Battle City Game

Final Report

CSEE4840 Embedded System Design

Under the guidance of Professor Stephen Edwards

Li Qi(lq2156)

Contents

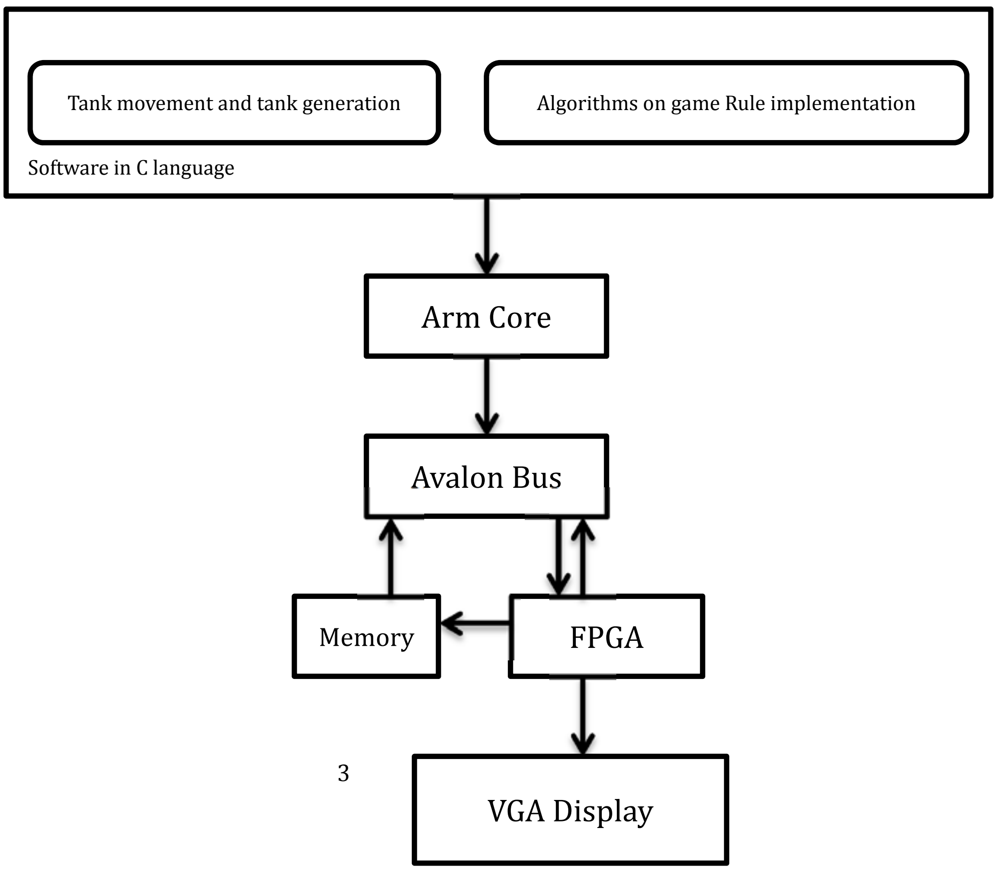
1.Overview

In this project I design a tank-shooting game with simple rules. And in the hardware part I use the system Verilog and Verilog both through Altera Cyclone IV FPGA board. As for software part, I use C programming to regulate the rules and make the game running. Users should be able to play this shooting game using keyboard as controller and monitor for display. The main challenges are using Verilog HDL for functions of configuring VGA and using C language for tank movements and applications of game rules.

1.1 Game Rules

To win this game, user need to destroy all enemy tanks and their base, which is surrounded by the brick walls. And if user's tank get destroyed three times or the user's base is destroyed, then the game will end. There are brick walls and rivers randomly located in the battlefield. The brick wall unit, which is shown as a white square, can be destroyed by one bullet shot. And the river unit, which is shown as a blue square, will block the movement of tanks.

1.2 Structure Description


Tank movement and tank generation
Algorithms on game Rule implementation
Software in C language
Arm Core
Avalon Bus
Memory
FPGA
VGA Display

In the software part I use C-program language to randomly generate two enemy tanks in the battlefield and depict user's tank movement by using the USBkeyboard. I also use C language to regulate the winning and losing game rules by algorithms.

Arm core and Avalon bus are the bridges connecting the software in C language and hardware designed in Verilog. Also the hardware contains VGA display that would generate everything belongs to the game window.

2.Hardware

2.1 keyboard

Keyboard is used to control the total game process; all the functions of the game should be realized with the use of keyboard. These functions include systematic functions such as pause, resume, start and exit; as well as gaming functions such as going forward, going back, making turns and fire. The functions of the keyboard are designed both in Verilog HDL and C language. In this design, we use following keys on keyboard and their functions are also listed below:

Keys	Functions
W,A,S,D	**Move tank up,left,down,right**
↑,↓	**Make selection between start and exit**
Space	**Tank fires**
ESC	**Pause& Resume the game**

To make the keyboard working correctly, we also need a driver document. And part of the codes of driver are listed below:

```
  /*
 * ===    FUNCTION    ======================================================================
 *    Name:    ps2_command
 *    Description:
 * =========================================================================================
 */
void ps2_command(alt_u8 cmd)
{
        alt_u16 i, j;
        IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_CLK_BASE, OUT);    //clk and data direction is output
        IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_DAT_BASE, OUT)
        IOWR_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE, 0);       //drive clk and data low
        IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 0);
        usleep(200);
        IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_CLK_BASE, IN); //Release the Clock line
```

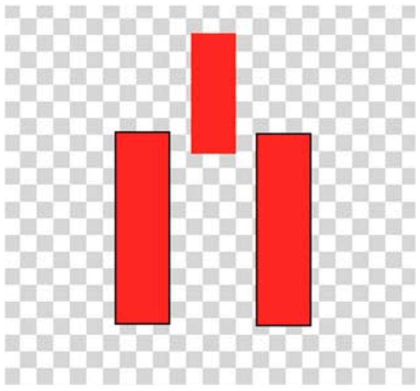
```
    usleep(10);
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01); //Wait for the device to bring the Clock line low.
    for(i=0,j=0; i<8; i++)
    {
        if(cmd & 0x01)              //Set/reset the Data line to send the first data bit.
        {
            IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 1);
            j++;
        }
        else
            IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 0);;
        cmd >>= 1;
        while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01));    //Wait for the device to bring Clock high.
        usleep(20);
        while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01);      //Wait for the device to bring Clock low.
        usleep(20);
    }
    if(!(j&0x01))
        IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 1);    //Send parity bit
    else
        IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 0);
    usleep(20);
    while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01));   //Wait for the device to bring Clock high.
    usleep(20);
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01);      //Wait for the device to bring Clock low.
    IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 1);                 //Send stop bit
    usleep(20);
    while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01));   //Wait for the device to bring Clock high.
    usleep(20);
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01);      //Wait for the device to bring Clock low.
    IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_DAT_BASE, IN);;          //Release the Data line.
    usleep(20);
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE) & 0x01);      //Wait for the device to bring Data low.
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01);      //Wait for the device to bring Clock low.
    usleep(20);
    while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01));   //Wait for the device to bring Clock high.
    while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE) & 0x01));   //Wait for the device to bring Data high.
}
```

2.2 VGA Block

VGA is hardware designed in Verilog to correctly connect the FPGA board with functions loaded to the monitor, as well as to display everything in the gaming window. VGA display would interact with memory, which consists of RAM and decoder to generate display at the current moment with respect to the display stored as the previous game window. This is the core part in terms of game display.

2.2.1 Tank

Here in this game I use a simple way to represent tanks----they are all represented by three lines as shown below.

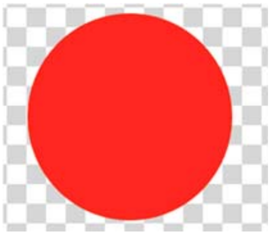


To present this image I use these main functions:

```
/*****************************************************************
*   Function: vid_draw_box
*   Purpose: Draws a box on the screen with the specified corner
*   points.   The fill parameter tells the function whether or not
*   to fill in the box.   1 = fill, 0 = do not fill.
*****************************************************************/
int vid_draw_box (int horiz_start, int vert_start, int horiz_end, int vert_end, int color, int fill, display_frame_buffer_struct*
frame_buffer)
{
  // If we want to fill in our box
  if (fill) {
       vid_paint_block (horiz_start, vert_start, horiz_end, vert_end, color, frame_buffer);
  // If we're not filling in the box, just draw four lines.
  } else {
     vid_draw_line(horiz_start, vert_start, horiz_start, vert_end-1, 1, color, frame_buffer);
     vid_draw_line(horiz_end-1, vert_start, horiz_end-1, vert_end-1, 1, color, frame_buffer);
     vid_draw_line(horiz_start, vert_start, horiz_end-1, vert_start, 1, color, frame_buffer);
     vid_draw_line(horiz_start, vert_end-1, horiz_end-1, vert_end-1, 1, color, frame_buffer);
  }
  return (0);
}
/*****************************************************************
*   Function: vid_move_block
*    Purpose: Moves a block around the screen, backfilling with
*    the backfill_color parameter.
*****************************************************************/
int vid_move_block(int xbegin, int ybegin, int xend, int yend, int x_distance, int y_distance, int backfill_color,
display_frame_buffer_struct* frame_buffer)
{
  int read_x, read_y, write_x, write_y;
  short temp_pixel;
  if(x_distance <= 0 && y_distance <= 0) {
    //Move by rows because they are contiguous in memory (could help speed if in SDRAM)
    for (read_y = ybegin; read_y < yend; read_y++) {
      write_y = read_y + y_distance;
      for(read_x = xbegin; read_x < xend; read_x++) {
        write_x = read_x + x_distance;
        temp_pixel = vid_get_pixel(read_x, read_y, frame_buffer);
        vid_set_pixel(write_x, write_y, temp_pixel, frame_buffer);
        if(read_x >= xend + x_distance || read_y >= yend + y_distance)
        {
              vid_set_pixel(read_x, read_y, backfill_color, frame_buffer);
        }
      }
    }
  }
  return (0);
}
```

2.2.2 Base

In this game I use two circles to represent the home base and enemy base.



The main function is drawing a circle. The codes are listed below:

```
/*****************************************************************
*    Function: vid_draw_circle
*    Purpose: Draws a circle on the screen with the specified center
*    and radius.    Draws symetric circles only.    The fill parameter
*    tells the function whether or not to fill in the box.    1 = fill,
*    0 = do not fill.
```

```
*****************************************************************/
int vid_draw_circle(int Hcenter, int Vcenter, int radius, int color, char fill, display_frame_buffer_struct* frame_buffer)
{
   int x = 0;
   int y = radius;
   int p = (5 - radius*4)/4;
   // Start the circle with the top, bottom, left, and right pixels.
   vid_circle_points(Hcenter, Vcenter, x, y, color, fill, frame_buffer);
   // Now start moving out from those points until the lines meet
   while (x < y) {
      x++;
      if (p < 0) {
         p += 2*x+1;
      } else {
         y--;
         p += 2*(x-y)+1;
      }
      vid_circle_points(Hcenter, Vcenter, x, y, color, fill, frame_buffer);
   }
   return (0);
}
int vid_draw_circle_ex(int Hcenter, int Vcenter, int radius, int color, char fill, display_frame_buffer_struct* frame_buffer,
unsigned char mode)
{
      int x = 0;
      int y = radius;
      int p = (5 - radius*4)/4;
      // Start the circle with the top, bottom, left, and right pixels.
      vid_circle_points_ex(Hcenter, Vcenter, x, y, color, fill, frame_buffer, mode);
      // Now start moving out from those points until the lines meet
      while (x < y)
      {
            x++;
            if (p < 0)
            {
                  p += 2*x+1;
            }
            else
            {
                  y--;
                  p += 2*(x-y)+1;
            }
            vid_circle_points_ex(Hcenter, Vcenter, x, y, color, fill, frame_buffer, mode);
      }
```

```
    return (0);
}
```

2.2.3 Brick wall& River

I use white squares to represent the brick wall. These walls can be destroyed by one shot from tanks. And I use blue squares to represent the river. The river can block the movement from the tanks.

Here I use the box function and the codes are listed below:

```
/****************************************************************
*   Function: vid_draw_box
*   Purpose: Draws a box on the screen with the specified corner
*   points.   The fill parameter tells the function whether or not
*   to fill in the box.   1 = fill, 0 = do not fill.
*
****************************************************************/
int vid_draw_box (int horiz_start, int vert_start, int horiz_end, int vert_end, int color, int fill, display_frame_buffer_struct*
frame_buffer)
{
  // If we want to fill in our box
  if (fill) {
     vid_paint_block (horiz_start, vert_start, horiz_end, vert_end, color, frame_buffer);
  // If we're not filling in the box, just draw four lines.
  } else {
    vid_draw_line(horiz_start, vert_start, horiz_start, vert_end-1, 1, color, frame_buffer);
    vid_draw_line(horiz_end-1, vert_start, horiz_end-1, vert_end-1, 1, color, frame_buffer);
    vid_draw_line(horiz_start, vert_start, horiz_end-1, vert_start, 1, color, frame_buffer);
    vid_draw_line(horiz_start, vert_end-1, horiz_end-1, vert_end-1, 1, color, frame_buffer);
  }
  return (0);
}
```

2.3 Audio Block

In this game I use the buzzer to simulate the audio I need in the game. So the system will output a high-voltage signal to make the buzzer sound if tank shoot, hit or move. And in other situations system will output a low-voltage signal to make the buzzer quiet.

The codes of buzzer are listed below:

```
module BUZZER (
              // inputs:
               address,
               chipselect,
               clk,
               reset_n,
               write_n,
                    writedata,
                   // outputs:
                    out_port,
                    readdata
                 )
  output            out_port;
  output            readdata;
  input     [   1: 0] address;
  input             chipselect;
  input             clk;
  input             reset_n;
  input             write_n;
  input             writedata;
  wire              clk_en;
  reg               data_out;
  wire              out_port;
  wire              read_mux_out;
  wire              readdata;
  assign clk_en = 1;
  //s1, which is an e_avalon_slave
  assign read_mux_out = {1 {(address == 0)}} & data_out;
  always @(posedge clk or negedge reset_n)
    begin
      if (reset_n == 0)
          data_out <= 0;
      else if (chipselect && ~write_n && (address == 0))
          data_out <= writedata;
    end
  assign readdata = read_mux_out;
  assign out_port = data_out;
endmodule
```

3. Software

3.1 Tank movement

There are two parts in tank movement—the user’s tank movement and enemy tanks movement. Enemy tanks are moving randomly across the battlefield. In the code I add a random number to realize this random movement.
This part of codes are listed below:

```
void enemy_control(int map[15][15],display_frame_buffer_struct* vga_frame_buffer)/*the control of enemy movement*/
{
    int i;
    for(i=0;i<3;i++)
        {
            enemy[i].control=rand()%50;/*random number*/
            if(enemy[i].life==1)
                {
                    if(enemy[i].control>=0&&enemy[i].control<=9&&map[enemy[i].j-1][enemy[i].i]==0)/*enemy
moves up*/
                        {
                            enemy[i].way=UPWAY;
                            map[enemy[i].j][enemy[i].i]=0;
                            blank(enemy[i].i,enemy[i].j,vga_frame_buffer);
                            enemy[i].j--;
                            uptank(enemy[i].i,enemy[i].j,RED_16,vga_frame_buffer);
                            map[enemy[i].j][enemy[i].i]=44+i;
                        }
```

As for user's tank, it is totally controlled by the keyboard. The part of the codes are listed below:

```
void judge_tank_my(int map[15][15],int key,display_frame_buffer_struct* vga_frame_buffer)/*the judgement of user
tank's movement*/
{
    switch(key)/*the keyboard button pressed*/
        {
        case a_UP:/*the W is pressed*/
            if(player[0].life==1){
            if(map[player[0].j-1][player[0].i]==0)/*if there is no block toward the tank*/
                {
                    if(player[0].way==UPWAY)/*if the tank is facing upwards*/
                        {
                            map[player[0].j][player[0].i]=0;/*emerge a new tank in a new place and wipe out the
former tank*/
                            blank(player[0].i,player[0].j,vga_frame_buffer);
                            player[0].j--;
                            uptank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);
                            map[player[0].j][player[0].i]=1;
                            player[0].way=UPWAY;
                        }
                    else /*if the tank doesn't face upward*/
                        {
                            blank(player[0].i,player[0].j,vga_frame_buffer);/*wipe the former tank*/
                            uptank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);/*emerge a new tank in the
same place which is facing upward*/
                            player[0].way=UPWAY;/*the tank's direction is up*/
                        }
                }
            else /*if there is a block */
                {
```

```
                blank(player[0].i,player[0].j,vga_frame_buffer);
                uptank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);player[0].way=UPWAY;
            }
                    fengming(100);      //the sound of tank movement
        }
        break;
```

3.2 Bullet movement

There are also two parts concerning bullet movement. First is the function of the bullet moving directions and the other function is about the all kinds of situation while a bullet is moving, such as the bullet is arriving at the edge of the battlefield or the bullet has hit the tank.

