ENGR 378 Digital Systems Design

Exp. 6: VGA Controller and Pong Game

Submitted by:
Moses Martinez
Dylan Wright
May 15, 2018

Introduction:

The purpose of this lab was to learn how to draw patterns onto the screen by using the red, blue, and green signals through the VGA port. Once we had a strong comprehension of how to manipulate the display on the screen, we were tasked on creating a copy of the Pong game. Using all our knowledge on FPGA boards we used two seven segment displays, four buttons, and a monitor to implement a replica of Pong.

Problem Analysis:

We began our process by drawing our borders we did this to give us a better understanding on how to draw the rest of the game. Once we had our borders established we were able to derive the logic behind when the paddles reaches one of the borders, when the ball passes a paddle, and when the ball hits a paddle. We use these limits to draw the rest of the components of the game and to determine the logic behind the functionality of the game. We then needed to find a balance between the movement of the ball and the velocity of the ball. Using the information provided to us in the lab hand out, we implemented the movement of our ball using a state machine. The logic is the if the ball is moving in the up direction and then hits a border it will switch to the state where it is moving down. Similarly we used the same state machine logic to determine the horizontal direction. When the ball is moving left the proceeds to hit a border it will switch to the right state. Using a combination of both horizontal or vertical, we determined the four states that ball can be in. Up to the left, up to the right, down to the left, and down to the right. The rest of the game's code could be implemented using code from previous lab.

Hardware Design:

Most of the hardware design for the pong game involved setting up the horizontal and vertical sync signals, the code for with was included in the VGA sample module given to us at the beginning of the presentation.

VHDL Modeling:

When the clock signal goes from left to right, top to bottom, a PixelPosition variable is set that determines where the scan is currently drawing. If statements are used to determine the x and y boundaries and which colors to be drawn.

Other variables, individual x and y components for two paddles and a ball, are used to determine the top left pixel of each. These variables are then modified and checked each redraw phase to control motion.

Collisions detection is performed by checking relative ball positions to relative paddle positions. If the ball is within the paddle bounds, it bounces back. If the ball is less than or greater than a certain y (the top and bottom borders) the ball will bounce. In order to control bouncing, the ball goes through states of moving up-left, up-right, down-left, and down-right. Each of the states causes the ball to move in a certain direction. When it hits a wall or a paddle, the ball transitions to another state based on the last state and the side of the wall.

When the ball goes past a paddle, an individual score variable is incremented, and the ball is reset back to the center of the screen. The score is then displayed on a seven segment display.

Conclusion and Discussion:

Upon completing this lab we were successful in creating a replica of Pong, using the FPGA on board buttons, seven segment displays, and the VGA signals. Some issues that we ran into was that our ball would jump down several pixel either up and down for no reason at all. We determined that the ball was experience an overflow issue which would cause the ball to jump to a random position. Using the state machine helped resolve this issue. We first implemented our ball to move constantly by adding or subtracting the x and y pixels. We also had some issues with our reset functionality. Our idea was that whenever the reset switch was turned on or the reset signal was set to on the paddles would move back to the middle and the ball would return the center of the screen. The issue we had was both a compiler error and runtime error.When we created separate processes for the paddles and the ball we were able to fix all issues that were related to the reset functionality. This lab taught us more about the possibilities that our available when using an FPGA board.

HDL Source Code:

