Tank Game

Zixiang Zheng zz2642

Wenzhe He wh2443

Table of Content

I. Introduction

1. Project overview

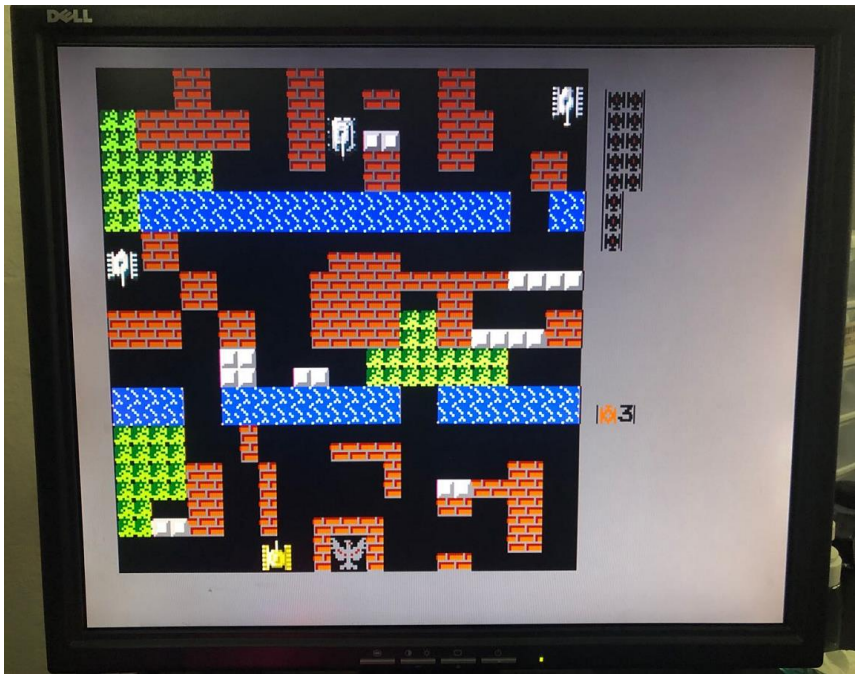


Fig.1 Game Overview

We design a reimplement a classic tank game ‘Battle City’. It was first introduced by Nintendo in year 1985, using the Nintendo joystick to play it. In this project, we bring the original taste of this outstanding game using modern technology and rebuilt it on DE1-SOC board. Player will be using a joystick to play this game.

2. Game Information

There is only one stage is this project, which is the original ‘stage 8’ of the original game. Player need to destroy all 16 enemy tanks to win the game. Player tank has 3 life and if all the lives are use or the based is attack by the enemy, player will lose the game.

II. System architecture

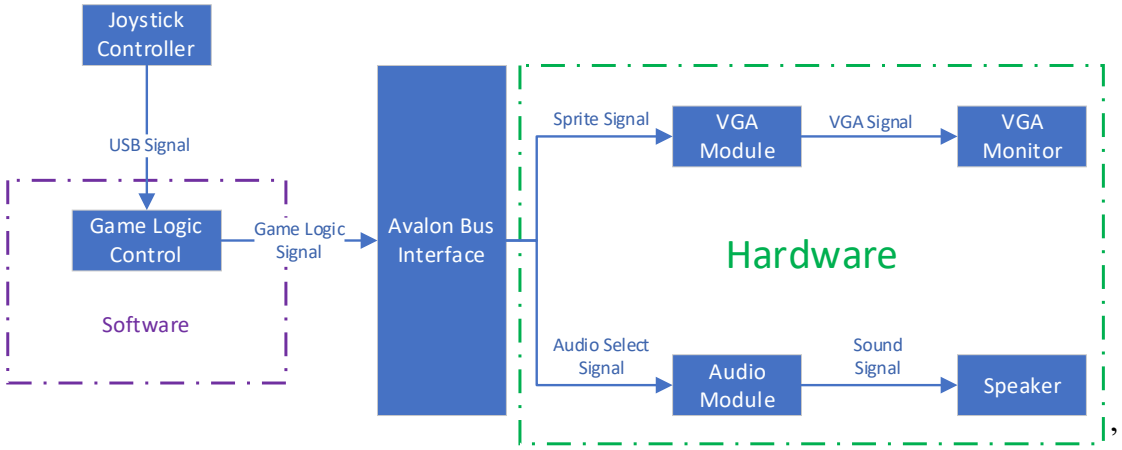


Fig. 2 System Architecture

The overall system module is shown in Fig.2. The system architecture is quite simple. First, the game is operate with a Joystick controller. The controller will send the control data from user to the software.

In software part, the software will calculate the game logic of the game. And send the game information to hardware through Avalon bus interface.

At each cycle, the software will send total 19 16bit data to hardware. The detail of the data will be discuss in software section.

Although all the hardware code is store in one System Verilog file, the function of the code can be split to 2 modules. In VGA module, the hardware will display map and tank based on On-chip RAM and data transmitted from software. In Audio display module, the hardware will produce sound based on the 'Sound' data transmitted from hardware. The detail of the design will be illustrated in the hardware section.

III. Hardware Design

1. Graphic Design

The Cyclone V SoC FPGA device on DE1-SoC board has a limited storage, so careful design should be taken into consideration.

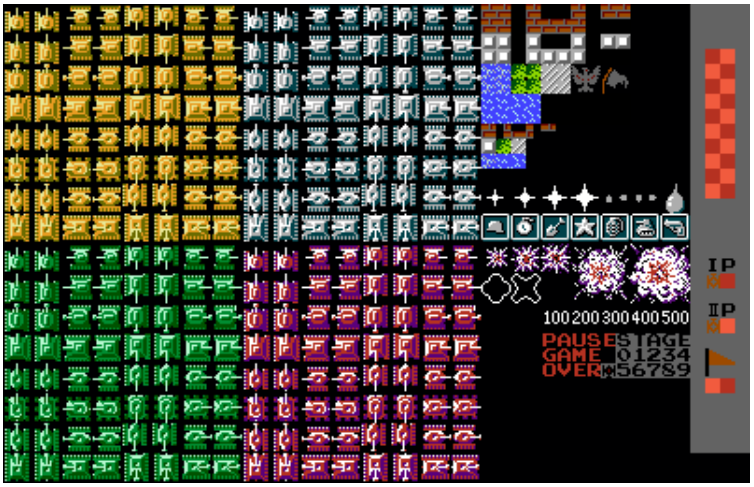


Fig. 3 General Sprite of Battle City tank game

Fig. 1 above shows all graphic needed to display in the original tank game, which determined by hardware. We will be only use some of them. Based on the display priority, the information can be divided in to three layers.

The first layer is the bullet, the second layer is the maps which cannot move but require update, the third layer is the tanks which can move around and fire the bullet. The detail of Layer is list in table 1.

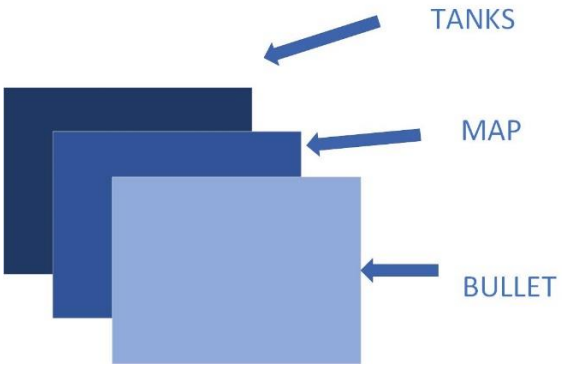


Fig. 4 Layer Design

Table 1 Layer of the game

Layer	Name	Description
1	Bullet	3*3 bullet
2	Background	background, wall, gadget, score
3	Tank	player's tank and enemy tank

The final performance of the project is shown in Fig.3. The battle field is located in the center left of the screen. The remain of enemy tank is shown on the top right of the screen. And the remain player life is shown on the middle right of the screen.

Fig. 5 Final Game Graph

To be specific, the original game display area is 224*256 and the battle field is 208*208, but this is too small on the VGA monitor, so all the size is double leading to a 448*512 area and a 416*416 battle field.

a. Color information

The total number of colors used in tank game is about 21. But it is important to notice that each tile contains no more than 4 colors. Which means that each pixel can be storage in 2 bits. Then use a color table to store the RBG information of the 21 different color.

b. Bullet Information

In order to simplify the whole process and save the memory, the bullet is a 3*3 square as in Fig. 6. The color is fix to 173 173 173 RGB.

Fig. 6 Bullet sprite

No colormap or other complex method is used, when the tank fire the bullet, hardware just assigns the bullet information.

c. Map Information

There are 5 different blocks in the game. We will at least finish first 5 of them. We named them our own. Each block 16*16 contains with four identical 8*8 basic block.

Table 2 Block Information

Image (8*8)	Name	Tank pass	Bullet pass	Note
	Blank	Yes	Yes	
	Brick	No	No	Destroyable
	Stone	No	No	
	Grass	Yes	Yes	Tank will display under it
	Water	No	Yes	

The whole map information is stored into on-chip RAM, to reduce the data transmitted from software at each cycle. For example, when a bullet hit a brick, the brick will be destroyed, by using RAM, only this brick information will be rewritten, instead of rewrite the whole map.

d. Tank information

As we mentioned, there are only limit tank used in the game. In this part we will list all of them. Each tank used two images to display, this is because when tank is moving, the track is not move. As shown in fig.7, the track of two tank is actually 1 pixel different, so

when tank is moving, they are displayed alternately.

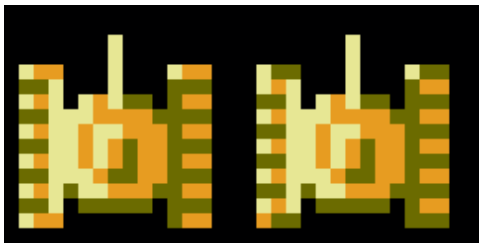


Fig. 7 Two Image of tank moving upward

Fig.8 shows all the tank sprite we used in the project. Notice that each tank only contains 3 colors, and the set of color is the same for same level of tank. So, we only need to store total 64 tanks and just need to change their color to obtain all 128 tanks.

