浙江大学
数字逻辑设计·课程报告

基于数字系统的“雷霆战机”游戏设计

组长：邹昂

组员：武思羽

2024 年 1 月 9 日

目录

一、游戏设计背景

1、FPGA 与开发平台

FPGA（Field－Programmable Gate Array，即现场可编程门阵列）是在 PAL、GAL、CPLD 等可编程器件的基础上进一步发展产物。以硬件描述语言（Verilog 或 VHDL）所完成的电路设计，可以经过简单的综合与布局，快速的烧录至 FPGA 上进行测试。这些可编程元件可以被用来实现一些基本的逻辑门电路或者更复杂的一些的组合功能。Xilinx 的 SWORD 实验板为可编程的实验板，而且实验板还可以和许多外部设备进行连接，例如 PS2、鼠标或者显示器，从而可以在 FPGA 的基础上设计使用 PS2、鼠标的调用，或者显示器上显示图片。

本次设计使用的软件为 Vivado 2023.1 软件，使用的语言为 Verilog 语言。Verilog 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。

2、游戏介绍

《雷霆战机》是一款为人熟知的飞机射击游戏。在游戏中，玩家可以通过控制方向键，实现飞机的上下左右移动，飞机会自动发射子弹攻击敌人。在游戏中会出现 Boss 来提升游戏的趣味性和难度，Boss 的血量更高。玩家需要控制飞机不被击中或者被敌机碰撞，同时击落尽可能多的飞机来获得游戏分数。

3、设计介绍

本课程设计利用 FPGA 板，通过底层逻辑设计以及硬件控制来实现雷霆战机游戏，游戏由于硬件水平限制，进行了一定的简化。

在该游戏中，我们使用 VGA 信号进行图像的显示和变化，封面为一张宇宙的图片，带有游戏名称和进入游戏的方式。开始后出现游戏界面，敌机会不断刷新，上方的中央会出现 boss，玩家飞机被击毁后会出现 Game Over 字样。

游戏中我们使用外接的 ps2 接口的键盘进行方向的控制和游戏的开始，通过上、下、左、右四个方向按键控制飞机的移动，以及 Enter 键实现从开始界面进入游戏的转换。敌机会在不同位置刷新，同时发射子弹，boss 也会不断发射子弹。玩家飞机共有三次被命中的机会，

游戏中有血量显示，血量清零游戏结束。

4、重难点分析

（1）利用 VGA 在显示器上进行图像的显示以及确定贴图的先后顺序，从而保证游戏流程的正常进行。

（2）ps2 模块获取键盘信号，并通过断码、通码等信号来表现不同按键是否按下，从而控制飞机的移动以及游戏的开始。

（3）对于敌机以及玩家飞机射出子弹的判断以及子弹不断循环射出进行正确的位置设定，同时实现子弹击中后出现扣血或爆炸的判断。

（4） Vivado 2023.1 软件中 IP core 的生成以及地址等参数的正确传输和使用，从而能够获得正确的 RGB 信号进行图形的显示。

（5）实现敌机爆炸后的延时效果，使爆炸动画持续一段时间，同时在一段时间后再进行新飞机的生成。

（6）通过对敌机爆炸信号的判断，决定不同的敌机类型所计得分不同，再通过七段数码管以及计数器来实现得分的显示。

二、整体结构

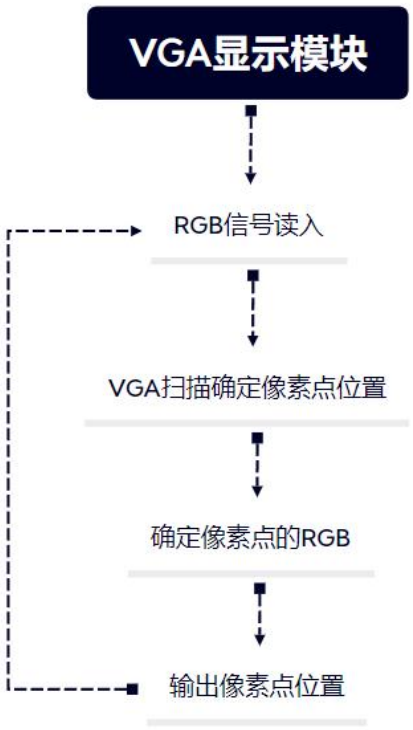
1、输入和输出

游戏中使用 ps2 键盘进行输入，通过上下左右四个按键控制飞机的移动，Enter 键控制游戏的开始。同时还有一个开关，控制在 Game over 后重置为开始界面，从而能够进行下一次游戏。

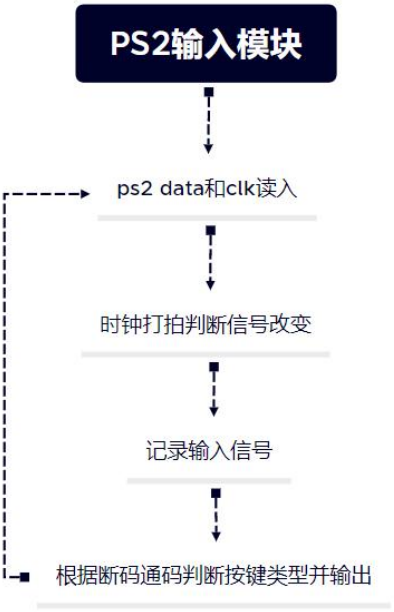
顶层模块的输出一部分为 VGA 显示所需要的行同步信号和场同步信号，以及每个像素点的 RGB 参数；另一部分为七段数码管输出数字所需的参数，从而在七段数码管上显示游戏得分。

