Poke-Pong Game Design and Implementation with FPGA and VGA

This report presents the development of the "Pokè-Pong" video game using Verilog description language on Vivado 2019.2, targeting the NEXYS-4DDR FPGA board. The game utilizes VGA signals for display, and the hardware is designed using a combination of sequential and combinational logic. The report covers background knowledge of FPGA boards and their principles, and it is structured as follows: Introduction to FPGA and VGA, Game Idea and Logical Procedures, Top View of the Project, Modules Design, Extra Features, Testbench and Tests, Conclusion, and Reflection.

1. Introduction to FPGA and VGA

1.1. FPGA Background

Field-Programmable Gate Arrays (FPGAs) are semiconductor devices containing programmable logic components and interconnects. These devices can be reprogrammed to implement custom digital circuits, making them a versatile choice for various applications, including video game development.

FPGAs consist of configurable logic blocks (CLBs) [1] interconnected through a programmable routing matrix. CLBs contain look-up tables (LUTs), [2] flip-flops, and other resources to implement different logic functions. The programmability of FPGAs allows designers to create custom hardware solutions tailored to their specific requirements.

In this project, we use the NEXYS-4DDR FPGA board, which features a Xilinx Artix-7 FPGA. This board provides a suitable platform for implementing our video game using Verilog description language on Vivado 2019.2.

1.2. VGA Background

VGA (Video Graphics Array) is an analog display interface that transmits video graphics from hardware, such as an FPGA board, to a CRT monitor using a VGA connector. The VGA system generates three 4-bit outputs representing red, blue, and green pixels, as well as control pulse signals. To produce the VGA signal pulses, horizontal and vertical counters are implemented, determining the active pixel areas across the screen [3]. In this project, we aim to create a video game using VGA signals produced by the NEXYS-4DDR FPGA board.

2. Game Idea and Logical Procedures

2.1. Game Idea

The game designed for this project is called "Poké-Pong", which is inspired by the classic ping-pong arcade game. It features a multiplayer mode where two players compete against each other, controlling paddles to prevent the Pokémon sprite from being hit by the opponent's paddle. The Pokémon moves in two dimensions (x and y-axis) and starts from the middle of the screen. The game offers three score limit options: first player to reach 20, 30, or 40 points.

2.2. Logical Procedures

The game logic includes paddle movement, Pokémon sprite movement, collision detection, and scoring. Players control their paddles using input devices, while the Pokémon sprite moves according to its current direction and speed. When the Pokémon hits a paddle or a screen boundary, it changes direction. The scoring system increases a player's score when the opponent fails to hit the Pokémon back.

3. Top View of the Project

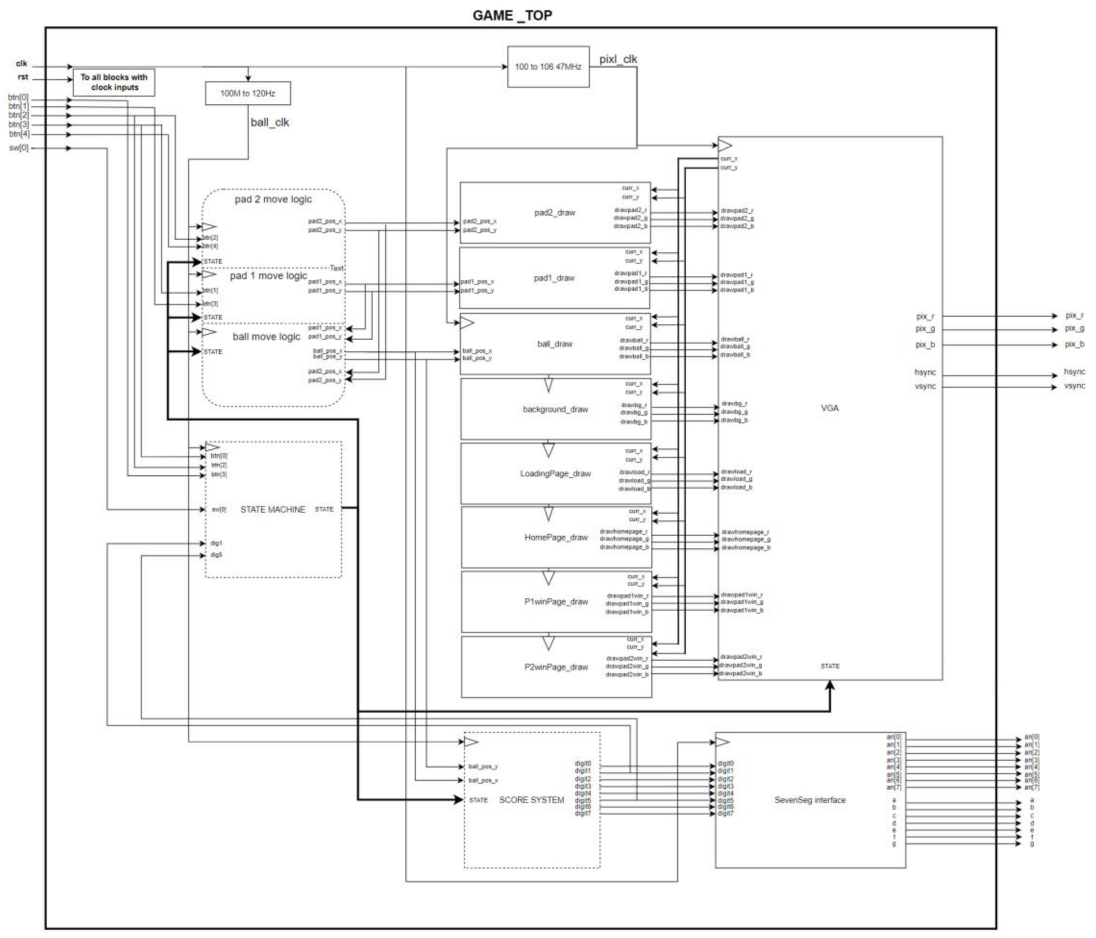
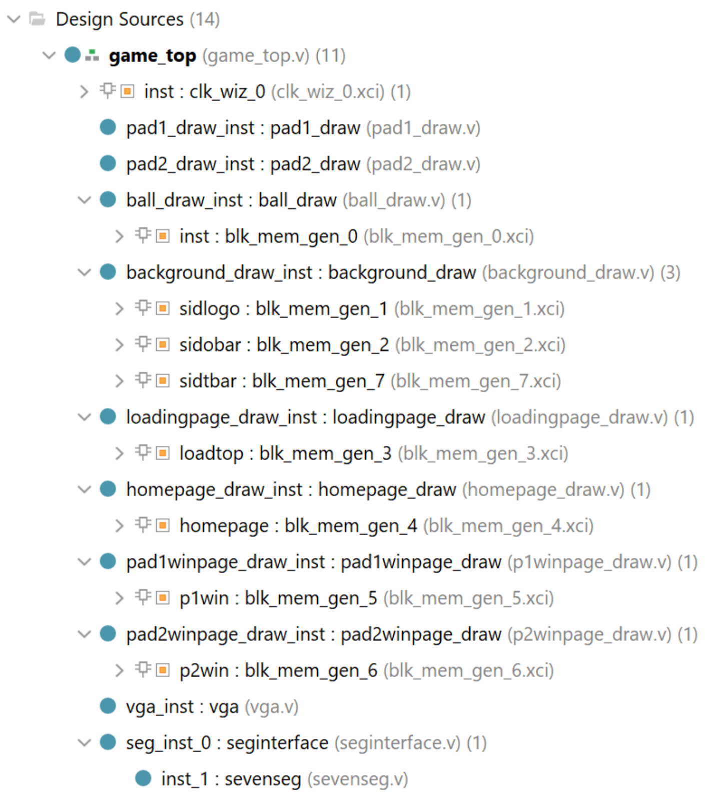


Figure 1- top view



- Design Sources (14)
  - game_top (game_top.v) (11)
    - inst : clk_wiz_0 (clk_wiz_0.xci) (1)
    - pad1_draw_inst : pad1_draw (pad1_draw.v)
    - pad2_draw_inst : pad2_draw (pad2_draw.v)
    - ball_draw_inst : ball_draw (ball_draw.v) (1)
      - inst : blk_mem_gen_0 (blk_mem_gen_0.xci)
    - background_draw_inst : background_draw (background_draw.v) (3)
      - sidlogo : blk_mem_gen_1 (blk_mem_gen_1.xci)
      - sidobar : blk_mem_gen_2 (blk_mem_gen_2.xci)
      - sidtbar : blk_mem_gen_7 (blk_mem_gen_7.xci)
    - loadingpage_draw_inst : loadingpage_draw (loadingpage_draw.v) (1)
      - loadtop : blk_mem_gen_3 (blk_mem_gen_3.xci)
    - homepage_draw_inst : homepage_draw (homepage_draw.v) (1)
      - homepage : blk_mem_gen_4 (blk_mem_gen_4.xci)
    - pad1winpage_draw_inst : pad1winpage_draw (p1winpage_draw.v) (1)
      - p1win : blk_mem_gen_5 (blk_mem_gen_5.xci)
    - pad2winpage_draw_inst : pad2winpage_draw (p2winpage_draw.v) (1)
      - p2win : blk_mem_gen_6 (blk_mem_gen_6.xci)
    - vga_inst : vga (vga.v)
    - seg_inst_0 : seginterface (seginterface.v) (1)
      - inst_1 : sevenseg (sevenseg.v)

Figure 2- System hierarchy

The top module of this project combines various lower-level modules to create a complete game experience. The game logic is implemented using a state machine and various visual elements are drawn using separate modules. The following sections provide an overview of the key elements of the top module.

3.1 State Machine

The game logic is controlled using a state machine that transitions between different states based on user inputs and game conditions. The state machine is driven by the game_clk signal and consists of the following states:

Loading page (3'd0)
Game option page (3'd1)
Game play state (3'd2)
Game pause state (3'd3)
Game over state - Player 2 win (3'd4)
Game over state - Player 1 win (3'd5)

The state machine transitions between these states depending on user inputs (buttons and switches) and the game's internal conditions (max_score) as shown down below.

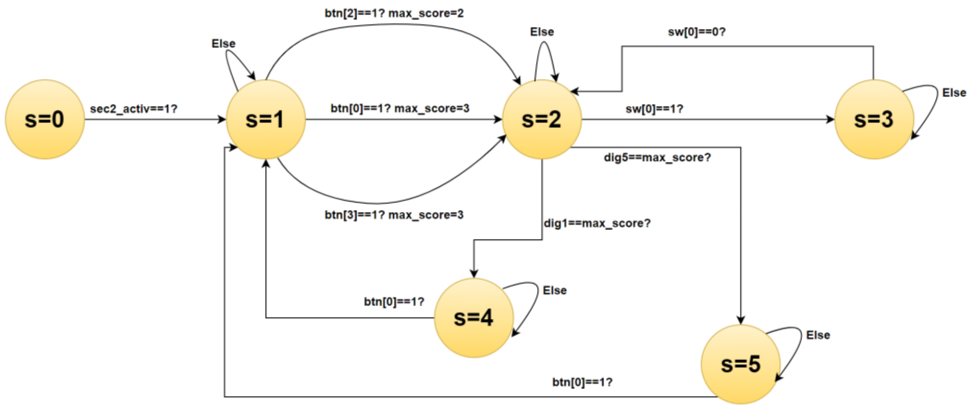


Figure 3- Finite state machine diagram of game

3.2 State 0 to 1 Timer

A timer module is used to control the transition from state 0 to 1. This timer runs for 2 seconds which after this sets the sec2_activ register to 1.