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

entity VGAInterface is
    Port ( CLOCK_50: in  STD_LOGIC;
           RESET : in  STD_LOGIC;
           VGA_R : out  STD_LOGIC_VECTOR (7 downto 0);
           VGA_G : out  STD_LOGIC_VECTOR (7 downto 0);
           VGA_B : out  STD_LOGIC_VECTOR (7 downto 0);
           VGA_HS : out  STD_LOGIC;
           VGA_VS : out  STD_LOGIC;
          VGA_BLANK_N : out  STD_LOGIC;
          VGA_CLK : out  STD_LOGIC;
          VGA_SYNC_N : out  STD_LOGIC;
           KEY : in  STD_LOGIC_VECTOR (3 downto 0);
           SW : in  STD_LOGIC_VECTOR (17 downto 0);
           HEX0 : out  STD_LOGIC_VECTOR (6 downto 0);
           HEX1 : out  STD_LOGIC_VECTOR (6 downto 0);
          HEX2 : out  STD_LOGIC_VECTOR (6 downto 0);
          HEX3 : out  STD_LOGIC_VECTOR (6 downto 0);
          HEX4 : out  STD_LOGIC_VECTOR (6 downto 0);
          HEX5 : out  STD_LOGIC_VECTOR (6 downto 0);
          HEX6 : out  STD_LOGIC_VECTOR (6 downto 0);
          HEX7 : out  STD_LOGIC_VECTOR (6 downto 0);
          LEDR : out  STD_LOGIC_VECTOR (17 downto 0);
          LEDG : out  STD_LOGIC_VECTOR (8 downto 0));
end VGAInterface;
```

```
architecture Behavioral of VGAInterface is

     component VGAFrequency is -- Altera PLL used to generate 108Mhz
clock
     PORT (areset          : IN STD_LOGIC;
           inclk0          : IN STD_LOGIC;
           C0              : OUT STD_LOGIC ;
           locked          : OUT STD_LOGIC);
     end component;

     component VGAController is -- Module declaration for the VGA
controller
     Port ( PixelClock : in  STD_LOGIC;
           inRed : in STD_LOGIC_VECTOR (7 downto 0);
                  inGreen : in STD_LOGIC_VECTOR (7 downto 0);
                  inBlue : in STD_LOGIC_VECTOR (7 downto 0);
                  outRed : out STD_LOGIC_VECTOR (7 downto 0);
                  outGreen : out STD_LOGIC_VECTOR (7 downto 0);
                  outBlue : out STD_LOGIC_VECTOR (7 downto 0);
           VertSynchOut : out  STD_LOGIC;
           HorSynchOut : out  STD_LOGIC;
           XPosition : out  STD_LOGIC_VECTOR (10 downto 0);
           YPosition : out  STD_LOGIC_VECTOR (10 downto 0));
     end component;

     -- Variables for screen resolution 1280 x 1024
     signal XPixelPosition : STD_LOGIC_VECTOR (10 downto 0);
     signal YPixelPosition : STD_LOGIC_VECTOR (10 downto 0);

     signal redValue : STD_LOGIC_VECTOR (7 downto 0)  := "00000000";
     signal greenValue :STD_LOGIC_VECTOR (7 downto 0)  := "00000000";
     signal blueValue : STD_LOGIC_VECTOR (7 downto 0)  := "00000000";

     -- Freq Mul/Div signals (PLL I/O variables used to generate
108MHz clock)
     constant resetFreq : STD_LOGIC := '0';
     signal PixelClock: STD_LOGIC;
     signal lockedPLL : STD_LOGIC; -- dummy variable

     -- Variables used for left paddle
     signal XPanelLeftPos : STD_LOGIC_VECTOR (10 downto 0)  :=
"00010101010";
     signal YPanelLeftPos : STD_LOGIC_VECTOR (10 downto 0)  :=
"00110110101";
```

```
     --signal displayPosition : STD_LOGIC_VECTOR (10 downto 0) :=
"01000000000";

     -- Variables used for right paddle
     signal XPanelRightPos : STD_LOGIC_VECTOR (10 downto 0) :=
"10000111000";
     signal YPanelRightPos : STD_LOGIC_VECTOR (10 downto 0) :=
"00110110101";

     -- Variables used for ball (square)
     signal XPanelBallPos : STD_LOGIC_VECTOR (10 downto 0) :=
"01001110001";
     signal YPanelBallPos : STD_LOGIC_VECTOR (10 downto 0) :=
"00111110001";

     -- Variables for slow clock counter to generate a slower clock
     signal slowClockCounter : STD_LOGIC_VECTOR (20 downto 0) :=
"000000000000000000000";
     signal slowClock : STD_LOGIC;

     -- Vertical and Horizontal Synch Signals
     signal HS : STD_LOGIC; -- horizontal synch
     signal VS : STD_LOGIC; -- vertical synch

