RYERSON UNIVERSITY

Faculty of Engineering, Architecture and Science

Department of Electrical and Computer Engineering

Course Number	COE 758
Course Title	Digital Systems Engineering
Semester/Year	F2023

Instructor	Dr. Vadim Guerkov

Project No.	2

Project Title	Video Game

Submission Date	November 30th, 2023
Due Date	December 4th, 2023

Student Name	Abdulrehman Khan	Kai Xu
Student ID	500968727	501041980
Signature*	A.K.	K.X.

**By signing above you attest that you have contributed to this written lab report and confirm that all work you have swung the lab contributed to this lab report is your own work.*

Table of Contents

Abstract

This project details the development and execution of a streamlined ping-pong video game using Xilinx ISE. The primary objective was to implement the game's logic through VHDL and subsequently run it on an FPGA device, generating visual outputs on a VGA monitor and functional waveforms through Xilinx ISE and VGA.

In our design, we utilized the red, blue, green, V Sync, and H Sync ports to efficiently distribute VGA outputs. Each player was represented by a paddle capable of vertical movement, tasked with hitting the dynamic ball. The scoring system employed seven-segment logic, dynamically updating when players miss and resetting after reaching a predefined score threshold.

Introduction

This project outlines the conception and execution of a straightforward ping-pong video game using the Xilinx ISE platform. To yield visual outputs on the VGA monitor and functional waveforms generated by both Xilinx ISE and VGA, the game was strategically designed to be logically constructed through VHDL and subsequently executed on an FPGA device. The distribution of VGA outputs in our design was facilitated through the utilization of ports such as red, blue, green, V Sync, and H Sync.

The VHDL coding process was initiated once the VGA module had been programmed in alignment with the specified requirements, incorporating horizontal and vertical parameters outlined in the project/lab manual. This sequential approach ensured the systematic development of the game logic.

Specification

The predetermined requirements for the horizontal and vertical parameters of the VGA module are crucial elements in this project, as elucidated 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

To craft VGA outputs with specific horizontal and vertical specifications, the provided table serves as a guiding reference. However, it is essential to note that these parameters do not comprehensively cover the entirety of the system's specifications. A thorough understanding of both the static and dynamic components of the game is imperative for a comprehensive examination of the complete system specifications.

Certain elements of the VGA output retain their static nature even after the commencement of gameplay. These include the white borders, center line, and green backdrop, constituting the static components of our design.

Outlined below is a preliminary specification schematic for the project:

