CSEE 4840 Embedded System Design

Phoenix on the DE1-SOC Board

Vaishnavi Murthy, Ignacio Ramirez, Brianna Williams

Columbia University, Spring 2020

Contents

1 Introduction

Phoenix is a space-themed “slide and shoot” game originally developed by Taito and Amstar Electronics in the early 1980s. It is known for having multiple levels and enemy variants at different points in the gameplay in addition to being one of the first games to have a video game boss at the final stage. Another unique feature of the game is also that it was one of the first to have continuously playing music during the initial levels, when in-game music was not common. In our project, we will implement one level of this game, which consists of the player destroying a formation of alien birds (as shown in Fig. 1) that move in a patterned, but somewhat unpredictable manner. Some of the birds in the formation will also shoot downward “kamikaze style” in an attempt to crash into the player’s spaceship.

2 System Architecture

The basic system architecture for our game is shown in the figure below.

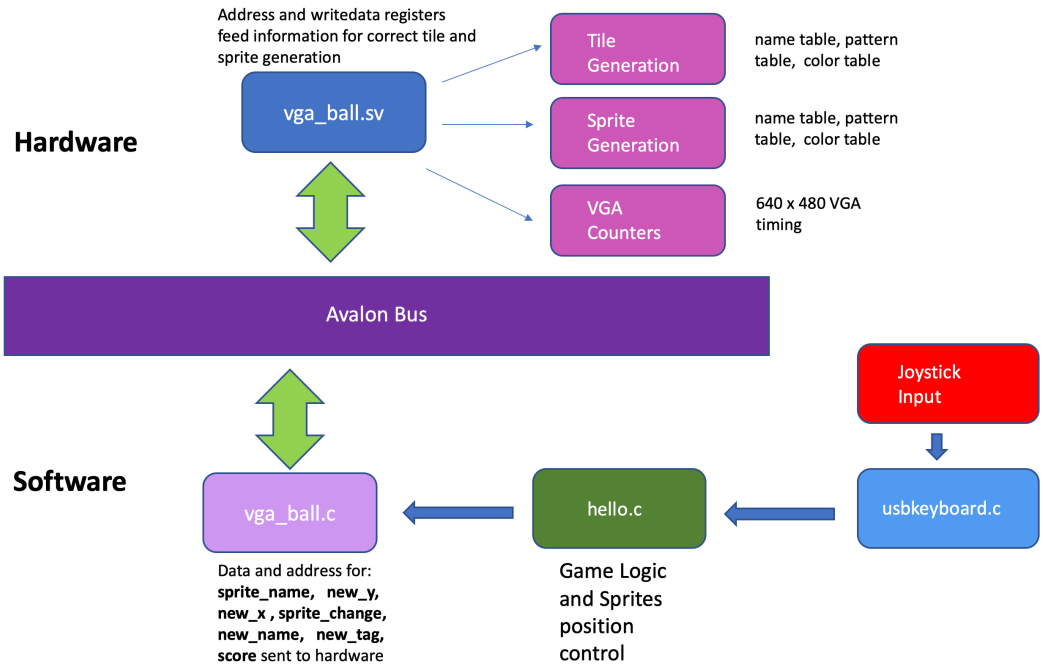

Figure 1. System Architecture

The player uses a joystick to control the horizontal position of the ship on the screen. On the joystick there is also a button, through which the player can shoot bullets at the enemy birds. The input is then registered by the software through the libusb library. That information is then fed to the main logic for our game that lies in the hello.c code. This code also contains information to control the positions of the sprites on the screen and which sprites appear at certain times. Next, vga_ball.c serves as the driver that communicates sprite and score information to the peripheral. The avalon bus is the interface between hardware and software, through which sprite and score information is delivered to the hardware module, vga_ball.sv, which creates the sprites and tiles that appear on the display. The sprite and tile generation are submodules located within vga_ball.sv, as is the submodule that controls the VGA display timing called vga_counters.

3 Hardware Design

To achieve the game display, we have designed an interface that communicates with the hardware to produce the color distribution shown below. The display consists of both characters that are consistently present as they move around the screen and the background setting that is designed to look as though the characters are in space. To determine what each pixel was to represent, we have designed 3 modules for this interface: the top module, established as the vga_ball module utilized in our third classroom assignment, the tile module, and the sprites module. The computation results in our display.

Figure 2. Graphics Display

The vga_ball module employs the tile and sprite modules to feed the correct color of each pixel on the screen. This module receives a possible color configuration from the sprite and tile modules at every vertical and horizontal placement combination that is available on the screen. To determine which is ultimately the one displayed, an indicator bit coming from the sprite module is scanned for whether the output color is one of a sprite or not. As sprites take precedence over the background (the sprite picture is placed on top of the background), this bit then tells us what color bits to assign to the vga color output of {VGA_R, VGA_G, VGA_B}. A description of each module will give further insight into its purpose for the display.

The tile module establishes the background pattern and color combination that is needed for the display of the game. To obtain the color at the pixel of the display's choosing, a pattern name table is first accessed to determine the type of pattern to be displayed. This allowed us to create tiles, or a set pattern of pixels that can be placed as a group on the screen. The tiles

allowed for greater control in location of color changes and better flexibility in design, as we were able to build more complex designs by controlling placement. With the tile design, we were able to create set blocks for letters and numbers, which are used to establish the lives of the player and the score. Our tiles are designed as a group of eight by eight pixels, and once accessed, can be indexed into to find the row of the set pattern defined in the pattern generator table. With the row and the column, we can find the intended color of the pixel with the color table, which is returned as output for possible analysis.

The sprite module establishes the colors of the characters that are animated within the game. These colors are found in the same workflow as the tiles, where a table is accessed to receive the name of the tiles, and this name will point to its complete design and the colors needed for the design. To determine what sprite will be displayed, the vertical and horizontal positions are checked for a desired placement of s sprite in the sprite attribute table. If a placement match is found at any point, the name of that sprite is stored for further accessing its

pattern and colors. Each sprite designed for our game is 32 by 32 pixels, enlarged for visual enjoyment. We have allowed the sprite to be represented in 4 colors, by assigning the color of a particular pixel using bit patterns rather than just present bits. If animation of the sprites were desired, the static sprite picture was slightly modified, and then the 2 sprite pictures were flipped back and forth to simulate movement. This especially pertained to the enemy birds of the game. The placements of the sprites on the screen were controlled by software using the avalon bus.

At every pixel, these modules are accessed and output a possible color combination to be displayed. The VGA then decided based on the sprite indicator bit which one actually gets displayed to the screen. This process gives us our display.

4 Software

4.1. User input

To achieve the goals of the game, the user is able to move the player ship around on the screen. This movement is controlled by a joystick, which we have provided for player usage. The joystick provides input in terms of signaling when physical buttons are pressed or not, which we can therefore translate into a movement on the screen. To receive this input, the libusb library is used to find, open, and return the raw data outputted by the joystick. Locating the joystick within the system was completed using the supplementary file usbkeyboard.h, where we were able to find the usb connected device based on its unique interface protocol. With this device, we were able to open and parse input using the interrup_transfer method of the library, and based on the signals received when the joystick is connected, assign these called signals to the movements created as a result. This input would additionally allow us to account for multiple signals, i.e. moving and shooting together for example, as each button was found to affect a separate keycode. This functionality allows the user to interact with the game.

4.2. Communicating with hardware code

There are 6 variables that are passed from software to the hardware through the ioctl, using the sprite_change_t struct in vga_ball.h:

1. Sprite_change[1:0] - if 1 indicates to the hardware code that new values have been written to the registers.
2. Sprite_num[7:0] - The index in the sprite attribute table that corresponds to the sprite we are changing. For example, the ship sprite entry is located in indexes 0-4 in the sprite attribute table, so the value of sprite_num for the ship sprite is 0. Another example, the ship bullet sprite entry is located in indexes 4-8 in the sprite attribute table, so the value of sprite_num to indicate we are changing information for the ship bullet is 1.
3. New_x[7:0] - the new x coordinate for the sprite. The 8 bits are compared to hcount[10:3].
4. New_y[7:0] - the new y coordinate for the sprite. The 8 bits are compared to vcount[9:1]
5. Sprite_name[7:0] - the index in the pattern table that contains the sprite image that is assigned to this sprite entry. For example, the name for the ship sprite (the information for this sprite is in indexes 0-4 in the sprite attribute table. The name for this sprite is in index 3) is usually 0, so it points to 0*32 = index 0 in the pattern table (indexes 0-32 contain the image information for the ship sprite), but when the ship sprite needs to explode, then the name of the ship sprite is changed to 5, which points to index 5*32 = 160 in the pattern table, since indexes 160-192 in the pattern table contain the explosion image.

6. New_tag[7:0] - the tag is stored in the fourth index for each sprite entry in the sprite attribute table. The tag is used as a boolean (1 or 0) to represent if the sprite is displayed on the screen or not.

4.3. Game Logic

The player's ship can only move left or right in the bottom row of the monitor. That is, when vcount[9:0] = "01 1110 0000" then the ship sprite is displayed as long as hcount[10:0] is less than "101 0000 0000". The ship can shoot a bullet at any time, when the player presses the button on the joystick. However, it only has one bullet sprite assigned, so until the previous bullet shot has disappeared through the top of the screen or hit a bird, pressing the button in the joystick does nothing. One thread handles joystick input and movement of the ship sprite. Another thread handles movement for the ship bullet and calculates whether the bullet has collided with any of the birds.

