Digital Systems Design

ENGR 378

Report-Lab 8: VGA Controller and Pong Game

Submitted by:

Jessica Yip
Dennis Lin

December 12, 2018

Objective

The objective of this lab is to learn how to draw color patterns on a computer monitor by using the red, blue, and green signals on the FPGA using the VGA port. Using what we learned to draw patterns on a monitor, we then had to write VHDL code in order to implement a game of pong.

Problem Analysis

The objective of this lab is to familiarize ourselves with using the FPGA to draw colors on the monitor. Using this knowledge we need to implement a PONG game with working paddles and ball. The FPGA will also need to display the score of the game. After each point is won the ball will have to reset in the middle of the board for a new round. We will also colorize all the items that we draw in the screen to our own desire.

Hardware Design

This lab used three hardware components; the FPGA, the monitor, and the a VGA cable. The monitor will be used to display the to display the game. We will implement the Pong game in the FPGA where we can play it on the monitor.

VHDL Modeling

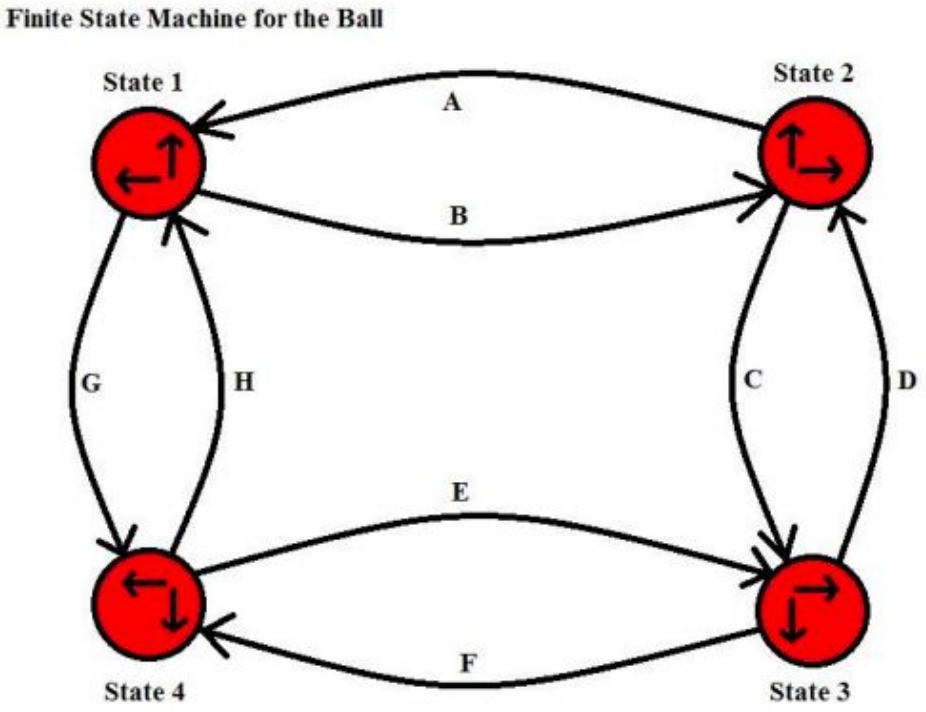

Fig 8: FSM for Pong Ball

State changes for the pong ball FSM are as follows:

A) The ball bounces off the right wall or the right paddle.
B) The ball bounces off the left wall or the left paddle.
C) The ball bounces off the top wall.
D) The ball bounces off the bottom wall.
E) The ball bounces off the left wall or the left paddle.
F) The ball bounces off the right wall or the right paddle.
G) The ball bounces off the top wall.
H) The ball bounces off the bottom wall.

Drawing the ball was the difficult aspect of this lab. The borders and paddles were something that we were able to model after the example lab to get working. The collisions for the ball was hard task to accomplish because we had to account for the position of the borders as well as the paddles' location and displaced location. The player that reaches 9 points is the winner, afterwards the score would reset back to 0.

Conclusion and Discussion

The objective of this lab was satisfied. We were able to produce the pong game that keeps track of score. The ball collisions with the paddles and the borders was accomplished. The game displays two paddles; one on each side of the background with the ball starting in the middle. We adjusted the speed of the paddles and ball so that the game play would not be bland and it would be somewhat exciting to play. The borders show the play area of where the ball is allowed to play without a player scoring. Once the ball is past the paddles of a player the player on the opposite end will score a point.