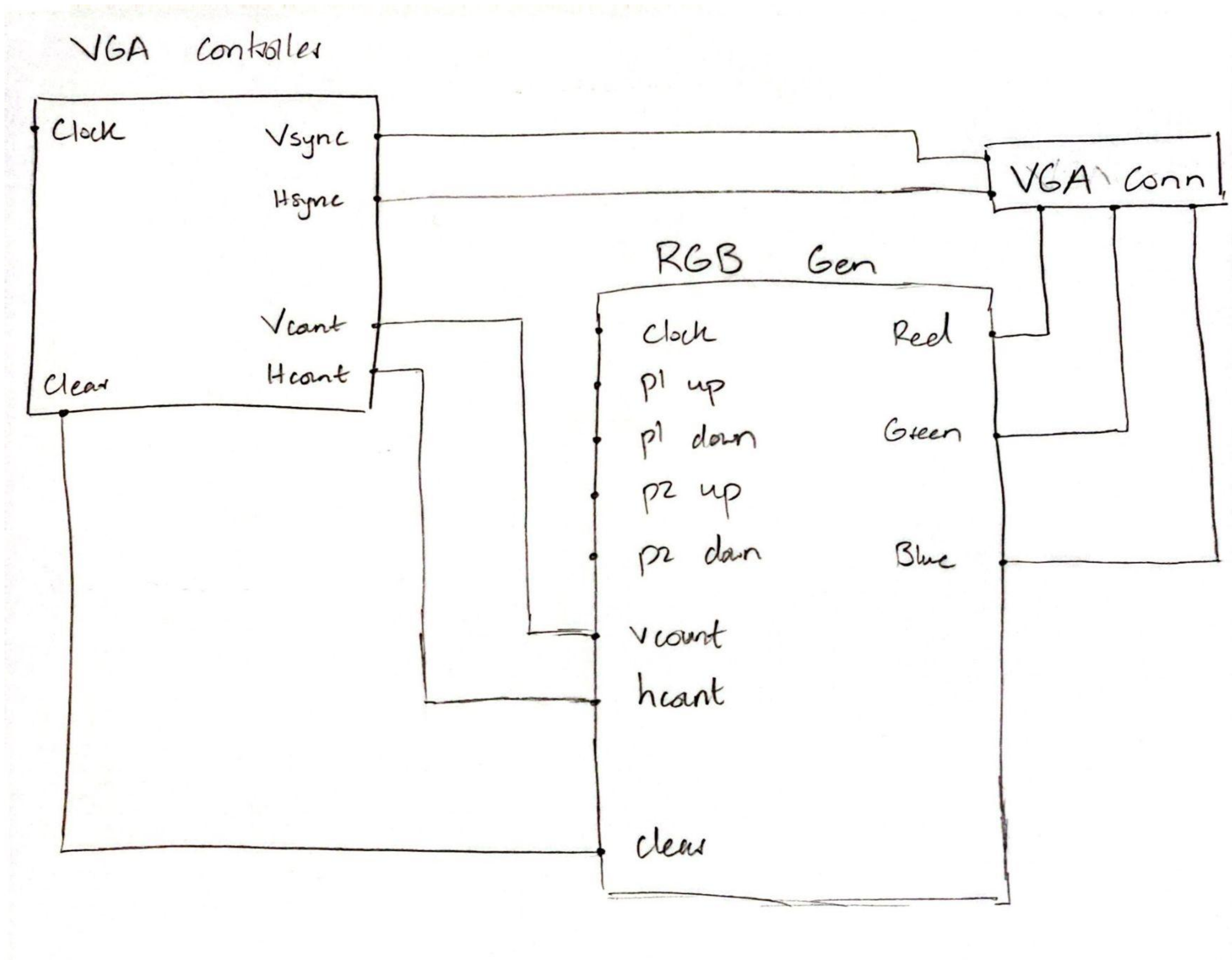

Figure 1: VGA Specification Schematic Diagram

The dynamic components of the generated VGA outputs encompass elements capable of repositioning themselves once the game is in progress. These dynamic elements comprise the vertically movable paddles for each player, the seven-segment scoreboard, and the continuously mobile ball. These elements collectively contribute to the evolving and interactive nature of the visual output as the game unfolds.

Device Design

Symbols

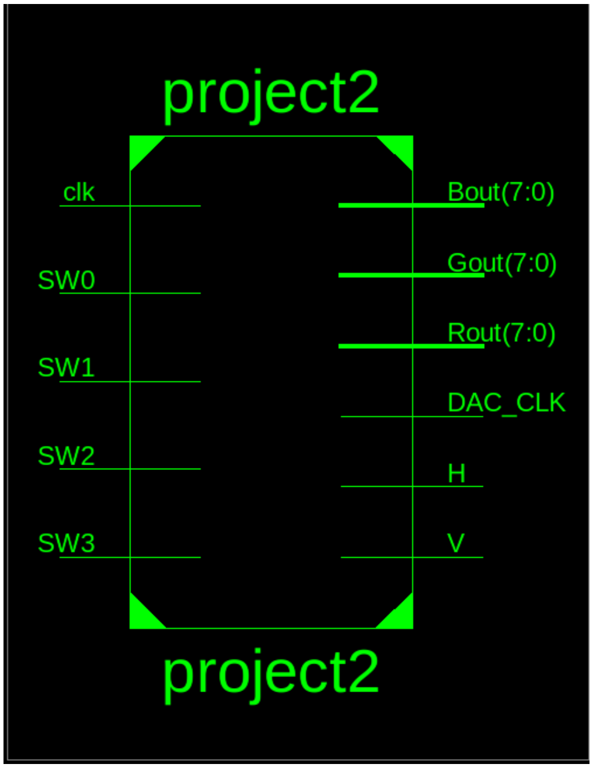

Figure 2: Project Main Game Block Symbol

This VHDL block serves as the core repository of the game's fundamental logic. Simultaneously, it interacts with input signals from the player, orchestrating the generation of VGA signals. As the primary architectural cornerstone of the project, this block stands as the central building component, defining the intricacies and operations of the entire system.

