Toronto Metropolitan University

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

Course Title:	Digital Systems
Course Number:	COE758
Semester/Year (e.g. F2017)	F2024

Instructor	Lev Kirischian

Assignment/Lab Number:	Project 2
Assignment/Lab Title:	Video Game Specification

Submission Date:	Nov. 20, 2024
Due Date:	Nov. 20, 2024

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Chedde	Rohan	501123581	7	Rohan
Wort	Austin	501111124	7	AJW

Abstract

The objective of this project is to design and emulate the functionality of a VGA controller using an FPGA to create a simple video game processor (SVGP). This involves generating real-time signals to display a static game field with dynamic elements, such as a moving ball and player-controlled paddles, on a VGA monitor. The system adheres to VGA timing standards, including precise synchronization signals and pixel clock timing, to ensure compatibility with the display. Interactive gameplay is achieved through paddle control via input switches, with the ball dynamically responding to collisions and edge cases, such as goals, where it resets to the center. The project provides practical experience in VHDL coding for real-time applications, implemented in the Xilinx ISE CAD environment and deployed on the Xilinx Spartan-3E FPGA. By integrating video output, signal processing, and interactive logic, this project reinforces key concepts in digital system design and hardware interfacing.

Introduction

VGA technology is an analog standard connector used for computer video input, defining both the hardware interface and analog signaling for transferring graphical data to the display. VGA utilizes a 15-pin connector, with pins for crucial signals such as horizontal and vertical sync, RGB, and ground; all signals emulated in this project. These signals collectively enable precise timing and data transmission, ensuring the accurate rendering of graphical information on the display.

VGA uses analog signals for video output and relies on two synchronization signals, horizontal and vertical sync, to properly position and display graphical data. The system "prints" pixels row-by-row, coordinated by the pixel clock, with the sync signals ensuring each row is completed before moving to the next. To allow the hardware time to stabilize and prepare for synchronization pulses, blanking periods are introduced, padding the active display area. These active and blanking intervals are precisely defined in terms of clock cycles for this project, as shown in Table 1 and Table 2 (Horizontal and Vertical parameter specifications).

VGA, like most graphical display systems, uses RGB data to represent colors on the monitor. The RGB values are transmitted from the VGA controller as binary data (typically 8 bits per channel in modern implementations, including this project) and converted into analog voltage levels by digital-to-analog converters (DACs). These analog signals, ranging from 0 to ~0.7V, drive the intensity of red, green, and blue channels, allowing the display to render millions of colors with smooth transitions and accurate timing. By integrating these principles, this project demonstrates the functionality and practical implementation of a VGA controller.

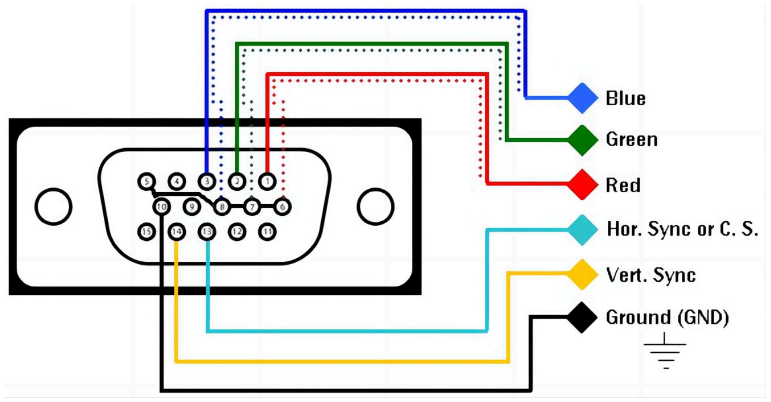

Figure 1. VGA Connector Pinout

System Specifications

The SVGP and VGA controller implemented in Xilinx ISE CAD have the following specifications:

INPUTS	DESCRIPTION
Pixel Clock / DAC_CLK	25 MHz clock generated from the 50 MHz FPGA core clock
RGB (Out)	8 bit RGB outputs to represent pixel color data
HSync / Vsync	Synchronization signals for electron beam to 'print' pixels row-by-row

Figure 2. System Specifications

The system also has three primary requirements that must be satisfied:

1. Display a static image of the background and game borders
2. Display two dynamically moving paddles that move vertically in front of the goals, triggered by the switches
3. Display a dynamically moving 'ball', that interacts with the borders and paddles by bouncing off of them at a 90 degree angle. Finally, the ball must be reset to the center of the screen upon entering the goal (scoring).

