浙江大学

课程设计报告

中文题目 ： 基于数字系统的躲避炸弹游戏

英文题目：Bomb-Dodging Game Based On a Digital System

小组成员 1: 董乙灿

学号: 3190105140

专业: 计算机科学与技术

学院: 计算机学院

小组成员 2: 张济开

学号: 3190103233

专业: 软件工程

学院: 计算机学院

指导教师: 洪奇军

报告提交日期 2021 年 01 月 19 日

目录

一、游戏设计背景

1.ISE 14.7 软件

本次设计所用的硬件编程软件为 Xilinx 公司设计的 ISE 14.7 软件，所使用的语言为 Verilog 语言。该软件的使用和 Verilog 语言的学习都较为简单，和 c 语言有相似之处。同方式或混合方式对设计建模。这些方式包括：行为描述方式建模；数据流方式建模；结构化方式建模等。Verilog HDL 中有两类数据类型：线网数据类型和寄存器数据类型。线网类型表示构件间的物理连线，而寄存器类型表示抽象的数据存储元件。通过模块化的设计与相应的外部接口，可以实现较好的人机交互。

本次设计通过行为描述的方式实现游戏的设计，显示器、PS2 外接设备优化游戏。

2.游戏介绍

本游戏名为《躲炸弹》，在游戏中我们需要控制主角史莱姆来躲避从四面八方随机生成飞来的炸弹，并尽可能躲避长久的时间，分数和时间挂钩，坚持的时间越久，获得的分数越高。

3.设计介绍

本次课程设计是利用 FPGA 板，通过底层硬件控制的方法实现躲炸弹这款游戏。在本组设计中，我们用外接的 ps2 键盘进行游戏的控制，采用四个按键（W、S、A、D），分别控制史莱姆进行上下左右的移动，来躲避从四面飞来的炸弹。当炸弹与史莱姆发生碰撞时，游戏结束；游戏界面由显示器显示，板子上的七段数码管显示游戏得分。

4.重难点分析

（1）在显示器上正确显示背景图、史莱姆图像、炸弹图像。

（2）能够通过对键盘的调用实现史莱姆的上下左右移动，画面可以及时对键盘传输的信号做出反应。

（3）能够从画面四周生成炸弹并使其周期性地运动。

（4）在炸弹和史莱姆发生碰撞的时候可以正确做出逻辑判断，切换到结束的场景并停止计分板和游戏的运行。

（5）炸弹和史莱姆的图片原本为矩形，在设计时用适当的方式选取采样点进行检测并进行碰撞条件判断。

二、整体结构

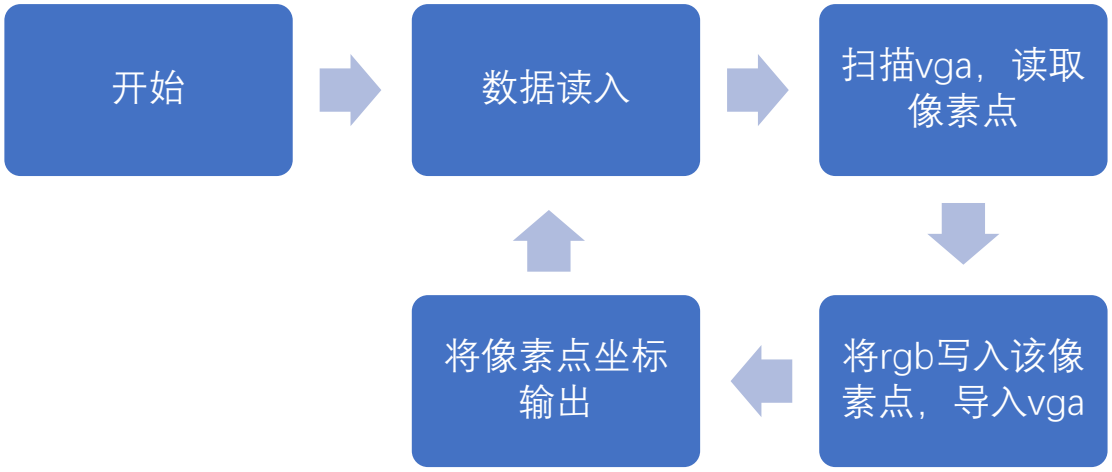
1.输入与输出

本游戏用外接键盘进行输入，采用四个按键（W、S、A、D），分别控制我方飞机进行上下左右的移动；空格键控制游戏从封面开始；rst 控制 gameover 后，重新回到主菜单。输出由两部分组成，一部分为 VGA 模块的显示器游戏界面输出；另一部分为板子上七段数码管显示的游戏得分。

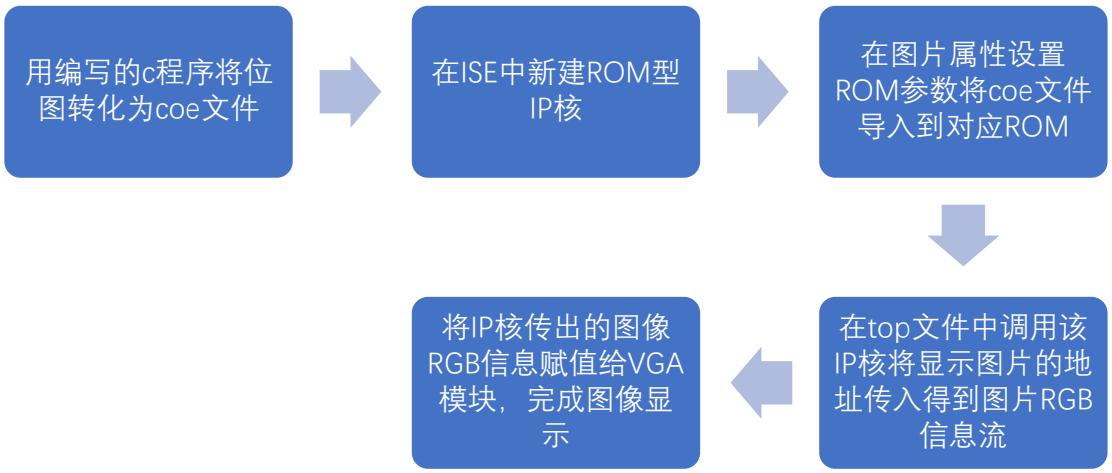
输出由两部分组成，一部分为 VGA 模块的显示器的游戏界面输出，另一部分则为七段数码管显示的游戏得分。

