北京邮电大学信息与通信工程学院

2017级数字电路与逻辑设计实验报告

简易迷宫游戏

班级：2017211128

学号：2017212040

班内序号：35

姓名：袁畅

1. **设计课题的任务和要求**

简易迷宫设计

设计并实现一个简易迷宫游戏机。

基本要求：

1、用 8×8 点阵进行游戏显示。

2、迷宫游戏如图 1 所示，采用双色点阵显示，其中红色LED 为迷宫墙壁，绿色LED

表示人物。通过BTN0～BTN3 四个按键控制迷宫中的人物进行上下左右移动，使人

物从起始点出发，走到迷宫的出口，游戏结束。

3、普通计时模式：通过按键BTN7 启动游戏，必须在30 秒内找到出口，否则游戏失败，

用两个数码管进行倒计时显示。游戏胜利或者失败均要在8×8 点阵上有相应的画面

出现。

4、迷宫中的人物在行走过程中，如果碰到墙壁，保持原地不动。

提高要求：

1、多种迷宫地图可以选择。

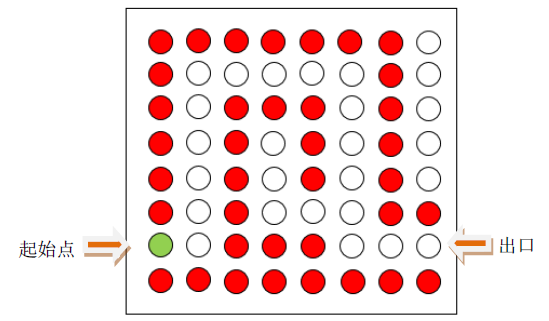
2、在计时的基础上增加计步的功能，每按一次控制按键步数加1，碰壁不计算步数，

计步结果用数码管显示。

3、为游戏增加提示音乐，在不同时间段采用不同频率的信号控制蜂鸣器发声报警。

4、增加其他游戏模式。

5、自拟其它功能。



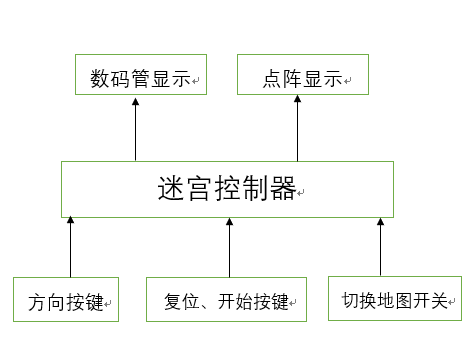
1. **系统设计**

**I、设计思路**

程序采用划分模块的思路，将具体任务分解为不同的任务子模块，通过模块例化的方式，将各模块连接起来。主程序maze中包括各个子模块的例化语句，子模块分为消抖模块、点阵显示模块、数码管显示模块、倒计时模块和二进制转十进制模块。

**II、总体框图**

（1）顶层框图



（2）逻辑划分框图

数码管

digit\_con digit\_seg0

数码管选位

数码管显示

地图切换

点阵显示

r\_col step\_cnt

change\_map g\_col suc time\_sign

30秒倒计时

坐标及显示控制

time\_sign

start\_lock clk\_1Hz direction1、2、3、4\_lock

分频器

消抖

clk\_1kHz

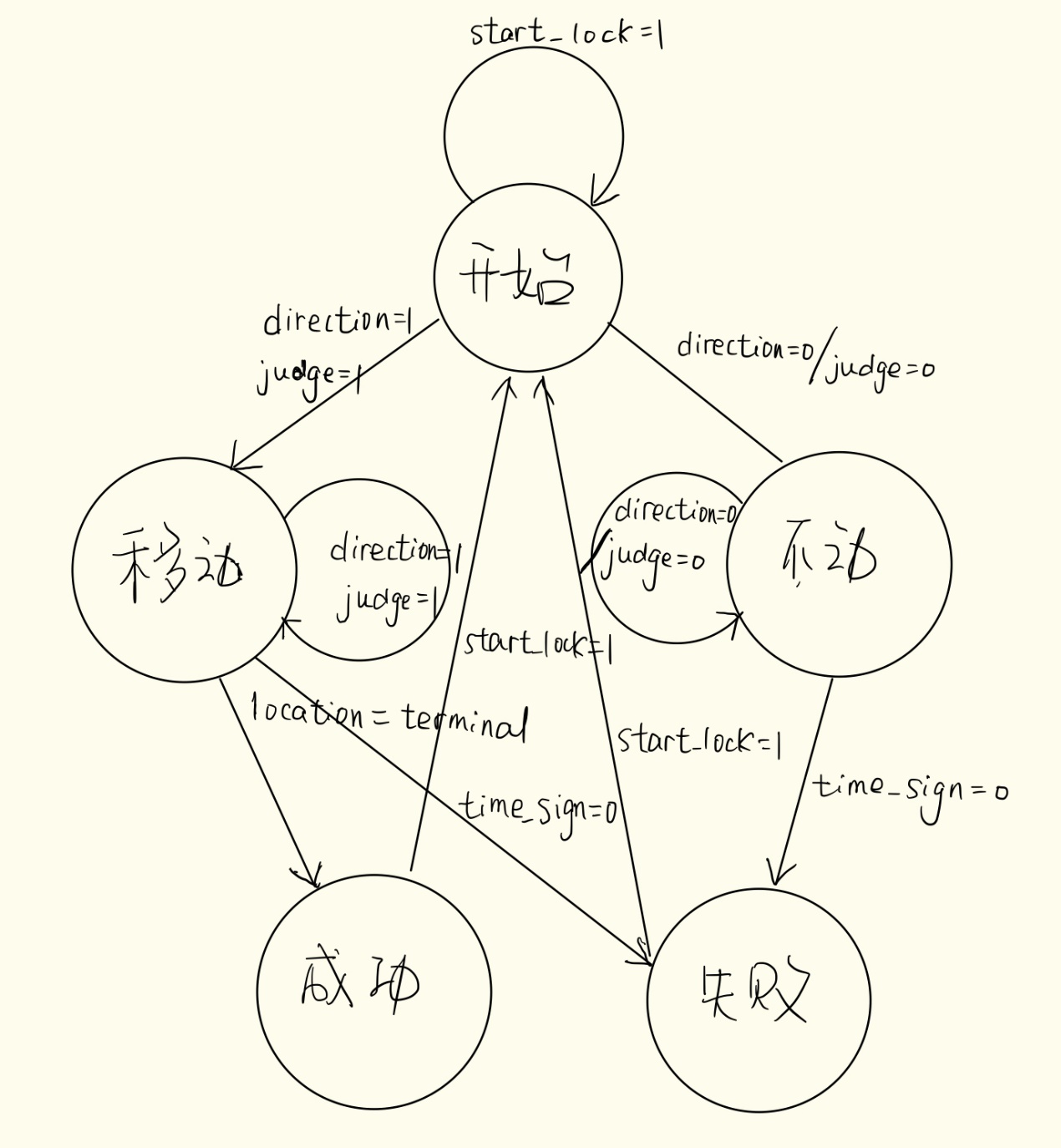
clk

时钟

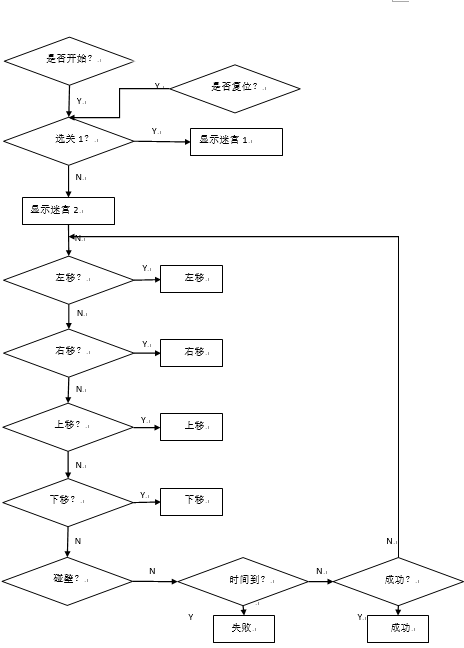
复位、开始按键

方向按键

（3）ASM图

****

(3)控制器逻辑图



**III、分块设计**

（1）消抖模块

module sw(clk,key\_in,key\_out);

