(
    input clk,                          //输入变频时钟
    output reg [9:0] Hcnt,              //输出行计数
    output reg [9:0] Vcnt,              //输出场计数
    output reg hs,                      //VGA hs
    output reg vs                       //VGA vs
    );
```

（3） 模块代码

- VGA 显示子系统

```
`include "Definition.h"

module VGA_Dispay(
      input clk,
       input to_left,
       input to_right,
       input [3:0] bar_move_speed,
       input start,
       input hard,
       input rst,
        input two,
        input to_left_2,
        input to_right_2,
        output hs,
        output vs,
        output reg [3:0] Red,
        output reg [3:0] Green,
        output reg [3:0] Blue,
       output    lose,
       output reg goal
      );

        //bound
            parameter UP_BOUND = 10;
            parameter DOWN_BOUND = 480;
            parameter LEFT_BOUND = 20;
            parameter RIGHT_BOUND = 630;

            // Radius of the ball
            parameter ball_r = 10;
            parameter block = 15;
            // The position of the downside bar
            reg [9:0] up_pos = 400;
            reg [9:0] down_pos = 417;
            // The position of the upside bar
           reg [9:0] up_pos_2 = 50;
            reg [9:0] down_pos_2 = 67;

      //end game
      reg end_g_b=0;
      reg end_g_k=0;
      wire end_g;
```

```
assign end_g = end_g_b | end_g_k;
//lose
reg lose_b =0;
reg lose_k =0;
 assign lose = lose_b | lose_k;

//register definition
wire [9:0] Hcnt;          // horizontal counter    if = PLD-1 -> Hcnt <= 0
wire [9:0] Vcnt;          // verical counter    if = LFD-1 -> Vcnt <= 0

reg h_speed = `RIGHT;
reg v_speed = `UP;

//block
reg h_speed_b = `RIGHT;
 reg v_speed_b = `UP;

// The position of the bar

reg [9:0] left_pos = 280;
reg [9:0] right_pos = 380;
 reg [9:0] left_pos_2 = 280;
 reg [9:0] right_pos_2 = 380;



// The circle heart position of the ball / beginning
reg [9:0] ball_x_pos = 330;
reg [9:0] ball_y_pos = 390;

// The center position of the block / beginning
reg [9:0] block_x_pos = 100;
 reg [9:0] block_y_pos = 100;

//--------------generate hs vs----------------
VGA_H_V_Ctrl generate_vga_t(
.clk(clk),
 .Hcnt(Hcnt),
 .Vcnt(Vcnt),
 .hs(hs),
 .vs(vs)
);
```

```
//---------------color control----------------------
                //Display the downside bar and the ball
always @ (posedge clk)
begin
                    if(!two)
                    begin
                    // Display the downside bar
                    if (Vcnt>=up_pos && Vcnt<=down_pos
                            && Hcnt>=left_pos && Hcnt<=right_pos)
                       if(hard)
                          begin
                              Red <= 4'b1000;
                              Green <= 4'b1111;
                              Blue <= 4'b1111;
                          end
                       else
                          begin
                             Red <= 4'b1111;
                             Green <= 4'b1111;
                             Blue <= 4'b0000;
                          end
                    // Display the ball
                    else if ( (Hcnt - ball_x_pos)*(Hcnt - ball_x_pos) + (Vcnt -
ball_y_pos)*(Vcnt - ball_y_pos) <= (ball_r * ball_r))
                    if (hard)
                    begin
                       Red <= 4'b0111;
                       Green <= 4'b0000;
                       Blue <= 4'b0111;
                    end
                    else
                       begin
                          Red <= 4'b0000;
                          Green <= 4'b1111;
                          Blue <= 4'b1111;
                       end
                    // Display the block
                    else if((Hcnt - block_x_pos)*(Hcnt - block_x_pos)<= block*block
&& (Vcnt - block_y_pos)*(Vcnt - block_y_pos)<= block*block    )
                    begin
                       //hard mode
                       if(hard)
                       begin
                          Red <= 4'b0000;
```

```
                Green <= 4'b1110;
                Blue <= 4'b1110;
            end
        end
        else
        begin
                Red <= 4'b0000;
                Green <= 4'b0000;
                Blue <= 4'b0000;
        end
    end
    else//two mode
    begin
        // Display the upside bar
                    if (Vcnt>=up_pos_2 && Vcnt<=down_pos_2
                                                    &&
Hcnt>=left_pos_2 && Hcnt<=right_pos_2)
                        if(hard)
                            begin
                                Red <= 4'b1000;
                                Green <= 4'b1111;
                                Blue <= 4'b1111;
                            end
                        else
                            begin
                                Red <= 4'b1100;
                                Green <= 4'b1110;
                                Blue <= 4'b0000;
                            end
                        // Display the upside bar
                        else if (Vcnt>=up_pos && Vcnt<=down_pos
                                    &&      Hcnt>=left_pos      &&
Hcnt<=right_pos)
                            if(hard)
                                begin
                                    Red <= 4'b1000;
                                    Green <= 4'b1111;
                                    Blue <= 4'b1111;
                                end
                            else
                                begin
                                    Red <= 4'b1100;
                                    Green <= 4'b1110;
                                    Blue <= 4'b0000;
```

```
                                        end
                                    // Display the ball
                                    else if ( (Hcnt - ball_x_pos)*(Hcnt - ball_x_pos)
+ (Vcnt - ball_y_pos)*(Vcnt - ball_y_pos) <= (ball_r * ball_r))
                                    if (hard)
                                    begin
                                       Red <= 4'b0111;
                                       Green <= 4'b0000;
                                       Blue <= 4'b0111;
                                    end
                                    else
                                       begin
                                          Red <= 4'b0110;
                                          Green <= 4'b1101;
                                          Blue <= 4'b1101;
                                       end
                                    // Display the block
                                    else if((Hcnt - block_x_pos)*(Hcnt -
block_x_pos)<= block*block && (Vcnt - block_y_pos)*(Vcnt - block_y_pos)<=
block*block  )
                                    begin
                                       //hard mode
                                       if(hard)
                                       begin
                                          Red <= 4'b0000;
                                          Green <= 4'b1110;
                                          Blue <= 4'b1110;
                                       end
                                    end
                                    else
                                    begin
                                          Red <= 4'b0000;
                                          Green <= 4'b0000;
                                          Blue <= 4'b0000;
                                    end
                        end
end

//------------------------position control-------
     always @ (posedge vs)
    begin
    if(!start||end_g==1)
        begin
           left_pos = 280;
```