Device Description / Design

The SVGP Xilinx project successfully compiles and runs on the Xilinx ISE CAD platform, displaying the Pong video game on the VGA monitor.The project consists of mainly two components: a VHDL file implementing the functionality and processes of the VGA controller, and a constraints file provided in the project manual. Additionally, there are Icon and ILA cores implemented from Xilinx for waveform simulation purposes, however they are not included in the code appendix for simplicity.

The VHDL file implements the functionality of a VGA controller by first generating a 25 MHz pixel clock from the 50 MHz core clock of the Spartan-3E FPGA. The pixel clock is the rate at which pixel data is transmitted to the display. The status of the display is controlled by crucial signals such as 'videoON' and 'RST' that determine whether the electron beam is in the active display region or in the back or front porch. To synchronize the transition of the electron beam amongst rows and columns, there are vertical and horizontal sync pulses that are high when the display is active, and go low when transitioning between the front porch and back porch. The RGB values displayed on the monitor are dependent on the position of the electron beam within the display. For example, if the beam is near the left edge of the monitor, it will display a white pixel as part of the border. These cases are handled by numerous if statements based on the current position of the beam. This process makes up essentially the entirety of the VHDL file, emulating the behavior and functionality of a VGA controller to create a SVGP.

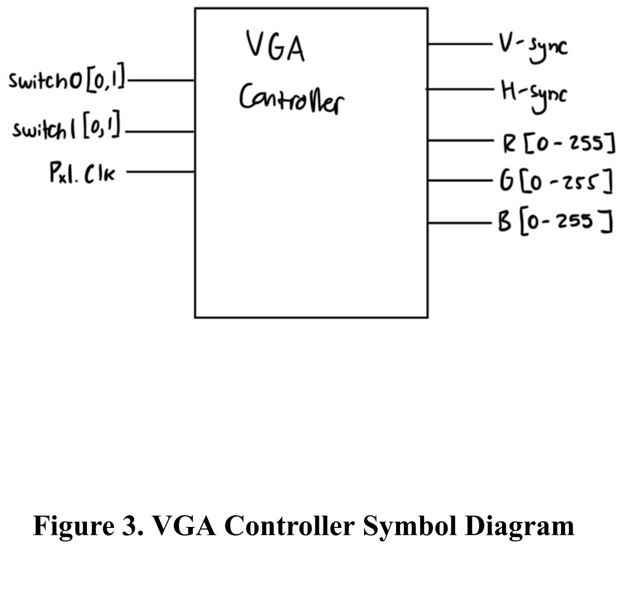

Figure 3. VGA Controller Symbol Diagram

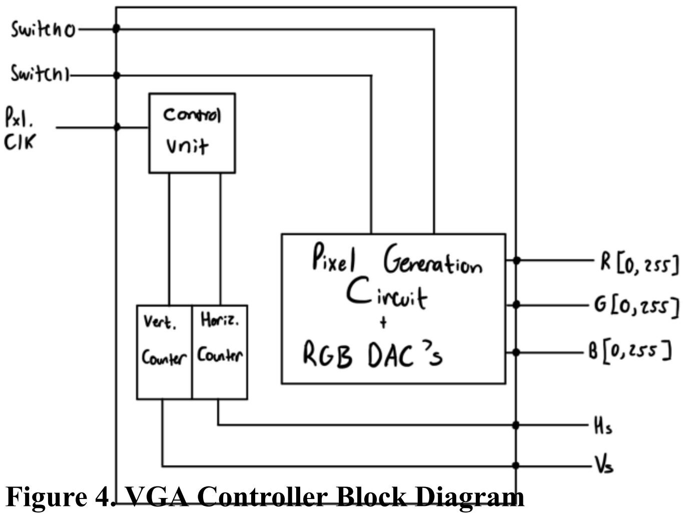

Figure 4. VGA Controller Block Diagram

VGA Specification

All horizontal time periods are specified in multiples of the VGA pixel clock, which is 25 MHz with a 60 Hz refresh rate display. All vertical (frame) time periods are specified in multiples of VGA lines.

