Ryerson University

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

Course Title:	Digital Systems Engineering
Course Number:	COE 758
Semester/Year (e.g.F2016)	F2023

Instructor:	Dr. Vadim Geurkov
TA:	Abdulrahman Abdulghafar

Design Project/Lab #:	2
Design Project/Lab Title:	Sample Video Game Processor for VGA

Submission Date:	December 4, 2023
Due Date:	December 4, 2023

Student LAST NAME	Student FIRST NAME	Student Number	Section	Signature*
Adil	Tuba	500947421	7	T.A.
Manav	Patel	500967756	7	M.P.

*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

Table of Contents

Sample Video Game Processor for VGA

Abstract

This project delves into the implementation of a VGA interface, leveraging VGA protocols to design controllers and develop applications. At its core, the objective was to create an active ping-pong game, employing VHDL as an instructional framework for program control and display. The system adhered to VGA standards, relying on key signal elements such as sync pulses and RGB signals for visual output. The outcomes encompassed a cohesive display through pixel transitions, offering an engaging visual experience reminiscent of an arcade classic

Information

The Video Graphics Array (VGA), introduced by IBM in 1987, stands as a fundamental milestone in display technology, setting the standard for screen resolutions, operational modes, and hardware prerequisites. This project critically depends on the intricacies unique to VGA protocols, emphasizing the design of controllers and the development of applications. Understanding VGA signaling elements—sync pulses, color signals, and timing sequences—is pivotal for comprehensive exploration.

Furthermore, the project's main goal is to create an active ping-pong game controlled via the VGA board. VHDL is utilized as an instructional framework for controlling, displaying, and maneuvering the ping pong program showcasing the software's interface interaction and control implementation potential.

In this project, the ping pong paddles, maneuvered by users via LED switches, dictate the game's dynamics. The paddle controls are mapped to the board's switches, ensuring practicality in accommodating multiple players. The pixel-based display of the ping pong table is created through H-sync and V-sync settings, enabling dynamic pixel placement, notably influenced by the ball's motion. Governing color association, the RGB control signals offer a spectrum of versatile and vivid color displays.

The game demonstrates a cohesive display through a series of pixel transitions and dynamic animations, utilizing VHDL's capabilities. This aims to offer a visual experience resembling constant motion, encapsulating the fundamental elements of a timeless arcade classic.

Specifications

The project's backbone relies on adhering to VGA standards, where crucial elements within the VGA signal orchestrate the system's behavior and visual output. These elements include vertical and horizontal sync pulses that direct the monitor's focus and pixel placement within each line. Alongside these, the analog red, green, and blue signals are pivotal for colorization, carefully navigating through specific sections of each line to ensure a coherent visual representation of the ping pong game.

In sync with VGA standards, the Microcontroller Unit (MCU) plays a central role in interfacing with the VGA hardware. It acts as a translator, converting VHDL commands into tangible actions on the VGA board. The MCU's hardware configuration intricately connects with the VGA interface, facilitating seamless communication and control over the display mechanisms. Precise hardware specifications tailored to meet the demands of interfacing with VGA hardware constitute a critical facet of the project's execution.

Memory requirements and constraints are critical considerations in system design and implementation. The project delves into meticulous memory management, addressing the storage demands inherent to pixel data, game logic, and visual rendering. It navigates the balancing act of optimizing memory allocation strategies within the MCU's limitations, accommodating code, pixel configurations, and game dynamics efficiently within these constraints.

