浙江大学

本科课程论文

坦克大战

课程名称：	数字逻辑设计
姓名：	李弘浩
姓名：	刘书含
姓名：	陈益扬
姓名：	缪晨露
学院：	竺可桢学院
专业：	计算机科学与技术
指导老师：	王总辉

2019 年 1 月 16 日

目录

1 绪论

1.1 设计背景

《坦克大战》是日本游戏开发商南梦宫与 1985 年开发的一款平面射击游戏。我们小组制作的为《坦克大战》的修改版双人对战游戏。

1.2 主要内容和难点

1. 主要内容

   双方玩家使用 PS2 键盘控制坦克在战场上的移动、射击，战场上有可破坏的砖块、可隐藏的草地。游戏的目的为击毁对方坦克，成功击毁对方坦克的玩家获胜。

2. 功能

   使用开关初始化游戏，玩家一使用 WSAD 控制上下左右移动，使用空格键控制子弹发射；玩家二使用上下左右箭头键控制移动，使用回车键控制子弹发射。双方坦克不能穿过边界和砖块等障碍物。当玩家击毁对方坦克时，显示胜利玩家，游戏结束。

3. 技术要求

   熟练掌握 verilog 语言设计；掌握 sword 开发板的使用方法了；掌握 VGA 显示原理与驱动模块设计；掌握 PS2 键盘工作原理与驱动模块设计；掌握并理解状态机的设计。

4. 目的

   加深对数字逻辑设计的掌握与理解、加强 verilog 硬件编程语言的掌握、增进团队合作能力

5. 实现重点与难点

   游戏逻辑、PS2 键盘输入、VGA 显示功能、自定义 IP 核显示图片

2 设计原理

2.1 设计相关软硬件基础

2.1.1 理论要点

1. VGA 显示

2. PS2 键盘输入

   PS/2 通信协议是一种双向同步串行通信协议。通信的两端通过 CLOCK(时钟脚) 同步，并通过国 DATA(数据脚) 交换数据。一般两设备间传输数据的最大时钟频率是 33kHz，大多数 PS/2 设备工作在 10–20kHz。推荐值在 15kHz 左右，也就是说，CLOCK 高、低电平的持续时间都为 40us。每一数据帧包含 11—12 位，具体含义如下表所示。

表 1: 通信数据帧格式

数据	含义
1 个起始位	总是逻辑 0
8 个数据位	(LSB) 地位在前
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

PS/2 到主机的通信时序如下图所示。数据在 PS/2 时钟的下降沿读取，PS/2 的时钟频率为 10—16.7kHz。对于 PS/2 设备，一般来说从时钟脉冲的上升沿到一个数据转变的时间至少要有 5us; 数据变化到下降沿的时间至少要有 5us，并且不大于 25us，这个时序非常重要应该严格遵循。主机可以再第 11 个时钟脉冲停止位之前把时钟线拉低，使设备放弃发送当前字节，当然这种情况比较少见。在停止位发送后设备在发送下个包前应该至少等待 50us，给主机时间做相应的处理。不过主机处理接收到的字节时一般会抑制发送 (主机在收到每个包时通常自动做这个)。在主机释放抑制后，设备至少应该在发送任何数据前等 50us。

3. 状态机的设计

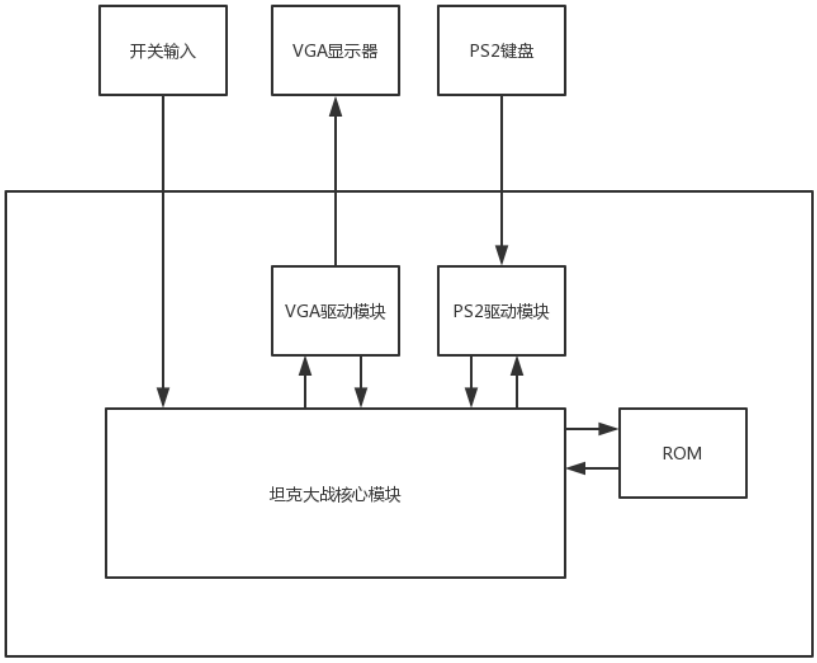
4. 硬件描述语言 Verilog HDL 是一种硬件描述语言 (HDL:Hardware Description Language)，以文本形式来描述数字系统硬件的结构和行为的语言，用它可以表示逻辑电路图、逻辑表达式，还可以表示数字逻辑系统所完成的逻辑功能。使用 Verilog 描述硬件的基本设计单元是模块 (module)。构建复杂的电子电路，主要是通过模块的相互连接调用来实现的。模块被包含在关键字 module、endmodule 之内。实际的电路元件。Verilog 中的模块类似 C 语言中的函数，它能够提供输入、输出端口。

