汕 头 大 学 工 学 院

二 级 项 目 报 告

课 程 名 称 ： 嵌入式综合设计二级项目

二级项目题目： 基于 FPGA 控制 VGA 显示的弹球游戏设计

Pinball Game Design Based on FPGA Control and VGA Display

指 导 教 师 ： 李芬兰

系 别： 工学院电子工程系 专 业： 电子信息工程

	姓名	学号
组长	梁辉鸿	2017141034
成员		

完成时间： **2020 年 9 月 6 日**

目录

1. VGA 概述及工作原理....4
1.1 VGA 概述....4
1.2 工作原理....5
1.3 VGA 参数....6
2. VGA_Ball—游戏顶层模块....7
2.1 模块说明....7
2.2 模块代码....7
2.3 顶层 RTLView....12
2.4 各子模块功能说明....13
3. vga_controller—VGA 时序控制模块....14
3.1 模块说明....14
3.2 模块代码....14
3.3 RTLView....17
4. game_start—游戏启动模块....18
4.1 模块说明....18
4.2 模块代码....18
4.3 RTLView....18
5. color_gen_char1—“游戏开始”字符产生模块....19
5.1 模块说明....19
5.2 模块代码....19
6. color_gen_char2—“游戏结束”字符产生模块....25
6.1 模块说明....25
6.2 模块代码....26
7. color_gen—颜色产生模块....32
7.1 模块说明....32
7.2 模块代码....32
7.3 RTLView....34
8. Ball—弹球生成模块....35
8.1 模块说明....35
8.2 模块代码....35
8.3 RTLView....38
9. Ball_speed—弹球速度键控模块....38
9.1 模块说明....38
9.2 模块代码....38
9.3 RTLView....40
10. block—挡板生成模块....40
10.1 模块说明....40
10.2 模块代码....40
10.3 RTLView....42
11. block_move—挡板键控模块....43
11.1 模块说明....43
11.2 模块代码....43

11.3 RTLView 44
12. Background_Color—背景颜色选择模块 44
12.1 模块说明 44
12.2 模块代码 44
12.3 RTLView 45
13. state_run—弹球游戏运行模块 45
13.1 模块说明 45
13.2 模块代码 45
13.3 RTLView 48
14. arykeyscan—键值采集和消抖模块 48
14.1 模块说明 48
14.2 模块代码 48
14.3 RTLView 54
15. LTM_Param.h—VGA 显示参数定义 55
15.1 内容说明 55
15.2 文件内容 55
16. Begin.h—字符点阵字模定义 56
16.1 内容说明 56
16.2 文件内容 56
17. 实验过程问题及解决描述 58
参考文献 59

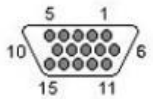
基于 FPGA 控制 VGA 显示的弹球游戏设计

1. VGA 概述及工作原理

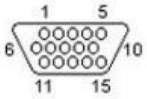
1.1 VGA 概述

VGA（Video Graphics Array）即视频图形阵列，是 IBM 在 1987 年随 PS/2（PS/2 原是“Personal System 2”的意思，“个人系统 2”，是 IBM 公司在 1987 年推出的一种个人电脑。PS/2 电脑上使用的键盘鼠标接口就是现在的 PS/2 接口。因为标准不开放，PS/2 电脑在市场中失败了。只有 PS/2 接口一直沿用到今天）一起推出的使用模拟信号的一种视频传输标准，在当时具有分辨率高、显示速率快、颜色丰富等优点，在彩色显示器领域得到了广泛的应用。这个标准对于现今的个人电脑市场已经十分过时。即使如此，VGA 仍然是最多制造商所共同支持的一个标准，个人电脑在加载自己的独特驱动程序之前，都必须支持 VGA 的标准。例如，微软 Windows 系列产品的开机画面仍然使用 VGA 显示模式，这也说明其在显示标准中的重要性和兼容性。

显示卡端的接口为15针母插座：



显示器连线端的接口为15针公插头：



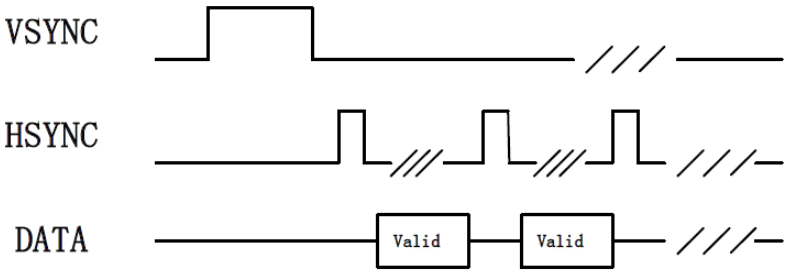
针脚定义：

1	Video-Red
2	Video-Green
3	Video-Blue
4	GND
5	CPU sense
6	GND-R
7	GND-G
8	GND-B
9	No pin present
10	GND-sync/self-raster
11	GND
12	DDC data
13	H-sync
14	V-sync
15	DDC clock

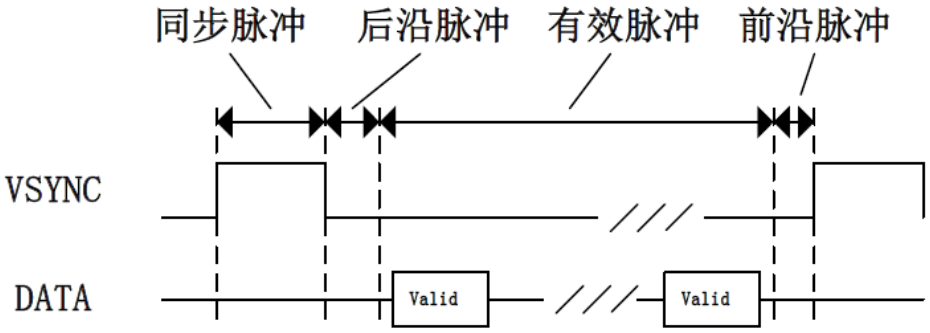
驱动 VGA 显示的接口，主要有以下 3 种信号：行同步信号 HSYNC，场同步信号 VSYNC 和 3 条色彩电压传输信号（R、G、B 分别对应）。色彩信号的电压为 0~0.7V，其同步是靠前面两个信号来协助的。至于 HSYNC 和 VSYNC 和色彩信号之间以什么样的关系进行传输，这都是相对固定的，虽然 VGA 收发双方没有时钟信号做同步，但我们通常会约定发送方有一个基本的时钟，VSYNC、HSYNC 和色彩信号都会按照这个时钟的节拍来确定状态。

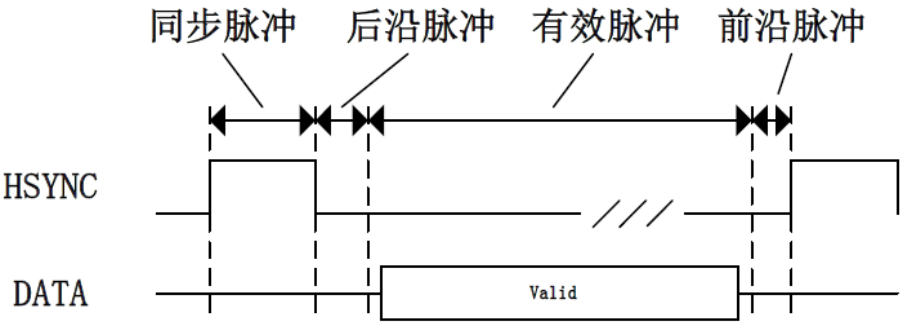
1.2 工作原理

VGA的接口时序如图所示，场同步信号VSYNC在每帧（即送一次全屏的图像）开始的时候产生一个固定宽度的高脉冲，行同步信号 HSYNC 在每行开始的时候产生一个固定宽度的高脉冲，色彩数据在某些固定的行和列交汇处有效。



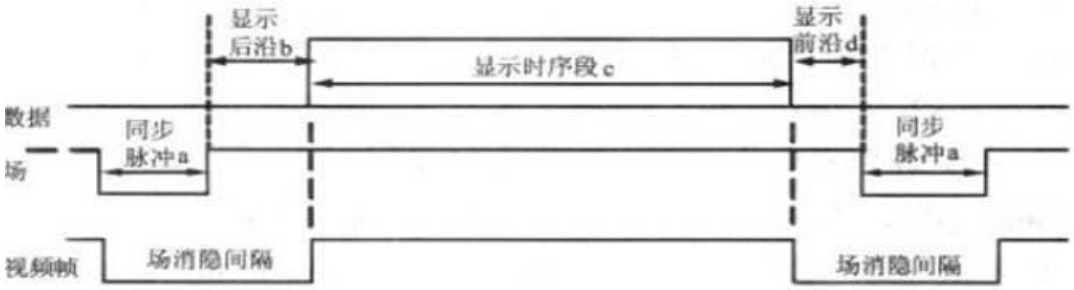
如前所述，我们通常以一个基准时钟驱动 VGA 信号的产生，用这个基准时钟为时间单位来产生的时序如图所示。





显示器的扫描方式：逐行扫描是扫描从屏幕左上角一点开始，从左像右逐点扫描，每扫描完一行,电子束回到屏幕的左边下一行的起始位置，在这期间，CRT对电子束进行消隐，每行结束时，用行同步信号进行同步；当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，同时进行场消隐,开始下一帧。

VGA 的行时序



VGA 的场时序

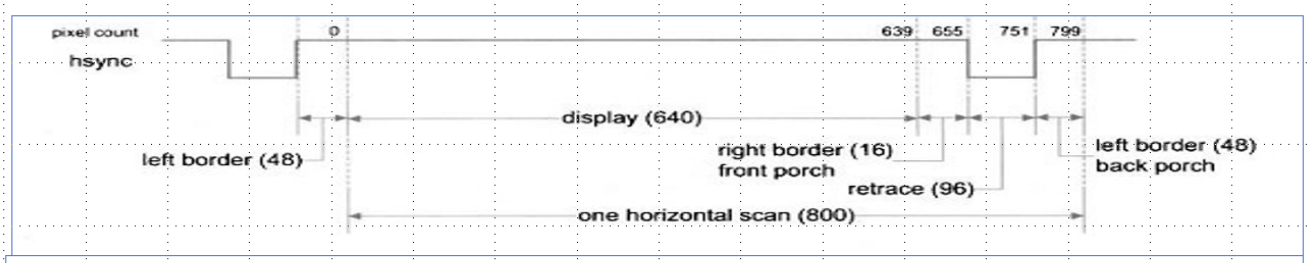
如上图，a 同步脉冲，b 显示后沿，c 显示时序段，d 显示后沿。这四段组成了一个完整的时序。图中最后的同步脉冲 a 是下一个时序的。所以消隐间隔就是上一个时序的显示前沿+本时序的同步脉冲+本时序的显示后沿。同步脉冲是包含在消音间隔中的。而显示时序段就是用来传输像素点的时钟周期。

1.3 VGA 参数

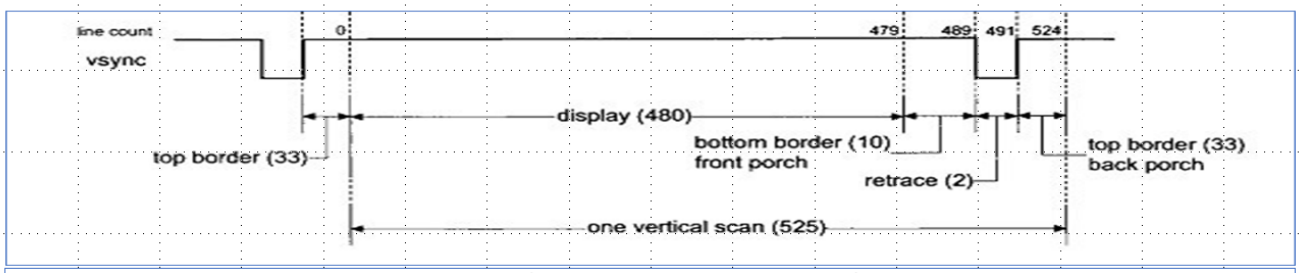
本实验程序用的是刷新频率为 60Hz，分辨率为 640X480 的标准 VGA 显示驱动，即每一行有 640 个像素，整个显示区域一共有 480 行。且基准驱动时钟为 25MHz，它的脉冲计数表如下所示。注意列的单位为“行”，而行的单位为“基准时钟周期数”，即 25MHz 时钟脉冲数。