The part of the codes concerning the direction of the bullet are shown below:

```
void judge_shootway(int map[15][15],int i,display_frame_buffer_struct* vga_frame_buffer)/*function of judging the direction of the bullet*/
{
    switch(shoot[i].way)
        {
            case
UPWAY:map[(shoot[i].y-15)/30-1][(shoot[i].x-15)/30]=0;blank((shoot[i].x-15)/30,(shoot[i].y-15)/30-1,vga_frame_buffer);break;
            case
DOWNWAY:map[(shoot[i].y-15)/30+1][(shoot[i].x-15)/30]=0;blank((shoot[i].x-15)/30,(shoot[i].y-15)/30+1,vga_frame_buffer);break;
            case
LEFTWAY:map[(shoot[i].y-15)/30][(shoot[i].x-15)/30-1]=0;blank((shoot[i].x-15)/30-1,(shoot[i].y-15)/30,vga_frame_buffer);break;
            case
RIGHTWAY:map[(shoot[i].y-15)/30][(shoot[i].x-15)/30+1]=0;blank((shoot[i].x-15)/30+1,(shoot[i].y-15)/30,vga_frame_buffer);break;
        }
}
```

And the following codes is about judging all situations while the bullet is moving.

```
  void judge_moveshootway(int map[15][15],int i,display_frame_buffer_struct* vga_frame_buffer)/*the function of the bullet's direction while the bullet is moving*/
{
    int shoot_color,per=9;
    if(i<1) shoot_color=GREEN_16;
    else    shoot_color=RED_16;
    switch(shoot[i].way)
        {
            case UPWAY :/*the bullet is moving upward*/
                if((shoot[i].y-15)%30==0)/*judging if the bullet has reached at the edge of the map*/
```

```
                    if(map[(shoot[i].y-15)/30-1][(shoot[i].x-15)/30]!=0)/*judging if there is any units in front of the
bullet*/
                        {
                            shoot[i].life=0;/*the bullet is destroyed*/

judge_shoot(map[(shoot[i].y-15)/30-1][(shoot[i].x-15)/30],map,i,vga_frame_buffer);/*use the function of the bullet
hitting unit*/
                            fengming(5000);     //the sound of hitting
                        }
                if(shoot[i].y<47+5+per)
                    shoot[i].life=0;
                if(shoot[i].life!=0)
                    {
//                          setfillstyle(1,1);/*picture the bullet*/
//                          bar(shoot[i].x+14,shoot[i].y-5,shoot[i].x+17,shoot[i].y-2);
//                          setcolor(7);
//                          line(shoot[i].x+14,shoot[i].y-1,shoot[i].x+17,shoot[i].y-1);
                     vid_draw_box(shoot[i].x+14,shoot[i].y-4,shoot[i].x+17,shoot[i].y-1,     BLACK_16,     DO_FILL,
vga_frame_buffer);
                            vga_flip_frame_buffers( vga_frame_buffer );
                            while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                    vid_draw_box(shoot[i].x+14,shoot[i].y-4,shoot[i].x+17,shoot[i].y-1,     BLACK_16,     DO_FILL,
vga_frame_buffer);
                            vga_flip_frame_buffers( vga_frame_buffer );
                            while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                            vid_draw_box(shoot[i].x+14,shoot[i].y-5-per,shoot[i].x+17,shoot[i].y-2-per,  shoot_color,
DO_FILL, vga_frame_buffer);
                            vga_flip_frame_buffers( vga_frame_buffer );
                            while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                    vid_draw_box(shoot[i].x+14,shoot[i].y-5-per,shoot[i].x+17,shoot[i].y-2-per,  shoot_color,  DO_FILL,
vga_frame_buffer);
                        //    vga_flip_frame_buffers( vga_frame_buffer );
                       //        while(  IORD_32DIRECT(  vga_frame_buffer->vga_controller_base,  0xC  )  !=
(int)vga_frame_buffer->frame0 );
                    //    vid_draw_line(shoot[i].x+14,shoot[i].y-1,shoot[i].x+17,shoot[i].y-1,    1,         RED_16,
vga_frame_buffer);
                     //       vga_flip_frame_buffers( vga_frame_buffer );
                     //           while(  IORD_32DIRECT(  vga_frame_buffer->vga_controller_base,  0xC  )  !=
(int)vga_frame_buffer->frame0 );
                     //    vid_draw_line(shoot[i].x+14,shoot[i].y-1,shoot[i].x+17,shoot[i].y-1,    1,        RED_16,
vga_frame_buffer);
```

```
                    //      vga_flip_frame_buffers( vga_frame_buffer );
                    //          while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                shoot[i].y=shoot[i].y-per-1;
            }
```

4. Lessons Learned

To be honest, I have learnt a lot from this project. First I have learnt the importance of frequently communication between group members. We failed because we have only a little communications between each other and all the problems haven't been solved until the presentation day. So if I have group task in the future I will not make this mistake again. Back to this project, one thing I think I have learned is that the architectures of hardware and software have to be designed carefully so they can cooperate well with each other. Also regularly back up source files with a modification date is really helpful.

5. Source Files

Main.c

```
#include <stdio.h>
#include <sys/alt_alarm.h>
#include "system.h"
#include "alt_types.h"
#include "../inc/vga_controller.h"
#include "sys/alt_cache.h"
#include "../inc/simple_graphics.h"
#include "../inc/vga_example.h"
#include "io.h"

void draw_grid(int scheme, display_frame_buffer_struct* vga_frame_buffer);
int main()
{
    unsigned char grid = 0;
    unsigned char ball_direct = 0;//bit0 x, bit1 y, bit2 ox, bit3 oy
    unsigned short ball_x = 100, ball_y = 100;
    unsigned short oball_x = 100, oball_y = 100;
    // VGA Device
    vga_controller_dev* vga;
    // VGA frame buffer
    display_frame_buffer_struct* vga_frame_buffer;
    //printf("+-----------------------------------------+\n");
    //printf("| Nios II VGA Controller Reference Design |\n");
    //printf("+-----------------------------------------+\n");
    // Open the VGA controller peripheral and allocate the frame buffers on the heap.
```

```
//printf(" - Initializing VGA controller.\n");
vga = (vga_controller_dev*)alt_vga_open_dev("/dev/VGA");
vga_frame_buffer = vga_init_no_interrupt( vga, HEAP );
// Run the cube rotation routine.
//run_rotate( vga_frame_buffer );
grid = 0;
draw_grid(grid, vga_frame_buffer);
vid_draw_circle_ex
(
    ball_x,
    ball_y,
    20,
    0,
    DO_FILL,
    vga_frame_buffer,
    DRAW_MODE_REVERSE
);
vga_flip_frame_buffers( vga_frame_buffer );
while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
draw_grid(grid, vga_frame_buffer);
/*vid_draw_circle_ex
(
    ball_x,
    ball_y,
    20,
    0,
    DO_FILL,
    vga_frame_buffer,
    DRAW_MODE_REVERSE
);*/
//vga_flip_frame_buffers( vga_frame_buffer );
while(1)
{
    usleep(1000);
    if(0 != (ball_direct & 0x01))
    {
        //x+∑ΩœÚ
        if(ball_x >= 619)
        {
            ball_direct ^= 0x01;
            ball_direct |= 0x10;
        }
        else
        {
```

```
        oball_x = ball_x;
        ball_x++;
    }
}
else
{
    //x-∑ΩœÚ
    if(ball_x <= 20)
    {
        ball_direct ^= 0x01;
        ball_direct |= 0x10;
    }
    else
    {
        oball_x = ball_x;
        ball_x--;
    }
}
if(0 != (ball_direct & 0x02))
{
    //y+∑ΩœÚ
    if(ball_y >= 459)
    {
        ball_direct ^= 0x02;
        ball_direct |= 0x20;
    }
    else
    {
        oball_y = ball_y;
        ball_y++;
    }
}
else
{
    //y-∑ΩœÚ
    if(ball_y <= 20)
    {
        ball_direct ^= 0x02;
        ball_direct |= 0x20;
    }
    else
    {
        oball_y = ball_y;
        ball_y--;
        }
}
if(0 != (ball_direct & 0x30))
{
        grid = (ball_direct & 0x03) + ((0 != (ball_direct & 0x10)) ? 4 : 0);
        ball_direct &= (~0x30);
        draw_grid(grid, vga_frame_buffer);
        vid_draw_circle_ex
        (
                ball_x,
                ball_y,
                20,
                0,
                DO_FILL,
                vga_frame_buffer,
                DRAW_MODE_REVERSE
        );
        vga_flip_frame_buffers( vga_frame_buffer );
        while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
        draw_grid(grid, vga_frame_buffer);
        /*vid_draw_circle_ex
        (
                ball_x,
                ball_y,
                20,
                0,
                DO_FILL,
                vga_frame_buffer,
                DRAW_MODE_REVERSE
        );*/
        //vga_flip_frame_buffers( vga_frame_buffer );
}
else
{
        //draw_grid(grid, vga_frame_buffer);
        vid_draw_circle_ex
        (
                ball_x,
                ball_y,
                20,
                0,
                DO_FILL,
                vga_frame_buffer,
                DRAW_MODE_REVERSE
                );
                vga_flip_frame_buffers( vga_frame_buffer );
                while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
                vid_draw_circle_ex
                (
                    oball_x,
                    oball_y,
                    20,
                    0,
                    DO_FILL,
                    vga_frame_buffer,
                    DRAW_MODE_REVERSE
                );
                //vga_flip_frame_buffers( vga_frame_buffer );
            }
            oball_x = ball_x;
            oball_y = ball_y;
        }
        exit(1);
        return 0;
    }
    // Rotate a point in three dimentions on three axises
    void rotate_point_deg
    (
        point_3d_struct* in_point,
        short rotx, short roty, short rotz,
        point_3d_struct axis_unit[3],
        point_3d_struct* out_point
    )
    {
        point_3d_struct temp_point[3];
        rotate_point( in_point, rotx, &(axis_unit[0]), &(temp_point[0]) );
        rotate_point( &(temp_point[0]), roty, &(axis_unit[1]), &(temp_point[1]) );
        rotate_point( &(temp_point[1]), rotz, &(axis_unit[2]), out_point );
    }
    // Rotate a point in three dimentions on one axis
    void rotate_point
    (
        point_3d_struct* in_point,
        int angle,
        point_3d_struct* r,
        point_3d_struct* out_point
    )
    {
        short int_cos_angle, int_sin_angle;
        out_point->x = 0;
        out_point->y = 0;
        out_point->z = 0;
        int_cos_angle = int_cos(angle);
        int_sin_angle = int_sin(angle);
        out_point->x += (int_cos_angle + (((((1024 - int_cos_angle) * r->x)>>10) * r->x)>>10)) * in_point->x;
        out_point->x += ((((1024 - int_cos_angle) * r->x * r->y)>>20) - ((r->z * int_sin_angle)>>10)) * in_point->y;
        out_point->x += ((((1024 - int_cos_angle) * r->x * r->z)>>20) + ((r->y * int_sin_angle)>>10)) * in_point->z;
        out_point->x = out_point->x >> 10;
        out_point->y += ((((1024 - int_cos_angle) * r->x * r->y)>>20) + ((r->z * int_sin_angle)>>10)) * in_point->x;
        out_point->y += (int_cos_angle + (((((1024 - int_cos_angle) * r->y)>>10) * r->y)>>10)) * in_point->y;
        out_point->y += ((((1024 - int_cos_angle) * r->y * r->z)>>20) - ((r->x * int_sin_angle)>>10)) * in_point->z;
        out_point->y = out_point->y >> 10;
        out_point->z += ((((1024 - int_cos_angle) * r->x * r->z)>>20) - ((r->y * int_sin_angle)>>10)) * in_point->x;
        out_point->z += ((((1024 - int_cos_angle) * r->y * r->z)>>20) + ((r->x * int_sin_angle)>>10)) * in_point->y;
        out_point->z += (int_cos_angle + (((((1024 - int_cos_angle) * r->z)>>10) * r->z)>>10)) * in_point->z;
        out_point->z = out_point->z >> 10;
        return;
}
// Integer sin lookup
int int_sin( int deg )
{
    deg %= 360;
    if( deg < 0 )
        deg += 360;
    return(int_sin_array[deg]);
}
// Integer cos lookup
int int_cos( int deg )
{
    deg %= 360;
    if( deg < 0 )
        deg += 360;
    return(int_cos_array[deg]);
}
// Rotate the 8 points of a cube
void rotate_cube_points
(
        point_3d_struct base_cube[8],
        point_3d_struct rotated_cube[8],
        point_3d_struct axis_unit[3],
        short rotx, short roty, short rotz
)
```

```
{
        alt_8 i;
        for( i = 0; i < 8; i++ )
        {
                rotate_point_deg( &(base_cube[i]), rotx, roty, rotz, axis_unit, &(rotated_cube[i]) );
        }
}
// Draw a cube with solid sides
void draw_filled_cube( point_3d_struct cube[8],
                                int color,
                                display_frame_buffer_struct* vga_frame_buffer )
{
    triangle_struct tri[12];
    alt_8 sorted[12] = {0,1,2,3,4,5,6,7,8,9,10,11};
    alt_8 i, j;
    alt_8 temp;
    static unsigned int cube_side_color[3] = {GREEN_16, RED_16, BLUE_16};
    // Project each point of the cube to 2D
    for( i = 0; i < 8; i++ )
    {
        project( &cube[i], DISTANCE, vga_frame_buffer );
    }
    // plot the triangles that make up the sides
    for( i = 0; i < 12; i++ )
    {
        tri[i].vertex_x[0] = cube[cube_tris[i][0]].screen_x;
        tri[i].vertex_y[0] = cube[cube_tris[i][0]].screen_y;
        tri[i].vertex_x[1] = cube[cube_tris[i][1]].screen_x;
        tri[i].vertex_y[1] = cube[cube_tris[i][1]].screen_y;
        tri[i].vertex_x[2] = cube[cube_tris[i][2]].screen_x;
        tri[i].vertex_y[2] = cube[cube_tris[i][2]].screen_y;
        tri[i].center_z = (cube[cube_tris[i][0]].z +
                                        cube[cube_tris[i][1]].z +
                                        cube[cube_tris[i][2]].z ) / 3;
        tri[i].fill = DO_FILL;
    }
    // Make cube three colors.
    for( i = 0; i < 4; i++ )
    {
        tri[i].col = color & cube_side_color[0];
        tri[i+4].col = color & cube_side_color[1];
        tri[i+8].col = color & cube_side_color[2];
    }
    // Sort triangles by z center.    This lets us know which ones are in the foreground
    for( i = 0; i < 12-2; i+=2 )
    {
        for( j = i+2; j < 12; j+=2 )
        {
            short square_center_z_j = (tri[sorted[j]].center_z + tri[sorted[j+1]].center_z) / 2;
            short square_center_z_i = (tri[sorted[i]].center_z + tri[sorted[i+1]].center_z) / 2;
            if( square_center_z_j < square_center_z_i )
            {
                temp = sorted[i];
                sorted[i] = sorted[j];
                sorted[j] = temp;
                temp = sorted[i+1];
                sorted[i+1] = sorted[j+1];
                sorted[j+1] = temp;
            }
        }
    }
    // Display the closest 6 triangles.    The others are hidden anyway, so dont draw them
    for( i = 5; i >= 0; i-- )
    {
        vid_draw_triangle( &tri[sorted[i]], vga_frame_buffer );
    }
}
// Draw a cube with just lines.
void draw_cube_skeleton( point_3d_struct cube[8],
                         int color,
                         display_frame_buffer_struct* vga_frame_buffer )
{
    alt_8 i;
    // Project each point of the cube to 2D
    for( i = 0; i < 8; i++ )
    {
        project( &cube[i], DISTANCE, vga_frame_buffer );
    }
    // Draw each edge of the cube (it has 12)
    for( i = 0; i < 12; i++ )
    {
        vid_draw_line( cube[cube_lines[i][0]].screen_x,
                       cube[cube_lines[i][0]].screen_y,
                       cube[cube_lines[i][1]].screen_x,
                       cube[cube_lines[i][1]].screen_y,
                       1, color , vga_frame_buffer);
    }
}
```

```
// Increase or decrease the size of a cube
int resize_cube( point_3d_struct cube[8], int magnitude )
{
   alt_8 i;
   // Define the 8 points of a cube
   for(i = 0; i < 8; i++ )
   {
      cube[i].x += (((i >> 0) & 0x1) * 2 * magnitude) + (-magnitude);
      cube[i].y += (((i >> 1) & 0x1) * 2 * magnitude) + (-magnitude);
      cube[i].z += (((i >> 2) & 0x1) * 2 * magnitude) + (-magnitude);
   }
   return 0;
}
// Run the cube rotation algorithm
void run_rotate( display_frame_buffer_struct* vga_frame_buffer )
{
      alt_8 i, j;
      int x, y, z;
      short rotx = 0, roty = 0, rotz = 0;
      point_3d_struct base_cube[8];
      point_3d_struct cube[3][8];
      point_3d_struct axis_unit[3];
      point_3d_struct base_axis_unit_vector;
      int ticks_now, msec_passed;
      int last_ticks = 0;
      int frames = 0;
      char frame_count_text[500];
      // Start things off
      alt_8 cube_to_draw = 0;
      alt_8 cube_to_erase = 1;
      // Define unit vectors for each axis x,y,z.
      axis_unit[0].x = 1024;
      axis_unit[0].y = 0;
      axis_unit[0].z = 0;
      axis_unit[1].x = 0;
      axis_unit[1].y = 1024;
      axis_unit[1].z = 0;
      axis_unit[2].x = 0;
      axis_unit[2].y = 0;
      axis_unit[2].z = 1024;
      base_axis_unit_vector.x = 1024;
      base_axis_unit_vector.y = 0;
      base_axis_unit_vector.z = 0;
      // Define the 8 points of a cube
```

```
    printf(" - Creating Cube.\n");
    for(i = 0; i < 8; i++ )
    {
        x = (((i >> 0) & 0x1) * 2 * SIDE_LENGTH) + (-SIDE_LENGTH);
        y = (((i >> 1) & 0x1) * 2 * SIDE_LENGTH) + (-SIDE_LENGTH);
        z = (((i >> 2) & 0x1) * 2 * SIDE_LENGTH) + (-SIDE_LENGTH);
        base_cube[i].x = x;
        base_cube[i].y = y;
        base_cube[i].z = z;
        for ( j = 0; j < 3; j++ )
        {
            cube[j][i].x = x;
            cube[j][i].y = y;
            cube[j][i].z = z;
        }
    }
    // Display Title at top of screen for both frame buffers
    for( i = 0; i < 2; i++ )
    {
        vid_print_string( TITLE_X, TITLE_Y, WHITE_16, cour10_font, vga_frame_buffer, "Nios II VGA Controller Reference
Design" );
        vid_draw_box( TITLE_X - 15, TITLE_Y - 10, TITLE_X + 330, TITLE_Y + 20, BLUE_16, DO_NOT_FILL,
vga_frame_buffer);
        // Flip the frame buffers.
        vga_flip_frame_buffers( vga_frame_buffer );
    }
    printf(" - Starting Cube Rotation.\n");
    while(1)
    {
        // Erase the last cube that was drawn in this buffer
        draw_filled_cube( cube[cube_to_erase], BLACK_16, vga_frame_buffer );
        // Keep track of time for calculating rotational speed
        ticks_now = alt_nticks();
        msec_passed = ( 1024 * (ticks_now - last_ticks)) / (alt_ticks_per_second());
        last_ticks = ticks_now;
        // Move the axis of rotation at a constant rate
        rotx++; //= (rotx + ((73 * msec_passed) / 1024)) % 360;
        roty+=2; //= (roty + ((123 * msec_passed) / 1024)) % 360;
        rotz+=3; //= (rotz + ((91 * msec_passed) / 1024)) % 360;
        // Rotate a new cube
        rotate_cube_points( base_cube, cube[cube_to_draw], axis_unit, rotx, roty, rotz );
        // And draw it
        draw_filled_cube(cube[cube_to_draw], WHITE_24, vga_frame_buffer );
        frames++;
```

```
            // Flip the double buffers.
            vga_flip_frame_buffers( vga_frame_buffer );
            // keep track of our old cubes so we can erase them when we get
            // back around to that frame
            cube_to_draw = (( cube_to_draw + 1 ) % 3 );
            cube_to_erase = (( cube_to_erase + 1 ) % 3 );
            // Wait until frame being displayed is done so we dont write to it
            // at the same time.
            while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) !=
                                    (int)vga_frame_buffer->frame0 );
    }
}
// Project a 3D point onto 2D space
void project(point_3d_struct* point, int dist, display_frame_buffer_struct* vga_fb) {
  // Prevent a divide by zero
  if(point->z == 0)
  {
    // Project the point
    point->screen_x = point->x + (vga_fb->width / 2);
    point->screen_y = -point->y + (vga_fb->height / 2);
  }
  else
  {
    // Project the point
    point->screen_x = ((point->x * dist) / (point->z + dist)) + (vga_fb->width / 2);
    point->screen_y = -((point->y * dist) / (point->z + dist)) + (vga_fb->height / 2);
  }
}
void draw_grid(int scheme, display_frame_buffer_struct* vga_frame_buffer)
{
    unsigned short x, y, color;
    unsigned char i, j, k;
    //for(k = 0; i< 2; k++)
    //{
    //      while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) !=
    //                              (int)vga_frame_buffer->frame0 );
    for(i = 0, x = 0; i < 16; i++, x += 40)
    {
        for(j = 0, y = 0; j < 16; j++, y += 30)
        {
            switch(scheme)
            {
                case 0:
                    color = (~i);
                    color <<= 6;
                    color += ((~j & 0x0f) << 1);
                    color <<= 5;
                    color += (~i & 0x0f);
                    break;
                case 1:
                    color = (~j);
                    color <<= 6;
                    color += ((~i & 0x0f) << 1);
                    color <<= 5;
                    color += j;
                    break;
                case 2:
                    color = (~i);
                    color <<= 6;
                    color += (j << 1);
                    color <<= 5;
                    color += (~i & 0x0f);
                    break;
                case 3:
                    color = (~j);
                    color <<= 6;
                    color += (i << 1);
                    color <<= 5;
                    color += j;
                    break;
                case 4:
                    color = i;
                    color <<= 6;
                    color += ((~j & 0x0f) << 1);
                    color <<= 5;
                    color += (~i & 0x0f);
                    break;
                case 5:
                    color = j;
                    color <<= 6;
                    color += ((~i & 0x0f) << 1);
                    color <<= 5;
                    color += j;
                    break;
                case 6:
                    color = i;
                    color <<= 6;
                    color += (j << 1);
```

```
                    color <<= 5;
                    color += (~i & 0x0f);
                    break;
                default:
                    color = j;
                    color <<= 6;
                    color += (i << 1);
                    color <<= 5;
                    color += j;
                    break;
            }
            vid_draw_box(x, y, x+40, y+30, color, DO_FILL, vga_frame_buffer);
        }
    }
    //}
}
```

