摘要

基于数字系统的设计原理，利用 SWORD 主板，开发一个小游戏——坦克大战（Tank War)。在 Xilinx ISE12.4 开发环境下，使用 Verilog HDL 硬件描述性语言，设计游戏逻辑的时序电路，利用 VGA 显示和 PS2 键盘控制，实现丰富的人机互动。

关键词：SWORD，Verilog HDL， 坦克大战，时序电路

目录

图目录

第1章 绪论

1.1 ……设计背景

在童年的时候，我和我的小伙伴们都特别喜欢玩像素游戏掌机，里面预置了好几个游戏，包括俄罗斯方块、贪吃蛇、坦克大战、走迷宫等等。在当时，能拥有一个这样的游戏机是十分幸福的一件事情。现在，我就想能不能复现一次游戏机里面的游戏出来。俄罗斯方块、贪吃蛇可以说是家喻户晓的小游戏了，很多人也都写过类似的游戏，因此我把目标放在了坦克大战上。

坦克大战的游戏内容是玩家控制一个坦克，可以朝上下左右四个方向前进和射击，游戏内会随机生成敌方的坦克，他们会不断的射击，玩家被敌方坦克击中则生命-1 或者游戏结束。这是一个充满互动性和趣味性的游戏，如果在 SWORD 主板上实现可以利用 VGA 显示和 PS2 键盘控制，贴合 project 的设计要求。和贪吃蛇和俄罗斯方块相比，坦克大战的实现需要同时控制多个目标（包括坦克和子弹），在设计上有一定的挑战性。

在这次的设计中，我的目标是完成基本坦克大战的游戏内容：玩家能自由移动、发射子弹，随机生成敌方坦克，敌方坦克会随机移动和射击，当被玩家的子弹击中后会消失，而玩家被击中后游戏结束。同时我会增加一个计分板的功能，利用七段数码管显示。

1.2 国内外现况分析

通过 Google 搜索和在 GitHub 上查找，发现几乎没有利用 Verilog HDL 编写的相同的游戏，虽然有同名的游戏，但实现的内容和我的设想不同：

网络上的坦克大战：

1. 游戏模式为双人模式，两个玩家同时控制
2. 只实现了两辆坦克的移动

3. 每个坦克只能同时发射一颗子弹
4. 增加了地形障碍物

我的设计目标：

1. 单人游戏模式
2. 存在多辆敌方坦克
3. 每辆坦克都可以发射多颗子弹
4. 无地形障碍，有一个长方形边界

可见，两者之间虽然类似但是从玩法和设计思路上都存在着较大的不同。反观贪吃蛇和俄罗斯方块，通过 Google 搜索可以得到大量由 Verilog HDL 编写的设计，在 GitHub 上甚至可以直接获取源代码。与贪吃蛇和俄罗斯方块相比，坦克大战的设计还是有很大实现价值的。

1.3 主要内容和难点

主要游戏功能：

1. 玩家可以通过键盘的 WASD 键操控坦克在上下左右四个方向自由移动
2. 玩家按下 J 键后会朝着坦克前进方向发射一颗子弹
3. 每隔一段时间随机生成一辆敌方坦克
4. 敌方坦克会自动发射子弹，并且能随机移动
5. 当玩家的坦克接触到敌人的子弹时游戏结束
6. 敌方坦克碰到玩家发射出的子弹后会消失
7. 玩家每击毁一个敌人加一定的分数，分数用七段数码管显示
8. 长按 RSTN 键后游戏重新开始

技术要求：

1. 能够在 640X480 的 VGA 显示器上正确显示游戏
2. 能通过 PS2 键盘控制，有长按和短按功能
3. 利用 IP 核 RAM 存储所需要用到的图片
4. 能通过七段数码管显示玩家所得分数
5. 实现正确的时序电路，完成游戏运行逻辑

实现重点、难点：

1. 多目标的追踪和控制，需要同时记录多辆坦克和多个子弹的位置和运动方向
2. 坦克与子弹碰撞的检测，在时序电路内实现多个坦克位置与多个子弹位置的比较
3. 敌我子弹的辨别，只有在被对方的子弹击中后才阵亡
4. 像素图片的加载和显示，正确计算出 VGA 扫描的地址，传入 RAM 中获得 RGB 值

第2章 ……设计原理

2.1 ……设计相关内容

本次设计主要应用时序电路的设计原理，在时序电路内实现坦克和子弹的移动、生成和销毁，检测坦克和子弹的碰撞；利用 VGA 显示的原理，呈现游戏的画面，同时通过 PS2 键盘控制坦克的移动和射击；在 IP Core RAM 里存储所需要用到的图片。

2.2 ……设计方案

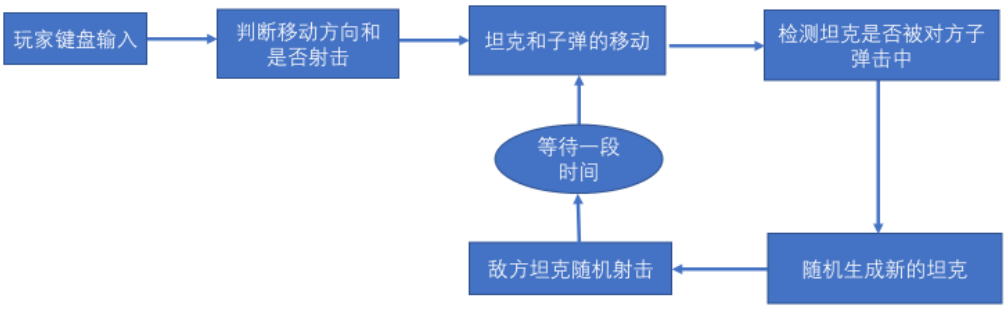


图 1 设计方案流程图

游戏中使用寄存器数组记录坦克和子弹的位置（x，y）和前进方向，同时记录下当前坦克的数量和子弹的数量。之后根据用户键盘的输入对坦克和子弹的位置进行移动，如果有射击动作发生则子弹数量+1，有坦克被击中则坦克数量-1，随机生成坦克时坦克数量+1。坦克和子弹每次的移动有时间间隔，坦克的时间间隔比子弹大一些。这样不断地更新坦克和子弹的信息，输出时判断当前扫描到的像素点是位于空地、坦克还是子弹上，给出相对的 Address，传入 RAM，得到相应图片的 RGB 值。

2.3 ……硬件设计

本次设计除了用到 SWORD 开发板，还利用了 VGA 显示器，PS2 键盘。

VGA（Video Graphics Array）是一个电脑显示标准，采用逐行扫描的方式进行显示。VGA 接口有两个扫描信号，一个是行同步信号，一个是场同步信号。行同步信号控制一行像素点的扫描，场同步信号控制一帧画面的扫描。本次设计 VGA 接口输入的 RGB

