

**数字逻辑设计实验报告**

**实验课程： 《数字逻辑设计》**

**实验项目： 乒乓球游戏设计**

**系 别： 通信工程**

**年 级： 23级**

**学生姓名： vann**

**学生学号： \*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

# 一、实验内容介绍

## 1.1、题目简述

兵乓球游戏设计：8个LED灯视为球桌，未开球时LED全灭，开球后任何时刻只有1个LED灯亮视为球的位置，按键作为球拍，按下即挥拍，过网接才有效，可以设置1-2个接球位置，只有球到了规定位置且接球球员按下该位置对应的按键才算接球成功，早接或者晚接都视为接球失败。4个数码管显示每局得分，11分制；2个数码管显示大局得分，7局4胜。初始化时，LED灯全灭，只有一方挥拍后球才相应方起始位置开始移动（注：按键按下为“0”）。

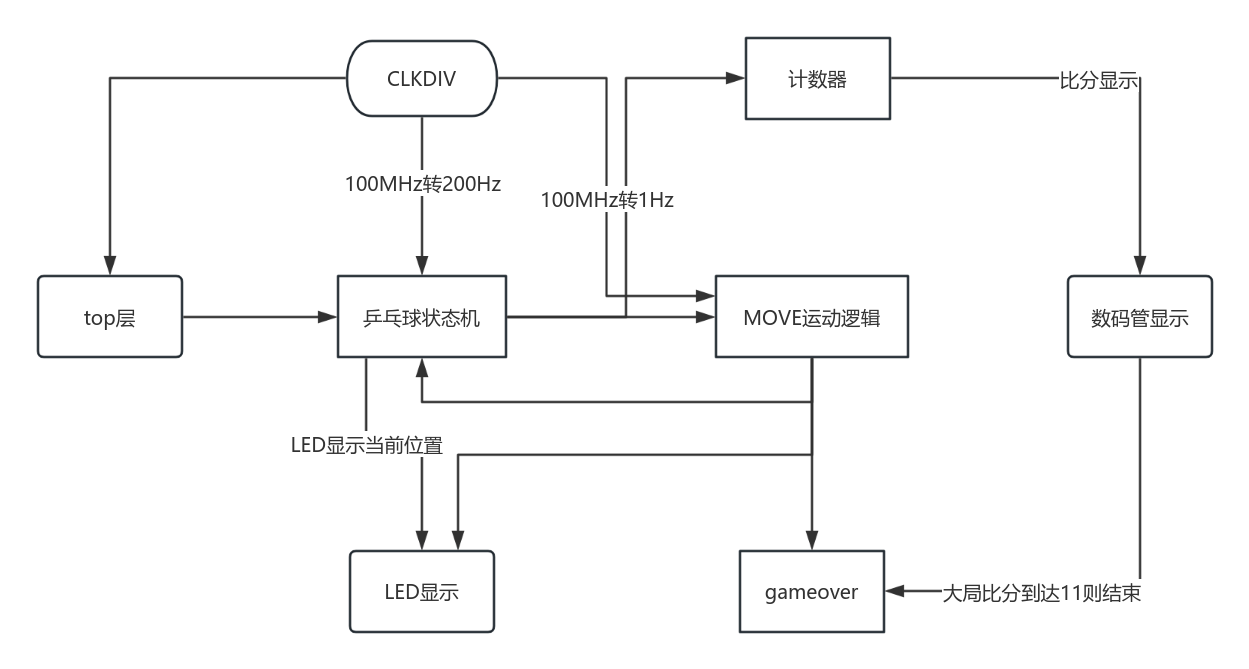
通过题目可知，本次实验是本实验使用两个按键分别模拟两个球员的挥拍动作，8个LED灯显示乒乓球的实时位置，动态数码管显示小局比分，静态数码管显示大局比分。实验成功实现了全部基本功能。此外，还增加了长接和短接两种模式，长接模式对应在球到达最后位置时挥拍，而短接模式则对应倒数第二个位置挥拍。系统自动检测先按下按键的一方并由其开始发球。当一方球员达到11分后，小局结束并大比分增加1分。

## 1.2、完成情况

利用两个按键作为接球按键，8个LED灯作为乒乓球位置指示灯，动态数码管作为小局分数指示，静态数码管作为大局分数指示，我顺利的完成了所有的功能。在此基础上，我还对接球进行了区分，不仅有长接接球还有短接接球。各自绑定两个按键。长接接球则是当乒乓球运动到最后一位的时候按下进行接球。短接则被我设定为倒数第二位的时候的接球动作。这两个按键只有乒乓球到了特定位置的时候才有效，提早按或者晚按都会导致失分。在开球是自动检测谁按下了按键，哪一边按下按键就从哪一边发球。并且，当一方得到11分之后，小居游戏结束，大局分数加1.双方的小局分数显示在动态数码管上，大局分数显示在静态数码管上。

# 二、设计思路

系统包括顶层模块、乒乓球状态机、时钟分频模块、数码管显示模块、计分模块和游戏结束检测模块。本次实验的模块主要有控制乒乓球整体状态的乒乓球状态机，控制乒乓球移动的move函数，对时钟进行分频的clkdiv函数，控制游戏结束的gameover函数，以及数码管显示函数以及动态数码管所需要的bin2bcd函数。整体框图如下：



**图一 整体框图**

# 设计代码

## 3.1顶层模块（top层）

顶层模块主要负责实例化各个子模块并完成模块间的数据连接与协调。

module top(

input clk,

input clr,

input play1,

input play2,

input play1l,

input play2l,

output [7:0] led,

output [6:0] a\_to\_g,

output [3:0] an,

output [3:0] win1,

output [3:0] win2

);

顶层实例化了时钟分频、状态机、数码管显示、bin转BCD、球运动、计分模块等，并完成数据连线。

## 3.2乒乓状态机

状态机一共有四个状态，分别为Init（初始化），L\_move(球向左边移动)，R\_move(球向右边移动)，Score（一方得分）。当处于初始化状态的情况下，将检测哪一边的按键按下就从哪一边来发球，然后进入相应的移动方向进行移动。然后如果提早接球或者为接到球则进入得分结算Score状态然后再次进入Init的状态。如果正常接到球(包括长接和短接)就让它的移动方向反向。

具体运行逻辑如下：Init：初始化并等待发球；L\_move/R\_move：球分别向左或向右移动；L\_hit/R\_hit：左或右侧球员击球后的短暂状态；L\_early/R\_early：球员提前击球状态；Score：得分状态。状态机通过检测按键的状态和球的位置来完成不同状态的转换，实现击球、移动、计分等逻辑。

关键状态：

parameter Init = 3'b000,

L\_hit = 3'b001,

R\_hit = 3'b010,

L\_move = 3'b011,

R\_move = 3'b100,

Score = 3'b101,

L\_early = 3'b110,

R\_early = 3'b111;

状态转移由按键状态和球的位置进行决定，满足条件则进入下个状态，例如：

Init: if (hit\_l == 0 && index == 10'b1000000001)

next\_state <= L\_hit;

