

课程设计报告

**中文题目 ：** **舞蹈革命**

**英文题目：** **Dance Revolution**

姓名/学号： 张瑾/3140105196

张昌琳/3140103541

指导教师： 董亚波、蒋方炎、洪奇军

专业类别： 张瑾/计科

张昌琳/软工

所在学院： 计算机学院

**论文提交日期 2016 年 1 月 23 日**

摘要

本次课程设计利用 Xilinx Spartan-3 设计平台完成了舞蹈革命的设计，借鉴了一些之前玩过的游戏，同时也融入了自己的一些想法。其中也包括了LED灯，七段数码管的显示等内容。 玩家可以通过扩展板上的按钮配合不同节奏进行对应手指舞蹈，除了游戏的核心模块， 还加入了计时、选择速度、选择单双人游戏模式等模块，使游戏更具趣味性。该报告将详细阐述这个游戏的实现原理、实现方法以及游戏操作等内容。

**关键词：**　Spartan-3、FPGA、舞蹈革命

目录

[摘要 ii](#_Toc441358128)

[1 绪论 3](#_Toc441358129)

[1.1舞蹈革命设计背景 3](#_Toc441358130)

[1.2同类作品比较 3](#_Toc441358131)

[1.3主要内容和难点 3](#_Toc441358132)

[2 Dance Revolution 设计原理 4](#_Toc441358133)

[2.1 Dance Revolution 设计相关内容 4](#_Toc441358134)

[2.2Dance Revolution 设计方案 5](#_Toc441358135)

[2.3 Dance Revolution硬件设计 5](#_Toc441358136)

[3 Dance Revolution 设计实现 19](#_Toc441358137)

[3.1实现方法及过程 19](#_Toc441358138)

[3.2仿真与调试 30](#_Toc441358139)

[4系统测试验证与结果分析 33](#_Toc441358140)

[4.1功能测试和结果分析 33](#_Toc441358141)

[4.2技术参数测试 38](#_Toc441358142)

[4.3系统演示与操作说明 39](#_Toc441358143)

[5结论与展望 40](#_Toc441358144)

# 1 绪论

## 1.1舞蹈革命设计背景

我们的课程设计“舞蹈革命”的灵感来自于几年前风靡一时的网络游戏“QQ炫舞”，已经很成熟的产品跳舞机游戏模式也如此。这个跳舞机实现的功能类似于“QQ炫舞”里的节奏模式，游戏系统随机产生一个方向符号，玩家需要在一定时间内按下对应的方向控制键，是一个锻炼反应能力和手指灵敏度的游戏。

### 1.2同类作品比较

|  |  |
| --- | --- |
| 舞蹈革命 | 原始游戏“QQ炫舞”以及其他类似游戏 |
| 游戏结束以生命值是否耗尽为标准 | 游戏结束以时间为标准 |
| 单人挑战模式、双人对抗模式 | 单人挑战模式、双人合作模式 |
| 可以暂停 | 不可以暂停 |
| 可以变速 | 需要在游戏开始前设置好速度 |

## 1.3主要内容和难点

1.3.1主要内容

游戏的主要内容是根据七段数码管显示的箭头方向按下对应方向键。



（1）欢迎界面

游戏处于待机状态时，扩展板七段数码管上显示“HELLO”字样，欢迎玩家进入游戏。

（2）模式选择

将switch0置0，则为选择单人模式，将switch0置1，则为选择双人模式。

（3）游戏开始

按button1按钮后触发游戏开始。若为单人模式，扩展板左侧七段数码管开始显示随机产生的方向符号，扩展板左四个LED灯亮起，代表生命值为4。若为双人模式，扩展板左侧和右侧七段数码管分别开始显示随机产生的方向符号（两个玩家方向符号分别随机生成），扩展板左侧和右侧四个LED灯亮起，分别代表一号玩家生命值为4和二号玩家生命值为4。

（4）游戏过程

扩展板上七段数码管显示符号根据当前速度档位在一定时间后显示下一个符号，玩家必须在变化的时间内按下七段数码管下方对应方向控制的button，按错或不按则生命值减一，即扩展板上LED灯熄灭一盏（双人模式下，LED灯熄灭顺序为从内向外顺次熄灭）。

（5）档位选择

档位一共有四个等级，默认值为最低速度档，将switch3置1，按一次button0则速度档位提升一个等级，将switch3置0，按一次button0则速度档位降低一个等级。当前速度等级显示于Spartan-III开发板的七段数码管上。

（6）游戏暂停

按button1按钮后游戏暂停。

（7）游戏结束

当单人模式生命值耗尽或双人模式下任一玩家生命值耗尽时，游戏结束。

1.3.2难点

以下为完成该游戏过程中遇到的难点，解决方法分析见报告2、3部分的技术实现部分。

（1）随机符号的生成

（2）按键错误检查。

（3）速度变化功能。

（4）暂停与重启的实现。

# 2 Dance Revolution 设计原理

## 2.1 Dance Revolution 设计相关内容

我们小组设计的Dance Revolution这个游戏的设计灵感来源一一款音乐舞蹈游戏《QQ炫舞》。此游戏的基本操作方式如下：界面中有一个判定框，节奏音符以“上下左右”的方式以固定的时间间隔出现，并以一定速度自下而上地滚动进判定框。玩家需正确选择滚动进判定框内的音符，并跟着音乐节奏选择对应的上下左右键。在规定时间点，正确按下滚动进判定框内的符号即可。如按错或未跟上节奏在规定时间点按下对应键，会减少生命值。当生命值减为零，游戏结束。我们小组此次在spartan－3板上设计的这款小游戏，即是参考了《QQ炫舞》的游戏方式，由于板上显示界面和操作按钮有限，我们设计的游戏只实现了其基本功能。但我们所设计出的游戏功能多，可玩性强，易于操作，充分体现了课程设计的理念和要求。