Figure 3: Pixel Clock generator Symbol

The significance of this block lies in its pivotal role in facilitating the game logic to function seamlessly. It holds responsibility for controlling the clock, a critical function in refreshing pixels to sustain the smooth operation of the game. The clock management provided by this block is essential for the timely and synchronized execution of processes, ensuring the effective and continuous running of the game.

Presented below is the comprehensive schematic diagram of the entire project:

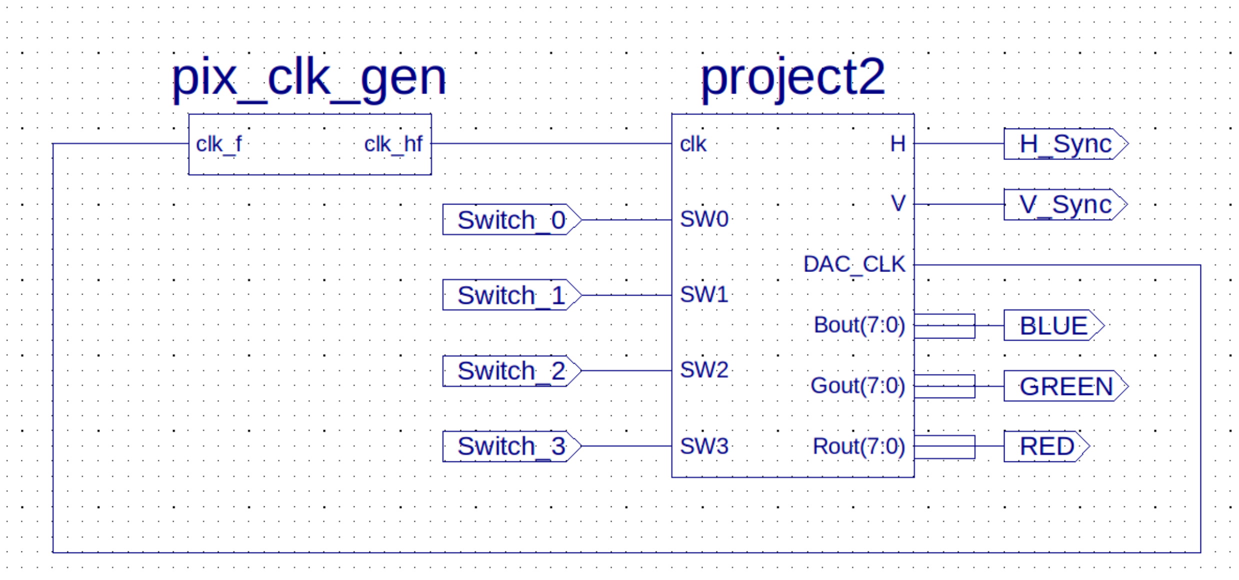

Figure 4: Schematic Diagram of the Diagram

State Diagram

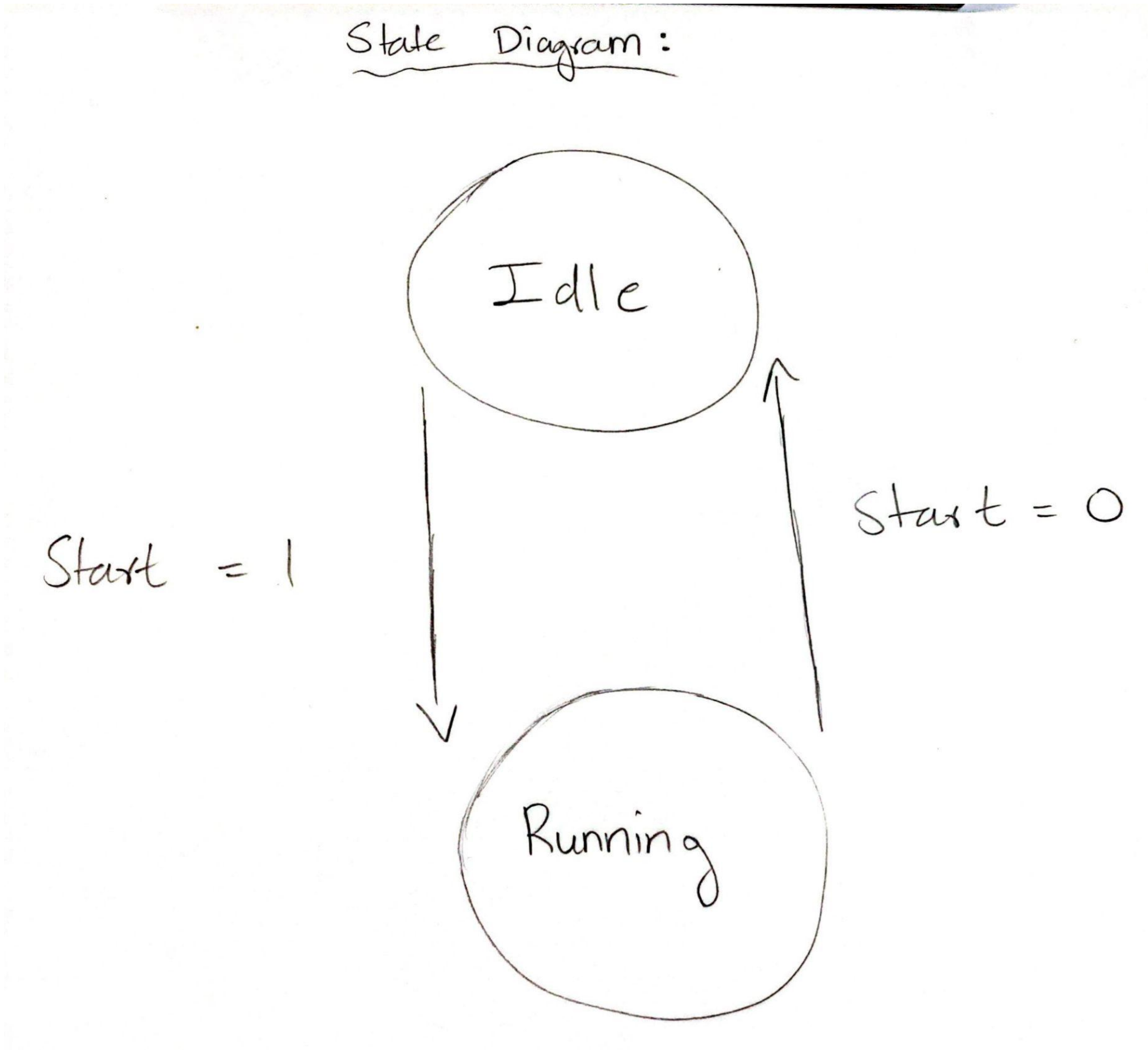

Figure 5: State Diagram

Process Diagram

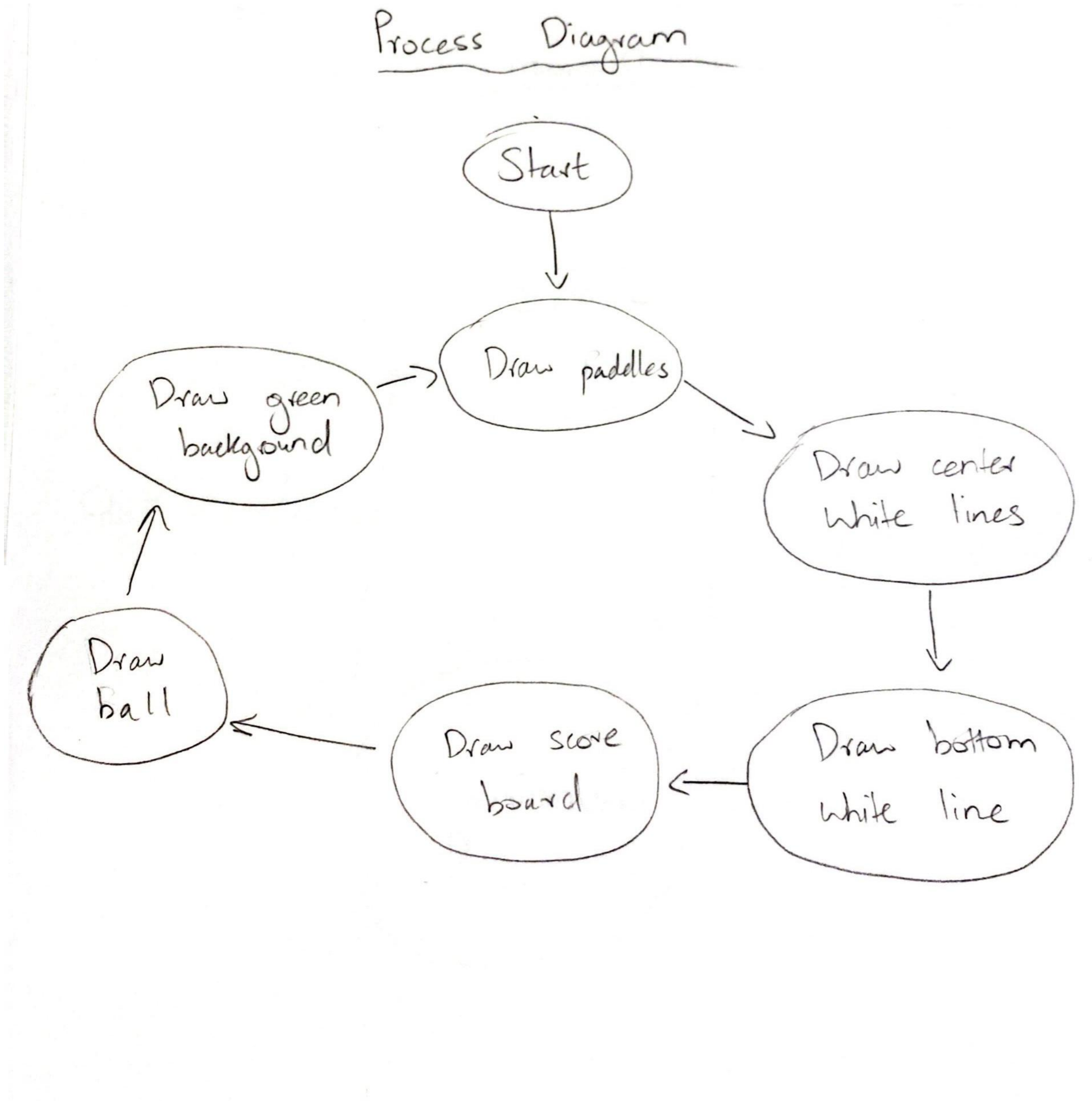