5. 可编程阵列逻辑

2.1.2 课程设计方法

利用 ISE 软件，通过 verilog HDL 硬件描述语言编写状态机，PS2 键盘输入外界数据，VGA 输出显示数据，编写后生成 bit 文件，利用可编程阵列逻辑烧制到板子上，最终实现人机互动。

2.2 设计方案



我们将整个工程分为 4 大部分，分别是：游戏核心模块（包括调度其他模块、实现游戏逻辑功能）；VGA 驱动模块；PS2 驱动模块；ROM（用于储存需要的图片）

3 设计实现

3.1 实现方法

3.1.1 地图显示逻辑

Listing 1: 地图显示逻辑

```
        //地图初始化
        save_map[0][1] = 4'b0010;
        // ......
        // 中间内容省略
        save_map[17][17] = 4'b0011;


        //地图显示
        if(row_addr >= 480 || col_addr >= 480)begin
            vga_data <= 0;
        end
        else if(save_map[row_addr / 24][col_addr / 24] == 4'b0001)begin  //green grass
            vga_data <= col_grass;
        end
        else if(bullet1_x_sqr[0] + bullet1_y_sqr[0] < bullet_radius * bullet_radius) begin
            vga_data <= 12'h347;
        end
        else if(bullet1_x_sqr[1] + bullet1_y_sqr[1] < bullet_radius * bullet_radius) begin
            vga_data <= 12'h347;
        end
        else if(bullet1_x_sqr[2] + bullet1_y_sqr[2] < bullet_radius * bullet_radius) begin
            vga_data <= 12'h347;
        end
        else if(bullet1_x_sqr[3] + bullet1_y_sqr[3] < bullet_radius * bullet_radius) begin
            vga_data <= 12'h347;
        end
        else if(bullet1_x_sqr[4] + bullet1_y_sqr[4] < bullet_radius * bullet_radius) begin
            vga_data <= 12'h347;
        end
        else if(bullet1_x_sqr[5] + bullet1_y_sqr[5] < bullet_radius * bullet_radius) begin
            vga_data <= 12'h347;
        end
        else if(bullet1_x_sqr[6] + bullet1_y_sqr[6] < bullet_radius * bullet_radius) begin
            vga_data <= 12'h347;
        end
        else if(bullet1_x_sqr[7] + bullet1_y_sqr[7] < bullet_radius * bullet_radius) begin
            vga_data <= 12'h347;
```

```
38          end
39          else if(bullet2_x_sqr[0] + bullet2_y_sqr[0] < bullet_radius * bullet_radius)
                begin
40              vga_data <= 12'hca9;
41          end
42          else if(bullet2_x_sqr[1] + bullet2_y_sqr[1] < bullet_radius * bullet_radius)
                begin
43              vga_data <= 12'hca9;
44          end
45          else if(bullet2_x_sqr[2] + bullet2_y_sqr[2] < bullet_radius * bullet_radius)
                begin
46              vga_data <= 12'hca9;
47          end
48          else if(bullet2_x_sqr[3] + bullet2_y_sqr[3] < bullet_radius * bullet_radius)
                begin
49              vga_data <= 12'hca9;
50          end
51          else if(bullet2_x_sqr[4] + bullet2_y_sqr[4] < bullet_radius * bullet_radius)
                begin
52              vga_data <= 12'hca9;
53          end
54          else if(bullet2_x_sqr[5] + bullet2_y_sqr[0] < bullet_radius * bullet_radius)
                begin
55              vga_data <= 12'hca9;
56          end
57          else if(bullet2_x_sqr[6] + bullet2_y_sqr[1] < bullet_radius * bullet_radius)
                begin
58              vga_data <= 12'hca9;
59          end
60          else if(bullet2_x_sqr[7] + bullet2_y_sqr[2] < bullet_radius * bullet_radius)
                begin
61              vga_data <= 12'hca9;
62          end
63          else if(row_addr >= UD_distance_tank1 && row_addr <= UD_distance_tank1 + 23
                && col_addr >= LR_distance_tank1 && col_addr <= LR_distance_tank1 +
                23)begin
64              if(tka >= 12'b110011001100) begin
65                  vga_data <= orimap;
66              end
67              else begin
68                  vga_data <= tka;
69              end
70          end
71          else if(row_addr >= UD_distance_tank2 && row_addr <= UD_distance_tank2 + 23
                && col_addr >= LR_distance_tank2 && col_addr <= LR_distance_tank2 +
                23)begin
72              if(tkb >= 12'b110011001100) begin
73                  vga_data <= orimap;
74              end
75              else begin
76                  vga_data <= tkb;
77              end
78
79          end
```

```
80          else begin
81                if(save_map[row_addr / 24][col_addr / 24] == 4'b0000)begin //black
                      orimap
82                   vga_data <= orimap;
83                end
84                else if(save_map[row_addr / 24][col_addr / 24] == 4'b0010)begin //red
                      brick
85                   vga_data <= col_brick;
86                end
87                else if(save_map[row_addr / 24][col_addr / 24] == 4'b0011)begin
                      //blue box
88                   vga_data <= col_box;
89                end
90
91          end
```