## 2.2Dance Revolution 设计方案

（1） 打歌面板为扩展板上的八个数码管。每个周期数码管上显示一个符号（上／下／左／右）作为给定音符，玩家通过按动显示屏下方的上／下／左／右键来击打音符进行游戏。

（2）未在要求的时间点击中音符（即未按中音符对应的按钮）判定失误，生民值减一。当血槽为空，即led灯全灭时，游戏结束。

（3）游戏分为单人模式和双人模式，双人模式下一方失败则游戏结束，另一方获胜。

（4）增加调速按钮，玩家可在游戏中更改音符出现的速度，以提升或降低游戏难度系数。

## 2.3 Dance Revolution硬件设计

2.3.1 Top 顶层模块

|  |
| --- |
| module top(input wire clk, input wire [1:0]btn\_in, input wire [3:0]switch, input wire [7:0]btn\_ctr\_in,  output wire [11:0]anode, output wire [15:0]segment, output reg [3:0]Y1, output reg [3:0]Y2);  reg [15:0] digt; //show on spartan-3  reg [31:0] digtex; //show on expand board  reg [1:0] speed; //control speed  reg [1:0] cnt; //used in time counter  reg [3:0] gameinter; // player’s game interface state  reg status; //control system stoping or running for status = 0 or 1  reg clk\_gamespeed; //control time interval in different speed status  reg [7:0] flag; //record witch button is pressed  reg [15:0] errorpress\_1; //record the number of errors player1 make  reg [15:0] errorpress\_2; //record the number of errors player2 make  wire [1:0] btn\_out; //buttons that change game’s setting  wire [7:0] btn\_ctr\_out; //buttons that players use to play the game  wire clk\_interval; //time interval that the counter produce  //Initial status///////////////////////////////////////////////  /////////////////////////////////////////////////////////////  initial begin  speed=0;  gameinter=0;  errorpress\_1=0;  errorpress\_2=0;  clk\_gamespeed=0;  status=0;  end  //buttons pbdebounce//////////////////////////////////////////  //////////////////////////////////////////////////////////////  pbdebounce p0(clk,btn\_in[0],btn\_out[0]);  pbdebounce p1(clk,btn\_in[1],btn\_out[1]);  pbdebounce p2(clk,btn\_ctr\_in[0],btn\_ctr\_out[0]);  pbdebounce p3(clk,btn\_ctr\_in[1],btn\_ctr\_out[1]);  pbdebounce p4(clk,btn\_ctr\_in[2],btn\_ctr\_out[2]);  pbdebounce p5(clk,btn\_ctr\_in[3],btn\_ctr\_out[3]);  pbdebounce p6(clk,btn\_ctr\_in[4],btn\_ctr\_out[4]);  pbdebounce p7(clk,btn\_ctr\_in[5],btn\_ctr\_out[5]);  pbdebounce p8(clk,btn\_ctr\_in[6],btn\_ctr\_out[6]);  pbdebounce p9(clk,btn\_ctr\_in[7],btn\_ctr\_out[7]);  //display patterns////////////////////////////////////////////  ////////////////////////////////////////////////////////////  display m0(clk, digt, anode[3:0], segment[7:0]);  display32bits m1(clk, digtex, anode[11:4], segment[15:8]);  //counter module////////////////////////////////////////////  ////////////////////////////////////////////////////////////  counter\_interval m2(clk, clk\_interval);  //player’s control ////////////////////////////////////////////  ////////////////////////////////////////////////////////////  always @(posedge btn\_out[0]) //when btn\_out[0] is pressed  begin  if(switch[3]==1) speed<=speed+1; //speed up  else speed<=speed-1; //speed down  end  always @(posedge btn\_out[1])  status<=~status; //stop or run  always @(posedge clk\_interval) //change game speed  begin  if(cnt<3-speed)  cnt<=cnt+1;  else  begin  clk\_gamespeed<=~clk\_gamespeed;  cnt<=0;  end  end  always @\* begin  if(switch[0] == 0) begin //one player  Y2<=8'b0000;  case(errorpress\_1) //numbers of errors determin numbers of LEDs on  0: Y1<=8'b1111;  1: Y1<=8'b1110;  2: Y1<=8'b1100;  3: Y1<=8'b1000;  4: Y1<=8’b0000; //4 permitted error times  endcase  end  else if(switch[0] == 1) begin //two player  case(errorpress\_1)  0: Y1<=8'b1111;  1: Y1<=8'b1110;  2: Y1<=8'b1100;  3: Y1<=8'b1000;  4: Y1<=8'b0000;  endcase  case(errorpress\_2)  0: Y2<=8'b1111;  1: Y2<=8'b0111;  2: Y2<=8'b0011;  3: Y2<=8'b0001;  4: Y2<=8'b0000;  endcase  end  end  always @(posedge clk\_gamespeed) begin  if(switch[0] == 0)begin//one player  if(errorpress\_1 != 4 && status)  begin  case(gameinter) //15 given status of interface  0:  begin  if(btn\_ctr\_out[7] == 0)//if the right button isn’t pressed  errorpress\_1 <= errorpress\_1+1;  //increase error time by 1  end  1:  begin  if(btn\_ctr\_out[5] == 0)  errorpress\_1 <= errorpress\_1+1;  end  2:  begin  if(btn\_ctr\_out[4] == 0)  errorpress\_1 <= errorpress\_1+1;  end  3:  begin  if(btn\_ctr\_out[4] == 0)  errorpress\_1 <= errorpress\_1+1;  end  4:  begin  if(btn\_ctr\_out[5] == 0)  errorpress\_1 <= errorpress\_1+1;  end  5:  begin  if(btn\_ctr\_out[6] == 0)  errorpress\_1 <= errorpress\_1+1;  end  6:  begin  if(btn\_ctr\_out[7] == 0)  errorpress\_1 <= errorpress\_1+1;  end  7:  begin  if(btn\_ctr\_out[5] == 0)  errorpress\_1 <= errorpress\_1+1;  end  8:  begin  if(btn\_ctr\_out[4] == 0)  errorpress\_1 <= errorpress\_1+1;  end  9:  begin  if(btn\_ctr\_out[6] == 0)  errorpress\_1 <= errorpress\_1+1;  end  10:  begin  if(btn\_ctr\_out[4] == 0)  errorpress\_1 <= errorpress\_1+1;  end  11:  begin  if(btn\_ctr\_out[7] == 0)  errorpress\_1 <= errorpress\_1+1;  end  12:  begin  if(btn\_ctr\_out[6] == 0)  errorpress\_1 <= errorpress\_1+1;  end  13:  begin  if(btn\_ctr\_out[5] == 0)  errorpress\_1 <= errorpress\_1+1;  end  14:  begin  if(btn\_ctr\_out[7] == 0)  errorpress\_1 <= errorpress\_1+1;  end  15:  begin  if(btn\_ctr\_out[6] == 0)  errorpress\_1 <= errorpress\_1+1;  end  endcase  end  end  else if(switch[0] == 1)begin  if(errorpress\_1 != 4 && errorpress\_2 != 4 && status)begin  case(gameinter)  0:  begin  if(btn\_ctr\_out[7] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[1] == 0)  errorpress\_2 <= errorpress\_2+1;  end  1:  begin  if(btn\_ctr\_out[5] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[3] == 0)  errorpress\_2 <= errorpress\_2+1;  end  2:  begin  if(btn\_ctr\_out[4] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[2] == 0)  errorpress\_2 <= errorpress\_2+1;  end  3:  begin  if(btn\_ctr\_out[4] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[0] == 0)  errorpress\_2 <= errorpress\_2+1;  end  4:  begin  if(btn\_ctr\_out[5] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[2] == 0)  errorpress\_2 <= errorpress\_2+1;  end  5:  begin  if(btn\_ctr\_out[6] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[0] == 0)  errorpress\_2 <= errorpress\_2+1;  end  6:  begin  if(btn\_ctr\_out[7] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[3] == 0)  errorpress\_2 <= errorpress\_2+1;  end  7:  begin  if(btn\_ctr\_out[5] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[3] == 0)  errorpress\_2 <= errorpress\_2+1;  end  8:  begin  if(btn\_ctr\_out[4] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[1] == 0)  errorpress\_2 <= errorpress\_2+1;  end  9:  begin  if(btn\_ctr\_out[6] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[0] == 0)  errorpress\_2 <= errorpress\_2+1;  end  10:  begin  if(btn\_ctr\_out[4] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[2] == 0)  errorpress\_2 <= errorpress\_2+1;  end  11:  begin  if(btn\_ctr\_out[7] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[1] == 0)  errorpress\_2 <= errorpress\_2+1;  end  12:  begin  if(btn\_ctr\_out[6] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[3] == 0)  errorpress\_2 <= errorpress\_2+1;  end  13:  begin  if(btn\_ctr\_out[5] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[1] == 0)  errorpress\_2 <= errorpress\_2+1;  end  14:  begin  if(btn\_ctr\_out[7] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[0] == 0)  errorpress\_2 <= errorpress\_2+1;  end  15:  begin  if(btn\_ctr\_out[6] == 0)  errorpress\_1 <= errorpress\_1+1;  if(btn\_ctr\_out[3] == 0)  errorpress\_2 <= errorpress\_2+1;  end  endcase  end  end  gameinter<=gameinter+1; // change into next interface status  end  //show patterns///////////////////////////////////////////  //////////////////////////////////////////////////////////  always @\* begin  if(switch[0] == 0)begin //one players module  if(errorpress\_1 == 4)begin  digtex<=32'he8b9\_ffff;//show "over"  digt<=16'h5bc6;//show "stop"  end  else if(~status)begin  digt<=16'h5bc6;//show "stop"  digtex<=32'hcbdd\_efff;//show "hello"  end  else begin  digt<=16'h5670+speed+1; //show speed  case(gameinter) //set patterns show in 15 status  0:digtex<=32'hf01f\_ffff; //up  1:digtex<=32'h4566\_ffff; //left  2:digtex<=32'h6657\_ffff; //right  3:digtex<=32'h6657\_ffff; //right  4:digtex<=32'h4566\_ffff; //left  5:digtex<=32'hf23f\_ffff; //down  6:digtex<=32'hf01f\_ffff; //up  7:digtex<=32'h4566\_ffff; //left  8:digtex<=32'h6657\_ffff; //right  9:digtex<=32'hf23f\_ffff; //down  10:digtex<=32'h6657\_ffff; //right  11:digtex<=32'hf01f\_ffff; //up  12:digtex<=32'hf23f\_ffff; //down  13:digtex<=32'h4566\_ffff; //left  14:digtex<=32'hf01f\_ffff; //up  15:digtex<=32'hf23f\_ffff; //down  endcase  end  end    else if(switch[0] == 1)begin //two players module  if(errorpress\_1 == 4 || errorpress\_2 == 4)begin  digtex<=32'he8b9\_ffff;//show "over"  digt<=16'h5bc6;//show "stop"  end  else if(~status)begin  digt<=16'h5bc6;//show "stop"  digtex<=32'hcbdd\_efff;//show "hello"  end  else begin  digt<=16'h5670+speed+1; //show speed  case(gameinter)  0:digtex<=32'hf01f\_4566; //up left  1:digtex<=32'h4566\_f01f; //left up  2:digtex<=32'h6657\_f23f; //right down  3:digtex<=32'h6657\_6657; //right right  4:digtex<=32'h4566\_f23f; //left down  5:digtex<=32'hf23f\_6657; //down right  6:digtex<=32'hf01f\_f01f; //up up  7:digtex<=32'h4566\_f01f; //left up  8:digtex<=32'h6657\_4566; //right left  9:digtex<=32'hf23f\_6657; //down right  10:digtex<=32'h6657\_f23f; //right down  11:digtex<=32'hf01f\_4566; //up left  12:digtex<=32'hf23f\_f01f; //down up  13:digtex<=32'h4566\_4566; //left left  14:digtex<=32'hf01f\_6657; //up right  15:digtex<=32'hf23f\_f01f; //down up  endcase  end  end  end  endmodule |

