同济大学计算机系

数字逻辑课程综合实验报告

学　　号 215

姓　　名 xx

专　　业 计算机科学与技术

授课老师 张冬冬

目录

一、实验内容

1. 项目内容梗概

基于 NEXYS4 FPGA、VS1003B，VGA，加速度传感器的多模式旋转音游小游戏

2. 输入输出

输入：开发板上的按键，拨码开关，整体器件旋转方位
输出：开发板上的 LED 灯，七段数码管，MP3，VGA 显示屏

3. 操作说明

刚开始为复位状态，打开拨码复位开关结束复位，vga 出现画面，MP3 开始播放音乐。MP3 播放音乐部分可以根据拨码开关来控制音量，选择音乐等，音量分为 4 档，音乐有三首。还可以通过拨码开关选择游戏模式。在模式 1 下，可以控制判定区旋转，判定区的标志将会根据板子角度变换位置，当小球到达判定区时，按下按键开关，则小球变大变白，说明成功选中，此时计分器加 1 分。在模式 2 下，可以将开发板和 vga 结合，在一起旋转，此时判定区不变，只有小球轨道根据 vga 和板子的旋转角度发生变化，若小球进入判定区，则计分器加 1 分。

在 vga 游戏画面进行的同时，开发板上的七段数码管从前到后依次显示第几首音乐、计分、播放音乐所用时间。播放完一首音乐或者音乐切换或者拨下复位开关，则数码管重新从 0 计时，游戏重新开始。

4. 实验工具及环境

4.1 实验工具：

Nexys 4 DDR Artix-7——由 Xilinx 公司开发出的一款现场可编程 门阵列（FPGA） 开发板；

VS1003B MP3——MP3 模块，其协议为 SPI 协议模式 0，实现音乐播放；

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；

ADXL362——板载加速度传感器，其协议为 SPI 协议模式 0，可以感知三轴上的重力加速度，进而获取角度值。

4.1 实验工具：

Vivado——FPGA 公司 Xilinx 开发的一种集成设计环境

Windows11——笔者所用的笔记本电脑的操作系统

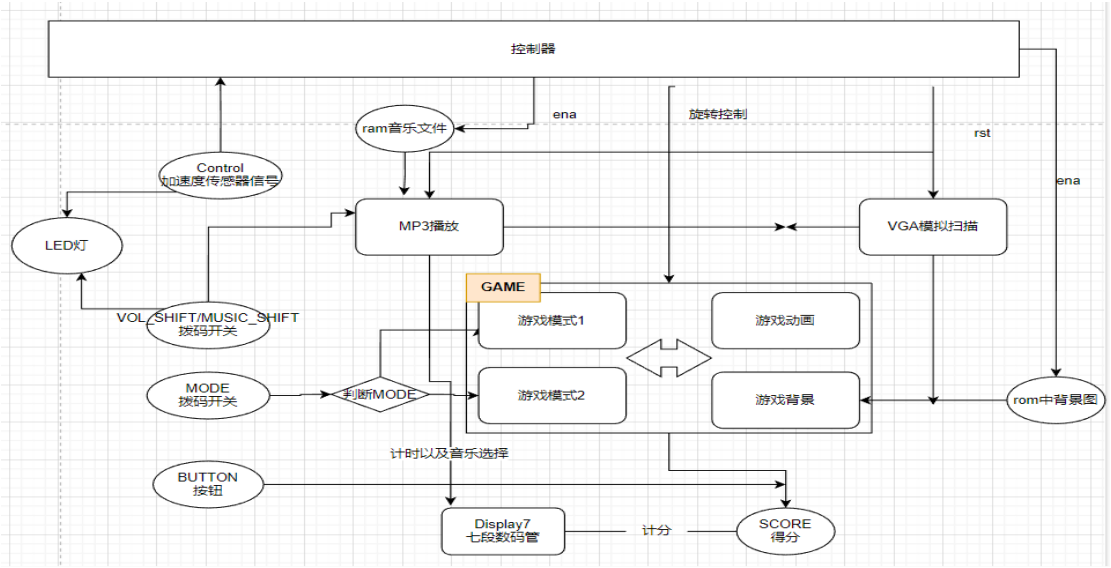
Modelsim———一款集成仿真软件

二、多模式旋转音游数字系统总框图

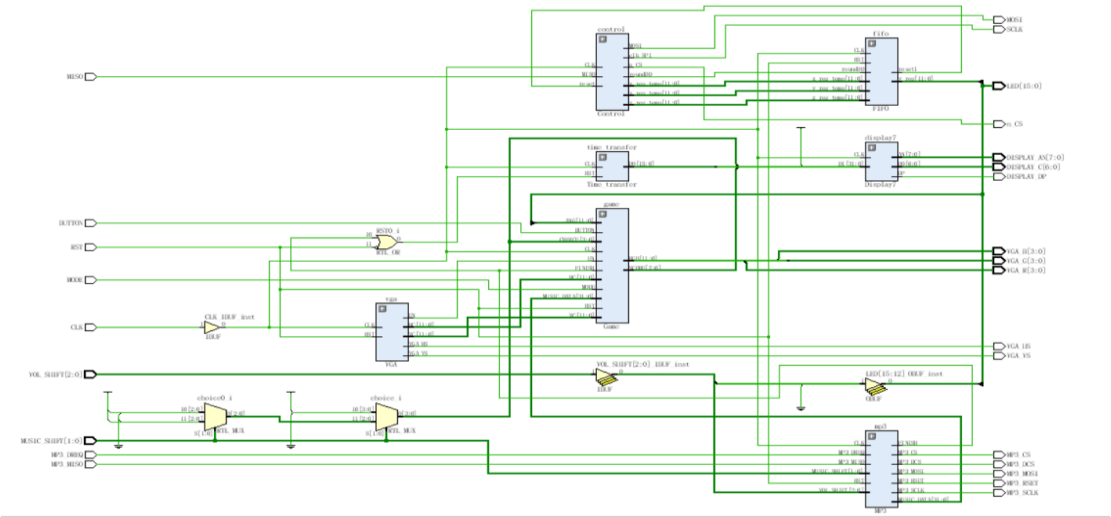
要实现的功能如下：

1. 实现 VGA 显示
2. 实现 MP3 音乐播放，音乐切换，音量调整
3. 实现对加速度传感器数据的读取
4. 实现基于七段数码管的数字钟计时，计分，以及音乐显示
5. 实现游戏整体功能
6. 实现基于 ROM IP 核的背景图片显示和 RAM IP 核的音乐存储
7. 实现 led 灯显示音量以及加速度传感器数据
8. 实现由音乐数据控制动画
9. 实现由加速度传感器数据控制动画