在按键信号上升沿或下降沿到来后，间隔2ms左右的时间再次检测按键的状态，将此时按键的状态值赋值给key\_out，输出信号。

（2）点阵显示模块

module lattice\_on(clk,rst,start,change\_map,direction1,direction2,direction3,direction4,time\_sign,row,g\_col,r\_col,suc,step\_cnt);

初始化两张地图map\_row1、map\_row2及各张地图的起点和终点信息。初始状态下start\_con状态值为0，此时数码管及点阵处于关闭状态，当按下start后，start\_con值持续赋1，数码管和点阵开始工作。拨码开关map\_change的状态值为0时，显示第一张地图，当map\_change=1时，点阵显示切换到第二张地图，按下start键，小绿点回到起点。

当小绿点达到终点，即suc=1时，点阵显示绿色的笑脸；当time\_sign=0，即30s倒计时时间结束后，而小绿点没有到达终点位置，则点阵显示红色的哭脸。

为使点阵能够正常显示图案，采用动态扫描的方法，即当扫描到row\_num行时，给r\_col、g\_col赋予不同的值，并且采用clk\_1kHz=1kHz的频率进行扫描，以产生视觉停留的效果，得到点阵显示。

（3）数码管显示模块

module digit\_tube(clk,start,change\_map,time\_sign,step\_cnt,digit\_con,digit\_seg0);

当start=0时，数码管不显示，当start=1时，数码管开始工作。

数码管分别由第0位、第1位显示倒计时，由第6位、第7位显示小绿点走过的步数。

当按下start键后，计步器重新开始，倒计时重新开始，当suc=1，即小绿点到达终点后，倒计时停止，等待重置。

数码管同样采用动态扫描的方法显示，当扫描到第0、1、6和7号数码管时，给digit\_seg0赋以相应的值，并以clk\_1kHz=1kHz的频率扫描，以产生视觉停留的效果，得到点阵显示。

（4）倒计时模块

module countdown(clk,but,one,ten,time1,suc);

由一个从30到0的计数变量，初始值为30，当suc=1 时，开始工作。每当遇到1Hz时钟的上升沿时，计数器减1，知道计数器减为0，或者suc=1为止。将计数变量作为输出变量传出模块。

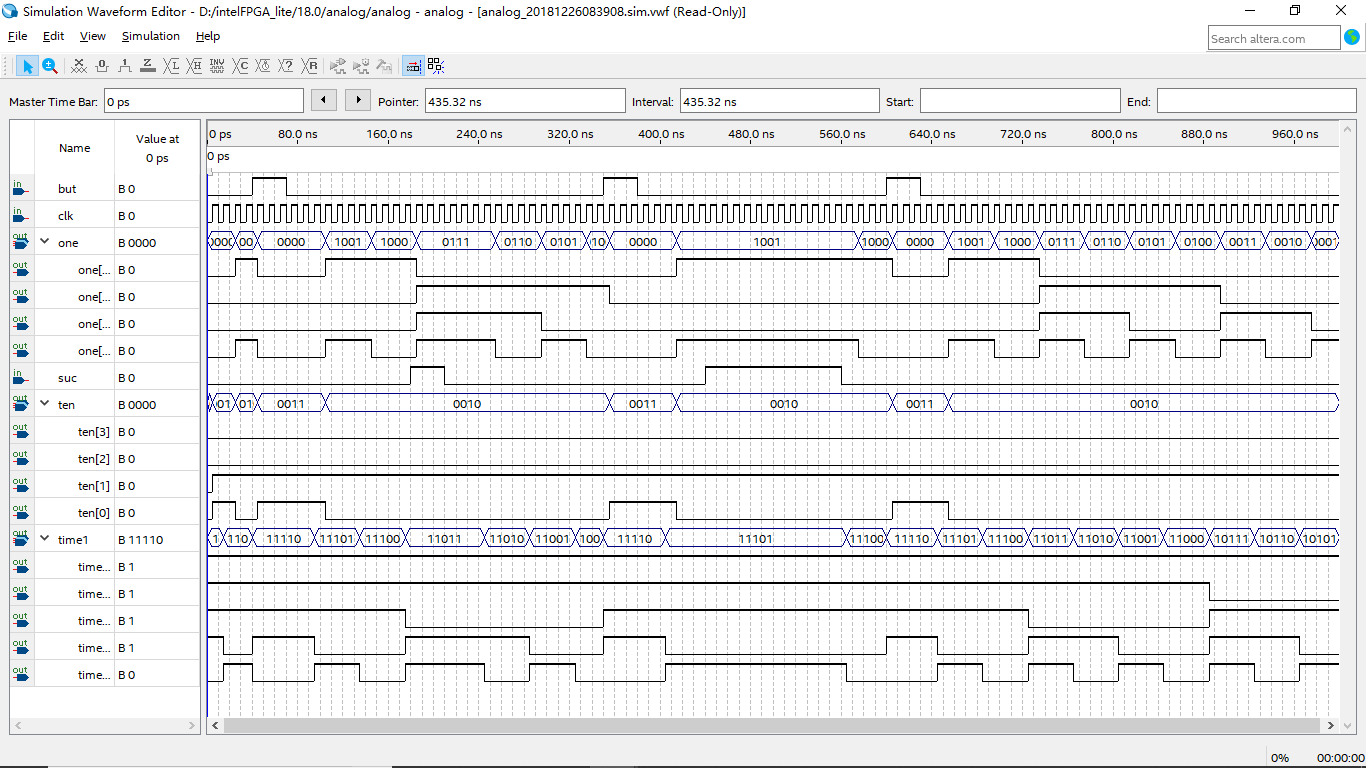
（5）二进制转十进制模块

module two\_ten(clk,num,ten,one);