The bird sprites move in unison from the left side of the screen to the right side, and backwards. The three bird sprites move one next to the other. The bird sprite name changes to point in the pattern table to the bird facing left (when it moves left) to the bird facing forward (when it stands still or is turning around) and to the bird facing right (when it moves right). They also randomly change rows vertically, in unison. Each bird has a bullet sprite assigned. The birds will shoot whenever the ship is directly underneath them. However, like the ship, they can each only shoot one bullet at a time, so they have to wait for their respective bullet sprite to disappear or hit the ship before shooting again. All the bullets can only move vertically at a constant speed once they are shot. One thread handles the movement of all the bird sprites. A new thread is

created every time a bullet is shot to handle the movement of that bullet and calculate whether the bullet has collided with the ship.

Each bird has two lives - it needs to be hit twice before exploding and disappearing from the screen. The ship has three lives, so it needs to be hit three times before exploding and disappearing. The explosion sprite image replaces the bird or ship sprite image for a few milliseconds whenever one of them is hit by a bullet before returning to the original bird/ship sprite image. If the sprite runs out of lives, then the explosion sprite image appears for two seconds and then the sprite disappears permanently.

There are two tiles for the score at the top of the screen. One for the player and one for the computer. The score increases appropriately whenever one of the sprites is hit. There is a tile below the score for the lives remaining for the ship. This tile starts at 3 and is reduced by one every time the ship is hit.

5. Challenges

Despite the extenuating conditions that have affected the progress of our game display, we have managed to establish an age old arcade game ready for play. With this process brought many challenges that we have faced and overcome to create the game. One major challenge that we have had is determining the memory allocation needed for the game. With the tiles and the sprites, we set out to determine the best way to keep all of the pattern information needed for display. This was resolved as we continued to iron out the accessing process by continuously drawing and discussing the process of which the board accesses this memory. Another challenge that we've worked through was developing the software process behind the game so that it

would be much like the actual game that you can play at the arcade. In the original game, the birds are set in a seemingly random pattern, and they each in their own time attack the ship by both shooting bullets and diving towards it. It was difficult to replicate this pattern in our game, because each bird needed its own thread for moving and shooting bullets, so the timing of actions would not be affected. To resolve this, we have implemented 3 enemy birds, and have hard coded a pattern that the birds use to shoot their bullets. This is how we worked to reach the conventions of the game.

6. Further Study

There are several ways that our game could be improved if we had more time. It is important to note that this project was done remotely by the three members of the team, while Columbia transitioned to online classes due to the COVID-19 epidemic in the spring semester of 2020. Ideas for improving the game include:

- Audio: We were able to convert the background song of the game to a .mif file, configure the hardware in qsys, and play this song through a .sv file when a register is activated from a .c software program. However, we were unable to integrate this with our hardware configuration and .sv file we had for the rest of the game and ran out of time while attempting to solve the errors. Further, more .mif files should be obtained that contain the different audio effects that occur in the original Phoenix game. The audio effect that plays should be controlled from the software .c code through a register.
- Movement of the birds: Our game contains a very simple implementation of the movement of the birds, as described in previous sections. However, the original Phoenix

game contained more complex movements by each individual bird sprite. This could be achieved by modifying our software .c code logic, but we ran out of time.

- Scrolling background: In the original Phoenix game, the tile background scrolls continuously. However, our implementation of the tile background does not move.
- More bullets: Our implementation of the game only allows for one bullet per bird and ship sprite. Adding more bullets would simply involve adding more entries in the sprite attribute table and controlling them from software, using the same logic we currently use for the bullet sprites we have.

7. References

[1] http://www.cs.columbia.edu/~sedwards/papers/TMS9918.pdf

[2] http://www.cs.columbia.edu/~sedwards/classes/2019/4840-spring/reports/BrickBreaker.pdf

[3] https://retro-game.fandom.com/wiki/Phoenix_(Arcade)

[4] https://www.usb.org/sites/default/files/documents/hid1_11.pdf

8. Source Code

vga_ball.sv

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

module vga_ball(input logic        clk,
```

```
            input logic        reset,
            input logic [7:0]  writedata,
            input logic        write,
            input              chipselect,
            input logic [2:0]  address,

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

  logic [10:0]    hcount;
  logic [9:0]     vcount;
  logic [23:0]    final_color;
  logic [23:0]    sprite_color;

  logic [7:0]    sprite_name;
  logic [7:0]    new_y;
  logic [7:0]    new_x;
  logic [7:0]    sprite_change;
  logic [7:0]    new_name;
  logic [7:0]    new_tag;
  logic [1:0]     is_sprite;
  logic [7:0]    score;


  vga_counters counters(.clk50(clk), .*);

        tile_generator tiles(.hcount(hcount[10:0]), .vcount(vcount), .clk(clk), .score(score),
.final_color(final_color));
                    sprite_generator sprites(.hcount(hcount[10:0]), .vcount(vcount), .clk(clk),
.sprite_change(sprite_change), .sprite_name(sprite_name), .new_y(new_y), .new_x(new_x),
.new_name(new_name), .new_tag(new_tag), .*);

  always_comb begin
    {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
    if (VGA_BLANK_n )

        if (is_sprite == 2'b01)
                    {VGA_R, VGA_G, VGA_B} = {sprite_color[23:16], sprite_color[15:8],
sprite_color[7:0]};
        else
                     {VGA_R, VGA_G, VGA_B} = {final_color[23:16], final_color[15:8],
final_color[7:0]};
```

```
  end


   always_ff @(posedge clk)
    if (reset) begin
       sprite_change <= 8'b0;
       sprite_name <= 8'b0;
       new_x <= 8'b0;
       new_y <= 8'b0;
          new_name <= 8'b0;
          new_tag <= 8'b0;
          score <= 8'h0;
    end else if (chipselect && write)
      case (address)
            3'h0 : sprite_change <= writedata;
            3'h1 : sprite_name <= writedata;
            3'h2 : new_x <= writedata;
            3'h3 : new_y <= writedata;
            3'h4 : new_name <= writedata;
        3'h5 : new_tag <= writedata;
            3'h6 : score <=writedata;
      endcase


endmodule

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

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0             1279       1599 0
 *              _______________           ________
 * _____________|    Video      |_____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _____________________     _____________
 * |____|       VGA_HS          |____|
 */
   // Parameters for hcount
```

```
parameter HACTIVE      = 11'd 1280,
          HFRONT_PORCH = 11'd 32,
          HSYNC        = 11'd 192,
          HBACK_PORCH  = 11'd 96,
          HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                         HBACK_PORCH; // 1600

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

logic endOfLine;

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

assign endOfLine = hcount == HTOTAL - 1;

logic endOfField;

always_ff @(posedge clk50 or posedge reset)
  if (reset)          vcount <= 0;
  else if (endOfLine)
    if (endOfField)   vcount <= 0;
    else              vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;

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

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

// Horizontal active: 0 to 1279     Vertical active: 0 to 479
// 101 0000 0000  1280              01 1110 0000  480
// 110 0011 1111  1599              10 0000 1100  524
```

```
  assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                       !( vcount[9] | (vcount[8:5] == 4'b1111) );

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

endmodule

module tile_generator (input logic [10:0] hcount, input logic [9:0] vcount, input logic clk, input
logic [7:0] score, output logic [23:0] final_color);

    logic [7:0] pattern_name_table [4799:0];
    logic [7:0] pattern_gen_table [2047:0];
    logic [47:0] color_table [255:0];

    logic [13:0] base_add;
    logic [7:0] tile_name;
    logic [10:0] gen_add;
    logic [7:0] pixel_row;
    logic [3:0] pixel_col;
    logic [47:0] pattern_colors;
    int i;
    int j;
    int m;
    int count =1;


    initial
      begin
        for (i = 0; i < 4800; i = i+1) pattern_name_table[i] = 34;
        for (i = 0; i < 2048; i = i+1) pattern_gen_table[i] = 8'h00;

        pattern_name_table[35]= 24;
           pattern_name_table[4799]=24;
        pattern_name_table[79]=24;
        pattern_name_table[77]=24;
           pattern_name_table[4720]=24;
```

```
   // Star band formation
for (m = 0; m < 16; m = m+1) begin

        if (m%2 ==0) begin
                for (i = 8; i < 18; i = i +1) begin
                        count = 80 *i;
                for (j = m*5; j < m*5 +5 ; j = j +1) begin
                        if (count%2==0)
                                pattern_name_table[count+j]=24;
                        else
                                        pattern_name_table[count+j]=23;
                end
                end
                for (i = 28; i < 38; i = i +1) begin
                        count = 80 *i;
                for (j = m*5; j < m*5 +5 ; j = j +1) begin
                        if (count%2==0)
                                pattern_name_table[count+j]=24;
                        else
                                        pattern_name_table[count+j]=23;
                end
                end

                for (i = 48; i < 58; i = i +1) begin
                        count = 80 *i;
                for (j = m*5; j < m*5 +5 ; j = j +1) begin
                        if (count%2==0)
                                pattern_name_table[count+j]=24;
                        else
                                        pattern_name_table[count+j]=23;
                end
                end
        end

        else begin
                for (i = 9; i < 19; i = i +1) begin
                        count = 80 *i;
                for (j = m*5; j < m*5 +5 ; j = j +1) begin
                        if (count%2==0)
                                pattern_name_table[count+j]=24;
                        else
                                        pattern_name_table[count+j]=23;
                end
                end
                for (i = 29; i < 39; i = i +1) begin
```