框图设计如下：

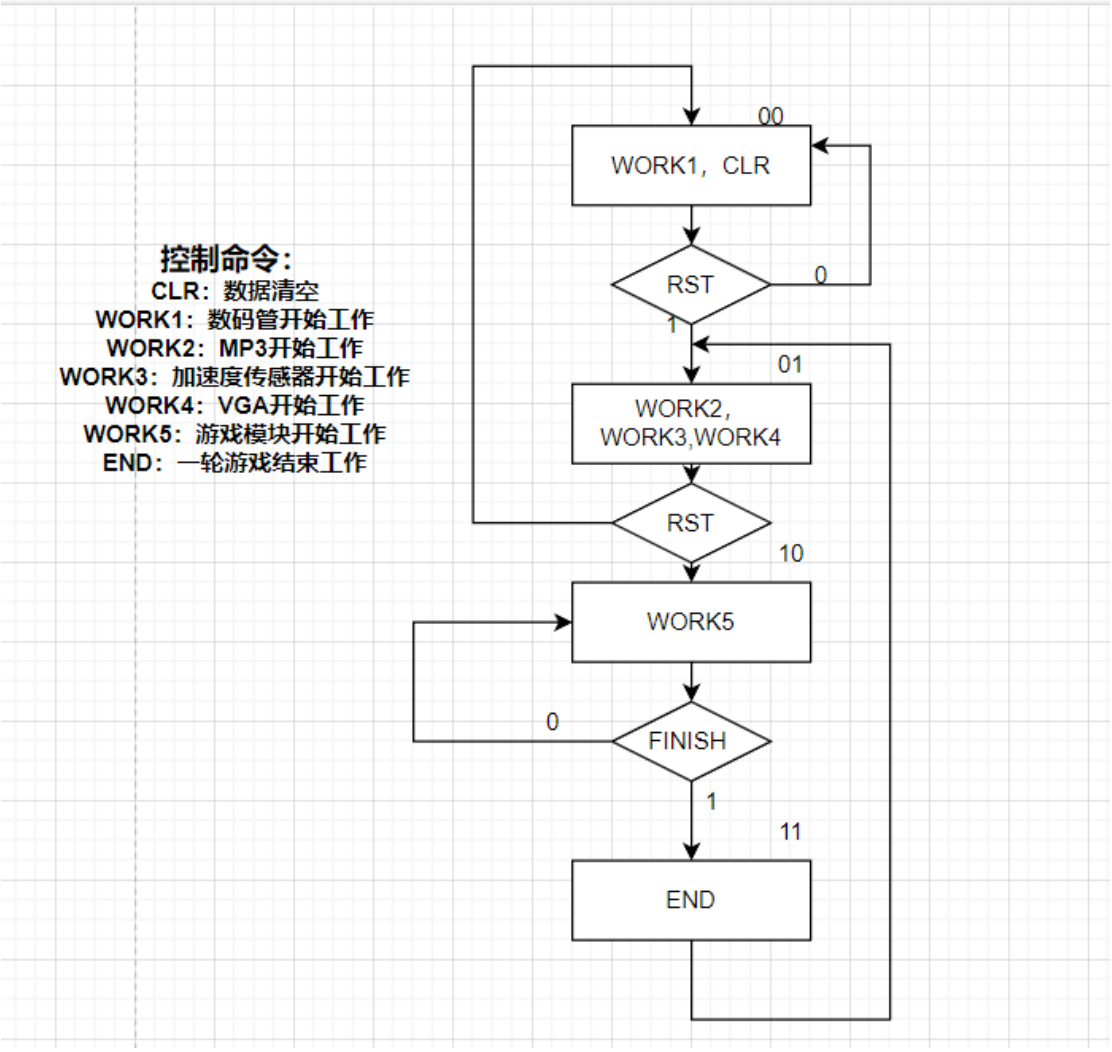


RTL 图如下：



三、系统控制器设计

1.ASM 流程图



2.状态转移真值表

PS	NS	转换条件
00	00	RST==0
00	01	RST==1
01	00	RST==0
01	10	RST==1
10	10	FINISH==0
10	11	FINISH==1
11	01	/

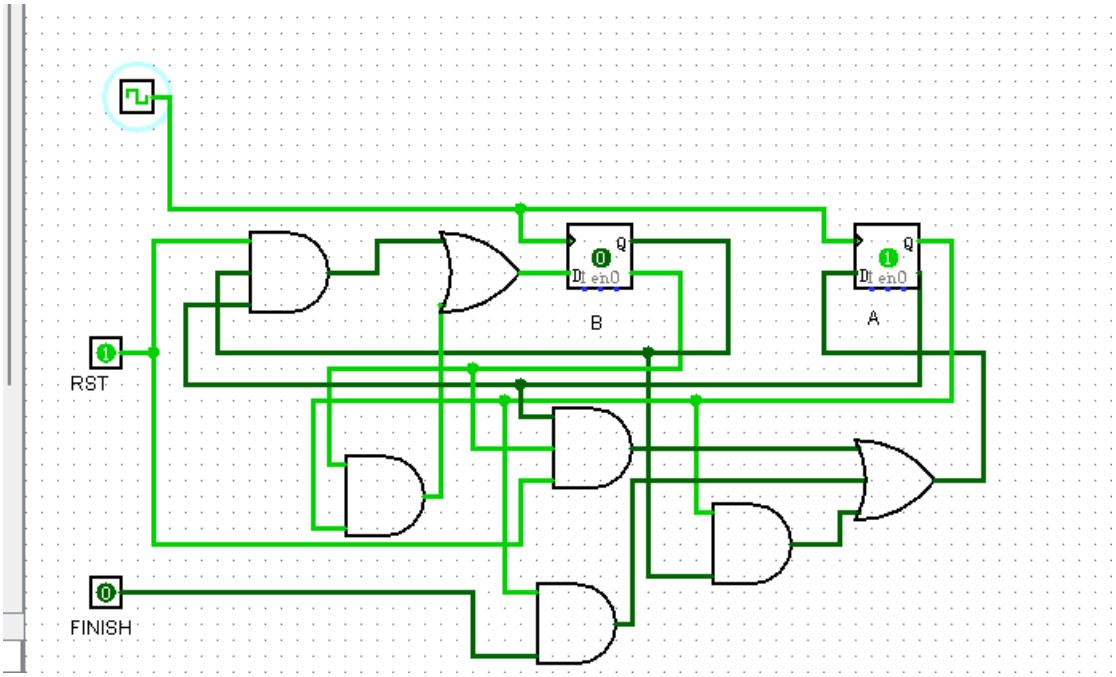
3.次态函数激励表达式

令编码首位为 B,末位为 A，得
B(D)=~A・B・RST+A・~B
A(D)=~A・~B・RST+A・FINISH+A・B

4.控制命令逻辑表达式

CLR=WORK1=~A・~B
WORK=WORK3=WORK4=~A・B
WORK5=A・~B
END=A・B

5.系统控制器逻辑方案图



四、子系统模块建模

1.top 顶层模块

功能：连接各个子模块，建立系统与外部的信号关联
接口信号定义：

input CLK,//时钟信号
input RST,//复位信号
input BUTTON,//按钮信号
output　MP3_RSET,//MP3 复位信号
output　MP3_CS,//MP3，spi 通信的命令传输有效信号，低电平有效

output MP3_DCS,//MP3，spi 通信的数据传输有效信号，低电平有效
output MP3_MOSI,//MP3，spi 通信的 master 输出端
input MP3_MISO,//MP3，spi 通信的 master 输入端
output MP3_SCLK,//MP3，spi 通信的时钟
input MP3_DREQ,//数据发送有效信号

output [3:0]VGA_R,//vga 的 r 颜色分量，下同理
output [3:0]VGA_G,
output [3:0]VGA_B,
output VGA_HS,//行扫描有效信号
output VGA_VS,//场扫描有效信号

input MISO,//加速度传感器的 spi 通信的 master 输入端，以下同 MP3
output MOSI,
output n_CS,
output SCLK,

output DISPLAY_DP,//数码管点
output [7:0] DISPLAY_AN,//数码管位选
output [6:0] DISPLAY_C,//七段数码管

input [2:0] VOL_SHIFT,//音量控制对应的拨码开关
input [1:0] MUSIC_SHIFT,//音乐切换对应的拨码开关
input MODE,//游戏模式选择对应的拨码开关

output[15:0] LED//led 灯

代码：

```
module top(
    input CLK,//时钟信号
    input RST,//复位信号
    input BUTTON,//按钮信号

    output  MP3_RSET,//MP3 复位信号
    output  MP3_CS,//MP3，spi 通信的命令传输有效信号，低电平有效
    output MP3_DCS,//MP3，spi 通信的数据传输有效信号，低电平有效
    output MP3_MOSI,//MP3，spi 通信的 master 输出端
    input MP3_MISO,//MP3，spi 通信的 master 输入端
    output MP3_SCLK,//MP3，spi 通信的时钟
    input MP3_DREQ,//数据发送有效信号

    output [3:0]VGA_R,//vga 的 r 颜色分量，下同理
    output [3:0]VGA_G,
    output [3:0]VGA_B,
    output VGA_HS,//行扫描有效信号
```

```
        output VGA_VS,//场扫描有效信号
        input   MISO,//加速度传感器的 spi 通信的 master 输入端，以下同 MP3
        output   MOSI,
        output n_CS,
        output SCLK,

        output DISPLAY_DP,//数码管点
        output [7:0] DISPLAY_AN,//数码管位选
        output [6:0] DISPLAY_C,//七段数码管

        input [2:0] VOL_SHIFT,//音量控制对应的拨码开关
        input [1:0] MUSIC_SHIFT,//音乐切换对应的拨码开关
        input MODE,//游戏模式选择对应的拨码开关

        output[15:0] LED//led 灯
        );
        wire FINISH;//一首音乐是否播放完毕
        wire [3:0] choice;//音乐选择
        wire [11:0] HC,VC;//行计数和场计数
        wire EN;//vga 扫描是否在有效范围内
        wire[31:0] MUSIC_DATA;//根据向 MP3 传送数据，来产生伪随机数
        wire [7:0] SCORE;//积分
        wire roundDD;//加速度传感器是否已经接收到了一轮数据
        wire [11:0] x_reg_temp, y_reg_temp, z_reg_temp;//加速度传感器下，x,y,z 方向的临时
数据
        wire reset1;//fifo 模块的延时，加速度传感器的复位信号
        wire [11:0]y_reg;//只需要 y 方向上的数据
        wire [15:0] ANG;//加速度衍生的角度相关值

        assign choice =( MUSIC_SHIFT==2'b10 ? 4'h2 :(MUSIC_SHIFT==2'b01 ? 4'h3 :
4'h1));

        assign LED[15:0]={VOL_SHIFT[2:0],1'b0,y_reg[11:0]};

        MP3
mp3(CLK,RST,MP3_RSET,MP3_CS,MP3_DCS,MP3_MOSI,MP3_MISO,MP3_SCLK,MP3_DR
EQ,VOL_SHIFT,MUSIC_SHIFT,FINISH,MUSIC_DATA);
       // Gyro gyro( CLK, RST,GYRO_SPC,GYRO_SDI,GYRO_SDO,GYRO_CS,ANG);
        VGA vga(CLK,RST,VGA_HS,VGA_VS,HC,VC,EN);//模拟 vga 扫描信号
        //游戏控制总体模块
        Game
game(CLK,RST,EN,FINISH,y_reg,HC,VC,BUTTON,SCORE,{VGA_R[3:0],VGA_G[3:0],VGA
_B[3:0]},MUSIC_DATA,CHOICE,MODE);
        //加速度传感器模拟队列
```

FIFO fifo(CLK, roundDD, RST,x_reg_temp, y_reg_temp, z_reg_temp,x_reg, y_reg, z_reg,reset1);

//加速度传感器

Control control(CLK,reset1, MISO,MOSI, n_CS, roundDD,x_reg_temp, y_reg_temp, z_reg_temp,SCLK);

//数码管显示

wire [15:0] time_cnt;

//时间转换和七段数码管

Time_transfer time_transfer(CLK,FINISH | (~RST),time_cnt);

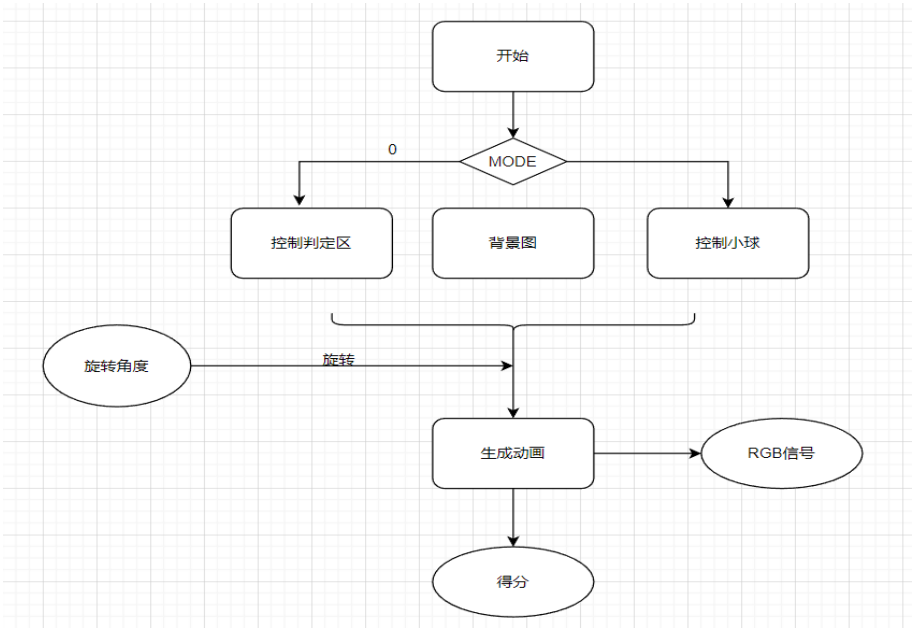
Display7 display7(CLK,{choice[3:0],4'b1111,SCORE[7:0],time_cnt[15:0]},DISPLAY_C,DISPLAY_AN,DISPLAY_DP);

endmodule

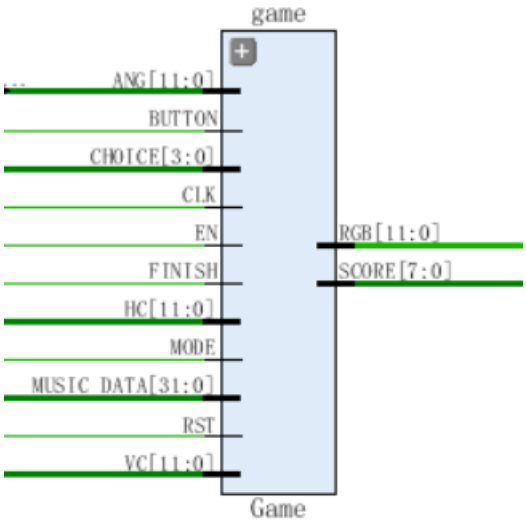
2.Game 游戏控制模块

功能：作为游戏主体，实现游戏相关功能

框图：



接口信号定义：



input CLK, //时钟

input RST, //复位信号

input EN,//vga 扫描有效信号

input FINISH, //音乐播放完成或者切换信号

input [11:0] ANG,//角度传感器

input [11:0] HC,//行计数

input [11:0] VC,//场计数

input BUTTON,//计分按钮

output [7:0] SCORE, //得分

```
    output reg [11:0] RGB,
    input[31:0] MUSIC_DATA,//音乐数据
    input[3:0] CHOICE,//选择的音乐
    input MODE//游戏模式
```

