EEL 4712C - Digital Design: Lab Report 5

Cole Rottenberg
11062528

April 14th, 2024

```
-- This is a test VHDL code block
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity test is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : out  STD_LOGIC);
end test;
```

Listing 1: Test VHDL Code

Lab Report

Problem Statement

Lab 5 builds off of lab 4 by turning our simple VGA lab with a bouncing ball into the game ”Pong”. There are four main parts of the game outlined by the lab assignment:

1. **Start and Game Over**: The Start scren should display the word ”PONG” in the middle of the screen. The Start screen should also display a ”Press B1” message at the bottom of the screen. The game should start when the user presses button 1. The game should end when one player reaches 11 points. The game should display a ”Game Over” message when the game ends.
2. **Ball Movement**: The ball should move in a straight line at a constant speed. It should bounce off the top and bottom of the scrren. It should also bounce off the paddles. When the ball hits the left or right side of the screen, the ball should be sent back the to middle of the screen.
3. **Paddle Movement**: The paddles should move up and down with the push buttons. The paddles should not be able to move off the screen.
4. **Scoreboard**: The game should keep track of the score. The game should end when one player reaches 11 points. The scoring of the game should be done using a bitmap of characters displayed on the screen.

Inputs

The inputs to control the game are the push buttons on the DE10-Lite board. Button 1 is used to start the game and button 2 is used to reset the game. The first two switches are used to control the ledt paddle and the third and fourth switches are used to control the right paddle. The ball moves at a constant speed and does not require any user input.

Outputs

The outputs are the VGA display output. This includes: VGA_HSYNC, VGA_VSYNC, VGA_R, VGA_G, and VGA_B. The VGA display will display the game screen, the paddles, the ball, the score, and the game over screen. The VGA display works best with a 640x480 resolution.

Function

The function of the system is broken into three main states of the game: the start screen, the game screen, and the game over screen. The start screen displays the word ”PONG” in the middle of the screen and a ”Press B1” message at the bottom of the screen. The game screen displays the paddles, the ball, and the score. The game over screen displays a ”Game Over” message. The game starts when the user presses button 1 and ends when one player reaches 11 points.

Start Screen: The start screen displays the word ”PONG” in the middle of the screen and a ”Press B1” message at the bottom of the screen. The game starts when the user presses button 1. **Game Screen**: The game screen displays the paddles, the ball, and the score. The ball moves in a diagonal line at a constant speed. It bounces off the top and bottom of the screen and the paddles. When the ball hits the left or right side of the screen, the ball is sent back to the middle of the screen. The paddles move up and down with the push buttons. The game keeps track of the score and ends when one player reaches 11 points. **Game Over Screen**: The game over screen displays a ”Game Over” message. The game ends when one player reaches 11 points.

Design

Components

The components used in the design are the VGA_sync module, a clock divider, and the a vga module. The design mimics the previous lab design with a change in the **vga.vhd** file to include the paddles and the ball. The vga module holds most of the logic for the game and incorporates the other two previously mentioned modules. The VGA_sync module is used to generate the horizontal and vertical sync signals for the VGA display. The clock divider is used to generate the 25MHz clock signal from the 50MHz clock signal. The clock divider is also used to create a slow clk that control the timing of the movement.

Signals

The signals that connect the components are the clk_50MHz signal, the clk_25MHz signal, the slow_clk signal, the VGA_HSYNC signal, the VGA_VSYNC signal, the VGA_R signal, the VGA_G signal, and the VGA_B signal. The clk_50MHz signal is the 50MHz clock signal from the DE10-Lite board. The clk_25MHz signal is the 25MHz clock signal generated by the clock divider. The slow_clk signal is the slow clock signal generated by the clock divider. The VGA_HSYNC signal is the horizontal sync signal generated by the VGA_sync module. The VGA_VSYNC signal is the vertical sync signal generated by the VGA_sync module. The VGA_R signal is the red signal generated by the vga module. The VGA_G signal is the green signal generated by the vga module. The VGA_B signal is the blue signal generated by the vga module.

Algorithms

The algorithms used in the design are the same as the previous lab with the addition of the paddles and the ball. The paddles move up and down with the push buttons. The ball moves in a diagonal line at a constant speed. It bounces off the top and bottom of the screen and the paddles. When the ball hits the left or right side of the screen, the ball is sent back to the middle of the screen. The game keeps track of the score and ends when one player reaches 11 points. In addition to the conditional logic used by the VGA, another component of the logic is the state machine responsible for controlling what to display on the screen.