3.3 Visual Elements

The top module includes several instances of lower-level modules for drawing various visual elements on the screen:

Player 1 paddle: pad1_draw
Player 2 paddle: pad2_draw
Ball: ball_draw
Background: background_draw
Loading screen: loadingpage_draw
Game option screen: homepage_draw
Player 1 win screen: pad1winpage_draw
Player 2 win screen: pad2winpage_draw

These modules handle the drawing of their respective visual elements, and these will be broadcasted to the VGA screen in the VGA module based on the state of the system. These make use of following custom designed sprites. These images were converted into a coe file using the mycoe.py script on visual studio. Then, we inserted them as single ports Read Only Memory blocks, using the memory block ip. The RGB data of the images are now stored and will be accessed by their respective draw functions.

3.4 VGA Interface

A VGA interface module (vga_inst) combines all the visual elements drawn by the lower-level modules and outputs the final RGB pixel values (pix_r, pix_g, pix_b) along with the horizontal and vertical synchronization signals (hsync, vsync). The VGA interface module takes the state of the game and the output signals from the visual element modules as its inputs to decide which visual stream to display on the screen.

3.5 Seven-Segment interface & sevenseg modules

A seven-segment interface module is used to display the scores of player 1, in anode 4 to 7, and player 2 , anode 0 to 3. This module takes the digits of the 8 anodes as input and uses a sevenseg module to convert these into digits in the respective anode.

4. Modules Design

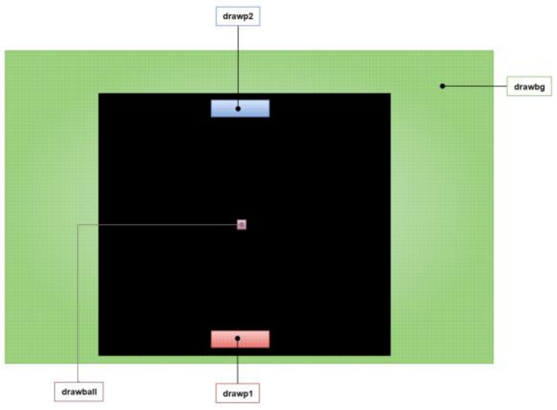
The game relies on several lower-level modules that handle specific tasks, such as drawing visual elements, managing timers, and controlling the display. This section provides an overview of the design and functionality of each module.

4.1 Loading Screen Module (drawload) (*state==0*).

The loading screen module is responsible for drawing the loading screen when the game is in the loading state (*state==0*). It outputs the colour signals (*drawload_r, drawload_g, drawload_b*) from the RGB data stored in the dedicated ROM address, based on the current pixel position (*curr_x* and *curr_y*).

Figure 4- Loading page snapshot

4.2 Game drawing modules (*state==2*).



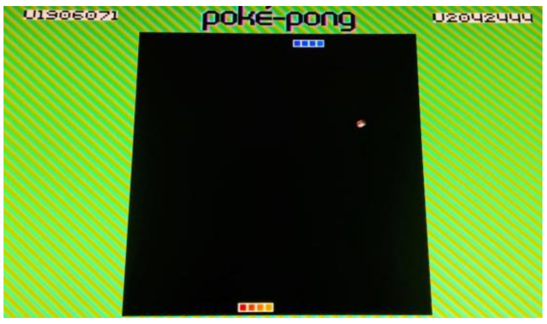


Figure 5- Game page snapshot

Figure 6- Game page drawing functions diagram

4.2.1 Player 1 paddle (drawp1)

The player 1 paddle module is responsible for drawing the first player's paddle on the screen. It receives the position inputs (p1pos_x and p1pos_y) from the position logic in the game_top and outputs the RGB colour signals (drawp1_r, drawp1_g, drawp1_b) by observing the current pixel position (curr_x and curr_y) from the VGA module. The module draws black everywhere in the addressable video region, except from the region encompassed by p1pos_x to p1pos_x+80 and p1pos_y to p1pos_y+20, where in this region the paddle designs are drawn using sequential logic.

4.2.2 Player 2 paddle (drawp2)

The player 2 player module is similarly designed as the previous module, this receives position inputs (p1pos_x and p2pos_y) from game_top and outputs the RGB colour signals (drawp2_r, drawp2_g, drawp2_b).

4.2.3 Ball Module (drawball)

The Ball module is responsible for drawing a poke ball on the screen, such as the ball in the game. This module receives the ball position inputs (ballpos_x and ballpos_y) from the game_top ball position logic, and outputs the RGB colour signals (drawball_r, drawball_g, drawball_b). The module uses the current pixel position (curr_x and curr_y) to determine whether to draw the sprite at a specific location.

4.2.4 Background Module (drawbg)

The background module is responsible for drawing the background during the game state. It outputs the colour signals (drawbg_r, drawbg_g, drawbg_b) from 3 memory blocks positioned at specific coordinates and sets the game background design. The sprites are the two university IDs and the game logo.

4.7 Game Option Screen Module (drawoption) (*state==1*).

The game option screen module is responsible for drawing the game options screen (*state==1*)., allowing the user to select game mode. It outputs the colour signals (*drawoption_r, drawoption_g, drawoption_b*) from the RGB data stored in the dedicated ROM address, based on the current pixel position (*curr_x* and *curr_y*).

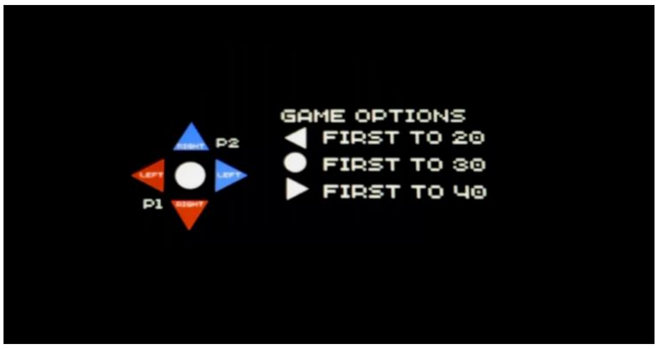


Figure 7- Home page snapshot

4.8 Player 1 Win Screen Module (drawp1win) (*state==5*) & Player 2 Win Screen Module (drawp2win) (*state==4*).

This module is responsible for drawing the screen when player 1 wins the game It outputs the colour signals (drawp1win_r, drawp1win_g, drawp1win_b) from the RGB data stored in the dedicated ROM address, based on the current pixel position (curr_x and curr_y). Similarly to the player 1 win screen module, this module is responsible for drawing the screen when player 2 wins the game. It outputs the colour signals (drawp2win_r, drawp2win_g, drawp2win_b) from the RGB data stored in the dedicated ROM address, based on the current pixel position (curr_x and curr_y).

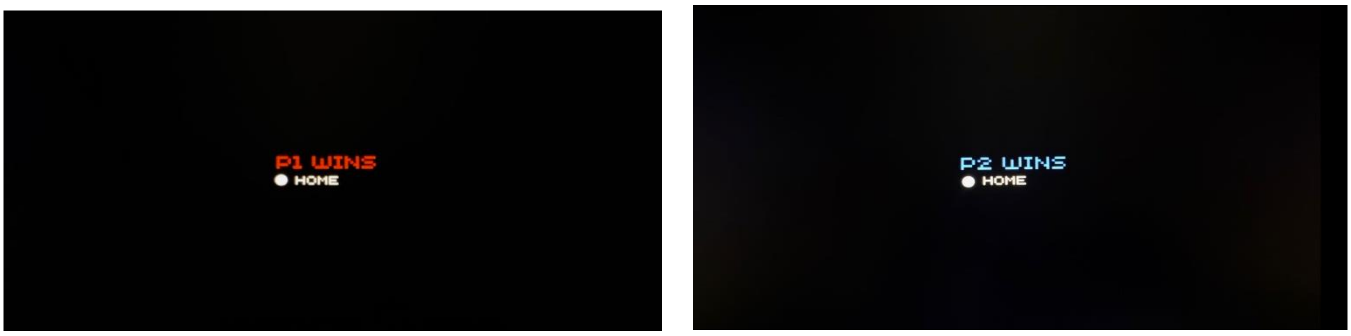


Figure 8- Player 1 & Player 2 win pages snapshot

4.9 Timer Module (timer t1s)

The timer module is responsible for controlling delay-based events in the game. It receives the clk, start, and length signals as inputs and generates an active output signal. The module uses an internal counter to keep track of the elapsed time and toggles the active signal on during the count, and off when not counting.

4.10 VGA Interface Module (vga_inst)

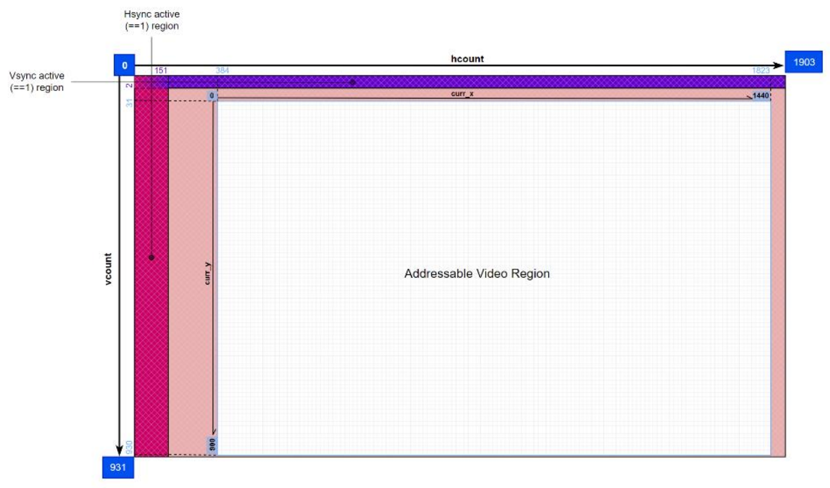


Figure 9- Pixel mapping for our design

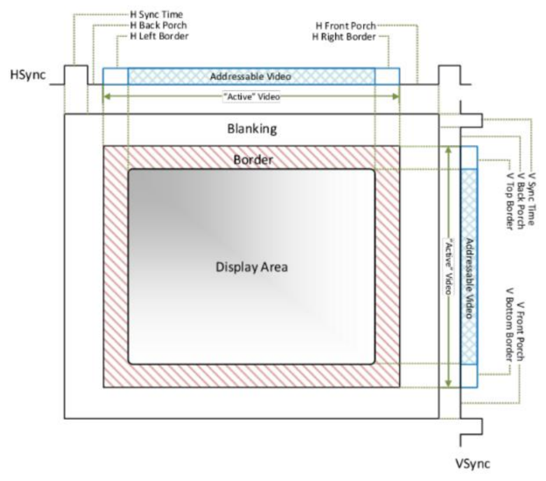


Figure 10- Understanding addressable pixels, taken from https://learn.digilentinc.com/Documents/269

The VGA interface module is responsible for sending the RGB video output (pix_r, pix_g, pix_b) and based the current game states signal and if we are in the addressable video region.

Internal signals within the module handle horizontal and vertical counts, as well as the current X and Y positions on the display. The VGA interface module generates the horizontal sync (hsync) and vertical sync (vsync) signals for the display using combinational logic. It defines the display region based on the horizontal and vertical count values as illustrated in the adjacent VGA module schematic. The module also manages horizontal and vertical counts, as well as the current X and Y positions on the display, through synchronous always blocks.