使用矩阵方式存储地图，利用映射逻辑减小空间使用量。因为最初地图是 480*480 大小，对每个像素点单独判断可达性有太多冗余。所以我们将地图划分为了 20*20 大小的矩阵，并且采用了寄存器数组对地图进行储存。初始状态，无障碍物部分用 0000 填入，在地图显示可摧毁砖块、不可摧毁砖块、草地处分别用 0010、0011、0001 填入矩阵。VGA 显示扫描过程中，用当前像素的横纵坐标分别除以 24 即可以得到对应的寄存器位置，并且显示对应模块。

此矩阵也用于之后的阻塞判断、子弹状态判断等逻辑中，实时记录当前地图状态。若砖块被摧毁则对应值恢复为 0000。

考虑到整个游戏的逻辑及显示方式，先将外部区域置为黑色。地图部分显示的优先级为草地 > 子弹 > 坦克 > 底图 > 可摧毁砖块 > 不可摧毁砖块。

3.1.2 坦克显示及移动逻辑

Listing 2: 相关参数定义

```
1  reg [11:0] LR_distance_tank1,TMP_LR_distance_tank1;
2     reg [10:0] UD_distance_tank1,TMP_UD_distance_tank1;
3     reg [1:0] direction_tank1;
4     initial begin LR_distance_tank1 = 0; end
5     initial begin UD_distance_tank1 = 455; end
6  reg [11:0] LR_distance_tank2,TMP_LR_distance_tank2;
7     reg [10:0] UD_distance_tank2,TMP_UD_distance_tank2;
8     reg  [1:0] direction_tank2;
9     initial begin LR_distance_tank2 = 431; end
10    initial begin UD_distance_tank2 = 0; end
```

Listing 3: 坦克初始化

```
1  if(SW[15])begin
2            LR_distance_tank2 = 431;
3            UD_distance_tank2 = 0;
4        end
5  if(SW[15])begin
6            LR_distance_tank1 = 0;
7            UD_distance_tank1 = 455;
8
9        end
```

我们设计的坦克使 UD_distance 记录上下坐标，LR_distance 记录左右坐标，Direction 记录坦克方向。最开始将坦克分别初始在左下角及右上角，为了实现独立控制，我们把坦克相关的量全部采用了不同的寄存器保存数据。

