DLD PROJECT REPORT

Nimra Sohail (06867), Razi Haider (06882), Taimoor Hameed (06920), Daniyal Murtaza (06880)

16 December 2021

1 Motivation

The aim of this project is to develop the masterpiece game of nineties "Pong Game" that will be displayed on the monitor's screen. The project will utilize the Verilog as the programming language and the Basys 3 FPGA will be used to carry out the coding and transferring the image using a Video Graphics Array (VGA) interface.

1.1 Overview

The report covers the following deliverable:

1. User Experience with Flow Diagram.
2. Input Peripherals.
3. Control Block.
4. Output Block.
5. All FSM Designs.
6. An FSM implemented on gate level.
7. Source Code of the Game.

2 User Experience

Our Project aims to provide a very friendly experience to the user. That is why it is necessary to establish a relation theoretically and mathematically that will be helpful in predicting this experience. Therefore, the user flow diagram which provides us perspective of the user for our project is given below:

2.1 User Flow Diagram

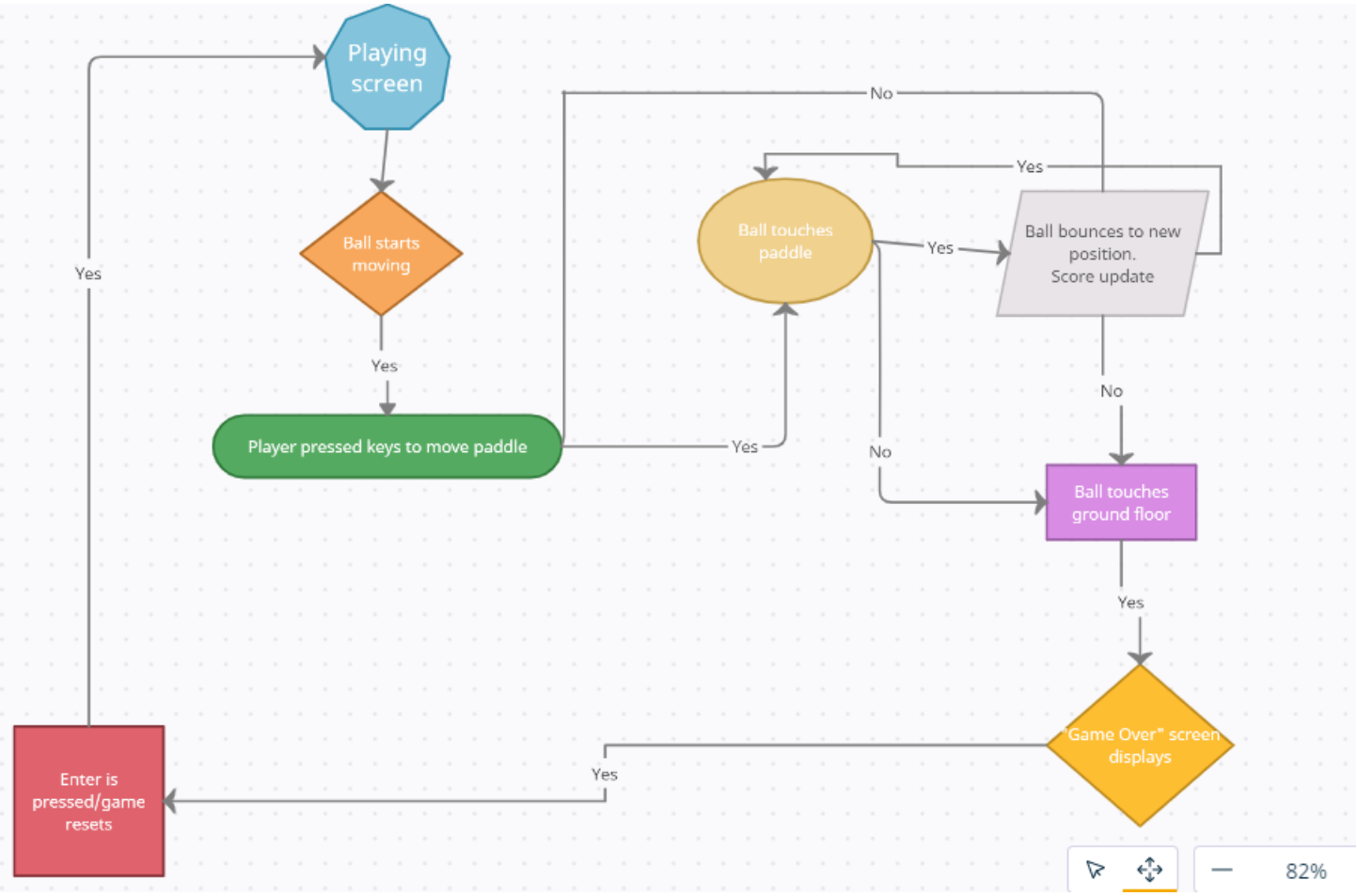


Figure 1: User Flow Diagram.

The User Flow Diagram basically consists of 3 states amalgamated into a single state given in Figure 1. These states are basically Reset State, Play state and lose state that defines the whole functionality of our project which will be elaborated in greater details in the Control Block of our project. Now, first we will define the Input Block then Control block and lastly Output to be able to completely understand the theory of our project.

3 Input Block

To understand how we implemented the inputs we needed to run our game using a keyboard we need to know how the Basys3's USB-HID and USB-UART ports work. When the user presses a key on the keyboard, this sends a keyboard PS/2 scan code to the Basys3 over the USB-HID port. The figure given below shows the keys and their respective codes:

Figure 2: PS-2 Keys with Codes.

This scan code is read and sent to a terminal application via the USB-UART Bridge. When the key is released, a scan code of F0XX is transmitted, indicating that the key with PS/2 code XX has been released. Our game uses the 'Enter' key to reset the game and 'A' key to make paddle move towards left and 'D' key to make paddle move towards right.

4 Control Block

The control block has been divided into Reset, Play and Lose States. The description of these states are given below:

1. Reset State
   This is the initial state of the game. The ball will start moving from a certain location.

2. Play State
   The game will start as soon as Hardware is programmed using FPGA. The player will be able to move the paddle using the "A" and "D" arrow keys on the keyboard. Each time the ball collides with the paddle, the player score increments by 1 on the screen. This state will continue unless the reset state or lose state is triggered.

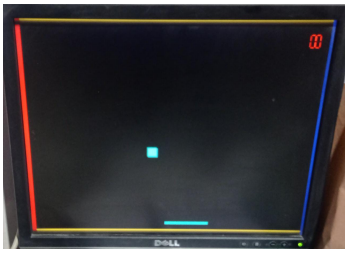


Figure 3: Play screen.

3. Lose State
   The game will enter this state only when the ball collides with the bottom wall. "GAME OVER" screen will appear at this stage.

