BREAKOUT GAME REMASTERED

Wang Chen, Youfeng Chen, Zheyuan Song, Angzi Xu
Columbia University
{wc2794,yc3999,zs2527,ax2157}@columbia.edu

1 INTRODUCTION

Our goal is to rebuild the classic "Arkanoid" breakout-like arcade game initially published in 1986. For our game, the player's goal is to use an old-fashioned controller to control a paddle at the bottom of the screen to bounce a ball onto some bricks and clear all bricks to win the game. If the player fails to catch the ball three times, the player will lose.

There are a total of two stages, and stage 2 will be a little more complicated than stage 1. Each time the player lets the ball hit a brick, the brick will disappear, and the player will get 10 points. Other than the play area, information about the current score, current stage, and HP remaining will also be shown on the screen.

The system architecture is shown in the Figure 1.

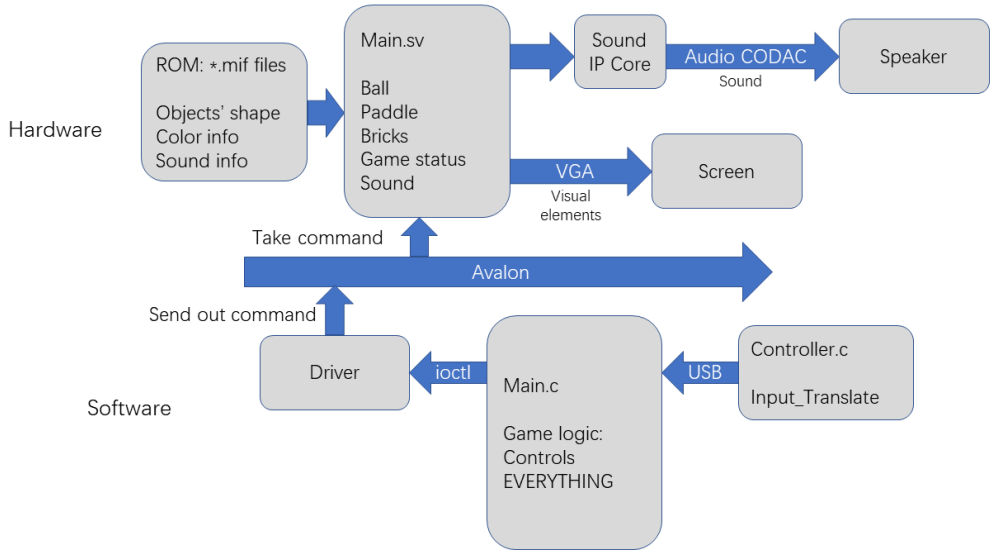

Figure 1: System Architecture.

2 HARDWARE DESIGN

2.1 GRAPHICS

a) General information

The Breakout game uses a lot of graphic elements repeatedly. In order to save the onboard storage, also for good displaying quality, we decide to use the tile and sprite method to build up our graphics module. Prepared tilemap for graphics elements are listed in the Table 1.

All tiles have the same size of 16 pixels * 16 pixels, and they will be stored in the ROM. The output of the tilemap will have 32 bits, every 2 bits for one pixel. The output is shifted according to the

Address (HEX)	Interpretation	Address (HEX)	Interpretation
0-f	Horiz. Border	E0-ef	Number 3
10-1f	Vertical Border	F0-ff	Number 4
20-2f	L Corner	100-10f	Number 5
30-3f	R Corner	110-11f	Number 6
40-4f	L Half Brick	120-12f	Number 7
50-5f	R Half Brick	130-13f	Number 8
60-6f	Letter S	140-14f	Number 9
70-7f	Letter C	150-15f	Heart Symbol
80-8f	Letter O	160-16f	Letter T
90-9f	Letter R	170-17f	Letter A
A0-af	Letter E	180-18f	Letter G
B0-bf	Number 0	190-19f	Letter N
C0-cf	Number 1	1a0-1af	Symbol "!"
D0-df	Number 2		

Table 1: Tilemap for Graphics Elements.

variable hcount so that the two bits at the rightmost of the output always describe their corresponding pixel. Other graphics elements of regular shape and solid color were directly assigned to specific areas and did not use tiles. For sprites, all elements are in a specific layer to ensure proper overlay of all elements. Objects at an upper layer will be displayed on top of the lower layers in case of conflict. The order of layers is shown in the Table 2

Layer Number	Interpretation
1 (top)	Ball
2	Paddle
3	Gray unused area at the sides
4	All elements from the tilemap
5 (bottom)	Background color

Table 2: Order of Layers.

The result display of the game's initial status is shown in Figure 2.

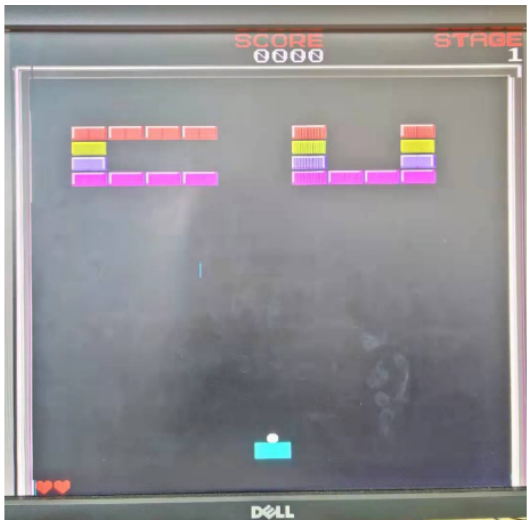

Figure 2: Display of game's initial status.

The play area has a size of 448 pixels * 480 pixels, which means it has 28 tiles horizontally and 30 tiles vertically.

b) Color assignment for tile-map-based elements

According to the original game design of "Arkanoid," each tile only takes up to four different colors, which 2-bit binary numbers can interpret. So that in the tilemap file, every two bits in a line indicates one color. The arrangement of the color map is shown in Table 3.

Address (HEX)	Colors for
0-3	All Border
4-7	All letters/symbols
8-b	Grey brick
c-f	Red brick
10-13	Yellow brick
14-17	Blue brick
18-1b	Purple brick
1c-1f	Green brick

Table 3: Arrangement of Color Map.

The output of the tilemap will be used as the address of the color map, and the 6-HEX RGB output from the color map will be used for display. Although the output of the tilemap for each pixel only could contain numbers from 00 to 11, we manually add offsets when designing the layout so that tiles know which set colors they should use. For example, if we want to display a grey brick, we add 8 to the 2-bit number for each pixel so that the actual color output is within the correct range, from 8 (2'b0 + 8) to B (2'b11+8).

c) Directly assigned elements

Ball, paddle, and two unused gray areas are directly assigned without using a tilemap. They are all regular-shaped elements with only one solid color. The ball and the paddle, which need to move around, are set based on a single-pixel point. As updated coordinate information about that point comes in from the software side, their assigned area can be automatically changed so that they can move freely.

d) Graphic Design block diagram

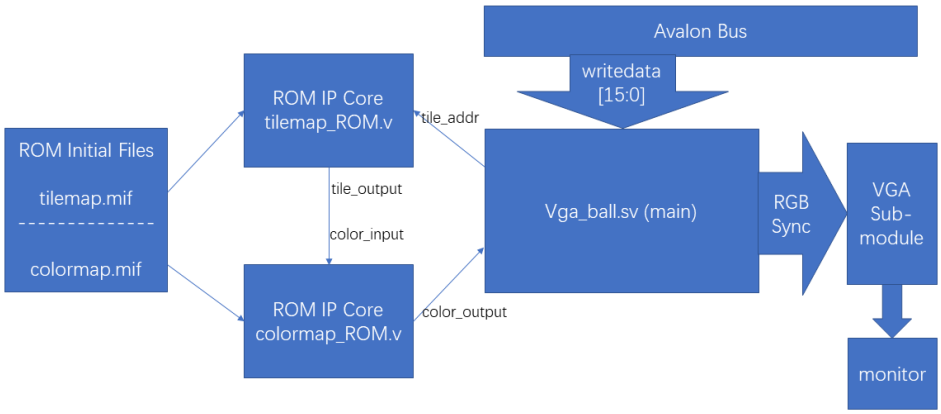

Figure 3: The block diagram of design.

Figure 3 shows the block diagram of design. The ROM cores will return the bits stored at a specific line according to the address they receive from the main program. At every 16 pixels * 16 pixels

block, if it requires tilemap, the main program will send addresses to tilemap-ROM.v to get the color arrangement information of each pixel line. Then, the output from the tilemap-ROM.v will be processed and sent to colormap-ROM.v, two bits (recall that every two bits represent one pixel) at a time as the address so that the colormap-ROM.v could return a 6-HEX color code for each pixel for displaying.

For parts that do not require tilemap, a specific area will be assigned as ranges of hcount and vcount. The hardware will display one color if the hcount and vcount are in that range. Depending on the information received from software via Avalon Bus, the main program will know, for example, if it needs or does not need to display something or where it should move the ball and paddle. According to these instructions, signals will be sent to different modules and executed to see the game running on the screen.

2.2 AUDIO

There are three audio jacks on the DE1-SoC: a line out, a line in, and a microphone jack. The connectors are connected to a Wolfson WM8731 audio CODEC (coder/decoder) chip. This chip contains ADCs (analog to digital converters) and DACs (digital to analog converters) attached to the analog connectors, and a digital interface links to the FPGA. Before utilizing this audio chip, the user must first set up it using the I2C interface. A unique feature of the DE1-Soc board is the presence of an I2C multiplexer, which allows users to configure the WM8731 from both the FPGA and the HPS system on board. The default mode is FPGA, and we will set it in this manner.

Figure 4 is the block diagram of our audio architecture.

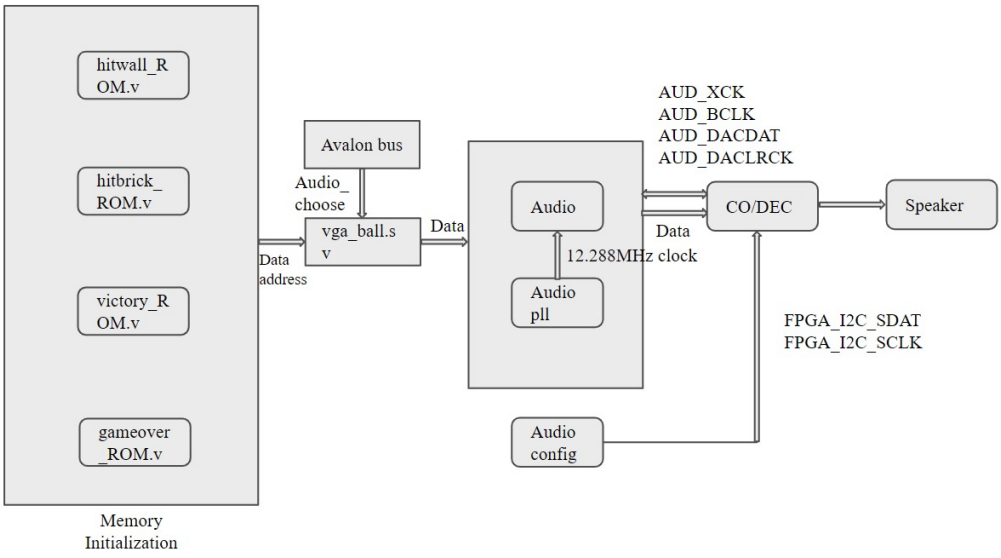

Figure 4: Audio Architecture.

We must first process the audio. We took four MP3 audio files, hitwall, hitbrick, gameover, and victory, and subsampled them at 8KHz with audacity. We conduct subsampling to retain one channel and eventually convert it to 16-bit mono PCM.wav since we configured the WM8731 in left-justified mode, which means the left and the right channel information is the same. Following that, we ensure that the bitrate of the.wav is 128Kbps. Second, use a hex editor to remove any incorrect information from the beginning of the.wav file. Finally, convert the generated .wav files to .mif files using Python.

The user-space audio chooses a signal used to control which sound is played. So the VGA module sends data to the IP core audio module, which includes a FIFO and manages timing for us (Here, we write the audio processing code directly into the VGA main code). Because the FIFO frequency

is 48kHz, we manually inserted a frequency divider of 6 into the VGA module. The audio module is then connected to the WM8731 through the port.

The WM8731 is programmed with a 16-bit data width, an 8kHz sampling rate, and a left-justified mode. The audio_pll_0 module provides a clock for the audio chip at 12.288 MHz and is attached to the board's AUD XCK port. The audio_pll_0 setup is shown Figure 5

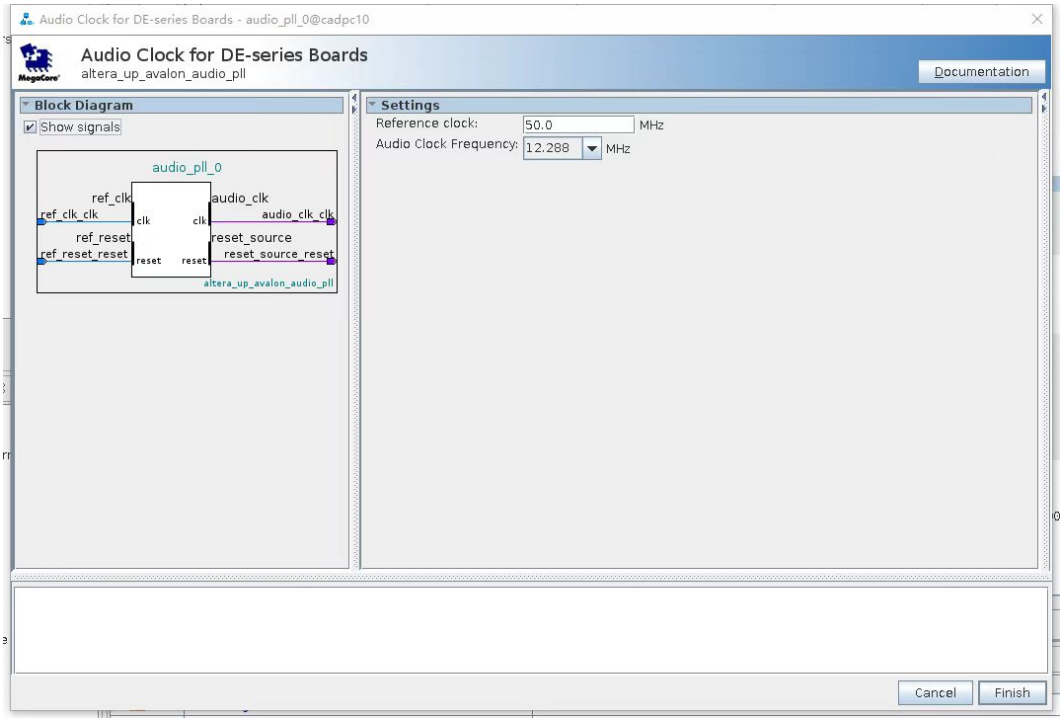

Figure 5: Audio_PLL_0 Configuration.

The cabling between the chip and the audio setup comes next. We adhere to the University of Cambridge's guidelines Aud.

The configuration procedure is summarized below. SCL for clock and SDAT for data are required by the I2C interface. Both signals are high in the idle state. They will pull the SDAT line low while holding the clock line high resulting in a start condition. The host then communicates the device's address to which it wishes to communicate. After delivering the 7-bit address and R/W bit, the host releases the data line. While the data line's status is not actively controlled, a pull-up resistor pulls it high. If the I2C device gets its address, it pulls the data line low on the following clock cycle, referred to as the ACK signal. After that, send two bytes of data. After all data transfers have been completed, a STOP condition is established by raising the clock line while keeping the data line low. The configuration is shown in Figure 6 and Figure 7.

Finally, we need to reconnect the configured QYSY. The reconnected QYSY is shown in the Figure 8.

Audio Issues and Solutions

Our first test found that the audio could not be played correctly when quickly hitting multiple walls or bricks. For example, when two bricks were hit in a row, the audio for hitting a brick was only triggered once. So we shortened the sound effect time and increased the sampling frequency in the hardware code. This problem disappeared on the second test.