Listing 4: 坦克移动逻辑

```
if(tank1_move[8] == 1'b0 && tank1_move[7:0] == 8'h1C && LR_distance_tank1 > 0) begin
                    TMP_LR_distance_tank1 = LR_distance_tank1 - 8'h01;
                    direction_tank1 = 2'b00;//left
                    if(TMP_LR_distance_tank1 > 0 && (save_map[(UD_distance_tank1 +
                        tanksize)/24][TMP_LR_distance_tank1/24] == 4'b0000 ||
                        save_map[(UD_distance_tank1 +
                        tanksize)/24][TMP_LR_distance_tank1/24] ==
                        4'b0001)&&(save_map[UD_distance_tank1/24][TMP_LR_distance_tank1/24]
                        == 4'b0000 ||
                        save_map[UD_distance_tank1/24][TMP_LR_distance_tank1/24] ==
                        4'b0001))begin
                         LR_distance_tank1 = TMP_LR_distance_tank1;
                    end
            end
            else if(tank1_move[8] == 1'b0 && tank1_move[7:0] == 8'h23 &&
                LR_distance_tank1 + tanksize < 479) begin
                    TMP_LR_distance_tank1 = LR_distance_tank1 + 8'h01;
                    direction_tank1 = 2'b01;//right
                    if(TMP_LR_distance_tank1 + tanksize <= 479 &&
                        (save_map[(UD_distance_tank1)/24][(TMP_LR_distance_tank1 +
                        tanksize)/24] == 4'b0000 ||
                        save_map[(UD_distance_tank1)/24][(TMP_LR_distance_tank1 +
                        tanksize)/24] == 4'b0001)&&(save_map[(UD_distance_tank1 +
                        tanksize)/24][(TMP_LR_distance_tank1 + tanksize)/24] == 4'b0000
                        || save_map[(UD_distance_tank1 +
                        tanksize)/24][(TMP_LR_distance_tank1 + tanksize)/24] ==
                        4'b0001))begin
                         LR_distance_tank1 = TMP_LR_distance_tank1;
                    end
            end
            else if(tank1_move[8] == 1'b0 && tank1_move[7:0] == 8'h1D &&
                UD_distance_tank1 > tanksize) begin
                      TMP_UD_distance_tank1 = UD_distance_tank1 - 8'h01;
                      direction_tank1 = 2'b10;//up
                      if(TMP_UD_distance_tank1 > 0 &&
                          (save_map[TMP_UD_distance_tank1/24][LR_distance_tank1/24] ==
                          4'b0000 ||
                          save_map[TMP_UD_distance_tank1/24][LR_distance_tank1/24] ==
                          4'b0001)&&(save_map[TMP_UD_distance_tank1/24][(LR_distance_tank1
                          + tanksize)/24] == 4'b0000 ||
                          save_map[TMP_UD_distance_tank1/24][(LR_distance_tank1 +
                          tanksize)/24] == 4'b0001))begin
                          UD_distance_tank1 = TMP_UD_distance_tank1;
                      end
            end
            else if(tank1_move[8] == 1'b0 && tank1_move[7:0] == 8'h1B &&
```

```
            UD_distance_tank1 + tanksize < 479) begin
23                  TMP_UD_distance_tank1 = UD_distance_tank1 + 8'h01;
24                  direction_tank1 = 2'b11;//down
25                  if(TMP_UD_distance_tank1 + tanksize < 479 &&
                        (save_map[(TMP_UD_distance_tank1 +
                        tanksize)/24][LR_distance_tank1/24] == 4'b0000 ||
                        save_map[(TMP_UD_distance_tank1 +
                        tanksize)/24][LR_distance_tank1/24] ==
                        4'b0001)&&(save_map[(TMP_UD_distance_tank1 +
                        tanksize)/24][(LR_distance_tank1 + tanksize)/24] == 4'b0000 ||
                        save_map[(TMP_UD_distance_tank1 +
                        tanksize)/24][(LR_distance_tank1 + tanksize)/24] ==
                        4'b0001))begin
26                      UD_distance_tank1 = TMP_UD_distance_tank1;
27                  end
28          end
```

整个游戏中采用玩家对战模式，所以设计中需要完成两架坦克的完整移动逻辑、显示逻辑。但彼此之间除了初始化及显示以外没有太大区别，故以下大部分内容以初始位置在左下角的坦克一为例。坦克的移动分为两方面，一方面是普通情况下的移动，包括位移和旋转。另一方面是坦克的移动阻塞。

普通情况下的移动仅需要判断键盘信号，所以我们在每一个触发沿对四个方向分别进行判断，首先判断是否按下对应的方向键，如果按下了相应的方向键，则在当前基础上改变相应坐标表示值及方向表示寄存器。

加上坦克的移动阻塞之后，出现了不可移动的情况。对不可消除的砖块、可消除的砖块、边界，坦克都只会发生方向改变而不会发生位移，对草坪则可以发生方向和位移的变化。所以我们引入了一个 TMP 系列的寄存器，用于记录位移改变后的量，然后对这个量进行边界判断。但是由于我们的坦克只记录了一个左上角的点坐标，在可移动性判断时，就需要使其变成正对障碍物的那条边。我们采用了两个边界点作为判断，如果 `tmp_up`、`tmp_lr` 在边界阈值内且 `tmp_up` 和 `tmp_lr` 映射到阻塞地图上是非障碍物就可以移动，即把 `tmp` 的位移赋值给对应变量 Reg。此处判断可移动性也利用到了之前的地图矩阵。由于记录的是边界点，如果边界不满足条件则中间的其他点一定不满足条件。