2、模块工作流程

（1）VGA 显示模块

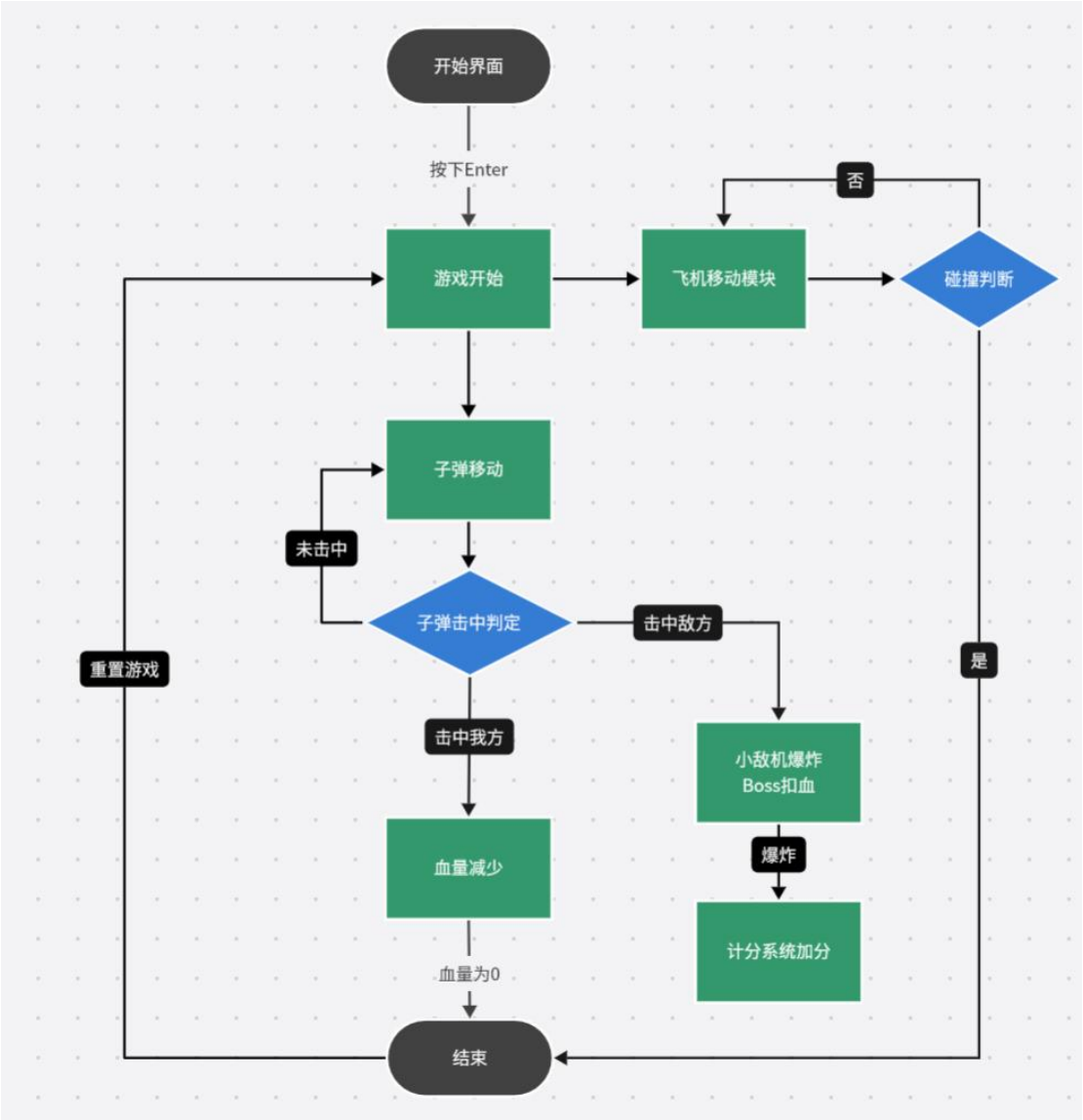


（2）ps2 输入模块

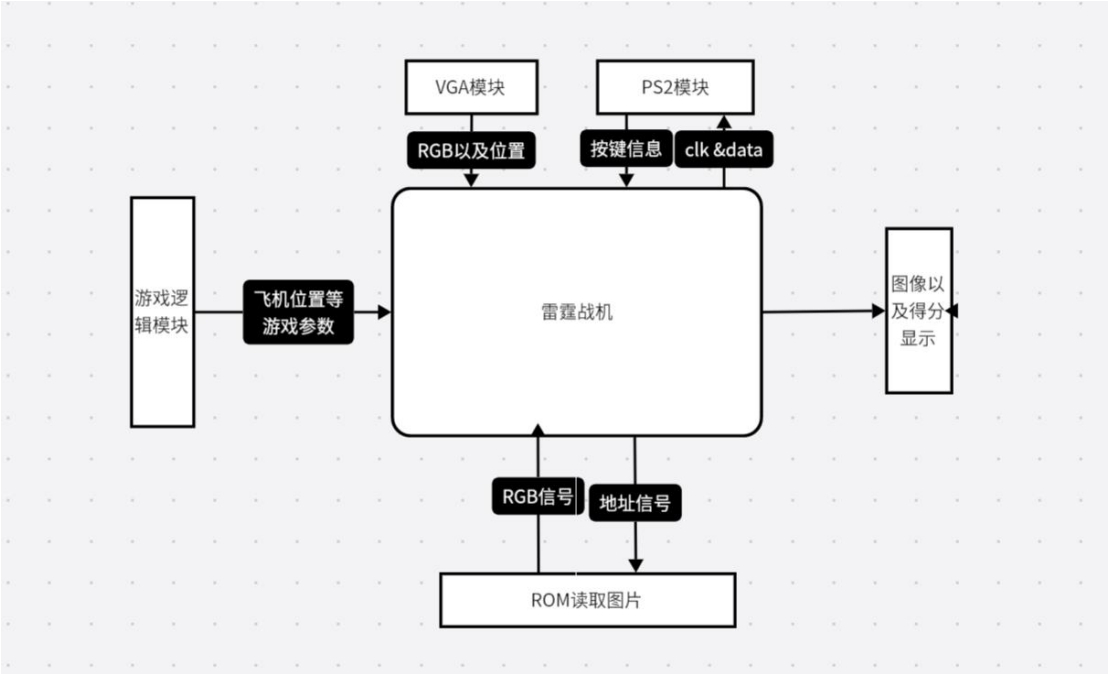


（3）IP core 的建立

（4）游戏逻辑示意图


开始界面
按下Enter
游戏开始
飞机移动模块
否
碰撞判断
是
子弹移动
未击中
子弹击中判定
击中敌方
重置游戏
击中我方
小敌机爆炸
Boss扣血
爆炸
血量减少
计分系统加分
血量为0
结束

（5）整体设计



3、游戏过程

上板后显示游戏封面，按下 Enter 键进入游戏，在屏幕上方有一架敌机，会从不同的位置出现，同时发射子弹，爆炸后经过一段时间会从上方重新出现。我方飞机在屏幕下方中央，Boss 在屏幕上方中央。

游戏开始后，敌机、Boss 与玩家飞机会不断发射子弹，玩家通过键盘上的上下左右四个按键控制飞机的移动，玩家飞机击中敌方飞机，敌方飞机爆炸，子弹消失，获得 1 分；击中 Boss，Boss 血量减少，血量为 0 时爆炸，获得 10 分；敌机与 Boss 在经过一段时间后会重新生成。

若敌机与玩家飞机碰撞，或者被击中三次，血量减为 0，则游戏结束。

游戏结束后，控制 SW[0]开关，重新返回游戏封面，按下 Enter 重新开始游戏。

三、各模块结构说明

（一）输入输出显示部分

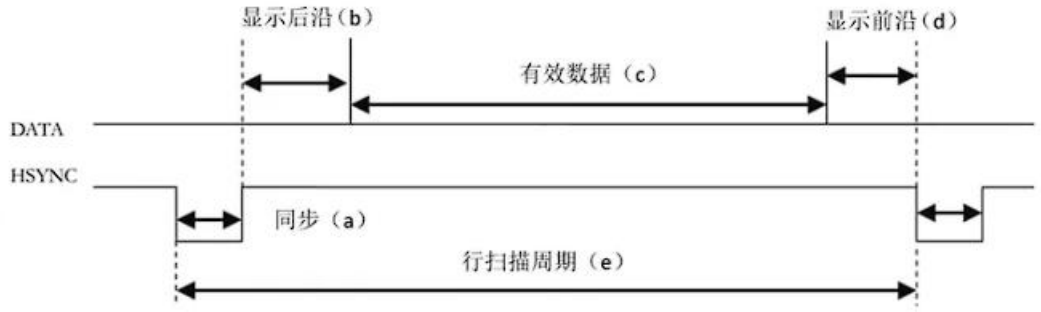
1. VGA 驱动模块的实现以及图片的导入

VGA 驱动模块的代码如下所示：

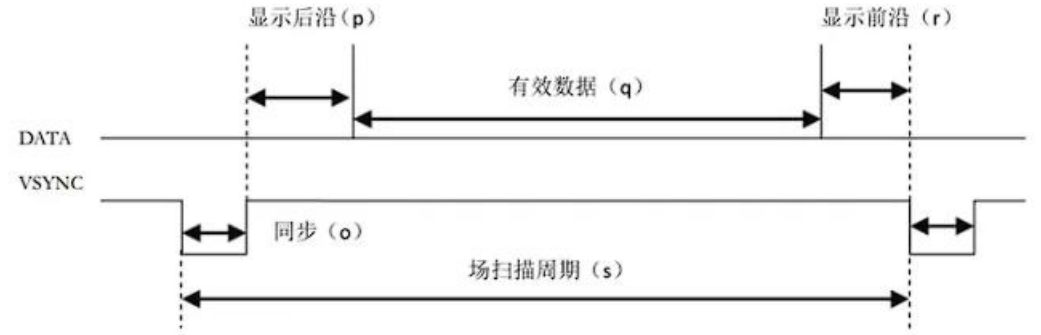
```
module Test (
    input clk,
    input rst,
    input [11:0] Din,
    output reg [11:0] rgb,
    output reg vsync,hsync,
    output reg [9:0] x,y,
    output reg EN
);
    reg [9:0] vcnt,hcnt;
    always @(posedge clk or posedge rst) begin
        if(rst)begin
            hcnt <= 10'b0;
        end
        else begin
            if(hcnt == 10'd799)hcnt <= 10'b0;
            else hcnt <= hcnt + 1;
        end
    end
    always @(posedge clk or posedge rst) begin
        if(rst)vcnt <= 10'b0;
        else begin
            if(hcnt == 10'd799)begin
                if(vcnt == 10'd524)vcnt <= 10'b0;
                else vcnt <= vcnt + 1;
            end
        end
    end

    always @(posedge clk) begin
        vsync <= vcnt > 2 ? 1 : 0;
        hsync <= hcnt > 96 ? 1 : 0;
        x <= hcnt - 10'd144;
        y <= vcnt - 10'd35;
        EN <= (vcnt > 10'd35) && (vcnt < 10'd515) && (hcnt > 144) && (hcnt < 784);
        if(EN) begin
```

```
            rgb <= Din;
        end
        else rgb <= 12'h000;
    end
endmodule
```

