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)	F2022

Instructor:	Lev Kirischian

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

Submission Date:	December 2nd, 2022
Due Date:	December 2nd, 2022

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Iqbal	Hamza	500973673	032	H.I
Ayntabli	Krikor	500895492	032	K.A

*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

Abstract

The design and execution of a straightforward video game of ping-pong on the Xilinx ISE is described in this project. It was intended for the game to be logically implemented using VHDL and then run on an FPGA device to provide outputs on the VGA monitor as well as functional waveforms produced by Xilinx ISE and VGA.

For our design, the ports red, blue, green, V Sync, and H Sync were used to distribute the VGA outputs. With a paddle that can move vertically up and down and hit the dynamic ball, each player was represented. Seven segment logic, which updates when players miss and resets after a certain score, was used to display the score.

Introduction

This project describes the creation and implementation of a simple ping-pong video game on the Xilinx ISE. In order to give outputs on the VGA monitor and functional waveforms generated by Xilinx ISE and VGA, the game was designed to be logically built using VHDL and then executed on an FPGA device. The VGA outputs for our design were dispersed across the ports red, blue, green, V Sync, and H Sync.

The game logic required to be built using VHDL coding once the VGA module had been programmed in accordance with the requirements using the horizontal and vertical parameters listed in the project/lab manual.

Specification

The requirements for the horizontal and vertical parameters of the VGA module were predetermined for this project and are described below:

Table I: VGA Horizontal Parameters

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

Table II: VGA Vertical Parameters

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

For us to construct the VGA outputs with specific horizontal and vertical specifications, the table above serves as a guidance. The system's specs are not entirely covered by these parameters, though. It is necessary to fully comprehend both the static and dynamic components of the game in order to examine the complete system specs. The parts of the output that are displayed on the VGA that are static even after the game has begun to play include the white borders, center line, and green backdrop. These are the static parts of our design.

The following is a rough specification schematic of the project:

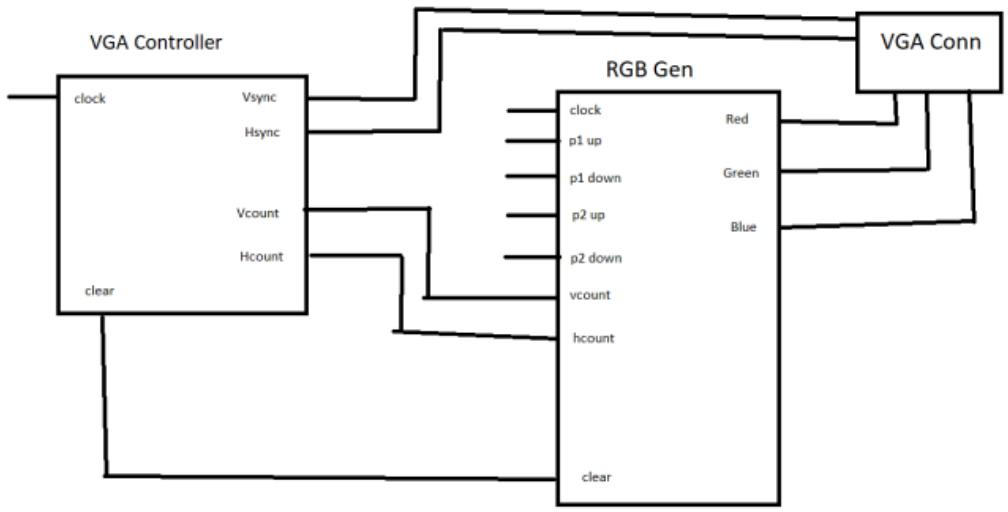

Figure 1: VGA Specification Schematic Diagram

The generated VGA outputs' dynamic elements are those parts that can reposition themselves once the game has begun. These elements include the paddles for each player that can move vertically up and down, the seven segment score board and the constantly moving ball.

Device Design

Symbols

The game's essential logic is contained in this VHDL block, which also receives input signals from the player and outputs VGA signals. It is the project's main building component.