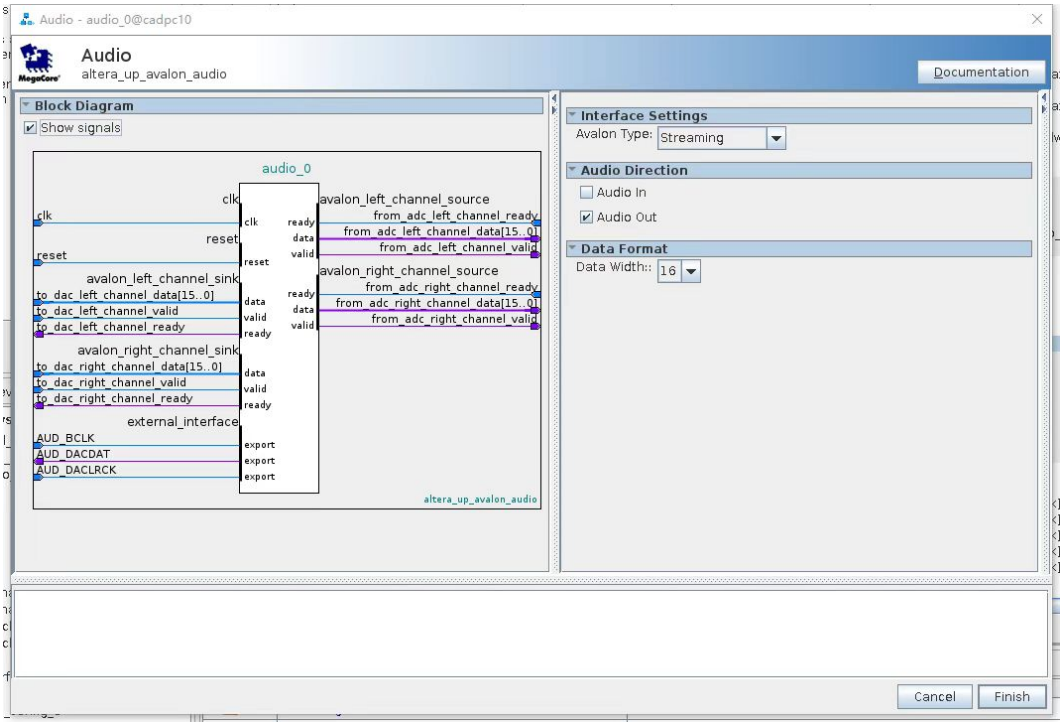

Figure 6: Audio IP core.

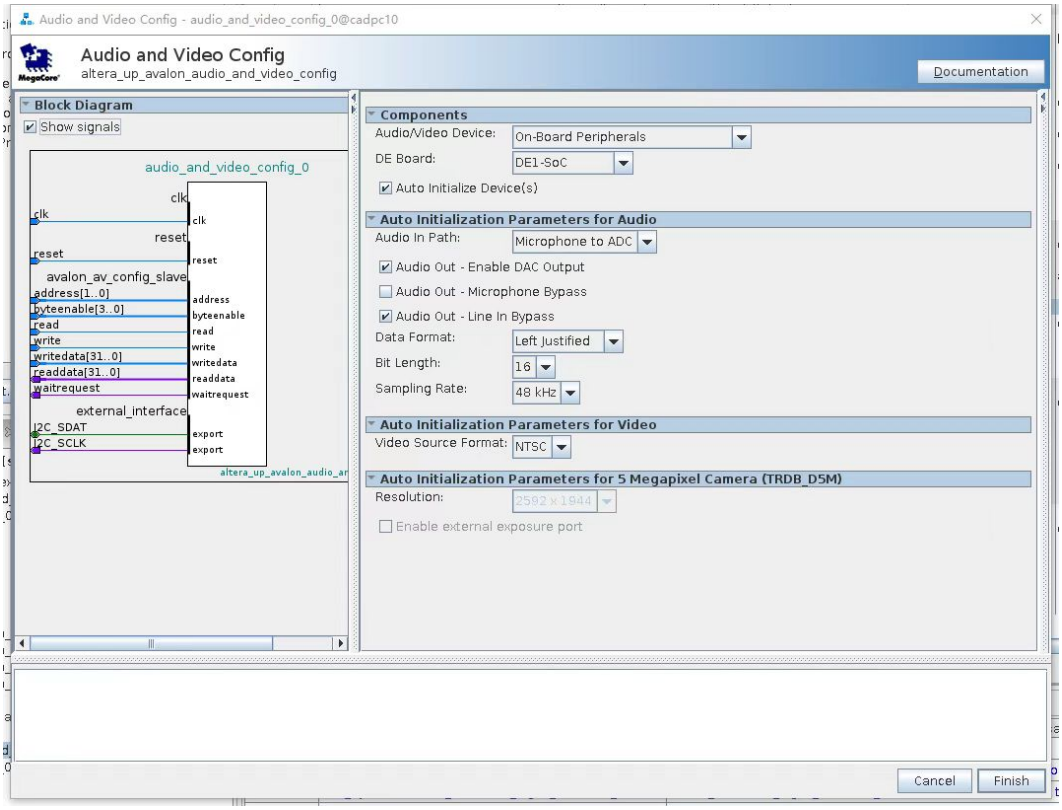

Figure 7: Audio and Video Configuration.

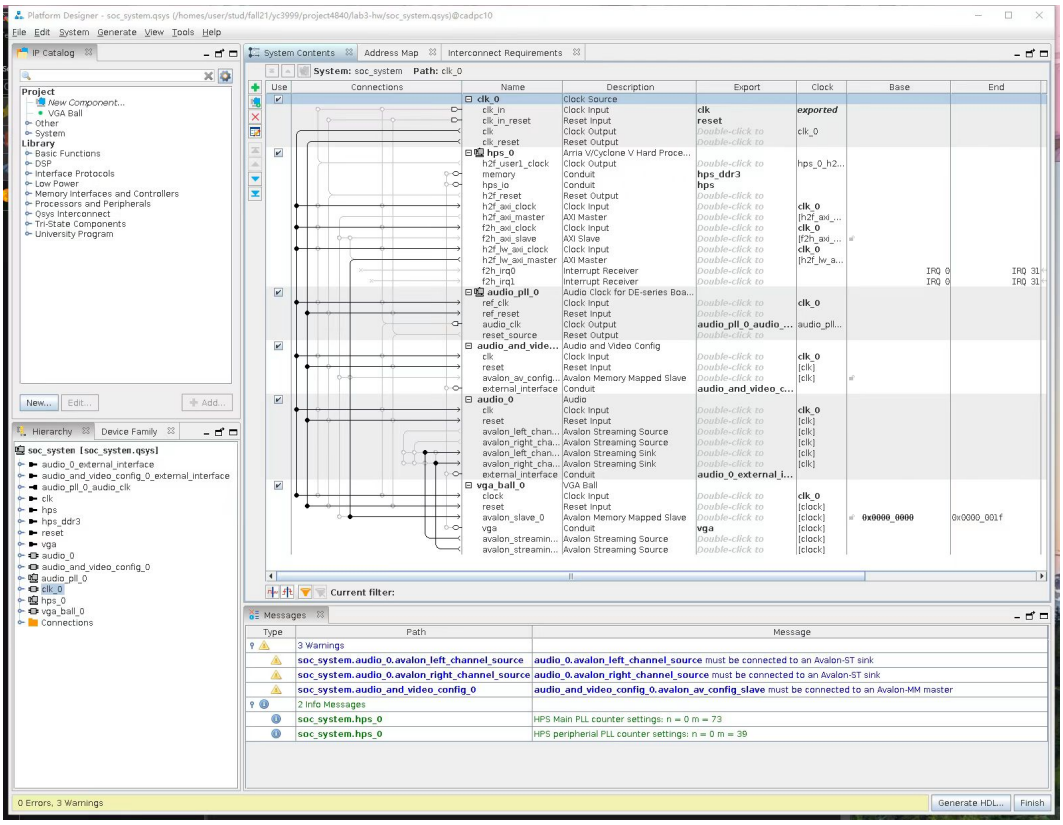

Figure 8: The connection of QYSY.

3 SOFTWARE DESIGN

3.1 INPUT

a) General information

In this game, we will use the controller as the input tool. The controller we used is shown in Figure 9.

Figure 9: "Model_GP 100" controller.

This controller will work just like a USB keyboard, and it can be connected to the FPGA board by its wired USB Port. It has ten keys, and in this game, we will use 6 of 10 to realize our desired functions. The function of each key of the controller is shown in Table 4:

Button	Function
Left arrow	Move the paddle to the left
Right arrow	Move the paddle to the right
Start	When one round of the game ends, restart
A	Launch the ball from the paddle
X+Y	Implement cheating mode to quickly end the game

Table 4: Button Function of controller.

Our controller can send data through the USB interface at each interrupt. To find the data obtained when different keys are pressed, we can print the value of each keypress.As shown below, we can see the values corresponding to the "Up", "Down", "Left", "Right", "Start", "A", "X","Y",and "X+Y" buttons, as shown in Table 5.

Button	value0	value1	value2	value3	value4	value5	value6
left	0	127	0	128	128	15	
right	255	127	0	128	128	15	
up	127	0	0	128	128	15	
down	127	255	0	128	128	15	
A	127	127	0	128	128	47	
restart	127	127	0	128	128	15	32
X + Y	127	127	0	128	128	15	9

Table 5: Value of Each Key Press.

With these data, we can use buff[] (the location of the data) to judge the condition of the keypress.

b) Audio-sounds effect

In the beginning, the audio from the hardware part is transferred to the software part. The software part will send a test signal to the hardware part. Using "data.sound_effect = sound_effect;" to realize

this function.Then the software part will process the audio. When the game loses, it will have the lost background music. Here using ”sound_effect = SOUND_GAME_OVER;” to gain the music. When the ball hits bricks, use ”sound_effect = SOUND_HIT_BRICK;” to gain the music. When the ball hits walls or paddle, use ”sound_effect = SOUND_WALL_PAD;” to gain the music. When the game wins, it will play the victory sound. Here using ”sound_effect = SOUND_DEFULT;” to gain the music.

3.2 GAME LOGIC

a) Flow chart

Before the game starts, the ball is stationary on the paddle, and the paddle can be moved left and right at this time. The ball will move in sync with the paddle. Press the ”A” key to launch the ball. When all the bricks in Stage 1 are destroyed, the game will automatically enter stage 2. Then, after all the bricks in Stage 2 are destroyed, the player wins. Whenever the paddle doesn't catch the ball, the player loses an HP. When all three HP points are lost, the game fails and ends. Pressing the ”START” key allows the player to restart the game conveniently. Press the ”X+Y” keys to cheat in the game, making the game easy so that the player can quickly win the game.

The entire game flow chart of the Breakout game is shown in Figure 10.

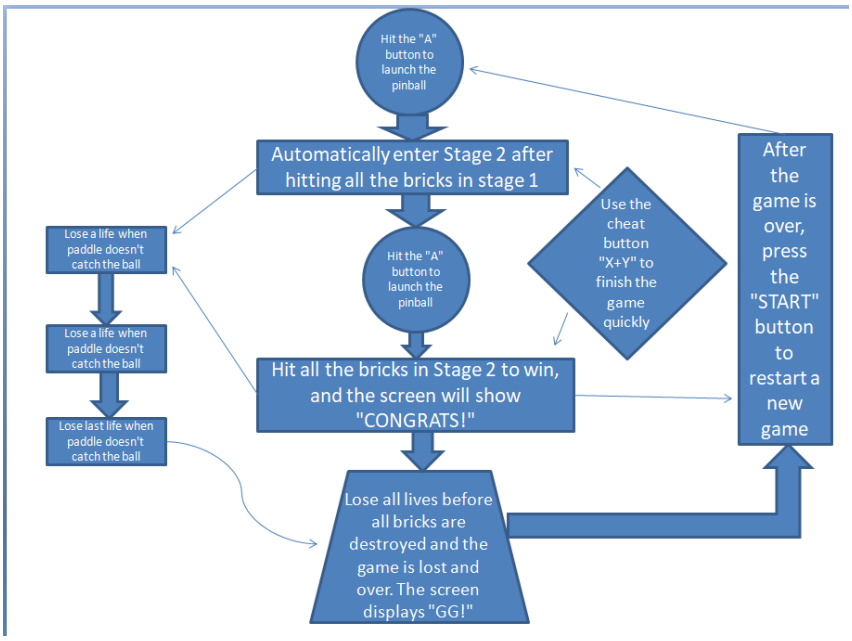

Figure 10: Game Flowchart.

b) Initialization

Before making the whole game, we need to initialize the game. Define some items and assign them corresponding initialized values.

First, we use the horizontal coordinate data.x_ball and the vertical coordinate data.y_ball to determine the ball's position. Let him be located at the coordinate point (208,425) at the beginning, at this time ball_h = 208; ball_v = 425;.

For paddle, we use the same method as for small balls. Let the initial horizontal position of the paddle be 208, and the vertical coordinates remain unchanged. Using “data.x_pad = 208;”. Besides, in order to control the movement range of the paddle within the game screen, use ”(screen_left¡data.x_pad + pad_diff) (data.x_pad + pad_diff¡screen_right)” to limit it.

For bricks, use ”data.brick1 = convert2bin(brick_matrix[0], 0);” to ”data.brick6 = convert2bin(brick_matrix[5], 5);” to assign data to bricks. In this process, the conversion between decimal and binary is required, so a function convert2bin needs to be defined here.

```
long long convert2bin( int x[13], int color) {
  long long  y = color; // 000 001 010 011 100 101

  // 1.color
  for (int i = 0; i < 13; i++){
    y = y * 2 + x[i];
  }
  return(y);
}

data.brick1  = convert2bin( brick_matrix[0], 0 );
data.brick2  = convert2bin( brick_matrix[1], 1 );
data.brick3  = convert2bin( brick_matrix[2], 2 );
data.brick4  = convert2bin( brick_matrix[3], 3 );
data.brick5  = convert2bin( brick_matrix[4], 4 );
data.brick6  = convert2bin( brick_matrix[5], 5 );
```

After defining each brick, we need to layout the bricks throughout the game map. Here the matrix brick_matrix[i][j] is used to set up the bricks. The whole map has six rows and 13 columns, ”1” means there are bricks, and ”0” means no bricks. The method is constructed as shown below:

```
 int brick_matrix[6][13] = {
    {0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,1,1,1,1,0,0,1,0,0,1,0,0},
    {0,1,0,0,0,0,0,1,0,0,1,0,0},
    {0,1,0,0,0,0,0,1,0,0,1,0,0},
    {0,1,1,1,1,0,0,1,1,1,1,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0}
  };
```

After completing this, the initialization of all statements is completed.

c) Movement logic

The speed of the ball can be seen as a vector, composed of a horizontal velocity and a vertical velocity. We decided to make the vertical speed direction twice as fast as the horizontal, as shown in Figure 11.

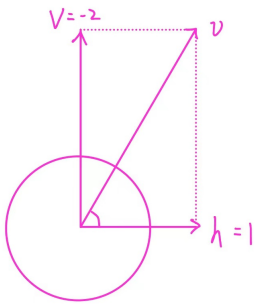

Figure 11: Ball’s Movement.

For the player's paddle movement. In order to let the player move the paddle left and right, we use "data.x_pad = data.x_pad + pad_diff;" to control. The positive and negative values of "pad_diff" can represent different directions.

3.3 HIT LOGIC

a) Wall

Define the boundaries of the three walls: the top wall's vertical position is 53, the right wall's horizontal position is 411, and the left wall's horizontal position is 5. When the ball hits the wall, there are three situations: when it hits the wall on the right, the vertical direction of the ball does not change, and the speed in the horizontal direction is reversed. When hitting the top wall, the horizontal direction of the ball does not change, and the vertical velocity is reversed. When hitting the wall on the left, the ball's vertical velocity does not change, and the horizontal velocity direction is reversed. Collisions occur immediately when the distance between the center coordinates of the ball and the boundary coordinates of the wall is equal to the radius of the ball. As shown in figure 15, the change of the vector of the ball before and after the collision can be clearly seen in Figure 12.