VGA 驱动显示的扫描时序图如下所示：



行同步时序



场同步时序

由时序图可知，我们需要根据 640*480@60Hz 的显示模式，确定行、列扫描中 a、b、c、d、e、p、r、o、s 的值，并根据图像像素的坐标来确定图像的扫描顺序。我们将每个像素的坐标值以及对应的 12 位 RGB 信号作为输入值，输出真实扫描的 RGB 信号以及周期使能信号。如果像素坐标在可显示坐标的范围内，则输出像素的真实值；如果像素坐标不在可显示的范围内，则输出 12 位全为 0 的黑色像素代表不显示。

在完成 VGA 驱动模块接口后，我们还需要完成对一张图片像素与坐标对应关系的导入。通过对应的 Python 程序（详见附录），我们能够将标准 24 位的 BMP 图像转化为对应的 coe 文件。利于 coe 文件，我们可以生成图片对应的 ip 核。在 Vivado 程序中依照相应的操作生成 ip 核后，我们将得到如下所示的模块接口：

```
explode_mem Boom (.clka(clk),.addra(col + row * 50),.douta(boom_rgb));
```

clka 为使用的时钟，addra 为像素所处位置，注意此处的位置是一维位置而非实际坐标，douta 为该像素对应的 RGB 值。我们将这些坐标与像素值接入 VGA 驱动模块，就可以在 VGA 板上得到想要显示的图像。

2. PS2 驱动模块的实现

PS2 驱动模块的实现如下所示：

```
module PS2 (
```

```
    input clk,
    input rst,
    input ps2_clk,
    input ps2_data,
    output reg up,
    output reg down,
    output reg left,
    output reg right,
    output reg enter
);

reg ps2_clk_flag0, ps2_clk_flag1, ps2_clk_flag2;
reg [7:0] temp_data;
reg [9:0] data;
reg data_break, data_done, data_expand;
reg [3:0] num;
always@(posedge clk or posedge rst)begin
    if(rst)begin
        ps2_clk_flag0<=1'b0;
        ps2_clk_flag1<=1'b0;
        ps2_clk_flag2<=1'b0;
    end
    else begin
        ps2_clk_flag0<=ps2_clk;
        ps2_clk_flag1<=ps2_clk_flag0;
        ps2_clk_flag2<=ps2_clk_flag1;
    end
end
wire negedge_ps2_clk = !ps2_clk_flag1 & ps2_clk_flag2;  //1 when negedge of ps2_clk
reg negedge_ps2_clk_shift;
always@(posedge clk)begin
    negedge_ps2_clk_shift<=negedge_ps2_clk;
end
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
always@(posedge clk or posedge rst)begin
    if(rst)
        temp_data<=8'd0;
```

```
    else if (negedge_ps2_clk_shift)begin
        case(num)
            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
    else temp_data<=temp_data;
end
always@(posedge clk or posedge rst)begin
    if(rst)begin
        data_break<=1'b0;
        data<=10'd0;
        data_expand<=1'b0;
    end
    else if(num==4'd11)begin
        if(temp_data==8'hE0)begin
            data_expand<=1'b1;
        end
        else if(temp_data==8'hF0)begin
            data_break<=1'b1;
        end
        else begin
            data<={data_expand,data_break,temp_data};
            data_expand<=1'b0;
            data_break<=1'b0;  //reset
        end
    end
    else begin
        data<=data;
        data_expand<=data_expand;
        data_break<=data_break;
    end
end
always @(posedge clk) begin
    case (data)
        10'h05A:enter <= 1;
        10'h15A:enter <= 0;
```

```
        10'h275:up <= 1;
        10'h375:up <= 0;
        10'h272:down <= 1;
        10'h372:down <=0;
        10'h26B:left <= 1;
        10'h36B:left <= 0;
        10'h274:right <= 1;
        10'h374:right <= 0;
    endcase
end
endmodule
```

本游戏按键操作只需要 PS2 键盘的箭头上、下、左、右键以及 ENTER 键，因此我们通过查询按键通码-断码表，将这五个按键特例化并作为输出，对应按键处于高位代表被按下、处于低位则代表被松开。我们使用这些输出来控制游戏的进程。

3. Arduino 数码管显示的实现

数码管显示实现代码如下所示：

```
module My74LS161(
    input CP,
    input CRn,
    input LDn,
    input [3:0] D,
    input CTT,
    input CTP,
```



```

    output [3:0] Q,
    output CO
);

    reg [3:0] Q_reg = 4'b0;
    always @(posedge CP or negedge CRn) begin
        if(!CRn) begin
            Q_reg <= 4'b0;
        end else begin
            if(!LDn) begin
                Q_reg <= D;
            end
            else if(CTT && CTP) begin
                Q_reg <= Q_reg + 4'b1;
            end
        end
```

```
    end
    assign Q = Q_reg;
    assign CO = (Q == 4'hF);
Endmodule

module DisplayNumber(
    input        clk,
    input        rst,
    input [15:0] hexs,
    input [ 3:0] points,
    input [ 3:0] LEs,
    output[ 3:0] AN,
    output[ 7:0] SEGMENT
);
    wire [31:0] w1;
    wire [3:0] w2;
    wire point,LE;
    clkdiv m0 (clk,rst,w1);
    DisplaySync m1 (w1[18:17],hexs,points,LEs,w2,AN,point,LE);
    MyMC14495 m2
(w2[0],w2[1],w2[2],w2[3],LE,SEGMENT[0],SEGMENT[1],SEGMENT[2],SEGMENT[3],SEGMENT
[4],SEGMENT[5],SEGMENT[6],SEGMENT[7],point);
endmodule
```

该模块是课程实验中实现过的部分。在显示部分。我们在 top 模块中实例化 4 个 My74LS161 模块分别作为 4 个数码管的输出计算模块，将计算所得的值导入 DIsplayNumber 模块显示在 Arduino 数码管上。