Figure 6: Process Diagram

Results

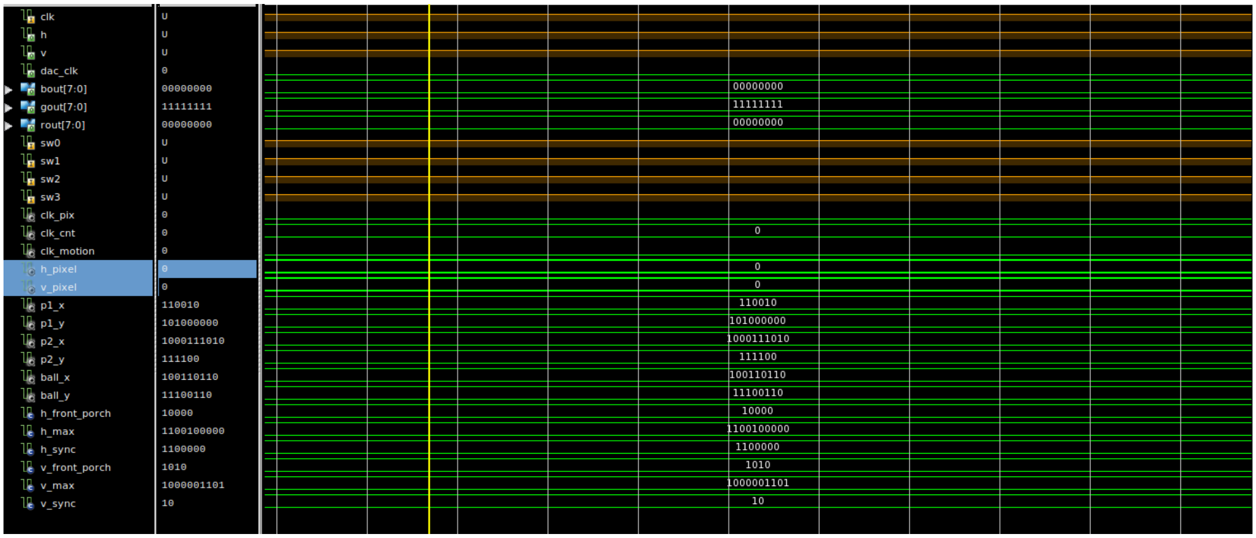

Figure 7: Waveform pic of Idle State

As depicted in Figure 5, the waveform illustrates the game's state during an idle phase, signifying that the game is not currently in progress. At this point, neither dynamic nor static elements are active, reflecting a state where all signals are low.