## 3.3 数码管显示

动态数码管：采用时钟分频技术，每5ms切换一次显示，达到视觉暂留效果，用于显示小局比分。

静态数码管：固定显示大局比分。

### **3.4 bin转bcd码（bin2bcd）**

该模块用于将二进制数转换为BCD码，考虑到本实验计数范围仅为0-11，因此采用直接逻辑判断进行快速转换。实现分数二进制转BCD码以便于数码管显示。

module bin2bcd(

input [3:0]in\_data,

output reg [4:0]data\_bcd

);

always @ (in\_data) begin

if(in\_data > 4'd9)

data\_bcd <= {1'b1,in\_data-4'd10};

else

data\_bcd <= {1'b0,in\_data};

end

## 3.5游戏结束判断逻辑（gameover 模块）

将状态机中的Score得到得计数值引出，接到gameover模块上。当检测到得分到达11，则将对应玩家的大比分加一。一直直到加到4的时候，输出结束游戏信号，使得乒乓球状态机状态回归Init状态并把数据清零。

## 3.6时钟分频逻辑（clkdiv）

由于本次移动乒乓球的位置需要用到1hz的时钟，因此需要将高频时钟分频为较低频的时钟。时钟的分频逻辑是通过计数器对寄存器变量进行反转，从而得到较为低频的时钟。该模块实现了对时钟100MHz进行分频，使用的FPGA开发板有100MHz、50MHz、3Hz三个频率的时钟，这三个频率不能满足设计需求，需要对时钟进行分频。

always @ (posedge clk or negedge clr) begin

if(clr == 0) begin

clk\_200\_cnt <= 0; clk\_200 <= 0;

end else if(clk\_200\_cnt == 19'd249999) begin

clk\_200\_cnt <= 0; clk\_200 <= ~clk\_200;

end else

clk\_200\_cnt <= clk\_200\_cnt + 1;

end

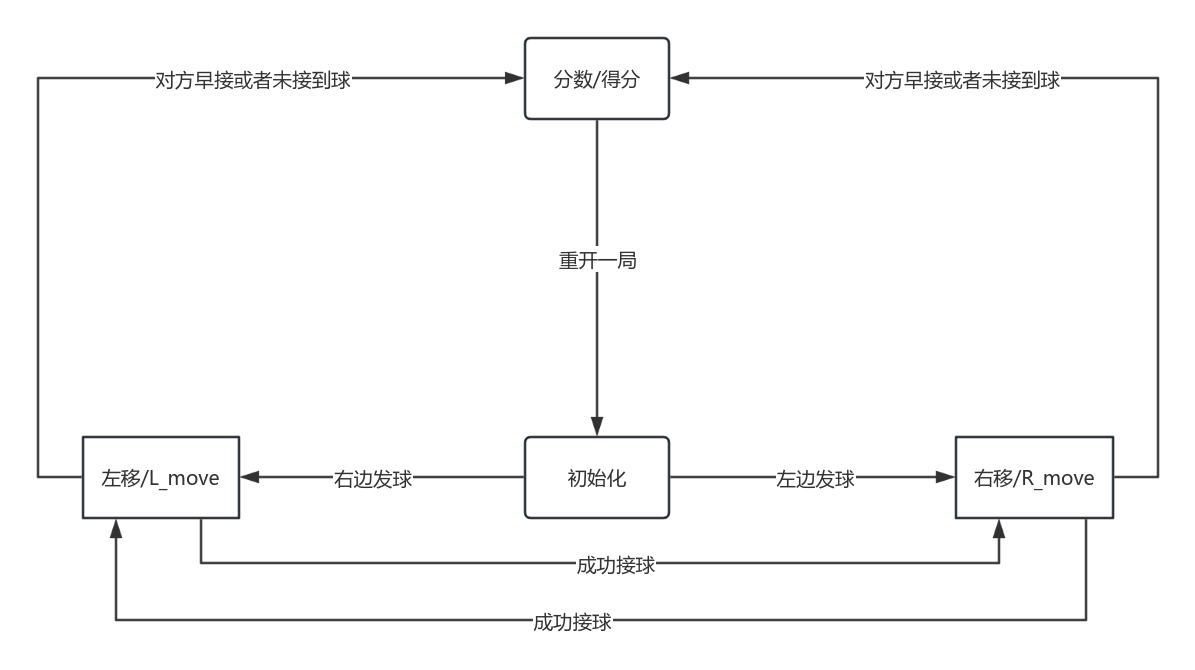
将系统高频时钟（100MHz）分频成所需的200Hz和1Hz。

### 3.7 数码管驱动模块（x7seg）

实现动态数码管和静态数码管的驱动与显示控制。

### 3.8 球运动模块（move模块）

move模块负责控制乒乓球的实时位置与移动方向，根据来自状态机的控制信号（球的起始位置和移动方向），生成对应的LED灯控制信号。模块内根据输入的球起始信号和方向信号，在每个时钟周期更新球的位置。当球到达边界位置时停止移动，直到下一次接球或发球指令到来。move模块确保乒乓球位置移动精确可靠，并与状态机紧密配合，实现游戏逻辑的顺畅运行。



**图二 游戏逻辑**

根据状态机的信号(ball\_start和ball\_dir)实时更新球的位置：

module move(

input clk,

input clr,

input [1:0] ball\_start,

input [1:0] ball\_dir,

output reg [9:0] index

);

always @(posedge clk or negedge clr) begin

if (!clr)

index <= 10'b1000000001;

else begin

case (ball\_start)

2'b01: index <= 10'b1000000000;

2'b10: index <= 10'b0000000001;

2'b11: index <= 10'b1000000001;

2'b00: begin

case (ball\_dir)

2'b01: if (index != 10'b1000000000) index <= index << 1;

2'b10: if (index != 10'b0000000001) index <= index >> 1;

default: index <= index;

endcase

end

default: index <= index;

