Embedded System
Final Project: Pokemon Breaker

Rui Chen, rc3205
Dajing Xu, dx2178
Shaofu Wu, sw3385
Bingyao Shi, bs3119

Contents

1 Introduction

1.1 System Architecture

The system architecture is shown below. In this section, we would directly talking about the system level design of our project.

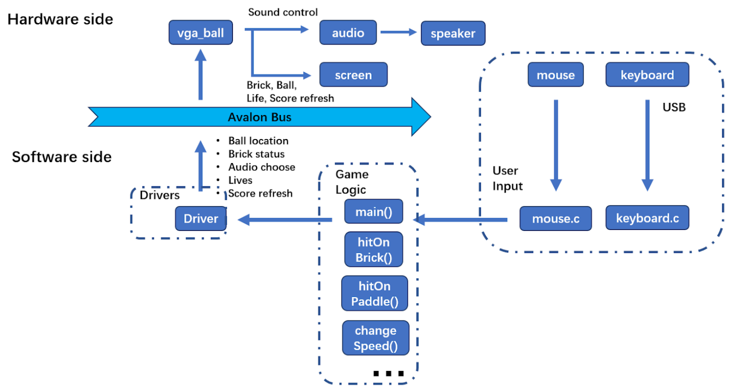


Figure 1.1: System Architecture

There are two ways to control the paddle in our brick breaker game. Players can either use mouse or keyboard to play. Keyboard uses two key, left-arrows and right-arrow. And mouse has two ways to play the game. One is scrolling, scroll up and the paddle will go right, scroll down and the paddle will go left.

For the software part, there are several functions in our main code. For example, *hitOnBrick()*, *hitOnPaddle()*, or *changeVelocityandChangeBrickStatus()*. These functions will return some parameters determine the state of the game. Moreover, these parameters will pass to the FPGA hardware part through one Avalon Bus.

There are two parts in hardware, audio and screen. Which will be fed by proper signal and the it will decide which music to be played or what to displayed on the monitor. For example, ball location, brick status, lives and score are some parameters that will be given to the screen. And sound control signal will be transferred to the audio part.

1.2 Game Rules

The game inherits most of the features of a traditional brick-breaker game: it can be controlled with multiple devices, it strictly follows physical laws when the ball bounces on the corner of a brick or the edge of the paddle, it has special sound effects when the ball hits on a brick/paddle/wall.

We have designed several levels with different brick layouts and the speed of the ball. The difficulty grows gradually as the player completes the previous level. The player wins the game when s/he has cleared the bricks in all levels within 3 lives wasted. The game restarts and returns to level 1 if the player has no lives left. On every new round, the ball is released by pressing ENTER key, and the ball is given a random initial direction, with an absolute speed growing with the level.

Green bricks are worth 2 points while blue ones are worth one. The score on the top of the screen also resets as the game restarts.

2 Hardware

2.1 Graphic Display

2.1.1 Memory Budget

The Cyclone V SoC FPGA device on DE1-SoC board include an embedded memory of 4,450 Kbits, which limits the maximum size of the graphic storage. The sprites we use have different sizes, ranging from 16×16 pixels to 45×45 pixels. Each pixel needs a storage size of 24 bits, 8 bits each for R, G, B signals. For this games graphic display, we will need a storage of 353 Kbits. The detailed calculation is shown below.

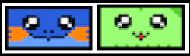
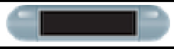
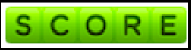
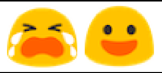
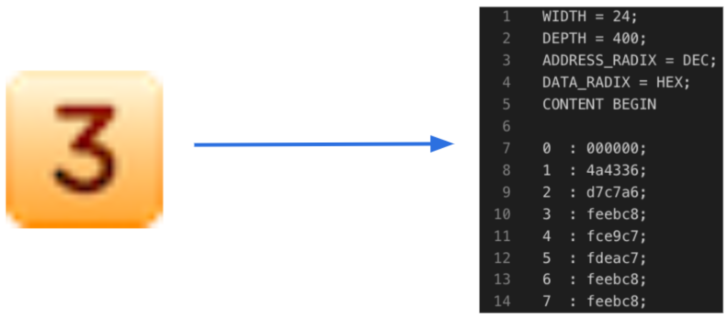
Category	Graphics	Size (bits)	# of images	Total size (bits)
Bricks		64*32	2	98,304
Ball		16*16	1	6,144
Paddel		90*20	1	43,200
Lives		24*22	1	12,672
Number		20*20	10	96,000
Score		100*20	1	48,000
Game Status		45*45	2	48,600
	Memory Budget (bits)			352920

Figure 2.1: Graph Memory Budget

We use memory initialization files to define on chip memory. All the .png images are first converted into .mif files using Matlab. The .mif file contains the address information and RGB data for each pixel. An example is shown in fig.2.2. Then a single-port ROM memory block is configured through MegaWizard for every sprite.



```
WIDTH = 24;
DEPTH = 400;
ADDRESS_RADIX = DEC;
DATA_RADIX = HEX;
CONTENT BEGIN

0  : 000000;
1  : 4a4336;
2  : d7c7a6;
3  : feebc8;
4  : fce9c7;
5  : fdeac7;
6  : feebc8;
7  : feebc8;
```

Figure 2.2: Example of .mif File

2.1.2 Graphic Display Architecture

Fig. 2.3 shows an example display of our game. It contains several parts: character score, 3-digit number, lives (shown as pikachu icons), 6 lines of bricks, poke ball and a paddle.

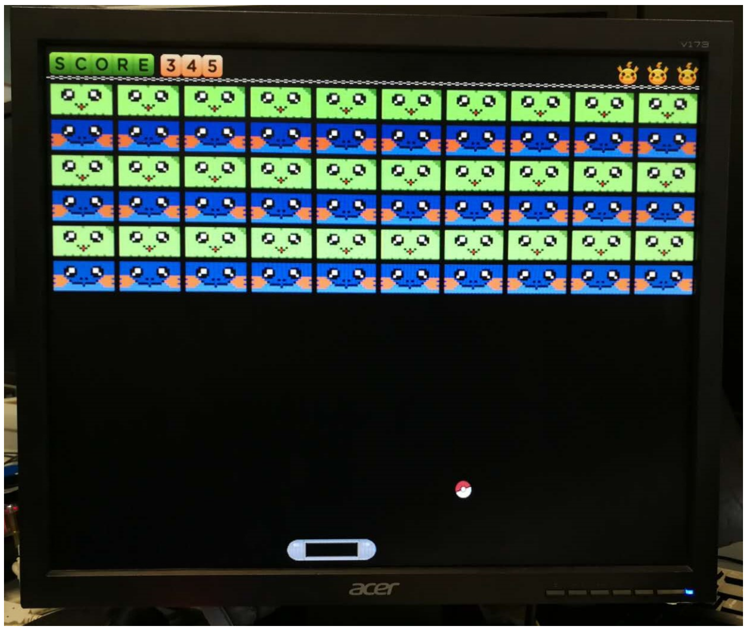


Figure 2.3: Our Game Start Window

The game consists of 4 layers. The order of layers is shown in Fig. 2.4.

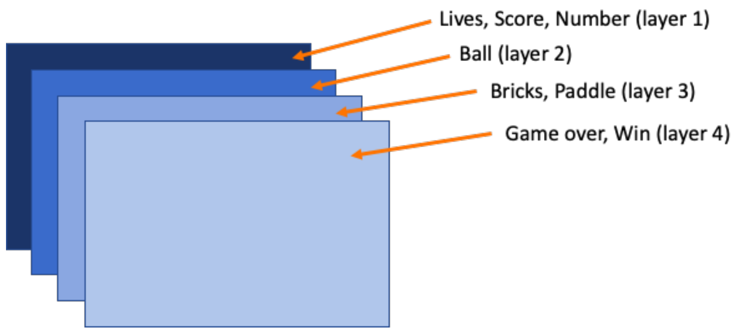


Figure 2.4: Graph Layers

The VGA output connector (15-pin D-SUB connector) on DE1-SoC board requires 5 output signals: VGA_R, VGA_G, VGA_B (color information), VGA_VS (vertical synchronization) and VGA_HS (horizontal synchronization). Cyclone V SoC FPGA generates thes 5 signals, of which VGA_HS and VGA_VS go directly into the connector output while VGA_R, VGA_G and VGA_B are transformed from digital to analog first by the Analog Devices ADV7123 triple 10-bit high-speed video DAC.

Fig.2.5 shows architecture for graphics display. vga_ball.sv is the core controller of the whole display function. It include modules to generate the mentioned 5 outputs. The VGA_HS and VGA_VS are generated by module vga_counter, which is the same as we used in Lab3. This vga_counter also generate hcount and vcount signals representing the pixel coordination being scanned. The 16-bit input writedata from Avalon bus directs vga_ball.sv to call for subcircuits such as paddle.v or ball.v to instantiate the ROM memory. Address data calculated from hcount and vcount are given to these subcircuits to retrieve RGB data. The required signals are then sent to their destination pins. Desired graphic will show on the VGA monitor.

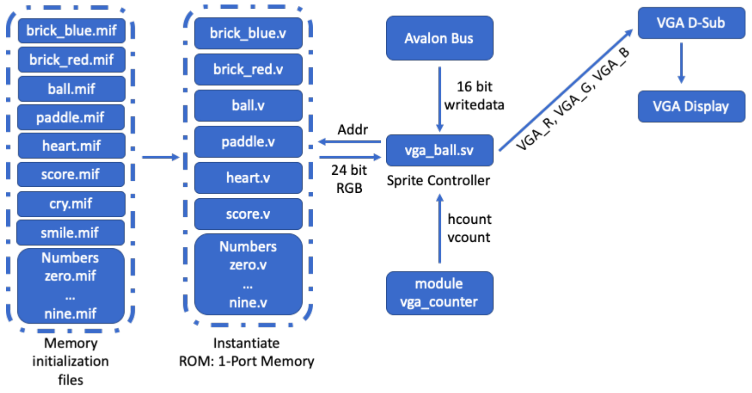


Figure 2.5: Graphic Architecture

2.2 Audio Output

2.2.1 Audio Architecture

The DE1-SoC board is using a audio chip WM8731. WM8731 allows CD quality (48kHz sampling rate at most) 24-bit data input and output. Before using this audio chip, user must configure it through I2C interface in advance. An interesting phenomenon in DE1-Soc board is that there is an I2C multiplexer, which allows users to configure WM8731 both from FPGA and HPS system on board. The default mode is FPGA, here we would also use this mode to configure it.

Below is the block diagram of our audio part.

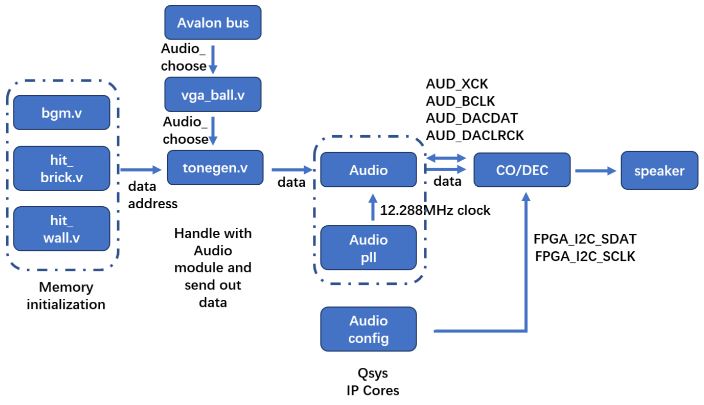


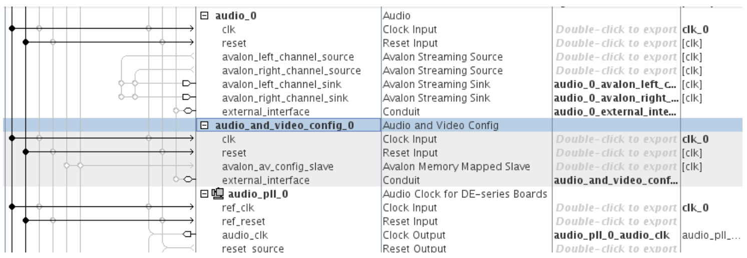
Figure 2.6: Audio Architecture

To use the CODEC WM8731, the first task is configure it through I2C bus. The default mode for DE1-SOC is fpga mode, and one can also configure it through hps. Here we make use of IP core audio_and_video_config, connect to the fpga ports to configure for us.

