同济大学计算机系

数字逻辑课程综合实验报告

学　　号 **1953484**

姓　　名 孙卓阳

专　　业 信息安全

授课老师 郭玉臣

目录

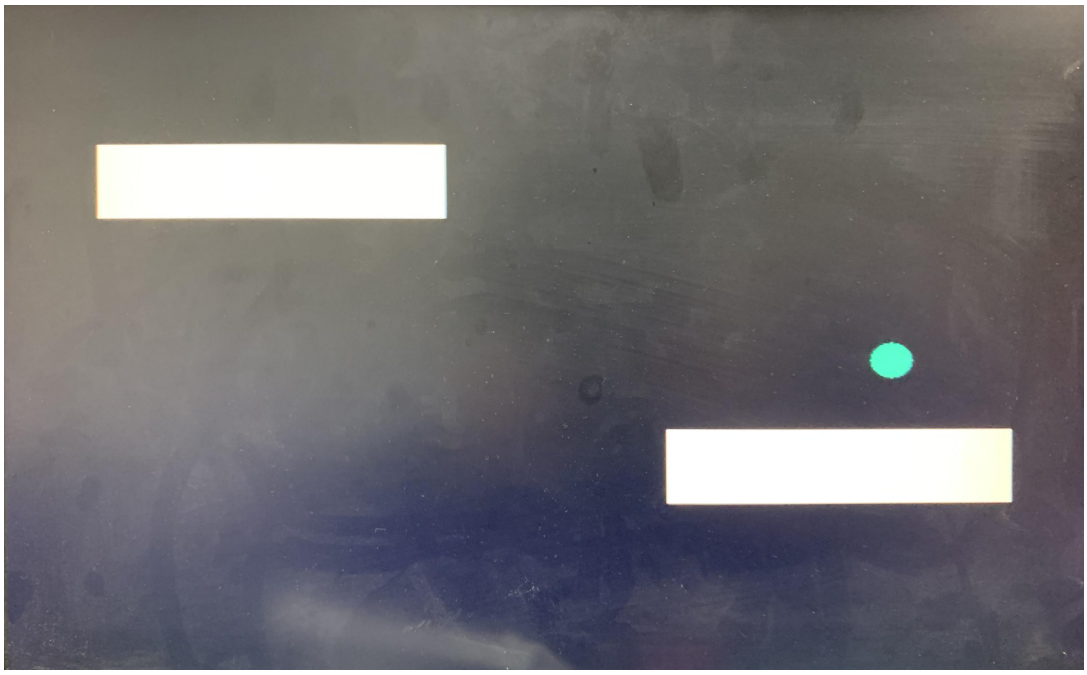
一、实验内容

1.1 项目内容概括

基于 FPGA 以及 LM386 声音传感器的一款弹球小游戏

1.2 界面样式

如图所示，界面包括移动的挡板(规律地左右移动)，小球(起始位置为屏幕左上角，起始运动方向为右下)



1.3 操作说明

设置 R15 管脚为 start，当 start 为 0 游戏未开始，start 为 1 游戏开始。M13、L16、J15 为小球速度控制管脚，设置了四档速度，从 V10 到 R17 共 12 个管脚为小球颜色控制管脚，调节可以使小球表现为不同的颜色(2^12 种颜色)。发出声音或者点击声音传感器可以使小球在下边界成功反弹。游戏结束后重置 start 即可再次开始游戏。

1.4 规则说明

有上下左右四个边界和两个左右移动的挡板，小球碰到挡板或左、

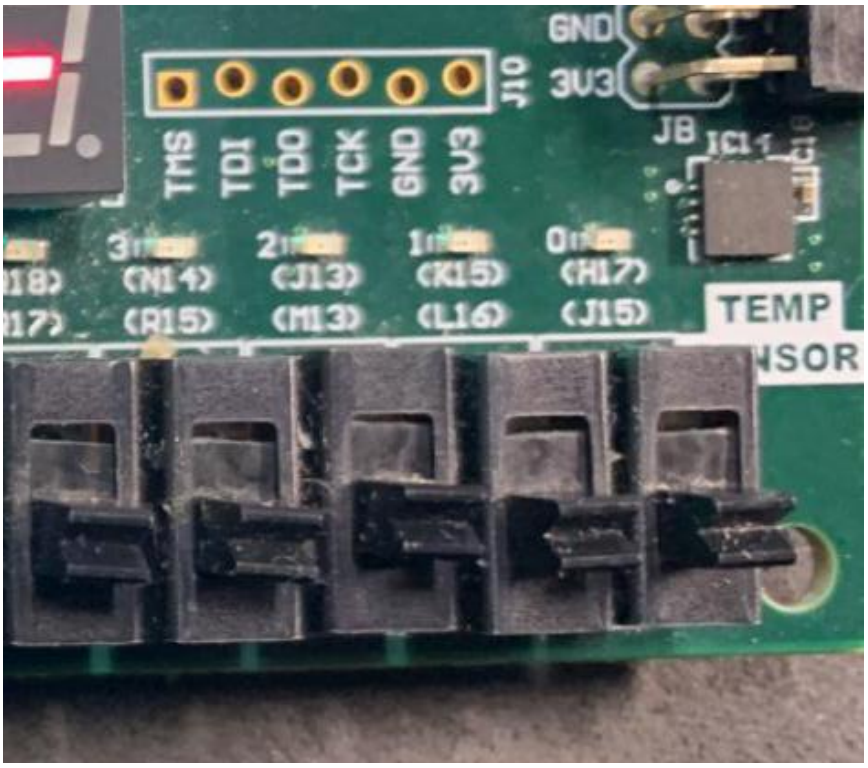
右、上边界会自动反弹，当小球到达下边界的时候需要发出声音给声音传感器一个信号使下边界的反弹有效，如果未及时给出声音信号时小球进入下边界以下则判定游戏结束。

Start 有效后开始计时，左四位数码管显示一个倒计时（源码设置的是 60s），右四位数码管显示的是分数（成功通过一次下边界加一分），同时设置有速度控制管角，有四种小球移动速度选项，需要在倒计时结束前得到尽量多的分数。

开关控制按键图文说明如下：

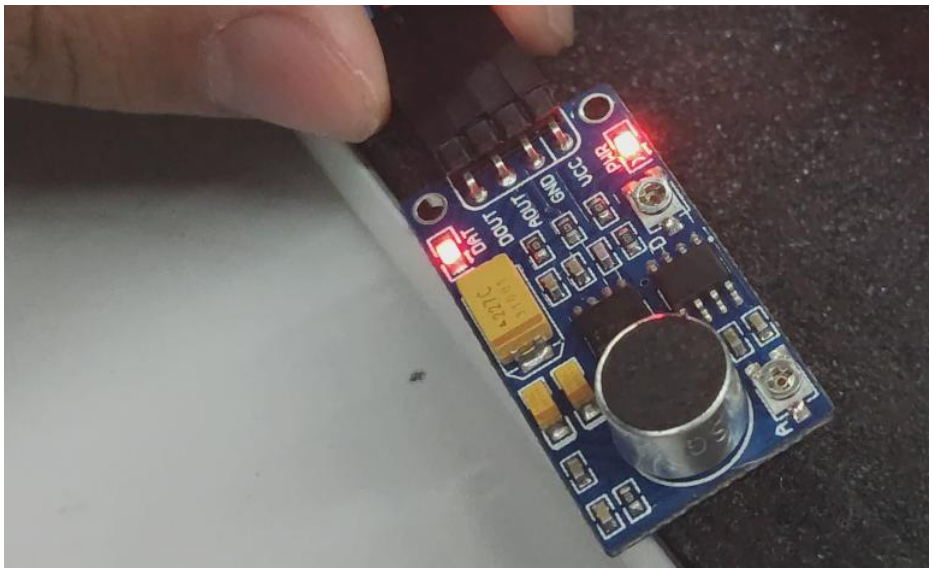
1.4.1 管脚

开关编号	用途	详细说明
R15	开始开关	置 0 游戏未开始 置 1 游戏开始
M13，L16，J15	控制小球速度	001 速度为 2 011 速度为 4 111 速度为 6 其他状态速度为 1
V10，U11，U12，H6， T13,R16， U8,T8,R13,U18,T18,R17	控制小球颜色	V10-H6 控制蓝色系 T13-T8 控制绿色系 R13-R17 控制红色系