信号为 12 位，R、G、B 各四位，可显示丰富的色彩。

以下是 VGA 模块的代码：

```
module VGA_CTRL (
       input wire clk, // 25mhz
       input wire RSTN,

       output wire hsync, // horizontal synchronization signal
       output wire vsync, // vertical synchronization signal

       output wire[9:0] pixel_x, // horizontal pos
       output wire[9:0] pixel_y  // vertical pos
       );


   reg[9:0] x, y;

   initial begin
       x <= 10'd0;
       y <= 10'd0;
   end

   always @(posedge clk or posedge RSTN) begin
       if (RSTN)
           x <= 10'd0;
       else begin
           if (x == 10'd799) begin
               x <= 10'd0;
           end
           else
               x <= x + 10'd1;
       end
   end

   always @(posedge clk or posedge RSTN) begin
       if (RSTN)
           y <= 10'd0;
       else begin
           if (x == 10'd799)
               if (y == 10'd524)
                   y <= 10'd0;
               else
                   y <= y + 10'd1;
```

```
        end
    end

    assign hsync = (x > 10'd95);
    assign vsync = (y > 10'd1);

    assign pixel_x = x - 10'd142;
    assign pixel_y = y - 10'd35;

endmodule
```

PS/2 接口是一种 PC 兼容型的接口，可以用来连接键盘和鼠标。它使用一个时钟频率 clk 和一个数据输出口 data 来向外输出信息。在使用键盘时，每次输出一个 11 位的信息，其中 8 位是数据位，当有按键按下时 8 位数据为按下按键的通码，当按键松开时，一个断码和一个通码会产生，由此得出键盘的输入。

以下是 PS/2 键盘模块的代码：

```
module ps2_keyboard (
    input clk,
    input clrn,
    input ps2_clk,
    input ps2_data,
    input rdn,
    output [7:0] data,
    output [7:0] prev_data,
    output ready, overflow
    );

reg overflow;       // fifo overflow
reg [3:0] count;        // count ps2_data bits, internal signal, for
test
reg [9:0] buffer;       // ps2_data bits
reg [7:0] fifo[7:0];    // data fifo
reg [2:0] w_ptr, r_ptr; // fifo write and read pointers
reg [2:0] ps2_clk_sync;

always @ (posedge clk) begin
    ps2_clk_sync <= {ps2_clk_sync[1:0],ps2_clk};
end
```

```
wire sampling = ps2_clk_sync[2] & ~ps2_clk_sync[1];

always @ (posedge clk) begin
    if (clrn == 0) begin
        count <= 0;
        w_ptr <= 0;
        r_ptr <= 0;
        overflow <= 0;
    end
    else if (sampling) begin
        if (count == 4'd10) begin
            if (buffer[0] == 0 && ps2_data) begin
                fifo[w_ptr] <= buffer[8:1];  // kbd scan code
                w_ptr <= w_ptr + 3'd1;
                overflow <= overflow |  (r_ptr == (w_ptr + 3'd1));
            end
            count <= 0; // for next
        end
        else begin
            buffer[count] <= ps2_data;   // store ps2_data
            count <= count + 3'd1;   // count ps2_data bits
        end
    end
    if (!rdn && ready) begin
        r_ptr <= r_ptr + 3'd1;
        overflow <= 0;
    end
end

assign ready = (w_ptr != r_ptr);
assign data = fifo[r_ptr];
assign prev_data = fifo[r_ptr - 1'b1];

endmodule
```

第3章 ……设计实现

3.1 实现方法

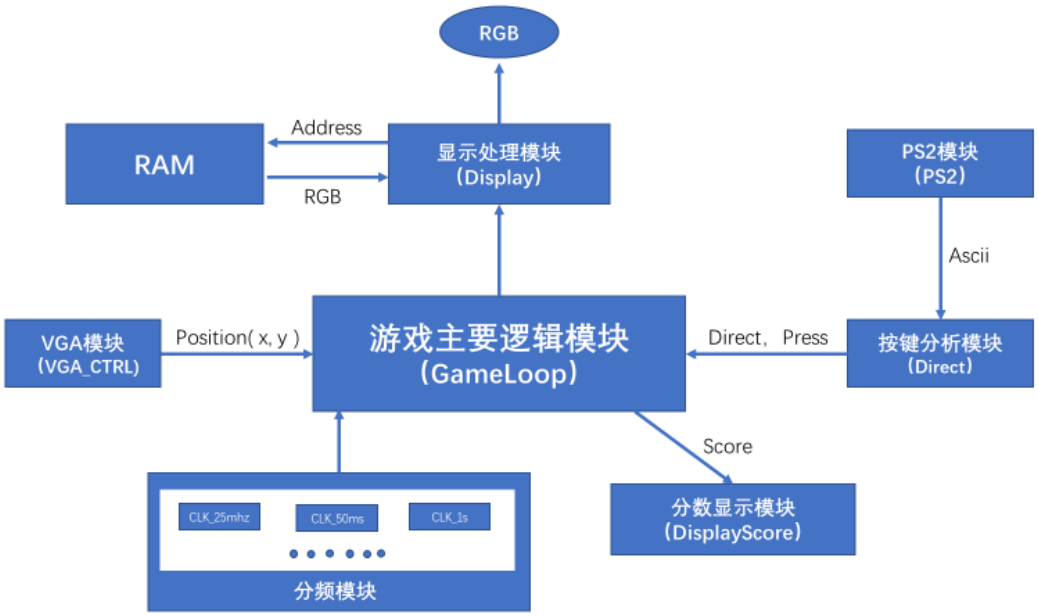


图 2 设计模块图

游戏的实现主要分为 8 个模块：

1．游戏主要逻辑模块（GameLoop）是游戏逻辑模块，负责坦克、子弹的移动、生成和销毁，记录坦克和子弹的位置和方向，运算之后把相应的信息传给显示处理模块进行显示

2．分频模块（CLK）负责为游戏逻辑模块提供不同频率的时钟信号，包括坦克移动频率、子弹移动频率、随机生成坦克的频率和发射子弹的频率等等

3．显示处理模块（Display）从 GameLoop 模块中接受信息，根据像素点的显示类别和相对位置选择合适的 IPCore RAM，传入计算得出的地址，得到 RGB 色值并输出。

4．ROM 模块存储所有需要用到的图片 RGB 值，包括坦克图片、启动界面和游戏结束界面等。

5．分数显示模块（DisplayScore）从游戏逻辑模块中获取玩家所得的分数，在七段数码管上采用十进制显示。

6．PS2 模块负责读取键盘的输入，获取由 PS/2 接口传入的 8 位数据值，经过分析后转化为对应的 ASCII 码，并给出一个 press 信号，当按钮按下是为 1，否则为 0

7. 按键分析模块（Direct）获得 ASCII 码后进行分析，判断玩家是否有移动或射击，把玩家的动作信息传给游戏逻辑模块。

8. VGA 模块（VGA_CTRL）生成行扫描信号和场扫描信号，传出当前扫描像素点的坐标

3.2 实现过程

由于游戏比较复杂，可能会出现很多问题，一次性写完如果出现 BUG 就难以调试，因此我是采用逐步设计的方法。

首先编写的是 VGA 模块，能在显示器上显示稳定的彩条之后，就开始设计 PS2 模块了。PS2 模块相较于 VGA 模块耗时要长许多，设计难度比 VGA 要大不少，但在仔细研究与多次尝试之后还是完成了。