VGA_Controller.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "vga_controller.h"
#include "io.h"
#include "sys/alt_alarm.h"
#include "sys/alt_cache.h"
#include "system.h"
#include "priv/alt_file.h"
// VGA device list (usualy just one)
ALT_LLIST_HEAD(vga_dev_list);
/******************************************************************
*   Function: vga_dev_init
*
*   Purpose: HAL device driver initialization.
*            Called by alt_sys_init.
*
******************************************************************/
int vga_dev_init ( vga_controller_dev* vga )
{
  int ret_code = 0;
  ret_code = vga_device_register( &(vga->dev) );
  return ret_code;
}
```

```
/****************************************************************
*    Function: alt_vga_open_dev
*    Purpose: Opens the VGA controller for use.
*                     Returns a file descriptor for the controller
****************************************************************/
alt_dev* alt_vga_open_dev(const char* name)
{
   alt_dev* dev = (alt_dev*)alt_find_dev(name, &vga_dev_list);
   return dev;
}
/****************************************************************
*    Function: alt_vga_close_dev
*    Purpose: Closes the VGA controller.
****************************************************************/
void alt_vga_close_dev(alt_dev* fd)
{
   return;
}
/****************************************************************
*    Function: vga_init_no_interrupt
*
*    Purpose: Initializes the VGA controller for the Lancelot VGA
*                     daughterboard.    Gets memory for the frame buffer, sets
*                     the resolution of the frame buffer, resets the VGA
*                     controller hardware, gives the controller the base
*                     address of the frame buffer, then enables
*                     the controller.    Interrupts are NOT enabled.
*
****************************************************************/
display_frame_buffer_struct* vga_init_no_interrupt ( vga_controller_dev* vga, int buffer_location)
{
        display_frame_buffer_struct* vga_frame_buffer;
      int bytes_per_pixel, bytes_per_frame;
      // We'll need these values more than once, so let's pre-calculate them.
      bytes_per_pixel = vga->color_depth >> 3; // same as /8
      bytes_per_frame = (( vga->width * vga->height ) * bytes_per_pixel);
        // Allocate our frame buffers
        if( buffer_location == HEAP )
        {
                vga_frame_buffer = (display_frame_buffer_struct*) alt_uncached_malloc(sizeof (display_frame_buffer_struct));
              vga_frame_buffer->frame0 = (frame_array*) alt_uncached_malloc(( bytes_per_frame ));
            // If we're double-buffering, grab an extra buffer, if not, just make
            // both pointers point to the same buffer.
            if(vga->frame_buffers == 2)
            {
                    vga_frame_buffer->frame1 = (frame_array*) alt_uncached_malloc(( bytes_per_frame ));
            }
            else
            {
                    vga_frame_buffer->frame1 = vga_frame_buffer->frame0;
            }
        }
        else
        {
               vga_frame_buffer = (display_frame_buffer_struct*)buffer_location;
               vga_frame_buffer->frame0 = (frame_array*)(buffer_location + sizeof(display_frame_buffer_struct));
          // If we're double-buffering, grab an extra buffer, if not, just make
          // both pointers point to the same buffer.
          if(vga->frame_buffers == 2)
          {
               vga_frame_buffer->frame1 = (vga_frame_buffer->frame0 + bytes_per_frame);
          }
          else
          {
               vga_frame_buffer->frame1 = vga_frame_buffer->frame0;
          }
        }
        vga_frame_buffer->width = vga->width;
        vga_frame_buffer->height = vga->height;
        vga_frame_buffer->color_depth = vga->color_depth;
        vga_frame_buffer->frame_buffers = vga->frame_buffers;
        vga_frame_buffer->bytes_per_frame = bytes_per_frame;
        vga_frame_buffer->bytes_per_pixel = (vga->color_depth / 8);
        vga_frame_buffer->vga_controller_base = vga->base_addr;
      //Clear all frame buffers to black
        vga_clear_screen( vga_frame_buffer, BLACK_8 );
      vga_flip_frame_buffers( vga_frame_buffer );
        vga_clear_screen( vga_frame_buffer, BLACK_8 );
        IOWR_32DIRECT( vga->base_addr, 0, 0x0 ); /* Reset the VGA controller */
        IOWR_32DIRECT( vga->base_addr, 4, ( int )vga_frame_buffer->frame0 ); /* Where our frame buffer starts */
        IOWR_32DIRECT( vga->base_addr, 8, ( ( vga->width * vga->height ) * bytes_per_pixel ) ); /* amount of memory
needed */
        IOWR_32DIRECT( vga->base_addr, 0, 0x1 ); /* Set the go bit. */
   return ( vga_frame_buffer );
}
int vga_flip_frame_buffers( display_frame_buffer_struct* vga_frame_buffer )
{
        frame_array* temp_frame;
        int ret_code;
        if( vga_frame_buffer->frame_buffers == 2 )
        {
                temp_frame = vga_frame_buffer->frame0;
                vga_frame_buffer->frame0 = vga_frame_buffer->frame1;
                vga_frame_buffer->frame1 = temp_frame;
                IOWR_32DIRECT( vga_frame_buffer->vga_controller_base, 4, ( int )vga_frame_buffer->frame0 );
                ret_code = 0;
        }
        else
        {
                ret_code = 1;
        }
        return(ret_code);
}
int vga_stop ( vga_controller_dev* vga )
{
        IOWR_32DIRECT( vga->base_addr, 0, 0x0 ); /* Reset the VGA controller */
   return (0);
}
/******************************************************************
*    Function: vga_clear_screen
*
*    Purpose: Uses the fast memset routine to clear the entire frame
*                 buffer.    User can specify black(0x00) or white(0xFF).
*
******************************************************************/
inline void vga_clear_screen (display_frame_buffer_struct* vga_frame_buffer, char color)
{
        memset( (void*)(vga_frame_buffer->frame1), color, vga_frame_buffer->bytes_per_frame );
}
```

KB(Keyboard).c

```
/*-----------------------------------------------------------------------------------
  *    Include
  *-----------------------------------------------------------------------------------*/
#include "kb.h"
#include "altera_avalon_pio_regs.h"
#include "system.h"
#include "sys/alt_cache.h"
#include "io.h"

/*-----------------------------------------------------------------------------------
```

```
 *   Define
 *-----------------------------------------------------------------------------------------*/
#define     OUT        1
#define     IN         0

