Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Digital Systems Engineering
Course Number:	COE758
Semester/Year (e.g.F2016)	F2023

Instructor:	Dr. Geurkov

Assignment/Lab Number:	Design Project 2
Assignment/Lab Title:	Simple Video Game Processor for VGA

Submission Date:	2023-11-28
Due Date:	2023-11-29

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Zelenovic	Danilo	501032542	11	DZ

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: http://www.ryerson.ca/senate/current/pol60.pdf

Abstract

For design project 2, we will create a video game processor using a Video Graphic Adaptor (VGA) on the Xilinx Spartan-3E FPGA. The game that we are tasked with creating is the classic pong game where 2 players control a paddle and play a virtual game of table tennis. Each player will be able to control a paddle with the switches on the board, and the goal is to block the ball from entering your goal, while scoring by putting the ball past your opponent's goal line. Each time the ball is hit by a paddle, the ball's speed is flipped, and the result causes the ball to move in the opposing direction, towards the goal of the other player. If a wall is hit, the y-direction is flipped, acting as a ball would it hit a real wall. Each time the ball passes a goal line, it will change colors before returning to its original color, and reappearing in the center of the screen. The program must send RGB, HSYNC and VSYNC values to the display using the VGA.

Introduction

Games are a great way to relax and destress. The entire point is to stop thinking about boring tasks in mundane everyday life and engrossing yourself in a virtual world. One of the earliest and most important video games is Pong. In this project, we investigate how to implement this simple video game processor using a VGA and a spartan-3e board. The VGA will control our display, which uses CRT technology to scan across the screen to display pixels. Students will gain practical insight in designing and implementing real-time signal generators with custom logic programming. As students are creating pong using a VGA, the display will need to be updated constantly in real-time. Because of this fact, we will define a few specifications like the vertical and horizontal parameters of the game, clock cycles for ach parameter and more. Both players will be allowed to control 1 paddle, their own, and move it upwards or downwards to prevent the ball from crossing their goal line. The display must be updated with each input. Basic pong rules apply for this game.

System Specifications

3 main system specifications for the video game processor must be noted.

1. Static video frame: We must see all the elements such as the game borders/walls and the static background image/color of the game, and dynamic components that impact gameplay.

2. Dynamic elements: There are 3 dynamic elements that must be accounted for in the pong game; the ball and 2 paddles. Each element is moving and are required to change locations as the game continues in order to play the game. The paddles can hit the ball which will cause it to move opposite in the horizontal direction, towards the opponent. The paddles themselves can only move vertically to defend their goals. Players will be able to input directions while the game is playing. The program will interpret the input and subsequently move the corresponding player's paddle up or down.
3. The game's behaviour: What happens if the ball is either missed or hit by a paddle? When a ball is hit by a paddle, the ball's trajectory must be changed by 90°, from the ball's initial trajectory before it was hit. The next scenario occurs when the ball is missed by a paddle and ends up behind it. Once this occurs, the "goal" is entered, causing the ball's color to change, indicating that a point was scored. Then the ball will reappear in its original color in the middle of the screen.

Device Description/Design

VGA specification

The VGA specifications are as follows: The Horizontal and Vertical parameters. The horizontal time periods are specified in multiples of the given VGA pixel clock, noted as 25MHz for a 60Hz refresh rate. The vertical lines are noted in multiples of the VGA lines. The following tables note the respective parameters.

Parameter	Clock Cycles
Complete Line	800
Front Porch	16
Sync Pulse	96
Back Porch	48
Active Area	640

Table 1. VGA Horizontal Parameters

Parameter	Clock Cycles
Complete Frame	525
Front Porch	10
Sync Pulse	2
Back Porch	33
Active Area	480