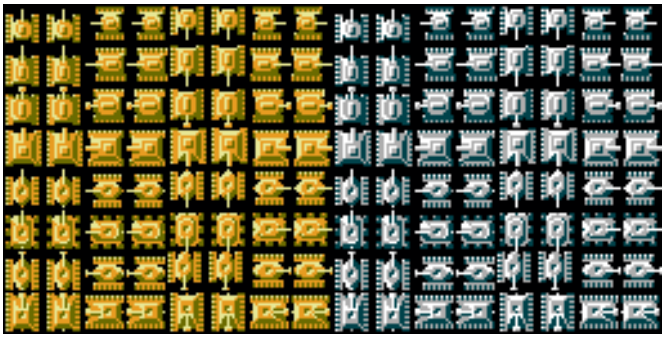


Fig. 8 All Tank Sprite

2. Hardware Block Diagram

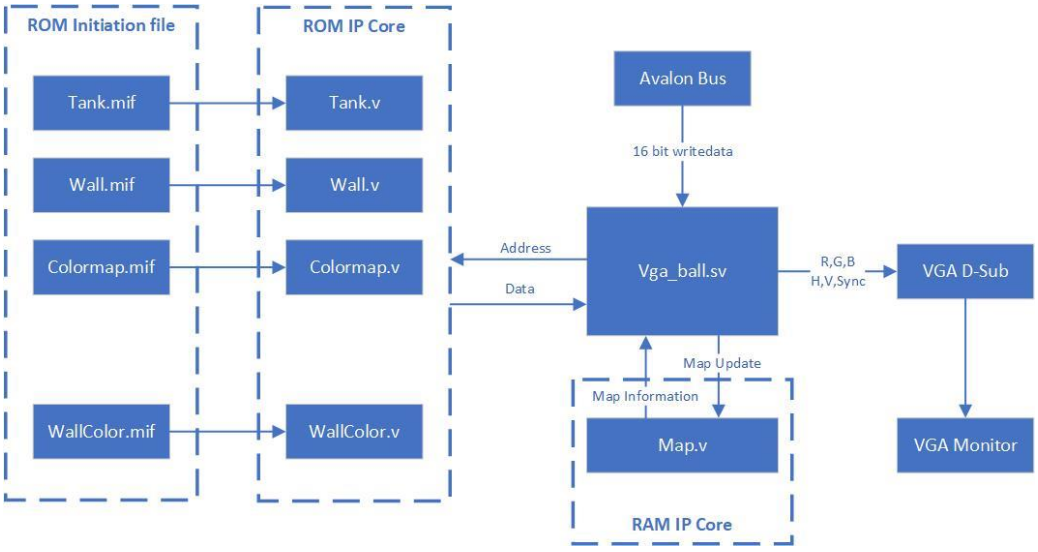


Fig.9 VGA Module Block Diagram

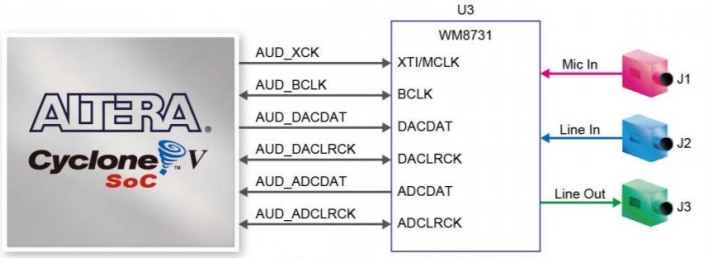
Fig.9 is the block diagram for VGA part. We use ROM IP Core to store tile pixel and color map. And we use RAM to store current Map wall information. At each cycle, software will only send updated wall information instead of send whole map information. This can be seen as an 'Differential Encoder'. Based on the changed address, map information can be easily modified in RAM.

The different map information is store in software. When the stage starts, the software will send the current entire stage map to the hardware. After the stage is running, it will only send data when some wall is change. By doing this we can greatly reduce the data send at each time through Avalon interface.

When display each pixel, first check the register to find out the name of tile or sprite on this pixel. Then check the ROM to find the pixel data, then check the ROM colormap to find the color of this pixel. Then send the color to VGA D-sub to display on the monitor. We will use the same technic in Lab3 to process the data. From the analysis of the game logic, we need four layers to display the graphics.

3. Audio Interface

On the DE1-SoC, there is a device called Wolfson WM8731 which has 2 24bits ADC, 2 24bits DAC and a headphone amplifier and support 8-96 kHz sampling rates. The WM8731 is controlled via serial I^2C bus, which is connected to HPS or Cyclone V SoC FPGA through an I2C multiplexer. The connection of the circuit is shown as following.



Pin Assignment of Audio CODEC

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
AUD_ADCLRCK	PIN_K8	Audio CODEC ADC LR Clock	3.3V
AUD_ADCDAT	PIN_K7	Audio CODEC ADC Data	3.3V
AUD_DACLRCK	PIN_H8	Audio CODEC DAC LR Clock	3.3V
AUD_DACDAT	PIN_J7	Audio CODEC DAC Data	3.3V
AUD_XCK	PIN_G7	Audio CODEC Chip Clock	3.3V
AUD_BCLK	PIN_H7	Audio CODEC Bit-stream Clock	3.3V
I2C_SCLK	PIN_J12 or PIN_E23	I2C Clock	3.3V
I2C_SDAT	PIN_K12 or PIN_C24	I2C Data	3.3V

Fig.10 Block Diagram Connection of circuit

As for our case, we assume that a sample rate of 8kHz is adequate. All the configuration process is accomplished with the help of a built-in IP core, so the things we need to do is to generator mif file and write a FIFO program to output the audio information to the IP-Core, which process can be found in the references. The whole structure is shown as following.

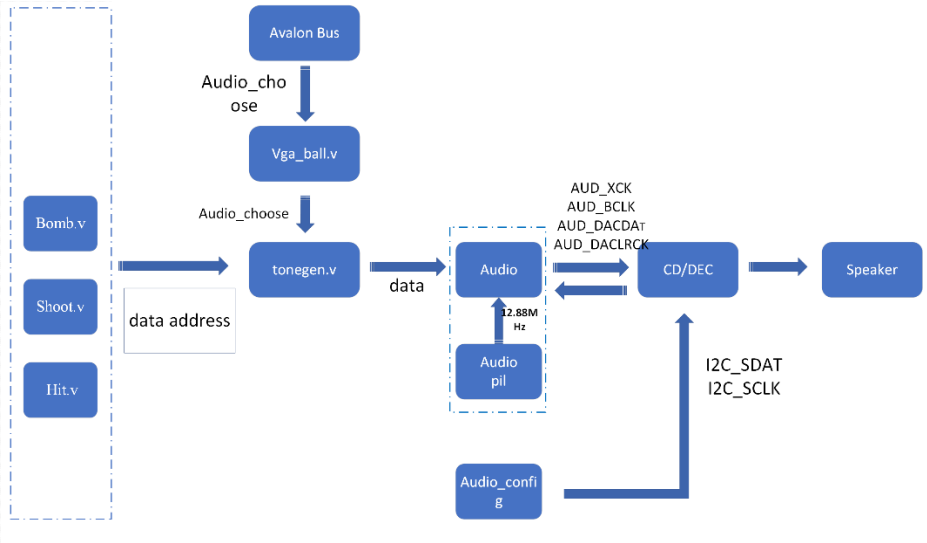


Fig. 11 audio configuration

First, on the internet, we found the game Battle City audio file (.mp3) and using audacity to subsample it at 8KHz, and because we will config WM8731 to Left Justified Mode which means the right and left channel information are the same, subsample only one channel, finally convert it to 16 bits mono PCM .wav. After that, make sure the bit rate of .wav is 128Kbps. Second, using a Hex editor to eliminate undesirable information at the beginning of the .wav file. Third, use MATLAB to generate .mif file.

Accordingly, tonegen module feed the data to ip core audio, which includes a FIFO in it and take care of the timing for us. The FIFO frequency is 48kHz, thus in tonegen module we manually add a frequency divider of 6. Then Audio module is connected to WM8731 through the ports. In the final file, tonegen code is added to main code directly.

The next step is to config the bulid-in IP core. The IP core offer pil clock at 12.88MHz to

drive the audio chip.

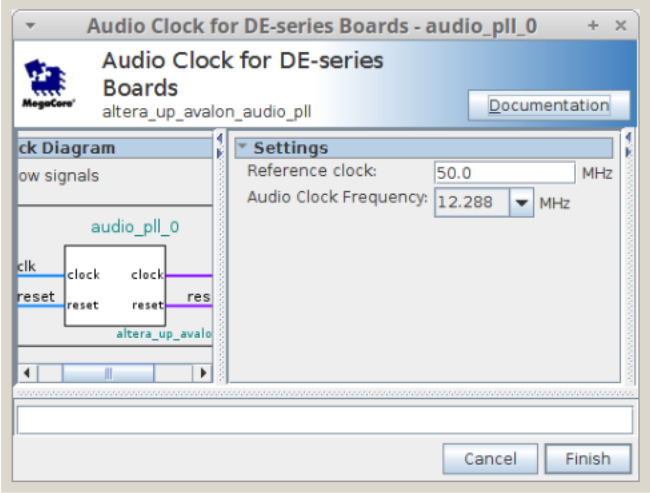


Fig. 12 Audio PLL Configuration

The next thing is to connect lines between the chip and audio configuration. Here we follow the instruction by Cambridge University [5]. To make things even simpler, combine the code of tongen.sv to the vga_ball.sv so that there is only one System Verilog file in our design. The Final QSYS connection is shown in Fig.13

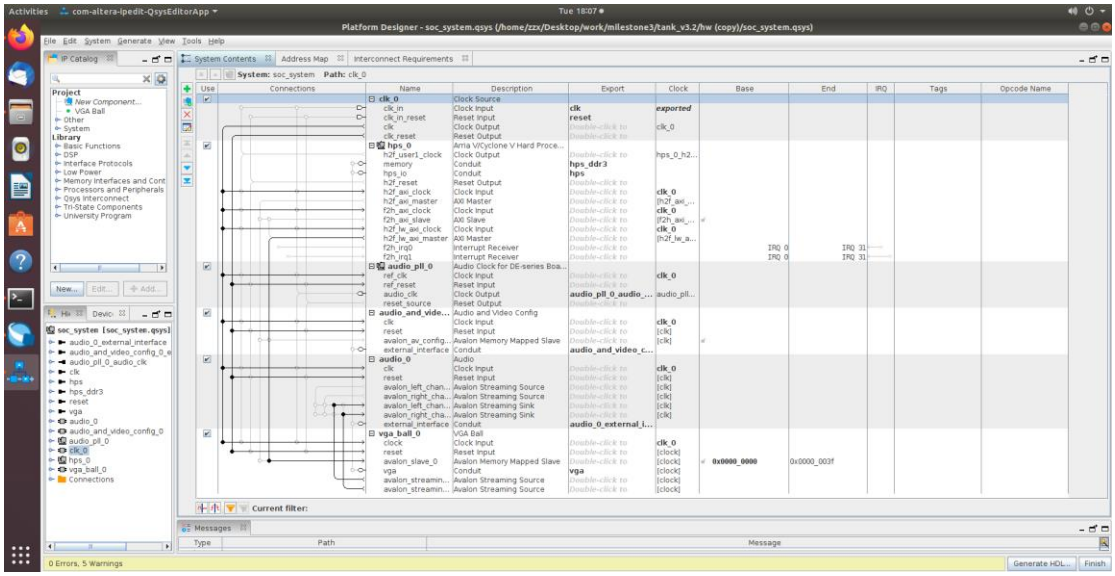


Fig. 13 Final QSYS connection

IV. Software design

1. User Input

We will be using Joystick to control the game. The Joystick we used is show in Fig. 14.

Fig. 14 iNNEXT Joystick Gamepad

The function of this game is list in Table 3.

Table 3 Button Function of Joystick

Button	Function	Button	Function
Up	Tank move up	Start	end game
Down	Tank move down	A	Shoot
Left	Tank move left	B	Summon Player tank
Right	Tank move right		

The joystick works just like USB keyboard, it follows USB protocols 0. So it can be perfectly operate following the code in 'Lab 2' by changing the judgement from 1 to 0

```
if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
     inter->bInterfaceProtocol == 0) {
```

At each interrupt, the joystick will send 8 Byte data through USB interface. The detail of the default data and data when each key is pressed is list in table 4.

Table 4 Joystick USB data

	0	1	2	3	4	5	6	7
default	01	7F	7F	7F	7F	0F	0	0
A						2F		
B						4F		
X						1F		
Y						8F		
up					00			
down					FF			
left				00				
right				FF				
L2								01
R2								02
Start							20	
Select							10	

2. Game Logic

a. Movement logic

All the tank, player tank and enemy tank will follow the movement logic below.

(1) Auto adjustment

To improve the controllability of tank and player user experience, the original tank game implements a function of auto adjustment. When tank turned, it will be auto adjusted to the ‘center’ of the road. Therefore, the all the tank can only be move on the redline in fig.15.