The design flows from the state machine which determinse waht to display on the screen through a set of conditionals within the display logic. The movement of both the paddles and the ball are controlled by the push buttons and the slow clock signal within three seperate processes. The score is only modified by that draw process when the ball hits the left or right side of the screen. The paddles are only modified by the draw process when the push buttons are pressed.

Flowchart

Implementation

The first block of code describes all the constants and signals needed for the game. The constants are the size of the ball, the speed of the ball, the size of the paddles, the speed of the paddles, the maximum and minimum values for the paddles, the maximum values for the x and y coordinates of the ball, and the maximum values for the x and y coordinates of the paddles. The signals are the horizontal and vertical count signals, the slow clock signal, the internal signals for the horizontal and vertical sync signals, the internal signal for the video on signal, the x and y coordinates of the ball, the x and y coordinates of the paddles, the scores of the players, the bitmaps for the scores, the bitmaps for the letters, and the game state. They can be seen in Listing 2.

The second block of code describes the state machine for the game. The state machine is responsible for controlling the game state and this helps with logic within the display process. This code can be seen in Listing 3.

The third block of code describes the movement processes for the ball and the paddles. The ball moves in a diagonal line at a constant speed. It bounces off the top and bottom of the screen and the paddles. When the ball hits the left or right side of the screen, the ball is sent back to the middle of the screen. The paddles move up and down with the push buttons. The paddles are not able to move off the screen. The ball and the paddles are only modified by the draw process when the game is in the playing state. The score is only modified by the draw process when the ball hits the left or right side of the screen. The paddles are only modified by the draw process when the push buttons are pressed. This code can be seen in Listing 4.

The fourth block of code describes the draw process. The draw process is responsible for displaying the game screen, the paddles, the ball, the score, and the game over screen. The draw process is dependent on the game state and the video on signal. The draw process is also dependent on the x and y coordinates of the ball, the x and y coordinates of the paddles, the x and y coordinates of the scores, and the x and y coordinates of the letters. The draw process is also dependent on the bitmaps for the scores and the letters. The draw process is also dependent on the horizontal and vertical count signals. The draw process is also dependent on the slow clock signal. The draw process is also dependent on the red, green, and blue signals. The draw process is also dependent on the size of the bitmaps. The draw process is also dependent on the maximum values for the x and y coordinates of the ball, the paddles, the scores, and the letters. The draw process is also dependent on the maximum values for the x and y coordinates of the screen. The draw process is also dependent on the maximum values for the x and y coordinates of the paddles. The draw process is also dependent on the maximum values for the x and y coordinates of the scores. The draw process is also dependent on the maximum values for the x and y coordinates of the letters. The draw process is also dependent on the maximum values for the x and y coordinates of the screen. The code can be seen in Listing 5.

Testing

The testing part of the code mainly took process within a basic testbench from the previous lab which was used just to manually monitor the changing of states and other conditions. Other methods of testing were not necessary due to the nature of working with internals of the game. One piece of troublesome code was around the movement process of scoring when the ball touched the edge of the screen. This was fixed allowing for the use of variables to store the new position of the ball and the direction of the ball.

Conclusions

The problems encountered in this lab centered around conditionals within states and the implementation of a bitmap for the scores as well as lettering in the beginning and end of the game. The success of the lab was in the implementation of the game and the movement of the ball and paddles. From lab 4 to the initial steps of implementing paddles were straightforward. The next steps would be to implement a more complex scoring system and to add more features to the game. The most complex piece to the project centered around the use of bitmaps and how I could access the boolean values of a bitmap in certain display locations. This development was the most challenging as well and the most satisfying after completion. This lab is entirely complete and the game is fully functional, however some of the bitmaps could be improved and the slow clock could be adjusted to make the game more challenging as well as differing the paddles and balls speed making it more challenging.