2.模块工作流程

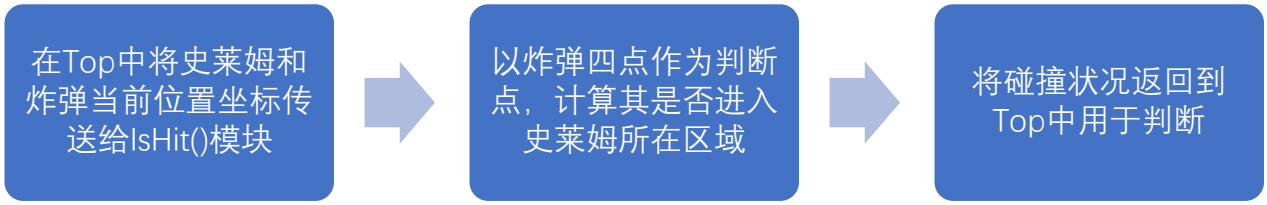
（一）VGA 模块流程示意图



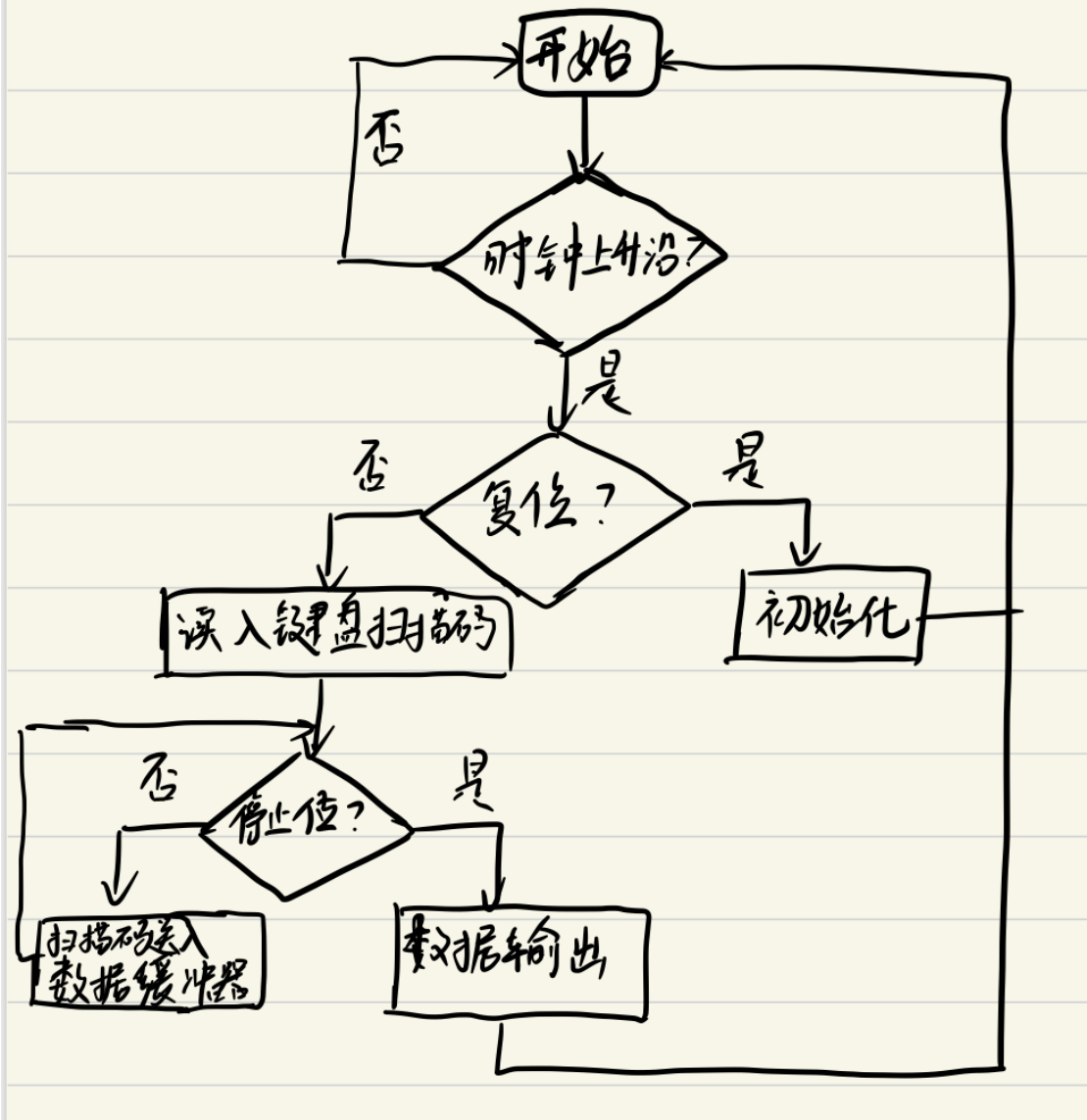
（二）图片导入模块程序流程示意图



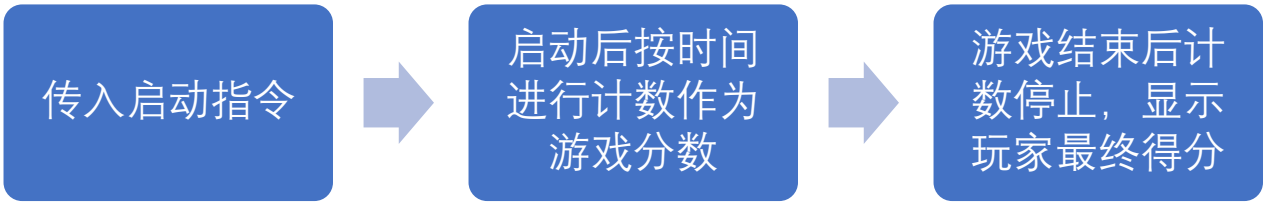
（三）碰撞判断模块 Ishit（）判断流程示意图



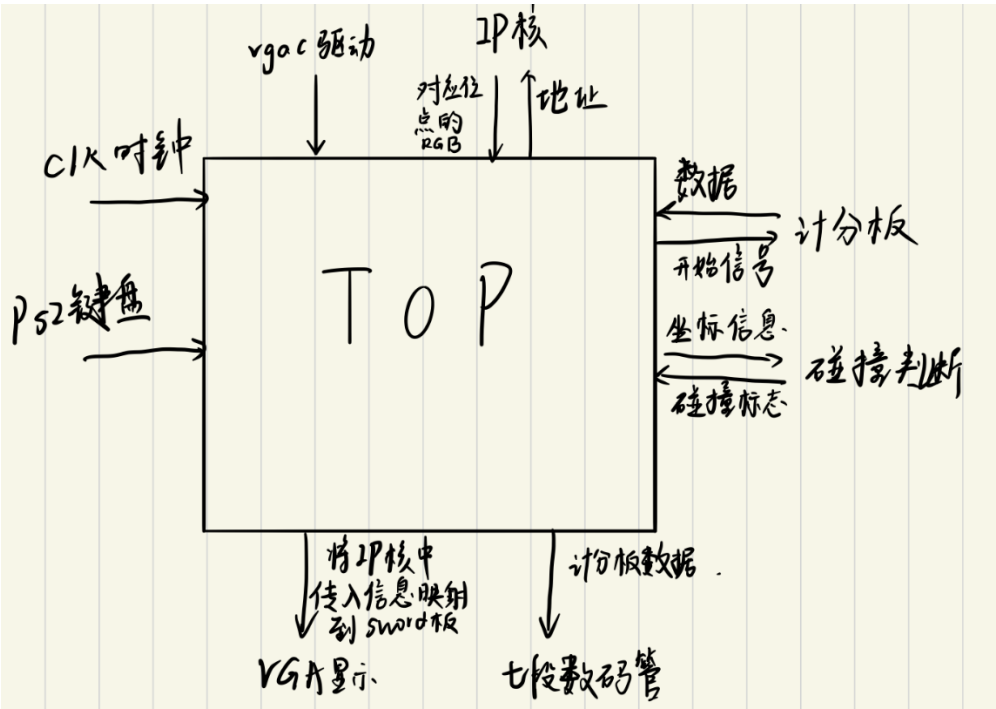
（四）Ps2 程序流程示意图



（五）计分板模块流程示意图



（六）各模块之间的关系图



3.游戏过程

下面介绍游戏的总体过程:

首先进入游戏封面，按下空格键后进入游戏界面开始游戏。在屏幕的四周会出现运动速度、方向不一的炸弹，我们控制的史莱姆在界面正中央，开始时计分板开始计分，分数跟游戏时间挂钩。我们需要通过键盘控制史莱姆躲避从四面八方来的炸弹。在游戏运行时 ps2 module 会将 ps2 键盘 10 位的 data 传给 topmodule，在 top 中进行判定和检测，以判断是不是方向控制键，进而由史莱姆控制逻辑控制史莱姆位置改变。与此同时 VGA 扫描将更新界面。