```
        right_pos = 380;
        ball_y_pos = 390;
        ball_x_pos = 330;
        //hard mode
        if(hard)
           begin
              block_x_pos <= 100;
              block_y_pos <= 100;
           end
        //two players
        if(two)
        begin
           left_pos_2 = 280;
           right_pos_2 = 380;
        end
     end
else if(start)
   begin
          // movement of the downside bar
           if (to_left && left_pos >= LEFT_BOUND)
           begin
                if(hard)
                begin
                   left_pos <= left_pos - 2*bar_move_speed;
                      right_pos <= right_pos - 2*bar_move_speed;
                end
                else
                begin
                         left_pos <= left_pos - bar_move_speed;
                    right_pos <= right_pos - bar_move_speed;
                end
            end
           else if(to_right && right_pos <= RIGHT_BOUND)
           begin
              if(hard)
                   begin
                   left_pos <= left_pos + 2*bar_move_speed;
                   right_pos <= right_pos + 2*bar_move_speed;
                   end
              else
                 begin
                left_pos <= left_pos + bar_move_speed;
                right_pos <= right_pos + bar_move_speed;
                   end
```

```
        end

        //movement of the upside bar
        if(two)
        begin
                if (to_left_2 && left_pos_2 >= LEFT_BOUND)
                begin
                        if(hard)
                        begin
                           left_pos_2 <= left_pos_2 - 3*bar_move_speed;
                           right_pos_2 <= right_pos_2 -
3*bar_move_speed;
                        end
                        else
                        begin
                           left_pos_2 <= left_pos_2 - 2*bar_move_speed;
                           right_pos_2 <= right_pos_2 -
2*bar_move_speed;
                        end
                 end
                else if(to_right_2 && right_pos_2 <= RIGHT_BOUND)
                begin
                   if(hard)
                        begin
                        left_pos_2 <= left_pos_2 + 3*bar_move_speed;
                        right_pos_2 <= right_pos_2 + 3*bar_move_speed;
                        end
                   else
                        begin
                        left_pos_2 <= left_pos_2 + 2*bar_move_speed;
                        right_pos_2 <= right_pos_2 + 2*bar_move_speed;
                        end
                end
        end

       //movement of the ball
        if (v_speed == `UP) // go up
             ball_y_pos <= ball_y_pos - bar_move_speed;
          else //go down
             ball_y_pos <= ball_y_pos + bar_move_speed;
        if (h_speed == `RIGHT) // go right
             ball_x_pos <= ball_x_pos + bar_move_speed;
          else //go down
             ball_x_pos <= ball_x_pos - bar_move_speed;
```

```
                    //hard mode
                    if(hard)
                        begin
                     //movement of the block
                      if (v_speed_b == `UP) // go down
                            block_y_pos <= block_y_pos - bar_move_speed;
                        else //go down
                            block_y_pos <= block_y_pos + bar_move_speed;
                        if (h_speed_b == `RIGHT) // go left
                            block_x_pos <= block_x_pos + bar_move_speed;
                        else //go left
                            block_x_pos <= block_x_pos - bar_move_speed;
                         end
            end
        end



//------------ball directions control ---------------------
//change directions when reach the edge or crush the bar
      always @ (negedge vs)
     begin
            if(!two)
            begin
                if (ball_y_pos <= UP_BOUND)      // Here, all the jugement should
use >= or <= instead of ==
                begin
                   goal = 0;
                  v_speed <= 1;                        // Because when the offset is more than
1, the axis may step over the line
                   lose_b <= 0;
                end

                //downside
                else if (ball_y_pos >= (up_pos - ball_r) && ball_x_pos <= right_pos &&
ball_x_pos >= left_pos)
                begin
                   goal <= 1;
                     v_speed <= 0;
                     end_g_b <= 1'b0;
                 end
                 //downside
                else if (ball_y_pos >= down_pos && ball_y_pos < (DOWN_BOUND -
```

```
ball_r))
                begin
                      // miss the ball
                     //end game
                   goal <= 0;
                   end_g_b <= 1'b1;
                   lose_b <= 1;
                end
                else if (ball_y_pos >= (DOWN_BOUND - ball_r + 1))
                    v_speed <= 0;
                 else
                    begin
                       goal <=0;
                       v_speed <= v_speed;
                       end_g_b <= 1'b0;
                    end

                 if (ball_x_pos <= LEFT_BOUND)
                       h_speed <= 1;
                 else if (ball_x_pos >= RIGHT_BOUND)
                       h_speed <= 0;
                 else
                       h_speed <= h_speed;
           end
           else//two mode
           begin
           // upside
                 if (ball_y_pos <= UP_BOUND)      // Here, all the jugement should
use >= or <= instead of ==
                    begin
                              goal = 0;
                              v_speed <= 1;                    // Because when the offset is
more than 1, the axis may step over the line
                              lose_b <= 0;
                    end
               else if (ball_y_pos<= (down_pos_2+ball_r) &&ball_y_pos >=
(up_pos_2+10) && ball_x_pos <= right_pos_2 && ball_x_pos >= left_pos_2)
                         begin
                            goal <=1;
                            v_speed <= 1;
                            lose_b <= 0;
                         end
               else if(ball_y_pos <= ( up_pos_2+10 - 1)&& (ball_x_pos >=
right_pos_2+1 || ball_x_pos <= left_pos_2-1))
```

```
                begin
                    // miss the ball
                    //end game
                    goal <= 0;
                    end_g_b <= 1'b1;
                    lose_b <= 1;
                end
            //downside
            else if ( ball_y_pos >= (up_pos - ball_r) && ball_x_pos <= right_pos &&
ball_x_pos >= left_pos)
                                begin
                                   goal <= 1;
                                   v_speed <= 0;
                                   end_g_b <= 1'b0;
                                end
                                //downside
            else if ( ball_y_pos >= down_pos && ball_y_pos < (DOWN_BOUND -
ball_r))
                                begin
                                      // miss the ball
                                      //end game
                                   goal <= 0;
                                   end_g_b <= 1'b1;
                                        lose_b <= 1;
                                end
            else if (ball_y_pos >= (DOWN_BOUND - ball_r + 1))
                                     v_speed <= 0;
            else
                                   begin
                                      goal <=0;
                                      v_speed <= v_speed;
                                      end_g_b <= 1'b0;
                                   end

            if (ball_x_pos <= LEFT_BOUND)
                                      h_speed <= 1;
            else if (ball_x_pos >= RIGHT_BOUND)
                                      h_speed <= 0;
            else
                                      h_speed <= h_speed;
        end
    end