Appendix

```

    signal v_count : std_logic_vector(COUNT_RANGE);
    signal h_count : std_logic_vector(COUNT_RANGE);
    -- Clocks
    signal slow_clk : std_logic := '0'; -- Slow clock
    -- Internal Temp signals
    signal temp_h_sync, temp_v_sync, temp_video_on : std_logic;
␣␣␣␣-- x and y coordinates of ball
    signal x_pos, y_pos : integer := 200; -- 200 is the center of the screen
    signal mov_x, mov_y : integer := 1;
    -- Constants of the ball
    constant speed : integer := 2;
    constant size : integer := 64;
    constant X_MAX : integer := 638;
    constant Y_MAX : integer := 478;
    -- Constants for the paddles
    constant PADDLE_WIDTH : integer := 10;
    constant PADDLE_HEIGHT : integer := 50;
    constant PADDLE_SPEED : integer := 2;
    constant PADDLE_MAX : integer := 428;
    constant PADDLE_MIN : integer := 0;
    -- x and y coordinates for each paddle, one on the left and one on the right
    -- Paddle 1 is on the left, Paddle 2 is on the right
    signal x_pos_p1 : integer := 0;
    signal y_pos_p1 : integer := 200;
    signal x_pos_p2 : integer := 638 - PADDLE_WIDTH;
    signal y_pos_p2 : integer := 200;

    signal P1_score : integer := 0;
    signal P2_score : integer := 0;

-- BITMAPS for PONG, P1 and P2 scores, 0-9 , game over and start
-- using the 2D array to store the bitmaps
-- The dimensions of the array are 8x4
-- PONG

    constant BM_SIZE : integer := 5;

    type bitmap is array(4 downto 0) of std_logic_vector(4 downto 0);

    constant ZERO : bitmap := (
        "11111",
        "10001",
        "10001",
        "10001",
        "11111"
    );

    constant ONE : bitmap := (
        "00100",
        "01100",
        "10100",
        "00100",
        "11111"
    );

    constant TWO : bitmap := (
        "11111",
        "00001",
        "11111",
        "10000",
        "11111"
    );

    constant THREE : bitmap := (
        "11111",
        "00001",
        "11111",
        "00001",
        "11111"
    );

    constant FOUR : bitmap := (
        "10001",
        "10001",
        "11111",
        "00001",
        "00001"
    );

    constant FIVE : bitmap := (
        "11111",
        "10000",
        "11111",
        "00001",
        "11111"
    );

    constant SIX : bitmap := (
        "11111",
        "10000",
        "11111",
        "10001",
        "11111"
    );

```

```vhdl
    constant SEVEN : bitmap := (
        "11111",
        "00001",
        "00010",
        "00100",
        "01000"
    );

    constant EIGHT : bitmap := (
        "11111",
        "10001",
        "11111",
        "10001",
        "11111"
    );

    constant NINE : bitmap := (
        "11111",
        "10001",
        "11111",
        "00001",
        "11111"
    );

    -- using a function to convert score to a bitmap
    function score_to_bitmap(score : integer) return bitmap is
    begin
        case score is
            when 0 =>
                return ZERO;
            when 1 =>
                return ONE;
            when 2 =>
                return TWO;
            when 3 =>
                return THREE;
            when 4 =>
                return FOUR;
            when 5 =>
                return FIVE;
            when 6 =>
                return SIX;
            when 7 =>
                return SEVEN;
            when 8 =>
                return EIGHT;
            when 9 =>
                return NINE;
            when others =>
                return ZERO;
        end case;
    end score_to_bitmap;

    -- Adding the PONG Letters
    constant P : bitmap := (
        "11111",
        "10001",
        "10001",
        "11111",
        "10000"
    );

    constant O : bitmap := (
        "11111",
        "10001",
        "10001",
        "10001",
        "11111"
    );

    constant N : bitmap := (
        "10001",
        "11001",
        "10101",
        "10011",
        "10001"
    );

    constant G : bitmap := (
        "11111",
        "10000",
        "10011",
        "10001",
        "11111"
    );

    -- Adding the WIN Letters
    constant W : bitmap := (
        "10001",
        "10001",
        "10001",
        "10101",
        "11011"
    );

    constant I : bitmap := (
        "11111",
        "00100",
        "00100",
        "00100",
        "11111"
    );

    constant S : bitmap := (
        "11111",
        "10000",
        "11111",
        "00001",
        "11111"
    );
    -- Position of the bitmap

    -- Position of the P1 score
    signal x_pos_p1_score : integer := 100;
    signal y_pos_p1_score : integer := 100;

    -- Position of the P2 score
    signal x_pos_p2_score : integer := 500;
    signal y_pos_p2_score : integer := 100;

    -- Position of the PONG letters
    -- Letter P
    signal x_pos_p : integer := 250;
    signal y_pos_p : integer := 100;

    -- Letter O
    signal x_pos_o : integer := 275;
    signal y_pos_o : integer := 100;

    -- Letter N
    signal x_pos_n : integer := 300;
    signal y_pos_n : integer := 100;

    -- Letter G
    signal x_pos_g : integer := 325;
    signal y_pos_g : integer := 100;

    -- Adding Letters for Wins
    -- Position of the P1 win
    signal x_pos_p1_win : integer := 250;
    signal y_pos_p1_win : integer := 200;

    -- Position of the P2 win
    signal x_pos_p2_win : integer := 500;
    signal y_pos_p2_win : integer := 200;

    -- Position of the WIN letters
    -- Letter W
    signal x_pos_w : integer := 300;
    signal y_pos_w : integer := 200;

    -- Letter I
    signal x_pos_i : integer := 350;
    signal y_pos_i : integer := 200;

    -- Letter N
    signal x_pos_n_win : integer := 400;
    signal y_pos_n_win : integer := 200;

    -- Letter S
    signal x_pos_s : integer := 450;
    signal y_pos_s : integer := 200;

    -- Position of the START letters
    -- Letter S
    signal x_pos_s_start : integer := 300;
    signal y_pos_s_start : integer := 300;

    -- Letter T
    signal x_pos_t : integer := 350;
    signal y_pos_t : integer := 300;

    -- Letter A
    signal x_pos_a : integer := 400;
    signal y_pos_a : integer := 300;

    -- Letter R
    signal x_pos_r : integer := 450;
    signal y_pos_r : integer := 300;

    -- Letter T
    signal x_pos_t_start : integer := 500;
    signal y_pos_t_start : integer := 300;

    -- Letter S
    signal x_pos_s_start2 : integer := 550;
    signal y_pos_s_start2 : integer := 300;

    -- Defining Game States
    -- 0: Start
    -- 1: Playing
    -- 2: P1 Wins
    -- 3: P2 Wins

    signal game_state : integer := 0; -- Initial state is start
    -- In the start state the words PONG are displayed
    -- Whenever the player presses the start button, the game state changes to playing
    -- If the player presses the rst button, the game state changes to start
    -- If the player wins, the game state changes to the respective win state
```