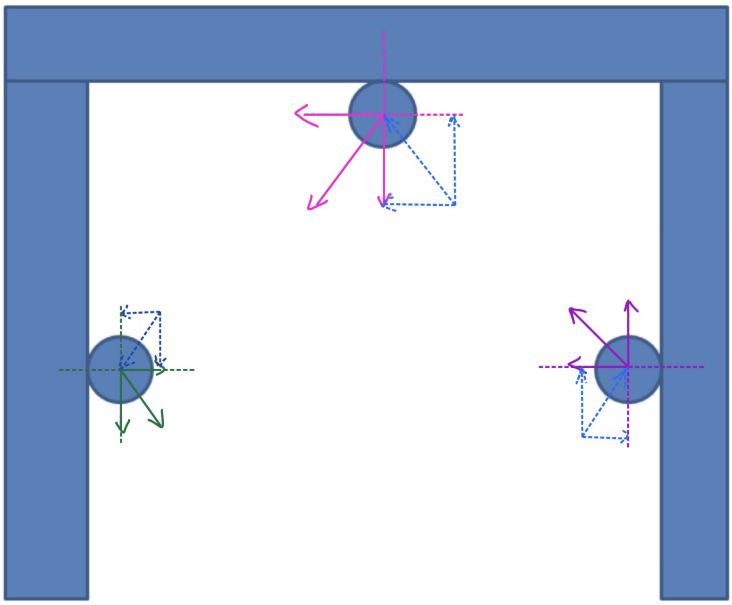

Figure 12: Ball Hit Wall.

b) Paddle

When the ball hits the paddle, it can only collide from above the paddle. When the ball collides with the paddle, the horizontal direction of the ball does not change, and the speed in the vertical direction is reversed. A judgment range can be defined to determine better when the ball should be seen as hitting the paddle. We made a rectangle with the upper boundary of the paddle as the base and the radius of the ball as the width. Command to collide immediately when the coordinate point of the sphere's center falls within this rectangle. As shown in the Figure 13, the red shaded part on the figure is the rectangular collision range.

c) Bricks

When the ball hits the brick, it can go from four directions. Here we define a hitOnBrick function to indicate that the ball hits the brick. Return flag = 0 if not hit on the brick; return flag = 1 if hit on the brick from the top; return flag = 2 if hit on the brick from the bottom; return flag = 3 if hit on the brick from the left; return flag = 4 if hit on the brick from right; use the flag to determine the ball's

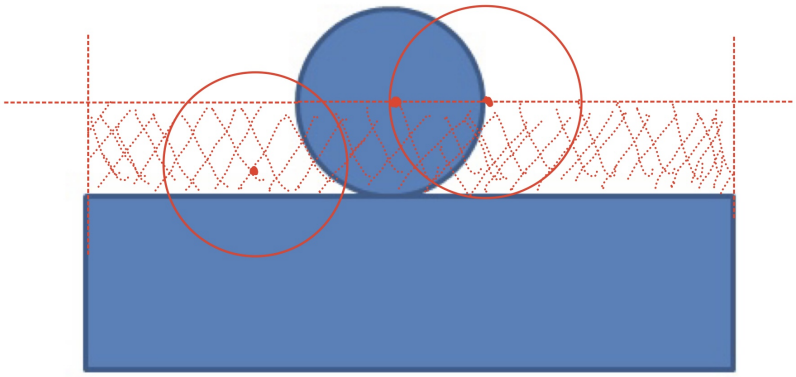

Figure 13: Ball Hit Paddle.

velocity change; after the collision, the brick disappears, and the corresponding brick status should be changed into 0. To better determine the four collision situations, we also use the rectangular range to define the collision range. Make four rectangles with the four sides of the brick as the base and the radius of the ball as the width. The four rectangles represent the determination range when the ball collides from the top, bottom, left, and right. As shown in the figure, the yellow shaded range in the figure is the collision judgment range when the ball hits the brick, as shown in Figure 14

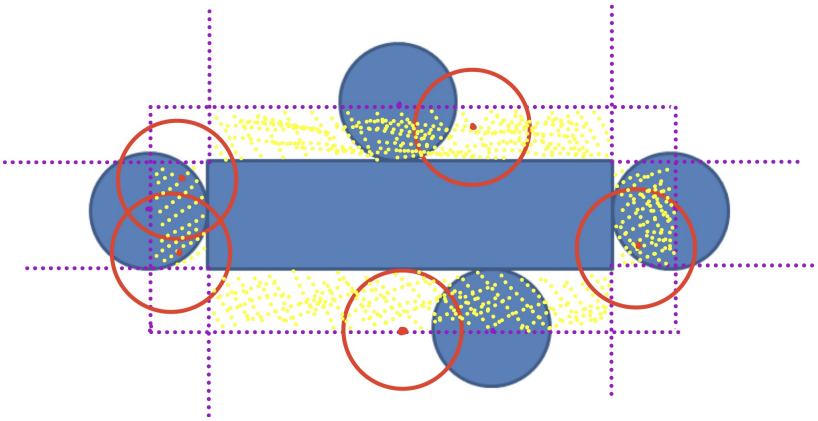

Figure 14: Ball Hit Brick.

The ball hits the brick, the wall, or the paddle. All hits' conditions are taken into account, and this information will be used in other sections to make the game more playable.

3.4 PLAYER LOGIC

a) Score

In the "score" section, start with 0 points and add 10 points each time you hit and destroy a ball until the end of the game. The "score" part in the hardware part is connected to the software part in hexadecimal form. That is to say, each of the four digits in "score" is represented by four digits: 0000 0000 0000 0000; so the function hex2hexadecimal is introduced to convert between hexadecimal and binary.

```
25 // vec to binary
26 long hex2hexadecimal(int score) {
```

```
27    int x[16];
28
29    for (int j = 0; j < 16; j++) {
30      x[j] = 0;
31    }
32
33    int idx = 15; // 12-15
34
35    //printf("original score %d\n",score);
36    while (score > 0) {
37      int num = score % 10;
38
39      int cur_idx = idx;
40      while (num > 0) {
41        x[cur_idx] = num % 2;
42        num = (int)(num/2);
43        cur_idx -= 1;
44      }
45      score = (int)(score / 10);
46      idx -= 4;
47    }
48
49    long y = 0;
50    for (int i = 0; i < 16; i++){
51      y = y * 2 + x[i];
52    }
53
54    //printf("score %d, %lld \n", score, y);
55    return(y);
56    //vec 2 10
57  }
```

b) Player's three lives

Use game_hp to control the player's life count. When the ordinate of the ball is greater than or equal to 475, execute "game_hp -= 1;". This condition means that the ball is missed by the paddle and falls off the map. When game_hp = 3, it corresponds to two red hearts so that x[14] and x[13] are both 1 to display both hearts. When game_hp = 2, it corresponds to a red heart. At this time, only x[13] is set to 1, which means that only one red heart is displayed. When game_hp = 1, no hearts are displayed. When game_hp = 0, the game ends.

c) Game stage upgrade

The game state is set to two Stage 1 and Stage 2. Stage 1 is set to relatively easy difficulty. Fewer bricks and the ball moves slower. When entering the next stage, Stage 2, the difficulty increases. There are more bricks, and the ball is faster. As shown below is the layout of the bricks corresponding to each of the two stages.

```
58    // stage 1
59    int brick_matrix[6][13] = {
60      {0,0,0,0,0,0,0,0,0,0,0,0,0},
61      {0,1,1,1,1,0,0,1,0,0,1,0,0},
62      {0,1,0,0,0,0,0,1,0,0,1,0,0},
63      {0,1,0,0,0,0,0,1,0,0,1,0,0},
64      {0,1,1,1,1,0,0,1,1,1,1,0,0},
65      {0,0,0,0,0,0,0,0,0,0,0,0,0}
66    };
67
68  // stage 2 map
69    int stage2matrix[6][13] = {
```

```
    {1,0,1,0,1,0,1,0,1,0,1,0,1},
    {1,0,1,0,1,0,1,0,1,0,1,0,1},
    {1,0,1,0,1,0,1,0,1,0,1,0,1},
    {1,0,1,0,1,0,1,0,1,0,1,0,1},
    {1,0,1,0,1,0,1,0,1,0,1,0,1},
    {1,0,1,0,1,0,1,0,1,0,1,0,1}
  };
```

3.5 Win and Loss

a) Win and Loss Logic

Use the function get_game_status to represent the game's victory and defeat logic. get_game_status contains game_stage, game_hp, game_over. Use the value of the contained items to determine the status and value of get_game_status. Win the game when all bricks are cleared, and the game is currently in Stage 2. That is, matrixclear == 1 game_stage == 1 (starting from 0). The function of matrixclear can be realized by defining a function check_clear, as shown below.

```
int check_clear(int matrix[6][13],int x, int y){
  for(int i = 0; i < x; i++){
    for (int j = 0; j < y; j++) {
      if (matrix[i][j] != 0){
            return 0;
      }
    }
  }
  return 1;
}
```

When the game fails, use game_over to indicate that the game fails when game_over = 1. The game is successful when game_over = 2. Then you can define a new parameter re_start, and add some judgment conditions to the loop to realize the restart function of the game.

b) Win and Loss Graph

When the player fails, the screen will display "GG!"; when the player succeeds, the screen will display "CONGRATS!", as shown in Figure 15 and Figure 16.

3.6 Avalon Bus Interface

Figure 17 shows the Avalon bus interface.

(1) Ball X: 9-bit number, controlling the horizontal position of the ball.
(2) Ball Y: 9-bit number, controlling the vertical position of the ball.
(3) Paddle X: 9-bit number, controlling the horizontal position of the paddle.
(4) Score: 4 of 4-bit numbers, each number controls one bit of the score.
(5) Brick lines: 16-bit information, bits [15:13] select color for the bricks, and every bit of [12:0] controls if a specific brick needs to be displayed.
(6) Sound: 3-bit number, 3'b0 means do not play anything, and any number greater than 0 corresponds to one sound effect.
(7) Game status: Bit [0] controls the stage number: input 0 to display 1 and input 1 to display 2. Bits [2:1] control the HP display: 2'b11 for 2 HP, 2'b10 for 1 HP, and 2'b00 for 0 HP. Bit [3] controls if the Win/Lose interface needs to be displayed, and bit [4] determines which to display: 0 for lose and 1 for win.

Figure 15: Win Status Graph.

Figure 16: Fails Status Graph.

	writedata[15:0]																
Address [6:0]	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Note
0								Horizontal Position of the Ball									Ball X
1								Vertical Position of the Ball									Ball Y
2								Horizontal Position of the Paddle									Paddle X
3	Digit 1				Digit 2				Digit 3				Digit 4				SCORE
4	Color Sel.			En 1	En 2	En 3	En 4	En 5	En 6	En 7	En 8	En 9	En 10	En 11	En 12	En 13	Brick line 1
5	Color Sel.			En 1	En 2	En 3	En 4	En 5	En 6	En 7	En 8	En 9	En 10	En 11	En 12	En 13	Brick line 2
...																	
9	Color Sel.			En 1	En 2	En 3	En 4	En 5	En 6	En 7	En 8	En 9	En 10	En 11	En 12	En 13	Brick line 6
a														Sound Sel.			Sound
b												Fin Sel	En Fin	HP		St	Game Status

Figure 17: Avalon bus interface.

4 DIFFICULTIES ENCOUNTERED AND ATTEMPTS

The determination conditions of the ball can be further optimized. At present, the judgment condition of the ball is judged by the rectangular range, which can realize the expected function of a collision. But when the speed of the ball is very fast, or the impact position is in the direction of the side vertex, there may be some collision bugs. To solve this problem, we can make the outer side of the ”hitbox” zigzag so that the ball will not remain still in the judgment zone after it is bounced back.

Our initial design set the tile size to be 15 pixels * 15 pixels for hardware. While 15 is not a power of 2, we encountered some STA problems because the board needed a long time to process those complex calculations. The initial Fmax required for our design is 40 MHz, and our given clock frequency is 50 MHz. This brought some display issues, including glares and unexpected colors, as shown in Figure .

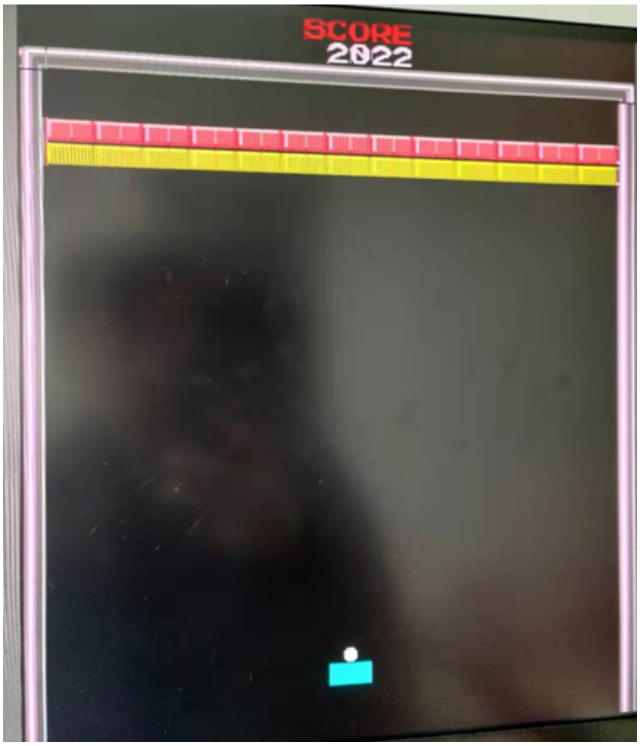

Figure 18: Low-quality display due to STA problem.

We did some optimization, including changing the tile sizing from 15*15 to 16*16 and trying to avoid any unnecessary if-else statements. As a result, our hardware design speeds up significantly, increasing the Fmax from 40 MHz to 70MHz, which will not produce any issue using the given 50 MHz clocks.

ACKNOWLEDGEMENT

We would like to thank Professor Stephen Edwards, and all of the TAs, Martha Barker and Abhijeet Nayak, for your hard work this semester. Thank you for the instructions and help for the 2022 Spring semester!

REFERENCES

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

APPENDIX

A. CODE FOR HARDWARE

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 *
 * Hardware Part of Breakout Game Remastered Project
 * CSEE 4840, Spring 2022, Columbia University
 */

module vga_ball(input logic        clk,
          input logic      reset,
                input logic [15:0]  writedata,
                input logic        write,
                input              chipselect,
                input logic [3:0]  address,

                input left_chan_ready,
                input right_chan_ready,

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

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

   logic [10:0]    hcount, hcount_adj;
   logic [9:0]     vcount;

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

   logic [15:0] score, x_ball, y_ball, x_pad, brick1,
      brick2, brick3, brick4, brick5, brick6, sound_effect,
       game_status;
   logic [4:0] tile_x, tile_y;
   logic [10:0] vcount1;

   assign vcount1 = {1'b0,vcount};

   logic circle;
   logic peddle;
   logic waste;

   logic [10:0] x_ball_adj, x_pad_adj, y_ball_adj;

   assign x_ball_adj = {2'b0,x_ball[8:0]}*2 + 11'd224;
```

```
49      assign x_pad_adj  = {2'b0,x_pad[8:0]}*2 + 11'd224;
50      assign y_ball_adj = {2'b0, y_ball[8:0]};
51
52      assign circle = (((hcount - x_ball_adj) * (hcount -
            x_ball_adj)) + 4 * ((vcount1 - y_ball_adj) * (vcount1
             - y_ball_adj))) < 11'd100; // ball r = 10
53      assign peddle = ((x_pad_adj - 11'd32) < hcount) && (
            hcount < (x_pad_adj + 11'd32)) && (10'd430 <= vcount)
             && (vcount < 10'd446); // pad 32 * 16
54      assign waste = (hcount < 11'd192) || (hcount > 11'd1088)
            ; //needless area
55
56      //Setup tile coordinates
57      assign tile_y = vcount / 16;
58
59      always_comb begin
60          if (hcount >= 192 && hcount < 1088)
61                  tile_x = ((hcount / 2) - 96) / 16;
62          else
63                  tile_x = 5'b11111;
64      end
65
66
67      always_ff @(posedge clk) begin
68        if (reset) begin
69       x_ball <= 16'd208;
70          y_ball <= 16'd425;
71       x_pad <= 16'd208;
72          score <= 16'h4840;
73          brick1 <= 16'b0001000000000000;
74          brick2 <= 16'b0010100000000000;
75          brick3 <= 16'b0100010000000000;
76          brick4 <= 16'b0110001000000000;
77          brick5 <= 16'b1000000100000000;
78          brick6 <= 16'b1010000010000000;
79          sound_effect <= 16'b000;
80       game_status <= 16'b0000000000000110;
81
82
83        end else if (chipselect && write) begin
84          case (address)
85            4'h0 : x_ball <= writedata;
86            4'h1 : y_ball <= writedata;
87            4'h2 : x_pad <= writedata;
88            4'h3 : score <= writedata;
89            4'h4 : brick1 <= writedata;
90            4'h5 : brick2 <= writedata;
91            4'h6 : brick3 <= writedata;
92            4'h7 : brick4 <= writedata;
93            4'h8 : brick5 <= writedata;
94            4'h9 : brick6 <= writedata;
95            4'ha : sound_effect <= writedata;
96            4'hb : game_status <= writedata;
97          endcase
98       end
99       end
100         //----------------Sound----------------
101         reg [13:0] counter;
102         logic flag1;
```