```
                count = 80 *i;
            for (j = m*5; j < m*5 +5 ; j = j +1) begin
                if (count%2==0)
                    pattern_name_table[count+j]=24;
                else
                        pattern_name_table[count+j]=23;
            end
            end
            for (i = 49; i < 59; i = i +1) begin
                count = 80 *i;
            for (j = m*5; j < m*5 +5 ; j = j +1) begin
                if (count%2==0)
                    pattern_name_table[count+j]=24;
                else
                        pattern_name_table[count+j]=23;
            end
            end

        end
end


pattern_name_table[1764]=25;
   pattern_name_table[2470]=25;
   pattern_name_table[789]=25;
   pattern_name_table[1560]=25;
   pattern_name_table[353]=25;
   pattern_name_table[4589]=25;
   pattern_name_table[19]=25;
   pattern_name_table[3147]=25;
pattern_name_table[1273]=25;
   pattern_name_table[4591]=25;
   pattern_name_table[1485]=25;
   pattern_name_table[73]=25;
   pattern_name_table[1697]=25;
   pattern_name_table[1236]=25;
   pattern_name_table[2421]=25;
   pattern_name_table[630]=25;
pattern_name_table[3593]=25;
   pattern_name_table[3057]=25;
   pattern_name_table[2768]=25;
   pattern_name_table[3522]=25;
   pattern_name_table[1284]=25;
   pattern_name_table[1555]=25;
   pattern_name_table[2978]=25;
```

```
   pattern_name_table[1735]=25;
pattern_name_table[2190]=25;
   pattern_name_table[1910]=25;
   pattern_name_table[2008]=25;
   pattern_name_table[1400]=25;
   pattern_name_table[1300]=25;
   pattern_name_table[1200]=25;
   pattern_name_table[2100]=25;
   pattern_name_table[2250]=25;


   pattern_name_table[1100]=22;
   pattern_name_table[2025]=22;
   pattern_name_table[3025]=22;
   pattern_name_table[2525]=22;
pattern_name_table[2750]=22;
   pattern_name_table[3625]=22;
   pattern_name_table[3825]=22;
   pattern_name_table[3715]=22;
   pattern_name_table[1925]=22;
pattern_name_table[356]=22;
   pattern_name_table[512]=22;
   pattern_name_table[1054]=22;
   pattern_name_table[777]=22;
   pattern_name_table[320]=22;

   pattern_name_table[2800]=21;
   pattern_name_table[4067]=21;
   pattern_name_table[1009]=21;
   pattern_name_table[256]=21;
   pattern_name_table[156]=21;

for (i = 12; i < 17; i = i+1) pattern_name_table[i] = i-1; // Score 1
pattern_name_table[17] = 1;

pattern_name_table[32] = 16;
pattern_name_table[33] = 17;
pattern_name_table[34] = 10;
for (i = 35; i < 40; i = i+1) pattern_name_table[i] = i-24;  // Hi Score

for (i = 55; i < 60; i = i+1) pattern_name_table[i] = i-44; // Score 2
pattern_name_table[60] = 9;

for (i = 93; i < 97; i = i+1) pattern_name_table[i] = 0;
for (i = 114; i < 118; i = i+1) pattern_name_table[i] = 0;
```

```
for (i = 136; i < 140; i = i+1) pattern_name_table[i] = 0;

pattern_name_table[174] = 20; // Life ship for score 1
pattern_name_table[175] = 3;

//pattern_name_table[194] = 20;
pattern_name_table[217] = 20; // Life ship for score 2
pattern_name_table[218] = 3;

// Number 0
pattern_gen_table[0] = 8'h08;
pattern_gen_table[1] = 8'h14;
pattern_gen_table[2] = 8'h22;
pattern_gen_table[3] = 8'h22;
pattern_gen_table[4] = 8'h22;
pattern_gen_table[5] = 8'h22;
pattern_gen_table[6] = 8'h14;
pattern_gen_table[7] = 8'h08;

// Number 1
pattern_gen_table[15] = 8'h7C;
pattern_gen_table[14] = 8'h10;
pattern_gen_table[13] = 8'h10;
pattern_gen_table[12] = 8'h10;
pattern_gen_table[11] = 8'h10;
pattern_gen_table[10] = 8'h14;
pattern_gen_table[9] = 8'h18;
pattern_gen_table[8] = 8'h10;

// Number 2
pattern_gen_table[23] = 8'h00; //3c
pattern_gen_table[22] = 8'h3E; //44
pattern_gen_table[21] = 8'h02; //40
pattern_gen_table[20] = 8'h02; //20
pattern_gen_table[19] = 8'h3E; //10
pattern_gen_table[18] = 8'h20; //08
pattern_gen_table[17] = 8'h20; //04
pattern_gen_table[16] = 8'h3E; //4c


// Number 3
pattern_gen_table[31] = 8'h3E;
pattern_gen_table[30] = 8'h40; //20
pattern_gen_table[29] = 8'h40; //10
pattern_gen_table[28] = 8'h20; //08
```

```
pattern_gen_table[27] = 8'h30; //10
pattern_gen_table[26] = 8'h40; //20
pattern_gen_table[25] = 8'h40; //20
pattern_gen_table[24] = 8'h3E;

   // Number 4
pattern_gen_table[215] = 8'h20;
pattern_gen_table[214] = 8'h20; //20
pattern_gen_table[213] = 8'h20; //10
pattern_gen_table[212] = 8'h20; //08
pattern_gen_table[211] = 8'h7E; //10
pattern_gen_table[210] = 8'h22; //20
pattern_gen_table[209] = 8'h22; //20
pattern_gen_table[208] = 8'h22;

   // Number 5
pattern_gen_table[223] = 8'h00;
pattern_gen_table[222] = 8'h3E; //20
pattern_gen_table[221] = 8'h20; //10
pattern_gen_table[220] = 8'h20; //08
pattern_gen_table[219] = 8'h3E; //10
pattern_gen_table[218] = 8'h02; //20
pattern_gen_table[217] = 8'h02; //20
pattern_gen_table[216] = 8'h3E;

// Number 6
pattern_gen_table[231] = 8'h00;
pattern_gen_table[230] = 8'h3E; //20
pattern_gen_table[229] = 8'h22; //10
pattern_gen_table[228] = 8'h22; //08
pattern_gen_table[227] = 8'h3E; //10
pattern_gen_table[226] = 8'h02; //20
pattern_gen_table[225] = 8'h02; //20
pattern_gen_table[224] = 8'h3E;

// Number 7
pattern_gen_table[239] = 8'h00;
pattern_gen_table[238] = 8'h20; //20
pattern_gen_table[237] = 8'h20; //10
pattern_gen_table[236] = 8'h20; //08
pattern_gen_table[235] = 8'h20; //10
pattern_gen_table[234] = 8'h20; //20
pattern_gen_table[233] = 8'h20; //20
pattern_gen_table[232] = 8'h3E;
```

```
// Number 8
pattern_gen_table[247] = 8'h00;
pattern_gen_table[246] = 8'h3E; //20
pattern_gen_table[245] = 8'h22; //10
pattern_gen_table[244] = 8'h22; //08
pattern_gen_table[243] = 8'h3E; //10
pattern_gen_table[242] = 8'h22; //20
pattern_gen_table[241] = 8'h22; //20
pattern_gen_table[240] = 8'h3E;

// Number 9
    pattern_gen_table[79] = 8'h00;
    pattern_gen_table[78] = 8'h20;
    pattern_gen_table[77] = 8'h20;
    pattern_gen_table[76] = 8'h20;
    pattern_gen_table[75] = 8'h3E;
    pattern_gen_table[74] = 8'h22;
    pattern_gen_table[73] = 8'h22;
    pattern_gen_table[72] = 8'h3E;

// Dash --> 10
pattern_gen_table[80] = 8'h00;
pattern_gen_table[81] = 8'h00;
pattern_gen_table[82] = 8'h00;
pattern_gen_table[83] = 8'h00;
pattern_gen_table[84] = 8'h7C;
pattern_gen_table[85] = 8'h00;
pattern_gen_table[86] = 8'h00;
pattern_gen_table[87] = 8'h00;

// S --> 11
pattern_gen_table[95] = 8'h3E;
pattern_gen_table[94] = 8'h40;
pattern_gen_table[93] = 8'h20;
pattern_gen_table[92] = 8'h10;
pattern_gen_table[91] = 8'h08;
pattern_gen_table[90] = 8'h04;
pattern_gen_table[89] = 8'h02;
pattern_gen_table[88] = 8'h7C;

// C --> 12
pattern_gen_table[103] = 8'h7C;
pattern_gen_table[102] = 8'h02;
pattern_gen_table[101] = 8'h02;
pattern_gen_table[100] = 8'h02;
```

```
pattern_gen_table[99] = 8'h02;
pattern_gen_table[98] = 8'h02;
pattern_gen_table[97] = 8'h02;
pattern_gen_table[96] = 8'h7C;

// R --> 14
pattern_gen_table[112] = 8'h3E;
pattern_gen_table[113] = 8'h22;
pattern_gen_table[114] = 8'h22;
pattern_gen_table[115] = 8'h3E;
pattern_gen_table[116] = 8'h06;
pattern_gen_table[117] = 8'h0A;
pattern_gen_table[118] = 8'h12;
pattern_gen_table[119] = 8'h22;