endcase

end

end

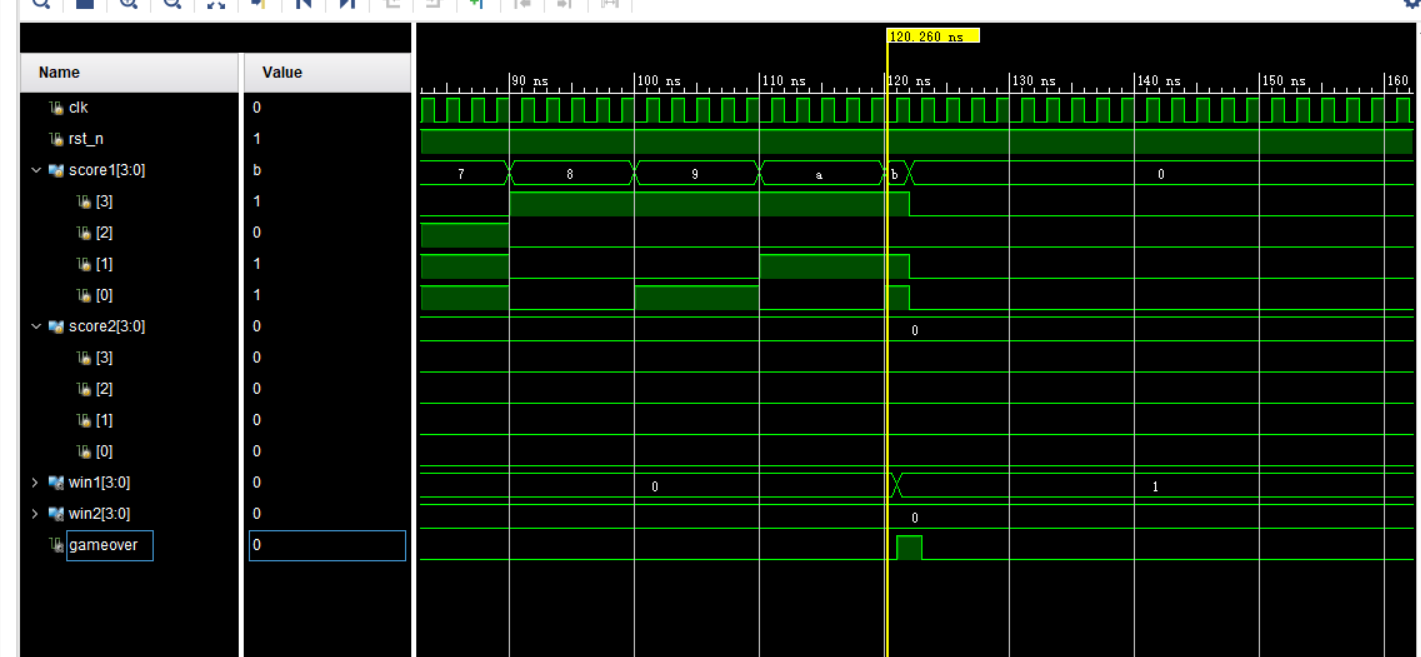
# 四、仿真结果

## 4.1计数模块仿真

### 4.1.1仿真代码

|  |
| --- |
| `timescale 1ns / 1ps  module fz\_gameover(  );  reg clk;  reg rst\_n;  reg [3:0]score1;  reg [3:0]score2;  wire [3:0]win1;  wire [3:0]win2;  wire gameover;    gameover u1(  .clk(clk),  .rst\_n(rst\_n),  .score1(score1),  .score2(score2),  .win1(win1),  .win2(win2),  .gameover(gameover)  );  initial  begin  rst\_n=0;  clk=0;  score1=0;  score2=0;  #10  rst\_n=1;  repeat(11)  begin  #10  score1=score1+1;  end  #2  score1=0;  repeat(11)  begin  #10  score1=score1+1;  end  #2  score1=0;  repeat(11)  begin  #10  score1=score1+1;  end  #2  score1=0;  repeat(11)  begin  #10  score1=score1+1;  end  #2  score1=0;  end            always #1 clk=~clk;  endmodule |

### 4.1.2仿真结果







### 4.1.3仿真分析

由图可知，在计数到11时，win1（1号选手的大比分）加一，并且输出了结束信号，分数进行清零，因此计数模块没有错误。

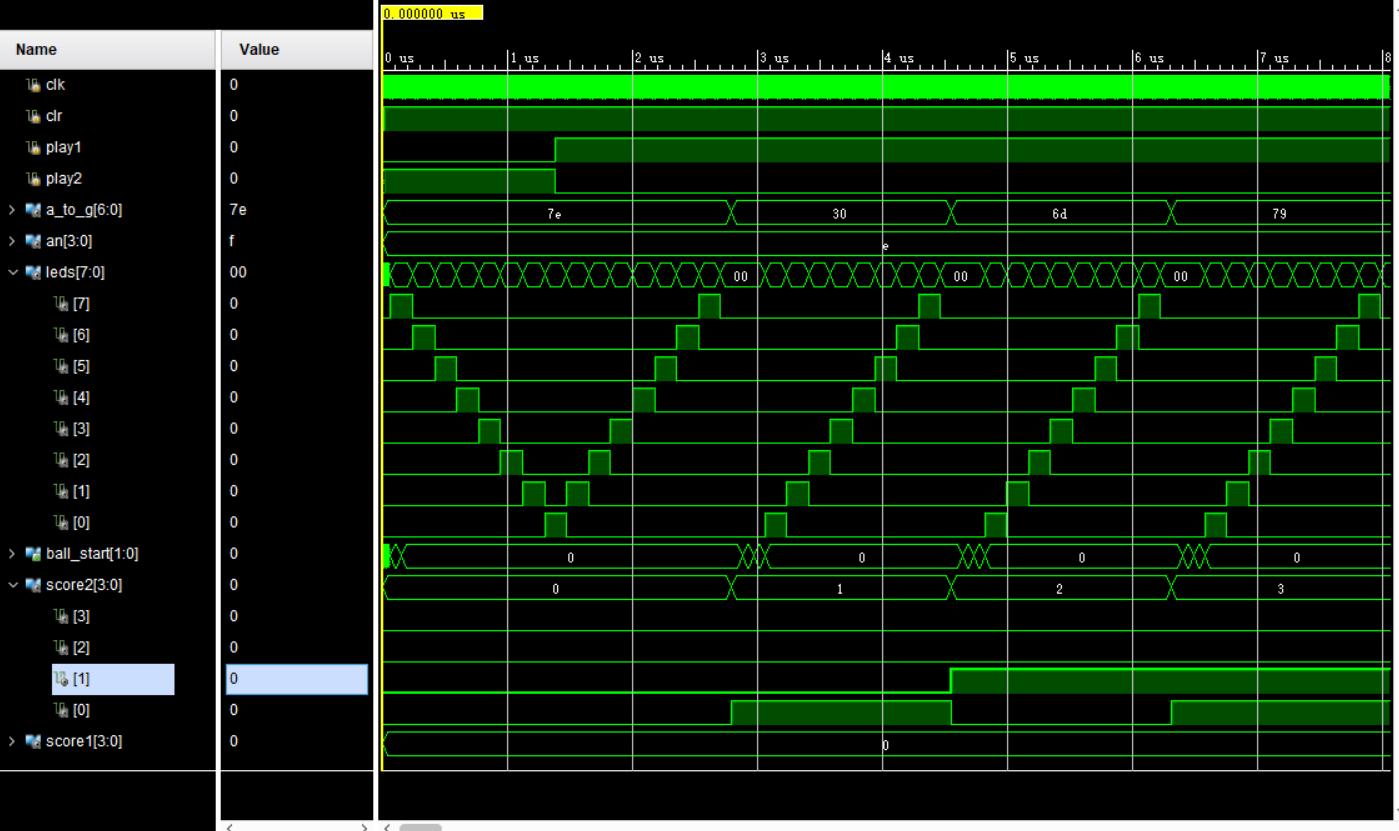
同时，当大比分记到4的时候，此时说明有一方已经获得了比赛的胜利，此时将大小比分同时清零，并且输出游戏结束信号。