Then, we store three section of sounds into the on-chip memory. 13s bgm, 0.5s hit brick and hit wall sound effects, sampling frequency is 8kHz to save the memory. The audio choose signal sent from user space is used to control which sound to play. Accordingly, tonegen module feed the data to ip core audio, which includes a fifo in it and take care of the timing for us. The fifo frequency is 48kHz, thus in tonegen module we manually add a frequency divider of 6. Then Audio module is connected to WM8731 through the ports.

The configuration process can be summarized as below. The I2C interface requires two signals: SCL for clock and SDAT for data. In the idle state both signal are high. By pulling the SDAT line LOW while the clock line remains HIGH is the start condition. Then the master sends an address of a device that it wants to talk to. After a 7-bit address and a R/W bit are sent, the master releases the data line. The data line has a pull up resistor, thich pulls the line HIGH while its state is not actively controlled. If an I2C device receives its address, it pulls the data line LOW at the next clock cycle, which is called ACK. Then two byte of data is sent. After all the data are transmitted, a stop condition is generated by pulling the clock line High, while the data line is kept LOW[1].

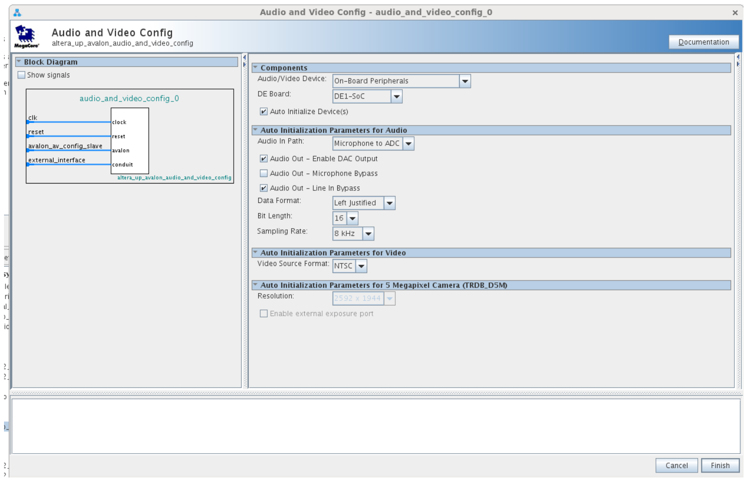
This process is quite complicated, one can make use of Qsys built-in IP core to automatically configure the audio chip. Refer to [2], another 3 audio modules are added into our project too. The Qsys interface is shown below:



⊟ **audio_0**	Audio		
clk	Clock Input	*Double-click to export*	**clk_0**
reset	Reset Input	*Double-click to export*	[clk]
avalon_left_channel_source	Avalon Streaming Source	*Double-click to export*	[clk]
avalon_right_channel_source	Avalon Streaming Source	*Double-click to export*	[clk]
avalon_left_channel_sink	Avalon Streaming Sink	**audio_0_avalon_left_c...**	[clk]
avalon_right_channel_sink	Avalon Streaming Sink	**audio_0_avalon_right_...**	[clk]
external_interface	Conduit	**audio_0_external_inte...**	
⊟ **audio_and_video_config_0**	Audio and Video Config		
clk	Clock Input	*Double-click to export*	**clk_0**
reset	Reset Input	*Double-click to export*	[clk]
avalon_av_config_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
external_interface	Conduit	**audio_and_video_conf...**	
⊟ **audio_pll_0**	Audio Clock for DE-series Boards		
ref_clk	Clock Input	*Double-click to export*	**clk_0**
ref_reset	Reset Input	*Double-click to export*	
audio_clk	Clock Output	**audio_pll_0_audio_clk**	audio_pll_...
reset_source	Reset Output	*Double-click to export*	

Figure 2.7: Qsys Setting for Audio Module

The audio_pll_0 module is providing a clock of 12.288 MHz for the audio chip, which would connect to AUD_XCK port on the board. The WM8731 is configured to work at a 16-bit data width, 8kHz sampling rate and working at Left Justified Mode. In this project, we dont care about audio input.



Audio and Video Config - audio_and_video_config_0

Audio and Video Config
altera_up_avalon_audio_and_video_config

Components
Audio/Video Device: On-Board Peripherals
DE Board: DE1-SoC
[✓] Auto Initialize Device(s)

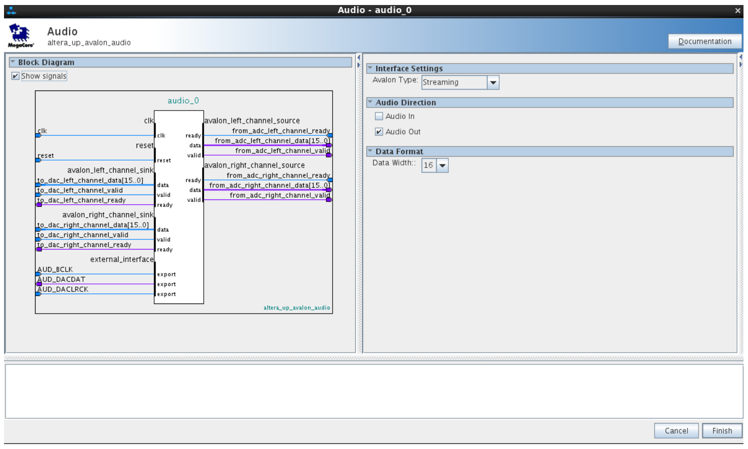
Auto Initialization Parameters for Audio
Audio In Path: Microphone to ADC
[✓] Audio Out - Enable DAC Output
[ ] Audio Out - Microphone Bypass
[✓] Audio Out - Line In Bypass
Data Format: Left Justified
Bit Length: 16
Sampling Rate: 8 kHz

Auto Initialization Parameters for Video
Video Source Format: NTSC

Auto Initialization Parameters for 5 Megapixel Camera (TRDB_D5M)
Resolution: 2592 x 1944
[ ] Enable external exposure port

Figure 2.8: Audio Configure IP Core

The Audio module is taking care of the timing and also gives a 48kHz output fifo for data transmission. Notice the dac part, there are two different channels here. We are actually using a single channel sound, so the same data is sent to both channel. The ready signal in each channel tells the tonegen module written by ourselves that the fifo is not full, data can be sent now. The valid signal tells audio module to accept the data in this time circle.


Audio - audio_0
Audio
altera_up_avalon_audio

Interface Settings
Avalon Type: Streaming

Audio Direction
☐ Audio In
☑ Audio Out

Data Format
Data Width:: 16

Documentation
Block Diagram
Show signals
audio_0
clk
reset
avalon_left_channel_sink
to_dac_left_channel_data[15..0]
to_dac_left_channel_valid
to_dac_left_channel_ready
avalon_right_channel_sink
to_dac_right_channel_data[15..0]
to_dac_right_channel_valid
to_dac_right_channel_ready
external_interface
AUD_BCLK
AUD_DACDAT
AUD_DACLRCK
avalon_left_channel_source
from_adc_left_channel_ready
from_adc_left_channel_data[15..0]
from_adc_left_channel_valid
avalon_right_channel_source
from_adc_right_channel_ready
from_adc_right_channel_data[15..0]
from_adc_right_channel_valid
data
valid
ready
export
altera_up_avalon_audio
Cancel
Finish

Figure 2.9: Audio IP Core

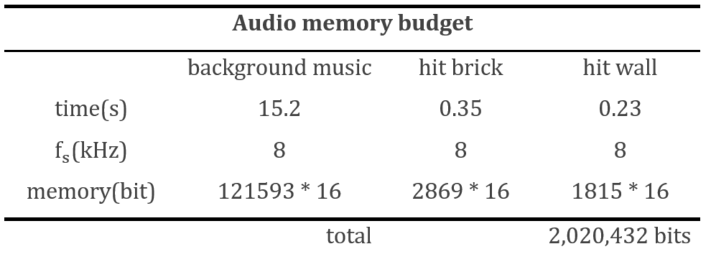
2.2.2 Audio Issues and Solutions

When ball hits on brick or wall, only in a single loop the audio_choose is set, in the next loop it goes back to 0. In our design, a loop is roughly 1.2ms, which is much shorter than the sound effects which are around 0.3s. Solve this from hardware side:

Use a flag to mark whether the sound effect is over. This method arise another problem is once the ball is hit on the wall, the sound must be played until finishing. During this period of time, hitting on brick, hitting on wall would not produce new sound.

2.2.3 Audio Memory Budget

Below is the budget for audio part in our project, the total memory cost is 2000kbits out of 4450 Kbits.



Audio memory budget			
	background music	hit brick	hit wall
time(s)	15.2	0.35	0.23
f_s(kHz)	8	8	8
memory(bit)	121593 * 16	2869 * 16	1815 * 16
	total		2,020,432 bits

Figure 2.10: Audio Memory Budget

3 Software

3.1 User Input

3.1.1 Mouse

Created a mouse thread in the main file to get the reading from the mouse. This thread is meant to not to pause the game. In the mouse thread, read the return value from the USB mouse connected to the board. And the four bytes can be categorized into two parts. Moving data and scrolling data. Datain[3] store the scrolling information. Datain[1] store the moving information (x axis) Designed to be the faster you move the mouse the faster the paddle will move.

Use the function below to open the mouse and get raw data from the mouse.

dev_handle = libusb_open_device_with_vid_pid(ctx, 16700, 12314); //open mouse

16700 and 12314 is the distinct vendorID and ProductID for the device, some more details are shown in Appendix A.

rr = libusb_interrupt_transfer(dev_handle, 0x81, datain, 0x0004, &size, 0);

Where *dev_handle* is a *libusb_device_handle*. It is mouse in this case. $0x81$ is the endpoint for input. Datain is a 1024 bytes buffer. 0x0004 limits the maximum number of bytes to receive. Size is number of bytes that actual transferred 0 indicates that this function will wait until getting a response, if it is some other integer, it means that this function will wait for some milliseconds. If timeout, it will return nothing and start the next timeout.

One more tricky thing that should be mentioned, the USB device might be connected to kernel so we have to detach it before we use it.

3.1.2 Keyboard

Created a keyboard in the main file to get two reading from the keyboard. Left-arrow and Right-arrow were detected by the thread, and control the paddle by certain speed. This is part of the keyboard thread which convert the keypressed to paddle movement.

3.2 Game Logic

3.2.1 Flow Chart

Game logic of our project can be shown in figure below.

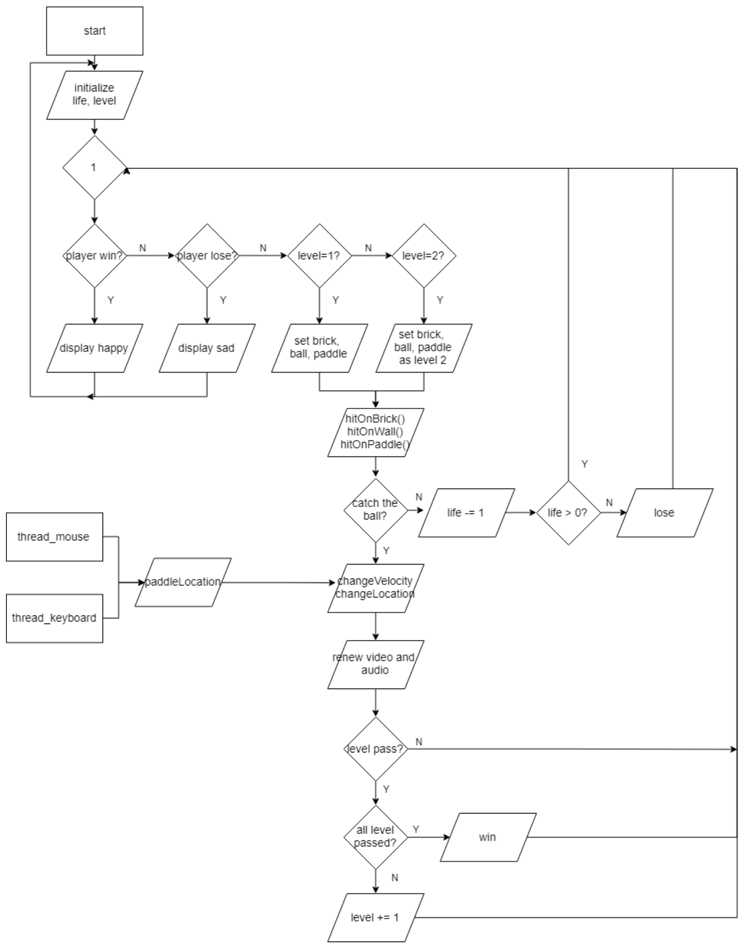


Figure 3.1: Flowchart of Game Logic

3.2.2 Control Logic Design

The hardware and software parts cooperate in a way similar to what we did in Lab3. The software side is responsible to do all the core logical calculations, and then send control parameters to the hardware. The hardware takes these parameters, displays the graphics accordingly and the screen, and plays the corresponding audio.