// E --> 15
pattern_gen_table[120] = 8'h3E;
pattern_gen_table[121] = 8'h02;
pattern_gen_table[122] = 8'h02;
pattern_gen_table[123] = 8'h1E;
pattern_gen_table[124] = 8'h02;
pattern_gen_table[125] = 8'h02;
pattern_gen_table[126] = 8'h02;
pattern_gen_table[127] = 8'h3E;

// O --> 13
pattern_gen_table[111] = 8'h7E;
pattern_gen_table[110] = 8'h42;
pattern_gen_table[109] = 8'h42;
pattern_gen_table[108] = 8'h42;
pattern_gen_table[107] = 8'h42;
pattern_gen_table[106] = 8'h42;
pattern_gen_table[105] = 8'h42;
pattern_gen_table[104] = 8'h7E;

// H --> 16
pattern_gen_table[128] = 8'h44;
pattern_gen_table[129] = 8'h44;
pattern_gen_table[130] = 8'h44;
pattern_gen_table[131] = 8'h7C;
pattern_gen_table[132] = 8'h44;
pattern_gen_table[133] = 8'h44;
pattern_gen_table[134] = 8'h44;
pattern_gen_table[135] = 8'h44;
```

```
// I --> 17
pattern_gen_table[136] = 8'h7C;
pattern_gen_table[137] = 8'h10;
pattern_gen_table[138] = 8'h10;
pattern_gen_table[139] = 8'h10;
pattern_gen_table[140] = 8'h10;
pattern_gen_table[141] = 8'h10;
pattern_gen_table[142] = 8'h10;
pattern_gen_table[143] = 8'h7C;

// Life spaceship
pattern_gen_table[167] = 8'hAA; //49
pattern_gen_table[166] = 8'h6C; //2a
pattern_gen_table[165] = 8'h38; //1c
pattern_gen_table[164] = 8'h10; //08
pattern_gen_table[163] = 8'h10; //08
pattern_gen_table[162] = 8'h7C; //1c
pattern_gen_table[161] = 8'h54; //3e
pattern_gen_table[160] = 8'h94; //5d


   //Stars

   // 25
   pattern_gen_table[203]= 8'h01;

   //  24
   pattern_gen_table[199]= 8'h40;
   pattern_gen_table[194] = 8'h01;

//  23
pattern_gen_table[184]= 8'h00;
   pattern_gen_table[187] = 8'h20;
   pattern_gen_table[191] = 8'h80;

   //  22
   pattern_gen_table[181] = 8'h18;
   pattern_gen_table[182] = 8'h18;

//  21
   pattern_gen_table[175] = 8'h18;
pattern_gen_table[174] = 8'h24;
   pattern_gen_table[173] = 8'h42;
   pattern_gen_table[172] = 8'h99;
   pattern_gen_table[171] = 8'h99;
```

```
            pattern_gen_table[170] = 8'h42;
            pattern_gen_table[169] = 8'h24;
            pattern_gen_table[168] = 8'h18;


          //for (i = 0; i < 255; i = i + 1) color_table[i] = {24'h000000, 24'hffffff}; // Background is
black and white
           for (i = 0; i < 11; i = i+1) color_table[i] = {24'hff0000, 24'h000000};  // Numbers are red
and black
           for (i = 11; i < 18; i = i+1) color_table[i] = {24'h00ffff, 24'h000000}; // Letters are blue
and black
         color_table[20] = {24'h00ff00, 24'h000000}; // Color scheme for spaceship


         for (i = 21; i < 26; i = i+1) color_table[i]= {24'h3374ff, 24'h000000};
            color_table[22]= {24'hffffff, 24'h000000};

         for (i = 26; i < 31; i = i+1) color_table[i] = {24'hff0000, 24'h000000};
         color_table[34]= {24'h000000, 24'h000000};


       end

     assign base_add={vcount[8:3], hcount[10:4]} - 48*vcount[8:3];
     assign gen_add = {tile_name, vcount[2:0]};
     assign pixel_col = hcount[3:1];
                assign final_color = pixel_row[pixel_col] == 1 ? pattern_colors[47:24] :
pattern_colors[23:0];


     always_ff @(posedge clk) begin

               case (score)
                       8'h0: pattern_name_table[96]=0;
                       8'h1: pattern_name_table[96]=1;
                       8'h2: pattern_name_table[96]=2;
                       8'h3: pattern_name_table[96]=3;
              8'h4: pattern_name_table[96]=26;
                       8'h5: pattern_name_table[96]=27;
                 8'h6: pattern_name_table[96]=28;
                 8'h7: pattern_name_table[96]=29;
                 8'h8: pattern_name_table[96]=30;
                 8'h9: pattern_name_table[96]=9;
               endcase
               tile_name = pattern_name_table[base_add];
```

```
        pixel_row <= pattern_gen_table[gen_add];

        pattern_colors <= color_table[tile_name];
    end

endmodule

module sprite_generator (input logic [10:0] hcount, input logic [9:0] vcount, input logic clk, input logic [7:0] sprite_change, input logic [7:0] sprite_name, input logic [7:0] new_y, input logic [7:0] new_x, input logic [7:0] new_name, input logic [7:0] new_tag, output logic [23:0] sprite_color, output logic [1:0] is_sprite);

        logic [15:0] sprite_att_table [32:0];
        logic [31:0] pattern_table [2047:0];
        logic [95:0] color_table [6:0];

    //logic [] pos_y;
    logic [12:0] pattern_add;
    logic [15:0] name;
    logic [9:0] vertical;
    logic [10:0] horizontal;
    logic [31:0] pixel_row;
    logic [5:0] pixel_col;
       logic [5:0] before_col;
       logic [5:0] after_col;
    logic [95:0] colors;
    int i;
    int j;


    initial
      begin
          sprite_att_table[0] = 400;  // vertical position to start
          sprite_att_table[1] = 550;  // horizontal position to start
          sprite_att_table[2] = 0;      // Name [Address to find ship]
          sprite_att_table[3] = 1;      // tag

                sprite_att_table[4] = 300; // Player bullet
                sprite_att_table[5] = 550;
                sprite_att_table[6] = 1;
                sprite_att_table[7] = 1;

                sprite_att_table[8] = 200; // Bird 1
                sprite_att_table[9] = 200;
                sprite_att_table[10] = 2;
```

```
sprite_att_table[11] = 1;

sprite_att_table[12] = 200; // Bird 2
sprite_att_table[13] = 250;
sprite_att_table[14] = 2;
sprite_att_table[15] = 1;

sprite_att_table[16] = 200; // Bird 3
sprite_att_table[17] = 150;
sprite_att_table[18] = 2;
sprite_att_table[19] = 1;

sprite_att_table[20] = 225; // bird bullet 1
sprite_att_table[21] = 200;
sprite_att_table[22] = 5;
sprite_att_table[23] = 1;

sprite_att_table[24] = 200; // bird bullet 2
sprite_att_table[25] = 650;
sprite_att_table[26] = 5;
sprite_att_table[27] = 1;

sprite_att_table[28] = 225; // bird bullet 3
sprite_att_table[29] = 650;
sprite_att_table[30] = 5;
sprite_att_table[31] = 1;

sprite_att_table[32] = 9; // blank name

pattern_table[0] = {8'h00, 8'h00, 8'h00, 8'h00};
pattern_table[1] = {8'h00, 8'h00, 8'h00, 8'h00};
pattern_table[2] = {8'h00, 8'h01, 8'h80, 8'h00};
pattern_table[3] = {8'h00, 8'h01, 8'h80, 8'h00};
pattern_table[4] = {8'h00, 8'h01, 8'h80, 8'h00};
pattern_table[5] = {8'h00, 8'h43, 8'hC2, 8'h00};
pattern_table[6] = {8'h00, 8'h43, 8'hC2, 8'h00};
pattern_table[7] = {8'h00, 8'h43, 8'hC2, 8'h00};
pattern_table[8] = {8'h00, 8'h63, 8'hC6, 8'h00};
pattern_table[9] = {8'h00, 8'h73, 8'hCE, 8'h00};
pattern_table[10] = {8'h00, 8'h7B, 8'hDE, 8'h00};
pattern_table[11] = {8'h00, 8'h7F, 8'hFE, 8'h00};
pattern_table[12] = {8'h00, 8'h5F, 8'hFA, 8'h00};
pattern_table[13] = {8'h00, 8'h4F, 8'hF2, 8'h00};
pattern_table[14] = {8'h00, 8'h07, 8'hE0, 8'h00};
pattern_table[15] = {8'h00, 8'h03, 8'hC0, 8'h00};
```