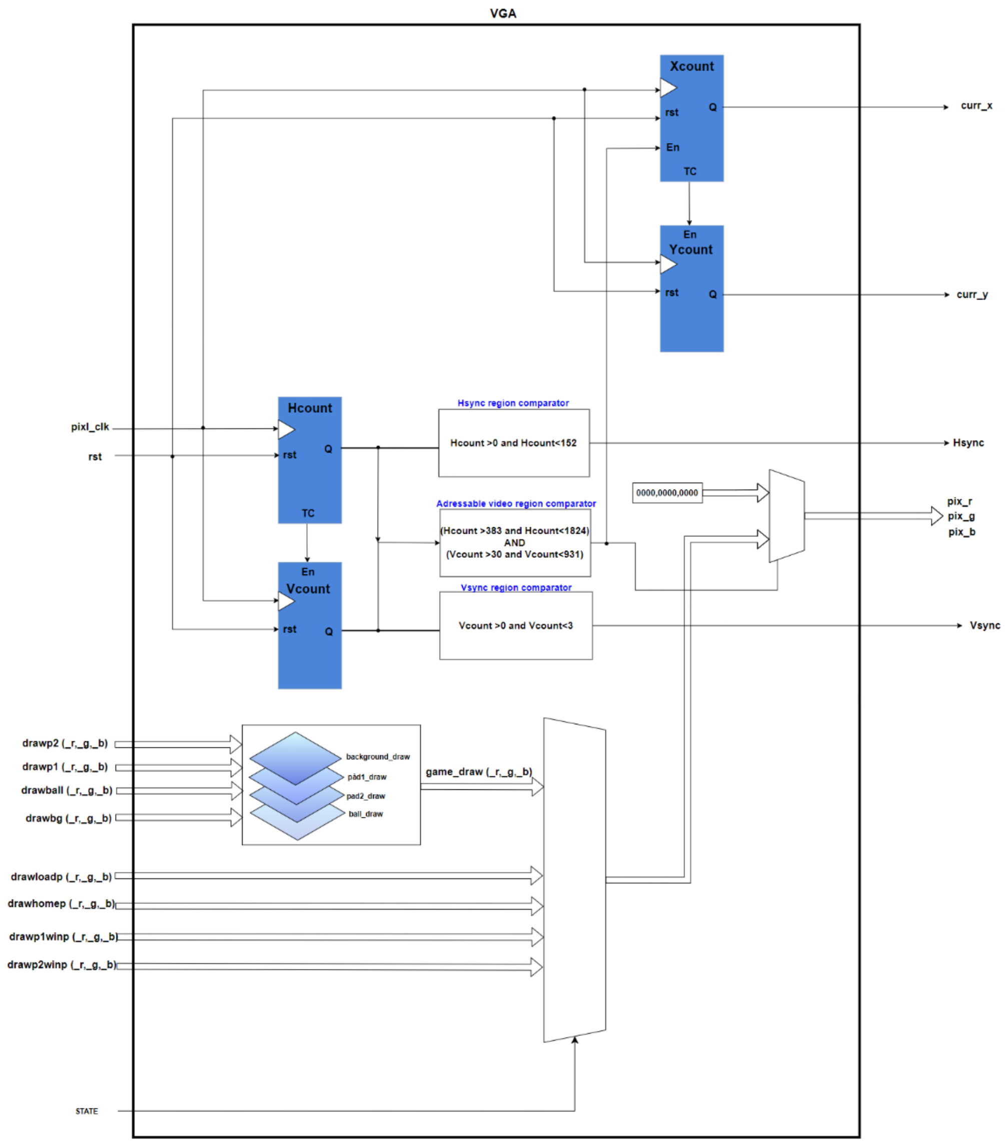


Figure 11- VGA module schematics

The module uses a mux which outputs the draw function from one of the system states based on the current state and output the appropriate RGB signals.

The game_draw function takes in the draw functions from the game play state

(state 2) and overlaps the draw function as layers, and the hierarchy is illustrated in the VGA schematic where the background_draw function is at the most forward, and the ball_draw is at the most backward layer.

5. Game Logic

The game logic module handles object movement, collision detection, and game state updates. It uses the state value from the state machine to switch between game states such as playing state (state==2) and paused (state==2).

5.1 Position, Direction and Collision Detection of the Ball

The ball's position on the screen is given by the coordinates ballpos_x and ballpos_y. The direction of the ball is represented by two binary variables, dir_x and dir_y, which indicate the horizontal and vertical direction of the ball, respectively. When dir_x is zero the ball moves to the left of the screen, and when dir_x is one the ball moves towards the right side of the screen, both by 3 pixels per positive edge of the ball_clk.

During game state, we checks for boundary collisions between the vertical walls and the front facing edge of the paddle. a collision where change in x direction is required, occurs when the coordinates of the top left corner of the ball (which is 20 by 20) crosses the left wall (x<360) or the right wall x>1030. Meanwhile a change in y direction is induced when the collision with the paddles is detected. This occours when the position of the ball is within the 10x80 rectangular region from the current coordinates of pad1, for pad2 this rectangle is offset from the paddle coordinate by +30 in y.

When the player fails to bounce the ball, the ball will travel past the paddles and hit the top or bottom game zone boundaries (when y<100 and y>890). This causes the ball to be redirected to the central y of the game zone at y=550, meanwhile the x position is randomly assigned by adding the most left wall x coordinates with the current value of the ball_count register. At this instant the timer is activated and the ball remains in this position until “active” variable of the timer is zero. This allows for the ball to be suspended in the middle of the screen for 1 second before restarting the gameplay.

When no collision is detected, the ball moves at the same direction and speed.

5.2 Position, Movement and Collision Detection of Paddles

The game has 2 paddles which are controlled by the buttons from the fpga board. Paddle 1 uses buttons btn[2] to move right and btn[4] to move left, meanwhile paddle 2 uses btn[1] to move right and btn[3] to move left. The paddle positions is compared to the two vertical walls at and movement is not permitted when the x coordinate of the paddle is x<360 and x>1000, which takes in account that the paddle is 20 by 80, sow the right boundary (1080) is offsetted by its width. The y coordinates for the two paddles differ for paddle 1 to be located at the bottom of the screen at y=800, and paddle 2 to be displayed at the top of the screen at y=120. The y position of the paddle is kept constant so the paddles can only move within the x plane and between the set boundaries. When a boundary is crossed the position is updated to take the block 1 unit out of the intersection to avoid paddles, or ball, being stuck in the intersection.

5.3 Game Pause

When in game mode (state 2), we can transition into game pause mode by raising sitch 0 to 1, now the state is set to 3 until sw[0] is brought back down to zero. In game pause mode, the ball and paddles control system will keep the ball and paddles in their current coordinates and current direction, not allowing movement, and it will restart from where it left when we go back to state 2, after lowering sw[0].

5.4 Score system and Game over

Points are attributed to a player when the opponent fails to bounce back the ball, when this happens a point is added to the users seven segment display section, player one on the left and player 2 on the right. This continus until one of the player reaches the max score determined by the second digit of the players display. The max_score is set when transitioning to state 2 (game) from state 1 (home). This can either be 2, 3 or 4 depending on which button was pressed to transition (see state machine). When the second digit of player 1 score pad, which is the value displayed by the dig4 register, or the second digit of player 2 which is displayed by dig1, is seen to equal the max_score; the state machine will change state to either 4 or 5 depending on which player reached the score first. The users can now see the winner of the matchin the screen and the final score in the seven segment displays. The user will now be able to press the middle button btn[0] to go back to the home screen to select a new or the same game mode.

Info Bar Display

The drawbg module is responsible for rendering the info bar, which includes the two university IDs and game logo. This is displayed on the very top of the game screen 100 pixels in hight and it spans the screen as required. The player

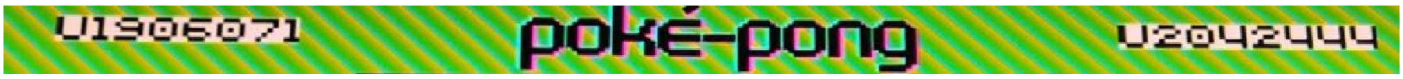


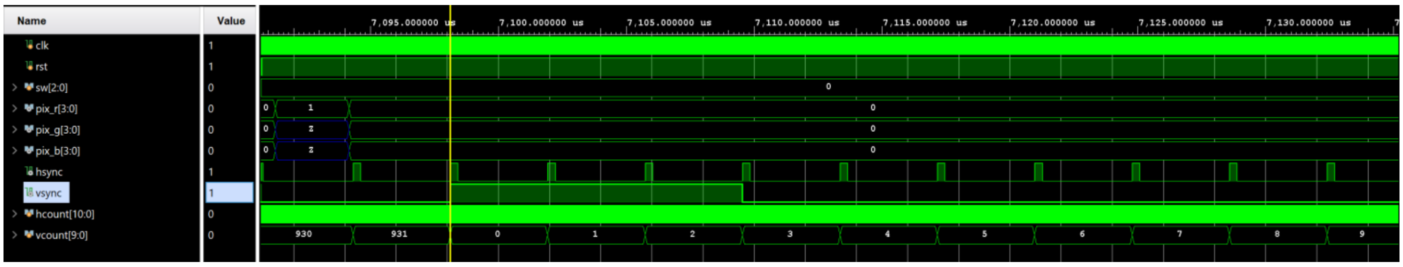
Figure 12- Info bar

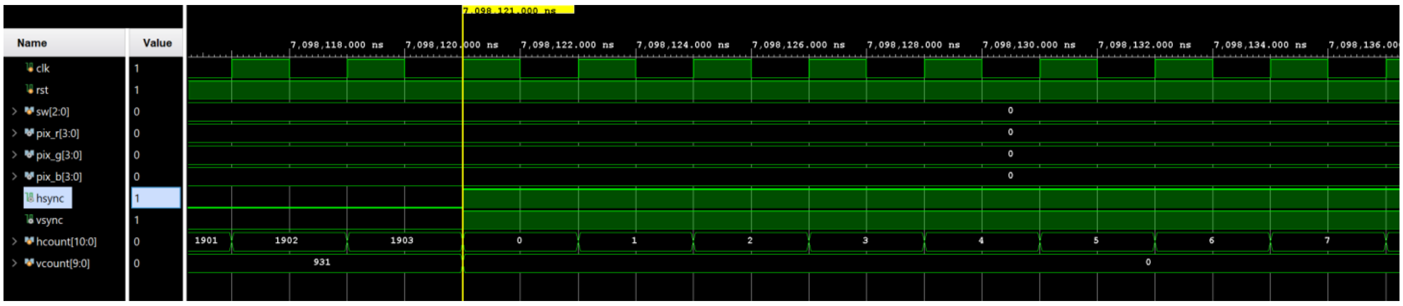
Extra Features

A *loading page* was added to the game to give the player a realistic game experience. The game futures a *home screen* where the user can select the max score that the game will be set at, giving *more options* to the players and allowing an increase *variety* of game modes. The 3 options are first to 20, 30 and 40. Another future that was added to the game is the *display of the winner* of the game once the max score have been reached by one of the players. The *seven-segment displays* was used to display the game scores of the two players, the display also retains the score when the winner is announced and it resets when the player goes back to the home page.

Testbench and Tests

A testbench was developed to simulate the game and verify its functionality. This involved creating test vectors for different game states, object movement, and collisions. The testbench was used to validate the correct operation of each module and their interactions. The test bench was initially used to verify the correct working of the vga module for checking that the Vsync and Hsysnc signal are working as expected. Here we can see that the vsync signal is active for vcount between 0 and 2 as expected, and the next snapshot shows hsync being active when hcount is between 0 and 151.





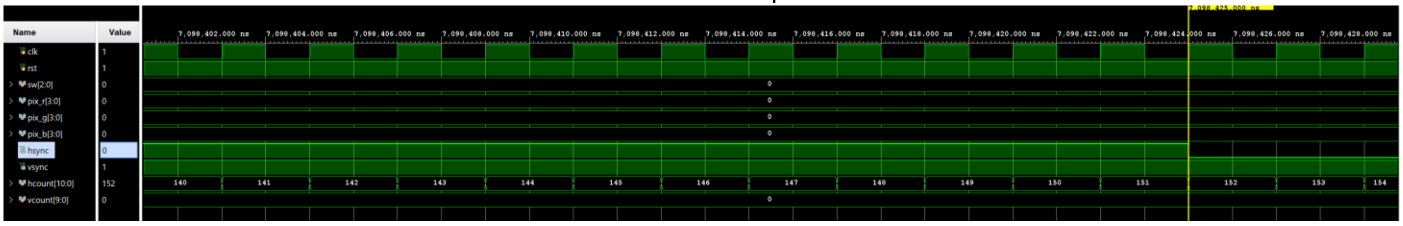


Figure 13- VGA Testbench symulation results

Conclusion and Reflections

In this section, we discuss the key achievements and challenges encountered during the development of our competitive ping-pong game, which we named poke-pong. The primary goal of the player is to outwit their opponent by missing fewer times the moving poke ball with their respective paddles.