（二）飞机运行逻辑部分

1. 我方飞机的控制模块

我方飞机的控制模块代码如下所示：

```
module Plane_Judge (
    input clk,rst,
    input clk_move,
    input [9:0] x,y,
    input [3:0] direction,
    input boom,
    output reg [9:0] p_x,p_y,
    output EN,
    output myplane_exist,
    output reg [11:0] rgb
);
```

```
reg [9:0] p_x_next,p_y_next;
reg EN_reg;
reg [7:0] invincible_time,invincible_time_next;
wire [11:0] plane_rgb,boom_rgb;
wire [9:0] col,row;
reg [7:0] boom_count;

assign col = x - p_x;
assign row = y - p_y;
plane_mem Plane (.clka(clk),.addra(col + row * 50),.douta(plane_rgb));
explode_mem Boom (.clka(clk),.addra(col + row * 50),.douta(boom_rgb));
always @(posedge clk or posedge rst) begin
    if(rst) begin
        p_x <= 320 - 25;
        p_y <= 480 - 50;
    end
    else begin
        p_x <= p_x_next;
        p_y <= p_y_next;
    end
end
always @(posedge clk_move) begin
    p_x_next = p_x;
    if(boom_count == 8'b0) begin
        case (direction)
            4'b0100: begin
                if(p_x > 0) p_x_next <= p_x - 1;
            end
            4'b1000: begin
                if(p_x < 640 - 50) p_x_next <= p_x + 1;
            end
        endcase
    end
end
always @(posedge clk_move) begin
    p_y_next = p_y;
    if(boom_count == 8'b0) begin
        case (direction)
            4'b0001:begin
                if(p_y > 0)p_y_next <= p_y - 1;
            end
            4'b0010:begin
                if(p_y < 480-50) p_y_next <= p_y + 1;
```

```
                end
            endcase
        end
    end
    // boom_count is used to create a time duration to play the boom picture, in
this time, the plane can't move.
    always @(posedge clk_move or posedge rst) begin
        if(rst) begin
            boom_count <= 8'b0;
        end
        else if(boom) begin
            if(boom_count < 8'b11111111) boom_count <= boom_count + 1;
            else boom_count <= boom_count;
        end
        else begin
            boom_count <= 8'b0;
        end
    end

    always @* begin
        EN_reg <= 0;
        if(boom_count > 8'b0 && boom_count < 8'b11111111) begin
            rgb <= boom_rgb;
            if(boom_rgb != 12'b111111111111) EN_reg <= 1;
        end
        else begin
            rgb <= plane_rgb;
            if(plane_rgb != 12'b111111111111 && boom_count != 8'b11111111) EN_reg
<= 1;
        end
    end
    assign EN = (x >= p_x && x < p_x + 50 && y >= p_y && y < p_y + 50 & EN_reg);
    assign myplane_exist = (boom_count == 8'b0);
endmodule
```

模块核心逻辑为：判断我方飞机是否产生移动以及是否发生爆炸。

如果通过 PS2 键盘输入产生移动（即 direction[4:0]的值发生变化），则执行对应的移动操作（即对应移动飞机坐标至对应位置）。

如果发生爆炸（即 boom 的值发生变化），则爆炸计时寄存器开始计时。在计时期间，飞机图像被替换为爆炸图像，同时飞机无法通过 PS2 键盘的输入进行移动操作。在爆炸计时结束后，飞机死亡，游戏结束。

2. 敌方飞机与 BOSS 的控制模块

敌方飞机的控制模块如下所示：

```
module Enemyplane_Judge (
    input clk,rst,
    input clk_move,
    input boom,
    input revive,
    input [9:0] x,y,
    output reg [9:0] enemy_x,enemy_y,
    output reg [11:0] rgb,
    output enemyplane_exist,
    output enemy_en,
    output reg Counter
);

    reg [9:0] e_next_x,e_next_y;
    reg EN_reg;
    wire [11:0] enemy_rgb,boom_rgb;
    wire [9:0] col,row;
    reg [7:0] boom_count;
    reg counter1 ,counter2, counter3;
    assign col = x - enemy_x;
    assign row = y - enemy_y;
    enemyplane_mem Enemy (.clka(clk),.addra(col + row * 50),.douta(enemy_rgb));
    explode_mem Boom (.clka(clk),.addra(col + row * 50),.douta(boom_rgb));
    always @(posedge clk or posedge rst) begin
        if(rst)begin
            enemy_x <= 120-25;
            enemy_y <= 0;
        end
        else begin
            enemy_x <= e_next_x;
            enemy_y <= e_next_y;
        end
    end
    always @(posedge clk_move) begin
        e_next_x <= enemy_x;
        e_next_y <= enemy_y;
        if(boom_count == 8'b0) begin
            if(revive) begin
                e_next_x <= (enemy_x + 100 > 640 ? 120 : enemy_x + 100);
                e_next_y <= 0;
            end
            else if(enemy_y < 430) begin
                e_next_y <= enemy_y + 1;
```

```
        end
        else begin
            e_next_x <= (enemy_x + 100 > 640 ? 120 : enemy_x + 100);
            e_next_y <= 0;
        end
    end
end
always @(posedge clk_move or posedge rst) begin
    if(rst) begin
        boom_count <= 8'b0;
    end
    else if(boom) begin
        if(boom_count < 8'b11111111) begin
            boom_count <= boom_count + 1;
        end
        else begin
            boom_count <= boom_count;
        end
    end
    else begin
        boom_count <= 8'b0;
    end
end

always@ (posedge clk_move or posedge rst) begin
    if(rst) begin
        counter1 <= 1'b0;
        counter2 <= 1'b0;
        counter3 <= 1'b0;
        Counter <= 1'b0;
    end
    else begin
        counter1 <= ~enemyplane_exist;
        counter2 <= counter1;
        counter3 <= counter2;
        if(counter2 == 1 && counter3 == 0) begin
            Counter <= 1'b1;
        end
        else Counter <= 1'b0;
    end
end
always @* begin
    EN_reg <= 0;
    if(boom_count > 8'b0 && boom_count < 8'b11111111) begin
```

```
            rgb <= boom_rgb;
            if(boom_rgb != 12'b111111111111) EN_reg <= 1;
        end
        else begin
            rgb <= enemy_rgb;
            if(enemy_rgb != 12'b111111111111 && boom_count != 8'b11111111) EN_reg
<= 1;
        end
    end
    assign enemy_en = (EN_reg && x >= enemy_x && x < enemy_x + 50 && y >= enemy_y
&& y < enemy_y + 50 && EN_reg);
    assign enemyplane_exist = (boom_count == 8'b0);

endmodule
```

敌方 BOSS 的控制模块与敌方飞机的控制模块类似，在此不再重复展示。

模块核心逻辑为：使敌机按照固定的移动逻辑在地图中移动，并判断敌机是否爆炸。

如果未发生爆炸，则敌机按照自上到下的固定速度在地图中移动。当移动到地图的最下端时，其会水平平移一个机位并出现在地图的最上端。

如果发生爆炸（即 boom 的值发生变化），则爆炸计时寄存器开始计时。在计时期间，敌机图像被替换为爆炸图像，同时敌机不再移动及发射子弹。在爆炸计时结束后，敌机死亡消失，重生寄存器开始计时。重生计时结束后，敌机重新出现在死亡时的横坐标最上端，重复先前的逻辑。

与敌机不同的是，敌方 BOSS 不会移动，重生时长长于敌机。

（三）子弹运行逻辑部分

1. 子弹飞行逻辑

我方飞机、敌机以及 BOSS 子弹的飞行逻辑均相同，在此只展示一种：

```
module Bullet_Judge (
    input clk,rst,
    input clk2,
    input [9:0] p_x,p_y,
    input [9:0] startp_x,startp_y,
    input [9:0] x,y,
    input collide,  //0 for my bullet collide, 1 for my bullet exist
    output reg [9:0] b_x,b_y,
    output mybullet_en,  //1 for my bullet can show on the VGA
    output mybullet_exist,  //1 for my bullet exist
    output reg [11:0] mybullet_rgb
);
```

```
    reg [9:0] b_x_next,b_y_next;
    reg EN_reg;
    reg collide_EN;  //1 for my bullet collide, 0 for my bullet exist
    wire [11:0] bullet_rgb;
    wire [9:0] col,row;

    assign row = y + 480 - b_y;
    assign col = x - b_x;

    bullet_mem Bullet (.clka(clk),.addra(col + row * 10),.douta(bullet_rgb));
    always @(posedge clk or posedge rst) begin
        if(rst) begin
            b_x <= startp_x;
            b_y <= startp_y;
        end
        else begin
            b_x <= b_x_next;
            b_y <= b_y_next;
        end
    end

    always @(posedge clk2 or posedge rst) begin
        if(rst)begin
            b_x_next <= startp_x;
            b_y_next <= startp_y;
            collide_EN <= 1'b0;
        end
        else if(b_y + 480 < p_y) begin
            b_x_next <= p_x + 10'd23;
            b_y_next <= p_y - 10'd40;
            collide_EN <= 1'b0;
        end
        else begin
            b_x_next <= b_x;
            b_y_next <= b_y - 1;
            if(~collide) collide_EN <= 1'b1;
        end
    end
    always @* begin
        mybullet_rgb <= bullet_rgb;
    end
    assign mybullet_en = (x >= b_x && x < b_x + 10 && y + 480 >= b_y && y + 480 <
b_y + 40 && b_y > 480 & ~collide_EN);
    assign mybullet_exist = (collide_EN == 1'b0);
```

```
endmodule
```

