EEL 4712C - Digital Design: Lab Report 5

Cole Rottenberg
11062528

April 14th, 2024

```vhdl
-- 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.

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

```
1
2       signal v_count : std_logic_vector(COUNT_RANGE);
3       signal h_count : std_logic_vector(COUNT_RANGE);
4       -- Clocks
5       signal slow_clk : std_logic := '0'; -- Slow clock
6       -- Internal Temp signals
7       signal temp_h_sync, temp_v_sync, temp_video_on : std_logic;
8       -- x and y coordinates of ball
9       signal x_pos, y_pos : integer := 200; -- 200 is the center of the
            screen
10      signal mov_x, mov_y : integer := 1;
11      -- Constants of the ball
12      constant speed : integer := 2;
13      constant size : integer := 64;
14      constant X_MAX : integer := 638;
15      constant Y_MAX : integer := 478;
16      -- Constants for the paddles
17      constant PADDLE_WIDTH : integer := 10;
18      constant PADDLE_HEIGHT : integer := 50;
19      constant PADDLE_SPEED : integer := 2;
20      constant PADDLE_MAX : integer := 428;
21      constant PADDLE_MIN : integer := 0;
22      -- x and y coordinates for each paddle, one on the left and one on
            the right
23      -- Paddle 1 is on the left, Paddle 2 is on the right
24      signal x_pos_p1 : integer := 0;
25      signal y_pos_p1 : integer := 200;
26      signal x_pos_p2 : integer := 638 - PADDLE_WIDTH;
27      signal y_pos_p2 : integer := 200;
28
29      signal P1_score : integer := 0;
30      signal P2_score : integer := 0;
31
32  -- BITMAPS for PONG, P1 and P2 scores, 0-9 , game over and start
33  -- using the 2D array to store the bitmaps
34  -- The dimensions of the array are 8x4
35  -- PONG
36
37      constant BM_SIZE : integer := 5;
38
39      type bitmap is array(4 downto 0) of std_logic_vector(4 downto 0);
40
41      constant ZERO : bitmap := (
42          "11111",
43          "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"
    );

    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"
```

```
206     );
207     -- Position of the bitmap
208
209     -- Position of the P1 score
210     signal x_pos_p1_score : integer := 100;
211     signal y_pos_p1_score : integer := 100;
212
213     -- Position of the P2 score
214     signal x_pos_p2_score : integer := 500;
215     signal y_pos_p2_score : integer := 100;
216
217     -- Position of the PONG letters
218     -- Letter P
219     signal x_pos_p : integer := 250;
220     signal y_pos_p : integer := 100;
221
222     -- Letter O
223     signal x_pos_o : integer := 275;
224     signal y_pos_o : integer := 100;
225
226     -- Letter N
227     signal x_pos_n : integer := 300;
228     signal y_pos_n : integer := 100;
229
230     -- Letter G
231     signal x_pos_g : integer := 325;
232     signal y_pos_g : integer := 100;
233
234     -- Adding Letters for Wins
235     -- Position of the P1 win
236     signal x_pos_p1_win : integer := 250;
237     signal y_pos_p1_win : integer := 200;
238
239     -- Position of the P2 win
240     signal x_pos_p2_win : integer := 500;
241     signal y_pos_p2_win : integer := 200;
242
243     -- Position of the WIN letters
244     -- Letter W
245     signal x_pos_w : integer := 300;
246     signal y_pos_w : integer := 200;
247
248     -- Letter I
249     signal x_pos_i : integer := 350;
250     signal y_pos_i : integer := 200;
251
252     -- Letter N
253     signal x_pos_n_win : integer := 400;
254     signal y_pos_n_win : integer := 200;
255
256     -- Letter S
257     signal x_pos_s : integer := 450;
258     signal y_pos_s : integer := 200;
259
```

```
260     -- Position of the START letters
261     -- Letter S
262     signal x_pos_s_start : integer := 300;
263     signal y_pos_s_start : integer := 300;
264
265     -- Letter T
266     signal x_pos_t : integer := 350;
267     signal y_pos_t : integer := 300;
268
269     -- Letter A
270     signal x_pos_a : integer := 400;
271     signal y_pos_a : integer := 300;
272
273     -- Letter R
274     signal x_pos_r : integer := 450;
275     signal y_pos_r : integer := 300;
276
277     -- Letter T
278     signal x_pos_t_start : integer := 500;
279     signal y_pos_t_start : integer := 300;
280
281     -- Letter S
282     signal x_pos_s_start2 : integer := 550;
283     signal y_pos_s_start2 : integer := 300;
284
285     -- Defining Game States
286     -- 0: Start
287     -- 1: Playing
288     -- 2: P1 Wins
289     -- 3: P2 Wins
290
291     signal game_state : integer := 0; -- Initial state is start
292     -- In the start state the words PONG are displayed
293     -- Whenever the player presses the start button, the game state
        changes to playing
294     -- If the player presses the rst button, the game state changes to
        start
295     -- If the player wins, the game state changes to the respective win
        state
```

Listing 2: Constants and Signals