输入一个100（十进制）以内的数，输出它的十位数（ten）和个位数（one）。

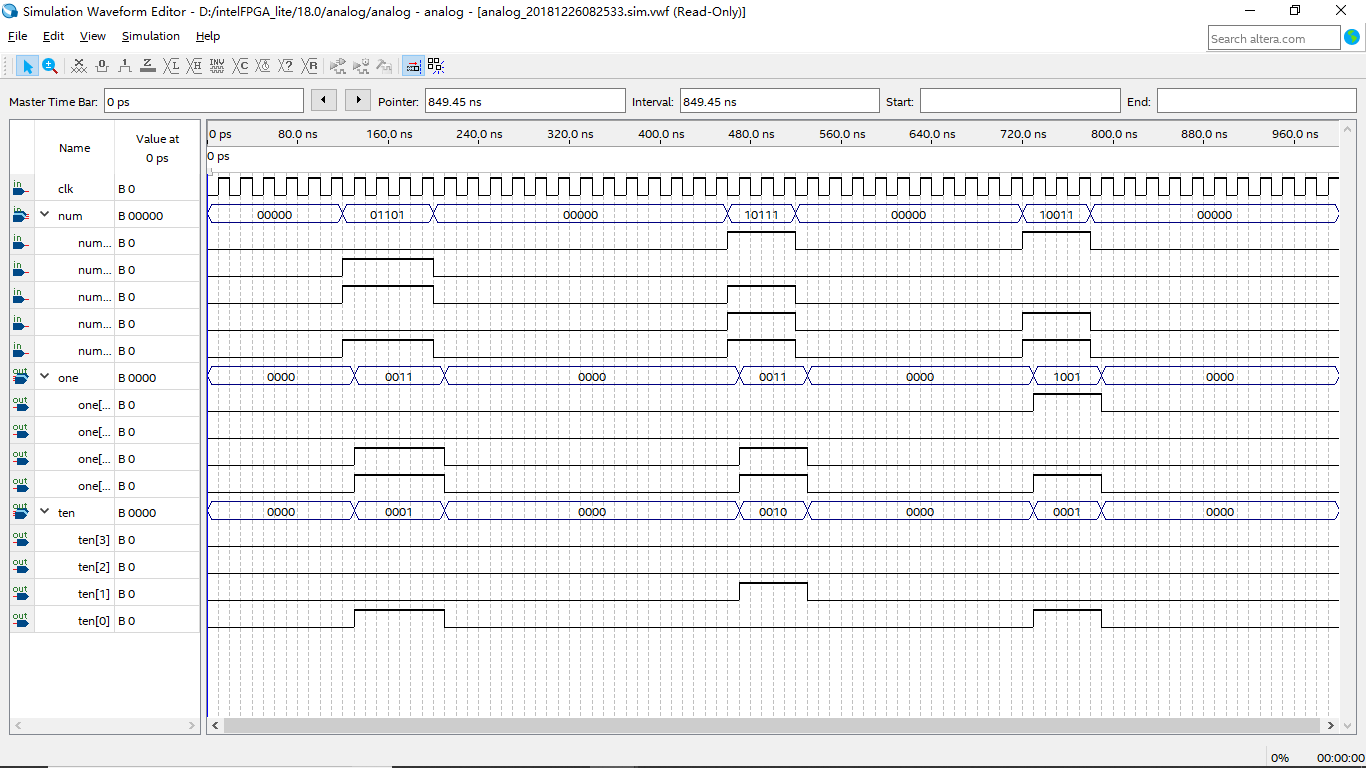
**三、仿真波形及波形分析**

（1）倒计时仿真



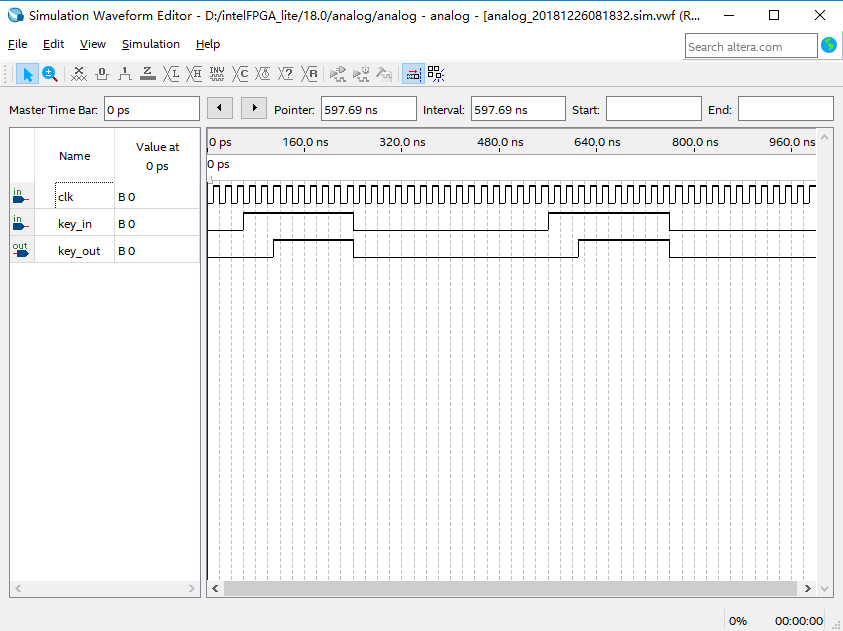
当按下but键，即当but=1时，倒计时器重新倒计时；当suc=1时，即代表小绿点已经到达终点时，倒计时停止。输出信号为所剩时间（time1）、时间十进制显示的十位（ten）以及个位显示（one）。

2、二进制转十进制模块仿真



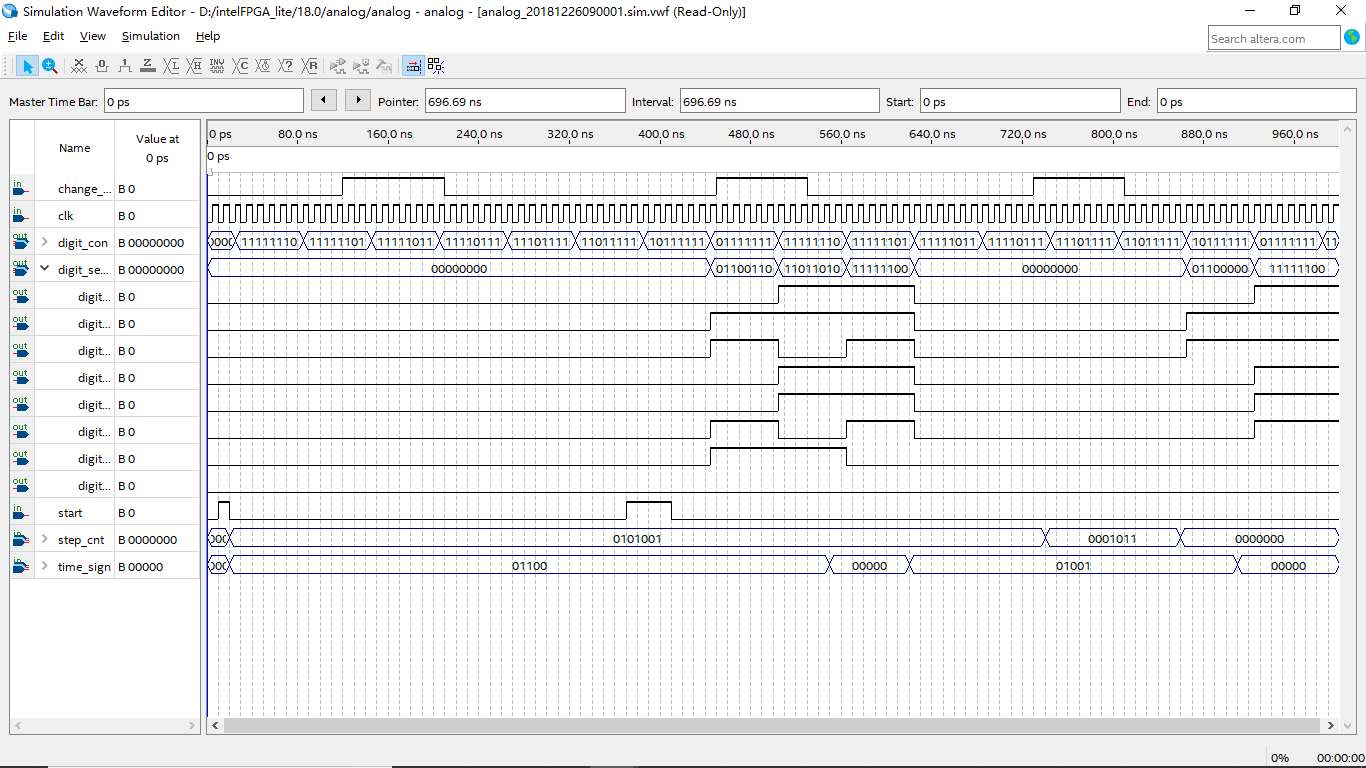
将待转化的二进制数num传进模块中，模块输出转化后的十位和个位，即ten、one。

3、按键消抖仿真



当key\_out迎来key\_in的下降沿时，将key\_in的值赋给key\_out，当key\_in的上升沿到来时，间隔一段时间后再次检测key\_in的电平状态，若依然为高电平，则将key\_in的状态值赋给key\_out，借此达到按键消抖的目的。