1.4.2 声音传感器

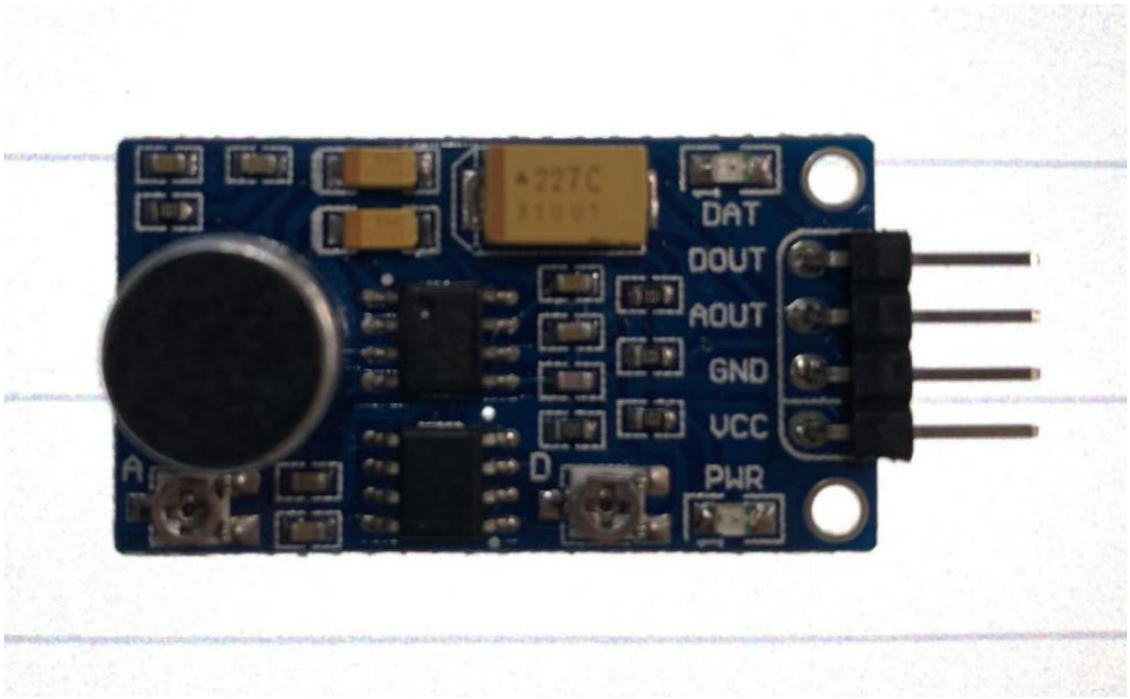
操作	用途
点击咪头	使声音传感器的信号有效
发出声音	使声音传感器的信号有效



1.5 外设介绍

1.5.1LM386 声音传感器

1LM386 是一种音频集成功率放大器，具有自身功耗低、更新内链增益可调整、电源电压范围大、外接元件少和总谐波失真小等优点。主要应用于低电压消费类产品。

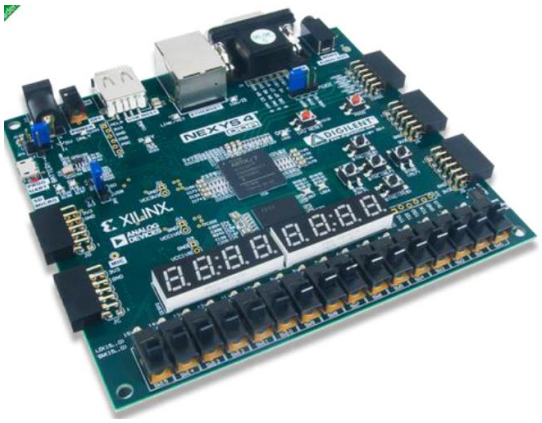


1.5.2 VGA——VGA(Video Graphics Array)是 IBM 在 1987 年随 PS/2 机一起推出的一种

视频传输标准，具有分辨率高、显示速率快、颜色丰富等优点；

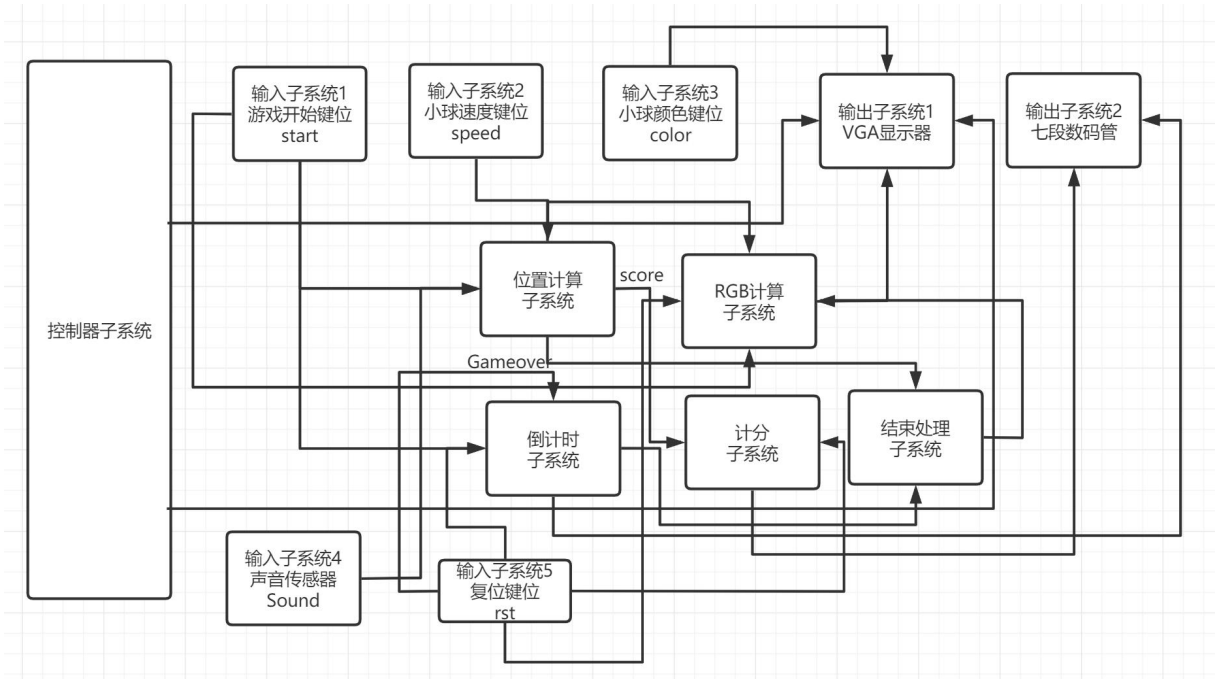
1.5.3 Nexys 4 DDR Artix-7 FPGA Trainer Board:

由 Xilinx 公司开发出的一款可编程门阵列（FPGA）开发板。



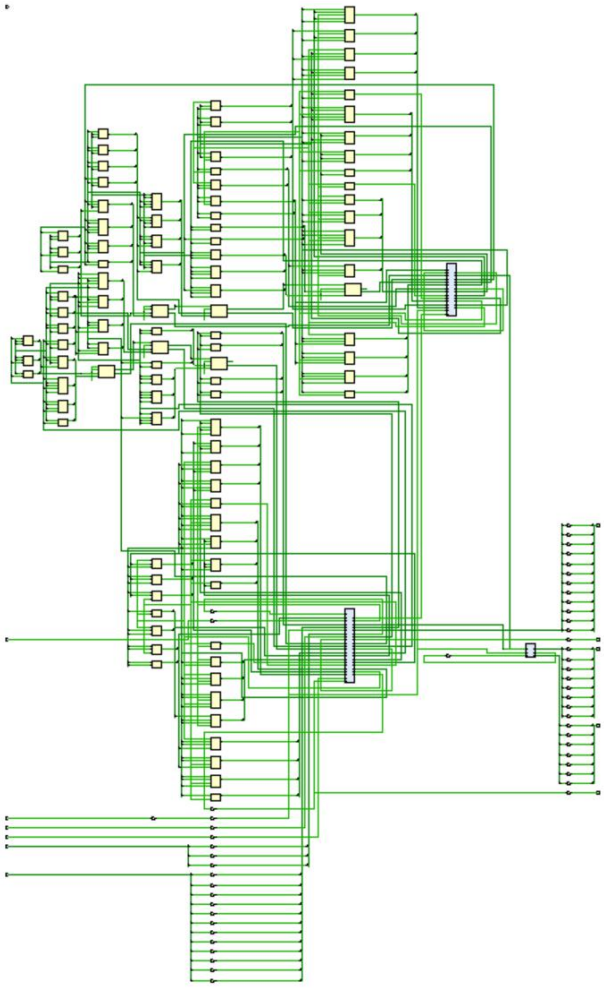
一、 弹球小游戏数字系统总框图

2.1 总框图:



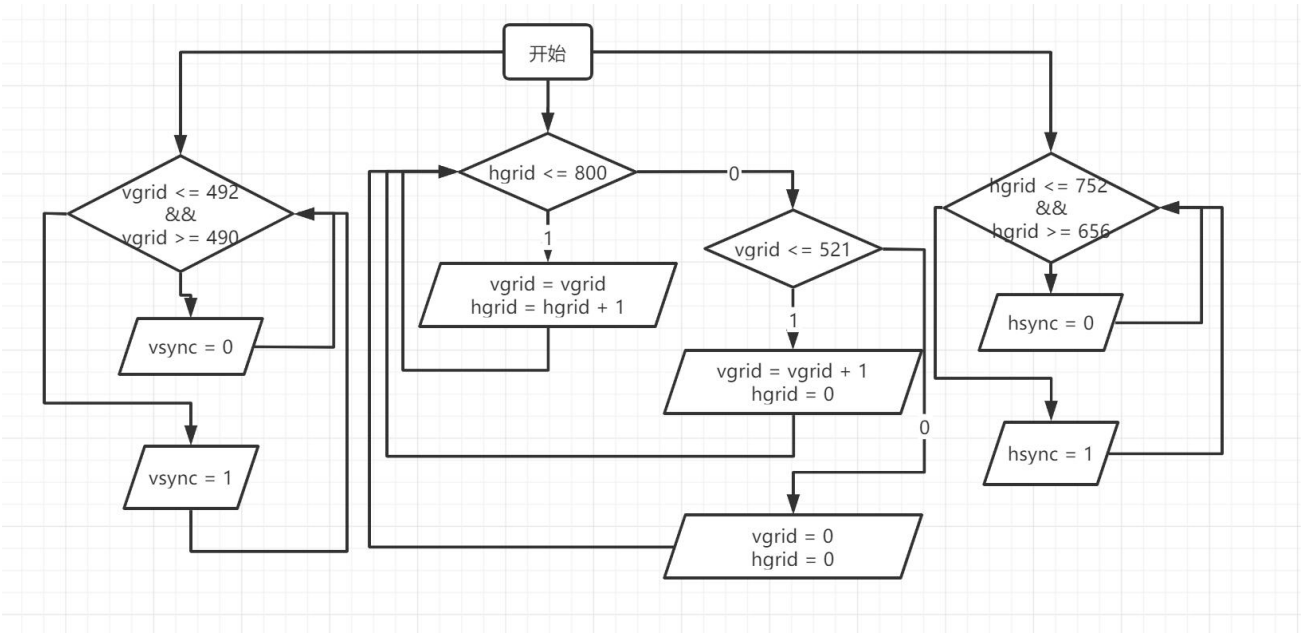
2.2 RTL 图片

如下:



二、系统控制器设计

3.1 hsync 和 vsync 信号设计



根据 basic VGA controller 的电路图，在水平方向扫描一次后，使得垂

直方向开始扫描。因为水平方向是时钟计数的，垂直方向根据水平方向的脉冲计数的。设置行选 hsync，列选 vsync 信号有效。 由于有建立的 Tpw 时间，所以要把 Tpw(脉冲宽度）时间段内的坐标视为无效

3.2 小球碰撞运动状态逻辑

PS	NS	转换条件
开始(1000)	生成小球运动方向(0000)	/
生成小球运动方向(0000)	生成小球位置(0001)	/
生成小球位置(0001)	碰撞检测(0010)	/
碰撞检测(0010)	生成小球运动方向(0001)	碰撞到墙面或者挡板
碰撞检测(0010)	游戏结束(0011)	碰撞到底边且无声音信号
碰撞检测(0010)	加分(0100)	碰撞到底边且有声音信号
加分(0100)	生成小球运动方向(0001)	/