Throughout the development of the game, we encountered various challenges and implemented several key adjustments to ensure an engaging and fair gaming experience. This process involved overcoming technical constraints and enhancing gameplay mechanics to support a smooth transition from single-player to multiplayer mode.

Initially, the project faced memory limitations due to the large size of the sprites when uploading the COE files. To tackle this constraint, we strategically resized the height and width of the BMP files, enabling the sprites to fit within the available memory without compromising the game's visual appeal. This optimization facilitated efficient resource allocation and smoother gameplay.

To guarantee a fair and unbiased experience for all players, we made crucial adjustments to the game's mechanics. In the original design, the Pokémon consistently started at the midpoint of the visible screen region, raising concerns about potential bias. To address this issue, we predetermined the Pokémon's position along the x-axis while maintaining the ball at the midpoint of the y-axis. Additionally, we implemented a feature that moved the Pokémon in the direction of the player's paddle that scored a point, ensuring balanced gameplay dynamics.

The most notable achievement of the project was the successful transition from a single-player mode to a multiplayer mode. This accomplishment required an in-depth understanding of the Pokémon's interaction with horizontal visible region boundaries and its interaction with the paddles. With this knowledge, we effectively implemented a second paddle, transforming the game into a fully functional multiplayer experience. The incorporation of these design elements and gameplay mechanics not only addressed technical challenges but also enriched the overall gaming experience, demonstrating the project's successful execution. Reflecting on the project, there were several aspects that was liked and disliked, as well as valuable lessons learned and areas for future improvement.

The collaborative environment fostered effective teamwork, allowing us to brainstorm ideas, share knowledge, and collectively overcome obstacles. The project offered an opportunity to exercise our creativity in designing the game's visual elements, mechanics, and overall gameplay experience. Moreover, the development process deepened our understanding of Verilog and FPGA programming, while familiarizing ourselves with various software tools and hardware components.

On the other hand, the project presented some challenges that occasionally proved frustrating. Encountering memory limitations and hardware constraints required us to adjust our initial plans and make compromises in certain aspects of the game design. Additionally, the tight deadline for the project added stress and limited the scope of features we could implement.

Throughout this journey, we learned valuable lessons and identified areas for improvement. The project emphasized the importance of thorough planning, setting realistic goals, and considering potential challenges and constraints during the initial design phase. Moreover, we developed flexibility and adaptability in our approach, enabling us to find creative solutions and continue making progress despite obstacles.

In conclusion, the development of the competitive ping-pong game has been a rewarding experience, marked by overcoming challenges and achieving key milestones. The final product is a fair and engaging multiplayer game that successfully integrates various design elements and gameplay mechanics. Reflecting on the experience, we value the lessons learned and opportunities for personal and professional growth, and we eagerly anticipate applying these insights to future projects

References:

[1] AMD, “What is an FPGA,” AMD Xilinx, [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.htm. [Accessed 3 April 2023].

[2] H. Bee, “Overview of Lookup Tables (LUT) in FPGA Design,” Hardware Bee, [Online]. Available: https://hardwarebee.com/overview-of-lookup-tables-in-fpga-design/. [Accessed 4 April 2023].

[3] D. E. Wachter, “Lab Experiment 5 – Design Project II,” [Online]. [Accessed 3 April 2023].

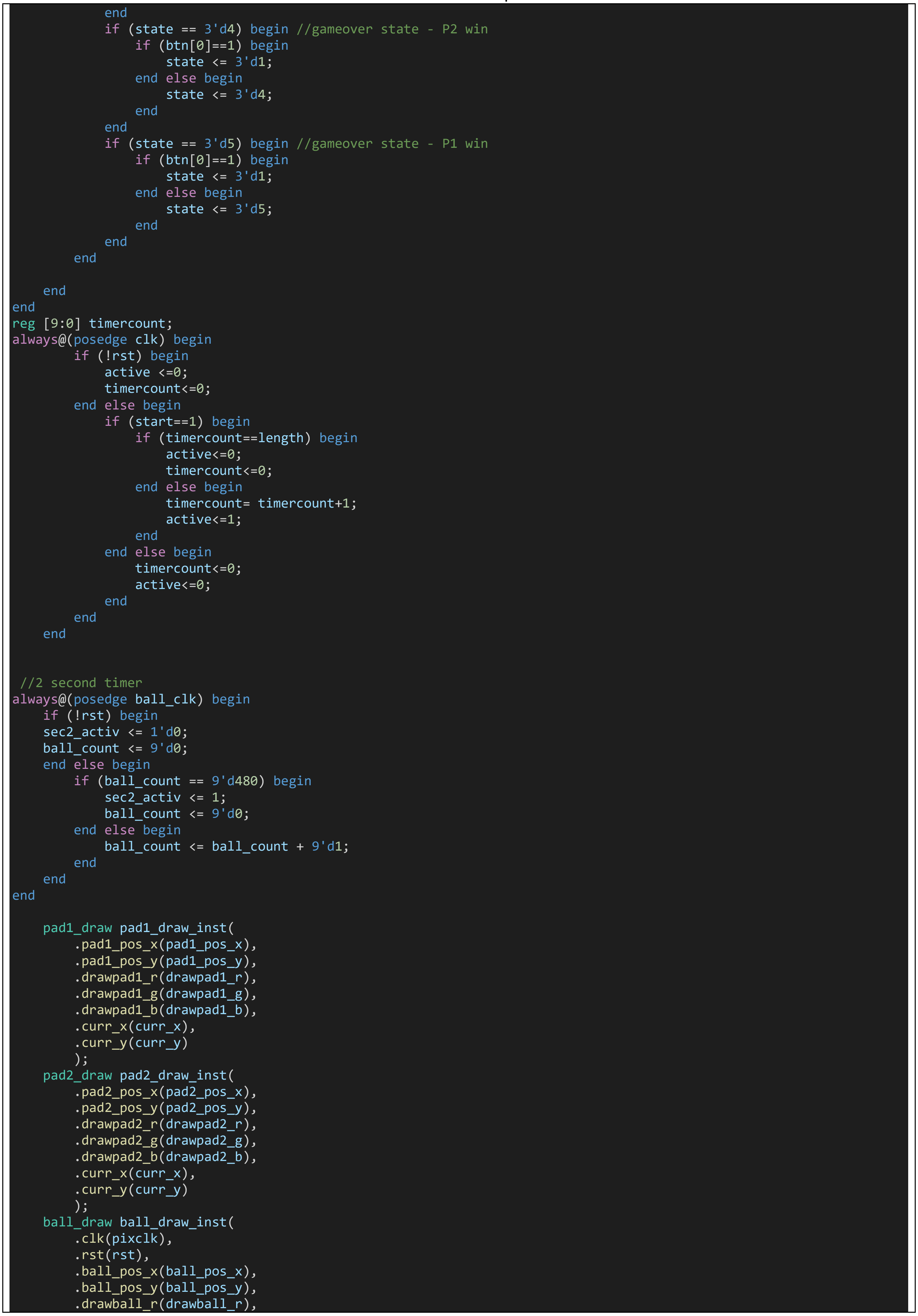
Appendix:

GAME_TOP module

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 25.04.2023 18:23:58
// Design Name: 
// Module Name: game_top
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module game_top(
input clk,
    input rst,
    input [2:0] sw,
    input [4:0] btn,
    output a,
    output b,
    output c,
    output d,
    output e,
    output f,
    output g,
    output [7:0] an,
    output [3:0] pix_r,
    output [3:0] pix_g,
    output [3:0] pix_b,
    output hsync,
    output vsync


    );
```

```
// internal wires
wire pixclk;
wire [3:0] drawpad1_r;
wire [3:0] drawpad1_g;
wire [3:0] drawpad1_b;
wire [3:0] drawpad2_r;
wire [3:0] drawpad2_g;
wire [3:0] drawpad2_b;
wire [3:0] drawball_r;
wire [3:0] drawball_g;
wire [3:0] drawball_b;
wire [10:0] curr_x;
wire [10:0] curr_y;
wire [3:0] drawbg_r;
wire [3:0] drawbg_g;
wire [3:0] drawbg_b;
wire [3:0] drawload_r;
wire [3:0] drawload_g;
wire [3:0] drawload_b;
wire [3:0] drawhomepage_r;
wire [3:0] drawhomepage_g;
wire [3:0] drawhomepage_b;
wire [3:0] drawpad1win_r;
wire [3:0] drawpad1win_g;
wire [3:0] drawpad1win_b;
wire [3:0] drawpad2win_r;
wire [3:0] drawpad2win_g;
wire [3:0] drawpad2win_b;

// registers for game_top module
reg [21:0] clk_div;
reg game_clk;
reg [18:0] ball_div;
reg ball_clk;
reg [10:0] pad1_pos_x;
reg [10:0] pad1_pos_y;
reg [10:0] pad2_pos_x;
reg [10:0] pad2_pos_y;
reg [10:0] ball_pos_x;
reg [10:0] ball_pos_y;

//state machine components
reg [2:0] state;
//wire state;
reg sec2_activ =0;
reg [8:0] ball_count;
wire game_over;
reg [2:0] max_score;

// clock generator
  clk_wiz_0 inst
  (
  // Clock out ports
  .clk_out1(pixclk),
 // Clock in ports
  .clk_in1(clk)
  );

// game clock generation
always@(posedge clk) begin
    if (!rst) begin
        clk_div <= 0;
    end else begin
        if(clk_div == 22'd3333333) begin /// max value att 15 Hz
            clk_div <=0;
            game_clk<= !game_clk;
        end else begin
            clk_div <= clk_div + 1;
        end
    end
end
always@(posedge clk) begin
    if (!rst) begin
        ball_div <= 0;
    end else begin
        if(ball_div == 19'd416666) begin // at 120 Hz
            ball_div <=0;
            ball_clk<= !ball_clk;
        end else begin
```

```
            ball_div <= ball_div + 1;
        end
    end
end

// POSITION LOGIC FOR BAR (player 1 - LEFT AND DOWN) /////////////////////////////////////////////////////
always @(posedge ball_clk) begin
     if (!rst) begin
        pad1_pos_x <= 11'd760;
        pad1_pos_y <= 11'd860; // initial state of the first bar
    end else begin
        if (state==3'd2) begin // game page state
                    pad1_pos_y <= 11'd860;
                    case ({btn[2],btn[4]}) // left (LEFT button)
                        4'b10: begin // right (DOWN button)
                                if (pad1_pos_x > 11'd360) begin // motion disabled below this point
                                    pad1_pos_x <= pad1_pos_x - 11'd10; // move 10 pixel every clock edge
                                end
                               end
                        4'b01: begin // right (DOWN button)
                                if (pad1_pos_x < 11'd1000) begin // motion disabled after this point
                                    pad1_pos_x <= pad1_pos_x + 11'd10; // move pixel every clock edge
                                end
                               end
                        default: begin
                             end
                    endcase
            end else begin
                        pad1_pos_x <= pad1_pos_x;
                        pad1_pos_y <= pad1_pos_y; // motion disabled when bar 1 is not at state 2
            end
    end
end
//////////////////////////////////////////////////////////////////////////////////////////////////////////

///////////// 2ND BAR (player 2 - UP AND RIGHT) ///////////////////////////////////////////////////////////
always @(posedge ball_clk) begin
     if (!rst) begin
        pad2_pos_x <= 11'd760;
        pad2_pos_y <= 11'd120; // initial state of the second bar
    end else begin
        if (state==3'd2)begin
                    pad2_pos_y <= 11'd120;
                    case ({btn[3],btn[1]})
                        4'b10: begin //left (RIGHT button)
                            if (pad2_pos_x > 11'd360) begin // motion disabled below this point
                                pad2_pos_x <= pad2_pos_x - 11'd10; // move 10 pixel every clock edge
                            end
                        end