/*-----------------------------------------------------------------------------------------
 *   Struct
 *-----------------------------------------------------------------------------------------*/
KEYBOARD_STRUCT key1_lut[]=
{
      0x16, 0x31, 0x21, 0,    0,    // { 1 }       { ! }
      0x1e, 0x32, 0x40, 0,    0,    // { 2 }       { @ }
      0x26, 0x33, 0x23, 0,    0,    // { 3 }       { # }
      0x25, 0x34, 0x24, 0,    0,    // { 4 }       { $ }
      0x2e, 0x35, 0x25, 0,    0,    // { 5 }       { % }
      0x36, 0x36, 0x5e, 0,    0,    // { 6 }       { ^ }
      0x3d, 0x37, 0x26, 0,    0,    // { 7 }       { & }
      0x3e, 0x38, 0x2a, 0,    0,    // { 8 }       { * }
      0x46, 0x39, 0x28, 0,    0,    // { 9 }       { ( }
      0x45, 0x30, 0x29, 0,    0,    // { 0 }       { ) }
      0x1c, 0x61, 0x41, 0,    0,    // { a }       { A }
      0x32, 0x62, 0x42, 0,    0,    // { b }       { B }
      0x21, 0x63, 0x43, 0,    0,    // { c }       { C }
      0x23, 0x64, 0x44, 0,    0,    // { d }       { D }
      0x24, 0x65, 0x45, 0,    0,    // { e }       { E }
      0x2b, 0x66, 0x46, 0,    0,    // { f }       { F }
      0x34, 0x67, 0x47, 0,    0,    // { g }       { G }
      0x33, 0x68, 0x48, 0,    0,    // { h }       { H }
      0x43, 0x69, 0x49, 0,    0,    // { i }       { I }
      0x3b, 0x6a, 0x4a, 0,    0,    // { j }       { J }
      0x42, 0x6b, 0x4b, 0,    0,    // { k }       { K }
      0x4b, 0x6c, 0x4c, 0,    0,    // { l }       { L }
      0x3a, 0x6d, 0x4d, 0,    0,    // { m }       { M }
      0x31, 0x6e, 0x4e, 0,    0,    // { n }       { N }
      0x44, 0x6f, 0x4f, 0,    0,    // { o }       { O }
      0x4d, 0x70, 0x50, 0,    0,    // { p }       { P }
      0x15, 0x71, 0x51, 0,    0,    // { q }       { Q }
      0x2d, 0x72, 0x52, 0,    0,    // { r }       { R }
      0x1b, 0x73, 0x53, 0,    0,    // { s }       { S }
      0x2c, 0x74, 0x54, 0,    0,    // { t }       { T }
      0x3c, 0x75, 0x55, 0,    0,    // { u }       { U }
      0x2a, 0x76, 0x56, 0,    0,    // { v }       { V }
      0x1d, 0x77, 0x57, 0,    0,    // { w }       { W }
      0x22, 0x78, 0x58, 0,    0,    // { x }       { X }
```

```
0x35, 0x79, 0x59, 0,    0,    // { y }        { Y }
0x1a, 0x7a, 0x5a, 0,    0,    // { z }        { Z }
0x0e, 0x60, 0x7e, 0,    0,    // { ` }        { ~ }
0x4e, 0x2d, 0x5f, 0,    0,    // { - }        { _ }
0x55, 0x3d, 0x2b, 0,    0,    // { = }        { + }
0x5d, 0x5c, 0x7c, 0,    0,    // { \ }        { | }
0x54, 0x5b, 0x7b, 0,    0,    // { [ }        { { }
0x5b, 0x5d, 0x7d, 0,    0,    // { ] }        { } }
0x4c, 0x3b, 0x3a, 0,    0,    // { ; }        { : }
0x52, 0x27, 0x22, 0,    0,    // { ' }        { " }
0x41, 0x2c, 0x3c, 0,    0,    // { , }        { < }
0x49, 0x2e, 0x3e, 0,    0,    // { . }        { > }
0x4a, 0x2f, 0x3f, 0,    0,    // { / }        { ? }
0x5a, 0x0a, 0,        0,    0,    // u{ enter }
0x29, 0x20, 0,        0,    0,    // { space }
0x05,      0,        0,        0,    "F1",
0x06,      0,        0,        0,    "F2",
0x04,      0,        0,        0,    "F3",
0x0c,      0,        0,        0,    "F4",
0x03,      0,        0,        0,    "F5",
0x0b,      0,        0,        0,    "F6",
0x83,      0,        0,        0,    "F7",
0x0a,      0,        0,        0,    "F8",
0x01,      0,        0,        0,    "F9",
0x09,      0,        0,        0,    "F10",
0x78,      0,        0,        0,    "F11",
0x07,      0,        0,        0,    "F12",
0x66,      0,        0,        0,    "BKSP",
0x0d,      0,        0,        0,    "TAB",
0x14,      0,        0,        0,    "L_CTRL",
0xe01f, 0,        0,        0,    "L_GUI",
0x11,      0,        0,        0,    "L_ALT",
0xe014, 0,        0,        0,    "R_CTRL",
0xe027, 0,        0,        0,    "R_GUI",
0xe011, 0,        0,        0,    "R_ALT",
0xe02f, 0,        0,        0,    "APPS",
0x76,      0x21,        0,        0,    "ESC",
0xe070, 0,        0,        0,    "Ins",
0xe06c, 0,        0,        0,    "Home",
0xe07d, 0,        0,        0,    "PgUP",
0xe071, 0,        0,        0,    "Del",
0xe069, 0,        0,        0,    "End",
0xe07a, 0,        0,        0,    "PgDn",
0xe075, 0,        0,        0,    "U_ARROW",
    0xe06b, 0,      0,      0,    "L_ARROW",
    0xe072, 0,      0,      0,    "D_ARROW",
    0xe074, 0,      0,      0,    "R_ARROW",
    0xe04a, 0,      0,      0,    "KP_/",
    0x7c,   0,      0,      0,    "KP_*",
    0x7b,   0,      0,      0,    "KP_-",
    0x79,   0,      0,      0,    "KP_+",
    0xe05a, 0,      0,      0,    "KP_EN",
    0x71,   0,      0,      0,    "KP_.",
    0x70,   0,      0,      0,    "KP_0",
    0x69,   0,      0,      0,    "KP_1",
    0x72,   0,      0,      0,    "KP_2",
    0x7a,   0,      0,      0,    "KP_3",
    0x6b,   0,      0,      0,    "KP_4",
    0x73,   0,      0,      0,    "KP_5",
    0x74,   0,      0,      0,    "KP_6",
    0x6c,   0,      0,      0,    "KP_7",
    0x75,   0,      0,      0,    "KP_8",
    0x7d,   0,      0,      0,    "KP_9",
    0xe037, 0,      0,      0,    "Power",
    0xe03f, 0,      0,      0,    "Sleep",
    0xe05e, 0,      0,      0,    "Wake",
    0,      0,      0,      0,    0,
};
#define KEY_LSHIFT_MASK      (0x12)
#define KEY_RSHIFT_MASK      (0x59)
#define KEY_CAPS_MASK        (0x58)
#define KEY_NUM_MASK         (0x77)
#define KEY_LSHIFT_INDEX    (0)
#define KEY_RSHIFT_INDEX    (1)
#define KEY_CAPS_INDEX      (2)
#define KEY_NUM_INDEX       (3)
KEYBOARD_STRUCT keys_lut[]=
{
    KEY_LSHIFT_MASK,    0,    0,    0,    0,        //L_shift
    KEY_RSHIFT_MASK,    0,    0,    0,    0,        //R_shift
    KEY_CAPS_MASK,      0,    0,    0,    0,        //caps
    KEY_NUM_MASK,       0,    0,    0,    0,        //num
    0,                  0,    0,    0,    0,
};
/*
 * ===   FUNCTION   ======================================================================
 *           Name:   ps2_restart
 *   Description:
```

```
 * =====================================================================================
 */
void ps2_restart(void)
{
    IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_CLK_BASE, OUT);    //clk and data direction is output
    IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_DAT_BASE, OUT);
    IOWR_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE, 0);     //drive clk and data low
    IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 1);
    usleep(200);
}
/*
 * ===  FUNCTION  ======================================================================
 *         Name:  ps2_input
 *  Description:
 * =====================================================================================
 */
alt_u8 ps2_input(void)
{
    alt_u16 i, dat;

    IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_CLK_BASE, IN);
    IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_DAT_BASE, IN);
    usleep(1);

    for(i=0; i<11; i++)
    {
        while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01); //Wait for the device to bring Clock low.
        while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01)); //Wait for the device to bring Clock High.
        dat >>= 1;
        if(IORD_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE))
            dat |= 0x200;
    }

    return dat & 0xff;

}

alt_u8 ps2_kbhit(void)
{
  alt_u8   dat_hitt =0;;
  int ps2_count;

   IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_CLK_BASE, IN);
   IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_DAT_BASE, IN);
```

```
    usleep(1);

     for (ps2_count =0;ps2_count<100000;ps2_count++)
    {
       if(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01)) //º¸≈Ã ‰»Î
          {
               dat_hitt = 0xff;
                 break;
          }
       else    dat_hitt = 0x00;
    }
  return dat_hitt;
}
/*
 * ===   FUNCTION   ======================================================================
 *                Name:   ps2_getch
 *  Description:
 * =====================================================================================
 */
alt_8 ps2_getch(void)
{
    alt_u16 i;
    alt_u8 ps2_dat;
    alt_8 mchar;
    ps2_dat = ps2_input();
    if(ps2_dat == 0xf0)      //∂Œ¬Î
    {
        ps2_input();
        return 0xff;
    }
    else                              //Õ®¬Î
    {
        for(i=0; i<50; i++)
             if(ps2_dat == key1_lut[i].mask)
                 mchar = key1_lut[i].value;
        return mchar;
    }
}
/*
 * ===   FUNCTION   ======================================================================
 *                Name:   ps2_command
 *  Description:
 * =====================================================================================
 */
```

```
void ps2_command(alt_u8 cmd)
{
    alt_u16 i, j;
    IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_CLK_BASE, OUT);    //clk and data direction is output
    IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_DAT_BASE, OUT);
    IOWR_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE, 0);      //drive clk and data low
    IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 0);
    usleep(200);
    IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_CLK_BASE, IN); //Release the Clock line.
    usleep(10);
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01); //Wait for the device to bring the Clock line low.
    for(i=0,j=0; i<8; i++)
    {
        if(cmd & 0x01)                  //Set/reset the Data line to send the first data bit.
        {
            IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 1);
            j++;
        }
        else
            IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 0);;
        cmd >>= 1;
        while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01));     //Wait for the device to bring Clock
high.
        usleep(20);
        while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01);         //Wait for the device to bring Clock
low.
        usleep(20);
    }
    if(!(j&0x01))
        IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 1);      //Send parity bit
    else
        IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 0);
    usleep(20);
    while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01));    //Wait for the device to bring Clock high.
    usleep(20);
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01);          //Wait for the device to bring Clock low.
    IOWR_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE, 1);                            //Send stop bit
    usleep(20);
    while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01));    //Wait for the device to bring Clock high.
    usleep(20);
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01);          //Wait for the device to bring Clock low.
    IOWR_ALTERA_AVALON_PIO_DIRECTION(PS2_DAT_BASE, IN);;                 //Release the Data line.
    usleep(20);
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE) & 0x01);          //Wait for the device to bring Data low.
    while(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01);          //Wait for the device to bring Clock low.
    usleep(20);
    while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_CLK_BASE) & 0x01));          //Wait for the device to bring Clock high.
    while(!(IORD_ALTERA_AVALON_PIO_DATA(PS2_DAT_BASE) & 0x01));          //Wait for the device to bring Data high.
}
```

Tank.c

```
#include <stdio.h>
#include <sys/alt_alarm.h>
#include "system.h"
#include "alt_types.h"
#include "vga_controller.h"
#include "sys/alt_cache.h"
#include "simple_graphics.h"
#include "vga_example.h"
#include "io.h"
#include "kb.h"
#include "altera_avalon_pio_regs.h"

  /*define the control of user's tank*/
#define a_UP       0x77     //W
#define a_DOWN    0x73     //S
#define a_LEFT    0x61     //A
#define a_RIGHT 0x64     //D
#define a_shoot 0x20     //SPACE

#define ESC     0x21

#define UPWAY       1/*define the direction*/
#define DOWNWAY    2
#define LEFTWAY    3
#define RIGHTWAY 4

int enemynum=2;/*define the enemy number*/
int player_life=3;/*define the situation of ending the game*/
int ESC_time=0;
struct SHOOT      /*define the structure of shoot*/
{
    int life;
    int x,y;
    int way;
}shoot[3]={{0,0,0,UPWAY},{0,0,0,UPWAY},{0,0,0,UPWAY}};
```

```
struct PLAYER    /*define the structure of the user*/
{
    int life;
    int x,y;
    int i,j;
    int way;
}player[1]={1,0,0,3,13,UPWAY};

struct ENEMY    /*define enemy's structure*/
{
    int life;
    int x,y;
    int i,j;
    int way;
    int control;
}enemy[2]={{1,0,0,1,1,DOWNWAY,0},{1,0,0,1,2,DOWNWAY,0}};

void play(display_frame_buffer_struct* vga_frame_buffer);/*define the main function of game*/
void map_all(int map[15][15],    display_frame_buffer_struct* vga_frame_buffer);/*initialize the map function*/

void judge_tank_my(int map[15][15],int key,display_frame_buffer_struct* vga_frame_buffer);/*judge the function of user's
tank*/
void enemy_control(int map[15][15],display_frame_buffer_struct* vga_frame_buffer);
void judge_moveshootway(int map[15][15],int i,display_frame_buffer_struct* vga_frame_buffer);
void judge_shoot(int m,int map[15][15],int i,display_frame_buffer_struct* vga_frame_buffer);
void judge_shootway(int map[15][15],int i,display_frame_buffer_struct* vga_frame_buffer);

void uptank(int i,int j,int color,display_frame_buffer_struct* vga_frame_buffer);/*the function of draw a tank*/
void downtank(int i,int j,int color,display_frame_buffer_struct* vga_frame_buffer);
void lefttank(int i,int j,int color,display_frame_buffer_struct* vga_frame_buffer);
void righttank(int i,int j,int color,display_frame_buffer_struct* vga_frame_buffer);

void allcircle(int i,int j,    display_frame_buffer_struct* vga_frame_buffer);
void blank(int i,int j,    display_frame_buffer_struct* vga_frame_buffer);

void map_water(int i,int j,    display_frame_buffer_struct* vga_frame_buffer);
void map_steel(int i,int j,    display_frame_buffer_struct* vga_frame_buffer);
void map_wall(int i,int j,    display_frame_buffer_struct* vga_frame_buffer);
void map_border(int i,int j,    display_frame_buffer_struct* vga_frame_buffer);
void map_base_enemy(int i,int j,    display_frame_buffer_struct* vga_frame_buffer);
void map_base_player(int i,int j,    display_frame_buffer_struct* vga_frame_buffer);

void fengming(int n);
void end();/*define the function of ending the game */

alt_u8 count= 0;

void main()
{
      // VGA Device
    // ps2_command(0xff);      //reset keyboard
    // ps2_restart();
    vga_controller_dev* vga;

    // VGA frame buffer
    display_frame_buffer_struct* vga_frame_buffer;
    printf("| Nios II VGA Controller Reference Design |\n");
    // Open the VGA controller peripheral and allocate the frame buffers on the heap.
    vga = (vga_controller_dev*)alt_vga_open_dev("/dev/VGA");
    vga_frame_buffer = vga_init_no_interrupt( vga, HEAP );

    IOWR_ALTERA_AVALON_PIO_DATA(BUZZER_BASE,0x1);

   {play( vga_frame_buffer);}
 //    end();
}

void play( display_frame_buffer_struct* vga_frame_buffer)/*main function*/
{
 alt_u8 key,flag;int i;int num=0;