These parameters consist of:

- Ball location: (x,y) (2*10 bits).
- Paddle location (10 bits).
- Brick status: 1, 0 for each brick (1*6*10 bits).

- Score: 3 digits decimal, transmitted separately (3*4 bits).
- Lives 3, 2, 1, 0 (2 bits).
- Game status: normal, won(display smiling faces), lost(display crying faces) (2 bits).
- Audio control normal, hit_wall, hit_brick (2 bits).

There are 60 fixed slots reserved to display bricks. We use a 1-bit variable to record its status. Once a brick is hit, the corresponding bit is set to zero.

3.2.3 Hit on Single Brick Model

Clearly, if we have a model to describe the ball hitting on a single brick, use the same model on every brick can tell us whether the ball hit on the bricks or not. The first thing to determine is whether the ball hits on the brick. The ball is hitting the brick, if it falls in the blue region in picture below. This gives us a quick way to determine the event happening. According to different regions the ball falls at, the velocity changes accordingly.

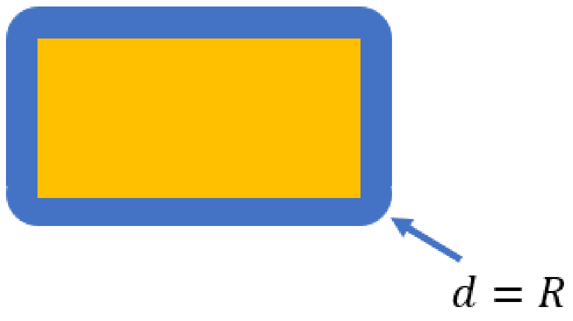


Figure 3.2: Determine Hit on Brick

The next step is changing the ball velocity according to the region it falls in. If the ball falls in the left or right region, the horizontal velocity changes into the opposite number. Similarly, if the ball falls in the upper or bottom region, vertical velocity changes into the opposite number. A complicated case is if the ball falls in the four corners.

Here we view the corner as a ball with a radius of 0 and built the following model.

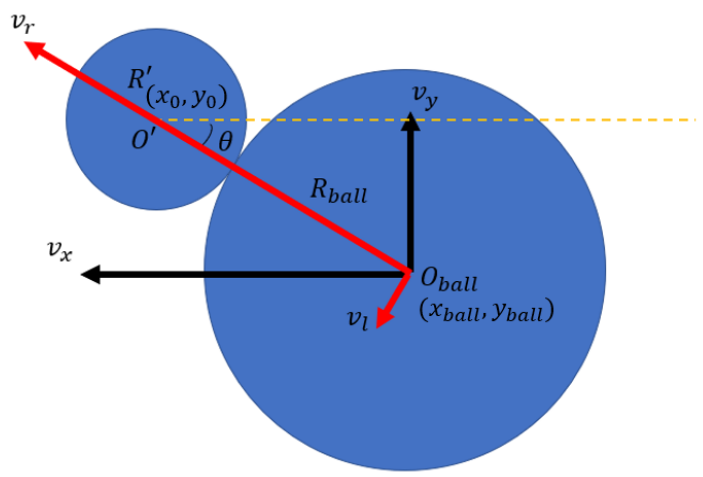


Figure 3.3: Corner Situation Consideration

One can decompose the vertical and horizontal velocity into radial and lateral direction. One assumes that the smaller ball is fixed and represents the corner. Its obvious if the larger ball hit on it, the lateral velocity remain unchanged while the radial velocity changes into opposite direction. The algebra below is a little complicated but not difficult. The final result gives:

$$\begin{cases} v'_x = -\cos 2\theta \cdot v_x - \sin 2\theta \cdot v_y \\ v'_y = -\sin 2\theta \cdot v_x + \cos 2\theta \cdot v_y \end{cases} \tag{3.1}$$

Where v_x and v_y are original horizontal and vertical velocity, v'_x and v'_y are changed horizontal and vertical velocity, θ is determined by:

$$\theta = \tan^{-1} \frac{y_0 - y_{ball}}{x_0 - x_{ball}} \tag{3.2}$$

3.2.4 Hit on Paddle Model

The paddle model is similar to the brick model. The shape of the paddle can be view as two half circles with radius equals to 10 locate at both sides and in the middle is a simple plane.

4 Discussion

4.1 Shining Points

- Provide the user with 3 lives.
- Random generates the direction of initial velocity.
- Score system.
- Audio sound effects on top of background music.
- Multiple ways of control.
- Realistic bouncing models.

4.2 Challenges

- When the paddle is moving towards the ball, there is a small probability that the ball falls into the paddle, which is not realistic.
- When the ball is hitting the wall and before the sound is finish it hits on another object, no sound would be produced.
- User experience of mouse has to be improved. The return data for scrolling up the mouse is 0x1, and scrolling down the mouse is 0xff.

  The problem here is that when we scroll one time, the data will change, but not until user moves it. Therefore, some bugs exist in our mouse controlling algorithm. Moreover, we find that different mice can have different change when we scroll the mice.
- Another big challenge for controlling mouse is the difference between normal mouse and gaming mouse. The methods and function used for controlling normal mouse cant be used directly to control gaming mouse.
- Next challenge for controlling mouse is how to mimic the movement of the mouse to the movement of the paddle in our game.

References

[1] *https://www.youtube.com/watch?v=zzIi7ErWhAA.*

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

APEENDIX

//**

/***************vga_ball.sv*******************/**

//**

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

               output logic [7:0] VGA_R, VGA_G, VGA_B,
               output logic       VGA_CLK, VGA_HS, VGA_VS,
                                   VGA_BLANK_n,
               output logic       VGA_SYNC_n,
               output logic [1:0] audio_choose);

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

   logic [7:0]        background_r, background_g, background_b;
   logic [15:0] paddle_left, ball_h, ball_v, heart_status;
   logic [15:0] brick_status0, brick_status1, brick_status2, brick_status3,
brick_status4, brick_status5, brick_status6;
   logic [15:0] score_number1, score_number2, score_number3;
   logic [15:0] game_status;

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

        always_ff @(posedge clk)
               if (reset) begin
                       background_r <= 8'h00;
                       background_g <= 8'h00;
                       background_b <= 8'h00;
                       paddle_left <= 16'd230;//9 bits should be enough
                       ball_h <= 16'd400;
                        ball_v <= 16'd400;
                        brick_status0 <= 16'b0000000000000000;//0000001111111000;
                        brick_status1 <= 16'b0000001111111111;
                        brick_status2 <= 16'b0000001111111111;
                        brick_status3 <= 16'b0000001111111111;
                        brick_status4 <= 16'b0000001111111111;
                        brick_status5 <= 16'b0000001111111111;
                        brick_status6 <= 16'b0000001111111111;
                       heart_status  <= 16'b0000000000000111;
                       score_number1  <= 16'd3;
                       score_number2  <= 16'd4;
                       score_number3  <= 16'd5;
                       game_status    <= 16'd0;
                       audio_choose  <= 2'b00;
               end
               else if (chipselect && write)
                       case (address)
                               4'd0 : paddle_left <= writedata;
                               4'd1 : ball_h <= writedata;
                                 4'd2 : ball_v <= writedata;
                                 4'd3 : brick_status0 <= writedata;
                                 4'd4 : brick_status1 <= writedata;
                                 4'd5 : brick_status2 <= writedata;
                                 4'd6 : brick_status3 <= writedata;
                                 4'd7 : brick_status4 <= writedata;
                                 4'd8 : brick_status5 <= writedata;
                                 4'd9 : brick_status6 <= writedata;
                                 4'd10: heart_status <= writedata;
```

```
                                4'd11: audio_choose  <= writedata;
                                4'd12: score_number1 <= writedata;
                                4'd13: score_number2 <= writedata;
                                4'd14: score_number3 <= writedata;
                                4'd15: game_status <= writedata;
                        endcase

        logic [10:0] cry_address;
        logic [23:0] cry_output;
        cry cry_unit(.address(cry_address),.clock(clk),.q(cry_output));
        logic [1:0] cry_en;

        always_ff @(posedge clk) begin
                //cry1 upper left corner h=248, v=228
                if (hcount[10:1] >= 10'd248 && hcount[10:1] <= 10'd292 && vcount >=
10'd228 && vcount <= 10'd272 && game_status == 2) begin
                        cry_en <= 2'b1;
                        cry_address <= hcount[10:1]-10'd248 + (vcount-10'd228)*45;
                end
                //number2 upper left corner h=298, v=228
                else if (hcount[10:1] >= 10'd298 && hcount[10:1] <= 10'd342 && vcount >=
10'd228 && vcount <= 10'd272 && game_status == 2) begin
                        cry_en <= 2'b1;
                        cry_address <= hcount[10:1]-10'd298 + (vcount-10'd228)*45;
                end
                //number3 upper left corner h=348, v=228
                else if (hcount[10:1] >= 10'd348 && hcount[10:1] <= 10'd392 && vcount >=
10'd228 && vcount <= 10'd272 && game_status == 2) begin
                        cry_en <= 2'b1;
                        cry_address <= hcount[10:1]-10'd348 + (vcount-10'd228)*45;
                end
                else begin
                        cry_en <= 2'b0;
                end
        end

        logic [10:0] smile_address;
        logic [23:0] smile_output;
        smile smile_unit(.address(smile_address),.clock(clk),.q(smile_output));
        logic [1:0] smile_en;

        always_ff @(posedge clk) begin
                //smile1 upper left corner h=248, v=228
                if (hcount[10:1] >= 10'd248 && hcount[10:1] <= 10'd292 && vcount >=
10'd228 && vcount <= 10'd272 && game_status == 1) begin
                        smile_en <= 2'b1;
                        smile_address <= hcount[10:1]-10'd248 + (vcount-10'd228)*45;
                end
                //smile2 upper left corner h=298, v=228
                else if (hcount[10:1] >= 10'd298 && hcount[10:1] <= 10'd342 && vcount >=
10'd228 && vcount <= 10'd272 && game_status == 1) begin
                        smile_en <= 2'b1;
                        smile_address <= hcount[10:1]-10'd298 + (vcount-10'd228)*45;
                end
                //smile3 upper left corner h=348, v=228
                else if (hcount[10:1] >= 10'd348 && hcount[10:1] <= 10'd392 && vcount >=
10'd228 && vcount <= 10'd272 && game_status == 1) begin
                        smile_en <= 2'b1;
                        smile_address <= hcount[10:1]-10'd348 + (vcount-10'd228)*45;
                end
                else begin
                        smile_en <= 2'b0;
                end
        end

        logic [8:0] zero_address;
        logic [23:0] zero_output;
        zero zero_unit(.address(zero_address),.clock(clk),.q(zero_output));
        logic [1:0] zero_en;

        always_ff @(posedge clk) begin
```

```
		//number1 upper left corner h=108, v=4
		if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6
&& vcount <= 5'd25 && score_number1 == 0) begin
			zero_en <= 2'b1;
			zero_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
		end
		//number2 upper left corner h=128, v=4
		else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >=
5'd6 && vcount <= 5'd25 && score_number2 == 0) begin
			zero_en <= 2'b1;
			zero_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
		end
		//number3 upper left corner h=148, v=5
		else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >=
5'd6 && vcount <= 5'd25 && score_number3 == 0) begin
			zero_en <= 2'b1;
			zero_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
		end
		else begin
			zero_en <= 2'b0;
		end
	end