完成了 VGA 和 PS2 模块的设计，基本上设计的框架也就有了。接下来，我先写了一个控制一个大方块的程序，并且能够发射一个小方块，这是坦克和子弹的原型。接下来我尝试实现多颗子弹的连发以及多个坦克的控制，这一部分是游戏设计的过程中最难的一部分，花的时间也是最久的。在完成了这一部分之后，整个游戏已经完成了一大半了，剩下的就是随机生成坦克、子弹，以及游戏效果的一些美化了。

在前期坦克的显示我是利用一个 30X30 的方块表示，后来我利用 IPCore 的 RAM，把坦克四个方向的图片存入进去，这样就能显示更加美观的坦克。之后我设置了游戏启动画面和游戏结束画面，并在游戏启动画面增加了一些基本操作的说明，让游戏更加的容易上手。

在最后，我编写了一个七段数码管显示模块，用来显示玩家的分数，给予玩家一些反馈。

设计出来代码的 RTL 逻辑图如下：

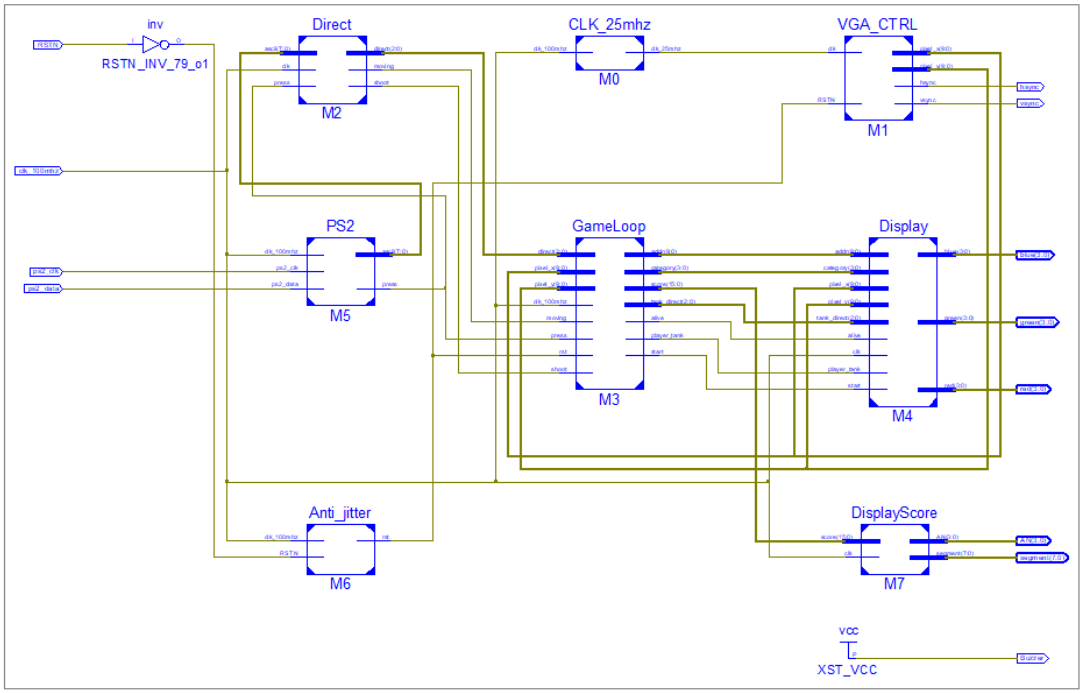


图 3 RTL 逻辑图

其中时钟分频模块是需要仿真测试的，保证时钟的频率符合要求，特别是 VGA 的时钟信号。

部分核心代码：

顶层模块 TankWar.v:

```
////////////////////////////////////////////////
//
//  Module: TankWar
//  Author: CheckDeng
//  Description: Top module of Tank War
//
```

```
//////////////////////////////////////////////

module TankWar (
    input wire clk_100mhz,
    // input wire shoot,
    input wire RSTN,

    input wire ps2_clk,
    input wire ps2_data,

    output wire Buzzer,

    output wire hsync,
    output wire vsync,
    output wire[3:0] red,
    output wire[3:0] green,
    output wire[3:0] blue,

    output wire[7:0] segment,
    output wire[3:0] AN
    );

wire clk_25mhz;
wire[9:0] pixel_x, pixel_y;
wire[3:0] category;
wire[2:0] moving_direct;
wire moving;
wire[7:0] ascii;
wire press;
wire rst;
wire shoot;
wire[9:0] addr;
wire[2:0] tank_direct;

wire[31:0] score;
wire player_tank;
wire start;

assign Buzzer = 1'b1;



CLK_25mhz M0 (
    .clk_100mhz(clk_100mhz),
    .clk_25mhz(clk_25mhz)
```

```
    );

VGA_CTRL M1 (
    .clk(clk_25mhz),
    .rst(rst),

    .hsync(hsync),
    .vsync(vsync),
    .pixel_x(pixel_x),
    .pixel_y(pixel_y)
    );

// modify
Direct M2 (
    .clk(clk_100mhz),
    .ascii(ascii),
    .press(press),
    .direct(moving_direct),
    .moving(moving),
    .shoot(shoot)
    );


GameLoop M3 (
    .clk_100mhz(clk_100mhz),
    .direct(moving_direct),
    .moving(moving),
    .pixel_x(pixel_x),
    .pixel_y(pixel_y),
    .shoot(shoot),
    .rst(rst),
    .press(press),
    .category(category),
    .addr(addr),
    .tank_direct(tank_direct),
    .alive(alive),
    .start(start),
    .score(score),
    .player_tank(player_tank)
    );

Display M4 (
    .clk(clk_100mhz),
    .category(category),
```

```
    .addr(addr),
    .tank_direct(tank_direct),
    .pixel_x(pixel_x),
    .pixel_y(pixel_y),
    .player_tank(player_tank),
    .start(start),
    .red(red),
    .green(green),
    .blue(blue),
    .alive(alive)
    );

PS2 M5 (
    .clk_100mhz(clk_100mhz),
    .ps2_data(ps2_data),
    .ps2_clk(ps2_clk),
    .ascii(ascii),
    .press(press)
    );

Anti_jitter M6 (
    .clk_100mhz(clk_100mhz),
    .RSTN(1'b1),
    .rst(rst)
    );

DisplayScore M7 (
    .clk(clk_100mhz),
    .score(score),
    .segment(segment),
    .AN(AN)
    );

endmodule
```