模块核心逻辑为：是否飞出屏幕外以及是否发生与飞机的碰撞。

如果没有发生与飞机的碰撞，则子弹按照一个固定速度飞行（我方飞机为向上飞行，敌方飞机与 BOSS 为向下飞行）。对子弹进行初始位置赋值为飞机的前端一个子弹的位置。当子弹射出屏幕后，由于坐标不满足显示要求不再显示。当坐标满足重置位置的判断要求后，将子弹位置重置为初始位置并重新显示。由于子弹的移动速度较快而地图较小，在视觉上可达到飞机不断发射子弹的效果。

如果发生与飞机的碰撞（即 collide 信号的值发生变化），则子弹立刻不再显示。但由于此时子弹的坐标仍在移动，所以仍然可满足上述坐标判定的规则。当坐标满足重置位置的判断要求后，将子弹位置重置为初始位置并重新显示。

2. 子弹碰撞逻辑

我方飞机、敌机以及 BOSS 子弹的碰撞逻辑均相同，在此只展示一种：

```
module Collision_Judge (
    input clk,rst,
    input [9:0] mp_x,mp_y,ep_x,ep_y,
    input mp_exist,ep_exist,
    output reg collision
);

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            collision <= 0;
        end
        else begin
            if(mp_x + 50 >= ep_x && mp_x <= ep_x + 50 && mp_y + 50 >= ep_y && mp_y
<= ep_y + 50 && mp_exist && ep_exist) begin
                collision <= 1;
            end
        end
    end

endmodule
```

在设置各个模块显示坐标时，我们将坐标原点均设置在图像的左上角，飞机与敌机的大小为 50*50，BOSS 的大小为 128*128；飞机与敌机子弹的大小为 10*40，BOSS 子弹的大小为 20*60。在判断是否碰撞时，我们只要根据实际几何关系，得出在飞机坐标原点与子弹坐标原点存在什么样的大小关系时，飞机与子弹的实际图像会产生重叠，即可得出碰撞逻辑的实现。输出 collision 作为发生碰撞的使能信号，传入各飞机与子弹的运行模块中，执行对应的逻辑。

（四）血量与分数逻辑部分

1. 血量模块实现

血量模块的实现如下所示：

```
module Health_Judge (
    input clk, rst,
    input [9:0] x, y,
    input [3:0] present_health,
    input [3:0] present_bhealth,
    output reg [11:0] health1_rgb,
    output reg [11:0] health2_rgb,
    output reg [11:0] health3_rgb,
    output health_EN1,
    output health_EN2,
    output health_EN3,
    output bhealth_EN1,
    output bhealth_EN2,
    output bhealth_EN3,
    output bhealth_EN4,
    output bhealth_EN5,
    output bhealth_EN6,
    output bhealth_EN7,
    output bhealth_EN8
);

    always @* begin
        health1_rgb <= 12'b111111111111;
        health2_rgb <= 12'b111111111111;
        health3_rgb <= 12'b111111111111;
    end
    assign health_EN1 = (x >= 420 && x <= 480 && y >= 460 && y <= 470 && present_health >=
4'b0001);
    assign health_EN2 = (x >= 480 && x <= 540 && y >= 460 && y <= 470 && present_health >=
4'b0010);
    assign health_EN3 = (x >= 540 && x <= 600 && y >= 460 && y <= 470 && present_health >=
4'b0011);
    assign bhealth_EN1 = (x >= 400 && x <= 425 && y >= 10 && y <= 20 && present_bhealth >=
4'b0001);
    assign bhealth_EN2 = (x >= 425 && x <= 450 && y >= 10 && y <= 20 && present_bhealth >=
4'b0010);
    assign bhealth_EN3 = (x >= 450 && x <= 475 && y >= 10 && y <= 20 && present_bhealth >=
4'b0011);
```

```
    assign bhealth_EN4 = (x >= 475 && x <= 500 && y >= 10 && y <= 20 && present_bhealth >=
4'b0100);
    assign bhealth_EN5 = (x >= 500 && x <= 525 && y >= 10 && y <= 20 && present_bhealth >=
4'b0101);
    assign bhealth_EN6 = (x >= 525 && x <= 550 && y >= 10 && y <= 20 && present_bhealth >=
4'b0110);
    assign bhealth_EN7 = (x >= 550 && x <= 575 && y >= 10 && y <= 20 && present_bhealth >=
4'b0111);
    assign bhealth_EN8 = (x >= 575 && x <= 600 && y >= 10 && y <= 20 && present_bhealth >=
4'b1000);
endmodule
```

这部分主要是计算各个血量部分的使能信号是否显示。因为我们定义了我方飞机血量上限为 3，敌方飞机血量上限为 1，敌方 BOSS 血量上限为 8，我们可以由传入的血量值来计算对应血量显示的使能。特别地，敌方飞机不需要特意显示血量。

2. 计分模块实现

计分模块实现如下所示：

```
//Counter
    My74LS161 Counter1 (.CP(clk_move),.LDn(~rst & play_en),.CRn(~(one ==
4'd10)),.CTT(Counter),.CTP(1'b1),.D(4'b0),.Q(one),.CO());
    My74LS161 Counter2 (.CP(clk_move),.LDn(~rst & play_en),.CRn(~(ten ==
4'd10)),.CTT(one == 4'd9 || Counter2),.CTP(1'b1),.D(4'b0),.Q(ten),.CO());
    My74LS161 Counter3 (.CP(clk_move),.LDn(~rst & play_en),.CRn(~(hundred ==
4'd10)),.CTT(one == 4'd9 && ten == 4'd9),.CTP(1'b1),.D(4'b0),.Q(hundred),.CO());
    My74LS161 Counter4 (.CP(clk_move),.LDn(~rst & play_en),.CRn(~(thousand ==
4'd10)),.CTT(one == 4'd9 && ten == 4'd9 && hundred ==
4'd9),.CTP(1'b1),.D(4'b0),.Q(thousand),.CO());

    DisplayNumber
display_inst(.clk(clk), .hexs({thousand,hundred,ten,one}), .points(4'b1111), .rst
(1'b0), .LEs(4'b0000), .AN(AN), .SEGMENT(SEGMENT));
```

在 top 模块中我们实例化了 4 个 My74LS161 模块用于计算分值。在敌机爆炸时，敌机计分使能 Counter 会在一个时钟周期内输出 1；在 BOSS 爆炸时，BOSS 计分使能 Counter2 会在一个时钟周期内输出 1。我们设计敌机爆炸加 1 分，BOSS 爆炸加 10 分，则可以将 Counter 接在表示个位的 74LS161 模块计数使能端，将 Counter2 接在表示十位的 74LS161 模块计数使能端。再依照十进制计数器的性质完善其他使能，就实现了一个可以计分的显示模块。

（五）游戏结束部分

游戏结束部分显示代码如下所示：

```
module Gameover (
```

```
    input clk,rst,
    input gameover,
    input [9:0] x,y,
    output EN,
    output reg [11:0] rgb
);

    reg EN_reg;
    wire [9:0] col;
    wire [11:0] game_rgb;
    assign col = y - 180;
    gameover_mem Game (.addra(col * 640 + x),.douta(game_rgb),.clka(clk));

    always @* begin
        EN_reg = 0;
        if(game_rgb != 12'b111100000000)EN_reg = 1;
        rgb <= game_rgb;
    end
    assign EN = (x >= 0 && x <= 640 && y >= 160 && y <= 320 && gameover && EN_reg);

endmodule
```