Table 2. VGA Vertical Parameters

Symbols and Block Diagrams

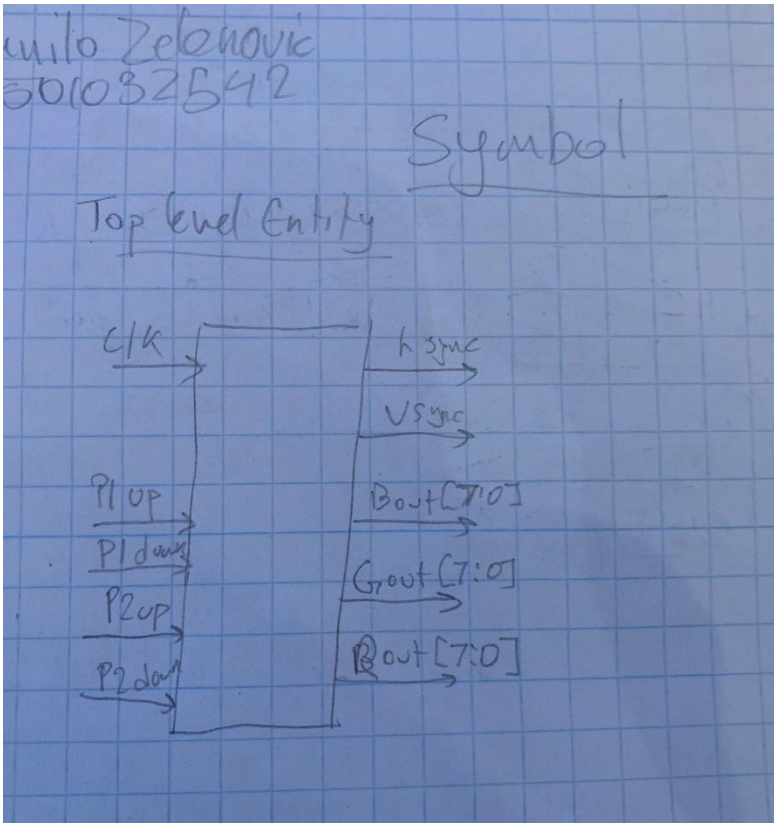

Figure 1. Symbol of top level entity.

In Figure 1, the components of the pong game are shown. The left side has the inputs, while the right side contains the output signals. Player up and down pins are on the left side, as the game requires these signals, processes them, and changes the display accordingly. They are used to change the paddle locations up or down, depending on the signal. When one direction is set to high, the other direction is set to low, ensuring they do not get locked. When both directions are set low, this means that the paddle is not moving. The CLK pin is set to 25MHz this way the image of the board is displayed correctly without issues as the board cuts off at greater clock speeds.

The output pins HSYNC and VSYNC notify the VGA when new lines or frames must be drawn. The R, G, B output pins are used to select different colors for the board with mixtures of red, green, and blue color combinations. These colors are used to indicate the border walls, the paddles, the ball, as well as the background. The VGA will receive all these output signals every 20ns to draw the game's interface and to be able to update it to make it playable.