Listing 2: Constants and Signals

```

␣␣clk_div: entity work.clk_div
␣␣␣␣generic map(
␣␣␣␣␣␣clk_in_freq => 50e6,
␣␣␣␣␣␣clk_out_freq => 50
␣␣␣␣)
␣␣␣␣port map(
␣␣␣␣␣␣clk_in => clk,
␣␣␣␣␣␣rst => rst,
␣␣␣␣␣␣clk_out => slow_clk
␣␣␣␣);

```

```
␣␣-- VGA SYNC_GEN BEGINS
␣␣sync: entity work.vga_sync_gen
␣␣␣␣port map (clk => clk,
␣␣␣␣␣␣␣␣    rst => rst,
␣␣␣␣␣␣␣␣    h_count => h_count,
␣␣␣␣␣␣␣␣    v_count => v_count,
␣␣␣␣␣␣␣␣    h_sync => temp_h_sync,
␣␣␣␣␣␣␣␣␣␣v_sync => temp_v_sync,
␣␣␣␣␣␣␣␣␣␣video_on => temp_video_on);
   -- VGA_SYNC_GEN ENDS␣

    -- State Machine for the game
    game_state_machine: process(slow_clk, rst)
        variable temp_game_state: integer;
    begin
        temp_game_state := game_state;
        if rising_edge(slow_clk) then
            if rst = '1' then
                game_state <= 0;
                -- Reset the scores
                -- P1_score <= 0;
                -- P2_score <= 0;
                -- Reset the ball position
            else
                case game_state is
                    when 0 =>
                        if en = '1' then
                            temp_game_state := 1;
                        end if;
                    when 1 =>
                        if P1_score = 10 then
                            temp_game_state := 2;
                        elsif P2_score = 10 then
                            temp_game_state := 3;
                        end if;
                    when 2 =>
                        if en = '1' then
                            temp_game_state := 0;
                        end if;
                    when 3 =>
                        if en = '1' then
                            temp_game_state := 0;
                        end if;
                    when others =>
                        temp_game_state := 0;
                end case;
                game_state <= temp_game_state;
            end if;
        end if;
    end process game_state_machine;
```