```
1
2   clk_div: entity work.clk_div
3       generic map(
4           clk_in_freq => 50e6,
5           clk_out_freq => 50
6       )
7       port map(
8           clk_in => clk,
9           rst => rst,
10          clk_out => slow_clk
11      );
12
13  -- 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;
```

```
4           variable temp_mov_y: integer;
5           variable temp_x_pos: integer;
6           variable temp_y_pos: integer;
7       begin
8           temp_mov_x := mov_x;
9           temp_mov_y := mov_y;
10          temp_x_pos := x_pos;
11          temp_y_pos := y_pos;
12          if rising_edge(slow_clk) then
13
14              if rst = '1' then
15                  x_pos <= 200;
16                  y_pos <= 200;
17                  temp_mov_x := 0;
18                  temp_mov_y := 0;
19                  mov_x <= 0; -- Stopping the balls movement
20                  mov_y <= 0;
21
22              else
23                  -- If the ball hits the left or right wall, reset the
                       ball to the center
24                  if x_pos + size >= X_MAX or x_pos <= 0 then
25                      -- Reset the ball to the center
26                      temp_x_pos := 200;
27                      temp_y_pos := 200;
28                      mov_x <= 1;
29                      mov_y <= 1;
30                      -- Increment the score of the player who scored
31                      if x_pos + size >= X_MAX then
32                          P1_score <= P1_score + 1;
33                      else
34                          P2_score <= P2_score + 1;
35                      end if;
36                  -- If the ball hits the top or bottom wall, reverse the
                       direction of the ball
37                  elsif y_pos + size >= Y_MAX or y_pos <= 0 then
38                      mov_y <= -1 * mov_y;
39                      temp_mov_y := -1 * temp_mov_y;
40                  -- If the ball hits the paddle 1, reverse the direction
                       of the ball
41                  elsif
42                      x_pos <= x_pos_p1 + PADDLE_WIDTH and
43                      y_pos + size >= y_pos_p1 and
44                      y_pos <= y_pos_p1 + PADDLE_HEIGHT then
45                      mov_x <= -1 * mov_x;
46                      temp_mov_x := -1 * temp_mov_x;
47                  -- If the ball hits the paddle 2, reverse the direction
                       of the ball
48                  elsif
49                      x_pos + size >= x_pos_p2 and
50                      y_pos + size >= y_pos_p2 and
51                      y_pos <= y_pos_p2 + PADDLE_HEIGHT then
52                      mov_x <= -1 * mov_x;
53                      temp_mov_x := -1 * temp_mov_x;
```

```
54                  end if;
55                  x_pos <= temp_x_pos + (temp_mov_x * speed);
56                  y_pos <= temp_y_pos + (temp_mov_y * speed);
57              end if;
58          end if;
59      end process ball_move;
60
61      -- Paddle 1 movement
62      -- Paddle 1 is dependent on the switches to move up and down
63      -- Switches 0 and 1 are used to move the paddle up and down
64      paddle1_move: process(slow_clk, rst)
65          variable temp_y_pos_p1: integer;
66          variable temp_x_pos_p1: integer;
67      begin
68          temp_y_pos_p1 := y_pos_p1;
69          temp_x_pos_p1 := x_pos_p1;
70          if rising_edge(slow_clk) then
71              if rst = '1' then
72                  y_pos_p1 <= 200;
73                  x_pos_p1 <= 0;
74              else
75                  if switch(0) = '1' and y_pos_p1 - PADDLE_SPEED >=
                        PADDLE_MIN then
76                      y_pos_p1 <= y_pos_p1 - PADDLE_SPEED;
77                      temp_y_pos_p1 := y_pos_p1 - PADDLE_SPEED;
78                  elsif switch(1) = '1' and y_pos_p1 + PADDLE_SPEED <=
                        PADDLE_MAX then
79                      y_pos_p1 <= y_pos_p1 + PADDLE_SPEED;
80                      temp_y_pos_p1 := y_pos_p1 + PADDLE_SPEED;
81                  end if;
82                  y_pos_p1 <= temp_y_pos_p1;
83                  x_pos_p1 <= temp_x_pos_p1;
84              end if;
85          end if;
86      end process paddle1_move;
87
88      -- Paddle 2 movement
89      paddle2_move: process(slow_clk, rst)
90          variable temp_y_pos_p2: integer;
91          variable temp_x_pos_p2: integer;
92      begin
93          temp_y_pos_p2 := y_pos_p2;
94          temp_x_pos_p2 := x_pos_p2;
95          if rising_edge(slow_clk) then
96              if rst = '1' or game_state = 0 or game_state = 2 or
                    game_state = 3 then
97                  y_pos_p2 <= 200;
98                  x_pos_p2 <= 638 - PADDLE_WIDTH;
99              else
100                 if switch(2) = '1' and y_pos_p2 - PADDLE_SPEED >=
                        PADDLE_MIN then
101                     y_pos_p2 <= y_pos_p2 - PADDLE_SPEED;
102                     temp_y_pos_p2 := y_pos_p2 - PADDLE_SPEED;
103                 elsif switch(3) = '1' and y_pos_p2 + PADDLE_SPEED <=
```