//    //------------------block control-----------------
```

```
   //change directions block
always @ (negedge vs)
begin
   if(hard)
   begin
          if(!two)
          begin
              if (block_y_pos <= UP_BOUND)
                 begin
                    v_speed_b <= 1;
                    lose_k <= 0;
                    end_g_k <= 1'b0;
                 end
              else if (block_y_pos >= (up_pos + 2) && block_x_pos <= right_pos    &&
block_x_pos >= left_pos)
                  begin    //touch the block
               v_speed_b <= 0;
               end_g_k <= 1'b1;
               lose_k <= 1;
                end
              else if(block_y_pos >= DOWN_BOUND)
                 v_speed_b <= 0;
              else
                 begin
                   v_speed_b <= v_speed_b;
                   lose_k <= 0;
                   end_g_k <= 1'b0;
                 end

              if (block_x_pos <= LEFT_BOUND)
                h_speed_b <= 1;
            else if (block_x_pos >= RIGHT_BOUND)
                h_speed_b <= 0;
            else
                h_speed_b <= h_speed_b;

            end
         else //two mode
            begin
              // upside
                           if (block_y_pos <= UP_BOUND)      // Here, all the jugement
should use >= or <= instead of ==
                              begin
```

```
                    v_speed_b <= 1;
                    lose_k <= 0;
                    end_g_k <= 1'b0;
                end
            else if (block_y_pos<= (down_pos_2+block) &&block_y_pos >= (up_pos_2+10) && block_x_pos <= right_pos_2 && block_x_pos >= left_pos_2)
                    begin
                      v_speed_b <= 0;
                      end_g_k <= 1'b1;
                      lose_k <= 1;
                    end
            else if(block_y_pos >= (up_pos + 2) && block_x_pos <= right_pos   && block_x_pos >= left_pos)
                    begin
                      v_speed_b <= 0;
                      end_g_k <= 1'b1;
                      lose_k <= 1;
                    end
            else if(block_y_pos >= DOWN_BOUND)
                            v_speed_b <= 0;
            else
                     begin
                       v_speed_b <= v_speed_b;
                       lose_k <= 0;
                       end_g_k <= 1'b0;
                     end

                          if (block_x_pos <= LEFT_BOUND)
                            h_speed_b <= 1;
                        else if (block_x_pos >= RIGHT_BOUND)
                            h_speed_b <= 0;
                        else
                            h_speed_b <= h_speed_b;
        end
   end
end

endmodule
```

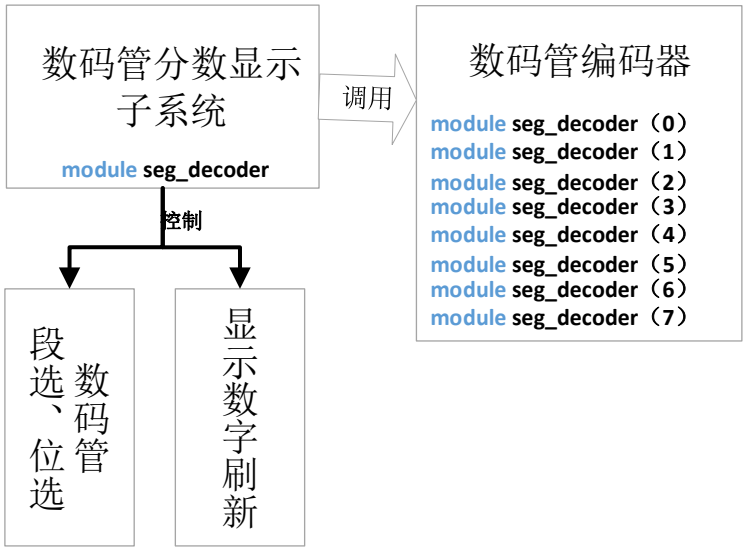
- VGA 行场频控制模块

```
module VGA_H_V_Ctrl(
    input clk,
    output reg [9:0] Hcnt,
    output reg [9:0] Vcnt,
    output reg hs,
    output reg vs
    );

    //parameter definition
        parameter PAL = 640;            //Pixels/Active Line (pixels)
        parameter LAF = 480;            //Lines/Active Frame (lines)
        parameter PLD = 800;            //Pixel/Line Divider(Whole Line)
        parameter LFD = 521;            //Line/Frame Divider(Whole Frame)
        parameter HPW = 96;                 //Horizontal synchro Sync Pulse Width
(pixels)
        parameter HFP = 16;                 //Horizontal synchro Front Porch (pixels)
        parameter VPW = 2;                      //Verical synchro Sync Pulse Width
(lines)
        parameter VFP = 10;                 //Verical synchro Front Porch (lines)
/*generate the hs && vs timing*/
        always@(posedge(clk))
        begin
            /*conditions of reseting Hcnter && Vcnter*/
            if( Hcnt == PLD-1 ) //have reached the edge of one line
            begin
                Hcnt <= 0; //reset the horizontal counter
                if( Vcnt == LFD-1 ) //only when horizontal pointer reach the edge
can the vertical counter ++
                    Vcnt <=0;
                else
                    Vcnt <= Vcnt + 1;
            end
            else
                Hcnt <= Hcnt + 1;

            /*generate hs timing*/
            if( Hcnt == PAL - 1 + HFP)
                hs <= 1'b0;
            else if( Hcnt == PAL - 1 + HFP + HPW )
                hs <= 1'b1;

            /*generate vs timing*/
            if( Vcnt == LAF - 1 + VFP )
                vs <= 1'b0;
```

```
            else if( Vcnt == LAF - 1 + VFP + VPW )
                vs <= 1'b1;
        end

endmodule
```

3、数码管分数显示子系统

（1） 功能框图：



（2） 接口定义

```
module seven_seg(
    input clk,                          //输入变频时钟
    input rst,                          //输入复位信号
    input lose,                         //游戏结束信号
    input goal,                         //游戏得分信号
    input hard,                         //困难模式选择
    input start,                        //游戏开始
    input [3:0] bar_move_speed,         //游戏速度
    output reg [7:0] select,            //数码管位选信号
    output reg [6:0] seg                //数码管段选信号
    );

module seg_decoder(
    input clk,                          //输入变频时钟
    input [3:0] num,                    //输入待显示的数字
    output reg [6:0] code               //数码管段选编码
    );
```

（3） 模块代码

- 数码管分数显示子系统