记声音信号为 X，有信号为 1，无信号为 0

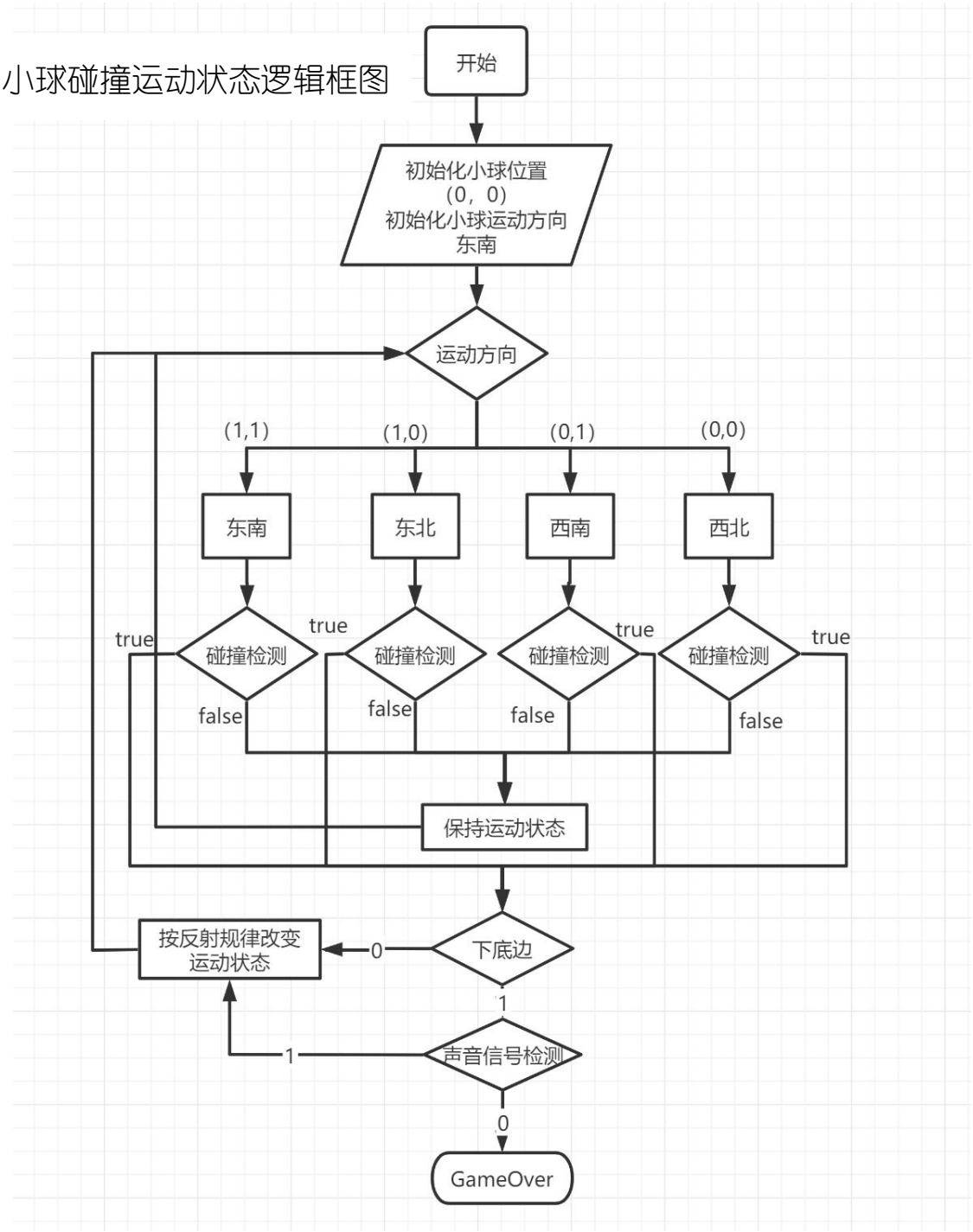
$$D^{n+1} = \overline{D^n} C^n B^n A^n$$

$$C^{n+1} = X \bar{B}^n$$

$$B^{n+1} = \bar{B}^n A^n + B^n \bar{X}$$

$$A^{n+1} = \bar{C}\bar{B} + (B+C)$$

小球碰撞运动状态逻辑框图



3.3 游戏逻辑

PS	NS	转换条件
开始(1000)	初始化(0000)	\
初始化(0000)	按规则正常运动(0001)	\
按规则正常运动(0001)	检测倒计时(0010)	
检测倒计时(0010)	GameOver(0011)	倒计时结束
检测倒计时(0010)	检测底边(0100)	倒计时未结束
检测底边(0100)	按规则正常运动(0001)	未到达下边界
检测底边(0100)	检测声音信号(0101)	到达下边界
检测声音信号(0101)	GameOver(0011)	无声音信号
检测声音信号(0101)	加分(0110)	有声音信号
加分(0110)	按规则正常运动(0001)	\

记倒计时状态为 X，倒计时结束为 0，未结束为 1

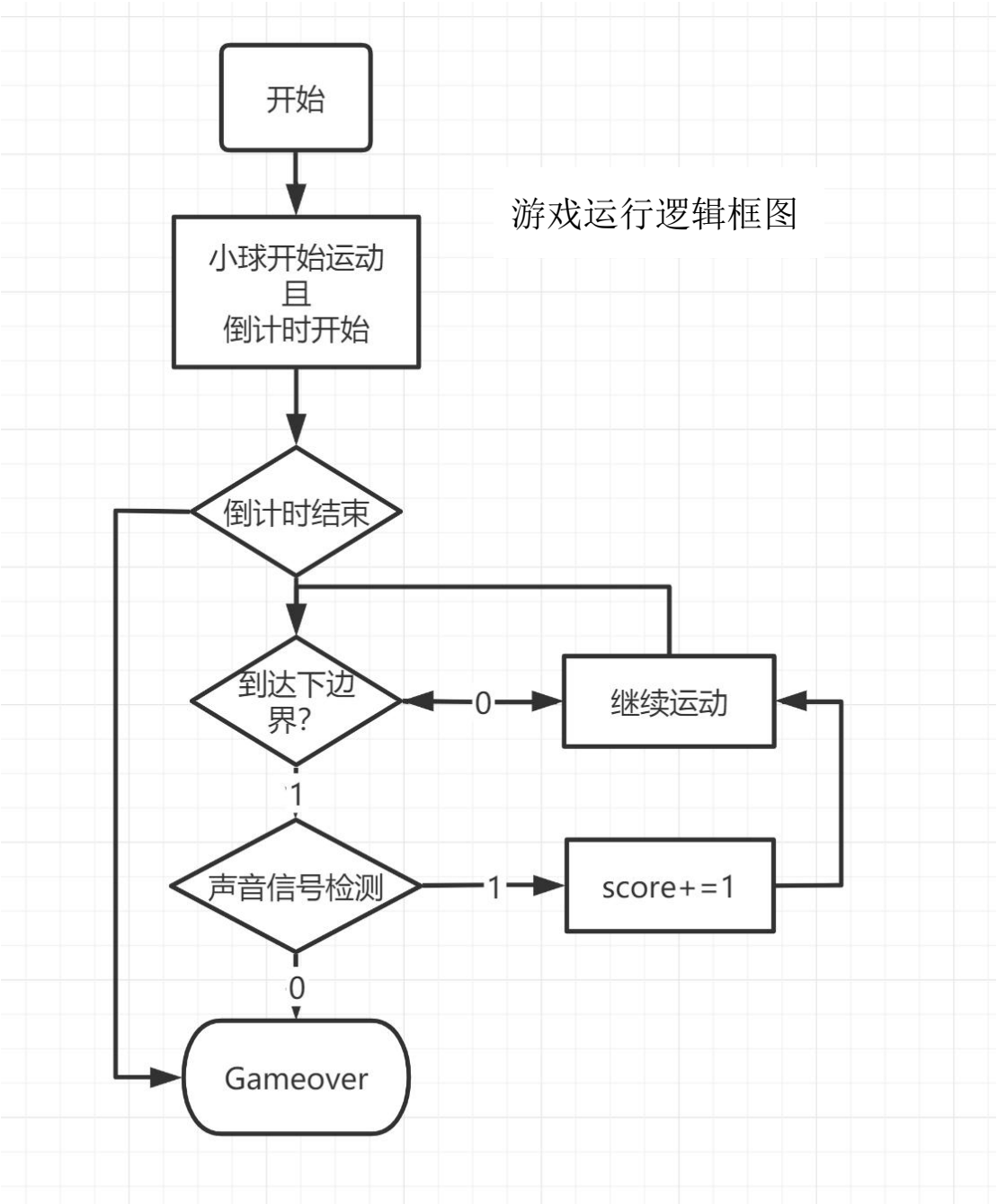
记声音信号状态为 Y，有声音信号为 1，无声音信号为 0

$$D^{n+1} = \overline{D}^{n} C^{n} B^{n} A^{n}$$

$$C^{n+1} = X\overline{C}^{n} B^{n} + \overline{C}^{n} \overline{B}^{n} A^{n}$$

$$B^{n+1} = A^{n} + X$$

$$A^{n+1} = \overline{X}\overline{C}^{n} B^{n} + \overline{Y} C^{n} \overline{B}^{n} + C^{n} B^{n} + C^{n} \overline{B}^{n} \overline{A}^{n} + \overline{D}^{n} \overline{C}^{n} \overline{B}^{n} \overline{A}^{n} + \overline{D}^{n} C^{n} B^{n} A^{n}$$



游戏运行逻辑框图

三、子系统模块建模

4.1 Ball_game 顶层模块

顶层模块，连接各个子模块，列出了所有需要与外部进行数据数据交流的数据

```
module Ball_game(
```

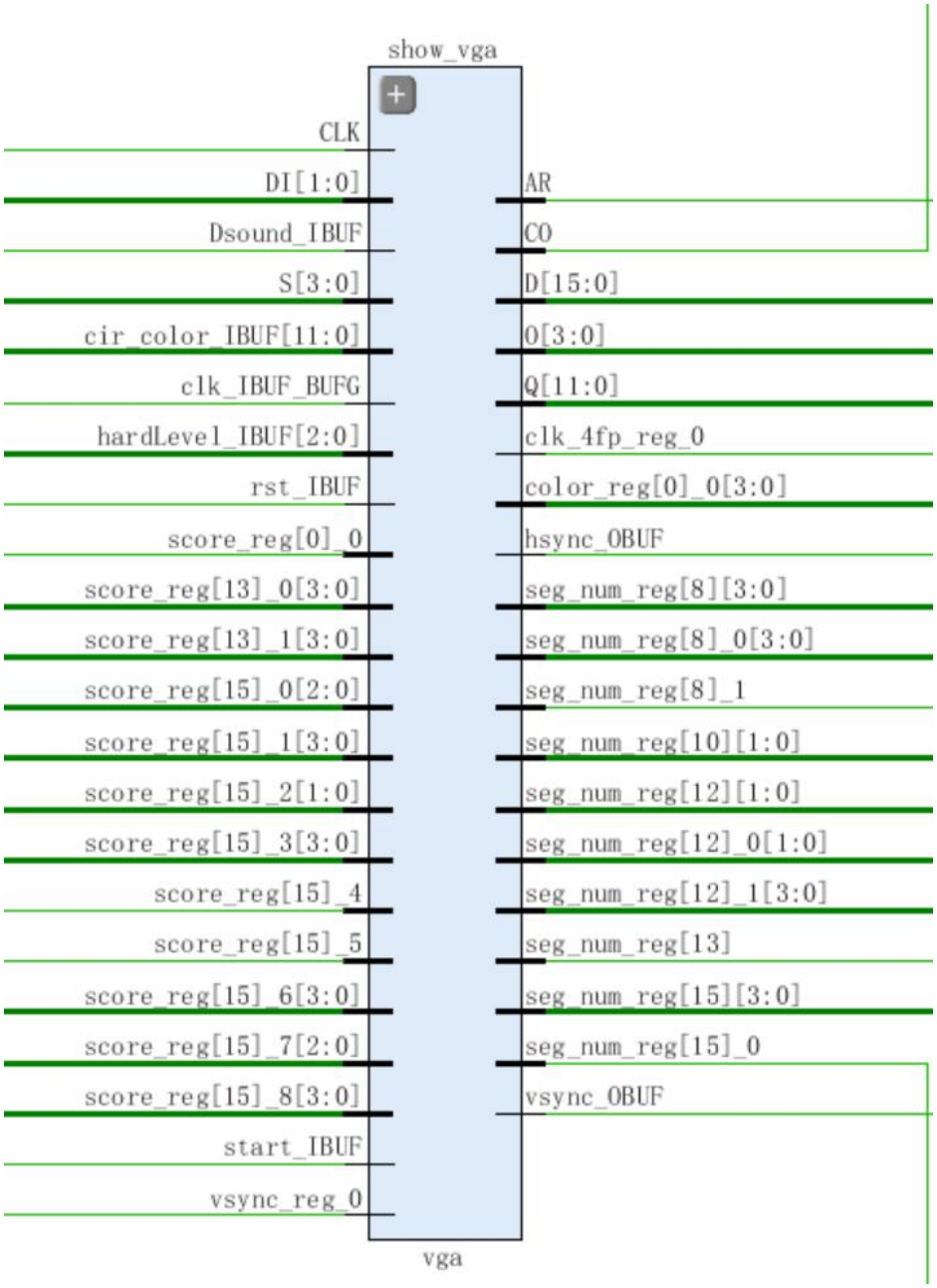
```
    input clk,                  //系统时钟
    input rst,                  //复位键
    input Dsound,               //声音传感器数字信号
    input Asound,               //声音传感器模拟信号
    input start,                //游戏开关
    input[2:0] speed,          //小球速度
    input[11:0] cir_color, //小球颜色

    output[6:0] seg_led,    //七段数码管显示数据
    output[7:0] id_led,     //七段数码管选择
    output hsync, vsync,      //行有效&列有效信号
    output[11:0] color      //VGA 颜色输出
);
wire time_over, gameover;
wire[15:0] score, Time;

vga show_vga(clk, rst, start, Dsound, speed, cir_color,
time_over, gameover, score, hsync, vsync, color);
timecnt(clk, start, rst, gameover, time_over, Time);
Display7 display_score(clk, rst, score, Time, seg_led,
id_led);
```