     -- State of Ball
     type STATEBALL is (UL, UR, DL, DR);
     signal STATE : STATEBALL := UR;

     -- Reset the game
     signal RESETSIGNAL : STD_LOGIC := '0';

     -- Player Scores
     signal P1SCORE : STD_LOGIC_VECTOR (3 downto 0) := "0000";
     signal P2SCORE : STD_LOGIC_VECTOR (3 downto 0) := "0000";

begin

     process (CLOCK_50)-- control process for a large counter to
generate a slow clock
     begin
          if CLOCK_50'event and CLOCK_50 = '1' then
               slowClockCounter <= slowClockCounter + 1;
          end if;
     end process;
```

```vhdl
    slowClock <= slowClockCounter(20); -- slow clock signal

    process (slowClock)-- move right paddle
    begin
        if slowClock'event and slowClock= '1' then
            if RESET = '1' then
                YPanelRightPos <= "00110110101";
            elsif KEY(0) = '0' and YPanelRightPos < 744 then --
detect button 0 pressed
                YPanelRightPos <= YPanelRightPos + 9;
            elsif KEY(1) = '0' and YPanelRightPos > 130 then --
detect button 1 pressed
                YPanelRightPos <= YPanelRightPos - 9;
            end if;
        end if;
    end process;

    process (slowClock)-- move left paddle
    begin
        if slowClock'event and slowClock = '1' then
            if RESET = '1' then
                YPanelLeftPos <= "00110110101";
            elsif KEY(2) = '0' and YPanelLeftPos < 744 then --
detect button 2 pressed
                YPanelLeftPos <= YPanelLeftPos + 9;
            elsif KEY(3) = '0' and YPanelLeftPos > 130 then--
detect button 3 pressed
                YPanelLeftPos <= YPanelLeftPos - 9;
            end if;
        end if;
    end process;

    process (slowClock)-- move ball
    begin
        if slowClock'event and slowClock = '1' then
            -- Handle Resets
            if RESET = '1' or RESETSIGNAL = '1' then
                -- Randomizes ball initial position
                case STATE is
                    when UL => STATE <= DL;
                    when DL => STATE <= DR;
                    when DR => STATE <= UR;
                    when others => STATE <= UL;
                end case;
                -- Resets score if hard reset
```

```
      if RESET = '1' then
            P1SCORE <= "0000";
            P2SCORE <= "0000";
      else
            P1SCORE <= P1SCORE;
            P2SCORE <= P2SCORE;
      end if;
      -- Forces ball to center
      XPanelBallPos <= "01001110001";
      YPanelBallPos <= "00111110001";
-- Moves ball based on state
elsif STATE = UL then
      XPanelBallPos <= XPanelBallPos - 15;
      YPanelBallPos <= YPanelBallPos - 15;
elsif STATE = UR then
      XPanelBallPos <= XPanelBallPos + 15;
      YPanelBallPos <= YPanelBallPos - 15;
elsif STATE = DL then
      XPanelBallPos <= XPanelBallPos - 15;
      YPanelBallPos <= YPanelBallPos + 15;
else
      XPanelBallPos <= XPanelBallPos + 15;
      YPanelBallPos <= YPanelBallPos + 15;
end if;

-- Bounce off top
if YPanelBallPos <= 130 then
      if STATE = UL then
            STATE <= DL;
      elsif STATE = UR then
            STATE <= DR;
      else
            STATE <= STATE;
      end if;
-- Bounce off bottom
elsif YPanelBallPos >= 866 then
      if STATE = DL then
            STATE <= UL;
      elsif STATE = DR then
            STATE <= UR;
      else
            STATE <= STATE;
      end if;
end if;
```

```vhdl
-- Bounce off left paddle
if YPanelBallPos <= YPanelLeftPos + 150 and
    YPanelBallPos + 30 >= YPanelLeftPos and
    XPanelBallPos <= XPanelLeftPos + 30 and
    XPanelBallPos >= XPanelLeftPos then
    if STATE = UL then
        STATE <= UR;
    elsif STATE = DL then
        STATE <= DR;
    else
        STATE <= STATE;
    end if;
-- Bounce off right paddle
elsif YPanelBallPos <= YPanelRightPos + 150 and
    YPanelBallPos + 30 >= YPanelRightPos and
    XPanelBallPos + 30 >= XPanelRightPos and
    XPanelBallPos + 30 <= XPanelRightPos + 30 then
    if STATE = UR then
        STATE <= UL;
    elsif STATE = DR then
        STATE <= DL;
    else
        STATE <= STATE;
    end if;
end if;