在史莱姆没有躲过炸弹而发生碰撞时，游戏界面会跳转至结束界面，同时计分板停止运行，其上显示的为游戏的最终得分。

在游戏结束时可以通过 rst 键重置游戏进程，使游戏回归到最初状态。

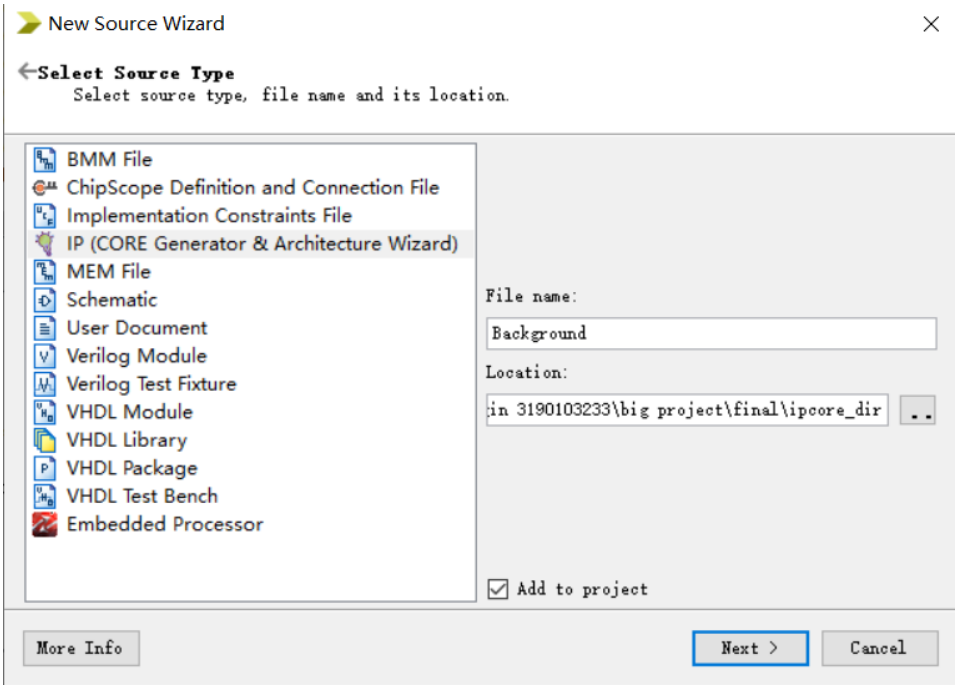
三、模块介绍

1.图片导入

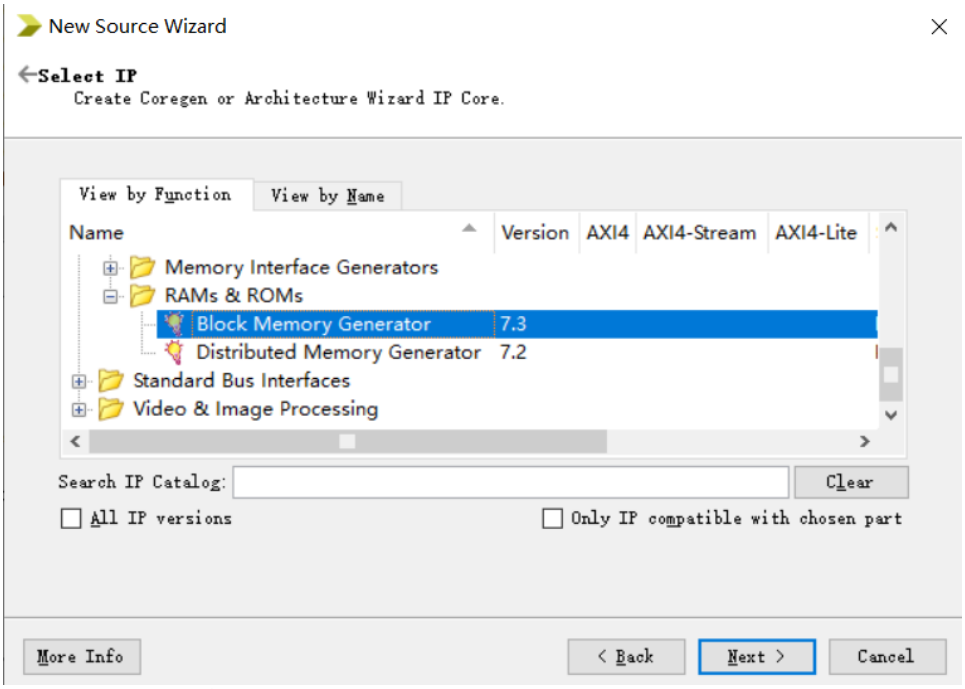
我们使用网上类似小游戏的封面进行转化，先转化为.bmp 文件再通过 c 程序处理图像颜色信息转化为.coe 文件。

具体操作:

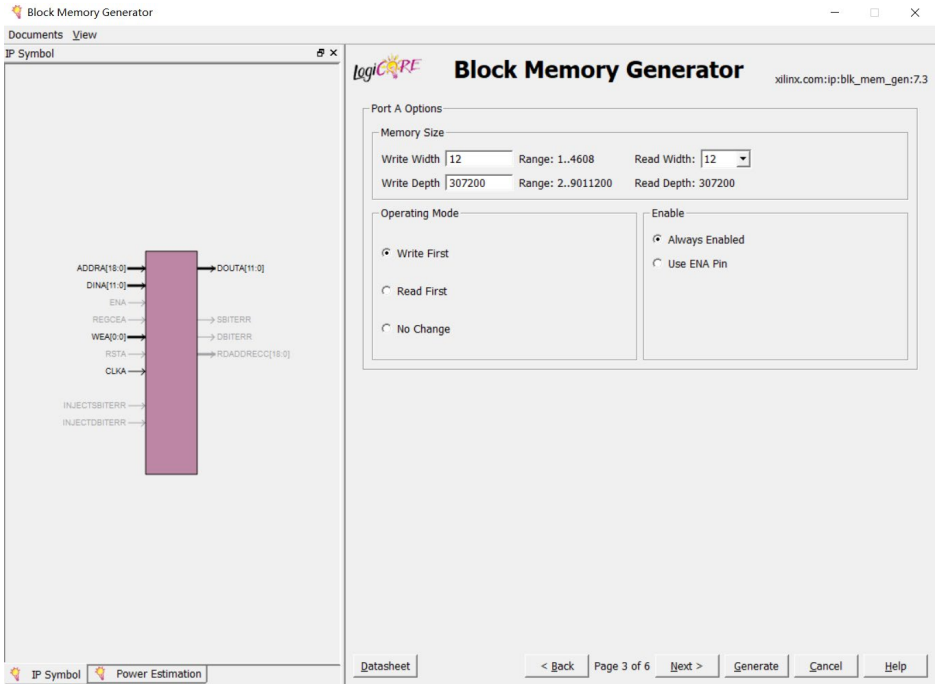
（1）创建 IP 核，命名为 Background。



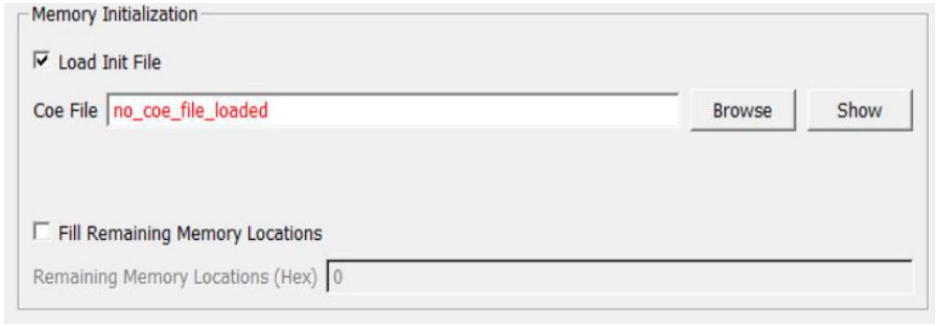
（2）在种类中选择 Block Memory Generator，点击 Next 进入下一步。



（3）调整合适的参数。其中 Write Width 是输出的位数，由于 vga 是 12 位输出，所以我们选择 12。Wirte Depth 是图片的大小，由于 vga 的显示屏大小是 640*480，所以我们填入的参数是 307200（640*480）。