```
103         logic flag2;
104         logic flag3;
105         logic flag4;
106 
107         reg     [8:0]  address1;
108         wire [15:0]  q1;
109         hitbrick_ROM audio1(.address(address1), .clock(clk)
                , .q(q1));//290
110 
111         reg     [9:0]  address2;
112         wire [15:0]  q2;
113         hitwall_ROM  audio2(.address(address2), .clock(clk)
                , .q(q2));//784
114 
115         reg     [13:0]  address3;
116         wire [15:0]  q3;
117         gameover_ROM audio3(.address(address3), .clock(clk)
                , .q(q3));//8489
118 
119         reg [12:0] address4;
120         wire [15:0] q4;
121         victory_ROM audio4(.address(address4), .clock(clk),
                .q(q4));//4504
122 
123         always_ff @(posedge clk) begin
124                 if(reset) begin
125                         counter <= 0;
126                         sample_valid_l <= 0; sample_valid_r
                            <= 0;
127                 end
128                 else if(left_chan_ready == 1 &&
                    right_chan_ready == 1 && counter < 6250)
                     begin
129                         counter <= counter + 1;
130                         sample_valid_l <= 0; sample_valid_r
                            <= 0;
131                 end
132                 else if(left_chan_ready == 1 &&
                    right_chan_ready == 1 && counter ==
                    6250) begin
133                         counter <= 0;
134                         sample_valid_l <= 1; sample_valid_r
                            <= 1;
135                         //------Game over------
136                         if (sound_effect[2:0]==3'd3 ||
                            flag3 ==1'b0) begin
137                                 if (address3 < 14'd8489)
                                    begin
138                                         address3 <=
                                            address3+1;
139                                         flag3 <= 1'b0;
140                                 end
141                                 else begin
142                                         address3 <=0;
143                                         flag3 <= 1'b1;
144                                 end
145                                 sample_data_l <= q3;
146                                 sample_data_r <= q3;
147                         end
```

```
148                             //------Hit Brick------
149                             else if (sound_effect[2:0]==3'd1 ||
                                    flag1 ==1'b0) begin
150                                 if (address1 < 9'd290)
                                        begin
151                                         address1 <=
                                                address1+1;
152                                         flag1 <= 1'b0;
153                                     end
154                                     else begin
155                                         address1 <=0;
156                                         flag1 <= 1'b1;
157                                     end
158                                     sample_data_l <= q1;
159                                     sample_data_r <= q1;
160                             end
161                             //------Hit Wall------
162                             else if (sound_effect[2:0]==3'd2 ||
                                    flag2 ==1'b0) begin
163                                 if (address2 < 10'd784)
                                        begin
164                                         address2 <=
                                                address2+1;
165                                         flag2 <= 1'b0;
166                                     end
167                                     else begin
168                                         address2 <=0;
169                                         flag2 <= 1'b1;
170                                     end
171                                     sample_data_l <= q2;
172                                     sample_data_r <= q2;
173                             end
174                             //------Win------
175                             else if (sound_effect[2:0]==3'd4 ||
                                    flag4 == 1'b0) begin
176                                 if (address4 < 13'd4504)
                                        begin
177                                         address4 <=
                                                address4 + 1;
178                                         flag4 <= 1'b0;
179                                     end else begin
180                                         address4 <= 0;
181                                         flag4 <= 1'b1;
182                                     end
183                                     sample_data_l <= q4;
184                                     sample_data_r <= q4;
185                             end
186
187                             else begin
188                                     sample_data_l <= 0;
189                                     sample_data_r <= 0;
190                             end
191                 end
192                 else begin
193                 sample_valid_l <= 0; sample_valid_r <= 0;
194                 end
195         end
196
197
```

```
198           // ---------------ROM tilemap_ROM.v---------------
199           // Assign objects to specific locations of the
                  screen
200           logic [8:0]  tile_addr;
201           logic [31:0] tile_output;
202           tilemap_ROM tile1(.address(tile_addr),.clock(clk),.
                  q(tile_output));
203
204           always_ff @(posedge clk) begin
205         //-----Static elements------
206         //Side boarder
207         if (tile_y >= 3 && (tile_x == 0 || tile_x == 27))
               begin
208            tile_addr <= 9'h13;
209         //Top boarder
210         end else if (tile_x <= 26 && tile_x >= 1 && tile_y ==
               2) begin
211           tile_addr <= vcount - 32;
212         //L Corner
213         end else if (tile_x == 0 && tile_y == 2) begin
214           tile_addr <= vcount;
215         //R Corner
216         end else if (tile_x == 27 && tile_y == 2) begin
217           tile_addr <= vcount + 16;
218         //SCORE
219         end else if (tile_x <= 16 && tile_x >= 12 && tile_y
              == 0) begin
220                 case (tile_x)
221                         5'd12 : tile_addr <= vcount + 96;
222                         5'd13 : tile_addr <= vcount + 112;
223                         5'd14 : tile_addr <= vcount + 128;
224                         5'd15 : tile_addr <= vcount + 144;
225           5'd16 : tile_addr <= vcount + 160;
226                 endcase
227            //STAGE
228            end else if (tile_x <= 27 && tile_x >= 23 &&
                  tile_y == 0) begin
229                 case (tile_x)
230                         5'd23 : tile_addr <= vcount + 96;
231                         5'd24 : tile_addr <= vcount + 352;
232                         5'd25 : tile_addr <= vcount + 368;
233                         5'd26 : tile_addr <= vcount + 384;
234                         5'd27 : tile_addr <= vcount + 160;
235                 endcase
236         //------Dynamic elements------
237         //Score Numbers
238         end else if (tile_x <= 16 && tile_x >= 13 && tile_y
              == 1) begin
239           case (tile_x)
240                         5'd13 : tile_addr <= vcount + 160 +
                                    (score[15:12]*16);
241                         5'd14 : tile_addr <= vcount + 160 +
                                    (score[11:8]*16);
242                         5'd15 : tile_addr <= vcount + 160 +
                                    (score[7:4]*16);
243                         5'd16 : tile_addr <= vcount + 160 +
                                    (score[3:0]*16);
244                 endcase
245            //Stage Numbers
```

```
246         end else if (tile_x == 27 && tile_y == 1) begin
247                 tile_addr <= game_status[0] ? (vcount +
                        192) : (vcount + 176);
248             //HP Indicator
249             end else if ((tile_x == 1 || tile_x == 2) &&
                    tile_y == 29) begin
250                 case (tile_x)
251                         5'd1 :  begin
252                                                 if (game_status[2])
253                                                         tile_addr
                                                            <=
                                                            vcount -
                                                             128;
254                                                 else
255                                                         tile_addr
                                                            <= 0;
256                                                 end
257                         5'd2 :  begin
258                                                 if (game_status[1])
259                                                         tile_addr
                                                            <=
                                                             vcount -
                                                             128;
260                                                 else
261                                                         tile_addr
                                                            <= 0;
262                                                 end
263                 endcase
264             //Win & Lose
265             end else if (tile_y == 15 && tile_x <= 18 &&
                    tile_x >= 10) begin
266                 if (game_status[3]) begin
267                         if (game_status[4]) begin
268                                 case (tile_x)
269                                                 5'd10 : tile_addr
                                                    <= vcount - 128;
270                                                 5'd11 : tile_addr
                                                    <= vcount - 112;
271                                                 5'd12 : tile_addr
                                                    <= vcount + 160;
272                                                 5'd13 : tile_addr
                                                    <= vcount + 144;
273                                                 5'd14 : tile_addr
                                                    <= vcount - 96;
274                                                 5'd15 : tile_addr
                                                    <= vcount + 128;
275                                                 5'd16 : tile_addr
                                                    <= vcount + 112;
276                                                 5'd17 : tile_addr
                                                    <= vcount - 144;
277                                                 5'd18 : tile_addr
                                                    <= vcount + 178;
278                                 endcase
279                         end else begin
280                                 case (tile_x)
281                                                 5'd13 : tile_addr
                                                    <= vcount + 144;
282                                                 5'd14 : tile_addr
                                                    <= vcount + 144;
```

```
283                                         5'd15 : tile_addr
                                                <= vcount + 178;
284                                         default : tile_addr
                                                <= 0;
285                                 endcase
286                         end
287                 end else begin
288                         tile_addr <= 0;
289                 end
290         //Bricks
291         end else if (tile_x >= 1 && tile_x <= 26 && tile_y >=
                5 && tile_y <= 10) begin
292                 case (tile_y)
293                 //Brick line 1
294                 5'd5 : begin
295                         if (brick1[12]) begin
296                                 case (tile_x)
297                                         5'd1 : tile_addr <=
                                                vcount - 16;
298                                         5'd2 : tile_addr <=
                                                vcount;
299                                 endcase
300                         end else begin
301                                 case (tile_x)
302                                         5'd1 : tile_addr <=
                                                0;
303                                         5'd2 : tile_addr <=
                                                0;
304                                 endcase
305                         end
306                         if (brick1[11]) begin
307                                 case (tile_x)
308                                         5'd3 : tile_addr <=
                                                vcount - 16;
309                                         5'd4 : tile_addr <=
                                                vcount;
310                                 endcase
311                         end else begin
312                                 case (tile_x)
313                                         5'd3 : tile_addr <=
                                                0;
314                                         5'd4 : tile_addr <=
                                                0;
315                                 endcase
316                         end
317                         if (brick1[10]) begin
318                                 case (tile_x)
319                                         5'd5 : tile_addr <=
                                                vcount - 16;
320                                         5'd6 : tile_addr <=
                                                vcount;
321                                 endcase
322                         end else begin
323                                 case (tile_x)
324                                         5'd5 : tile_addr <=
                                                0;
325                                         5'd6 : tile_addr <=
                                                0;
326                                 endcase
```

```
327                             end
328                             if (brick1[9]) begin
329                                     case (tile_x)
330                                             5'd7 : tile_addr <=
                                                    vcount - 16;
331                                             5'd8 : tile_addr <=
                                                    vcount;
332                                     endcase
333                             end else begin
334                                     case (tile_x)
335                                             5'd7 : tile_addr <=
                                                    0;
336                                             5'd8 : tile_addr <=
                                                    0;
337                                     endcase
338                             end
339                             if (brick1[8]) begin
340                                     case (tile_x)
341                                             5'd9 : tile_addr <=
                                                    vcount - 16;
342                                             5'd10: tile_addr <=
                                                    vcount;
343                                     endcase
344                             end else begin
345                                     case (tile_x)
346                                             5'd9 : tile_addr <=
                                                    0;
347                                             5'd10: tile_addr <=
                                                    0;
348                                     endcase
349                             end
350                             if (brick1[7]) begin
351                                     case (tile_x)
352                                             5'd11: tile_addr <=
                                                    vcount - 16;
353                                             5'd12 : tile_addr
                                                  <= vcount;
354                                     endcase
355                             end else begin
356                                     case (tile_x)
357                                             5'd11: tile_addr <=
                                                    0;
358                                             5'd12 : tile_addr
                                                  <= 0;
359                                     endcase
360                             end
361                             if (brick1[6]) begin
362                                     case (tile_x)
363                                             5'd13: tile_addr <=
                                                    vcount - 16;
364                                             5'd14: tile_addr <=
                                                    vcount;
365                                     endcase
366                             end else begin
367                                     case (tile_x)
368                                             5'd13: tile_addr <=
                                                    0;
369                                             5'd14: tile_addr <=
                                                    0;
```

```
370                         endcase
371                     end
372                     if (brick1[5]) begin
373                         case (tile_x)
374                             5'd15: tile_addr <= vcount - 16;
375                             5'd16: tile_addr <= vcount;
376                         endcase
377                     end else begin
378                         case (tile_x)
379                             5'd15: tile_addr <= 0;
380                             5'd16: tile_addr <= 0;
381                         endcase
382                     end
383                     if (brick1[4]) begin
384                         case (tile_x)
385                             5'd17: tile_addr <= vcount - 16;
386                             5'd18: tile_addr <= vcount;
387                         endcase
388                     end else begin
389                         case (tile_x)
390                             5'd17: tile_addr <= 0;
391                             5'd18: tile_addr <= 0;
392                         endcase
393                     end
394                     if (brick1[3]) begin
395                         case (tile_x)
396                             5'd19: tile_addr <= vcount - 16;
397                             5'd20: tile_addr <= vcount;
398                         endcase
399                     end else begin
400                         case (tile_x)
401                             5'd19: tile_addr <= 0;
402                             5'd20: tile_addr <= 0;
403                         endcase
404                     end
405                     if (brick1[2]) begin
406                         case (tile_x)
407                             5'd21: tile_addr <= vcount - 16;
408                             5'd22: tile_addr <= vcount;
409                         endcase
410                     end else begin
411                         case (tile_x)
412                             5'd21: tile_addr <= 0;
```

```
413                                     5'd22: tile_addr <=
                                            0;
414                             endcase
415                     end
416                     if (brick1[1]) begin
417                             case (tile_x)
418                                     5'd23: tile_addr <=
                                            vcount - 16;
419                                     5'd24: tile_addr <=
                                            vcount;
420                             endcase
421                     end else begin
422                             case (tile_x)
423                                     5'd23: tile_addr <=
                                            0;
424                                     5'd24: tile_addr <=
                                            0;
425                             endcase
426                     end
427                     if (brick1[0]) begin
428                             case (tile_x)
429                                     5'd25: tile_addr <=
                                            vcount - 16;
430                                     5'd26: tile_addr <=
                                            vcount;
431                             endcase
432                     end else begin
433                             case (tile_x)
434                                     5'd25: tile_addr <=
                                            0;
435                                     5'd26: tile_addr <=
                                            0;
436                             endcase
437                     end
438             end
439             //Brick line 2
440             5'd6 : begin
441                     if (brick2[12]) begin
442                             case (tile_x)
443                                     5'd1 : tile_addr <=
                                            vcount - 32;
444                                     5'd2 : tile_addr <=
                                            vcount - 16;
445                             endcase
446                     end else begin
447                             case (tile_x)
448                                     5'd1 : tile_addr <=
                                            0;
449                                     5'd2 : tile_addr <=
                                            0;
450                             endcase
451                     end
452                     if (brick2[11]) begin
453                             case (tile_x)
454                                     5'd3 : tile_addr <=
                                            vcount - 32;
455                                     5'd4 : tile_addr <=
                                            vcount - 16;
456                             endcase
```

```
457                         end else begin
458                                 case (tile_x)
459                                         5'd3 : tile_addr <=
                                                    0;
460                                         5'd4 : tile_addr <=
                                                    0;
461                                 endcase
462                         end
463                         if (brick2[10]) begin
464                                 case (tile_x)
465                                         5'd5 : tile_addr <=
                                                    vcount - 32;
466                                         5'd6 : tile_addr <=
                                                    vcount - 16;
467                                 endcase
468                         end else begin
469                                 case (tile_x)
470                                         5'd5 : tile_addr <=
                                                    0;
471                                         5'd6 : tile_addr <=
                                                    0;
472                                 endcase
473                         end
474                         if (brick2[9]) begin
475                                 case (tile_x)
476                                         5'd7 : tile_addr <=
                                                    vcount - 32;
477                                         5'd8 : tile_addr <=
                                                    vcount - 16;
478                                 endcase
479                         end else begin
480                                 case (tile_x)
481                                         5'd7 : tile_addr <=
                                                    0;
482                                         5'd8 : tile_addr <=
                                                    0;
483                                 endcase
484                         end
485                         if (brick2[8]) begin
486                                 case (tile_x)
487                                         5'd9 : tile_addr <=
                                                    vcount - 32;
488                                         5'd10: tile_addr <=
                                                    vcount - 16;
489                                 endcase
490                         end else begin
491                                 case (tile_x)
492                                         5'd9 : tile_addr <=
                                                    0;
493                                         5'd10: tile_addr <=
                                                    0;
494                                 endcase
495                         end
496                         if (brick2[7]) begin
497                                 case (tile_x)
498                                         5'd11: tile_addr <=
                                                    vcount - 32;
499                                         5'd12: tile_addr <=
                                                    vcount - 16;
```