                        4'b01: begin //right (UP button)
                            if (pad2_pos_x < 11'd1000) begin // motion disabled above this point
                                pad2_pos_x <= pad2_pos_x + 11'd10; // move 10 pixel every clock edge
                            end
                        end
                        default: begin
                                 end
                    endcase
                  end else begin
                    pad2_pos_x <= pad2_pos_x;
                    pad2_pos_y <= pad2_pos_y; // motion disabled when bar 2 is not at state 2
                  end
    end
end

//POSITION LOGIC FOR SPRITE//////////////////////////////////////////////////////////////////////////////
reg dir_y; //direction in y
reg dir_x; //direction in x
reg start;
reg [9:0] length;
reg [9:0] active;
always @(posedge ball_clk) begin
     if (!rst) begin
        ball_pos_y <= 11'd720 ; // original y position of sprite at reset
        dir_y<=1'd0;
     end else begin
        if (state==3'd2)begin
                    if ((ball_pos_y > 11'd890 || ball_pos_y < 11'd100)) begin // after passing top and bottom
boundary
                        start<=1;
                        length<=9'd500;
                        ball_pos_y <= 11'd550;
                        dir_y<=!dir_y;
                    end else begin
                     if (active==0) begin
                        start<=0;
                        if ((((ball_pos_x+11'd20)>pad2_pos_x) && (ball_pos_x< pad2_pos_x+11'd80) &&
                            (ball_pos_y < pad2_pos_y+11'd20 )&&(ball_pos_y>(pad2_pos_y+11'd10)))
                            ||(((ball_pos_y+11'd20)>pad1_pos_y) && ((ball_pos_x+11'd19 > pad1_pos_x )
                            && (ball_pos_x < pad1_pos_x+11'd80 )&&((ball_pos_y+11'd20)<(pad1_pos_y+11'd10))))) begin

                            dir_y<=!dir_y;
                            if (((ball_pos_x+11'd20)>pad2_pos_x) && (ball_pos_x< pad2_pos_x+11'd80) &&
                                (ball_pos_y < pad2_pos_y+11'd20 )&&(ball_pos_y>(pad2_pos_y+11'd10))) begin
                                ball_pos_y<=11'd141;
                            end
                            if (((ball_pos_y+11'd20)>pad1_pos_y) && ((ball_pos_x+11'd49 > pad1_pos_x ) &&
                                (ball_pos_x < pad1_pos_x+11'd80 )&&((ball_pos_y+11'd20)<(pad1_pos_y+11'd10)))) begin
                                ball_pos_y<=11'd839;
                            end
                        end else begin
                            dir_y<=dir_y;
                            if (dir_y) begin
                                ball_pos_y<=ball_pos_y-11'd2;
                            end else begin
                                ball_pos_y<=ball_pos_y+11'd2;
                            end
                        end
                     end
                    end
              end else begin
                    ball_pos_y <= ball_pos_y;
                    dir_y<=dir_y;
              end
     end
end

//direction in x
always @(posedge ball_clk) begin
     if (!rst) begin // centre button - brings it back to the middle
        ball_pos_x <= 11'd720; // midpoint of screen
        dir_x<=1'd0;
     end else begin
        if (state==3'd2) begin
                    if ((ball_pos_y > 11'd890 || ball_pos_y < 11'd100)) begin
                        ball_pos_x <= 11'd480+ball_count;
                        dir_x<=!dir_x;
                    end else begin
                     if (active==0) begin
                        if (((ball_pos_x)<11'd360)||((ball_pos_x)>11'd1060) )begin
                            dir_x<=!dir_x;
                            if (ball_pos_x < 11'd360) begin
                                ball_pos_x<=11'd361;
                            end
                            if ((ball_pos_x)>11'd1060) begin
                                ball_pos_x<=11'd1059;
                            end
                        end else begin
                            dir_x<=dir_x;
                            if (dir_x) begin
                                ball_pos_x<=ball_pos_x+11'd3;
                            end else begin
                                ball_pos_x<=ball_pos_x-11'd3;
                            end
                        end
                     end
                    end
              end else begin
                    ball_pos_x <= ball_pos_x;
                    dir_x<=dir_x;
              end
     end
end
//POINTS SYSTEM ///////////////////////////////////////////////////////////////////////////////
reg [3:0] dig0;
reg [3:0] dig1;
reg [3:0] dig2;
reg [3:0] dig3;
reg [3:0] dig4;
reg [3:0] dig5;
reg [3:0] dig6;
reg [3:0] dig7;


 always@(posedge ball_clk) // for synchronous logic we neeed an always@posedege block - going up - 0 to 1
 begin
    if (!rst) begin
        dig0 <= 0;
        dig1 <= 0;
        dig2 <= 0;
        dig3 <= 0;
        dig4 <= 0;
        dig5 <= 0;
        dig6 <= 0;
        dig7 <= 0;

    end else begin    // at every interval the div clock is implementing we should add one to hexcnt

        case (state)
            (3'd0||3'd1): begin
                    dig0 <= 0;
                    dig1 <= 0;
                    dig2 <= 0;
                    dig3 <= 0;
                    dig4 <= 0;
                    dig5 <= 0;
                    dig6 <= 0;
                    dig7 <= 0;
                  end
            (3'd2): begin
                    //player 1 (bottom) point system
                    if ((ball_pos_y+11'd20 > pad1_pos_y+11'd20)&&(ball_pos_y+11'd20 < pad1_pos_y+11'd23)) begin
                        if(dig0 < 4'h9) begin
                               dig0 <= dig0 + 1;
                           end else begin
                               dig0 <= 0;
                               //dig1 <= dig1 + 1;
                               if(dig1 < 4'h9) begin
                                   dig1 <= dig1 + 1;
                               end else begin
                                   dig1 <= 0;
                                   //hxc2 <= hxc2 + 1;
                                   if(dig2 < 4'h9) begin
                                       dig2 <= dig2 + 1;
                                   end else begin
                                       dig2 <= 0;
                                       //hxc3 <= hxc3 + 1;'
                                       if(dig3 < 4'h9) begin
                                           dig3 <= dig3 + 1;
                                       end else begin
                                           dig3 <= 0;
                                       end
                                   end
                               end
                           end
                    end
                    //player 2 (top) point system
                    if ((ball_pos_y < pad2_pos_y)&&(ball_pos_y >= pad2_pos_y-11'd3)) begin
                        if(dig4 < 4'h9) begin
                               dig4 <= dig4 + 1;
                           end else begin
                               dig4 <= 0;
                               //dig1 <= dig1 + 1;
                               if(dig5 < 4'h9) begin
                                   dig5 <= dig5 + 1;
                               end else begin
                                   dig5 <= 0;
                                   //hxc2 <= hxc2 + 1;
                                   if(dig6 < 4'h9) begin
                                       dig6 <= dig6 + 1;
                                   end else begin
                                       dig6 <= 0;
                                       //hxc3 <= hxc3 + 1;'
                                       if(dig7 < 4'h9) begin
                                           dig7 <= dig7 + 1;
                                       end else begin
                                           dig7 <= 0;
                                       end
                                   end
                               end
                           end
                    end
                  end
              (3'd4): begin
                    dig0 <= dig0;
                    dig1 <= dig1;
                    dig2 <= dig2;
                    dig3 <= dig3;
                    dig4 <= dig4;
                    dig5 <= dig5;
                    dig6 <= dig6;
                    dig7 <= dig7;
                  end
              (3'd5): begin
                    dig0 <= dig0;
                    dig1 <= dig1;
                    dig2 <= dig2;
                    dig3 <= dig3;
                    dig4 <= dig4;
                    dig5 <= dig5;
                    dig6 <= dig6;
                    dig7 <= dig7;
                  end
        endcase
    end

 end

 ////////////////STATE MACHINE//////////////////////////////

always@(posedge game_clk) begin
    if (!rst) begin
    state <= 0;
    end else begin
        if (sec2_activ==0) begin
            state <= 3'd0;
        end else begin
            if (state == 3'd0 ) begin // loading page
                state <= 3'd1;
            end
            if (state == 3'd1) begin //game homepage
                if (btn[0]==1) begin
                    state <= 3'd2;
                    max_score<= 3'd3;
                end else begin
                    if (btn[2]==1) begin
                        state <= 3'd2;
                        max_score<= 3'd2;
                    end else begin
                        if (btn[3]==1) begin
                            state <= 3'd2;
                            max_score<= 3'd4;
                        end else begin
                            state <= 3'd1;
                        end
                    end
                end
            end
            if (state == 3'd2) begin // game state
                if (sw[0]==1) begin
                    state <= 3'd3; //pause
                end else begin
                    if (dig1 == max_score) begin //player 2 win
                        state <= 3'd4;
                    end else begin
                        if (dig5 == max_score) begin //player 1 win
                            state <= 3'd5;
                        end else begin
                            state <= 3'd2;
                        end
                    end
                end

            end
            if (state == 3'd3) begin //pause state
                if (sw[0]==0) begin
                    state <= 3'd2;
                end else begin
                    state <= 3'd3;
                end
```



```
            end
            if (state == 3'd4) begin //gameover state - P2 win
                if (btn[0]==1) begin
                    state <= 3'd1;
                end else begin
                    state <= 3'd4;
                end
            end
            if (state == 3'd5) begin //gameover state - P1 win
                if (btn[0]==1) begin
                    state <= 3'd1;
                end else begin
                    state <= 3'd5;
                end
            end
        end

    end
end
reg [9:0] timercount;
always@(posedge clk) begin
        if (!rst) begin
            active <=0;
            timercount<=0;
        end else begin
            if (start==1) begin
                if (timercount==length) begin
                    active<=0;
                    timercount<=0;
                end else begin
                    timercount= timercount+1;
                    active<=1;
                end
            end else begin
                timercount<=0;
                active<=0;
            end
        end
    end


 //2 second timer
always@(posedge ball_clk) begin
    if (!rst) begin
    sec2_activ <= 1'd0;
    ball_count <= 9'd0;
    end else begin
        if (ball_count == 9'd480) begin
            sec2_activ <= 1;
            ball_count <= 9'd0;
        end else begin
            ball_count <= ball_count + 9'd1;
        end
    end
end

    pad1_draw pad1_draw_inst(
        .pad1_pos_x(pad1_pos_x),
        .pad1_pos_y(pad1_pos_y),
        .drawpad1_r(drawpad1_r),
        .drawpad1_g(drawpad1_g),
        .drawpad1_b(drawpad1_b),
        .curr_x(curr_x),
        .curr_y(curr_y)
        );
    pad2_draw pad2_draw_inst(
        .pad2_pos_x(pad2_pos_x),
        .pad2_pos_y(pad2_pos_y),
        .drawpad2_r(drawpad2_r),
        .drawpad2_g(drawpad2_g),
        .drawpad2_b(drawpad2_b),
        .curr_x(curr_x),
        .curr_y(curr_y)
        );
    ball_draw ball_draw_inst(
        .clk(pixclk),
        .rst(rst),
        .ball_pos_x(ball_pos_x),
        .ball_pos_y(ball_pos_y),
        .drawball_r(drawball_r),
        .drawball_g(drawball_g),
        .drawball_b(drawball_b),
        .curr_x(curr_x),
        .curr_y(curr_y)
        );

     background_draw background_draw_inst(
            .clk(pixclk),
            .rst(rst),
            .drawbg_r(drawbg_r),
            .drawbg_g(drawbg_g),
        .drawbg_b(drawbg_b),
        .curr_x(curr_x),
        .curr_y(curr_y)
    );

    loadingpage_draw loadingpage_draw_inst(
        .clk(pixclk),
        .rst(rst),
        .drawload_r(drawload_r),
        .drawload_g(drawload_g),
        .drawload_b(drawload_b),
        .curr_x(curr_x),
        .curr_y(curr_y)
    );

    homepage_draw homepage_draw_inst(
        .clk(pixclk),
        .rst(rst),
        .drawhomepage_r(drawhomepage_r),
        .drawhomepage_g(drawhomepage_g),
        .drawhomepage_b(drawhomepage_b),
        .curr_x(curr_x),
        .curr_y(curr_y)
    );

    pad1winpage_draw pad1winpage_draw_inst(
        .clk(pixclk),
        .rst(rst),
        .drawpad1win_r(drawpad1win_r),
        .drawpad1win_g(drawpad1win_g),
        .drawpad1win_b(drawpad1win_b),
        .curr_x(curr_x),
        .curr_y(curr_y)
    );

    pad2winpage_draw pad2winpage_draw_inst(
        .clk(pixclk),
        .rst(rst),
        .drawpad2win_r(drawpad2win_r),
        .drawpad2win_g(drawpad2win_g),
        .drawpad2win_b(drawpad2win_b),
        .curr_x(curr_x),
        .curr_y(curr_y)
    );



vga vga_inst(
    .state(state),
    .clk(pixclk),
    .rst(rst),

    .drawload_r(drawload_r),
    .drawload_g(drawload_g),
    .drawload_b(drawload_b),
    .drawhomepage_r(drawhomepage_r),
    .drawhomepage_g(drawhomepage_g),
    .drawhomepage_b(drawhomepage_b),
    .drawpad1win_r(drawpad1win_r),
    .drawpad1win_g(drawpad1win_g),
    .drawpad1win_b(drawpad1win_b),
    .drawpad2win_r(drawpad2win_r),
    .drawpad2win_g(drawpad2win_g),
    .drawpad2win_b(drawpad2win_b),