Endmodule

4.2 vga 子模块




```
module vga(
    input clk, rst, start, Dsound,
    input[2:0] hardLevel,
    input[11:0] cir_color,
    input time_over,
```

```
    output reg gameover,
    output reg[15:0] score,
    output reg hsync, vsync,
    output reg[11:0] color
);
//VGA 显示变量定义
reg[9:0] hgrid = 0; // 800 TS , x
reg[9:0] vgrid = 0; // 521 Ts , y
reg clk_4fp = 0;
reg[1:0] count = 0;
//游戏难度
integer speed = 2;
// 显示移动小球
parameter  WIDTH = 24,    //矩形长
HEIGHT = 24,  //矩形宽
// 显示区域的边界
DISV_TOP = 10'd480,  // display top bound
DISV_DOWN = 10'd0,     // display down bound
DISH_LEFT = 10'd0,     // display left bound
DISH_RIGHT = 10'd640;  // display right bound
//
```

```
reg[1:0] flag = 0;
reg[9:0] rec_count = 0;
reg[9:0] rec_top_1 = 10'd150;     // rec1 top bound
reg[9:0] rec_down_1 = 10'd100;    // rec1 down bound
reg[9:0] rec_left_1 = 0;          // rec1 left bound
reg[9:0] rec_right_1 = 200;       // rec1 right bound
reg[9:0] rec_top_2 = 10'd350;     // rec2 top bound
reg[9:0] rec_down_2 = 10'd300;  // rec2 down bound
reg[9:0] rec_left_2 = DISH_RIGHT - 10'd200; reg[9:0]
// rec2 left bound
rec_right_2 = DISH_RIGHT;        // rec2 right bound
reg[2:0] loc = 0;
//初始小球的位置，在显示区的左下角
wire[9:0] center_v;
wire[9:0] center_h;
reg[9:0] topbound = DISV_DOWN + HEIGHT;
reg[9:0] downbound = DISV_DOWN;
reg[9:0] leftbound = DISH_LEFT;
reg[9:0] rightbound = DISH_LEFT + WIDTH; assign center_v
= (topbound + downbound) / 2; assign center_h =
(leftbound + rightbound) / 2; //初始方向为东南方向
reg[1:0] movexy = 2'b11;
```

```
//////////////////////////////////////////////VGA 显示驱动
//分频 25M Hz
always@(posedge clk)begin
if (rst)begin
   count <= 0;
end
else begin
if (count < 1)begin
   count <= count + 1;
end
else
begin

   clk_4fp <= ~clk_4fp;
   count <= 0;
end
end
end
// 扫描整个屏幕，包括非显示区,确定什么时候
always @ (posedge clk_4fp or posedge rst) begin
if (rst) begin
   hgrid <= 0;
```

```
    vgrid <= 0;
end
else begin
//根据 basic VGA controller 的电路图，在水平方向扫描一次后，使得垂直方向开始扫描
// 因为水平方向是时钟计数的，垂直方向是根据水平方向的脉冲计数的
if (hgrid >= 800) begin
    hgrid <= 0;
    vgrid <= (vgrid >= 521 ? 0 : vgrid + 1'b1);
    end
else
    hgrid <= hgrid + 1'b1;
end
end
//设置行选，列选信号有效。 由于有建立的 Tpw 时间，所以要把 Tpw(脉冲宽度) 时间段内的坐标视为无效
always @(posedge clk_4fp or posedge rst) begin
if (rst) begin
    hsync <= 0;
    vsync <= 0;
end
```

```
else begin
if (hgrid < 752 && hgrid >= 656)// 脉冲内为 0 (800 - Tbp -Tpw)
~ (800 - Tbp)
   hsync <= 0;
else
hsync <= 1;

if (vgrid < 492 && vgrid >= 490)// 脉冲内为 0 (521 - Tbp -Tpw)
~ (521 - Tbp)
   vsync <= 0;
else
vsync <= 1;
end
end
/*
根据时间选择不同范围坐标的像素显示颜色，使其成为一个移动的矩形。
由于是 60/s，vsync 的 Ts 恰好是移动 1px 所花的时间，所以用 vsync 信号的上升沿判断
*/
//游戏难度选择
always@(hardLevel)begin
```

```
case(hardLevel)
    3'b001:speed = 2;
    3'b011:speed = 4;
    3'b111:speed = 6;
    default:speed = 1;
endcase
end
//游戏结束条件检测
//确立每一个像素时钟里球的坐标范围
always @ (posedge vsync or posedge rst or posedge start or
posedge gameover) begin
if (rst || !start) begin
    topbound = DISV_DOWN + HEIGHT;
    downbound = DISV_DOWN;
    leftbound = DISH_LEFT;
    rightbound = DISH_LEFT + WIDTH;
    movexy = 2'b11;
    score = 0;
    gameover = 0;
end
else if (gameover || time_over) begin
    topbound = DISV_DOWN + HEIGHT;
```

```
    downbound = DISV_DOWN;
    leftbound = DISH_LEFT;
    rightbound = DISH_LEFT + WIDTH;
    movexy = 2'b11;
    score = 0;
end
//碰到边界，改变方向
else begin
case(movexy[1:0])
    2'b11: begin // 东南
    if (topbound <= DISV_TOP && topbound >= DISV_TOP - 20 &&
rightbound < DISH_RIGHT)begin
        if (Dsound > 0)begin
            movexy = 2'b10;
            score = score + 1;
end
        else if (!(Dsound > 0) && topbound <= DISV_TOP &&
topbound >= DISV_TOP - 8)
            gameover = 1;
end
        else if ((topbound <= rec_down_1 && topbound >=
rec_down_1 - 8 && rightbound > rec_left_1 && rightbound <
```

```
rec_right_1) || (topbound <= rec_down_2 && topbound >=
rec_down_2 - 8 && rightbound < rec_right_2 && rightbound >
rec_left_2))
        movexy = 2'b10;
    else if ((topbound < DISV_TOP && rightbound <= DISH_RIGHT
&& rightbound >= DISH_RIGHT - 8) || (topbound < rec_top_1 &&
topbound > rec_down_1 && rightbound <= rec_left_1 &&
rightbound >= rec_left_1 - 8) || (topbound < rec_top_2 &&
topbound > rec_down_2 && rightbound <= rec_right_2 &&
rightbound >= rec_left_2 - 8))
        movexy = 2'b01;
    else if ((topbound <= rec_down_1 && topbound >=
rec_down_1 - 8 && rightbound <= rec_left_1 && rightbound >=
rec_left_1 - 8) || (topbound <= rec_down_2 && topbound >=
rec_down_2 - 8 && rightbound <= rec_left_2 && rightbound >=
rec_left_2 - 8))
        movexy = 2'b00;
    else if (topbound <= DISV_TOP && topbound >= DISV_TOP -
10 && rightbound <= DISH_RIGHT && rightbound >= DISH_RIGHT -
20)begin
        if (Dsound > 0)begin
            movexy = 2'b00;
```

```
            score = score + 1;
end
        else if (!(Dsound > 0) && topbound >= DISV_TOP &&
topbound >= DISV_TOP - 8)
            gameover = 1;
end
end
2'b10: begin // 东北
if ((downbound >= DISV_DOWN && downbound <= DISV_DOWN + 8 &&
rightbound < DISH_RIGHT) || (downbound >= rec_top_1 &&
downbound <= rec_top_1 + 8 && rightbound < rec_right_1 &&
rightbound > rec_left_1) || (downbound >= rec_top_2 &&
downbound <= rec_top_2 + 8 && rightbound < rec_right_2 &&
rightbound > rec_left_2))
   movexy = 2'b11;
else if ((downbound > DISV_DOWN && rightbound <= DISH_RIGHT &&
rightbound >= DISH_RIGHT - 8) || (downbound <= rec_top_1 &&
downbound >= rec_down_1 && rightbound <= rec_left_1 &&
rightbound >= rec_left_1 - 8) || (downbound <= rec_top_2 &&
downbound >= rec_down_2 && rightbound <= rec_left_2 &&
rightbound >= rec_left_2 - 8))
   movexy = 2'b00;
```

```
else if ((downbound >= DISV_DOWN && downbound <= DISV_DOWN +
8 && rightbound <= DISH_RIGHT && rightbound >= DISH_RIGHT + 8)
|| (downbound >= rec_top_1 && downbound <= rec_top_1 + 8 &&
rightbound <= rec_left_1 && rightbound >= rec_left_1 - 8) ||
(downbound >= rec_top_2 && downbound <= rec_top_2 + 8 &&
rightbound <= rec_left_2 && rightbound >= rec_left_2 - 8))
   movexy = 2'b01;
end
2'b00: begin // 西北
if ((downbound >= DISV_DOWN && downbound <= DISV_DOWN + 8 &&
leftbound > DISH_LEFT) || (downbound >= rec_top_1 && downbound
<= rec_top_1 + 8 && rightbound < rec_right_1 && rightbound >
rec_left_1) || (downbound >= rec_top_2 && downbound <=
rec_top_2 + 8 && rightbound < rec_right_2 && rightbound >
rec_left_2))
   movexy = 2'b01;
else if ((downbound > DISV_DOWN && leftbound >= DISH_LEFT &&
leftbound <= DISH_LEFT + 8) || (downbound < rec_top_1 &&
downbound > rec_down_1 && leftbound >= rec_right_1 && leftbound
<= rec_right_1 + 8) || (downbound < rec_top_2 && downbound >
rec_down_2 && leftbound >= rec_right_2 && leftbound <=
rec_right_2 + 8))
```

```
   movexy = 2'b10;
else if ((downbound >= DISV_DOWN && downbound <= DISV_DOWN +
8 && leftbound >= DISH_LEFT && leftbound <= DISH_LEFT + 8) ||
(downbound >= rec_top_1 && downbound <= rec_top_1 + 8 &&
leftbound >= rec_right_1 - 4 && leftbound <= rec_right_1 + 8)
|| (downbound >= rec_top_2 && downbound <= rec_top_2 + 8 &&
leftbound >= rec_right_2 && leftbound <= rec_right_2 + 8))
   movexy = 2'b11;
end
2'b01: begin // 西南
if (topbound <= DISV_TOP && topbound >= DISV_TOP - 20 &&
leftbound > DISH_LEFT)begin
if (Dsound > 0)begin
   movexy = 2'b00;
   score = score + 1;
end
else if (!(Dsound > 0) && topbound >= DISV_TOP && topbound >=
DISV_TOP - 8)
   gameover = 1;
end
else if ((topbound <= rec_down_1 && topbound >= rec_down_1 -
8 && leftbound > rec_left_1 && leftbound < rec_right_1) ||
```

```
(topbound <= rec_down_2 && topbound >= rec_down_2 - 8 &&
leftbound > rec_left_2 && leftbound < rec_right_2))begin
   movexy = 2'b00;
   end
else if ((topbound < DISV_TOP && leftbound >= DISH_LEFT &&
leftbound <= DISH_LEFT + 8) || (topbound < rec_top_1 &&
topbound > rec_down_1 && leftbound >= rec_right_1 && leftbound
<= rec_right_1 + 8) || (topbound < rec_top_2 && topbound >
rec_down_2 && leftbound >= rec_right_2 && leftbound <=
rec_right_2 + 8))
   movexy = 2'b11;
else if ((topbound <= rec_down_1 && topbound >= rec_down_1 -
8 && leftbound >= rec_right_1 && leftbound <= rec_right_1 + 8)
|| (topbound <= rec_down_2 && topbound >= rec_down_2 - 8 &&
leftbound >= rec_right_2 && leftbound <= rec_right_2 + 8))
   movexy = 2'b10;
else if (topbound <= DISV_TOP && topbound >= DISV_TOP - 20 &&
leftbound >= DISH_LEFT && leftbound <= DISH_LEFT + 20)begin
if (Dsound > 0)begin
   movexy = 2'b10;
   score = score + 1;
end
```

```
else if (!(Dsound > 0) && topbound >= DISV_TOP && topbound >=
DISV_TOP - 8)
   gameover = 1;
end
end
default: movexy = 2'b11;
   endcase
   //操纵方块移动
   topbound <= topbound + (movexy[0] ? speed : -speed);
downbound <= downbound + (movexy[0] ? speed : -speed);
leftbound <= leftbound + (movexy[1] ? speed : -speed);
rightbound <= rightbound + (movexy[1] ? speed : -speed);
end
end
//生成边框的位置
always @ (posedge vsync or posedge rst or posedge start) begin
if (rst || !start) begin
rec_count <= 0;
rec_left_1 <= 0;
rec_right_1 <= 0;
rec_left_2 <= DISH_RIGHT;
rec_right_2 <= DISH_RIGHT + 10'd200;
```

```
end
else if (rec_count >= DISH_RIGHT + 10'd200)begin
   rec_count <= 0;
   rec_left_1 <= 0;
   rec_right_1 <= 0;
   rec_left_2 <= DISH_RIGHT;
   rec_right_2 <= DISH_RIGHT + 200;
   end
else begin
rec_count <= rec_count + 1'b1;
rec_right_1 <= rec_count;
if (rec_count < DISH_RIGHT)
   rec_left_2 <= DISH_RIGHT - rec_count;
else if (rec_count >= DISH_RIGHT)
rec_left_2 <= 0;
if (rec_count >= 10'd200)begin
   rec_left_1 <= rec_count - 10'd200;
   rec_right_2 <= DISH_RIGHT + 10'd200 - rec_count;
   end
   end
   end
```

```
    // 确定扫描到哪一个像素该显示什么颜色
    always @(posedge clk_4fp or posedge rst) begin
    if (rst)begin
       color <= 12'b0000_0000_0000;
       end
    else if (!start)begin  //游戏未开始时显示GO!
       if ((hgrid >= 124 && hgrid <= 144) && (vgrid >= 150 &&
vgrid <= 300))
          color <= 12'b1111_1111_1111;
       else if ((hgrid >= 124 && hgrid <= 244) && (vgrid >= 150
&& vgrid <= 170))
          color <= 12'b1111_1111_1111;
       else if ((hgrid >= 124 && hgrid <= 244) && (vgrid >= 280
&& vgrid <= 300))
          color <= 12'b1111_1111_1111;
       else if ((hgrid >= 184 && hgrid <= 244) && (vgrid >= 225
&& vgrid <= 245))
          color <= 12'b1111_1111_1111;
       else if ((hgrid >= 224 && hgrid <= 244) && (vgrid >= 225
&& vgrid <= 300))
          color <= 12'b1111_1111_1111;
       else if ((hgrid >= 304 && hgrid <= 424) && (vgrid >= 150
```

```
&& vgrid <= 170))
            color <= 12'b1111_1111_1111;
        else if ((hgrid >= 304 && hgrid <= 424) && (vgrid >= 280
&& vgrid <= 300))
            color <= 12'b1111_1111_1111;
        else if ((hgrid >= 304 && hgrid <= 324) && (vgrid >= 150
&& vgrid <= 300))
            color <= 12'b1111_1111_1111;
        else if ((hgrid >= 404 && hgrid <= 424) && (vgrid >= 150
&& vgrid <= 300))
            color <= 12'b1111_1111_1111;
        else if ((hgrid >= 484 && hgrid <= 504) && (vgrid >= 150
&& vgrid <= 260))
            color <= 12'b1111_1111_1111;
        else if ((hgrid >= 484 && hgrid <= 504) && (vgrid >= 280
&& vgrid <= 290))
            color <= 12'b1111_1111_1111;
        else
            color <= 12'b0;
    end
    else if (gameover || time_over)begin  //游戏判定结束时显示GG!
        if ((hgrid >= 124 && hgrid <= 144) && (vgrid >= 150 &&
```