State/Process Diagrams

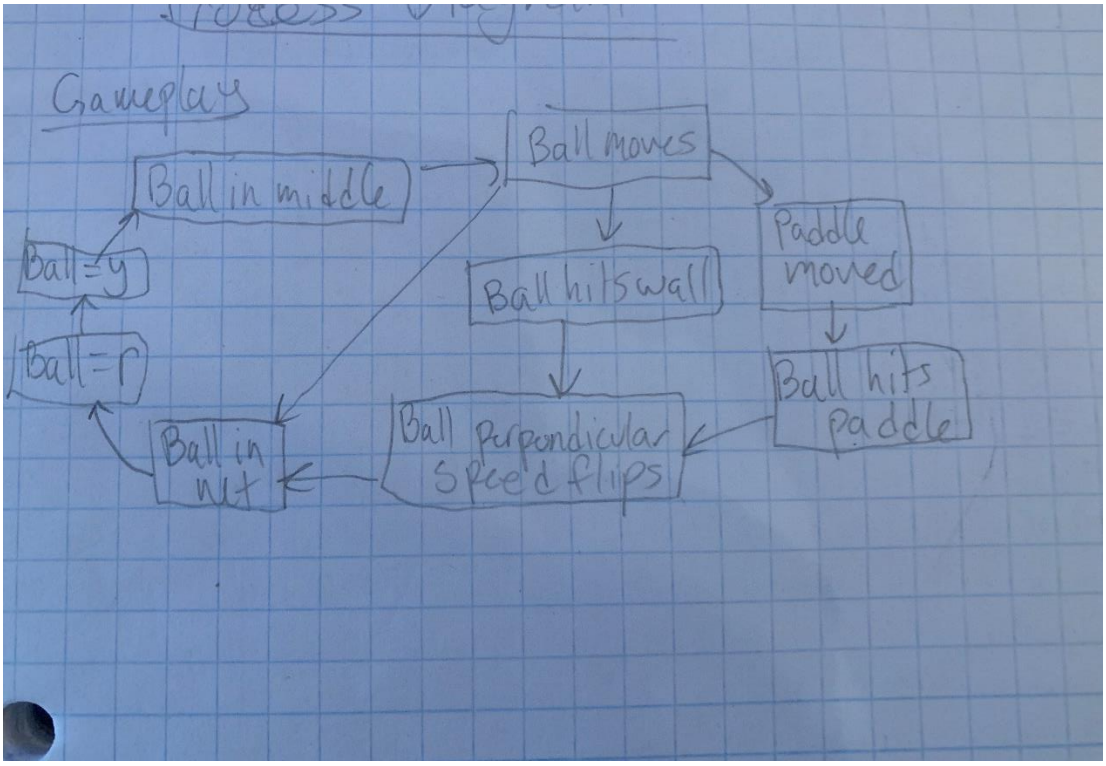

Figure 2. Process diagram of gameplay

Figure 2 displays the process diagram of the game. The game begins with a yellow ball in the center of the screen. The ball moves towards one of the paddles once the game begins. The ball's speed is constant, however, once it hits a wall or a player paddle, the direction will be inverted. If the ball hits a wall, the y-direction is flipped, while if it hits a paddle, the x-direction is flipped. If the ball enters a goal, located behind a player paddle, the ball will turn red for a moment, before it is repositioned back in the center and turns the regular yellow color again, starting the game over again.