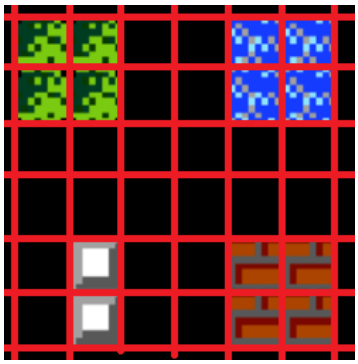


Fig. 15 Tank Moving Line

This function is achieved by doing round-off of tank position when tank is turning. In this project, this auto adjustment fixes the tank's movements also tank's bullets' position which greatly reduce unnecessary logic and avoid bugs.

(2) Brick and Tank detection

In our game, all types of tank can only move on blank or grass area. It cannot move across wall, brick and water or another tank. When a tank wants to move in a certain direction, the software will calculate its next position and judge whether there is another object that block the road.

b. Bullet logic.

(1) Bullet movement

The normal speed of bullet is twice the movement speed of normal tank. It will continue move on the screen. Bullet will stop move and disappear in the following situation:

i. Hit wall.

Notice that one single bullet and only destroy half of the wall. The wall will change in following corresponding way. Each bullet will destroy 4*8 size pixel both side of it's moving direction. The corresponding change is of fire bullet from left to right is shown is table.

Table 5 Brick change fire from left side

Original				
attack from top left				
attack from bottom left				

ii. Hit Brick, or the edge of the war field.

Only the bullet will disappear.

iii. Hit enemy tank or players tank.

The corresponding tank will disappear on the screen. The bullet will also disappear.

(2) Bullet number.

Each tank can only fire at most 1 bullet. Tank cannot fire until the previous bullet disappears based on previous logic.

c. Player tank logic

(1) Player tank movement

The movement of player tank is based on data of the user press on joystick. The actual move of the tank follows the 'Movement logic' as illustrate previously.

(2) Player tank life.

This game implements a life system. Initially, player have 3 tank life. When tank is destroyed by enemy, users can press B to summon a new one. And the remaining life will be decreased by 1. When all 3 life are used, the user will lose the game.

(3) Player tank upgrade.

Player tank will be upgrade by destroy enemy tank. There are total 4 different type as illustrate in Section Fig…. When player tank upgrade to level 3 or above, the speed of player tank will be doubled which allows player to destroy enemy more accurately.

Table 6 Player 1 tank information

Tank image (16*16 pixel *8)	Feature	Bullet Speed
	Level 1	Slow
	Level 2	Slow
	Level 3	Fast
	Level 4	Fast

d. Enemy tank logic

(1) Enemy tank movement

Enemy tank will also follow the ‘Movement logic’. Besides, the direction of enemy tank will change based on following two logic:

i. If enemy tank reach wall, edge or other tank and stop moving, it will change its direction randomly.

ii. Enemy tank will have a chance 1/64pixels to change its direction when it is moving. This allow enemy tank to reach all position in the field.

(2) Enemy fire bullet

The enemy will fire bullet based on ‘Bullet Logic’. When the previous bullet disappears, the enemy tank will fire a new bullet randomly (with probability 1/32pixel) when moving.

(3) Enemy tank class

There are four types of enemy tank, and some tank will have special character. The detail of tank is list in table 7.

Table 7 Enemy normal tank information

Tank image (16*16 pixel)	Feature	Bullet Speed	Movement Speed
	Level 1	Slow	Slow
	Level 2	Slow	Fast
	Level 3	Fast	Slow
	Level 4	Slow	Slow

e. Win and Lose logic

i. Win logic: Player destroy all 16 enemy tank.

ii. Lose logic: Player use all 3 tank or base is under attack.

f. Win and Lose graph

We design to simple lose, win graph which will be display at the end of the game. The graph is shown in Fig.16.

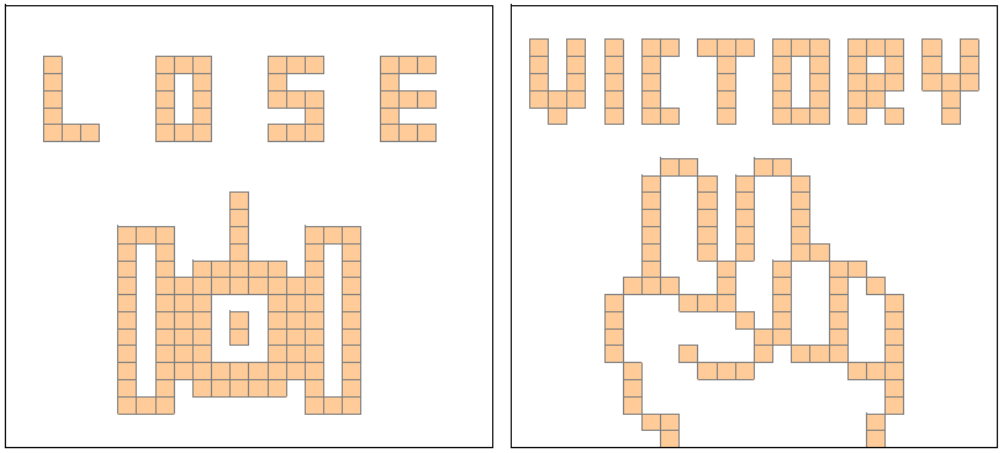


Fig. 16 Win Lose graph

3. Avalon Bus interface

At each iteration cycle, the software will send 19*16 bits information to hardware side, the detail of the information is list in table 8.

Table 8 Avalon Bus data

Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remark
00	X Position					Y Position					Brick					En.	Wall update, each will update 8×8
01	X Position					Y Position					Brick					En.	Score, Explosive, Gadget Tile
02	X Position										Y Position						Player tank Sprite
03	Tank Class				Color number						Direction					En.	
04	X Position										Y Position						Emeny tank 1 Sprite
05	Tank Class				Color number						Direction					En.	
06	X Position										Y Position						Emeny tank 2 Sprite
07	Tank Class				Color number						Direction					En.	
08	X Position										Y Position						Emeny tank 3 Sprite
09	Tank Class				Color number						Direction					En.	
10	X Position										Y Position						Emeny tank 4 Sprite
11	Tank Class				Color number						Direction					En.	
12	X Position										Y Position						Bullet 1 Position
13	X Position										Y Position						Bullet 2 Position
14	X Position										Y Position						Bullet 3 Position
15	X Position										Y Position						Bullet 4 Position
16	X Position										Y Position						Bullet 5 Position
17	Direction5		Direction 4		Direction 3		Direction2		Direction 1			En. 5	En. 4	En. 3	En. 2	En. 1	Bullet direction and visiable
18															Sound		Game Information

V. Challenges

1. Enlarge the whole game.

The original game is just 224*256, it will only use 1/4 of the whole VGA screen. To fully use the screen and improve the user experience, we enlarge the whole game once. To make it a 448*512 game and fill the rest of the space with proper color.

2. Display image correctly using ROM and RAM

Tank and map image information are store in ROM. But to decrease the memory, the shape and color are store in two different ROM. Figuring out the time sequence and make the output image display correctly is not easy.

3. Achieve the movement logic correctly.

The enemy tank will still have a change to run into each other, but they will not be stuck, they can continue moving.

4. Design the AI of the enemy tank.

The enemy tank should move ‘randomly’ and also have a reasonable fire mechanism.

VI. Reference

1. Battle city online game

https://www.retrogames.cz/play_014-NES.php

2. Battle city General Sprite

http://mirrors.pdp-11.ru/_sprites/www.spriters-resource.com/nes/batcity/sheet/60016/index.html

3. DE1-SoC User Manal

4. ROM IP Core usage

https://blog.csdn.net/weixin_41445387/article/details/86835663

5. Audio CODEC tutorial

https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/optional-tonegen.html

VII. Appendix

1. Vga_ball.sv

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module vga_ball(input logic              clk,
                input logic          reset,
            input logic [15:0]    writedata,
            input logic           write,
            input                 chipselect,
            input logic [4:0]    address,

            input left_chan_ready,
            input right_chan_ready,
            output logic [15:0] sample_data_l,
            output logic sample_valid_l,
            output logic [15:0] sample_data_r,
            output logic sample_valid_r,

            output logic [7:0] VGA_R, VGA_G, VGA_B,
            output logic        VGA_CLK, VGA_HS, VGA_VS,
                                    VGA_BLANK_n,
            output logic        VGA_SYNC_n);

    logic [10:0]          hcount;
    logic [9:0]         vcount;
    logic [10:0]          hcount_data;
    logic [10:0]          hcount_data2;
    logic [9:0]         vcount_data;
    logic [9:0]         vcount_data2;
    logic [7:0]           background_r, background_g,
background_b,ball_x1,ball_x2,ball_y1,ball_y2;
    logic [15:0]
wall1,wall2,tank1_p,tank1_d,enemy1_p,enemy1_d,enemy2_p,enemy2_d,enemy3_p,enemy3_d,e
nemy4_p,enemy4_d;
    logic [15:0]       bullet1_p,bullet2_p,bullet3_p,bullet4_p,bullet5_p,bullet_data,game_data;
    logic [10:0]    x;
```

```
logic [8:0]      y;

 reg [11:0] counter;
 logic flag1;
 logic flag2;
 logic flag3;

vga_counters counters(.clk50(clk), .*);
reg [255:0] image;

always_ff @(posedge clk) begin
   if (reset) begin
 background_r <= 8'hff;
 background_g <= 8'hff;
 background_b <= 8'hff;
        tank1_d <= 16'b0111001100100000;
 tank1_p <= 16'hA079;
        wall1 <= 16'h0000;
        wall2 <= 16'h0000;
        enemy1_p <= 16'h5018;
        enemy1_d <= 16'h4201;
        enemy2_p <= 16'h0000;
        enemy2_d <= 16'h0000;
        enemy3_p <= 16'h0000;
        enemy3_d <= 16'h0000;
        enemy4_p <= 16'h0000;
        enemy4_d <= 16'h0000;
        bullet1_p <= 16'h7880;
        bullet2_p <= 16'h0000;
        bullet3_p <= 16'h0000;
        bullet4_p <= 16'h0000;
        bullet5_p <= 16'h0000;
        bullet_data <= 16'h0001;
        game_data <= 16'h0003;
   end else if (chipselect && write)
      case (address)
   5'h0 : wall1 <= writedata;
   5'h1 : wall2 <= writedata;
          5'h2 : tank1_p <= writedata;
          5'h3 : tank1_d <= writedata;
          5'h4 : enemy1_p <= writedata;
```

```
            5'h5 : enemy1_d <= writedata;
            5'h6 : enemy2_p <= writedata;
            5'h7 : enemy2_d <= writedata;
            5'h8 : enemy3_p <= writedata;
            5'h9 : enemy3_d <= writedata;
            5'ha : enemy4_p <= writedata;
            5'hb : enemy4_d <= writedata;
            5'hc : bullet1_p <= writedata;
            5'hd : bullet2_p <= writedata;
            5'he : bullet3_p <= writedata;
            5'hf : bullet4_p <= writedata;
            5'h10 : bullet5_p <= writedata;
            5'h11: bullet_data <= writedata;
            5'h12: game_data <= writedata;
        endcase
end

    reg  [10:0]   address1;
    wire [15:0]   q1;

    shoot audio1(.address(address1), .clock(clk), .q(q1)); //1653


    reg  [10:0]   address2;
    wire [15:0]   q2;
    hit   audio2(.address(address2), .clock(clk), .q(q2));//1235

    reg  [12:0]   address3;
    wire [15:0]   q3;
    bomb audio3(.address(address3), .clock(clk), .q(q3));//5832

    always_ff @(posedge clk) begin
        if(reset) begin
            counter <= 0;
            sample_valid_l <= 0; sample_valid_r <= 0;
        end

        else if(left_chan_ready == 1 && right_chan_ready == 1 && counter < 3125) begin
            counter <= counter + 1;
            sample_valid_l <= 0; sample_valid_r <= 0;
        end
```