代码：

```
module Game(
    input     CLK,
    input     RST,
    input    EN,//有效信号
    input FINISH,
    input [11:0] ANG,//角度
    input    [11:0] HC,
    input [11:0] VC,
    input BUTTON,
    output [7:0] SCORE,          //得分
    output reg [11:0] RGB,
    input[31:0] MUSIC_DATA,
    input[3:0] CHOICE,
    input MODE
    );

   //画面坐标数据和颜色参数
    parameter center_x=512,
              center_y=384;
    parameter cycle_r=450;
    parameter angle1=12'b0001_1000_0000,

angle2=12'b0010_1000_0000;
    parameter red=12'hf_0_0,
              orange=12'hf_a_0,
              cyan=12'h0_f_f,
              green=12'h0_f_0,
              yellow=12'hf_f_0,
              blue=12'h0_0_f,
              purple=12'ha_2_f,
              black=12'h0_0_0,
              white=12'hf_f_f,
              cray=12'hb_b_b;
wire clk;
wire locked;
clk_wiz_0 clk_65hz(
          .reset(~RST),
          .locked(locked),
          .clk_in1(CLK),
          .clk_out1(clk)
    );

    reg [18:0] addra = 0;          //背景图片存储 ROM 地址
    wire [11:0] douta;
//ROM 输出
    integer hc;
    integer vc;

    background
image(.clka(clk),.addra(addra),.douta(douta));


    reg[9:0] px_ball,py_ball;//小球位置
    reg[9:0] r_ball;//小球半径
    reg[11:0] c_ball;//小球颜色
    reg[2:0] final_path;

    //画图和画圆
   always@(posedge clk or negedge RST or
posedge FINISH)
    if(~RST || FINISH ) begin
        RGB<=black;
    end
    else begin
        if(EN) begin
            hc<=HC;
            vc<=VC;

if((hc-px_ball)*(hc-px_ball)+(vc-py_ball)*(vc
-py_ball)<r_ball*r_ball
                ||
(hc+px_ball-2*center_x)*(hc+px_ball-2*cente
r_x)+(vc+py_ball-2*center_y)*(vc+py_ball-2
*center_y)<r_ball*r_ball)begin   //一对小球
                RGB<=c_ball;
```

```
                end
                else
if((hc-center_x)*(hc-center_x)+(vc-center_y)*
(vc-center_y)<=cycle_r*cycle_r+3600 &&

(hc-center_x)*(hc-center_x)+(vc-center_y)*(v
c-center_y)>=cycle_r*cycle_r-3600) begin//识
别区

if(MODE==1'b0)begin

if(ANG[11]==1 && ~ANG>angle2) begin
                                        if
(vc>=center_y+325 && hc>center_x
||vc<=center_y-325 && hc<center_x )

RGB<=red;
                                        else

RGB<=white;

final_path<=1;
                                end
                                else
if(ANG[11]==1 && ~ANG>angle1 &&
~ANG<=angle2)begin

if( (vc<=center_y+325 && vc>=center_y+125
&& hc>center_x || vc>=center_y-325 &&
vc<=center_y-125 && hc<center_x ))

RGB<=red;
                                        else

RGB<=white;

final_path<=2;
                                end
                                else
if(ANG[11]==0 && ANG>angle2) begin
                                        if
(vc>=center_y+325 && hc<center_x
||vc<=center_y-325 && hc>center_x )

RGB<=red;
                                        else

RGB<=white;

final_path<=5;
                                end
                                else
if(ANG[11]==0 && ANG>angle1 &&
ANG<=angle2)begin

if( (vc<=center_y+325 && vc>=center_y+125
&& hc<center_x || vc>=center_y-325 &&
vc<=center_y-125 && hc>center_x ))

RGB<=red;
                                        else

RGB<=white;

final_path<=4;
                                end
                                else begin

if(vc>=center_y-100 && vc<=center_y+100)

RGB<=red;
                                        else

RGB<=white;

final_path<=3;
                                end
                        end
                        else begin

if(vc>=center_y-100 && vc<=center_y+100)

RGB<=red;
                                else

RGB<=white;
                                end
                        end
```

```
                else  if(hc<832  &&  hc>=192
&& vc>=144 && vc<624) begin   //背景

addra<=(vc-144)*640+hc-192;

RGB<={douta[11:8],douta[7:4],douta[3:0]};
                    end
                else begin
                    RGB<=black;
                end
            end
            else begin
                RGB<=black;
            end
        end

        wire clk_ball;
        Divider
#(100)divider_ball(CLK,clk_ball); //1MHz


 parameter     beat0=1000_000,     //由于是
120bpm，1 秒两拍
                  beat1=1200_000,
//100bpm
                  beat2=750_000;
//160bpm
//状态
 parameter   SET=0,
                PATH1=1,
                PATH2=2,
                PATH3=3,
                PATH4=4,
                PATH5=5,
                GET_SCORE=6,
                OVER=7;
parameter angle3=12'b0010_0000_0000,
            angle4=12'b0011_1000_0000;

   reg[31:0] rand;
   integer cnt=0;
   integer beat;
   reg[3:0]change=0;
   reg[1:0] path;
   reg[2:0]   state;
   integer sco;

   //使小球以及转动
   always @(posedge clk_ball or negedge RST
or posedge FINISH) begin
      if(~RST || FINISH)begin
          cnt<=0;
          state<=SET;
          sco<=0;
      end
      else begin
      case(state)
          SET:begin
              change<=0;
              cnt<=0;
          case(CHOICE)
          2'b01:
              beat<=beat0;
          2'b10:
              beat<=beat1;
          2'b11:
              beat<=beat2;
          default:
              beat<=beat0;
          endcase

          rand<=MUSIC_DATA%60;// 由 音
乐数据产生随机轨道和颜色
          if(rand<10)begin
//颜色,轨道
              c_ball<=red;
              state<=PATH1;
          end
          else if(rand>10&& rand<20) begin
              c_ball<=orange;
                state<=PATH2;
          end
          else if(rand>=20&&rand <30)begin
              c_ball<=yellow;
                state<=PATH3;
          end
          else if(rand>=30&&rand <40)begin
```

```
                c_ball<=green;
                  state<=PATH3;
            end
            else if(rand>=40&&rand <50)begin
                c_ball<=blue;
                  state<=PATH4;
            end
            else begin
                c_ball<=purple;
                  state<=PATH5;
            end
            end
            PATH1: begin
          if((change==0 && cnt>=beat/6) || (change==1 && cnt>=2*beat/6) ||(change==2 && cnt>=3*beat/6)||(change==3 && cnt>=4*beat/6))begin
          if(MODE==1 && ANG[11]==1 && ~ANG>angle3 && ~ANG<=angle4 )begin
                change<=change+1;
                state<=PATH2;
            end
            else if(MODE==1 && ANG[11]==1 && ~ANG>angle4)begin
                change<=change+1;
                state<=PATH3;
            end
          end
            if(cnt<beat/6)begin
                px_ball<=center_x;
                py_ball<=center_y;
                r_ball<=10;
            end
            else                if(cnt>=beat/6&& cnt<2*beat/6)begin
                px_ball<=center_x+45;
                py_ball<=center_y+77;
                r_ball<=20;
            end
            else                if(cnt>=2*beat/6&& cnt<3*beat/6)begin
                px_ball<=center_x+90;
                py_ball<=center_y+154;
                r_ball<=30;
            end
            else                if(cnt>=3*beat/6&& cnt<4*beat/6)begin
                px_ball<=center_x+135;
                py_ball<=center_y+231;
                r_ball<=45;
            end
            else                if(cnt>=4*beat/6&& cnt<5*beat/6)begin
                px_ball<=center_x+180;
                py_ball<=center_y+308;
                r_ball<=60;
            end
            else begin
                  px_ball<=center_x+225;
                py_ball<=center_y+385;
                r_ball<=75;

if(MODE==0&&final_path==1)
                    state<=GET_SCORE;
              else
                  state<=OVER;
            end
            cnt<=cnt+1;
          end
            PATH2: begin
                  if((change==0                && cnt>=beat/6) || (change==1 && cnt>=2*beat/6) ||(change==2 && cnt>=3*beat/6)||(change==3 && cnt>=4*beat/6))begin
          if(MODE==1 && ANG[11]==1 && ~ANG>angle3 && ~ANG<=angle4)begin
                  change<=change+1;
                state<=PATH3;
            end
             else        if(MODE==1          && ANG[11]==0 && ANG>angle3 )begin
                change<=change+1;
                state<=PATH1;
            end
            else if(MODE==1 && ANG[11]==1 && ~ANG>angle4)begin
              change<=change+1;
                state<=PATH4;
```

```
            end
                end
            if(cnt<beat/6)begin
                px_ball<=center_x;
                py_ball<=center_y;
                r_ball<=10;
            end
            else                if(cnt>=beat/6&&
cnt<2*beat/6)begin
                px_ball<=center_x+77;
                py_ball<=center_y+45;
                r_ball<=20;
            end
            else                if(cnt>=2*beat/6&&
cnt<3*beat/6)begin
                px_ball<=center_x+154;
                py_ball<=center_y+90;
                r_ball<=30;
            end
            else                if(cnt>=3*beat/6&&
cnt<4*beat/6)begin
                px_ball<=center_x+231;
                py_ball<=center_y+135;
                r_ball<=45;
            end
            else                if(cnt>=4*beat/6&&
cnt<5*beat/6)begin
                px_ball<=center_x+308;
                py_ball<=center_y+180;
                r_ball<=60;
            end
            else    begin
                px_ball<=center_x+385;
                py_ball<=center_y+225;
                r_ball<=75;

if(MODE==0&&final_path==2)
                    state<=GET_SCORE;
                else
                    state<=OVER;
            end
            cnt<=cnt+1;
           end
            PATH3: begin
            if(cnt<beat/6)begin
                px_ball<=center_x;
                py_ball<=center_y;
                r_ball<=10;
            end
            else                if(cnt>=beat/6&&
cnt<2*beat/6)begin
                px_ball<=center_x+90;
                py_ball<=center_y;
                r_ball<=20;
            end
            else                if(cnt>=2*beat/6&&
cnt<3*beat/6)begin
                px_ball<=center_x+180;
               py_ball<=center_y;
                r_ball<=30;
            end
            else                if(cnt>=3*beat/6&&
cnt<4*beat/6)begin
                px_ball<=center_x+270;
                  py_ball<=center_y;
                r_ball<=45;
            end
            else                if(cnt>=4*beat/6&&
cnt<5*beat/6)begin
                px_ball<=center_x+360;
                py_ball<=center_y;
                r_ball<=60;
            end
            else    begin
                  px_ball<=center_x+450;
               py_ball<=center_y;
                r_ball<=75;

if((MODE==0&&final_path==3)                ||
MODE==1)
                    state<=GET_SCORE;
                else
                    state<=OVER;
            end
            cnt<=cnt+1;
           end
            PATH4: begin
```

```
            if((change==0 && cnt>=beat/6) ||
(change==1 && cnt>=2*beat/6) ||(change==2
&&      cnt>=3*beat/6)||(change==3      &&
cnt>=4*beat/6))begin
            if(MODE==1 && ANG[11]==1 &&
~ANG>angle3)begin
                change<=change+1;
                 state<=PATH5;
            end
            else if(MODE==1 && ANG[11]==0
&& ANG>angle4 )begin
                change<=change+1;
                 state<=PATH2;
            end
             else      if(MODE==1      &&
ANG[11]==0  &&  ANG>angle3  &&
ANG<=angle4)begin
                change<=change+1;
                state<=PATH3;
            end
             end

            if(cnt<beat/6)begin
                px_ball<=center_x;
                py_ball<=center_y;
                r_ball<=10;
            end
            else              if(cnt>=beat/6&&
cnt<2*beat/6)begin
                px_ball<=center_x+77;
                py_ball<=center_y-45;
                r_ball<=20;
            end
            else              if(cnt>=2*beat/6&&
cnt<3*beat/6)begin
                px_ball<=center_x+154;
                py_ball<=center_y-90;
                r_ball<=30;
            end
            else              if(cnt>=3*beat/6&&
cnt<4*beat/6)begin
                px_ball<=center_x+231;
                py_ball<=center_y-135;
                r_ball<=45;
            end
            else              if(cnt>=4*beat/6&&
cnt<5*beat/6)begin
                px_ball<=center_x+308;
                py_ball<=center_y-180;
                r_ball<=60;
            end
            else begin
                 px_ball<=center_x+385;
                py_ball<=center_y-225;
                r_ball<=75;

if(MODE==0&&final_path==4)
                    state<=GET_SCORE;
                else
                    state<=OVER;
             end
             cnt<=cnt+1;
            end
            PATH5: begin
              if((change==0 && cnt>=beat/6) ||
(change==1 && cnt>=2*beat/6) ||(change==2
&&      cnt>=3*beat/6)||(change==3      &&
cnt>=4*beat/6))begin
            if(MODE==1 && ANG[11]==0 &&
ANG>angle3 && ANG<=angle4)begin
                change<=change+1;
                 state<=PATH4;
            end
            else if(MODE==1 && ANG[11]==0
&& ANG>angle4)begin
                change<=change+1;
                state<=PATH3;
            end
             end
            if(cnt<beat/6)begin
                px_ball<=center_x;
                py_ball<=center_y;
                r_ball<=10;
            end
            else              if(cnt>=beat/6&&
cnt<2*beat/6)begin
                px_ball<=center_x+45;
                py_ball<=center_y-77;
```