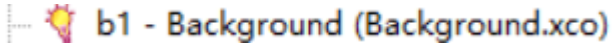


（4）在 page4 勾选 Load Init File，选择想要显示的图片的 coe 文件。



（5）选择完毕后，点击 generate 即可生成 ip 核。

（6）生成完毕后会显示出这样的图标，即为生成成功。

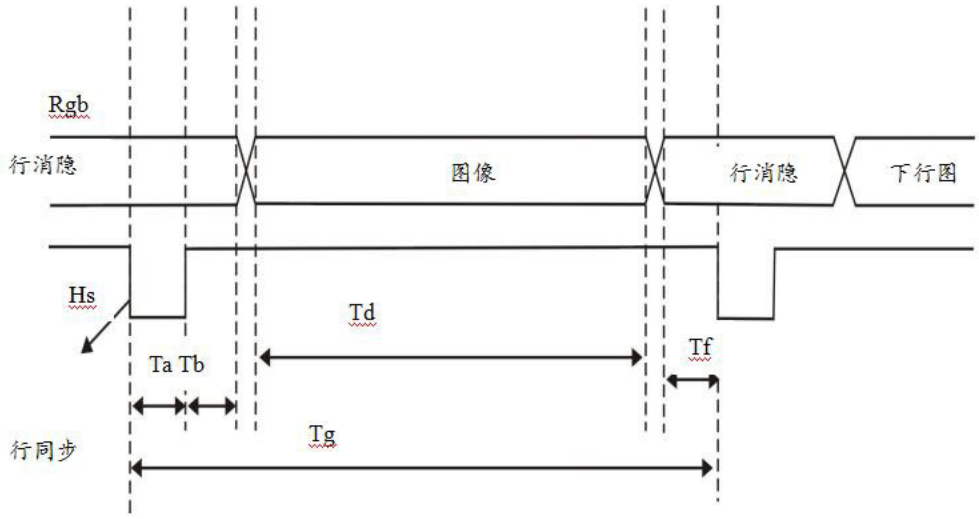


2.VGA 显示

计算机系统基本交互的标准输出显示设备是 VGA 显示器，使用 LCD 显示器，采取了点阵扫描显示技术。

每个显示点都称为一个“像素”，要点亮在某个像素点，首先要定位这个像素点。像素点定位的最有效办法扫描这些像素点，并对每个像素点进行是否点亮的判断。

VGA 扫描时序图：



（1）首先需要调用自己生成的 IP 核的模块

```
    //开始界面 IP 核
    reg [18:0]addr_start_ground;
    wire [11:0]start_ground;
    Start_graph
s1(.addra(addr_start_ground),.douta(start_ground),.clka(clk));
```

（2）接着，我们需要给 IP 核的输入进行赋值。我们用图片左上角的坐标来表示图片的位置。我们判定如果游戏还为开始（Start = 0 时）vga 扫描的纵坐标 col_addr 和横坐标 row_addr 在显示屏范围内，我们就给输入赋值成(480 - row_addr) * 640 + col_addr。

（3）在给输入赋值完毕后，只需将 ip 核的地址赋给 vga_data 即可。

```
    always@(posedge clk) begin
        if(Start == 0)begin
            if((col_addr >= 0) && (col_addr <= 639) && (row_addr >=
0) && (row_addr <= 479))begin
                addr_start_ground <= (480 - row_addr) * 640 + col_addr;
                vga_data <= start_ground[11:0];
            end
        end
```

做出的改进：

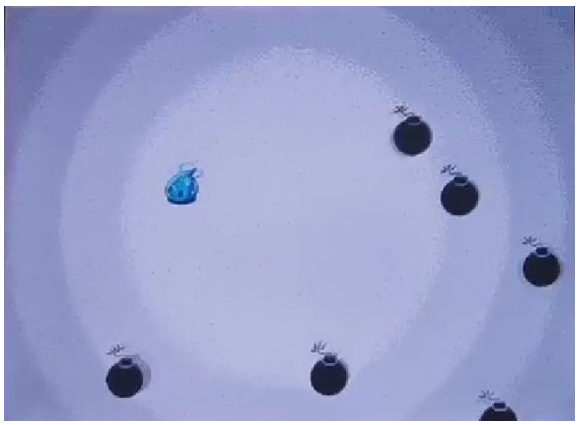
由于本实验中所显示的图片图形本身是不规则的，所以在导入 IP 核时会在图像周围产生空白，而不是透明状态:

为了让显示更加美观，需要将四周的空白删去。

参考图像的特殊性：本身非白色，而不需要的部分均为白色，所以在进行 VGA 显示时加入颜色判断：若扫描到的像素点是白色，那就显示背景；否则显示图像内容。

```
            if (((row_addr >= y) && (row_addr <= y + 64) && (col_addr >= x) && (col_addr <= x + 64)))begin
                temp_dot <= slime[11:0];
                addr_slime <= (64 - (row_addr - y))*64 + (col_addr - x);
                if(temp_dot == 12'hfff)begin
                    addr_background <= row_addr * 640 + col_addr;
                    vga_data <= background[11:0];
                end
            else
                vga_data <= slime[11:0];
            end
```

显示效果：



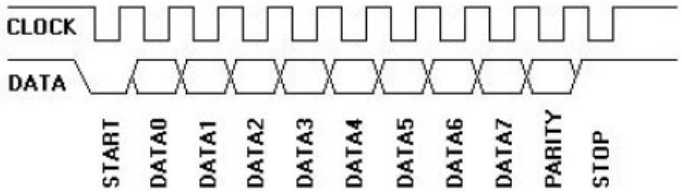
3.PS2 键盘输入

PS2 键盘作为计算机基本交互设备，使用 PS2 通信协议实现双向同步串行通信。通信的两端通过 CLOCK(时钟脚)同步，并通过 DATA(数据脚)交换数据。每个数据共有 11-12 位，分别为：

数据	含义
1个起始位	总是逻辑0
8个数据位	(LSB)低位在前
1个奇偶校验位	奇校验
1个停止位	总是逻辑1
1个应答位	仅用在主机对设备的通信中

数据在 PS/2 时钟的下降沿读取，其上升沿到一个数据转变的时间至少有 5us，数据变化到下降沿的时间又至少需要 5us 并且不大于 25us，这个时序需要严格遵循。

主机箱设备发送数据帧格式与时序:



所以本模块重点在于实现对数据帧的处理:

设置 num，每读取十一位进行重置:

```
always @(posedge clk or posedge rst)begin
    if(rst) num <= 4'd0;
    else if(num==4'd11) num <= 4'd0;
    else if(negedge_ps2_clk) num <= num +1'b1;
end
```

首先时钟上升沿时检测采样，读取键盘扫描码:

```
always @(posedge clk or posedge rst)begin
    if(rst) temp_data <= 8'd0;
    else if(negedge_ps2_clk_shift) begin //键盘时钟连续低电平接收数据
        case(num) //存储八个数据位
            4'd2 : temp_data[0] <= ps2_data;
            4'd3 : temp_data[1] <= ps2_data;
            4'd4 : temp_data[2] <= ps2_data;
            4'd5 : temp_data[3] <= ps2_data;
            4'd6 : temp_data[4] <= ps2_data;
            4'd7 : temp_data[5] <= ps2_data;
            4'd8 : temp_data[6] <= ps2_data;
            4'd9 : temp_data[7] <= ps2_data;
            default : temp_data <= temp_data;
        endcase
    end
```

对当前状态进行判断，若已读取到结束位置，则进行数据判断:

```
    else if(num == 4'd11)begin
        if(temp_data == 8'hE0) data_expand <= 1'b1; //键盘扩张键
        else if(temp_data == 8'hF0) data_break <= 1'b1;
        else begin
            data <= {data_expand,data_break,temp_data};
            data_done <= 1'b1; //键盘数据已准备好输出
            data_expand <= 1'b0;
            data_break <=1'b0;
        end
    end
```