```
else if(left_chan_ready == 1 && right_chan_ready == 1 && counter == 3125) begin
    counter <= 0;
    sample_valid_l <= 1; sample_valid_r <= 1;

    if (game_data[1:0]==3'b11 || flag3 ==1'b0) begin
        if (address3 < 13'd5832) begin
            address3 <= address3+1;
            flag3 <= 1'b0;
        end
        else begin
            address3 <=0;
            flag3 <= 1'b1;
        end
        sample_data_l <= q3;
        sample_data_r <= q3;
    end


    else if (game_data[1:0]==3'b01 || flag1 ==1'b0) begin
        if (address1 < 11'd1653) begin
            address1 <= address1+1;
            flag1 <= 1'b0;
        end
        else begin
            address1 <=0;
            flag1 <= 1'b1;
        end
        sample_data_l <= q1;
        sample_data_r <= q1;
    end

    else if (game_data[1:0]==3'b10 || flag2 ==1'b0) begin
        if (address2 < 11'd1235) begin
            address2 <= address2+1;
            flag2 <= 1'b0;
        end
        else begin
            address2 <=0;
            flag2 <= 1'b1;
        end
        sample_data_l <= q2;
```

```
                sample_data_r <= q2;
            end

            else begin
                sample_data_l <= 0;
                sample_data_r <= 0;
            end
        end

        else begin
        sample_valid_l <= 0; sample_valid_r <= 0;
        end
    end


  //---------------------- RAM map.v ------------------------------------
    logic map_en;
    logic [9:0] map_address;
    logic [4:0] map_data;
    logic [4:0] map_output;
    map
map_unit(.address(map_address),.clock(clk),.data(map_data),.wren(map_en),.q(map_output));

    always_ff @(posedge clk) begin
        if(wall1[0]) begin
          map_en <=1;
          map_address <= wall1[15:11] + wall1[10:6]*32;
          map_data <= wall1[5:1];
        end
        else begin
          map_en <=0;
          map_address <= hcount_data2[10:5]-2+vcount_data2[8:4]*32-32;
        end
    end
    //---------------------- ROM wall.v ------------------------------------
    //------------------ store the image of wall --------------------------
    logic [7:0] wall_address;
    logic [15:0] wall_output;
    wall wall_unit(.address(wall_address),.clock(clk),.q(wall_output));
    always_ff @(posedge clk) begin
          wall_address <= map_output*8 + vcount_data2[3:1];
```

```
    // end
    end

    // -------------------------ROM tank.v -------------------------
    logic [9:0] tank_address;
    logic [31:0] tank_output;
    tank tank_unit(.address(tank_address),.clock(clk),.q(tank_output));
    always_ff @(posedge clk) begin
    if (hcount_data <(tank1_p[15:8]*4+64) && vcount_data <(tank1_p[7:0]*2+32) &&
hcount_data>=(tank1_p[15:8]*4) && vcount_data >=(tank1_p[7:0]*2))
            tank_address <= vcount_data[8:1]-
tank1_p[7:0]+tank1_d[15:12]*8*16+tank1_d[5:4]*32+(tank1_p[8]^tank1_p[0])*16; //display
player1 tank

    else if (enemy1_d[0] && hcount_data <(enemy1_p[15:8]*4+64) && vcount_data
<(enemy1_p[7:0]*2+32) && hcount_data>=(enemy1_p[15:8]*4) &&
vcount_data >=(enemy1_p[7:0]*2))
            tank_address <= vcount_data[8:1]-
enemy1_p[7:0]+enemy1_d[15:12]*8*16+enemy1_d[5:4]*32+(enemy1_p[8]^enemy1_p[0])*16;
//display enemy1 tank

    else if (enemy2_d[0] && hcount_data <(enemy2_p[15:8]*4+64) && vcount_data
<(enemy2_p[7:0]*2+32) && hcount_data>=(enemy2_p[15:8]*4) &&
vcount_data >=(enemy2_p[7:0]*2))
            tank_address <= vcount_data[8:1]-
enemy2_p[7:0]+enemy2_d[15:12]*8*16+enemy2_d[5:4]*32+(enemy2_p[8]^enemy2_p[0])*16;
//display enemy2 tank

    else if (enemy3_d[0] && hcount_data <(enemy3_p[15:8]*4+64) && vcount_data
<(enemy3_p[7:0]*2+32) && hcount_data>=(enemy3_p[15:8]*4) &&
vcount_data >=(enemy3_p[7:0]*2))
            tank_address <= vcount_data[8:1]-
enemy3_p[7:0]+enemy3_d[15:12]*8*16+enemy3_d[5:4]*32+(enemy3_p[8]^enemy3_p[0])*16;
//display enemy3 tank

    else if (enemy4_d[0] && hcount_data <(enemy4_p[15:8]*4+64) && vcount_data
<(enemy4_p[7:0]*2+32) && hcount_data>=(enemy4_p[15:8]*4) &&
vcount_data >=(enemy4_p[7:0]*2))
            tank_address <= vcount_data[8:1]-
enemy4_p[7:0]+enemy4_d[15:12]*8*16+enemy4_d[5:4]*32+(enemy4_p[8]^enemy4_p[0])*16;
//display enemy4 tank
```

```
end

   //------------------------ ROM colormap.v ---------------------------
   //------------------------ store tank color -----------------
     logic [4:0] colormap_address;
     logic [23:0] colormap_output;
     colormap colormap_unit(.address(colormap_address),.clock(clk),.q(colormap_output));
     always_ff @(posedge clk) begin
     if (hcount_data <(tank1_p[15:8]*4+64) && vcount_data <(tank1_p[7:0]*2+32) &&
hcount_data>=(tank1_p[15:8]*4) && vcount_data >=(tank1_p[7:0]*2))
            colormap_address    <= tank_output>>(30-2*(hcount_data[9:2]-
tank1_p[15:8]))&32'd3|(tank1_d[9:6]&4'b1100);      // display player1 tank

     else if (enemy1_d[0] && hcount_data <(enemy1_p[15:8]*4+64) && vcount_data
<(enemy1_p[7:0]*2+32) && hcount_data>=(enemy1_p[15:8]*4) &&
vcount_data >=(enemy1_p[7:0]*2))
            colormap_address    <= tank_output>>(30-2*(hcount_data[9:2]-
enemy1_p[15:8]))&32'd3|(enemy1_d[9:6]&4'b1100);      // display enemy1 tank

     else if (enemy2_d[0] && hcount_data <(enemy2_p[15:8]*4+64) && vcount_data
<(enemy2_p[7:0]*2+32) && hcount_data>=(enemy2_p[15:8]*4) &&
vcount_data >=(enemy2_p[7:0]*2))
            colormap_address    <= tank_output>>(30-2*(hcount_data[9:2]-
enemy2_p[15:8]))&32'd3|(enemy2_d[9:6]&4'b1100);      // display enemy2 tank

     else if (enemy3_d[0] && hcount_data <(enemy3_p[15:8]*4+64) && vcount_data
<(enemy3_p[7:0]*2+32) && hcount_data>=(enemy3_p[15:8]*4) &&
vcount_data >=(enemy3_p[7:0]*2))
            colormap_address    <= tank_output>>(30-2*(hcount_data[9:2]-
enemy3_p[15:8]))&32'd3|(enemy3_d[9:6]&4'b1100);      // display enemy3 tank

     else if (enemy4_d[0] && hcount_data <(enemy4_p[15:8]*4+64) && vcount_data
<(enemy4_p[7:0]*2+32) && hcount_data>=(enemy4_p[15:8]*4) &&
vcount_data >=(enemy4_p[7:0]*2))
            colormap_address    <= tank_output>>(30-2*(hcount_data[9:2]-
enemy4_p[15:8]))&32'd3|(enemy4_d[9:6]&4'b1100);      // display enemy4 tank


     end
```

```
// --------------------------ROM wallcolor.v ------------------------

    logic [4:0] wallcolor_address;
    logic [23:0] wallcolor_output;
    wallcolor wallcolor_unit(.address(wallcolor_address),.clock(clk),.q(wallcolor_output));
    always_ff @(posedge clk) begin
    if (hcount_data>=64 && hcount_data <1024 && vcount_data>=16 && vcount_data< 432)
begin
        case (map_output)
        4'b0000: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b00000); // blank
        4'b0001: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b00000); // wall
        4'b0010: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b00000); // wall
        4'b0011: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b00000); // wall
        4'b0100: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b00000); // wall
        4'b0101: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b00000); // wall
        4'b0110: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b00100); // brick
        4'b0111: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b00100); // ice
        4'b1000: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b01000); // grass
        4'b1001: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b01100); // water
        4'b1010: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b01100); // water
        4'b1011: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b01100); // water

        default: wallcolor_address   <= wall_output>>(14-
2*(hcount_data[4:2]))&32'd3|(5'b10000); // other
        endcase
    end
end
```

```
always_comb begin
            {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
            if(hcount==1278) hcount_data = 1;
            else if(hcount==1279) hcount_data = 2;
            else if(hcount==1277) hcount_data =0;
      else hcount_data = hcount+3;
            hcount_data2 = hcount +8; //??????????????????????????
            vcount_data = vcount;
            vcount_data2 = vcount;

      //----------------------- display on VGA -------------------------------------
      if (VGA_BLANK_n ) begin
      // display bullet
      // bullet 1
      if ((bullet_data[0]!=0) && hcount>= (bullet1_p[15:8]*4) && hcount<=
(bullet1_p[15:8]*4+11) &&    vcount >=(bullet1_p[7:0]*2) &&    vcount
<=(bullet1_p[7:0]*2+5))
            {VGA_R, VGA_G, VGA_B} = {8'hAD, 8'hAD, 8'hAD};
      // bullet 2
      else if ((bullet_data[1]!=0) && hcount>= (bullet2_p[15:8]*4) && hcount<=
(bullet2_p[15:8]*4+11) &&    vcount >=(bullet2_p[7:0]*2) &&    vcount
<=(bullet2_p[7:0]*2+5))
            {VGA_R, VGA_G, VGA_B} = {8'hAD, 8'hAD, 8'hAD};
      // bullet 3
      else if ((bullet_data[2]!=0) && hcount>= (bullet3_p[15:8]*4) && hcount<=
(bullet3_p[15:8]*4+11) &&    vcount >=(bullet3_p[7:0]*2) &&    vcount
<=(bullet3_p[7:0]*2+5))
              {VGA_R, VGA_G, VGA_B} = {8'hAD, 8'hAD, 8'hAD};
      // bullet 4
      else if ((bullet_data[3]!=0) && hcount>= (bullet4_p[15:8]*4) && hcount<=
(bullet4_p[15:8]*4+11) &&    vcount >=(bullet4_p[7:0]*2) &&    vcount
<=(bullet4_p[7:0]*2+5))
              {VGA_R, VGA_G, VGA_B} = {8'hAD, 8'hAD, 8'hAD};
      // bullet 5
      else if ((bullet_data[4]!=0) && hcount>= (bullet5_p[15:8]*4) && hcount<=
(bullet5_p[15:8]*4+11) &&    vcount >=(bullet5_p[7:0]*2) &&    vcount
<=(bullet5_p[7:0]*2+5))
              {VGA_R, VGA_G, VGA_B} = {8'hAD, 8'hAD, 8'hAD};
      // map
            else if (hcount>=64 && hcount<1024 && vcount>=16 && vcount<432 &&
wallcolor_output)
```