## 4.2 接球积分功能仿真

### 4.2.1仿真代码

|  |
| --- |
| `timescale 1ns / 1ps      module fzdd(      );      reg clk;      reg clr;      reg play1;      reg play2;      reg play1l;      reg play2l;      wire[6:0]a\_to\_g;      wire[3:0]an;      wire[7:0]leds;      top u1(          .clk(clk),          .clr(clr),          .play1(play1),          .play2(play2),          .a\_to\_g(a\_to\_g),          .an(an),          .led(leds)      );      initial      begin         clk=0;         clr=0;         play1=0;         play2=0;         #5         clr=1;         play1=0;         play2=1;         #176         play1=0;         play2=1;         #1200         play1=1;         play2=0;      end      always #1 clk=~clk;  endmodule |

### 4.2.2仿真结果



### 4.2.3仿真分析

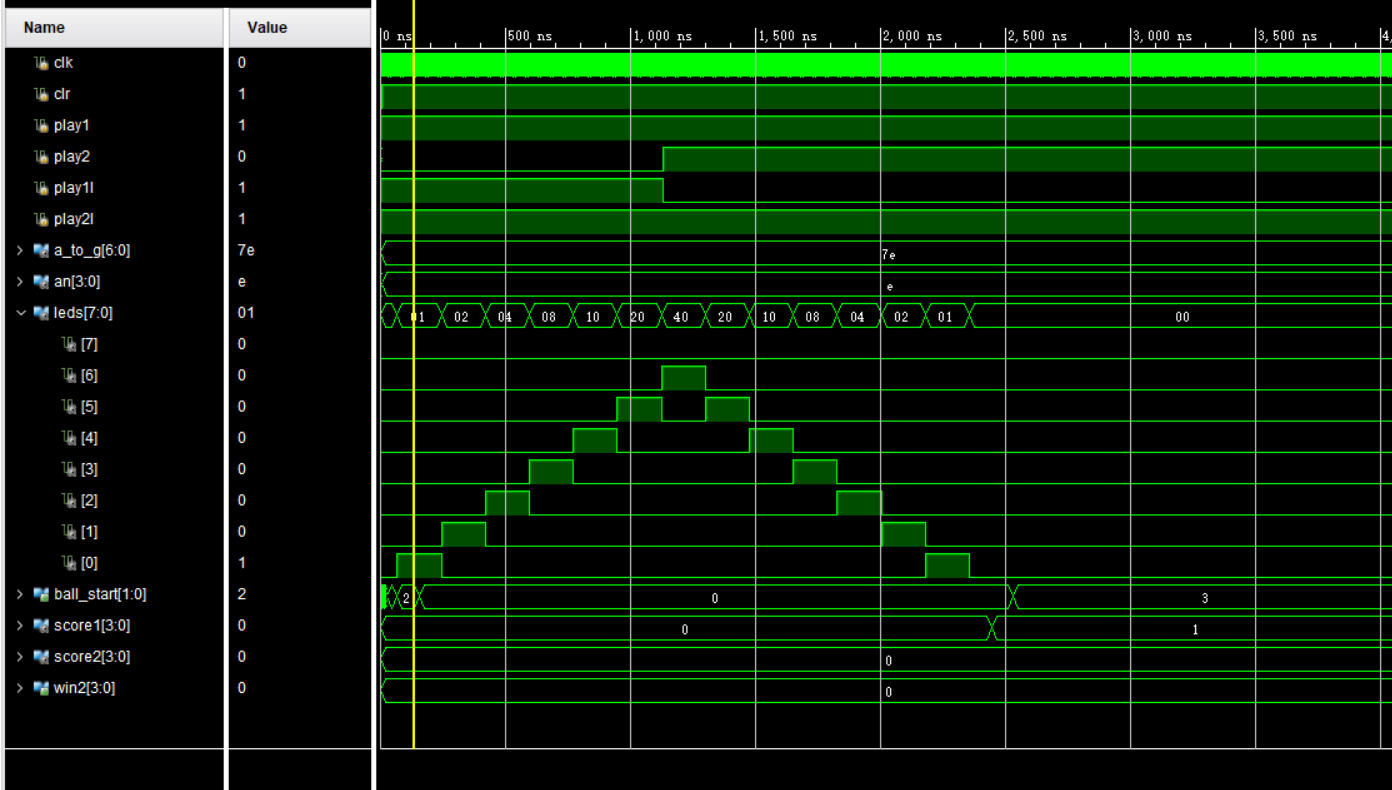
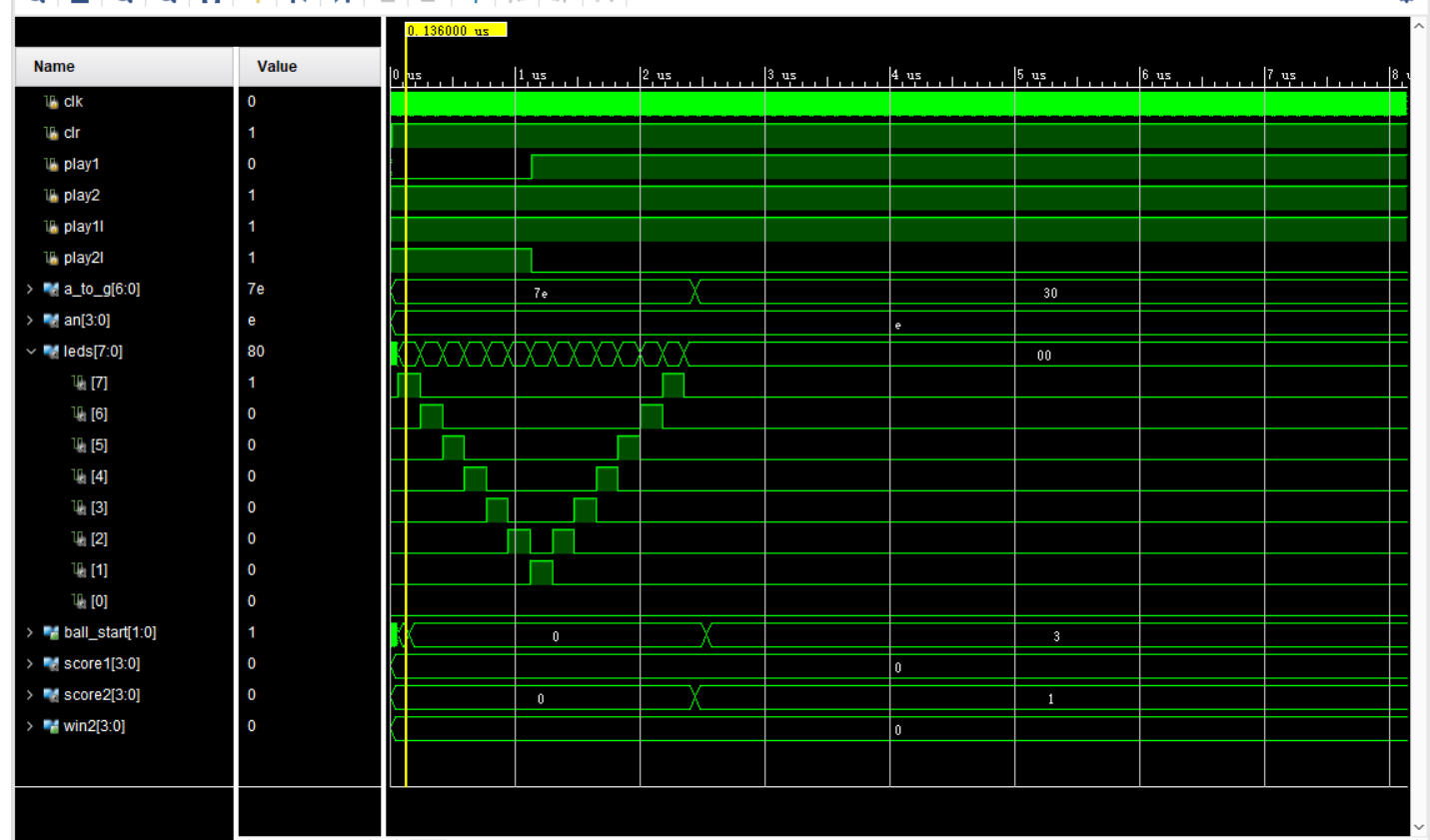
在一号选手发球后，球开始向二号选手移动。并且二号选手在最后接住了球。在其之后二号选手连续发球但是一号选手没能接球。在图上显示可知二号选手得分Score2正常。每次到一号选手未接上球就多得一分。