-- Update score
if XPanelBallPos <= 100 then
    if RESETSIGNAL = '0' then
        if P1SCORE < 9 then
            P1SCORE <= P1SCORE + 1;
        else
            P1SCORE <= "0000";
        end if;
    else
        P1SCORE <= P1SCORE;
    end if;
    RESETSIGNAL <= '1';
elsif XPanelBallPos >= 1150 then
    if RESETSIGNAL = '0' then
        if P2SCORE < 9 then
            P2SCORE <= P2SCORE + 1;
        else
            P2SCORE <= "0000";
        end if;
```

```
                else
                    P2SCORE <= P2SCORE;
                end if;
                RESETSIGNAL <= '1';
            else
                RESETSIGNAL <= '0';
            end if;
        end if;
    end process;


    -- Generates a 108Mhz frequency for the pixel clock using the
PLL (The pixel clock determines how much time there is between
drawing one pixel at a time)
    VGAFreqModule : VGAFrequency port map (resetFreq, CLOCK_50,
PixelClock, lockedPLL);

    -- Module generates the X/Y pixel position on the screen as
well as the horizontal and vertical synch signals for monitor with
1280 x 1024 resolution at 60 frams per second
    VGAControl : VGAController port map (PixelClock, redValue,
greenValue, blueValue, VGA_R, VGA_G, VGA_B, VS, HS, XPixelPosition,
YPixelPosition);

    -- OUTPUT ASSIGNMENTS FOR VGA SIGNALS
    VGA_VS <= VS;
    VGA_HS <= HS;
    VGA_BLANK_N <= '1';
    VGA_SYNC_N <= '1';
    VGA_CLK <= PixelClock;

    -- OUTPUT ASSIGNEMNTS TO SEVEN SEGMENT DISPLAYS
    HEX1 <= "1111111"; -- display 0
    HEX2 <= "1111111"; -- display 0
    HEX3 <= "1111111"; -- display 0
    HEX4 <= "1111111"; -- display 0
    HEX5 <= "1111111"; -- display 0
    HEX6 <= "1111111"; -- display 0

    -- COLOR ASSIGNMENT STATEMENTS
    process (PixelClock)-- MODIFY CODE HERE TO DISPLAY COLORS IN
DIFFERENT REGIONS ON THE SCREEN
    begin
        if PixelClock'event and PixelClock = '1' then
```

```
                -- Define RGB
--              redValue <= XPixelPosition(7 downto 0);
--              blueValue <= YPixelPosition(7 downto 0);
--              greenValue <= YPixelPosition(9 downto 2);

                -- Draw borders
                --TOP
                if (XPixelPosition < 100) then
                     redValue <= "00000000";
                     blueValue <= "00000000";
                     greenValue <= "11111111";
                elsif (XPixelPosition > 1180) then
                     redValue <= "00000000";
                     blueValue <= "00000000";
                     greenValue <= "11111111";
                elsif (YPixelPosition < 130) then
                     redValue <= "11111111";
                     blueValue <= "11111111";
                     greenValue <= "00000000";
                elsif (YPixelPosition > 894) then
                     redValue <= "11111111";
                     blueValue <= "11111111";
                     greenValue <= "00000000";
                -- Draw left paddle
                elsif (XPixelPosition > XPanelLeftPos
                     and XPixelPosition < XPanelLeftPos + 30
                     and YPixelPosition > YPanelLeftPos
                     and YPixelPosition < YPanelLeftPos + 150) then
                     redValue <= "00000000";
                     blueValue <= "11111111";
                     greenValue <= "00000000";
                -- Draw right paddle
                elsif (XPixelPosition > XPanelRightPos
                     and XPixelPosition < XPanelRightPos + 30
                     and YPixelPosition > YPanelRightPos
                     and YPixelPosition < YPanelRightPos + 150) then
                     redValue <= "00000000";
                     blueValue <= "11111111";
                     greenValue <= "00000000";
                -- Draw ball
                elsif (XPixelPosition > XPanelBallPos
                     and XPixelPosition < XPanelBallPos + 30
                     and YPixelPosition > YPanelBallPos
                     and YPixelPosition < YPanelBallPos + 30) then
                     redValue <= "11111111";
                     blueValue <= "00000000";
                     greenValue <= "00000000";
                else
                     redValue <= "00000000";
                     blueValue <= "00000000";
                     greenValue <= "00000000";
                end if;