Figure 4: "GAME OVER" screen

4.1 Controllers Description

All inputs from the keyboard will be taken using the ps2 receiver emulator on the basys 3 board. The keys "A" and "D" will make the paddle move left or right. The movement of the ball will be controlled by another FSM. State transitions will depend on the current state of the ball as well as the particular wall it collides with."ENTER" will reset the game.

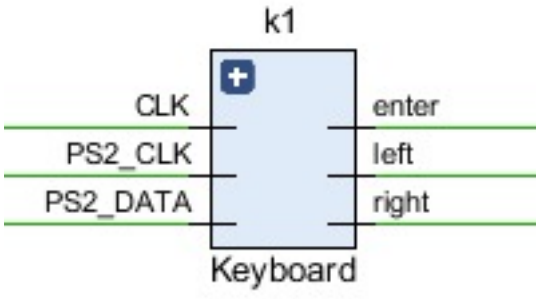


Figure 5: Input and Control Peripheral Block

4.2 Pin Configuration

Clk input gets connected to W5 pin of FPGA, PS2 CLK gets connected to c17 pin and ps2 data gets connected to b17 pin of FPGA. The keyboard is connected to FPGA board via USB host connector of FPGA board.

5 Output Block

We'll be using a VGA controller to drive a frame from our game onto a monitor. The figure below shows the VGA port connections:

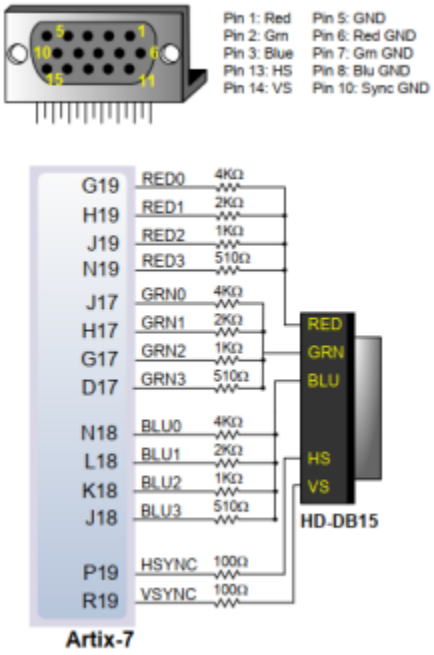


Figure 6: VGA port Connections

To create a VGA controller, we generated HSYNC and VSYNC signals from the Pixel clock. The timing of HSYNC and VSYNC signals depends on number of parameters: Horizontal and Vertical Frontporch, Horizontal and Vertical Backporch, Horizontal and Vertical Display Pixels, Horizontal and Vertical Sync Pulse Widths and Polarities. These parameters are usually standardized for a chosen VGA display.

6 RTL anaylysis

The display area of our screen is 640x480 pixels. Our pixel generator module defines all four of the walls/borders of our game as red and 8 pixels thick. The paddle is cyan in colour. The ball is square and cyan in color. Lastly our background is completely black.

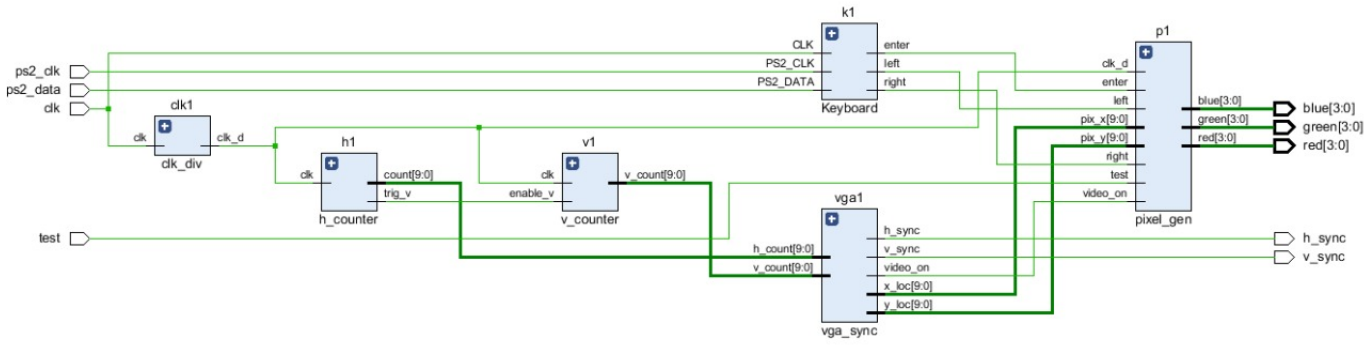


Figure 7: Block Diagram of Output Block.

Overall, the RTL analysis of our so far run code is given above. The output block or the input to the pixel generator in general consist of clk d from the clock divider module, pix x and pix y vedio on signal, left, right and test inputs. The pixel generator takes left and right signals from keyboard module that tells whenever a shift of paddle towards left or right has to be made. Correspondingly the pixels on the screen gets updated.

7 Finite State Machine (FSM)

This section consist of the factored FSM as well as the final Main FSM of the game. The Main FSM is factorized into two FSM namely FSM for Ball and FSM for Paddle that basically constitutes the FSM factoring in the Control Block.

7.1 Factored FSM

Since, the project consists of two significant elements i.e., Ball and Paddle therefore, the elemental FSM with illustration is given in the below section.

7.1.1 FSM for Ball

The FSM of the ball is given below:

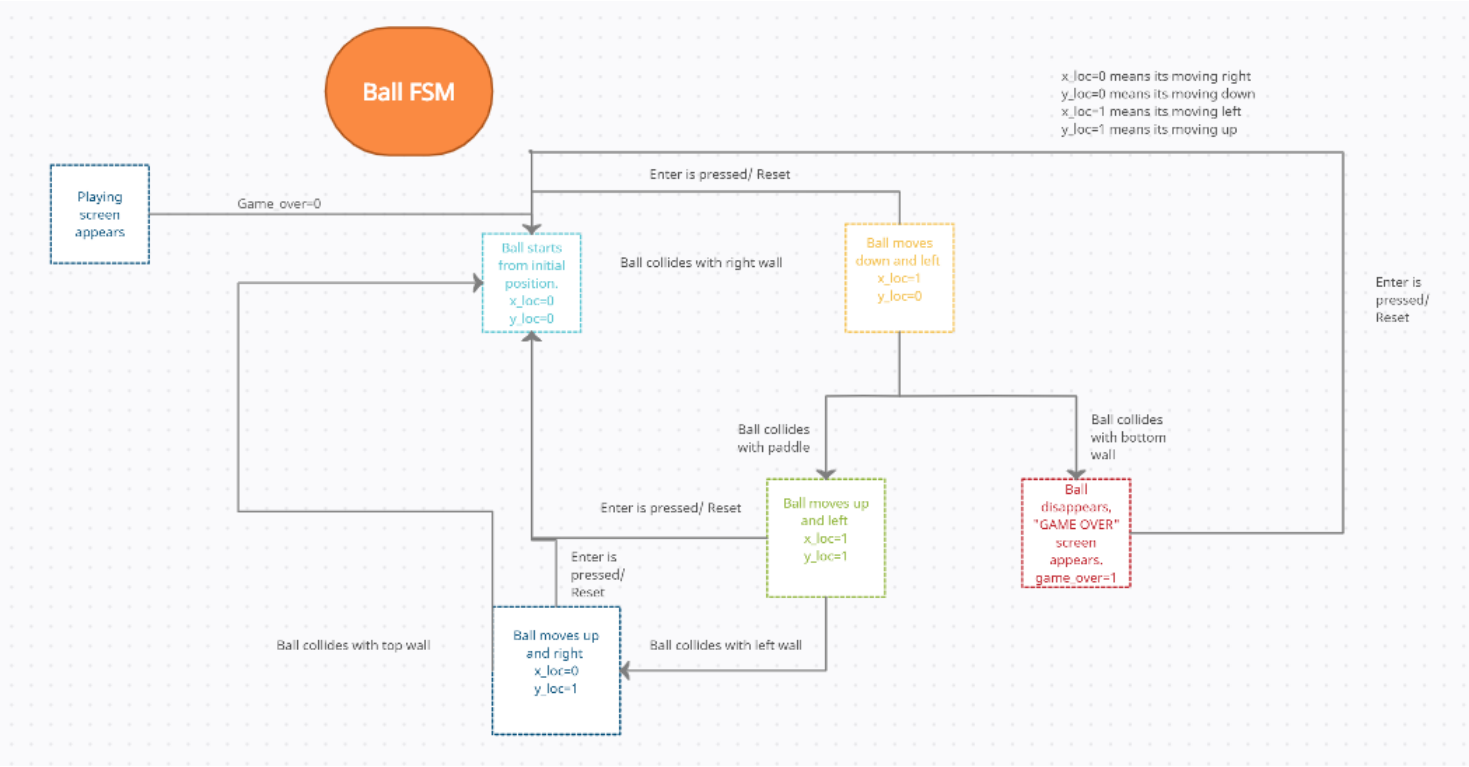


Figure 8: FSM Diagram for Ball.

FSM of ball consists of the following variables that are responsible for transitioning the pixels or in a blatant manner location of the ball. x_loc at 0 implies that the ball is moving right while for moving the ball to left x_loc becomes 1. Similarly, Y_loc at 1 implies that the ball is moving up while Y_loc at 0 defines that the ball is moving downwards.

When the Enter key is pressed the game will start from the Reset state and the Ball begins moving from it's initial position.

7.1.2 FSM for Paddle

The FSM of the paddle is given below:

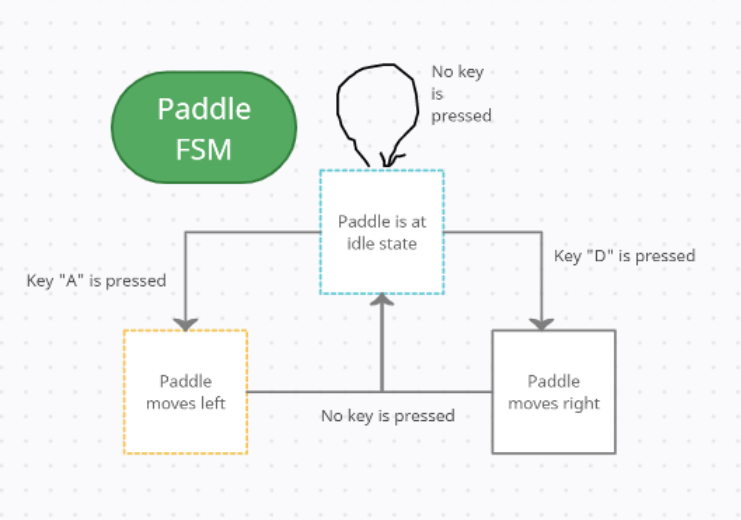


Figure 9: FSM Diagram for Paddle.

The FSM of the paddle is very simple as it contains only 3 states. The paddle remains at the idle state until no key is pressed. If "A" key is pressed then paddle moves to the left. Similarly, if "D" key is pressed, it will take the paddle to the right.

7.2 Main FSM

The main FSM which is actually acting as the cluster of the ball and paddle FSM is given below:

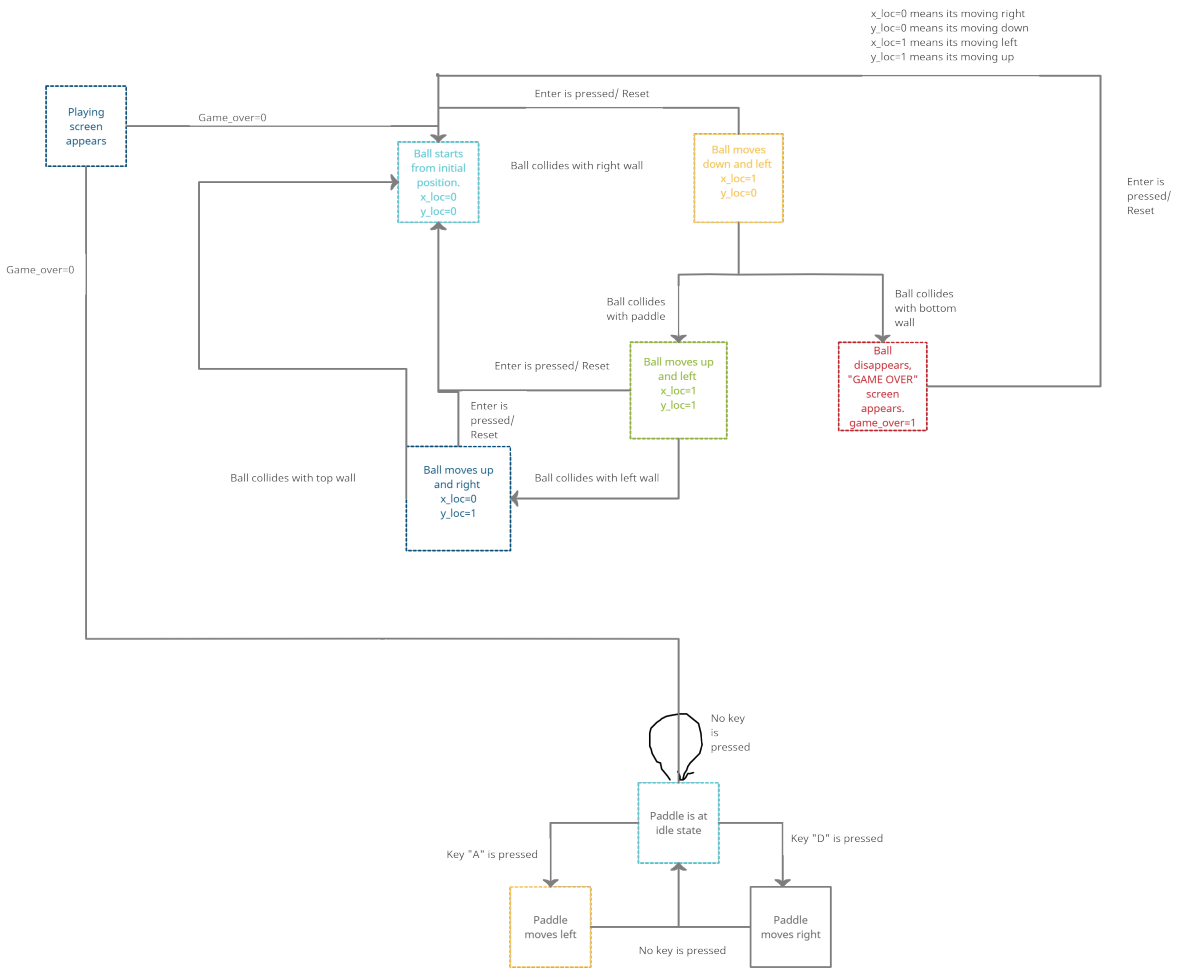
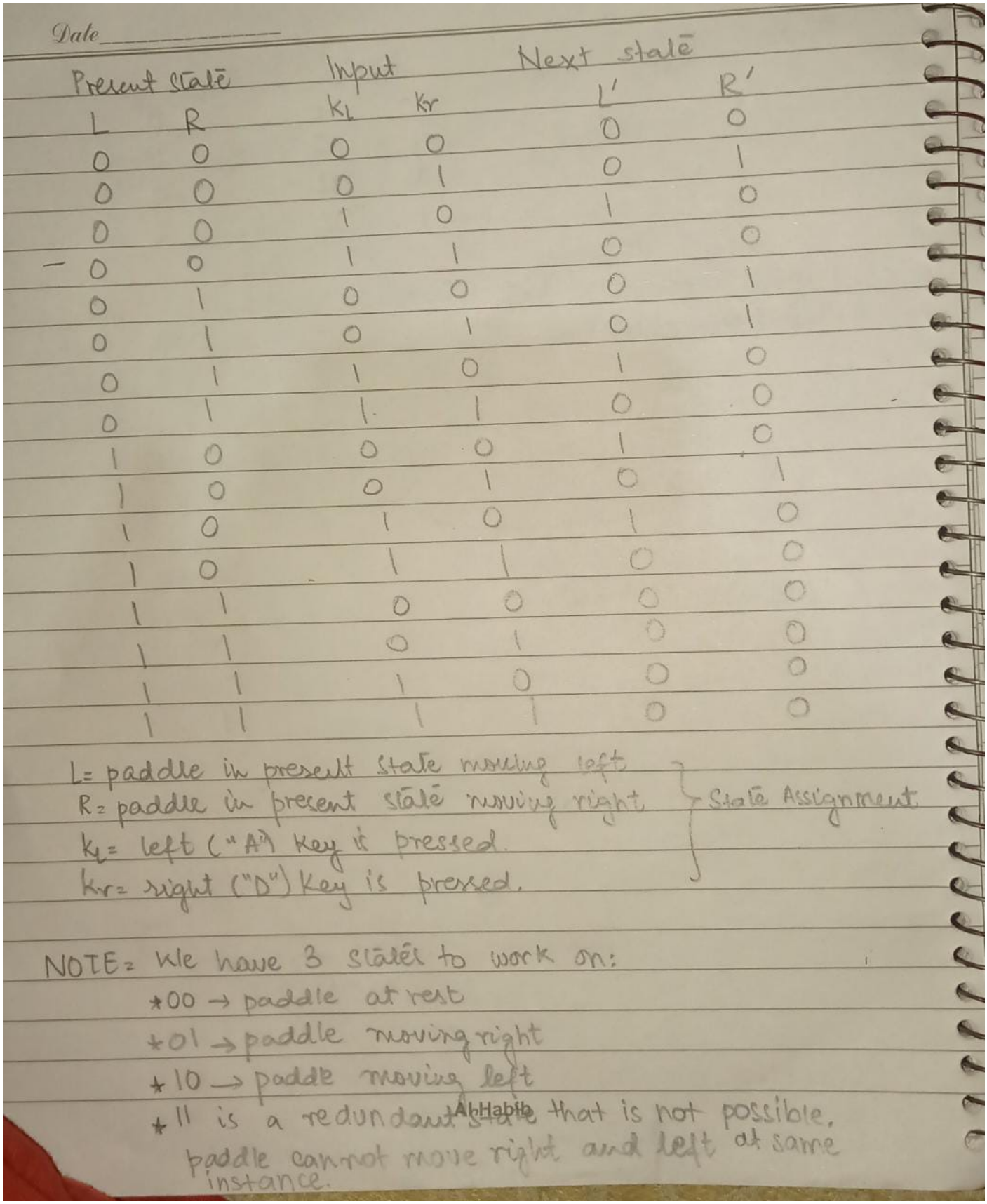


Figure 10: Main Finite State Machine of Pong Game.

The Main FSM which is actually the combination of Ball and Paddle FSM has the same specification as illustrated in the factored FSM section but with the addition that it has linked Ball FSM with the Paddle FSM on the Condition that if Game over has value equal to zero then the paddle FSM is going to execute.

8 PADDLE FSM implemented on gate level.



Present state L	Present state R	Input K_L	Input K_R	Next state L′	Next state R′
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

L = paddle in present state moving left
R = paddle in present state moving right
K_L = left ("A") key is pressed.
K_R = right ("D") key is pressed.

} State Assignment

NOTE = We have 3 states to work on:
- *00 → paddle at rest
- *01 → paddle moving right
- *10 → padde moving left
- *11 is a redundant state that is not possible, paddle cannot move right and left at same instance.