子弹的生成和移动：

```
// bullets
always @(posedge clk_100mhz) begin

if (start) begin //---------game start----------

if (rst) begin
```

```
    n_bullets <= 7'd0;
    bullets_ptr <= 7'd0;
    bullets_go <= 1'b0;
    tanks_cnt <= 6'd1;
end
else begin
    // shoot
    if (shoot_edge && n_bullets < MAX_BULLETS) begin
        case (tanks_direct[0])
        LEFT: begin
            if (tanks_x[0] > 10'd0) begin
                n_bullets <= n_bullets + 7'd1;
                bullets_x[n_bullets] <= tanks_x[0] - 10'd1;
                bullets_y[n_bullets] <= tanks_y[0] + 10'd1;
                bullets_direct[n_bullets] <= tanks_direct[0];
                is_player[n_bullets] <= 1'b1;
                // map[tanks_x[0] - 10'd1 + (tanks_y[0] + 10'd1) * WIDTH] <= 1'b1;  // map
            end
        end
        RIGHT: begin
            if (tanks_x[0] < WIDTH - 1) begin
                n_bullets <= n_bullets + 7'd1;
                bullets_x[n_bullets] <= tanks_x[0] + 10'd3;
                bullets_y[n_bullets] <= tanks_y[0] + 10'd1;
                bullets_direct[n_bullets] <= tanks_direct[0];
                is_player[n_bullets] <= 1'b1;
                // map[tanks_x[0] + 10'd3 + (tanks_y[0] + 10'd1) * WIDTH] <= 1'b1; // map
            end
        end
        UP: begin
            if (tanks_y[0] > 10'd0) begin
                n_bullets <= n_bullets + 7'd1;
                bullets_x[n_bullets] <= tanks_x[0] + 10'd1;
                bullets_y[n_bullets] <= tanks_y[0] - 10'd1;
                bullets_direct[n_bullets] <= tanks_direct[0];
                is_player[n_bullets] <= 1'b1;
                // map[tanks_x[0] + 10'd1 + (tanks_y[0] - 10'd1) * WIDTH] <= 1'b1; // map
            end
        end
        DOWN: begin
            if (tanks_y[0] < HEIGHT - 1) begin
                n_bullets <= n_bullets + 7'd1;
                bullets_x[n_bullets] <= tanks_x[0] + 10'd1;
                bullets_y[n_bullets] <= tanks_y[0] + 10'd3;
```

```
                bullets_direct[n_bullets] <= tanks_direct[0];
                is_player[n_bullets] <= 1'b1;
                // map[tanks_x[0] + 10'd1 + (tanks_y[0] + 10'd3) * WIDTH] <= 1'b1;  // map
            end
        end
        endcase
    end
    // bullets move
    else if (clkbullet_edge | bullets_go) begin
        if (clkbullet_edge & ~bullets_go) begin
            bullets_go <= 1'b1;
        end

        if (bullets_ptr < n_bullets) begin
            // delete bullets
            if (bullets_x[bullets_ptr] < 0 | bullets_x[bullets_ptr] > WIDTH |
bullets_y[bullets_ptr] < 0 | bullets_y[bullets_ptr] > HEIGHT) begin
                n_bullets <= n_bullets - 7'd1;
                if (n_bullets != 7'd0) begin
                    bullets_x[bullets_ptr] <= bullets_x[n_bullets - 7'd1];
                    bullets_y[bullets_ptr] <= bullets_y[n_bullets - 7'd1];
                    bullets_direct[bullets_ptr] <= bullets_direct[n_bullets - 7'd1];
                    is_player[bullets_ptr] <= is_player[n_bullets - 7'd1];
                    // map[bullets_x[bullets_ptr] + bullets_y[bullets_ptr] * WIDTH] <= 1'b0;
// map
                end
            end
            else begin
                // map[bullets_x[bullets_ptr] + bullets_y[bullets_ptr] * WIDTH] <= 1'b0; //
map
                case (bullets_direct[bullets_ptr])
                    LEFT: begin
                        bullets_x[bullets_ptr] <= bullets_x[bullets_ptr] - 10'd1;
                        // map[bullets_x[bullets_ptr] - 1 + bullets_y[bullets_ptr] * WIDTH]
<= 1'b1; // map
                    end
                    RIGHT: begin
                        bullets_x[bullets_ptr] <= bullets_x[bullets_ptr] + 10'd1;
                        // map[bullets_x[bullets_ptr] + 1 + bullets_y[bullets_ptr] * WIDTH]
<= 1'b1; // map
                    end
                    UP: begin
                        bullets_y[bullets_ptr] <= bullets_y[bullets_ptr] - 10'd1;
                        // map[bullets_x[bullets_ptr] + (bullets_y[bullets_ptr] - 1) * WIDTH]
```

```
<= 1'b1; // map
                    end
                    DOWN: begin
                        bullets_y[bullets_ptr] <= bullets_y[bullets_ptr] + 10'd1;
                        // map[bullets_x[bullets_ptr] + (bullets_y[bullets_ptr] + 1) * WIDTH]
<= 1'b1; // map
                    end
                endcase
                bullets_ptr <= bullets_ptr + 7'd1;
            end
        end
        else begin
            bullets_ptr <= 7'd0;
            bullets_go <= 1'b0;
        end
    end
    else if (clk250_edge && n_bullets < MAX_BULLETS) begin
        if (tanks_cnt < n_tanks && tanks_cnt != 6'd0) begin
            case (tanks_direct[tanks_cnt])
            LEFT: begin
                if (tanks_x[tanks_cnt] > 10'd0) begin
                    n_bullets <= n_bullets + 7'd1;
                    bullets_x[n_bullets] <= tanks_x[tanks_cnt] - 10'd1;
                    bullets_y[n_bullets] <= tanks_y[tanks_cnt] + 10'd1;
                    bullets_direct[n_bullets] <= LEFT;
                    is_player[n_bullets] <= 1'b0;
                    // map[tanks_x[tanks_cnt] - 1 + (tanks_y[tanks_cnt] + 1) * WIDTH] <= 1'b1;
// map
                end
            end
            RIGHT: begin
                if (tanks_x[tanks_cnt] < WIDTH - 1) begin
                    n_bullets <= n_bullets + 7'd1;
                    bullets_x[n_bullets] <= tanks_x[tanks_cnt] + 10'd3;
                    bullets_y[n_bullets] <= tanks_y[tanks_cnt] + 10'd1;
                    bullets_direct[n_bullets] <= RIGHT;
                    is_player[n_bullets] <= 1'b0;
                    // map[tanks_x[tanks_cnt] + 3 + (tanks_y[tanks_cnt] + 1) * WIDTH] <= 1'b1;
// map
                end
            end
            UP: begin
                if (tanks_y[tanks_cnt] > 10'd0) begin
                    n_bullets <= n_bullets + 7'd1;
```

```
                    bullets_x[n_bullets] <= tanks_x[tanks_cnt] + 10'd1;
                    bullets_y[n_bullets] <= tanks_y[tanks_cnt] - 10'd1;
                    bullets_direct[n_bullets] <= UP;
                    is_player[n_bullets] <= 1'b0;
                    // map[tanks_x[tanks_cnt] + 1 + (tanks_y[tanks_cnt] - 1) * WIDTH] <= 1'b1;
// map
                end
            end
            DOWN: begin
                if (tanks_y[tanks_cnt] < HEIGHT - 1) begin
                    n_bullets <= n_bullets + 7'd1;
                    bullets_x[n_bullets] <= tanks_x[tanks_cnt] + 10'd1;
                    bullets_y[n_bullets] <= tanks_y[tanks_cnt] + 10'd3;
                    bullets_direct[n_bullets] <= DOWN;
                    is_player[n_bullets] <= 1'b0;
                    // map[tanks_x[tanks_cnt] + 1 + (tanks_y[tanks_cnt] + 3) * WIDTH] <= 1'b1;
// map
                end
            end
            endcase
            tanks_cnt <= tanks_cnt + 6'd1;
        end
        else begin
            tanks_cnt <= 6'd1;
        end
    end
end

end //---------game start----------

end
```