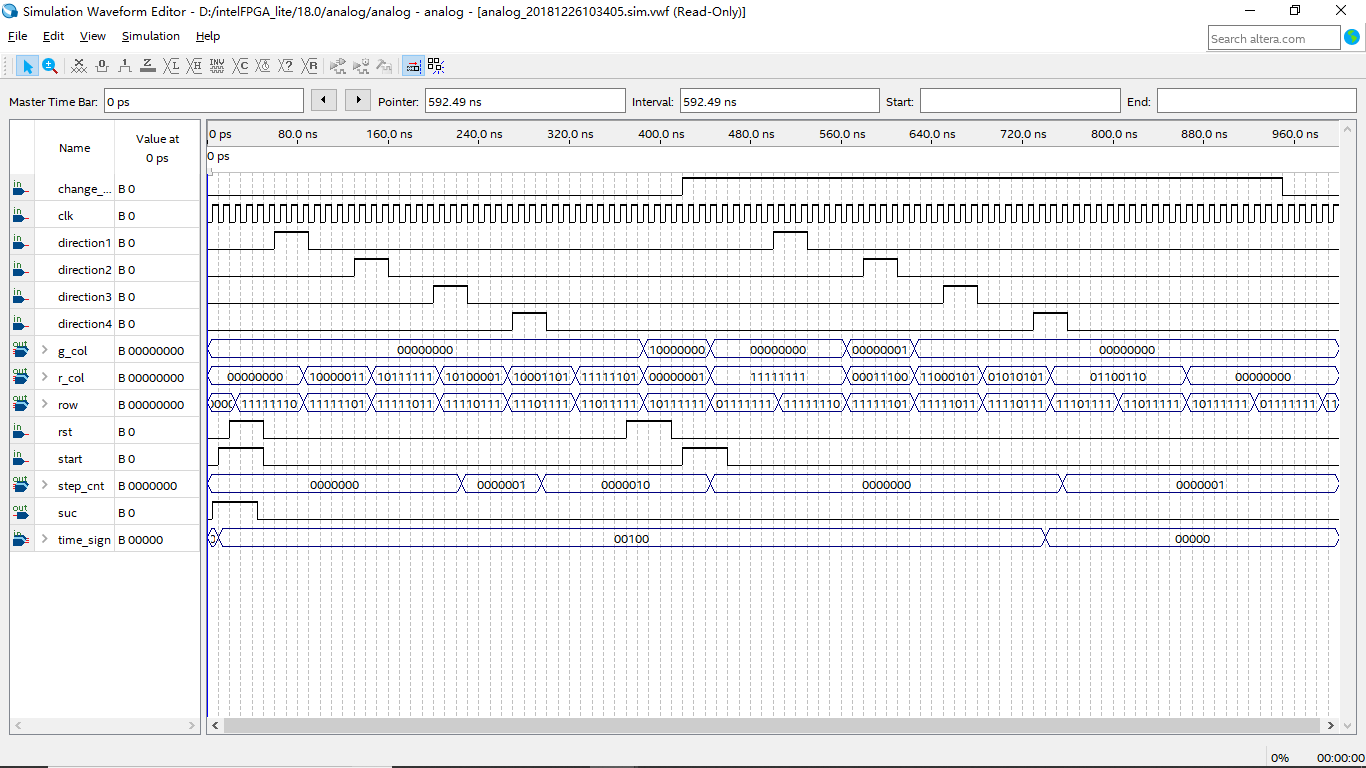
4、数码管显示模块仿真



按下start键后，数码管开始工作。

数码管显示采用动态扫描的方法，当扫描到第0号数码管时，显示time\_sign，即时间的个位；当扫描到1号数码管时，给digit\_seg0赋值time\_sign的十位；当扫描到6号数码管时，给digit\_seg0赋值step\_cnt的个位，即计步器的个位数；当扫描到7号数码管时，给digit\_seg0赋值step\_cnt的十位。扫描到其他数码管时，数码管各位为低电平，数码管不显示。