```
            {VGA_R, VGA_G, VGA_B} = wallcolor_output;
            // player tank
        else if ((tank1_d[0]!=0) && hcount <(tank1_p[15:8]*4+64) && vcount
<(tank1_p[7:0]*2+32) && hcount>=(tank1_p[15:8]*4) && vcount >=(tank1_p[7:0]*2))
            {VGA_R, VGA_G, VGA_B} = colormap_output;
            // enemy tank1
             else if ((enemy1_d[0]!=0) && hcount <(enemy1_p[15:8]*4+64) && vcount
<(enemy1_p[7:0]*2+32) && hcount>=(enemy1_p[15:8]*4) && vcount >=(enemy1_p[7:0]*2))
            {VGA_R, VGA_G, VGA_B} = colormap_output;
            // enemy tank2
             else if ((enemy2_d[0]!=0) && hcount <(enemy2_p[15:8]*4+64) && vcount
<(enemy2_p[7:0]*2+32) && hcount>=(enemy2_p[15:8]*4) && vcount >=(enemy2_p[7:0]*2))

            {VGA_R, VGA_G, VGA_B} = colormap_output;
            // enemy tank3
             else if ((enemy3_d[0]!=0) && hcount <(enemy3_p[15:8]*4+64) && vcount
<(enemy3_p[7:0]*2+32) && hcount>=(enemy3_p[15:8]*4) && vcount >=(enemy3_p[7:0]*2))
            {VGA_R, VGA_G, VGA_B} = colormap_output;
            // enemy tank4
             else if ((enemy4_d[0]!=0) && hcount <(enemy4_p[15:8]*4+64) && vcount
<(enemy4_p[7:0]*2+32) && hcount>=(enemy4_p[15:8]*4) && vcount >=(enemy4_p[7:0]*2))

            {VGA_R, VGA_G, VGA_B} = colormap_output;
        // background
        else if (hcount>=64 && hcount<896 && vcount>=16 && vcount<432)
            {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
             else
            {VGA_R, VGA_G, VGA_B} = {8'h63, 8'h63, 8'h63};

           end
end

endmodule

module vga_counters(
 input logic             clk50, reset,
 output logic [10:0] hcount,    // hcount[10:1] is pixel column
 output logic [9:0]    vcount,    // vcount[9:0] is pixel row
 output logic                 VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
```

```
* 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
*
* HCOUNT 1599 0                    1279          1599 0
*                _________________                 _________
* _____________|       Video       |_____________|   Video
*
*
* |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
*          _____________________        _____________
* |____|          VGA_HS              |____|
*/
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                            HBACK_PORCH; // 1600

   // Parameters for vcount
   parameter VACTIVE      = 10'd 480,
             VFRONT_PORCH = 10'd 10,
             VSYNC        = 10'd 2,
             VBACK_PORCH  = 10'd 33,
             VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                            VBACK_PORCH; // 525

   logic endOfLine;

   always_ff @(posedge clk50 or posedge reset)
      if (reset)              hcount <= 0;
      else if (endOfLine) hcount <= 0;
      else                    hcount <= hcount + 11'd 1;

   assign endOfLine = hcount == HTOTAL - 1;

   logic endOfField;

   always_ff @(posedge clk50 or posedge reset)
      if (reset)              vcount <= 0;
      else if (endOfLine)
```

```
        if (endOfField)     vcount <= 0;
        else                vcount <= vcount + 10'd 1;

    assign endOfField = vcount == VTOTAL - 1;

    // Horizontal sync: from 0x520 to 0x5DF (0x57F)
    // 101 0010 0000 to 101 1101 1111
    assign VGA_HS = !( (hcount[10:8] == 3'b101) &
                !(hcount[7:5] == 3'b111));
    assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

    assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

    // Horizontal active: 0 to 1279       Vertical active: 0 to 479
    // 101 0000 0000    1280          01 1110 0000    480
    // 110 0011 1111    1599          10 0000 1100    524
    assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
            !( vcount[9] | (vcount[8:5] == 4'b1111) );

    /* VGA_CLK is 25 MHz
     *                    __      __      __
     * clk50        __|   |__|   |__|
     *
     *                    _____          __
     * hcount[0]__|       |_____|
     */
    assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