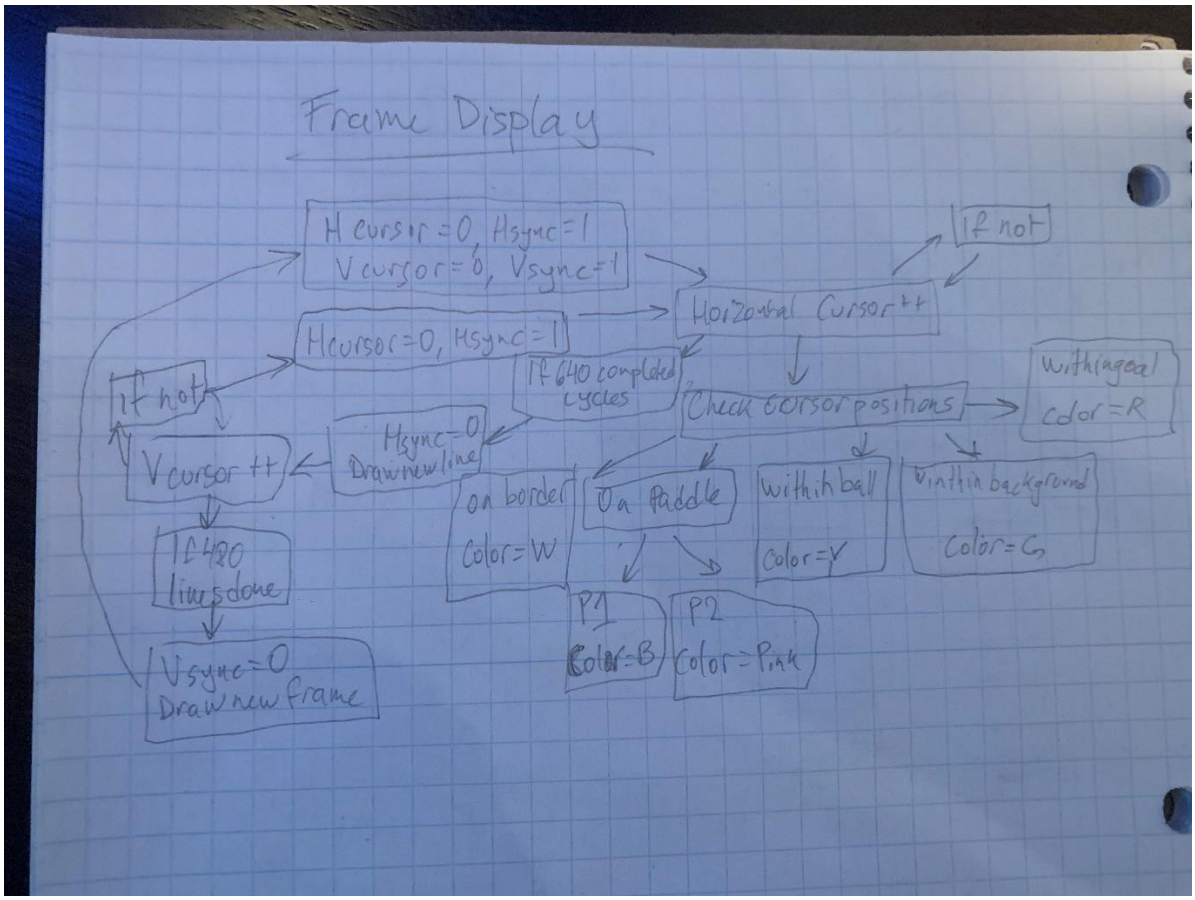

Figure 3. Process diagram for updating frame display.

Figure 3 shows how frames are displayed and updated. Horizontal and vertical cursors are used in the VGA to determine the position of the pixel that must be drawn. These will be controlled by the position counter processes, both initially set to 0. HSYNC and VSYNC are both set to high once the frame-drawing process starts. Within each cycle, which lasts 20ns, the horizontal position counter is increased by 1 so that the next pixel is drawn. When the horizontal counter reaches the maximum value (640 pixels, as per the lab manual specifications), the vertical counter will be incremented, moving the line to be updated down by 1 pixel. The HSYNC will

also be returned to low to note that the line has been completed, and the horizontal counter is set back to 0. This process continues until the vertical counter reaches the maximum value (480 pixels, as per lab manual specifications). This means that it will require 307 200 cycles in total to complete. Once both counters are completed, HSYNC and VSYNC are set to 0, as an indicator that the frame is drawn. The counters will be reset to 0, which the sync signals are set to 1, in order to begin drawing the next frame. This process runs the entire time the program is running. And ball or paddle movement will happen once every frame in order to prevent screen tear.