////////////////////////////////////////////////////////////

2.3.2 pbdebounce 按键去抖模块

|  |
| --- |
| module pbdebounce  (input wire clk,  input wire button,  output reg pbreg);    reg [7:0] pbshift;  wire clk\_1ms;  timer\_1ms m0(clk, clk\_1ms);  always@(posedge clk\_1ms) begin  pbshift=pbshift<<1;  pbshift[0]=button;  if (pbshift==0)  pbreg=0;  if (pbshift==8'hFF)  pbreg=1;  end  endmodule |

////////////////////////////////////////////////////////////

2.3.3 timer\_1ms 1ms计时模块（作为按键去抖动的触发时钟）

|  |
| --- |
| module timer\_1ms  (input wire clk,  output reg clk\_1ms);    reg [15:0] cnt;  initial begin  cnt [15:0] <=0;  clk\_1ms <= 0;  end  always@(posedge clk)  if(cnt>=25000) begin  cnt<=0;  clk\_1ms <= ~clk\_1ms;  end  else begin  cnt<=cnt+1;  end  endmodule |

////////////////////////////////////////////////////////////

2.3.4 display spartan－3上数码管的显示模块

|  |
| --- |
| module display(  input wire clk,  input wire [15:0] digit,// digits shows  output reg [ 3:0] node, //four digital pipes choice  output reg [ 7:0] segment //eight digital tube display  );  reg [3:0] code = 4'b0;  reg [15:0] count = 15'b0;  always @(posedge clk) begin  case (count[15:14])  2'b00 : begin  node <= 4'b1110;  code <= digit[3:0];  end  2'b01 : begin  node <= 4'b1101;  code <= digit[7:4];  end  2'b10 : begin  node <= 4'b1011;  code <= digit[11:8];  end  2'b11 : begin  node <= 4'b0111;  code <= digit[15:12];  end  endcase    case (code)  4'b0000: segment <= 8'b11000000;//0  4'b0001: segment <= 8’b11111001; //1  4'b0010: segment <= 8'b10100100;//2  4'b0011: segment <= 8'b10110000;//3  4'b0100: segment <= 8'b10011001;//4  4'b0101: segment <= 8'b10010010;//s  4'b0110: segment <= 8'b10001100;//p  4'b0111: segment <= 8'b00100001;//d.  4'b1000: segment <= 8'b10101111;//r  4'b1001: segment <= 8'b11100011;//u  4'b1010: segment <= 8'b10101011;//n  4'b1011: segment <= 8'b10000111;//t  4'b1100: segment <= 8'b10100011;//o  4'b1101: segment <= 8'b10100001;  4'b1110: segment <= 8'b10000110;  4'b1111: segment <= 8'b11111111;//none  default: segment <= 8'b11111111;  endcase  count <= count + 1;  end  endmodule |

////////////////////////////////////////////////////////////

2.3.5 display32bits spartan－3扩展板上数码管显示模块

|  |
| --- |
| module display32bits(clk, disp\_num, digit\_anode, segment);  input clk;  input [31:0] disp\_num;  output [7:0] digit\_anode;  output [7:0] segment;    reg [7:0] digit\_anode;  reg [7:0] segment;  reg [12:0] cnt=0;  wire [31:0] disp\_num;  reg [3:0] num;    always@(posedge clk)begin  case(cnt[12:10])  3'b000:begin  digit\_anode <= 8'b11111110;  num <= disp\_num[3:0];  end  3'b001:begin  digit\_anode <= 8'b11111101;  num <= disp\_num[7:4];  end  3'b010:begin  digit\_anode <= 8'b11111011;  num <= disp\_num[11:8];  end  3'b011:begin  digit\_anode <= 8'b11110111;  num <= disp\_num[15:12];  end  3'b100:begin  digit\_anode <= 8'b11101111;  num <= disp\_num[19:16];  end  3'b101:begin  digit\_anode <= 8'b11011111;  num <= disp\_num[23:20];  end  3'b110:begin  digit\_anode <= 8'b10111111;  num <= disp\_num[27:24];  end  3'b111:begin  digit\_anode <= 8'b01111111;  num <= disp\_num[31:28];  end  endcase    case(num)  4'b0000:segment<=8'b11011000;//7  4'b0001:segment<=8'b11001100;//反7  4'b0010:segment<=8'b11100001;//倒7  4'b0011:segment<=8'b11000011;//倒反7  4'b0100:segment<=8'b10000110;//E  4'b0101:segment<=8'b10110110;//三  4'b0110:segment<=8'b10111111;//一  4'b0111:segment<=8'b10110000;//反E  4'b1000:segment<=8'b11100011;//v  4'b1001:segment<=8'b10101111;//r  4'b1010:segment<=8'b10111111;  4'b1011:segment<=8'b10000110;//e  4'b1100:segment<=8'b10001001;//h  4'b1101:segment<=8'b11000111;//l  4'b1110:segment<=8'b11000000;//o  4'b1111:segment<=8'b11111111;//show nothing  endcase  end  always@(posedge clk) begin  cnt<=cnt+1;  end  endmodule |

////////////////////////////////////////////////////////////

2.3.6 counter\_interval 计时器模块

|  |
| --- |
| module counter\_interval(clk, clk\_seconds); //produce time interval  input wire clk;  output reg clk\_seconds;  reg[31:0] cnt=0;  always@(posedge clk) begin  if(cnt<8000000) //set time interval  begin  cnt<=cnt+1;  end    else  begin  cnt<=0;  clk\_seconds<=~clk\_seconds;  end  end  endmodule |

# 3 Dance Revolution 设计实现

## 3.1实现方法及过程

（内容要点：实现的具体思路和方法、重点难点在实现中的经验和方法等）

**1.打歌面板及符号显示**

spartan－3板及其扩展板上共有4+8=12个七段数码管。扩展板上的八个数码管用来显示符号（即代替音符），左边四个为player1的显示面板，右边四个为player2的显示面板。当游戏选择单人模式时，player2的面板不作任何显示。