```
500                             endcase
501                         end else begin
502                             case (tile_x)
503                                 5'd11: tile_addr <=
                                        0;
504                                 5'd12: tile_addr <=
                                        0;
505                             endcase
506                         end
507                         if (brick2[6]) begin
508                             case (tile_x)
509                                 5'd13: tile_addr <=
                                        vcount - 32;
510                                 5'd14: tile_addr <=
                                        vcount - 16;
511                             endcase
512                         end else begin
513                             case (tile_x)
514                                 5'd13: tile_addr <=
                                        0;
515                                 5'd14: tile_addr <=
                                        0;
516                             endcase
517                         end
518                         if (brick2[5]) begin
519                             case (tile_x)
520                                 5'd15: tile_addr <=
                                        vcount - 32;
521                                 5'd16: tile_addr <=
                                        vcount - 16;
522                             endcase
523                         end else begin
524                             case (tile_x)
525                                 5'd15: tile_addr <=
                                        0;
526                                 5'd16: tile_addr <=
                                        0;
527                             endcase
528                         end
529                         if (brick2[4]) begin
530                             case (tile_x)
531                                 5'd17: tile_addr <=
                                        vcount - 32;
532                                 5'd18: tile_addr <=
                                        vcount - 16;
533                             endcase
534                         end else begin
535                             case (tile_x)
536                                 5'd17: tile_addr <=
                                        0;
537                                 5'd18: tile_addr <=
                                        0;
538                             endcase
539                         end
540                         if (brick2[3]) begin
541                             case (tile_x)
542                                 5'd19: tile_addr <=
                                        vcount - 32;
```

```
543                                     5'd20: tile_addr <=
                                            vcount - 16;
544                             endcase
545                         end else begin
546                             case (tile_x)
547                                 5'd19: tile_addr <=
                                        0;
548                                 5'd20: tile_addr <=
                                        0;
549                             endcase
550                         end
551                         if (brick2[2]) begin
552                             case (tile_x)
553                                 5'd21: tile_addr <=
                                        vcount - 32;
554                                 5'd22: tile_addr <=
                                        vcount - 16;
555                             endcase
556                         end else begin
557                             case (tile_x)
558                                 5'd21: tile_addr <=
                                        0;
559                                 5'd22: tile_addr <=
                                        0;
560                             endcase
561                         end
562                         if (brick2[1]) begin
563                             case (tile_x)
564                                 5'd23: tile_addr <=
                                        vcount - 32;
565                                 5'd24: tile_addr <=
                                        vcount - 16;
566                             endcase
567                         end else begin
568                             case (tile_x)
569                                 5'd23: tile_addr <=
                                        0;
570                                 5'd24: tile_addr <=
                                        0;
571                             endcase
572                         end
573                         if (brick2[0]) begin
574                             case (tile_x)
575                                 5'd25: tile_addr <=
                                        vcount - 32;
576                                 5'd26: tile_addr <=
                                        vcount - 16;
577                             endcase
578                         end else begin
579                             case (tile_x)
580                                 5'd25: tile_addr <=
                                        0;
581                                 5'd26: tile_addr <=
                                        0;
582                             endcase
583                         end
584                     end
585                     //Brick line 3
586                     5'd7 : begin
```

```
587                             if (brick3[12]) begin
588                                     case (tile_x)
589                                             5'd1 : tile_addr <=
                                                    vcount - 48;
590                                             5'd2 : tile_addr <=
                                                    vcount - 32;
591                                     endcase
592                             end else begin
593                                     case (tile_x)
594                                             5'd1 : tile_addr <=
                                                    0;
595                                             5'd2 : tile_addr <=
                                                    0;
596                                     endcase
597                             end
598                             if (brick3[11]) begin
599                                     case (tile_x)
600                                             5'd3 : tile_addr <=
                                                    vcount - 48;
601                                             5'd4 : tile_addr <=
                                                    vcount - 32;
602                                     endcase
603                             end else begin
604                                     case (tile_x)
605                                             5'd3 : tile_addr <=
                                                    0;
606                                             5'd4 : tile_addr <=
                                                    0;
607                                     endcase
608                             end
609                             if (brick3[10]) begin
610                                     case (tile_x)
611                                             5'd5 : tile_addr <=
                                                    vcount - 48;
612                                             5'd6 : tile_addr <=
                                                    vcount - 32;
613                                     endcase
614                             end else begin
615                                     case (tile_x)
616                                             5'd5 : tile_addr <=
                                                    0;
617                                             5'd6 : tile_addr <=
                                                    0;
618                                     endcase
619                             end
620                             if (brick3[9]) begin
621                                     case (tile_x)
622                                             5'd7 : tile_addr <=
                                                    vcount - 48;
623                                             5'd8 : tile_addr <=
                                                    vcount - 32;
624                                     endcase
625                             end else begin
626                                     case (tile_x)
627                                             5'd7 : tile_addr <=
                                                    0;
628                                             5'd8 : tile_addr <=
                                                    0;
629                                     endcase
```

```
630                         end
631                         if (brick3[8]) begin
632                                 case (tile_x)
633                                         5'd9 : tile_addr <=
                                                vcount - 48;
634                                         5'd10: tile_addr <=
                                                vcount - 32;
635                                 endcase
636                         end else begin
637                                 case (tile_x)
638                                         5'd9 : tile_addr <=
                                                0;
639                                         5'd10: tile_addr <=
                                                0;
640                                 endcase
641                         end
642                         if (brick3[7]) begin
643                                 case (tile_x)
644                                         5'd11: tile_addr <=
                                                vcount - 48;
645                                         5'd12: tile_addr <=
                                                vcount - 32;
646                                 endcase
647                         end else begin
648                                 case (tile_x)
649                                         5'd11: tile_addr <=
                                                0;
650                                         5'd12: tile_addr <=
                                                0;
651                                 endcase
652                         end
653                         if (brick3[6]) begin
654                                 case (tile_x)
655                                         5'd13: tile_addr <=
                                                vcount - 48;
656                                         5'd14: tile_addr <=
                                                vcount - 32;
657                                 endcase
658                         end else begin
659                                 case (tile_x)
660                                         5'd13: tile_addr <=
                                                0;
661                                         5'd14: tile_addr <=
                                                0;
662                                 endcase
663                         end
664                         if (brick3[5]) begin
665                                 case (tile_x)
666                                         5'd15: tile_addr <=
                                                vcount - 48;
667                                         5'd16: tile_addr <=
                                                vcount - 32;
668                                 endcase
669                         end else begin
670                                 case (tile_x)
671                                         5'd15: tile_addr <=
                                                0;
672                                         5'd16: tile_addr <=
                                                0;
```

```
673                     endcase
674                 end
675                 if (brick3[4]) begin
676                     case (tile_x)
677                         5'd17: tile_addr <=
                                vcount - 48;
678                         5'd18: tile_addr <=
                                vcount - 32;
679                     endcase
680                 end else begin
681                     case (tile_x)
682                         5'd17: tile_addr <=
                                0;
683                         5'd18: tile_addr <=
                                0;
684                     endcase
685                 end
686                 if (brick3[3]) begin
687                     case (tile_x)
688                         5'd19: tile_addr <=
                                vcount - 48;
689                         5'd20: tile_addr <=
                                vcount - 32;
690                     endcase
691                 end else begin
692                     case (tile_x)
693                         5'd19: tile_addr <=
                                0;
694                         5'd20: tile_addr <=
                                0;
695                     endcase
696                 end
697                 if (brick3[2]) begin
698                     case (tile_x)
699                         5'd21: tile_addr <=
                                vcount - 48;
700                         5'd22: tile_addr <=
                                vcount - 32;
701                     endcase
702                 end else begin
703                     case (tile_x)
704                         5'd21: tile_addr <=
                                0;
705                         5'd22: tile_addr <=
                                0;
706                     endcase
707                 end
708                 if (brick3[1]) begin
709                     case (tile_x)
710                         5'd23: tile_addr <=
                                vcount - 48;
711                         5'd24: tile_addr <=
                                vcount - 32;
712                     endcase
713                 end else begin
714                     case (tile_x)
715                         5'd23: tile_addr <=
                                0;
```

```
716                                     5'd24: tile_addr <=
                                            0;
717                             endcase
718                     end
719                     if (brick3[0]) begin
720                             case (tile_x)
721                                     5'd25: tile_addr <=
                                            vcount - 48;
722                                     5'd26: tile_addr <=
                                            vcount - 32;
723                             endcase
724                     end else begin
725                             case (tile_x)
726                                     5'd25: tile_addr <=
                                            0;
727                                     5'd26: tile_addr <=
                                            0;
728                             endcase
729                     end
730             end
731             //Brick line 4
732             5'd8 : begin
733                     if (brick4[12]) begin
734                             case (tile_x)
735                                     5'd1 : tile_addr <=
                                            vcount - 64;
736                                     5'd2 : tile_addr <=
                                            vcount - 48;
737                             endcase
738                     end else begin
739                             case (tile_x)
740                                     5'd1 : tile_addr <=
                                            0;
741                                     5'd2 : tile_addr <=
                                            0;
742                             endcase
743                     end
744                     if (brick4[11]) begin
745                             case (tile_x)
746                                     5'd3 : tile_addr <=
                                            vcount - 64;
747                                     5'd4 : tile_addr <=
                                            vcount - 48;
748                             endcase
749                     end else begin
750                             case (tile_x)
751                                     5'd3 : tile_addr <=
                                            0;
752                                     5'd4 : tile_addr <=
                                            0;
753                             endcase
754                     end
755                     if (brick4[10]) begin
756                             case (tile_x)
757                                     5'd5 : tile_addr <=
                                            vcount - 64;
758                                     5'd6 : tile_addr <=
                                            vcount - 48;
759                             endcase
```

```
760                         end else begin
761                                 case (tile_x)
762                                         5'd5 : tile_addr <=
                                                0;
763                                         5'd6 : tile_addr <=
                                                0;
764                                 endcase
765                         end
766                         if (brick4[9]) begin
767                                 case (tile_x)
768                                         5'd7 : tile_addr <=
                                                vcount - 64;
769                                         5'd8 : tile_addr <=
                                                vcount - 48;
770                                 endcase
771                         end else begin
772                                 case (tile_x)
773                                         5'd7 : tile_addr <=
                                                0;
774                                         5'd8 : tile_addr <=
                                                0;
775                                 endcase
776                         end
777                         if (brick4[8]) begin
778                                 case (tile_x)
779                                         5'd9 : tile_addr <=
                                                vcount - 64;
780                                         5'd10: tile_addr <=
                                                vcount - 48;
781                                 endcase
782                         end else begin
783                                 case (tile_x)
784                                         5'd9 : tile_addr <=
                                                0;
785                                         5'd10: tile_addr <=
                                                0;
786                                 endcase
787                         end
788                         if (brick4[7]) begin
789                                 case (tile_x)
790                                         5'd11: tile_addr <=
                                                vcount - 64;
791                                         5'd12: tile_addr <=
                                                vcount - 48;
792                                 endcase
793                         end else begin
794                                 case (tile_x)
795                                         5'd11: tile_addr <=
                                                0;
796                                         5'd12: tile_addr <=
                                                0;
797                                 endcase
798                         end
799                         if (brick4[6]) begin
800                                 case (tile_x)
801                                         5'd13: tile_addr <=
                                                vcount - 64;
802                                         5'd14: tile_addr <=
                                                vcount - 48;
```

```
803                             endcase
804                         end else begin
805                             case (tile_x)
806                                 5'd13: tile_addr <=
                                        0;
807                                 5'd14: tile_addr <=
                                        0;
808                             endcase
809                         end
810                         if (brick4[5]) begin
811                             case (tile_x)
812                                 5'd15: tile_addr <=
                                        vcount - 64;
813                                 5'd16: tile_addr <=
                                        vcount - 48;
814                             endcase
815                         end else begin
816                             case (tile_x)
817                                 5'd15: tile_addr <=
                                        0;
818                                 5'd16: tile_addr <=
                                        0;
819                             endcase
820                         end
821                         if (brick4[4]) begin
822                             case (tile_x)
823                                 5'd17: tile_addr <=
                                        vcount - 64;
824                                 5'd18: tile_addr <=
                                        vcount - 48;
825                             endcase
826                         end else begin
827                             case (tile_x)
828                                 5'd17: tile_addr <=
                                        0;
829                                 5'd18: tile_addr <=
                                        0;
830                             endcase
831                         end
832                         if (brick4[3]) begin
833                             case (tile_x)
834                                 5'd19: tile_addr <=
                                        vcount - 64;
835                                 5'd20: tile_addr <=
                                        vcount - 48;
836                             endcase
837                         end else begin
838                             case (tile_x)
839                                 5'd19: tile_addr <=
                                        0;
840                                 5'd20: tile_addr <=
                                        0;
841                             endcase
842                         end
843                         if (brick4[2]) begin
844                             case (tile_x)
845                                 5'd21: tile_addr <=
                                        vcount - 64;
```

```
846                                         5'd22: tile_addr <=
                                                vcount - 48;
847                                 endcase
848                             end else begin
849                                 case (tile_x)
850                                         5'd21: tile_addr <=
                                                0;
851                                         5'd22: tile_addr <=
                                                0;
852                                 endcase
853                             end
854                             if (brick4[1]) begin
855                                 case (tile_x)
856                                         5'd23: tile_addr <=
                                                vcount - 64;
857                                         5'd24: tile_addr <=
                                                vcount - 48;
858                                 endcase
859                             end else begin
860                                 case (tile_x)
861                                         5'd23: tile_addr <=
                                                0;
862                                         5'd24: tile_addr <=
                                                0;
863                                 endcase
864                             end
865                             if (brick4[0]) begin
866                                 case (tile_x)
867                                         5'd25: tile_addr <=
                                                vcount - 64;
868                                         5'd26: tile_addr <=
                                                vcount - 48;
869                                 endcase
870                             end else begin
871                                 case (tile_x)
872                                         5'd25: tile_addr <=
                                                0;
873                                         5'd26: tile_addr <=
                                                0;
874                                 endcase
875                             end
876                         end
877                         //Brick line 5
878                         5'd9 : begin
879                             if (brick5[12]) begin
880                                 case (tile_x)
881                                         5'd1 : tile_addr <=
                                                vcount - 80;
882                                         5'd2 : tile_addr <=
                                                vcount - 64;
883                                 endcase
884                             end else begin
885                                 case (tile_x)
886                                         5'd1 : tile_addr <=
                                                0;
887                                         5'd2 : tile_addr <=
                                                0;
888                                 endcase
889                             end
```

```
890                         if (brick5[11]) begin
891                                 case (tile_x)
892                                         5'd3 : tile_addr <=
                                                    vcount - 80;
893                                         5'd4 : tile_addr <=
                                                    vcount - 64;
894                                 endcase
895                         end else begin
896                                 case (tile_x)
897                                         5'd3 : tile_addr <=
                                                    0;
898                                         5'd4 : tile_addr <=
                                                    0;
899                                 endcase
900                         end
901                         if (brick5[10]) begin
902                                 case (tile_x)
903                                         5'd5 : tile_addr <=
                                                    vcount - 80;
904                                         5'd6 : tile_addr <=
                                                    vcount - 64;
905                                 endcase
906                         end else begin
907                                 case (tile_x)
908                                         5'd5 : tile_addr <=
                                                    0;
909                                         5'd6 : tile_addr <=
                                                    0;
910                                 endcase
911                         end
912                         if (brick5[9]) begin
913                                 case (tile_x)
914                                         5'd7 : tile_addr <=
                                                    vcount - 80;
915                                         5'd8 : tile_addr <=
                                                    vcount - 64;
916                                 endcase
917                         end else begin
918                                 case (tile_x)
919                                         5'd7 : tile_addr <=
                                                    0;
920                                         5'd8 : tile_addr <=
                                                    0;
921                                 endcase
922                         end
923                         if (brick5[8]) begin
924                                 case (tile_x)
925                                         5'd9 : tile_addr <=
                                                    vcount - 80;
926                                         5'd10: tile_addr <=
                                                    vcount - 64;
927                                 endcase
928                         end else begin
929                                 case (tile_x)
930                                         5'd9 : tile_addr <=
                                                    0;
931                                         5'd10: tile_addr <=
                                                    0;
932                                 endcase
```