5、点阵显示模块仿真

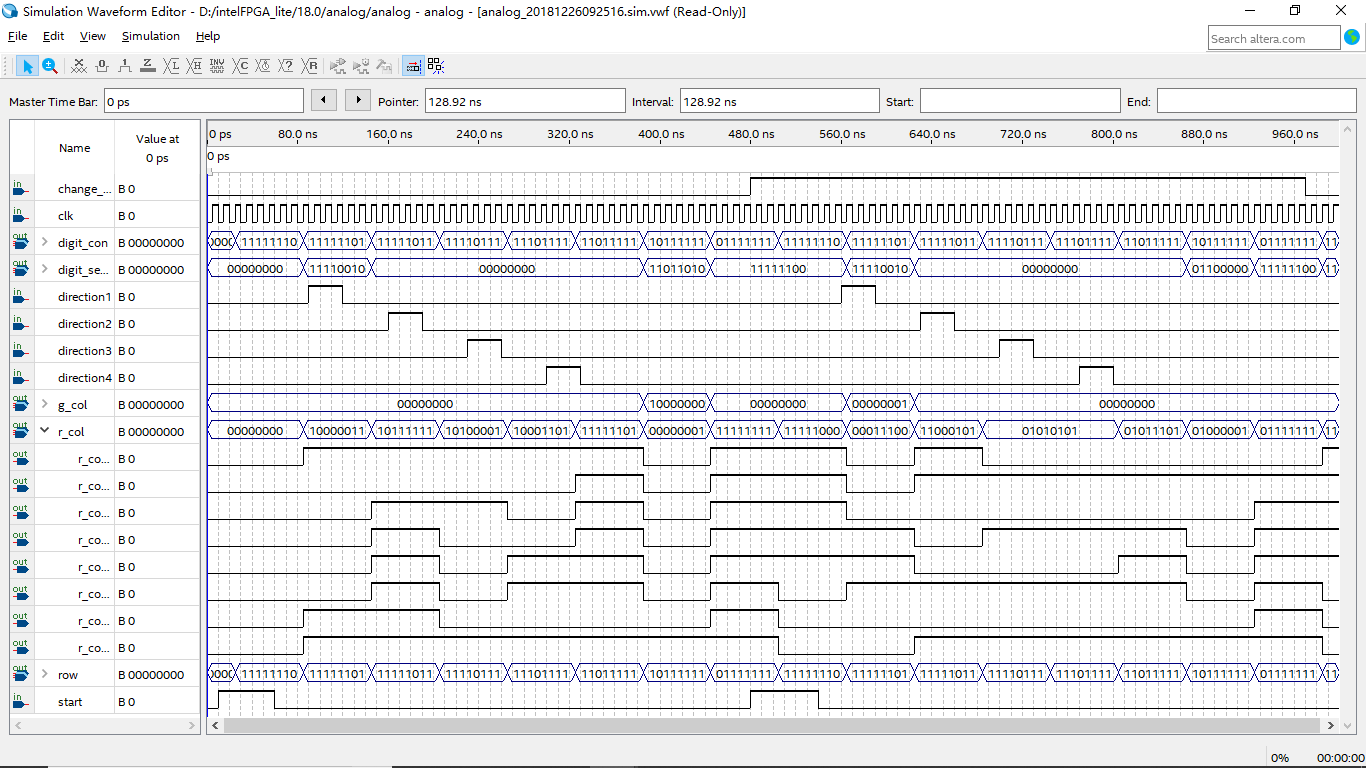


当按下start键后，点阵开始显示。

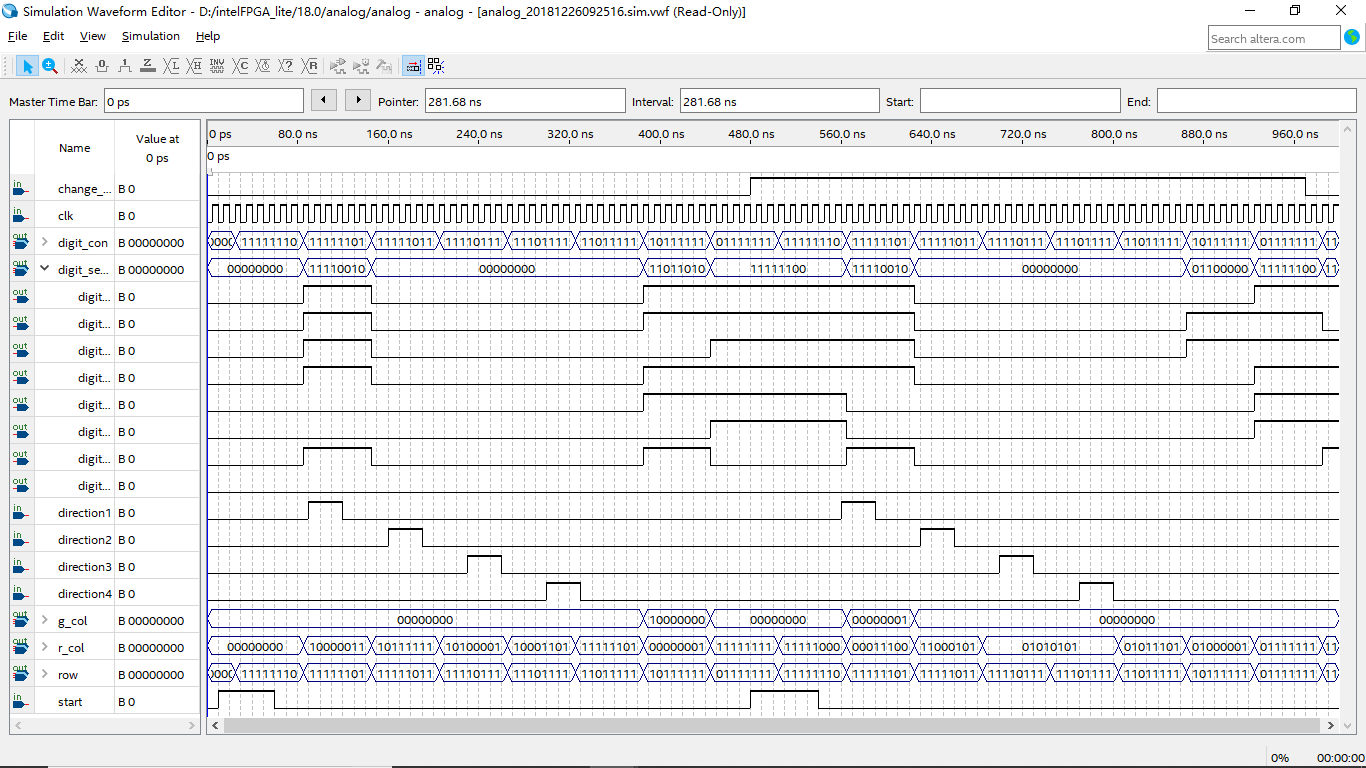
当change\_map=0时，点阵显示第一张地图。当chang\_map=1时，点阵显示第二张地图。点阵采用动态扫描的方法，当扫描到相应的行时，分别给r\_col、g\_col赋予相应的值，当时间信号time\_sign不等于0时，点阵正常显示，当time\_sign=1时，点阵显示红色的哭脸。

小绿点的方向上下左右分别由direction1、direction2、direction3和direction4控制，在经过判断之后，若小绿点可以移动，则计步器step\_cnt加1，否则值不变。而小绿点的位置也作出相应地改变或者是原地不动。

6、总体模块仿真



总体仿真点阵红色部分



总体仿真数码管部分

初始状态下，点阵和数码管不显示。当按下start后，数码管和点阵开始显示。再次按下start键后，计步器、小绿点已经倒计时装置回到初始状态，游戏重新开始。

当change\_map=0时，点阵显示第一张地图；当change\_map=1时，点阵显示第二张地图。点阵和数码管采用动态扫描的方式，当扫描到相应的点阵行或者是数码管时，为其赋上相应的值。控制方向的按键按下后，经过判断，对小绿点的位置信息和计步器做出相应的改变或者是保持不变。

**四、源代码**

module maze(clk,start,change\_map,direction1,direction2,direction3,direction4,digit\_seg0,digit\_con,r\_col,g\_col,row);

input clk; //时钟信号

input start; //开始按键（复位按键）

input change\_map; //切换地图

input direction1; //控制小绿点的移动方向，分别为上、下、左、右

input direction2;

input direction3;

input direction4;

output [7:0] digit\_seg0; //数码管

output [7:0] digit\_con; //数码管选位控制

output [7:0] r\_col; //红色点阵

output [7:0] g\_col; //绿色点阵

output [7:0] row; //点阵行数选择器

wire [4:0] time\_sign;

wire suc;

wire [6:0] step\_cnt;

wire [3:0] one;

wire [3:0] ten;

reg rst1=1'b0;

wire rst;

assign rst=rst1;

countdown l3(.clk(clk),.but(start),.time1(time\_sign),.ten(ten),.one(one),.suc(suc));

lattice\_on l1(.clk(clk),.rst(rst),.start(start),.change\_map(change\_map),.direction1(direction1),.direction2(direction2),.direction3(direction3),.direction4(direction4),.time\_sign(time\_sign),.row(row),.g\_col(g\_col),.r\_col(r\_col),.suc(suc),.step\_cnt(step\_cnt));

digit\_tube l2(.clk(clk),.start(start),.change\_map(change\_map),.time\_sign(time\_sign),.step\_cnt(step\_cnt),.digit\_con(digit\_con),.digit\_seg0(digit\_seg0));

endmodule

//数码管显示模块

module lattice\_on(clk,rst,start,change\_map,direction1,direction2,direction3,direction4,time\_sign,row,g\_col,r\_col,suc,step\_cnt);

input clk;

input start;

input rst;

input change\_map;

input direction1;

input direction2;

input direction3;

input direction4;

input [4:0] time\_sign;

output reg [7:0] row;

output reg [7:0] g\_col;

output reg [7:0] r\_col;

output reg suc=1'b0; //判断小绿点是否到达终点

output reg [6:0] step\_cnt=7'b000\_0000; //计步器

reg [7:0] map\_row [7:0]; //地图信息

reg [7:0] map\_row\_1 [7:0]; //第一张地图信息

reg [7:0] map\_row\_2 [7:0]; //第二张地图信息

//地图信息初始化

initial

begin

map\_row\_1[0]=8'b1111\_1000;

map\_row\_1[1]=8'b1000\_0011;

map\_row\_1[2]=8'b1011\_1111;

map\_row\_1[3]=8'b1010\_0001;

map\_row\_1[4]=8'b1000\_1101;

map\_row\_1[5]=8'b1111\_1101;

map\_row\_1[6]=8'b0000\_0001;

map\_row\_1[7]=8'b1111\_1111;

map\_row\_2[0]=8'b1111\_1111;

map\_row\_2[1]=8'b0001\_1100;

map\_row\_2[2]=8'b1100\_0101;

map\_row\_2[3]=8'b0101\_0101;

map\_row\_2[4]=8'b0101\_0101;

map\_row\_2[5]=8'b0101\_1101;

map\_row\_2[6]=8'b0100\_0001;

map\_row\_2[7]=8'b0111\_1111;

end

reg [2:0] start\_point\_col; //起始点行数信息

reg [2:0] start\_point\_row; //起点列数信息

reg [2:0] terminal\_point\_row; //终点信息

reg [7:0] terminal\_point\_col;

reg [7:0] start\_point\_1=6'b110\_111;

reg [5:0] terminal\_point\_1=6'b000\_000;

reg [5:0] start\_point\_2=6'b001\_000;

reg [5:0] terminal\_point\_2=6'b001\_111;

reg [2:0] location\_row\_num; //小绿点位置信息

reg [2:0] location\_col\_num;

wire [2:0] maze\_con; //三位信息分别代表开始，切换地图，以及切换模式

reg start\_con=1'b0;

reg map\_con=1'b0;

reg mode\_con=1'b0;

assign maze\_con = {mode\_con,map\_con,start\_con};

wire start\_lock;

wire change\_map\_lock;

sw a1(.clk(clk),.key\_in(start),.key\_out(start\_lock)); //start按键消抖

sw a2(.clk(clk),.key\_in(change\_map),.key\_out(change\_map\_lock)); //change\_map按键消抖

always @(posedge start\_lock) begin

if(start\_lock==1) begin

start\_con<=1'b1;

end

end

always @(change\_map\_lock) begin

if(change\_map\_lock==1) begin

map\_con<=1'b1;

end

else begin

map\_con<=1'b0;

end

end

always @(posedge clk) begin //由map\_con的值决定map\_row的取值，即地图信息及起点、终点位置信息

if(map\_con==0) begin

map\_row[0]=map\_row\_1[0];