```
`include "Definition.h"
module seven_seg(
        input clk,
       input rst,
       input lose,
       input goal,
       input hard,
       input start,
       input [3:0] bar_move_speed,
      output reg [7:0] select,
      output reg [6:0] seg
      );

reg [3:0] num0 = 4'b0;

reg [3:0] num1 = 4'b0;

reg [3:0] num2 = 4'b0;

reg [3:0] num3 = 4'b0;

reg [3:0] num4 = 4'b0;

reg [3:0] num5 = 4'b0;

reg [3:0] num6 = 4'b0;

reg [3:0] num7 = 4'b0;

reg [5:0] max = 0;
reg [5:0] score;

reg [3:0] cnt = 0;

reg [7:0] clk_cnt = 0;

reg sclk = 0;

always@(posedge clk)
begin
    if(clk_cnt == 8'd255)
    begin
```

```
            sclk <= ~sclk;
            clk_cnt <= 0;
        end
        else
            clk_cnt <= clk_cnt + 1;
end

wire [6:0] out0;
wire [6:0] out1;
wire [6:0] out2;
wire [6:0] out3;
wire [6:0] out4;
wire [6:0] out5;
wire [6:0] out6;
wire [6:0] out7;

seg_decoder seg0(
        .clk(clk),
        .num(num0),
        .code(out0)
        );

seg_decoder seg1(
        .clk(clk),
        .num(num1),
        .code(out1)
        );

seg_decoder seg2(
        .clk(clk),
        .num(num2),
        .code(out2)
        );

seg_decoder seg3(
        .clk(clk),
        .num(num3),
        .code(out3)
        );

seg_decoder seg4(
         .clk(clk),
         .num(num4),
         .code(out4)
```

```
        );

seg_decoder seg5(
        .clk(clk),
        .num(num5),
        .code(out5)
        );

seg_decoder seg6(
        .clk(clk),
        .num(num6),
        .code(out6)
        );

seg_decoder seg7(
        .clk(clk),
        .num(num7),
        .code(out7)
        );

// Display eight    seg
always@(posedge sclk)
begin
      if(!rst) //low active
      begin
            cnt <= 0;
      end
      else
      begin
            case(cnt)
            4'b0000:
            begin
                  seg <= out0;
                  select <= 8'b1111_1110;
            end
            4'b0001:
            begin
                  seg <= out1;
                  select <= 8'b1111_1101;
            end
            4'b0010:
            begin
                  seg <= out2;
                  select <= 8'b1111_1011;
```

```
            end
            4'b0011:
            begin
                seg <= out3;
                select <= 8'b1111_0111;
            end
            4'b0100:
                        begin
                            seg <= out4;
                            select <= 8'b1110_1111;
                        end
                        4'b0101:
                        begin
                            seg <= out5;
                            select <= 8'b1101_1111;
                        end
                        4'b0110:
                        begin
                            seg <= out6;
                            select <= 8'b1011_1111;
                        end
                        4'b111:
                        begin
                            seg <= out7;
                            select <= 8'b0111_1111;
                        end
            endcase
            cnt <= cnt + 1;
            if(cnt == 3'b111)
                cnt<=0;
        end
end


// Flush data each time you lose or gain a goal
always@(negedge rst or posedge lose or posedge goal or posedge start or negedge start)
begin
        if(!rst||lose||!start)
        begin
            if(!rst)
            begin
               num0 = 0;
                num1 = 0;
                num2 = 0;
```

```
            num3 = 0;
            max =0 ;
        end
        else
        begin
           num0 = 0;
            num1 = 0;
            num2 = 0;
            num3 = 0;
            score = 0;
            num4 = max % 10;
            num5 = (max/10) % 10;
            num6 = (max/100) % 10;
        end
    end
    else if(goal)
    begin

        if(hard) score = score + 2*bar_move_speed;
        else     score = score + bar_move_speed;
     if(max <= score)    max = score;
     num0 = score % 10;
     num1 = (score/10) % 10;
     num2 = (score/100) % 10;
     num3 = (score/1000) % 10;
     num4 = max % 10;
     num5 = (max/10) % 10;
     num6 = (max/100) % 10;
     num7 = (max/1000) % 10;
     end


end

endmodule
```

- 数码管编码器

```
`include "Definition.h"
module seg_decoder(
    input clk,
    input [3:0] num,
    output reg [6:0] code
    );
```

```
always@(posedge clk)
begin
    case(num)
    4'b0000:
        code <= `ZERO;
    4'b0001:
        code <= `ONE;
    4'b0010:
        code <= `TWO;
    4'b0011:
        code <= `THREE;
    4'b0100:
        code <= `FOUR;
    4'b0101:
        code <= `FIVE;
    4'b0110:
        code <= `SIX;
    4'b0111:
        code <= `SEVEN;
    4'b1000:
        code <= `EIGHT;
    4'b1001:
        code <= `NINE;
    default:
        code <= code;
    endcase
end

endmodule
```

4、头文件代码

```
// ball speed direction
`define RIGHT 1'b1
`define LEFT    1'b0
`define UP        1'b0
`define DOWN    1'b1

// 7 seg LED definition
`define ZERO 7'b1000000
`define ONE 7'b1111001
`define TWO 7'b0100100
`define THREE 7'b0110000
`define FOUR 7'b0011001
`define FIVE 7'b0010010
```

```
`define SIX 7'b0000010
`define SEVEN 7'b1111000
`define EIGHT 7'b0000000
`define NINE 7'b0010000
```

五、测试模块建模

```
module SBgame(
                    input clk_in,
                    input rst,
                    input to_left,
                    input to_right,
                    input [3:0] bar_move_speed,
                    input start,
                    input hard,
                     input two,
                     input to_left_2,
                     input to_right_2,
                     output HSync,
                     output [3:0] OutBlue,
                     output [3:0] OutGreen,
                     output [3:0] OutRed,
                     output VSync,
                    output [7:0] seg_select,
                    output [6:0] seg_LED
        );


wire mclk;
wire lose;
wire goal;

clk_vag clk
  (
  // Clock in ports
    .clk_in(clk_in),
    // Clock out ports
    .clk_out(mclk),
    // Status and control signals
    .resetn(rst)
  );

VGA_Dispay u_VGA_Disp(
```

```
        .clk(mclk),
        .to_left(to_left),
        .to_right(to_right),
        .bar_move_speed(bar_move_speed),
        .start(start),
        .hard(hard),
        .hs(HSync),
        .Blue(OutBlue),
        .Green(OutGreen),
        .Red(OutRed),
        .vs(VSync),
        .lose(lose),
        .goal(goal),
        .rst(rst),
        .two(two),
        .to_left_2(to_left_2),
        .to_right_2(to_right_2)
        );

seven_seg score_board(
        .clk(mclk),
        .rst(rst),
        .goal(goal),
        .hard(hard),
        .lose(lose),
        .start(start),
        .select(seg_select),
        .seg(seg_LED),
        .bar_move_speed(bar_move_speed)
        );