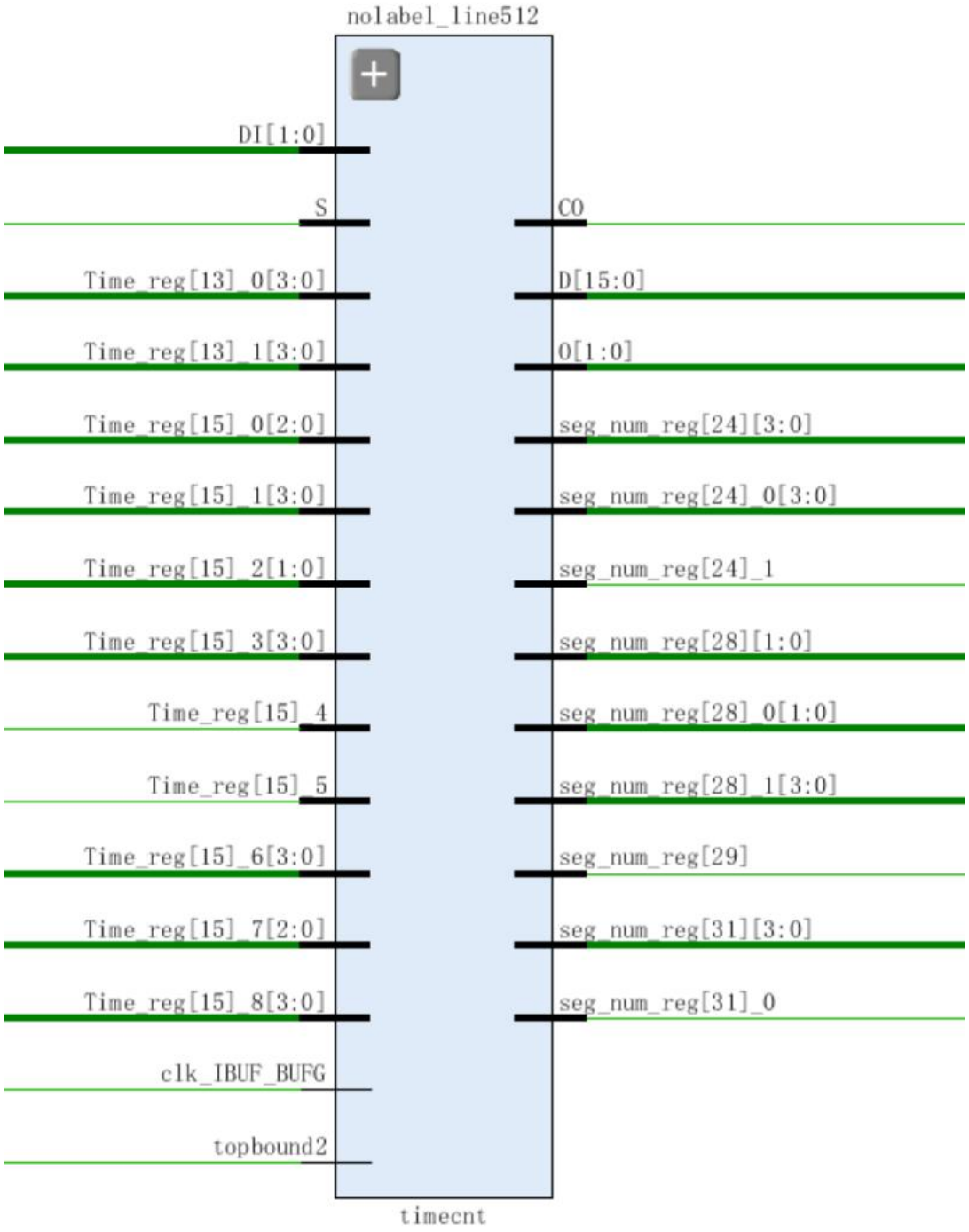
```
vgrid <= 300))
         color <= 12'b0000_1111_0000;
      else if ((hgrid >= 124 && hgrid <= 244) && (vgrid >= 150
&& vgrid <= 170))
         color <= 12'b0000_1111_0000;
      else if ((hgrid >= 124 && hgrid <= 244) && (vgrid >= 280
&& vgrid <= 300))
         color <= 12'b0000_1111_0000;
      else if ((hgrid >= 184 && hgrid <= 244) && (vgrid >= 225
&& vgrid <= 245))
         color <= 12'b0000_1111_0000;
      else if ((hgrid >= 224 && hgrid <= 244) && (vgrid >= 225
&& vgrid <= 300))
         color <= 12'b0000_1111_0000;
      else if ((hgrid >= 304 && hgrid <= 324) && (vgrid >= 150
&& vgrid <= 300))
         color <= 12'b0000_1111_0000;
      else if ((hgrid >= 304 && hgrid <= 424) && (vgrid >= 150
&& vgrid <= 170))
         color <= 12'b0000_1111_0000;
      else if ((hgrid >= 304 && hgrid <= 424) && (vgrid >= 280
&& vgrid <= 300))
```

```
            color <= 12'b0000_1111_0000;
        else if ((hgrid >= 364 && hgrid <= 424) && (vgrid >= 225
&& vgrid <= 245))
            color <= 12'b0000_1111_0000;
        else if ((hgrid >= 404 && hgrid <= 424) && (vgrid >= 225
&& vgrid <= 300))
            color <= 12'b0000_1111_0000;
        else if ((hgrid >= 484 && hgrid <= 504) && (vgrid >= 150
&& vgrid <= 260))
            color <= 12'b0000_1111_0000;
        else if ((hgrid >= 484 && hgrid <= 504) && (vgrid >= 280
&& vgrid <= 300))
            color <= 12'b0000_1111_0000;
        else
            color <= 12'b0;
            end
    else begin                    //游戏进行时小球与方块的显示
        if (hgrid >= DISH_LEFT && hgrid <= DISH_RIGHT && vgrid >=
DISV_DOWN && vgrid <= DISV_TOP) begin
            if ((hgrid - center_h) * (hgrid - center_h) + (vgrid
- center_v) * (vgrid - center_v) <= 144)
                color <= cir_color;//小球颜色
```

```
        else if ((hgrid <= rec_right_1 && hgrid >= rec_left_1
&& vgrid >= rec_down_1 && vgrid <= rec_top_1) || (hgrid >=
rec_left_2 && hgrid <= rec_right_2 && vgrid >= rec_down_2 &&
vgrid <= rec_top_2))
            color <= 12'b1111_1111_1111;//方块颜色设定为白色
        else
            color <= 12'b0001_0000_0000;
            end
      else begin
         color <= 12'b0000_0000_0000;
      end
   end
end
Endmodule
```