AbHabib

Figure 11: State assignment and state table

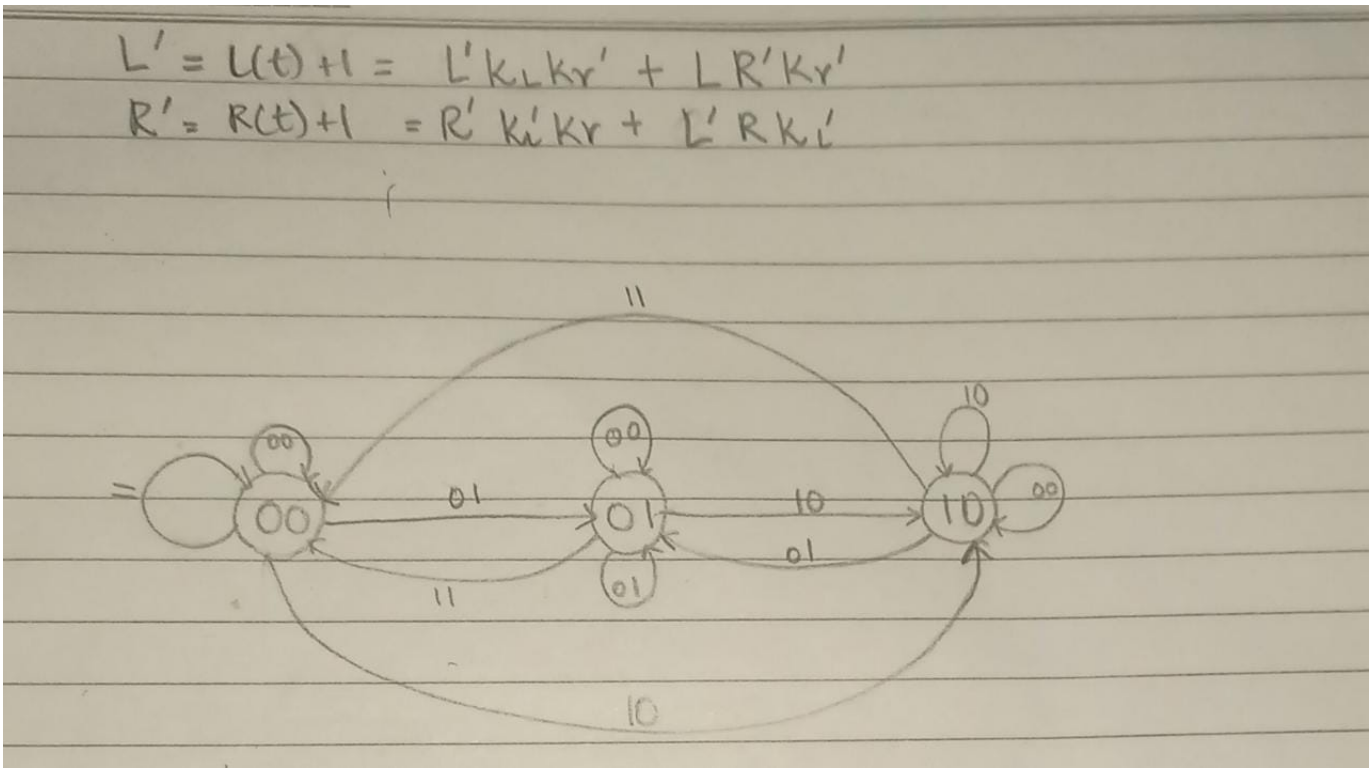


Figure 12: Next state equations and state transition diagram

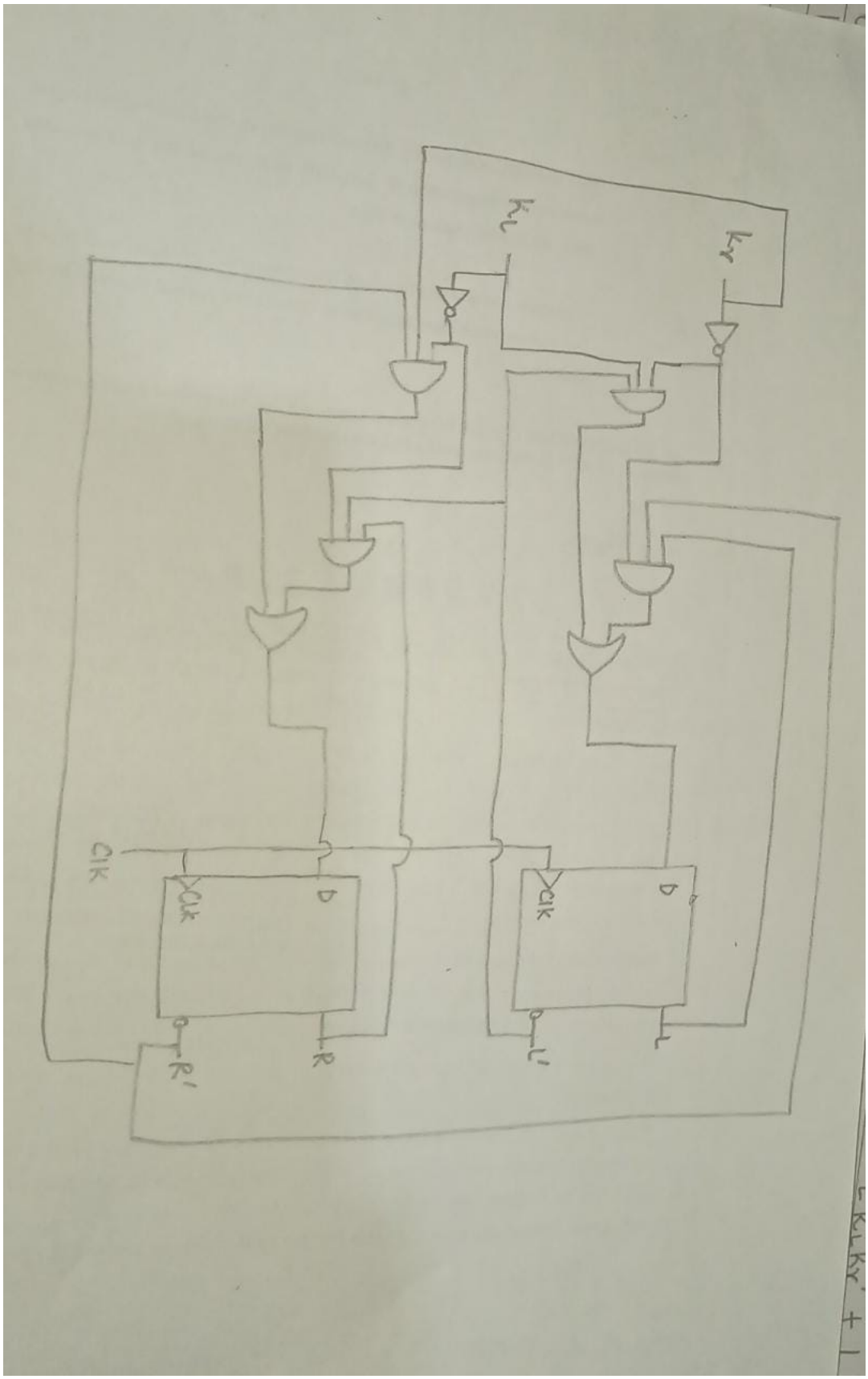


Figure 13: Gate level circuit diagram.