要显示符号，可以使用display32bits模块来实现。模块及参数定义如下：

module display32bits(clk, disp\_num, digit\_anode, segment);

input clk;

input [31:0] disp\_num;

output [7:0] digit\_anode;

output [7:0] segment;

reg [7:0] digit\_anode; //八个数码管的位选

reg [7:0] segment; //控制七段数码管每段的显示

reg [12:0] cnt=0; //用来计时分频的变量

wire [31:0] disp\_num; //用32位二进制数值表示的待显示符号

reg [3:0] num; //数码管的显示状态

游戏中共有上键（↑），下键（↓），左键（←），右键（→）四种符号。在数码管上显示为下图。

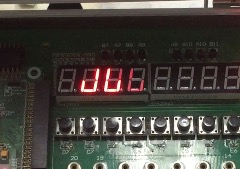


图1.下键

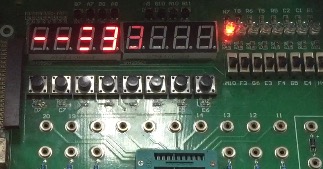


图2.右键

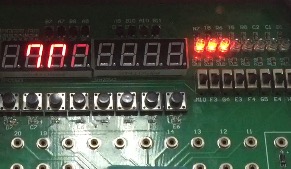


图3.上键

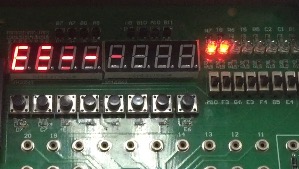


图4.左键

根据七段数码管的显示原理，要显示如图1-图4的四种符号，可写出每个数码管的赋值如下：

case(num)

4'b0000:segment<=8'b11011000;//7

4'b0001:segment<=8'b11001100;//反7

4'b0010:segment<=8'b11100001;//倒7

4'b0011:segment<=8'b11000011;//倒反7

4'b0100:segment<=8'b10000110;//E

4'b0101:segment<=8'b10110110;//三

4'b0110:segment<=8'b10111111;//一

4’b0111:segment<=8’b10110000;//反E

……

据此，四个符号可表示如下：

上键：16’hf01f

下键：16’hf23f

左键：16’h4566

右键：16’h6657

用digtex记录上述每个键对应的16进制数值（共八个数码管），作为display32bits的传入参数disp\_num。调用display32bits模块，即可实现符号的显示。

我们定义了reg [15:0] gameinter这个变量来控制符号的显示和变化。共设置了16个状态，gameinter在0000～1111 共16个值之间变化，每个值代表一种面板上的符号状态，作为伪随机状态。gameinter赋值及对应状态设置如下（以双人模式为例）：

case(gameinter)

0:digtex<=32'hf01f\_4566; //up left

1:digtex<=32'h4566\_f01f; //left up

2:digtex<=32'h6657\_f23f; //right down

3:digtex<=32'h6657\_6657; //right right

4:digtex<=32'h4566\_f23f; //left down

5:digtex<=32'hf23f\_6657; //down right

6:digtex<=32'hf01f\_f01f; //up up

7:digtex<=32'h4566\_f01f; //left up

8:digtex<=32'h6657\_4566; //right left

9:digtex<=32'hf23f\_6657; //down right

10:digtex<=32'h6657\_f23f; //right down

11:digtex<=32'hf01f\_4566; //up left

12:digtex<=32'hf23f\_f01f; //down up

13:digtex<=32'h4566\_4566; //left left

14:digtex<=32'hf01f\_6657; //up right

15:digtex<=32'hf23f\_f01f; //down up

endcase

在每个clk\_gamespeed上升沿（clk\_gamespeed产生每个上升沿的时间间隔为面板每个状态发生变化的时间间隔），gameinter<=gameinter+1，这样就实现了面板上符号状态的转换。

**2.玩家操作和失误判定**

玩家通过扩展板上八个数码管下方的八个（单人模式下为四个）按钮来进行游戏，左右的四个按钮分别为玩家1和玩家2的上／下／左／右控制按钮。数码管上显示的符号每隔一定的时间间隔发生变化，玩家需按下与此符号对应的按钮。如面板上显示“↑”，玩家1就必须按下第一个按钮，以此类推。

注意：玩家必须在下一个状态上升沿发生的瞬间按下对应按钮，才算击中符号，否则判定失误，失误次数加一。（即相当于炫舞游戏中玩家必须在判定框内击中音符）但考虑到玩家无法预知下一个上升沿到来的准确时间，可允许玩家在本次状态的时间间隔内一直按着按钮，以保证在下一个时钟周期上升沿到来的时候按钮处于按下状态。

为实现此比对功能，我们为打歌面板上可能出现的16个状态分别写了比对代码如下（以双人模式为例）：

if(errorpress\_1 != 4 && errorpress\_2 != 4 && status)begin

//错误在允许次数内且游戏处于运行状态

case(gameinter)

0:

begin

if(btn\_ctr\_out[7] == 0) //如果对应的正确按钮未处于按下状态，则认定失误一次

errorpress\_1 <= errorpress\_1+1; //失误次数加一

if(btn\_ctr\_out[1] == 0)

errorpress\_2 <= errorpress\_2+1;

end

1:

begin

if(btn\_ctr\_out[5] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[3] == 0)

errorpress\_2 <= errorpress\_2+1;

end

2:

begin

if(btn\_ctr\_out[4] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[2] == 0)

errorpress\_2 <= errorpress\_2+1;

end

3:

begin

if(btn\_ctr\_out[4] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[0] == 0)

errorpress\_2 <= errorpress\_2+1;

end

4:

begin

if(btn\_ctr\_out[5] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[2] == 0)

errorpress\_2 <= errorpress\_2+1;

end

5:

begin

if(btn\_ctr\_out[6] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[0] == 0)

errorpress\_2 <= errorpress\_2+1;

end

6:

begin

if(btn\_ctr\_out[7] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[3] == 0)

errorpress\_2 <= errorpress\_2+1;

end

7:

begin

if(btn\_ctr\_out[5] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[3] == 0)

errorpress\_2 <= errorpress\_2+1;

end

8:

begin

if(btn\_ctr\_out[4] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[1] == 0)

errorpress\_2 <= errorpress\_2+1;

end

9:

begin

if(btn\_ctr\_out[6] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[0] == 0)

errorpress\_2 <= errorpress\_2+1;

end

10:

begin

if(btn\_ctr\_out[4] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[2] == 0)

errorpress\_2 <= errorpress\_2+1;

end

11:

begin

if(btn\_ctr\_out[7] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[1] == 0)

errorpress\_2 <= errorpress\_2+1;

end

12:

begin

if(btn\_ctr\_out[6] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[3] == 0)

errorpress\_2 <= errorpress\_2+1;

end

13:

begin

if(btn\_ctr\_out[5] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[1] == 0)

errorpress\_2 <= errorpress\_2+1;

end

14:

begin

if(btn\_ctr\_out[7] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[0] == 0)

errorpress\_2 <= errorpress\_2+1;

end

15:

begin

if(btn\_ctr\_out[6] == 0)

errorpress\_1 <= errorpress\_1+1;

if(btn\_ctr\_out[3] == 0)

errorpress\_2 <= errorpress\_2+1;

end

endcase

end

失误反馈：我们使用了扩展板上的八个led灯作为生命值的反映，每个玩家共有四条生命。（单人模式下血槽为左四个，双人模式下玩家1血槽为左四个，玩家2血槽为右四个）以errorpress\_1和errorpress\_2为失误次数的反馈，玩家每失误一次，errorpress\_1（玩家1）或errorpress\_2（玩家2）增加一。 双人模式下两人的游戏进行相互独立。通过errorpress\_x的值来控制led灯亮灭的个数，errorpress\_x每加一，灯灭一盏。当犯错四次即errorpress\_x从0增加到4的时候，游戏结束。双人模式下，只要有一人生命结束，则另一个玩家获胜，游戏结束。对血槽（led灯）的控制实现如下（以双人模式为例，[7:0]Y1/[7:0]Y2的引脚设置为八个led灯）：

case(errorpress\_1)

0: Y1<=8'b1111;