坦克的生成和移动

```

// tanks
always @(posedge clk_100mhz) begin

if (start) begin //---------game start----------

if (rst) begin
    tanks_x[0] <= 10'd0;
    tanks_y[0] <= 10'd0;
```

```
    tanks_direct[0] <= RIGHT;
    n_tanks <= 6'd1;

    tanks_ptr <= 6'd1;
    tanks_go <= 1'b0;
    score <= 16'b0;
end
else begin
    // generate tanks
    if (clk5s_edge && (n_tanks < MAX_TANKS)) begin
        n_tanks <= n_tanks + 6'd1;
        tanks_x[n_tanks] <= {5'b00, random_pos};
        tanks_y[n_tanks] <= {5'b00, random_pos};
        tanks_direct[n_tanks] <= {1'b0, random_direct};
        random_direct <= random_direct + 2'b01;
        // tanks_direct[n_tanks] <= RIGHT;
    end


    // player's tank move
    if (moving_edge | clk500_edge) begin // player move
        if (moving) begin
            if (tanks_x[0] >= 10'd0 && tanks_x[0] < WIDTH - 10'd2 && tanks_y[0] >= 0 &&
tanks_y[0] < HEIGHT - 10'd2) begin
                case (direct)
                LEFT:
                    if (tanks_x[0] > 10'd0) begin
                        tanks_x[0] <= tanks_x[0] - 10'd1;
                    end
                RIGHT:
                    if (tanks_x[0] < WIDTH - 10'd3) begin
                        tanks_x[0] <= tanks_x[0] + 10'd1;
                    end
                UP:
                    if (tanks_y[0] > 10'd0) begin
                        tanks_y[0] <= tanks_y[0] - 10'd1;
                    end
                DOWN:
                    if (tanks_y[0] < HEIGHT - 10'd3) begin
                        tanks_y[0] <= tanks_y[0] + 10'd1;
                    end
                endcase
                tanks_direct[0] <= direct;
            end
```

```
        end
    end

    n = 0;
    // enemy tanks move
    if (clk250_edge | tanks_go) begin   // enemey move
        if (clk250_edge & ~tanks_go) begin
            tanks_go <= 1'b1;
        end

        if (tanks_ptr != n_tanks) begin
            for (m = 0; m < MAX_BULLETS; m = m + 1) begin
                if (m < n_bullets && is_player[m] == 1'b1) begin
                    if (bullets_x[m] >= tanks_x[tanks_ptr] && bullets_x[m] <
tanks_x[tanks_ptr] + 10'd3 &&
                        bullets_y[m] >= tanks_y[tanks_ptr] && bullets_y[m] <
tanks_y[tanks_ptr] + 10'd3) begin
                        n = 1;
                    end
                end
            end
            if (n == 1) begin
                n_tanks <= n_tanks - 6'b1;
                tanks_x[tanks_ptr] <= tanks_x[n_tanks - 6'd1];
                tanks_y[tanks_ptr] <= tanks_y[n_tanks - 6'd1];
                tanks_direct[tanks_ptr] <= tanks_direct[n_tanks - 6'd1];
                if (score[3:0] < 4'b1001)
                    score[3:0] <= score[3:0] + 4'b0001;  //-----------------------add
player's score
                else if (score[7:4] < 4'b1001) begin
                    score[7:4] <= score[7:4] + 4'b0001;
                    score[3:0] <= 4'b0000;
                end
                else if (score[11:8] < 4'b1001) begin
                    score[11:8] <= score[11:8] + 4'b0001;
                    score[7:4] <= 4'b0000;
                    score[3:0] <= 4'b0000;
                end
                else if (score[15:12] < 4'b1001) begin
                    score[15:12] <= score[15:12] + 4'b0001;
                    score[11:8] <= 4'b0000;
                    score[7:4] <= 4'b0000;
                    score[3:0] <= 4'b0000;
                end
```

```
            end
            else begin
                case (tanks_direct[tanks_ptr])
                LEFT:
                    if (tanks_x[tanks_ptr] > 10'd0) begin
                        tanks_x[tanks_ptr] <= tanks_x[tanks_ptr] - 10'd1;
                    end
                    else begin
                        tanks_direct[tanks_ptr] <= {1'b0, random_direct};
                        random_direct <= random_direct + 2'b01;
                    end
                RIGHT: begin
                    if (tanks_x[tanks_ptr] < WIDTH - 10'd3) begin
                        tanks_x[tanks_ptr] <= tanks_x[tanks_ptr] + 10'd1;
                    end
                    else begin
                        tanks_direct[tanks_ptr] <= {1'b0, random_direct};
                        random_direct <= random_direct + 2'b01;
                    end
                end
                UP:
                    if (tanks_y[tanks_ptr] > 10'd0) begin
                        tanks_y[tanks_ptr] <= tanks_y[tanks_ptr] - 10'd1;
                    end
                    else begin
                        tanks_direct[tanks_ptr] <= {1'b0, random_direct};
                        random_direct <= random_direct + 2'b01;
                    end
                DOWN:
                    if (tanks_y[tanks_ptr] < HEIGHT - 10'd3) begin
                        tanks_y[tanks_ptr] <= tanks_y[tanks_ptr] + 10'd1;
                    end
                    else begin
                        tanks_direct[tanks_ptr] <= {1'b0, random_direct};
                        random_direct <= random_direct + 2'b01;
                    end
                endcase
                tanks_ptr <= tanks_ptr + 6'd1;
            end
        end
        else begin
            tanks_ptr <= 6'd1;
            tanks_go <= 1'b0;
        end
```

```
    end

    if (clk5s_edge | tank_turn) begin
        if (clk5s_edge & ~tank_turn) begin
            tank_turn <= 1'b1;
        end
        if (tanks_p < n_tanks && tanks_p != 6'd0) begin
            tanks_direct[tanks_p] <= random_direct;
            random_direct <= random_direct + 2'b01;
            tanks_p <= tanks_p + 6'd1;
        end
        else begin
            tanks_p <= 6'd1;
            tank_turn <= 1'b0;
        end
    end
end
```