## 4.3 短接功能仿真

### 4.3.1仿真代码

|  |
| --- |
| `timescale 1ns / 1ps      module fzdd(      );      reg clk;      reg clr;      reg play1;      reg play2;      reg play1l;      reg play2l;      wire[6:0]a\_to\_g;      wire[3:0]an;      wire[7:0]leds;      top u1(          .clk(clk),          .clr(clr),          .play1(play1),          .play2(play2),          .play1l(play1l),          .play2l(play2l),          .a\_to\_g(a\_to\_g),          .an(an),          .led(leds)      );      initial      begin         clk=0;         clr=0;         play1=1;         play2=1;         play1l=1;         play2l=1;         #5         clr=1;         play2=0;         play1=1;         #176         play2=0;         play1=1;         #950         play2=1;         play1l=0;         play1=1;      end      always #1 clk=~clk;  endmodule |

### 4.3.2仿真结果

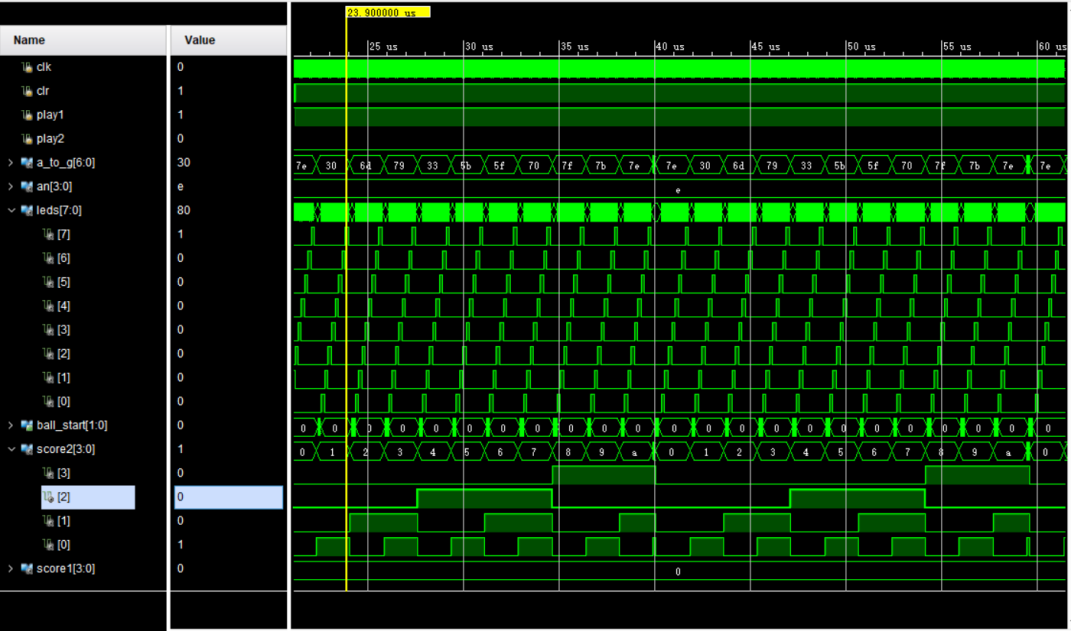
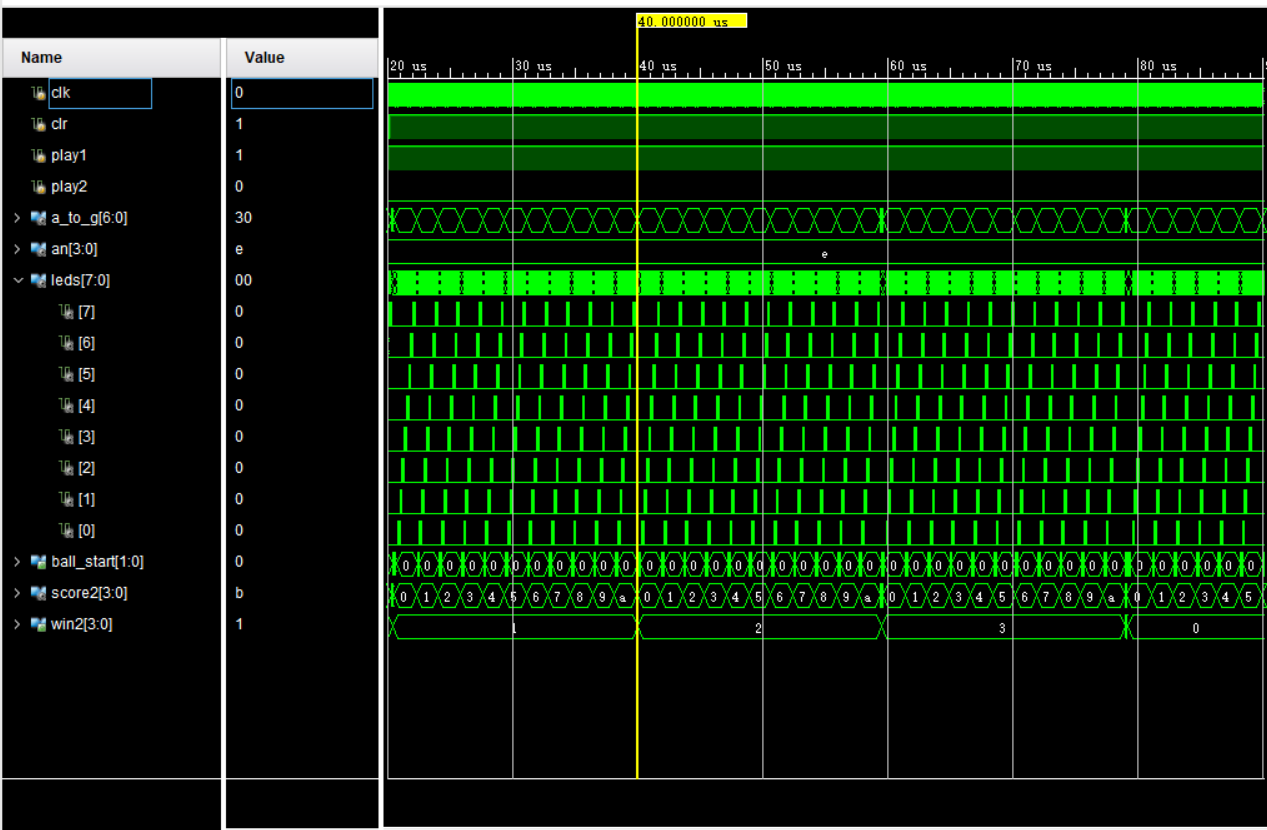