VGA 驱动时序参数表

行/列	同步脉冲	后沿脉冲	显示脉冲	前沿脉冲	帧长
列	2	33	480	10	525
行	96	48	640	16	800



行同步信号：左边框：48个，显示区：640个，右边框：16个，返回区：96个； 总共800个



场同步信号： 顶边框:33个，显示区:480个，底边框:10个，返回区:2个，总共525个

2. VGA_Ball—游戏顶层模块

2.1 模块说明

顶层模块的功能是将所有独立的子模块连接到一起，定义 IO 端口，从而实现整个弹球游戏运行的功能。

2.2 模块代码

```
//基于 FPGA 的 VGA 显示弹球的游戏设计
//***********键控说明**********************************
//       S1                  增大 x 方向的速度
//       S2                  减小 x 方向的速度
//       S3                  增大 y 方向的速度
//       S4                  减小 y 方向的速度
//       S5                  切换到上一个背景颜色
//       S6                  切换到下一个背景颜色
//       S7                  增大挡板的大小
//       S8                  减小挡板的大小
//       S13                     控制挡板左移
//       S14                     控制挡板右移
//       S9                  开始游戏
//       RESET                重启游戏，回到游戏启动页面
//  SW3  SW4  SW5  SW6      输入弹球的半径，调整弹球大小
```

```
//*****************************************************

module VGA_Ball(
       input ext_clk_25m,  //外部输入 25MHz 时钟信号
       input ext_rst_n,        //外部输入复位信号，低电平有效
       input [3:0]switch,  //拨码开关输入弹球半径大小
       input [3:0]key_v,       //4 个列按键输入，未按下为高电平，按下后为低
电平
       output [3:0]key_h,  //4 个行按键输出
       output vga_r,           //可见显示区的 R 分量
       output vga_g,           //可见显示区的 G 分量
       output vga_b,           //可见显示区的 B 分量
       output vga_hsy,     //行同步信号
       output vga_vsy          //场同步信号
       );

   //--------------------------------------
   //键值采集和消抖模块--产生键控指令
   wire [15:0]key_num;

   arykeyscan     arykeyscan_inst(
                    .clk(ext_clk_25m),              //时钟信号
                    .rst_n(ext_rst_n),      //复位信号，低电平有效
                    .key_v(key_v),              //4 个按键输入，未按下为
高电平，按下后为低电平
                    .key_h(key_h),              //4 个行按键输出
                    .display_num(key_num)    //产生键控指令
                    );

   //--------------------------------------
   //背景颜色选择模块--产生背景选择信号
   wire [15:0]iDISPLAY_MODE;

   Background_Color    Background_Color_inst(
                        .iCLK(ext_clk_25m),
                        .iRST_n(ext_rst_n),
                        .key_num(key_num),
                        .oBackground_set(iDISPLAY_MODE)
                        );

   //----------------------------------------
   //颜色产生模块--产生球色、挡板色、背景色（可选）
   wire [7:0]Ball_S;
   wire [10:0]block_X1;
```

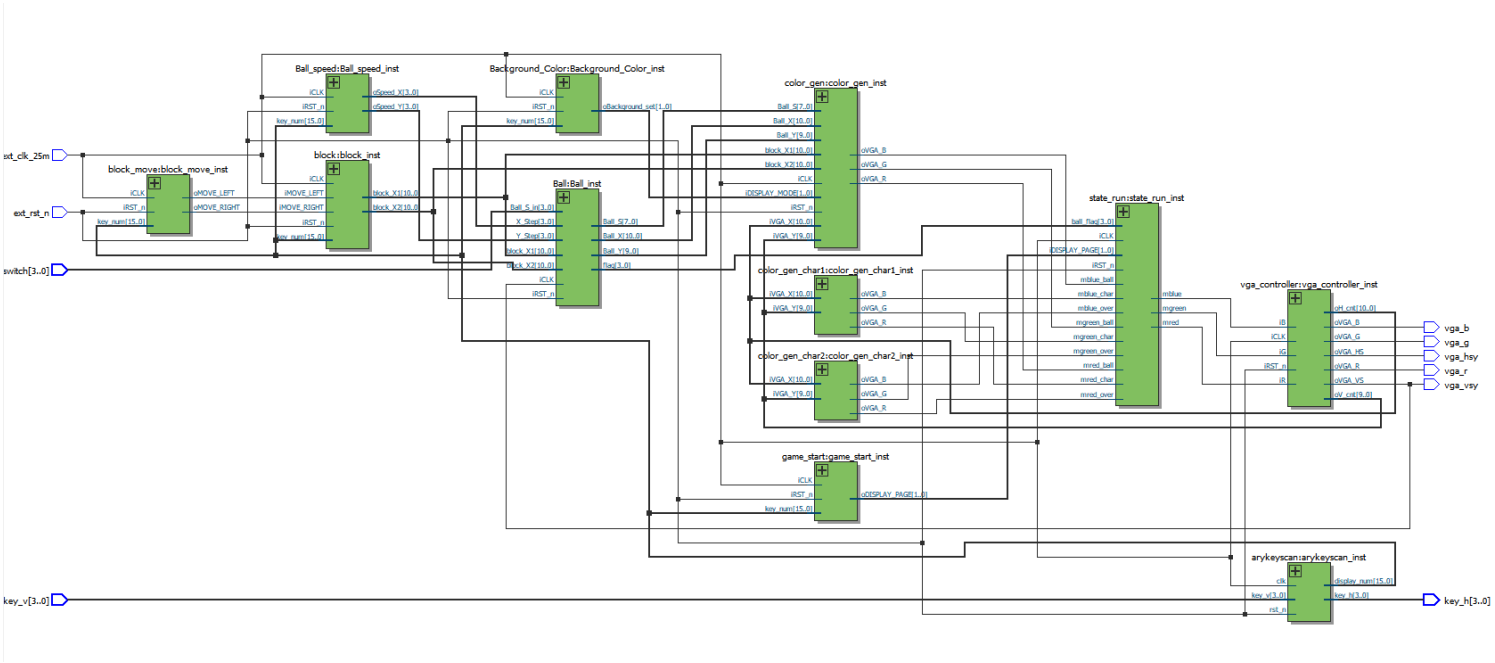
```
    wire [10:0]block_X2;
    wire [10:0]Ball_X;
    wire [9:0]Ball_Y;

    color_gen   color_gen_inst(
                  .iCLK(ext_clk_25m),
                  .iRST_n(ext_rst_n),
                  .Ball_X(Ball_X),
                  .Ball_Y(Ball_Y),
                  .iVGA_X(oH_cnt),
                  .iVGA_Y(oV_cnt),
                  .Ball_S(Ball_S),
                  .block_X1(block_X1),
                  .block_X2(block_X2),
                  .iDISPLAY_MODE(iDISPLAY_MODE),
                  .oVGA_R(mred_ball),
                  .oVGA_G(mgreen_ball),
                  .oVGA_B(mblue_ball)
                 );

    //-----------------------------------------------
    //游戏启动模块--按下 S9 键开始游戏
    wire [1:0]iDISPLAY_PAGE;

    game_start      game_start_inst(
                      .iCLK(ext_clk_25m),
                      .iRST_n(ext_rst_n),
                      .key_num(key_num),
                      .oDISPLAY_PAGE(iDISPLAY_PAGE)
                     );

    //-------------------------------------------
    //弹球游戏运行模块--控制游戏启动、运行、结束的界面
    wire [3:0]ball_flag;
    wire mred_char,mgreen_char,mblue_char;  //游戏启动页面
    wire mred_ball,mgreen_ball,mblue_ball;  //游戏运行页面
    wire mred_over,mgreen_over,mblue_over;  //游戏结束页面
    wire mred,mgreen,mblue;                  //实际输出

    state_run   state_run_inst(
                  .iCLK(ext_clk_25m),
                  .iRST_n(ext_rst_n),
                  .iDISPLAY_PAGE(iDISPLAY_PAGE),
                  .ball_flag(ball_flag),
```

```
                .mred_char(mred_char),
                .mgreen_char(mgreen_char),
                .mblue_char(mblue_char),
                .mred_ball(mred_ball),
                .mgreen_ball(mgreen_ball),
                .mblue_ball(mblue_ball),
                .mred_over(mred_over),
                .mgreen_over(mgreen_over),
                .mblue_over(mblue_over),
                .mred(mred),
                .mgreen(mgreen),
                .mblue(mblue)
                );

//------------------------------------------------
//弹球生成模块--控制弹球大小、位置、位置更新、输出标志
wire [3:0]oSpeed_X;
wire [3:0]oSpeed_Y;

Ball    Ball_inst(
        .iCLK(vga_vsy),
        .iRST_n(ext_rst_n),
        .Ball_S_in(switch),
        .X_Step(oSpeed_X),
        .Y_Step(oSpeed_Y),
        .block_X1(block_X1),
        .block_X2(block_X2),
        .Ball_X(Ball_X),
        .Ball_Y(Ball_Y),
        .Ball_S(Ball_S),
        .flag(ball_flag)
        );

//------------------------------------------------------------
//弹球速度键控模块--通过 S1、S2、S3、S4 按键改变弹球在 x 和 y 方向的速度
Ball_speed  Ball_speed_inst(
                .iCLK(ext_clk_25m),
                .iRST_n(ext_rst_n),
                .key_num(key_num),
                .oSpeed_X(oSpeed_X),
                .oSpeed_Y(oSpeed_Y)
                );

//------------------------------------------------------------
```

```
//挡板生成模块--通过 S7、S8 按键改变挡板大小，通过左右移信号实现左右移动
wire oMOVE_RIGHT;
wire oMOVE_LEFT;

block      block_inst(
          .iCLK(ext_clk_25m),
          .iRST_n(ext_rst_n),
          .iMOVE_RIGHT(oMOVE_RIGHT),
          .iMOVE_LEFT(oMOVE_LEFT),
          .key_num(key_num),
          .block_X1(block_X1),
          .block_X2(block_X2)
          );

//-------------------------------------------------------------
//挡板键控模块--按下按键 S13、S14，输出左移和右移信号
block_move     block_move_inst(
                 .iCLK(ext_clk_25m),
                 .iRST_n(ext_rst_n),
                 .key_num(key_num),
                 .oMOVE_LEFT(oMOVE_LEFT),
                 .oMOVE_RIGHT(oMOVE_RIGHT)
                 );

//-----------------------------------------------
//"游戏开始"字符产生模块--游戏启动页面
color_gen_char1 color_gen_char1_inst(
                    .iVGA_X(oH_cnt),
                    .iVGA_Y(oV_cnt),
                    .oVGA_R(mred_char),
                    .oVGA_G(mgreen_char),
                    .oVGA_B(mblue_char)
                    );

//-----------------------------------------------
//"游戏结束"字符产生模块--游戏结束页面
color_gen_char2 color_gen_char2_inst(
                    .iVGA_X(oH_cnt),
                    .iVGA_Y(oV_cnt),
                    .oVGA_R(mred_over),
                    .oVGA_G(mgreen_over),
                    .oVGA_B(mblue_over)
                    );
```

```
    //-------------------------------------------------------------
-------------
    //VGA 时序控制模块--完成时序扫描，输出 LCD 显示的 RGB 数值及水平和垂直方向的
同步信号
    wire [10:0]oH_cnt;  //行计数器--VGA 扫描点的 x 坐标
    wire [9:0]oV_cnt;       //列计数器--VGA 扫描点的 y 坐标

    vga_controller      vga_controller_inst(
                         .iCLK(ext_clk_25m),
                         .iRST_n(ext_rst_n),
                         .iR(mred),
                         .iG(mgreen),
                         .iB(mblue),
                         .oVGA_R(vga_r),
                         .oVGA_G(vga_g),
                         .oVGA_B(vga_b),
                         .oVGA_HS(vga_hsy),
                         .oVGA_VS(vga_vsy),
                         .oH_cnt(oH_cnt),
                         .oV_cnt(oV_cnt)
                         );

endmodule
```