Listing 3: State Machine

```

    ball_move: process(slow_clk, rst)
```

```
        variable temp_mov_x: integer;
        variable temp_mov_y: integer;
        variable temp_x_pos: integer;
        variable temp_y_pos: integer;
    begin
        temp_mov_x := mov_x;
        temp_mov_y := mov_y;
        temp_x_pos := x_pos;
        temp_y_pos := y_pos;
        if rising_edge(slow_clk) then

            if rst = '1' then
                x_pos <= 200;
                y_pos <= 200;
                temp_mov_x := 0;
                temp_mov_y := 0;
                mov_x <= 0; -- Stopping the balls movement
                mov_y <= 0;

            else
                -- If the ball hits the left or right wall, reset the ball to the center
                if x_pos + size >= X_MAX or x_pos <= 0 then
                    -- Reset the ball to the center
                    temp_x_pos := 200;
                    temp_y_pos := 200;
                    mov_x <= 1;
                    mov_y <= 1;
                    -- Increment the score of the player who scored
                    if x_pos + size >= X_MAX then
                        P1_score <= P1_score + 1;
                    else
                        P2_score <= P2_score + 1;
                    end if;
                -- If the ball hits the top or bottom wall, reverse the direction of the ball
                elsif y_pos + size >= Y_MAX or y_pos <= 0 then
                    mov_y <= -1 * mov_y;
                    temp_mov_y := -1 * temp_mov_y;
                -- If the ball hits the paddle 1, reverse the direction of the ball
                elsif
                    x_pos <= x_pos_p1 + PADDLE_WIDTH and
                    y_pos + size >= y_pos_p1 and
                    y_pos <= y_pos_p1 + PADDLE_HEIGHT then
                    mov_x <= -1 * mov_x;
                    temp_mov_x := -1 * temp_mov_x;
                -- If the ball hits the paddle 2, reverse the direction of the ball
                elsif
                    x_pos + size >= x_pos_p2 and
                    y_pos + size >= y_pos_p2 and
                    y_pos <= y_pos_p2 + PADDLE_HEIGHT then
                    mov_x <= -1 * mov_x;
                    temp_mov_x := -1 * temp_mov_x;
                end if;
                x_pos <= temp_x_pos + (temp_mov_x * speed);
                y_pos <= temp_y_pos + (temp_mov_y * speed);
            end if;
        end if;
    end process ball_move;

    -- Paddle 1 movement
    -- Paddle 1 is dependent on the switches to move up and down
    -- Switches 0 and 1 are used to move the paddle up and down
    paddle1_move: process(slow_clk, rst)
        variable temp_y_pos_p1: integer;
        variable temp_x_pos_p1: integer;
    begin
        temp_y_pos_p1 := y_pos_p1;
        temp_x_pos_p1 := x_pos_p1;
        if rising_edge(slow_clk) then
            if rst = '1' then
                y_pos_p1 <= 200;
                x_pos_p1 <= 0;
            else
                if switch(0) = '1' and y_pos_p1 - PADDLE_SPEED >=
                   PADDLE_MIN then
                    y_pos_p1 <= y_pos_p1 - PADDLE_SPEED;
                    temp_y_pos_p1 := y_pos_p1 - PADDLE_SPEED;
                elsif switch(1) = '1' and y_pos_p1 + PADDLE_SPEED <=
                   PADDLE_MAX then
                    y_pos_p1 <= y_pos_p1 + PADDLE_SPEED;
                    temp_y_pos_p1 := y_pos_p1 + PADDLE_SPEED;
                end if;
                y_pos_p1 <= temp_y_pos_p1;
                x_pos_p1 <= temp_x_pos_p1;
            end if;
        end if;
    end process paddle1_move;

    -- Paddle 2 movement
    paddle2_move: process(slow_clk, rst)
        variable temp_y_pos_p2: integer;
        variable temp_x_pos_p2: integer;
    begin
        temp_y_pos_p2 := y_pos_p2;
        temp_x_pos_p2 := x_pos_p2;
        if rising_edge(slow_clk) then
            if rst = '1' or game_state = 0 or game_state = 2 or
               game_state = 3 then
                y_pos_p2 <= 200;
                x_pos_p2 <= 638 - PADDLE_WIDTH;
            else
                if switch(2) = '1' and y_pos_p2 - PADDLE_SPEED >=
                   PADDLE_MIN then
                    y_pos_p2 <= y_pos_p2 - PADDLE_SPEED;
                    temp_y_pos_p2 := y_pos_p2 - PADDLE_SPEED;
                elsif switch(3) = '1' and y_pos_p2 + PADDLE_SPEED <= PADDLE_MAX then
                    y_pos_p2 <= y_pos_p2 + PADDLE_SPEED;
                    temp_y_pos_p2 := y_pos_p2 + PADDLE_SPEED;
                end if;
                y_pos_p2 <= temp_y_pos_p2;
                x_pos_p2 <= temp_x_pos_p2;
            end if;
        end if;
    end process paddle2_move;
```