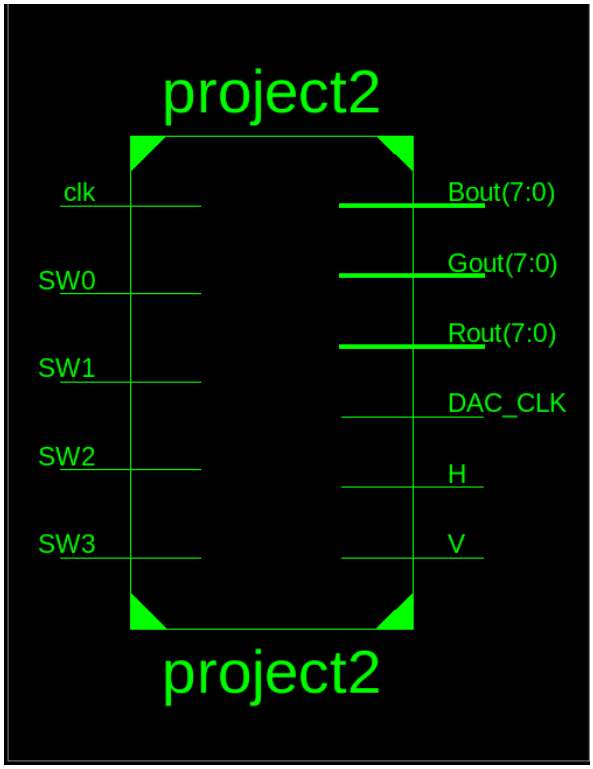

Figure 2: Project Main Game Block Symbol

The game's essential logic is contained in this VHDL block, which also receives input signals from the player and outputs VGA signals. It is the project's main building component.

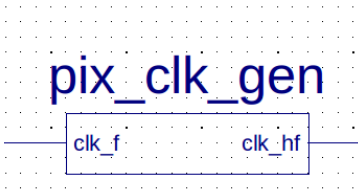

Figure 3: Pixel Clock generator Symbol

In order for the game logic to function, this block is vital since it is responsible for controlling the clock, which will be used for refreshing the pixels to run the game.

The following is the complete schematic diagram of the project:

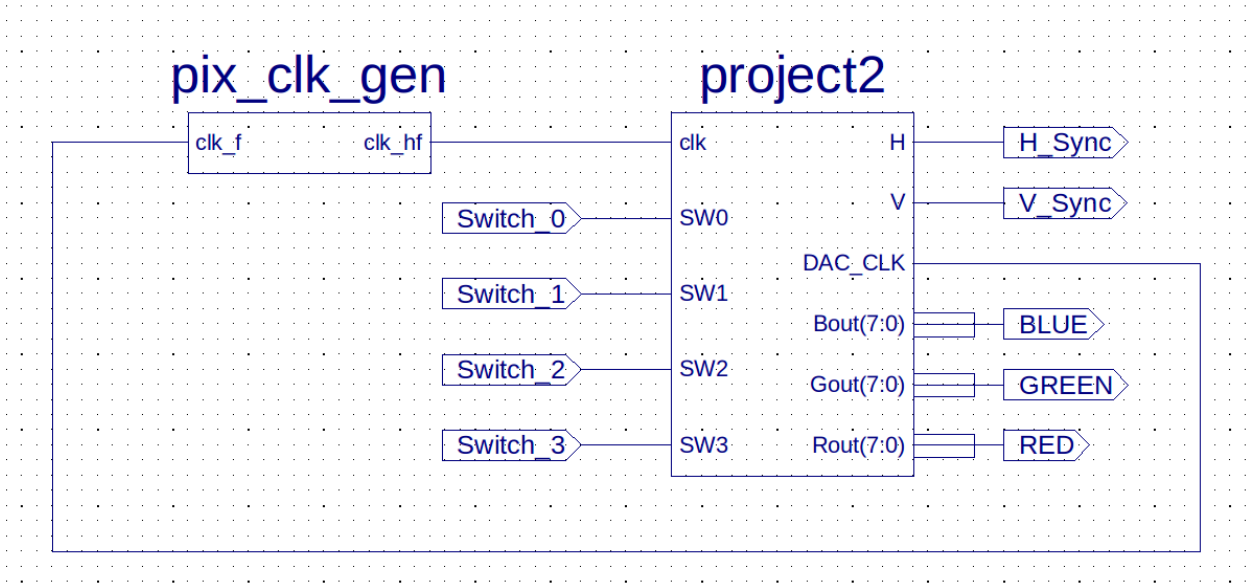

Figure 4: Schematic Diagram of the Diagram

State Diagram

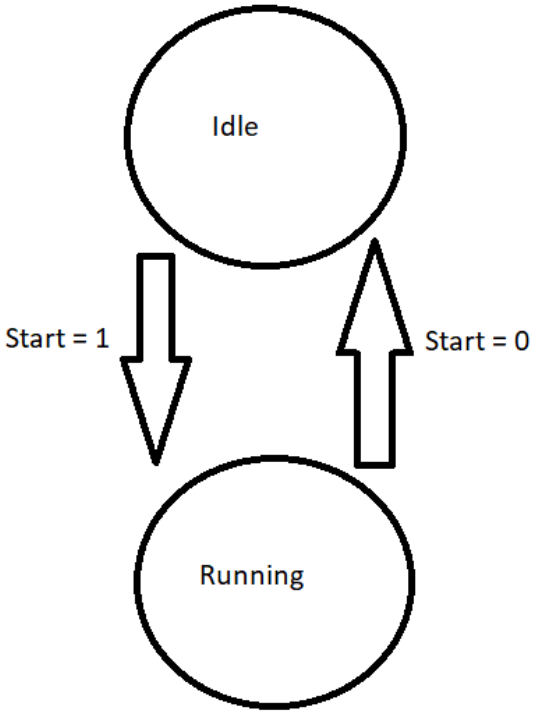

Figure 5: State Diagram

Process Diagram

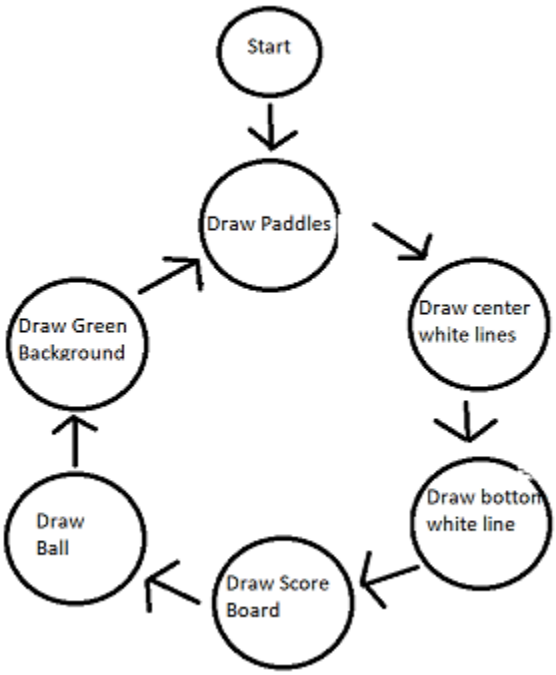

Figure 6: Process Diagram

Results

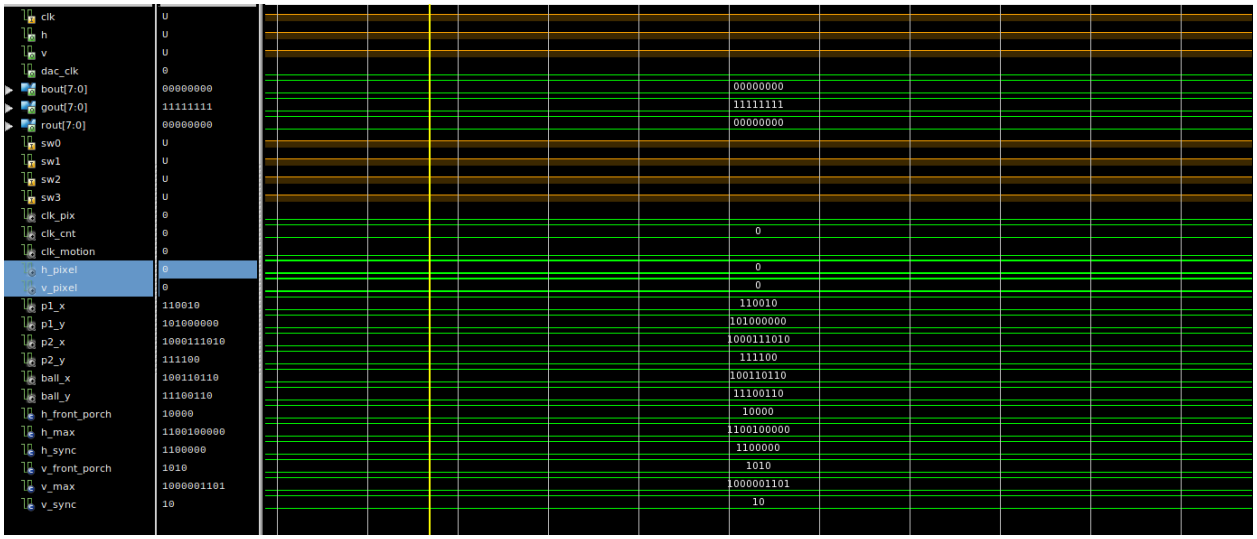

Figure 7: Waveform pic of Idle State

As seen in figure 5, this waveform showcases the state of the game when it is in an idle state, that is, the game is not running. There are currently no dynamic or static elements. This is as indicated as all the signals are low.

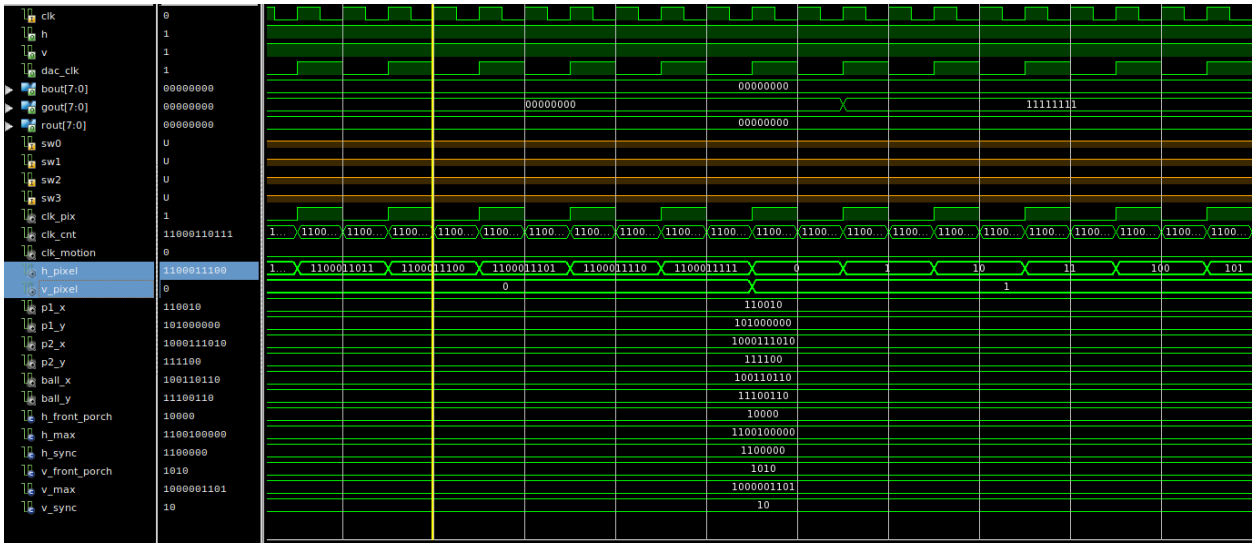

Figure 8: Waveform pic of Running State

As seen in figure 6, this waveform showcases the state of the game when it is running, indicated by the clock running. The paddles can be moved using the switch signals. Pixels on the monitor are refreshed 1 pixel at a time. For every pixel on a column, it refreshes each horizontal pixel on its respective row, then moves down the column to the next row of pixels to refresh. This is shown with the signals V_sync & H_sync.

Conclusion

Overall, The ping-pong game project was implemented and completed, on the Xilinx ISE. It was intended for the game to be logically implemented using VHDL and then run on an FPGA device to provide outputs on the VGA monitor as well as functional waveforms produced by Xilinx ISE and VGA. The ping pong game's functional waveforms were effectively used to show its operation.