```
933                             end
934                             if (brick5[7]) begin
935                                     case (tile_x)
936                                             5'd11: tile_addr <=
                                                    vcount - 80;
937                                             5'd12: tile_addr <=
                                                    vcount - 64;
938                                     endcase
939                             end else begin
940                                     case (tile_x)
941                                             5'd11: tile_addr <=
                                                    0;
942                                             5'd12: tile_addr <=
                                                    0;
943                                     endcase
944                             end
945                             if (brick5[6]) begin
946                                     case (tile_x)
947                                             5'd13: tile_addr <=
                                                    vcount - 80;
948                                             5'd14: tile_addr <=
                                                    vcount - 64;
949                                     endcase
950                             end else begin
951                                     case (tile_x)
952                                             5'd13: tile_addr <=
                                                    0;
953                                             5'd14: tile_addr <=
                                                    0;
954                                     endcase
955                             end
956                             if (brick5[5]) begin
957                                     case (tile_x)
958                                             5'd15: tile_addr <=
                                                    vcount - 80;
959                                             5'd16: tile_addr <=
                                                    vcount - 64;
960                                     endcase
961                             end else begin
962                                     case (tile_x)
963                                             5'd15: tile_addr <=
                                                    0;
964                                             5'd16: tile_addr <=
                                                    0;
965                                     endcase
966                             end
967                             if (brick5[4]) begin
968                                     case (tile_x)
969                                             5'd17: tile_addr <=
                                                    vcount - 80;
970                                             5'd18: tile_addr <=
                                                    vcount - 64;
971                                     endcase
972                             end else begin
973                                     case (tile_x)
974                                             5'd17: tile_addr <=
                                                    0;
975                                             5'd18: tile_addr <=
                                                    0;
```

```
976                         endcase
977                     end
978                     if (brick5[3]) begin
979                         case (tile_x)
980                             5'd19: tile_addr <=
                                    vcount - 80;
981                             5'd20: tile_addr <=
                                    vcount - 64;
982                         endcase
983                     end else begin
984                         case (tile_x)
985                             5'd19: tile_addr <=
                                    0;
986                             5'd20: tile_addr <=
                                    0;
987                         endcase
988                     end
989                     if (brick5[2]) begin
990                         case (tile_x)
991                             5'd21: tile_addr <=
                                    vcount - 80;
992                             5'd22: tile_addr <=
                                    vcount - 64;
993                         endcase
994                     end else begin
995                         case (tile_x)
996                             5'd21: tile_addr <=
                                    0;
997                             5'd22: tile_addr <=
                                    0;
998                         endcase
999                     end
1000                    if (brick5[1]) begin
1001                        case (tile_x)
1002                            5'd23: tile_addr <=
                                   vcount - 80;
1003                            5'd24: tile_addr <=
                                   vcount - 64;
1004                        endcase
1005                    end else begin
1006                        case (tile_x)
1007                            5'd23: tile_addr <=
                                   0;
1008                            5'd24: tile_addr <=
                                   0;
1009                        endcase
1010                    end
1011                    if (brick5[0]) begin
1012                        case (tile_x)
1013                            5'd25: tile_addr <=
                                   vcount - 80;
1014                            5'd26: tile_addr <=
                                   vcount - 64;
1015                        endcase
1016                    end else begin
1017                        case (tile_x)
1018                            5'd25: tile_addr <=
                                   0;
```

```
1019                                        5'd26: tile_addr <=
                                                0;
1020                                endcase
1021                        end
1022                end
1023                //Brick line 6
1024                5'd10: begin
1025                        if (brick6[12]) begin
1026                                case (tile_x)
1027                                        5'd1 : tile_addr <=
                                                vcount - 96;
1028                                        5'd2 : tile_addr <=
                                                vcount - 80;
1029                                endcase
1030                        end else begin
1031                                case (tile_x)
1032                                        5'd1 : tile_addr <=
                                                0;
1033                                        5'd2 : tile_addr <=
                                                0;
1034                                endcase
1035                        end
1036                        if (brick6[11]) begin
1037                                case (tile_x)
1038                                        5'd3 : tile_addr <=
                                                vcount - 96;
1039                                        5'd4 : tile_addr <=
                                                vcount - 80;
1040                                endcase
1041                        end else begin
1042                                case (tile_x)
1043                                        5'd3 : tile_addr <=
                                                0;
1044                                        5'd4 : tile_addr <=
                                                0;
1045                                endcase
1046                        end
1047                        if (brick6[10]) begin
1048                                case (tile_x)
1049                                        5'd5 : tile_addr <=
                                                vcount - 96;
1050                                        5'd6 : tile_addr <=
                                                vcount - 80;
1051                                endcase
1052                        end else begin
1053                                case (tile_x)
1054                                        5'd5 : tile_addr <=
                                                0;
1055                                        5'd6 : tile_addr <=
                                                0;
1056                                endcase
1057                        end
1058                        if (brick6[9]) begin
1059                                case (tile_x)
1060                                        5'd7 : tile_addr <=
                                                vcount - 96;
1061                                        5'd8 : tile_addr <=
                                                vcount - 80;
1062                                endcase
```

```
1063                        end else begin
1064                                case (tile_x)
1065                                        5'd7 : tile_addr <=
                                                    0;
1066                                        5'd8 : tile_addr <=
                                                    0;
1067                                endcase
1068                        end
1069                        if (brick6[8]) begin
1070                                case (tile_x)
1071                                        5'd9 : tile_addr <=
                                                    vcount - 96;
1072                                        5'd10: tile_addr <=
                                                    vcount - 80;
1073                                endcase
1074                        end else begin
1075                                case (tile_x)
1076                                        5'd9 : tile_addr <=
                                                    0;
1077                                        5'd10: tile_addr <=
                                                    0;
1078                                endcase
1079                        end
1080                        if (brick6[7]) begin
1081                                case (tile_x)
1082                                        5'd11: tile_addr <=
                                                    vcount - 96;
1083                                        5'd12: tile_addr <=
                                                    vcount - 80;
1084                                endcase
1085                        end else begin
1086                                case (tile_x)
1087                                        5'd11: tile_addr <=
                                                    0;
1088                                        5'd12: tile_addr <=
                                                    0;
1089                                endcase
1090                        end
1091                        if (brick6[6]) begin
1092                                case (tile_x)
1093                                        5'd13: tile_addr <=
                                                    vcount - 96;
1094                                        5'd14: tile_addr <=
                                                    vcount - 80;
1095                                endcase
1096                        end else begin
1097                                case (tile_x)
1098                                        5'd13: tile_addr <=
                                                    0;
1099                                        5'd14: tile_addr <=
                                                    0;
1100                                endcase
1101                        end
1102                        if (brick6[5]) begin
1103                                case (tile_x)
1104                                        5'd15: tile_addr <=
                                                    vcount - 96;
1105                                        5'd16: tile_addr <=
                                                    vcount - 80;
```

```
1106                    endcase
1107                end else begin
1108                    case (tile_x)
1109                        5'd15: tile_addr <= 0;
1110                        5'd16: tile_addr <= 0;
1111                    endcase
1112                end
1113                if (brick6[4]) begin
1114                    case (tile_x)
1115                        5'd17: tile_addr <= vcount - 96;
1116                        5'd18: tile_addr <= vcount - 80;
1117                    endcase
1118                end else begin
1119                    case (tile_x)
1120                        5'd17: tile_addr <= 0;
1121                        5'd18: tile_addr <= 0;
1122                    endcase
1123                end
1124                if (brick6[3]) begin
1125                    case (tile_x)
1126                        5'd19: tile_addr <= vcount - 96;
1127                        5'd20: tile_addr <= vcount - 80;
1128                    endcase
1129                end else begin
1130                    case (tile_x)
1131                        5'd19: tile_addr <= 0;
1132                        5'd20: tile_addr <= 0;
1133                    endcase
1134                end
1135                if (brick6[2]) begin
1136                    case (tile_x)
1137                        5'd21: tile_addr <= vcount - 96;
1138                        5'd22: tile_addr <= vcount - 80;
1139                    endcase
1140                end else begin
1141                    case (tile_x)
1142                        5'd21: tile_addr <= 0;
1143                        5'd22: tile_addr <= 0;
1144                    endcase
1145                end
1146                if (brick6[1]) begin
1147                    case (tile_x)
1148                        5'd23: tile_addr <= vcount - 96;
```

```
1149                                        5'd24: tile_addr <=
                                                vcount - 80;
1150                                endcase
1151                            end else begin
1152                                case (tile_x)
1153                                        5'd23: tile_addr <=
                                                0;
1154                                        5'd24: tile_addr <=
                                                0;
1155                                endcase
1156                            end
1157                            if (brick6[0]) begin
1158                                case (tile_x)
1159                                        5'd25: tile_addr <=
                                                vcount - 96;
1160                                        5'd26: tile_addr <=
                                                vcount - 80;
1161                                endcase
1162                            end else begin
1163                                case (tile_x)
1164                                        5'd25: tile_addr <=
                                                0;
1165                                        5'd26: tile_addr <=
                                                0;
1166                                endcase
1167                            end
1168                    end
1169                    endcase
1170                    end
1171            end
1172
1173            // ----------------ROM colormap_ROM.v-------------
1174            // Assign specific color to each pixel
1175            assign addr_adj = tile_output >> (30 - ({hcount
                    [10:1],1'b0} - (192 + (tile_x * 32))));
1176            logic [4:0] color_addr;
1177            logic [23:0] color_output;
1178            logic [31:0] addr_adj;
1179            colormap_ROM color1(.address(color_addr),.clock(clk
                    ),.q(color_output));
1180
1181            always_ff @(posedge clk) begin
1182        //------Static elements------
1183                    if ((tile_x <= 27 && tile_y == 2)|| //
                            Corners + Top
1184                 ((tile_x == 0 || tile_x == 27) && tile_y >= 3)
                            //Side
1185                            )
1186                                color_addr <= {3'b0,addr_adj[1:0]};
1187            //SCORE & Stage & Win/Lose
1188                    else if ((tile_x <= 16 && tile_x >= 12 &&
                            tile_y == 0)||
1189                    (tile_x <= 16 && tile_x >= 13 && tile_y ==
                            1)||
1190                    (tile_x <= 27 && tile_x >= 23 && tile_y ==
                            0)||
1191                    (tile_x == 27 && tile_y == 1)||
1192                    ((tile_x == 1 || tile_x == 2) && tile_y ==
                            29)|| //HP
```

```
1193                    (tile_y == 15 && tile_x <= 18 && tile_x >=
                            10) // Win or Lose
1194                    )
1195                            color_addr <= addr_adj[1:0] + 4;
1196        //------Dynamic elements------
1197            //Brick lines
1198                    else if (tile_x >= 1 && tile_x <= 26) begin
1199                            case (tile_y)
1200                                    5'd5 : color_addr <=
                                            addr_adj[1:0] + 8 + (4*
                                            brick1[15:13]);
1201                                    5'd6 : color_addr <=
                                            addr_adj[1:0] + 8 + (4*
                                            brick2[15:13]);
1202                                    5'd7 : color_addr <=
                                            addr_adj[1:0] + 8 + (4*
                                            brick3[15:13]);
1203                                    5'd8 : color_addr <=
                                            addr_adj[1:0] + 8 + (4*
                                            brick4[15:13]);
1204                                    5'd9 : color_addr <=
                                            addr_adj[1:0] + 8 + (4*
                                            brick5[15:13]);
1205                                    5'd10: color_addr <=
                                            addr_adj[1:0] + 8 + (4*
                                            brick6[15:13]);
1206                            endcase
1207                    end
1208            end
1209
1210            //-------Display-------
1211            // Put everything on screen
1212        always_comb begin
1213          {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
1214          if (VGA_BLANK_n)
1215                if (circle) //Ball
1216                    {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'
                            hff};
1217            else if (peddle) //Pad
1218                    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'hff, 8'hff
                            };
1219                else if (waste) //Gray needless area
1220                    {VGA_R, VGA_G, VGA_B} = {8'h69, 8'h69, 8'
                            h69};
1221            else if ((tile_x <= 27 && tile_y == 2)|| //Corners
                    + Top
1222                    ((tile_x == 0 || tile_x == 27)&& tile_y >=
                            3)|| //Side
1223                    (tile_x <= 16 && tile_x >= 12 && tile_y ==
                            0)|| //SCORE
1224                    (tile_x <= 16 && tile_x >= 13 && tile_y ==
                            1)|| //Score Number
1225                                    (tile_x <= 27 && tile_x >= 23
                                            && tile_y == 0)|| //STAGE
1226                            (tile_x == 27 && tile_y == 1)|| //
                                    Stage Number
1227                                    ((tile_x == 1 || tile_x ==
                                            2) && tile_y == 29)|| //
                                            HP Indicator
```

```
1228                                    (tile_y == 15 && tile_x <=
                                            18 && tile_x >= 10)|| //
                                            Win or Lose
1229                                (tile_x >= 1 && tile_x <= 26 &&
                                        tile_y >= 5 && tile_y <=
                                        10) //Bricks
1230                                )
1231                            {VGA_R, VGA_G, VGA_B} =
                                    color_output;
1232          else //Background
1233                  {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
1234
1235      end
1236
1237  endmodule
1238
1239  module vga_counters(
1240   input logic          clk50, reset,
1241   output logic [10:0] hcount,  // hcount[10:1] is pixel
         column
1242   output logic [9:0]  vcount,  // vcount[9:0] is pixel row
1243   output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
         VGA_SYNC_n);
1244
1245  /*
1246   * 640 X 480 VGA timing for a 50 MHz clock: one pixel every
         other cycle
1247   *
1248   * HCOUNT 1599 0              1279       1599 0
1249   *             ---------------              --------
1250   * ------------|     Video       |------------|  Video
1251   *
1252   *
1253   * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
1254   *       -----------------------     -------------
1255   * |____|       VGA_HS          |____|
1256   */
1257     // Parameters for hcount
1258     parameter HACTIVE      = 11'd 1280,
1259               HFRONT_PORCH = 11'd 32,
1260               HSYNC        = 11'd 192,
1261               HBACK_PORCH  = 11'd 96,
1262               HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC
                      +
1263                              HBACK_PORCH; // 1600
1264
1265     // Parameters for vcount
1266     parameter VACTIVE      = 10'd 480,
1267               VFRONT_PORCH = 10'd 10,
1268               VSYNC        = 10'd 2,
1269               VBACK_PORCH  = 10'd 33,
1270               VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC
                      +
1271                              VBACK_PORCH; // 525
1272
1273     logic endOfLine;
1274
1275     always_ff @(posedge clk50 or posedge reset)
1276       if (reset)          hcount <= 0;
```

```
1277        else if (endOfLine) hcount <= 0;
1278        else                hcount <= hcount + 11'd 1;
1279
1280      assign endOfLine = hcount == HTOTAL - 1;
1281
1282      logic endOfField;
1283
1284      always_ff @(posedge clk50 or posedge reset)
1285        if (reset)          vcount <= 0;
1286        else if (endOfLine)
1287          if (endOfField)   vcount <= 0;
1288          else              vcount <= vcount + 10'd 1;
1289
1290      assign endOfField = vcount == VTOTAL - 1;
1291
1292      // Horizontal sync: from 0x520 to 0x5DF (0x57F)
1293      // 101 0010 0000 to 101 1101 1111
1294      assign VGA_HS = !( (hcount[10:8] == 3'b101) &
1295                         !(hcount[7:5] == 3'b111));
1296      assign VGA_VS = !( vcount[9:1] == (VACTIVE +
             VFRONT_PORCH) / 2);
1297
1298      assign VGA_SYNC_n = 1'b0; // For putting sync on the
             green signal; unused
1299
1300      // Horizontal active: 0 to 1279     Vertical active: 0
             to 479
1301      // 101 0000 0000  1280              01 1110 0000  480
1302      // 110 0011 1111  1599              10 0000 1100  524
1303      assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount
             [8]) ) &
1304                           !( vcount[9] | (vcount[8:5] == 4'
                                  b1111) );
1305
1306      /* VGA_CLK is 25 MHz
1307       *             __    __    __
1308       * clk50    __|  |__|  |__|
1309       *
1310       *             _____       __
1311       * hcount[0]__|     |_____|
1312       */
1313      assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge
             sensitive
1314
1315  endmodule
```

B. Code for software

```
1
2   /*****************************************************/ /*
        ***********************vga_ball.h*********************** */
         /*****************************************************/
3
4   #ifndef _VGA_BALL_H
5   #define _VGA_BALL_H
6
7   #include <linux/ioctl.h>
```