```
pattern_table[16] = {8'h00, 8'h03, 8'hC0, 8'h00};
pattern_table[17] = {8'h00, 8'h07, 8'hE0, 8'h00};
pattern_table[18] = {8'h00, 8'h8F, 8'hF1, 8'h00};
pattern_table[19] = {8'h00, 8'h9F, 8'hF9, 8'h00};
pattern_table[20] = {8'h00, 8'hBF, 8'hFD, 8'h00};
pattern_table[21] = {8'h00, 8'hFF, 8'hFF, 8'h00};
pattern_table[22] = {8'h00, 8'hFB, 8'hDF, 8'h00};
pattern_table[23] = {8'h00, 8'hF3, 8'hCF, 8'h00};
pattern_table[24] = {8'h00, 8'hE3, 8'hC7, 8'h00};
pattern_table[25] = {8'h00, 8'hC3, 8'hC3, 8'h00};
pattern_table[26] = {8'h00, 8'h83, 8'hC1, 8'h00};
pattern_table[27] = {8'h00, 8'h9F, 8'hF9, 8'h00};
pattern_table[28] = {8'h00, 8'h99, 8'h99, 8'h00};
pattern_table[29] = {8'h00, 8'h99, 8'h99, 8'h00};
pattern_table[30] = {8'h00, 8'h00, 8'h00, 8'h00};
pattern_table[31] = {8'h00, 8'h00, 8'h00, 8'h00};

for (i = 32; i < 64; i = i+1) pattern_table[i] = {8'h00, 8'h00, 8'h00, 8'h00};
for (i = 45; i < 51; i = i+1) pattern_table[i] = {8'h00, 8'h01, 8'h00, 8'h00};

// Bird
for (i = 64; i < 69; i = i+1) pattern_table[i] = {8'h00, 8'h00, 8'h00, 8'h00};
pattern_table[69] = {8'h00, 8'h06, 8'h60, 8'h00};
pattern_table[70] = {8'h00, 8'h86, 8'h61, 8'h00};
pattern_table[71] = {8'h01, 8'hC1, 8'h83, 8'h80};
pattern_table[72] = {8'h03, 8'hE1, 8'h87, 8'hC0};
pattern_table[73] = {8'h07, 8'hF1, 8'h8F, 8'hE0};
pattern_table[74] = {8'h0F, 8'hF9, 8'h9F, 8'hF0};
pattern_table[75] = {8'h0F, 8'hFF, 8'hFF, 8'hF0};
pattern_table[76] = {8'h0F, 8'hFF, 8'hFF, 8'hF0};
pattern_table[77] = {8'h0F, 8'h8F, 8'hF1, 8'hF0};
pattern_table[78] = {8'h0F, 8'h8F, 8'hF1, 8'hF0};
pattern_table[79] = {8'h0F, 8'h8F, 8'hF1, 8'hF0};
pattern_table[80] = {8'h0F, 8'h8F, 8'hF1, 8'hF0};
pattern_table[81] = {8'h07, 8'h8F, 8'hF1, 8'hE0};
pattern_table[82] = {8'h03, 8'hCF, 8'hF3, 8'hC0};
pattern_table[83] = {8'h01, 8'hE7, 8'hE7, 8'h80};
pattern_table[84] = {8'h00, 8'hE3, 8'hC7, 8'h00};
pattern_table[85] = {8'h00, 8'h61, 8'h86, 8'h00};
pattern_table[86] = {8'h00, 8'h21, 8'h84, 8'h00};
pattern_table[87] = {8'h00, 8'h21, 8'h84, 8'h00};
for (i = 88; i < 96; i = i+1) pattern_table[i] = {8'h00, 8'h00, 8'h00, 8'h00};

// flying bird moving left
for (i = 96; i < 104; i = i+1) pattern_table[i] = {8'h00, 8'h00, 8'h00, 8'h00};
```

```
pattern_table[104] = {8'h00, 8'h1F, 8'hFF, 8'hC0};
pattern_table[105] = {8'h00, 8'h1C, 8'h1F, 8'h80};
pattern_table[106] = {8'h00, 8'h78, 8'h1F, 8'h00};
pattern_table[107] = {8'h00, 8'h00, 8'h1E, 8'h00};
pattern_table[108] = {8'h00, 8'h07, 8'hFC, 8'h00};
pattern_table[109] = {8'h00, 8'h0F, 8'hF8, 8'h60};
pattern_table[110] = {8'h00, 8'h1F, 8'hF8, 8'h60};
pattern_table[111] = {8'h00, 8'hFF, 8'hFF, 8'h80};
pattern_table[112] = {8'h00, 8'hFF, 8'hFF, 8'h80};
pattern_table[113] = {8'h00, 8'h1F, 8'hF8, 8'h60};
pattern_table[114] = {8'h00, 8'h0F, 8'hF8, 8'h60};
pattern_table[115] = {8'h00, 8'h07, 8'hFC, 8'h00};
pattern_table[116] = {8'h00, 8'h00, 8'h1E, 8'h00};
pattern_table[117] = {8'h00, 8'h78, 8'h1F, 8'h00};
pattern_table[118] = {8'h00, 8'h1C, 8'h1F, 8'h80};
pattern_table[119] = {8'h00, 8'h1F, 8'hFF, 8'hC0};
for (i = 120; i < 128; i = i+1) pattern_table[i] = {8'h00, 8'h00, 8'h00, 8'h00};

// flying bird moving right
for (i = 128; i < 136; i = i+1) pattern_table[i] = {8'h00, 8'h00, 8'h00, 8'h00};
pattern_table[136] = {8'h03, 8'hFF, 8'hFC, 8'h00};
pattern_table[137] = {8'h01, 8'hF8, 8'h3C, 8'h00};
pattern_table[138] = {8'h00, 8'hF8, 8'h1E, 8'h00};
pattern_table[139] = {8'h00, 8'h78, 8'h00, 8'h00};
pattern_table[140] = {8'h00, 8'h3F, 8'h07, 8'h00};
pattern_table[141] = {8'h06, 8'h1F, 8'hF0, 8'h60};
pattern_table[142] = {8'h06, 8'h1F, 8'hF8, 8'h60};
pattern_table[143] = {8'h01, 8'hFF, 8'hFF, 8'h80};
pattern_table[144] = {8'h01, 8'hFF, 8'hFF, 8'h80};
pattern_table[145] = {8'h06, 8'h1F, 8'hF8, 8'h60};
pattern_table[146] = {8'h06, 8'h1F, 8'hF0, 8'h60};
pattern_table[147] = {8'h00, 8'h3F, 8'h07, 8'h00};
pattern_table[148] = {8'h00, 8'h78, 8'h00, 8'h00};
pattern_table[149] = {8'h00, 8'hF8, 8'h1E, 8'h00};
pattern_table[150] = {8'h01, 8'hF8, 8'h3C, 8'h00};
pattern_table[151] = {8'h03, 8'hFF, 8'hFC, 8'h00};
for (i = 152; i < 160; i = i+1) pattern_table[i] = {8'h00, 8'h00, 8'h00, 8'h00};


// Bullets for the bird
for (i = 160; i < 192; i = i+1) pattern_table[i] = {8'h00, 8'h00, 8'h00, 8'h00};
for (i = 174; i < 178; i = i+1) pattern_table[i] = {8'h00, 8'h01, 8'h80, 8'h00};

for (i = 192; i < 224; i = i+1) pattern_table[i] = {8'h00, 8'h00, 8'h00, 8'h00};
pattern_table[192] = {8'h00, 8'h00, 8'h00, 8'h00};
```

```
        pattern_table[193] = {8'h00, 8'h00, 8'h00, 8'h00};
        pattern_table[194] = {8'h00, 8'h00, 8'h00, 8'h00};
        pattern_table[195] = {8'h00, 8'h08, 8'h00, 8'h00};
        pattern_table[196] = {8'h00, 8'h08, 8'h00, 8'h00};
        pattern_table[197] = {8'h18, 8'h60, 8'h00, 8'h00};
        pattern_table[198] = {8'h3C, 8'hF0, 8'h82, 8'h08};
        pattern_table[199] = {8'h3C, 8'hF8, 8'h98, 8'h1C};

        pattern_table[200] = {8'h19, 8'hFC, 8'h3C, 8'h1C};
        pattern_table[201] = {8'h01, 8'hFE, 8'h7E, 8'h08};
        pattern_table[202] = {8'h03, 8'hFF, 8'hFF, 8'h80};
        pattern_table[203] = {8'h0F, 8'hFF, 8'hFF, 8'hC0};
        pattern_table[204] = {8'h1F, 8'hFF, 8'hFF, 8'hE0};
        pattern_table[205] = {8'h1F, 8'hFF, 8'hFF, 8'hE0};
        pattern_table[206] = {8'h0C, 8'hFF, 8'hFF, 8'hE0};
        pattern_table[207] = {8'h22, 8'hFF, 8'hFF, 8'hC0};

        pattern_table[208] = {8'h01, 8'hFF, 8'hFF, 8'h90};
        pattern_table[209] = {8'h00, 8'hFF, 8'hFF, 8'hC0};
        pattern_table[210] = {8'h20, 8'hFF, 8'hFF, 8'hE0};
        pattern_table[211] = {8'h71, 8'hFF, 8'hFF, 8'hF0};
        pattern_table[212] = {8'h23, 8'hFF, 8'hFF, 8'hF0};
        pattern_table[213] = {8'h0F, 8'hFF, 8'hFF, 8'hF0};
        pattern_table[214] = {8'h1F, 8'hFF, 8'hFF, 8'hF0};
        pattern_table[215] = {8'h0F, 8'hDF, 8'hFB, 8'hE0};

        pattern_table[216] = {8'h47, 8'h8F, 8'hF1, 8'hC0};
        pattern_table[217] = {8'h03, 8'h07, 8'hE0, 8'h00};
        pattern_table[218] = {8'h00, 8'h03, 8'hC0, 8'h00};
        pattern_table[219] = {8'h00, 8'h00, 8'h00, 8'h00};
        pattern_table[220] = {8'h00, 8'h00, 8'h00, 8'h00};
        pattern_table[221] = {8'h00, 8'h00, 8'h00, 8'h00};
        pattern_table[222] = {8'h00, 8'h00, 8'h00, 8'h00};
        pattern_table[223] = {8'h00, 8'h00, 8'h00, 8'h00};


        color_table[0] = {24'hffff00, 24'hff0000, 24'hffffff, 24'h000000};
        color_table[1] = {24'hffffff, 24'h000000, 24'h000000, 24'h000000};
        color_table[2] = {24'hff00ff, 24'h00ff00, 24'h000000, 24'h000000};
        color_table[3] = {24'hff00ff, 24'h00ff00, 24'h000000, 24'h000000};
        color_table[4] = {24'hff00ff, 24'h00ff00, 24'h000000, 24'h000000};
        color_table[5] = {24'hffffff, 24'h000000, 24'h000000, 24'h000000};
        color_table[6] = {24'hff00ff, 24'hffff00, 24'h000000, 24'h000000};
end
```