2.3 顶层 RTLView



2.4 各子模块功能说明

模块	名称	功能
VGA 时序控制模块	vga_controller	完成时序扫描，输出 LCD 显示的 RGB 数值及水平方向和垂直方向的同步信号
游戏启动模块	game_start	键控开始游戏
“游戏开始”字符产生模块	color_gen_char1	生成游戏启动页面的“游戏开始”字符
“游戏结束”字符产生模块	color_gen_char2	生成游戏结束页面的“游戏结束”字符
颜色产生模块	color_gen	显示球色，挡板色，背景色（模式不同背景色可选）
弹球生成模块	Ball	控制弹球大小、位置、位置更新、输出标志
弹球速度键控模块	Ball_speed	通过按键改变弹球在 x 和 y 方向的速度
挡板生成模块	block	通过按键改变挡板大小，通过左右移信号实现左右移动
挡板键控模块	block_move	按下按键，输出左移和右移信号
背景颜色选择模块	Background_Color	键控产生背景颜色切换信号
弹球游戏运行模块	state_run	通过状态机控制游戏启动、运行、结束的界面切换
键值采集和消抖模块	arykeyscan	对矩阵键盘进行键值采集和消抖处理，输出键控指令
VGA 显示参数定义	LTM_Param.h	对本实验所用 VGA 显示驱动的具体参数以及小球相关坐标进行定义
字符点阵字模定义	Begin.h	对“游戏开始”和“游戏结束”页面的字符点阵字模进行定义

3. vga_controller—VGA 时序控制模块

3.1 模块说明

此模块是对 VGA 时序扫描原理的具体实现，对实际参数进行定义，输出 LCD 显示的具体 RGB 数值和生成行同步信号与场同步信号，并且对 VGA 扫描中的实时行坐标和纵坐标进行输出，供其他模块使用。

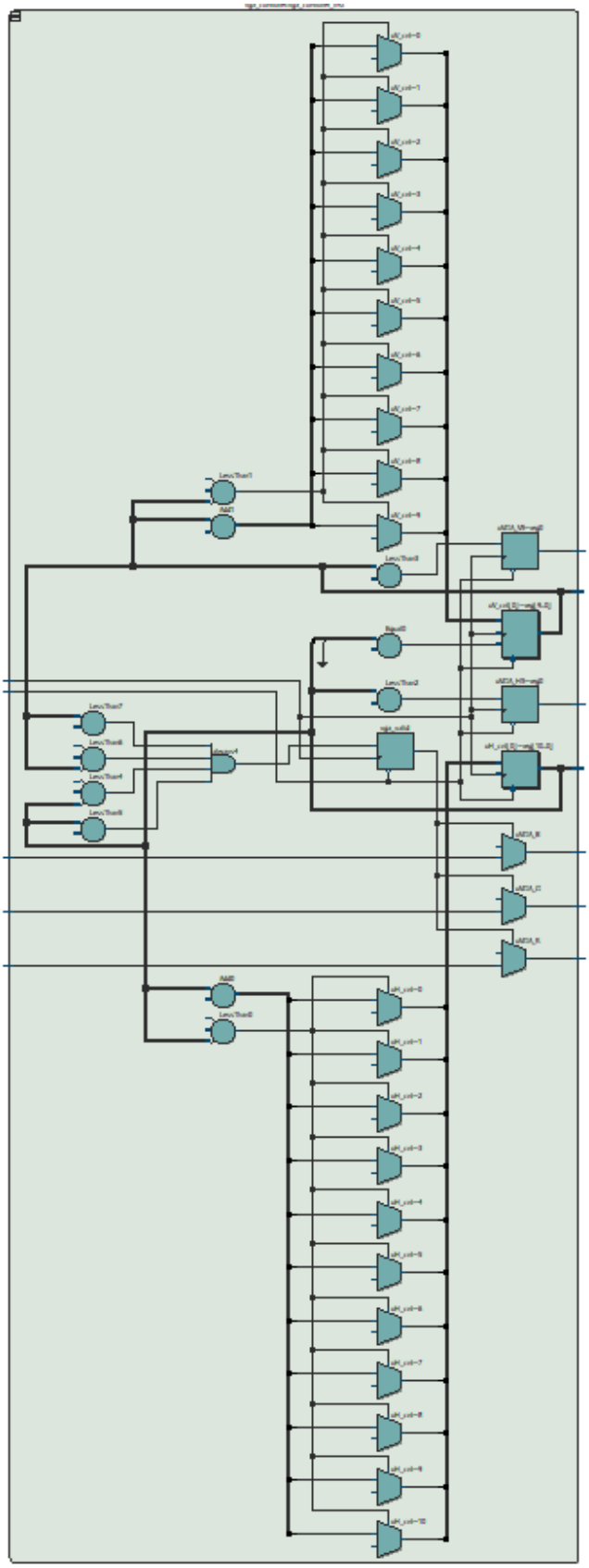
3.2 模块代码

```
//VGA 时序控制模块--完成时序扫描，输出 LCD 显示的 RGB 数值及水平方向和垂直方向的同步信号
module vga_controller(
          input iCLK,                //PLL 输出 25MHz 时钟
          input iRST_n,              //复位信号，低电平有效
          input iR,
          input iG,
          input iB,
          output oVGA_R,
          output oVGA_G,
          output oVGA_B,
          output reg oVGA_HS,     //行同步
          output reg oVGA_VS,        //场同步
          output reg [10:0]oH_cnt,//行计数器
          output reg [9:0]oV_cnt  //列计数器
          );

   //-------------------------------------------------------------
   `define VGA_640_480

   //-------------------------------------------------------------
   `ifdef VGA_640_480
      //VGA Timing 640*480 & 25MHz & 60Hz
      parameter VGA_HTT = 12'd800-12'd1;     //Hor Total Time
      parameter VGA_HST = 12'd96;           //Hor Sync  Time
      parameter VGA_HBP = 12'd48;//+12'd16;   //Hor Back Porch
      parameter VGA_HVT = 12'd640;             //Hor Valid Time
      parameter VGA_HFP = 12'd16;           //Hor Front Porch

      parameter VGA_VTT = 12'd525-12'd1;     //Ver Total Time
      parameter VGA_VST = 12'd2;               //Ver Sync Time
      parameter VGA_VBP = 12'd33;//-12'd4;    //Ver Back Porch
```

```
    parameter VGA_VVT = 12'd480;              //Ver Valid Time
    parameter VGA_VFP = 12'd10;             //Ver Front Porch
`endif

always @(posedge iCLK or negedge iRST_n)
    if(!iRST_n)
        oH_cnt <= 12'd0;
    else if(oH_cnt >= VGA_HTT)
        oH_cnt <= 12'd0;
    else
        oH_cnt <= oH_cnt+1'b1;

always @(posedge iCLK or negedge iRST_n)
    if(!iRST_n)
        oV_cnt <= 12'd0;
    else if(oH_cnt == VGA_HTT)
        begin
            if(oV_cnt >= VGA_VTT) oV_cnt <= 12'd0;
            else oV_cnt <= oV_cnt+1'b1;
        end
    else ;

//-----------------------------------------------------------
//行、场同步信号生成
always @(posedge iCLK or negedge iRST_n)
    if(!iRST_n)
        oVGA_HS <= 1'b0;
    else if(oH_cnt < VGA_HST)
        oVGA_HS <= 1'b1;
    else
        oVGA_HS <= 1'b0;

always @(posedge iCLK or negedge iRST_n)
    if(!iRST_n)
        oVGA_VS <= 1'b0;
    else if(oV_cnt < VGA_VST)
        oVGA_VS <= 1'b1;
    else
        oVGA_VS <= 1'b0;

//-----------------------------------------------------------
//显示有效区域标志信号生成
reg vga_valid;  //显示区域内，该信号高电平
```

```
    always @(posedge iCLK or negedge iRST_n)
        if(!iRST_n)
            vga_valid <= 1'b0;
        else if((oH_cnt >= (VGA_HST+VGA_HBP)) && (oH_cnt <
(VGA_HST+VGA_HBP+VGA_HVT))
                    && (oV_cnt >= (VGA_VST+VGA_VBP)) && (oV_cnt <
(VGA_VST+VGA_VBP+VGA_VVT)))
            vga_valid <= 1'b1;
        else
            vga_valid <= 1'b0;

    assign oVGA_R = vga_valid ? iR:1'b0;
    assign oVGA_G = vga_valid ? iG:1'b0;
    assign oVGA_B = vga_valid ? iB:1'b0;

endmodule
```

3.3 RTLView



4. game_start—游戏启动模块

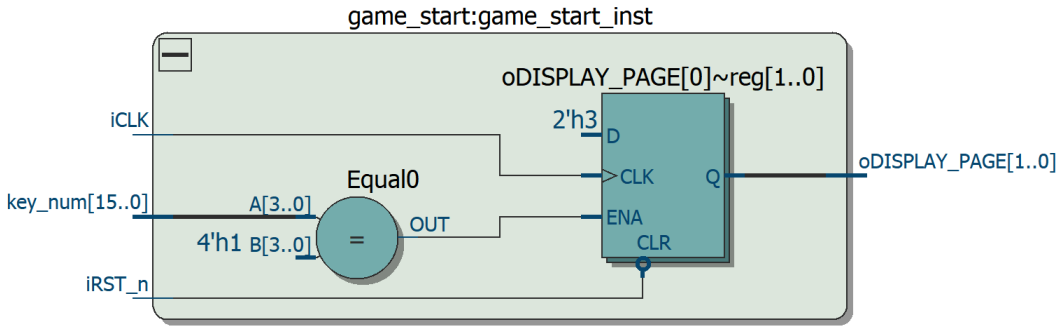
4.1 模块说明

检测按键输入，若为 S9 按下则输出游戏页面切换信号到弹球游戏运行模块，从游戏开始页面切换到游戏运行页面。

4.2 模块代码

```
//游戏启动模块--键控开始游戏
module game_start(iCLK, iRST_n, key_num, oDISPLAY_PAGE);
    input iCLK;
    input iRST_n;
    input [15:0]key_num;
    output [1:0]oDISPLAY_PAGE;

    reg [1:0]oDISPLAY_PAGE;

    always @(posedge iCLK or negedge iRST_n)
        begin
            if(!iRST_n)
                oDISPLAY_PAGE<=2'b00;
            else if(key_num[3:0]==4'h8)      //按下按键 S9 后，进入游戏启动界面
                oDISPLAY_PAGE<=2'b11;
        end

endmodule
```