1: Y1<=8'b1110;

2: Y1<=8'b1100;

3: Y1<=8'b1000;

4: Y1<=8'b0000;

endcase

case(errorpress\_2)

0: Y2<=8'b1111;

1: Y2<=8'b0111;

2: Y2<=8'b0011;

3: Y2<=8'b0001;

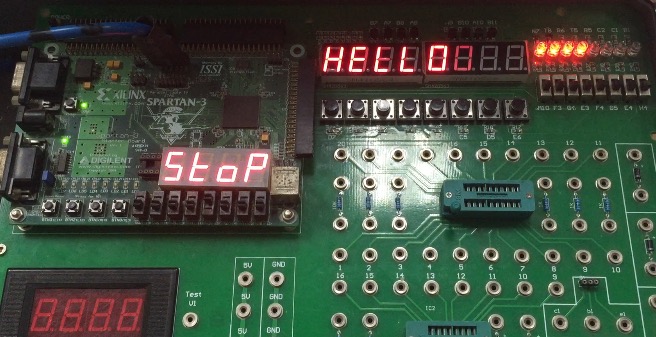
4: Y2<=8'b0000;

endcase

**3.装饰性界面**

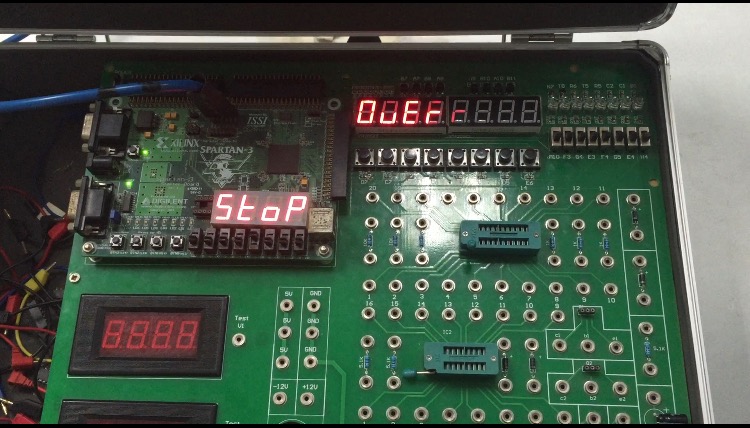
欢迎界面／暂停界面／结束界面：

游戏开始时显示欢迎界面如下图：



按btn[1]键游戏开始。游戏进行中按btn[1]键游戏暂停，再按一下游戏继续。暂停界面与开始界面相同。

游戏结束时显示结束界面如下图：



这些显示可通过display和display32bits两个模块实现。

**4.速度切换**

将switch[3]打到高电平可对游戏进行正向调速，switch[3]打到低电平可对游戏进行反向调速。游戏过程中，按下btn[0]键即提升／降低一个速度级别并将当前速度等级显示在spartan－3板的数码管上。共四个级别。实现方法如下：

module counter\_interval(clk, clk\_seconds) 模块作为计时器，产生的clk\_interval每隔一定时间发生变化，作为新的时钟脉冲。以clk\_interval代替clk为新的时钟脉冲，我们写了一个可调速的新的计时器。实现如下：

always @(posedge clk\_interval) begin

if(cnt<3-speed)

cnt<=cnt+1;

else

begin

clk\_gamespeed<=~clk\_gamespeed;

cnt<=0;

end

end

speed为0，1，2，3时，clk\_gamespeed变化的速度不同。speed＝1时为最低速，speed＝3时为最高速。调速实现如下：

always @(posedge btn\_out[0]) //when btn\_out[0] is pressed

begin

if(switch[3]==1) speed<=speed+1; //speed up

else speed<=speed-1; //speed down

end

**5. 引脚分配**

NET "clk" LOC= "T9";

NET "anode[0]" LOC="D14";

NET "anode[1]" LOC="G14";

NET "anode[2]" LOC="F14";

NET "anode[3]" LOC="E13";

NET "anode[4]" LOC = "B11";

NET "anode[5]" LOC = "A10";

NET "anode[6]" LOC = "B10";

NET "anode[7]" LOC = "A9";

NET "anode[8]" LOC = "A8";

NET "anode[9]" LOC = "B8";

NET "anode[10]" LOC = "A7";

NET "anode[11]" LOC = "B7";

NET "segment[0]" LOC="E14";

NET "segment[1]" LOC="G13";

NET "segment[2]" LOC="N15";

NET "segment[3]" LOC="P15";

NET "segment[4]" LOC="R16";

NET "segment[5]" LOC="F13";

NET "segment[6]" LOC="N16";

NET "segment[7]" LOC="P16";

NET "segment[8]" LOC = "C8";

NET "segment[9]" LOC = "D8";

NET "segment[10]" LOC = "C9";

NET "segment[11]" LOC = "D10";

NET "segment[12]" LOC = "A3";

NET "segment[13]" LOC = "B4";

NET "segment[14]" LOC = "A4";

NET "segment[15]" LOC = "B5";

NET "btn\_in[0]" loc="M13";

NET "btn\_in[1]" loc="M14";

NET "btn\_ctr\_in[0]" loc="E6";

NET "btn\_ctr\_in[1]" loc="D5";

NET "btn\_ctr\_in[2]" loc="C5";

NET "btn\_ctr\_in[3]" loc="D6";

NET "btn\_ctr\_in[4]" loc="C6";

NET "btn\_ctr\_in[5]" loc="E7";

NET "btn\_ctr\_in[6]" loc="C7";

NET "btn\_ctr\_in[7]" loc="D7";

NET "switch[0]" LOC="F12";

NET "switch[1]" LOC="G12";

NET "switch[2]" LOC="H14";

NET "switch[3]" LOC="H13";

NET "Y2[0]" LOC="B1";

NET "Y2[1]" LOC="C1";

NET "Y2[2]" LOC="C2";

NET "Y2[3]" LOC="R5";

NET "Y1[0]" LOC="T5";

NET "Y1[1]" LOC="R6";

NET "Y1[2]" LOC="T8";

NET "Y1[3]" LOC="N7";

## 3.2仿真与调试

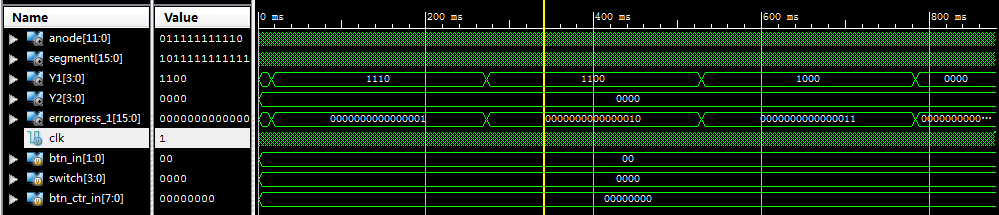
（1）单人模式仿真

仿真输入： switch0置为0（选择单人游戏），btn1置为1后变为0（开始游戏），btn\_ctr\_in全为0且没有输入。

仿真输入代码如下：

|  |
| --- |
| module top\_test;  // Inputs  reg clk;  reg [1:0] btn\_in;  reg [3:0] switch;  reg [7:0] btn\_ctr\_in;  // Outputs  wire [11:0] anode;  wire [15:0] segment;  wire [3:0] Y1;  wire [3:0] Y2;  wire [15:0]errorpress\_1;  wire clk\_gamespeed;  wire [3:0] gameinter;  // Instantiate the Unit Under Test (UUT)  top uut (  .clk(clk),  .btn\_in(btn\_in),  .switch(switch),  .btn\_ctr\_in(btn\_ctr\_in),  .anode(anode),  .segment(segment),  .Y1(Y1),  .Y2(Y2),  .errorpress\_1(errorpress\_1),  .clk\_gamespeed(clk\_gamespeed),  .gameinter(gameinter)  );  initial begin  // Initialize Inputs  clk = 0;  btn\_in = 0;  switch = 0;  btn\_ctr\_in = 0;  // Wait 100 ns for global reset to finish  #100;    // Add stimulus here  switch[1]=0;  btn\_in[1]=1;  #100;  btn\_in[1]=0;    end  initial begin  forever #1 clk=~clk;  end  endmodule |