2. Hello.c

```
#include <stdio.h>
#include "vga_ball.h"
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
```

```
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include "usbkeyboard.h"
#include <pthread.h>

#define SERVER_HOST "128.59.64.152"
#define SERVER_PORT 42000

#define BUFFER_SIZE 128

int vga_ball_fd;

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

pthread_t network_thread;
void *network_thread_f(void *);


/* Read and print the background color */
void print_background_color() {
  vga_ball_arg_t vla;

  if (ioctl(vga_ball_fd, VGA_BALL_READ_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_READ_BACKGROUND) failed");
      return;
  }
  printf("%04x\n",vla.background.game_data);
}

/* Set the background color */
void set_background_color(vga_ball_color_t *c)
{
  vga_ball_arg_t vla;
  vla.background = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
      return;
  }
}
```

```
// attack bullet

void bullet_cal(char *c,short *d,char *e,char flag)
{
	char x,y;


	if(*(c+3)){
	switch(*(c+2)){
	case 0x00 : *(c+1) = *(c+1)-0x02; break;
	case 0x01: *c = *c-0x02;break;
	case 0x02: *(c+1) = *(c+1)+0x02;break;
	case 0x03: *c = *c + 0x02;break;}

	if(*c<16||*c>220||*(c+1)<8||*(c+1)>216)    *(c+3) = 0x00;


	if(*c<128&&*c>112&&*(c+1)<216&&*(c+1)>200)
	{ *(d+18) = *(d+18)&0x0fff|0x1000; *(d+18) = (*(d+18))&0xfffc|0x0003;
	set_background_color(d);}


if(flag){
	// attack emeny1 tank
	if(((*(d+5))&0x0001)== 0x0001){
	if((*c)<(((*(d+4))&0xff00)/256+16)&&(*c)>(((*(d+4))&0xff00)/256))
		{if ((*(c+1))<(((*(d+4))&0x00ff)%256+16)&&(*(c+1))>(((*(d+4))&0x00ff)%256) )
	{	*(d+5) = (*(d+5))&0xfffe;
		*(d+18) = (*(d+18))&0xfffc|0x0003;
		set_background_color(d);
				*(c+3) = 0x00;}}}

	// attack emeny2 tank
	if(((*(d+7))&0x0001)== 0x0001){
	if((*c)<(((*(d+6))&0xff00)/256+16)&&(*c)>(((*(d+6))&0xff00)/256))
		{if ((*(c+1))<(((*(d+6))&0x00ff)%256+16)&&(*(c+1))>(((*(d+6))&0x00ff)%256) )
	{	*(d+7) = (*(d+7))&0xfffe;
		*(d+18) = (*(d+18))&0xfffc|0x0003;
		set_background_color(d);
				*(c+3) = 0x00;}}}
```

```
	// attack emeny3 tank
	if(((*(d+9))&0x0001)== 0x0001){
	if((*c)<(((*(d+8))&0xff00)/256+16)&&(*c)>(((*(d+8))&0xff00)/256))
		{if ((*(c+1))<(((*(d+8))&0x00ff)%256+16)&&(*(c+1))>(((*(d+8))&0x00ff)%256) )
	{	*(d+9) = (*(d+9))&0xfffe;
		*(d+18) = (*(d+18))&0xfffc|0x0003;
		set_background_color(d);
				*(c+3) = 0x00;}}}

	// attack emeny4 tank
	if(((*(d+11))&0x0001)== 0x0001){
	if((*c)<(((*(d+10))&0xff00)/256+16)&&(*c)>(((*(d+10))&0xff00)/256))
		{if ((*(c+1))<(((*(d+10))&0x00ff)%256+16)&&(*(c+1))>(((*(d+10))&0x00ff)%256) )
	{	*(d+11) = (*(d+11))&0xfffe;
		*(d+18) = (*(d+18))&0xfffc|0x0003;
		set_background_color(d);
				*(c+3) = 0x00;}}}
}

else {
	if(((*(d+3))&0x0001)== 0x0001){
	if((*c)<(((*(d+2))&0xff00)/256+16)&&(*c)>(((*(d+2))&0xff00)/256))
		{if ((*(c+1))<(((*(d+2))&0x00ff)%256+16)&&(*(c+1))>(((*(d+2))&0x00ff)%256) )
	{	*(d+3) = (*(d+3))&0xfffe;
		*(d+18) = (*(d+18))&0xfffc|0x0003;
		set_background_color(d);
				*(c+3) = 0x00;}}}
}


	x = (*c-16)/8;
	y = (*(c+1)-8)/8;

	switch(*(c+2))    //table[0] = i*2048+j*64+map[j][i]*2+1;
	{
	    case 0x00:{ if( (*(e+y*26+x)) ==1)             {*(e+y*26+x) = 2;*(c+3) =
0x00;*d=x*2048+y*64+2*2+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                  else if( (*(e+y*26+x)) ==4)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                  else if( (*(e+y*26+x)) ==2)    {*(e+y*26+x) = 0;*(c+3) =
```

```
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==3)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==6)    {*(c+3) = 0x00;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);} //brick bullet disapper
                                    if( (*(e+y*26+x+1)) ==1)    {*(e+y*26+x+1) = 2;*(c+3) =
0x00;*d=(x+1)*2048+2*2+y*64+1;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+1)) ==4)    {*(e+y*26+x+1) = 0;*(c+3) =
0x00;*d=(x+1)*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+1)) ==2)    {*(e+y*26+x+1) = 0;*(c+3) =
0x00;*d=(x+1)*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+1)) ==5)    {*(e+y*26+x+1) = 0;*(c+3) =
0x00;*d=(x+1)*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+1)) ==6)    {*(c+3) = 0x00;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);} //brick bullet disapper
                                    break;}

          case 0x01:{ if( (*(e+y*26+x)) ==1)              {*(e+y*26+x) = 5;*(c+3) =
0x00;*d=x*2048+y*64+5*2+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==5)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==4)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==3)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==6)    {*(c+3) = 0x00;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);} //brick bullet disapper
                                    if( (*(e+y*26+x+26)) ==1)    {*(e+y*26+x+26) = 5;*(c+3) =
0x00;*d=x*2048+5*2+(y+1)*64+1;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+26)) ==3)    {*(e+y*26+x+26) = 0;*(c+3) =
0x00;*d=x*2048+(y+1)*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+26)) ==2)    {*(e+y*26+x+26) = 0;*(c+3) =
0x00;*d=x*2048+(y+1)*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+26)) ==5)    {*(e+y*26+x+26) = 0;*(c+3) =
0x00;*d=x*2048+(y+1)*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+26)) ==6)    {*(c+3) = 0x00;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);} //brick bullet disapper
                                    break;          }
```

```
        case 0x02:{ if( (*(e+y*26+x)) ==1)            {*(e+y*26+x) = 4;*(c+3) =
0x00;*d=x*2048+y*64+4*2+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==4)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==2)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==3)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==6)    {*(c+3) = 0x00;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);} //brick bullet disapper
                                    if( (*(e+y*26+x+1)) ==1)    {*(e+y*26+x+1) = 4;*(c+3) =
0x00;*d=(x+1)*2048+4*2+y*64+1;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+1)) ==4)    {*(e+y*26+x+1) = 0;*(c+3) =
0x00;*d=(x+1)*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+1)) ==2)    {*(e+y*26+x+1) = 0;*(c+3) =
0x00;*d=(x+1)*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+1)) ==5)    {*(e+y*26+x+1) = 0;*(c+3) =
0x00;*d=(x+1)*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+1)) ==6)    {*(c+3) = 0x00;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);} //brick bullet disapper
                                    break;          }

        case 0x03:{ if( (*(e+y*26+x)) ==1)            {*(e+y*26+x) = 3;*(c+3) =
0x00;*d=x*2048+y*64+3*2+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==5)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==4)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==3)    {*(e+y*26+x) = 0;*(c+3) =
0x00;*d=x*2048+y*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x)) ==6)    {*(c+3) = 0x00;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);} //brick bullet disapper
                                    if( (*(e+y*26+x+26)) ==1)    {*(e+y*26+x+26) = 3;*(c+3) =
0x00;*d=x*2048+3*2+(y+1)*64+1;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+26)) ==3)    {*(e+y*26+x+26) = 0;*(c+3) =
0x00;*d=x*2048+(y+1)*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+26)) ==2)    {*(e+y*26+x+26) = 0;*(c+3) =
0x00;*d=x*2048+(y+1)*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                    else if( (*(e+y*26+x+26)) ==5)    {*(e+y*26+x+26) = 0;*(c+3) =
```

```
0x00;*d=x*2048+(y+1)*64+1;*(d+18) = (*(d+18))&0xfffc|0x0002;set_background_color(d);}
                                else if( (*(e+y*26+x+26)) ==6)    {*(c+3) = 0x00;*(d+18) =
(*(d+18))&0xfffc|0x0002;set_background_color(d);} //brick bullet disapper
                                break;          }
      }
            *d=x*2048+y*64+0;

            set_background_color(d);
     }
}

void rote(char i, char dir,char *c,short *d)    //tank,direction,map,table
{
            int temp1,temp2,temp3,temp4,temp5,temp6;
            short temp_check=0;
            char j;
            char check1=0;
            char check2=0;
            char check3=0;
            char check4=0;

               temp1 = (((*(d+2*i))&0xff00)/256-0x10)/8 + ((((*(d+2*i))&0x00ff)%256-
0x09)/8)*26;
               temp2 = (((*(d+2*i))&0xff00)/256-0x11)/8 + ((((*(d+2*i))&0x00ff)%256-
0x08)/8)*26;
               temp3 = (((*(d+2*i))&0xff00)/256-0x10)/8 +
((((*(d+2*i))&0x00ff)%256+0x08)/8)*26;
               temp4 = (((*(d+2*i))&0xff00)/256+0x00)/8 + ((((*(d+2*i))&0x00ff)%256-
0x08)/8)*26;

      for(j=1;j<6;j++){
            if(j!=i && (*(d+2*j+1)&0x01) )
            if ( ((*(d+2*i))&0x00ff ) <= ((*(d+2*j)+0x0011)&0x00ff))
            if ( ((*(d+2*i))&0x00ff ) > ((*(d+2*j)-0x0010)&0x00ff))
           if (    ((*(d+2*i))&0xff00) > ( (*(d+2*j)-0x1000 )&0xff00)    )
           if (    ((*(d+2*i))&0xff00) < ( (*(d+2*j)+0x1000)&0xff00 )    )
                  check1=1;

            if(j!=i && (*(d+2*j+1)&0x01) )
            if ( ((*(d+2*i))&0x00ff ) > ((*(d+2*j)-0x10)&0x00ff))
            if ( ((*(d+2*i))&0x00ff ) < ((*(d+2*j)+0x10)&0x00ff))
```

```
        if (    ((*(d+2*i))&0xff00) > ( (*(d+2*j)-0x1000 )&0xff00)    )
        if (    ((*(d+2*i))&0xff00) <= ( (*(d+2*j)+0x1100)&0xff00 )    )
                check2=1;


          if(j!=i && (*(d+2*j+1)&0x01) )
          if ( ((*(d+2*i))&0x00ff ) >= ((*(d+2*j)-0x11)&0x00ff))
          if ( ((*(d+2*i))&0x00ff ) < ((*(d+2*j)+0x10)&0x00ff))
        if (    ((*(d+2*i))&0xff00) > ( (*(d+2*j)-0x1000 )&0xff00)    )
        if (    ((*(d+2*i))&0xff00) < ( (*(d+2*j)+0x1000)&0xff00 )    )
                check3=1;

           if(j!=i && (*(d+2*j+1)&0x01) )
          if ( ((*(d+2*i))&0x00ff ) > ((*(d+2*j)-0x10)&0x00ff))
          if ( ((*(d+2*i))&0x00ff ) < ((*(d+2*j)+0x10)&0x00ff))
        if (    ((*(d+2*i))&0xff00) >= ( (*(d+2*j)-0x1100 )&0xff00)    )
        if (    ((*(d+2*i))&0xff00) < ( (*(d+2*j)+0x1000)&0xff00 )    )
                check4=1;
    }

       temp_check = *(d+2*i);
        // up
        switch(dir){

        case 0:{

                    *(d+2*i)= (*(d+2*i)+0x0400)&0xf8ff; //adjustment
        for(j=1;j<6;j++){
                if(j!=i && (*(d+2*j+1)&0x01) )
                if ( ((*(d+2*i))&0x00ff ) <= ((*(d+2*j)+0x0011)&0x00ff))
                if ( ((*(d+2*i))&0x00ff ) > ((*(d+2*j)-0x0010)&0x00ff))
           if (    ((*(d+2*i))&0xff00) > ( (*(d+2*j)-0x1000 )&0xff00)    )
           if (    ((*(d+2*i))&0xff00) < ( (*(d+2*j)+0x1000)&0xff00 )    )
                    check1=1;
        }
        if(!check1 &&
           ( (*(c+temp1) == 0||*(c+temp1) ==8||(*(c+temp1) ==2 &&
(((*(d+2*i)&0x00ff)%256-0x09)&0x07) >0x03))
                &&(*(c+temp1+1) == 0||*(c+temp1+1) == 8 || (*(c+temp1+1) ==2 &&
(((*(d+2*i)&0x00ff)%256-0x09)&0x07) >0x03)))
        && (((*(d+2*i))&0x00ff)>8) )      *(d+2*i) = *(d+2*i) -0x01;
```

```
                else {
            *(d+2*i)=temp_check;
            if (i!=1)    *(d+2*i+1)= ((*(d+2*i+1))&0xffcf)+(rand()%4)*16;
            }

          if (i==1) *(d+2*i+1) = (*(d+2*i+1))&0xffcf;    //change direction
          else if((rand()%64)==0) *(d+2*i+1)= ((*(d+2*i+1))&0xffcf)+(rand()%4)*16;

            break;
            }

          case 1:{
           *(d+2*i)= (*(d+2*i)+0x0004)&0xfff8;//adjustment
           for(j=1;j<6;j++){
            if(j!=i && (*(d+2*j+1)&0x01) )
            if ( ((*(d+2*i))&0x00ff ) > ((*(d+2*j)-0x10)&0x00ff))
            if ( ((*(d+2*i))&0x00ff ) < ((*(d+2*j)+0x10)&0x00ff))
            if (    ((*(d+2*i))&0xff00) > ( (*(d+2*j)-0x1000 )&0xff00)    )
            if (    ((*(d+2*i))&0xff00) <= ( (*(d+2*j)+0x1100)&0xff00 )    )
            check2=1;
            }
             if(!check2
             &&( (*(c+temp2) == 0||*(c+temp2) ==8||(*(c+temp2) ==5 &&
(((*(d+2*i)&0xff00)/256-0x11)&0x07) >0x03))
                  &&(*(c+temp2+26) == 0||*(c+temp2+26) == 8 || (*(c+temp2+26) ==5 &&
(((*(d+2*i)&0xff00)/256-0x11)&0x07) >0x03)))
             &&(((*(d+2*i))&0xff00)>0x1000) ) *(d+2*i) = *(d+2*i) - 0x0100;
          else {
            *(d+2*i)=temp_check;
            if (i!=1)    *(d+2*i+1)= ((*(d+2*i+1))&0xffcf)+(rand()%4)*16;
            }


            if (i==1) *(d+2*i+1) = ((*(d+2*i+1))&0xffcf)|0x0010;    //change direction

          else if((rand()%64)==0) *(d+2*i+1)= ((*(d+2*i+1))&0xffcf)+(rand()%4)*16;

          break;}

          case 2:{
```

```
            *(d+2*i)= (*(d+2*i)+0x0400)&0xf8ff;
            for(j=1;j<6;j++){
                  if(j!=i && (*(d+2*j+1)&0x01) )
                  if ( ((*(d+2*i))&0x00ff ) >= ((*(d+2*j)-0x11)&0x00ff))
                  if ( ((*(d+2*i))&0x00ff ) < ((*(d+2*j)+0x10)&0x00ff))
                  if (    ((*(d+2*i))&0xff00) > ( (*(d+2*j)-0x1000 )&0xff00)    )
                  if (    ((*(d+2*i))&0xff00) < ( (*(d+2*j)+0x1000)&0xff00 )    )
                        check3=1;
                  }

            if(!check3
                  && ( (*(c+temp3) == 0||*(c+temp3) ==8||(*(c+temp3) ==4 &&
(((*(d+2*i)&0x00ff)%256+0x08)&0x07) <0x04))
                        &&(*(c+temp3+1) == 0||*(c+temp3+1) == 8 || (*(c+temp3+1) ==4 &&
(((*(d+2*i)&0x00ff)%256+0x08)&0x07) <0x04)))
                  && ((*(d+2*i)&0x00ff)<200) ) *(d+2*i) = *(d+2*i) + 0x01;
            else {
                  *(d+2*i)=temp_check;
                  if (i!=1)    *(d+2*i+1)= ((*(d+2*i+1))&0xffcf)+(rand()%4)*16;

                  }
                  if (i==1)    *(d+2*i+1) = ((*(d+2*i+1))&0xffcf)|0x0020;    //change direction

            else if((rand()%64)==0) *(d+2*i+1)= ((*(d+2*i+1))&0xffcf)+(rand()%4)*16;

                  break;}


           case 3:{
            *(d+2*i)= (*(d+2*i)+0x0004)&0xfff8;
            for(j=1;j<6;j++){
                  if(j!=i && (*(d+2*j+1)&0x01) )
                  if ( ((*(d+2*i))&0x00ff ) > ((*(d+2*j)-0x10)&0x00ff))
                  if ( ((*(d+2*i))&0x00ff ) < ((*(d+2*j)+0x10)&0x00ff))
                  if (    ((*(d+2*i))&0xff00) >= ( (*(d+2*j)-0x1100 )&0xff00)    )
                  if (    ((*(d+2*i))&0xff00) < ( (*(d+2*j)+0x1000)&0xff00 )    )
                        check4=1;
            }
            if(!check4
            && ( (*(c+temp4) == 0||*(c+temp4) ==8||(*(c+temp4) ==3 &&
(((*(d+2*i)&0xff00)/256+0x00)&0x07) <0x04))
```

```
                &&(*(c+temp4+26) == 0||*(c+temp4+26) == 8 || (*(c+temp4+26) ==3 &&
(((*(d+2*i)&0xff00)/256+0x00)&0x07) <0x04)))
            &&(((*(d+2*i))&0xff00)<0xd000) ) *(d+2*i) = *(d+2*i) + 0x0100;


        else {
            *(d+2*i)=temp_check;
            if (i!=1)    *(d+2*i+1)= ((*(d+2*i+1))&0xffcf)+(rand()%4)*16;

            }

            if (i==1)    *(d+2*i+1)= ((*(d+2*i+1))&0xffcf)|0x0030;    //change direction

        else if((rand()%64)==0) *(d+2*i+1)= ((*(d+2*i+1))&0xffcf)+(rand()%4)*16;


            break;}
        }
}

int main()
{
   vga_ball_arg_t vla;

   struct sockaddr_in serv_addr;

   struct usb_keyboard_packet packet;
    char map[][26]={{0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0},
                    {0,0,0,0,1,1,0,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,0,0,0,0},
                    {8,8,1,1,1,1,1,1,0,0,1,1,0,0,0,0,0,0,0,1,1,1,0,0,0,0},
                    {8,8,1,1,1,1,1,1,0,0,1,1,0,0,6,6,0,0,1,1,1,0,0,0,0,0},
                    {8,8,8,8,8,8,8,0,0,0,0,1,1,0,0,1,1,0,0,1,1,0,0,0,1,1,0},
                    {8,8,8,8,8,8,8,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0},

{8,8,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,0,0,11,11},
                {8,8,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,0,0,11,11},
                    {0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                {0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,1,1,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,6,6,6,6},
                    {0,0,0,0,1,1,0,0,0,0,0,0,1,1,1,1,1,0,0,1,1,0,0,0,0,0,0},
                    {1,1,1,1,0,0,1,1,0,0,0,1,1,1,1,1,1,8,8,1,1,0,0,0,0,1,1},
```

```
                    {1,1,1,1,0,0,1,1,0,0,0,1,1,1,1,1,8,8,1,1,6,6,6,6,1,1},
                    {0,0,0,0,0,0,6,6,0,0,0,0,0,0,8,8,8,8,8,8,8,8,0,0,0,0},
                    {0,0,0,0,0,0,6,6,0,0,6,6,0,0,8,8,8,8,8,8,8,8,0,0,0,0},

{11,11,11,11,0,0,11,11,11,11,11,11,11,11,11,11,0,0,11,11,11,11,11,11,11,11},

{11,11,11,11,0,0,11,11,11,11,11,11,11,11,11,11,0,0,11,11,11,11,11,11,11,11},
                    {8,8,8,8,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {8,8,8,8,0,0,0,1,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0},
                    {8,8,8,8,1,1,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,0},
                    {8,8,8,8,1,1,0,0,1,0,0,0,0,0,0,1,0,0,6,6,1,1,1,1,0,0},
                    {8,8,0,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0},
                    {8,8,6,6,1,1,0,0,1,0,0,1,1,1,1,0,0,0,0,0,0,0,1,1,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,1,12,13,1,0,0,0,0,0,0,0,1,1,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,1,14,15,1,0,0,0,1,1,0,0,0,0,0,0}};


    char map2[][26]={{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,1,0,0,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0},
                    {0,0,1,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0},
                    {0,0,1,0,0,0,0,0,1,0,1,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0},
                    {0,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0},
                {0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0,1,1,1,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,1,0,1,1,1,1,1,0,1,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,1,0,1,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,1,1,1,0,0,0,1,1,1,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,1,1,1,0,1,0,1,1,1,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,1,1,1,0,0,0,1,1,1,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,1,1,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,1,0,1,1,1,1,1,0,1,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0},
```

```
                    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};

char map3[][26]={{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,1,0,1,0,1,0,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0,1,0,1,0},
                    {0,1,0,1,0,1,0,1,0,0,0,1,0,0,1,0,1,0,1,0,1,0,1,0,1,0},
                    {0,1,0,1,0,1,0,1,0,0,0,1,0,0,1,0,1,0,1,1,1,0,1,1,1,0},
                    {0,1,0,1,0,1,0,1,0,0,0,1,0,0,1,0,1,0,1,1,0,0,0,1,0,0},
                    {0,0,1,0,0,1,0,1,1,0,0,1,0,0,1,1,1,0,1,0,1,0,0,1,0,0},
                {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
                {0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,0,1,1,0,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,1,0,0,1,1,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,1,1,1,0,0,1,0,0,1,0,0,1,0,1,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0,1,1,1,0,0,1,0,0,1,0,0,1,0,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,1,0,0,1,0,0,1,0,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,1,0,0,1,0,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,1,1,1,0,0,1,0,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0,1,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
                    {0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0},
                    {0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0}};


int transferred;
char keystate[12];

int i;
char flag_enemy1=0;
char flag=0;
char flag2=0;
char flag4 = 0;
char flag8 =0;
char flag16 =0;
```

```
    char flag32 = 0;

    char game = 0x00;

    char flag_count=0x00;

    char tank_num1 = 16;      // total tank
    char tank_num2 = 0;       //current tank number
    char tank_ku[] = {0x42,0x52,0x62,0x72,0x42,0x52,0x62,0x72,
                      0x42,0x52,0x62,0x72,0x42,0x52,0x62,0x72};

    char life =0x03;

    char bullet1[]={0x00,0x00,0x00,0x00};          // x,y,direction,en;
    char bullet2[]={0x00,0x00,0x00,0x00};          // x,y,direction,en;
    char bullet3[]={0x00,0x00,0x00,0x00};          // x,y,direction,en;
    char bullet4[]={0x00,0x00,0x00,0x00};          // x,y,direction,en;
    char bullet5[]={0x00,0x00,0x00,0x00};          // x,y,direction,en;

    short table[]={0x0000,0x0000,0x50c8,0x0001,0xa008,0x5200,0x1008,0x4200,
                           0x1038,0x7100,0x7008,0x4200,0x0000,0x0000,0x0000,0x0000,
                           0x0000,0x0000,0x0000};

    static const char filename[] = "/dev/vga_ball";
    printf("VGA ball Userspace program started\n");
    if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
        fprintf(stderr, "could not open %s\n",filename);
        return -1;
    }

    printf("initial state: ");
    print_background_color();

// clear map
for(int j=0;j<26;j++)
{
     for(int i=0;i<32;i++)
{    table[0] = i*2048+j*64+1;
     set_background_color(&table);
}
}
```

```
    table[0] = table[0]-1;
    set_background_color(&table);
     usleep(40000);

// load one map
for(int j=0;j<26;j++)
{
    for(int i=0;i<26;i++)
{

    table[0] = i*2048+j*64+map[j][i]*2+1;
    set_background_color(&table);
 //   print_background_color();
}
     usleep(10000);
}

    table[0] = table[0]-1;
    set_background_color(&table);

// load other item
for(int j=1;j<9;j++)
{
    for(int i=27;i<29;i++)
{
    table[0] = i*2048+j*64+16*2+1;
    set_background_color(&table);
}
}

    table[0] = 27*2048+17*64+17*2+1;
    set_background_color(&table);
    table[0] = 28*2048+17*64+21*2+1;
    set_background_color(&table);
    table[0] = table[0]-1;
    set_background_color(&table);

// initail tank

    table[12] = 0x3030;
    table[17] = 0x0000;
```

```
    set_background_color(&table);
    print_background_color();

   /* Open the keyboard */
   if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
      fprintf(stderr, "Did not find a keyboard\n");
      exit(1);
   }

  /* Look for and handle keypresses */
   for (;;) {
      libusb_interrupt_transfer(keyboard, endpoint_address,
                        (unsigned char *) &packet, sizeof(packet),
                        &transferred, 0);
      if (transferred == sizeof(packet)) {
      //    sprintf(keystate, "%02x %02x %02x %02x %02x %02x %02x %02x",
packet.keycode[0],
//              packet.keycode[1],packet.keycode[2],packet.keycode[3],
      //                  packet.keycode[4],packet.keycode[5],packet.keycode[6],packet.keycode[7]);
      //    printf("%s\n", keystate);
      flag = ~flag;
            flag_count=flag_count+1;
            if(flag) flag2 = ~flag2;
            if(flag2) flag4 = ~flag4;
            if(flag4) flag8 = ~flag8;
            if(flag8) flag16 = ~flag16;
            if(flag16) flag32 = ~flag32;



      if(flag){

                              table[18]=table[18]&0xfffc;


//*********************************************************************
        //******************** detect key press *********************************

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