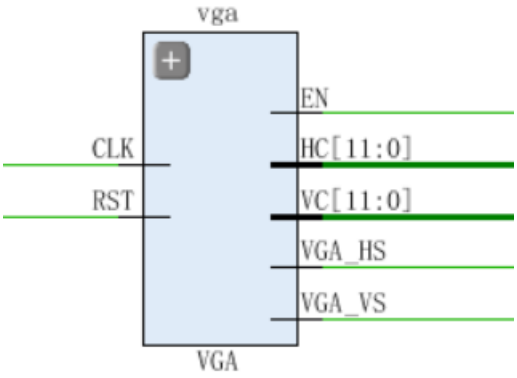
```
                r_ball<=20;
            end
            else                if(cnt>=2*beat/6&&
cnt<3*beat/6)begin
                px_ball<=center_x+90;
                py_ball<=center_y-154;
                r_ball<=30;
            end
            else                if(cnt>=3*beat/6&&
cnt<4*beat/6)begin
                px_ball<=center_x+135;
                py_ball<=center_y-231;
                r_ball<=45;
            end
            else                if(cnt>=4*beat/6&&
cnt<5*beat/6)begin
                px_ball<=center_x+180;
                py_ball<=center_y-308;
                r_ball<=60;
            end
            else begin
                  px_ball<=center_x+225;
                py_ball<=center_y-385;
                r_ball<=75;

if(MODE==0&&final_path==5)
                        state<=GET_SCORE;
                else
                        state<=OVER;
                end
                cnt<=cnt+1;
            end
            //计分
            GET_SCORE: begin
                if((MODE==0&&BUTTON)        ||
MODE==1) begin
                    sco<=sco+1;
                    r_ball<=150;
                    c_ball<=white;
                end
                state<=OVER;
            end
            OVER:begin//结束缓冲
                if(cnt<beat)
                    cnt<=cnt+1;
                else
                    state<=SET;
            end
        endcase


        end
    end
    //赋分
    assign SCORE[3:0]=sco%10;
    assign SCORE[7:4]=sco/10;
endmodule
```

3.VGA 模块

功能：产生行扫描、场扫描、扫描有效信号

接口信号定义：

input CLK;//100MHz 时钟
input RST;//复位信号
output VGA_HS; //行扫描有效信号
output VGA_VS; //场扫描有效信号
output [11:0]HC ;//行扫描计数
output[11:0] VC;//场扫描计数
output EN; //扫描有效



代码：

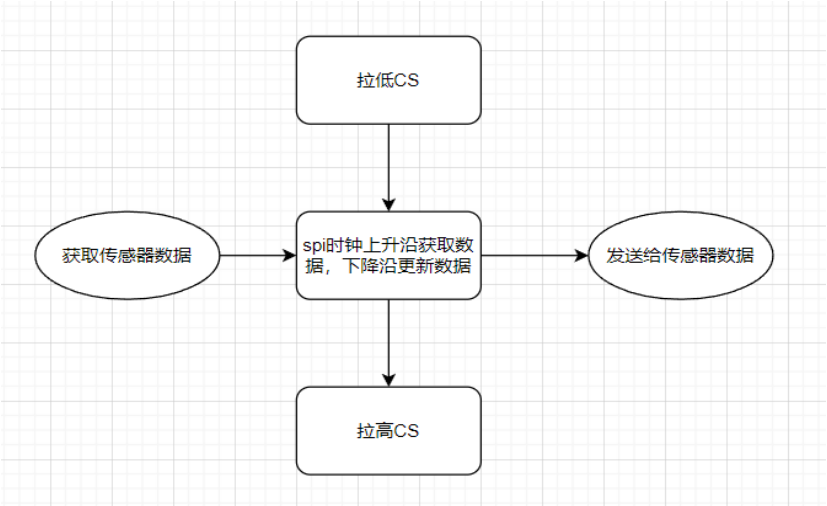
```
module VGA(                                CLK,
```

```
    RST,
    VGA_HS,
    VGA_VS,
    HC,
    VC,
    EN
    );
input CLK;
input RST;
output VGA_HS;
output VGA_VS;
output [11:0]HC ;
output[11:0] VC;
output EN;

parameter hsync_end = 12'd135,//0-136-160<--1024-->24
          hdat_begin = 12'd295,
          hdat_end = 12'd1319,
          hpixel_end = 12'd1343,

          vsync_end = 10'd5,//0-6-29<--768-->3
          vdat_begin = 10'd34,
          vdat_end = 10'd802,
          vline_end = 10'd805;

wire vga_clk;
wire locked;
//ip 核产生 1024X768 的 65Hz 时钟
clk_wiz_0 clk_65hz(
        .reset(~RST),
        .locked(locked),
        .clk_in1(CLK),
        .clk_out1(vga_clk)
    );

reg [12:0] hcount=0; //行扫描计数
reg [12:0] vcount=0; //场扫描计数

//行扫描
always @(posedge vga_clk or negedge locked)begin
if(!locked)
        hcount<=0;
else if (hcount==hpixel_end)
    hcount <= 0;
else
    hcount <= hcount + 1;
end

assign VGA_HS=(hcount<hsync_end+1) ? 1'b0:1'b1;
//场扫描
always @(posedge vga_clk or negedge locked)//场
begin
if(!locked)
        vcount<=0;
else if(vcount==vline_end)
    vcount<=1'b0;
else if(hcount==hpixel_end)
    vcount<=vcount+1;
else
     vcount<=vcount;
end

assign VGA_VS=(vcount<vsync_end+1)? 1'b0:1'b1;

assign EN = ((hcount >= hdat_begin) && (hcount < hdat_end))&& ((vcount >= vdat_begin) && (vcount < vdat_end));

assign HC=hcount-hdat_begin;
assign VC=vcount-vdat_begin;

endmodule
```