仿真输出：anode和segment为随机数。Y1初始值为1111，一个游戏间隔后依次变为1110、1100、1000、0000。（即依次失去四次生命）



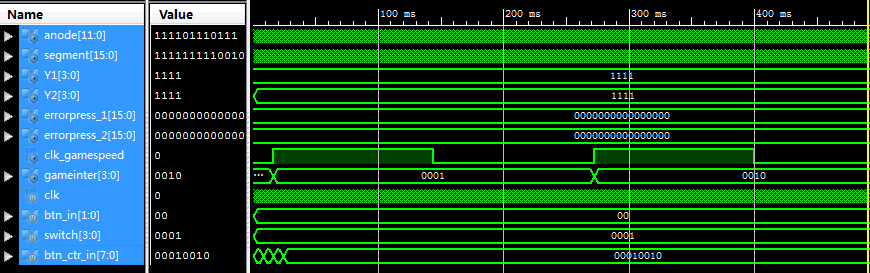
（2）双人模式仿真

仿真输入： switch0置为1（选择双人人游戏），btn1置为1后变为0（开始游戏），btn\_ctr\_in输入如代码所示。

仿真输入代码如下：

|  |
| --- |
| module top\_test;  // Inputs  reg clk;  reg [1:0] btn\_in;  reg [3:0] switch;  reg [7:0] btn\_ctr\_in;  // Outputs  wire [11:0] anode;  wire [15:0] segment;  wire [3:0] Y1;  wire [3:0] Y2;  wire [15:0]errorpress\_1;  wire clk\_gamespeed;  wire [3:0] gameinter;  // Instantiate the Unit Under Test (UUT)  top uut (  .clk(clk),  .btn\_in(btn\_in),  .switch(switch),  .btn\_ctr\_in(btn\_ctr\_in),  .anode(anode),  .segment(segment),  .Y1(Y1),  .Y2(Y2),  .errorpress\_1(errorpress\_1),  .clk\_gamespeed(clk\_gamespeed),  .gameinter(gameinter)  );  initial begin  // Initialize Inputs  clk = 0;  btn\_in = 0;  switch = 0;  btn\_ctr\_in = 0;  // Wait 100 ns for global reset to finish  #100;    // Add stimulus here  switch[0]=1;  btn\_in[1]=1;  #100;  btn\_in[1]=0;  btn\_ctr\_in[7]=1;  btn\_ctr\_in[2]=1;  #8000000;  btn\_ctr\_in[7]=0;  btn\_ctr\_in[5]=1;  btn\_ctr\_in[2]=0;  btn\_ctr\_in[3]=1;  #8000000;  btn\_ctr\_in[5]=0;  btn\_ctr\_in[6]=1;  btn\_ctr\_in[3]=0;  btn\_ctr\_in[0]=1;  #8000000;  btn\_ctr\_in[6]=0;  btn\_ctr\_in[4]=1;  btn\_ctr\_in[0]=0;  btn\_ctr\_in[1]=1;    end  initial begin  forever #1 clk=~clk;  end  endmodule |

仿真输出：anode和segment为随机数。输入正确，Y1，Y2不变。



# 4系统测试验证与结果分析

## 4.1功能测试和结果分析

（1）欢迎界面

游戏代码下载到板子后，扩展板七段数码管正确显示欢迎界面，LED灯正确显示初始生命值。

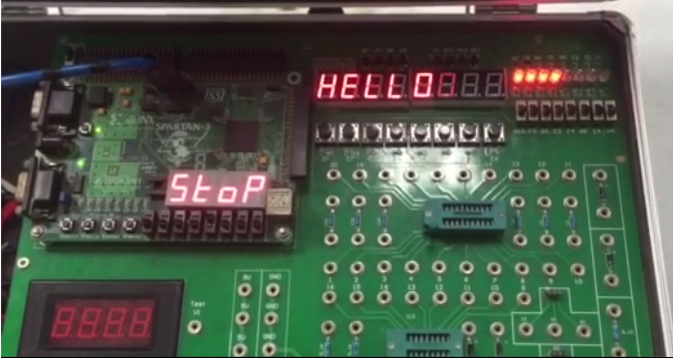


图1 单人模式欢迎界面

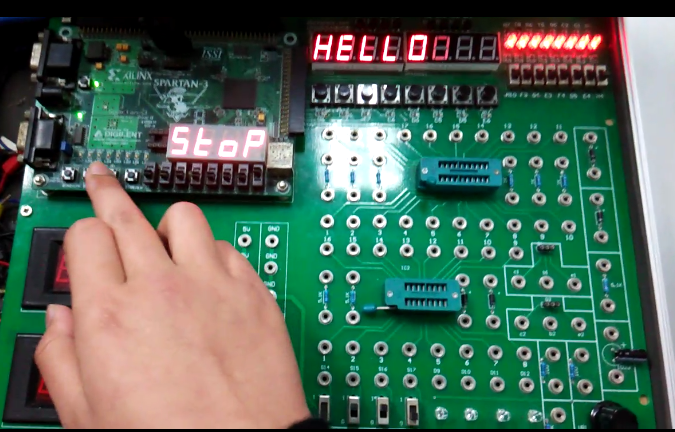


图2 双人模式欢迎界面

（2）模式选择

将switch0置0，正确显示单人模式，将switch0置1，正确显示双人模式。

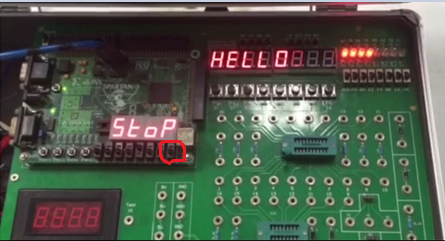


图3 switch0置0选择单人模式

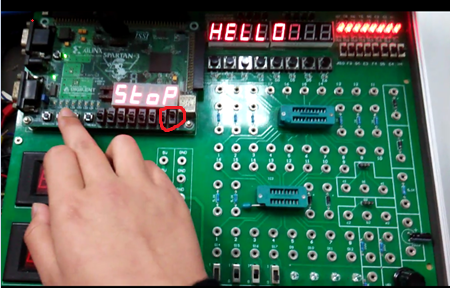


图4 switch0置1选择双人模式

（3）游戏开始

按button1按钮后游戏正确开始。扩展板左侧七段数码管正确显示随机产生的方向符号，扩展板LED灯正确显示当前生命值。

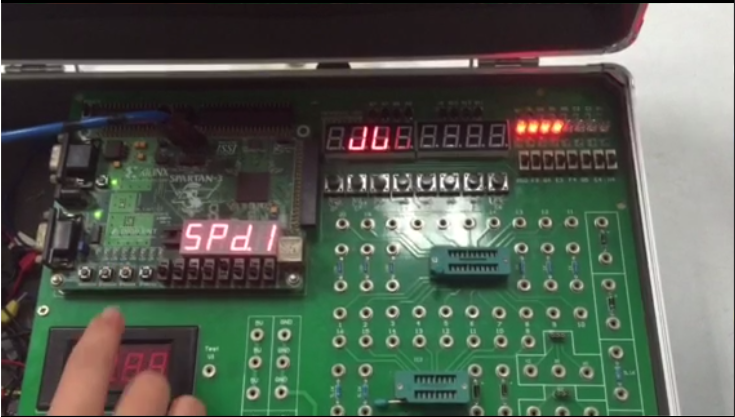


图5 单人模式游戏开始

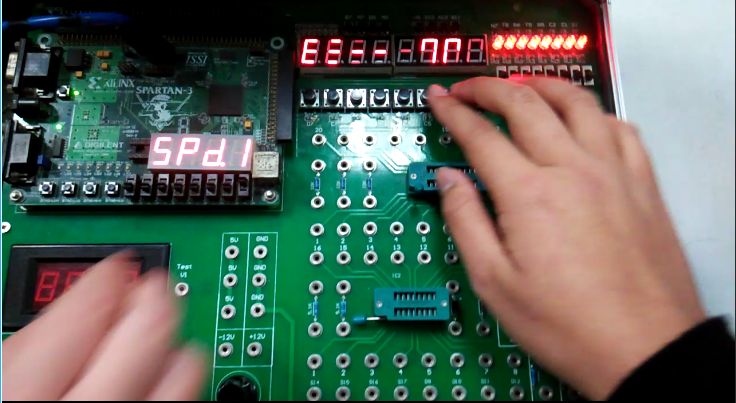


图6 双人模式游戏开始

（4）游戏过程

扩展板上七段数码管显示符号根据当前速度档位在一定时间后显示下一个符号，玩家必须在变化的时间内按下七段数码管下方对应方向控制的button，按错或不按则生命值减一，即扩展板上LED灯熄灭一盏（双人模式下，LED灯熄灭顺序为从内向外顺次熄灭）。