Results

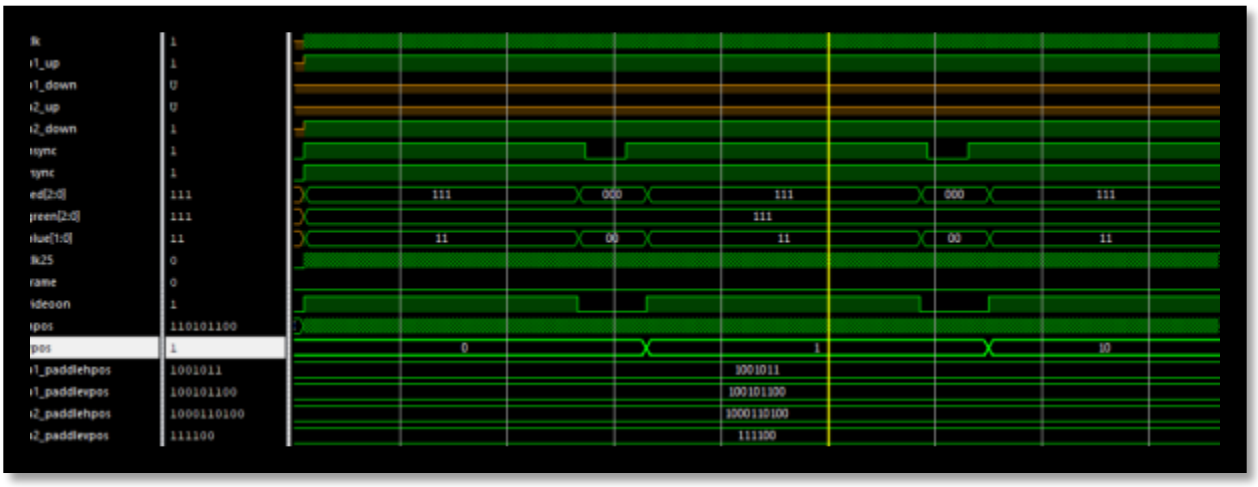

Figure 4. Timing diagram when sync signals are 1.

Figure 4 is a diagram of the timings when both sync signals are set high, meaning that the frame is being drawn. The RGB signals indicate that the color is white. This means the border is being drawn.

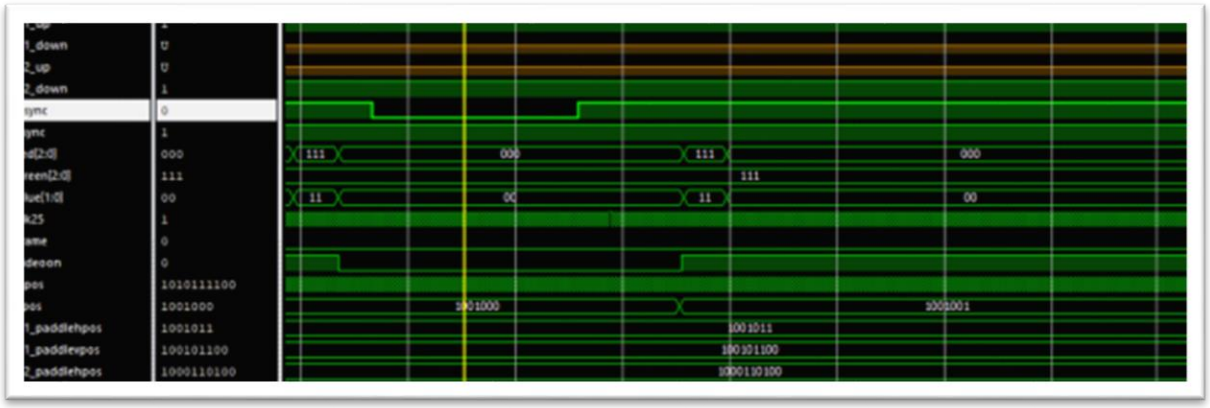

Figure 5. Timing diagram when HSYNC is low and VSYNC is high.

Figure 5 indicates that a horizontal line has finished drawing and is incrementing to a new line. The colors indicate that a green pixel is being drawn, indicating the playing area. The vertical position is still within the drawing region.