HDL Source Code:

```
----------------------------------------------------------------------------------
-- ENGR 378 San Francisco State University -- VGA Lab 8
-- This code was written for a computer monitor with a 1280 by 1024 resolution
(60 fps)
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.NUMERIC_STD.ALL;

entity VGAInterface is -- Define project interface
      Port ( CLOCK_50: 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 displaying the white dot to screen for demo
signal XDotPosition : STD_LOGIC_VECTOR (10 downto 0) := "01010000000";
signal YDotPosition : STD_LOGIC_VECTOR (10 downto 0) := "01000000000";
signal XDotPositionR : STD_LOGIC_VECTOR (10 downto 0) := "01010000000";
signal YDotPositionR : STD_LOGIC_VECTOR (10 downto 0) := "01000000000";
signal displayPosition : STD_LOGIC_VECTOR (10 downto 0) := "01000000000";

-- 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

-- Ball Position
signal XBallPosition : STD_LOGIC_VECTOR (10 downto 0) := "01010000000";
signal YBallPosition : STD_LOGIC_VECTOR (10 downto 0) := "01000000000";

-- Set Values
signal State: integer := 1; -- Initial Starting State is 1
```

```
signal XLeft: integer := 96; -- Left Wall Boundary
signal XRight: integer := 1184; -- Right Wall Boundary
signal YTop: integer := 80; -- Top Wall Boundary
signal YBot: integer := 944; -- Bottom Wall Boundary
signal radius: integer := 10; -- Radius of Ball
signal LeftScore: integer := 0; -- Counts Player Left
signal Rightscore: integer := 0; -- Counts Player Right
signal reset: integer := 5; -- Signal to Reset Game
signal BallSpeed: integer := 30; -- Controls Ball Speed
signal PaddleSpeed: integer := 20; -- Controls Paddle Speed
signal PaddleLength: integer := 70; -- Designates Paddle Length
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;
slowClock <= slowClockCounter(20);

-- slow clock signal
process (slowclock)
-- Move Ball Position and Scoring/Display to SSD
begin if slowClock'event and slowClock= '1' then
if Sw(0) = '1' then
XBallPosition <= "01010000000";
YBallPosition <= "01000000000";
YDotPosition <= "01000000000";
YDotPositionR <= "01000000000";
BallSpeed <= 0;
LeftScore <= 0;
RightScore <= 0;
else
BallSpeed <= 15; if (LeftScore >= 9 or RightScore >= 9) then
BallSpeed <= 0; XBallPosition <= "01010000000";
YBallPosition <= "01000000000";
end if;
if KEY(0) = '0' then -- Detect Button 0 Pressed
if YDotPositionR >= YBot - PaddleLength - 7 then
YDotPositionR <= YDotPositionR; else
YDotPositionR <= YDotPositionR + PaddleSpeed;-- Moves Right Paddle Down
end if;
elsif KEY(1) = '0' then -- Detect Button 2 Pressed
if YDotPositionR <= YTop + PaddleLength + 7 then
YDotPositionR <= YDotPositionR; else
YDotPositionR <= YDotPositionR - PaddleSpeed;-- Moves Right Paddle Up
end if; end if;
if KEY(2) = '0' then -- Detect Button 2 Pressed
```

```
if YDotPosition >= YBot - PaddleLength - 7 then
YDotPosition <= YDotPosition; else
YDotPosition <= YDotPosition + PaddleSpeed; -- Moves Left Paddle Down
end if;
elsif KEY(3) = '0' then -- Detect Button 3 Pressed
if YDotPosition <= YTop + PaddleLength + 7 then
YDotPosition <= YDotPosition; else
YDotPosition <= YDotPosition - PaddleSpeed; -- Moves Left Paddle Up
end if;
end if;
if State = 1 then -- State 1 has ball moving in +X/-Y direction
XBallPosition <= XBallPosition + BallSpeed;
YBallPosition <= YBallPosition - BallSpeed;
if XBallPosition >= XRight - radius then
LeftScore <= LeftScore + 1;
XBallPosition <= "01010000000";
YBallPosition <= "01000000000";

State <= 4;
elsif YBallPosition <= YTop + radius then
YBallPosition <= YBallPosition + radius;
State <= 4; elsif (XBallPosition >= 1110 - radius) and (XBallPosition <= 1131 -