3.3 仿真与调试

CLK_25m.v 的仿真结果：

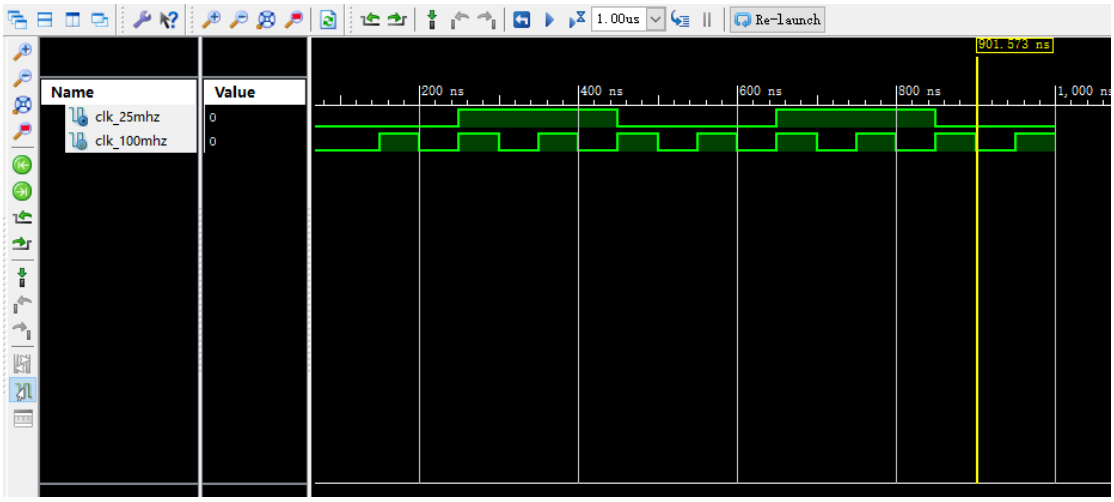


图 4 CLK_25mhz 仿真结果图

由图上可以看到，clk_25mhz 的周期是 clk_100mhz 的 4 倍，证明该模块是正常工作的，实现了分频的要求

CLK_500ms.v 的仿真结果：

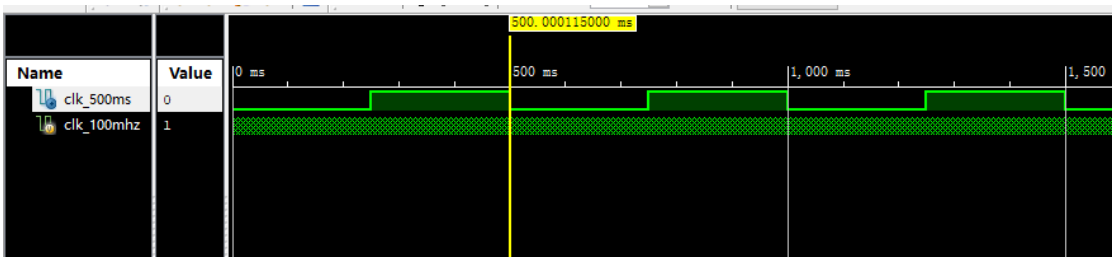


图 5 CLK_500ms 仿真结果图

由图中可以看到，clk_500ms 的周期为 500ms，符合设计预期

第4章 系统测试验证与结果分析

4.1 功能测试

进入游戏后，操控坦克战斗，并没有明显的 BUG，能正常展示游戏画面，坦克和子弹动作流畅，没有掉帧或者卡屏的情况。

4.2 技术参数测试

运行游戏后，检测游戏设计目标是否达到：

1. 能通过 WASD 键控制坦克运动方向 √
2. 支持长按和短按的移动方式 √
3. 能通过 J 键控制发射子弹 √
4. 每隔 5s 自动生成敌方坦克 √
5. 敌方坦克能随机发射子弹 √
6. 坦克碰到界面边缘后会自动调整方向 √
7. 玩家的子弹击中坦克后坦克消失 √
8. 玩家被敌方子弹击中后游戏结束 √
9. 有游戏启动和游戏结束界面 √

10. 计分板正常显示，玩家每击中一辆坦克加一分 √

4.3 结果分析

总的来说，验证的结果是不错的，完成了所有预期的设计目标，还增设了游戏启动和结束画面，可以说超额完成了目标。但仍存在着一些不足，坦克与坦克之间的体积碰撞没有实现，导致缺乏了一些实感，影响了游戏体验。