    .drawpad1_r(drawpad1_r),
    .drawpad1_g(drawpad1_g),
    .drawpad1_b(drawpad1_b),
    .drawpad2_r(drawpad2_r),
    .drawpad2_g(drawpad2_g),
    .drawpad2_b(drawpad2_b),
    .drawball_r(drawball_r),
    .drawball_g(drawball_g),
    .drawball_b(drawball_b),
    .drawbg_r(drawbg_r),
    .drawbg_g(drawbg_g),
    .drawbg_b(drawbg_b),
    .pix_r(pix_r),
    .pix_g(pix_g),
    .pix_b(pix_b),
    .curr_x(curr_x),
    .curr_y(curr_y),
    .hsync(hsync),
    .vsync(vsync)
);

seginterface seg_inst_0 (.an(an), .dig0(dig0),.dig1(dig1), .dig2(dig2), .dig3(dig3),
    .dig4(dig4),.dig5(dig5), .dig6(dig6), .dig7(dig7),
    .clk(clk), .rst(rst), .a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g));

endmodule
```

PAD1_DRAW module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 26.04.2023 15:17:13
// Design Name:
// Module Name: pad1_draw
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module pad1_draw(
    input [10:0] pad1_pos_x,
    input [10:0] pad1_pos_y,
    output [3:0] drawpad1_r,
    output [3:0] drawpad1_g,
    output [3:0] drawpad1_b,
    input [10:0] curr_x,
    input [10:0] curr_y
    );

reg [3:0] pad1_r, pad1_b, pad1_g;

 //block
colour

    // 2 0 x
80

    always@*
begin

        if((curr_x < pad1_pos_x) || (curr_x > (pad1_pos_x + 11'd80))
||
          (curr_y < pad1_pos_y) || (curr_y > (pad1_pos_y + 11'd80)))
begin

            pad1_r <=
4'b0000;

            pad1_g <=
4'b0000;

            pad1_b <=
4'b0000;

        end else
begin

            //

            if(((curr_y >= pad1_pos_y) && (curr_y <= pad1_pos_y +11'd20) && (curr_x >= (pad1_pos_x + 11'd0)) &&
(curr_x <= (pad1_pos_x + 11'd80)))) begin  // 4 by 4
space
                if ((((curr_y >= pad1_pos_y) && (curr_y <= pad1_pos_y +11'd20) && (((curr_x >= (pad1_pos_x)) &&
(curr_x <= (pad1_pos_x + 11'd2))) || ((curr_x >= (pad1_pos_x+ 11'd19)) && (curr_x <= (pad1_pos_x + 11'd22)))
                || ((curr_x >= (pad1_pos_x+ 11'd39)) && (curr_x <= (pad1_pos_x + 11'd42))) || ((curr_x >=
(pad1_pos_x+ 11'd59)) && (curr_x <= (pad1_pos_x +
11'd62)))
                || ((curr_x >= (pad1_pos_x+ 11'd79)) && (curr_x <= (pad1_pos_x + 11'd80))))) || (((curr_x >=
pad1_pos_x) && (curr_x <= pad1_pos_x +11'd80)) &&
                ((curr_y == (pad1_pos_y)) || (curr_y == (pad1_pos_y + 11'd1)) || (curr_y == (pad1_pos_y+ 11'd19)) ||
(curr_y == (pad1_pos_y + 11'd20)))))) begin
                    pad1_r <= 4'b1111; //boarder of
square

                    pad1_g <=
4'b1111;

                    pad1_b <=
4'b1111;

                end else
begin

                    pad1_r <= 4'b1100; // red component of the block at max value
                    pad1_g <= curr_x - pad1_pos_x >> 3; // reduces the range of the green value by 3 bits
                    pad1_b <= curr_y - pad1_pos_y >> 3; // // reduces the range of the blue value by 3
bits

                end

            end else
begin

            pad1_r <= 4'b0000;
//black

            pad1_g <=
4'b0000;

            pad1_b <=
4'b0000;

            end

        end

    end

assign drawpad1_r = (pad1_r != 4'b0000) ? pad1_r : 4'b0000;
assign drawpad1_g = (pad1_g != 4'b0000) ? pad1_g : 4'b0000;
assign drawpad1_b = (pad1_b != 4'b0000) ? pad1_b : 4'b0000;

endmodule
```

PAD2_DRAW module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
```

```
// Create Date: 29.04.2023 20:40:42
// Design Name:
// Module Name: pad2_draw
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module pad2_draw(
    input [10:0] pad2_pos_x,
    input [10:0] pad2_pos_y,
    output [3:0] drawpad2_r,
    output [3:0] drawpad2_g,
    output [3:0] drawpad2_b,
    input [10:0] curr_x,
    input [10:0] curr_y
    );

reg [3:0] pad2_r, pad2_b, pad2_g;
reg [3:0] bgpad2_r, bgpad2_b, bgpad2_g;


 //block
colour

    // 20 x
80

    always@*
begin

        if((curr_x < pad2_pos_x) || (curr_x > (pad2_pos_x + 11'd80))
||

          (curr_y < pad2_pos_y) || (curr_y > (pad2_pos_y + 11'd80)))
begin

            pad2_r <= 4'b0000;
//black

            pad2_g <=
4'b0000;

            pad2_b <=
4'b0000;

        end else
begin

            //

            if(((curr_y >= pad2_pos_y) && (curr_y <= pad2_pos_y +11'd20) && (curr_x >= (pad2_pos_x + 11'd0)) &&
(curr_x <= (pad2_pos_x + 11'd80)))) begin  // 4 by 4
space
                if ((((curr_y >= pad2_pos_y) && (curr_y <= pad2_pos_y +11'd20) && (((curr_x >= (pad2_pos_x)) &&
(curr_x <= (pad2_pos_x + 11'd2))) || ((curr_x >= (pad2_pos_x+ 11'd19)) && (curr_x <= (pad2_pos_x + 11'd22)))
                || ((curr_x >= (pad2_pos_x+ 11'd39)) && (curr_x <= (pad2_pos_x + 11'd42))) || ((curr_x >=
(pad2_pos_x+ 11'd59)) && (curr_x <= (pad2_pos_x +
11'd62)))
                || ((curr_x >= (pad2_pos_x+ 11'd79)) && (curr_x <= (pad2_pos_x + 11'd80))))) || (((curr_x >=
pad2_pos_x) && (curr_x <= pad2_pos_x +11'd80)) &&
                ((curr_y == (pad2_pos_y)) || (curr_y == (pad2_pos_y + 11'd1)) || (curr_y == (pad2_pos_y+ 11'd19)) ||
(curr_y == (pad2_pos_y + 11'd20)))))) begin
                    pad2_r <= 4'b1111; //boarder of
square

                    pad2_g <=
4'b1111;
```

```
                    pad2_b <=
4'b1111;

                end else
begin

                    pad2_r <= curr_y - pad2_pos_y >> 3; // // reduces the range of the red value by 3 bits
                    pad2_g <= curr_x - pad2_pos_x >> 3; // reduces the range of the green value by 3 bits
                    pad2_b <=
4'b1111;

                end

            end else
begin

            pad2_r <= 4'b0000;
//black

            pad2_g <=
4'b0000;

            pad2_b <=
4'b0000;

            end

        end

    end

assign drawpad2_r = (pad2_r != 4'b0000) ? pad2_r : bgpad2_r;
assign drawpad2_g = (pad2_g != 4'b0000) ? pad2_g : bgpad2_g;
assign drawpad2_b = (pad2_b != 4'b0000) ? pad2_b : bgpad2_b;

endmodule
```

BALL_DRAW module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 27.04.2023 21:47:31
// Design Name:
// Module Name: ball_draw
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ball_draw(

    input clk,
    input rst,
    input [10:0] ball_pos_x,
    input [10:0] ball_pos_y,
    output [3:0] drawball_r,
    output [3:0] drawball_g,
    output [3:0] drawball_b,
    input [10:0] curr_x,
    input [10:0] curr_y
    );

reg [3:0] drawball_r_r, drawball_r_b, drawball_r_g;
reg [3:0] bg_r, bg_b, bg_g;

//signals for image
parameter blk_size_x = 20, blk_size_y =20;   // 20 x 20
reg [8:0] addr;
```

```verilog
wire [11:0] rom_pixel; // read only memory


//image block of pokemon
always@(posedge clk) begin
    if(!rst) begin
        drawball_r_r <= 4'b0000;
        drawball_r_g <= 4'b0000;
        drawball_r_b <= 4'b0000;
        addr <= 0;
    end
    else begin
        if ((curr_x < ball_pos_x) || (curr_x > ball_pos_x + blk_size_x -1 ) ||
        (curr_y < ball_pos_y) || (curr_y > ball_pos_y + blk_size_y -1)) begin
            drawball_r_r <= 4'b0000;
            drawball_r_g <= 4'b0000;
            drawball_r_b <= 4'b0000;
        end
        else begin
            addr= (curr_y - ball_pos_y)*blk_size_x+(curr_x - ball_pos_x);
            drawball_r_r <= rom_pixel [11:8];
            drawball_r_g <= rom_pixel [7:4];
            drawball_r_b <= rom_pixel [3:0];

            if (addr == (blk_size_x*blk_size_y) -1)
                addr <= 0;
            else addr <= addr + 1;
        end
    end
end

assign drawball_r = (drawball_r_r != 4'b0000) ? drawball_r_r : bg_r;
assign drawball_g = (drawball_r_g != 4'b0000) ? drawball_r_g : bg_g;
assign drawball_b = (drawball_r_b != 4'b0000) ? drawball_r_b : bg_b;

// instantiate
blk_mem_gen_0 inst
(
.clka(clk),
.addra(addr),
.douta(rom_pixel)
);

endmodule
```

BACKGROUND_DRAW module

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 27.04.2023 21:47:31
// Design Name:
// Module Name:
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module background_draw(

    input clk,
    input rst,
    output [3:0] drawbg_r, // draw logo red
    output [3:0] drawbg_g, // draw logo green
    output [3:0] drawbg_b, // draw logo blue
    input [10:0] curr_x,
    input [10:0] curr_y
    );
```

```
reg [3:0] bg_r, bg_b, bg_g;