Device Design

Symbol Diagrams

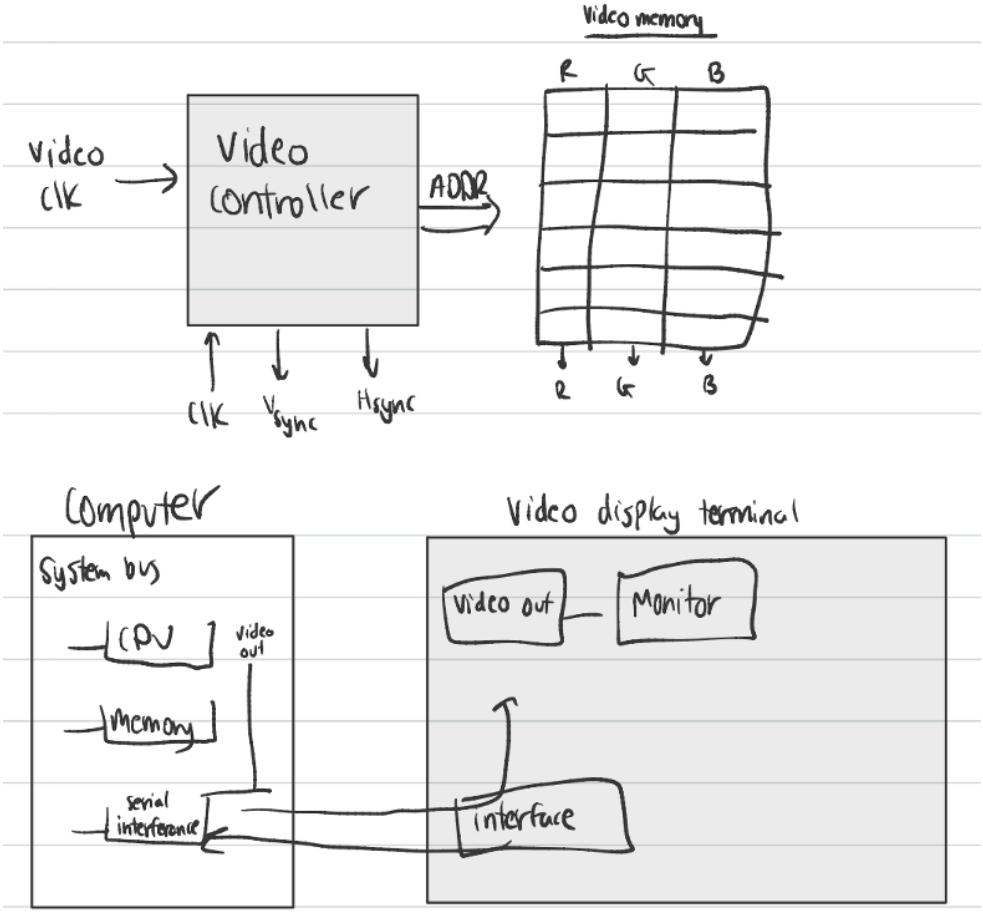

Figure 1: Symbol Diagrams of Individual components utilized for Ping Pong Game

VGA Specifications

The following VGA parameters were provided in the project manual:

Table I: VGA Horizontal Parameters

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

Figure 2: VGA Horizontal Parameters

The H-sync goes through the row waiting for the clock cycles to complete which is the units for the horizontal parameters. While the V-sync uses the H-sync lines to measure the length of time it takes, hence, the units were measured in Lines.

Table II: VGA Vertical Parameters

Parameter	Lines
Complete Frame	525
Front Porch	10
Sync Pulse	2
Back Porch	33
Active image area	480