4.3 RTLView



5. color_gen_char1—“游戏开始”字符产生模块

5.1 模块说明

通过“Begin.h”中定义的“游戏开始”字符显示的字模，相应的输出 RGB 值，对于字符输出红色，非字符显示区域输出蓝色。

5.2 模块代码

```
//“游戏开始”字符产生模块--游戏启动页面
module color_gen_char1(iVGA_X,iVGA_Y,oVGA_R,oVGA_G,oVGA_B);
    input [10:0]iVGA_X;     //行列扫描对应点坐标值
    input [9:0]iVGA_Y ;
    output reg oVGA_R;
    output reg oVGA_G;
    output reg oVGA_B;

    `include "Begin.h"          //包含的头文件为所显示字符的点阵数值

    always@(iVGA_X or iVGA_Y)
        begin
            if((iVGA_Y<CHAR_START_Y)||(iVGA_Y>(CHAR_START_Y+CHAR_Y-
1))||(iVGA_X<CHAR_START_X)||
            (iVGA_X>(CHAR_START_X+CHAR_X-1)))    //非字符显示区域，输出相
应的颜色
                begin
                    oVGA_R=1'b0;
                    oVGA_G=1'b0;
                    oVGA_B=1'b1;
                end
            else     //在字符显示区域显示相应的颜色
                begin
                    case(iVGA_Y)  //case 语句判断显示的行数，实现字符的点阵输
出

                    231:
                    begin
                        oVGA_R={1{charline_a0[CHAR_START_X+CHAR_X-
iVGA_X-1]}}; //R 分量的值为字模点阵的值，用红色字体显示
                        oVGA_G={1{1'b0}};
                        oVGA_B={1{1'b0}};
                    end
```

```
                232:
                begin
                    oVGA_R={1{charline_a1[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                233:
                begin
                    oVGA_R={1{charline_a2[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                234:
                begin
                    oVGA_R={1{charline_a3[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                235:
                begin
                    oVGA_R={1{charline_a4[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                236:
                begin
                    oVGA_R={1{charline_a5[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                237:
                begin
```

```
                    oVGA_R={1{charline_a6[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                238:
                begin
                    oVGA_R={1{charline_a7[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                239:
                begin
                    oVGA_R={1{charline_a8[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                240:
                begin
                    oVGA_R={1{charline_a9[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                241:
                begin
                    oVGA_R={1{charline_a10[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                242:
                begin
                    oVGA_R={1{charline_a11[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
```

```
                end

                243:
                begin
                    oVGA_R={1{charline_a12[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                244:
                begin
                    oVGA_R={1{charline_a13[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                245:
                begin
                    oVGA_R={1{charline_a14[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                246:
                begin
                    oVGA_R={1{charline_a15[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                247:
                begin
                    oVGA_R={1{charline_a16[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                248:
                begin
```

```
                    oVGA_R={1{charline_a17[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                249:
                begin
                    oVGA_R={1{charline_a18[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                250:
                begin
                    oVGA_R={1{charline_a19[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                251:
                begin
                    oVGA_R={1{charline_a20[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                252:
                begin
                    oVGA_R={1{charline_a21[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                253:
                begin
                    oVGA_R={1{charline_a22[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
```

```
                end

                254:
                begin
                    oVGA_R={1{charline_a23[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                255:
                begin
                    oVGA_R={1{charline_a24[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                256:
                begin
                    oVGA_R={1{charline_a25[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                257:
                begin
                    oVGA_R={1{charline_a26[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                258:
                begin
                    oVGA_R={1{charline_a27[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                259:
                begin
```

```
                    oVGA_R={1{charline_a28[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                260:
                begin
                    oVGA_R={1{charline_a29[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                261:
                begin
                    oVGA_R={1{charline_a30[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                262:
                begin
                    oVGA_R={1{charline_a31[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                endcase
            end
        end

endmodule
```

6. color_gen_char2—“游戏结束”字符产生模块

6.1 模块说明

通过“Begin.h”中定义的“游戏结束”字符显示的字模，相应的输出 RGB 值，对于字符输出红色，非字符显示区域输出蓝色。

6.2 模块代码

```
//"游戏结束"字符产生模块--游戏结束页面
module color_gen_char2(iVGA_X,iVGA_Y,oVGA_R,oVGA_G,oVGA_B);
    input [10:0]iVGA_X;     //行列扫描对应点坐标值
    input [9:0]iVGA_Y ;
    output reg oVGA_R;
    output reg oVGA_G;
    output reg oVGA_B;

    `include "Begin.h"          //包含的头文件为所显示字符的点阵数值

    always@(iVGA_X or iVGA_Y)
        begin
            if((iVGA_Y<CHAR_START_Y)||(iVGA_Y>(CHAR_START_Y+CHAR_Y-
1))||(iVGA_X<CHAR_START_X)||
            (iVGA_X>(CHAR_START_X+CHAR_X-1)))   //非字符显示区域，输出相
应的颜色
                begin
                    oVGA_R=1'b0;
                    oVGA_G=1'b0;
                    oVGA_B=1'b1;
                end
            else     //在字符显示区域显示相应的颜色
                begin
                    case(iVGA_Y)  //case 语句判断显示的行数，实现字符的点阵输
出

                    231:
                    begin
                        oVGA_R={1{charline_b0[CHAR_START_X+CHAR_X-
iVGA_X-1]}}; //R 分量的值为字模点阵的值，用红色字体显示
                        oVGA_G={1{1'b0}};
                        oVGA_B={1{1'b0}};
                    end

                    232:
                    begin
                        oVGA_R={1{charline_b1[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                        oVGA_G={1{1'b0}};
                        oVGA_B={1{1'b0}};
                    end
```

```
                233:
                begin
                    oVGA_R={1{charline_b2[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                234:
                begin
                    oVGA_R={1{charline_b3[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                235:
                begin
                    oVGA_R={1{charline_b4[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                236:
                begin
                    oVGA_R={1{charline_b5[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                237:
                begin
                    oVGA_R={1{charline_b6[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                238:
                begin
                    oVGA_R={1{charline_b7[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
```

```
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                239:
                begin
                    oVGA_R={1{charline_b8[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                240:
                begin
                    oVGA_R={1{charline_b9[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                241:
                begin
                    oVGA_R={1{charline_b10[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                242:
                begin
                    oVGA_R={1{charline_b11[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                243:
                begin
                    oVGA_R={1{charline_b12[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end
```

```
                244:
                begin
                    oVGA_R={1{charline_b13[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                245:
                begin
                    oVGA_R={1{charline_b14[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                246:
                begin
                    oVGA_R={1{charline_b15[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                247:
                begin
                    oVGA_R={1{charline_b16[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                248:
                begin
                    oVGA_R={1{charline_b17[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                249:
                begin
                    oVGA_R={1{charline_b18[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
```

```
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                250:
                begin
                    oVGA_R={1{charline_b19[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                251:
                begin
                    oVGA_R={1{charline_b20[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                252:
                begin
                    oVGA_R={1{charline_b21[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                253:
                begin
                    oVGA_R={1{charline_b22[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                254:
                begin
                    oVGA_R={1{charline_b23[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end
```

```
                255:
                begin
                    oVGA_R={1{charline_b24[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                256:
                begin
                    oVGA_R={1{charline_b25[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                257:
                begin
                    oVGA_R={1{charline_b26[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                258:
                begin
                    oVGA_R={1{charline_b27[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                259:
                begin
                    oVGA_R={1{charline_b28[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                260:
                begin
                    oVGA_R={1{charline_b29[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
```

```
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                261:
                begin
                    oVGA_R={1{charline_b30[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                262:
                begin
                    oVGA_R={1{charline_b31[CHAR_START_X+CHAR_X-
iVGA_X-1]}};
                    oVGA_G={1{1'b0}};
                    oVGA_B={1{1'b0}};
                end

                endcase
            end
        end

endmodule
```

7. color_gen—颜色产生模块

7.1 模块说明

此模块负责生成小球、挡板以及背景的颜色，若当前扫描坐标为小球区域，则输出白色，若当前扫描坐标为挡板区域，则输出蓝色，若两者都不是，则为背景区域，则输出背景色，并且背景色可根据输入的背景切换指令进行切换，默认为蓝绿色。

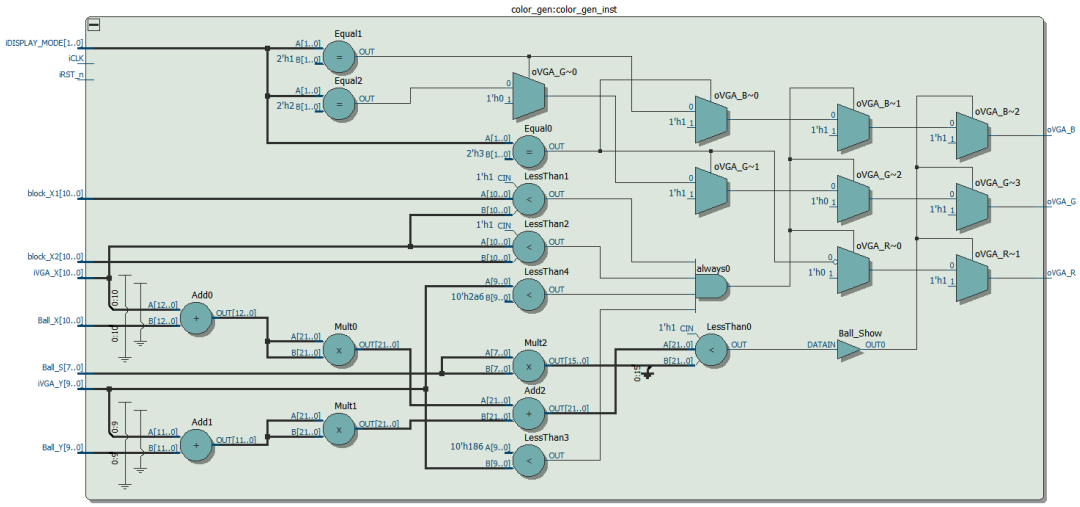
7.2 模块代码

```
//颜色产生模块  显示球色，挡板色，背景色（模式不同背景色可选）
module
color_gen(iCLK,iRST_n,Ball_X,Ball_Y,iVGA_X,iVGA_Y,Ball_S,block_X1,b
lock_X2,iDISPLAY_MODE,oVGA_R,oVGA_G,oVGA_B);
```

```
`include "LTM_Param.h"

    input iCLK, iRST_n;
    input [10:0]Ball_X;
    input [9:0]Ball_Y;
    input [10:0]iVGA_X;
    input [9:0]iVGA_Y;
    input [7:0]Ball_S;
    input [10:0]block_X1,block_X2;
    input [1:0]iDISPLAY_MODE;
    output reg oVGA_R;
    output reg oVGA_G;
    output reg oVGA_B;

    (*keep*)wire Ball_Show;
    wire[21:0]Delta_X2,Delta_Y2,R2;

    assign Delta_X2=(iVGA_X-Ball_X)*(iVGA_X-Ball_X);
    assign Delta_Y2=(iVGA_Y-Ball_Y)*(iVGA_Y-Ball_Y);
    assign R2=(Ball_S*Ball_S);
    assign Ball_Show =(Delta_X2+Delta_Y2)<=R2?1'b1:1'b0;

    always@(Ball_Show,iVGA_X,iVGA_Y,iDISPLAY_MODE)
        begin
            if(Ball_Show) //显示球
                begin
                    oVGA_R=1'b1;
                    oVGA_G=1'b1;
                    oVGA_B=1'b1;
                end
            else
                begin

if((iVGA_X>=block_X1)&&(iVGA_X<=block_X2)&&(iVGA_Y>390)&&(iVGA_Y<40
5)) //挡板位置
                        begin
                            oVGA_R={1{1'b0}};
                            oVGA_G={1{1'b0}};
                            oVGA_B={1{1'b1}};
                        end
                    else
                        begin
                            if(iDISPLAY_MODE==2'b11)  //蓝绿色
```

```
                    begin
                        oVGA_R=1'b0;
                        oVGA_G=1'b1;
                        oVGA_B=1'b1;
                    end
                else if(iDISPLAY_MODE==2'b10) //品红色
                    begin
                        oVGA_R=1'b1;
                        oVGA_G=1'b0;
                        oVGA_B=1'b1;
                    end
                else if(iDISPLAY_MODE==2'b01) //黄色
                    begin
                        oVGA_R=1'b1;
                        oVGA_G=1'b1;
                        oVGA_B=1'b0;
                    end
                else
                    begin       //红色
                        oVGA_R=1'b1;
                        oVGA_G=1'b0;
                        oVGA_B=1'b0;
                    end
            end
        end
    end
endmodule
```

7.3 RTLView



8. Ball—弹球生成模块

8.1 模块说明

此模块负责根据输入的半径、运动步长、挡板位置而生成弹球的大小，位置坐标，位置更新，以及输出弹球的标志位。实现弹球游戏运行的主要逻辑，弹球在触碰到上左右三壁以及挡板的情况下弹回继续运动，若没有被挡板接住而掉下，则游戏结束。并将弹球的实时坐标和标志位输出供其他模块使用。