```
    assign pattern_add = {name[7:0], vertical[4:0]};
       assign pixel_col = horizontal[5:1]; // OG 5:1
       assign before_col = pixel_col - 5'b00001;
       assign after_col = pixel_col + 5'b00001;
       assign is_sprite = name == sprite_att_table[32] ? 2'b00 : 2'b01;

    always_comb begin
          horizontal = hcount;
          vertical = vcount;
          name = sprite_att_table[32];
                for (j = 0; j < 32; j = j+4) // Make this more dynamic
                      if (vcount >= sprite_att_table[j] && vcount < sprite_att_table[j] + 32 &&
hcount >= sprite_att_table[j+1] && hcount < sprite_att_table[j+1] + 64 && sprite_att_table[j+3]
== 1) begin
                                  horizontal = hcount - sprite_att_table[j+1];
                                  vertical = vcount - sprite_att_table[j];
                                  name = sprite_att_table[j+2];
                      end
    end


    always_ff @(posedge clk) begin
                if (sprite_change[0] == 1'b1) begin
         sprite_att_table[(sprite_name << 2)] = {7'b0, new_y[7:0], 1'b0};
         sprite_att_table[(sprite_name << 2) + 1'b1] = {5'b0, new_x[7:0], 3'b0};
              sprite_att_table[(sprite_name << 2) + 2'b10] = {8'b0, new_name};
              sprite_att_table[(sprite_name << 2) + 2'b11] = {8'b0, new_tag};
          end

          pixel_row <= pattern_table[pattern_add];

          colors <= color_table[name];
       end


       always_comb begin
                case ({pixel_row[before_col], pixel_row[pixel_col], pixel_row[after_col]})
                      3'b010: sprite_color = colors[95:72];
                      3'b011: sprite_color = colors[95:72];
                      3'b110: sprite_color = colors[95:72];
                      3'b111: sprite_color = colors[71:48];
                default: sprite_color = colors[23:0];
       endcase
       end
```

```
endmodule
```

vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>


typedef struct {
        unsigned char sprite_change, sprite_num, new_x, new_y, new_name, new_tag;
} sprite_change_t;

typedef struct {
        unsigned char x, y;
} coordinates_t;

typedef struct {
        unsigned short score1;

}hardware_p;

typedef struct {
  sprite_change_t sprite_args;
  hardware_p alldata;
} vga_ball_arg_t;



#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1,
vga_ball_arg_t *)
#define VGA_BALL_READ_BACKGROUND  _IOR(VGA_BALL_MAGIC, 2, vga_ball_arg_t
*)
#define VGA_BALL_WRITE_ALLDATA  _IOW(VGA_BALL_MAGIC, 3, vga_ball_arg_t *)


#endif
```

vga_ball.c

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

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_ball.h"

#define DRIVER_NAME "vga_ball"

/* Device registers */
#define SPRITE_CHANGE(x) (x)
#define SPRITE_NAME(x) ((x)+1)
#define NEW_X(x) ((x)+2)
#define NEW_Y(x) ((x)+3)
#define NEW_NAME(x) ((x)+4)
```

```
#define NEW_TAG(x) ((x)+5)
#define SCORE1(x) ((x)+6)


/*
 * Information about our device
 */
struct vga_ball_dev {
	struct resource res; /* Resource: our registers */
	void __iomem *virtbase; /* Where registers can be accessed in memory */
	sprite_change_t sprite_args;
	hardware_p alldata;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_background(sprite_change_t *sprite_args)
{
	iowrite8(sprite_args->sprite_change, SPRITE_CHANGE(dev.virtbase) );
	iowrite8(sprite_args->sprite_num, SPRITE_NAME(dev.virtbase) );
	iowrite8(sprite_args->new_x, NEW_X(dev.virtbase) );
	iowrite8(sprite_args->new_y, NEW_Y(dev.virtbase) );
	iowrite8(sprite_args->new_name, NEW_NAME(dev.virtbase) );
	iowrite8(sprite_args->new_tag, NEW_TAG(dev.virtbase) );
	dev.sprite_args = *sprite_args;
}

static void write_alldata(hardware_p *alldata){
	iowrite8(alldata->score1, SCORE1(dev.virtbase));
	dev.alldata = *alldata;

}

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

	switch (cmd) {
```

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

        case VGA_BALL_READ_BACKGROUND:
                vla.sprite_args = dev.sprite_args;
                if (copy_to_user((vga_ball_arg_t *) arg, &vla,
                                 sizeof(vga_ball_arg_t)))
                        return -EACCES;
                break;
        case VGA_BALL_WRITE_ALLDATA:
           if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
                        sizeof(vga_ball_arg_t)))
                return -EACCES;
           write_alldata(&vla.alldata);
           break;

        default:
                return -EINVAL;
        }

        return 0;
}

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

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

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev)
```

```
{
      sprite_change_t initial_args = {0,0,0,0,0,0};
        hardware_p initial_data = {0};
        int ret;

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

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

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

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

        /* Set an initial color */
      write_background(&initial_args);
        write_alldata(&initial_data);

        return 0;

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

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

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

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

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

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

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

module_init(vga_ball_init);
module_exit(vga_ball_exit);

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

hello.c

```
/*
 * Userspace program that communicates with the vga_ball device driver
 * through ioctls
 *
 * Stephen A. Edwards
 * Columbia University
 */

#include <stdio.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include <pthread.h>
#include "usbkeyboard.h"
#include <stdlib.h>

#define SHIP_NAME 0
#define SHIP_BULLET_NAME 1
#define BIRD1_NAME 2
#define BIRD2_NAME 3
#define BIRD3_NAME 4
#define BIRD1_BULLET_NAME 5
#define BIRD2_BULLET_NAME 6
#define BIRD3_BULLET_NAME 7
#define EXPLOSION_NAME 6

#define BIRD_STANDSTILL_NAME 2
#define BIRD_RIGHT_NAME 4
#define BIRD_LEFT_NAME 3

#define TOP_ROW 16
#define LAST_COLUMN 150
#define DISPLAY 1
#define DONT_DISPLAY 0
#define ROW_BIRD 40
#define ROW_SHIP 224
```

```
#define TIME_CONSTANT 10000
int vga_ball_fd;
int ship_lives = 2;
int bird1_lives = 1;
int bird2_lives = 1;
int bird3_lives = 1;
unsigned short score = 0;
hardware_p data;


struct libusb_device_handle *keyboard;
uint8_t endpoint_address;

// new_y goes from 0 to 224
// new_x goes from 0 to 112

coordinates_t bird1_coor = {0,ROW_BIRD};
coordinates_t bird2_coor = {0,ROW_BIRD};
coordinates_t bird3_coor = {0,ROW_BIRD};
coordinates_t ship_coor = {60,ROW_SHIP};

pthread_t bird1_thread_shoot;
pthread_t bird2_thread_shoot;
pthread_t bird3_thread_shoot;
pthread_t ship_thread_shoot;
pthread_t move_birds_thread;

sprite_change_t ship = {1, SHIP_NAME, 60, ROW_SHIP, SHIP_NAME, DISPLAY};

sprite_change_t bird1 = {1, BIRD1_NAME, 0, ROW_BIRD, BIRD1_NAME, DISPLAY};

sprite_change_t bird2 = {1, BIRD2_NAME, 0, ROW_BIRD, BIRD1_NAME, DISPLAY};

sprite_change_t bird3 = {1, BIRD3_NAME, 0, ROW_BIRD, BIRD1_NAME, DISPLAY};

sprite_change_t bird1_bullet = {1, BIRD1_BULLET_NAME, 0, 0,
BIRD1_BULLET_NAME,DONT_DISPLAY};

sprite_change_t bird2_bullet = {1, BIRD2_BULLET_NAME, 0, 0,
BIRD1_BULLET_NAME,DONT_DISPLAY};

sprite_change_t bird3_bullet = {1, BIRD3_BULLET_NAME, 0, 0,
BIRD1_BULLET_NAME,DONT_DISPLAY};
```

```
sprite_change_t ship_bullet = {1, SHIP_BULLET_NAME, 0, 0, SHIP_BULLET_NAME,DONT_DISPLAY};

/* Read and print the coordinates */
/*void print_coordinates() {
  vga_ball_arg_t vla;

  if (ioctl(vga_ball_fd, VGA_BALL_READ_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_READ_BACKGROUND) failed");
      return;
  }
  printf("New_x: %d New_y: %d\n",
           vla.sprite_args.new_x, vla.sprite_args.new_y);
}*/