4.Control 模块

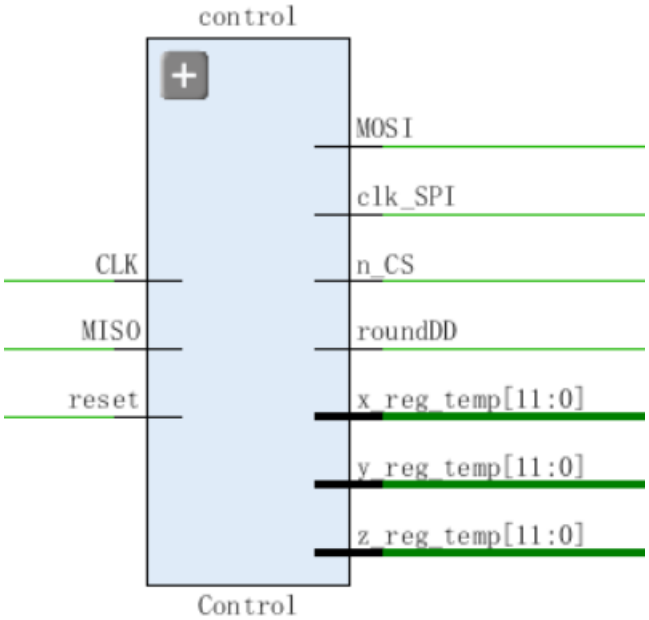
功能：与加速度传感器进行通信，获取加速度数据

框图：



接口信号定义：

input　CLK,//100MHz 时钟
　　reset,//加速度传感器复位信号
　　MISO,//spi 通信的 master 的输入
output reg MOSI,//spi 通信的 master 的输出
　　n_CS,//spi 通信的有效片选信号
　　roundDD,//读完一轮数据的有效信号
output reg [11:0] x_reg_temp, y_reg_temp, z_reg_temp,//加速度传感器三个方向上的临时数据
output clk_SPI　//spi 通信时钟



代码：

```
module Control(
    input  CLK, reset, MISO,
    output reg MOSI, n_CS, roundDD,
    output reg [11:0] x_reg_temp,
    y_reg_temp, z_reg_temp,
    output clk_SPI
```

```
    );

    Divider #(20) divider_con(CLK,clk_SPI);

    parameter
            //寄存器地址
            CON_REG = 8'h2D,
            X_L_REG = 8'h0E,
            X_H_REG = 8'h0F,
            Y_L_REG = 8'h10,
            Y_H_REG = 8'h11,
            Z_L_REG = 8'h12,
            Z_H_REG = 8'h13,

            //寄存器的读写指令
            REGISTER_READ = 8'h0B,
            REGISTER_WRITE = 8'h0A,

            //状态定义
            START = 3'd0,
            INSTRUCTION = 3'd1,
            ADDRESS = 3'd2,
            DATA_READ = 3'd3,
            DATA_WRITE = 3'd4,
            DATA_PROCESS = 3'd5,
            OVER = 3'd6,

            //读和写的数据位
            WRITE = 1'b1,
            READ = 1'b0;


    reg rw, roundDone;
    reg [7:0] address; //寄存器地址
    reg [7:0] data; //读取的数据信息
    reg [2:0] counter, state; //字节位计数和状态
    reg [7:0] instruction;


    always@(posedge clk_SPI) begin
        if(!reset) begin
            rw <= WRITE;
            x_reg_temp <= 0;
            y_reg_temp <= 0;
            z_reg_temp <= 0;
            n_CS <= 1;
            instruction <= REGISTER_WRITE;
            address <= CON_REG;
            roundDone <= 1;
            roundDD <= 1;
            state <= START;
            counter <= 7; //从 7 数到 0

            data<= 8'b0000_0010;//mesurement 模式
        end
        else begin
            roundDD <= roundDone;

            if(state == INSTRUCTION || state == ADDRESS || state == DATA_READ || state == DATA_WRITE) begin
                if(counter == 0) begin //couter 必须为 0，即数据读完才能切换状态
                    counter <= 7;
                    if(state == INSTRUCTION)
                        state <= ADDRESS;
                    else if(state == ADDRESS && rw)
                        state <= DATA_WRITE; //进入写出数据的状态
                    else if(state == DATA_WRITE || state == DATA_READ) begin
                        state <= OVER; //结束，将 CS 拉高
                    end
                    else begin
                        roundDone <= 0; //新数据来了，一个数据接受轮结束
                        state <= DATA_READ; //转换到读数据阶段
```

```
                    end
                end
                else counter <= counter - 1;
            end

             if(state == START) begin
                n_CS <= 0;
                state <= INSTRUCTION;
            end
            else if(state == DATA_PROCESS) begin//数据处理并进行地址转换
                state <= START;
                data <= 0;
                case(address) //地址字节转换，读取下一个字节
                    X_L_REG : begin
                        x_reg_temp[7:0] <= data;
                        address <= X_H_REG;
                    end
                    X_H_REG : begin
                        x_reg_temp[11:8] <= data[3:0];
                        address <= Y_L_REG;
                    end
                    Y_L_REG : begin
                        y_reg_temp[7:0] <= data;
                        address <= Y_H_REG;
                    end
                    Y_H_REG : begin
                        y_reg_temp[11:8] <= data[3:0];
                        address <= Z_L_REG;
                    end
                    Z_L_REG : begin
                        z_reg_temp[7:0] <= data;
                        address <= Z_H_REG;
                    end
                    Z_H_REG : begin
                        z_reg_temp[11:8] <= data[3:0];
                        roundDone <= 1;
                        address <= X_L_REG;
                    end
                    CON_REG : begin //控制寄存器，
                        address <= X_L_REG;
                        rw <= 0;
                        instruction <= REGISTER_READ;
                    end
                endcase
            end
            else if(state == DATA_READ) begin //spi 时钟的上升沿从 miso 捕获数据（SPI MODE0）
                data[counter] <= MISO;
            end
            else if(state == OVER) begin
                state <= DATA_PROCESS;
                n_CS <= 1;
            end
        end
    end

    always@(negedge clk_SPI) begin //spi 时钟的下降沿更新数据（SPI MODE0）
        if(!reset) begin
            MOSI <= 0;
        end
        else begin
            if(state == INSTRUCTION)//写入指令字节
                MOSI <= instruction[counter];
            else if(state == ADDRESS) //写入地址指令字节
```

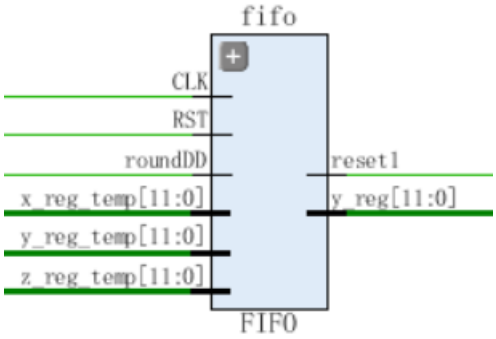
```
                MOSI <= end
address[counter];
        else if(state == DATA_WRITE)        end
//写入数据
                MOSI <= data[counter];      endmodule
```

5.FIFO 模块

功能：模拟队列处理加速度传感器的临时数据，来提高系统的工作效率

接口信号定义：

```
input CLK,//时钟
    roundDD,//control 读完一轮数据的信号
     RST,//整体复位信号
input [11:0] x_reg_temp,
          y_reg_temp,
          z_reg_temp,//x,y,z 方向的临时值
output reg [11:0] x_reg,
           y_reg//y 方向的角度数据
         z_reg
output reg reset1//control 模块的复位信号
```



代码：

```
module FIFO(
        input CLK, roundDD, RST,
        input [11:0] x_reg_temp,
y_reg_temp, z_reg_temp,
        output reg [11:0] x_reg, y_reg,
z_reg,
        output reg reset1
    );

    reg en1, en2, en3;
    reg [6:0] rstHold;



    Divider
#(2)divider_fifo(CLK,clk);//50MHZ

    always@(posedge clk) begin
            if(!RST) begin
                x_reg <= 0;
                y_reg <= 0;
                z_reg <= 0;
                en1 <= 0;
                en2 <= 0;
                en3 <= 0;
                reset1 <= 0;
                rstHold <= 0;
            end
            else begin
                en1 <= roundDD;
                en2 <= en1;
                en3 <= en2;

                //延迟一段时间来使三轴
工作
                if(rstHold == 63)
                    reset1 <= 1;
                else if(!reset1)
                    rstHold <= rstHold
+ 1;
                else reset1 <= reset1;

                if(en3) begin //三轮结束，
数据更新，避免数据的频繁访问
                    x_reg <=
x_reg_temp;
                    y_reg <=
y_reg_temp;
                    z_reg <=
```