Listing 4: Movement Processes

```
␣draw: process(clk, rst)
␣begin
␣␣if rising_edge(clk) then
␣␣    -- If in the start state, display the PONG letters
␣␣    -- If in the win state, display the WIN letters with the respective player

            -- Draw the PONG letters
            if unsigned(h_count) >= to_unsigned(x_pos_p, h_count'length) and unsigned(h_count) <= to_unsigned(x_pos_p + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_p, v_count'length) and unsigned(v_count) <= to_unsigned(y_pos_p + BM_SIZE * 5, v_count'length) and
            game_state = 0 and
            temp_video_on = '1' then
                if P(BM_SIZE - 1 - to_integer(unsigned(v_count) - y_pos_p) / 5)(BM_SIZE - 1 - to_integer(unsigned(h_count) - x_pos_p) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            -- Display O
            elsif unsigned(h_count) >= to_unsigned(x_pos_o, h_count'length) and unsigned(h_count) <= to_unsigned(x_pos_o + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_o, v_count'length) and unsigned(v_count) <= to_unsigned(y_pos_o + BM_SIZE * 5, v_count'length) and
            game_state = 0 and
            temp_video_on = '1' then
                if O(BM_SIZE - 1 - to_integer(unsigned(v_count) - y_pos_o) / 5)(BM_SIZE - 1 - to_integer(unsigned(h_count) - x_pos_o) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            -- Display N
            elsif unsigned(h_count) >= to_unsigned(x_pos_n,
               h_count'length) and unsigned(h_count) <=
               to_unsigned(x_pos_n + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_n, v_count'length) and
               unsigned(v_count) <= to_unsigned(y_pos_n + BM_SIZE * 5,
               v_count'length) and
            game_state = 0 and
            temp_video_on = '1' then
                if N(BM_SIZE - 1 - to_integer(unsigned(v_count) -
                   y_pos_n) / 5)(BM_SIZE - 1 -
                   to_integer(unsigned(h_count) - x_pos_n) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            -- Display G
            elsif unsigned(h_count) >= to_unsigned(x_pos_g,
               h_count'length) and unsigned(h_count) <=
               to_unsigned(x_pos_g + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_g, v_count'length) and
               unsigned(v_count) <= to_unsigned(y_pos_g + BM_SIZE * 5,
               v_count'length) and
            game_state = 0 and
            temp_video_on = '1' then
                if G(BM_SIZE - 1 - to_integer(unsigned(v_count) -
                   y_pos_g) / 5)(BM_SIZE - 1 -
                   to_integer(unsigned(h_count) - x_pos_g) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;

        -- Drawing the ball
            elsif unsigned(h_count) >= to_unsigned(x_pos, h_count'length)
               and unsigned(h_count) <= to_unsigned(x_pos + size,
               h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos, v_count'length) and
               unsigned(v_count) <= to_unsigned(y_pos + size,
               v_count'length) and
            game_state = 1 and
            temp_video_on = '1' then
                red <= "0111";
                green <= "0011";
                blue <= "1011";
            --- Paddle 1
            elsif unsigned(h_count) >= to_unsigned(x_pos_p1,
               h_count'length) and unsigned(h_count) <=
               to_unsigned(x_pos_p1 + PADDLE_WIDTH, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_p1, v_count'length)
               and unsigned(v_count) <= to_unsigned(y_pos_p1 +
               PADDLE_HEIGHT, v_count'length) and
            game_state = 1 and
            temp_video_on = '1' then
                red <= "0000";
                green <= "0000";
                blue <= "1111";
            --- Paddle 2
            elsif unsigned(h_count) >= to_unsigned(x_pos_p2,
               h_count'length) and unsigned(h_count) <=
               to_unsigned(x_pos_p2 + PADDLE_WIDTH, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_p2, v_count'length)
               and unsigned(v_count) <= to_unsigned(y_pos_p2 +
               PADDLE_HEIGHT, v_count'length) and
            game_state = 1 and
            temp_video_on = '1' then
                red <= "1111";
                green <= "0000";
                blue <= "0000";
            elsif
            unsigned(h_count) >= to_unsigned(x_pos_p1_score,