在我方飞机血量降至 0 时，gameover 信号被置为 1，随后该模块将图像显示改变为游戏结束的图像，同时其他所有模块停止运行，等待游戏重启。

四、模块仿真分析

由于实验中我们小组在外设接口模块方面实现较为顺利，产生的困难主要出现在内部逻辑实现部分，因此小组成员的调试部分更多还是通过生成 bitstream 下板观察现象实现，使用的仿真分析数量较少，现介绍如下。

1. 对子弹判断逻辑的仿真分析

仿真文件如下所示：

```
`timescale 1ns/1ps

module tb();
reg clk,rst,clk2;
reg [9:0] p_x,p_y,startp_x,startp_y;
reg [9:0] x,y;
reg boom;
wire [9:0] b_x,b_y;
wire [11:0] mybullet_rgb;
```

```
wire mybullet_en;
Bullet_Judge gg(
    .clk(clk),
    .rst(rst),
    .clk2(clk2),
    .p_x(p_x),
    .p_y(p_y),
    .startp_x(startp_x),
    .startp_y(startp_y),
    .x(x),.y(y),
    .boom(boom),
    .b_x(b_x),.b_y(b_y),
    .mybullet_rgb(mybullet_rgb),
    .mybullet_en(mybullet_en)
);
initial begin
    clk = 0;
    clk2 = 0;
    rst = 1;
    p_x = 10'd270;
    p_y = 10'd430 + 10'd480;
    startp_x = 10'd270;
    startp_y = 10'd430 + 10'd520;
    #5;
    rst = 0;
    boom = 0;
end
    always #10 clk <= ~clk;
    always #20 clk2 <= ~clk2;
endmodule
```

在测试子弹运行的功能时，我们在仿真文件中对子弹的各个使能以及初位置进行了规定，意在探究子弹的移动是否会跟随时钟周期变化，以及时钟周期的大小变化是否会带来子弹运行使能的紊乱。仿真时通过坐标寄存器的结果，我们可以清楚地知道子弹有没有按照设计好的逻辑进行移动。最终小组成员得出结论：子弹移动的时钟周期 clk_move 必须小于坐标同步的时钟周期 clk，并且在 rst 信号置 1 重置时必须给有关坐标的所有变量都赋好初值，否则在重置时会因为时钟周期不同步的问题而出现不确定的值。

2. 对敌机运行逻辑的仿真分析

仿真文件如下所示：

```
`timescale 1ns/1ps

module tb2();
```

```
reg clk,rst,clk2;
reg [9:0] x,y;
reg boom;
wire [9:0] e_x,e_y;
wire [11:0] rgb;
wire e_en;
Enemyplane_Judge gg(
    .clk(clk),
    .rst(rst),
    .clk_move(clk2),
    .x(x),.y(y),
    .boom(boom),
    .enemy_x(e_x),.enemy_y(e_y),
    .enemy_en(e_en),
    .rgb(rgb)
);
initial begin
    clk = 0;
    clk2 = 0;
    rst = 1;
    #5;
    rst = 0;
    boom = 0;
end
    always #10 clk <= ~clk;
    always #20 clk2 <= ~clk2;
endmodule
```

在测试敌机飞行的功能是否正常时，我们在仿真文件中设置了敌机位置重置的时间以及不同的时钟周期。由于敌机的移动逻辑是由上至下自行移动，因此通过重置敌机位置并开始时钟周期，仿真时我们可以通过敌机坐标寄存器清楚地知道敌机运行逻辑是否正常。小组成员在仿真后，发现敌机运行时不会自行左右移动，在修改后小组成员也最终得到了正确的逻辑模块。

五、游戏实现演示

1. 游戏开始界面

游戏开始界面介绍了游戏名称“雷霆战机”。在开始界面还有进入游戏的提示，按下 ENTER 键即可开始游戏。

2. 游戏运行界面

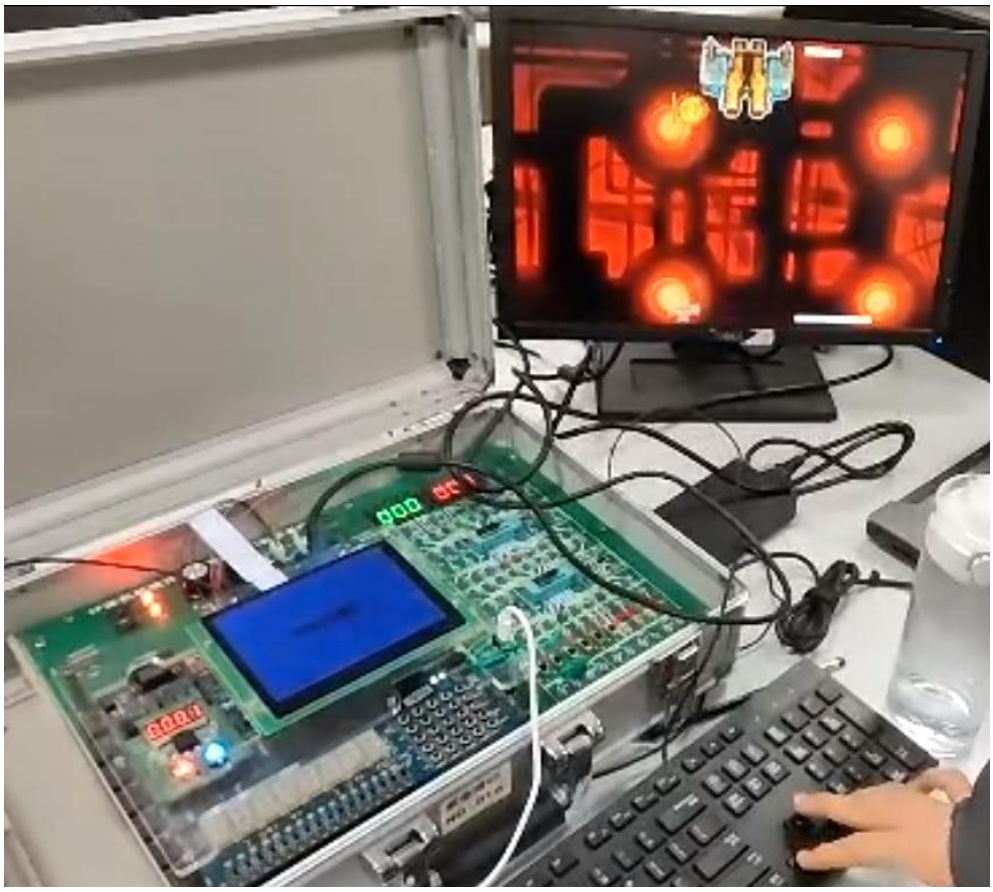
进入游戏后，可看到我方飞机、敌机及 BOSS 其中我方飞机可通过 PS2 键盘进行移动控制。我方飞机发生子弹为蓝色，敌机发生子弹为黄色，BOSS 发射子弹为绿色。在我方飞机与 BOSS 初始位置旁（即屏幕的最下端与最上端）分别显示我方飞机与 BOSS 的血量值，飞机被子弹击中后，血量值会对应减少。

3. 游戏结束界面

若我方飞机血量降至 0 或我方飞机与敌机发生碰撞，则我方飞机阵亡，游戏结束。屏幕显示“GAME OVER”字样。此时可通过 SWORD 板上的开关按钮控制游戏重新开始。

4. 计分界面

在我方飞机击杀敌方飞机一次后，计分板计数会增加 1；在我方飞机击杀敌方 BOSS 一次后，计分板计数会上升 10。玩家可通过 Arduino 板上数码管的计数观察到自己获得的分数。



六、反思与总结

本次实验设计总体来说难度较大，涉及到的知识面比较广，对于理论与实践的结合要求也较高。由于最开始的设想较为复杂，在实现时也遇到了诸多困难，最终小组成员集思广益，在有限的时间内舍弃了部分华而不实的功能，换取了较高的完成度。

首先在子弹的移动逻辑部分，由于使能约束的原因，子弹运行的逻辑总是与预想的逻辑不符，比如碰撞后不再显示、重置位置不是飞机当前位置等等。这些问题有时通过仿真不一定能发现，只能在模块代码书写时就事先规定好使能是高位有效还是低位有效，防止之后的逻辑出现错误。