/* Set the new position of the sprite */
void set_new_args(const sprite_change_t *c)
{
  vga_ball_arg_t vla;
  vla.sprite_args = *c;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
      return;
  }
}

void set_alldata(const hardware_p *c)
{
  vga_ball_arg_t vla;
  vla.alldata = *c;

  if( ioctl(vga_ball_fd, VGA_BALL_WRITE_ALLDATA, &vla))
  {
       perror("ioctl(VGA_BALL_SET_ALLDATA) failed");
       return;
  }
}

/* translate the joystick input into moves */
int input_moves(char *ks) {
          int move;

          if (ks[6] == 'f' && ks[12] == '1') move = 2;
          else if (ks[6] == 'f' && ks[12] == '0') move = 1;
          else if (ks[6] == '0' && ks[12] == '1') move = 0;
```

```
        else if (ks[6] == '0' && ks[12] == '0') move = -1;
        else if (ks[6] == '7' && ks[12] == '1') move = 3;
        else move = -2;

        return move;
}

void move_ship_left()
{
        if(ship_coor.x > 0) ship_coor.x -= 1;
        ship.new_x = ship_coor.x;

        set_new_args(&ship);
        usleep(TIME_CONSTANT);
}

void move_ship_right()
{
        if(ship_coor.x > LAST_COLUMN-5) ship_coor.x += 0;
        else ship_coor.x+=1;
    ship.new_x = ship_coor.x;

    set_new_args(&ship);
        usleep(TIME_CONSTANT);
}

int check_collision_with_birds(sprite_change_t *ship_bullet)
{
        if(ship_bullet->new_x >= bird1_coor.x - 2 && ship_bullet->new_x <= bird1_coor.x + 2
&& ship_bullet->new_y >= bird1_coor.y && ship_bullet->new_y <= bird1_coor.y + 4)
            return 1;
        if(ship_bullet->new_x >= bird2_coor.x - 2 && ship_bullet->new_x <= bird2_coor.x + 2
&& ship_bullet->new_y >= bird2_coor.y && ship_bullet->new_y <= bird2_coor.y + 4)
            return 2;
        if(ship_bullet->new_x >= bird3_coor.x - 2 && ship_bullet->new_x <= bird3_coor.x + 2
&& ship_bullet->new_y >= bird3_coor.y && ship_bullet->new_y <= bird3_coor.y + 4)
            return 3;
    return 0;
}


void *ship_shoot_bullet()
{
        char prev_name;
    ship_bullet.new_tag = DISPLAY;
```

```
ship_bullet.new_x = ship_coor.x;
ship_bullet.new_y = ship_coor.y - 8;
set_new_args(&ship_bullet);


while((ship_bullet.new_y) > TOP_ROW)
{
      int bird_hit = check_collision_with_birds(&ship_bullet);
   switch (bird_hit)
      {
                  case (1):
                  if (bird1_lives == 0)
                  {
                     bird1.new_name = EXPLOSION_NAME;
                     set_new_args(&bird1);
                                 usleep(TIME_CONSTANT*2);
                                 bird1.new_tag = DONT_DISPLAY;
                                 set_new_args(&bird1);
                                 bird1_lives--;
                                 score++;
                                 data.score1 = score;
                                 set_alldata(&data);

               }
               else
                     bird1_lives--;
                                 prev_name = bird1.new_name;
                                 bird1.new_name = EXPLOSION_NAME;
                     set_new_args(&bird1);
                     usleep(TIME_CONSTANT/2);
                                 bird1.new_name = prev_name;
                     set_new_args(&bird1);

                                 score++;
                     data.score1 = score;
                     set_alldata(&data);
               ship_bullet.new_tag = DONT_DISPLAY;
               set_new_args(&ship_bullet);
               break;

                  case (2):
               if (bird2_lives == 0)
               {
                     bird2.new_name = EXPLOSION_NAME;
                     set_new_args(&bird2);
```

```
                    usleep(TIME_CONSTANT*2);
                    bird2.new_tag = DONT_DISPLAY;
        set_new_args(&bird2);
                    bird2_lives--;
                    score++;
        data.score1 = score;
        set_alldata(&data);
}
else
        bird2_lives--;
                    prev_name = bird2.new_name;
        bird2.new_name = EXPLOSION_NAME;
        set_new_args(&bird2);
        usleep(TIME_CONSTANT/2);
        bird1.new_name = prev_name;
        set_new_args(&bird2);

                    score++;
        data.score1 = score;
        set_alldata(&data);
ship_bullet.new_tag = DONT_DISPLAY;
set_new_args(&ship_bullet);
break;

   case (3):
if (bird3_lives == 0)
{
        bird3.new_name = EXPLOSION_NAME;
        set_new_args(&bird3);
                    usleep(TIME_CONSTANT*2);
                    bird3.new_tag = DONT_DISPLAY;
        set_new_args(&bird3);
                    bird3_lives--;
                    score++;
        data.score1 = score;
        set_alldata(&data);
}
else
        bird3_lives--;
                    prev_name = bird3.new_name;
        bird3.new_name = EXPLOSION_NAME;
        set_new_args(&bird3);
        usleep(TIME_CONSTANT/2);
        bird3.new_name = prev_name;
        set_new_args(&bird3);
```

```
                                        score++;
                                    data.score1 = score;
                                    set_alldata(&data);
                            ship_bullet.new_tag = DONT_DISPLAY;
                            set_new_args(&ship_bullet);
                            break;
                               case (0):
                                        ship_bullet.new_y -= 1;
                                        set_new_args(&ship_bullet);
                            break;
                               default:
                            ship_bullet.new_y -= 1;
                            set_new_args(&ship_bullet);
                            break;
                    }

                    usleep(TIME_CONSTANT/2);
                        if (ship_bullet.new_tag == DONT_DISPLAY)
                                break;
            }
            ship_bullet.new_tag = DONT_DISPLAY;
            set_new_args(&ship_bullet);
}


int check_collision_with_ship(sprite_change_t *bird_bullet)
{
        if(bird_bullet->new_x >= ship_coor.x - 2 && bird_bullet->new_x <= ship_coor.x + 2)
                return 1;
        return 0;
}

void *bird1_shoot_bullet()
{
        bird1_bullet.new_tag = DISPLAY;
        bird1_bullet.new_x = bird1_coor.x;
        bird1_bullet.new_y = bird1_coor.y + 8;
        set_new_args(&bird1_bullet);

        while((bird1_bullet.new_y) < 240)
        {
                if(bird1_bullet.new_y >= 212) {
                        if (check_collision_with_ship(&bird1_bullet))
                        {
```

```
                        if (ship_lives == 0)
                        {
                                ship.new_name = EXPLOSION_NAME;
                                set_new_args(&ship);
                                usleep(TIME_CONSTANT*2);
                        ship.new_tag = DONT_DISPLAY;
                        set_new_args(&ship);
                                ship_lives--;
                        }
                        else
                                ship_lives--;
                                char prev_name = ship.new_name;
                        ship.new_name = EXPLOSION_NAME;
                        set_new_args(&ship);
                        usleep(TIME_CONSTANT/2);
                        ship.new_name = prev_name;
                        set_new_args(&ship);
                            bird1_bullet.new_tag = DONT_DISPLAY;
                            set_new_args(&bird1_bullet);
                            break;
                    }
                }
                bird1_bullet.new_y += 1;
                set_new_args(&bird1_bullet);
                usleep(TIME_CONSTANT/2);
        }
           bird1_bullet.new_tag = DONT_DISPLAY;
        set_new_args(&bird1_bullet);
}


void *bird2_shoot_bullet()
{
        bird2_bullet.new_tag = DISPLAY;
        bird2_bullet.new_x = bird2_coor.x;
        bird2_bullet.new_y = bird2_coor.y + 8;
        set_new_args(&bird2_bullet);

        while((bird2_bullet.new_y) < 240)
        {
            if(bird2_bullet.new_y >= 212) {
                if (check_collision_with_ship(&bird2_bullet))
                {
                    if (ship_lives == 0)
                    {
```

```
                        ship.new_name = EXPLOSION_NAME;
                        set_new_args(&ship);
                                usleep(TIME_CONSTANT*2);
                        ship.new_tag = DONT_DISPLAY;
                        set_new_args(&ship);
                                ship_lives--;
                    }
                    else
                        ship_lives--;
                                char prev_name = ship.new_name;
                        ship.new_name = EXPLOSION_NAME;
                        set_new_args(&ship);
                        usleep(TIME_CONSTANT/2);
                        ship.new_name = prev_name;
                        set_new_args(&ship);
                    bird2_bullet.new_tag = DONT_DISPLAY;
                    set_new_args(&bird2_bullet);
                        break;
                }
            }
            bird2_bullet.new_y += 1;
            set_new_args(&bird2_bullet);
            usleep(TIME_CONSTANT/2);
        }
          bird2_bullet.new_tag = DONT_DISPLAY;
        set_new_args(&bird2_bullet);
}


void *bird3_shoot_bullet()
{
        bird3_bullet.new_tag = DISPLAY;
        bird3_bullet.new_x = bird3_coor.x;
        bird3_bullet.new_y = bird3_coor.y + 8;
        set_new_args(&bird3_bullet);

        while((bird3_bullet.new_y) < 240)
        {
            if(bird3_bullet.new_y >= 212) {
                if (check_collision_with_ship(&bird3_bullet))
                {
                    if (ship_lives == 0)
                    {
                        ship.new_name = EXPLOSION_NAME;
                        set_new_args(&ship);
```