图7 单人模式随机符号和指示灯

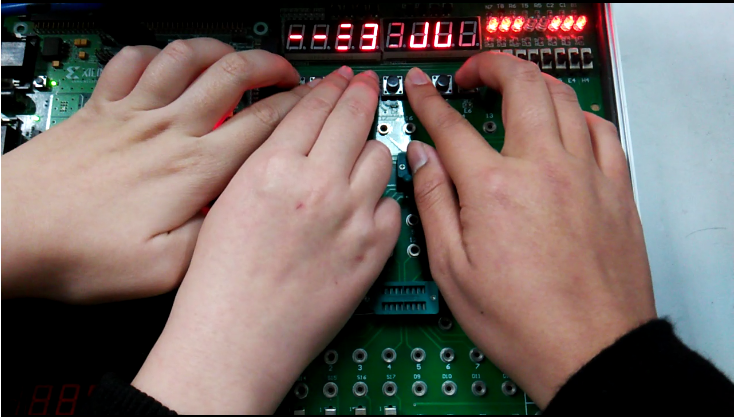


图8 双人模式随机符号和指示灯

（5）档位选择

游戏开始默认档位为最低速度档，将switch0置1，按一次button0则速度档位提升一个等级，将switch0置0，按一次button0则速度档位降低一个等级。

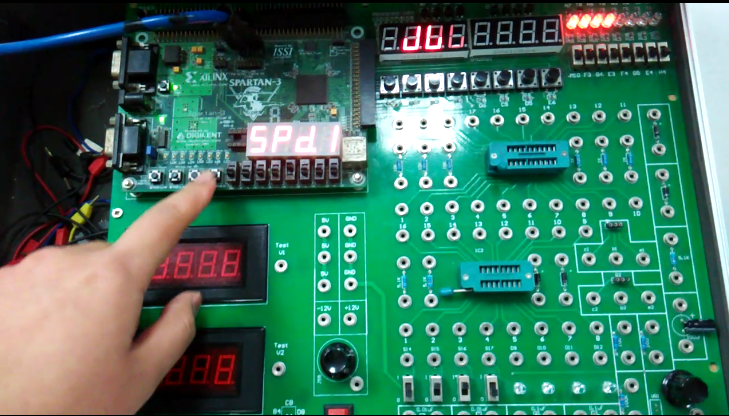


图9 初始速度

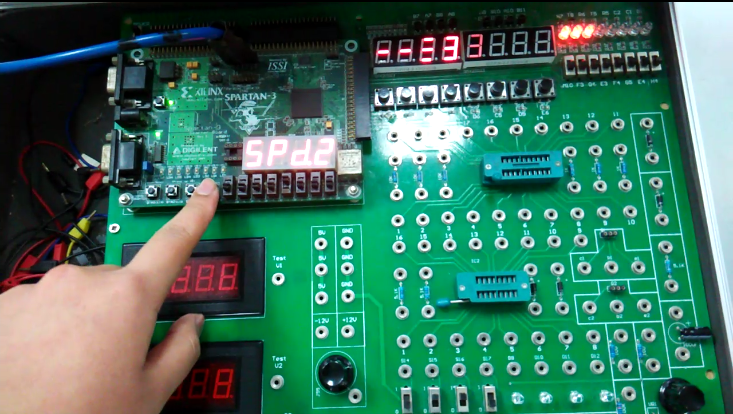


图10 提高速度

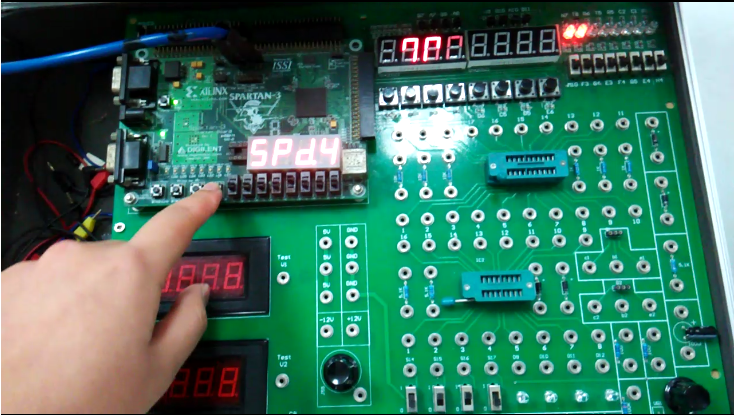
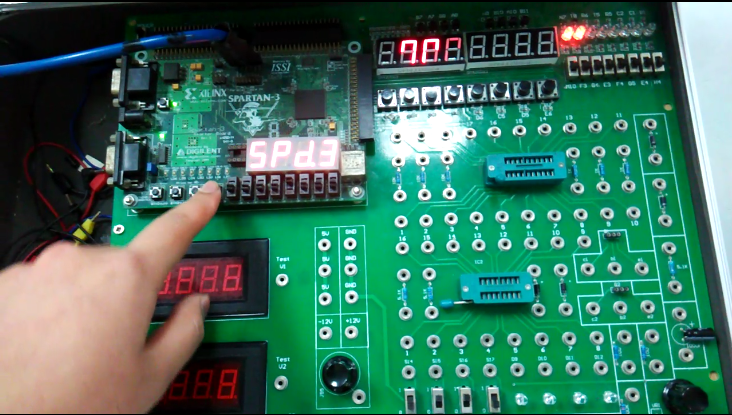