	logic [8:0] one_address;
	logic [23:0] one_output;
	one one_unit(.address(one_address),.clock(clk),.q(one_output));
	logic [1:0] one_en;

	always_ff @(posedge clk) begin
		//number1 upper left corner h=108, v=4
		if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6
&& vcount <= 5'd25 && score_number1 == 1) begin
			one_en <= 2'b1;
			one_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
		end
		//number2 upper left corner h=128, v=4
		else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >=
5'd6 && vcount <= 5'd25 && score_number2 == 1) begin
			one_en <= 2'b1;
			one_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
		end
		//number3 upper left corner h=148, v=5
		else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >=
5'd6 && vcount <= 5'd25 && score_number3 == 1) begin
			one_en <= 2'b1;
			one_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
		end
		else begin
			one_en <= 2'b0;
		end
	end

	logic [8:0] two_address;
	logic [23:0] two_output;
	two two_unit(.address(two_address),.clock(clk),.q(two_output));
	logic [1:0] two_en;

	always_ff @(posedge clk) begin
		//number1 upper left corner h=108, v=4
		if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6
&& vcount <= 5'd25 && score_number1 == 2) begin
			two_en <= 2'b1;
			two_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
		end
		//number2 upper left corner h=128, v=4
		else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >=
5'd6 && vcount <= 5'd25 && score_number2 == 2) begin
			two_en <= 2'b1;
			two_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
		end
		//number3 upper left corner h=148, v=5
		else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >= 
5'd6 && vcount <= 5'd25 && score_number3 == 2) begin
			two_en <= 2'b1;
			two_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
		end
		else begin
			two_en <= 2'b0;
		end
	end

	logic [8:0] three_address;
	logic [23:0] three_output;
	three three_unit(.address(three_address),.clock(clk),.q(three_output));
	logic [1:0] three_en;

	always_ff @(posedge clk) begin
		//number1 upper left corner h=108, v=4
		if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6 
&& vcount <= 5'd25 && score_number1 == 3) begin
			three_en <= 2'b1;
			three_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
		end
		//number2 upper left corner h=128, v=4
		else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >= 
5'd6 && vcount <= 5'd25 && score_number2 == 3) begin
			three_en <= 2'b1;
			three_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
		end
		//number3 upper left corner h=148, v=5
		else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >= 
5'd6 && vcount <= 5'd25 && score_number3 == 3) begin
			three_en <= 2'b1;
			three_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
		end
		else begin
			three_en <= 2'b0;
		end
	end

	logic [8:0] four_address;
	logic [23:0] four_output;
	four four_unit(.address(four_address),.clock(clk),.q(four_output));
	logic [1:0] four_en;

	always_ff @(posedge clk) begin
		//number1 upper left corner h=108, v=4
		if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6 
&& vcount <= 5'd25 && score_number1 == 4) begin
			four_en <= 2'b1;
			four_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
		end
		//number2 upper left corner h=128, v=4
		else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >= 
5'd6 && vcount <= 5'd25 && score_number2 == 4) begin
			four_en <= 2'b1;
			four_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
		end
		//number3 upper left corner h=148, v=5
		else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >= 
5'd6 && vcount <= 5'd25 && score_number3 == 4) begin
			four_en <= 2'b1;
			four_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
		end
		else begin
			four_en <= 2'b0;
		end
	end

	logic [8:0] five_address;
	logic [23:0] five_output;
	five five_unit(.address(five_address),.clock(clk),.q(five_output));
	logic [1:0] five_en;

	always_ff @(posedge clk) begin
		//number1 upper left corner h=108, v=4
		if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6
&& vcount <= 5'd25 && score_number1 == 5) begin
			five_en <= 2'b1;
			five_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
		end
		//number2 upper left corner h=128, v=4
		else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >=
5'd6 && vcount <= 5'd25 && score_number2 == 5) begin
			five_en <= 2'b1;
			five_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
		end
		//number3 upper left corner h=148, v=5
		else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >=
5'd6 && vcount <= 5'd25 && score_number3 == 5) begin
			five_en <= 2'b1;
			five_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
		end
		else begin
			five_en <= 2'b0;
		end
	end

	logic [8:0] six_address;
	logic [23:0] six_output;
	six six_unit(.address(six_address),.clock(clk),.q(six_output));
	logic [1:0] six_en;

	always_ff @(posedge clk) begin
		//number1 upper left corner h=108, v=4
		if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6
&& vcount <= 5'd25 && score_number1 == 6) begin
			six_en <= 2'b1;
			six_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
		end
		//number2 upper left corner h=128, v=4
		else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >=
5'd6 && vcount <= 5'd25 && score_number2 == 6) begin
			six_en <= 2'b1;
			six_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
		end
		//number3 upper left corner h=148, v=5
		else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >=
5'd6 && vcount <= 5'd25 && score_number3 == 6) begin
			six_en <= 2'b1;
			six_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
		end
		else begin
			six_en <= 2'b0;
		end
	end

	logic [8:0] seven_address;
	logic [23:0] seven_output;
	seven seven_unit(.address(seven_address),.clock(clk),.q(seven_output));
	logic [1:0] seven_en;

	always_ff @(posedge clk) begin
		//number1 upper left corner h=108, v=4
		if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6
&& vcount <= 5'd25 && score_number1 == 7) begin
			seven_en <= 2'b1;
			seven_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
		end
		//number2 upper left corner h=128, v=4
		else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >=
5'd6 && vcount <= 5'd25 && score_number2 == 7) begin
			seven_en <= 2'b1;
```

```
                        seven_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
                end
                //number3 upper left corner h=148, v=5
                else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >=
5'd6 && vcount <= 5'd25 && score_number3 == 7) begin
                        seven_en <= 2'b1;
                        seven_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
                end
                else begin
                        seven_en <= 2'b0;
                end
        end

        logic [8:0] eight_address;
        logic [23:0] eight_output;
        eight eight_unit(.address(eight_address),.clock(clk),.q(eight_output));
        logic [1:0] eight_en;

        always_ff @(posedge clk) begin
                //number1 upper left corner h=108, v=4
                if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6
&& vcount <= 5'd25 && score_number1 == 8) begin
                        eight_en <= 2'b1;
                        eight_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
                end
                //number2 upper left corner h=128, v=4
                else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >=
5'd6 && vcount <= 5'd25 && score_number2 == 8) begin
                        eight_en <= 2'b1;
                        eight_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
                end
                //number3 upper left corner h=148, v=5
                else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >=
5'd6 && vcount <= 5'd25 && score_number3 == 8) begin
                        eight_en <= 2'b1;
                        eight_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
                end
                else begin
                        eight_en <= 2'b0;
                end
        end

        logic [8:0] nine_address;
        logic [23:0] nine_output;
        nine nine_unit(.address(nine_address),.clock(clk),.q(nine_output));
        logic [1:0] nine_en;

        always_ff @(posedge clk) begin
                //number1 upper left corner h=108, v=4
                if (hcount[10:1] >= 10'd108 && hcount[10:1] <= 10'd127 && vcount >= 5'd6
&& vcount <= 5'd25 && score_number1 == 9) begin
                        nine_en <= 2'b1;
                        nine_address <= hcount[10:1]-10'd108 + (vcount-5'd6)*20;
                end
                //number2 upper left corner h=128, v=4
                else if (hcount[10:1] >= 10'd128 && hcount[10:1] <= 10'd147 && vcount >=
5'd6 && vcount <= 5'd25 && score_number2 == 9) begin
                        nine_en <= 2'b1;
                        nine_address <= hcount[10:1]-10'd128 + (vcount-5'd6)*20;
                end
                //number3 upper left corner h=148, v=5
                else if (hcount[10:1] >= 10'd148 && hcount[10:1] <= 10'd167 && vcount >=
5'd6 && vcount <= 5'd25 && score_number3 == 9) begin
                        nine_en <= 2'b1;
                        nine_address <= hcount[10:1]-10'd148 + (vcount-5'd6)*20;
                end
                else begin
                        nine_en <= 2'b0;
                end
        end
```

```
	logic [10:0] score_address;
	logic [23:0] score_output;
	score score_unit(.address(score_address),.clock(clk),.q(score_output));
	logic [1:0] score_en;

	always_ff @(posedge clk) begin
		//score upper left corner h=25, v=6
		if (hcount[10:1] >= 10'd2 && hcount[10:1] <= 10'd101 && vcount >= 5'd6 &&
vcount <= 5'd25) begin
			score_en <= 2'b1;
			score_address <= hcount[10:1]-10'd2 + (vcount-10'd6)*100;
		end
		else begin
			score_en <= 2'b0;
		end
	end

	logic [10:0] heart_address;
	logic [23:0] heart_output;
	heart heart_unit(.address(heart_address),.clock(clk),.q(heart_output));
	logic [1:0] heart_en;
	logic [4:0] heart_v, heart_h;

	always_ff @(posedge clk) begin
		//heart1 upper left corner h=555, v=5
		if (hcount[10:1] >= 10'd555 && hcount[10:1] <= 10'd578 && vcount >= 5'd5
&& vcount <= 5'd26 && heart_status[0] == 1) begin
			heart_en <= 2'b1;
			heart_v <= vcount-5'd5;
			  heart_h <= hcount[10:1]-10'd555+1;
			if(heart_h==0)begin
				heart_v <= heart_v+1;
			end
			heart_address <= heart_h + heart_v*24;
		end
		//heart2 upper left corner h=585, v=5
		else if (hcount[10:1] >= 10'd585 && hcount[10:1] <= 10'd608 && vcount >=
5'd5 && vcount <= 5'd26 && heart_status[1] == 1) begin
			heart_en <= 2'b1;
			heart_v <= vcount-5'd5;
			  heart_h <= hcount[10:1]-10'd585+1;
			if(heart_h==0)begin
				heart_v <= heart_v+1;
			end
			heart_address <= heart_h + heart_v*24;
		end
		//heart3 upper left corner h=615, v=5
		else if (hcount[10:1] >= 10'd615 && hcount[10:1] <= 10'd638 && vcount >=
5'd5 && vcount <= 5'd26 && heart_status[2] == 1) begin
			heart_en <= 2'b1;
			heart_v <= vcount-5'd5;
			  heart_h <= hcount[10:1]-10'd615+1;
			if(heart_h==0)begin
				heart_v <= heart_v+1;
			end
			heart_address <= heart_h + heart_v*24;
		end
		else begin
			heart_en <= 2'b0;
		end
	end

	logic [10:0] brick_blue_address;
	logic [23:0] brick_blue_output;
	brick_blue
brick_blue_unit(.address(brick_blue_address),.clock(clk),.q(brick_blue_output));
	logic [1:0] brick_blue_en;

	always_ff @(posedge clk) begin
		//line 0
		/*if (hcount[10:7] == 4'd0 && vcount[9:5] == 5'd0 && brick_status0[0] ==
1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd1 && vcount[9:5] == 5'd0 && brick_status0[1]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd2 && vcount[9:5] == 5'd0 && brick_status0[2]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd3 && vcount[9:5] == 5'd0 && brick_status0[3]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd4 && vcount[9:5] == 5'd0 && brick_status0[4]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd5 && vcount[9:5] == 5'd0 && brick_status0[5]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd6 && vcount[9:5] == 5'd0 && brick_status0[6]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd7 && vcount[9:5] == 5'd0 && brick_status0[7]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd8 && vcount[9:5] == 5'd0 && brick_status0[8]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd9 && vcount[9:5] == 5'd0 && brick_status0[9]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end*/

                        //line 2
                        if (hcount[10:7] == 4'd0 && vcount[9:5] == 5'd2 && brick_status2[0] == 1)
begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd1 && vcount[9:5] == 5'd2 && brick_status2[1]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd2 && vcount[9:5] == 5'd2 && brick_status2[2]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
                        else if (hcount[10:7] == 4'd3 && vcount[9:5] == 5'd2 && brick_status2[3]
== 1) begin
                                brick_blue_en <= 2'b1;
                                brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                        end
```