Figure 3: VGA Horizontal Parameters

Block Diagrams

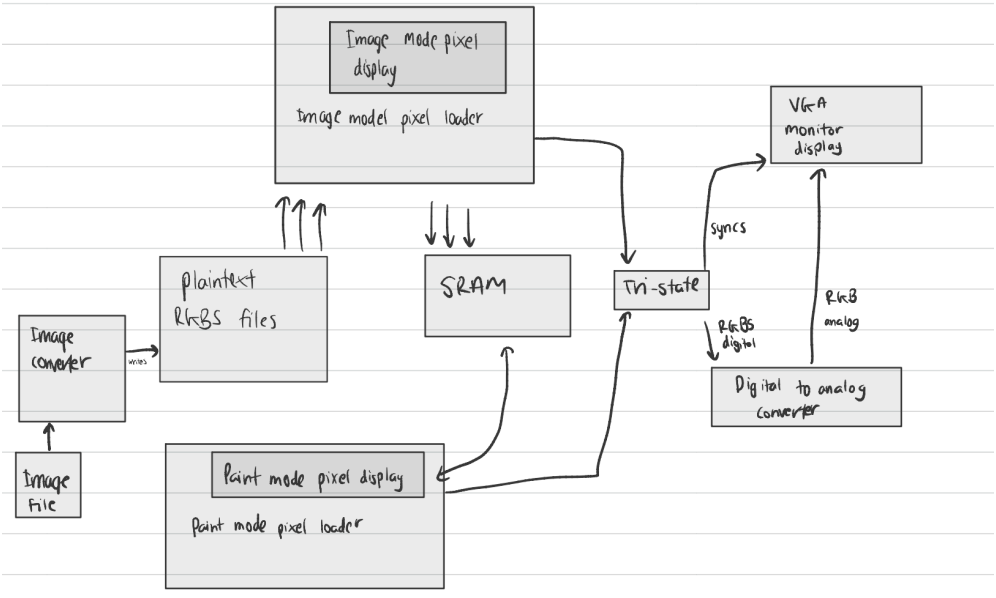

Figure 2: Block Diagram of Ping Pong Game

Process Diagram

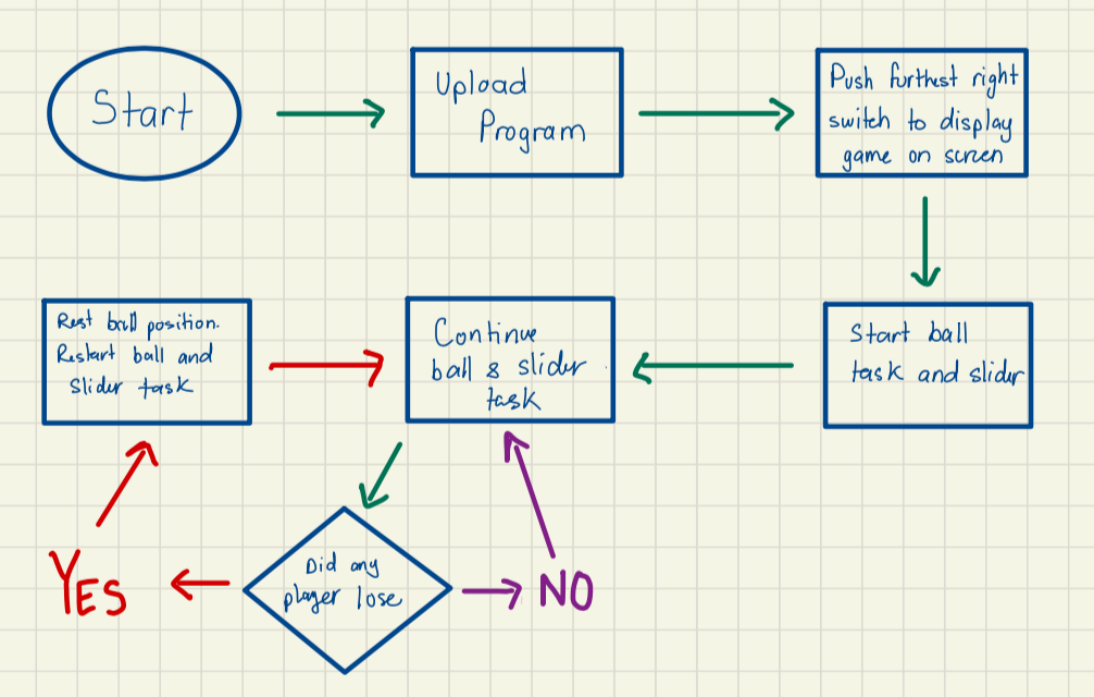

Figure 3: Process Diagram of Ping Pong Game