在最初设计的时候发现对尺寸较大的坦克而言，由于坦克比障碍物大，会出现坦克能够压着障碍物通过的情况。后来我们把坦克的大小缩小到和障碍物一样就解决了这个问题。

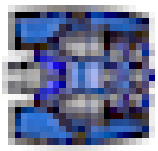
Listing 5: 坦克显示逻辑

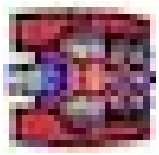
```
1  if(direction_tank1 == 2'b00) begin
2           if(row_addr >= UD_distance_tank1 && row_addr <= UD_distance_tank1 + tanksize
                && col_addr >= LR_distance_tank1 && col_addr <= LR_distance_tank1 +
                tanksize) begin
3               atkaddr = (row_addr - UD_distance_tank1) * 24 + col_addr -
                    LR_distance_tank1;
4           end
5       end
6       else if(direction_tank1 == 2'b01) begin
7           if(row_addr >= UD_distance_tank1 && row_addr <= UD_distance_tank1 + tanksize
                && col_addr >= LR_distance_tank1 && col_addr <= LR_distance_tank1 +
                tanksize) begin
8               atkaddr = (row_addr - UD_distance_tank1) * 24 + tanksize - col_addr +
                    LR_distance_tank1;
```

```
            end
        end
        else if(direction_tank1 == 2'b10) begin
            if(row_addr >= UD_distance_tank1 && row_addr <= UD_distance_tank1 + tanksize
                && col_addr >= LR_distance_tank1 && col_addr <= LR_distance_tank1 +
                tanksize) begin
                atkaddr = (row_addr - UD_distance_tank1) + (col_addr -
                    LR_distance_tank1)*24;
            end
        end
        else if(direction_tank1 == 2'b11) begin
            if(row_addr >= UD_distance_tank1 && row_addr <= UD_distance_tank1 + tanksize
                && col_addr >= LR_distance_tank1 && col_addr <= LR_distance_tank1 +
                tanksize) begin
                atkaddr = (tanksize - row_addr + UD_distance_tank1) + (tanksize -
                    col_addr + LR_distance_tank1)*24;
            end
        end

FTanka atank(
     .clka(clk),
     .wea(0),
     .addra(atkaddr),
     .dina(0),
     .douta(tka)
   );
```

对所有显示逻辑而言，主要需要实现的就是当前坐标下某点的位置和 COE 文件中地址的对应关系。我们导入的坦克图像初始方向是朝左的，记录了左上角点。如图所示：





地址的运算和方向及左上角点都相关，就是旋转坐标下的各点表示。具体见代码实现。

另外，因为坦克是带白色底图背景的，为了让他透明显示，我们设置了透明逻辑，即如果当前的色值高于某个阈值（接近白色）就不显示。

Listing 6: 坦克透明显示

```
if(row_addr >= UD_distance_tank1 && row_addr <= UD_distance_tank1 + 23 && col_addr >=
    LR_distance_tank1 && col_addr <= LR_distance_tank1 + 23)begin
                if(tka >= 12'b110011001100) begin
                    vga_data <= orimap;
                end
                else begin
                    vga_data <= tka;
                end
            end
```

3.1.3 子弹逻辑

Listing 7: 相关参数定义

```
localparam bullet_radius = 4; // 子弹大小
localparam bullet_num = 8; // 子弹数量
localparam bullet_speed = 4'd12; //子弹速度
reg [11:0] bullet1_x [bullet_num - 1:0],bullet2_x [bullet_num - 1:0]; // 子弹x坐标
reg [10:0] bullet1_y [bullet_num - 1:0],bullet2_y [bullet_num - 1:0]; // 子弹y坐标
reg [2:0] bullet1_direction [bullet_num- 1:0],bullet2_direction [bullet_num-1:0];
    // 子弹运动方向
reg [2:0] bullet1_count,bullet2_count; // 子弹计数器
wire [19:0] bullet1_x_sqr[bullet_num - 1:0],bullet2_x_sqr[bullet_num - 1:0]; //
    判断VGA显示
wire [19:0] bullet1_y_sqr[bullet_num - 1:0],bullet2_y_sqr[bullet_num - 1:0];
```

Listing 8: 子弹初始化

```
always @(posedge clkdiv[22] or posedge SW[15]) begin
   if(SW[15])begin
      for(i=0; i<bullet_num; i=i+1) begin
         // 初始化默认位置
         bullet1_x[i] = 500;
         bullet1_y[i] = 500;
         // 初始化默认方向，最高位为1表示静止，其余两位与坦克方向的表示相同
         bullet1_direction[i] = 3'b100;
         bullet1_count = 0;
         // 坦克2的子弹初始化同理，不再赘述
         // ......
      end
      // 逻辑地图
      save_map[0][1] = 4'b0010;
      save_map[0][3] = 4'b0010;
      // ......
      // 中间部分省略
      save_map[17][16] = 4'b0011;
      save_map[17][17] = 4'b0011;

      lock = 0;
      winner = 2'b11;
      gameover = 0;
   end
   else begin
      // 以下为子弹逻辑
      // ......
   end
end
```

我们设计的子弹使用 xy 坐标记录位置，并且有固定的速度和方向，在每个时钟上升沿更新子弹的信息。因为硬件编程最后生成的是逻辑电路，与软件编程的思路完全不同。在软件编程中，我们可以将子弹封装成一个对象，在需要新子弹的时候 new 一个新的对象；在硬件编程中这就完全行不通。所以我们想到固定子弹的数量 `bullet_num`，然后在需要的时候直接循环重复使用这些子弹。