               h_count'length)
            and unsigned(h_count) <= to_unsigned(x_pos_p1_score + BM_SIZE
               * 3, h_count'length)
            and unsigned(v_count) >= to_unsigned(y_pos_p1_score,
               v_count'length)
            and unsigned(v_count) <= to_unsigned(y_pos_p1_score + BM_SIZE
               * 5, v_count'length)
            and game_state = 1
            and temp_video_on = '1' then
                -- Convert the y_pos_p1_score and x_pos_p1_score to
                   unsigned to perform the subtraction
            --- Drawing score for P1
            -- We need to draw the score and check if first we are in the
               display area
            -- and then check what specific pixel we are in
            -- we then check the value of the bitmap for that score at
               that pixel
            -- Our bitmap is 8x4 so it has 5 rows and 5 columns
                if score_to_bitmap(P1_score)(BM_SIZE - 1 -
                   to_integer(unsigned(v_count) - y_pos_p1_score) /
                   5)(BM_SIZE - 1 - to_integer(unsigned(h_count) -
                   x_pos_p1_score) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
```

```vhdl
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;

            -- Drawing score for P2
            elsif unsigned(h_count) >= to_unsigned(x_pos_p2_score, h_count'length)
            and unsigned(h_count) <= to_unsigned(x_pos_p2_score + BM_SIZE * 3, h_count'length)
            and unsigned(v_count) >= to_unsigned(y_pos_p2_score, v_count'length)
            and unsigned(v_count) <= to_unsigned(y_pos_p2_score + BM_SIZE * 5, v_count'length)
            and game_state = 1
            and temp_video_on = '1' then
                if score_to_bitmap(P2_score)(BM_SIZE - 1 - to_integer(unsigned(v_count) - y_pos_p2_score) / 5)(BM_SIZE - 1 - to_integer(unsigned(h_count) - x_pos_p2_score) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            -- Display the WIN letters for P1
            elsif unsigned(h_count) >= to_unsigned(x_pos_w, h_count'length) and unsigned(h_count) <= to_unsigned(x_pos_w + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_w, v_count'length) and unsigned(v_count) <= to_unsigned(y_pos_w + BM_SIZE * 5, v_count'length) and
            game_state = 2 and
            temp_video_on = '1' then
                if W(BM_SIZE - 1 - to_integer(unsigned(v_count) - y_pos_w) / 5)(BM_SIZE - 1 - to_integer(unsigned(h_count) - x_pos_w) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            elsif unsigned(h_count) >= to_unsigned(x_pos_i, h_count'length) and unsigned(h_count) <= to_unsigned(x_pos_i + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_i, v_count'length) and
```

```
               unsigned(v_count) <= to_unsigned(y_pos_i + BM_SIZE * 5,
               v_count'length) and
            game_state = 2 and
            temp_video_on = '1' then
                if I(BM_SIZE - 1 - to_integer(unsigned(v_count) -
                   y_pos_i) / 5)(BM_SIZE - 1 -
                   to_integer(unsigned(h_count) - x_pos_i) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            elsif unsigned(h_count) >= to_unsigned(x_pos_n_win,
               h_count'length) and unsigned(h_count) <=
               to_unsigned(x_pos_n_win + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_n_win, v_count'length)
               and unsigned(v_count) <= to_unsigned(y_pos_n_win + BM_SIZE
               * 5, v_count'length) and
            game_state = 2 and
            temp_video_on = '1' then
                if N(BM_SIZE - 1 - to_integer(unsigned(v_count) -
                   y_pos_n_win) / 5)(BM_SIZE - 1 -
                   to_integer(unsigned(h_count) - x_pos_n_win) / 3) = '1'
                   then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            elsif unsigned(h_count) >= to_unsigned(x_pos_s,
               h_count'length) and unsigned(h_count) <=
               to_unsigned(x_pos_s + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_s, v_count'length) and
               unsigned(v_count) <= to_unsigned(y_pos_s + BM_SIZE * 5,
               v_count'length) and
            game_state = 2 and
            temp_video_on = '1' then
                if S(BM_SIZE - 1 - to_integer(unsigned(v_count) -
                   y_pos_s) / 5)(BM_SIZE - 1 -