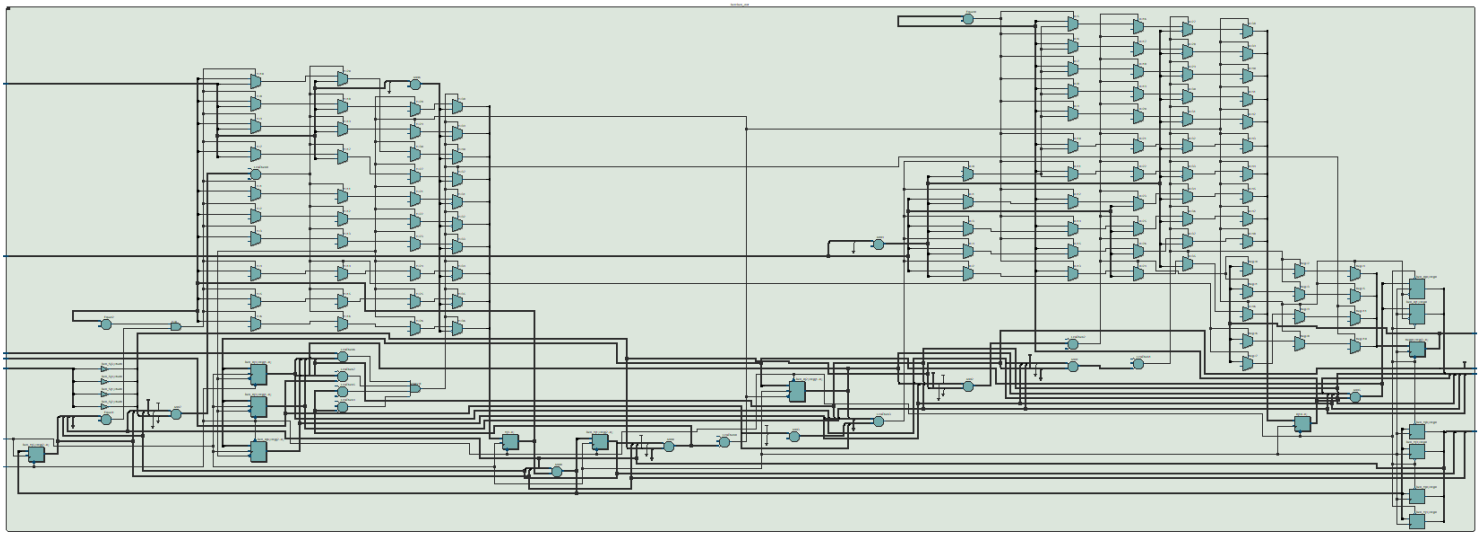
8.2 模块代码

```
//弹球生成模块--控制弹球大小、位置、位置更新、输出标志
module
Ball(iCLK,iRST_n,Ball_S_in,X_Step,Y_Step,block_X1,block_X2,Ball_X,B
all_Y,Ball_S,flag);

    `include "LTM_Param.h"

    input iCLK,iRST_n;
    input [3:0]Ball_S_in;
    input [3:0]X_Step;
    input [3:0]Y_Step;
    input [10:0]block_X1;
    input [10:0]block_X2;
    output [10:0]Ball_X;
    output [9:0]Ball_Y;
    output [7:0]Ball_S;
    output [3:0]flag;

    //中间变量
    (*keep*)wire[7:0]Ball_S;
    reg [10:0]X;        //球在 x 轴的增量
    reg [10:0]Ball_X;  //球在 x 轴的位置
    reg [9:0]Y;          //球在 y 轴的增量
    reg [9:0]Ball_Y;     //球在 y 轴的位置
    reg [3:0]flag;       //标志位

    assign Ball_S={3'b000,Ball_S_in,1'b1}; //球大小的赋值，最小 1 个像素

    always@(posedge iCLK or negedge iRST_n)  //球的 x 轴坐标位置的输出
       begin
          if(!iRST_n) // 异步复位
```

```
            begin
                Ball_X<=Ball_X_Center; //球的默认位置在屏中央
                X<=0;                  //x 方向移动速度为 0
                flag[1:0]<=2'b00;          //标志位的默认输出
            end
        else if(Ball_Y+Ball_S>=Ball_Y_Max) //球没有被挡板挡住,掉下去
了
            begin
                X<=11'b0;                   //球掉到底部，x 轴方向速度为 0
                flag[1:0]<=2'b11;                 //输出标志位
            end
        else
            begin
                if(Ball_X+Ball_S>=Ball_X_Max)           //球到达最右边
                    begin
                        X<=~{7'b0000000,X_Step}+11'b1;      //x 轴步进
变为负
                        flag[1:0]<=2'b01;              //输出标志位
                    end
                else
                    begin
                        if(Ball_X-Ball_S<=Ball_X_Min) //球到达最左边
                            begin
                                X<={7'b0000000,X_Step};         //x 轴步
进变为正
                                flag[1:0]<=2'b10;           //输出标志位
                            end
                        else
                            begin
                                X<=(X==11'b0)?((Ball_X<block_X2-
20)?(~{7'b0000000,X_Step}+11'b1):({7'b0000000,X_Step})):X;  //判断球
在中间位置时，让球动起来
                            end
                    end

                Ball_X<=Ball_X+X; //更新弹球 X 轴位置
            end
    end

    always@(posedge iCLK or negedge iRST_n) //球的 y 轴坐标位置的输出
        begin
            if(!iRST_n)
                begin
                    Ball_Y<=Ball_Y_Center;      //球的默认位置在屏中央
```

```
                Y<=0;                                   //y 方向移动速度为 0
                flag[3:2]<=2'b00;               //标志位的默认输出
            end
        else
            begin

if((Ball_Y+Ball_S>=390)&&(Ball_Y+Ball_S<=400)&&(Ball_X>block_X1)&&(
Ball_X<=block_X2))   //球被挡板挡住
                begin
                    Y<=~{6'b000000,Y_Step}+10'b1;       //y 轴步进
变为负
                    flag[3:2]<=2'b01;                   //输出相应标
志位
                end
            else if(Ball_Y+Ball_S>=Ball_Y_Max)  //球没有被挡板挡
住，掉下去了
                begin
                    Y<=0;                                   //球掉到底
部，y 轴方向速度为 0
                    flag[3:2]<=2'b11;                   //输出标志位
                end
            else
                begin
                    if(Ball_Y-Ball_S<=Ball_Y_Min) //球碰到顶部
                        begin
                            Y<={6'b000000,Y_Step};  //y 轴步进变为正
                            flag[3:2]<=2'b10;       //输出标志位
                        end
                    else
                        begin

Y<=((Ball_Y==Ball_Y_Center)&&(Y==10'b0))?{6'b000000,Y_Step}:Y;
//判断球在中间位置时，让球动起来
                        end
                end

            Ball_Y<=Ball_Y+Y;  //更新弹球 Y 轴位置
        end
    end

endmodule
```

8.3 RTLView



9. Ball_speed—弹球速度键控模块

9.1 模块说明

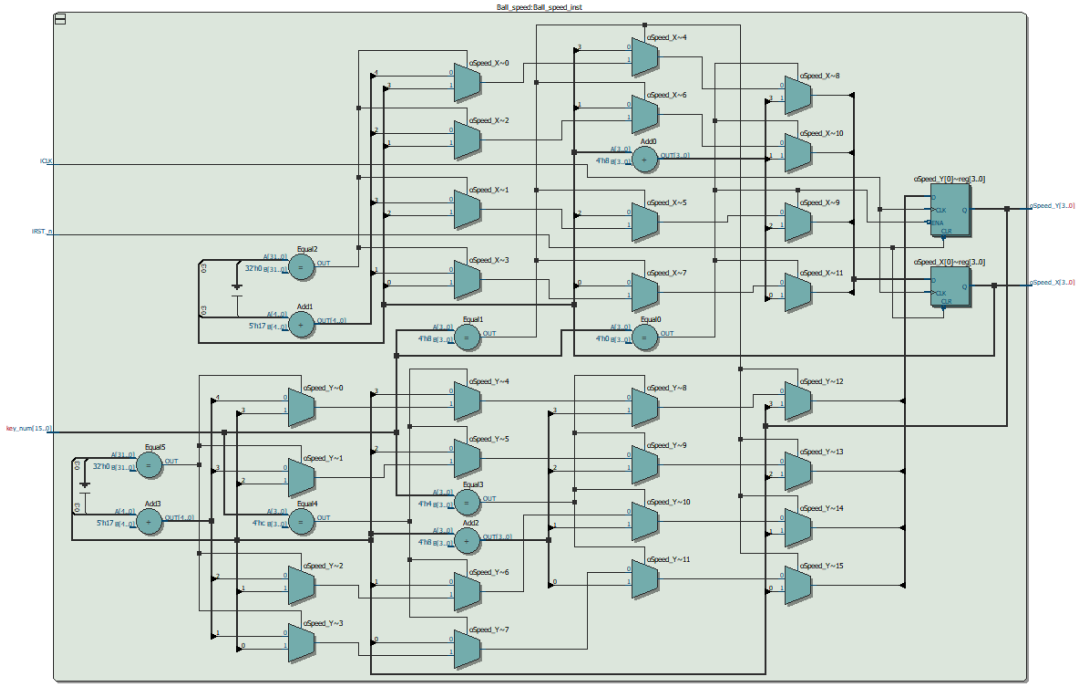
检测按键输入，通过 S1、S2 按键增大和减小弹球在 x 方向的运动速度，通过 S3、S4 按键增大和减小弹球在 y 方向的运动速度。根据具体按键指令，更新弹球在 x 方向和 y 方向的运动步长，输出到弹球生成模块。

9.2 模块代码

```
//弹球速度键控模块--通过 S1、S2、S3、S4 按键改变弹球在 x 和 y 方向的速度
module Ball_speed(iCLK,iRST_n,key_num,oSpeed_X,oSpeed_Y);
    input iCLK,iRST_n;
    input [15:0]key_num;
    output [3:0]oSpeed_X;
    output [3:0]oSpeed_Y;

    reg [3:0]oSpeed_X;
    reg [3:0]oSpeed_Y;

    always@(posedge iCLK or negedge iRST_n)
       begin
          if(!iRST_n)
             begin
```

```
                oSpeed_X<=0;
                oSpeed_Y<=0;
            end
        else
            begin
                if(key_num[3:0]==4'h0)      //按下 S1 按键时，增大 x 方向
的速度
                    oSpeed_X<=oSpeed_X+1;
                else if(key_num[3:0]==4'h1) //按下 S2 按键时，减小 x 方
向的速度
                    begin
                        if(oSpeed_X==0) //当 x 方向速度为 0 时，将不会继续
减小
                            oSpeed_X<=oSpeed_X;
                        else
                            oSpeed_X<=oSpeed_X-1;
                    end
                else if(key_num[3:0]==4'h2) //按下 S3 按键时，增大 y 方
向的速度
                    oSpeed_Y<=oSpeed_Y+1;
                else if(key_num[3:0]==4'h3) //按下 S4 按键时，减小 y 方
向的速度
                    begin
                        if(oSpeed_Y==0) //当 y 方向速度为 0 时，将不会继续
减小
                            oSpeed_Y<=oSpeed_Y;
                        else
                            oSpeed_Y<=oSpeed_Y-1;
                    end
                else
                    begin
                        oSpeed_X<=oSpeed_X;
                        oSpeed_Y<=oSpeed_Y;
                    end
            end
    end

endmodule
```

9.3 RTLView



10.block—挡板生成模块

10.1 模块说明

此模块负责生成挡板的大小，位置，位置更新。检测按键输入，通过 S7、S8 按键增大和减小挡板的长度。根据输入的左右移动信号实现挡板的左右移动。并将挡板的最左端坐标和最右端坐标输出供其他模块使用。