```
                else if (hcount[10:7] == 4'd4 && vcount[9:5] == 5'd2 && brick_status2[4]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd5 && vcount[9:5] == 5'd2 && brick_status2[5]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd6 && vcount[9:5] == 5'd2 && brick_status2[6]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd7 && vcount[9:5] == 5'd2 && brick_status2[7]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd8 && vcount[9:5] == 5'd2 && brick_status2[8]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd9 && vcount[9:5] == 5'd2 && brick_status2[9]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                //line 4

                else if (hcount[10:7] == 4'd0 && vcount[9:5] == 5'd4 && brick_status4[0]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd1 && vcount[9:5] == 5'd4 && brick_status4[1]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd2 && vcount[9:5] == 5'd4 && brick_status4[2]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd3 && vcount[9:5] == 5'd4 && brick_status4[3]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd4 && vcount[9:5] == 5'd4 && brick_status4[4]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd5 && vcount[9:5] == 5'd4 && brick_status4[5]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd6 && vcount[9:5] == 5'd4 && brick_status4[6]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd7 && vcount[9:5] == 5'd4 && brick_status4[7]
== 1) begin
                        brick_blue_en <= 2'b1;
                        brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
```

```
		end
		else if (hcount[10:7] == 4'd8 && vcount[9:5] == 5'd4 && brick_status4[8]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd9 && vcount[9:5] == 5'd4 && brick_status4[9]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		//line 6
		else if (hcount[10:7] == 4'd0 && vcount[9:5] == 5'd6 && brick_status6[0]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd1 && vcount[9:5] == 5'd6 && brick_status6[1]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd2 && vcount[9:5] == 5'd6 && brick_status6[2]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd3 && vcount[9:5] == 5'd6 && brick_status6[3]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd4 && vcount[9:5] == 5'd6 && brick_status6[4]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd5 && vcount[9:5] == 5'd6 && brick_status6[5]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd6 && vcount[9:5] == 5'd6 && brick_status6[6]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd7 && vcount[9:5] == 5'd6 && brick_status6[7]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd8 && vcount[9:5] == 5'd6 && brick_status6[8]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd9 && vcount[9:5] == 5'd6 && brick_status6[9]
== 1) begin
			brick_blue_en <= 2'b1;
			brick_blue_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else begin
			brick_blue_en <= 2'b0;
		end
	end

	logic [10:0] brick_red_address;
	logic [23:0] brick_red_output;
	brick_red
brick_red_unit(.address(brick_red_address),.clock(clk),.q(brick_red_output));
```

```
	logic [1:0] brick_red_en;

	always_ff @(posedge clk) begin
		//line 1
		if (hcount[10:7] == 4'd0 && vcount[9:5] == 5'd1 && brick_status1[0] == 1)
begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd1 && vcount[9:5] == 5'd1 && brick_status1[1]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd2 && vcount[9:5] == 5'd1 && brick_status1[2]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd3 && vcount[9:5] == 5'd1 && brick_status1[3]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd4 && vcount[9:5] == 5'd1 && brick_status1[4]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd5 && vcount[9:5] == 5'd1 && brick_status1[5]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd6 && vcount[9:5] == 5'd1 && brick_status1[6]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd7 && vcount[9:5] == 5'd1 && brick_status1[7]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd8 && vcount[9:5] == 5'd1 && brick_status1[8]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd9 && vcount[9:5] == 5'd1 && brick_status1[9]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		//line 3
		else if (hcount[10:7] == 4'd0 && vcount[9:5] == 5'd3 && brick_status3[0]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd1 && vcount[9:5] == 5'd3 && brick_status3[1]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd2 && vcount[9:5] == 5'd3 && brick_status3[2]
== 1) begin
			brick_red_en <= 2'b1;
			brick_red_address <= hcount[6:1]+vcount[4:0]*64;
		end
		else if (hcount[10:7] == 4'd3 && vcount[9:5] == 5'd3 && brick_status3[3]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd4 && vcount[9:5] == 5'd3 && brick_status3[4]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd5 && vcount[9:5] == 5'd3 && brick_status3[5]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd6 && vcount[9:5] == 5'd3 && brick_status3[6]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd7 && vcount[9:5] == 5'd3 && brick_status3[7]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd8 && vcount[9:5] == 5'd3 && brick_status3[8]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd9 && vcount[9:5] == 5'd3 && brick_status3[9]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                //line 5
                else if (hcount[10:7] == 4'd0 && vcount[9:5] == 5'd5 && brick_status5[0]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd1 && vcount[9:5] == 5'd5 && brick_status5[1]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd2 && vcount[9:5] == 5'd5 && brick_status5[2]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd3 && vcount[9:5] == 5'd5 && brick_status5[3]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd4 && vcount[9:5] == 5'd5 && brick_status5[4]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd5 && vcount[9:5] == 5'd5 && brick_status5[5]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd6 && vcount[9:5] == 5'd5 && brick_status5[6]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd7 && vcount[9:5] == 5'd5 && brick_status5[7]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd8 && vcount[9:5] == 5'd5 && brick_status5[8]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else if (hcount[10:7] == 4'd9 && vcount[9:5] == 5'd5 && brick_status5[9]
== 1) begin
                        brick_red_en <= 2'b1;
                        brick_red_address <= hcount[6:1]+vcount[4:0]*64;
                end
                else begin
                        brick_red_en <= 2'b0;
                end
        end

        logic [10:0] ball_address;
        logic [23:0] ball_output;
        ball ball_unit(.address(ball_address),.clock(clk),.q(ball_output));
        logic [1:0] ball_en;
        logic [3:0] b_v;
        logic [3:0] b_h;
        always_ff @(posedge clk) begin
                //ball
                //if (hcount[10:5] == 6'd20 && vcount[9:4] == 6'd22) begin
                if (hcount[10:1] >= ball_h && hcount[10:1] < ball_h+16 && vcount[9:0] >=
ball_v && vcount[9:0]<ball_v+16)begin
                         b_v <= vcount[9:0]-ball_v;
                         b_h <= hcount[10:1]-ball_h +1;

                        if(b_h==0)begin
                                b_v <= b_v+1;
                        end
                        ball_address <= b_h + b_v*16;
                        ball_en <= 2'b1;
                end
                else begin
                        ball_en <= 2'b0;
                end
        end

        logic [10:0] paddle_address;
        logic [23:0] paddle_output;
        paddle paddle_unit(.address(paddle_address),.clock(clk),.q(paddle_output));
        logic [1:0] paddle_en;
        logic [4:0] r_v;
        logic [6:0] r_h;
        always_ff @(posedge clk) begin
                //paddle
                if (hcount[10:1] >= paddle_left && hcount[10:1] < paddle_left + 90 &&
vcount[9:0] >= 10'd455 && vcount[9:0] <= 10'd474) begin
                         r_v <= vcount[9:0]-10'd455;
                         r_h <= hcount[10:1]-paddle_left+1;
                         paddle_address <= r_h + r_v * 90;
                        paddle_en <= 2'b1;
                end
                else begin
                        paddle_en <= 2'b0;
                end
        end
        //broken_blue bric_unit(.address(brick_address),.clock(clk),.q(brick_output));

        always_comb begin
                {VGA_R, VGA_G, VGA_B} = {background_r, background_g, background_b};
                if (VGA_BLANK_n ) begin
                        if(heart_en) begin
                                {VGA_R, VGA_G, VGA_B} = heart_output;
                        end
                        else if(score_en) begin
                                {VGA_R, VGA_G, VGA_B} = score_output;
                        end
                        else if(zero_en) begin
                                {VGA_R, VGA_G, VGA_B} = zero_output;
                        end
                        else if(one_en) begin
                                {VGA_R, VGA_G, VGA_B} = one_output;
                        end
                        else if(two_en) begin
                                {VGA_R, VGA_G, VGA_B} = two_output;
                        end
                        else if(three_en) begin
                                {VGA_R, VGA_G, VGA_B} = three_output;
                        end
                        else if(four_en) begin
                                {VGA_R, VGA_G, VGA_B} = four_output;
                        end
                        else if(five_en) begin
                                {VGA_R, VGA_G, VGA_B} = five_output;
                        end
                        else if(six_en) begin
                                {VGA_R, VGA_G, VGA_B} = six_output;
                        end
                        else if(seven_en) begin
                                {VGA_R, VGA_G, VGA_B} = seven_output;
                        end
                        else if(eight_en) begin
                                {VGA_R, VGA_G, VGA_B} = eight_output;
                        end
                        else if(nine_en) begin
                                {VGA_R, VGA_G, VGA_B} = nine_output;
                        end

                        if(ball_en) begin
                                {VGA_R, VGA_G, VGA_B} = ball_output;
                        end

                        if(brick_blue_en) begin
                                {VGA_R, VGA_G, VGA_B} = brick_blue_output;
                        end
                        else if(brick_red_en) begin
                                {VGA_R, VGA_G, VGA_B} = brick_red_output;
                        end
                        else if(paddle_en) begin
                                {VGA_R, VGA_G, VGA_B} = paddle_output;
                        end
                        if(cry_en) begin
                                {VGA_R, VGA_G, VGA_B} = cry_output;
                        end
                        else if(smile_en) begin
                                {VGA_R, VGA_G, VGA_B} = smile_output;
                        end

                        if(hcount[3] == 1 && vcount == 27)begin
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                        end
                        if(hcount[3] == 0 && vcount == 28)begin
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                        end
                        if(hcount[3] == 1 && vcount == 29)begin
                                {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                        end

                end
        end

endmodule

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

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0             1279       1599 0
 *             _______________              ________
 * ___________|    Video      |____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
 *       _______________________      _____________
 * |____|       VGA_HS          |____|
 */
   // Parameters for hcount
   parameter HACTIVE      = 11'd 1280,
             HFRONT_PORCH = 11'd 32,
             HSYNC        = 11'd 192,
             HBACK_PORCH  = 11'd 96,
             HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                            HBACK_PORCH; // 1600,800

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

   logic endOfLine;

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

   assign endOfLine = hcount == HTOTAL - 1;

   logic endOfField;

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

   assign endOfField = vcount == VTOTAL - 1;

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

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

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

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

endmodule
```

//**

/***************vga_ball.h*******************/**

//**

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

typedef struct {
unsigned short red,
green,blue,paddle_left,ball_h,ball_v,brick_status0,brick_status1,brick_status2,brick_stat
us3,brick_status4,brick_status5,brick_status6,
audio_choose,heart_status,score1,score2,score3,gamestatus;
} hardware_p;


typedef struct {
  hardware_p alldata;
} vga_ball_arg_t;

#define VGA_BALL_MAGIC 'q'

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

#endif
```

//**

/***************vga_ball.c*******************/**

//**

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

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

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

#define DRIVER_NAME "vga_ball"

/* Device registers */
#define BG_RED(x) ((x))
#define BG_GREEN(x) ((x)+2)
#define BG_BLUE(x) ((x)+4)
#define PADDLE(x) ((x))
#define BALL_H(x) ((x)+2)
#define BALL_V(x) ((x)+4)
#define BRICK_STATUS0(x) ((x)+6)
#define BRICK_STATUS1(x) ((x)+8)
#define BRICK_STATUS2(x) ((x)+10)
#define BRICK_STATUS3(x) ((x)+12)
#define BRICK_STATUS4(x) ((x)+14)
#define BRICK_STATUS5(x) ((x)+16)
#define BRICK_STATUS6(x) ((x)+18)
#define HEART_STATUS(x)  ((x)+20)
#define AUDIO(x)         ((x)+22)
#define SCORE1(x)        ((x)+24)
#define SCORE2(x)        ((x)+26)
#define SCORE3(x)        ((x)+28)
#define GAMESTATUS(x)        ((x)+30)