9 Game code

```
// Code your design here
```

```
`timescale 1ns/1ps
module clk_div(clk,clk_d);
      parameter div_value=1;
      input clk;
      output clk_d;
      reg clk_d;
      reg count;
      initial
        begin
          clk_d=0;
          count=0;
        end
      always @(posedge clk)
        begin
          if (count == div_value)
            count <=0;
          else
            count<=count+1;
        end
      always @(posedge clk)
        begin
          if (count == div_value)
            clk_d <= ~ clk_d;
        end
    endmodule


module v_counter(clk,enable_v,v_count);
      input clk,enable_v;
      output [9:0] v_count;
      reg [9:0] v_count;

      initial v_count = 0;
      always @ (posedge clk)
        begin
          if (enable_v==1)
            begin
              if (v_count<524)
                begin

                    v_count<=v_count+1;
                end
              else
                begin
                  v_count <=0;

                end

            end

        end
    endmodule


module h_counter(clk,count,trig_v);
      input clk;
      output [9:0] count;
      output trig_v;
      reg [9:0] count;
      reg trig_v;
```

```
    initial begin count =0;
      trig_v =0;
    end
    always @ (posedge clk)
      begin
        if (count <799)
          begin
            count<=count+1;
            trig_v <=0;
          end
        else
          begin
            count <=0;
            trig_v <=1;
          end
      end
  endmodule


module vga_sync(
    input [9:0]h_count ,
    input [9:0]v_count,
    output h_sync,
    output v_sync,
    output video_on, //active area
    output [9:0] x_loc, //current pixel, x location
    output [9:0] y_loc // current pixel y location
  );

    //for h_sync
    localparam HD = 640;
    localparam HF = 16;              //Right border
    localparam HB = 48;        //left border
    localparam HR = 96; //retrace

    //for v_sync
    localparam VD = 480;
    localparam VF =   10; //bottom border
    localparam VB = 33; //top border
    localparam VR = 2; //vertical retrace

    assign video_on = (h_count<HD) && (v_count<VD);
    assign h_sync= (h_count <(HD+HF)) || (h_count >= (HD+HF+HR));
    assign v_sync=  (v_count <(VD+VF))  || (v_count >= (VD+VF+VR));
    assign x_loc = h_count;
    assign y_loc = v_count;


  endmodule //vga_sync

  // Code your design here



module Keyboard(
input CLK,
  input PS2_CLK,
  input PS2_DATA,
```

```
// output reg [3:0]COUNT,
// output reg TRIG_ARR,
// output reg [7:0]CODEWORD,
   output reg up, reg down, reg left, reg right,reg enter //8 LEDs
   );

  wire [7:0] ARROW_UP = 8'h1D; //codes for arrows
  wire [7:0] ARROW_DOWN = 8'h1B;
  wire [7:0] ARROW_LEFT = 8'h1C;
  wire [7:0] ARROW_RIGHT = 8'h23;
  wire [7:0] ENTER = 8'h5A;
//wire [7:0] EXTENDED = 8'hE0; //codes
//wire [7:0] RELEASED = 8'hF0;

reg read;
reg [11:0] count_reading;
reg PREVIOUS_STATE;
reg scan_err;
reg [10:0] scan_code;
reg [7:0] CODEWORD;
reg TRIG_ARR;
reg [3:0]COUNT;
reg TRIGGER = 0;
reg [7:0]DOWNCOUNTER = 0;


initial begin
PREVIOUS_STATE = 1;
scan_err = 0;
scan_code = 0;
COUNT = 0;
CODEWORD = 0;
read = 0;
count_reading = 0;
end

always @(posedge CLK) begin
if (DOWNCOUNTER < 249) begin
DOWNCOUNTER <= DOWNCOUNTER + 1;
TRIGGER <= 0;
end
else begin
DOWNCOUNTER <= 0;
TRIGGER <= 1;
end
end

always @(posedge CLK) begin
if (TRIGGER) begin
if (read)
count_reading <= count_reading + 1;
else
count_reading <= 0;
end
end


always @(posedge CLK) begin
if (TRIGGER) begin
```

```
if (PS2_CLK != PREVIOUS_STATE) begin
if (!PS2_CLK) begin
read <= 1;
scan_err <= 0;
scan_code[10:0] <= {PS2_DATA, scan_code[10:1]};
COUNT <= COUNT + 1;
end
end
else if (COUNT == 11) begin
COUNT <= 0;
read <= 0;
TRIG_ARR <= 1;

if (!scan_code[10] || scan_code[0] || !(scan_code[1]^scan_code[2]^scan_code[3]^scan_code[4]
^scan_code[5]^scan_code[6]^scan_code[7]^scan_code[8]
^scan_code[9]))
scan_err <= 1;
else
scan_err <= 0;
end
else  begin
TRIG_ARR <= 0;
if (COUNT < 11 && count_reading >= 4000) begin
COUNT <= 0;
read <= 0;
end
end
PREVIOUS_STATE <= PS2_CLK;
end
end


always @(posedge CLK) begin
if (TRIGGER) begin
if (TRIG_ARR) begin
if (scan_err) begin
CODEWORD <= 8'd0;
end
else begin
CODEWORD <= scan_code[8:1];
end
end
else CODEWORD <= 8'd0;
end
else CODEWORD <= 8'd0;
end

always @(posedge CLK) begin
        up <= 0;
        down <= 0;
        right <= 0;
        left <= 0;
        enter<=0;
if (CODEWORD == ARROW_UP)
begin
up <= 1;
down <= 0;
left <= 0;
right <=0;
enter<=0;
```

```
end
else if (CODEWORD == ARROW_DOWN)
begin
up <= 0;
down <= 1;
left <= 0;
right <= 0;
enter<=0;
end
        else if (CODEWORD == ARROW_LEFT)
begin
up <= 0;
down <= 0;
left <= 1;
right <= 0;
enter<=0;
end
        else if (CODEWORD == ARROW_RIGHT)
begin
up <= 0;
down <= 0;
left <= 0;
right <= 1;
enter<=0;
end

else if (CODEWORD == ENTER)
begin
up <= 0;
down <= 0;
left <= 0;
right <= 0;
enter<=1;

end
else begin
up <= 0;
down <= 0;
left <=0;
right <= 0;
enter<=0;
end
end

endmodule
```

```
module pixel_gen ( input clk_d,
        input wire [9:0] pix_x,
        input wire [9:0] pix_y,
        input wire video_on,
        output reg [3:0] red=0,
        output reg [3:0] green=0,
        output reg [3:0] blue=0,
        input left,
        input right,
```

```
        input enter,
        input test
        );



        localparam x_wall_left = 0;
        localparam x_wall_right = 10;
        localparam y_wall_left = 629;
        localparam y_wall_right = 639;
        localparam top_left_wall = 0;
        localparam top_right_wall = 10;
        localparam bottom_wall_left = 469;
        localparam bottom_right_wall = 479;
        reg [23:0] counter = 0;
        // bar left , right boundary
        reg [10:0] x_left_paddle = 221;
        reg [10:0] x_right_paddle = 321;
      // bar top, bowadaattom boundary
          localparam y_top_paddle =449;