图11、12 速度档的全部展示

（6）游戏暂停

按button1按钮后游戏暂停。

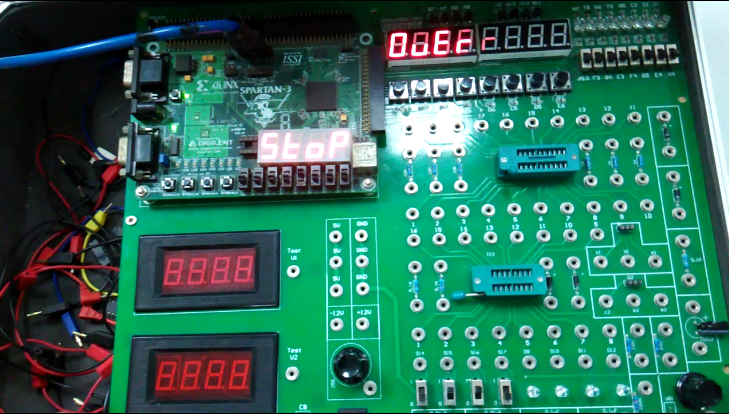


图13 游戏暂停

（7）游戏结束

单人模式生命值耗尽，正确显示游戏结束界面。双人模式任一玩家生命值耗尽，正确显示游戏结束界面。

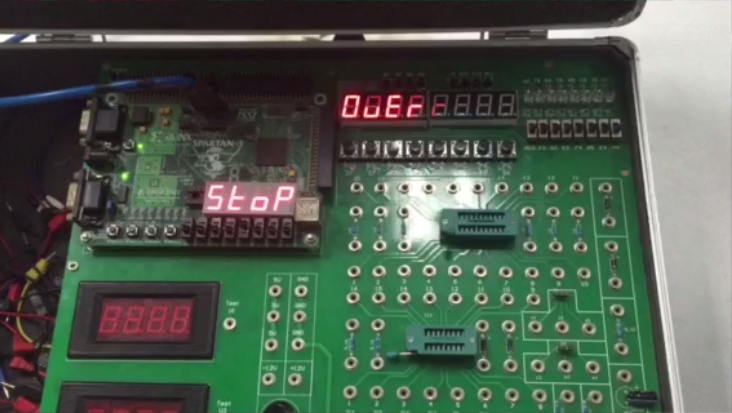


图14 单人模式玩家生命值耗尽

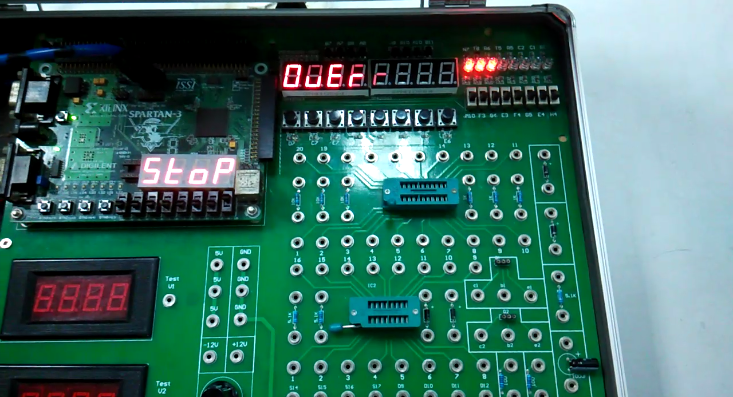


图15 双人模式玩家2生命值耗尽

## 4.2技术参数测试

技术参数1：随机数生成符号

随机数的生成其实是伪随机数的出现，对于玩家而言呈现无规律性。

（1）单人模式随机符号出现测试数据

|  |  |
| --- | --- |
| 随机符号位次 | 随机符号 |
| 1 | 上 |
| 2 | 左 |
| 3 | 右 |
| 4 | 右 |
| 5 | 左 |
| 6 | 下 |

（2）单人模式随机符号出现测试数据

|  |  |  |
| --- | --- | --- |
| 随机符号位次 | 玩家1 | 玩家2 |
| 1 | 上 | 上 |
| 2 | 左 | 上 |
| 3 | 右 | 左 |
| 4 | 下 | 右 |
| 5 | 右 | 下 |
| 6 | 上 | 左 |

技术参数2：生命值

|  |  |
| --- | --- |
| 失误次数 | LED灯为亮的数量 |
| 0 | 4 |
| 1 | 3 |
| 2 | 2 |
| 3 | 1 |
| 4 | 0 |

技术参数3：四个速度档位

|  |  |
| --- | --- |
| 档位 | 随机符号变化时间间隔 |
| 1 | 64ms |
| 2 | 48ms |
| 3 | 32ms |
| 4 | 16ms |

## 4.3系统演示与操作说明

4.3.1系统操作说明

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | 引脚 | 取值 | 含义 |
| 输入 | switch0 | 0 | 选择单人挑战模式 |
| 1 | 选择双人对抗模式 |
| switch3 | 0 | 开启降速功能 |
| 1 | 开启升速功能 |
| btn0 | 1 | 开始游戏或暂停游戏 |
| btn1 | 1 | 升速或降速 |
| 扩展板btn[0--7] | 1 | 玩家输入方向 |
| 输出 | / | / | 显示随机符号在扩展板上的七段数码管 |
| / | / | 显示当前生命值在扩展板上的LED灯 |
| / | / | 显示当前速度等级在Spartan-3板上七段数码管 |

4.3.2主要界面说明

（1）游戏设置键switch0、switch3、btn0、btn1



（2）游戏输入方向键



（3）HELLO显示



（4）STOP显示



（5）OVER显示



（6）生命值显示

单人模式

初始生命值 游戏中生命值 游戏结束生命值

双人模式



初始生命值 游戏中生命值 游戏结束生命值

（7）方向符号显示



上 下 左 右

（8）速度档位显示

第一档 第二档 第三档 第四档

# 5结论与展望

在本次课程设计中，我们小组设计了一个小游戏，运用到了本学期数字逻辑实验课所学到的很多知识。比如计时器的使用，如何进行分频调速，数码管的显示及利用分时扫描原理同时显示多个数码管上的内容。在实验课上，我们没有独立写过一个完整的模块甚至是程序，只是对ppt上给出的成品进行改进，然后仿真并下载运行。这次让我们独立设计一个游戏，可以说刚开始着手设计的时候完全没有头绪，不知道该从何处下手。考虑到spartan板上可利用的有限个按钮和数码管，我们初步定下以数码管作为显示界面，按钮作为玩家控制按键来设计游戏。而扩展板上的八个数码管下方又正好对应八个按钮，我们就想到了QQ炫舞和节奏大师的游戏原理。这两个游戏都是要在规定时间点击打音符得分，QQ炫舞是要玩家选择上下左右按键来击打对应音符，节奏大师是要玩家通过手机屏幕上的四个击打点来击打到来的音符。而利用同步时序电路的一些知识，就很容易实现这个过程。

我们通过设计计时器来设置音符出现的时间间隔，利用display和display32bits模块来显示游戏界面以及音符，利用button作为玩家的上下左右控制按钮......这些模块在实验课中都作了详细介绍，当时做实验时也对其进行了透彻理解，对其改进和运用并不是很难。

然而在写好每个模块，进行综合的时候却出现了一大堆error。其中遇到的最难解决的一个错误是在不同的always块里，不能改变同一个变量的值。后来我们折腾了半天，最后将两个always块里的同一个变量拆分成两个变量，再在另一个always块里对用这两个变量，在不同的条件下对原来的变量进行赋值，这才解决了这个问题。

游戏还有很多值得改进的地方，比如必须在下一个时钟上升沿到来的瞬间，对应的btn\_ctr\_out按键处于按下状态才表示击中音符。虽然其原理和我们所参考的QQ炫舞及节奏大师的原理一样，但由于板子的限制，我们无法将这个有效的击打时间点可视化，即无法告诉玩家准确的击打时间点。玩家只能通过大致判断和一直保持按键处于按下的状态才行，不然很难判断下个时钟周期到来的时间。对于这个问题，可能的解决方法有：在spartan板上显示倒计时，倒计时一结束瞬间要按下按钮，或者是只要在当前的时钟周期内按下对应按钮即表示击中。 还有一个问题，就是随机音符的实现只是一个伪随机实现方法，通过写足够多的状态来手动实现“随机”。但实际上可以写一个能产生真正随机数的模块，让音符的产生更加随机以增加游戏的可玩性。以上这些功能都有更大的改进空间，但鉴于时间原因我们没能一一实现，若今后有机会我们会将这个游戏实现得更加完美。

这次课程设计大大增进了我们对verilog语言的理解。从一个独立的模块都写不出来，到完成一个小游戏的设计，对于我们组内的每个成员来说都是突破性的进步。在写代码，综合，仿真，以及debug过程中，我们了解到了在课程上所没有学习到的很多内容，以及如何处理一些棘手的bug，组内成员的合作能力也有了很大的提升。

刚做project时我们头脑风暴想出了很多游戏和功能，真正着手时却很迷茫，不知道该从哪里开始入手。后来我们提炼出核心就是要能显示随机符号，并将用户输入和正确值对比，以判断是否丢失生命值。我们就先完成了这一核心功能，再在这一功能上拓展开去，project的完成就有计划有条理了许多，最终实现了我们想要实现的各种功能。这或许也正是project除了强化基础知识之外的意义，完成一个较大的工程时，要能先去繁求简，再不断修饰添彩，这样完成过程能明晰而且有把握，也易于控制时间进度。

本次课程设计顺利结束，虽然有一些小小的缺憾，但在实践过程中我们收获了很多很多。