radius) and (YBallPosition >= YDotPositionR - PaddleLength - radius) and
(YBallPosition <= YDotPositionR + PaddleLength + radius) then -- Changes ball
direction when colliding with Right Paddle
XBallPosition <= XBallPosition - radius;
State <= 2; end if;
elsif State = 2 then -- State 2 has ball moving in -X/-Y direction
XBallPosition <= XBallPosition - BallSpeed;
YBallPosition <= YBallPosition - BallSpeed;
if XBallPosition <= XLeft + radius then
RightScore <= RightScore + 1;
XBallPosition <= "01010000000";
YBallPosition <= "01000000000";
State <= 3;
elsif YBallPosition <= YTop + radius then
YBallPosition <= YBallPosition + radius;
State <= 3;
elsif (XBallPosition <= 171 + radius)
      and (XBallPosition >= 150 + radius)
      and (YBallPosition >= YDotPosition - PaddleLength - radius)
      and (YBallPosition <= YDotPosition + PaddleLength + radius) then
XBallPosition <= XBallPosition + radius;
State <= 1; end if;
elsif State = 3 then -- State 3 has ball moving in -X/+Y direction
XBallPosition <= XBallPosition - BallSpeed;
YBallPosition <= YBallPosition + BallSpeed;
if XBallPosition <= XLeft + radius then
```

```
RightScore <= RightScore + 1;
XBallPosition <= "01010000000";
YBallPosition <= "01000000000";
State <= 2;
elsif YBallPosition > YBot - radius then
YBallPosition <= YBallPosition - radius;
State <= 2;
elsif (XBallPosition <= 171 + radius) and (XBallPosition >= 150 + radius) and
(YBallPosition >= YDotPosition - PaddleLength - radius) and (YBallPosition <=
YDotPosition + PaddleLength + radius) then -- Changes ball direction when
colliding with Left Paddle
XBallPosition <= XBallPosition + radius;
State <= 4; end if;
elsif State = 4 then -- State 4 has ball moving in +X/+Y direction
XBallPosition <= XBallPosition + BallSpeed;
YBallPosition <= YBallPosition + BallSpeed;
if XBallPosition >= XRight - radius then
LeftScore <= LeftScore + 1;
XBallPosition <= "01010000000";
YBallPosition <= "01000000000";

State <= 1;
elsif YBallPosition >= YBot - radius then
YBallPosition <= YBallPosition - radius;

State <= 1;
elsif (XBallPosition >= 1110 - radius) and (XBallPosition <= 1131 - radius) and
(YBallPosition >= YDotPositionR - PaddleLength - radius) and (YBallPosition <=
YDotPositionR + PaddleLength + radius) then -- Changes ball direction when
colliding with Right Paddle
XBallPosition <= XBallPosition - radius;

State <= 3;
end if;
end if;
end if;
end if;
end process;

process (LeftScore, RightScore)
begin
If LeftScore = 0 then -- Display for when Left person makes a point
HEX6 <= "1000000"; elsif LeftScore = 1 then
HEX6 <= "1111001"; elsif LeftScore = 1 then
HEX6 <= "1111001"; elsif LeftScore = 2 then
HEX6 <= "0100100"; elsif LeftScore = 3 then
HEX6 <= "0110000"; elsif LeftScore = 4 then
HEX6 <= "0011001"; elsif LeftScore = 5 then
```

```
HEX6 <= "0010010"; elsif LeftScore = 6 then
HEX6 <= "0000010"; elsif LeftScore = 7 then
HEX6 <= "1111000"; elsif LeftScore = 8 then
HEX6 <= "0000000"; elsif LeftScore = 9 then
HEX6 <= "0011000";
end if;


if RightScore = 0 then-- Display for when Right person makes a point
HEX0 <= "1000000";
elsif RightScore = 1 then
HEX0 <= "1111001"; elsif RightScore = 2 then
HEX0 <= "0100100"; elsif RightScore = 3 then
HEX0 <= "0110000"; elsif RightScore = 4 then
HEX0 <= "0011001"; elsif RightScore = 5 then
HEX0 <= "0010010"; elsif RightScore = 6 then
HEX0 <= "0000010"; elsif RightScore = 7 then
HEX0 <= "1111000"; elsif RightScore = 8 then
HEX0 <= "0000000"; elsif RightScore = 9 then
HEX0 <= "0011000"; end if; end process;

-- Displays the value for the X coordinate or Y coordinate to the LEDS
depending on switch 1
LEDR(10 downto 0) <= YDotPosition when SW(1) = '1'
else XDotPosition;

-- 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;