同时在飞机的爆炸逻辑书写部分，小组成员也遇到了很多问题。爆炸图片该如何显示？计分模块该如何运行？敌机重生位置与敌方子弹显示在爆炸后该如何规定？这些问题都曾经困扰着小组。由于模块代码书写遵循着循序渐进的原则，在添加功能时也会有“牵一发而动全身”的问题，有时需要将使能重新规范约束。日后书写程序时还是应当实现设计好模块功能，尽量做到简洁易懂。

最后在 VGA 与 PS2 的输入输出部分，由于这部分理论部分并未涉及，需要小组成员自行查阅资料书写，同时这一部分直接影响到最终下板的演示结果，因此小组决定在最开始就解决这方面问题。通过查阅助教提供的资料以及学长的代码资源，小组理解了 VGA 与 PS2 背后的运行逻辑，并成功实现了代码的自行书写。

设计完成后，小组认为该游戏还有很大改进的空间：

七、小组分工说明及贡献比例

1. 成员分工

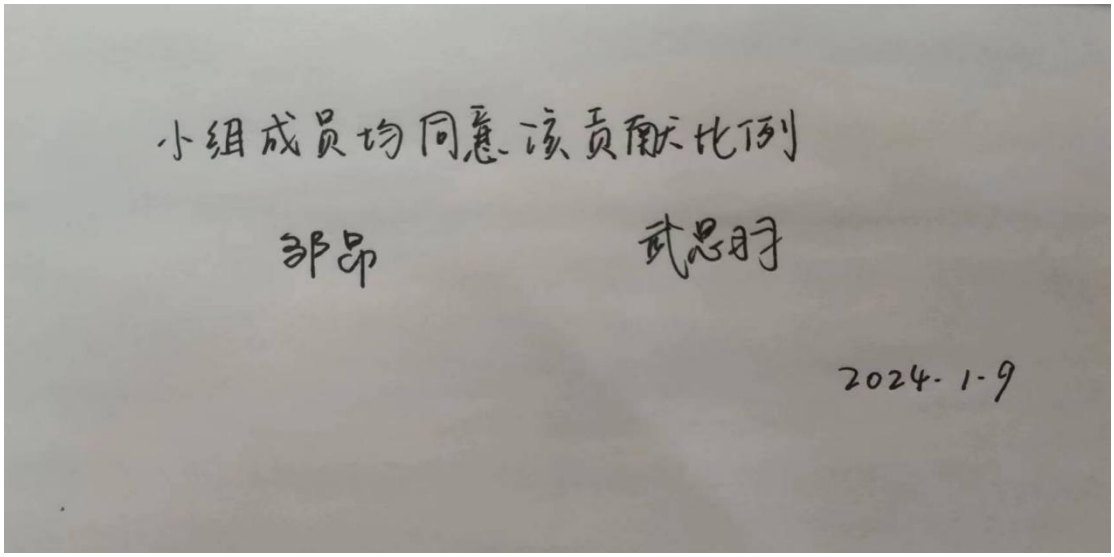
邹昂：VGA 驱动模块的书写、飞机运行和碰撞爆炸模块的书写、开始结束界面的书写、下板验证、撰写报告。

武思羽：PS2 驱动模块的书写、子弹运行和碰撞爆炸模块的书写、血量与计分模块的书写、下板验证、撰写报告。

2. 贡献比例

3220105648 邹昂： 50%

3220105846 武思羽： 50%



八、参考资料与资源

1. 课程郭助教的“外设使用”链接：https://guahao31.github.io/2023_DD/final_project/device/
2. 潘学长工程代码的外设接口部分：https://github.com/PAN-Ziyue/FPGA--JOJO/blob/master/JOJO/Framework/PS2.v