//     int map[15][15]={{8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 },
//                       {8 ,44,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,8 },
//                       {8 ,45,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,8 },
//                       {8 ,0,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,8 },
//                       {8 ,5 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ,5 ,5 ,8 },
//                       {8 ,5 ,0 ,0 ,7 ,7 ,7 ,6 ,7 ,7 ,7 ,5 ,5 ,5 ,8 },
//                       {8 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ,5 ,5 ,8 },
//                       {8 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,8 },
//                       {8 ,0 ,0 ,0 ,0 ,0 ,0 ,6 ,0 ,0 ,0 ,6 ,0 ,0 ,8 },
//                       {8 ,5 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ,5 ,5 ,8 },
//                       {8 ,5 ,0 ,0 ,5 ,5 ,5 ,5 ,5 ,5 ,0 ,5 ,5 ,5 ,8 },
//                       {8 ,5 ,0 ,0 ,5 ,5 ,5 ,5 ,5 ,5 ,0 ,5 ,5 ,5 ,8 },
//                       {8 ,0 ,0 ,0 ,0 ,5 ,5 ,5 ,5 ,5 ,0 ,0 ,0 ,0 ,8 },
//                       {8 ,0 ,0 ,1 ,0 ,5 ,5 ,9 ,5 ,5 ,2 ,0 ,0 ,0 ,8 },
//                       {8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 }};
```

```
    int map[15][15]={{8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 },
                     {8 ,44,0 ,0 ,0 ,5 ,5 ,9 ,5 ,5 ,0 ,0 ,0 ,0 ,8 },
                     {8 ,45,0 ,0 ,0 ,5 ,5 ,5 ,5 ,5 ,0 ,0 ,0 ,0 ,8 },
                     {8 ,0,0 ,0 ,0 ,5 ,5 ,5 ,5 ,5 ,0 ,0 ,0 ,0 ,8 },
                     {8 ,5 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ,5 ,5 ,8 },
                     {8 ,5 ,0 ,0 ,5 ,5 ,5 ,6 ,5 ,5 ,5 ,5 ,5 ,5 ,8 },
                     {8 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ,5 ,5 ,8 },
                     {8 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,8 },
                     {8 ,0 ,0 ,0 ,0 ,0 ,0 ,6 ,0 ,0 ,0 ,6 ,0 ,0 ,8 },
                     {8 ,5 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ,5 ,5 ,8 },
                     {8 ,5 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ,5 ,5 ,8 },
                     {8 ,5 ,0 ,0 ,5 ,5 ,5 ,5 ,5 ,5 ,0 ,5 ,5 ,5 ,8 },
                     {8 ,0 ,0 ,0 ,0 ,5 ,5 ,5 ,5 ,5 ,0 ,0 ,0 ,0 ,8 },
                     {8 ,0 ,0 ,1 ,0 ,5 ,5 ,10 ,5 ,5 ,0 ,0 ,0 ,0 ,8 },
                     {8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 ,8 }};


   map_all(map, vga_frame_buffer);/*initialize map*/
     vid_print_string(470, 90, GREEN_16,cour10_font,vga_frame_buffer ,"Player Life : 3");
                vga_flip_frame_buffers( vga_frame_buffer );
                while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
     vid_print_string(470, 90, GREEN_16,cour10_font,vga_frame_buffer ,"Player Life : 3");
                vga_flip_frame_buffers( vga_frame_buffer );
                while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
 //      ps2_command(0xff);
 //     flag=ps2_kbhit();
    while(1)


        {
                IOWR_ALTERA_AVALON_PIO_DATA(BUZZER_BASE,0x1);


               flag=ps2_kbhit();




                if(enemynum==0||player_life==0)/*end game if enemy is 0*/
                 {    if(enemynum==0)
                        {
                          vid_print_string(500, 150, GREEN_16,cour10_font,vga_frame_buffer ,"YOU WIN!");
                           vga_flip_frame_buffers( vga_frame_buffer );
                          while(     IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
                      vid_print_string(500, 150, GREEN_16,cour10_font,vga_frame_buffer ,"YOU WIN!");
                          vga_flip_frame_buffers( vga_frame_buffer );
                          while(     IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );

                        }
                      else if(player_life==0)
                              {
                               vid_print_string(500, 150, RED_16,cour10_font,vga_frame_buffer ,"YOU LOSE!");
                                  vga_flip_frame_buffers( vga_frame_buffer );
                                 while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                            vid_print_string(500, 150, RED_16,cour10_font,vga_frame_buffer ,"YOU LOSE!");
                                vga_flip_frame_buffers( vga_frame_buffer );
                               while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                              }
                     goto aaa;
                    }
                  for(i=0;i<3;i++)
                        if(shoot[i].life==1)
                           judge_moveshootway(map,i,vga_frame_buffer);
                   usleep(1);num++;
                   if(num==10)
                        {
                             enemy_control(map, vga_frame_buffer);
                             num=0;
                         }
              if(player_life!=0&&player[0].life==0)
                  {   player[0].life=1;
                      player[0].x=0;
                      player[0].y=0;
                      player[0].i=3;
                      player[0].j=13;
                      player[0].way=UPWAY;
                  }
               printf("%x", flag);
               if(flag == 0xff)
                   {     ps2_restart();
                         key=ps2_getch();
                         if(key != 0xff)
                           { if(ESC_time == 0)
```

```
                                                if(key !=ESC)
                                                   judge_tank_my(map,key, vga_frame_buffer);/*judge the user tank's function*/
                                                else    { ESC_time =1 ; }
                                          else
                                           if(key == ESC)
                                           {
                                                      judge_tank_my(map,key, vga_frame_buffer);
                                                        ESC_time =0 ;
                                            }
                                            else      ESC_time =1 ;
                                        }
                                     count =0;
                                 }
                          else    printf("22");
                      //show the remain life
                }
              aaa: ;
}
void map_all(int map[15][15], display_frame_buffer_struct* vga_frame_buffer)
{
       int i,j;
       for(i=0;i<15;i++)
              for(j=0;j<15;j++)
                     switch(map[j][i])
                            {
                                   case 0: break;
                                   case 5:map_wall(i,j, vga_frame_buffer);break;
                                   case 6:map_steel(i,j, vga_frame_buffer);break;
                                   case 7:map_water(i,j, vga_frame_buffer);break;
                                   case 8:map_border(i,j, vga_frame_buffer);break;
                                   case 9:map_base_enemy(i,j, vga_frame_buffer);break;
                                   case 10:map_base_player(i,j, vga_frame_buffer);break;
                            }
  //uptank(10, 23,GREEN_16,vga_frame_buffer);

      //        vga_flip_frame_buffers( vga_frame_buffer );
     //          while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
    //    lefttank(10,16,GREEN_16,vga_frame_buffer);
//              vga_flip_frame_buffers( vga_frame_buffer );
//                  while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
}

//void allcircle(int i,int j)
//{
```

```
//      int x,y;x=30*i+15,y=30*j+15;
//
//      arc(x+15,y+15, 90,180,8);     //
//      arc(x+16,y+15,   0, 90,8);
//      arc(x+15,y+16,180,270,8);
//      arc(x+16,y+16,270,360,8);
//      arc(x+15,y+15, 90,180,4);
//      arc(x+16,y+15,   0, 90,4);
//      arc(x+15,y+16,180,270,4);
//      arc(x+16,y+16,270,360,4);
//
//   、、    vid_draw_circle(int Hcenter, int Vcenter, int radius, int color, char fill, display_frame_buffer_struct* frame_buffer)
//}
void blank(int i,int j,display_frame_buffer_struct* vga_frame_buffer)/*画空白函数*/
{
    int x,y;
    x=30*i+15,y=30*j+15;
   // setfillstyle(1,7);
   // bar(x+1,y+1,x+30,y+30);
    vid_draw_box(x+1, y+1,x+30 , y+30, BLACK_16, DO_FILL, vga_frame_buffer);
          vga_flip_frame_buffers( vga_frame_buffer );
          while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
    vid_draw_box(x+1, y+1,x+30 , y+30, BLACK_16, DO_FILL, vga_frame_buffer);
      //   vga_flip_frame_buffers( vga_frame_buffer );
    //          while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
}
void map_water(int i,int j,display_frame_buffer_struct* vga_frame_buffer)
{
    int x,y;x=30*i+15,y=30*j+15;
 //     setfillstyle(1,9);
 //     bar(x+1,y+1,x+30,y+30);
   // setfillstyle(1,7);
  //    bar(x+14,y+1,x+17,y+30);
  //    bar(x+1,y+14,x+30,y+17);
   vid_draw_box(x+1, y+1,x+30 , y+30, BLUE_16, DO_FILL, vga_frame_buffer);
      vga_flip_frame_buffers( vga_frame_buffer );
     while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_box(x+1, y+1,x+30 , y+30, BLUE_16, DO_FILL, vga_frame_buffer);
      vga_flip_frame_buffers( vga_frame_buffer );
     while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
   vid_draw_box(x+14, y+1,x+17 , y+30, BLUE_16, DO_FILL, vga_frame_buffer);
       vga_flip_frame_buffers( vga_frame_buffer );
          while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
          vid_draw_box(x+14, y+1,x+17 , y+30, BLUE_16, DO_FILL, vga_frame_buffer);
```

```
            vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
     vid_draw_box(x+1,y+14,x+30,y+17, BLUE_16, DO_FILL, vga_frame_buffer);
          vga_flip_frame_buffers( vga_frame_buffer );
              while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
        vid_draw_box(x+1,y+14,x+30,y+17, BLUE_16, DO_FILL, vga_frame_buffer);
              vga_flip_frame_buffers( vga_frame_buffer );
              while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
}
void map_steel(int i,int j,display_frame_buffer_struct* vga_frame_buffer)
{
      int x,y;x=30*i+15,y=30*j+15;
     // setfillstyle(1,8);
     // bar(x+1,y+1,x+30,y+30);
      vid_draw_box(x+1,y+1,x+30,y+30, GOLD_16, DO_FILL, vga_frame_buffer);
           vga_flip_frame_buffers( vga_frame_buffer );
              while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
           vid_draw_box(x+1,y+1,x+30,y+30, GOLD_16, DO_FILL, vga_frame_buffer);
              vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
 //      setfillstyle(1,15);
   //    bar(x+3,y+3,x+12,y+12);
   //    bar(x+19,y+3,x+28,y+12);
   //    bar(x+3,y+19,x+12,y+28);
     // bar(x+19,y+19,x+28,y+28);
      vid_draw_box(x+3,y+3,x+12,y+12, GRAY_16, DO_FILL, vga_frame_buffer);
           vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
                  vid_draw_box(x+3,y+3,x+12,y+12, GRAY_16, DO_FILL, vga_frame_buffer);
           vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );

      vid_draw_box(x+19,y+3,x+28,y+12, GRAY_16, DO_FILL, vga_frame_buffer);
          vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
           vid_draw_box(x+19,y+3,x+28,y+12, GRAY_16, DO_FILL, vga_frame_buffer);
          vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );


               vid_draw_box(x+3,y+19,x+12,y+28, GRAY_16, DO_FILL, vga_frame_buffer);
               vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
                 vid_draw_box(x+3,y+19,x+12,y+28, GRAY_16, DO_FILL, vga_frame_buffer);
               vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );


            vid_draw_box(x+19,y+19,x+28,y+28, GRAY_16, DO_FILL, vga_frame_buffer);
                  vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
            vid_draw_box(x+19,y+19,x+28,y+28, GRAY_16, DO_FILL, vga_frame_buffer);
                  vga_flip_frame_buffers( vga_frame_buffer );
               while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );



}
void map_wall(int i,int j,display_frame_buffer_struct* vga_frame_buffer)
{
        int x,y;
        x=30*i+15,y=30*j+15;
//        setfillstyle(1,13);
//        bar(x+1,y+1,x+30,y+30);
//        setcolor(15);
//        line(x+1,y+1,x+30,y+30);
//        line(x+1,y+30,x+30,y+1);
    vid_draw_box(x+1, y+1,x+30 , y+30, WHITE_16, DO_FILL, vga_frame_buffer);
            vga_flip_frame_buffers( vga_frame_buffer );
                while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_box(x+1, y+1,x+30 , y+30, WHITE_16, DO_FILL, vga_frame_buffer);
        //    vga_flip_frame_buffers( vga_frame_buffer );
        //        while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
}

void map_border(int i,int j,display_frame_buffer_struct* vga_frame_buffer)
{
        int x,y;x=30*i+15,y=30*j+15;
//        setfillstyle(1,8);
//        bar(x+1,y+1,x+30,y+30);
        vid_draw_box(x+1, y+1,x+30 , y+30, WHEAT_16, DO_FILL, vga_frame_buffer);
            vga_flip_frame_buffers( vga_frame_buffer );
                while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
            vid_draw_box(x+1, y+1,x+30 , y+30, WHEAT_16, DO_FILL, vga_frame_buffer);
                  vga_flip_frame_buffers( vga_frame_buffer );
              while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
}


void map_base_enemy(int i,int j,display_frame_buffer_struct* vga_frame_buffer)
{
    int x,y;x=30*i+15,y=30*j+15;
 //     setcolor(15);
//      circle(x+15,y+15,10);
//      circle(x+11,y+11,2);
//      circle(x+19,y+11,2);

   vid_draw_circle(x+15, y+15, 10, RED_16, DO_FILL, vga_frame_buffer);
      vga_flip_frame_buffers( vga_frame_buffer );
            while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_circle(x+15, y+15, 10, RED_16, DO_FILL, vga_frame_buffer);
      vga_flip_frame_buffers( vga_frame_buffer );
            while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );

}

void map_base_player(int i,int j,display_frame_buffer_struct* vga_frame_buffer)
{
    int x,y;x=30*i+15,y=30*j+15;

   vid_draw_circle(x+15, y+15, 10, GREEN_16, DO_FILL, vga_frame_buffer);
      vga_flip_frame_buffers( vga_frame_buffer );
            while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
   vid_draw_circle(x+15, y+15, 10, GREEN_16, DO_FILL, vga_frame_buffer);
      vga_flip_frame_buffers( vga_frame_buffer );
            while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );

}

void uptank(int i,int j,int color,display_frame_buffer_struct* vga_frame_buffer)

{
    int x,y;x=30*i+15,y=30*j+15;