### 4.3.3仿真分析

由上图可知，二位选手发球后，对手可以通过短接按键来在倒数第二位来接球并使球的运动方向取反。此时，记分的功能正常。

## 4.4整体仿真

### 4.4.1仿真结果



### 4.4.2仿真分析

让一位选手不断的发球，观察计分情况，可以看到小局积分和大局积分都正常。

# 五、实际展示结果



### 初始状态时在系统刚刚上电或按下复位按钮后，所有8个LED灯处于熄灭状态，4位动态数码管和2位静态数码管均显示数字0，此时游戏处于等待发球状态。任意一方球员按下对应按键时，8个LED灯立即从该球员一侧的初始位置开始亮起，表示球开始向对方运动。在运动阶段——LED灯依次逐个亮起并熄灭，以平滑的方式从球员一侧向另一侧逐步移动，直观地展示乒乓球在球桌上的运动轨迹。球到达对方指定接球位置时（长接在最边缘位置，短接在倒数第三个位置），对应玩家需准确按下相应按键完成挥拍动作，若接球成功，LED运动方向立即反转，球向对方方向返回。若球未到指定位置，玩家提前按键挥拍（包含过半检测，必须要过半场才会扣分），则视为提前击球失败，LED熄灭，短暂停顿后重新进入待发球状态，对方比分自动增加1分。若球已经超过指定位置，玩家按键挥拍，则视为延迟击球失败，同样进入重新发球状态，对方比分增加1分。

### 积分显示——每次得分后，动态数码管显示小局比分实时增加，直观且迅速地更新当前分数。当某方达到11分时，小局结束，大比分数码管相应地自动增加1分。每局结束（达到11分）后，8个LED灯全灭，数码管显示的小比分归零，大比分继续保留。当任意一方大比分达到4分时，整个比赛结束，静态和动态数码管同时闪烁提示最终胜利方，随后所有得分归零重新进入待发球状态，准备新一轮比赛。

# 附录：

**Top层代码：**

`timescale 1ns / 1ps

// Description: 顶层模块，用于一个乒乓球游戏。包括时钟分频、游戏逻辑、得分显示和游戏结束检测。

module top(

input clk, // 100MHz时钟

input clr, // 复位信号

input play1, // 玩家1长按按钮

input play2, // 玩家2长按按钮

input play1l, // 玩家1短按按钮

input play2l, // 玩家2短按按钮

input fast\_mode,

output [7:0] led, // 8个LED灯

output [6:0] a\_to\_g, // 7段数码管的段选信号

output [3:0] an, // 7段数码管的位选信号

output [3:0] win1, // 玩家1胜利指示

output [3:0] win2 // 玩家2胜利指示

);

// 内部连线

wire [9:0] index;

wire clk\_200;

wire clk\_1;

// 时钟分频模块实例化

clkdiv u1 (

.clk(clk),

.clr(clr),

.fast\_mode(fast\_mode),

.clk\_200(clk\_200),

.clk\_1(clk\_1)

);

// 内部连线，用于分数和游戏逻辑

wire [3:0] score1, score2;

wire [1:0] ball\_dir, ball\_start;

wire gameover;

// 乒乓球游戏状态机

pingpangztj u2 (

.clk(clk\_200),

.clr(clr),

.index(index),

.ball\_dir(ball\_dir),

.ball\_start(ball\_start),

.hit\_l(play1),

.hit\_r(play2),

.hit\_l\_s(play1l),

.hit\_r\_s(play2l),

.score1(score1),

.score2(score2),

.gameover(gameover)

);

// 二进制转BCD码转换器，用于分数显示

wire [7:0] score1\_bcd, score2\_bcd;

bin2bcd u3 (

.in\_data(score1),

.data\_bcd(score1\_bcd)

);

bin2bcd u4 (

.in\_data(score2),

.data\_bcd(score2\_bcd)

);

// 7段数码管驱动

x7seg u5 (

.x({score1\_bcd, score2\_bcd}),

.clk(clk),

.clr(clr),

.a\_to\_g(a\_to\_g),

.an(an)

);

// 球运动逻辑

move u6 (

.clk(clk\_1),

.clr(clr),

.index(index),

.ball\_start(ball\_start),

.ball\_dir(ball\_dir)

);

// 游戏结束逻辑

gameover u7 (

.clk(clk\_200),

.rst\_n(clr),

.score1(score1),

.score2(score2),

.win1(win1),

.win2(win2),

.gameover(gameover)

);

// 分配LED输出 - 显示球的位置

assign led = index[8:1];

endmodule

**clkdiv分频代码：**

`timescale 1ns / 1ps

module clkdiv(

input clk,

input clr,

input fast\_mode,

output reg clk\_200,

output reg clk\_1

);

reg [18:0] clk\_200\_cnt;

reg [6:0] clk\_1\_cnt;