map\_row[1]=map\_row\_1[1];

map\_row[2]=map\_row\_1[2];

map\_row[3]=map\_row\_1[3];

map\_row[4]=map\_row\_1[4];

map\_row[5]=map\_row\_1[5];

map\_row[6]=map\_row\_1[6];

map\_row[7]=map\_row\_1[7];

start\_point\_row=start\_point\_1[5:3];

start\_point\_col=start\_point\_1[2:0];

terminal\_point\_row=terminal\_point\_1[5:3];

terminal\_point\_col=terminal\_point\_1[2:0];

end

else if(map\_con==1)begin

map\_row[0]=map\_row\_2[0];

map\_row[1]=map\_row\_2[1];

map\_row[2]=map\_row\_2[2];

map\_row[3]=map\_row\_2[3];

map\_row[4]=map\_row\_2[4];

map\_row[5]=map\_row\_2[5];

map\_row[6]=map\_row\_2[6];

map\_row[7]=map\_row\_2[7];

start\_point\_row=start\_point\_2[5:3];

start\_point\_col=start\_point\_2[2:0];

terminal\_point\_row=terminal\_point\_2[5:3];

terminal\_point\_col=terminal\_point\_2[2:0];

end

end

wire direction\_lock1;

wire direction\_lock2;

wire direction\_lock3;

wire direction\_lock4;

//方向控制键按键消抖

debounce c1(clk,~rst,direction1,direction2,direction3,direction4,direction\_lock1,direction\_lock2,direction\_lock3,direction\_lock4);

//sw c1(.clk(clk),.key\_in(direction1),.key\_out(direction\_lock1));

//sw c2(.clk(clk),.key\_in(direction2),.key\_out(direction\_lock2));

//sw c3(.clk(clk),.key\_in(direction3),.key\_out(direction\_lock3));

//sw c4(.clk(clk),.key\_in(direction4),.key\_out(direction\_lock4));

always @(posedge clk)

begin

if(start\_lock==1) begin //按下start键后，计步器归零，小绿点回到起始位置

step\_cnt<=7'b0000000;

location\_row\_num=start\_point\_row;

location\_col\_num=start\_point\_col;

end

if(location\_row\_num!=terminal\_point\_row||location\_col\_num!=terminal\_point\_col) //小绿点移动及判断

begin

if(direction\_lock1&&location\_row\_num<7) //按下向上键，并且小绿点的行数小于7时，执行系列操作

begin

if(map\_row[location\_row\_num+1][location\_col\_num]==0) //当小绿点的上方为‘0’，即小绿点上方没有墙时，小绿点向上移动一个，并且计步器加1

begin

step\_cnt<=step\_cnt+1;

location\_row\_num=location\_row\_num+1;

end

else begin

location\_col\_num=location\_col\_num;

location\_row\_num=location\_row\_num;

end

end

if(direction\_lock2&&location\_row\_num>0)

begin

if(map\_row[location\_row\_num-1][location\_col\_num]==0) begin

step\_cnt<=step\_cnt+1;

location\_row\_num=location\_row\_num-1;

end

else begin

location\_row\_num=location\_row\_num;

location\_col\_num=location\_col\_num;

end

end

if(direction\_lock3&&location\_col\_num>0)

begin

if(map\_row[location\_row\_num][location\_col\_num-1]==0)

begin

step\_cnt<=step\_cnt+1;

location\_col\_num=location\_col\_num-1;

end

else begin

location\_col\_num=location\_col\_num;

location\_row\_num=location\_row\_num;

end

end

if(direction\_lock4&&location\_col\_num<7)

begin

if(map\_row[location\_row\_num][location\_col\_num+1]==0)

begin

step\_cnt<=step\_cnt+1;

location\_col\_num=location\_col\_num+1;

end

else begin

location\_row\_num=location\_row\_num;

location\_col\_num=location\_col\_num;

end

end

end

end

//当小绿点到达终点后，成功

always @(posedge clk) begin

if((location\_row\_num==terminal\_point\_row)&&(location\_col\_num==terminal\_point\_col)) begin

suc<=1'b1;

end

else

suc<=1'b0;

end

reg [31:0] count=32'd0;

reg clk\_1kHz;

always @(posedge clk) begin //1k频率的时钟

if(count==50000) begin

count<=32'd0;

clk\_1kHz<=~clk\_1kHz;

end

else begin

count<=count+1;

end

end

//点阵行数控制器，采用1kHz的频率动态扫描点阵

reg [3:0] row\_num=4'b0000;

always @(posedge clk\_1kHz) begin

if(row\_num!=4'b0111) begin

row\_num<=row\_num+1;

end

else row\_num<=4'b0000;

end

//当扫描到不同的行时，给点阵选行器赋予相应的值

always @(posedge clk\_1kHz) begin

case(row\_num)

4'h0: begin

row<=8'b1111\_1110;

end

4'h1: begin

row<=8'b1111\_1101;

end

4'h2: begin

row<=8'b1111\_1011;

end

4'h3: begin

row<=8'b1111\_0111;

end

4'h4: begin

row<=8'b1110\_1111;

end

4'h5: begin

row<=8'b1101\_1111;

end

4'h6: begin

row<=8'b1011\_1111;

end

4'h7: begin

row<=8'b0111\_1111;

end

endcase

end

always @(posedge clk\_1kHz) begin

if(start\_con==0) begin //当没有按下start键时，点阵不显示

r\_col<=8'b000\_0000;

g\_col<=8'b000\_0000;

end

else if(start\_con==1&&time\_sign!=0&&suc==0) begin //当倒计时没有结束，小绿点没有到达终点，点阵显示

case(row\_num)

4'd0:r\_col<=map\_row[0];

4'd1:r\_col<=map\_row[1];

4'd2:r\_col<=map\_row[2];

4'd3:r\_col<=map\_row[3];

4'd4:r\_col<=map\_row[4];

4'd5:r\_col<=map\_row[5];

4'd6:r\_col<=map\_row[6];

4'd7:r\_col<=map\_row[7];

endcase

if(row\_num==location\_row\_num) begin

case(location\_col\_num)

3'd0:g\_col<=8'b0000\_0001;

3'd1:g\_col<=8'b0000\_0010;

3'd2:g\_col<=8'b0000\_0100;

3'd3:g\_col<=8'b0000\_1000;

3'd4:g\_col<=8'b0001\_0000;

3'd5:g\_col<=8'b0010\_0000;

3'd6:g\_col<=8'b0100\_0000;

3'd7:g\_col<=8'b1000\_0000;

endcase

end

if((row\_num)!=location\_row\_num)

begin

g\_col<=8'b0000\_0000;

end

end

else if(start\_con==1&&time\_sign==0&&suc==0) begin //倒计时结束时的哭脸显示

case(row\_num)

4'd0:r\_col<=8'b0000\_0000;

4'd1:r\_col<=8'b0100\_0010;

4'd2:r\_col<=8'b0011\_1100;

4'd3:r\_col<=8'b0000\_0000;

4'd4:r\_col<=8'b0110\_0110;

4'd5:r\_col<=8'b0110\_0110;

4'd6:r\_col<=8'b0000\_0000;

4'd7:r\_col<=8'b0000\_0000;

endcase

g\_col<=8'b0000\_0000;

end

else if(start\_con==1&&time\_sign!=0&&suc==1) begin //小绿点到达终点的绿色笑脸显示

case(row\_num)

4'd0:g\_col<=8'b0000\_0000;

4'd1:g\_col<=8'b0011\_1100;

4'd2:g\_col<=8'b0100\_0010;

4'd3:g\_col<=8'b0000\_0000;

4'd4:g\_col<=8'b0110\_0110;

4'd5:g\_col<=8'b0110\_0110;

4'd6:g\_col<=8'b0000\_0000;