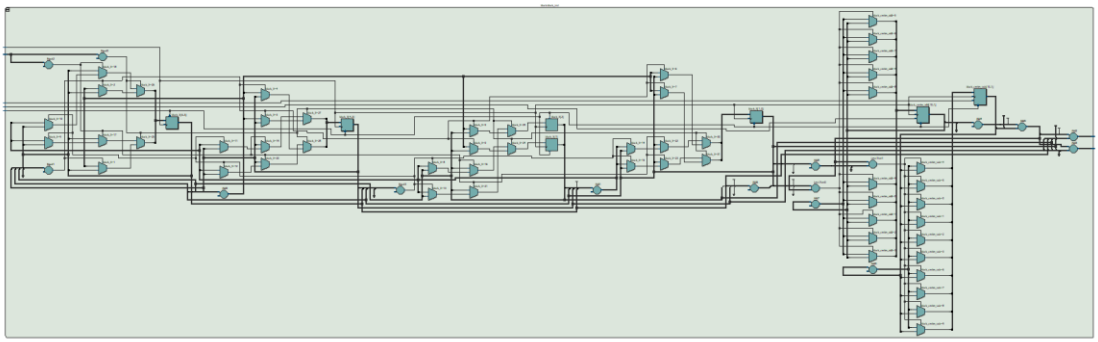
10.2 模块代码

```
//挡板生成模块--通过 S7、S8 按键改变挡板大小，通过左右移信号实现左右移动
module
block(iCLK,iRST_n,iMOVE_RIGHT,iMOVE_LEFT,key_num,block_X1,block_X2);

`include  "LTM_Param.h"

    input iCLK,iRST_n;
    input iMOVE_RIGHT;          //挡板右移信号
    input iMOVE_LEFT;           //挡板左移信号
    input [15:0]key_num;        //按下 S7、S8 按键，改变挡板大小
```

```
    output [10:0] block_X1;     //挡板最左端位置
    output [10:0] block_X2;     //挡板最右端位置

    reg [7:0]block_X;   //挡板大小的中间变量，表示挡板一半大小
    wire [10:0]block_center; //挡板中点坐标 X 轴位置变量

    reg [10:0]block_center_add,block_center_sub; //挡板中点坐标位置增减变
量，含义为挡板左右移动

    always@(posedge iCLK or negedge iRST_n) //实现挡板大小改变
        begin
            if(!iRST_n)
                block_X<=100;              //挡板默认长度为 100
            else
                begin
                    if(key_num[3:0]==4'h6)
                        begin
                            if(block_X==200) block_X<=block_X;      //挡板
达到最大长度 200，不再增长
                            else block_X<=block_X+5;                  //每
按一次 S7 键，挡板长度+5
                        end
                    else if(key_num[3:0]==4'h7)
                        begin
                            if(block_X==10) block_X<=block_X;       //挡板
达到最小长度 10，不再减短
                            else block_X<=block_X-5;                  //每
按一次 S8 键，挡板长度-5
                        end
                    else
                        block_X<=block_X;
                end
        end

    assign block_X1=block_center-block_X;
    assign block_X2=block_center+block_X;

    assign block_center=Ball_X_Center+block_center_add-
block_center_sub;

    always@(posedge iCLK or negedge iRST_n) //挡板右移
        begin
            if(!iRST_n) block_center_add=0;  //中心位置增量为 0
            else if(iMOVE_RIGHT)           //判断右移信号是否到来
```

```
            begin
                if(block_center>Ball_X_Max-block_X)          //判断是否
到最右
                    block_center_add<=block_center_add;      //是，则中
心保持不变
                else
                    block_center_add<=block_center_add+10; //否，则右
移 10
            end
        else block_center_add<=block_center_add;             //无右移信
号，坐标值不变
    end

  always@(posedge iCLK or negedge iRST_n) //挡板左移
    begin
        if(!iRST_n) block_center_sub<=0;    //中心位置减量为 0
        else if(iMOVE_LEFT)                 //判断左移信号是否到来
            begin
                if(block_center<Ball_X_Min+block_X)          //判断是否
到最左
                    block_center_sub<=block_center_sub;      //是，则中
心保持不变
                else
                    block_center_sub<=block_center_sub+10;  //否，则左
移 10
            end
        else block_center_sub<=block_center_sub;         //无左移信号，
坐标值不变
    end

endmodule
```

10.3 RTLView



11.block_move—挡板键控模块

11.1 模块说明

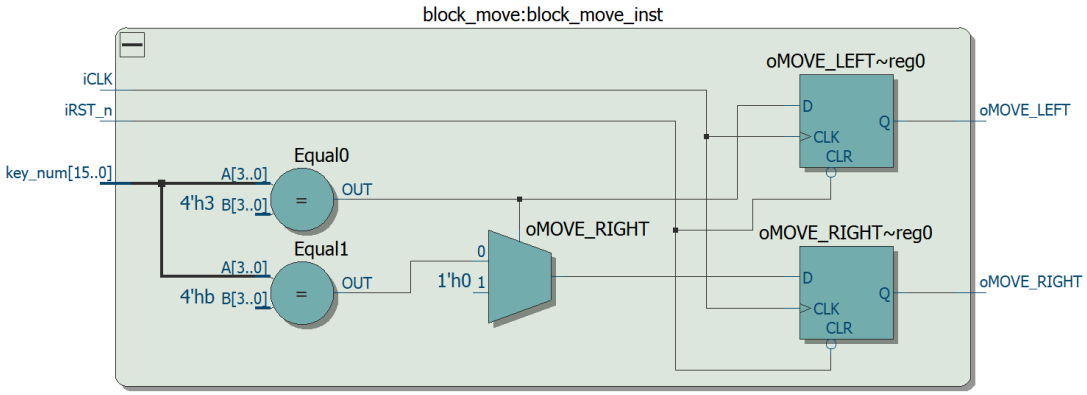
检测按键输入，通过 S13、S14 按键生成挡板左移和右移信号，并输出到挡板生成模块实现挡板的左右移动。

11.2 模块代码

```
//挡板键控模块--按下按键 S13、S14，输出左移和右移信号
module block_move(iCLK,iRST_n,key_num,oMOVE_LEFT,oMOVE_RIGHT);
    input iCLK,iRST_n;
    input [15:0]key_num;        //按键输入信号
    output oMOVE_LEFT;      //挡板左移输出信号
    output oMOVE_RIGHT;     //挡板右移输出信号

    reg oMOVE_LEFT;
    reg oMOVE_RIGHT;

    always@(posedge iCLK or negedge iRST_n) // 实现挡板左移和右移信号的
产生
        begin
            if(!iRST_n)
                begin
                    oMOVE_LEFT<=1'b0;       //默认无效输出
                    oMOVE_RIGHT<=1'b0;
                end
            else
                begin
                    if(key_num[3:0]==4'hC)      //按下 S13 键，挡板左移
                        begin
                            oMOVE_LEFT<=1'b1;
                            oMOVE_RIGHT<=1'b0;
                        end
                    else if(key_num[3:0]==4'hD) //按下 S14 键，挡板右移
                        begin
                            oMOVE_LEFT<=1'b0;
                            oMOVE_RIGHT<=1'b1;
                        end
                    else
                        begin
                            oMOVE_LEFT<=1'b0;
```

```
                        oMOVE_RIGHT<=1'b0;
                    end
                end
            end

endmodule
```

11.3 RTLView



12. Background_Color—背景颜色选择模块

12.1 模块说明

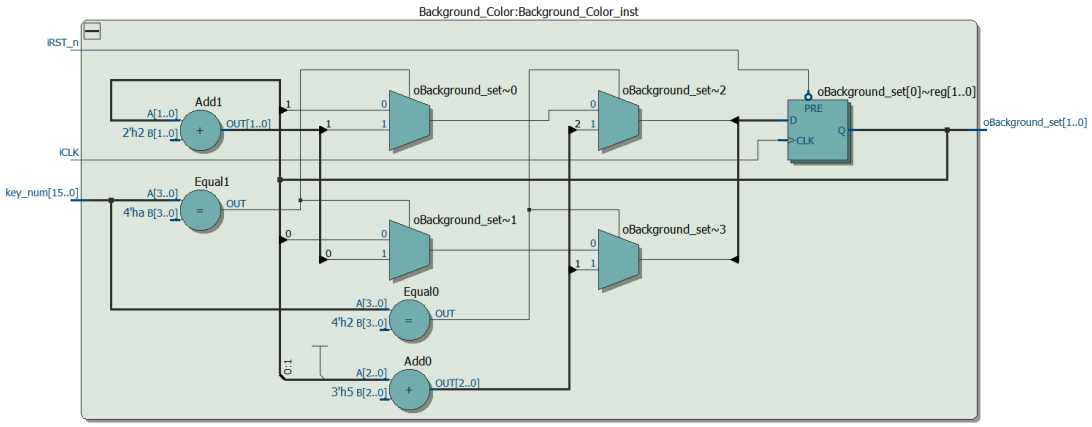
检测按键输入，通过 S5、S6 按键生成背景颜色切换信号，并输出到颜色生成模块实现背景颜色的切换。

12.2 模块代码

```
//背景颜色选择模块--键控产生背景颜色切换信号
module Background_Color(iCLK,iRST_n,key_num,oBackground_set);
    input iCLK, iRST_n;
    input [15:0]key_num;
    output [1:0]oBackground_set;

    reg [1:0]oBackground_set;

    always@(posedge iCLK or negedge iRST_n)
       begin
```

```
            if(!iRST_n)
                oBackground_set<=2'b11;
            else if(key_num[3:0]==4'h4)      //按下按键 S5 后，切换到上一个
背景颜色
                oBackground_set<=oBackground_set-1;
            else if(key_num[3:0]==4'h5)      //按下按键 S6 后，切换到下一个
背景颜色
                oBackground_set<=oBackground_set+1;
            else
                oBackground_set<=oBackground_set;
        end

endmodule
```

12.3 RTLView



13.state_run—弹球游戏运行模块

13.1 模块说明

此模块根据输入的页面切换信号、小球运动标志位、RGB 值，通过状态机控制实现三个页面（游戏开始页面、游戏运行页面、游戏结束页面）的切换，并将具体页面的 RGB 值输出到时序控制模块，进而输出到 LCD 显示屏。