```
                    PADDLE_MAX then
104                     y_pos_p2 <= y_pos_p2 + PADDLE_SPEED;
105                     temp_y_pos_p2 := y_pos_p2 + PADDLE_SPEED;
106                 end if;
107                 y_pos_p2 <= temp_y_pos_p2;
108                 x_pos_p2 <= temp_x_pos_p2;
109             end if;
110         end if;
111     end process paddle2_move;
```

Listing 4: Movement Processes

```
1  draw: process(clk, rst)
2  begin
3      if rising_edge(clk) then
4          -- If in the start state, display the PONG letters
5          -- If in the win state, display the WIN letters with the
       respective player
6
7              -- Draw the PONG letters
8              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
9              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
10             game_state = 0 and
11             temp_video_on = '1' then
12                 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
13                     red <= "1111";
14                     green <= "1111";
15                     blue <= "1111";
16                 else
17                     red <= "0000";
18                     green <= "0000";
19                     blue <= "0000";
20                 end if;
21             -- Display O
22             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
23             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
24             game_state = 0 and
25             temp_video_on = '1' then
26                 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
27                     red <= "1111";
28                     green <= "1111";
29                     blue <= "1111";
30                 else
```

```
31                      red <= "0000";
32                      green <= "0000";
33                      blue <= "0000";
34                  end if;
35              -- Display N
36              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
37              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
38              game_state = 0 and
39              temp_video_on = '1' then
40                  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
41                      red <= "1111";
42                      green <= "1111";
43                      blue <= "1111";
44                  else
45                      red <= "0000";
46                      green <= "0000";
47                      blue <= "0000";
48                  end if;
49              -- Display G
50              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
51              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
52              game_state = 0 and
53              temp_video_on = '1' then
54                  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
55                      red <= "1111";
56                      green <= "1111";
57                      blue <= "1111";
58                  else
59                      red <= "0000";
60                      green <= "0000";
61                      blue <= "0000";
62                  end if;
63
64          -- Drawing the ball
65              elsif unsigned(h_count) >= to_unsigned(x_pos, h_count'length)
                    and unsigned(h_count) <= to_unsigned(x_pos + size,
                    h_count'length) and
66              unsigned(v_count) >= to_unsigned(y_pos, v_count'length) and
                    unsigned(v_count) <= to_unsigned(y_pos + size,
                    v_count'length) and
67              game_state = 1 and
68              temp_video_on = '1' then
```

```
69                  red <= "0111";
70                  green <= "0011";
71                  blue <= "1011";
72              --- Paddle 1
73              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
74              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
75              game_state = 1 and
76              temp_video_on = '1' then
77                  red <= "0000";
78                  green <= "0000";
79                  blue <= "1111";
80              --- Paddle 2
81              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
82              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
83              game_state = 1 and
84              temp_video_on = '1' then
85                  red <= "1111";
86                  green <= "0000";
87                  blue <= "0000";
88              elsif
89              unsigned(h_count) >= to_unsigned(x_pos_p1_score,
                    h_count'length)
90              and unsigned(h_count) <= to_unsigned(x_pos_p1_score + BM_SIZE
                    * 3, h_count'length)
91              and unsigned(v_count) >= to_unsigned(y_pos_p1_score,
                    v_count'length)
92              and unsigned(v_count) <= to_unsigned(y_pos_p1_score + BM_SIZE
                    * 5, v_count'length)
93              and game_state = 1
94              and temp_video_on = '1' then
95                  -- Convert the y_pos_p1_score and x_pos_p1_score to
                        unsigned to perform the subtraction
96              --- Drawing score for P1
97              -- We need to draw the score and check if first we are in the
                    display area
98              -- and then check what specific pixel we are in
99              -- we then check the value of the bitmap for that score at
                    that pixel
100             -- Our bitmap is 8x4 so it has 5 rows and 5 columns
101                 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
102                     red <= "1111";
103                     green <= "1111";
104                     blue <= "1111";
```