4.3 timecnt 子模块

倒计时模块，对系统时钟 50Mhz 分频，传出剩余时间与是否游戏结束

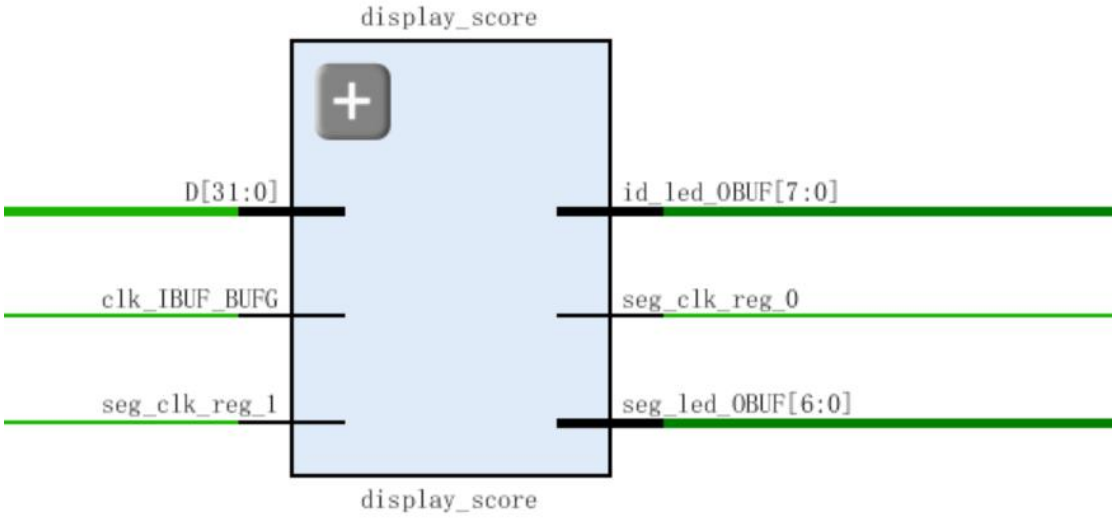



```
module timecnt(
    input clk,
    input start,
    input rst,
```

```
    output reg time_over,
    output reg[15:0] Time
);
integer counter = 0;
reg out = 0;
always@(posedge clk or posedge rst or posedge start or posedge
time_over) begin
if (rst || !start || time_over) begin
counter <= 0;
out <= 0;
end
else begin
if (counter >= 50000000) begin
counter <= 0;
out <= ~out;
end
else begin
counter <= counter + 1'b1;
out <= out;
end
end
end
```

```
always@(posedge out or posedge rst or posedge start or posedge
gameover) begin
if (rst || !start || gameover)begin
Time <= 10'd30;
time_over<= 0;
end
else begin
if (Time > 0)begin
Time <= Time - 1'b1;
gameover <= 0;
end
else begin
Time <= 0;
gameover <= 1;
end
end
end
Endmodule
```

4.4 display_score 子模块




```
module display7(
    input clk, rst,
    input[15:0] score, Time,

    output reg[6:0] seg_led,
    output reg[7:0] id_led
);
//数码管显示当前得分模块所需变量定义
localparam CLK_DIV = 4'd10;          //时钟分频系数
localparam update = 13'd5000;       //5Mhz 分频

reg[3:0]    clk_cnt;
reg         seg_clk;
reg[31:0]   seg_num;
```

```
reg[12:0]   cnt;
reg         flag;
reg[2:0]    cnt_id;
reg[3:0]    status;

//每位数码管需要显示的数据
wire[3:0] data1;
wire[3:0] data2;
wire[3:0] data3;
wire[3:0] data4;
wire[3:0] data5;
wire[3:0] data6;
wire[3:0] data7;
wire[3:0] data8;

//分配个数位对应的十进制数据
assign data1 = score % 4'd10;
assign data2 = score / 4'd10 % 4'd10;
assign data3 = score / 7'd100 % 4'd10;
assign data4 = score / 10'd1000 % 4'd10;
//倒计时
assign data5 = Time % 4'd10;
```

```
assign data6 = Time / 4'd10 % 4'd10;
assign data7 = Time / 7'd100 % 4'd10;
assign data8 = Time / 10'd1000 % 4'd10;
//////////////////////////////////////////////////////分数显
示
//数码管分频 5MHz
always@(posedge clk)begin
if (clk_cnt < CLK_DIV / 2 - 1)begin
   clk_cnt <= clk_cnt + 1'b1;
   seg_clk <= seg_clk;
end
else begin
   seg_clk <= ~seg_clk;
   clk_cnt <= 4'd0;
   end
end
//将 32 位的数码管数据转换成 8421 码
always@(posedge seg_clk)begin
   seg_num[31:28] <= data8;
   seg_num[27:24] <= data7;
   seg_num[23:20] <= data6;
   seg_num[19:16] <= data5;
```

```
    seg_num[15:12] <= data4;
    seg_num[11:8] <= data3;
    seg_num[7:4] <= data2;
    seg_num[3:0] <= data1;
end
//定义一个 1ms 的定时器
always@(posedge seg_clk)begin
if (cnt < update - 1)begin
    cnt <= cnt + 1'b1;
    flag <= 1'b0;
    end
else begin
    flag = 1'b1;
    cnt <= 13'd0;
end
end
//cnt_cs 片选计数器，根据该计数器的值确定当前要选那个数码管
always@(posedge seg_clk)begin
if (flag)begin
if (cnt_id < 4'd8)
    cnt_id <= cnt_id + 1'b1;
else
```

```
    cnt_id <= 3'd0;
end
else
    cnt_id <= cnt_id;
end
//根据片选计数器 cnt_cs 的值，轮流显示每个数码管
always@(posedge seg_clk)begin
case(cnt_id)
4'd0:begin
    id_led <= 8'b1111_1110;
    status <= seg_num[3:0];
end
4'd1:begin
    id_led <= 8'b1111_1101;
    status <= seg_num[7:4];
end
4'd2:begin
    id_led <= 8'b1111_1011;
    status <= seg_num[11:8];
end
4'd3:begin
    id_led <= 8'b1111_0111;
```

```
    status <= seg_num[15:12];
end
4'd4:begin
    id_led <= 8'b1110_1111;
    status <= seg_num[19:16];
end
4'd5:begin
    id_led <= 8'b1101_1111;
    status <= seg_num[23:20];
end
4'd6:begin
    id_led <= 8'b1011_1111;
    status <= seg_num[27:24];
end
4'd7:begin
    id_led <= 8'b0111_1111;
    status <= seg_num[31:28];
end
default:begin
    id_led <= 8'b1111_1111;
    status <= 4'd10;
    end
```

```
endcase
end
//数字的具体显示
always@(*)
begin
case(status)
    4'b0000:seg_led=7'b1000000;//0
    4'b0001:seg_led=7'b1111001;//1
    4'b0010:seg_led=7'b0100100;//2
    4'b0011:seg_led=7'b0110000;//3
    4'b0100:seg_led=7'b0011001;//4
    4'b0101:seg_led=7'b0010010;//5
    4'b0110:seg_led=7'b0000010;//6
    4'b0111:seg_led=7'b1111000;//7
    4'b1000:seg_led=7'b0000000;//8
    4'b1001:seg_led=7'b0010000;//9
    default:seg_led = 7'b1111111;//无显示，全关闭
endcase
end
```

四、 测试模块建模

VGA 子模块

由于 VGA 模块显示的特殊性，以及小球显示需要在屏幕上显示等

因素，直接下板测试更为便捷，因此该模块暂不使用 test bench 测试，直接使用最简单的四色彩条测试 VGA 是否能正常工作，经测试，预期中的 8 色彩条能够正常显示，因此 VGA 模块逻辑正确。