Results

Timing Diagrams

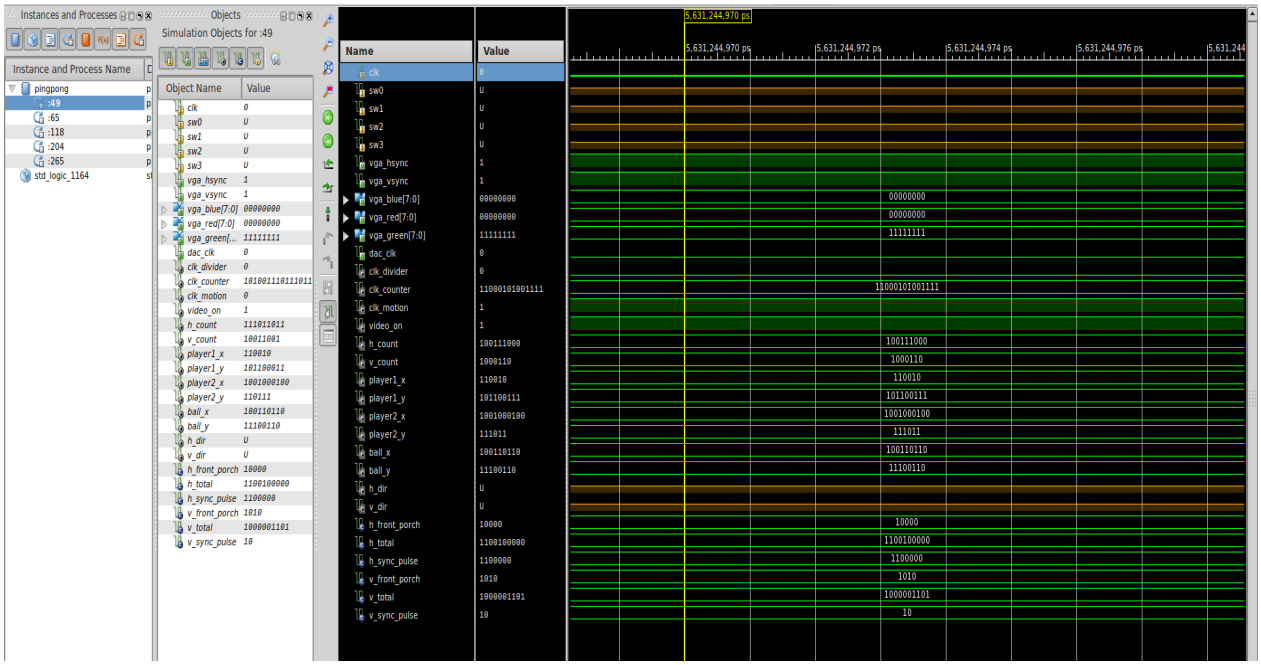

Figure 4: Vsync and Hsync Timing Diagrams

Static Video Frame

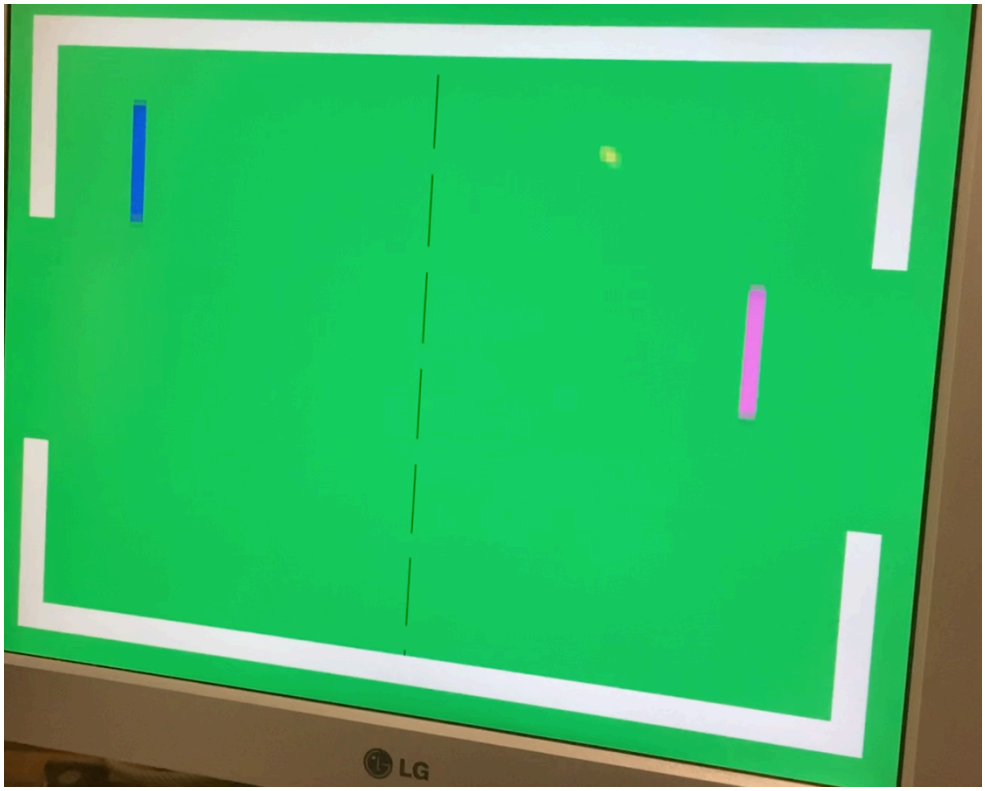

Figure 5: Ping Pong Game Static Frames (players playing game)