Endmodule
```

六、实验结果

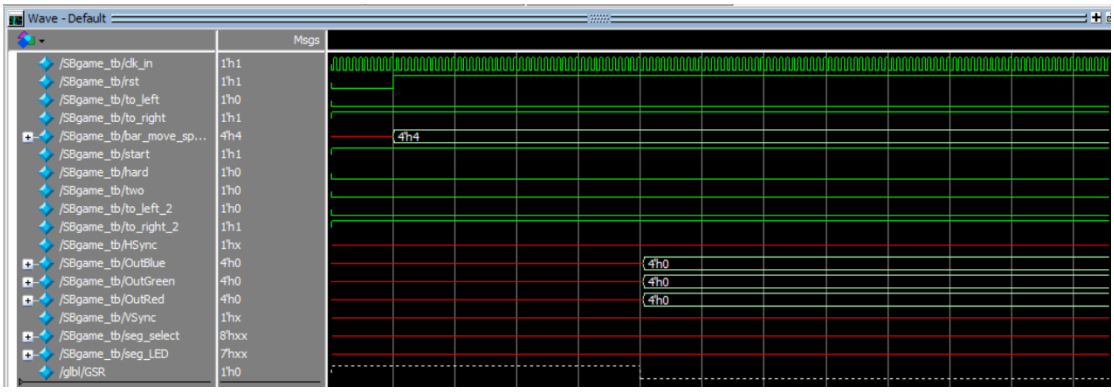
（一）、测试部分

- **说明：**

（1） 由于其他作业已测试过分频模块、数码管编码器模块，因此不再进行 modelsim 仿真测试。

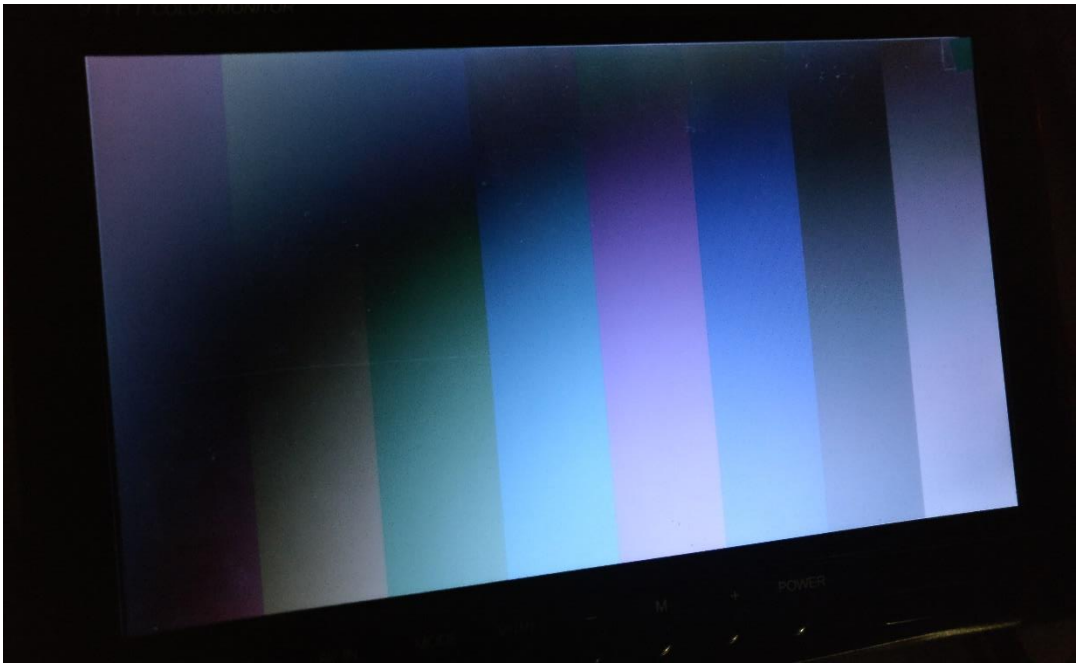
（2） 由于测试 VGA 显示屏无法通过 modelsim 测试其能否正常显示，因此显示屏模块测试由直接下板观察完成。

1、总控制模块 modelsim 仿真

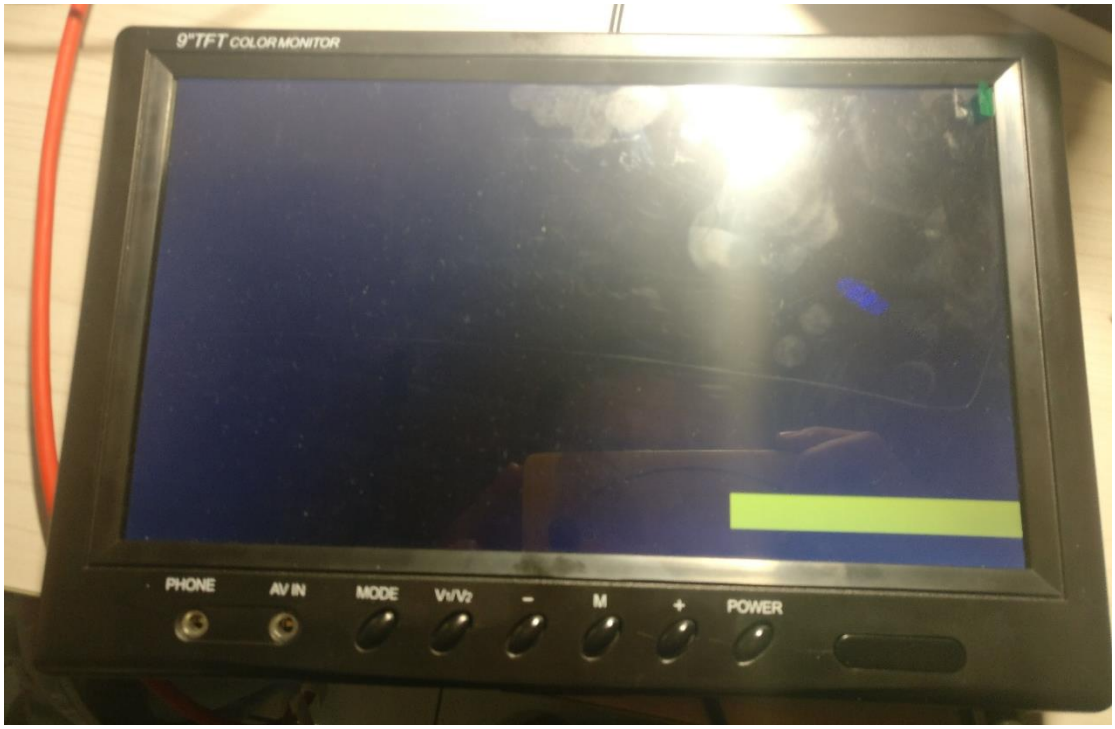


2、VGA 测试

测试 1：测试显示屏驱动模块能否正常显示指定颜色的彩条。