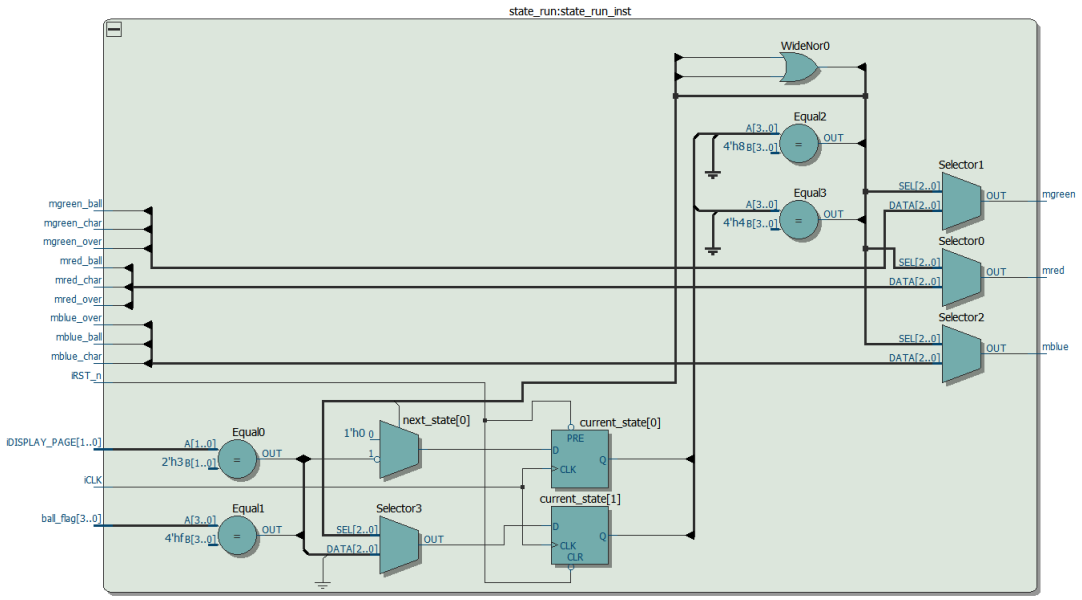
13.2 模块代码

```
//弹球游戏运行模块—通过状态机控制游戏启动、运行、结束的界面
```

```
module
state_run(iCLK,iRST_n,iDISPLAY_PAGE,ball_flag,mred_char,mgreen_char
,mblue_char,

mred_ball,mgreen_ball,mblue_ball,mred_over,mgreen_over,mblue_over,m
red,mgreen,mblue);
    input iCLK,iRST_n;
    input [1:0]iDISPLAY_PAGE;
    input [3:0]ball_flag;
    input mred_char,mgreen_char,mblue_char;
    input mred_ball,mgreen_ball,mblue_ball;
    input mred_over,mgreen_over,mblue_over;
    output reg mred;
    output reg mgreen;
    output reg mblue;

    parameter S0=4'b0001, S1=4'b0010, S2=4'b0100;
    reg [1:0] current_state, next_state;

    always @(posedge iCLK or negedge iRST_n)
        begin
            if(!iRST_n)
                current_state<=S0;
            else
                current_state<=next_state;
        end

    always@(current_state or iDISPLAY_PAGE or ball_flag or iRST_n)
        begin
            case(current_state)
            S0:
            begin    //S0 状态下屏幕显示游戏开始的界面
                mred=mred_char;
                mgreen=mgreen_char;
                mblue=mblue_char;
                if(iDISPLAY_PAGE==2'b11)  //判断如果有按下游戏开始按键
（S9），进入 S1 状态，游戏运行
                    next_state=S1;
                else next_state=S0;  //否则一直处于 S0 起始界面
            end

            S1:
            begin            // S1 状态下屏幕显示游戏运行的弹球界面
                mred=mred_ball;
```

```
                mgreen=mgreen_ball;
                mblue=mblue_ball;
                if(ball_flag==4'b1111) //判断球运行的标志，如果掉到底部，进
入 S2 状态，游戏结束
                    next_state=S2;
                else
                    next_state=S1;  //否则，在 S1 弹球运行界面
            end

            S2:
            begin               //S2 状态下屏幕显示游戏结束界面
                mred=mred_over;
                mgreen=mgreen_over;
                mblue=mblue_over;
                if(iRST_n) next_state=S2; //复位信号到来后，重新进入游戏开
始界面
                else next_state=S0;
            end

            default:
            begin       //default 默认 S2 状态
                next_state=S2;
                mred=mred_over;
                mgreen=mgreen_over;
                mblue=mblue_over;
            end
            endcase
        end

endmodule
```

13.3 RTLView



14.arykeyscan—键值采集和消抖模块

14.1 模块说明

此模块负责对开发板的矩阵键盘进行键值采集和消抖处理，将具体的键控指令输出到其他模块，以实现改变弹球速度、改变挡板长度、改变背景颜色、控制挡板左右移动、启动游戏等功能。

14.2 模块代码

```
//键值采集和消抖模块--产生键控指令
module arykeyscan(
          input clk,  //外部输入 25MHz 时钟信号
          input rst_n,    //外部输入复位信号，低电平有效
          input[3:0] key_v,   //4 个按键输入，未按下为高电平，按下后为低电平
          output reg[3:0] key_h,  //4 个行按键输出
          output reg[15:0] display_num    //数码管显示数据，[15:12]--数码管千位，[11:8]--数码管百位，[7:4]--数码管十位，[3:0]--数码管个位
          );

//----------------------------------------
//列按键键值采样
```

```
wire[3:0] keyv_value;       //列按键按下键值，高电平有效

sigkeyscan      uut_sigkeyscan(
                  .clk(clk),  //外部输入 25MHz 时钟信号
                  .rst_n(rst_n),  //外部输入复位信号，低电平有效
                  .key_v(key_v),  //4 个独立按键输入，未按下为高电平，按下后为
低电平
                  .keyv_value(keyv_value)     //列按键按下键值，高电平有效
               );

//-------------------------------------
//状态机采样键值
reg[3:0] nstate,cstate;
parameter K_IDLE = 4'd0;    //空闲状态，等待
parameter K_H1OL = 4'd1;    //key_h[0]拉低
parameter K_H2OL = 4'd2;    //key_h[1]拉低
parameter K_H3OL = 4'd3;    //key_h[2]拉低
parameter K_H4OL = 4'd4;    //key_h[3]拉低
parameter K_CHCK = 4'd5;

    //状态切换
always @(posedge clk or negedge rst_n)
    if(!rst_n) cstate <= K_IDLE;
    else cstate <= nstate;

always @(cstate or keyv_value or key_v)
    case(cstate)
        K_IDLE: if(keyv_value != 4'b0000) nstate <= K_H1OL;
                else nstate <= K_IDLE;
        K_H1OL: nstate <= K_H2OL;
        K_H2OL: if(key_v != 4'b1111) nstate <= K_IDLE;
                else nstate <= K_H3OL;
        K_H3OL: if(key_v != 4'b1111) nstate <= K_IDLE;
                else nstate <= K_H4OL;
        K_H4OL: if(key_v != 4'b1111) nstate <= K_IDLE;
                else nstate <= K_CHCK;
        K_CHCK: nstate <= K_IDLE;
        default: ;
    endcase

//-------------------------------------
//采样键值
reg[3:0] new_value; //新采样数据
reg new_rdy;    //新采样数据有效
```

```
always @(posedge clk or negedge rst_n)
    if(!rst_n) begin
        key_h <= 4'b0000;
        new_value <= 4'd0;
        new_rdy <= 1'b0;
    end
    else begin
        case(cstate)
            K_IDLE: begin
                key_h <= 4'b0000;
                new_value <= 4'd0;
                new_rdy <= 1'b0;
            end
            K_H1OL: begin
                key_h <= 4'b1110;
                new_value <= 4'd0;
                new_rdy <= 1'b0;
            end
            K_H2OL: begin
                case(key_v)
                    4'b1110: begin
                            key_h <= 4'b0000;
                            new_value <= 4'd0;
                            new_rdy <= 1'b1;
                        end
                    4'b1101: begin
                            key_h <= 4'b0000;
                            new_value <= 4'd1;
                            new_rdy <= 1'b1;
                        end
                    4'b1011: begin
                            key_h <= 4'b0000;
                            new_value <= 4'd2;
                            new_rdy <= 1'b1;
                        end
                    4'b0111: begin
                            key_h <= 4'b0000;
                            new_value <= 4'd3;
                            new_rdy <= 1'b1;
                        end
                    default: begin
                            key_h <= 4'b1101;
                            new_value <= 4'd0;
```

```
                    new_rdy <= 1'b0;
                end
            endcase
        end
        K_H3OL: begin
            case(key_v)
                4'b1110: begin
                        key_h <= 4'b0000;
                        new_value <= 4'd4;
                        new_rdy <= 1'b1;
                    end
                4'b1101: begin
                        key_h <= 4'b0000;
                        new_value <= 4'd5;
                        new_rdy <= 1'b1;
                    end
                4'b1011: begin
                        key_h <= 4'b0000;
                        new_value <= 4'd6;
                        new_rdy <= 1'b1;
                    end
                4'b0111: begin
                        key_h <= 4'b0000;
                        new_value <= 4'd7;
                        new_rdy <= 1'b1;
                    end
                default: begin
                        key_h <= 4'b1011;
                        new_value <= 4'd0;
                        new_rdy <= 1'b0;
                    end
            endcase
        end
        K_H4OL: begin
            case(key_v)
                4'b1110: begin
                        key_h <= 4'b0000;
                        new_value <= 4'd8;
                        new_rdy <= 1'b1;
                    end
                4'b1101: begin
                        key_h <= 4'b0000;
                        new_value <= 4'd9;
                        new_rdy <= 1'b1;
```

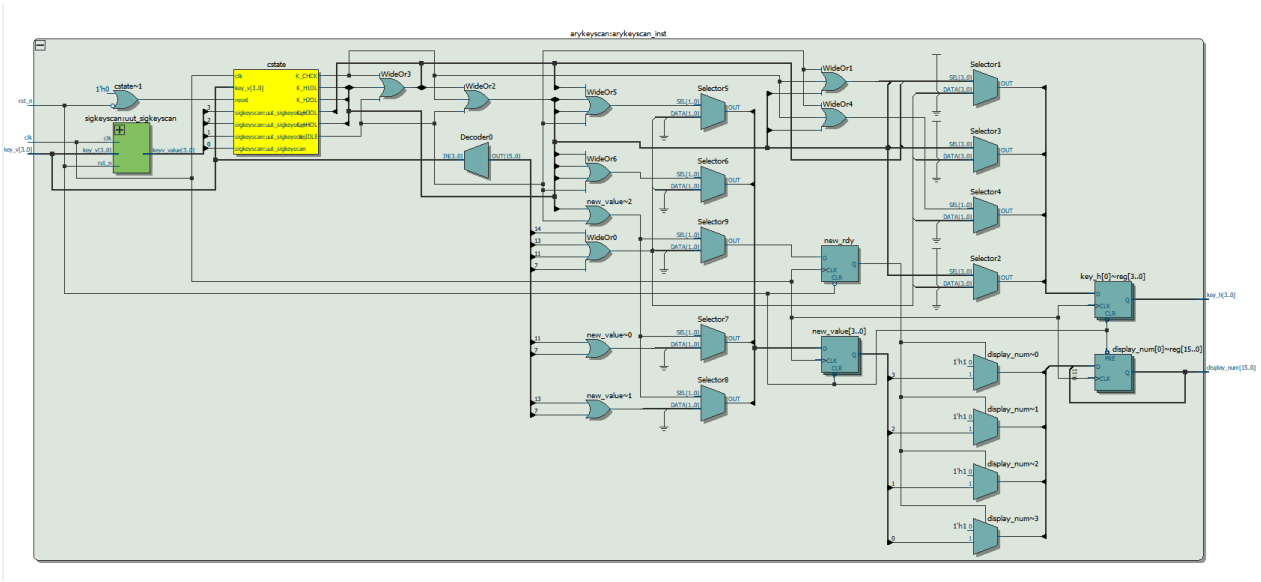
```
                end
            4'b1011: begin
                    key_h <= 4'b0000;
                    new_value <= 4'd10;
                    new_rdy <= 1'b1;
                end
            4'b0111: begin
                    key_h <= 4'b0000;
                    new_value <= 4'd11;
                    new_rdy <= 1'b1;
                end
            default: begin
                    key_h <= 4'b0111;
                    new_value <= 4'd0;
                    new_rdy <= 1'b0;
                end
        endcase
    end
    K_CHCK: begin
        case(key_v)
            4'b1110: begin
                    key_h <= 4'b0000;
                    new_value <= 4'd12;
                    new_rdy <= 1'b1;
                end
            4'b1101: begin
                    key_h <= 4'b0000;
                    new_value <= 4'd13;
                    new_rdy <= 1'b1;
                end
            4'b1011: begin
                    key_h <= 4'b0000;
                    new_value <= 4'd14;
                    new_rdy <= 1'b1;
                end
            4'b0111: begin
                    key_h <= 4'b0000;
                    new_value <= 4'd15;
                    new_rdy <= 1'b1;
                end
            default: begin
                    key_h <= 4'b0000;
                    new_value <= 4'd0;
                    new_rdy <= 1'b0;
```

```
                    end
                endcase
            end
            default: ;
        endcase
    end

//-------------------------------------
//产生最新键值

always @(posedge clk or negedge rst_n)
    if(!rst_n) display_num <= 16'hffff;
    else
        begin
            if(new_rdy) display_num <= {display_num[11:0],new_value};
            else display_num <= {display_num[11:0],4'b1111};
        end

endmodule
```

```
module sigkeyscan(
            input clk,  //外部输入 25MHz 时钟信号
            input rst_n,    //外部输入复位信号，低电平有效
            input [3:0]key_v,   //4 个列按键输入，未按下为高电平，按下后为低电平
            output [3:0]keyv_value      //列按键按下键值，高电平有效
            );

//-------------------------------------
//按键抖动判断逻辑
wire key;   //所有按键值相与的结果，用于按键触发判断
reg[3:0] keyr;  //按键值 key 的缓存寄存器

assign key = key_v[0] & key_v[1] & key_v[2] & key_v[3];

always @(posedge clk or negedge rst_n)
    if (!rst_n) keyr <= 4'b1111;
    else keyr <= {keyr[2:0],key};

wire key_neg = ~keyr[2] & keyr[3];  //有按键被按下
wire key_pos = keyr[2] & ~keyr[3];  //有按键被释放

//-------------------------------------
//定时计数逻辑，用于对按键的消抖判断
reg[19:0]  cnt;
```

```
    //按键消抖定时计数器
always @ (posedge clk or negedge rst_n)
   if (!rst_n) cnt <= 20'd0;
    else if(key_pos || key_neg) cnt <= 20'd0;
    else if(cnt < 20'd999_999) cnt <= cnt + 1'b1;
    else cnt <= 20'd0;

reg[3:0] key_value[1:0];

    //定时采集按键值
always @(posedge clk or negedge rst_n)
    if (!rst_n) begin
        key_value[0] <= 4'b1111;
        key_value[1] <= 4'b1111;
    end
    else begin
        key_value[1] <= key_value[0];
        if(cnt == 20'd999_999) key_value[0] <= key_v;    //定时键值采集
        else ;
    end

assign keyv_value = key_value[1] & ~key_value[0];       //消抖后按键值变化
标志位

endmodule
```