Listing 9: 子弹逻辑

```
always @(posedge clkdiv[22] or posedge SW[15]) begin
    if(SW[15])begin
        // 子弹初始化，见上
    end
    else begin
        // 从PS2模块获取到坦克1射击的按键触发
        if(tank1_attack[8] == 1'b0 && tank1_attack[7:0] == 8'h29) begin
            // 获取子弹的坐标，因为要显示在坦克炮管上，所以需要对坦克坐标进行变换
            if(direction_tank1 == 2'b00)begin
                bullet1_x[bullet1_count] = LR_distance_tank1;
                bullet1_y[bullet1_count] = UD_distance_tank1 + 11'd12;
                waittodel = {bullet1_count, 1'b0, bullet1_y[bullet1_count]};
            end
            else if(direction_tank1 == 2'b01)begin
                bullet1_x[bullet1_count] = LR_distance_tank1 + 12'd23;
                bullet1_y[bullet1_count] = UD_distance_tank1 + 11'd12;
            end
            else if(direction_tank1 == 2'b10)begin
                bullet1_x[bullet1_count] = LR_distance_tank1 + 12'd12;
                bullet1_y[bullet1_count] = UD_distance_tank1;
            end
            else if(direction_tank1 == 2'b11)begin
                bullet1_x[bullet1_count] = LR_distance_tank1 + 12'd12;
                bullet1_y[bullet1_count] = UD_distance_tank1 + 11'd23;
            end
            // 更新子弹运动方向
            bullet1_direction[bullet1_count] = {1'b0, direction_tank1};
            // 更新子弹计数器
            bullet1_count = bullet1_count + 4'b0001;
        end

        // 每个时钟都更新所有子弹的位置
        for(i=0; i<bullet_num; i=i+1) begin
            if(bullet1_direction[i]==3'b000) begin
                bullet1_x[i] = bullet1_x[i] - bullet_speed;
            end
            else if(bullet1_direction[i]==3'b001) begin
                bullet1_x[i] = bullet1_x[i] + bullet_speed;
            end
            else if(bullet1_direction[i]==3'b010) begin
                bullet1_y[i] = bullet1_y[i] - bullet_speed;
            end
            else if(bullet1_direction[i]==3'b011) begin
                bullet1_y[i] = bullet1_y[i] + bullet_speed;
            end

            // 检测到子弹超出边界时重置子弹
            if(bullet1_y[i] <= 0 || bullet1_y[i] >= 479 || bullet1_x[i] <= 0 ||
                bullet1_x[i] >= 479)begin
                bullet1_x[i] = 500;
                bullet1_y[i] = 500;
                bullet1_direction[i] = 3'b100;
```

```
            end
            // 检测到子弹与可摧毁砖块碰撞时，重置子弹、摧毁砖块
            else if(save_map[bullet1_y[i]/24][bullet1_x[i]/24] == 4'b0010)begin
                save_map[bullet1_y[i]/24][bullet1_x[i]/24] = 4'b0;
                bullet1_x[i] = 500;
                bullet1_y[i] = 500;
                bullet1_direction[i] = 3'b100;
            end
            // 检测到子弹与不可摧毁砖块碰撞时，重置子弹
            else if(save_map[bullet1_y[i]/24][bullet1_x[i]/24] == 4'b0011)begin
                bullet1_x[i] = 500;
                bullet1_y[i] = 500;
                bullet1_direction[i] = 3'b100;
            end
            //
                检测到子弹与对方坦克碰撞时，重置子弹、锁定游戏、设置胜利玩家、进入gameover流程
            else if(bullet1_x[i] >= LR_distance_tank2 && bullet1_x[i] <=
                LR_distance_tank2 +23 && bullet1_y[i] >= UD_distance_tank2 &&
                bullet1_y[i] <= UD_distance_tank2 + 23) begin
                bullet1_x[i] = 500;
                bullet1_y[i] = 500;
                bullet1_direction[i] = 3'b100;
                lock = 1;
                winner = 2'b0;
                gameover = 1;
            end
        end
    end
end
```

如果在上升沿从 PS2 键盘获得坦克 1 发射子弹的指令，则复用子弹 `bullet1_*[bullet1_count]`，将新的坐标、运动方向赋予该子弹，然后更新子弹计数器 `bullet1_count`。子弹计数器 `bullet1_count` 在溢出时会回到 0，形成一个循环。

如果没有获得发射子弹的指令，就根据每个子弹的运动方向更新坐标，接着判断子弹与其他物体是否碰撞。碰撞检测的逻辑也较为简单：我们使用 `save_map` 保存游戏中的物体信息 00 表示无物体、01 表示为草地、10 表示为可摧毁的砖块、11 表示为不可摧毁的砖块。当子弹超出边界或者与砖块碰撞的时候，重置子弹的信息。