//signals for image
parameter sidlogo_size_x = 410, sidlogo_size_y = 92;   // 100 by 100
parameter sidlogopos_x =11'd515; //position of logo
parameter sidlogopos_y =11'd3; //position of logo
parameter sidopos_x =11'd59; //position of logo
parameter sidopos_y =11'd34; //position of logo
parameter sido_size_x = 242, sido_size_y = 31;
reg [15:0] sidlogo_addr;
reg [12:0] sido_addr;

parameter sidtpos_x =11'd1128; //position of logo
parameter sidtpos_y =11'd34; //position of logo
parameter sidt_size_x = 264, sidt_size_y = 31;
reg [12:0] sidt_addr;
wire [11:0] sidt_rom_pixel; // read only memory

wire [11:0] sidlogo_rom_pixel; // read only memory
wire [11:0] sido_rom_pixel; // read only memory
always@* begin
    if ((curr_x<11'd360) || (curr_x > 11'd1080) || (curr_y<11'd100) || (curr_y > 11'd890)) begin
        bg_r <= curr_x-curr_y>> 1;
        bg_g <= 4'b1101;
        bg_b <= curr_y-curr_x>> 1;
    end else begin
        bg_r <= 4'b0000;
        bg_g <= 4'b0000;
        bg_b <= 4'b0000;
    end
end
reg [3:0] draw_out_r;
reg [3:0] draw_out_g;
reg [3:0] draw_out_b;
//image block
always@(posedge clk) begin
    if(!rst) begin
        draw_out_r <= bg_r;
        draw_out_g <= bg_g;
        draw_out_b <= bg_b;
        sidlogo_addr <= 0;
        sido_addr <= 0;
    end
    else begin
        if ((curr_x < sidlogopos_x) || (curr_x > sidlogopos_x + sidlogo_size_x -1 ) ||
        (curr_y < sidlogopos_y) || (curr_y > sidlogopos_y + sidlogo_size_y -1)) begin
            if ((curr_x < sidopos_x) || (curr_x > sidopos_x + sido_size_x -1 ) ||
            (curr_y < sidopos_y) || (curr_y > sidopos_y + sido_size_y -1)) begin
                if ((curr_x < sidtpos_x) || (curr_x > sidtpos_x + sidt_size_x -1 ) ||
                    (curr_y < sidtpos_y) || (curr_y > sidtpos_y + sidt_size_y -1)) begin
                        draw_out_r <= bg_r;
                        draw_out_g <= bg_g;
                        draw_out_b <= bg_b;
                end else begin
                    sidt_addr= (curr_y - sidtpos_y)*sidt_size_x+(curr_x - sidtpos_x);
                    draw_out_r <= sidt_rom_pixel [11:8];
                    draw_out_g <= sidt_rom_pixel [7:4];
                    draw_out_b <= sidt_rom_pixel [3:0];

                    if (sidt_addr == (sidt_size_x*sidt_size_y) -1)
                        sidt_addr <= 0;
                    else sidt_addr <= sidt_addr + 1;
                    end
            end
            else begin
                sido_addr= (curr_y - sidopos_y)*sido_size_x+(curr_x - sidopos_x);
                draw_out_r <= sido_rom_pixel [11:8];
                draw_out_g <= sido_rom_pixel [7:4];
                draw_out_b <= sido_rom_pixel [3:0];

                if (sido_addr == (sido_size_x*sido_size_y) -1)
                    sido_addr <= 0;
                else sido_addr <= sido_addr + 1;
            end
        end
        else begin
            sidlogo_addr= (curr_y - sidlogopos_y)*sidlogo_size_x+(curr_x - sidlogopos_x);
            if ((sidlogo_rom_pixel [11:8]==4'b1111)&&(sidlogo_rom_pixel [7:4]==4'b1111)&&(sidlogo_rom_pixel
[3:0]==4'b1111)) begin
                draw_out_r <= bg_r;
                draw_out_g <= bg_g;
                draw_out_b <= bg_b;
            end else begin
                draw_out_r <= sidlogo_rom_pixel [11:8];
                draw_out_g <= sidlogo_rom_pixel [7:4];
                draw_out_b <= sidlogo_rom_pixel [3:0];
            end


            if (sidlogo_addr == (sidlogo_size_x*sidlogo_size_x) -1)
                sidlogo_addr <= 0;
            else sidlogo_addr <= sidlogo_addr + 1;
        end
    end
end

assign drawbg_r = draw_out_r;
assign drawbg_g = draw_out_g;
assign drawbg_b = draw_out_b;

// instantiate
blk_mem_gen_1 sidlogo
(
.clka(clk),
.addra(sidlogo_addr),
.douta(sidlogo_rom_pixel)
);

blk_mem_gen_2 sidobar
(
.clka(clk),
.addra(sido_addr),
.douta(sido_rom_pixel)
);

blk_mem_gen_7 sidtbar
(
.clka(clk),
.addra(sidt_addr),
.douta(sidt_rom_pixel)
);
endmodule
```

LOADINGPAGE_DRAW module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01.05.2023 19:52:52
// Design Name:
// Module Name: loadingpage_draw
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module loadingpage_draw(
    input clk,
    input rst,
    output [3:0] drawload_r, // draw logo red
    output [3:0] drawload_g, // draw logo green
    output [3:0] drawload_b, // draw logo blue
    input [10:0] curr_x,
    input [10:0] curr_y
    );

    parameter load_size_x = 527, load_size_y = 119;   // 100 by 100
    parameter loadpos_x =11'd446; //position of logo
    parameter loadpos_y =11'd318; //position of logo
    reg [15:0] load_addr;
    wire [11:0] load_rom_pixel; // read only memory
    reg [3:0] draw_out_r;
    reg [3:0] draw_out_g;
    reg [3:0] draw_out_b;

    always@(posedge clk) begin
        if(!rst) begin
            draw_out_r <= 4'b0000;
            draw_out_g <= 4'b0000;
            draw_out_b <= 4'b0000;
            load_addr <= 0;
        end
        else begin
            if ((curr_x < loadpos_x) || (curr_x > loadpos_x + load_size_x -1 ) ||
            (curr_y < loadpos_y) || (curr_y > loadpos_y + load_size_y -1)) begin
                draw_out_r <= 4'b0000;
                draw_out_g <= 4'b0000;
                draw_out_b <= 4'b0000;
            end
            else begin
                load_addr= (curr_y - loadpos_y)*load_size_x+(curr_x - loadpos_x);
                draw_out_r <= load_rom_pixel [11:8];
                draw_out_g <= load_rom_pixel [7:4];
                draw_out_b <= load_rom_pixel [3:0];

                if (load_addr == (load_size_x*load_size_x) -1)
                        load_addr <= 0;
                    else load_addr <= load_addr + 1;
                end
        end
    end

    assign drawload_r = draw_out_r;
    assign drawload_g = draw_out_g;
    assign drawload_b = draw_out_b;

    // instantiate
    blk_mem_gen_3 loadtop
    (
    .clka(clk),
    .addra(load_addr),
    .douta(load_rom_pixel)
    );

endmodule
```

HOMEPAGE_DRAW module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01.05.2023 19:52:52
// Design Name:
// Module Name: homepage_draw
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module homepage_draw(
    input clk,
    input rst,
    output [3:0] drawhomepage_r, // draw logo red
    output [3:0] drawhomepage_g, // draw logo green
    output [3:0] drawhomepage_b, // draw logo blue
    input [10:0] curr_x,
    input [10:0] curr_y
```

```verilog
    );

    parameter homepage_size_x = 400, homepage_size_y = 159;   // 100 by 100
    parameter homepagepos_x =11'd520; //position of logo
    parameter homepagepos_y =11'd370; //position of logo

    reg [15:0] homepage_addr;
    wire [11:0] homepage_rom_pixel; // read only memory

    reg [3:0] draw_out_r;
    reg [3:0] draw_out_g;
    reg [3:0] draw_out_b;

    always@(posedge clk) begin
        if(!rst) begin
            draw_out_r <= 4'b0000;
            draw_out_g <= 4'b0000;
            draw_out_b <= 4'b0000;
            homepage_addr <= 0;
        end
        else begin
            if ((curr_x < homepagepos_x) || (curr_x > homepagepos_x + homepage_size_x -1 ) ||
            (curr_y < homepagepos_y) || (curr_y > homepagepos_y + homepage_size_y -1)) begin
                draw_out_r <= 4'b0000;
                draw_out_g <= 4'b0000;
                draw_out_b <= 4'b0000;
            end else begin
                homepage_addr= (curr_y - homepagepos_y)*homepage_size_x+(curr_x - homepagepos_x);
                draw_out_r <= homepage_rom_pixel [11:8];
                draw_out_g <= homepage_rom_pixel [7:4];
                draw_out_b <= homepage_rom_pixel [3:0];

                if (homepage_addr == (homepage_size_x*homepage_size_y) -1)
                    homepage_addr <= 0;
                else homepage_addr <= homepage_addr + 1;
            end
        end
    end

    assign drawhomepage_r = draw_out_r;
    assign drawhomepage_g = draw_out_g;
    assign drawhomepage_b = draw_out_b;

    // instantiate
    blk_mem_gen_4 homepage
    (
    .clka(clk),
    .addra(homepage_addr),
    .douta(homepage_rom_pixel)
    );

endmodule
```

PAD1WINPAGE_DRAW module

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01.05.2023 19:52:52
// Design Name:
// Module Name: pad1winpage_draw
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module pad1winpage_draw(
    input clk,
    input rst,
    output [3:0] drawpad1win_r, // draw logo red
    output [3:0] drawpad1win_g, // draw logo green
    output [3:0] drawpad1win_b, // draw logo blue
    input [10:0] curr_x,
    input [10:0] curr_y
    );

    parameter p1win_size_x = 166, p1win_size_y = 55;   // 100 by 100
    parameter p1winpos_x =11'd637; //position of logo
    parameter p1winpos_y =11'd422; //position of logo

    reg [13:0] p1win_addr;
    wire [11:0] p1win_rom_pixel; // read only memory

    reg [3:0] draw_out_r;
    reg [3:0] draw_out_g;
    reg [3:0] draw_out_b;

    always@(posedge clk) begin
        if(!rst) begin
            draw_out_r <= 4'b0000;
            draw_out_g <= 4'b0000;
            draw_out_b <= 4'b0000;
            p1win_addr <= 0;
        end
        else begin
            if ((curr_x < p1winpos_x) || (curr_x > p1winpos_x + p1win_size_x -1 ) ||
            (curr_y < p1winpos_y) || (curr_y > p1winpos_y + p1win_size_y -1)) begin
                draw_out_r <= 4'b0000;
                draw_out_g <= 4'b0000;
                draw_out_b <= 4'b0000;
            end else begin
                p1win_addr= (curr_y - p1winpos_y)*p1win_size_x+(curr_x - p1winpos_x);
                draw_out_r <= p1win_rom_pixel [11:8];
                draw_out_g <= p1win_rom_pixel [7:4];
                draw_out_b <= p1win_rom_pixel [3:0];

                if (p1win_addr == (p1win_size_x*p1win_size_y) -1)
                    p1win_addr <= 0;
                else p1win_addr <= p1win_addr + 1;
            end
        end
    end

    assign drawpad1win_r = draw_out_r;
    assign drawpad1win_g = draw_out_g;
    assign drawpad1win_b = draw_out_b;

    // instantiate
    blk_mem_gen_5 p1win
    (
    .clka(clk),
    .addra(p1win_addr),
    .douta(p1win_rom_pixel)
    );

endmodule
```