--              if SW(0) = '0' then -- display three different colors
to screen
--                   redValue <= XPixelPosition(7 downto 0);
--                   blueValue <= YPixelPosition(7 downto 0);
--                   greenValue <= YPixelPosition(9 downto 2);
--                   -- RED
--                   if (XPixelPosition < 160) then
--                        redValue <= "11111111";
--                        blueValue <= "00000000";
--                        greenValue <= "00000000";
--                   -- BLUE
--                   elsif (XPixelPosition < 320) then
--                        redValue <= "00000000";
--                        blueValue <= "11111111";
--                        greenValue <= "00000000";
--                   -- GREEN
--                   elsif (XPixelPosition < 480) then
--                        redValue <= "00000000";
--                        blueValue <= "00000000";
--                        greenValue <= "11111111";
--                   -- PURPLE
--                   elsif (XPixelPosition < 640) then
--                        redValue <= "11111111";
--                        blueValue <= "11111111";
--                        greenValue <= "00000000";
--                   -- TURQUOISE
--                   elsif (XPixelPosition < 800) then
--                        redValue <= "00000000";
--                        blueValue <= "11111111";
--                        greenValue <= "11111111";
--                   -- YELLOW
--                   elsif (XPixelPosition < 960) then
--                        redValue <= "11111111";
--                        blueValue <= "00000000";
--                        greenValue <= "11111111";
--                   -- WHITE
--                   elsif (XPixelPosition < 1120) then
--                        redValue <= "11111111";
--                        blueValue <= "11111111";
--                        greenValue <= "11111111";
--                   -- BLACK
--                   else
--                        redValue <= "00000000";
--                        blueValue <= "00000000";
--                        greenValue <= "00000000";
--                   end if;
--              else -- display the white dot and the black
background
--                   if (XPixelPosition = XPanelLeftPos AND
YPixelPosition = YPanelLeftPos) then
--                        redValue <= "11111111";
--                        blueValue <= "11111111";
--                        greenValue <= "11111111";
--                   else
--                        redValue <= "00000000";
--                        blueValue <= "00000000";
--                        greenValue <= "00000000";
--                   end if;
--              end if;

          end if;
     end process;

process (P2SCORE)
begin
  case P2SCORE is
    when "0000" => HEX7 <= "1000000";
      when "0001" => HEX7 <= "1111001";
      when "0010" => HEX7 <= "0100100";
      when "0011" => HEX7 <= "0110000";
      when "0100" => HEX7 <= "0011001";
      when "0101" => HEX7 <= "0010010";
      when "0110" => HEX7 <= "0000010";
      when "0111" => HEX7 <= "1111000";
      when "1000" => HEX7 <= "0000000";
      when "1001" => HEX7 <= "0010000";
      when others  => HEX7 <= "1111111";
  end case;
end process;

process (P1SCORE)
```

```
begin

  case P1SCORE is
    when "0000" => HEX0 <= "1000000";
      when "0001" => HEX0 <= "1111001";
      when "0010" => HEX0 <= "0100100";
      when "0011" => HEX0 <= "0110000";
      when "0100" => HEX0 <= "0011001";
      when "0101" => HEX0 <= "0010010";
      when "0110" => HEX0 <= "0000010";
      when "0111" => HEX0 <= "1111000";
      when "1000" => HEX0 <= "0000000";
      when "1001" => HEX0 <= "0010000";
      when others  => HEX0 <= "1111111";
  end case;
end process;

end Behavioral;
```