4'd7:g\_col<=8'b0000\_0000;

endcase

r\_col<=8'b0000\_0000;

end

end

endmodule

//数码管显示模块

module digit\_tube(clk,start,change\_map,time\_sign,step\_cnt,digit\_con,digit\_seg0);

input clk;

input start;

input change\_map;

input [4:0] time\_sign; //倒计时所剩时间

input [6:0] step\_cnt; //计步器

output reg [7:0] digit\_con;

output reg [7:0] digit\_seg0;

wire [3:0] ten;

wire [3:0] one;

reg [3:0] ten\_step;

reg [3:0] one\_step;

//将时间信号二进制数转化为十进制

two\_ten t1(.clk(clk),.num(time\_sign),.ten(ten),.one(one));

//将步数的二进制数转化为十进制

integer i;

always @(posedge clk)

begin

ten\_step=4'd0;

one\_step=4'd0;

for(i=6;i>=0;i=i-1)

begin

if(ten\_step>4)

ten\_step=ten\_step+4'd3;

if(one\_step>4)

one\_step=one\_step+4'd3;

ten\_step=ten\_step<<1;

ten\_step[0]=one\_step[3];

one\_step=one\_step<<1;

one\_step[0]=step\_cnt[i];

end

end

reg [31:0] count=32'd0;

reg clk\_1kHz;

always @(posedge clk) begin //1k频率的时钟

if(count==50000) begin

count<=32'd0;

clk\_1kHz<=~clk\_1kHz;

end

else begin

count<=count+1;

end

end

reg [3:0] row\_num=4'b0000;

//与点阵显示模块中的选行控制器相同，这里做数码管选位控制器

always @(posedge clk\_1kHz) begin

if(row\_num!=4'b0111) begin

row\_num<=row\_num+1;

end

else row\_num<=4'b0000;

end

always @(posedge clk\_1kHz) begin

case(row\_num)

4'h0: begin

digit\_con<=8'b1111\_1110;

end

4'h1: begin

digit\_con<=8'b1111\_1101;

end

4'h2: begin

digit\_con<=8'b1111\_1011;

end

4'h3: begin

digit\_con<=8'b1111\_0111;

end

4'h4: begin

digit\_con<=8'b1110\_1111;

end

4'h5: begin

digit\_con<=8'b1101\_1111;

end

4'h6: begin

digit\_con<=8'b1011\_1111;

end

4'h7: begin

digit\_con<=8'b0111\_1111;

end

endcase

end

always @(posedge clk\_1kHz) begin

end

wire [2:0] maze\_con; //三位信息分别代表开始，切换地图，以及切换模式

reg start\_con=1'b0;

reg map\_con=1'b0;

reg mode\_con=1'b0;

assign maze\_con = {mode\_con,map\_con,start\_con};

wire start\_lock;

wire change\_map\_lock;

//start、change\_map键的消抖

sw b1(.clk(clk),.key\_in(start),.key\_out(start\_lock));

sw b2(.clk(clk),.key\_in(change\_map),.key\_out(change\_map\_lock));

always @(posedge start\_lock) begin

if(start\_lock==1) begin

start\_con<=1'b1;

end

end

always @(change\_map\_lock) begin

if(change\_map\_lock==0) begin

map\_con<=1'b0;

end

else begin

map\_con<=1'b1;

end

end

always @(posedge clk\_1kHz) begin

if(start\_con==0) begin //初始状态，数码管不显示

digit\_seg0<=8'b0000\_0000;

end

if(start\_con==1) begin

if(row\_num==1) begin //扫描到1号数码管，显示倒计时的十位

case(ten)

4'd0:digit\_seg0<=8'b11111100;

4'd1:digit\_seg0<=8'b01100000;

4'd2:digit\_seg0<=8'b11011010;

4'd3:digit\_seg0<=8'b11110010;

4'd4:digit\_seg0<=8'b01100110;

4'd5:digit\_seg0<=8'b10110110;

4'd6:digit\_seg0<=8'b10111110;

4'd7:digit\_seg0<=8'b11100000;

4'd8:digit\_seg0<=8'b11111110;

4'd9:digit\_seg0<=8'b11110110;

endcase

end

if(row\_num==0) begin //扫描到0号数码管，显示倒计时的个位

case(one)

4'd0:digit\_seg0<=8'b11111100;

4'd1:digit\_seg0<=8'b01100000;

4'd2:digit\_seg0<=8'b11011010;

4'd3:digit\_seg0<=8'b11110010;

4'd4:digit\_seg0<=8'b01100110;

4'd5:digit\_seg0<=8'b10110110;

4'd6:digit\_seg0<=8'b10111110;

4'd7:digit\_seg0<=8'b11100000;

4'd8:digit\_seg0<=8'b11111110;

4'd9:digit\_seg0<=8'b11110110;

endcase

end

if(row\_num==2||row\_num==3||row\_num==4||row\_num==5) begin //其余位数码管不显示

digit\_seg0<=8'b0000\_0000;

end

if(row\_num==6) begin //扫描到6号数码管，显示计步器的个位

case(one\_step)

4'd0:digit\_seg0<=8'b11111100;

4'd1:digit\_seg0<=8'b01100000;

4'd2:digit\_seg0<=8'b11011010;

4'd3:digit\_seg0<=8'b11110010;

4'd4:digit\_seg0<=8'b01100110;

4'd5:digit\_seg0<=8'b10110110;

4'd6:digit\_seg0<=8'b10111110;

4'd7:digit\_seg0<=8'b11100000;

4'd8:digit\_seg0<=8'b11111110;

4'd9:digit\_seg0<=8'b11110110;

endcase

end

if(row\_num==7) begin //扫描到7号数码管，显示计步器的十位

case(ten\_step)

4'd0:digit\_seg0<=8'b11111100;

4'd1:digit\_seg0<=8'b01100000;

4'd2:digit\_seg0<=8'b11011010;

4'd3:digit\_seg0<=8'b11110010;

4'd4:digit\_seg0<=8'b01100110;

4'd5:digit\_seg0<=8'b10110110;

4'd6:digit\_seg0<=8'b10111110;

4'd7:digit\_seg0<=8'b11100000;

4'd8:digit\_seg0<=8'b11111110;

4'd9:digit\_seg0<=8'b11110110;

endcase

end

end

end

endmodule

//倒计时模块

module countdown(clk,but,one,ten,time1,suc);

input but; //复位按键

input clk;

input suc; //小绿点是否到达终点的信息

output [3:0] ten;

output [3:0] one;

output reg[4:0] time1=5'd30;

reg [31:0] time\_count=32'd0;

always @(posedge div\_clk or posedge but) begin

if(but==1) time1<=5'd30; //按下复位键，重新倒计时

else time1<=time1-1;

end

reg div\_clk;

always @(posedge clk)

begin

if(time1!=0&&suc==0) begin

if(time\_count==25000000)

begin

time\_count<=32'd0;

div\_clk<=~div\_clk;

end

else begin

time\_count<=time\_count+1;

end

end

end

two\_ten t1(.clk(clk),.num(time1),.ten(ten),.one(one));

endmodule

//二进制转十进制模块

module two\_ten(clk,num,ten,one);

input [4:0] num; //待转的二进制数

input clk;

output reg [3:0] ten; //转化后的十位

output reg [3:0] one; //转化后的个位

integer i;

always @(posedge clk)

begin

ten=4'd0;

one=4'd0;

for(i=4;i>=0;i=i-1)

begin

if(ten>4)

ten=ten+4'd3;

if(one>4)

one=one+4'd3;

ten=ten<<1;

ten[0]=one[3];

one=one<<1;

one[0]=num[i];

end

end

endmodule

//多个按键消抖模块

module debounce (clk,rst,key[0],key[1],key[2],key[3],key\_pulse[0],key\_pulse[1],key\_pulse[2],key\_pulse[3]);

parameter N = 4; //要消除的按键的数量

input clk;

input rst;

input [N-1:0] key; //输入的按键

output [N-1:0] key\_pulse; //按键动作产生的脉冲

reg [N-1:0] key\_rst\_pre; //定义一个寄存器型变量存储上一个触发时的按键值

reg [N-1:0] key\_rst; //定义一个寄存器变量储存储当前时刻触发的按键值

wire [N-1:0] key\_edge; //检测到按键由高到低变化是产生一个高脉冲

//利用非阻塞赋值特点，将两个时钟触发时按键状态存储在两个寄存器变量中

always @(posedge clk or negedge rst)