          localparam y_bottom_paddle = 455;
        // square ball
        // Bottom wall
          reg [10:0] size_ball = 26; // ball left , right boundary
      reg [10:0] x_left_ball = 100; // make ball a register so taht its value can change.
      reg [10:0] x_right_ball =120;
        // ball top, bottom boundary
      reg [10:0] y_top_ball= 100;
      reg [10:0] y_bottom_ball = 120 ;

      // THE 7 SEGMENT HAS A TOTAL OF 7 LEDs (OR RECTANGLES IN OUR CASE).
//THE FOLLOWING 7 VARIABLES ARE USED FOR THE "RED" COLOR OF THE 7 RECTANGLES.4'h2 IS A DULL RED C
      reg [3:0]r1 =4'h2; // R1 IS FOR THE TOP MOST HORIZONTAL RECTANGLE
      reg [3:0]r2 =4'h2; // R2 IS FOR THE LEFT TOP MOST VERTICAL RECTANGLE
      reg [3:0]r3 =4'h2; // R3 IS FOR THE RIGHT TOP MOST VERTICAL RECCTANGLE
      reg [3:0]r4 =4'h2; // R4 IS THE MIDDLE HORIZONTAL RECTANGLE
      reg [3:0]r5 =4'h2; // R5 IS FOR THE BOTTOM LEFT VERTICAL
      reg [3:0]r6 =4'h2; // R6 IS FOR BOTTOM RIGHT VERTICAL
      reg [3:0]r7 =4'h2; // R7 IS FOR BOTTOM MOST HORIZONTAL

  // THE 7 SEGMENT HAS A TOTAL OF 7 LEDs (OR RECTANGLES IN OUR CASE).
//THE FOLLOWING 7 VARIABLES ARE USED FOR THE "RED" COLOR OF THE 7 RECTANGLES.4'h2 IS
A DULL RED COLOR WHICH IS USED FOR INITALIAZATION PURPOSES
  reg [3:0]l1 =4'h2; // R1 IS FOR THE TOP MOST HORIZONTAL RECTANGLE
  reg [3:0]l2 =4'h2; // R2 IS FOR THE LEFT TOP MOST VERTICAL RECTANGLE
  reg [3:0]l3 =4'h2; // R3 IS FOR THE RIGHT TOP MOST VERTICAL RECCTANGLE
  reg [3:0]l4 =4'h2; // R4 IS THE MIDDLE HORIZONTAL RECTANGLE
  reg [3:0]l5 =4'h2; // R5 IS FOR THE BOTTOM LEFT VERTICAL
  reg [3:0]l6 =4'h2; // R6 IS FOR BOTTOM RIGHT VERTICAL
  reg [3:0]l7 =4'h2; // R7 IS FOR BOTTOM MOST HORIZONTAL
        // object output signals
//      wire wall_l, wall_r,wall_t, wall_b, bar_on , sq_ball_on ;
        //
        // (wall) left vertical strip // pixel within wall

//      assign wall_l = (WALL_X_L<=pix_x) && (pix_x<=WALL_X_R);
//      assign wall_r =(WALL_Y_L<=pix_x) && (pix_x<=WALL_Y_R);
//      assign wall_t =(WALL_T_L<=pix_y) && (pix_y<=WALL_T_R);
//      assign wall_b =(WALL_B_L<=pix_y) && (pix_y<=WALL_B_R);
//        // right vertical bar // pixel within bar
```

```
//         assign bar_on = (BAR_X_L<=pix_x) && (pix_x<=BAR_X_R) &&
(BAR_Y_T<=pix_y)&& (pix_y<=BAR_Y_B);
//        // square ball
//        // pixel within squared ball
//         assign sq_ball_on =(BALL_X_L <=pix_x) && (pix_x <=BALL_X_R) && (BALL_Y_T <=pix_y) &&
          // red
       // rgb multiplexing circuit
       reg x_mov = 1'b0;// when this is 0 means ball goes right.
       reg y_mov = 1'b0;// when this is 0 means ball goes down.
       reg game_over=1'b0;
       reg [6:0] score =0;
       reg [6:0] score1 =0;
       always @(posedge clk_d) begin



     if (score ==2 || score ==3 || score ==5 || score ==6 || score ==7 || score ==8 || score ==
     else begin r1 <= 4'h2; end  // THE TOP MOST RECTANGLE SHOULD BE ENLIGHTED WHEN THE SCORE IS
     if (score == 4 || score == 5 || score == 6 || score == 8|| score ==9 || score==0) begin r2
     else begin r2<= 4'h2; end
     if (score == 1 || score == 2 || score == 3 || score == 4 || score == 7 || score == 8 || sc
     else begin r3<=4'h2; end
     if (score == 2 || score == 3 || score == 4 || score == 5 || score == 6 || score == 8 || sc
     else begin r4 <= 4'h2; end
     if (score == 2 || score == 6 || score == 8 ||  score==0) begin r5 <= 4'hF; end
     else begin r5 <= 4'h2; end
     if (~(score == 2)) begin r6 <= 4'hF; end
     else begin r6 <= 4'h2; end
     if (score == 2 || score == 3 || score == 5 || score == 6 || score == 8 || score ==9 || sco
     else begin r7 <= 4'h2; end



         if (score1 ==2 || score1 ==3 || score1 ==5 || score1 ==6 || score1 ==7 || score1 ==8 ||
     else begin r1 <= 4'h2; end  // THE TOP MOST RECTANGLE SHOULD BE ENLIGHTED WHEN THE SCORE IS
         if (score1 == 4 || score1 == 5 || score1 == 6 || score1 == 8|| score1 ==9 || score1==0
     else begin l2<= 4'h2; end
         if (score1 == 1 || score1 == 2 || score1 == 3 || score1 == 4 || score1 == 7 || score1 =
     else begin l3<=4'h2; end
         if (score1 == 2 || score1 == 3 || score1 == 4 || score1 == 5 || score1 == 6 || score1 =
     else begin l4 <= 4'h2; end
         if (score1 == 2 || score1 == 6 || score1 == 8 ||  score1==0) begin l5 <= 4'hF; end
     else begin l5 <= 4'h2; end
         if (~(score1 == 2)) begin l6 <= 4'hF; end
     else begin l6 <= 4'h2; end
         if (score1 == 2 || score1 == 3 || score1 == 5 || score1 == 6 || score1 == 8 || score1==
     else begin l7 <= 4'h2; end



       counter <= counter +1;

        if (counter == 250000) begin

                  counter <=0;


                  if (x_right_ball >= y_wall_left) begin // collide cases
```

```
            x_left_ball <= x_left_ball - 1;//it collided with right wall and now goes lef
            x_right_ball <=x_right_ball -1;
            x_mov = 1'b1;//it should now go to left side.
            end

            if (x_left_ball <= x_wall_right) begin // collide cases
            x_left_ball <= x_left_ball + 1;//collides on left side of wall, then it moves
            x_right_ball <=x_right_ball + 1;
            x_mov = 1'b0;//it should now go to left side.
            end

            if (y_top_ball <= top_right_wall) begin // collide cases
            y_top_ball<= y_top_ball + 1;//collides on top wall, the moves down. y value i
            y_bottom_ball <=y_bottom_ball + 1;
            y_mov = 1'b0;//it should go down.
            end



            if (y_top_ball>=bottom_right_wall) begin

                game_over<=1'b1;
                score=0;
                score1=0;


             end

          else if (y_bottom_ball == y_top_paddle) begin

//                          // collide cases


                if ((x_left_ball)<=(x_right_paddle) && (x_right_ball)>=(x_left_paddle))