z_reg_temp;　　end

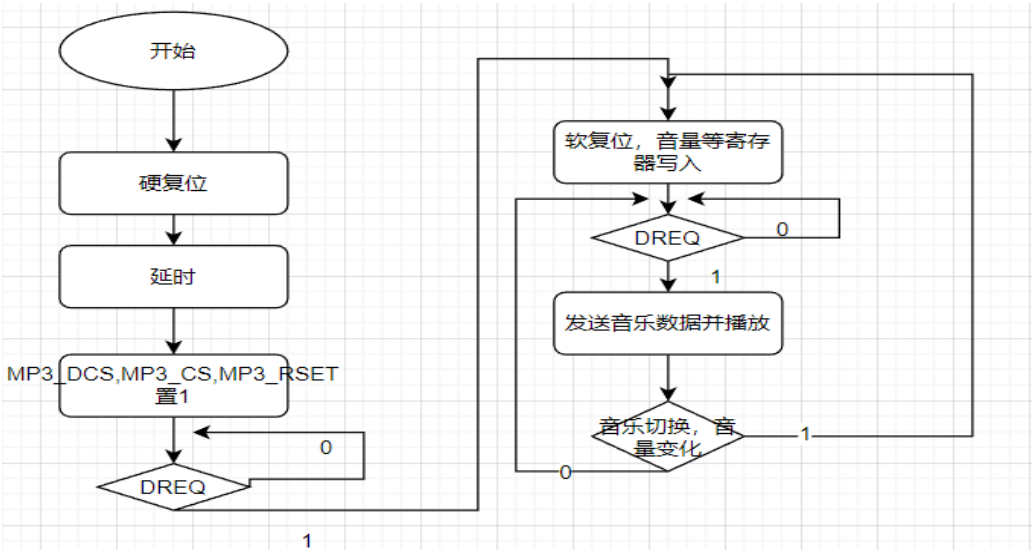
　　end　　endmodule

　　end

6.MP3 模块

功能：播放音乐，调节音量，切换音乐

框图：



接口信号定义：

input CLK;//100MHz 时钟

input RST;//复位信号

output　reg MP3_RSET,//MP3 复位信号

output　reg MP3_CS,//MP3，spi 通信的命令传输有效信号，低电平有效

output　reg MP3_DCS,//MP3，spi 通信的数据传输有效信号，低电平有效

output　reg MP3_MOSI,//MP3，spi 通信的 master 输出端

input　MP3_MISO,//MP3，spi 通信的 master 输入端

output　reg MP3_SCLK,//MP3，spi 通信的时钟

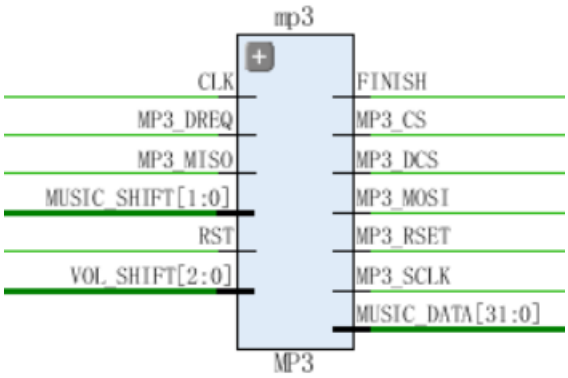
input　MP3_DREQ,//数据发送有效信号

input [2:0] VOL_SHIFT;//音量切换

input [1:0] MUSIC_SHIFT;//音乐切换

output reg FINISH=0;//音乐切换或音乐播放结束

output [31:0]MUSIC_DATA;//音乐数据



代码：

```
module MP3
(
    CLK,
    RST,

    MP3_RSET,
    MP3_CS,//命令
    MP3_DCS,//数据
    MP3_MOSI,
    MP3_MISO,
    MP3_SCLK,
    MP3_DREQ,

    VOL_SHIFT,
    MUSIC_SHIFT,

    FINISH,

    MUSIC_DATA

);
    input CLK;
    input RST;

    output reg MP3_RSET=1;
    output reg MP3_CS=1;
    output reg MP3_DCS=1;
    output reg MP3_MOSI=0;
    input MP3_MISO;
    output reg MP3_SCLK=0;
    input MP3_DREQ;

    input [2:0] VOL_SHIFT;
    input [1:0] MUSIC_SHIFT;
    output reg FINISH=0;
    output [31:0]MUSIC_DATA;

    wire mp3_clk;
    Divider #(100)divider_mp3(CLK,mp3_clk);// 分 频 到 1MHz

    //设置命令
    integer cnt=0;
    integer cmd_cnt=0;
    parameter cmd_cnt_max=4;
    reg [31:0] next_cmd;

    reg [127:0] cmd_init={32'h02000804,32'h02020055,32'h02039800,32'h020B7070};
    reg [127:0] cmd={32'h02000804,32'h02020055,32'h02039800,32'h020B7070};


    //读取音乐
    wire [31:0] data0;
    wire [31:0] data1;
    wire [31:0] data2;
    reg [31:0] data;
    reg [17:0] pos=0;
    blk_mem_gen_0 music_0(.clka(CLK),.dina(0),.wea(0),.addra(pos[17:0]),.douta(data0));
    blk_mem_gen_1
```

```
music_1(.clka(CLK),.dina(0),.wea(0),.addra(pos[17:0]),.douta(data1));
    blk_mem_gen_2 music_2(.clka(CLK),.dina(0),.wea(0),.addra(pos[17:0]),.douta(data2));

    //音乐切换
    reg   [1:0] pre_music=0;
    reg   [1:0] now_music=0;
    integer  delay=0;//设置延迟，延迟 0.5s 来保证此阶段命令完成发送
    always @(negedge mp3_clk ) begin
        if(delay==0) begin
            if(pre_music!=now_music)
                delay<=50_0000;
            case(MUSIC_SHIFT)
            2'b00:
              now_music<=0;
            2'b10:
               now_music<=1;
            2'b01:
               now_music<=2;
            default:
               now_music<=0;
            endcase
        end
        else
             delay<=delay-1;
    end

    //音量控制
    reg [15:0]pre_vol=16'h7070;
    reg [15:0]now_vol=16'h7070;
    integer vol_delay=0;
    always @(negedge mp3_clk) begin
        if(vol_delay==0) begin
            if(pre_vol!=now_vol)
                  vol_delay<=50_0000;

now_vol<=16'h7070-VOL_SHIFT*16'h1010;
        end
        else
            vol_delay<=vol_delay-1;
    end

  //MP3 状态
    parameter INIT = 3'd0;
    parameter CMD_WRITE = 3'd1;
    parameter VOL_CHANGE = 3'd2;
    parameter DATA_WRITE = 3'd3;
    parameter RSET_OVER = 3'd4;
    parameter VOL_SET_PRE = 3'd5;
    parameter VOL_SET = 3'd6;

    parameter len0=8'd125;
    parameter len1=8'd100;
    parameter len2=13'd90;


    reg[2:0] state=0;
    reg[7:0] len;
    integer len_cnt=0;


    always @(posedge mp3_clk or negedge RST) begin//接收到复位信号，切换音乐，或者一首音乐放完，则复位
            if( ~RST|| pre_music!=now_music || (MUSIC_SHIFT==2'b10 ? (len>len1):(MUSIC_SHIFT==2'b01 ? (len>len2):(len>len0) ))) begin
                pos<=0;
                pre_music<=now_music;
                cnt<=0;
                MP3_RSET<=0;
                cmd_cnt<=0;
                state<=RSET_OVER;
                cmd<=cmd_init;
                MP3_SCLK<=0;
                MP3_CS<=1;
                MP3_DCS<=1;
                len<=0;
                len_cnt<=0;
                FINISH<=1&RST;
            end
            else begin
                if(len_cnt<1000_000)//计算大致播放时长
```

```
                len_cnt<=len_cnt+1;
            else begin
                len_cnt<=0;
                len<=len+1;
            end
            case(state)
            INIT:begin// 由 于 是 SPI MODE0 模式，在第一个时钟上升沿之前要准备好数据
                MP3_SCLK<=0;

if(cmd_cnt>=cmd_cnt_max) begin

state<=VOL_CHANGE;
                end
                else if(MP3_DREQ) begin
                    MP3_CS<=0;
                    cnt<=1;

state<=CMD_WRITE;

MP3_MOSI<=cmd[127];

cmd<={cmd[126:0],cmd[127]};
                end
            end
            CMD_WRITE:begin// 写 入 控制寄存器
                if(MP3_DREQ) begin
                    if(MP3_SCLK) begin
                        if(cnt<32)begin

cnt<=cnt+1;

MP3_MOSI<=cmd[127];

cmd<={cmd[126:0],cmd[127]};
                            end
                            else begin

MP3_CS<=1;
                                cnt<=0;

cmd_cnt<=cmd_cnt+1;

state<=INIT;
                            end
                        end

MP3_SCLK<=~MP3_SCLK;
                    end
                end
                VOL_CHANGE:begin  // 判 断音量是否变化

if(now_vol[15:0]!=cmd_init[15:0]) begin

state<=VOL_SET_PRE;

next_cmd<={16'h020B,now_vol[15:0]};
                    end
                    else if(MP3_DREQ) begin
                        MP3_DCS<=0;
                        MP3_SCLK<=0;

state<=DATA_WRITE;
                        case (now_music)
                            3'd0:begin

data<={data0[30:0],data0[31]};

MP3_MOSI<=data0[31];
                            end
                            3'd1:begin

data<={data1[30:0],data1[31]};

MP3_MOSI<=data1[31];
                            end
                            3'd2:begin

data<={data2[30:0],data2[31]};

MP3_MOSI<=data2[31];
                            end
                        endcase
                        cnt<=1;
                    end
```

```
cmd_init[15:0]<=now_vol;
                end
                DATA_WRITE:begin //向 MP3
写入数据
                    if(MP3_SCLK)begin
                        if(cnt<32)begin
                            cnt<=cnt+1;

MP3_MOSI<=data[31];

data<={data[30:0],data[31]};
                        end
                        else begin
                            MP3_DCS<=1;
                            pos<=pos+1;

state<=VOL_CHANGE;
                        end
                    end

MP3_SCLK<=~MP3_SCLK;
                end
                RSET_OVER:begin    //复位结
束等待一点时间
                    if(cnt<1000000) begin
                        cnt<=cnt+1;
                    end
                    else begin
                        cnt<=0;
                        state<=INIT;
                        MP3_RSET<=1;
                        FINISH<=0;
                    end
                end
                VOL_SET_PRE:begin     //SPI
MODE0 模式在 CS 拉低后第一个上升沿之
前准备好数据
                    if(MP3_DREQ) begin
                        MP3_CS<=0;
                        cnt<=1;
                        state<=VOL_SET;

MP3_MOSI<=next_cmd[31];

next_cmd<={next_cmd[30:0],next_cmd[31]};
                    end
                end
                VOL_SET:begin//设置音量
                    if(MP3_DREQ) begin
                        if(MP3_SCLK) begin
                            if(cnt<32)begin

cnt<=cnt+1;

MP3_MOSI<=next_cmd[31];

next_cmd<={next_cmd[30:0],next_cmd[31]};
                            end
                            else begin

MP3_CS<=1;
                                cnt<=0;

state<=VOL_CHANGE;
                            end
                        end

MP3_SCLK<=~MP3_SCLK;
                    end
                end
                default:begin
                    ;
                end
                endcase
            end
        end

        assign MUSIC_DATA=data;

endmodule
```

7.Time_transfer 模块

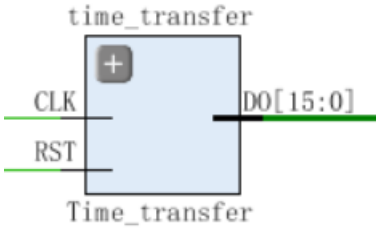
功能：进行计时，并将时间转化为分：秒的形式。

接口信号定义：

input CLK;//100MHz

input RST;//复位信号

output [15:0]DO;//输出的对应的分：秒信号，以显示在七段数码管



代码：

```
module Time_transfer
(
    CLK,
    RST,
    DO
);

    input CLK;//100MHz
    input RST;
    output [15:0]DO;

    Divider #(100)divider_transfer(CLK,clk);// 分 频 到 1MHz

    integer t=0,T=0;
    always @(negedge clk) begin
        if(RST)begin
            t<=0;
            T<=0;
        end
        else begin
            if(t<999999) begin
                t<=t+1;
            end
            else begin
                t<=0;
                T<=T+1;
            end
        end
    end
    assign DO[3:0]=T%10;
    assign DO[7:4]=(T/10)%6;
    assign DO[11:8]=(T/60)%10;
    assign DO[15:12]=T/600;

endmodule
```