```
    // up,down,left,right
    if (packet.keycode[4] == 0x00)
{
            rote(1,0,map,table);
}

    else if (packet.keycode[4] == 0xff)
{
            rote(1,2,map,table);
}

    else if (packet.keycode[3] == 0x00)
{
        rote(1,1,map,table);
}

    else if (packet.keycode[3] == 0xff)
{
        rote(1,3,map,table);
}

// press b get new player tank
if (packet.keycode[5] == 0x4f)
{
      if ((table[3]&0x01)==0x00 ) {
            table[2]=0x50c8;
                  table[3]=table[3]|0x01;
             life = life -1;
      switch (life) {
      case 2: {table[0] = 28*2048+17*64+20*2+1; break;}
      case 1: {table[0] = 28*2048+17*64+19*2+1;break;}
      case 0: {table[0] = 28*2048+17*64+18*2+1;break;}
      }
         set_background_color(&table);
      }
}

// press a
  if (packet.keycode[5] == 0x2f)
{
      if((table[3]&0x01)==0x01 && bullet1[3]==0x00)
```

```
                    {
                        bullet1[0] = table[2]/256+6;
                        bullet1[1] = table[2]%256+6;
                        bullet1[2] = (table[3]/16)&0x03;
                        bullet1[3] =    0x01;
                table[18]=table[18]&0xfffc|0x0001;

                     }
    }


//*************************************************************************
        //********************** end key press *********************************

//*************************************************************************
    if( ((table[3]&0xf000)==0x2000 )|| ( (table[3]&0xf000)==0x3000))
bullet_cal(bullet1,table,map,1);
    bullet_cal(bullet1,table,map,1);
          table[12] = bullet1[0]*256+bullet1[1];
          table[17] = (table[17]&0xfffe)|(bullet1[3]&0x01);


//************************* movement of tank ****************************

   //enemy tank1
    if ((table[5]&0x01)==0x01&&flag2){

              rote(2,(table[5]&0x0030)/16,map,table);

    if((table[5]&0xf000)==0x5000)     rote(2,(table[5]&0x0030)/16,map,table);

         if(bullet2[3]==0x00 && !(rand()%32))
                  {
                      bullet2[0] = table[4]/256+6;
                      bullet2[1] = table[4]%256+6;
                      bullet2[2] = (table[5]/16)&0x03;
                      bullet2[3] =    0x01;


                    }
    }
```

```
if((table[5]&0xf000)==0x6000)      bullet_cal(bullet2,table,map,0);

bullet_cal(bullet2,table,map,0);
table[13] = bullet2[0]*256+bullet2[1];
table[17] = (table[17]&0xfffd)|((bullet2[3]&0x01)*0x02);


//enemy tank2
      if ((table[7]&0x01)==0x01&&flag2){

if((table[7]&0xf000)==0x5000) rote(3,(table[7]&0x0030)/16,map,table);
            rote(3,(table[7]&0x0030)/16,map,table);
      if(bullet3[3]==0x00 && !(rand()%32))
                  {
                        bullet3[0] = table[6]/256+6;
                        bullet3[1] = table[6]%256+6;
                        bullet3[2] = (table[7]/16)&0x03;
                        bullet3[3] =    0x01;


                  }
}
      if((table[7]&0xf000)==0x6000)      bullet_cal(bullet3,table,map,0);
bullet_cal(bullet3,table,map,0);
table[14] = bullet3[0]*256+bullet3[1];
table[17] = (table[17]&0xfffb)|((bullet3[3]&0x01)*0x04);