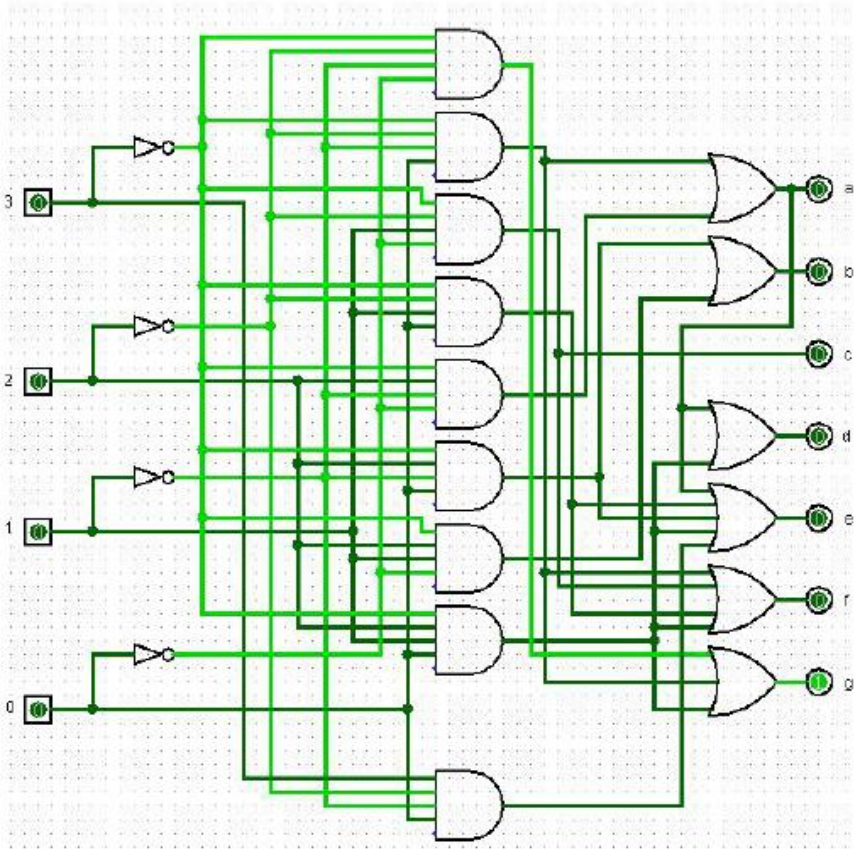
其他部分的实现也是主要基于在板上上显示出的问题来查找问题，进行进一步的调试，根据显示情况的不同进行分析，再更改相应的代码即可。

所以并未编写 tb 模块

五、 实验结果

6.1.七段数码管

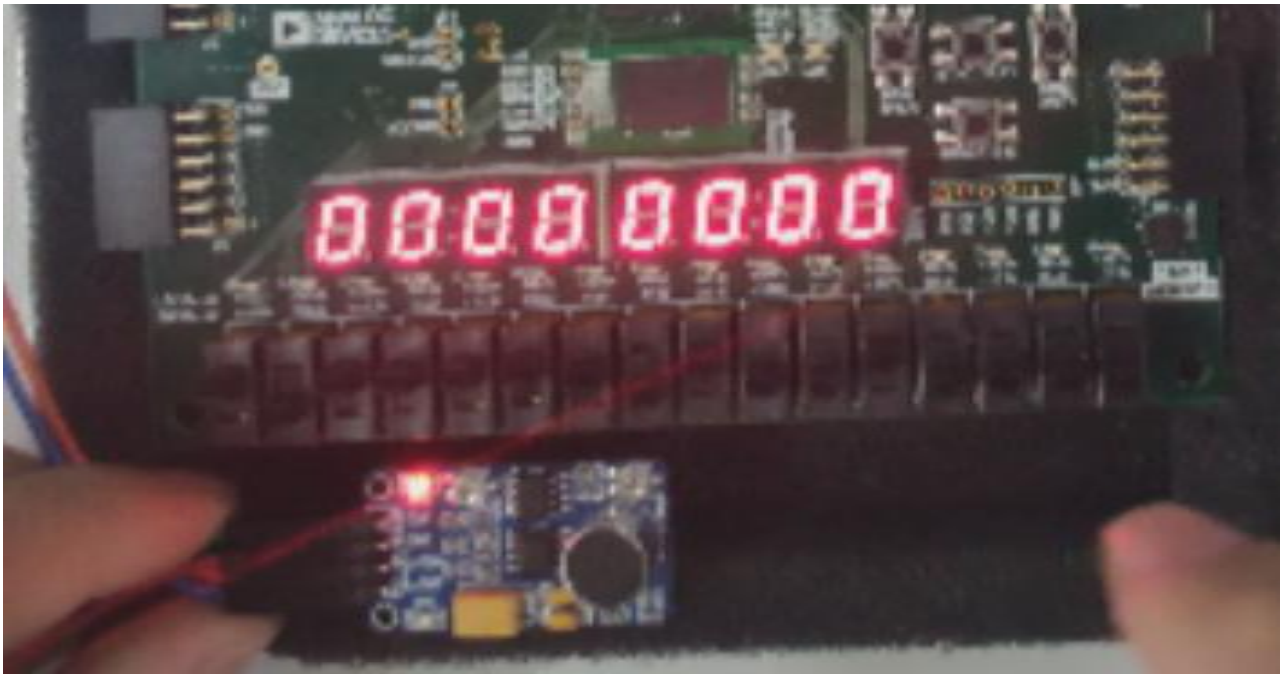
Logism 贴图

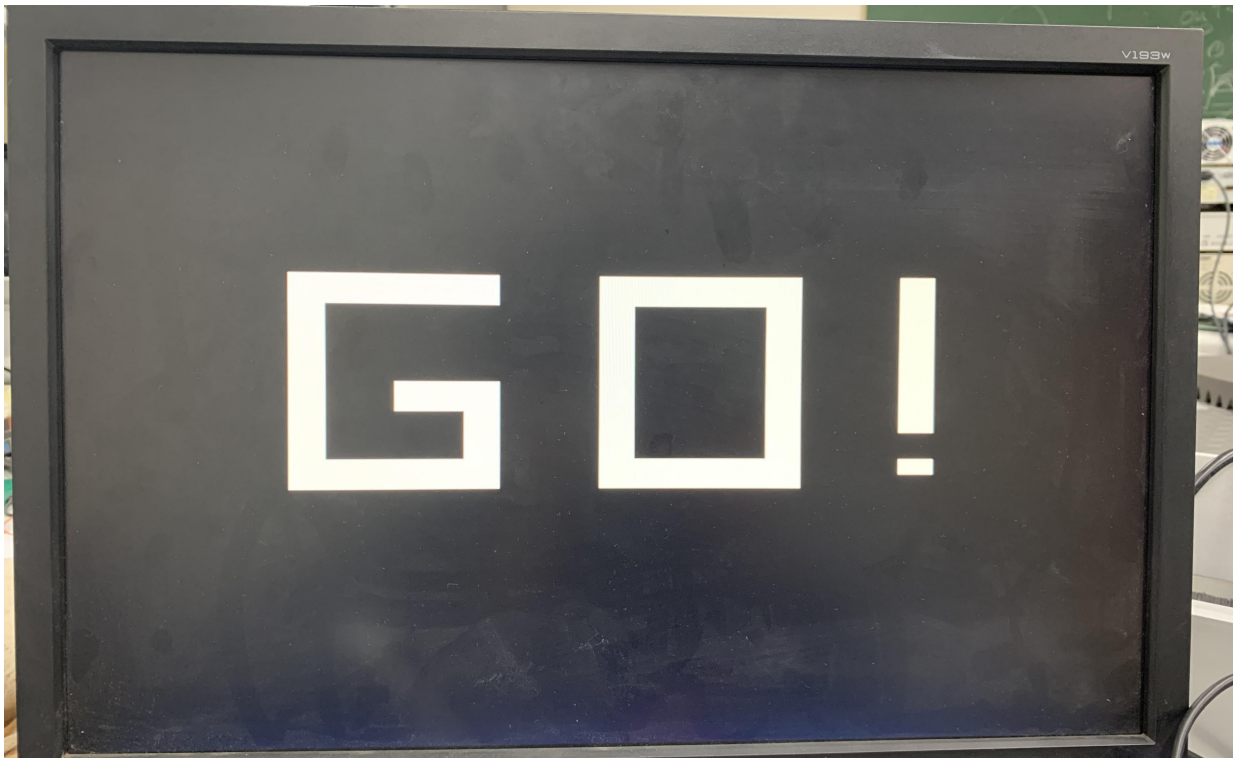


6.2 总实验结果

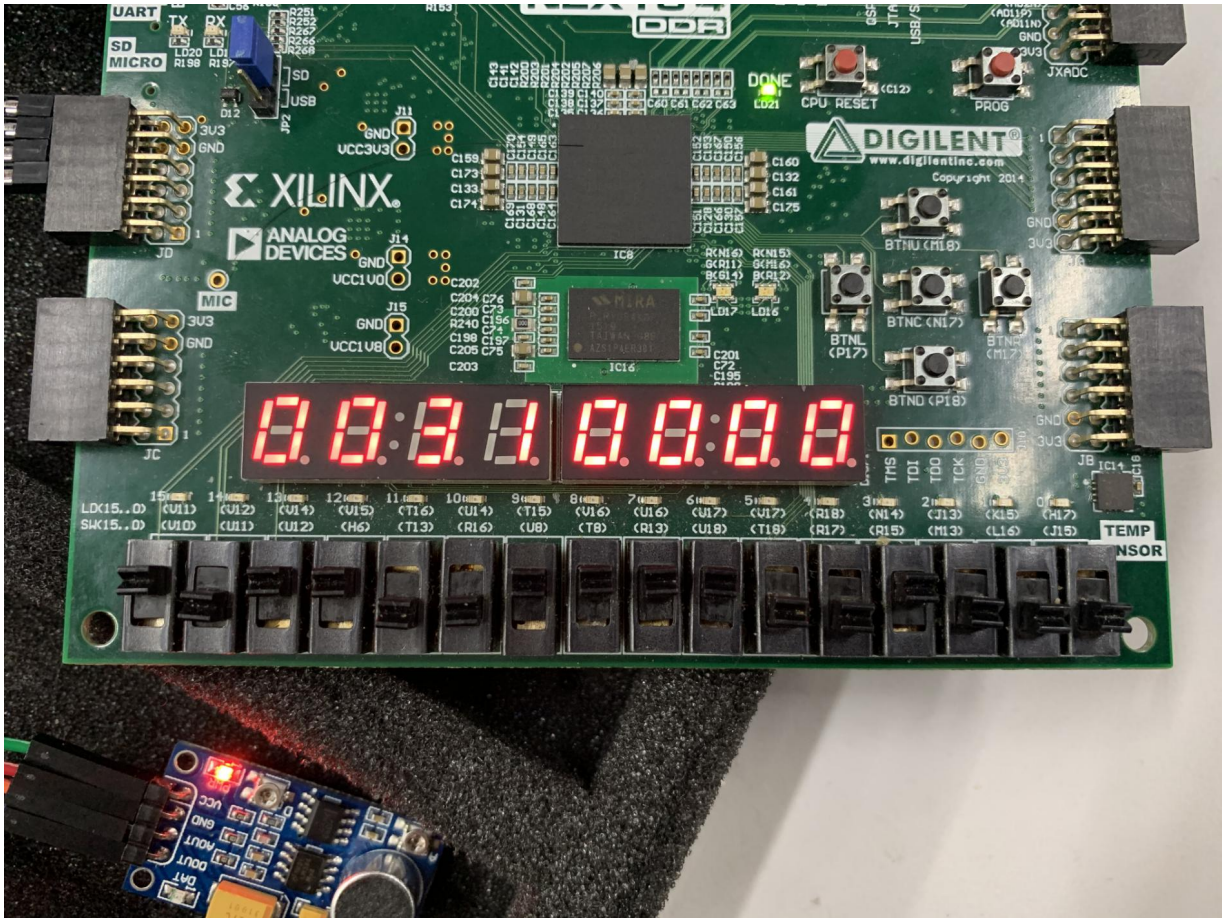
如下:

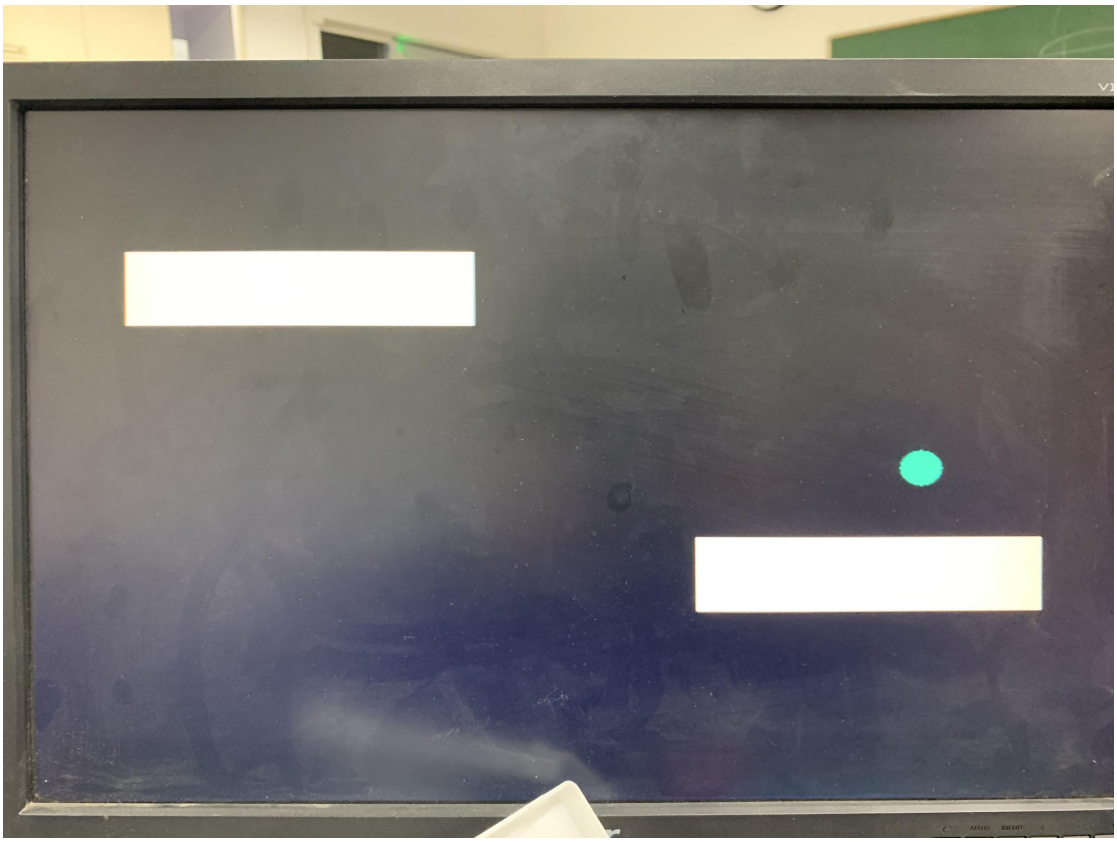
①如图所示，未推上 start 前，VGA 显示初始界面“GO”



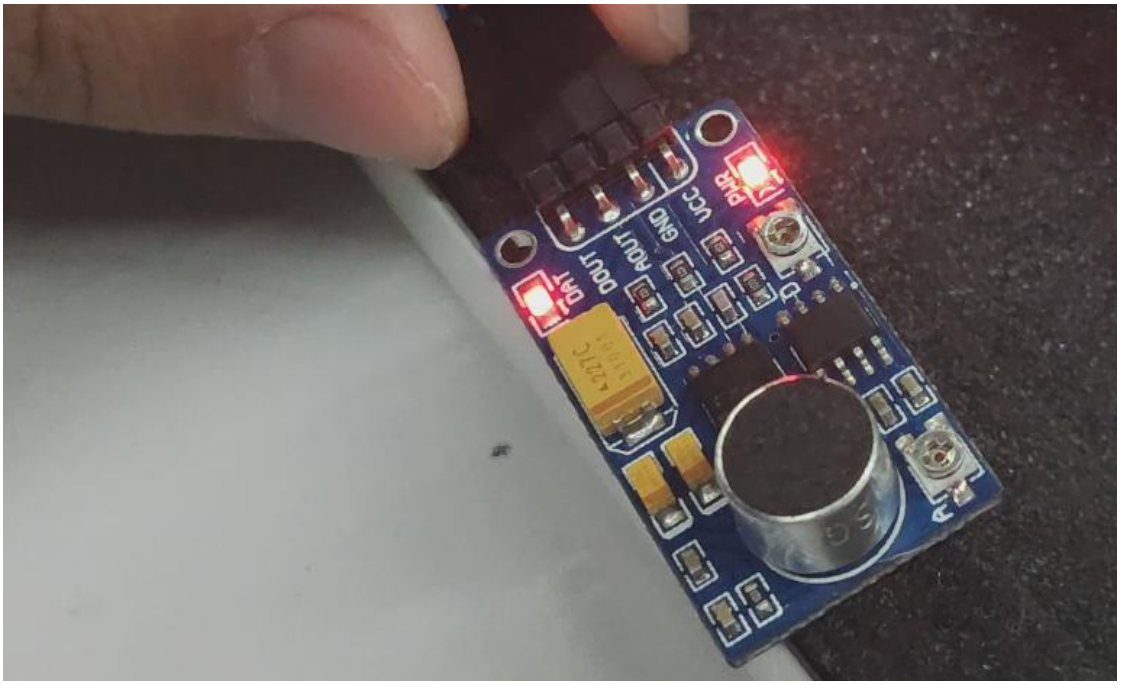


②按下 start 后，小球开始运动，同时开始倒计时





③到达底边后，对声音传感器吼一声或者点击声音传感器的咪头即可使小球正常反弹



④若小球在底边未正常反弹则游戏结束，输出“GG！”，重置 start 可以重新开始游戏

⑤到倒计时结束时，游戏正常结束，VGA 同样输出 GG！

六、 结论

从实际下板的情况来看，各个子模块无论是单独验证，还是合在一起下板，均能按照预期正常工作。因此这一数字系统是能够正常工作的。

七、 心得体会及建议

7.1.本课程收获

7.1.1 日常课程收获

计算机学科是一个实践性很强的学科，包括计算机科学与技术、计算机科学、计算机工程、信息安全、信息系统等诸多专业，数字逻辑作为计算机学科基础 12 门核心课程三硬中的第一门课程起到了硬件启蒙的作用。

课程安排的实践环节培养了学生的创新能力，中科院杨叔子院士有句至理名言--创新源于实践，工程院袁隆平院士历经辛苦发明的杂交水稻再一次证明--创新源于实践。胡锦涛在清华大学百年校庆讲话中强调指出:“科学理论、创新理论来源是实践，有服务于实践。要坚持理论联系，积极投身社会实践，。。。在实践中发现新知，运用真知”

创新是实践之果，实践是创新之根，二者相互依存。

同时，通过学习这门课程，我们的理论思维、实验思维、计算思维也得到了提升。这三种思维正对应着推动人类文明进步和科学技术发展的三大支柱--理论科学、实验科学、计算科学。

数字逻辑作为一门基础的硬件课程，让我学习了以前从未了解过的硬件的世界，学到了数字电路的设计方法和一门硬件描述语言，从最简单的与或非组合逻辑，到如何从零开始设计一个功能模块，从高级语言编程的思维方式转变到开始思考怎样用硬件的方式去思考设计 verilog 程序。思维方式拓宽了，也逐渐熟悉硬件编程的特性以及

电子电路设计的技巧与方法。

7.1.2 对于这次的大作业:

①对于设计完成一个规模较大的数字系统工程时，需要在重复全面调研，合理估计后，初步拟定集合，划分任务进度，然后逐步推进。在设计顶层模块时，需要考虑到如何合理将一个完整的想法分割成不同的小模块，使程序更加简洁，更容易编写。

例如这次大作业，我初步的想法需要 VGA 显示器，声音传感器以及 mp3 模块，是小球以声音传感器为信号与显示器的边界进行碰撞，在正常碰撞后 mp3 模块会发出弹球碰撞墙壁的声音。基本任务是让小球能正常与显示器边界进行碰撞，进阶任务是在显示器上显示各种障碍物妨碍小球到达下边界，减慢得分速度。由于显示器是做这个项目的基础，所以我先尝试弄懂 VGA 的显示部分，从显示画面的原理到能熟练控制 VGA 显示我想要的东西花了不少时间，以至于 mp3 模块无暇开发，所以我深深感受到动手开始做项目前规划好整个项目的进程是多么重要。这样可以保证在即便部分问题无法解决或解决不到位的情况，也能不影响整体项目的进度。

②在调试过程中，发现当一个子模块的逻辑有错误、混乱的时候，会影响到另一个与之毫不相干的子模块，以至于整体调试时，不知道是哪个模块产生问题。因此在编写时，可以先分别编写子模块、单独测试各个子模块是否功能完善。待确保所有子模块功能完善时，再拼接成一个完整的系统，进行调试。这样出错的可能性更小，在进行整体调试的时候也更容易 debug。

7.2.建议

希望采购外设时可以选购一些更常见或资料充分的模块，避免一些陈旧且资料不全的模块。

7.3 对数字芯片设计的认识与体会

课程的作业需要用 vivado 编程，modelsim 模拟实验结果，用 logisim 画原理图，这三个软件国外的软件，从这一点可以看出我们芯片设计需要的 EDA 软件，并没有完全国产化，仍然非常依赖国外的软件。软件配套的仿真模拟软件也是同样的，而且仿真模拟的软件成熟需要业界积累经验，优化设计，需要无数用户反馈与经验总结才能出一个好产品。

根据我查到的资料，芯片的生产技术在我国也是巨大的瓶颈。相关的技术困难在生产的机器和相关的人才。相关人才的缺失更是比缺少精密的机器更加严重的问题。业界有个说法，即使我给你机器，精度如此的高的机器，即使是有经验的人在调试中往往也可能会出错. 何况，在我国芯片产业积累如此少量的人才，绝大部分都是国外经验人员挖回来的. 以本人的愚见，现在媒体过度宣扬机器的问题，但是缺忽略了人才的培训等战略资源. 一旦我们得到了机器后，但是无人懂得如何操作，岂不是很丢脸么？所以本人觉得，我国应该要开始培训芯片产业链上的工人(国外往往是博士级别). 当硅圆生产完成后，就需要封装了，切成一小片，然后变成了一片片价格昂贵的芯片了. 这也是完全机器化分割，在这一类的机器中，外国到底有没有掐我国的脖子？因为信息的缺乏，所以本人并不清楚.