并将数据输出模块。

4.七段数码管显示分数

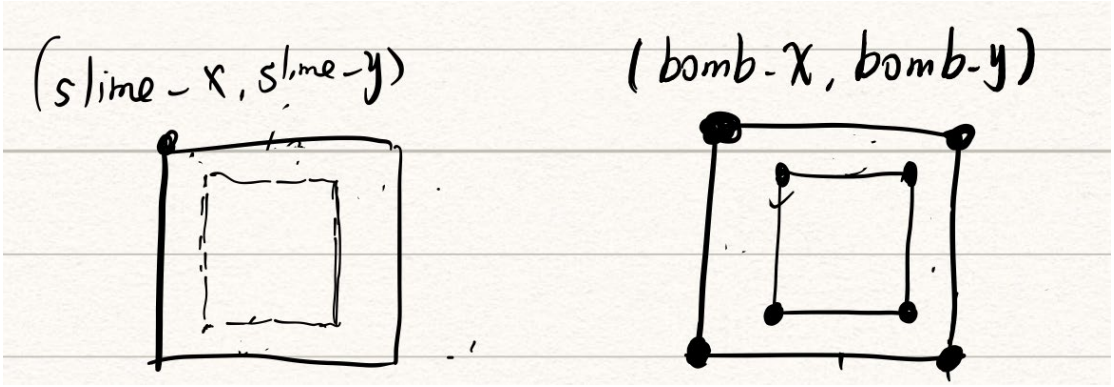
本模块将获得的分数显示在七段数码管上。七段数码管的显示驱动使用之前的实验工程 MyRevCounter，实现正向计数，并将模块根据需要进行一定修改：

程序要求在游戏开始后计时，结束后停止计时并保持显示，所以需要对 MyRevCounter 加入游戏开始的输入标识，并且添加游戏结束的判断。

```
module RevCounter(input wire clk,
input wire Start,//等于 1 表示游戏开始
input wire s,
output reg [15:0] cnt
      );
initial cnt= 0;
always@ (posedge clk) begin
      if(Start == 1)begin//游戏开始，计时同时开始
            if(s) cnt<=cnt+1;
      end
      else cnt <= 0;//游戏未开始时计分记为 0，复位时使用
end
endmodule
```

5.IsHit 模块——判断史莱姆和炸弹是否发生碰撞

本模块的实质是对两个坐标的判断，对于史莱姆和炸弹两个显示色块四点坐标进行计算与判断。



以史莱姆所在的色块区域作为判断面，以炸弹的顶角四点作为判断点，进行坐标计算判断其是否进入史莱姆所在区域，若进入则已撞到。

```
    always@(posedge clk)begin
        if((start == 1) &&((((bomb_y + 15 >= slime_y + 10) && (bomb_y + 15 <= slime_y
+ 54) && (bomb_x + 15 >= slime_x + 10) && (bomb_x + 15 <= slime_x + 54)) ||
        ((bomb_y + 57 >= slime_y + 10) && (bomb_y + 57 <= slime_y + 54) && (bomb_x
+ 15 >= slime_x + 10) && (bomb_x + 15 <= slime_x + 54)) ||
        ((bomb_y + 15 >= slime_y + 10) && (bomb_y + 15 <= slime_y + 54) && (bomb_x
+ 57 >= slime_x + 10) && (bomb_x + 57 <= slime_x + 54)) ||
        ((bomb_y + 57 >= slime_y + 10) && (bomb_y + 57 <= slime_y + 54) && (bomb_x
+ 57 >= slime_x + 10) && (bomb_x + 57 <= slime_x + 54)))))
            done = 1'b1;
        else
            done = 1'b0;
    end
```

6.炸弹运动模块

（1）变量定义和初始化

```
    reg [9:0] x;
    reg [8:0] y;
    reg [9:0] bomb1_x;
    reg [8:0] bomb1_y;
    reg [9:0] bomb2_x;
    reg [8:0] bomb2_y;
    reg [9:0] bomb3_x;
    reg [8:0] bomb3_y;
    reg [9:0] bomb4_x;
    reg [8:0] bomb4_y;
    reg [9:0] bomb5_x;
    reg [8:0] bomb5_y;
    reg [9:0] bomb6_x;
    reg [8:0] bomb6_y;
    initial begin
        bomb1_x <= 10'd640;
        bomb1_y <= 9'd0;
        bomb2_x <= 10'd500;
        bomb2_y <= 9'd0;
        bomb3_x <= 10'd380;
        bomb3_y <= 9'd80;
        bomb4_x <= 10'd0;
        bomb4_y <= 9'd320;
        bomb5_x <= 10'd500;
        bomb5_y <= 9'd320;
        bomb6_x <= 10'd640;
        bomb6_y <= 9'd0;
    end
```

（2）调用炸弹 IP 核

```
    //炸弹 IP 核
    reg [11:0] addr_bomb1;
    reg [11:0] b1_temp;
    wire[11:0] bomb1;
    reg [11:0] addr_bomb2;
```

```
    reg [11:0] b2_temp;
    wire[11:0] bomb2;
    reg [11:0] addr_bomb3;
    reg [11:0] b3_temp;
    wire[11:0] bomb3;
    reg [11:0] addr_bomb4;
    reg [11:0] b4_temp;
    wire[11:0] bomb4;
    reg [11:0] addr_bomb5;
    reg [11:0] b5_temp;
    wire[11:0] bomb5;
    reg [11:0] addr_bomb6;
    reg [11:0] b6_temp;
    wire[11:0] bomb6;
    bomb_graph bb1(.addra(addr_bomb1),.douta(bomb1),.clka(clk));
    bomb_graph bb2(.addra(addr_bomb2),.douta(bomb2),.clka(clk));
    bomb_graph bb3(.addra(addr_bomb3),.douta(bomb3),.clka(clk));
    bomb_graph bb4(.addra(addr_bomb4),.douta(bomb4),.clka(clk));
    bomb_graph bb5(.addra(addr_bomb5),.douta(bomb5),.clka(clk));
    bomb_graph bb6(.addra(addr_bomb6),.douta(bomb6),.clka(clk));
```