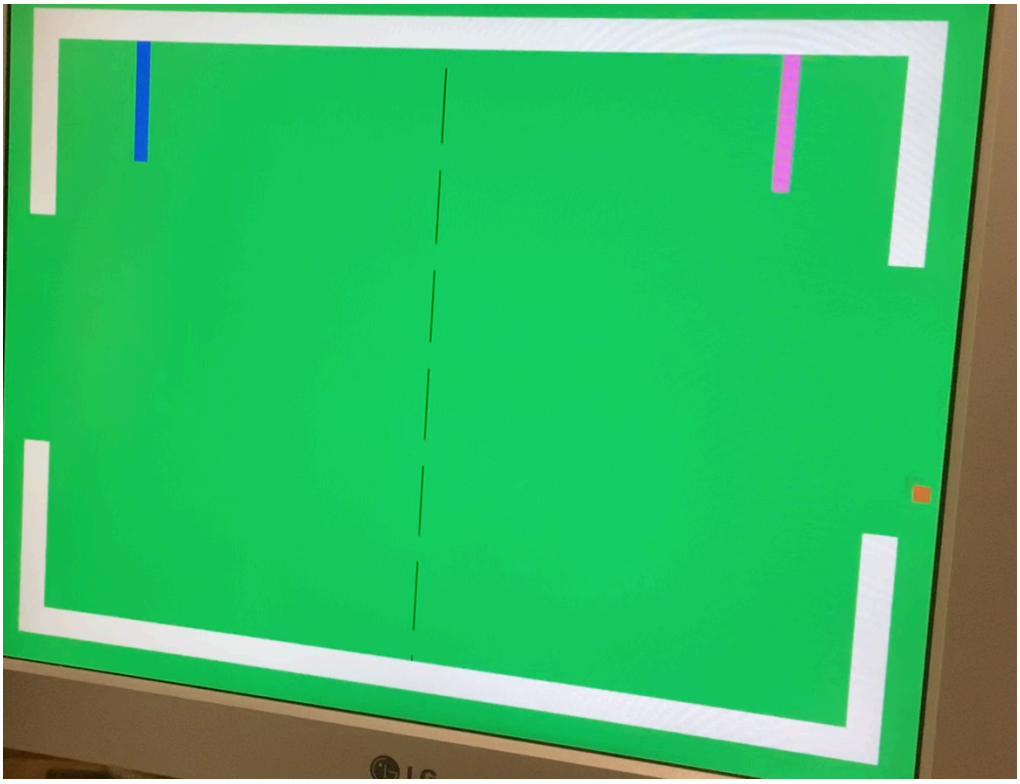

Figure 6: Ping Pong Game Static Frames (Ball Outside Boundary)

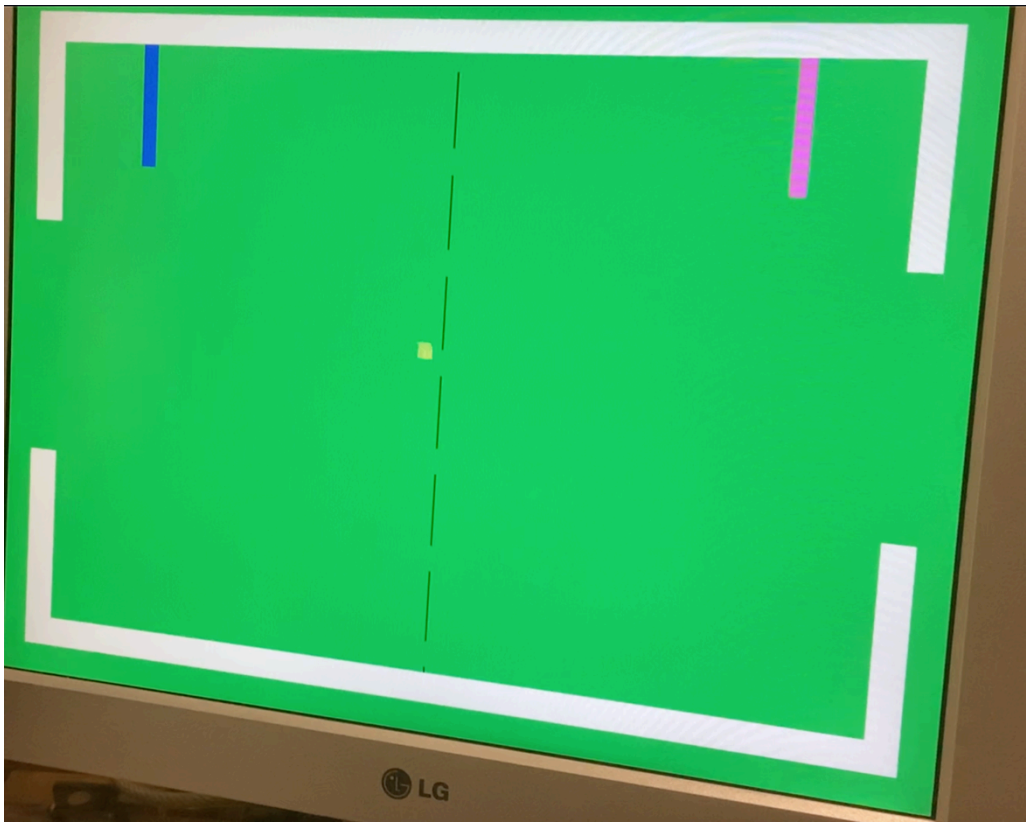

Figure 7: Ping Pong Game Static Frames (Ball Reset after Score)