     //enemy tank3
      if ((table[9]&0x01)==0x01&&flag2){
      if((table[9]&0xf000)==0x5000)      rote(4,(table[9]&0x0030)/16,map,table);
            rote(4,(table[9]&0x0030)/16,map,table);
      if(bullet4[3]==0x00&& !(rand()%32))
                  {
                        bullet4[0] = table[8]/256+6;
                        bullet4[1] = table[8]%256+6;
                        bullet4[2] = (table[9]/16)&0x03;
                        bullet4[3] =    0x01;


                  }
```

```
    }

        if((table[9]&0xf000)==0x6000)     bullet_cal(bullet4,table,map,0);
        bullet_cal(bullet4,table,map,0);
        table[15] = bullet4[0]*256+bullet4[1];
        table[17] = (table[17]&0xfff7)|((bullet4[3]&0x01)*0x08);


      //enemy tank4
        if ((table[11]&0x01)==0x01&&flag2){
if((table[11]&0xf000)==0x5000)    rote(5,(table[11]&0x0030)/16,map,table);
            rote(5,(table[11]&0x0030)/16,map,table);
        if(bullet5[3]==0x00 && !(rand()%32))
                {
                        bullet5[0] = table[10]/256+6;
                        bullet5[1] = table[10]%256+6;
                        bullet5[2] = (table[11]/16)&0x03;
                        bullet5[3] =    0x01;


                    }
        }

        if((table[11]&0xf000)==0x6000)     bullet_cal(bullet5,table,map,0);
    bullet_cal(bullet5,table,map,0);
        table[16] = bullet5[0]*256+bullet5[1];
        table[17] = (table[17]&0xffef)|((bullet5[3]&0x01)*0x10);


        set_background_color(&table);

if( ((flag_count%128)==1) && tank_num1){

     for(i=0;i<4;i++)
     if ((table[5+2*i]&0x01)==0x00)
     {
        switch (tank_num1%3)
        {
        case 0: {table[4+2*i]=0x7008;break;}
        case 1: {table[4+2*i]=0x1008;break;}
        case 2:    {table[4+2*i]=0xd008;break;}
```

```
            }
            table[5+2*i] = tank_ku[16-tank_num1]*256+0x21;
            tank_num1 = tank_num1 -1;
            if (tank_num1 >=8) {table[0] = 28*2048+(tank_num1-7)*64+1;}
            else    { table[0] = 27*2048+(tank_num1+1)*64+1;}

            switch(tank_num1) {
            case 10: {table[3] = table[3]&0x0fff|0x1000;break;}
            case 6: {table[3] = table[3]&0x0fff|0x2000;break;}
            case 1: {table[3] = table[3]&0x0fff|0x3000;break;}
            }
            break;
      }
}

      set_background_color(&table);
            table[0] = 0x0000;
      set_background_color(&table);



    if ((table[18]&0xf000)||life==0x00) {game = 1;break;}    // game over

    if (    (table[5]&0x01)==0x00 && (table[7]&0x01)==0x00 && (table[9]&0x01)==0x00
              &&(table[11]&0x01)==0x00      && tank_num1==0x00) {game = 2;break;} //win

    if (packet.keycode[6] == 0x20) { /* ESC pressed? */
      break;

         }
      }
   }
}


usleep(100000);

for (i=0;i<18;i++)
{
      table[i] = 0x00;
}
```

```
// clear map
for(int j=0;j<26;j++)
{
    for(int i=0;i<32;i++)
{   table[0] = i*2048+j*64+1;
    set_background_color(&table);
}
}
    table[0] = table[0]-1;
    set_background_color(&table);
     usleep(40000);

if(game==1) {
// load one map
for(int j=0;j<26;j++)
{
    for(int i=0;i<26;i++)
{

    table[0] = i*2048+j*64+map2[j][i]*2*6+1;
    set_background_color(&table);
}
     usleep(10000);
}

    table[0] = table[0]-1;
    set_background_color(&table);
}

else {
// load one map
for(int j=0;j<26;j++)
{
    for(int i=0;i<26;i++)
{

    table[0] = i*2048+j*64+map3[j][i]*2*6+1;
    set_background_color(&table);
}
     usleep(10000);
```

```
}

    table[0] = table[0]-1;
    set_background_color(&table);
}
    table[18] = 0x0000;
    set_background_color(&table);
  usleep(100000);

  printf("VGA BALL Userspace program terminating\n");
  return 0;
}
```

3. vga_ball.c

```
/* * Device driver for the VGA video generator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *                          drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_ball.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_ball.c
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
```

```
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_ball.h"

#define DRIVER_NAME "vga_ball"

/* Device registers */
#define BG_RED(x) (x)
#define BG_GREEN(x) ((x)+2)
#define TANK1_P(x) ((x)+4)
#define TANK1_D(x) ((x)+6)
#define ENEMY1_P(x) ((x)+8)
#define ENEMY1_D(x) ((x)+10)
#define ENEMY2_P(x) ((x)+12)
#define ENEMY2_D(x) ((x)+14)
#define ENEMY3_P(x) ((x)+16)
#define ENEMY3_D(x) ((x)+18)
#define ENEMY4_P(x) ((x)+20)
#define ENEMY4_D(x) ((x)+22)
#define BULLET1(x) ((x)+24)
#define BULLET2(x) ((x)+26)
#define BULLET3(x) ((x)+28)
#define BULLET4(x) ((x)+30)
#define BULLET5(x) ((x)+32)
#define BULLET_D(x) ((x)+34)
#define GAME_D(x) ((x)+36)
/*
 * Information about our device
 */
struct vga_ball_dev {
	struct resource res; /* Resource: our registers */
	void __iomem *virtbase; /* Where registers can be accessed in memory */
        vga_ball_color_t background;
} dev;
```

```
/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_background(vga_ball_color_t *background)
{
	iowrite16(background->wall1, BG_RED(dev.virtbase) );	//display wall
	iowrite16(background->wall2, BG_GREEN(dev.virtbase) );	//display wall
	iowrite16(background->tank1_p, TANK1_P(dev.virtbase) );
	iowrite16(background->tank1_d, TANK1_D(dev.virtbase) ); //display player tank
	iowrite16(background->enemy1_p, ENEMY1_P(dev.virtbase) );
	iowrite16(background->enemy1_d, ENEMY1_D(dev.virtbase) ); //display player tank
	iowrite16(background->enemy2_p, ENEMY2_P(dev.virtbase) );
	iowrite16(background->enemy2_d, ENEMY2_D(dev.virtbase) ); //display player tank
	iowrite16(background->enemy3_p, ENEMY3_P(dev.virtbase) );
	iowrite16(background->enemy3_d, ENEMY3_D(dev.virtbase) ); //display player tank
	iowrite16(background->enemy4_p, ENEMY4_P(dev.virtbase) );
	iowrite16(background->enemy4_d, ENEMY4_D(dev.virtbase) ); //display player tank
		iowrite16(background->bullet1_p, BULLET1(dev.virtbase) );
		iowrite16(background->bullet2_p, BULLET2(dev.virtbase) );
		iowrite16(background->bullet3_p, BULLET3(dev.virtbase) );
		iowrite16(background->bullet4_p, BULLET4(dev.virtbase) );
		iowrite16(background->bullet5_p, BULLET5(dev.virtbase) );
		iowrite16(background->bullet_data, BULLET_D(dev.virtbase) ); //display bullet
	iowrite16(background->game_data, GAME_D(dev.virtbase) ); //music


	//iowrite16(0x7320, TANK1_D(dev.virtbase) );
	dev.background = *background;
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static long vga_ball_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
	vga_ball_arg_t vla;

	switch (cmd) {
```

```
	case VGA_BALL_WRITE_BACKGROUND:
		if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
				   sizeof(vga_ball_arg_t)))
			return -EACCES;
		write_background(&vla.background);
		break;

	case VGA_BALL_READ_BACKGROUND:
		vla.background = dev.background;
		if (copy_to_user((vga_ball_arg_t *) arg, &vla,
				 sizeof(vga_ball_arg_t)))
			return -EACCES;
		break;

	default:
		return -EINVAL;
	}

	return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_ball_fops = {
	.owner		= THIS_MODULE,
	.unlocked_ioctl = vga_ball_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice vga_ball_misc_device = {
	.minor		= MISC_DYNAMIC_MINOR,
	.name		= DRIVER_NAME,
	.fops		= &vga_ball_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev)
{
	vga_ball_color_t beige = { 0xf9, 0xe4, 0xb7 };
```

```
    int ret;

    /* Register ourselves as a misc device: creates /dev/vga_ball */
    ret = misc_register(&vga_ball_misc_device);

    /* Get the address of our registers from the device tree */
    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
    if (ret) {
        ret = -ENOENT;
        goto out_deregister;
    }

    /* Make sure we can use these registers */
    if (request_mem_region(dev.res.start, resource_size(&dev.res),
                           DRIVER_NAME) == NULL) {
        ret = -EBUSY;
        goto out_deregister;
    }

    /* Arrange access to our registers */
    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
    if (dev.virtbase == NULL) {
        ret = -ENOMEM;
        goto out_release_mem_region;
    }

    /* Set an initial color */
        write_background(&beige);

    return 0;

out_release_mem_region:
    release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
    misc_deregister(&vga_ball_misc_device);
    return ret;
}

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev)
{
```

```
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&vga_ball_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_ball_of_match[] = {
    { .compatible = "csee4840,vga_ball-1.0" },
    {},
};
MODULE_DEVICE_TABLE(of, vga_ball_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_ball_driver = {
    .driver    = {
        .name     = DRIVER_NAME,
        .owner    = THIS_MODULE,
        .of_match_table = of_match_ptr(vga_ball_of_match),
    },
    .remove  = __exit_p(vga_ball_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_ball_init(void)
{
    pr_info(DRIVER_NAME ": init\n");
    return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
}

/* Calball when the module is unloaded: release resources */
static void __exit vga_ball_exit(void)
{
    platform_driver_unregister(&vga_ball_driver);
    pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_ball_init);
module_exit(vga_ball_exit);
```

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA ball driver");
```

4. vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

typedef struct {
      unsigned short wall1,
wall2,tank1_p,tank1_d,enemy1_p,enemy1_d,enemy2_p,enemy2_d,enemy3_p,enemy3_d,enemy4
_p,enemy4_d,bullet1_p,bullet2_p,
                                    bullet3_p,bullet4_p,bullet5_p,bullet_data,game_data;
} vga_ball_color_t;


typedef struct {
   vga_ball_color_t background;
} vga_ball_arg_t;

#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1,
vga_ball_arg_t *)
#define VGA_BALL_READ_BACKGROUND    _IOR(VGA_BALL_MAGIC, 2,
vga_ball_arg_t *)

#endif
```

5. Makefile

```
ifneq (${KERNELRELEASE},)
# KERNELRELEASE defined: we are being compiled as part of the Kernel
            obj-m := vga_ball.o

else
```

```
# We are being compiled as a module: use the Kernel build system
	KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
		PWD := $(shell pwd)
#CFLAGS = -Wall
default: module hello
hello: hello.o usbkeyboard.o
	cc $(CFLAGS) -o hello hello.o usbkeyboard.o -lusb-1.0 -pthread -lm
hello.o: hello.c usbkeyboard.h
usbkeyboard.o: usbkeyboard.c usbkeyboard.h
module:
	${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules
clean:
	${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
	${RM} hello
TARFILES = Makefile README vga_ball.h vga_ball.c hello.c usbkeyboard.h usbkeyboard.c
TARFILE = lab3-sw.tar.gz
.PHONY : tar
tar : $(TARFILE)

$(TARFILE) : $(TARFILES)
	tar zcfC $(TARFILE) .. $(TARFILES:%=lab3-sw/%)

endif
```