// 生成200Hz时钟（用于游戏逻辑）

always @ (posedge clk or negedge clr) begin

if (clr == 0) begin

clk\_200\_cnt <= 0;

clk\_200 <= 0;

end

else if (clk\_200\_cnt == 19'd249999) begin // 100MHz/250000 = 200Hz

clk\_200\_cnt <= 0;

clk\_200 <= ~clk\_200;

end

else

clk\_200\_cnt <= clk\_200\_cnt + 1;

end

always @ (posedge clk\_200 or negedge clr) begin

if (clr == 0) begin

clk\_1\_cnt <= 0;

clk\_1 <= 0;

end

else if ((fast\_mode && clk\_1\_cnt == 7'd19) || (!fast\_mode && clk\_1\_cnt == 7'd99)) begin

clk\_1\_cnt <= 0;

clk\_1 <= ~clk\_1;

end

else

clk\_1\_cnt <= clk\_1\_cnt + 1;

end

/\*

// 生成1Hz时钟（用于球移动）

always @ (posedge clk\_200 or negedge clr) begin

if (clr == 0) begin

clk\_1\_cnt <= 0;

clk\_1 <= 0;

end

else if (clk\_1\_cnt == 7'd99) begin // 200Hz/100 = 2Hz

clk\_1\_cnt <= 0;

clk\_1 <= ~clk\_1;

end

else

clk\_1\_cnt <= clk\_1\_cnt + 1;

end

\*/

endmodule

**乒乓球状态机部分代码：**

`timescale 1ns / 1ps

module pingpangztj(

input wire clk, // 时钟信号

input clr, // 复位信号

input [9:0] index, // 球移动到的位置

input hit\_l, // 玩家1长按按钮

input hit\_r, // 玩家2长按按钮

input hit\_l\_s, // 玩家1短按按钮

input hit\_r\_s, // 玩家2短按按钮

input gameover, // 游戏结束信号

output reg [1:0] ball\_start, // 球开始状态

output reg [1:0] ball\_dir, // 球运行方向

output reg [3:0] score1, // 玩家1得分

output reg [3:0] score2 // 玩家2得分

);

// 状态定义

parameter Init = 3'b000, // 初始化状态，等待发球

L\_hit = 3'b001, // 左侧玩家击球状态

R\_hit = 3'b010, // 右侧玩家击球状态

L\_move = 3'b011, // 球向左移动状态

R\_move = 3'b100, // 球向右移动状态

Score = 3'b101, // 计分状态

L\_early = 3'b110,// 左侧玩家提前击球状态

R\_early = 3'b111;// 右侧玩家提前击球状态

reg [2:0] present\_state; // 当前状态寄存器

reg [2:0] next\_state; // 下一个状态寄存器

// 用于下降沿检测的寄存器

// \_reg 后缀表示上一时钟周期的信号状态

reg hit\_l\_reg;

reg hit\_r\_reg;

reg hit\_l\_s\_reg;

reg hit\_r\_s\_reg;

// 状态寄存器逻辑：在时钟上升沿或复位信号下降沿更新当前状态和按钮状态寄存器

always @ (posedge clk or negedge clr) begin

if (!clr) begin

present\_state <= Init; // 复位时进入初始状态

// 同时复位下降沿检测寄存器，初始化为高电平，以便检测到第一次按下（下降沿）

hit\_l\_reg <= 1'b1;

hit\_r\_reg <= 1'b1;

hit\_l\_s\_reg <= 1'b1;

hit\_r\_s\_reg <= 1'b1;

end else begin

present\_state <= next\_state; // 更新当前状态

// 更新下降沿检测寄存器，存储当前输入信号的值，用于下一个时钟周期的比较

hit\_l\_reg <= hit\_l;

hit\_r\_reg <= hit\_r;

hit\_l\_s\_reg <= hit\_l\_s;

hit\_r\_s\_reg <= hit\_r\_s;

end

end

// 状态转移逻辑：根据当前状态和输入信号（下降沿）决定下一个状态

// 此处恢复为组合逻辑 always @ (\*) 块来计算 next\_state

always @ (\*) begin

next\_state = present\_state; // 默认情况下，下一个状态保持当前状态

case (present\_state)

Init: // 等待发球的状态

// 玩家1发球（长按按钮的下降沿检测）

if (hit\_l\_reg == 1'b1 && hit\_l == 1'b0 && index == 10'b1000000001) // 检测到左侧玩家发球按钮下降沿

next\_state = L\_hit;

// 玩家2发球（长按按钮的下降沿检测）

else if (hit\_r\_reg == 1'b1 && hit\_r == 1'b0 && index == 10'b1000000001) // 检测到右侧玩家发球按钮下降沿

next\_state = R\_hit;

else

next\_state = Init;

L\_hit: // 左侧玩家击球后球向右移动

next\_state = R\_move;

R\_hit: // 右侧玩家击球后球向左移动

next\_state = L\_move;

R\_move: // 球向右移动

// 玩家2短按按钮 (play2l) 在从右到左第三个位置 (index = 10'b0000000100) 接球，检测下降沿

if (index == 10'b0000001000 && hit\_r\_s\_reg == 1'b1 && hit\_r\_s == 1'b0)

next\_state = L\_move;

// 玩家2长按按钮 (play2) 在从右到左第二个位置 (index = 10'b0000000010) 接球，检测下降沿

else if (index == 10'b0000000010 && hit\_r\_reg == 1'b1 && hit\_r == 1'b0)

next\_state = L\_move;

// 提前击球（包括未过半场击球）逻辑 - 仅修改按钮检测为下降沿

else if ((index & 10'b0000011100 && (hit\_r\_reg == 1'b1 && hit\_r == 1'b0)) || (index & 10'b00000011000 && (hit\_r\_s\_reg == 1'b1 && hit\_r\_s == 1'b0)))

next\_state = R\_early;

// 球到达右边界，玩家1得分

else if (index == 10'b0000000001)

next\_state = Score;

else

next\_state = R\_move;

L\_move: // 球向左移动

// 玩家1短按按钮 (play1l) 在从左到右第三个位置 (index = 10'b0010000000) 接球，检测下降沿

if (index == 10'b0001000000 && hit\_l\_s\_reg == 1'b1 && hit\_l\_s == 1'b0)

next\_state = R\_move;

// 长按按钮 (play1) 在从左到右第一个位置 (index = 10'b1000000000) 接球，检测下降沿

else if (index == 10'b0100000000 && hit\_l\_reg == 1'b1 && hit\_l == 1'b0)

next\_state = R\_move;

// 提前击球（包括未过半场击球）逻辑 - 仅修改按钮检测为下降沿

else if ((index & 10'b0011100000 && (hit\_l\_reg == 1'b1 && hit\_l == 1'b0)) || (index & 10'b00011100000 && (hit\_l\_s\_reg == 1'b1 && hit\_l\_s == 1'b0)))

next\_state = L\_early;

// 球到达左边界，玩家2得分

else if (index == 10'b1000000000)

next\_state = Score;

else

next\_state = L\_move;

L\_early: // 左侧玩家提前击球

next\_state = Init;

R\_early: // 右侧玩家提前击球

next\_state = Init;