```
8
9   typedef struct {
10          unsigned short x_ball, y_ball, x_pad, score, brick1,
                brick2, brick3, brick4, brick5, brick6, sound_effect,
                game_status;
11  } hardware_data;
12
13
14  typedef struct {
15    hardware_data data;
16  } vga_ball_arg_t;
17
18  #define VGA_BALL_MAGIC 'q'
19
20  /* ioctls and their arguments */
21  #define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1,
        vga_ball_arg_t *)
22  #define VGA_BALL_READ_BACKGROUND  _IOR(VGA_BALL_MAGIC, 2,
        vga_ball_arg_t *)
23
24  #endif
25
26
27
28  /*****************************************************/ /*
        ********************vga_ball.c*********************** */
         /*****************************************************/
29
30  /* * Device driver for the VGA video generator
31   *
32   * A Platform device implemented using the misc subsystem
33   *
34   * Stephen A. Edwards
35   * Columbia University
36   *
37   * References:
38   * Linux source: Documentation/driver-model/platform.txt
39   *               drivers/misc/arm-charlcd.c
40   * http://www.linuxforu.com/tag/linux-device-drivers/
41   * http://free-electrons.com/docs/
42   *
43   * "make" to build
44   * insmod vga_ball.ko
45   *
46   * Check code style with
47   * checkpatch.pl --file --no-tree vga_ball.c
48   */
49
50  #include <linux/module.h>
51  #include <linux/init.h>
52  #include <linux/errno.h>
53  #include <linux/version.h>
54  #include <linux/kernel.h>
55  #include <linux/platform_device.h>
56  #include <linux/miscdevice.h>
57  #include <linux/slab.h>
58  #include <linux/io.h>
59  #include <linux/of.h>
60  #include <linux/of_address.h>
```

```
61  #include <linux/fs.h>
62  #include <linux/uaccess.h>
63  #include "vga_ball.h"
64
65  #define DRIVER_NAME "vga_ball"
66
67  /* Device registers */
68  #define X_BALL(x) ((x) + 0)
69  #define Y_BALL(x) ((x)+ 2)
70  #define X_PAD(x) ((x) + 4)
71  #define SCORE(x) ((x) + 6)
72
73  #define BRICK1(x) ((x) + 8)
74  #define BRICK2(x) ((x) + 10)
75  #define BRICK3(x) ((x) + 12)
76  #define BRICK4(x) ((x) + 14)
77  #define BRICK5(x) ((x) + 16)
78  #define BRICK6(x) ((x) + 18)
79  #define SOUND_EFFECT(x) ((x) + 20)
80  #define GAME_STATUS(x) ((x) + 22)
81
82  /*
83   * Information about our device
84   */
85  struct vga_ball_dev {
86          struct resource res; /* Resource: our registers */
87          void __iomem *virtbase; /* Where registers can be
                accessed in memory */
88          hardware_data data;
89  } dev;
90
91  /*
92   * Write segments of a single digit
93   * Assumes digit is in range and the device information has
        been set up
94   */
95  static void write_background(hardware_data *data)
96  {
97    //     iowrite16(data->red, BG_RED(dev.virtbase) );
98          iowrite16(data->x_ball, X_BALL(dev.virtbase));
99          iowrite16(data->y_ball, Y_BALL(dev.virtbase));
100         iowrite16(data->x_pad, X_PAD(dev.virtbase));
101         iowrite16(data->score, SCORE(dev.virtbase));
102
103         iowrite16(data->brick1, BRICK1(dev.virtbase));
104         iowrite16(data->brick2, BRICK2(dev.virtbase));
105         iowrite16(data->brick3, BRICK3(dev.virtbase));
106         iowrite16(data->brick4, BRICK4(dev.virtbase));
107         iowrite16(data->brick5, BRICK5(dev.virtbase));
108         iowrite16(data->brick6, BRICK6(dev.virtbase));
109
110
111         iowrite16(data->sound_effect, SOUND_EFFECT(dev.
                virtbase));
112         iowrite16(data->game_status, GAME_STATUS(dev.
                virtbase));
113         dev.data = *data;
114 }
115
```

```
116 /*
117  * Handle ioctl() calls from userspace:
118  * Read or write the segments on single digits.
119  * Note extensive error checking of arguments
120  */
121 static long vga_ball_ioctl(struct file *f, unsigned int cmd
        , unsigned long arg)
122 {
123         vga_ball_arg_t vla;
124
125         switch (cmd) {
126         case VGA_BALL_WRITE_BACKGROUND:
127                 if (copy_from_user(&vla, (vga_ball_arg_t *)
                        arg,
128                                    sizeof(vga_ball_arg_t)))
129                         return -EACCES;
130                 write_background(&vla.data);
131                 break;
132
133         case VGA_BALL_READ_BACKGROUND:
134                 vla.data = dev.data;
135                 if (copy_to_user((vga_ball_arg_t *) arg, &
                        vla,
136                                  sizeof(vga_ball_arg_t)))
137                         return -EACCES;
138                 break;
139
140         default:
141                 return -EINVAL;
142         }
143
144         return 0;
145 }
146
147 /* The operations our device knows how to do */
148 static const struct file_operations vga_ball_fops = {
149         .owner          = THIS_MODULE,
150         .unlocked_ioctl = vga_ball_ioctl,
151 };
152
153 /* Information about our device for the "misc" framework --
        like a char dev */
154 static struct miscdevice vga_ball_misc_device = {
155         .minor          = MISC_DYNAMIC_MINOR,
156         .name           = DRIVER_NAME,
157         .fops           = &vga_ball_fops,
158 };
159
160 /*
161  * Initialization code: get resources (registers) and
        display
162  * a welcome message
163  */
164 static int __init vga_ball_probe(struct platform_device *
        pdev)
165 {
166     hardware_data beige = { 0xf9, 0xe4, 0xb7, 0xb7,0xb7};
167         int ret;
168
```

```
169         /* Register ourselves as a misc device: creates /
                dev/vga_ball */
170         ret = misc_register(&vga_ball_misc_device);
171
172         /* Get the address of our registers from the device
                tree */
173         ret = of_address_to_resource(pdev->dev.of_node, 0,
                &dev.res);
174         if (ret) {
175                 ret = -ENOENT;
176                 goto out_deregister;
177         }
178
179         /* Make sure we can use these registers */
180         if (request_mem_region(dev.res.start, resource_size
                (&dev.res),
181                                DRIVER_NAME) == NULL) {
182                 ret = -EBUSY;
183                 goto out_deregister;
184         }
185
186         /* Arrange access to our registers */
187         dev.virtbase = of_iomap(pdev->dev.of_node, 0);
188         if (dev.virtbase == NULL) {
189                 ret = -ENOMEM;
190                 goto out_release_mem_region;
191         }
192
193         /* Set an initial color */
194         write_background(&beige);
195
196         return 0;
197
198 out_release_mem_region:
199         release_mem_region(dev.res.start, resource_size(&
                dev.res));
200 out_deregister:
201         misc_deregister(&vga_ball_misc_device);
202         return ret;
203 }
204
205 /* Clean-up code: release resources */
206 static int vga_ball_remove(struct platform_device *pdev)
207 {
208         iounmap(dev.virtbase);
209         release_mem_region(dev.res.start, resource_size(&
                dev.res));
210         misc_deregister(&vga_ball_misc_device);
211         return 0;
212 }
213
214 /* Which "compatible" string(s) to search for in the Device
        Tree */
215 #ifdef CONFIG_OF
216 static const struct of_device_id vga_ball_of_match[] = {
217         { .compatible = "csee4840,vga_ball-1.0" },
218         {},
219 };
220 MODULE_DEVICE_TABLE(of, vga_ball_of_match);
```

```
221 #endif
222
223 /* Information for registering ourselves as a "platform"
        driver */
224 static struct platform_driver vga_ball_driver = {
225         .driver = {
226                 .name   = DRIVER_NAME,
227                 .owner  = THIS_MODULE,
228                 .of_match_table = of_match_ptr(
                        vga_ball_of_match),
229         },
230         .remove = __exit_p(vga_ball_remove),
231 };
232
233 /* Called when the module is loaded: set things up */
234 static int __init vga_ball_init(void)
235 {
236         pr_info(DRIVER_NAME ": init\n");
237         return platform_driver_probe(&vga_ball_driver,
                vga_ball_probe);
238 }
239
240 /* Calball when the module is unloaded: release resources
        */
241 static void __exit vga_ball_exit(void)
242 {
243         platform_driver_unregister(&vga_ball_driver);
244         pr_info(DRIVER_NAME ": exit\n");
245 }
246
247 module_init(vga_ball_init);
248 module_exit(vga_ball_exit);
249
250 MODULE_LICENSE("GPL");
251 MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
252 MODULE_DESCRIPTION("VGA ball driver");
253
254
255
256 /*****************************************************/ /*
        ********************hello.c*********************/ /*
        *****************************************************/
257
258 /*
259  * Userspace program that communicates with the vga_ball
        device driver
260  * through ioctls
261  *
262  * Stephen A. Edwards
263  * Columbia University
264
265  * Software Part of Breakout Game Remastered Project
266  * CSEE 4840, Spring 2022, Columbia University
267  */
268
269
270 #include <stdio.h>
271 #include "vga_ball.h"
272 #include <sys/ioctl.h>
```

```
273 #include <sys/types.h>
274 #include <sys/stat.h>
275 #include <fcntl.h>
276 #include <string.h>
277 #include <unistd.h>
278 #include <stdlib.h>
279 #include <pthread.h>
280 #include <math.h>
281 #include <time.h>
282 #include "usbkeyboard.h"
283 #define BRICK_W 30
284 #define BRICK_H 15
285 #define BALL_R 10
286 #define PADDLE_W 90
287 #define PADDLE_H 20
288 #define PADDLE_R 10
289 #define PADDLE_L 70
290
291 #define SOUND_DEFULT 0
292 #define SOUND_HIT_BRICK 1
293 #define SOUND_WALL_PAD 2
294 #define SOUND_GAME_OVER 3
295
296
297 double h_location;
298 //double speed_paddle, ball_h = 312.0, ball_v = 440.0,
        speed_h, speed_v;
299 int brick_status[7][10];
300 int reset = 1;
301 int lives = 3;
302 int game_start = 0;
303 int finalstatus = 0;
304 //int size;
305 int vga_ball_fd;
306 hardware_data data = {0, 0, 0, 0, 0}; // TODO
307 struct libusb_device_handle *mouse;    // a mouse device
        handle
308 pthread_t mouse_thread;
309 void *mouse_thread_f(void *);
310 int ball_h = 195;
311 int ball_v = 425;
312 int sound_effect = 0;
313 int gloabl_score = 0;
314 int game_stage = 0; // 1, 2
315 int game_hp = 3;
316 int game_over = 0; // 0 default 1 game over 2 win
317 int game_over_sound = 1;
318 int re_start = 1;
319 int easy_mode = 0;
320
321
322
323 /* Read and print the background color */
324 void print_background_color()
325 {
326   vga_ball_arg_t vla;
327
328   if (ioctl(vga_ball_fd, VGA_BALL_READ_BACKGROUND, &vla))
329   {
```

```
330       perror("ioctl(VGA_BALL_READ_BACKGROUND) failed");
331       return;
332     }
333     // printf("%02x %02x %02x\n", vla.data.red, vla.data.
          green, vla.data.blue);
334   }
335
336   /* Set the background color */
337   void set_background_color(const hardware_data *c)
338   {
339     vga_ball_arg_t vla;
340     vla.data = *c;
341     if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla))
342     {
343       perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
344       return;
345     }
346   }
347
348   /* The 4  ways of hit */
349   /* return flag = 0 if not hit on the brick */
350   /* return flag = 1 if hit on the brick from top */
351   /* return flag = 2 if hit on the brick from bottom */
352   /* return flag = 3 if hit on the brick from left  */
353   /* return flag = 4 if hit on the brick from right */
354   /* use the flag to determine the ball velocity change */
355   /* the corresponding brick status should be changed into 0
        */
356   int hitOnBrick(int ball_v, int ball_h, int brick_width, int
        brick_height, int brick_h, int brick_v)
357   {
358     int flag;
359     // h:horizontal; v:vertical
360     // hit on top: flag= 1
361     if (ball_h >= brick_h && ball_h <= brick_h + brick_width
          && ball_v >= brick_v - 5 && ball_v <=brick_v /*ball_h-
           5  >= brick_h && ball_h+5<= brick_h + brick_width &&
          ball_v + 5 >    brick_v && ball_v + 5 <=brick_v +
          brick_height*/)
362       {
363         flag = 1;
364       }
365     // hit on bottom: flag= 2
366     else if (ball_h >= brick_h && ball_h <= brick_h +
          brick_width && ball_v <= brick_v + brick_height + 5 &&
           ball_v >= brick_v + brick_height /*ball_h - 5 >=
          brick_h && ball_h + 5 <= brick_h + brick_width &&
          ball_v <= brick_v + brick_height + 5 && ball_v - 5 >
          brick_v*/)
367       {
368         flag = 2;
369       }
370     // hit from left: flag= 3
371     else if (ball_v >= brick_v && ball_v <= brick_v +
          brick_height && ball_h >= brick_h - 5 && ball_h <=
          brick_h /* ball_h + 5 >= brick_h && ball_h < brick_h +
           brick_width - 5*/)
372       {
373         flag = 3;
```

```
374       }
375     // hit from right: flag= 4
376     else if (ball_v >= brick_v && ball_v <= brick_v +
            brick_height && ball_h <= brick_h + brick_width + 5 &&
             ball_h >= brick_h + brick_width /*ball_h - 5 <=
            brick_h + brick_width && ball_h > brick_h + 5*/)
377     {
378       flag = 4;
379     }
380     // doesn't hit on the brick: flag= 0
381     else
382     {
383       flag = 0;
384     }
385
386     return flag;
387   }
388
389
390   // conv game status
391   long get_game_status(int game_stage, int game_hp, int
        game_over) {
392     int x[16];
393     for (int j = 0; j < 16; j++) {
394       x[j] = 0;
395     }
396
397     x[15] = game_stage;
398     if (game_hp == 3) {
399       x[14] = 1;
400       x[13] = 1;
401     } else if (game_hp == 2) {
402       x[13] = 1;
403     }
404
405     if (game_over == 1) {
406       x[12] = 1;
407     } else if (game_over == 2) {
408       x[12] = 1;
409       x[11] = 1;
410     }
411
412
413     long y = 0;
414     for (int i = 0; i < 16; i++){
415       y = y * 2 + x[i];
416     }
417
418     //printf("get_game_status %d", y);
419     return(y);
420   }
421
422
423
424   // vec to binary
425   long long convert2bin( int x[13], int color) {
426     long long  y = color; // 000 001 010 011 100 101
427
428     // 1.color
```

```
429     for (int i = 0; i < 13; i++){
430       y = y * 2 + x[i];
431     }
432     return(y);
433   }
434
435   // vec to binary
436   long hex2hexadecimal(int score) {
437     int x[16];
438
439     for (int j = 0; j < 16; j++) {
440       x[j] = 0;
441     }
442
443     int idx = 15; // 12-15
444
445     //printf("original score %d\n",score);
446     while (score > 0) {
447       int num = score % 10;
448
449       int cur_idx = idx;
450       while (num > 0) {
451         x[cur_idx] = num % 2;
452         num = (int)(num/2);
453         cur_idx -= 1;
454       }
455       score = (int)(score / 10);
456       idx -= 4;
457     }
458
459     long y = 0;
460     for (int i = 0; i < 16; i++){
461       y = y * 2 + x[i];
462     }
463
464     //printf("score %d, %lld \n", score, y);
465     return(y);
466     //vec 2 10
467   }
468
469
470
471   int check_clear(int matrix[6][13],int x, int y){
472     for(int i = 0; i < x; i++){
473       for (int j = 0; j < y; j++) {
474         if (matrix[i][j] != 0){
475                   return 0;
476         }
477       }
478     }
479     return 1;
480   }
481
482
483
484   int main()
485   {
486     vga_ball_arg_t vla;
487     int i;
```