//      setcolor(color);
//      line(x+1,y+4,x+1,y+27);
//      line(x+7,y+4,x+7,y+27);
//      line(x+24,y+4,x+24,y+27);
//      line(x+30,y+4,x+30,y+27);
//      arc(x+4,y+4,0,180,3);
//      arc(x+4,y+27,180,360,3);
//      arc(x+27,y+4,0,180,3);
//      arc(x+27,y+27,180,360,3);
//      allcircle(i,j);
//      setfillstyle(1,color);
//        bar(x+14,y+1,x+17,y+15);
//
        vid_draw_box(x+1,y+4,x+7,y+27, color, DO_FILL, vga_frame_buffer);
                vga_flip_frame_buffers( vga_frame_buffer );
                  while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_box(x+1,y+4,x+7,y+27, color, DO_FILL, vga_frame_buffer);
                vga_flip_frame_buffers( vga_frame_buffer );
                  while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_box(x+24,y+4,x+30,y+27, color, DO_FILL, vga_frame_buffer);
                  vga_flip_frame_buffers( vga_frame_buffer );
                  while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );

        vid_draw_box(x+24,y+4,x+30,y+27, color, DO_FILL, vga_frame_buffer);
                  vga_flip_frame_buffers( vga_frame_buffer );
                  while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );



          vid_draw_circle(x+15,y+14, 2, color, DO_FILL, vga_frame_buffer);
                vga_flip_frame_buffers( vga_frame_buffer );
                  while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
              vid_draw_circle(x+15,y+14, 2, color, DO_FILL, vga_frame_buffer);
                vga_flip_frame_buffers( vga_frame_buffer );
                while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );

      vid_draw_box(x+14,y+1,x+17,y+15, color, DO_FILL, vga_frame_buffer);
              vga_flip_frame_buffers( vga_frame_buffer );
                  while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
          vid_draw_box(x+14,y+1,x+17,y+15, color, DO_FILL, vga_frame_buffer);
          //    vga_flip_frame_buffers( vga_frame_buffer );
            //      while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
}


void downtank(int i,int j,int color,display_frame_buffer_struct* vga_frame_buffer)
{
      int x,y;x=30*i+15,y=30*j+15;
//        setcolor(color);
//        line(x+1,y+4,x+1,y+27);
//        line(x+7,y+4,x+7,y+27);
//        line(x+24,y+4,x+24,y+27);
//        line(x+30,y+4,x+30,y+27);
//        arc(x+4,y+4,0,180,3);
//        arc(x+4,y+27,180,360,3);
//        arc(x+27,y+4,0,180,3);
//       arc(x+27,y+27,180,360,3);
//       allcircle(i,j);
//       line(x+14,y+16,x+14,y+30);
//       line(x+15,y+16,x+15,y+30);
//       line(x+16,y+16,x+16,y+30);
//       line(x+17,y+16,x+17,y+30);
         vid_draw_box(x+1,y+4,x+7,y+27, color, DO_FILL, vga_frame_buffer);
             vga_flip_frame_buffers( vga_frame_buffer );
             while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
         vid_draw_box(x+1,y+4,x+7,y+27, color, DO_FILL, vga_frame_buffer);
             vga_flip_frame_buffers( vga_frame_buffer );
             while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
       vid_draw_box(x+24,y+4,x+30,y+27, color, DO_FILL, vga_frame_buffer);
             vga_flip_frame_buffers( vga_frame_buffer );
             while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
        vid_draw_box(x+24,y+4,x+30,y+27, color, DO_FILL, vga_frame_buffer);
             vga_flip_frame_buffers( vga_frame_buffer );
             while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
         vid_draw_circle(x+15,y+14, 2, color, DO_FILL, vga_frame_buffer);
         vga_flip_frame_buffers( vga_frame_buffer );
             while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
       vid_draw_circle(x+15,y+14, 2, color, DO_FILL, vga_frame_buffer);
            vga_flip_frame_buffers( vga_frame_buffer );
             while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
       vid_draw_box(x+14,y+16,x+17,y+30, color, DO_FILL, vga_frame_buffer);
             vga_flip_frame_buffers( vga_frame_buffer );
             while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
         vid_draw_box(x+14,y+16,x+17,y+30, color, DO_FILL, vga_frame_buffer);
           //    vga_flip_frame_buffers( vga_frame_buffer );
           //    while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 )
}
void lefttank(int i,int j,int color,display_frame_buffer_struct* vga_frame_buffer)
{
        int x,y;x=30*i+15,y=30*j+15;
//       setcolor(color);
//       line(x+4,y+1,x+27,y+1);
//       line(x+4,y+7,x+27,y+7);
//       line(x+4,y+24,x+27,y+24);
//       line(x+4,y+30,x+27,y+30);
//       arc(x+4,y+4,90,270,3);
//       arc(x+27,y+4,270,90,3);
//       arc(x+4,y+27,90,270,3);
//       arc(x+27,y+27,270,90,3);
//       line(x+15,y+14,x+1,y+14);
//      line(x+15,y+15,x+1,y+15);
//      line(x+15,y+16,x+1,y+16);
//      line(x+15,y+17,x+1,y+17);
//      allcircle(i,j);

        vid_draw_box(x+4,y+1,x+27,y+7, color, DO_FILL, vga_frame_buffer);
                vga_flip_frame_buffers( vga_frame_buffer );
              while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_box(x+4,y+1,x+27,y+7, color, DO_FILL, vga_frame_buffer);
              vga_flip_frame_buffers( vga_frame_buffer );
            while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );



      vid_draw_box(x+4,y+24,x+27,y+30, color, DO_FILL, vga_frame_buffer);
                vga_flip_frame_buffers( vga_frame_buffer );
                while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
          vid_draw_box(x+4,y+24,x+27,y+30, color, DO_FILL, vga_frame_buffer);
                vga_flip_frame_buffers( vga_frame_buffer );
                while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );

          vid_draw_circle(x+15,y+14, 2, color, DO_FILL, vga_frame_buffer);
          vga_flip_frame_buffers( vga_frame_buffer );
              while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
        vid_draw_circle(x+15,y+14, 2, color, DO_FILL, vga_frame_buffer);
          vga_flip_frame_buffers( vga_frame_buffer );
            while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_box(x+1,y+14,x+15,y+17, color, DO_FILL, vga_frame_buffer);
                vga_flip_frame_buffers( vga_frame_buffer );
              while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
        vid_draw_box(x+1,y+14,x+15,y+17, color, DO_FILL, vga_frame_buffer);
          //      vga_flip_frame_buffers( vga_frame_buffer );
            //    while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );

}

void righttank(int i,int j,int color,display_frame_buffer_struct* vga_frame_buffer)
{
        int x,y;x=30*i+15,y=30*j+15;
//      setcolor(color);
//      line(x+4,y+1,x+27,y+1);
//      line(x+4,y+7,x+27,y+7);
//      line(x+4,y+24,x+27,y+24);
//      line(x+4,y+30,x+27,y+30);
//      arc(x+4,y+4,90,270,3);
//       arc(x+27,y+4,270,90,3);
//       arc(x+4,y+27,90,270,3);
//       arc(x+27,y+27,270,90,3);
//       line(x+16,y+14,x+30,y+14);
//       line(x+16,y+15,x+30,y+15);
//       line(x+16,y+16,x+30,y+16);
//       line(x+16,y+17,x+30,y+17);
//       allcircle(i,j);

        vid_draw_box(x+4,y+1,x+27,y+7, color, DO_FILL, vga_frame_buffer);
            vga_flip_frame_buffers( vga_frame_buffer );
            while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
        vid_draw_box(x+4,y+1,x+27,y+7, color, DO_FILL, vga_frame_buffer);
            vga_flip_frame_buffers( vga_frame_buffer );
            while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_box(x+4,y+24,x+27,y+30, color, DO_FILL, vga_frame_buffer);
            vga_flip_frame_buffers( vga_frame_buffer );
           while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
         vid_draw_box(x+4,y+24,x+27,y+30, color, DO_FILL, vga_frame_buffer);
            vga_flip_frame_buffers( vga_frame_buffer );
           while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
        vid_draw_circle(x+15,y+14, 2, color, DO_FILL, vga_frame_buffer);
        vga_flip_frame_buffers( vga_frame_buffer );
           while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_circle(x+15,y+14, 2, color, DO_FILL, vga_frame_buffer);
        vga_flip_frame_buffers( vga_frame_buffer );
           while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );


      vid_draw_box(x+16,y+14,x+30,y+17, color, DO_FILL, vga_frame_buffer);
            vga_flip_frame_buffers( vga_frame_buffer );
        while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
      vid_draw_box(x+16,y+14,x+30,y+17, color, DO_FILL, vga_frame_buffer);
      //        vga_flip_frame_buffers( vga_frame_buffer );
       // while( IORD_32DIRECT( vga_frame_buffer->vga_controller_base, 0xC ) != (int)vga_frame_buffer->frame0 );
}
void judge_shoot(int m,int map[15][15],int i,display_frame_buffer_struct* vga_frame_buffer)
{
    char *string_life="\0";
       switch(m)
           {
               case 1 :shoot[i].life=0;
                       if(i>=1&&i<=2)
                       {
                           player[0].life=0;
```

```
                                player_life--;
                           if(player_life==1)
                                 {string_life="Player Life : 1\n";
                                  }
                            else if(player_life==2)
                            {
                                string_life="Player Life : 2\n";
                            }
                           else if(player_life==3)
                            {
                                string_life="Player Life : 3\n";

                             }

                          vid_draw_box(570, 80 , 600 , 100, BLACK_16, DO_FILL, vga_frame_buffer);
                          vga_flip_frame_buffers( vga_frame_buffer );
                         while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0 );
                         vid_draw_box(570, 80 , 600 , 100, BLACK_16, DO_FILL, vga_frame_buffer);
                           vga_flip_frame_buffers( vga_frame_buffer );
                         while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0 );

                      vid_print_string(470, 90, GREEN_16,cour10_font,vga_frame_buffer ,string_life);
                        vga_flip_frame_buffers( vga_frame_buffer );
                        while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0 );
                      vid_print_string(470, 90, GREEN_16,cour10_font,vga_frame_buffer ,string_life);
                        vga_flip_frame_buffers( vga_frame_buffer );
                        while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0 );
                                judge_shootway(map,i, vga_frame_buffer);
                            }
                            break;

                 case 5 :shoot[i].life=0;
                            judge_shootway(map,i, vga_frame_buffer);
                            break;
                 case 7 :shoot[i].life=1;
                            break;
                 case 9 :shoot[i].life=0;
                            if(i==0) enemynum=0;
                            break;
                case 10: shoot[i].life=0;
```

```
                                if((i>=1&&i<=2)) player_life=0;
                                 break;

                        case 44:shoot[i].life=0;
                                 if(i==0)
                                 {
                                        enemy[0].life=0;enemynum--;
                                        judge_shootway(map,i, vga_frame_buffer);
                                 }
                                 break;
                        case 45:shoot[i].life=0;
                                 if(i==0)
                                 {
                                        enemy[1].life=0;enemynum--;
                                        judge_shootway(map,i, vga_frame_buffer);
                                 }
                                 break;

                }
}

void enemy_control(int map[15][15],display_frame_buffer_struct* vga_frame_buffer)
{
        int i;
        for(i=0;i<3;i++)
                {
                        enemy[i].control=rand()%50;
                        if(enemy[i].life==1)
                                {
                                        if(enemy[i].control>=0&&enemy[i].control<=9&&map[enemy[i].j-1][enemy[i].i]==0)
                                                {
                                                        enemy[i].way=UPWAY;
                                                        map[enemy[i].j][enemy[i].i]=0;
                                                        blank(enemy[i].i,enemy[i].j,vga_frame_buffer);
                                                        enemy[i].j--;
                                                        uptank(enemy[i].i,enemy[i].j,RED_16,vga_frame_buffer);
                                                        map[enemy[i].j][enemy[i].i]=44+i;
                                                }
                                        else if(enemy[i].control>=10&&enemy[i].control<=19&&map[enemy[i].j+1][enemy[i].i]==0)
                                                {
                                                        enemy[i].way=DOWNWAY;
                                                        map[enemy[i].j][enemy[i].i]=0;
                                                        blank(enemy[i].i,enemy[i].j,vga_frame_buffer);
                                                        enemy[i].j++;
                                                        downtank(enemy[i].i,enemy[i].j,RED_16,vga_frame_buffer);
                                                        map[enemy[i].j][enemy[i].i]=44+i;
                                                    }
                                 else if(enemy[i].control>=20&&enemy[i].control<=29&&map[enemy[i].j][enemy[i].i-1]==0)
                                                    {
                                                        enemy[i].way=LEFTWAY;
                                                        map[enemy[i].j][enemy[i].i]=0;
                                                        blank(enemy[i].i,enemy[i].j,vga_frame_buffer);
                                                        enemy[i].i--;
                                                        lefttank(enemy[i].i,enemy[i].j,RED_16,vga_frame_buffer);
                                                        map[enemy[i].j][enemy[i].i]=44+i;
                                                    }
                                 else if(enemy[i].control>=30&&enemy[i].control<=39&&map[enemy[i].j][enemy[i].i+1]==0)
                                                    {
                                                        enemy[i].way=RIGHTWAY;
                                                        map[enemy[i].j][enemy[i].i]=0;
                                                        blank(enemy[i].i,enemy[i].j,vga_frame_buffer);
                                                        enemy[i].i++;
                                                        righttank(enemy[i].i,enemy[i].j,RED_16,vga_frame_buffer);
                                                        map[enemy[i].j][enemy[i].i]=44+i;
                                                    }
                                 else if(enemy[i].control>=40&&enemy[i].control<=49)
                                                    {
                                                        if(shoot[1+i].life==0&&enemy[i].life==1)
                                                            {
                                                                shoot[1+i].life=1;
                                                                shoot[1+i].x=enemy[i].i*30+15;    shoot[1+i].y=enemy[i].j*30+15;
                                                                shoot[1+i].way=enemy[i].way;break;
                                                            }
                                                    }
                                }
                }
}


void judge_shootway(int map[15][15],int i,display_frame_buffer_struct* vga_frame_buffer)
{
        switch(shoot[i].way)
                {
                        case
UPWAY:map[(shoot[i].y-15)/30-1][(shoot[i].x-15)/30]=0;blank((shoot[i].x-15)/30,(shoot[i].y-15)/30-1,vga_frame_buffer
);break;
                        case
DOWNWAY:map[(shoot[i].y-15)/30+1][(shoot[i].x-15)/30]=0;blank((shoot[i].x-15)/30,(shoot[i].y-15)/30+1,vga_frame_b
uffer);break;
            case
LEFTWAY:map[(shoot[i].y-15)/30][(shoot[i].x-15)/30-1]=0;blank((shoot[i].x-15)/30-1,(shoot[i].y-15)/30,vga_frame_buff
er);break;
            case
RIGHTWAY:map[(shoot[i].y-15)/30][(shoot[i].x-15)/30+1]=0;blank((shoot[i].x-15)/30+1,(shoot[i].y-15)/30,vga_frame_b
uffer);break;
        }
}

void judge_moveshootway(int map[15][15],int i,display_frame_buffer_struct* vga_frame_buffer)
{
    int shoot_color,per=9;
    if(i<1) shoot_color=GREEN_16;
    else    shoot_color=RED_16;
    switch(shoot[i].way)
        {
            case UPWAY :
                if((shoot[i].y-15)%30==0)
                    if(map[(shoot[i].y-15)/30-1][(shoot[i].x-15)/30]!=0)
                        {
                            shoot[i].life=0;
                            judge_shoot(map[(shoot[i].y-15)/30-1][(shoot[i].x-15)/30],map,i,vga_frame_buffer);
fengming(5000);
                        }
                if(shoot[i].y<47+5+per)
                    shoot[i].life=0;
                if(shoot[i].life!=0)
                    {
//                          setfillstyle(1,1);
//                          bar(shoot[i].x+14,shoot[i].y-5,shoot[i].x+17,shoot[i].y-2);
//                          setcolor(7);
//                          line(shoot[i].x+14,shoot[i].y-1,shoot[i].x+17,shoot[i].y-1);
                      vid_draw_box(shoot[i].x+14,shoot[i].y-4,shoot[i].x+17,shoot[i].y-1,   BLACK_16,   DO_FILL,
vga_frame_buffer);
                            vga_flip_frame_buffers( vga_frame_buffer );
                            while(   IORD_32DIRECT(   vga_frame_buffer->vga_controller_base,   0xC   )   !=
(int)vga_frame_buffer->frame0 );
                     vid_draw_box(shoot[i].x+14,shoot[i].y-4,shoot[i].x+17,shoot[i].y-1,   BLACK_16,   DO_FILL,
vga_frame_buffer);
                            vga_flip_frame_buffers( vga_frame_buffer );
                            while(   IORD_32DIRECT(   vga_frame_buffer->vga_controller_base,   0xC   )   !=
(int)vga_frame_buffer->frame0 );
```

```
                    vid_draw_box(shoot[i].x+14,shoot[i].y-5-per,shoot[i].x+17,shoot[i].y-2-per,   shoot_color,   DO_FILL,
vga_frame_buffer);
                              vga_flip_frame_buffers( vga_frame_buffer );
                              while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
                   vid_draw_box(shoot[i].x+14,shoot[i].y-5-per,shoot[i].x+17,shoot[i].y-2-per,    shoot_color,   DO_FILL,
vga_frame_buffer);
                         //      vga_flip_frame_buffers( vga_frame_buffer );
                        //             while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );


                   //        vid_draw_line(shoot[i].x+14,shoot[i].y-1,shoot[i].x+17,shoot[i].y-1,        1,                RED_16,
vga_frame_buffer);
                     //           vga_flip_frame_buffers( vga_frame_buffer );
                     //                     while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                    //       vid_draw_line(shoot[i].x+14,shoot[i].y-1,shoot[i].x+17,shoot[i].y-1,        1,               RED_16,
vga_frame_buffer);
                        //      vga_flip_frame_buffers( vga_frame_buffer );
                         //        while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0 );
                     shoot[i].y=shoot[i].y-per-1;
                    }
                else
                    {
                       // setfillstyle(1,getpixel(shoot[i].x+14,shoot[i].y-6));
                      //    bar(shoot[i].x+14,shoot[i].y-5,shoot[i].x+17,shoot[i].y-1);
                      if(shoot[i].y<50)
                         {    vid_draw_box(shoot[i].x+14,shoot[i].y-5,shoot[i].x+17,shoot[i].y-1, WHEAT_16, DO_FILL,
vga_frame_buffer);
                              vga_flip_frame_buffers( vga_frame_buffer );
                              while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
                         vid_draw_box(shoot[i].x+14,shoot[i].y-5,shoot[i].x+17,shoot[i].y-1,     WHEAT_16,     DO_FILL,
vga_frame_buffer);
                              // vga_flip_frame_buffers( vga_frame_buffer );
                            //     while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0 );
                         }
                       else
                       {     vid_draw_box(shoot[i].x+14,shoot[i].y-5,shoot[i].x+17,shoot[i].y-1,   BLACK_16,   DO_FILL,
vga_frame_buffer);
                              vga_flip_frame_buffers( vga_frame_buffer );
```

```
                                        while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
                                   vid_draw_box(shoot[i].x+14,shoot[i].y-5,shoot[i].x+17,shoot[i].y-1,      BLACK_16,      DO_FILL,
vga_frame_buffer);
                                        vga_flip_frame_buffers( vga_frame_buffer );
                                        while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
                              }
                         }
                    break;
               case DOWNWAY:
                    if((shoot[i].y-15)%30==0)
                         if(map[(shoot[i].y-15)/30+1][(shoot[i].x-15)/30]!=0)
                              {
                                   shoot[i].life=0;
                                   judge_shoot(map[(shoot[i].y-15)/30+1][(shoot[i].x-15)/30],map,i,vga_frame_buffer);
                                        fengming(5000);
                              }
                    if(shoot[i].y>403-per)
                         shoot[i].life=0;
                    if(shoot[i].life!=0)
                         {
//                                   setfillstyle(1,1);
//                                   bar(shoot[i].x+14,shoot[i].y+35,shoot[i].x+17,shoot[i].y+32);setcolor(7);
//                                   line(shoot[i].x+14,shoot[i].y+31,shoot[i].x+17,shoot[i].y+31);
//
                                   vid_draw_box(shoot[i].x+14,shoot[i].y+31,shoot[i].x+17,shoot[i].y+34,   BLACK_16,   DO_FILL,
vga_frame_buffer);
                                         vga_flip_frame_buffers( vga_frame_buffer );
                                        while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
                                    vid_draw_box(shoot[i].x+14,shoot[i].y+31,shoot[i].x+17,shoot[i].y+34,  BLACK_16,  DO_FILL,
vga_frame_buffer);
                                         vga_flip_frame_buffers( vga_frame_buffer );
                                        while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );


                                    vid_draw_box(shoot[i].x+14,shoot[i].y+32+per,shoot[i].x+17,shoot[i].y+35+per,   shoot_color,
DO_FILL, vga_frame_buffer);
                                         vga_flip_frame_buffers( vga_frame_buffer );
                                        while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
                                    vid_draw_box(shoot[i].x+14,shoot[i].y+32+per,shoot[i].x+17,shoot[i].y+35+per,   shoot_color,
DO_FILL, vga_frame_buffer);
```

```
                                    //      vga_flip_frame_buffers( vga_frame_buffer );
                                    //        while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );

                              //              vid_draw_line(shoot[i].x+14,shoot[i].y+31,shoot[i].x+17,shoot[i].y+31,    1,      RED_16,
vga_frame_buffer);
                                //      vga_flip_frame_buffers( vga_frame_buffer );
                               //                  while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                               //        vid_draw_line(shoot[i].x+14,shoot[i].y+31,shoot[i].x+17,shoot[i].y+31,    1,        RED_16,
vga_frame_buffer);
                                    //vga_flip_frame_buffers( vga_frame_buffer );
                                  //        while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                                 shoot[i].y= shoot[i].y+per+1;
                              }
                           else
                              {
                                //    setfillstyle(1,getpixel(shoot[i].x+14,shoot[i].y+36));
                                //    bar(shoot[i].x+14,shoot[i].y+35,shoot[i].x+17,shoot[i].y+31);

                                 vid_draw_box(shoot[i].x+14,shoot[i].y+31,shoot[i].x+17,shoot[i].y+35,    BLACK_16,    DO_FILL,
vga_frame_buffer);
                                      vga_flip_frame_buffers( vga_frame_buffer );
                                     while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0);
                               vid_draw_box(shoot[i].x+14,shoot[i].y+31,shoot[i].x+17,shoot[i].y+35,    BLACK_16,    DO_FILL,
vga_frame_buffer);
                                    //      vga_flip_frame_buffers( vga_frame_buffer );
                                  //          while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0);
                              }
                           break;
                        case LEFTWAY:
                           if((shoot[i].x-15)%30==0)
                              if(map[(shoot[i].y-15)/30][(shoot[i].x-15)/30-1]!=0)
                                 {
                                    shoot[i].life=0;
                                    judge_shoot(map[(shoot[i].y-15)/30][(shoot[i].x-15)/30-1],map,i,vga_frame_buffer);
                                       fengming(5000);


                                 }
                           if(shoot[i].x<48+5+per)
                              shoot[i].life=0;
```

```
                if(shoot[i].life!=0)
                    {
                     //     setfillstyle(1,1);
                     //     bar(shoot[i].x-5,shoot[i].y+14,shoot[i].x-2,shoot[i].y+17);setcolor(7);
                     //     line(shoot[i].x-1,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17);

                      vid_draw_box(shoot[i].x-4,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17,       BLACK_16,       DO_FILL,
vga_frame_buffer);
                              vga_flip_frame_buffers( vga_frame_buffer );
                             while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
                          vid_draw_box(shoot[i].x-4,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17,    BLACK_16,    DO_FILL,
vga_frame_buffer);
                          //    vga_flip_frame_buffers( vga_frame_buffer );
                            //while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );



                         vid_draw_box(shoot[i].x-5-per,shoot[i].y+14,shoot[i].x-2-per,shoot[i].y+17,          shoot_color,
DO_FILL, vga_frame_buffer);
                              vga_flip_frame_buffers( vga_frame_buffer );
                             while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0 );
                          vid_draw_box(shoot[i].x-5-per,shoot[i].y+14,shoot[i].x-2-per,shoot[i].y+17,        shoot_color,
DO_FILL, vga_frame_buffer);
                       //          vga_flip_frame_buffers( vga_frame_buffer );
                        //              while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );



                       //      vid_draw_line(shoot[i].x-1,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17,      2,            RED_16,
vga_frame_buffer);
                        //      vga_flip_frame_buffers( vga_frame_buffer );
                        //              while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );
                      //      vid_draw_line(shoot[i].x-1,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17,      2,            RED_16,
vga_frame_buffer);
                        //      vga_flip_frame_buffers( vga_frame_buffer );
                         //          while(    IORD_32DIRECT(    vga_frame_buffer->vga_controller_base,    0xC    )    !=
(int)vga_frame_buffer->frame0 );



                        shoot[i].x= shoot[i].x-per-1;
                    }
                else
                    {
                      //    setfillstyle(1,getpixel(shoot[i].x-6,shoot[i].y+14));
                      //    bar(shoot[i].x-5,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17);
                        if(shoot[i].x<50)
                     {      vid_draw_box(shoot[i].x-5,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17,    WHEAT_16,    DO_FILL,
vga_frame_buffer);
                              vga_flip_frame_buffers( vga_frame_buffer );
                             while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0);
                        vid_draw_box(shoot[i].x-5,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17,    WHEAT_16,    DO_FILL,
vga_frame_buffer);
                         //      vga_flip_frame_buffers( vga_frame_buffer );
                          //     while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0);
                      }
                      else
                        {vid_draw_box(shoot[i].x-5,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17,     BLACK_16,     DO_FILL,
vga_frame_buffer);
                              vga_flip_frame_buffers( vga_frame_buffer );
                             while(      IORD_32DIRECT(      vga_frame_buffer->vga_controller_base,      0xC      )      !=
(int)vga_frame_buffer->frame0);
                        vid_draw_box(shoot[i].x-5,shoot[i].y+14,shoot[i].x-1,shoot[i].y+17,     BLACK_16,     DO_FILL,
vga_frame_buffer);
                         //      vga_flip_frame_buffers( vga_frame_buffer );
                          //     while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0);
                        }
                    }
                break;

            case RIGHTWAY:
                if((shoot[i].x-15)%30==0)
                    if(map[(shoot[i].y-15)/30][(shoot[i].x-15)/30+1]!=0)
                        {
                            shoot[i].life=0;
                            judge_shoot(map[(shoot[i].y-15)/30][(shoot[i].x-15)/30+1],map,i,vga_frame_buffer);
                                fengming(5000);
                        }
                if(shoot[i].x>402-per)
                    shoot[i].life=0;
                if(shoot[i].life!=0)
                    {
                      //    setfillstyle(1,1);
                //bar(shoot[i].x+35,shoot[i].y+14,shoot[i].x+32,shoot[i].y+17);setcolor(7);
                //line(shoot[i].x+31,shoot[i].y+14,shoot[i].x+31,shoot[i].y+17);

                vid_draw_box(shoot[i].x+31,shoot[i].y+14,shoot[i].x+34,shoot[i].y+17,   BLACK_16,   DO_FILL,
vga_frame_buffer);
                    vga_flip_frame_buffers( vga_frame_buffer );
                  while(   IORD_32DIRECT(   vga_frame_buffer->vga_controller_base,   0xC   )   !=
(int)vga_frame_buffer->frame0 );
                vid_draw_box(shoot[i].x+31,shoot[i].y+14,shoot[i].x+34,shoot[i].y+17,   BLACK_16,   DO_FILL,
vga_frame_buffer);
                    vga_flip_frame_buffers( vga_frame_buffer );
                  while(   IORD_32DIRECT(   vga_frame_buffer->vga_controller_base,   0xC   )   !=
(int)vga_frame_buffer->frame0 );



                 vid_draw_box(shoot[i].x+32+per,shoot[i].y+14,shoot[i].x+35+per,shoot[i].y+17,  shoot_color,
DO_FILL, vga_frame_buffer);
                    vga_flip_frame_buffers( vga_frame_buffer );
                  while(   IORD_32DIRECT(   vga_frame_buffer->vga_controller_base,   0xC   )   !=
(int)vga_frame_buffer->frame0 );
                vid_draw_box(shoot[i].x+32+per,shoot[i].y+14,shoot[i].x+35+per,shoot[i].y+17,   shoot_color,
DO_FILL, vga_frame_buffer);
                 //    vga_flip_frame_buffers( vga_frame_buffer );
                 //      while(   IORD_32DIRECT(   vga_frame_buffer->vga_controller_base,   0xC   )   !=
(int)vga_frame_buffer->frame0 );



              //     vid_draw_line(shoot[i].x+31,shoot[i].y+14,shoot[i].x+31,shoot[i].y+17,   2,     RED_16,
vga_frame_buffer);
              //     vga_flip_frame_buffers( vga_frame_buffer );
               //          while(   IORD_32DIRECT(   vga_frame_buffer->vga_controller_base,   0xC   )   !=
(int)vga_frame_buffer->frame0 );
             //     vid_draw_line(shoot[i].x+31,shoot[i].y+14,shoot[i].x+31,shoot[i].y+17,   2,     RED_16,
vga_frame_buffer);
               //    vga_flip_frame_buffers( vga_frame_buffer );
                //        while(   IORD_32DIRECT(   vga_frame_buffer->vga_controller_base,   0xC   )   !=
(int)vga_frame_buffer->frame0 );
                shoot[i].x=shoot[i].x+per+1;
              }
            else
              {
        //            setfillstyle(1,getpixel(shoot[i].x+36,shoot[i].y+14));
           //       bar(shoot[i].x+35,shoot[i].y+14,shoot[i].x+31,shoot[i].y+17);
```

```
                                        vid_draw_box(shoot[i].x+31,shoot[i].y+14,shoot[i].x+35,shoot[i].y+17,   BLACK_16,   DO_FILL,
vga_frame_buffer);
                                                  vga_flip_frame_buffers( vga_frame_buffer );
                                                 while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0);
                                     vid_draw_box(shoot[i].x+31,shoot[i].y+14,shoot[i].x+35,shoot[i].y+17,     BLACK_16,     DO_FILL,
vga_frame_buffer);
                                        //        vga_flip_frame_buffers( vga_frame_buffer );
                                        //         while(     IORD_32DIRECT(     vga_frame_buffer->vga_controller_base,     0xC     )     !=
(int)vga_frame_buffer->frame0);
                                   }
                          break;
                 }
}



void judge_tank_my(int map[15][15],int key,display_frame_buffer_struct* vga_frame_buffer)
{
        switch(key)
                {
                case a_UP:
                        if(player[0].life==1){
                        if(map[player[0].j-1][player[0].i]==0)
                                {
                                        if(player[0].way==UPWAY)
                                                {
                                                        map[player[0].j][player[0].i]=0;
                                                        blank(player[0].i,player[0].j,vga_frame_buffer);
                                                        player[0].j--;
                                                        uptank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);
                                                        map[player[0].j][player[0].i]=1;
                                                        player[0].way=UPWAY;
                                                }
                                        else
                                                {
                                                        blank(player[0].i,player[0].j,vga_frame_buffer);
                                                        uptank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);
                                                        player[0].way=UPWAY;
                                                }
                                }
                        else
                                {
                                        blank(player[0].i,player[0].j,vga_frame_buffer);
```

```
                    uptank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);player[0].way=UPWAY;
                }
                        fengming(100);
            }
            break;
        case a_DOWN:
            if(player[0].life==1){
            if(map[player[0].j+1][player[0].i]==0)
                {
                    if(player[0].way==DOWNWAY)
                        {
                            map[player[0].j][player[0].i]=0;
                            blank(player[0].i,player[0].j,vga_frame_buffer);
                            player[0].j++;
                            downtank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);
                            map[player[0].j][player[0].i]=1;
                            player[0].way=DOWNWAY;
                        }
                    else
                        {
                            blank(player[0].i,player[0].j,vga_frame_buffer);

downtank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);player[0].way=DOWNWAY;
                        }
                }
            else
                {

blank(player[0].i,player[0].j,vga_frame_buffer);downtank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);player[0].w
ay=DOWNWAY;
                }
                    fengming(100);
            }
            break;
        case a_LEFT:
            if(player[0].life==1){
            if(map[player[0].j][player[0].i-1]==0)
                {
                    if(player[0].way==LEFTWAY)
                        {
                            map[player[0].j][player[0].i]=0;
                            blank(player[0].i,player[0].j,vga_frame_buffer);
                            player[0].i--;
                            lefttank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);
                            map[player[0].j][player[0].i]=1;
                            player[0].way=LEFTWAY;
                        }
                    else
                        {

blank(player[0].i,player[0].j,vga_frame_buffer);lefttank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);player[0].way
=LEFTWAY;
                        }
                }
            else
                {

blank(player[0].i,player[0].j,vga_frame_buffer);lefttank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);player[0].way
=LEFTWAY;
                } fengming(100);
                }


            break;
        case a_RIGHT:
            if(player[0].life==1){
            if(map[player[0].j][player[0].i+1]==0)
                {
                    if(player[0].way==RIGHTWAY)
                        {
                            map[player[0].j][player[0].i]=0;
                            blank(player[0].i,player[0].j,vga_frame_buffer);
                            player[0].i++;
                            righttank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);
                            map[player[0].j][player[0].i]=1;
                            player[0].way=RIGHTWAY;
                        }
                    else
                        {

blank(player[0].i,player[0].j,vga_frame_buffer);righttank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);player[0].wa
y=RIGHTWAY;
                        }
                }
            else
                {

blank(player[0].i,player[0].j,vga_frame_buffer);righttank(player[0].i,player[0].j,GREEN_16,vga_frame_buffer);player[0].wa
y=RIGHTWAY;
```

```
                        }

                     fengming(100);
                  }
                  break;
              case a_shoot:
                  if(player[0].life==1){
                  if(shoot[0].life==0)
                        {
                              shoot[0].life=1;
                              shoot[0].x=player[0].i*30+15;    shoot[0].y=player[0].j*30+15;
                              shoot[0].way=player[0].way;

                              break;
                        } fengming(500);
                        }
                        break;

                 player_life=0;
      }
}

void fengming(int n)
{
      int j;
      for(j=0;j<100;j++)
          {
          IOWR_ALTERA_AVALON_PIO_DATA(BUZZER_BASE,0x1);
          usleep(n);
          IOWR_ALTERA_AVALON_PIO_DATA(BUZZER_BASE,0x0);
          usleep(n);
          }
}
```

Verilog Part

VGA.v

```
//Legal Notice: (C)2015 Altera Corporation. All rights reserved.    Your
//use of Altera Corporation's design tools, logic functions and other
//software and tools, and its AMPP partner logic functions, and any
//output files any of the foregoing (including device programming or
//simulation files), and any associated documentation or information are
//expressly subject to the terms and conditions of the Altera Program
```

```
//License Subscription Agreement or other applicable license agreement,
//including, without limitation, that your use is for the sole purpose
//of programming logic devices manufactured by Altera and sold by Altera
//or its authorized distributors.    Please refer to the applicable
//agreement for further details.