-- 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
if SW(0) = '0' then

-- Vertical Border Colors and Background Color
if (XPixelPosition < XLeft) then -- Left Border Color
redValue <= "11111111";
blueValue <= "11111111";
greenValue <= "00000000";

elsif (XPixelPosition > XRight) then -- Right Border Color
redValue <= "11111111";
blueValue <= "11111111";
greenValue <= "00000000";
else

-- Background Color
redValue <= "00000000";
blueValue <= "00000000";
greenValue <= "00000000";
end if;


-- Horizontal Border Colors
-- Top border color
if (XPixelPosition > XLeft
      and XPixelPosition < XRight
      and YPixelPosition < YTop) then
redValue <= "00000000";
blueValue <= "00000000";
greenValue <= "11111111";

-- Bottom border color
elsif (XPixelPosition > XLeft
      and XPixelPosition < XRight
      and YPixelPosition > YBot) then
redValue <= "00000000";
blueValue <= "00000000";
greenValue <= "11111111";
end if;


-- Paddles
-- Left Paddle Position
if (XPixelPosition >= 150
      and XPixelPosition <= 171
      and YPixelPosition < (YDotPosition + PaddleLength)
      and YPixelPosition > (YDotPosition - PaddleLength)) then
redValue <= "11111111";
```

```
blueValue <= "11111111";
greenValue <= "11111111";

-- Right Paddle Position
elsif (XPixelPosition >= 1110
      and XPixelPosition <= 1131
      and YPixelPosition < (YDotPositionR + PaddleLength)
      and YPixelPosition > (YDotPositionR - PaddleLength)) then
redValue <= "11111111";
blueValue <= "11111111";
greenValue <= "11111111";
end if;


-- Ball Position Color and Sizing
if (XPixelPosition < (XBallPosition + radius)
      and XPixelPosition > (XBallPosition - radius)
      and YPixelPosition < (YBallPosition + radius)
      and YPixelPosition > (YBallPosition - radius)) then
redValue <= "11111111";
blueValue <= "00000000";
greenValue <= "00000000";
end if;
elsif Sw(0) = '1' then

-- Vertical Border Colors and Background Color
if (XPixelPosition < XLeft) then

-- Left Border Color
redValue <= "11111111";
blueValue <= "11111111";
greenValue <= "00000000";

elsif (XPixelPosition > XRight) then
redValue <= "11111111";
blueValue <= "11111111";
greenValue <= "00000000";

else

-- Background Color
redValue <= "00000000";
blueValue <= "00000000";
greenValue <= "00000000";
end if;

-- Horizontal Border Colors
-- Top border color
```

```
if (XPixelPosition > XLeft and XPixelPosition < XRight and YPixelPosition <
YTop) then
redValue <= "00000000";
blueValue <= "00000000";
greenValue <= "11111111";

-- Bottom border color
elsif (XPixelPosition > XLeft and XPixelPosition < XRight and YPixelPosition >
YBot) then
redValue <= "00000000";
blueValue <= "00000000";
greenValue <= "11111111";
end if;

-- Paddle
-- Left Paddle Position
if (XPixelPosition >= 150
      and XPixelPosition <= 171
      and YPixelPosition < (YDotPosition + PaddleLength)
      and YPixelPosition > (YDotPosition - PaddleLength)) then
redValue <= "11111111";
blueValue <= "11111111";
greenValue <= "11111111";

-- Right Paddle Position
elsif (XPixelPosition >= 1110
      and XPixelPosition <= 1131
      and YPixelPosition < (YDotPositionR + PaddleLength)
      and YPixelPosition > (YDotPositionR - PaddleLength)) then
redValue <= "11111111";
blueValue <= "11111111";
greenValue <= "11111111";
end if;

-- Ball Position Color and Sizing
if (XPixelPosition < (XBallPosition + radius)
      and XPixelPosition > (XBallPosition - radius)
      and YPixelPosition < (YBallPosition + radius)
      and YPixelPosition > (YBallPosition - radius)) then
redValue <= "11111111";
blueValue <= "00000000";
greenValue <= "00000000";
end if;
end if;
end if;

end process;
```

```
end Behavioral;
```