4.4 系统演示与操作说明

操作说明：

控制前进方向： W—上 S—下 A—左 D—右

射击： J

游戏重新开始： 长按 RSTN

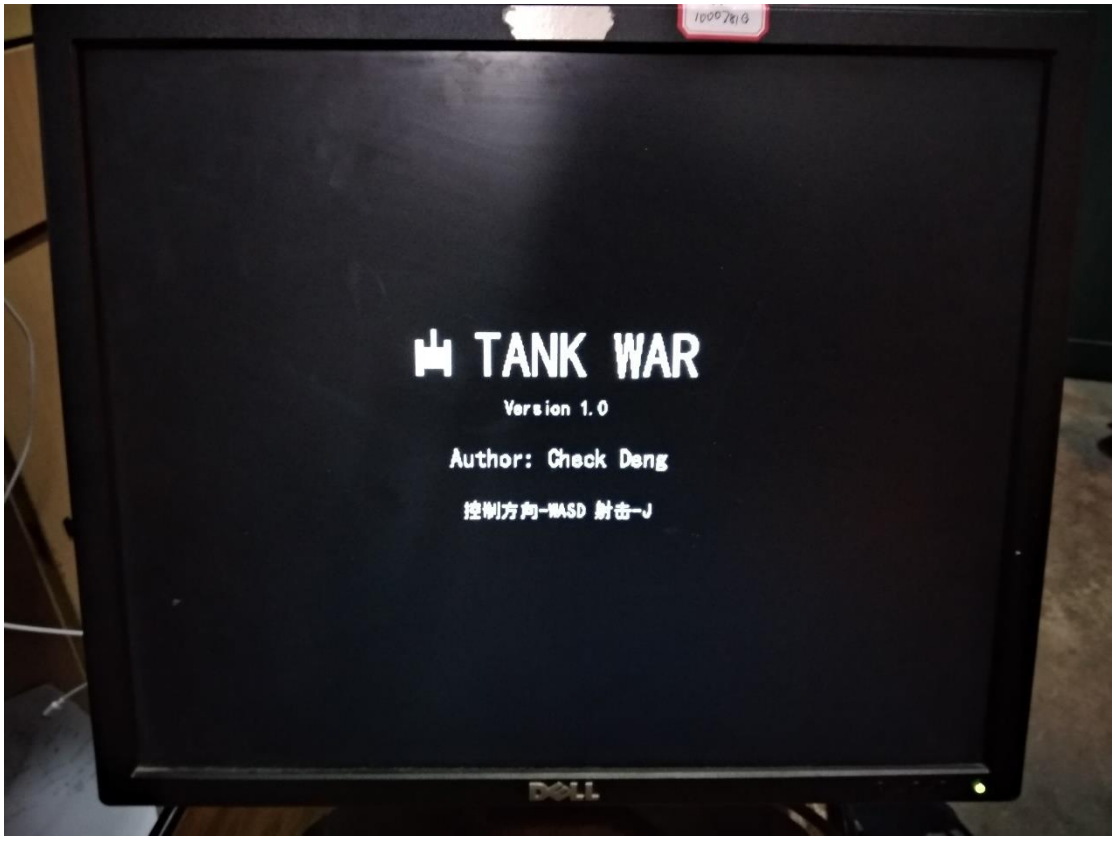


图 6 游戏开始画面

游戏开始画面，有简单的操作说明

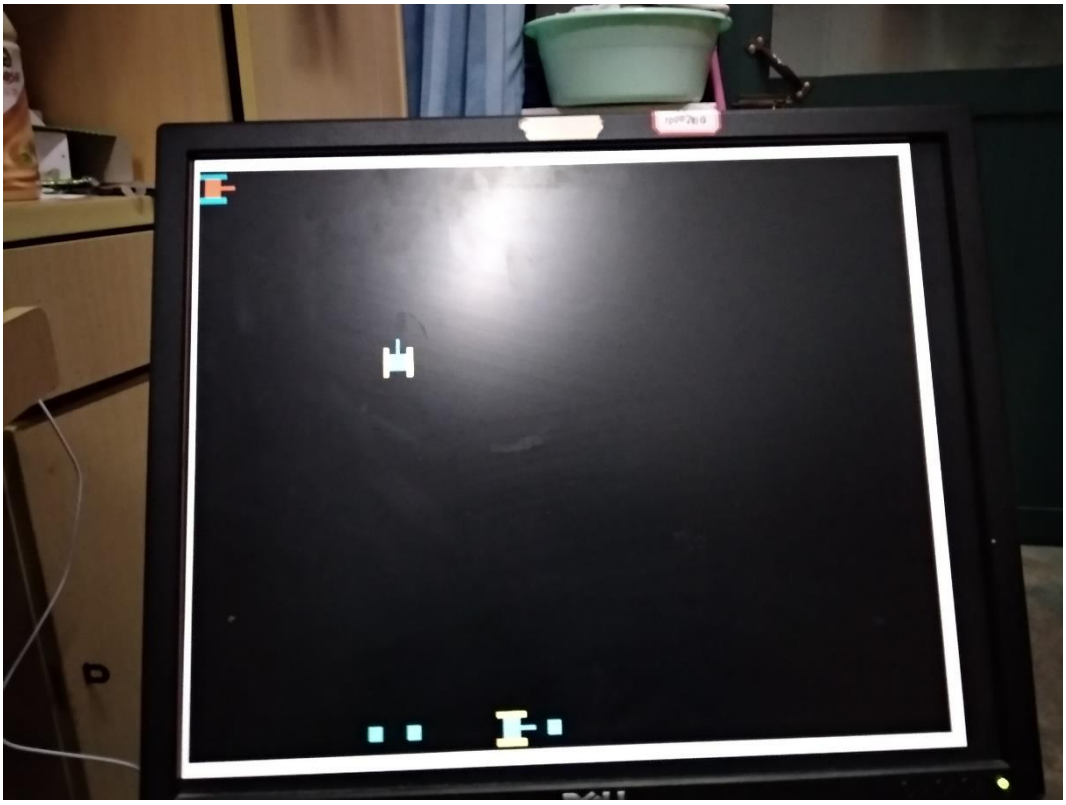


图 7 初始游戏状态

游戏开始时，玩家的坦克位于左上角，坦克颜色与其他坦克不同

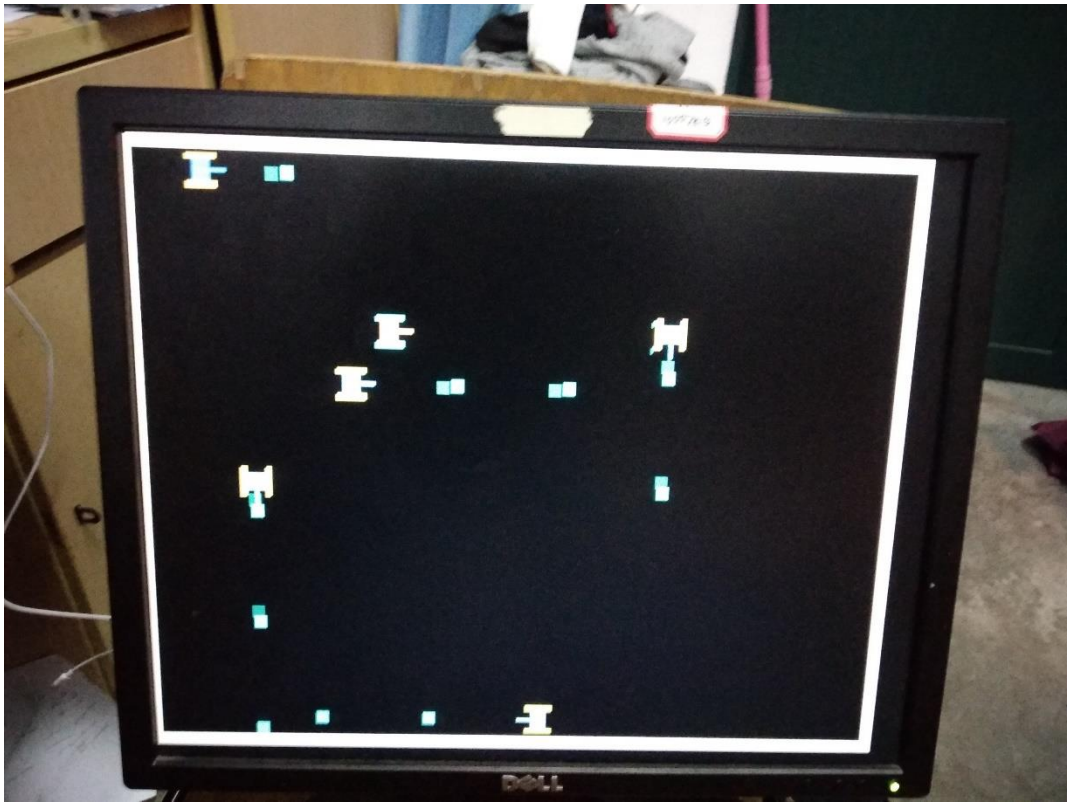


图 8 游戏进行一段时间后的画面

随着时间的推移，敌方坦克的数量会越来越多

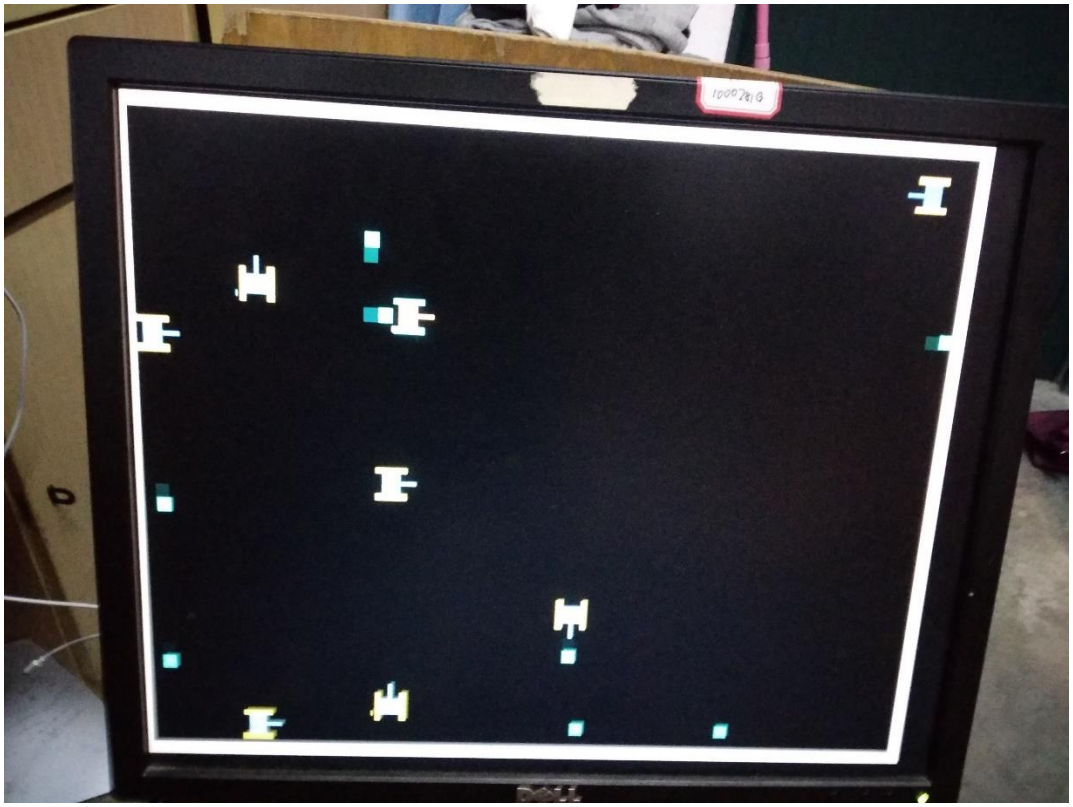


图 9 坦克被击中的画面

坦克被击中后会消失，玩家分数+1

图 10 游戏结束的画面

玩家被击中后游戏结束，图为游戏结束画面

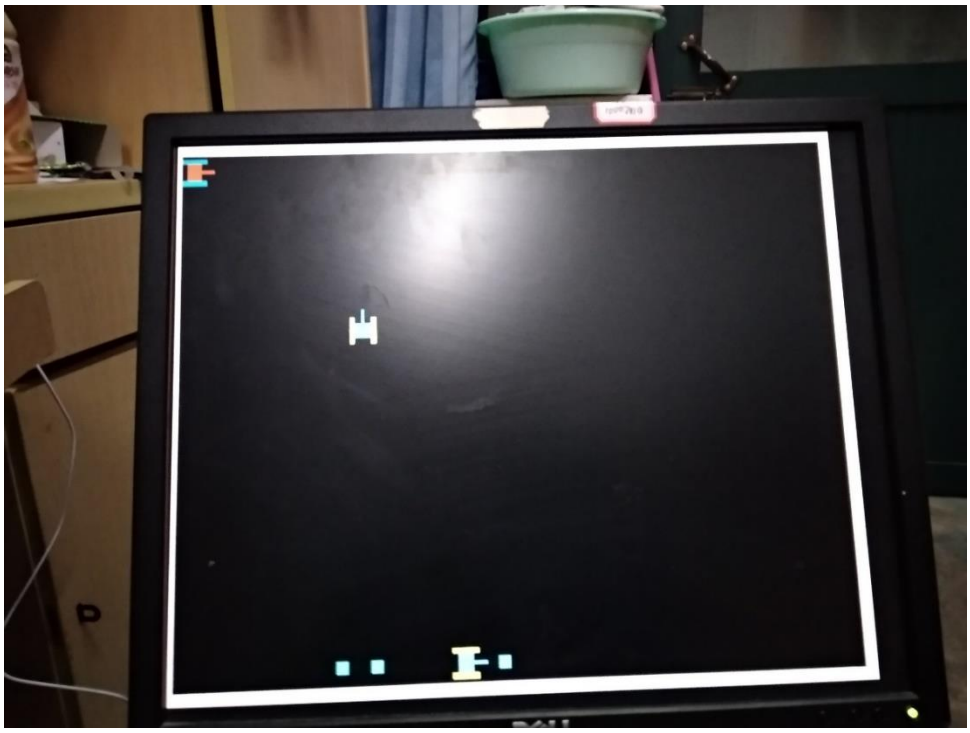
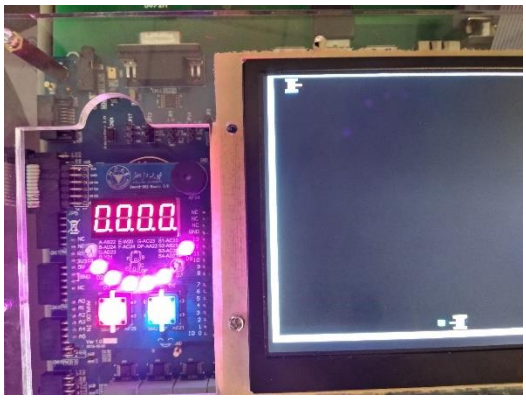


图 11 游戏重新开始画面

长按 RSTN 键后游戏重新开始



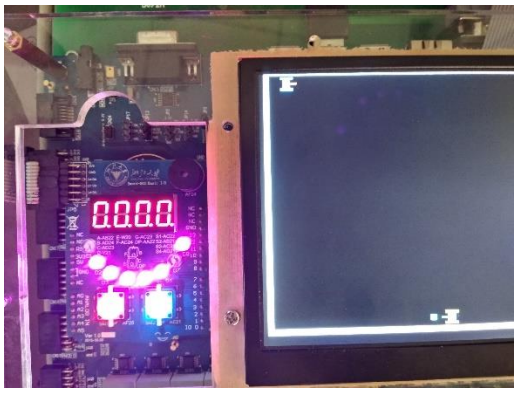


图 11 计分功能

计分功能

第5章 结论与展望

这次的 Project 设计有一定的难度，当初选这个题目的时候心里也是有些不安的，担心不能完成设计内容，可喜的是最后完成了全部的设计目标。因为这次的 project，我第一次