```
488     int j;
489
490     static const char filename[] = "/dev/vga_ball";
491
492     printf("VGA ball Userspace program started\n");
493
494     if ((vga_ball_fd = open(filename, O_RDWR)) == -1)
495     {
496       fprintf(stderr, "could not open %s\n", filename);
497       return -1;
498     }
499
500     printf("initial state: ");
501     print_background_color();
502
503     // input from device
504     libusb_context *ctx = NULL; // a libusb session
505     libusb_device **devs;       // pointer to pointer of
          device, used to retrieve a list of devices
506     int r;                      // for return values
507     ssize_t cnt;                // holding number of devices
          in list
508     r = libusb_init(&ctx);      // initialize a library
          session
509     if (r < 0)
510     {
511       printf("%s  %d\n", "Init Error", r); // there was an
            error
512       return 1;
513     }
514     libusb_set_debug(ctx, 3);                  // set
          verbosity level to 3, as suggested in the
          documentation
515     cnt = libusb_get_device_list(ctx, &devs); // get the list
           of devices
516     if (cnt < 0)
517     {
518       printf("%s\n", "Get Device Error"); // there was an
            error
519     }
520
521     printf("\n11111\n");
522     // mouse = libusb_open_device_with_vid_pid(ctx, 12943,
          33); //open mouse
523     // 081f:e401
524     // mouse = libusb_open_device_with_vid_pid(ctx, 0x1c4f, 0
          x0002);
525     mouse = libusb_open_device_with_vid_pid(ctx, 0x081f, 0
          xe401);
526     printf("\n2222\n");
527
528
529
530
531
532     if (mouse == NULL)
533     {
534       printf("%s\n", "Cannot open device");
```

```
535       libusb_free_device_list(devs, 1); // free the list,
              unref the devices in it
536       libusb_exit(ctx);                 // close the session
537       return 0;
538     }
539     else
540     {
541       printf("%s\n", "Device opened");
542       libusb_free_device_list(devs, 1); // free the list,
              unref the devices in it
543       if (libusb_kernel_driver_active(mouse, 0) == 1)
544       { // find out if kernel driver is attached
545         printf("%s\n", "Kernel Driver Active");
546         if (libusb_detach_kernel_driver(mouse, 0) == 0) //
              detach it
547           printf("%s\n", "Kernel Driver Detached!");
548       }
549       r = libusb_claim_interface(mouse, 0); // claim
              interface 0 (the first) of device (mine had just 1)
550       if (r < 0)
551       {
552         printf("%s\n", "Cannot Claim Interface");
553         return 1;
554       }
555     }
556     printf("%s\n", "Claimed Interface");
557
558     pthread_create(&mouse_thread, NULL, mouse_thread_f, NULL)
          ;
559
560
561
562     while(1) {
563       // ===========
564         // 0. define
565         int brick_row = 6;
566         int brick_col = 13;
567         //int brick_matrix[brick_row][brick_col]; // 1 has
              brick 0 empty
568         // int brick_width = BRICK_W;
569         // int brick_height = BRICK_H;
570         int ball_radius = BALL_R;
571         int paddle_length = PADDLE_W;
572         int brick_h, brick_v;
573         int game_status;
574         int flag[6][13];
575         int flag_paddle = 0;
576         int matrixclear;
577
578         // 1. init
579         // initialize
580
581         h_location = 275;
582         game_status = 1;
583         game_over_sound = 1;
584         game_start = 0;
585
586
587         // 1.2 bricks
```

```
588         // stage 1
589         int brick_matrix[6][13] = {
590           {0,0,0,0,0,0,0,0,0,0,0,0,0},
591           {0,1,1,1,1,0,0,1,0,0,1,0,0},
592           {0,1,0,0,0,0,0,1,0,0,1,0,0},
593           {0,1,0,0,0,0,0,1,0,0,1,0,0},
594           {0,1,1,1,1,0,0,1,1,1,1,0,0},
595           {0,0,0,0,0,0,0,0,0,0,0,0,0}
596         };
597 
598         /*
599         // simifiy stage 1
600         for(int i = 0; i < brick_row; i++){
601           for (int j = 0; j < brick_col; j++) {
602               if ( i == 5 && j == 9){
603                   brick_matrix[i][j] = 1;
604               }
605               else if ( i == 4 && j == 11){
606                   brick_matrix[i][j] = 1;
607               }
608               else {
609                   brick_matrix[i][j] = 0;
610               }
611           }
612         } */
613 
614 
615         // stage 2 map
616         int stage2matrix[6][13] = {
617           {1,0,1,0,1,0,1,0,1,0,1,0,1},
618           {1,0,1,0,1,0,1,0,1,0,1,0,1},
619           {1,0,1,0,1,0,1,0,1,0,1,0,1},
620           {1,0,1,0,1,0,1,0,1,0,1,0,1},
621           {1,0,1,0,1,0,1,0,1,0,1,0,1},
622           {1,0,1,0,1,0,1,0,1,0,1,0,1}
623         };
624 
625         /*
626         // simifiy stage 2
627         for(int i = 0; i < brick_row; i++){
628           for (int j = 0; j < brick_col; j++) {
629               if ( i == 5 && j == 9){
630                   stage2matrix[i][j] = 1;
631               }
632               else if ( i == 4 && j == 11){
633                   stage2matrix[i][j] = 1;
634               }
635               else {
636                   stage2matrix[i][j] = 0;
637               }
638           }
639         }*/
640 
641 
642         // 3. easy mode
643         int easy_matrix[6][13] = {
644           {0,0,0,0,0,0,0,0,0,0,0,0,0},
645           {0,0,0,0,0,0,0,0,0,0,0,0,0},
646           {0,0,0,0,0,0,0,0,0,0,0,0,0},
```

```
647           {0,0,0,0,0,0,0,0,0,0,0,0,0},
648           {0,0,0,0,0,0,0,0,0,0,1,0,0},
649           {0,0,0,0,0,0,0,0,0,0,0,0,0}
650         };
651
652
653
654         // 1.3 ball
655         ball_h = 208;
656         ball_v = 425;
657         int step = 1;
658         int speed_h = 1;
659         int speed_v = -2;
660         int brick_width = 32;
661         int brick_height = 16;
662
663
664
665
666         // init data
667         data.x_pad = 208;  // work
668         gloabl_score = 0;
669         game_stage = 0;
670         game_over = 0;
671         game_hp = 3;
672         sound_effect = 0;
673         easy_mode = 0;
674
675         data.x_ball = ball_h;
676         data.y_ball = ball_v;
677         data.brick1  = convert2bin( brick_matrix[0], 0 );
678         data.brick2  = convert2bin( brick_matrix[1], 1 );
679         data.brick3  = convert2bin( brick_matrix[2], 2 );
680         data.brick4  = convert2bin( brick_matrix[3], 3 );
681         data.brick5  = convert2bin( brick_matrix[4], 4 );
682         data.brick6  = convert2bin( brick_matrix[5], 5 );
683         data.score = hex2hexadecimal(gloabl_score);
684         data.game_status = get_game_status(game_stage,
              game_hp, game_over);
685
686         set_background_color(&data); // TODO
687
688
689       // game logitic
690       while (1)
691       {
692
693         sound_effect = SOUND_DEFULT;
694         if (game_start == 0) {
695           continue;
696         }
697
698
699
700
701         sound_effect = 0; // init
702
703         ball_h += speed_h * step;
704         ball_v += speed_v * step;
```

```
705         // data.x_ball       = (int) ball_h * 4; // TODO
706         // data.green = 195;
707         data.x_ball = ball_h;
708         data.y_ball = ball_v;
709
710
711         //printf(" %d, %d \n", data.x_ball, data.y_ball);
712
713
714         // check hit wall
715         if (ball_v <= 53) {
716           speed_v = 0 - speed_v;
717           sound_effect = SOUND_WALL_PAD;
718         }
719         if (ball_h >= 411) {
720           speed_h = 0 - speed_h;
721           sound_effect = SOUND_WALL_PAD;
722         }
723         if (ball_h <= 5) {
724           speed_h = 0 - speed_h;
725           sound_effect = SOUND_WALL_PAD;
726         }
727         // check hit pad
728         //if (ball_h <=  data.x_pad +20 && ball_h >= data.
              x_pad -20 && ball_v == 429/*ball_v +5 >= 430 &&
              ball_v + 5 <= 440*/){
729         if (ball_h <=  data.x_pad +20 && ball_h >= data.x_pad
               -20 && ball_v >= 425 && ball_v <= 430/*ball_v +5
             >= 430 && ball_v + 5 <= 440*/){
730           speed_v = 0 - speed_v;
731           sound_effect = SOUND_WALL_PAD;
732         }
733
734         //printf("easy_mode, %d", easy_mode);
735         // easy mode
736         if (easy_mode == 1) {
737             for(int i = 0; i < brick_row; i++){
738              for (int j = 0; j < brick_col; j++) {
739                brick_matrix[i][j] = easy_matrix[i][j] ;
740              }
741            }
742         }
743
744
745         // check hit brick
746         for (int i = 0; i < 6; i++) {
747           for (int j = 0; j < 13; j++) {
748             if (brick_matrix[i][j] == 1)
749                 {
750                   //printf("%d \n",brick_matrix[i][j]);
751                   brick_h = brick_width * j;
752                   brick_v = brick_height * i + 80;
753                   flag[i][j] = hitOnBrick(ball_v, ball_h,
                          brick_width, brick_height, brick_h,
                          brick_v);
754                   // update flag
755                   if (flag[i][j] == 1) {
756                         speed_v = 0 - speed_v;
757                         brick_matrix[i][j] = 0;
```

```
758                     }
759                     if (flag[i][j] == 2) {
760                           speed_v = 0 - speed_v;
761                           brick_matrix[i][j] = 0;
762                     }
763                     if (flag[i][j] == 3) {
764                           speed_h = 0 - speed_h;
765                           brick_matrix[i][j] = 0;
766                     }
767                     if (flag[i][j] == 4) {
768                           speed_h = 0 - speed_h;
769                           brick_matrix[i][j] = 0;
770                     }
771                     // hit brick
772                     if (flag[i][j] != 0) {
773                       sound_effect = SOUND_HIT_BRICK;
774                       gloabl_score += 10;
775                     }
776                   }
777               }
778           }
779
780
781           // check game over TODO
782           if (ball_v >= 475) {
783             game_hp -= 1;
784
785             ball_h = data.x_pad;
786             ball_v = 425;
787             game_start = 0;
788             speed_h = 1;
789             speed_v = -2;
790           }
791
792           if (game_hp == 0) {
793             sound_effect = SOUND_GAME_OVER;
794             data.sound_effect = sound_effect;
795             game_over = 1;
796             break;
797           }
798
799
800           //printf("%d %d, %d, %d \n", data.brick3, data.brick4
                , data.brick5, data.brick6);
801
802
803           matrixclear = check_clear(brick_matrix, 6, 13);
804
805           // win
806           if (matrixclear == 1 && game_stage == 1) {
807             game_over = 2;
808             sound_effect = 4;
809             data.brick1  = convert2bin( brick_matrix[0], 0 );
810             data.brick2  = convert2bin( brick_matrix[1], 1 );
811             data.brick3  = convert2bin( brick_matrix[2], 2 );
812             data.brick4  = convert2bin( brick_matrix[3], 3 );
813             data.brick5  = convert2bin( brick_matrix[4], 4 );
814             data.brick6  = convert2bin( brick_matrix[5], 5 );
815             data.sound_effect = sound_effect;
```

```
816           data.score = hex2hexadecimal(gloabl_score);
817           data.game_status = get_game_status(game_stage,
                  game_hp,game_over);
818           break;
819         }
820
821         //  2rd stage
822         if ( matrixclear == 1) {
823           data.x_pad = 208;
824           ball_h = data.x_pad;
825           ball_v = 425;
826           game_start = 0;
827           step = 2; // ball step
828           speed_h = 1;
829           speed_v = -2;
830           game_stage = 1;
831           sound_effect = 0;
832           easy_mode = 0;
833
834           for(int i = 0; i < brick_row; i++){
835                 for (int j = 0; j < brick_col; j++) {
836                   brick_matrix[i][j] = stage2matrix[i][j] ;
837                 }
838               }
839         }
840
841         // assign data
842         data.brick1  = convert2bin( brick_matrix[0], 0 );
843         data.brick2  = convert2bin( brick_matrix[1], 1 );
844         data.brick3  = convert2bin( brick_matrix[2], 2 );
845         data.brick4  = convert2bin( brick_matrix[3], 3 );
846         data.brick5  = convert2bin( brick_matrix[4], 4 );
847         data.brick6  = convert2bin( brick_matrix[5], 5 );
848         data.sound_effect = sound_effect;
849         data.score = hex2hexadecimal(gloabl_score);
850         data.game_status = get_game_status(game_stage,
              game_hp,game_over);
851
852
853         set_background_color(&data); // TODO
854         usleep(15000);               // 1000000
855
856       }
857
858       if (game_over == 2) {
859         data.game_status = get_game_status(game_stage,
              game_hp, game_over);
860         set_background_color(&data);
861       } else if (game_over == 1) {
862         data.game_status = get_game_status(game_stage,
              game_hp, game_over);
863         set_background_color(&data);
864       }
865
866
867       // reset sound aviod too long
868       usleep(250000);
869       sound_effect = 0;
870       data.sound_effect = sound_effect;
```

```
871       int original_ball_x = ball_h;
872       int original_pad_x = data.x_pad;
873       //set_background_color(&data);
874
875       re_start = 0;
876       while (re_start == 0) {
877           data.x_pad = original_pad_x;
878           data.x_ball = original_ball_x;
879           re_start = 0;
880           set_background_color(&data);
881       }
882     }
883
884     return 0;
885   }
886
887
888
889   //read the mouse
890   void *mouse_thread_f(void *ignored)
891   {
892     printf("mouse thread started\n");
893
894     vga_ball_arg_t vla;
895
896
897     while (1)
898     {
899       unsigned char buff[64];
900       int size = 8;
901       libusb_interrupt_transfer(mouse, 0x81, buff, 0x0008, &
              size, 0);
902
903       int t_speed = 0;
904       int pad_diff = 0;
905       int step = 2;
906       int screen_left = 16;
907       int screen_right = 400;
908
909       //printf("buff[0], buff[1], buff[2] %d, %d, %d,%d, %d, %d
              ,%d ,%d \n", buff[0], buff[1], buff[2], buff[3], buff
              [4], buff[5], buff[6], buff[7]);
910
911       // left:  0   127  0 128 128 15
912       // right: 255 127 0 128 128 15
913       // up:    127 0   0 128 128 15
914       // down   127 255 0 128 128 15
915       // A:     127 127 0 128 128 47
916       // restart: 127 127 0 128 128 15 32
917       // X + Y: 127 127 0 128 128 15  9
918       // X:     127 127 0 128 128 31
919       // Y:     127 127 0 128 128 143
920       if (buff[0] == 0) {
921         // 0 127 0 128 128 15
922         // left
923         pad_diff = -step;
924
925         // ball move with pad
```

```
926         if (game_start == 0 && (screen_left < ball_h - step))
                {
927           ball_h -= step;
928         }
929
930       } else if (buff[0] == 255) {
931         // right
932         // 255 127 0 128 128 15
933         pad_diff = step;
934
935         // ball move with pad
936         if (game_start == 0 && (ball_h + step < screen_right)
                ) {
937           ball_h += step;
938         }
939
940       //} else if (buff[0] == 127 && buff[1] == 0) {
941       } else if (buff[0] == 127 && buff[1] == 127 && buff[5]
            == 47) {
942         game_start = 1;
943         //printf("start \n");
944
945       //} else if (buff[0] == 127 && buff[1] == 255) {
946       } else if (buff[0] == 127 && buff[1] == 127 && buff[5]
            == 15 && buff[6] == 32) {
947         re_start = 1;
948         //printf("restart \n");
949       } else if (buff[0] == 127 && buff[1] == 127 && buff[5]
            == 159 && (game_start == 1)) {
950         //printf("easy");
951         easy_mode = 1;
952         //
953       }
954
955       //printf("%d, %d, %d, %d\n", buff[0],buff[1],buff[2],
            buff[3]);
956
957
958       // data.x_ball =  195;
959       // data.y_ball =  300;
960       //  30 - 360
961       // avoid pad hit pall
962       if ((screen_left < data.x_pad + pad_diff) && (data.
            x_pad + pad_diff < screen_right))
963       {
964         data.x_pad = data.x_pad + pad_diff; // work
965       }
966
967       data.x_ball = ball_h;
968       data.y_ball = ball_v;
969       set_background_color(&data); // TODO
970     }
971   }
```