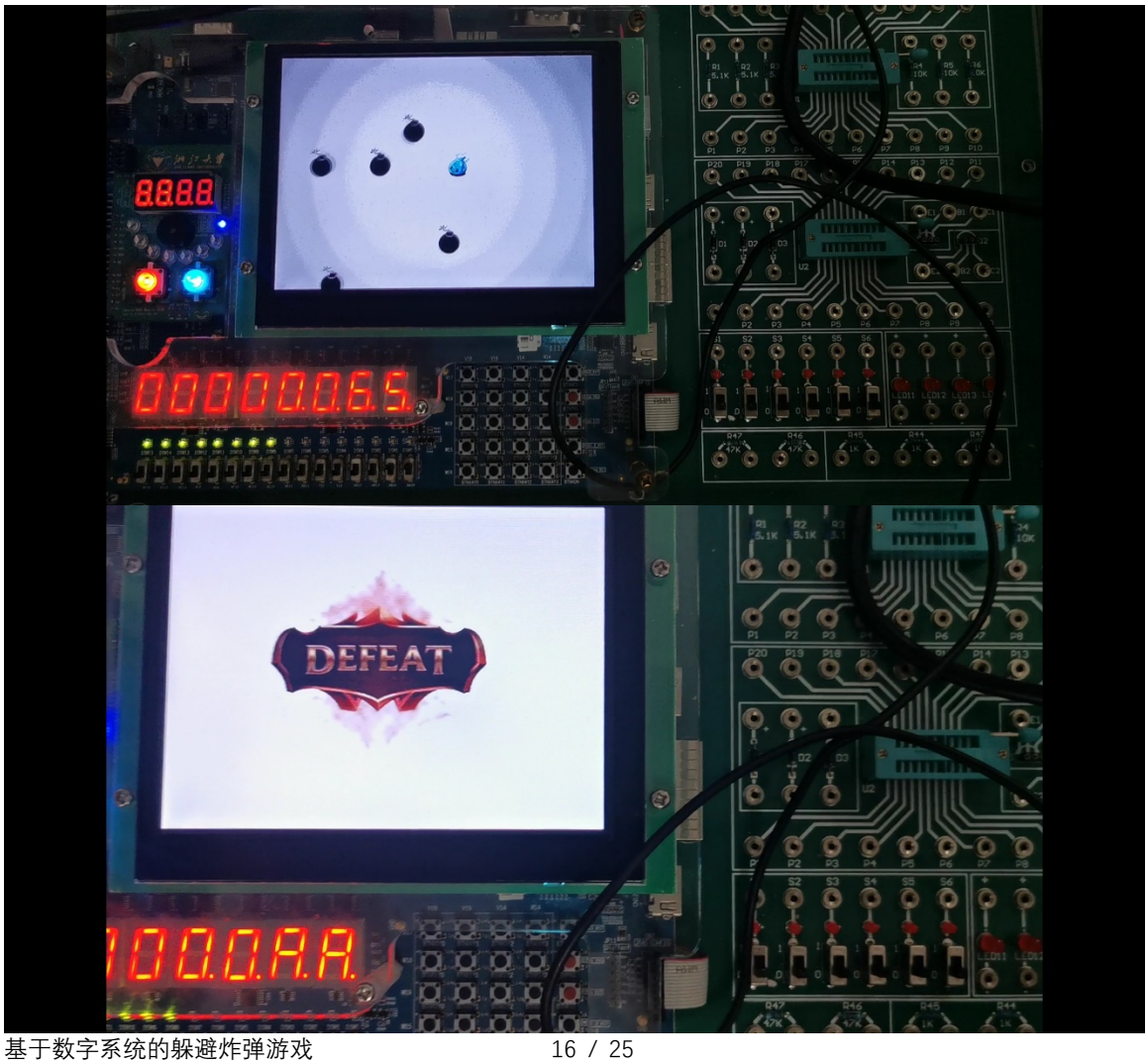
(3) 每当上升沿时，在游戏还在运行且没有发生碰撞时让炸弹运动

```
    always @(posedge clk) begin
    if(Start == 1)begin
    if(hit == 0)begin
    if(clkdiv[19:0] == 20'h00000) begin
        bomb1_y <= bomb1_y + 1;
        bomb1_x <= bomb1_x - 1;
        bomb2_x <= bomb2_x - 1;
        bomb2_y <= bomb2_y + 2;
        bomb3_x <= bomb3_x - 3;
        bomb3_y <= bomb3_y + 1;
        bomb4_x <= bomb4_x + 2;
        bomb4_y <= bomb4_y - 2;
        bomb5_x <= bomb5_x + 2;
        bomb5_y <= bomb5_y - 1;
        bomb6_x <= bomb6_x - 4;
        bomb6_y <= bomb6_y + 2;
        end
    if(bomb1_x <= 5)begin
        bomb1_x <= 10'd580;
        end
    if(bomb2_x <= 5)begin
        bomb2_x <= 10'd580;
        end
    if(bomb3_x <= 5)begin
        bomb3_x <= 10'd580;
        end
    if(bomb4_x >= 590)begin
        bomb4_x <= 0;
        end
    if(bomb5_x >= 590)begin
        bomb5_x <= 0;
        end
    if(bomb6_x <= 5)begin
        bomb6_x <= 10'd580;
        end
    end
    end
    end
    //动起来
```