14.3 RTLView



15.LTM_Param.h—VGA 显示参数定义

15.1 内容说明

此文件对本实验所用 VGA 显示驱动的具体参数进行定义，包括行总长度、列总长度、行有效区域长度、列有效区域长度、行同步信号长度、场同步信号长度、列前沿、列后沿长度等参数。同时，对小球中心点坐标、运动区域上下左右的边界值进行定义。

15.2 文件内容

```
//LTM_Param.h
//////////////////////////////////////////////////////////////////////
////////
//parameter declarations
//////////////////////////////////////////////////////////////////////
////////
parameter H_LINE =800;                          //行的总点数
parameter V_LINE = 525;                         //列的总点数
parameter Hsync_Blank =140;                   //
parameter Hsync_Front_Porch =16;                 //
parameter Vertical_Back_Porch =33;            //
parameter Vertical_Front_Porch =10;           //
//////////////////////////////////////////////////////////////////////
////////
// Horizontal Parameter
parameter H_TOTAL =H_LINE-1;     //total-1 行点数的最大值
799
parameter H_SYNC =96 ;             //sync-1 行同步信号点数最大值
95
parameter H_START = 140;           //sync+back-1-1-delay  行有效区域起
始的点      140
parameter H_END =780 ;             //H_START+800 行有效区域结束的点
780
// Vertical Parameter
parameter V_TOTAL = V_LINE-1;    //total-1 列点数的最大值
524
parameter V_SYNC = 2;              //sync-1 列同步信号点数最大值
2
parameter V_START =34 ;           //sync+back-1  pre 2 lines；列有效区
域起始的点     34
```

```
parameter V_END=514 ;            //V_START+480  pre 2 lines；列有效区
域结束的点     514

parameter X_START =Hsync_Blank;
parameter Y_START = Vertical_Back_Porch;

parameter Ball_X_Center =X_START-1+((H_LINE-Hsync_Blank-
Hsync_Front_Porch)>>1);              //screen center  x
parameter Ball_Y_Center=Y_START-1+((V_LINE-Vertical_Back_Porch-
Vertical_Front_Porch)>>1);      //screen center  y
parameter Ball_X_Min = X_START-1;                 //screen left up
x
parameter Ball_Y_Min=Y_START-1;                   //screen left up
y
parameter Ball_X_Max=H_LINE-Hsync_Front_Porch-1;    //screen rigt
down x
parameter Ball_Y_Max=V_LINE-Vertical_Front_Porch-1;  //screen rigt
down y
```

16.Begin.h—字符点阵字模定义

16.1 内容说明

通过“PCtoLCD2020”软件获取“游戏开始”和“游戏结束”的具体字模，并在此文件进行参数定义，并对字符显示的起始坐标、字符长度宽度进行定义，供字符产生模块使用，通过 RGB 值的输出生成具体字符。

16.2 文件内容

```
//Begin.h
//*****************************************************************
parameter CHAR_START_X=400,CHAR_START_Y=231; //字符显示的起始坐标
parameter CHAR_X=128, CHAR_Y=32;            //字符的长度和宽度


//*****************************************************************
//定义"游戏开始"的字符点阵字模
parameter
    charline_a0=128'h00000000_00000000_00000000_00000000,
    charline_a1=128'h00000000_00000000_00000000_00000000,
    charline_a2=128'h00000000_00004000_00000000_01000800,
```

```
    charline_a3=128'h08100700_00007000_00000020_01C00E00,
    charline_a4=128'h060C0600_00006400_00000070_01801C00,
    charline_a5=128'h070E0400_00006380_1FFFFFF8_01801800,
    charline_a6=128'h03060C00_000061C0_00300C00_01801000,
    charline_a7=128'h03040818_001860E0_00300C00_03003000,
    charline_a8=128'h00008FFC_3FFC6060_00300C00_030020C0,
    charline_a9=128'h007FD000_00186000_00300C00_03086060,
    charline_a10=128'h20881010_00306018_00300C00_3FFC4038,
    charline_a11=128'h30883FF8_003021FC_00300C00_0208801C,
    charline_a12=128'h19084030_003BFE00_00300C00_061981FC,
    charline_a13=128'h19188060_10343000_00300C00_0619FE0C,
    charline_a14=128'h091FC040_0C203020_00300C18_04198008,
    charline_a15=128'h0118C180_06603070_3FFFFFFC_04180000,
    charline_a16=128'h0218C180_03603060_00300C00_0C100000,
    charline_a17=128'h02188180_01C010C0_00300C00_0C308010,
    charline_a18=128'h02188188_00C019C0_00300C00_0830FFF8,
    charline_a19=128'h0610BFFC_00E01980_00300C00_1820C030,
    charline_a20=128'h04108180_01F01B00_00300C00_1860C030,
    charline_a21=128'h3C308180_01300E00_00200C00_0C60C030,
    charline_a22=128'h0C308180_03380C04_00600C00_0340C030,
    charline_a23=128'h0C218180_02181E04_00600C00_00E0C030,
    charline_a24=128'h0C618180_04083704_00C00C00_00B8C030,
    charline_a25=128'h0C418180_0C086308_00800C00_019CC030,
    charline_a26=128'h1CC18180_08018188_01000C00_030CC030,
    charline_a27=128'h1C930180_100300EC_02000C00_0600FFF0,
    charline_a28=128'h0D0F0F80_2004007C_04000C00_0C00C030,
    charline_a29=128'h02060300_4018003C_18000C00_1000C030,
    charline_a30=128'h04000000_0000000C_20000800_20008000,
    charline_a31=128'h00000000_00000000_00000000_00000000;

//*****************************************************************
//定义"游戏结束"的字符点阵字模
parameter
    charline_b0=128'h00000000_00000000_00000000_00000000,
    charline_b1=128'h00000000_00000000_00000000_00000000,
    charline_b2=128'h00000000_00004000_00000800_00010000,
    charline_b3=128'h08100700_00007000_01800E00_0001C000,
    charline_b4=128'h060C0600_00006400_01800C00_00018000,
    charline_b5=128'h070E0400_00006380_01800C00_00018018,
    charline_b6=128'h03060C00_000061C0_03000C00_3FFFFFFC,
    charline_b7=128'h03040818_001860E0_03000C18_00018000,
    charline_b8=128'h00008FFC_3FFC6060_0207FFFC_00018000,
    charline_b9=128'h007FD000_00186000_04100C00_00018000,
    charline_b10=128'h20881010_00306018_041C0C00_02018080,
```

```
    charline_b11=128'h30883FF8_003021FC_08300C00_03FFFFE0,
    charline_b12=128'h19084030_003BFE00_18600C00_030180C0,
    charline_b13=128'h19188060_10343000_3FE00C00_030180C0,
    charline_b14=128'h091FC040_0C203020_1CC00C20_030180C0,
    charline_b15=128'h0118C180_06603070_0183FFF0_030180C0,
    charline_b16=128'h0218C180_03603060_01000000_030180C0,
    charline_b17=128'h02188180_01C010C0_02000000_03FFFFC0,
    charline_b18=128'h02188188_00C019C0_04010020_0307C0C0,
    charline_b19=128'h0610BFFC_00E01980_0879FFF0_030FA000,
    charline_b20=128'h04108180_01F01B00_1F818030_000DB000,
    charline_b21=128'h3C308180_01300E00_1C018020_00199000,
    charline_b22=128'h0C308180_03380C04_00018020_00318800,
    charline_b23=128'h0C218180_02181E04_00018020_00618E00,
    charline_b24=128'h0C618180_04083704_00198020_00C18700,
    charline_b25=128'h0C418180_0C086308_00E18020_018183C0,
    charline_b26=128'h1CC18180_08018188_0F818020_030181F0,
    charline_b27=128'h1C930180_100300EC_3C01FFE0_0601807C,
    charline_b28=128'h0D0F0F80_2004007C_10018020_18018030,
    charline_b29=128'h02060300_4018003C_00018020_20018000,
    charline_b30=128'h04000000_0000000C_00010000_00010000,
    charline_b31=128'h00000000_00000000_00000000_00000000;
```

17.实验过程问题及解决描述

整体来说，这次实验很有挑战性，也是对之前所学 FPGA 内容的一次很好的梳理和实践，在本次设计的过程中，对 VGA 时序扫描原理的理解至关重要。

本次实验中遇到的问题：

（1） 游戏运行异常，无法通过按键输入正常控制小球速度、挡板移动、背景切换等。

（2） 小球速度异常，显示一片色带而非正常的运动画面。

（3） 背景颜色异常，应该属于背景的颜色却显示为小球和挡板的颜色，背景切换时背景颜色不变，切换的是小球和挡板的颜色。

对应分析：

（1） 问题出在键值采集模块，因为键值采集后 key_num 变量会一直保存采集到的键值，直至下一个按键的输入才会更新，而时钟频率很快，因此会出现按一次按键，挡板长度却一下子从最小增到最大的情况，其他键控功能同理。

（2） 问题在于控制小球运动（弹球生成模块）所使用的时钟频率过快，一秒内会进行无数次的位置更新，因此一个很小的步长却会造成小球运动过快形成一片色带的现象。

（3） 问题出在时序控制模块的 VGA 参数定义，因为使用的是慕课上的时序控制模块，进行简单的参数修改，但是由于 VGA 显示驱动的差异，所使用的参数并不能匹配实验中实际使用的 VGA 驱动，因此会出现颜色显示的异常。

对应解决措施：

（1） 在键值采集模块中，每次采集到键值后的下一个时钟，便将采集到的键值清零，让每次采集的键值只作用一次。

（2） 根据同学的建议，将时序控制模块输出的场同步信号作为弹球运动的时钟，即解决问题。除此之外，另外设计一个分频模块生成一个小频率时钟也是可以的。

（3） 修改时序控制模块的参数定义，使用本实验中实际使用的 VGA 显示驱动的参数，生成正确的行同步信号和场同步信号，即可实现颜色的正常显示和切换。

参考文献

[1] LupinLeo. VGA 原理详解. CSDN 论坛. 2018-08-19.
https://blog.csdn.net/shichao1470/article/details/81840978

[2] 朱敏. EDA 技术与实验. 哈尔滨工业大学. 中国大学 MOOC.
https://www.icourse163.org/course/HIT-1003359013