8.Display7 模块

功能：进行计时，并将时间转化为分：秒的形式。

接口信号定义：

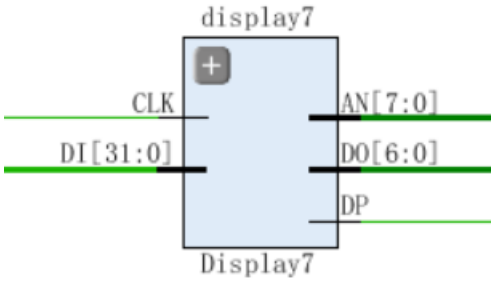
input CLK;//100MHz

input [31:0] DI;//相关数据输入

output reg [6:0] DO;//七段数码管

output reg [7:0] AN=8'b01111111;//数码管位选

output reg DP;//点号



代码：

```
module Display7
(
    CLK,
    DI,
);
```

```
    input CLK;
    input [31:0] DI;//4*8
    output reg [6:0] DO;
    output reg [7:0] AN=8'b01111111;//控制第一个阳极为低电平
    output reg DP;

    reg [5:0] p=0;//最大值 31

    wire clk_t;
    Divider #(200000)divider_display(CLK,clk_t);// 分 频到 2ms 的周期
    always @(posedge clk_t) begin
        AN<={AN[6:0],AN[7]};//先驱动阴极，再拉低阳极信号来显示
        p<=p+4;//向前一位数字
        if(AN[1]==0) begin
            DP<=0;
        end
        else begin
            DP<=1;
        end

case({DI[p+3],DI[p+2],DI[p+1],DI[p]})
            4'b0000: begin
                DO<=7'b1000000;
            end
            4'b0001: begin
                DO<=7'b1111001;
            end
            4'b0010: begin
                DO<=7'b0100100;
            end
            4'b0011: begin
                DO<=7'b0110000;
            end
            4'b0100: begin
                DO<=7'b0011001;
            end
            4'b0101: begin
                DO<=7'b0010010;
            end
            4'b0110: begin
                DO<=7'b0000010;
            end
            4'b0111: begin
                DO<=7'b1111000;
            end
            4'b1000: begin
                DO<=7'b0000000;
            end
            4'b1001: begin
                DO<=7'b0010000;
            end
            default: begin
                DO<=7'b1111111;
            end
        endcase
    end

endmodule
```

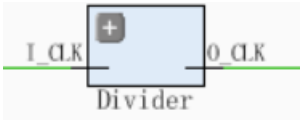
9.Divider 模块

功能：时钟分频。

接口信号定义：

input I_CLK,//分频前时钟

output reg O_CLK//分频后时钟



代码：

```
module Divider(
      input I_CLK,
      output reg O_CLK
      );
parameter cycle=100;
reg[31:0] count=0;
initial
O_CLK=0;

always@(posedge I_CLK)
begin
   if(count<cycle/2)
      count=count+1;
   else
      begin
         count=1;
         O_CLK=~O_CLK;
      end
end
endmodule
```

五、测试模块建模

由于 top 模块不需要测试，game 和 vga 由于时钟太过于繁琐复杂，Divider 模块本学期已经做过，都没有测试的必要性，因此只对其他 5 个模块进行了测试。

1.Control 模块

```
`timescale 1ns / 1ps

module Control_tb();
`timescale 1ns / 1ps
reg clk;
reg RST;
reg MISO;

wire MOSI;
wire n_CS;
wire roundDD;
wire [11:0] x_reg_temp,y_reg_temp,z_reg_temp;
wire clk_SPI;

initial
begin
clk=0;
```

```
RST=0;
#1000 RST=1;
end

always
begin
#10 clk=~clk;
end
Control control_inst(
        clk,
        RST,
        MISO,
        MOSI,
        n_CS,
        roundDD,
        x_reg_temp,y_reg_temp,z_reg_temp,
        clk_SPI
          );

endmodule
```

2.FIFO 模块

```
`timescale 1ns / 1ps
module FIFO_tb();

  reg CLK, roundDD, RST;
reg [11:0] x_reg_temp, y_reg_temp, z_reg_temp;
wire    [11:0] x_reg, y_reg, z_reg;
wire reset1;

initial
begin
        CLK=0;
        RST=0;
        roundDD=0;
      #100 RST=1;
end

always begin
        #10 CLK<=~CLK;

end
```

```
always begin
      #120 roundDD<=~roundDD;
end

initial begin
     x_reg_temp= 1;
     y_reg_temp=1;
     z_reg_temp=1;
     #200      x_reg_temp= 2;
     y_reg_temp=2;
     z_reg_temp=2;
     #200    x_reg_temp= 3;
     y_reg_temp=3;
     z_reg_temp=3;
end

FIFO FIFO_inst(
CLK, roundDD, RST,x_reg_temp, y_reg_temp, z_reg_temp,x_reg, y_reg, z_reg,reset1
      );
endmodule
```

3.Time_transfer 模块

```
`timescale 1ns / 1ps

module Time_transfer_tb( );

     reg CLK;
     reg RST;
     wire [15:0]DO;

initial
begin
      CLK=0;
      RST=0;
     #100 RST=1;
end
always begin
      #2 CLK<=~CLK;
end
Time_transfer Time_transfer_inst(CLK,
      RST,
      DO);
endmodule
```

4.MP3 模块

```
`timescale 1ns / 1ps
module MP3_tb(

    );
  reg CLK;
    reg RST;

    wire MP3_RSET;
    wire MP3_CS=1;
    wire MP3_DCS=1;
    wire MP3_MOSI=0;
    reg MP3_MISO;
    wire MP3_SCLK=0;
    reg MP3_DREQ;

    reg [2:0] VOL_SHIFT;
    reg [1:0] MUSIC_SHIFT;
    wire FINISH=0;
    wire [31:0]MUSIC_DATA;

  initial
begin
MP3_DREQ=1;
CLK=0;
RST=0;
#1000 RST=1;
end

initial
begin
VOL_SHIFT=3'b001;
#30000 VOL_SHIFT=3'b111;
end

initial
begin
MUSIC_SHIFT=3'b001;
#20000 MUSIC_SHIFT=3'b111;
end
```

```
always
begin
#10 CLK=~CLK;
end

MP3 MP3_inst(CLK,
    RST,

    MP3_RSET,
    MP3_CS,//命令
    MP3_DCS,//数据
    MP3_MOSI,
    MP3_MISO,
    MP3_SCLK,
    MP3_DREQ,

    VOL_SHIFT,
    MUSIC_SHIFT,

    FINISH,

    MUSIC_DATA);
endmodule
```

5.Display7 模块

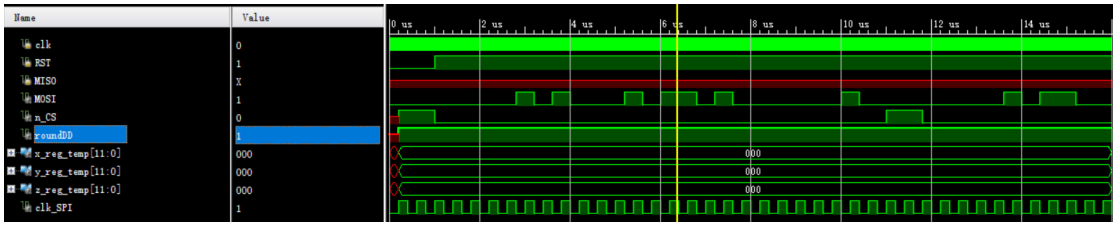
```
`timescale 1ns / 1ps

module Display7_tb( );
    reg CLK;
    reg [31:0] DI;//4*8
    wire [6:0] DO;
    wire [7:0] AN=8'b01111111;//控制第一个阳极为低电平
    wire DP;

    initial
begin
    CLK=0;
    DI=32'habcd_1234;
end

always begin
    #10 CLK<=~CLK;
end
```

```
Display7 Display7_inst( CLK,
        DI,
        DO,
        AN,
        DP      );

endmodule
```

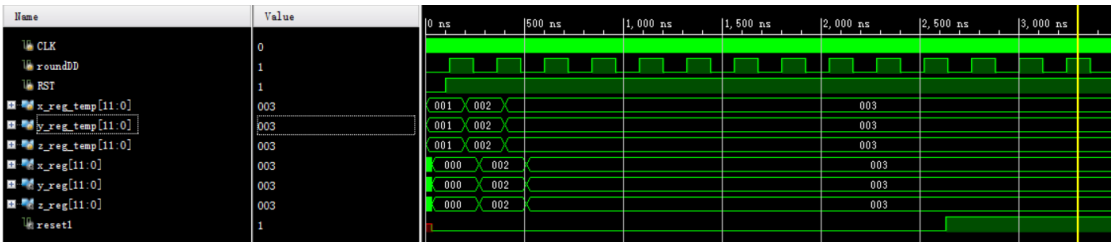
六、实验结果

1.Modelsim 仿真波形图

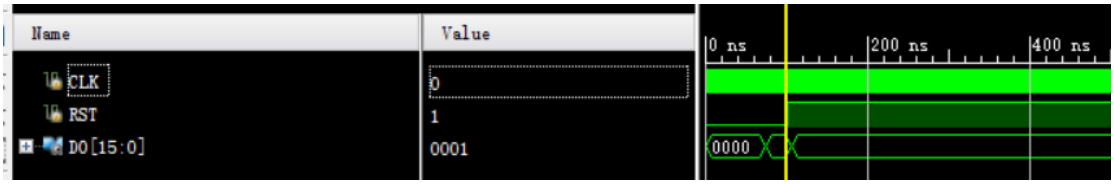
1.1　Control 模块（由于仿真没有外围模块交互，故有 MISO 为不确定值）