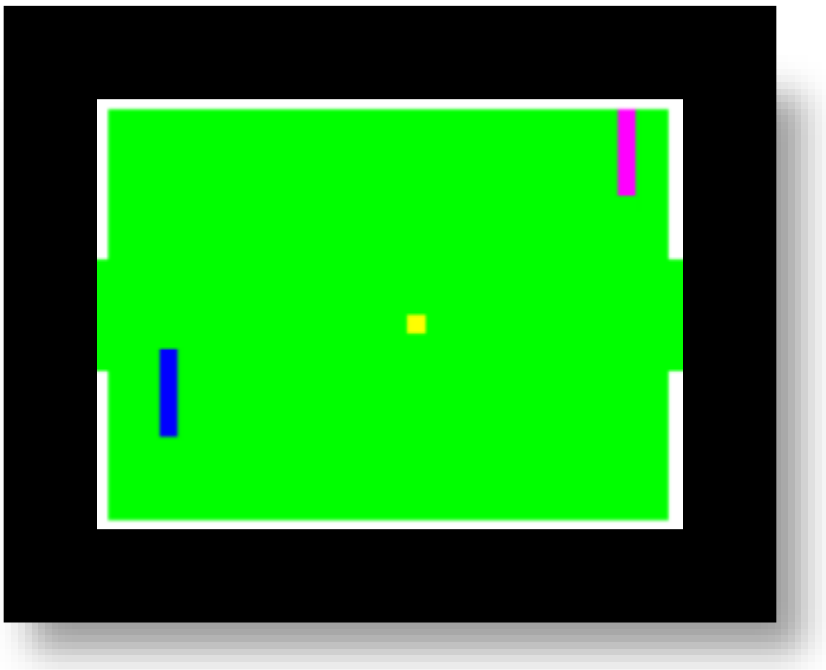

Figure 6. Simulated results of pong game

Conclusions

This project taught the importance of digital systems and their many uses and just how precise they can be. The design choices taken contribute to flow and gameplay. While a pong game may seem simple compared to most modern games, the background information required for even such a simple game is very complicated. The project was a great experience creating a pong game.

References

Eastwood, Eric. “VGA Simulator: Getting Started.” Eric Eastwood Web Developer, 14 Jan. 2018, ericeastwood.com/blog/8/vga-simulator-getting-started.

Kirischian, Lev. “Project #2 – Simply Video Game Processor for VGA” COE758, https://www.ee.ryerson.ca/~lkirisch/ele758/labs/SimpleVideoGame[11-11-11].pdf

FPGAKey. “A VGA Interface in Verilog”, https://www.fpgakey.com/tutorial/section892

https://m.youtube.com/watch?v=eJMYVLPX0no

Appendix

```
# 50 MHz input clock.
NET "clk" LOC = "c9";
# Synchronization signals.
NET "H" LOC = "c5";
NET "V" LOC = "d5";
# Pixel clock for the video DAC.
NET "DAC_CLK" LOC = "a4";
# Blue channel pins.
NET "Bout<7>" LOC = "b16";
NET "Bout<6>" LOC = "a16";
NET "Bout<5>" LOC = "d14";
NET "Bout<4>" LOC = "c14";
NET "Bout<3>" LOC = "b14";
NET "Bout<2>" LOC = "a14";
NET "Bout<1>" LOC = "b13";
NET "Bout<0>" LOC = "a13";
# Green channel pins.
NET "Gout<0>" LOC = "f9";
NET "Gout<1>" LOC = "e9";
NET "Gout<2>" LOC = "d11";
NET "Gout<3>" LOC = "c11";
NET "Gout<4>" LOC = "f11";
NET "Gout<5>" LOC = "e11";
NET "Gout<6>" LOC = "e12";
NET "Gout<7>" LOC = "f12";
# Red channel pins.
NET "Rout<0>" LOC = "a6";
NET "Rout<1>" LOC = "b6";
NET "Rout<2>" LOC = "e7";
NET "Rout<3>" LOC = "f7";
NET "Rout<4>" LOC = "d7";
NET "Rout<5>" LOC = "c7";
NET "Rout<6>" LOC = "f8";
NET "Rout<7>" LOC = "e8";
# On-board switches.
NET "SW0" LOC = "N17";
NET "SW1" LOC = "H18";
NET "SW2" LOC = "L14";
NET "SW3" LOC = "L13";
```

Game pins ucf file

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
-- use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity game_behavioral is
    Port ( clk : in  STD_LOGIC;
           SW0 : in  STD_LOGIC;
           SW1 : in  STD_LOGIC;
           SW2 : in  STD_LOGIC;
           SW3 : in  STD_LOGIC;
           H : out  STD_LOGIC;
           V : out  STD_LOGIC;
           DAC_CLK : out  STD_LOGIC;
           Rout : out  STD_LOGIC_VECTOR (7 downto 0);
           Gout : out  STD_LOGIC_VECTOR (7 downto 0);
           Bout : out  STD_LOGIC_VECTOR (7 downto 0));
end game_behavioral;