小组成员在这些参考资料与资源的基础上，理解了 VGA 与 PS2 模块的驱动原理，并自行书写了这些模块的代码，详见附录。

其余逻辑实现部分均由小组成员自行独立书写，无参考。

附录：源代码

Top.v

```
module Top (
    input clk,rst,
    input ps2_clk,ps2_data,
    output [11:0] rgb,
    output vsync,
    output hsync,
    output [3:0] AN,
    output [7:0] SEGMENT
);

    wire clk_out,clk_move,
```

```
             clk_move_bullet;
wire play_rst;
wire [11:0] background_rgb,
            start_rgb,
            myplane_rgb,
            enemy_rgb,
            boss_rgb,
            mybullet_rgb,
            gameover_rgb,
            enemybullet_rgb,
            bossbullet_rgb,
            health1_rgb,
            health2_rgb,
            health3_rgb;
reg [11:0] select_rgb;
wire [9:0] myplane_x,myplane_y, // The positions of the game's every elements
           enemy_x,enemy_y,
           mybullet_x,mybullet_y,
           ebullet_x,ebullet_y,
           boss_x,boss_y,
           bbullet_x, bbullet_y;
wire myplane_en, // Determine at the position (x,y) ,output what element
     enemy_en,
     mybullet_en,
     enemybullet_en,
     boss_en,
     gameover_en,
     end_en,
     mp_exist,
     mb_exist,
     ep_exist,
     eb_exist,
     bb_exist,
     health_EN1,
     health_EN2,
     health_EN3,
     bhealth_EN1,
     bhealth_EN2,
     bhealth_EN3,
     bhealth_EN4,
     bhealth_EN5,
     bhealth_EN6,
     bhealth_EN7,
     bhealth_EN8;
```

```
    wire p_boom, ep_boom, b_collide, b_collide2, eb_collide, bb_collide ,collision;
    wire [3:0] present_health,present_bhealth;
    wire Counter, Counter2;
    reg play_en;
    wire [3:0] direction;
    wire [9:0] x,y;
    reg [11:0] rgb_reg; // Use to store the RGB of the element needed to output
    //Clock modules
    clk_wiz_0 m1 (.clk_in1(clk),.reset(rst),.clk_out1(clk_out),.locked());
//output the clk of 25.175Mhz using MMCM
    clk_10ms CLK_MOVE (.clk(clk),.clk_10ms(clk_move));
    clk_2ms CLK_MOVE_BULLET (.clk(clk),.clk_2ms(clk_move_bullet));
    //PS2 module
    PS2 KeyBoard
        (.clk(clk),.rst(rst),.ps2_data(ps2_data),.ps2_clk(ps2_clk),.up(directio
n[0]),.down(direction[1]),.left(direction[2]),.right(direction[3]),.enter(play_
rst));
    //Boom Judge
    My_Boom_Judge MyBoom (.clk(clk_out), .rst(rst |
~play_en), .clk2(clk_move_bullet), .enemy_bullet_en(eb_exist), .boss_bullet_en(bb
_exist), .my_en(mp_exist),
                          .p_x(myplane_x), .p_y(myplane_y), .eb_x(ebullet_x), .eb_
y(ebullet_y), .bb_x(bbullet_x), .bb_y(bbullet_y), .my_health(4'b0011),
                          .boom(p_boom), .present_eb_en(eb_collide), .present_bb_
en(bb_collide), .present_health(present_health) );

    Enemy_Boom_Judge EnemyBoom (.clk(clk_out), .rst(rst |
~play_en), .clk2(clk_move_bullet), .mybullet_en(mb_exist), .enemy_en(ep_exist),
                               .ep_x(enemy_x), .ep_y(enemy_y), .b_x(mybullet_x),
 .b_y(mybullet_y), .revive(revive), .enemy_health(3'b001),
                               .present_mb_en(b_collide), .boom(ep_boom) );
    Boss_Boom_Judge BossBoom (.clk(clk_out), .rst(rst |
~play_en), .clk2(clk_move_bullet), .mybullet_en(mb_exist), .boss_en(boss_exist),
                              .boss_x(boss_x), .boss_y(boss_y), .b_x(mybullet_x),
 .b_y(mybullet_y), .boss_health(4'b1000),
                              .present_bhealth(present_bhealth), .boom(b_boom), .
present_mb_en(b_collide2) );
    //Heart Judge
    Health_Judge MyHealth (.clk(clk_out), .rst(rst |
~play_en), .x(x), .y(y), .present_health(present_health), .present_bhealth(presen
t_bhealth),
                           .health1_rgb(health1_rgb), .health2_rgb(health2_rgb),
 .health3_rgb(health3_rgb),
```

```
                        .health_EN1(health_EN1), .health_EN2(health_EN2), .hea
lth_EN3(health_EN3),.bhealth_EN1(bhealth_EN1),
                        .bhealth_EN2(bhealth_EN2), .bhealth_EN3(bhealth_EN3),
 .bhealth_EN4(bhealth_EN4), .bhealth_EN5(bhealth_EN5),
                        .bhealth_EN6(bhealth_EN6), .bhealth_EN7(bhealth_EN7),
 .bhealth_EN8(bhealth_EN8));
    //Planes Judge
    Plane_Judge MyPlane
          (.clk(clk_out), .clk_move(clk_move), .rst(rst |
~play_en), .x(x), .y(y), .boom(p_boom ||
collision), .direction(direction), .p_x(myplane_x), .p_y(myplane_y),
          .EN(myplane_en), .myplane_exist(mp_exist), .rgb(myplane_rgb) );
    Enemyplane_Judge Enemyplane (.clk(clk_out), .rst(rst |
~play_en), .clk_move(clk_move), .x(x), .y(y), .boom(ep_boom ||
collision), .Counter(Counter),
                                .enemy_x(enemy_x), .enemy_y(enemy_y), .rgb(enemy_r
gb), .enemy_en(enemy_en), .enemyplane_exist(ep_exist) ,.revive(revive));
    Boss_Judge BossPlane (.clk(clk_out), .rst(rst |
~play_en), .clk_move(clk_move), .x(x), .y(y), .boom(b_boom),
                        .boss_x(boss_x), .boss_y(boss_y), .rgb(boss_rgb), .boss_
EN(boss_en), .boss_exist(boss_exist), .Counter2(Counter2) );
    //Bullets Judge
    Bullet_Judge MyBullet (.clk(clk_out),.clk2(clk_move_bullet),.rst(rst |
~play_en),.x(x),.y(y),
        .p_x(myplane_x),.p_y(myplane_y + 10'd480),.startp_x(myplane_x +
23),.startp_y(myplane_y + 10'd440),.collide(b_collide & b_collide2),
        .b_x(mybullet_x),.b_y(mybullet_y),.mybullet_en(mybullet_en),.mybullet_r
gb(mybullet_rgb), .mybullet_exist(mb_exist) );
    Enemy_Bullet_Judge EnemyBullet (.clk(clk_out), .rst(rst |
~play_en), .clk2(clk_move_bullet), .x(x), .y(y), .enemyplane_exist(ep_exist),
                                    .ep_x(enemy_x), .ep_y(enemy_y +
10'd480), .startep_x(enemy_x + 23), .startep_y(enemy_y +
10'd520), .collide(eb_collide),
                                    .eb_x(ebullet_x), .eb_y(ebullet_y), .enemy_bul
let_en(enemybullet_en), .enemy_bullet_rgb(enemybullet_rgb), .enemybullet_exist(e
b_exist) );
    Boss_Bullet_Judge BossBullet (.clk(clk_out), .rst(rst |
~play_en), .clk2(clk_move_bullet), .x(x), .y(y), .boss_exist(boss_exist),
                                .boss_x(boss_x), .boss_y(boss_y +
10'd480), .startboss_x(boss_x + 54), .startboss_y(boss_y +
10'd540), .collide(bb_collide),
                                .bb_x(bbullet_x), .bb_y(bbullet_y), .boss_bullet
_en(bossbullet_en), .boss_bullet_rgb(bossbullet_rgb), .bossbullet_exist(bb_exist
) );
```

```
    Collision_Judge Collision (.clk(clk_out),.rst(rst |
~play_en),.mp_x(myplane_x),.mp_y(myplane_y),.ep_x(enemy_x),.ep_y(enemy_y),.coll
ision(collision),.mp_exist(mp_exist),.ep_exist(ep_exist));
    //Maps Judge
    Gameover Gameover_Judge (.clk(clk_out),.rst(rst |
~play_en),.x(x),.y(y),.EN(gameover_en),.rgb(gameover_rgb),.gameover(~mp_exist ||
collision));
    background_mem Background
(.clka(clk_out),.addra(y*640+x),.douta(background_rgb));
    start_mem Start (.clka(clk_out),.addra(y*640+x),.douta(start_rgb));
    //VGA module
    Test m0
(.clk(clk_out),.rst(rst),.Din(select_rgb),.rgb(rgb),.vsync(vsync),.hsync(hsync),
.x(x),.y(y),.EN());
    wire [3:0] one,ten,hundred,thousand;
    //Counter
    My74LS161 Counter1 (.CP(clk_move),.LDn(~rst & play_en),.CRn(~(one ==
4'd10)),.CTT(Counter),.CTP(1'b1),.D(4'b0),.Q(one),.CO());
    My74LS161 Counter2 (.CP(clk_move),.LDn(~rst & play_en),.CRn(~(ten ==
4'd10)),.CTT(one == 4'd9 || Counter2),.CTP(1'b1),.D(4'b0),.Q(ten),.CO());
    My74LS161 Counter3 (.CP(clk_move),.LDn(~rst & play_en),.CRn(~(hundred ==
4'd10)),.CTT(one == 4'd9 && ten == 4'd9),.CTP(1'b1),.D(4'b0),.Q(hundred),.CO());
    My74LS161 Counter4 (.CP(clk_move),.LDn(~rst & play_en),.CRn(~(thousand ==
4'd10)),.CTT(one == 4'd9 && ten == 4'd9 && hundred ==
4'd9),.CTP(1'b1),.D(4'b0),.Q(thousand),.CO());
    DisplayNumber
display_inst(.clk(clk), .hexs({thousand,hundred,ten,one}), .points(4'b1111), .rst
(1'b0), .LEs(4'b0000), .AN(AN), .SEGMENT(SEGMENT));
    always @(posedge clk or posedge rst) begin
        if(rst)begin
            play_en <= 0;
        end
        else if(play_rst) begin
            play_en <= 1;
        end
        else begin
            play_en <= play_en;
        end
    end
    always @* begin
        if(!play_en) rgb_reg = start_rgb;
        else begin
            if(mp_exist)begin
                if(myplane_en) rgb_reg = myplane_rgb;
```

```
                else if(health_EN1) rgb_reg = health1_rgb;
                else if(health_EN2) rgb_reg = health2_rgb;
                else if(health_EN3) rgb_reg = health3_rgb;
                else if(bhealth_EN1) rgb_reg = health1_rgb;
                else if(bhealth_EN2) rgb_reg = health1_rgb;
                else if(bhealth_EN3) rgb_reg = health1_rgb;
                else if(bhealth_EN4) rgb_reg = health1_rgb;
                else if(bhealth_EN5) rgb_reg = health1_rgb;
                else if(bhealth_EN6) rgb_reg = health1_rgb;
                else if(bhealth_EN7) rgb_reg = health1_rgb;
                else if(bhealth_EN8) rgb_reg = health1_rgb;
                else if(enemy_en) rgb_reg = enemy_rgb;
                else if(boss_en) rgb_reg = boss_rgb;
                else if(mybullet_en) rgb_reg = mybullet_rgb;
                else if(bossbullet_en) rgb_reg = bossbullet_rgb;
                else if(enemybullet_en) rgb_reg = enemybullet_rgb;
                else rgb_reg = background_rgb;
            end
            else begin
                if(gameover_en) rgb_reg = gameover_rgb;
                else rgb_reg = background_rgb;
            end
        end
    end
    always @(posedge clk_out) begin
        select_rgb <= rgb_reg;
    end

endmodule
```

其余代码在上述报告中已经提及，同时也已经提交 code 文件，包含了实验中用到的使用 logisim 生成的代码。