Listing 10: 子弹显示

```
genvar j;
generate
    for(j=0; j<bullet_num; j=j+1) begin
        assign bullet1_x_sqr[j] = (bullet1_x[j] - col_addr) * (bullet1_x[j] -
            col_addr);
        assign bullet1_y_sqr[j] = (bullet1_y[j] - row_addr) * (bullet1_y[j] -
            row_addr);
        assign bullet2_x_sqr[j] = (bullet2_x[j] - col_addr) * (bullet2_x[j] -
            col_addr);
        assign bullet2_y_sqr[j] = (bullet2_y[j] - row_addr) * (bullet2_y[j] -
            row_addr);
    end
endgenerate
```

```
10
11    always @(posedge clk or posedge gameover) begin
12        if(gameover == 1'b1) begin
13            // 胜利时显示胜利界面
14            // ......
15        end
16        else begin
17            if(row_addr >= 480 || col_addr >= 480)begin
18                vga_data <= 0;
19            end
20            // 草地显示逻辑
21            else if(save_map[row_addr / 24][col_addr / 24] == 4'b0001)begin
22                vga_data <= col_grass;
23            end
24            // 坦克1子弹1的显示逻辑
25            else if(bullet1_x_sqr[0] + bullet1_y_sqr[0] < bullet_radius * bullet_radius)
                  begin
26                vga_data <= 12'h347;
27            end
28            // 坦克1其他子弹的显示逻辑同理
29            // ......
30            else if(bullet1_x_sqr[7] + bullet1_y_sqr[7] < bullet_radius * bullet_radius)
                  begin
31                vga_data <= 12'h347;
32            end
33            else if(bullet2_x_sqr[0] + bullet2_y_sqr[0] < bullet_radius * bullet_radius)
                  begin
34                vga_data <= 12'hca9;
35            end
36            // ......
37            else if(bullet2_x_sqr[7] + bullet2_y_sqr[2] < bullet_radius * bullet_radius)
                  begin
38                vga_data <= 12'hca9;
39            end
40            // 以下为其他部件的显示逻辑
41            // ......
42        end
43    end
```

因为子弹的形状是圆形的，所以我们定义了 bullet1,2_x,y_sqr 来计算扫描的像素点与子弹间的距离的平方，如果 bullet1,2_x_sqr+bullet1,2_y_sqr<bullet_radius*bullet_radius，则说明扫描到的像素点在子弹显示半径之内，我们就给这个像素点不同的颜色。还有一点需要注意的就是子弹经过草地的显示。因为我们规定其他可移动物体经过草地时，都会被草地遮盖，所以草地的显示优先级是最高的。因此我们在判断像素点的颜色时，最先判定这个像素点是不是属于草地，如果不是，再判断子弹的情况。

3.1.4 获胜逻辑

Listing 11: 获胜逻辑

```
1  //以tank2获胜为例
2  if(bullet2_x[i] >= LR_distance_tank1 && bullet2_x[i] <= LR_distance_tank1 + 23 &&
       bullet2_y[i] >= UD_distance_tank1 && bullet2_y[i] <= UD_distance_tank1 + 23) begin
```

```
    bullet2_x[i] = 500;
    bullet2_y[i] = 500;
    bullet2_direction[i] = 3'b100;
    lock = 1;
    gameover = 1;
    winner = 2'b1;
end
```

以子弹中心作为判定点，敌方坦克所在正方形区域作为判定区间。子弹中心进入坦克敌方坦克所处范围时，判定发射子弹方获胜。此时回收子弹并锁定键盘，同时显示获胜界面。

3.1.5 复位逻辑

对不同的模块我们都采用时钟信号触发，仅有地址计算模块我们采用了异步触发。这是为了避免扫描信号发生改变时还没有来得及计算新的地址并且改变色值显示。所以在时钟触发信号后面增加一个 RST，然后加一层 IF-ELSE 判断，如果 rst 信号为真，则执行清空操作。否则按照正常逻辑执行。

3.1.6 IP 核

导入 COE 文件生成 IP 核按照正常步骤即可。但是具体的 COE 文件却并不太容易被生成，我们认真查阅了大量资料，最后将所需图片转化成 BMP 保存之后，将 24 位真彩图的 BMP 文件信息按 R-G-B 三个通道压缩真彩值，得到了二进制下各个像素点 12 位的色值（即十六进制下 3 位）。这样的 COE 文件才能和我们的程序显示模块一致。

Listing 12: 获胜逻辑

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;
char trash[2000];
int main()
{
    freopen("wintwo.coe","r",stdin);
    freopen("winsec.coe","w",stdout);
    gets(trash);
    puts(trash);
    //puts("\n");
    gets(trash);
    puts(trash);
    //puts("\n");
    gets(trash);
    puts(trash);
    //puts("\n");
    gets(trash);
    puts(trash);
    //puts("\n");
    gets(trash);
    puts(trash);
    //puts("\n");
    int x = 0;
    int N = 24*24;//
```