begin

if (!rst) begin

key\_rst <= {N{1'b1}}; //初始化时给key\_rst赋值全为1，{}中表示N个1

key\_rst\_pre <= {N{1'b1}};

end

else begin

key\_rst <= key; //第一个时钟上升沿触发之后key的值赋给key\_rst,同时key\_rst的值赋给key\_rst\_pre

key\_rst\_pre <= key\_rst; //非阻塞赋值。相当于经过两个时钟触发，key\_rst存储的是当前时刻key的值，key\_rst\_pre存储的是前一个时钟的key的值

end

end

assign key\_edge = key\_rst\_pre & (~key\_rst);//脉冲边沿检测。当key检测到下降沿时，key\_edge产生一个时钟周期的高电平

reg [19:0] cnt; //产生延时所用的计数器，系统时钟12MHz，要延时20ms左右时间，至少需要18位计数器

//产生20ms延时，当检测到key\_edge有效是计数器清零开始计数

always @(posedge clk or negedge rst)

begin

if(!rst)

cnt <= 20'h0;

else if(key\_edge)

cnt <= 20'h0;

else

cnt <= cnt + 1'h1;

end

reg [N-1:0] key\_sec\_pre; //延时后检测电平寄存器变量

reg [N-1:0] key\_sec;

//延时后检测key，如果按键状态变低产生一个时钟的高脉冲。如果按键状态是高的话说明按键无效

always @(posedge clk or negedge rst)

begin

if (!rst)

key\_sec <= {N{1'b1}};

else if (cnt==20'hf\_ffff)

key\_sec <= key;

end

always @(posedge clk or negedge rst)

begin

if (!rst)

key\_sec\_pre <= {N{1'b1}};

else

key\_sec\_pre <= key\_sec;

end

assign key\_pulse = key\_sec\_pre & (~key\_sec);

endmodule

//单个按键消抖模块

module sw(clk,key\_in,key\_out);

input key\_in;

input clk;

output reg key\_out;

reg [19:0]tcnt1;

always @(posedge clk or negedge key\_in)

begin

if(!key\_in)tcnt1<=20'd0;

else tcnt1<=tcnt1+1;

end

always @(posedge clk or negedge key\_in)

begin

if(!key\_in)key\_out<=0;

else if(tcnt1==20'hfffff)key\_out<=key\_in;

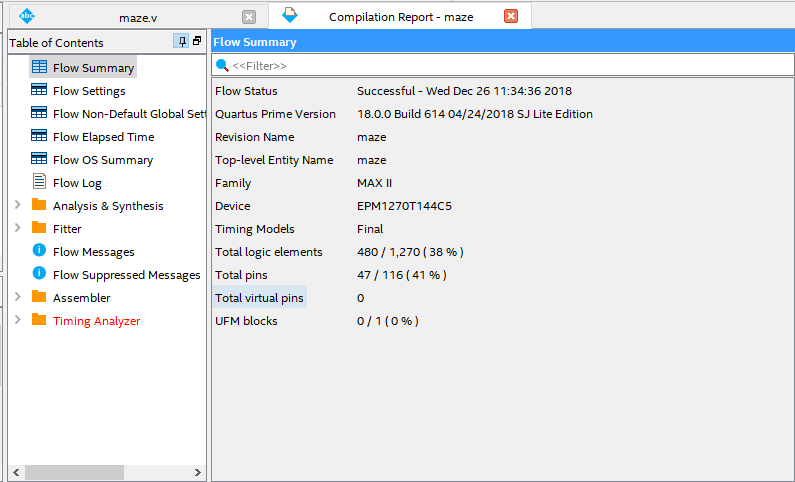
end

endmodule

**五、功能说明**

本迷宫实现了所有的基本功能，包括地图的点阵显示、BTN0~BTN3四个按键控制人物进行上下左右移动、普通计时模式功能、人物行走过程中碰壁不动等功能。在此基础之上，本迷宫还添加了切换地图功能，制作了成功和失败的界面，增加了小绿点的计步功能（碰到墙壁上步数不计）。

**六、资源利用情况**

****

**七、故障及问题分析**

（1）切换地图问题

由于地图的信息的过大，如果针对地图的两种情况分别进行考虑和编写代码，无疑会大幅度增加工作量和代码的冗余度，而且增加逻辑上出错的可能性。

解决办法：将两张地图分别存于map\_row1、map\_row2变量当中，当选择不同的地图时，将对应的值传给map\_row变量当中，对其他的诸如起始点信息和终点信息也是如此。在接下来的代码对map\_row进行考虑和编码就可以了。

（2）BTN7作为开始键的问题

由于BTN7键为按键，只有当按下后才处于高电平。而当代码烧录完成后，游戏立即开始，若用BTN7的高电平来控制游戏的开始只有在长时间按住按键才可实现，否则无法完成。

解决方法：创建另一个变量start\_con，使其初始值为0，当按下start键后，start\_con键赋1，则可根据start\_con的状态值来达到控制游戏的开始。

（3）游戏的结束问题

当小绿点到达终点或者是倒计时结束（游戏失败）时，点阵应当显示不同的图案，而游戏成功是以小绿点是否到达终点作为参考信息，游戏失败是以倒计时结束为标志。在一开始运行的时候，游戏成功后，倒计时仍然处于进行当中，当倒计时结束后，点阵显示游戏失败的图案。游戏出现逻辑错误。

解决方法：在点阵显示模块中，创建变量suc，用来监视小绿点是否到达终点。将suc变量传至倒计时模块，只有爱suc=0时，倒计时正常进行，而当suc=1时，倒计时暂停。

（4）判断碰壁的问题

通过上下左右按键来改变小绿点的位置信息较为容易实现，而开始的时候，因为迷宫的信息是根据动态扫描到相应的行数而给r\_col赋上相应的值，这种情况下，很难判断出是否可以移动小绿点。

解决办法：将地图的位置信息存储到memory型数组中，可以根据将要移动到的位置信息是否为0来判断是否可以移动。

（5）频率选择问题

由于点阵和数码管的显示是通过动态扫描的方式进行的，所以这其中涉及到频率的选择问题。实验的初始阶段，由于频率选择过大的问题，点阵和数码管出现明显的显示问题，除此之外，亮度方面也出现问题。

解决方法：之前数字电路的随堂实验中涉及到数码管的动态扫描，根据之前所使用的1kHz的频率，对本次实验的频率做出相应的修改后，数码管和点阵的显示正常。

（6）按键消抖问题

在按下按键后，会出现短时间的按键抖动问题，会严重的影响到输出。

解决办法：当迎来按键的上升沿之后，经过短时间后再次检验按键的状态值，将此时按键的状态值赋值给输出变量，借此达到按键消抖的目的。

**八、总结和结论**

本次实验是对我之前所学知识的一种检验，在巩固了知识的同时，又发现了自己的不足。通过此次实验，我的实践动手能力有了很大的提高。

本次实验历时四周的时间。从当初的迷茫，到逐渐梳理结构，划分功能模块，编写代码，到最后的调试，debug，运行成功，在这个过程中，我对使用Verilog编程的理解愈加深刻。在此之前，我对Verilog的使用仅局限于实现单独的某项功能，并没有划分功能模块的编程思想。而本次实验所实现的功能略显复杂，如若不进行模块划分，则代码显得十分冗余，且结构复杂，不易修改。模块化之后，整体逻辑显得清晰许多，便于修改。实验当中遇到了许多的问题，调试的过程是枯燥而困难的，但也正是在这个过程中，使我逐渐掌握了Verilog，对其有了更进一步的认识。

总体而言，数电实验这门课程具有十分重大的意义，在本科阶段，这门课程的开设无疑很大程度上提高学生的综合能力。而在之后的实验中，我也会充分利用在这门课程中学习到的方法，包括使用搭建框架，划分模块等方法，逐步实现和解决不断出现的问题，进而形成良好的编程习惯，进一步加强对Verilog语言的了解。