// synthesis translate_off
`timescale 1ns / 1ps
// synthesis translate_on

// turn off superfluous verilog processor warnings
// altera message_level Level1
// altera message_off 10034 10035 10036 10037 10230 10240 10030

module VGA (
              // inputs:
               clk,
               master_data_valid,
               master_readdata,
               master_waitrequest,
               reset_n,
               slave_address,
               slave_chipselect,
               slave_write,
               slave_writedata,
               vga_clk,

              // outputs:
               B,
               G,
               M1,
               M2,
               R,
               blank_n,
               hsync,
               master_address,
               master_read,
               slave_readdata,
               sync_n,
               sync_t,
               vsync
            )
;
```

```
output   [  7: 0] B;
output   [  7: 0] G;
output            M1;
output            M2;
output   [  7: 0] R;
output            blank_n;
output            hsync;
output   [ 31: 0] master_address;
output            master_read;
output   [ 31: 0] slave_readdata;
output            sync_n;
output            sync_t;
output            vsync;
input             clk;
input             master_data_valid;
input    [ 31: 0] master_readdata;
input             master_waitrequest;
input             reset_n;
input    [  1: 0] slave_address;
input             slave_chipselect;
input             slave_write;
input    [ 31: 0] slave_writedata;
input             vga_clk;

reg      [  7: 0] B;
reg      [  7: 0] G;
wire              M1;
reg               M2;
reg      [  7: 0] R;
wire     [ 31: 0] address_counter;
wire              address_counter_incr;
wire              address_counter_sload;
wire     [ 29: 0] address_counter_temp;
reg               blank_n;
wire     [  9: 0] column_counter;
wire     [  2: 0] config_counter;
wire              ctrl_reg_go_bit;
reg      [ 31: 0] current_dma;
wire              display_active;
reg      [ 31: 0] dma_modulus_reg;
reg      [ 31: 0] dma_source_reg;
wire              empty_the_fifo;
wire     [ 31: 0] fifo_data_in;
wire     [ 31: 0] fifo_data_out;
```

```
reg                 fifo_emptied;
reg                 fifo_has_data;
reg                 fifo_has_data_reg1;
reg                 fifo_has_room;
reg                 fifo_has_room_reg1;
wire                fifo_rdempty;
wire                fifo_read_req;
wire     [ 11: 0] fifo_used;
wire                fifo_write_clk;
wire                fifo_write_req;
reg                 go_bit;
reg                 go_bit_vga;
reg                 go_bit_vga_reg1;
reg                 hblank;
wire                hsync;
wire                hsync_temp;
reg      [ 31: 0] last_dma_addr_reg;
wire     [ 31: 0] master_address;
wire                master_read;
reg                 mux_toggle;
wire                read_16b;
wire     [  9: 0] row_counter;
reg      [ 31: 0] slave_control_reg;
wire     [ 31: 0] slave_readdata;
wire                stop_config_counter;
reg                 sync_n;
reg                 sync_n_init;
reg                 sync_t;
reg                 vblank;
wire     [ 15: 0] vga_16bit_out;
reg                 vga_start;
wire                vsync;
wire                vsync_temp;
dcfifo the_dcfifo
  (
    .aclr (!reset_n),
    .data (fifo_data_in),
    .q (fifo_data_out),
    .rdclk (vga_clk),
    .rdempty (fifo_rdempty),
    .rdreq (fifo_read_req),
    .wrclk (fifo_write_clk),
    .wrreq (fifo_write_req),
    .wrusedw (fifo_used)
  );

defparam the_dcfifo.LPM_NUMWORDS = 4096,
         the_dcfifo.LPM_SHOWAHEAD = "ON",
         the_dcfifo.LPM_WIDTH = 32;

assign fifo_write_clk = clk;
assign fifo_data_in = master_readdata;
always @(posedge clk or negedge reset_n)
  begin
    if (reset_n == 0)
        fifo_has_room_reg1 <= 0;
    else
      fifo_has_room_reg1 <= fifo_used < 3584;
  end


always @(posedge clk or negedge reset_n)
  begin
    if (reset_n == 0)
        fifo_has_room <= 0;
    else
      fifo_has_room <= fifo_has_room_reg1;
  end


always @(posedge vga_clk or negedge reset_n)
  begin
    if (reset_n == 0)
        fifo_has_data_reg1 <= 0;
    else
      fifo_has_data_reg1 <= fifo_used > 2048;
  end


always @(posedge vga_clk or negedge reset_n)
  begin
    if (reset_n == 0)
        fifo_has_data <= 0;
    else
      fifo_has_data <= fifo_has_data_reg1;
  end
```

```
lpm_counter dma_address_counter
   (
      .aclr (!reset_n),
      .clock (clk),
      .cnt_en (address_counter_incr),
      .data (dma_source_reg[31 : 2]),
      .q (address_counter_temp),
      .sload (address_counter_sload)
   );

defparam dma_address_counter.LPM_WIDTH = 30;

assign address_counter_incr = (master_read == 1) && (master_waitrequest == 0) && (go_bit == 1);
assign address_counter_sload = (go_bit == 0) ||(address_counter_incr && (address_counter == last_dma_addr_reg));
assign address_counter = {address_counter_temp, 2'b00};
always @(posedge clk or negedge reset_n)
   begin
      if (reset_n == 0)
            last_dma_addr_reg <= 0;
      else if (address_counter_sload)
            last_dma_addr_reg <= dma_source_reg + dma_modulus_reg - 4;
   end


lpm_counter vga_column_counter
   (
      .aclr (!reset_n),
      .clock (vga_clk),
      .cnt_en (vga_start),
      .q (column_counter),
      .sclr (!vga_start)
   );

defparam vga_column_counter.LPM_MODULUS = 800,
              vga_column_counter.LPM_WIDTH = 10;

lpm_counter vga_row_counter
   (
      .aclr (!reset_n),
      .clock (vga_clk),
      .cnt_en (vga_start && (column_counter == (800 - 1))),
      .q (row_counter),
      .sclr (!vga_start)
   );
```

```
defparam vga_row_counter.LPM_MODULUS = 524,
              vga_row_counter.LPM_WIDTH = 10;

always @(posedge clk or negedge reset_n)
   begin
      if (reset_n == 0)
            go_bit <= 0;
      else
         go_bit <= ctrl_reg_go_bit & stop_config_counter & fifo_emptied;
   end


assign stop_config_counter = config_counter == 3'b101;
lpm_counter vga_config_counter
   (
      .aclr (!reset_n),
      .clock (vga_clk),
      .cnt_en (!stop_config_counter),
      .q (config_counter)
   );

defparam vga_config_counter.LPM_WIDTH = 3;

always @(posedge vga_clk or negedge reset_n)
   begin
      if (reset_n == 0)
            sync_n_init <= 0;
      else
         sync_n_init <= config_counter[2] | (config_counter[0] & !config_counter[1]);
   end


always @(posedge vga_clk or negedge reset_n)
   begin
      if (reset_n == 0)
            M2 <= 0;
      else
         M2 <= config_counter[1];
   end


assign M1 = 0;
always @(posedge vga_clk or negedge reset_n)
```

```
  begin
    if (reset_n == 0)
        fifo_emptied <= 0;
    else
      fifo_emptied <= ctrl_reg_go_bit & (fifo_emptied | fifo_rdempty);
  end


assign empty_the_fifo = !fifo_emptied;
always @(posedge vga_clk or negedge reset_n)
  begin
    if (reset_n == 0)
        go_bit_vga_reg1 <= 0;
    else
      go_bit_vga_reg1 <= go_bit;
  end


always @(posedge vga_clk or negedge reset_n)
  begin
    if (reset_n == 0)
        go_bit_vga <= 0;
    else
      go_bit_vga <= go_bit_vga_reg1;
  end


always @(posedge vga_clk or negedge reset_n)
  begin
    if (reset_n == 0)
        vga_start <= 0;
    else
      vga_start <= (vga_start & go_bit_vga) | (fifo_has_data & go_bit_vga);
  end


assign vga_16bit_out = mux_toggle ? fifo_data_out[31 : 16] : fifo_data_out[15 : 0];
always @(posedge vga_clk or negedge reset_n)
  begin
    if (reset_n == 0)
        mux_toggle <= 0;
    else
      mux_toggle <= (!mux_toggle) & (read_16b);
  end
```

```
assign fifo_read_req = (mux_toggle & read_16b) | empty_the_fifo;
assign read_16b = hblank & vblank;
assign display_active = read_16b;
always @(posedge vga_clk or negedge reset_n)
   begin
      if (reset_n == 0)
            R <= 0;
      else
         R <= display_active ? ({vga_16bit_out[15 : 11], 3'b111}) : 8'b00000000;
   end


always @(posedge vga_clk or negedge reset_n)
   begin
      if (reset_n == 0)
            G <= 0;
      else
         G <= display_active ? ({vga_16bit_out[10 : 5], 2'b11}) : 8'b00000000;
   end


always @(posedge vga_clk or negedge reset_n)
   begin
      if (reset_n == 0)
            B <= 0;
      else
         B <= display_active ? ({vga_16bit_out[4 : 0], 3'b111}) : 8'b00000000;
   end


always @(posedge vga_clk or negedge reset_n)
   begin
      if (reset_n == 0)
            sync_n <= 0;
      else
         sync_n <= vga_start ? (vsync_temp ~^ hsync_temp) : sync_n_init;
   end


always @(posedge vga_clk or negedge reset_n)
   begin
      if (reset_n == 0)
```

```
            sync_t <= 0;
        else
          sync_t <= vga_start ? 0 : 0;
    end


always @(posedge vga_clk or negedge reset_n)
    begin
        if (reset_n == 0)
            blank_n <= 0;
        else
          blank_n <= display_active;
    end


always @(posedge vga_clk or negedge reset_n)
    begin
        if (reset_n == 0)
            vblank <= 0;
        else
          vblank <= (row_counter >= (33)) && (row_counter < (513));
    end


assign vsync_temp = (row_counter >= (2)) && (row_counter < (524));
lpm_shiftreg vsync_delay
    (
        .aclr (!reset_n),
        .clock (vga_clk),
        .shiftin (vsync_temp),
        .shiftout (vsync)
    );

defparam vsync_delay.LPM_WIDTH = 8;

always @(posedge vga_clk or negedge reset_n)
    begin
        if (reset_n == 0)
            hblank <= 0;
        else
          hblank <= (column_counter >= (144)) && (column_counter < (784));
    end
```

```
assign hsync_temp = (column_counter >= (96)) && (column_counter < (800));
lpm_shiftreg hsync_delay
   (
      .aclr (!reset_n),
      .clock (vga_clk),
      .shiftin (hsync_temp),
      .shiftout (hsync)
   );

defparam hsync_delay.LPM_WIDTH = 8;

assign slave_readdata = ((slave_address == 0))? slave_control_reg :
   ((slave_address == 1))? dma_source_reg :
   ((slave_address == 2))? dma_modulus_reg :
   ((slave_address == 3))? current_dma :
   slave_control_reg;

always @(posedge clk or negedge reset_n)
   begin
      if (reset_n == 0)
            slave_control_reg <= 0;
      else if (slave_write && slave_chipselect && (slave_address == 0))
            slave_control_reg <= slave_writedata;
   end
assign ctrl_reg_go_bit = slave_control_reg[0];
always @(posedge clk or negedge reset_n)
   begin
      if (reset_n == 0)
            dma_source_reg <= 0;
      else if (slave_write && slave_chipselect && (slave_address == 1))
            dma_source_reg <= slave_writedata;
   end
always @(posedge clk or negedge reset_n)
   begin
      if (reset_n == 0)
            dma_modulus_reg <= 0;
      else if (slave_write && slave_chipselect && (slave_address == 2))
            dma_modulus_reg <= slave_writedata;
   end
always @(posedge clk or negedge reset_n)
   begin
      if (reset_n == 0)
            current_dma <= 0;
      else if (address_counter_sload)
```

```
                current_dma <= dma_source_reg;
      end
   assign master_address = address_counter;
   assign master_read = fifo_has_room & go_bit;
   assign fifo_write_req = master_data_valid & go_bit;
   //s1, which is an e_avalon_slave
   //m1, which is an e_avalon_master

endmodule
```

Keyboard.v

```
//Legal Notice: (C)2015 Altera Corporation. All rights reserved.    Your
//use of Altera Corporation's design tools, logic functions and other
//software and tools, and its AMPP partner logic functions, and any
//output files any of the foregoing (including device programming or
//simulation files), and any associated documentation or information are
//expressly subject to the terms and conditions of the Altera Program
//License Subscription Agreement or other applicable license agreement,
//including, without limitation, that your use is for the sole purpose
//of programming logic devices manufactured by Altera and sold by Altera
//or its authorized distributors.    Please refer to the applicable
//agreement for further details.

// synthesis translate_off
`timescale 1ns / 1ps
// synthesis translate_on

// turn off superfluous verilog processor warnings
// altera message_level Level1
// altera message_off 10034 10035 10036 10037 10230 10240 10030

module PS2_CLK (
                          // inputs:
                           address,
                           clk,
                           in_port,
                           reset_n,

                          // outputs:
                           readdata
                        )
;
```

```
   output                  readdata;
   input     [    1: 0] address;
   input                   clk;
   input                   in_port;
   input                   reset_n;

   wire                    clk_en;
   wire                    data_in;
   wire                    read_mux_out;
   reg                     readdata;
   assign clk_en = 1;
   //s1, which is an e_avalon_slave
   assign read_mux_out = {1 {(address == 0)}} & data_in;
   always @(posedge clk or negedge reset_n)
      begin
         if (reset_n == 0)
               readdata <= 0;
         else if (clk_en)
               readdata <= read_mux_out;
      end
   assign data_in = in_port;
endmodule
```