```
                                usleep(TIME_CONSTANT*2);
                        ship.new_tag = DONT_DISPLAY;
                        set_new_args(&ship);
                                ship_lives--;
                    }
                    else
                        ship_lives--;
                                char prev_name = ship.new_name;
                        ship.new_name = EXPLOSION_NAME;
                        set_new_args(&ship);
                        usleep(TIME_CONSTANT/2);
                        ship.new_name = prev_name;
                        set_new_args(&ship);
                    bird3_bullet.new_tag = DONT_DISPLAY;
                    set_new_args(&bird3_bullet);
                            break;
                }
            }
            bird3_bullet.new_y += 1;
            set_new_args(&bird3_bullet);
            usleep(TIME_CONSTANT/2);
        }
           bird3_bullet.new_tag = DONT_DISPLAY;
        set_new_args(&bird3_bullet);
}


void *move_birds()
{
   while(1){
        for (int i = 0; i < LAST_COLUMN - 16; i++)
        {
                bird1_coor.x = i;
                bird1.new_x= bird1_coor.x;
                bird2_coor.x = i + 8;
            bird2.new_x= bird2_coor.x;
                bird3_coor.x = i + 16;
            bird3.new_x= bird3_coor.x;
                bird1.new_name = BIRD_RIGHT_NAME;
                bird2.new_name = BIRD_RIGHT_NAME;
                bird3.new_name = BIRD_RIGHT_NAME;
                set_new_args(&bird1);
            set_new_args(&bird2);
            set_new_args(&bird3);
```

```
                if (bird1_coor.x >= ship_coor.x - 2 && bird1_coor.x <= (ship_coor.x + 2) &&
bird1_bullet.new_tag == DONT_DISPLAY)
                {
                        pthread_create(&bird1_thread_shoot, NULL, bird1_shoot_bullet, NULL);
                }
                if (bird2_coor.x >= ship_coor.x - 2 && bird2_coor.x <= (ship_coor.x + 2) &&
bird2_bullet.new_tag == DONT_DISPLAY)
            {
                pthread_create(&bird2_thread_shoot, NULL, bird2_shoot_bullet, NULL);
            }
                if (bird3_coor.x >= ship_coor.x - 2 && bird3_coor.x <= (ship_coor.x + 2) &&
bird3_bullet.new_tag == DONT_DISPLAY)
            {
                pthread_create(&bird3_thread_shoot, NULL, bird3_shoot_bullet, NULL);
            }
                usleep(TIME_CONSTANT*10);
        }

        bird1.new_name = BIRD_STANDSTILL_NAME;
      bird2.new_name = BIRD_STANDSTILL_NAME;
      bird3.new_name = BIRD_STANDSTILL_NAME;
        set_new_args(&bird1);
      set_new_args(&bird2);
      set_new_args(&bird3);
        usleep(TIME_CONSTANT*10);

        for (int i = LAST_COLUMN - 16; i > 0; i--)
      {
            bird1_coor.x = i;
            bird1.new_x= bird1_coor.x;
            bird2_coor.x = i + 8;
            bird2.new_x= bird2_coor.x;
            bird3_coor.x = i + 16;
            bird3.new_x= bird3_coor.x;
                bird1.new_name = BIRD_LEFT_NAME;
            bird2.new_name = BIRD_LEFT_NAME;
            bird3.new_name = BIRD_LEFT_NAME;
            set_new_args(&bird1);
            set_new_args(&bird2);
            set_new_args(&bird3);
                if (bird1_coor.x >= ship_coor.x - 2 && bird1_coor.x <= (ship_coor.x + 2) &&
bird1_bullet.new_tag == DONT_DISPLAY)
            {
                pthread_create(&bird1_thread_shoot, NULL, bird1_shoot_bullet, NULL);
            }
```

```
                if (bird2_coor.x >= ship_coor.x - 2 && bird2_coor.x <= (ship_coor.x + 2) &&
bird2_bullet.new_tag == DONT_DISPLAY)
            {
                pthread_create(&bird2_thread_shoot, NULL, bird2_shoot_bullet, NULL);
            }
                 if (bird3_coor.x >= ship_coor.x - 2 && bird3_coor.x <= (ship_coor.x + 2) &&
bird3_bullet.new_tag == DONT_DISPLAY)
            {
                pthread_create(&bird3_thread_shoot, NULL, bird3_shoot_bullet, NULL);
            }

                usleep(TIME_CONSTANT*10);

        }
      bird1.new_name = BIRD_STANDSTILL_NAME;
      bird2.new_name = BIRD_STANDSTILL_NAME;
      bird3.new_name = BIRD_STANDSTILL_NAME;
      set_new_args(&bird1);
      set_new_args(&bird2);
      set_new_args(&bird3);
      usleep(TIME_CONSTANT*10);
   }
}

int main()
{
  struct usb_keyboard_packet packet;
  int transferred;
  char keystate[15];
  int m;
  static const char filename[] = "/dev/vga_ball";
  score = 0;
  data.score1=score;
  set_alldata(&data);


  printf("VGA ball Userspace program started\n");


  if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
  }

  if ( (keyboard = openkeyboard(&endpoint_address)) == NULL) {
```

```
        fprintf(stderr, "Did not find a joystick\n");
        exit(1);
  }

  set_new_args(&ship);
  pthread_create(&move_birds_thread, NULL, move_birds, NULL);

  while(1){
        libusb_interrupt_transfer(keyboard, endpoint_address, (unsigned char *) &packet,
sizeof(packet), &transferred, 0);
    if (transferred == sizeof(packet)) {
            sprintf(keystate, "%02x %02x %02x %02x %02x", packet.modifiers,
packet.keycode[0], packet.keycode[1], packet.keycode[2], packet.keycode[3]);

            m = input_moves(keystate);
            switch(m)
            {
                case(-1):
                    move_ship_left();
                    break;
                case(0):
                    if(ship_bullet.new_tag == DONT_DISPLAY)
                        pthread_create(&ship_thread_shoot, NULL,
ship_shoot_bullet, NULL);
                    move_ship_left();
                    break;
                case(1):
                    move_ship_right();
                    break;
                case(2):
                    if(ship_bullet.new_tag == DONT_DISPLAY)
                   pthread_create(&ship_thread_shoot, NULL, ship_shoot_bullet, NULL);
                move_ship_right();
                break;
                   case (3):
                    if(ship_bullet.new_tag == DONT_DISPLAY)
                   pthread_create(&ship_thread_shoot, NULL, ship_shoot_bullet, NULL);
                break;
            }

            if(bird1_lives < 0 && bird2_lives < 0 && bird3_lives < 0){
                printf("You won!");
                exit(1);
            }
            if(ship_lives <0){
```

```
                        printf("You lost");
                        exit(1);
                }

        }

        /*if(bird1_lives == 0)
            score1=1;
    else if(bird1_lives ==-1)
            score1=2;

    if(bird2_lives == 0)
            score2=1;
    else if(bird2_lives == -1)
            score2=2;

    if(bird3_lives == 0)
            score3=1;
    else if(bird3_lives == -1)
            score3=2;

    score=score1+score2+score3;
       printf("%d\n",score);
    data.score1 = score;
    set_alldata(&data);*/




 }



 pthread_join(move_birds_thread, NULL);

 printf("User Program Terminated\n");
 return 0;
}
```

usbkeyboard.h

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H
```

```
#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1

struct usb_keyboard_packet {
  uint8_t modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};

/* Find and open a USB keyboard device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

usbkeyboard.c

```
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 * http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *keyboard = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  /* Start the library */
```

```
if ( libusb_init(NULL) < 0 ) {
  fprintf(stderr, "Error: libusb_init failed\n");
  exit(1);
}

/* Enumerate all the attached USB devices */
if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
  fprintf(stderr, "Error: libusb_get_device_list failed\n");
  exit(1);
}

/* Look at each device, remembering the first HID device that speaks
   the keyboard protocol */

for (d = 0 ; d < num_devs ; d++) {
  libusb_device *dev = devs[d];
  if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
    fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
    exit(1);
  }

  if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
    struct libusb_config_descriptor *config;
    libusb_get_config_descriptor(dev, 0, &config);
    for (i = 0 ; i < config->bNumInterfaces ; i++)
         for ( k = 0 ; k < config->interface[i].num_altsetting ; k++ ) {
           const struct libusb_interface_descriptor *inter =
             config->interface[i].altsetting + k ;
           if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
                inter->bInterfaceProtocol == 0) {
             int r;
             if ((r = libusb_open(dev, &keyboard)) != 0) {
               fprintf(stderr, "Error: libusb_open failed: %d\n", r);
               exit(1);
             }
             if (libusb_kernel_driver_active(keyboard,i))
               libusb_detach_kernel_driver(keyboard, i);
             libusb_set_auto_detach_kernel_driver(keyboard, i);
             if ((r = libusb_claim_interface(keyboard, i)) != 0) {
               fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
               exit(1);
             }
             *endpoint_address = inter->endpoint[0].bEndpointAddress;
             goto found;
           }
```

```
            }
        }
    }

found:
    libusb_free_device_list(devs, 1);

    return keyboard;
}
```