四、游戏演示

（1）游戏初始封面（这里为了显示清楚将 vga 连接至电脑显示屏）
按空格即可开始游戏。

（2）游戏界面
按下空格键后进入游戏界面，背景、史莱姆在屏幕上出现，炸弹从四面八方生成并运动。图为游戏界面截图。



（3）游戏结束界面
在史莱姆和炸弹发生碰撞之后游戏结束，显示屏上显示失败的界面。
此时计分板上显示的为本局游戏分数，此时按下 rst 键可以重置游戏进行下一局。

附录：源代码

1.top.v

```
    `timescale 1ns / 1ps
    //////////////////////////////////////////////////////////////////
//////////////////
    // Company:
    // Engineer:
    //
    // Create Date: 2016/10/17 12:25:41
    // Design Name:
    // Module Name: Top
    // Project Name:
    // Target Devices:
    // Tool Versions:
    // Description:
    //
    // Dependencies:
    //
    // Revision:
    // Revision 0.01 - File Created
    // Additional Comments:
    //
    //////////////////////////////////////////////////////////////////
//////////////////


    module Top(
    input clk,
    input rstn,
    input [15:0]SW,
    output hs,
    output vs,
    output [3:0] r,
    output [3:0] g,
    output [3:0] b,

    output wire ledclk,
    output wire ledsout,
    output wire ledclrn,
    output wire LEDEN,
    output wire seg_clk,
    output wire seg_sout,
    output wire SEG_PEN,
    output wire seg_clrn,
    inout [4:0]BTN_X,
    inout [3:0]BTN_Y,
```

```
    output buzzer,
    input ps2_clk,
    input ps2_data
       );

    reg [31:0]clkdiv;
    always@(posedge clk) begin
        clkdiv <= clkdiv + 1'b1;
    end
    assign buzzer = 1'b1;
    wire [15:0] SW_OK;
    AntiJitter #(4) a0[15:0](.clk(clkdiv[15]), .I(SW), .O(SW_OK));

    wire [7:0] Ps2_keyCode;
    wire Ps2_keyReady;
    ps2_kb k1(.clk(clk),
     .rst(SW[15]),
     .ps2_clk(ps2_clk), //键盘时钟输入
     .ps2_data(ps2_data), //键盘数据输入
     .data_out(Ps2_keyCode), //键盘扫描输出
     .ready(Ps2_keyReady)
     );

    reg [9:0] x;
    reg [8:0] y;
    reg [9:0] bomb1_x;
    reg [8:0] bomb1_y;
    reg [9:0] bomb2_x;
    reg [8:0] bomb2_y;
    reg [9:0] bomb3_x;
    reg [8:0] bomb3_y;
    reg [9:0] bomb4_x;
    reg [8:0] bomb4_y;
    reg [9:0] bomb5_x;
    reg [8:0] bomb5_y;
    reg [9:0] bomb6_x;
    reg [8:0] bomb6_y;

    reg Start;
    //炸弹初始化
    initial begin
        bomb1_x <= 10'd640;
        bomb1_y <= 9'd0;
        bomb2_x <= 10'd500;
        bomb2_y <= 9'd0;
        bomb3_x <= 10'd380;
        bomb3_y <= 9'd80;
        bomb4_x <= 10'd0;
        bomb4_y <= 9'd320;
        bomb5_x <= 10'd500;
        bomb5_y <= 9'd320;
        bomb6_x <= 10'd640;
        bomb6_y <= 9'd0;
        Start <= 0;

    end


        reg [11:0] vga_data;
        wire [9:0] col_addr;
```

```
       wire [8:0] row_addr;

    vgac v0 (

    .vga_clk(clkdiv[1]), .clrn(SW_OK[0]), .d_in(vga_data), .row_add
r(row_addr), .col_addr(col_addr), .r(r), .g(g), .b(b), .hs(hs), .vs
(vs)
    );

    wire hit1,hit2,hit3,hit4,hit5,hit6;
    wire hit;

    IsHit
IsHit1(.clk(clk),.slime_x(x),.slime_y(y),.bomb_x(bomb1_x),.bomb_y(b
omb1_y),.hit(hit1),.start(Start));
    IsHit
IsHit2(.clk(clk),.slime_x(x),.slime_y(y),.bomb_x(bomb2_x),.bomb_y(b
omb2_y),.hit(hit2),.start(Start));
    IsHit
IsHit3(.clk(clk),.slime_x(x),.slime_y(y),.bomb_x(bomb3_x),.bomb_y(b
omb3_y),.hit(hit3),.start(Start));
    IsHit
IsHit4(.clk(clk),.slime_x(x),.slime_y(y),.bomb_x(bomb4_x),.bomb_y(b
omb4_y),.hit(hit4),.start(Start));
    IsHit
IsHit5(.clk(clk),.slime_x(x),.slime_y(y),.bomb_x(bomb5_x),.bomb_y(b
omb5_y),.hit(hit5),.start(Start));
    IsHit
IsHit6(.clk(clk),.slime_x(x),.slime_y(y),.bomb_x(bomb6_x),.bomb_y(b
omb6_y),.hit(hit6),.start(Start));

    assign hit = hit1 | hit2 | hit3 | hit4 | hit5 | hit6;

    wire clk_100ms;
    wire [31:0] cnt;
    wire [31:0] div;
    clk_100ms c0(.clk(clk),.clk_100ms(clk_100ms));
    RevCounter c1(.clk(clk_100ms),.Start(Start),.s(~hit),.cnt(cnt));
    clkdiv m3(.clk(clk),.rst(1'b0),.clkdiv(div[31:0]));
    LEDP2S             #(.DATA_BITS(16),.DATA_COUNT_BITS(4),.DIR(0))
U7(.clk(clk),.rst(1'b0),.Start(div[20]),

    .PData({8'hFF}),.sclk(ledclk),.sclrn(ledclrn),.sout(ledsout),.E
N(LEDEN));
    SSeg7_Dev
m4(.clk(clk),.rst(~rstn),.Start(div[20]),.SW0(SW_OK[0]),.flash(1'b0
),.Hexs(cnt),.point(8'b0000_1111),
    .LES(8'b1111_0000),.seg_clk(seg_clk),.seg_sout(seg_sout),.SEG_P
EN(SEG_PEN),.seg_clrn(seg_clrn));

    //开始界面 IP 核
    reg [18:0]addr_start_ground;
    wire [11:0]start_ground;
    Start_graph
s1(.addra(addr_start_ground),.douta(start_ground),.clka(clk));
    //背景 IP 核
    reg [18:0]addr_background;
    wire [11:0]background;
    Background
b1(.addra(addr_background),.douta(background),.clka(clk));
```

```
    //史莱姆 IP 核
    reg [11:0]addr_slime;
    reg [11:0]temp_dot;
    wire [11:0]slime;
    Slime g1(.addra(addr_slime),.douta(slime),.clka(clk));
    //炸弹 IP 核
    reg [11:0] addr_bomb1;
    reg [11:0] b1_temp;
    wire[11:0] bomb1;
    reg [11:0] addr_bomb2;
    reg [11:0] b2_temp;
    wire[11:0] bomb2;
    reg [11:0] addr_bomb3;
    reg [11:0] b3_temp;
    wire[11:0] bomb3;
    reg [11:0] addr_bomb4;
    reg [11:0] b4_temp;
    wire[11:0] bomb4;
    reg [11:0] addr_bomb5;
    reg [11:0] b5_temp;
    wire[11:0] bomb5;
    reg [11:0] addr_bomb6;
    reg [11:0] b6_temp;
    wire[11:0] bomb6;
    bomb_graph bb1(.addra(addr_bomb1),.douta(bomb1),.clka(clk));
    bomb_graph bb2(.addra(addr_bomb2),.douta(bomb2),.clka(clk));
    bomb_graph bb3(.addra(addr_bomb3),.douta(bomb3),.clka(clk));
    bomb_graph bb4(.addra(addr_bomb4),.douta(bomb4),.clka(clk));
    bomb_graph bb5(.addra(addr_bomb5),.douta(bomb5),.clka(clk));
    bomb_graph bb6(.addra(addr_bomb6),.douta(bomb6),.clka(clk));

    //失败界面 IP 核
    reg [17:0]addr_defeat;
    wire [11:0]defeat;
    Defeat_graph d1(.addra(addr_defeat),.douta(defeat),.clka(clk));

    always @(posedge clk) begin
    if(Start == 1)begin
    if(hit == 0)begin
    if(clkdiv[19:0] == 20'h00000) begin
        bomb1_y <= bomb1_y + 1;
        bomb1_x <= bomb1_x - 1;
        bomb2_x <= bomb2_x - 1;
        bomb2_y <= bomb2_y + 2;
        bomb3_x <= bomb3_x - 3;
        bomb3_y <= bomb3_y + 1;
        bomb4_x <= bomb4_x + 2;
        bomb4_y <= bomb4_y - 2;
        bomb5_x <= bomb5_x + 2;
        bomb5_y <= bomb5_y - 1;
        bomb6_x <= bomb6_x - 4;
        bomb6_y <= bomb6_y + 2;
        end
    if(bomb1_x <= 5)begin
        bomb1_x <= 10'd580;
        end
    if(bomb2_x <= 5)begin
        bomb2_x <= 10'd580;
        end
    if(bomb3_x <= 5)begin
```

```
        bomb3_x <= 10'd580;
        end
    if(bomb4_x >= 590)begin
        bomb4_x <= 0;
        end
    if(bomb5_x >= 590)begin
        bomb5_x <= 0;
        end
    if(bomb6_x <= 5)begin
        bomb6_x <= 10'd580;
        end
    end

    end
    end
    //动起来


    reg Ps2_wasReady;
    always @(posedge clk) begin
        if (!rstn) begin
            x <= 10'd320;
            y <= 9'd240;
            Start <= 0;
        end else begin
            Ps2_wasReady <= Ps2_keyReady;
            if (!Ps2_wasReady && Ps2_keyReady) begin
                    case (Ps2_keyCode)
                        8'h29: Start <= 1;
                        8'h1c: x <= x - 10'd20;
                        8'h23: x <= x + 10'd20;
                        8'h1B: y <= y + 9'd20;
                        8'h1D: y <= y - 9'd20;
                        default: ;
                    endcase
            end
        end
    end


    always@(posedge clk) begin

        if(Start == 0)begin
            if((col_addr >= 0) && (col_addr <= 639) && (row_addr >=
0) && (row_addr <= 479))begin
                addr_start_ground <= (480 - row_addr) * 640 + col_addr;
                vga_data <= start_ground[11:0];
            end
        end
        else begin

            if((col_addr >= 0) && (col_addr <= 639) && (row_addr >=
0) && (row_addr <= 479))begin
                addr_background <= row_addr * 640 + col_addr;
                vga_data <= background[11:0];
            end

            if((row_addr >= bomb1_y) && (row_addr <= bomb1_y+64) &&
(col_addr >= bomb1_x) && (col_addr <= bomb1_x+64)) begin
                b1_temp <= bomb1[11:0];
                addr_bomb1  <=  (64  -  (row_addr  -  bomb1_y))*64  +
```

```
(col_addr - bomb1_x);
              if(b1_temp == 12'hfff)begin
                  addr_background <= row_addr * 640 + col_addr;

                  vga_data <= background[11:0];
              end
              else
                  vga_data <= bomb1[11:0];
          end
          if((row_addr >= bomb2_y) && (row_addr <= bomb2_y+64) &&
(col_addr >= bomb2_x) && (col_addr <= bomb2_x+64)) begin
              b2_temp <= bomb2[11:0];
              addr_bomb2  <=  (64  -  (row_addr  -  bomb2_y))*64  +
(col_addr - bomb2_x);
              if(b2_temp == 12'hfff)begin
                  addr_background <= row_addr * 640 + col_addr;

                  vga_data <= background[11:0];
              end
              else
                  vga_data <= bomb2[11:0];
          end
          if((row_addr >= bomb3_y) && (row_addr <= bomb3_y+64) &&
(col_addr >= bomb3_x) && (col_addr <= bomb3_x+64)) begin
              b3_temp <= bomb3[11:0];
              addr_bomb3  <=  (64  -  (row_addr  -  bomb3_y))*64  +
(col_addr - bomb3_x);
              b3_temp <= bomb3[11:0];
              addr_bomb3  <=  (64  -  (row_addr  -  bomb3_y))*64  +
(col_addr - bomb3_x);
              if(b3_temp == 12'hfff)begin
                  addr_background <= row_addr * 640 + col_addr;

                  vga_data <= background[11:0];
              end
              else
                  vga_data <= bomb3[11:0];
          end
          if((row_addr >= bomb4_y) && (row_addr <= bomb4_y+64) &&
(col_addr >= bomb4_x) && (col_addr <= bomb4_x+64)) begin
              b4_temp <= bomb4[11:0];
              addr_bomb4  <=  (64  -  (row_addr  -  bomb4_y))*64  +
(col_addr - bomb4_x);
              b4_temp <= bomb4[11:0];
              addr_bomb4  <=  (64  -  (row_addr  -  bomb4_y))*64  +
(col_addr - bomb4_x);
              if(b4_temp == 12'hfff)begin
                  addr_background <= row_addr * 640 + col_addr;

                  vga_data <= background[11:0];
              end
              else
                  vga_data <= bomb4[11:0];
          end
          if((row_addr >= bomb5_y) && (row_addr <= bomb5_y+64) &&
(col_addr >= bomb5_x) && (col_addr <= bomb5_x+64)) begin
              b5_temp <= bomb5[11:0];
              addr_bomb5  <=  (64  -  (row_addr  -  bomb5_y))*64  +
(col_addr - bomb5_x);
              b5_temp <= bomb5[11:0];
              addr_bomb5  <=  (64  -  (row_addr  -  bomb5_y))*64  +
```

```
(col_addr - bomb5_x);
              if(b5_temp == 12'hfff)begin
                  addr_background <= row_addr * 640 + col_addr;

                  vga_data <= background[11:0];
              end
              else
                  vga_data <= bomb5[11:0];
          end
          if((row_addr >= bomb6_y) && (row_addr <= bomb6_y+64) &&
(col_addr >= bomb6_x) && (col_addr <= bomb6_x+64)) begin
              b6_temp <= bomb6[11:0];
              addr_bomb6  <=  (64  -  (row_addr  -  bomb6_y))*64  +
(col_addr - bomb6_x);
              b6_temp <= bomb6[11:0];
              addr_bomb6  <=  (64  -  (row_addr  -  bomb6_y))*64  +
(col_addr - bomb6_x);
              if(b6_temp == 12'hfff)begin
                  addr_background <= row_addr * 640 + col_addr;

                  vga_data <= background[11:0];
              end
              else
                  vga_data <= bomb6[11:0];
          end
          if  (((row_addr  >=  y)  &&  (row_addr  <=  y  +  64)  &&
(col_addr >= x) && (col_addr <= x + 64)))begin
              temp_dot <= slime[11:0];
              addr_slime <= (64 - (row_addr - y))*64 + (col_addr -
x);
              if(temp_dot == 12'hfff)begin
                  addr_background <= row_addr * 640 + col_addr;

                  vga_data <= background[11:0];
              end
          else
              vga_data <= slime[11:0];
          end
      end

      if(hit == 1)begin
          addr_defeat <= (480 - row_addr)*640 + col_addr;
          vga_data <= defeat[11:0];
      end

  end

  endmodule
```