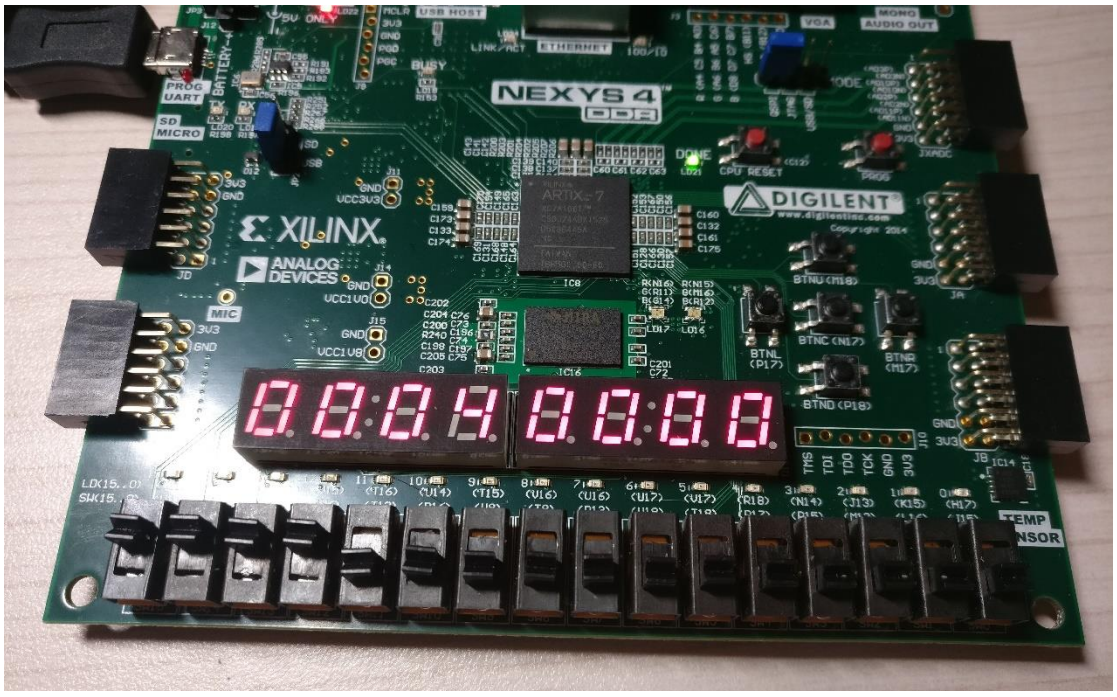


测试 2：小球、挡板的位置控制、速度控制模块是否达到预期

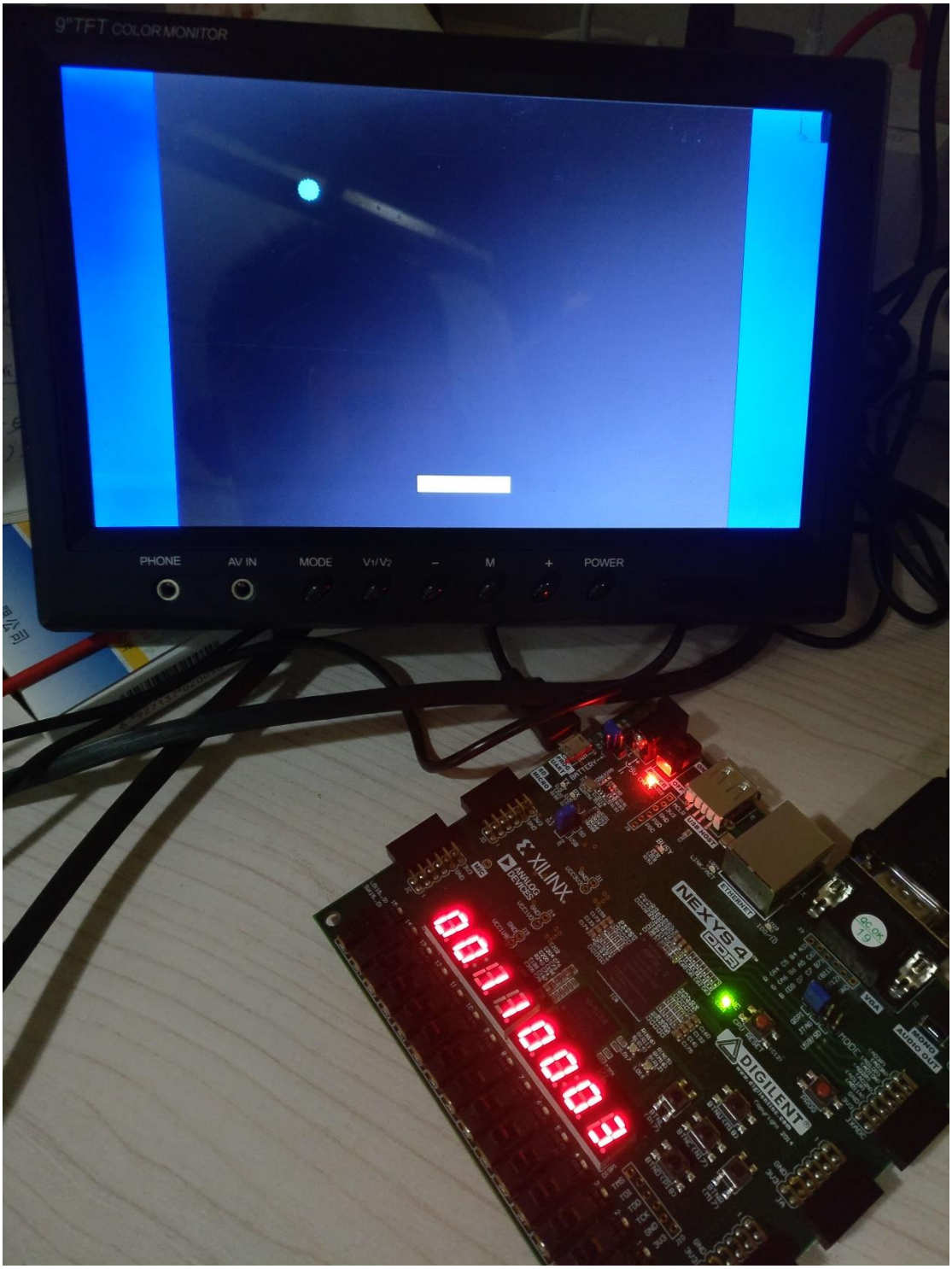


（二）、实验结果部分

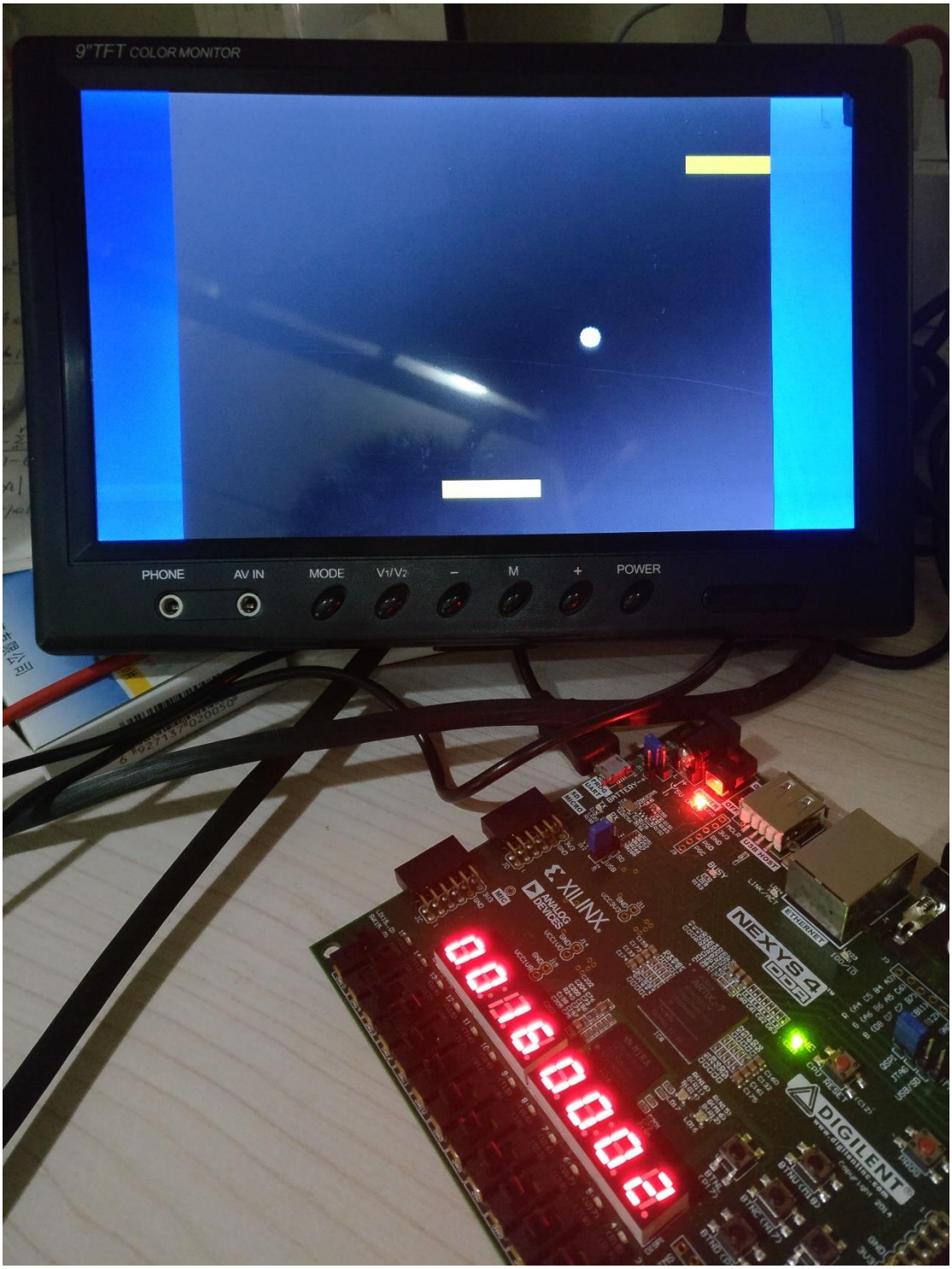
2、 数码管计分器：左侧最高分数 右侧当前分数



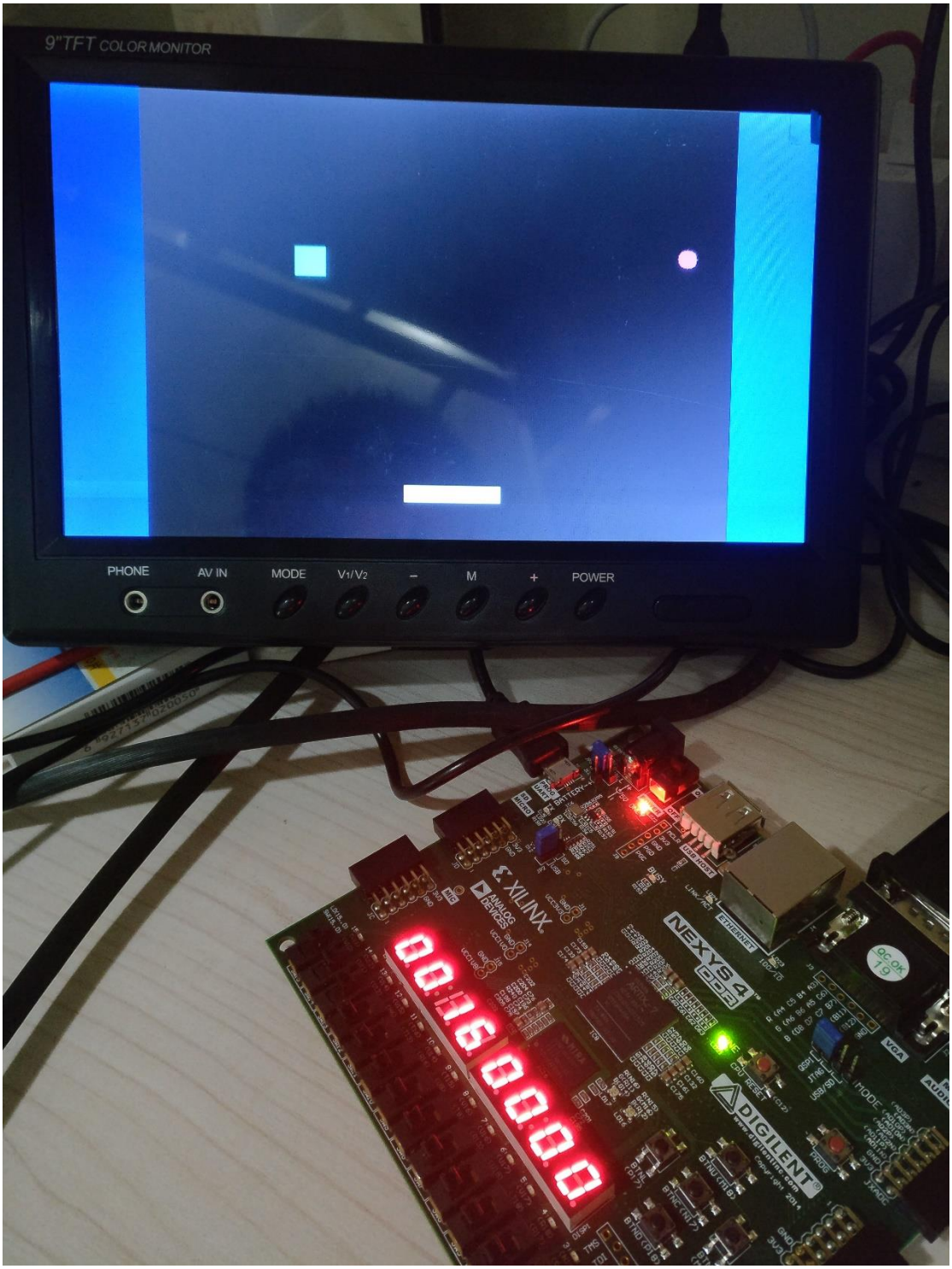
3、单人简单模式


9"TFT COLOR MONITOR
PHONE
AV IN
MODE
V+/V-
M
POWER
XILINX
NEXYS 4
DIGILENT