References

1) Khare, S. (2014) -*Design of Game "Pong" Using VHDL*. Retrieved November 28 from Design of Game "Pong" Using VHDL – IJERT

2) Kirischian, L. (n.d.). *Project #2 - Simple Video Game Processor for VGA*. Retrieved November 22, 2022, from https://www.ee.ryerson.ca/~lkirisch/ele758/labs/SimpleVideoGame%5b11-11-11%5d.pdf

3) *Pong Game*. fpga4fun.com. (n.d.). Retrieved November 28, 2022, from https://www.fpga4fun.com/PongGame.html

Appendix

```
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 project2 is
    Port ( clk : in  STD_LOGIC;
           H : out  STD_LOGIC;
           V : out  STD_LOGIC;
           DAC_CLK : out  STD_LOGIC;
           Bout : out  STD_LOGIC_VECTOR (7 downto 0);
           Gout : out  STD_LOGIC_VECTOR (7 downto 0);
           Rout : out  STD_LOGIC_VECTOR (7 downto 0);
           SW0 : in  STD_LOGIC;
           SW1 : in  STD_LOGIC;
           SW2 : in  STD_LOGIC;
           SW3 : in  STD_LOGIC);
end project2;

architecture Behavioral of project2 is

           ----------------------
           -- Consants
           ----------------------
           constant h_front_porch          : integer := 16;
           constant h_max                                  : integer := 800;
           constant h_sync                         : integer := 96;
           constant v_front_porch          : integer := 10;
           constant v_max                          : integer := 525;
           constant v_sync                         : integer := 2;
           ------------------------------
           -- Clock Generation Signals
           ------------------------------
           signal clk_pix          : STD_LOGIC := '0';
           signal clk_cnt          : integer := 0;
```

```
	signal clk_motion : STD_LOGIC := '0';
	---------------------------
	-- Pixel display signals
	---------------------------
	signal pix_enable : STD_LOGIC;
	signal h_pixel		: integer := 0;
	signal v_pixel		: integer := 0;
	signal p1_x			: integer := 50;
	signal p1_y			: integer := 320;
	signal p2_x			: integer := 570;
	signal p2_y			: integer := 60;
	signal ball_x		: integer := 310;
	signal ball_y		: integer := 230; -- center = (320, 240)
	signal h_dir		: STD_LOGIC; -- 0 = right, 1 = left
	signal v_dir		: STD_LOGIC; -- 0 = down,  1  = up


begin

	-----------------------------
	-- clock Generation
	-----------------------------
	process (clk)
	begin
		if (clk'Event and clk = '1') then
			clk_pix <= NOT clk_pix;
			if (clk_cnt  = 100000) then
				clk_motion <= NOT clk_motion;
				clk_cnt <= 0;
			else
				clk_cnt <= clk_cnt + 1;
			end if;
		end if;
	end process;

	-------------------------
	-- Pixel Configuration
	-------------------------
	process (clk_pix, SW0)
	begin

		if (clk_pix'Event and clk_pix = '1') then
			-----------------------------------------------
			-- Counter for pixel location calculation
```

```
                    ----------------------------------------
                    if (h_pixel < h_max - 1) then
                            h_pixel <= h_pixel + 1;
                    else
                            h_pixel <= 0;
                            if (v_pixel < v_max - 1) then
                                    v_pixel <= v_pixel + 1;
                            else
                                    v_pixel <= 0;
                            end if;
                    end if;
                    ----------------------------------------
                    -- Sync configuration
                    ----------------------------------------
                    -- Horizontal Sync
                    if ((h_pixel < 639 + h_front_porch) OR (h_pixel >= 639 + h_front_porch +
h_sync)) then
                            H <= '1';
                    else
                            H <= '0';
                    end if;
                    -- Vertical Sync
                    if ((v_pixel < 479 + v_front_porch) OR (v_pixel >= 479 + v_front_porch +
v_sync)) then
                            V <= '1';
                    else
                            V <= '0';
                    end if;
                    ----------------------------------------
                    -- Determine pixel avilability
                    ----------------------------------------
                    if ((h_pixel < 640) AND (v_pixel < 480)) then
                            pix_enable <= '1';
                    else
                            pix_enable <= '0';
                    end if;
            end if;

    end process;
    ------------------
    -- Motion Update
    ------------------
    process (clk_motion, SW0, SW1, SW2, SW3, p1_y, p2_y)
    begin
```

```
if (clk_motion'Event and clk_motion = '1') then
        ------------------------------
        -- Update Player 1 location
        ------------------------------