architecture Behavioral of game_behavioral is
```

```
constant H_FRONT_PORCH : integer := 16;
constant H_COMPLETE_LINE : integer := 800;
constant H_SYNC_PULSE : integer := 96;
constant V_FRONT_PORCH : integer := 10;
constant V_COMPLETE_FRAME : integer := 525;
constant V_SYNC_PULSE : integer := 2;
----------------------------
-- Clock Generation Signals
----------------------------
signal clk_divider : STD_LOGIC := '0';
signal clk_count : integer := 0;
signal clk_motion_enable : STD_LOGIC := '0';
------------------------
-- Pixel display signals
------------------------
signal video_active : STD_LOGIC;
signal horizontal_counter : integer := 0;
signal vertical_counter : integer := 0;
signal player1_x : integer := 50;
signal player1_y : integer := 360;
signal player2_x : integer := 580;
signal player2_y : integer := 60;
signal ball_x_pos : integer := 310;
signal ball_y_pos : integer := 230; -- center = (360, 240)
signal horizontal_direction : STD_LOGIC; -- 0 = right, 1 = left
signal vertical_direction : STD_LOGIC; -- 0 = down,  1  = up

begin

----------------------------
-- Clock Divider
----------------------------
process (clk)
begin
    if (rising_edge(clk)) then
        clk_divider <= NOT clk_divider;
        if (clk_count  = 100000) then
            clk_motion_enable <= NOT clk_motion_enable;
            clk_count <= 0;
        else
            clk_count <= clk_count + 1;
        end if;
    end if;
end process;

----------------------
-- Pixel Configuration
----------------------
process (clk_divider, SW3)
begin
    -- Reset Switch
    if (SW3 = '1') then
        horizontal_counter <= 0;
        vertical_counter <= 0;
        H <= '1';
        V <= '0';
        video_active <= '0';
    else
        if (rising_edge(clk_divider)) then
```

```
            if (horizontal_counter < H_COMPLETE_LINE - 1) then
                horizontal_counter <= horizontal_counter + 1;
            else
                horizontal_counter <= 0;
            if (vertical_counter < V_COMPLETE_FRAME - 1) then
                vertical_counter <= vertical_counter + 1;
            else
                vertical_counter <= 0;
            end if;
            end if;
            ---------------------------------------
            -- Sync configuration
            ---------------------------------------
            -- Horizontal Sync
            if ((horizontal_counter < 639 + H_FRONT_PORCH) OR (horizontal_counter
                H <= '1';
            else
                H <= '0';
            end if;
            -- Vertical Sync
            if ((vertical_counter < 479 + V_FRONT_PORCH) OR (vertical_counter >=
                V <= '1';
            else
                V <= '0';
            end if;
            ---------------------------------------
            -- Determine pixel avilability
            ---------------------------------------
            if ((horizontal_counter < 640) AND (vertical_counter < 480)) then
                video_active <= '1';
            else
                video_active <= '0';
            end if;
        end if;
    end if;
end process;
-----------------
-- Motion Update
-----------------
process (clk_motion_enable, SW0, SW2, player1_y, player2_y)
begin
    if (rising_edge(clk_motion_enable)) then
        ---------------------------
        -- Update Player 1 location
        ---------------------------
        if (SW0 = '0') then
            if (player1_y < 360) then
                player1_y <= player1_y + 1;
            else
                player1_y <= 360;
            end if;
        else
            if (player1_y > 40) then
                player1_y <= player1_y - 1;
            else
                player1_y <= 40;
            end if;
        end if;
```

```
    if (SW2 = '0') then
        if (player2_y < 360) then
            player2_y <= player2_y + 1;
        else
            player2_y <= 360;
        end if;
    else
        if (player2_y > 40) then
            player2_y <= player2_y - 1;
        else
            player2_y <= 40;
        end if;
    end if;
    ------------------------------
    -- Update direction
    ------------------------------
    if ( ball_x_pos >= player1_x and ball_x_pos < player1_x + 10) then
        -- Change direction when hit the left player
        if ( ((ball_y_pos >= player1_y) or (ball_y_pos + 10 >= player1_y)) an
            horizontal_direction <= '0';
        end if;
    elsif ( ball_x_pos = 40 ) then
        -- Change direction when hit left boundary
        if ( (ball_y_pos >=
```