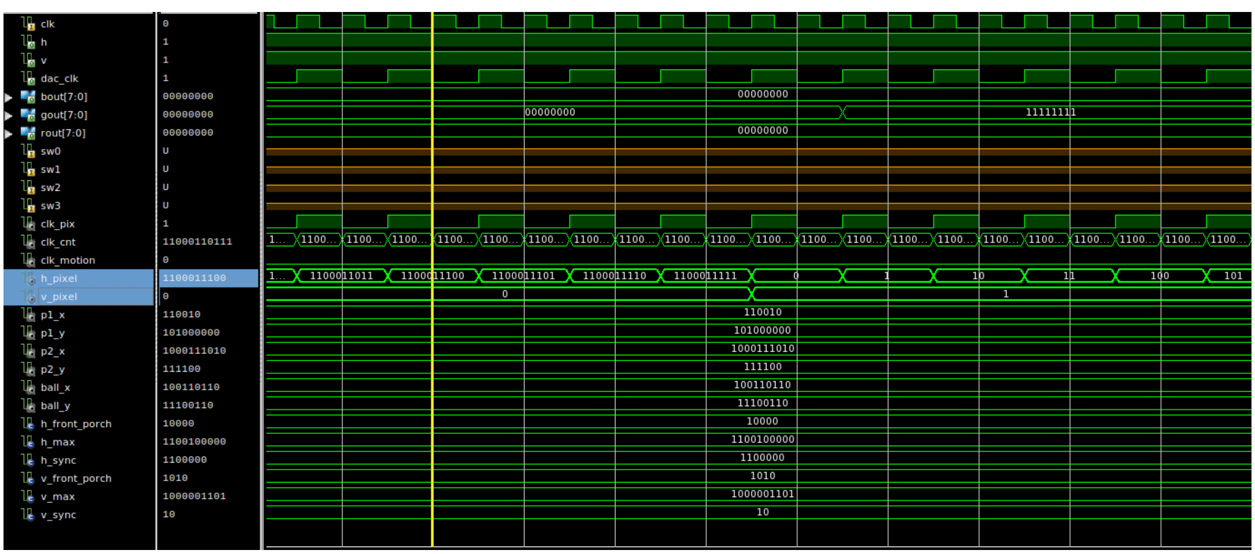

Figure 8: Waveform pic of Running State

As illustrated in Figure 6, this waveform captures the active state of the game, denoted by the running clock. During this phase, the paddles are responsive to switch signals, allowing for vertical movement. Notably, the pixel refresh on the monitor occurs incrementally, updating one pixel at a time. The synchronization of this process is evident through the signals V_sync and H_sync, orchestrating the systematic refresh of each horizontal pixel in its corresponding row before progressing to the next row and column.

Conclusion

In conclusion, the ping-pong game project was successfully implemented and finalized using Xilinx ISE. The strategic approach involved logical implementation through VHDL, followed by execution on an FPGA device. This setup allowed for the generation of outputs on the VGA monitor, accompanied by functional waveforms produced by both Xilinx ISE and VGA. The utilization of these functional waveforms effectively demonstrated the operational dynamics of the ping-pong game.

References

1. CS/EE 3710 – Computer Design Lab. (2010, September 30). Lab 3 – VGA. Retrieved from https://my.eng.utah.edu/~cs3710/labs/VGA.pdf

2. Fanelli, R., & Hartino, D. (n.d.). ECE 4760: Final Projects - Homemade VGA Adapter: An inexpensive solution, pushing the envelope on MCU clock cycle optimization. Cornell University ECE4760 Course Website. Retrieved from https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/raf225_dah322/raf225_dah322/

3. Tech Tangents. (n.d.). How VGA works [Video]. YouTube. https://www.youtube.com/watch?v=5exFKr-JJtg

4. Toronto Metropolitan University. 2023. Course Content: PROJECT 2 REPORT. In Digital Systems Engineering, COE 758. Toronto Metropolitan University's Learning Management System.

5. Toronto Metropolitan University. 2023. Course Content: SimpleVideoGame[11-11-11]. In Digital Systems Engineering, COE 758. Toronto Metropolitan University's Learning Management System.

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;
```