```
105                 else
106                     red <= "0000";
107                     green <= "0000";
108                     blue <= "0000";
109                 end if;
110 
111             -- Drawing score for P2
112             elsif unsigned(h_count) >= to_unsigned(x_pos_p2_score,
                    h_count'length)
113             and unsigned(h_count) <= to_unsigned(x_pos_p2_score + BM_SIZE
                    * 3, h_count'length)
114             and unsigned(v_count) >= to_unsigned(y_pos_p2_score,
                    v_count'length)
115             and unsigned(v_count) <= to_unsigned(y_pos_p2_score + BM_SIZE
                    * 5, v_count'length)
116             and game_state = 1
117             and temp_video_on = '1' then
118                 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
119                     red <= "1111";
120                     green <= "1111";
121                     blue <= "1111";
122                 else
123                     red <= "0000";
124                     green <= "0000";
125                     blue <= "0000";
126                 end if;
127             -- Display the WIN letters for P1
128             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
129             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
130             game_state = 2 and
131             temp_video_on = '1' then
132                 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
133                     red <= "1111";
134                     green <= "1111";
135                     blue <= "1111";
136                 else
137                     red <= "0000";
138                     green <= "0000";
139                     blue <= "0000";
140                 end if;
141             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
142             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
143         game_state = 2 and
144         temp_video_on = '1' then
145             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
146                 red <= "1111";
147                 green <= "1111";
148                 blue <= "1111";
149             else
150                 red <= "0000";
151                 green <= "0000";
152                 blue <= "0000";
153             end if;
154         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
155         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
156         game_state = 2 and
157         temp_video_on = '1' then
158             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
159                 red <= "1111";
160                 green <= "1111";
161                 blue <= "1111";
162             else
163                 red <= "0000";
164                 green <= "0000";
165                 blue <= "0000";
166             end if;
167         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
168         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
169         game_state = 2 and
170         temp_video_on = '1' then
171             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
172                 red <= "1111";
173                 green <= "1111";
174                 blue <= "1111";
175             else
176                 red <= "0000";
177                 green <= "0000";
178                 blue <= "0000";
179             end if;
180         -- Display 1 for P1
```

```
181             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
182             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
183             game_state = 2 and
184             temp_video_on = '1' then
185                 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
186                     red <= "1111";
187                     green <= "1111";
188                     blue <= "1111";
189                 else
190                     red <= "0000";
191                     green <= "0000";
192                     blue <= "0000";
193                 end if;
194             -- Display the WIN letters for P2
195             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
196             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
197             game_state = 3 and
198             temp_video_on = '1' then
199                 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
200                     red <= "1111";
201                     green <= "1111";
202                     blue <= "1111";
203                 else
204                     red <= "0000";
205                     green <= "0000";
206                     blue <= "0000";
207                 end if;
208             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
209             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
210             game_state = 3 and
211             temp_video_on = '1' then
212                 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
213                     red <= "1111";
```

```
214                     green <= "1111";
215                     blue <= "1111";
216                 else
217                     red <= "0000";
218                     green <= "0000";
219                     blue <= "0000";
220                 end if;
221             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
222             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
223             game_state = 3 and
224             temp_video_on = '1' then
225                 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
226                     red <= "1111";
227                     green <= "1111";
228                     blue <= "1111";
229                 else
230                     red <= "0000";
231                     green <= "0000";
232                     blue <= "0000";
233                 end if;
234             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
235             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
236             game_state = 3 and
237             temp_video_on = '1' then
238                 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
239                     red <= "1111";
240                     green <= "1111";
241                     blue <= "1111";
242                 else
243                     red <= "0000";
244                     green <= "0000";
245                     blue <= "0000";
246                 end if;
247             -- Display the 2 for P2
248             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
249             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
250             game_state = 3 and
251             temp_video_on = '1' then
252                 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
253                     red <= "1111";
254                     green <= "1111";
255                     blue <= "1111";
256                 else
257                     red <= "0000";
258                     green <= "0000";
259                     blue <= "0000";
260                 end if;
261             -- Else Conditions
262             else
263                 red <= "0000";
264                 green <= "0000";
265                 blue <= "0000";
266             end if;
267         end if;
268
269 end process draw;
```

Listing 5: Draw Process