The results encapsulated the functionality and efficacy of the VGA implementation. Screen captures displayed the ping-pong game with a green background, white boundaries, and pink/blue paddles for players one/two respectively. The ball, represented by a yellow square, turned red upon exiting the boundaries. Insights from timing diagrams and captures highlighted the system's functionality. However, limitations were observed where the ball occasionally passed through the paddles from their backside due to object space recognition issues and lack of virtual 2D space within the code. Overall, the project showcased a functional game, responsive controls, and successful VGA board integration, albeit with minor recognition challenges.

Conclusion

In conclusion, the Ping Pong project was shown to be a functioning game and was responsive to the controls mentioned in the lab manual. Also, the active ping-pong game was controlled via the VGA board. The LED switches dictate the game's dynamics. As switches move up and down, the paddle would respond to that and move up or down. The pixel-based display of the game was created through H-sync and V-sync settings. However, one problem that did occur was that the ball did not recognize some parts of the paddle, hence making it go right through the paddle. Overall, the project was successfully completed while understanding the necessary information.

References

1. Kirischian, L. COE 758 – Labs – Project 2: Simple Video Game [11-11-11]. (2023). Retrieved from http://www.ee.ryerson.ca/~lkirisch/ele758/labs/SimpleVideoGame%5B11-11-11%5D.pdf

2. Research. (n.d.). Video Graphics Array (VGA). [Online]. Retrieved from https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/raf225_dah322/

3. Resource. (n.d.) Computer Design Lab 3 VGA [Online]. Retrieved from https://my.eng.utah.edu/~cs3710/labs/VGA.pdf

Appendix

Ping Pong.vhd

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

entity PingPong is
    Port ( Clk : in  STD_LOGIC;
           SW0 : in  STD_LOGIC;
           SW1 : in  STD_LOGIC;
           SW2 : in  STD_LOGIC;
           SW3 : in  STD_LOGIC;
           VGA_HSYNC : out  STD_LOGIC;
           VGA_VSYNC : out  STD_LOGIC;
           VGA_BLUE : out  STD_LOGIC_VECTOR (7 downto 0);
           VGA_RED : out  STD_LOGIC_VECTOR (7 downto 0);
           VGA_GREEN : out  STD_LOGIC_VECTOR (7 downto 0);
           DAC_CLK : out  STD_LOGIC);
end PingPong;

architecture Behavioral of PingPong is

    -- VGA Known Parameters
    constant H_FRONT_PORCH : integer := 16;
    constant H_TOTAL : integer := 800;
    constant H_SYNC_PULSE : integer := 96;
    constant V_FRONT_PORCH : integer := 10;
    constant V_TOTAL : integer := 525;
    constant V_SYNC_PULSE : integer := 2;

    -- Clock Generation Signals
    signal Clk_DIVIDER : STD_LOGIC := '0';
    signal Clk_COUNTER : integer := 0;
    signal Clk_MOTION : STD_LOGIC := '0';

    -- Pixel display signals
    signal VIDEO_ON : STD_LOGIC;
    signal H_COUNT : integer := 0;
    signal V_COUNT : integer := 0;
    signal PLAYER1_X : integer := 100; -- 50
    signal PLAYER1_Y : integer := 360;
    signal PLAYER2_X : integer := 530; -- 580
    signal PLAYER2_Y : integer := 60;
    signal BALL_X : integer := 310;
    signal BALL_Y : integer := 230; -- center = (360, 240)
    signal H_DIR : STD_LOGIC; -- 0 = right, 1 = left
    signal V_DIR : STD_LOGIC; -- 0 = down,  1  = up
```

```
begin

    -- Clock Divider
    process (Clk)
    begin
        if (rising_edge(Clk)) then
            Clk_DIVIDER <= NOT Clk_DIVIDER;