写代码而熬夜到 2 点，也是第一次全天呆在实验室。设计途中遇到过许多问题，许多是由于不熟悉 Verilog HDL 的语法以及时序电路的设计原理造成的，碰了许多壁，有些问题甚至导致要重写代码。例如，在 always 语句里使用 for 语句是不能循环变量的范围是不能有变量的，本来写好的 100 多行代码因此需要重写，代码的架构也要改变，还好发现问题发现得早。

在整个设计当中，最难的部分是如何控制多个目标，由于时序电路里使用 for 语句的限制比较大，而且较大的 for 语句在综合时会展开成很大的电路，因此我会尽量避免使用 for 语句。为了替代 for 语句，我使用一个循环变量，在每次 always 结束后加一，直到达到当前坦克或子弹数量后归零，这样不仅解决了操控多个目标的问题还节省了电路设计的负担。不过在一些特定的地方，还是不得不使用 for 循环来达到设计的目标。

通过这次 Project 的设计，我学到了非常多的东西，特别是对时序电路的有了更深的理解，懂得了 VGA 和 PS/2 接口的工作原理和 IP Core 的创建、使用。我觉得这次 Project 设计最锻炼人的自我学习能力，VGA、PS/2 和 IP Core 都是需要自学的，老师不会教大家怎么用，如何从网上查找到有用的资料并灵活运用，这是非常重要的一项能力。我非常喜欢逻辑与计算机基础这门课程，能锻炼学生的动手能力，我们能够从中学习到计算机基础的工作原理。相信修完了这门课程，对以后的学习是有很大帮助的。