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

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_alldata(hardware_p *alldata)
{
	/*iowrite16(alldata->red, BG_RED(dev.virtbase) );
	iowrite16(alldata->green, BG_GREEN(dev.virtbase) );
	iowrite16(alldata->blue, BG_BLUE(dev.virtbase) );*/
	iowrite16(alldata->paddle_left, PADDLE(dev.virtbase) );
	iowrite16(alldata->ball_h, BALL_H(dev.virtbase) );
        iowrite16(alldata->ball_v, BALL_V(dev.virtbase) );
        iowrite16(alldata->brick_status0, BRICK_STATUS0(dev.virtbase) );
        iowrite16(alldata->brick_status1, BRICK_STATUS1(dev.virtbase) );
        iowrite16(alldata->brick_status2, BRICK_STATUS2(dev.virtbase) );
        iowrite16(alldata->brick_status3, BRICK_STATUS3(dev.virtbase) );
        iowrite16(alldata->brick_status4, BRICK_STATUS4(dev.virtbase) );
        iowrite16(alldata->brick_status5, BRICK_STATUS5(dev.virtbase) );
        iowrite16(alldata->brick_status6, BRICK_STATUS6(dev.virtbase) );
        iowrite16(alldata->heart_status,  HEART_STATUS(dev.virtbase));
        iowrite16(alldata->audio_choose,  AUDIO(dev.virtbase));
        iowrite16(alldata->score1,  SCORE1(dev.virtbase));
        iowrite16(alldata->score2,  SCORE2(dev.virtbase));
        iowrite16(alldata->score3,  SCORE3(dev.virtbase));
        iowrite16(alldata->gamestatus,  GAMESTATUS(dev.virtbase));

	dev.alldata = *alldata;
```

```
}

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

	switch (cmd) {
	case VGA_BALL_WRITE_alldata:
		if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
				   sizeof(vga_ball_arg_t)))
			return -EACCES;
		write_alldata(&vla.alldata);
		break;

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

	default:
		return -EINVAL;
	}

	return 0;
}

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

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

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

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

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

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

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

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

        return 0;

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

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev)
{
        iounmap(dev.virtbase);
        release_mem_region(dev.res.start, resource_size(&dev.res));
        misc_deregister(&vga_ball_misc_device);
        return 0;
}

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

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

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

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

module_init(vga_ball_init);
module_exit(vga_ball_exit);

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


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

/***************hello.c*********************/**

/*/**

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

#define N0            0
#define N1            1
#define N2            2
#define N3            3
#define N4            4
#define N5            5
#define N6            6
#define N7            7
#define N8            8
#define N9            9
#define BRICK_W               64
#define BRICK_H               32
#define BALL_R        8
#define PADDLE_W      90
#define PADDLE_H      20
#define PADDLE_R      10
#define PADDLE_L      70
/* version with both mouse and keyboard control */

int                           vga_ball_fd;
double                        h_location;
struct libusb_device_handle   *keyboard;
uint8_t                       endpoint_address;
struct usb_keyboard_packet    packet;
int                           transferred;
char                          keystate[12];
int                           brick_status[7][10];
double                        speed_paddle, ball_h = 312.0, ball_v = 440.0, speed_h,
speed_v;
double                        t_speed;
int                           audio_choose;
int                           finalstatus = 0;

hardware_p            data = { 0, 0, 0, 0, 0 };
libusb_device_handle  *mouse;    /* a mouse device handle */
unsigned char         datain[1024] = "\0";
int                   size;
int                   reset          = 1;
int                   lives          = 3;
int                   game_start     = 0;
int                   level          = 1;
int                   score          = 0;

pthread_t      keyboard_thread;
pthread_t      mouse_thread;
void *keyboard_thread_f( void * );


void *mouse_thread_f( void * );


double absoluteDouble( double x )
{
	if ( x < 0 )
		return(0 - x);
	else
		return(x);
}


double square( double x )
{
	return(x * x);
}


int getscore1( int x )
{
	int y = (int) (x / 100);
	return(y);
}


int getscore2( int x )
{
	int y = (int) (x - 100 * getscore1( x ) ) / 10;
	return(y);
}


int getscore3( int x )
{
	int y = x % 10;
	return(y);
}


int lives2bin( int lives )
{
	if ( lives == 3 )
		return(7);
	else if ( lives == 2 )
		return(6);
	else if ( lives == 1 )
		return(4);
	else
		return(0);
}


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

	if ( ioctl( vga_ball_fd, VGA_BALL_READ_alldata, &vla ) )
	{
		perror( "ioctl(VGA_BALL_READ_alldata) failed" );
		return;
	}
	printf( "%02x %02x %02x\n",
		vla.alldata.red, vla.alldata.green, vla.alldata.blue );
}


/* fresh the display */
void set_alldata( const hardware_p *c )
{
	vga_ball_arg_t vla;
	vla.alldata = *c;
	if ( ioctl( vga_ball_fd, VGA_BALL_WRITE_alldata, &vla ) )
	{
		perror( "ioctl(VGA_BALL_SET_alldata) failed" );
		return;
```

```
        }
}


unsigned short convert2bin( int x[10] )
{
        unsigned short y      = 0;
        int            i      = 0, j = 0, k;
        for ( i = 0; i < 10; i++ )
        {
                k = 1;
                for ( j = 0; j < i; j++ )
                        k = k * 2;
                y = y + x[i] * k;
        }
        return(y);
}


/* newly developed function */
/* return flag = 0 if not hit on the brick */
/* return flag = 1 if hit on the brick from top or bottom */
/* return flag = 2 if hit on the brick from left or right */
/* return flag = 3 if hit on the brick corner*/
/* use the flag to determine the ball velocity change */
/* the corresponding brick status should be changed into 0 */
/* notice */
int hitOnBrick( int brick_width, int brick_height, int brick_h, int brick_v, int
ball_radius,
                double *p_ball_h, double *p_ball_v,
                double *p_corner_theta, int *p_audio_choose )
{
        double  n_ball_h       = *p_ball_h + BALL_R;
        double  n_ball_v       = *p_ball_v + BALL_R;
        double  n_corner_theta = *p_corner_theta;
        int     flag;
        int     n_audio_choose = *p_audio_choose;
        /* hit on top 1 */
        if ( n_ball_h >= brick_h && n_ball_h <= brick_h + brick_width
             && n_ball_v >= brick_v - ball_radius && n_ball_v < brick_v + brick_height )
        {
                flag           = 1;
                n_audio_choose = 2;
        }
        /* hit on bottom 2 */
        else if ( n_ball_h >= brick_h && n_ball_h <= brick_h + brick_width
                  && n_ball_v < brick_v + brick_height + ball_radius && n_ball_v >
brick_v )
        {
                flag           = 2;
                n_audio_choose = 2;
        }
        /* hit from left 3 */
        else if ( n_ball_v >= brick_v && n_ball_v <= brick_v + brick_height
                  && n_ball_h >= brick_h - ball_radius && n_ball_h < brick_h +
brick_width )
        {
                flag           = 3;
                n_audio_choose = 2;
        }
        /* hit from right 4 */
        else if ( n_ball_v >= brick_v && n_ball_v <= brick_v + brick_height
                  && n_ball_h < brick_h + brick_width + ball_radius && n_ball_h >
brick_h )
        {
                flag           = 4;
                n_audio_choose = 2;
        }
        /* corner right bottom 5 */
        else if ( square( n_ball_h - (brick_h + brick_width) ) + square( n_ball_v -
(brick_v + brick_height) ) < square( ball_radius ) )
```

```
	{
		flag           = 5;
		n_audio_choose = 2;
		n_corner_theta = atan( (brick_v + brick_height - n_ball_v) / (brick_h + 
brick_width - n_ball_h) );
	}
	/* corner right top   6 */
	else if ( square( n_ball_h - (brick_h + brick_width) ) + square( n_ball_v - 
brick_v ) < square( ball_radius ) )
	{
		flag           = 6;
		n_audio_choose = 2;
		n_corner_theta = atan( (brick_v - n_ball_v) / (brick_h + brick_width - 
n_ball_h) );
	}
	/* corner left top    7 */
	else if ( square( n_ball_h - brick_h ) + square( n_ball_v - brick_v ) < 
square( ball_radius ) )
	{
		flag           = 7;
		n_audio_choose = 2;
		n_corner_theta = atan( (brick_v - n_ball_v) / (brick_h - n_ball_h) );
	}
	/* corner left bottom  8 */
	else if ( square( n_ball_h - brick_h ) + square( n_ball_v - (brick_v + 
brick_height) ) < square( ball_radius ) )
	{
		flag           = 8;
		n_audio_choose = 2;
		n_corner_theta = atan( (brick_v + brick_height - n_ball_v) / (brick_h - 
n_ball_h) );
	}
	/* doesn't hit on the brick 0 */
	else{
		flag = 0;
	}
	*p_audio_choose = n_audio_choose;
	*p_corner_theta = n_corner_theta;

	return(flag);
}


void hitOnWall( double *p_ball_h, double *p_ball_v, double *p_speed_h, double *p_speed_v, 
int *p_audio_choose )
{
	double  n_ball_h       = *p_ball_h + BALL_R;
	double  n_ball_v       = *p_ball_v + BALL_R;
	double  n_speed_h      = *p_speed_h;
	double  n_speed_v      = *p_speed_v;
	int     n_audio_choose = *p_audio_choose;
	/* hit from top */
	if ( n_ball_v + n_speed_v <= BALL_R + BRICK_H )
	{
		n_speed_v      = 0 - n_speed_v;
		n_audio_choose = 1;
	}

	if ( n_ball_h + BALL_R + n_speed_h >= 640 || n_ball_h + n_speed_h <= BALL_R )
	{
		n_speed_h      = 0 - n_speed_h;
		n_audio_choose = 1;
	}


	*p_ball_h      = n_ball_h - BALL_R;
	*p_ball_v      = n_ball_v - BALL_R;
	*p_speed_h     = n_speed_h;
	*p_speed_v     = n_speed_v;
	*p_audio_choose = n_audio_choose;
}
```

```
int hitOnPaddle( double *p_ball_h, double *p_ball_v, double *p_speed_v, int
*p_audio_choose,
                double *p_paddle_position, double *p_paddle_speed, int *p_game_status,
double *p_corner_theta )
{
       double  n_ball_h       = *p_ball_h + BALL_R;
       double  n_ball_v       = *p_ball_v + BALL_R;
       double  n_paddle_speed = *p_paddle_speed;
       double  n_paddle_x0    = *p_paddle_position + n_paddle_speed;
       double  n_paddle_y0    = 455.0;
       double  n_speed_v      = *p_speed_v;
       double  n_corner_theta = *p_corner_theta;
       int     n_audio_choose = *p_audio_choose;
       int     n_game_status  = *p_game_status;
       int     flag;

       /* center of the left half circle */
       double  n_paddle_x1    = n_paddle_x0 + PADDLE_R;
       double  n_paddle_y1    = n_paddle_y0 + PADDLE_R;

       /* center of the right half circle */
       double  n_paddle_x2    = n_paddle_x1 + PADDLE_L;
       double  n_paddle_y2    = n_paddle_y1;


       /*
        * same as hit on brick, return the ball status flag
        * hit from the top
        */
       if ( speed_v > 0 && n_ball_v <= 474 - 0.5 * PADDLE_H )
       {
              if ( n_ball_h >= n_paddle_x1 && n_ball_h <= n_paddle_x2 && n_ball_v >=
n_paddle_y0 - BALL_R && n_ball_v <= n_paddle_y0 )
              {
                     flag           = 9;
                     n_audio_choose = 1;
              }else if ( square( n_ball_h - n_paddle_x1 ) + square( n_ball_v -
n_paddle_y1 ) < square( BALL_R + PADDLE_R )
                        && square( n_ball_h - n_paddle_x1 ) + square( n_ball_v -
n_paddle_y1 ) > square( PADDLE_R ) )
              {
                     flag           = 10;
                     n_corner_theta = atan( (n_paddle_y1 - n_ball_v) / (n_paddle_x1 -
n_ball_h) );
                     n_audio_choose = 1;
              }else if ( square( n_ball_h - n_paddle_x2 ) + square( n_ball_v -
n_paddle_y2 ) < square( BALL_R + PADDLE_R )
                        && square( n_ball_h - n_paddle_x1 ) + square( n_ball_v -
n_paddle_y1 ) > square( PADDLE_R ) )
              {
                     flag           = 11;
                     n_corner_theta = atan( (n_paddle_y2 - n_ball_v) / (n_paddle_x2 -
n_ball_h) );
                     n_audio_choose = 1;
              }
       }


       if ( n_ball_v >= 488 )
       {
              lives          -= 1;
              game_start     = 0;
              printf( "%d lives left!\n", lives );


              if ( lives == 0 )
              {
                     n_game_status  = 0;
                     finalstatus    = 2;
                printf( "GAME OVER\n" );
            }else  {
                reset = 1;
            }
        }

        *p_ball_h      = n_ball_h - BALL_R;
        *p_ball_v      = n_ball_v - BALL_R;