            -- Reset the clock counter every 100,000 cycles to generate a motion signal.
            if (Clk_COUNTER  = 100000) then
                Clk_MOTION <= NOT Clk_MOTION;
                Clk_COUNTER <= 0;
            else
                Clk_COUNTER <= Clk_COUNTER + 1;
            end if;
        end if;
    end process;

    -- Pixel Configuration
    process (Clk_DIVIDER, SW3)
    begin
        -- Reset Switch
        if (SW3 = '1') then
            H_COUNT <= 0;
            V_COUNT <= 0;
            VGA_HSYNC <= '1';
            VGA_VSYNC <= '0';
            VIDEO_ON <= '0';
        else
            if (rising_edge(Clk_DIVIDER)) then

                -- Counter for pixel location calculation
                if (H_COUNT < H_TOTAL - 1) then
                    H_COUNT <= H_COUNT + 1;
                else
                    H_COUNT <= 0;

                    if (V_COUNT < V_TOTAL - 1) then
                        V_COUNT <= V_COUNT + 1;
                    else
                        V_COUNT <= 0;
                    end if;
                end if;

                -- Sync configuration
                -- Horizontal Sync
                if ((H_COUNT < 639 + H_FRONT_PORCH) OR (H_COUNT >= 639 + H_FRONT_PORCH + H_SYNC_PULSE))
then
                    VGA_HSYNC <= '1';
                else
                    VGA_HSYNC <= '0';
                end if;
```

```
                -- Vertical Sync
                if ((V_COUNT < 479 + V_FRONT_PORCH) OR (V_COUNT >= 479 + V_FRONT_PORCH + V_SYNC_PULSE))
then
                    VGA_VSYNC <= '1';
                else
                    VGA_VSYNC <= '0';
                end if;

                -- Determine pixel availability
                if ((H_COUNT < 640) AND (V_COUNT < 480)) then
                    VIDEO_ON <= '1';
                else
                    VIDEO_ON <= '0';
                end if;
            end if;
        end if;
    end process;

    -----------------
    -- Motion Update
    -- Motion Update process
process (Clk_MOTION, SW0, SW2, PLAYER1_Y, PLAYER2_Y)
begin
    if (rising_edge(Clk_Motion)) then
        -- Update Player 1 location
        if (SW0 = '0') then
            if (PLAYER1_Y < 359) then
                PLAYER1_Y <= PLAYER1_Y + 1; -- Increase the height of the paddle
            else
                PLAYER1_Y <= 359;
            end if;
        else
            if (PLAYER1_Y > 40) then
                PLAYER1_Y <= PLAYER1_Y - 1; -- Increase the height of the paddle
            else
                PLAYER1_Y <= 40;
            end if;
        end if;

        -- Update Player 2 location
        if (SW2 = '0') then
            if (PLAYER2_Y < 359) then
                PLAYER2_Y <= PLAYER2_Y + 1; -- Increase the height of the paddle
            else
                PLAYER2_Y <= 359;
            end if;
        else
            if (PLAYER2_Y > 40) then
                PLAYER2_Y <= PLAYER2_Y - 1; -- Increase the height of the paddle
            else
                PLAYER2_Y <= 40;
            end if;
        end if;

        -- Check if the ball goes out of bounds on the left or right
        if (BALL_X <= 0 or BALL_X >= 640) then
            -- Reset the ball to the center
            BALL_X <= 310; -- Adjust to the desired center X position
            BALL_Y <= 230; -- Adjust to the desired center Y position
```

```
            -- Change the direction of the ball (opposite direction)
        --      H_DIR <= NOT H_DIR;
          end if;

          -- Update direction
          if ( BALL_X >= PLAYER1_X and BALL_X < PLAYER1_X + 10) then
              -- Change direction when hit the left player
              if ( ((BALL_Y >= PLAYER1_Y) or (BALL_Y + 10 >= PLAYER1_Y)) and ((BALL_Y < PLAYER1_Y + 80) or
(BALL_Y + 10 < PLAYER1_Y + 80)) ) then
                  H_DIR <= '0';
              end if;
          elsif ( BALL_X = 40 ) then
              -- Change direction when hit the left boundary
              if ( (BALL_Y >= 40 and BALL_Y < 160) or (BALL_Y + 10 >= 360 and BALL_Y + 10 < 440) ) then
                  H_DIR <= '0';
              end if;
          elsif (BALL_X + 10 > PLAYER2_X and BALL_X + 10 <= PLAYER2_X + 10) then
              -- Change direction when hit the right player
              if ( ((BALL_Y >= PLAYER2_Y) or (BALL_Y + 10 >= PLAYER2_Y)) and ((BALL_Y < PLAYER2_Y + 80) or
(BALL_Y + 10 < PLAYER2_Y + 80)) ) then
                  H_DIR <= '1';
              end if;
          elsif (BALL_X + 10 = 600) then
              -- Change direction when hit the right boundary
              if ( (BALL_Y >= 40 and BALL_Y < 160) or (BALL_Y + 10 >= 360 and BALL_Y + 10 < 440) ) then
                  H_DIR <= '1';
              end if;
          end if;

          if ( BALL_Y - 1 <= 40 ) then
              V_DIR <= '1';
          elsif (BALL_Y + 11 >= 440) then
              V_DIR <= '0';
          end if;