```
            40 and ball_y_pos < 160) or (ball_y_pos + 10 >= 360 and ball_y_pos +
                horizontal_direction <= '0';
            end if;
        elsif (ball_x_pos + 10 > player2_x and ball_x_pos + 10 <= player2_x + 10)
            -- Change direction when hit the right player
            if ( ((ball_y_pos >= player2_y) or (ball_y_pos + 10 >= player2_y)) an
                horizontal_direction <= '1';
            end if;
        elsif (ball_x_pos + 10 = 600) then
            -- Change direction when hit right boundary
            if ( (ball_y_pos >= 40 and ball_y_pos < 160) or (ball_y_pos + 10 >= 3
                horizontal_direction <= '1';
            end if;
        end if;
        if ( ball_y_pos - 1 <= 40 ) then
            vertical_direction <= '1';
        elsif (ball_y_pos + 11 >= 440) then
            vertical_direction <= '0';
        end if;
        ----------------------------
        -- Update ball location
        ----------------------------
        if ((ball_x_pos > 0) and (ball_x_pos + 10) < 639) then
            -- Horizontal
            if (horizontal_direction = '0') then
                ball_x_pos <= ball_x_pos + 1;
            elsif (horizontal_direction = '1') then
                ball_x_pos <= ball_x_pos - 1;
            end if;
            -- Vertical
            if (vertical_direction = '0') then
                ball_y_pos <= ball_y_pos - 1;
            elsif(vertical_direction = '1') then
                ball_y_pos <= ball_y_pos + 1;
            end if;
        else
            ball_x_pos <= 300;
            ball_y_pos <= 220;
        end if;
    end if;
end process;
------------------------------
-- Pixel Color Set
------------------------------
process (video_active)
begin
    if (video_active = '0') then
        Rout <= (others => '0');
        Gout <= (others => '0');
        Bout <= (others => '0');
    else
        if (horizontal_counter >= 20 and horizontal_counter < 640 - 20 and vertic
            if ( vertical_counter < 40 or vertical_counter >= 440) then
                Rout <= (others => '1');  -- Display white for top & bottom borde
                Gout <= (others => '1');
                Bout <= (others => '1');
            elsif (((horizontal_counter < 40) OR (horizontal_counter >= 640 - 40)
                Rout <= (others => '1');  -- Display white for left & right borde
                Gout <= (others => '1');
                Bout <= (others => '1');
            elsif ((horizontal_counter >= ball_x_pos and horizontal_counter < bal
                Rout <= (others => '1');   -- Color Ball inside play field (gate
                Gout <= (others => '1');
```

```
                Bout <= (others => '1');
            elsif (((horizontal_counter < 40) OR (horizontal_counter >= 640 - 40
                Rout <= (others => '1');  -- Display white for left & right bord
                Gout <= (others => '1');
                Bout <= (others => '1');
            elsif ((horizontal_counter >= ball_x_pos and horizontal_counter < ba
                Rout <= (others => '1');   -- Color Ball inside play field (gate
                Gout <= (others => '1');
                Bout <= (others => '0');
            elsif ((horizontal_counter >= player1_x and horizontal_counter < pla
                Rout <= (others => '0');   -- Color Player 1
                Gout <= (others => '0');
                Bout <= (others => '1');
            elsif ((horizontal_counter >= player2_x and horizontal_counter < pla
                Rout <= (others => '1');   -- Color Player 2
                Gout <= (others => '0');
                Bout <= (others => '1');
            elsif (vertical_counter >= 40 and vertical_counter < 440 and horizon
                Rout <= (others => '1');   -- Color center line
                Gout <= (others => '1');
                Bout <= (others => '1');
            else
                Rout <= (others => '0');    -- Color background
                Gout <= (others => '1');
                Bout <= (others => '0');
            end if;
        elsif ((horizontal_counter >= ball_x_pos and horizontal_counter < ball_x
            Rout <= (others => '1');   -- Color Ball
            Gout <= (others => '0');
            Bout <= (others => '0');
        else
            Rout <= (others => '0');    -- Color background
            Gout <= (others => '1');
            Bout <= (others => '0');
        end if;
    end if;
end process;

DAC_CLK <= clk_divider;

end Behavioral;
```