        *p_audio_choose = n_audio_choose;
        *p_game_status = n_game_status;
        *p_corner_theta = n_corner_theta;

        return(flag);
}


/* according to flag change the velocity of ball and also the brick status */
void changeVelocityAndBrickStatus( int flag, double *p_speed_h, double *p_speed_v, int
*brick_status, double *p_corner_theta )
{
        double  n_corner_theta = *p_corner_theta;
        double  n_speed_h      = *p_speed_h;
        double  n_speed_v      = *p_speed_v;
        double  n_speed_h_temp;
        double  n_speed_v_temp;
        /* hit from top */
        if ( flag == 1 )
        {
                if ( n_speed_v > 0 )
                        n_speed_v = 0 - n_speed_v;
                *brick_status = 0;
                printf( "changed vertical velocity\n" );
        }
        /* hit from bottom */
        else if ( flag == 2 )
        {
                if ( n_speed_v < 0 )
                        n_speed_v = 0 - n_speed_v;
                *brick_status = 0;
                printf( "changed vertical velocity\n" );
        }
        /* hit from left */
        else if ( flag == 3 )
        {
                if ( n_speed_h > 0 )
                        n_speed_h = 0 - n_speed_h;
                *brick_status = 0;
                printf( "changed horizontal velocity\n" );
        }
        /* hit from right */
        else if ( flag == 4 )
        {
                if ( n_speed_h < 0 )
                        n_speed_h = 0 - n_speed_h;
                *brick_status = 0;
                printf( "changed horizontal velocity\n" );
        }else if ( flag == 5 || flag == 6 || flag == 7 || flag == 8 )
        {
                n_speed_h_temp = n_speed_h;
                n_speed_v_temp = n_speed_v;
                printf( "original speed_h is %f, original speed_v is %f\n", n_speed_h,
n_speed_v );
                n_speed_h      = 0.0 - cos( 2.0 * n_corner_theta ) * n_speed_h_temp -
sin( 2.0 * n_corner_theta ) * n_speed_v_temp;
                n_speed_v      = 0.0 - sin( 2.0 * n_corner_theta ) * n_speed_h_temp +
cos( 2.0 * n_corner_theta ) * n_speed_v_temp;
                *brick_status  = 0;
                printf( "hit on the corner\n angle is %f\n", n_corner_theta );
                printf( "speed_h is %f, speed_v is %f\n", n_speed_h, n_speed_v );
        }
        *p_corner_theta = n_corner_theta;
        *p_speed_h      = n_speed_h;
        *p_speed_v      = n_speed_v;
}


/* change velocity after hitting on the paddle */
void changeVelocityHitPaddle( int flag, double *p_speed_h, double *p_speed_v, double
*p_speed_paddle, double *p_corner_theta )
{
        double  n_corner_theta = *p_corner_theta;
        double  n_speed_h      = *p_speed_h;
        double  n_speed_v      = *p_speed_v;
        double  n_speed_h_temp;
        double  n_speed_v_temp;
        double  n_speed_paddle = *p_speed_paddle;

        if ( flag == 9 )
        {
                if ( n_speed_v > 0 )
                        n_speed_v = 0 - n_speed_v;
                printf( "detect hit on paddle top\n" );
        }else if ( flag == 10 || flag == 11 )
        {
                n_speed_h_temp = n_speed_h;
                n_speed_v_temp = n_speed_v;
                printf( "detect hit on paddle side\n" );
                printf( "original speed_h is %f, original speed_v is %f\n", n_speed_h,
n_speed_v );
                n_speed_h      = 0.0 - cos( 2.0 * n_corner_theta ) * n_speed_h_temp -
sin( 2.0 * n_corner_theta ) * n_speed_v_temp;
                n_speed_v      = 0.0 - sin( 2.0 * n_corner_theta ) * n_speed_h_temp +
cos( 2.0 * n_corner_theta ) * n_speed_v_temp;
                printf( "hit on the corner\n angle is %f\n", n_corner_theta );
                printf( "speed_h is %f, speed_v is %f\n", n_speed_h, n_speed_v );
        }

        *p_corner_theta = n_corner_theta;
        *p_speed_h      = n_speed_h;
        *p_speed_v      = n_speed_v;
}


int levelPass( int n_brick_status[7][10] )
{
        int     a, b;
        int     temp = 1; /* pass the level if it returns 1 */
        for ( a = 0; a < 7; a++ )
                for ( b = 0; b < 10; b++ )
                {
                        if ( n_brick_status[a][b] != 0 )
                        {
                                temp = 0;
                                break;
                        }
                }
        return(temp);
}


void velocityConstrain( double *p_speed_h, double *p_speed_v, double constrainSpeedV )
{
        double  n_speed_h      = *p_speed_h;
        double  n_speed_v      = *p_speed_v;
        double  n_speed        = sqrt( square( n_speed_h ) + square( n_speed_v ) );
        if ( absoluteDouble( n_speed_v ) < constrainSpeedV )
        {
                if ( n_speed_v < 0 && n_speed_h < 0 )
                {
                        n_speed_v      = 0 - constrainSpeedV;
                        n_speed_h      = 0 - sqrt( square( n_speed ) -
square( n_speed_v ) );
                }else if ( n_speed_v < 0 && n_speed_h >= 0 )
                {
                        n_speed_v      = 0 - constrainSpeedV;
                        n_speed_h      = sqrt( square( n_speed ) - square( n_speed_v ) );
                }else if ( n_speed_v >= 0 && n_speed_h < 0 )
                {
                        n_speed_v      = constrainSpeedV;
                        n_speed_h      = 0 - sqrt( square( n_speed ) -
square( n_speed_v ) );
                }else  {
                        n_speed_v      = constrainSpeedV;
                        n_speed_h      = sqrt( square( n_speed ) - square( n_speed_v ) );
                }
        }
        *p_speed_h     = n_speed_h;
        *p_speed_v     = n_speed_v;
}


int main()
{
        vga_ball_arg_t vla;
        int            err;
        double         corner_theta;
        int            brick_width    = BRICK_W;
        int            brick_height   = BRICK_H;
        int            ball_radius    = BALL_R;
        int            paddle_length  = PADDLE_W;
        int            brick_h, brick_v;
        int            game_status;
        int            flag[7][10];
        int            count = 0;

        int     flag_paddle    = 0;
        int     level_reset    = 1;

        /* initialize */


        speed_paddle   = 0;
        ball_h         = 312;
        ball_v         = 440;
        speed_h        = 0.5;
        speed_v        = -0.2;
        h_location     = 275;
        game_status    = 1;


        /* Open the keyboard */
        if ( (keyboard = openkeyboard( &endpoint_address ) ) == NULL )
        {
                fprintf( stderr, "Did not find a keyboard\n" );
                exit( 1 );
        }
        libusb_device  **devs;                                          /* pointer
to pointer of device, used to retrieve a list of devices */
        libusb_context *ctx = NULL;                                     /* a libusb
session */
        int            r;                                               /* for
return values */
        ssize_t        cnt;                                             /* holding
number of devices in list */
        r = libusb_init( &ctx );                                         /*
initialize a library session */
        if ( r < 0 )
        {
                printf( "%s  %d\n", "Init Error", r );                   /* there
was an error */
```

```
                return(1);
        }
        libusb_set_debug( ctx, 3 );                                             /* set
verbosity level to 3, as suggested in the documentation */
        cnt = libusb_get_device_list( ctx, &devs );                             /* get the
list of devices */
        if ( cnt < 0 )
        {
                printf( "%s\n", "Get Device Error" );                           /* there
was an error */
        }

        mouse = libusb_open_device_with_vid_pid( ctx, 16700, 12314 );           /* open
mouse */
        if ( mouse == NULL )
        {
                printf( "%s\n", "Cannot open device" );
                libusb_free_device_list( devs, 1 );                             /* free
the list, unref the devices in it */
                libusb_exit( ctx );                                             /* close
the session */
                return(0);
        }else  {
                printf( "%s\n", "Device opened" );
                libusb_free_device_list( devs, 1 );                             /* free
the list, unref the devices in it */
                if ( libusb_kernel_driver_active( mouse, 0 ) == 1 )             /* find
out if kernel driver is attached */
                {
                        printf( "%s\n", "Kernel Driver Active" );
                        if ( libusb_detach_kernel_driver( mouse, 0 ) == 0 )     /* detach
it */
                                printf( "%s\n", "Kernel Driver Detached!" );
                }
                r = libusb_claim_interface( mouse, 0 );                         /* claim
interface 0 (the first) of device (mine had just 1) */
                if ( r < 0 )
                {
                        printf( "%s\n", "Cannot Claim Interface" );
                        return(1);
                }
        }
        printf( "%s\n", "Claimed Interface" );

        pthread_create( &keyboard_thread, NULL, keyboard_thread_f, NULL );
        pthread_create( &mouse_thread, NULL, mouse_thread_f, NULL );

        /*
         * 0 bgm;
         * 1 bgm + hit_wall
         * 2 bgm + hit_brick
         */
        data.audio_choose = 0;


        /* filename corresponding to vga_ball */
        static const char filename[] = "/dev/vga_ball";


        if ( (vga_ball_fd = open( filename, O_RDWR ) ) == -1 ) /* opened some file */
        {
                fprintf( stderr, "could not open %s\n", filename );
                return(-1);
        }
        while ( 1 )
        {
                level_reset    = 1;
                level          = 1;
                game_status    = 1;
                lives          = 3;
                finalstatus    = 0;
```

```
            score          = 0;


            /* game start */
            while ( game_status )
            {
                    /* choose level */
                    if ( level_reset == 1 )
                    {
                            /* default status of the bricks are 0 */
                            for ( int i = 0; i < 7; i++ )
                                    for ( int j = 0; j < 10; j++ )
                                    {
                                            brick_status[i][j] = 0;
                                    }


                            /*
                             * the status of bricks can be initialized to generate
different levels of game
                             * Don't display any bricks in the first row
                             */
                            for ( int i = 3; i <= 5; i++ )
                                    for ( int j = 2; j <= 7; j++ )
                                            brick_status[i][j] = 1;
                            reset          = 1;
                            level_reset    = 0;
                    }

                    if ( level_reset == 2 )
                    {
                            /* default status of the bricks are 0 */
                            for ( int i = 0; i < 7; i++ )
                                    for ( int j = 0; j < 10; j++ )
                                    {
                                            brick_status[i][j] = 0;
                                    }

                            for ( int i = 2; i <= 6; i++ )
                                    for ( int j = 1; j <= 8; j++ )
                                            brick_status[i][j] = 1;
                            reset          = 1;
                            level_reset    = 0;
                    }

                    /* when player die or pass through a level, reset is set to 1 */
                    if ( reset )
                    {
                            /* if the player don't press on keyboard or mouse, the game
doesn't start */
                            while ( !game_start )
                            {
                                    ;
                            }
                            printf( "RESET\n" );
                            speed_paddle   = 0;
                            ball_h         = 312;
                            ball_v         = 440;
                            /* different level has different speed */
                            if ( level == 1 )
                            {
                                    srand( (unsigned) time( NULL ) );
                                    int     seed                 = rand() % 6;
                                    double  speed_h_range[6]      = { 0.4, -0.4, 0.3, -
0.3, 0.2, -0.2 };
                                    double  speed_v_range[6]      = { 0.2, -0.2, -0.33,
-0.33, -0.4, -0.4 };


                                    speed_h = speed_h_range[seed];
                                    speed_v = speed_v_range[seed];
```

```
                }else if ( level == 2 )
                {
                        srand( (unsigned) time( NULL ) );
                        int     seed                    = rand() % 6;
                        double  speed_h_range[6]        = { 0.4, -0.4, 0.3, -
0.3, 0.2, -0.2 };
                        double  speed_v_range[6]        = { 0.2, -0.2, -0.33,
-0.33, -0.4, -0.4 };


                        speed_h = speed_h_range[seed] * 1.2;
                        speed_v = speed_v_range[seed] * 1.2;
                }

                h_location      = 275;
                game_status     = 1;
                reset           = 0;
        }


        /*
         * set_alldata_color(&colors[i % COLORS ]);
         * print_alldata_color();
         */
        audio_choose    = 0;
        count           = 0;
        /*
         * if at corner, the paddle will not move at all
         * else the speed of paddle is set to a constant number
         */