          -- Update ball location
          if ((BALL_X > 0) and (BALL_X + 10) < 639) then
              -- Horizontal
              if (H_DIR = '0') then
                  BALL_X <= BALL_X + 1;
              elsif (H_DIR = '1') then
                  BALL_X <= BALL_X - 1;
              end if;
              -- Vertical
              if (V_DIR = '0') then
                  BALL_Y <= BALL_Y - 1;
              elsif(V_DIR = '1') then
                  BALL_Y <= BALL_Y + 1;
              end if;
          else
              BALL_X <= 300;
              BALL_Y <= 220;
          end if;
      end if;
end process;
```

```
    process (VIDEO_ON)
    begin
        if (VIDEO_ON = '0') then
            VGA_RED <= (others => '0');
            VGA_GREEN <= (others => '0');
            VGA_BLUE <= (others => '0');
        else
            if (H_COUNT >= 20 and H_COUNT < 640 - 20 and V_COUNT >=20 and V_COUNT < 460) then
                if ( V_COUNT < 40 or V_COUNT >= 440) then
                    VGA_RED <= (others => '1');  -- Display white for top & bottom border
                    VGA_GREEN <= (others => '1');
                    VGA_BLUE <= (others => '1');
                elsif (((H_COUNT < 40) OR (H_COUNT >= 640 - 40)) and (V_COUNT < 160 or V_COUNT >= 320))
then
                    VGA_RED <= (others => '1');  -- Display white for left & right border
                    VGA_GREEN <= (others => '1');
                    VGA_BLUE <= (others => '1');
                elsif ((H_COUNT >= BALL_X and H_COUNT < BALL_X + 10) and (V_COUNT >= BALL_Y and V_COUNT <
BALL_Y + 10) )then
                    VGA_RED <= (others => '1');   -- Color Ball inside play field (gate + border)
                    VGA_GREEN <= (others => '1');
                    VGA_BLUE <= (others => '0');
                elsif ((H_COUNT >= PLAYER1_X and H_COUNT < PLAYER1_X + 10) and (V_COUNT >= PLAYER1_Y and
V_COUNT < PLAYER1_Y + 80) )then
                    VGA_RED <= (others => '0');   -- Color Player 1
                    VGA_GREEN <= (others => '0');
                    VGA_BLUE <= (others => '1');
                elsif ((H_COUNT >= PLAYER2_X and H_COUNT < PLAYER2_X + 10) and (V_COUNT >= PLAYER2_Y and
V_COUNT < PLAYER2_Y + 80) )then
                    VGA_RED <= (others => '1');   -- Color Player 2
                    VGA_GREEN <= (others => '0');
                    VGA_BLUE <= (others => '1');
                elsif ((V_COUNT >= 40 and V_COUNT < 440) and (H_COUNT = 320)) and (((V_COUNT - 35) mod 64)
> 16) then
                    VGA_Red <= (others => '0');   -- Color center line
                    VGA_Green <= (others => '0');
                    VGA_Blue <= (others => '0');
                else
                    VGA_RED <= (others => '0'); -- Color background
                    VGA_GREEN <= (others => '1');
                    VGA_BLUE <= (others => '0');
                end if;
            elsif ((H_COUNT >= BALL_X and H_COUNT < BALL_X + 10) and (V_COUNT >= BALL_Y and V_COUNT <
BALL_Y + 10) )then
                VGA_RED <= (others => '1');   -- Color Ball
                VGA_GREEN <= (others => '0');
                VGA_BLUE <= (others => '0');
            else
                VGA_RED <= (others => '0'); -- Color background
                VGA_GREEN <= (others => '1');
                VGA_BLUE <= (others => '0');
            end if;
        end if;
    end process;
```

```
    -- Output the clock divider signal
    DAC_CLK <= Clk_DIVIDER;

end Behavioral;
```