```
    while(x < N)
    {
        x ++;
        char c = getchar();
        int a = 0;
        for(;!((c >= 'a' && c <= 'f')||(c >= '0' && c <= '9'));c = getchar());
        if(c >= 'a' && c <= 'f') a = a * 16 + c - 'a' + 10;
        if(c >= '0' && c <= '9') a = a * 16 + c - '0';
        c = getchar();
        for(;!((c >= 'a' && c <= 'f')||(c >= '0' && c <= '9'));c = getchar());
        if(c >= 'a' && c <= 'f') a = a * 16 + c - 'a' + 10;
        if(c >= '0' && c <= '9') a = a * 16 + c - '0';
        a = (double)a / 255.0 * 15.0 + 0.5;
        if(a > 15) a = 15;
        if(a >= 0 && a <= 9) printf("%d",a);
        else printf("%c",a + 'a' - 10);

        c = getchar();
        a = 0;
        for(;!((c >= 'a' && c <= 'f')||(c >= '0' && c <= '9'));c = getchar());
        if(c >= 'a' && c <= 'f') a = a * 16 + c - 'a' + 10;
        if(c >= '0' && c <= '9') a = a * 16 + c - '0';
        c = getchar();
        for(;!((c >= 'a' && c <= 'f')||(c >= '0' && c <= '9'));c = getchar());
        if(c >= 'a' && c <= 'f') a = a * 16 + c - 'a' + 10;
        if(c >= '0' && c <= '9') a = a * 16 + c - '0';
        a = (double)a / 255.0 * 15.0 + 0.5;
        if(a > 15) a = 15;
        if(a >= 0 && a <= 9) printf("%d",a);
        else printf("%c",a + 'a' - 10);

        c = getchar();
        a = 0;
        for(;!((c >= 'a' && c <= 'f')||(c >= '0' && c <= '9'));c = getchar());
        if(c >= 'a' && c <= 'f') a = a * 16 + c - 'a' + 10;
        if(c >= '0' && c <= '9') a = a * 16 + c - '0';
        c = getchar();
        for(;!((c >= 'a' && c <= 'f')||(c >= '0' && c <= '9'));c = getchar());
        if(c >= 'a' && c <= 'f') a = a * 16 + c - 'a' + 10;
        if(c >= '0' && c <= '9') a = a * 16 + c - '0';
        a = (double)a / 255.0 * 15.0 + 0.5;
        if(a > 15) a = 15;
        if(a >= 0 && a <= 9) printf("%d",a);
        else printf("%c",a + 'a' - 10);
        printf("\n");
    }
    puts(";");
    return 0;
}
```

IP 核的调用只用 View HDL 相关模块即可。因为我们只需要读入，所以计算出地址值，得到色值即可。

3.1.7 PS2 模块

4 设计结果

4.1 设计过程中所遇问题及解决方案

4.1.1 VGA 显示

最初使用多层透明逻辑，对可打破砖块、不可打破砖块、草坪、背景图分别导入了 480*480 的 COE 文件。又另外使用一个 COE 文件读取当前像素点对应位置的色值，红色显示砖块、蓝色显示不可移动砖块、绿色显示草坪、黑色显示底图。但最后发现这样实现会导致资源占用太多。所以修改为了前文所述的矩阵映射显示方式。只需要导入单位素材即可。

但在游戏结束界面的时候又遇到了不一样的问题。因为需要能正常阅读，所以我们导入了 24*24 像素的显示图，然后将其放大，采用最近邻插值。最终得到了正确的显示界面。

4.1.2 PS2 多按键输入

在 PS2 模块实现过程中，我们遇到了很多问题。实现过程时，我们用按下键盘坦克的移动来检测模块的准确性。开始时，我们发现按下键盘后坦克迅速从画面一端跳到了另一端，调试后发现是时钟信号给得太快的问题，于是，我们延长了读取一次键码的时间间隔。

上述问题解决后，我们遇到的第二个问题是，键盘按下后坦克一直保持移动，只有在另一个按下时，坦克才停止移动或者改变方向。调试后我们发现我们只考虑了通码，忽略了断码。我们在程序中增加了通码的判断，只有通码，才会引起坦克的移动。

新的问题是，键盘的按键相互干扰，一次只会发送一个键码。然而，在我们的设计中，两个玩家同时操作并且涉及到方向键和攻击键，所以在写 PS2 模块时，需要考虑多按键输入的问题。最终我们给出的解决方案是，在 PS2 模块 output 中增加了 4 个 reg，分别记录坦克一的移动，坦克一的攻击，坦克二的移动、坦克二的攻击。如果 ps2 的信号对应能上述四个 reg 之一对应的键码时，对其进行更新，否则不改变。一次我们达到了多按键输入的效果。

4.2 改进与展望

坦克虽然能够支持连发子弹，但每辆坦克各自最多八发。如果能够进一步优化设计或许有希望能增加子弹数量。

在效果方面，没有给坦克增加爆炸效果，因为资源占用已经很紧张，进一步增加 IP 核不知道是否会超限，所以没有进行最后的尝试。