                score<= score +1;
                  if (score > 9) begin
                    score <= 0;
                    score1 <= score1 +1;
                  end
                y_top_ball<= y_top_ball - 1;
                y_bottom_ball <=y_bottom_ball - 1;

                y_mov = 1'b1;//it should go up
               end




        end




            if (x_mov==1'b1) begin   // direction of movement
            x_left_ball <= x_left_ball - 1;
            x_right_ball<= x_right_ball - 1;//ball goes left side.
            end
            if (x_mov==1'b0) begin
```

```
            x_left_ball <= x_left_ball+ 1;
            x_right_ball <= x_right_ball + 1;
            end
            if (y_mov==1'b1) begin
            y_top_ball <= y_top_ball- 1;//y value decreases, ball goes up.
            y_bottom_ball <= y_bottom_ball - 1;
            end
            if (y_mov==1'b0) begin
            y_top_ball <= y_top_ball+ 1;//y value decreases, ball goes up.
            y_bottom_ball <= y_bottom_ball +1;
            end


    end




 if (left) begin


 x_left_paddle = x_left_paddle -18;
 x_right_paddle = x_right_paddle -18;
 end


 if (right) begin

 x_left_paddle = x_left_paddle +18;
 x_right_paddle = x_right_paddle +18;
 end

 if (enter) begin
 x_left_ball <=100;
 x_right_ball <=120;
 y_top_ball <=100;
 y_bottom_ball <=120;
 game_over<=1'b0;

 end
 if (game_over==1) begin
    if  (((((pix_x > 170 && pix_x <230) && (pix_y > 170 && pix_y<180)) || ((pix_x > 170 &
      red <= video_on?(4'hF): (4'h0);
      green <= video_on?(4'h0): (4'h0);
      blue <= video_on?(4'h0): (4'h0);
   end
   else begin
      red <= video_on?(4'h0): (4'h0);
      green <= video_on?(4'h0): (4'h0);
      blue <= video_on?(4'h0): (4'h0);
    end

end




else if ((x_wall_left<=pix_x) && (pix_x<= x_wall_right) ) begin
    red<=3'hF;
```

```
            blue<=3'h0;
            green<=3'h0; end
//            ||  (((pix_x > 608 && pix_x < 610) && (pix_y > 30 && pix_y < 40)) || ((pix_x > 608
```

```
// HERE THE VARIABLES R1,R2,R3,... ARE USED FOR ENLIGHTING OR DARKENING THE RECTANGLES
        else if (pix_x > 601 && pix_x < 608 && pix_y > 28 && pix_y < 30 ) begin
            red<=r1;
            blue<=3'h0;
            green<=3'h0; end
        else if (pix_x > 599 && pix_x < 601 && pix_y > 30 && pix_y < 40 ) begin
            red<=r2;
            blue<=3'h0;
            green<=3'h0; end
        else if (pix_x > 608 && pix_x < 610 && pix_y > 30 && pix_y < 40 ) begin
            red<=r3;
            blue<=3'h0;
            green<=3'h0; end
        else if (pix_x > 601 && pix_x < 608 && pix_y > 40 && pix_y < 42 ) begin
            red<=r4;
            blue<=3'h0;
            green<=3'h0; end
        else if (pix_x > 599 && pix_x < 601 && pix_y > 42 && pix_y < 52 ) begin
            red<=r5;
            blue<=3'h0;
            green<=3'h0; end
        else if (pix_x > 608 && pix_x < 610 && pix_y > 42 && pix_y < 52 ) begin
            red<=r6;
            blue<=3'h0;
            green<=3'h0; end
        else if (pix_x > 601 && pix_x < 608 && pix_y > 52 && pix_y < 54 ) begin
            red<=r7;
            blue<=3'h0;
            green<=3'h0; end


         else if (pix_x > 591 && pix_x < 598 && pix_y > 28 && pix_y < 30 ) begin
            red<=l1;
            blue<=3'h0;
            green<=3'h0; end
         else if (pix_x > 589 && pix_x < 591 && pix_y > 30 && pix_y < 40 ) begin
            red<=l2;
            blue<=3'h0;
            green<=3'h0; end
         else if (pix_x > 598 && pix_x < 600 && pix_y > 30 && pix_y < 40 ) begin
            red<=l3;
            blue<=3'h0;
            green<=3'h0; end
         else if (pix_x > 591 && pix_x < 598 && pix_y > 40 && pix_y < 42 ) begin
            red<=l4;
            blue<=3'h0;
            green<=3'h0; end
         else if (pix_x > 589 && pix_x < 591 && pix_y > 42 && pix_y < 52 ) begin
            red<=l5;
            blue<=3'h0;
```

```
            green<=3'h0; end
         else if (pix_x > 598 && pix_x < 600 && pix_y > 42 && pix_y < 52 ) begin
            red<=l6;
            blue<=3'h0;
            green<=3'h0; end
         else if (pix_x > 591 && pix_x < 598 && pix_y > 52 && pix_y < 54 ) begin
            red<=l7;
            blue<=3'h0;
            green<=3'h0; end









        else if ((y_wall_left<=pix_x) && (pix_x<=y_wall_right)) begin
            red<=3'h0;
            blue<=3'hF;
            green<=3'h0; end
        else if ((top_left_wall<=pix_y) && (pix_y<=top_right_wall )) begin
            red<=3'hF;
            blue<=3'h0;
            green<=3'hF; end
        else if ((bottom_wall_left<=pix_y) && (pix_y<=bottom_right_wall)) begin
            red<=3'hF;
            blue<=3'h0;
            green<=3'hF; end
        else if ((x_left_paddle<=pix_x) && (pix_x<=x_right_paddle) && (y_top_paddle<=pix_y)&& (p
            red<=3'h0;
            blue<=3'hF;
            green<=3'hF; end
        else if ((x_left_ball <=pix_x) && (pix_x <=x_right_ball) && (y_top_ball <=pix_y) && (pix
            red<=3'h0;
            blue<=3'hF;
            green<=3'hF; end
        else begin
            red<=3'h0;
            blue<=3'h0;
            green<=3'h0;
            end// Black background
      end
//         end

    endmodule

module TopLevel(
      input clk,
      output h_sync,
      output v_sync,
      output [3:0] red,
      output [3:0] green,
      output [3:0] blue,
      input ps2_data,
      input ps2_clk,
      input test
    );
```

```
  wire [9:0] h_count;
  wire [9:0] v_count;
  wire [9:0] x_loc;
  wire [9:0] y_loc;
  wire u,d,l,r,e;

  clk_div   clk1(clk,clk_d);
  h_counter  h1(clk_d,h_count,trig_v);
  v_counter  v1(clk_d,trig_v,v_count);
  vga_sync   vga1(h_count,v_count,h_sync,v_sync,video_on,x_loc,y_loc);
  Keyboard k1(clk,ps2_clk,ps2_data,u,d,l,r,e);
  pixel_gen   p1(clk_d,x_loc,y_loc,video_on,red,green,blue,l,r,e,test);


endmodule
```

Thankyou!