Score: // 计分状态

next\_state = Init;

default:

next\_state = Init; // 默认状态为初始状态

endcase

end

// 输出逻辑：根据当前状态和输入信号更新输出信号

always @ (posedge clk or negedge clr) begin

if (!clr) begin

ball\_dir <= 2'b00; // 复位时球方向置为0

ball\_start <= 2'b11; // 复位时球开始状态置为11（初始状态）

score1 <= 0; // 复位时玩家1得分置为0

score2 <= 0; // 复位时玩家2得分置为0

end else begin

if (gameover == 1) begin

score1 <= 0; // 游戏结束时玩家1得分置为0

score2 <= 0; // 游戏结束时玩家2得分置为0

end else begin

case (present\_state) // 注意这里是present\_state，确保在状态转换后才更新输出

Init: begin

ball\_dir <= 2'b00; // 初始状态时球方向置为0

ball\_start <= 2'b11; // 初始状态时球开始状态置为11

end

L\_hit: begin

ball\_start <= 2'b01; // 左侧玩家击球后球从左边开始

ball\_dir <= 2'b10; // 向右移动

end

R\_hit: begin

ball\_start <= 2'b10; // 右侧玩家击球后球从右边开始

ball\_dir <= 2'b01; // 向左移动

end

R\_move: begin

ball\_dir <= 2'b10; // 球向右移动时球方向置为10

ball\_start <= 2'b00; // 球向右移动时球开始状态置为00

end

L\_move: begin

ball\_dir <= 2'b01; // 球向左移动时球方向置为01

ball\_start <= 2'b00; // 球向左移动时球开始状态置为00

end

L\_early: begin

ball\_start <= 2'b11; // 左侧玩家提前击球时球开始状态置为11

ball\_dir <= 2'b00; // 左侧玩家提前击球时球方向置为00

score2 <= score2 + 1; // 更新玩家2得分

end

R\_early: begin

ball\_start <= 2'b11; // 右侧玩家提前击球时球开始状态置为11

ball\_dir <= 2'b00; // 右侧玩家提前击球时球方向置为00

score1 <= score1 + 1; // 更新玩家1得分

end

Score: begin

ball\_start <= 2'b11; // 计分后回到初始状态

ball\_dir <= 2'b00;

if (next\_state == Init) begin // 避免重复计分

if (index == 10'b0000000001) // 球在右边界，玩家1得分

score1 <= score1 + 1;

else if (index == 10'b1000000000) // 球在左边界，玩家2得分

score2 <= score2 + 1;

end

end

endcase

end

end

end

endmodule

**Bin转bcd代码：**

`timescale 1ns / 1ps

module bin2bcd(

input [3:0] in\_data,

output reg [7:0] data\_bcd // 修正位宽，支持两位BCD码

);

always @ (in\_data) begin

if (in\_data > 4'd9)

data\_bcd = {4'b0001, in\_data - 4'd10}; // 十位为1，个位为(in\_data-10)

else

data\_bcd = {4'b0000, in\_data}; // 十位为0，个位为in\_data

end

endmodule

数码管显示代码：

`timescale 1ns / 1ps

module x7seg(

input [15:0] x,

input clk,

input clr,

output reg [6:0] a\_to\_g,

output reg [3:0] an //一组4个数码管的片选信号

);

wire [1:0] s; // 选择数码管点亮的位置；

reg [3:0] digit;

reg [19:0] clkdiv;

assign s = clkdiv[19:18]; //18分频，每5.2ms变化一次，依次点亮4个数码管

//assign s = clkdiv[4:3]; //仿真测试，快速依次点亮4个数码

always @ ( \* )

case ( s ) //x16位，依次存放4位要显示的数

0:begin digit = x[3:0]; an=4'b1110; end

1:begin digit = x[7:4]; an=4'b1101; end

2:begin digit = x[11:8]; an=4'b1011; end

3:begin digit = x[15:12];an=4'b0111; end

default: digit = x[3:0];

endcase

always @ ( \* ) //段译码

case ( digit )

0:a\_to\_g=7'b1111110;

1:a\_to\_g=7'b0110000;

2:a\_to\_g=7'b1101101;

3:a\_to\_g=7'b1111001;

4:a\_to\_g=7'b0110011;

5:a\_to\_g=7'b1011011;

6:a\_to\_g=7'b1011111;

7:a\_to\_g=7'b1110000;

8:a\_to\_g=7'b1111111;

9:a\_to\_g=7'b1111011;

'hA:a\_to\_g=7'b1110111;

'hB:a\_to\_g=7'b0011111;

'hC:a\_to\_g=7'b1001110;

'hD:a\_to\_g=7'b0111101;

'hE:a\_to\_g=7'b1001111;

'hF:a\_to\_g=7'b1000111;

default:a\_to\_g=7'b1111110;

endcase

always @ ( posedge clk or posedge clr ) //使用计数器分频

begin

if ( clr == 0 )

clkdiv=0; //clkdiv清零；

else

clkdiv=clkdiv+1; //clkdiv+1;

end

endmodule

运动逻辑代码：

`timescale 1ns / 1ps

module move(

input clk, // 来自 clkdiv 的 clk\_1（慢速）

input clr, // 复位信号

input [1:0] ball\_start, // 起始控制：01左边发球，10右边发球，11中间，00继续运动

input [1:0] ball\_dir, // 方向控制：01向左，10向右，00停止

output reg [9:0] index // 控制乒乓球位置

);

always @(posedge clk or negedge clr) begin

if (!clr) begin

index <= 10'b1000000001; // 初始位置在中间

end

else begin

case (ball\_start)

2'b01: index <= 10'b1000000000; // 左侧发球，从最左边开始

2'b10: index <= 10'b0000000001; // 右侧发球，从最右边开始

2'b11: index <= 10'b1000000001; // 初始位置（中间）

2'b00: begin // 继续运动

case (ball\_dir)

2'b01: begin // 向左移动

if (index != 10'b1000000000) // 防止溢出

index <= index << 1;

end

2'b10: begin // 向右移动

if (index != 10'b0000000001) // 防止溢出

index <= index >> 1;

end

default: index <= index; // 暂停或保持

endcase

end

default: index <= index;

endcase

end

end

endmodule

游戏结束代码：

`timescale 1ns / 1ps

module gameover(

input clk,

input rst\_n,

input [3:0] score1,

input [3:0] score2,

output reg [3:0] win1,

output reg [3:0] win2,

output reg gameover

);

reg game\_end\_flag;

always @ (posedge clk or negedge rst\_n) begin

if (!rst\_n) begin

win1 <= 0;

win2 <= 0;

gameover <= 0;

game\_end\_flag <= 0;

end

else begin

// 检测单局结束（11分制）

if (score1 == 4'd11 && !game\_end\_flag) begin

win1 <= win1 + 1;

gameover <= 1;

game\_end\_flag <= 1;

end

else if (score2 == 4'd11 && !game\_end\_flag) begin

win2 <= win2 + 1;

gameover <= 1;

game\_end\_flag <= 1;

end

// 检测大局结束（7局4胜）

else if (win1 == 4 || win2 == 4) begin

win1 <= 0;

win2 <= 0;

gameover <= 0;

game\_end\_flag <= 0;

end

// 单局结束后重新开始

else if (gameover && (score1 == 0 && score2 == 0)) begin

gameover <= 0;

game\_end\_flag <= 0;

end

end

end

endmodule