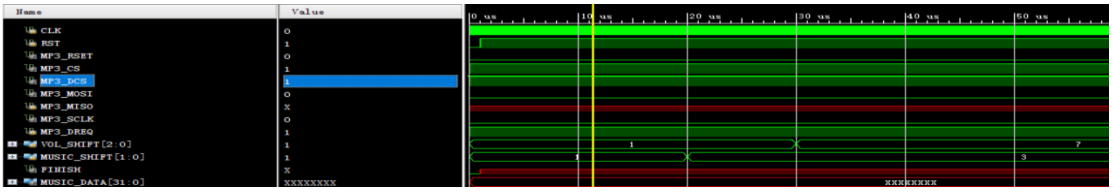
1.2　FIFO 模块



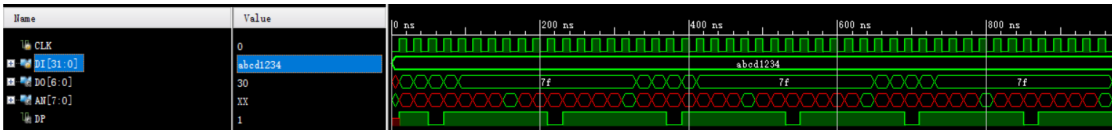
1.3 Time_transfer 模块（时间跨度太大）



1.4 MP3 模块



1.5 Display7 模块

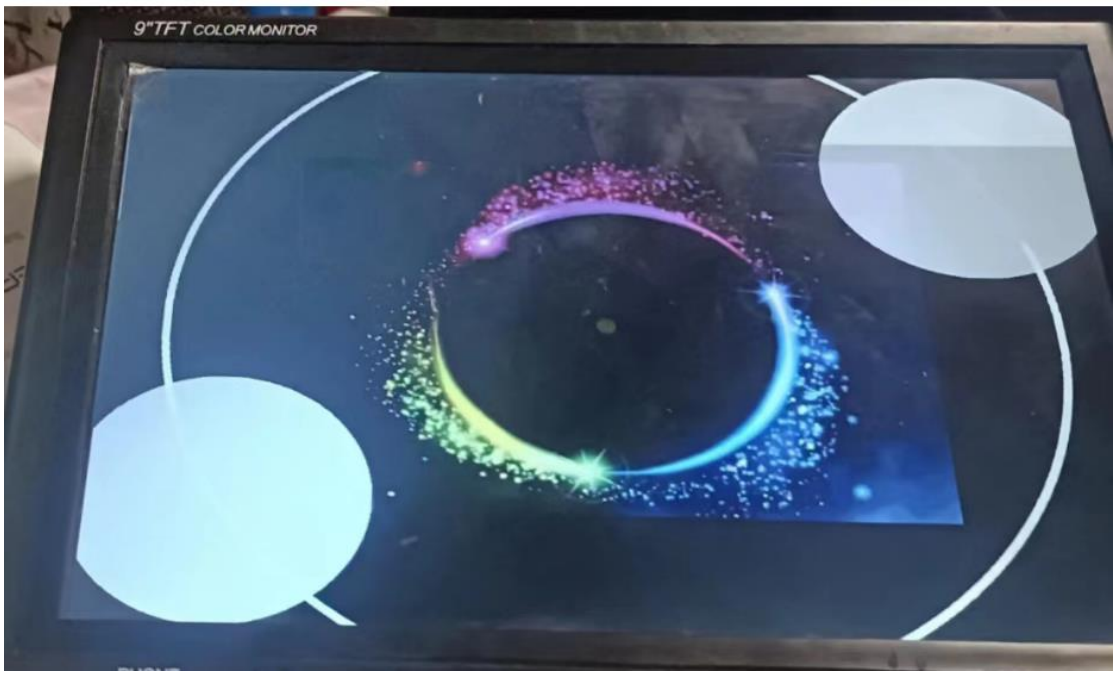


2.实验结果贴图

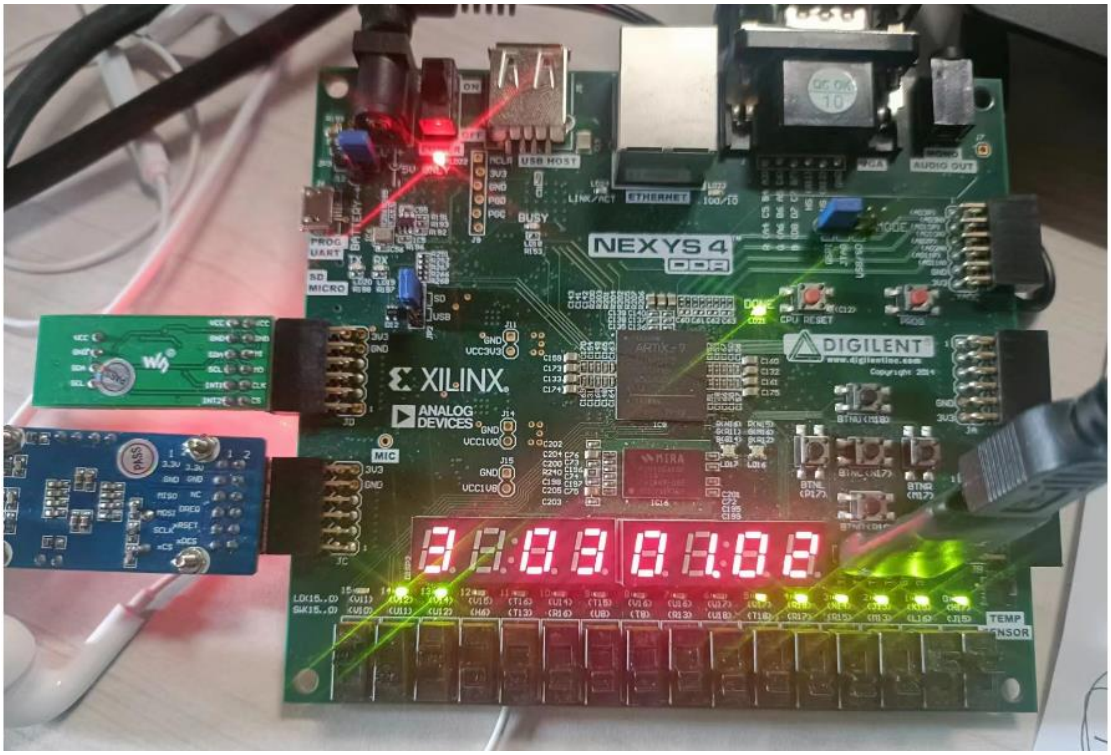
（由于 **MP3** 通过声音输出，这里不再展示）

2.1 vga 游戏运行时的画面

2.2 圆球成功进入判定区时的动画（圆变白变大）



2.3 LED 左边三个灯显示音量，右边显示角度数据;七段数码管的最左边显示音乐序号，中间靠左显示分数，最右边显示音乐播放时间



七、结论

此次实验基本达到预期效果。对于数字系统自上而下的设计方法，各个模块之间的时序和关联很重要，由于自上而下，各个模块之间的层次关系也要搞清楚。不同于软件设计的 C/C++语言，verilog 作为一种典型的硬件设计语言，其基本逻辑与前者有很大区别，后者常常基于偏物理电路层面理解，编写程序时要考虑物理条件和电路关系，这样才能不被之前的惯性思维带偏而出错。

同时，此次实验之中，无论是单独验证各个模块还是在一起综合下板，总体过程虽有瓶颈，但整体较为顺利。但是，对于本次实验，还有更多需要改进的地方，对于开发板和外围硬件的功能可以做进一步探索，游戏模式也有待完善。

八、心得体会及建议

1.课程收获

在一个学期的学习中，我认识到数字逻辑是一个比较挑战性的课程，但同时也是一门有趣的课程。虽然在课程中不断遭受挫折，无论是理论学习还是实验学习都有不懂的地方，但老师、同学会为我耐心解答。因此，我感到这个课程总体完成得非常充实，收获了很多东西。

在理论学习方面，我学到了基本计算机电路的底层逻辑，并由基本的组合逻

辑、时序逻辑、数字系统的学习为实验中的程序设计打下基础。其中由逻辑的层面理解电路是一大收获。

在实验学习方面，对于每次小实验在开发板上的成功的结果都很兴奋，他让我深深获得了动手的成就感。对于今后的实践学习起了很大的推动作用。

总之，此门课程让我认识到了理论实践结合的魅力，受益匪浅。

2.课程建议

2.1 希望老师在理论方面多一点笔头作业，来加深对相关知识的理解。
2.2 希望老师可以在实验课程中加上对开发板、vivodo,modelsim,logisim 相关软件的介绍与讲解
2.3 希望大作业提供之前学长的建议以及充实有效的外部模块的设计资料
2.4 希望老师在实验课上也可以结合相关理论知识，阐明原理
2.5 理论课偏电路，而实验课一般用行为描述写代码，体现不出底层电路，希望老师可以连接两者。

3.对于数字芯片设计大环境的认知与体会

从全球芯片产业链来看，美国是全球集成电路产业最发达的国家，也是最早进行集成电路大规模生产的国家，目前集成电路产业的技术水平和经济实力居世界领先地位。全球前 20 大半导体公司中，美国企业有 15 家，中国有两家企业入围。排名前 5 位的企业，美国就占了 3 家（英特尔、 AMD、三星），日本 1 家（东芝）；欧洲有 1 家（英飞凌）。中国大陆仅有中芯国际一家在研发和设计方面具有一定的能力。同时，我们仅有中芯国际一家能够制造 28 nm 的芯片

在国内现阶段，芯片技术属于我国的卡脖子技术之一。芯片是国之重器，是现代信息产业的基础和核心，我国已经成为世界上最大的集成电路市场，但是在基础关键领域受制于人，所以对于国内芯片产业的被动局面，我们必须要在半导体领域迎头赶上。这是国家意志也是人民期盼。芯片产业是信息产业的核心技术，具有基础性、战略性、先导性的重要作用，关系国计民生和国家安全。

我认为对于国内芯片产业的突破主要在于相关企业和国内高校。

企业方面：国内企业公司的芯片设计研发时还存在很多问题。既包括国内的产业升级，也包括国外巨头对我们产品的垄断。国内芯片企业整体落后，当然这并不是说他们没有努力，而是在这种情况下他们所付出的努力还不能和国外公司相匹敌。这些公司已经意识到了芯片设计的重要性，并开始将相关研发资源和人员投入到芯片设计中来。同时还成立了很多专门研究芯片的实验室。但除此之外还有一个重要原因：它们都是从 0 到 1 开始做芯片设计。就像华为一样，他们的最早也是从低端做起，慢慢积累实力和经验，最后才不断进步而走到现在这样一个高度。

因此，我国的芯片企业必须要有所为之。可以通过引进国外先进技术和管理经验或者加强与国外公司的合作来实现国内产业升级，最重要的是要学会自力更

生，依靠国内自身力量来发展自己，也就是要大力发展属于自己的核心技术。其中非常重要的一个策略就是自主创新。

同时可以看到，国内高校里存在着两个非常严重的问题：1)学生们只是为了毕业后好找工作，才去学一些相关知识；2)学校里很少有专门培养人才的机构或者实验室。我国在半导体行业发展上落后于西方国家几十年，与我国高校在芯片相关学科上存在一定差距有关。因此，应建立和完善研究生培养体系以加强本科生参与科研的力度和深度。对于大学生而言，一方面，大学生要研究硬件发展情况、研究领域以及设计方法；另一方面，大学生也要参与到新技术的开发中去。让学生们在这两个方面进行创新和探索，只有这样才能让中国芯片更好地发展。

国内很多大学生都喜欢钻研理论知识，很少有学生去深入了解实际的研发过程，没有将理论知识与实践结合起来。实际上，掌握好基础知识和专业技能后，学生们的发展潜力非常大。尤其是对于国内大学的大学生而言，应该利用好国家给予的优惠政策，在实习、实践过程中将基础知识和专业技能学好学透。笔者作为一名本科生，也应不断提高自身素质和能力来满足社会对人才的需求。为国家数字芯片产业做出自己的贡献。