                   to_integer(unsigned(h_count) - x_pos_s) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            -- Display 1 for P1
            elsif unsigned(h_count) >= to_unsigned(x_pos_p1_score, h_count'length) and unsigned(h_count) <= to_unsigned(x_pos_p1_score + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_p1_score, v_count'length) and unsigned(v_count) <= to_unsigned(y_pos_p1_score + BM_SIZE * 5, v_count'length) and
            game_state = 2 and
            temp_video_on = '1' then
                if score_to_bitmap(1)(BM_SIZE - 1 - to_integer(unsigned(v_count) - y_pos_p1_score) / 5)(BM_SIZE - 1 - to_integer(unsigned(h_count) - x_pos_p1_score) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            -- Display the WIN letters for P2
            elsif unsigned(h_count) >= to_unsigned(x_pos_w, h_count'length) and unsigned(h_count) <= to_unsigned(x_pos_w + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_w, v_count'length) and unsigned(v_count) <= to_unsigned(y_pos_w + BM_SIZE * 5, v_count'length) and
            game_state = 3 and
            temp_video_on = '1' then
                if W(BM_SIZE - 1 - to_integer(unsigned(v_count) - y_pos_w) / 5)(BM_SIZE - 1 - to_integer(unsigned(h_count) - x_pos_w) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            elsif unsigned(h_count) >= to_unsigned(x_pos_i, h_count'length) and unsigned(h_count) <= to_unsigned(x_pos_i + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_i, v_count'length) and unsigned(v_count) <= to_unsigned(y_pos_i + BM_SIZE * 5, v_count'length) and
            game_state = 3 and
            temp_video_on = '1' then
                if I(BM_SIZE - 1 - to_integer(unsigned(v_count) - y_pos_i) / 5)(BM_SIZE - 1 - to_integer(unsigned(h_count) - x_pos_i) / 3) = '1' then
```

```
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            elsif unsigned(h_count) >= to_unsigned(x_pos_n_win,
               h_count'length) and unsigned(h_count) <=
               to_unsigned(x_pos_n_win + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_n_win, v_count'length)
               and unsigned(v_count) <= to_unsigned(y_pos_n_win + BM_SIZE
               * 5, v_count'length) and
            game_state = 3 and
            temp_video_on = '1' then
                if N(BM_SIZE - 1 - to_integer(unsigned(v_count) -
                   y_pos_n_win) / 5)(BM_SIZE - 1 -
                   to_integer(unsigned(h_count) - x_pos_n_win) / 3) = '1'
                   then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            elsif unsigned(h_count) >= to_unsigned(x_pos_s,
               h_count'length) and unsigned(h_count) <=
               to_unsigned(x_pos_s + BM_SIZE * 3, h_count'length) and
            unsigned(v_count) >= to_unsigned(y_pos_s, v_count'length) and
               unsigned(v_count) <= to_unsigned(y_pos_s + BM_SIZE * 5,
               v_count'length) and
            game_state = 3 and
            temp_video_on = '1' then
                if S(BM_SIZE - 1 - to_integer(unsigned(v_count) -
                   y_pos_s) / 5)(BM_SIZE - 1 -
                   to_integer(unsigned(h_count) - x_pos_s) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            -- Display the 2 for P2
            elsif unsigned(h_count) >= to_unsigned(x_pos_p2_score,
               h_count'length) and unsigned(h_count) <=
               to_unsigned(x_pos_p2_score + BM_SIZE * 3, h_count'length)
               and
            unsigned(v_count) >= to_unsigned(y_pos_p2_score,
               v_count'length) and unsigned(v_count) <=
                to_unsigned(y_pos_p2_score + BM_SIZE * 5, v_count'length) and
            game_state = 3 and
            temp_video_on = '1' then
                if score_to_bitmap(2)(BM_SIZE - 1 - to_integer(unsigned(v_count) - y_pos_p2_score) / 5)(BM_SIZE - 1 - to_integer(unsigned(h_count) - x_pos_p2_score) / 3) = '1' then
                    red <= "1111";
                    green <= "1111";
                    blue <= "1111";
                else
                    red <= "0000";
                    green <= "0000";
                    blue <= "0000";
                end if;
            -- Else Conditions
            else
                red <= "0000";
                green <= "0000";
                blue <= "0000";
            end if;
        end if;

end process draw;
```

Listing 5: Draw Process