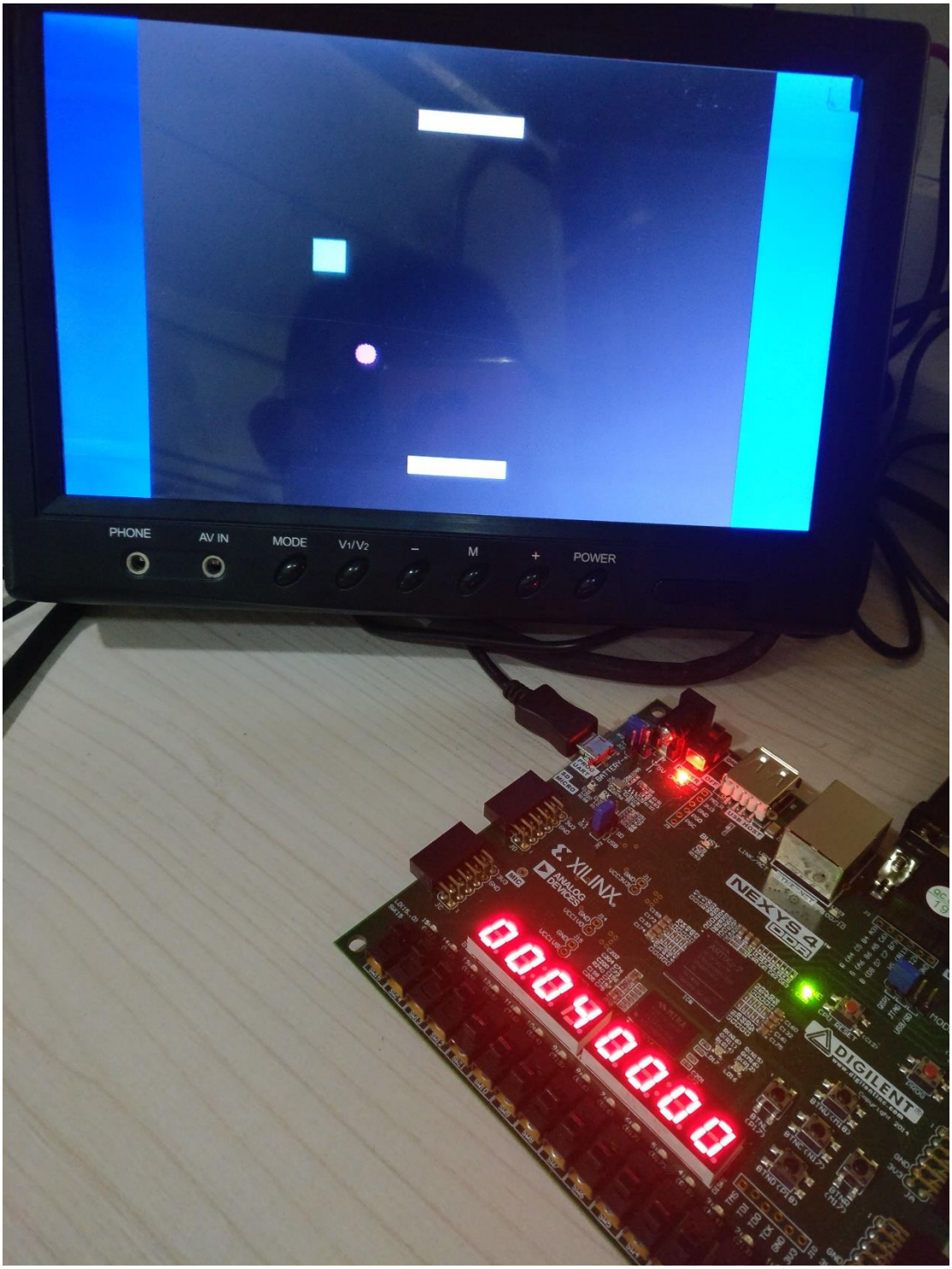
3、双人简单模式


9"TFT COLOR MONITOR
PHONE
AV IN
MODE
V1/V2
–
M
+
POWER
XILINX
ANALOG DEVICES
NEXYS 4
DIGILENT

4、单人困难模式


9"TFT COLOR MONITOR
PHONE
AV IN
MODE
V1/V2
–
M
+
POWER
XILINX
NEXYS 4
DIGILENT

5、双人困难模式



七、结论

能实现四种游戏模式，实现游戏状态判断并完成分数统计。

八、心得体会

1、 心得体会

（1） 第一次自己设计并完成一个数字系统的设计，对自上而下的系统设计方法有了更清晰的认识和体会。

（2） 对 Verilog 语法更加熟悉，能更熟练的完成不同模块间的参数传递、模块调用，对并序执行和时许逻辑有了更深刻的理解。

（3） 切身体会到阻塞赋值和非阻塞赋值之间的差别。在显示当前游戏分数时，一开始由于变量采用非阻塞复制，第一次接住小球以后，分数并没有发生改变，到第二次接住小球时，分数仅为接住一次球的分数，将其改为阻塞赋值，此 bug 解决。

（4） 初步了解了 VGA 接口技术，明白了 VGA 显示的原理，理解了场时序、行时序，能通过控制 VGA 三原色的输出显示所需颜色。

（5） 增强了对 vivado,logisim,modelsim,visio 等软件使用的熟练程度。

（6） 本次大作业提高了自己的资料搜集能力和自学能力。