        if (SW0 = '0' and SW1 = '0') then
                if (p1_y < 320) then
                        p1_y <= p1_y + 1;
                else
                        p1_y <= 320;
                end if;
        else
        if (SW0 = '1' and SW1 = '1') then
                if (p1_y > 40) then
                        p1_y <= p1_y - 1;
                else
                        p1_y <= 40;
                end if;
        end if;
        end if;
        ------------------------------
        -- Update Player 2 location
        ------------------------------
        if (SW2 = '0' and SW3 = '0') then
                if (p2_y < 320) then
                        p2_y <= p2_y + 1;
                else
                        p2_y <= 320;
                end if;
        else
        if (SW2 = '1' and SW3 = '1') then
                if (p2_y > 40) then
                        p2_y <= p2_y - 1;
                else
                        p2_y <= 40;
                end if;
        end if;
        end if;
        ------------------------------
        -- Update direction
        ------------------------------
        if ( ball_x >= p1_x and ball_x < p1_x + 10) then
          if ( ((ball_y >= p1_y) or (ball_y + 10 >= p1_y)) and ((ball_y < p1_y +
120) or (ball_y + 10 < p1_y + 120)) ) then
                        h_dir <= '0';  -- Change direction when hit the player
```

```
                        end if;
                    elsif ( ball_x = 40 ) then
                        if ( (ball_y >= 40 and ball_y < 160) or (ball_y + 10 >= 320 and
ball_y + 10 < 440) ) then
                            h_dir <= '0';  -- Change direction when hit left boundry
                        end if;
                    elsif (ball_x + 10 > p2_x and ball_x + 10 <= p2_x + 10) then
                        if ( ((ball_y >= p2_y) or (ball_y + 10 >= p2_y)) and ((ball_y < p2_y
+ 120) or (ball_y + 10 < p2_y + 120)) ) then
                            h_dir <= '1';
                        end if;
                    elsif (ball_x + 10 = 600) then
                        if ( (ball_y >= 40 and ball_y < 160) or (ball_y + 10 >= 320 and
ball_y + 10 < 440) ) then
                            h_dir <= '1';  -- Change direction when hit right boundry
                        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;
```

```
	---------------------------------50 300 (420
	-- Pixel Color Set
	---------------------------------**120
	process (pix_enable)
	begin
		if (pix_enable = '0') then
			Rout <= "00000000";
			Gout <= "00000000";
			Bout <= "00000000";
		else
			if (h_pixel >= 20 and h_pixel < 640 - 20 and v_pixel >=20 and v_pixel <
460) then
			  if ( v_pixel < 40 or v_pixel >= 440) then
					Rout <= (others => '1');  -- Display white for top & bottom
border
					Gout <= (others => '1');
					Bout <= (others => '1');
				elsif (((h_pixel < 40) OR (h_pixel >= 640 - 40)) and (v_pixel < 160
or v_pixel >= 320)) then
					Rout <= (others => '1');  -- Display white for left & right
border
					Gout <= (others => '1');
					Bout <= (others => '1');
				elsif ((h_pixel >= ball_x and h_pixel < ball_x + 10) and (v_pixel >=
ball_y and v_pixel < ball_y + 10) )then
					Rout <= "11111111";   -- Color Ball inside play field (gate +
border)
					Gout <= "11111111";
					Bout <= "00000000";
				elsif ((h_pixel >= p1_x and h_pixel < p1_x + 10) and (v_pixel >=
p1_y and v_pixel < p1_y + 120) )then
					Rout <= "00000000";   -- Color Player 1
					Gout <= "00000000";
					Bout <= "11111111";
				elsif ((h_pixel >= p2_x and h_pixel < p2_x + 10) and (v_pixel >=
p2_y and v_pixel < p2_y + 120) )then
					Rout <= "11111111";   -- Color Player 2
					Gout <= "00000000";
					Bout <= "11111111";
				elsif (v_pixel >= 40 and v_pixel < 440 and h_pixel = 320 and
v_pixel mod 16 <= 10) then
				  Rout <= (others => '0');   -- Color center line
					Gout <= (others => '0');
					Bout <= (others => '0');
```

```
                                else
                                        Rout <= (others => '0');
                                        Gout <= (others => '1');
                                        Bout <= (others => '0');
                                end if;
                        elsif ((h_pixel >= ball_x and h_pixel < ball_x + 10) and (v_pixel >= ball_y
and v_pixel < ball_y + 10) )then
                                Rout <= "11111111";   -- Color Ball
                                Gout <= "00000000";
                                Bout <= "00000000";
                        else
                                Rout <= (others => '0');
                                Gout <= (others => '1');
                                Bout <= (others => '0');
                        end if;
                end if;
        end process;
        DAC_CLK <= clk_pix;


end Behavioral;
```