        /* Don't display any bricks in the first row */

        flag_paddle = 0;

        for ( int i = 0; i < 7; i++ )
                for ( int j = 0; j < 10; j++ )
                        flag[i][j] = 0;

        if ( h_location + t_speed < 0 )
                h_location = 0;
        else if ( h_location + t_speed + PADDLE_W >= 639 )
                h_location = 639 - PADDLE_W;
        else
                h_location += t_speed;

        /*
         * determine whether hit on the brick
         * i represents line
         * j represents column
         */
        for ( int i = 0; i < 7; i++ )
                for ( int j = 0; j < 10; j++ )
                        if ( brick_status[i][j] == 1 )
                        {
                                brick_h         = brick_width * j;
                                brick_v         = brick_height * i;
                                flag[i][j]      = hitOnBrick( brick_width,
brick_height, brick_h, brick_v, ball_radius, &ball_h, &ball_v, &corner_theta,
&audio_choose );
                                if ( flag[i][j] != 0 )
                                {
                                        score   += (1 - i % 2) + 1;
                                        count   += 1;
                                }
                        }

        /* if only one brick get hit */
        if ( count == 1 )
        {
                for ( int i = 0; i < 7; i++ )
```

```
                              for ( int j = 0; j < 10; j++ )
                                     changeVelocityAndBrickStatus( flag[i][j],
&speed_h, &speed_v, &brick_status[i][j], &corner_theta );
                      }
                      /* if more than 1 brick get hit, we don't use corner one */
                      else if ( count > 1 )
                      {
                             for ( int i = 0; i < 7; i++ )
                                    for ( int j = 0; j < 10; j++ )
                                           if ( flag[i][j] == 1 || flag[i][j] == 2 ||
flag[i][j] == 3 || flag[i][j] == 4 )

       changeVelocityAndBrickStatus( flag[i][j], &speed_h, &speed_v, &brick_status[i][j],
&corner_theta );
                      }


                      /*
                       * refresh the score now
                       * if one brick get hit, score +3, if 2 get hit in a single loop +9
and so on
                       */


                      /* determine whether hit on the left and right and top wall and
change the velocity */
                      hitOnWall( &ball_h, &ball_v, &speed_h, &speed_v, &audio_choose );

                      /* determine whether paddle catches the ball and give the game
status */
                      flag_paddle = hitOnPaddle( &ball_h, &ball_v, &speed_v,
&audio_choose, &h_location, &speed_paddle, &game_status, &corner_theta );

                      changeVelocityHitPaddle( flag_paddle, &speed_h, &speed_v,
&speed_paddle, &corner_theta );

                      /* in case the situation a very small vertical velocity is
generated. */
                      velocityConstrain( &speed_h, &speed_v, 0.02 );

                      ball_h  += speed_h;
                      ball_v  += speed_v;


                      /* check whether level is finished */
                      if ( levelPass( brick_status ) )
                      {
                             level          += 1;
                             game_start     = 0;
                             if ( level > 2 )
                             {
                                    finalstatus    = 1;
                                    game_status    = 0;
                                    printf( "GAME FINISHED\n" );
                                    data.gamestatus = finalstatus;
                             }
                             level_reset = level;
                      }

                      /* //////////// WRITE DATA ////////////// */


                      data.paddle_left      = (int) h_location;
                      data.ball_h           = (int) ball_h;
                      data.ball_v           = (int) ball_v;
                      data.brick_status0    = convert2bin( brick_status[0] );
                      data.brick_status1    = convert2bin( brick_status[1] );
                      data.brick_status2    = convert2bin( brick_status[2] );
                      data.brick_status3    = convert2bin( brick_status[3] );
                      data.brick_status4    = convert2bin( brick_status[4] );
                      data.brick_status5    = convert2bin( brick_status[5] );
```

```
                        data.brick_status6     = convert2bin( brick_status[6] );
                        data.audio_choose      = audio_choose;
                        data.heart_status      = lives2bin( lives );
                        data.score1            = getscore1( score );
                        data.score2            = getscore2( score );
                        data.score3            = getscore3( score );
                        data.gamestatus                = finalstatus;

                        set_alldata( &data );


                        usleep( 1200 );
                        if ( game_status == 0 )
                        {
                                usleep( 1200000 );
                                break;
                        }
                }

                usleep( 3000 );
        }

        /* Terminate the blink thread */
        pthread_cancel( keyboard_thread );
        pthread_cancel( mouse_thread );
        /* Wait for the network thread to finish */
        pthread_join( keyboard_thread, NULL );
        pthread_join( mouse_thread, NULL );
        return(0);
}


void *keyboard_thread_f( void *ignored )
{
        printf( "thread started\n" );

        vga_ball_arg_t vla;


        while ( 1 )
        {
                libusb_interrupt_transfer( keyboard, endpoint_address,
                                          (unsigned char *) &packet, sizeof(packet),
                                          &transferred, 0 );

                if ( transferred == sizeof(packet) )
                {
                        sprintf( keystate, "%02x %02x %02x", packet.modifiers,
packet.keycode[0],
                                 packet.keycode[1] );


                        if ( packet.keycode[0] == 0x50 )        /* LEFTARROW Pressed */

                        {
                                t_speed = -0.5;
                        }else if ( packet.keycode[0] == 0x4f )  /* RIGHTARROW Pressed */
                        {
                                t_speed = 0.5;
                        }else if ( packet.keycode[0] == 0x28 )
                        {
                                game_start = 1;
                        }else  {
                                /* printf( "else\n" ); */
                                t_speed = 0;
                        }
                }else  {
                        printf( "else\n" );
                        t_speed = 0;
                }
```

```
		if ( h_location + t_speed < 0 )
			h_location = 0;
		else if ( h_location + t_speed + PADDLE_W >= 639 )
			h_location = 639 - PADDLE_W;
		else
			h_location += t_speed;

/* /////////// WRITE DATA ///////////// */

		data.audio_choose      = audio_choose;
		data.paddle_left       = (int) h_location;
		data.ball_h            = (int) ball_h;
		data.ball_v            = (int) ball_v;
		data.brick_status0     = convert2bin( brick_status[0] );
		data.brick_status1     = convert2bin( brick_status[1] );
		data.brick_status2     = convert2bin( brick_status[2] );
		data.brick_status3     = convert2bin( brick_status[3] );
		data.brick_status4     = convert2bin( brick_status[4] );
		data.brick_status5     = convert2bin( brick_status[5] );
		data.brick_status6     = convert2bin( brick_status[6] );
		data.heart_status      = lives2bin( lives );
		data.score1            = getscore1( score );
		data.score2            = getscore2( score );
		data.score3            = getscore3( score );
		data.gamestatus                = finalstatus;

		set_alldata( &data );
	}
}


void *mouse_thread_f( void *ignored )
{
	printf( "mouse thread started\n" );

	vga_ball_arg_t vla;


	while ( 1 )
	{
		libusb_interrupt_transfer( mouse, 0x81,
					  datain, 0x0004,
					  &size, 0 );
		printf( "transferred\n" );

		if ( datain[3] == 0x01 )	/* mouse */
		{
			t_speed = -0.5;
		}else if ( datain[3] == 0xff )	/* mouse */
		{
			t_speed = 0.5;
		}else  {
			printf( "else\n" );
			t_speed = 0;
		}

		if ( datain[1] > 0 && datain[1] < 0x2f )		/* mouse */
		{
			t_speed = 0;
		}else if ( datain[1] > 0x30 && datain[1] < 0x5f )	/* mouse */
		{
			t_speed = 0.8;
		}else if ( datain[1] > 0x60 && datain[1] < 0x7f )	/* mouse */
		{
			t_speed = 1.8;
		}else if ( datain[1] > 0x80 && datain[1] < 0xaf )	/* mouse */
		{
			t_speed = -1.8;
		}else if ( datain[1] > 0xb0 && datain[1] < 0xcf )	/* mouse */
		{
			t_speed = -0.8;
		}else if ( datain[1] > 0xd0 && datain[1] < 0xff )	/* mouse */
		{
			t_speed = 0;
		}


		h_location += t_speed;
		/* //////////// WRITE DATA ////////////// */
		data.audio_choose	= audio_choose;
		data.paddle_left	= (int) h_location;
		data.ball_h		= (int) ball_h;
		data.ball_v		= (int) ball_v;
		data.brick_status0	= convert2bin( brick_status[0] );
		data.brick_status1	= convert2bin( brick_status[1] );
		data.brick_status2	= convert2bin( brick_status[2] );
		data.brick_status3	= convert2bin( brick_status[3] );
		data.brick_status4	= convert2bin( brick_status[4] );
		data.brick_status5	= convert2bin( brick_status[5] );
		data.brick_status6	= convert2bin( brick_status[6] );
		data.score1		= getscore1( score );
		data.score2		= getscore2( score );
		data.score3		= getscore3( score );
		data.gamestatus			= finalstatus;

		set_alldata( &data );
	}
}
```

/*/**

/***************tonegen.sv********************/**

/*/**

```
// we generate a simple square wave here
module tonegen(
	input clk,	// 50MHz
	input reset,
	input left_chan_ready,
	input right_chan_ready,
	 input  logic [1:0]  audio_choose,
	output logic [15:0] sample_data,  // output data, to fit the data each time can be
transferred 16 bits
	output logic sample_valid

);

// our testing sampling rate is 8kHz
logic [2:0] counter1;	// change 48kHz to 8kHz, divide by 6
logic [16:0] bg_address;
logic [15:0] bg_data;
logic [10:0] hit_wall_address;
logic [15:0] hit_wall_data;
logic [11:0] hit_brick_address;
logic [15:0] hit_brick_data;
logic		flag;
logic		flag2;



// instantiate the memory for back ground music
bgm bgm0(.address(bg_address),.clock(clk),.q(bg_data));

// instantiate the memory for hit_wall sound
hit_wall hit_wall0(.address(hit_wall_address),.clock(clk),.q(hit_wall_data));

// instantiate the memory for hit_brick sound
hit_brick hit_brick0(.address(hit_brick_address),.clock(clk),.q(hit_brick_data));
```

```
// keep playing background music
always_ff @(posedge clk) begin
   if (reset) begin
      counter1 <= 3'b0;
      bg_address <= 0;
      sample_data <= (bg_data );
      sample_valid <= 0;
      flag <= 1'b1;
      flag2<= 1'b1;
      hit_brick_address <= 0;
      hit_wall_address <= 0;
   end else if (counter1 < 3'd6 && (left_chan_ready && right_chan_ready)) begin
      counter1 <= counter1 + 1;
      sample_valid <= 1'b0;
   end else if (counter1 == 3'd6 && (left_chan_ready && right_chan_ready)) begin
      if (audio_choose == 2'b00 && flag == 1'b1 && flag2 == 1'b1) begin
         if (bg_address < 17'd121593) begin
            bg_address <= bg_address + 1;
         end else begin
            bg_address <= 0;
         end
         sample_data <= (bg_data);
      end else if (audio_choose == 2'b01 || flag == 1'b0) begin // if hit the wall
         if (hit_wall_address < 11'd1815 && bg_address < 17'd121593) begin
            hit_wall_address <= hit_wall_address + 1;
            bg_address <= bg_address + 1;
            flag <= 1'b0;
         end else if (bg_address == 17'd121593) begin
           bg_address <= 0;
            flag <= 1'b0;
         end else if (hit_wall_address == 11'd1815) begin
            hit_wall_address <= 0;
            flag <= 1'b1;
         end
         sample_data <= (hit_wall_data) + (bg_data);
      end else if (audio_choose == 2'b10 || flag2 == 1'b0) begin // if hit the brick
         if (hit_brick_address < 12'd2869 && bg_address < 17'd121593) begin
            hit_brick_address <= hit_brick_address + 1;
            bg_address <= bg_address + 1;
            flag2 <= 1'b0;
         end else if (bg_address == 17'd121593) begin
           bg_address <= 0;
            flag2 <= 1'b0;
         end else if (hit_brick_address == 12'd2869) begin
            hit_brick_address <= 0;
            flag2 <= 1'b1;
         end
         sample_data <= (hit_brick_data) + (bg_data);
      end
      counter1 <= 0;
      sample_valid = 1'b1;
   end
end







endmodule
```