PAD2WINPAGE_DRAW module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01.05.2023 19:52:52
// Design Name:
// Module Name: pad2winpage_draw
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

```
module pad2winpage_draw(
    input clk,
    input rst,
    output [3:0] drawpad2win_r, // draw logo red
    output [3:0] drawpad2win_g, // draw logo green
    output [3:0] drawpad2win_b, // draw logo blue
    input [10:0] curr_x,
    input [10:0] curr_y
    );

    parameter p2win_size_x = 172, p2win_size_y = 55;   // 100 by 100
    parameter p2winpos_x =11'd634; //position of logo
    parameter p2winpos_y =11'd422; //position of logo

    reg [13:0] p2win_addr;
    wire [11:0] p2win_rom_pixel; // read only memory

    reg [3:0] draw_out_r;
    reg [3:0] draw_out_g;
    reg [3:0] draw_out_b;

    always@(posedge clk) begin
        if(!rst) begin
            draw_out_r <= 4'b0000;
            draw_out_g <= 4'b0000;
            draw_out_b <= 4'b0000;
            p2win_addr <= 0;
        end
        else begin
            if ((curr_x < p2winpos_x) || (curr_x > p2winpos_x + p2win_size_x -1 ) ||
            (curr_y < p2winpos_y) || (curr_y > p2winpos_y + p2win_size_y -1)) begin
                draw_out_r <= 4'b0000;
                draw_out_g <= 4'b0000;
                draw_out_b <= 4'b0000;
            end else begin
                p2win_addr= (curr_y - p2winpos_y)*p2win_size_x+(curr_x - p2winpos_x);
                draw_out_r <= p2win_rom_pixel [11:8];
                draw_out_g <= p2win_rom_pixel [7:4];
                draw_out_b <= p2win_rom_pixel [3:0];

                if (p2win_addr == (p2win_size_x*p2win_size_y) -1)
                    p2win_addr <= 0;
                else p2win_addr <= p2win_addr + 1;
            end
        end
    end

    assign drawpad2win_r = draw_out_r;
    assign drawpad2win_g = draw_out_g;
    assign drawpad2win_b = draw_out_b;

    // instantiate
    blk_mem_gen_6 p2win
    (
    .clka(clk),
    .addra(p2win_addr),
    .douta(p2win_rom_pixel)
    );

endmodule
```

VGA module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 25.04.2023 13:41:51
// Design Name:
// Module Name: vga
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
```

```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module vga(
    input [2:0]state,
    input clk,
    input rst,
    input [3:0] drawload_r,
    input [3:0] drawload_g,
    input [3:0] drawload_b,
    input [3:0] drawhomepage_r,
    input [3:0] drawhomepage_g,
    input [3:0] drawhomepage_b,
    input [3:0] drawpad1win_r,
    input [3:0] drawpad1win_g,
    input [3:0] drawpad1win_b,
    input [3:0] drawpad2win_r,
    input [3:0] drawpad2win_g,
    input [3:0] drawpad2win_b,
    input [3:0] drawpad1_r,
    input [3:0] drawpad1_g,
    input [3:0] drawpad1_b,
    input [3:0] drawpad2_r, // player 2 block bar
    input [3:0] drawpad2_g, // player 2 block bar
    input [3:0] drawpad2_b, // player 2 block bar
    input [3:0] drawball_r, // fraw sprite of the ball
    input [3:0] drawball_g, // draw sprite of the ball
    input [3:0] drawball_b, // draw sprite of the ball
    input [3:0] drawbg_r, // draw sprite for logo
    input [3:0] drawbg_g, // draw sprite for logo
    input [3:0] drawbg_b, // draw sprite for logo
    output [10:0] curr_x,
    output [10:0] curr_y,
    output [3:0] pix_r,
    output [3:0] pix_g,
    output [3:0] pix_b,
    output hsync,
    output vsync
    );

    // internal signals
reg [10:0] hcount;
reg [9:0] vcount;
reg [10:0] curr_x_r;
reg [10:0] curr_y_r;

//wire pixclk;
wire line_end = (hcount == 11'd1903);
wire frame_end = (vcount == 10'd931);
wire display_region;

// hsync vsync assining combinational logic
assign hsync = ((hcount >= 11'd0) && (hcount <= 11'd151));
assign vsync = ((vcount >= 10'd0) && (vcount <= 10'd2));

assign display_region = ((hcount >= 11'd384) && (hcount <= 11'd1823) && (vcount >= 10'd31) && (vcount <= 10'd930));

reg [3:0] out_drawpad1_r;
reg [3:0] out_drawpad1_g;
reg [3:0] out_drawpad1_b;

always@(posedge clk) begin
    if (!rst) begin
        out_drawpad1_r <= 4'b0000;
        out_drawpad1_g <= 4'b0000;
        out_drawpad1_b <= 4'b0000;
    end else begin
        case (state)
            3'd0: begin
                    out_drawpad1_r <= drawload_r;
                    out_drawpad1_g <= drawload_g;
                    out_drawpad1_b <= drawload_b;
                  end
            3'd1: begin
                    out_drawpad1_r <= drawhomepage_r;
                    out_drawpad1_g <= drawhomepage_g;
                    out_drawpad1_b <= drawhomepage_b;
                  end
            (3'd2): begin
                    if ((drawbg_r==4'b0000)&&(drawbg_g==4'b0000)&&(drawbg_b==4'b0000)) begin
                        if ((drawpad1_r==4'b0000)&&(drawpad1_g==4'b0000)&&(drawpad1_b==4'b0000)) begin
                            if ((drawpad2_r==4'b0000)&&(drawpad2_g==4'b0000)&&(drawpad2_b==4'b0000)) begin
                                out_drawpad1_r <= drawball_r;
                                out_drawpad1_g <= drawball_g;
                                out_drawpad1_b <= drawball_b;
                            end else  begin
                                out_drawpad1_r <= drawpad2_r;
                                out_drawpad1_g <= drawpad2_g;
                                out_drawpad1_b <= drawpad2_b;
                            end
                        end else  begin
                                out_drawpad1_r <= drawpad1_r;
                                out_drawpad1_g <= drawpad1_g;
                                out_drawpad1_b <= drawpad1_b;
                        end
                    end else begin
                        out_drawpad1_r <= drawbg_r;
                        out_drawpad1_g <= drawbg_g;
                        out_drawpad1_b <= drawbg_b;
                    end

                  end
            (3'd3): begin
                    if ((drawbg_r==4'b0000)&&(drawbg_g==4'b0000)&&(drawbg_b==4'b0000)) begin
                        if ((drawpad1_r==4'b0000)&&(drawpad1_g==4'b0000)&&(drawpad1_b==4'b0000)) begin
                            if ((drawpad2_r==4'b0000)&&(drawpad2_g==4'b0000)&&(drawpad2_b==4'b0000)) begin
                                out_drawpad1_r <= drawball_r;
                                out_drawpad1_g <= drawball_g;
                                out_drawpad1_b <= drawball_b;
                            end else  begin
                                out_drawpad1_r <= drawpad2_r;
                                out_drawpad1_g <= drawpad2_g;
                                out_drawpad1_b <= drawpad2_b;
                            end
                        end else  begin
                                out_drawpad1_r <= drawpad1_r;
                                out_drawpad1_g <= drawpad1_g;
                                out_drawpad1_b <= drawpad1_b;
                        end
                    end else begin
                        out_drawpad1_r <= drawbg_r;
                        out_drawpad1_g <= drawbg_g;
                        out_drawpad1_b <= drawbg_b;
                    end

                  end
            3'd4: begin
                    out_drawpad1_r <= drawpad2win_r;
                    out_drawpad1_g <= drawpad2win_g;
                    out_drawpad1_b <= drawpad2win_b;
                  end
            3'd5: begin
                    out_drawpad1_r <= drawpad1win_r;
                    out_drawpad1_g <= drawpad1win_g;
                    out_drawpad1_b <= drawpad1win_b;
                  end
            default: begin

            end
        endcase
    end
end

// pix assign combinational for block
assign pix_r = (display_region) ? out_drawpad1_r : 4'b0000;
// when we're inside the display region and depedninng on the value of sw - assert the pix_r values
assign pix_g = (display_region) ? out_drawpad1_g : 4'b0000;
assign pix_b = (display_region) ? out_drawpad1_b : 4'b0000;


// hcount synchronous
    always@(posedge clk) begin // clk for controlling synchronous always block
        if (!rst) // have to set the value of hcount when we rst
            hcount <=  11'd0;  // setting all the 11 bits of h count to 0
        else begin
            if (line_end) // a signal that will control the wrap around at max
```

```verilog
                hcount <= 11'd0;
            else
                hcount <= hcount + 11'd1;
        end
     end

// vcount synchronous
    always@(posedge clk) begin
        if (!rst)
            vcount <= 10'd0;
        else begin
            if (line_end)
                    vcount <= vcount + 10'd1;
            else begin
                if (frame_end)
                vcount <= 10'd0;
            end
        end
    end

 // curr_x synchronous
    always@(posedge clk) begin
        if (!rst)
            curr_x_r <= 11'd0;
        else begin
            if(hcount >= 11'd384 && hcount<= 11'd1824) begin
                curr_x_r <= curr_x_r + 11'd1;
            end else begin
                curr_x_r <= 11'd0; // every time ur outside the limit 384 to 1824 then currxr is set to 0
            end
        end
    end

 // curr_y synchronous
     always@(posedge clk) begin
        if (!rst) begin
            curr_y_r <= 11'd0;
        end else begin
            if (line_end) begin
            //else begin
                if(vcount >= 11'd31 && vcount<= 11'd931) begin
                    curr_y_r <= curr_y_r + 11'd1;
                end else begin
                    curr_y_r <= 11'd0;
                end
            end
          end
    end

 assign curr_x = curr_x_r;
 assign curr_y = curr_y_r;

endmodule
```

SEGINTERFACE module

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 13.04.2023 21:34:13
// Design Name:
// Module Name: seginterface
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module seginterface(
        input clk, rst, // takes input from clock and from reset - used for controlling the synchronous logic
        // clock - base our changes on a moving clock edge, allows us to create changes that occur in sync with
        // a clock
        input [3:0] dig7, dig6, dig5, dig4, dig3, dig2, dig1, dig0, // allows us to take 8, 4 bit wire digits
        output a, b, c, d, e, f, g, // outputs the seven segment elements
        output [7:0] an // anode for selecting each of the 7 segment display
    );

    wire led_clk;
    reg [3:0] dig_sel;
    reg [28:0] clk_count = 11'd0;
    // clock count is used to store how many times the clock changes
    always @(posedge clk) begin
        clk_count <= clk_count + 1'b1;
    end

    assign led_clk = clk_count[16]; // LED clock is sampled at the 16th value from the clock count
    reg [7:0] led_strobe = 8'b11111110;
    always @(posedge led_clk) begin
        led_strobe <= {led_strobe[6:0],led_strobe[7]};
    end

    assign an = led_strobe;

    reg [2:0] led_index = 3'd0;
    always @(posedge led_clk) begin
        led_index <= led_index + 3'd1;
    end


    always@*
        case (led_index)
            3'd0: dig_sel = dig0;
            3'd1: dig_sel = dig1;
            3'd2: dig_sel = dig2;
            3'd3: dig_sel = dig3;
            3'd4: dig_sel = dig4;
            3'd5: dig_sel = dig5;
            3'd6: dig_sel = dig6;
            3'd7: dig_sel = dig7;
        endcase

    sevenseg inst_1 (.num(dig_sel), .a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g));

endmodule
```

SEVENSEG module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 13.04.2023 21:52:26
// Design Name:
// Module Name: sevenseg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


`timescale 1ns / 1ps

module sevenseg(
    input [3:0] num,
    output a,
    output b,
    output c,
    output d,
    output e,
    output f,
    output g
    );

    reg [6:0] intseg;
    assign {a,b,c,d,e,f,g} = ~intseg;

    always@*
    begin
        case(num)
            4'h0: intseg = 7'b1111110;
            4'h1: intseg = 7'b0110000;
            4'h2: intseg = 7'b1101101;
            4'h3: intseg = 7'b1111001;
            4'h4: intseg = 7'b0110011;
            4'h5: intseg = 7'b1011011;
            4'h6: intseg = 7'b1011111;
            4'h7: intseg = 7'b1110000;
            4'h8: intseg = 7'b1111111;
            4'h9: intseg = 7'b1111011;
            4'ha: intseg = 7'b1110111;
            4'hb: intseg = 7'b0011111;
            4'hc: intseg = 7'b1001110;
            4'hd: intseg = 7'b0111101;
            4'he: intseg = 7'b1001111;
            4'hf: intseg = 7'b1000111;
        endcase
    end

endmodule
```