2.IsHit.v

```
  module IsHit(
  input clk,
  input start,
  input [9:0]slime_x,
  input [8:0]slime_y,
  input [9:0]bomb_x,
```

```
    input [8:0]bomb_y,
    output hit
        );
    reg done;
    initial begin
        done = 1'b0;
    end
    always@(posedge clk)begin
        if((start == 1) &&((((bomb_y + 15 >= slime_y + 10) && (bomb_y
+ 15 <= slime_y + 54) && (bomb_x + 15 >= slime_x + 10) && (bomb_x +
15 <= slime_x + 54)) ||
        ((bomb_y + 57 >= slime_y + 10) && (bomb_y + 57 <= slime_y +
54) && (bomb_x + 15 >= slime_x + 10) && (bomb_x + 15 <= slime_x +
54)) ||
        ((bomb_y + 15 >= slime_y + 10) && (bomb_y + 15 <= slime_y +
54) && (bomb_x + 57 >= slime_x + 10) && (bomb_x + 57 <= slime_x +
54)) ||
        ((bomb_y + 57 >= slime_y + 10) && (bomb_y + 57 <= slime_y +
54) && (bomb_x + 57 >= slime_x + 10) && (bomb_x + 57 <= slime_x +
54)))))
            done = 1'b1;
        else
            done = 1'b0;
    end

    assign hit = done;

    endmodule
```

3.UCF 文件

```
       NET "clk"  LOC = AC18 | IOSTANDARD = LVCMOS18 ;
       NET "rstn" LOC = W13  | IOSTANDARD = LVCMOS18 ;
     NET "seg_clk"LOC = M24 | IOSTANDARD = LVCMOS33;
     NET "seg_clrn"   LOC = M20 | IOSTANDARD = LVCMOS33;
     NET "seg_sout"   LOC = L24 | IOSTANDARD = LVCMOS33;
     NET "SEG_PEN"LOC = R18 | IOSTANDARD = LVCMOS33;
     NET "ledclk" LOC = N26 | IOSTANDARD = LVCMOS33;
     NET "ledclrn"LOC = N24 | IOSTANDARD = LVCMOS33;
     NET "ledsout"LOC = M26 | IOSTANDARD = LVCMOS33;
     NET "LEDEN"  LOC = P18 | IOSTANDARD = LVCMOS33;
       NET "BTN_X[0]"LOC = V17  | IOSTANDARD = LVCMOS18 | PULLUP;
       NET "BTN_X[1]"LOC = W18  | IOSTANDARD = LVCMOS18 | PULLUP;
       NET "BTN_X[2]"LOC = W19  | IOSTANDARD = LVCMOS18 | PULLUP;
       NET "BTN_X[3]"    LOC = W15  | IOSTANDARD = LVCMOS18 | PULLUP;
       NET "BTN_X[4]"LOC = W16  | IOSTANDARD = LVCMOS18 | PULLUP;
       NET "BTN_Y[0]"    LOC = V18  | IOSTANDARD = LVCMOS18 | PULLUP;
       NET "BTN_Y[1]"LOC = V19  | IOSTANDARD = LVCMOS18 | PULLUP;
       NET "BTN_Y[2]"    LOC = V14  | IOSTANDARD = LVCMOS18 | PULLUP;
       NET "BTN_Y[3]"LOC = W14  | IOSTANDARD = LVCMOS18 | PULLUP;
       NET "SW[0]"   LOC = AA10 | IOSTANDARD = LVCMOS15 ;
       NET "SW[1]"   LOC = AB10 | IOSTANDARD = LVCMOS15 ;
       NET "SW[2]"   LOC = AA13 | IOSTANDARD = LVCMOS15 ;
       NET "SW[3]"   LOC = AA12 | IOSTANDARD = LVCMOS15 ;
       NET "SW[4]"   LOC = Y13  | IOSTANDARD = LVCMOS15 ;
       NET "SW[5]"   LOC = Y12  | IOSTANDARD = LVCMOS15 ;
       NET "SW[6]"   LOC = AD11 | IOSTANDARD = LVCMOS15 ;
```

```
      NET "SW[7]"    LOC = AD10 | IOSTANDARD = LVCMOS15 ;
      NET "SW[8]"    LOC = AE10 | IOSTANDARD = LVCMOS15 ;
      NET "SW[9]"    LOC = AE12 | IOSTANDARD = LVCMOS15 ;
      NET "SW[10]"   LOC = AF12 | IOSTANDARD = LVCMOS15 ;
      NET "SW[11]"   LOC = AE8  | IOSTANDARD = LVCMOS15 ;
      NET "SW[12]"   LOC = AF8  | IOSTANDARD = LVCMOS15 ;
      NET "SW[13]"   LOC = AE13 | IOSTANDARD = LVCMOS15 ;
      NET "SW[14]"   LOC = AF13 | IOSTANDARD = LVCMOS15 ;
      NET "SW[15]"   LOC = AF10 | IOSTANDARD = LVCMOS15 ;
    NET "buzzer"  LOC = AF24 | IOSTANDARD = LVCMOS33;

   NET"ps2_clk" LOC = N18 | IOSTANDARD = LVCMOS33 | PULLUP;
   NET"ps2_data" LOC = M19 | IOSTANDARD = LVCMOS33 | PULLUP;

      NET "b[0]" LOC = T20 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "b[1]" LOC = R20 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "b[2]" LOC = T22 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "b[3]" LOC = T23 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "g[0]" LOC = R22 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "g[1]" LOC = R23 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "g[2]" LOC = T24 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "g[3]" LOC = T25 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "r[0]" LOC = N21 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "r[1]" LOC = N22 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "r[2]" LOC = R21 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "r[3]" LOC = P21 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "hs"   LOC = M22 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
      NET "vs"   LOC = M21 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
```