Parameter	**Clock Cycles**
Complete line	800
Front Porch	16
Sync Pulse	96
Back Porch	48
Active Image Area	640

Figure 5. Horizontal Frame Specificifications

Parameter	**Lines**
Complete Frame	525
Front Porch	10
Sync Pulse	2
Back Porch	33
Active Image Area	480

Figure 6. Vertical Frame Specifications

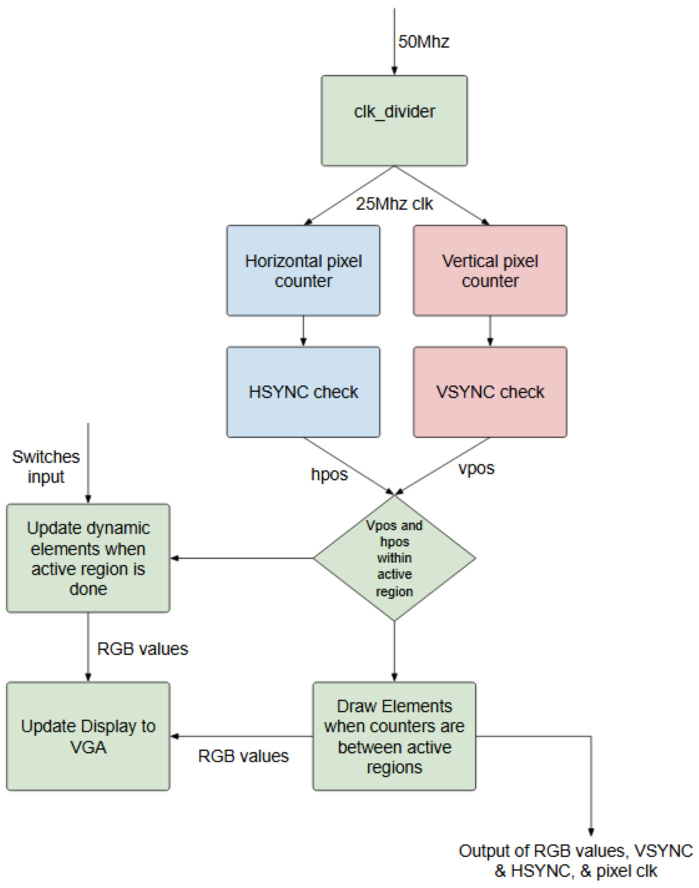
50Mhz
clk_divider
25Mhz clk
Horizontal pixel counter
Vertical pixel counter
HSYNC check
VSYNC check
hpos
vpos
Switches input
Update dynamic elements when active region is done
Vpos and hpos within active region
RGB values
Update Display to VGA
RGB values
Draw Elements when counters are between active regions
Output of RGB values, VSYNC & HSYNC, & pixel clk

Figure 7. Process Diagram

Results

Timing Diagrams

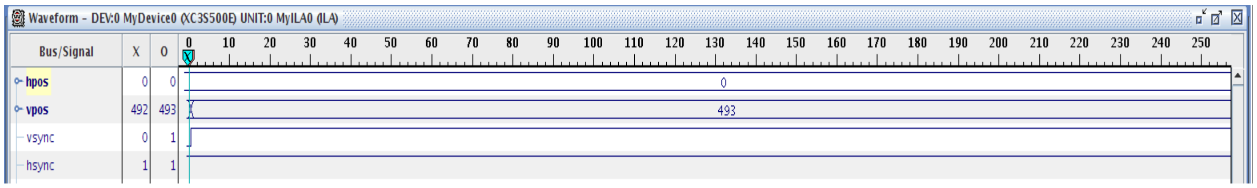

Figure 8. VPOS Synchronization Period (V-Sync Low to High)

This figure illustrates the VGA timing signals during the vertical synchronization pulse. The hpos signal cycles from 0 to 800 clock cycles, representing one complete horizontal line..When hpos reaches 800, it resets to 0, incrementing the vpos signal by one line (printing pixels row-by-row). In this portion of the waveform, vpos increments from 492 to 493, progressing through the Vsync pulse duration. The vsync signal is low at the very beginning of the waveform, as this is the 2 clock cycles specified for the sync pulse before incrementing the vpos. After the 2 clock cycles, vsync returns to high.

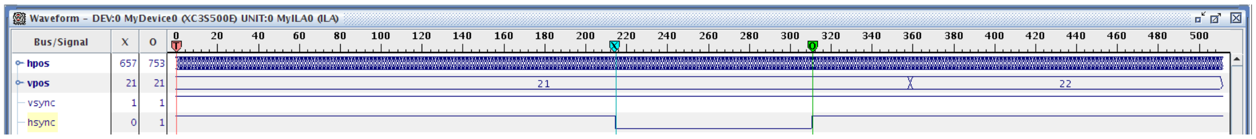

Figure 9. HPOS Synchronization Period (H-sync Low to HIgh)

This figure illustrates the behavior of the hsync signal during the horizontal synchronization period. The hpos signal increments from 657 to 753 clock cycles, representing the horizontal blanking interval of 96 clock cycles (753 - 657 = 96). Within this range, the hsync signal is low, indicating the horizontal sync pulse.

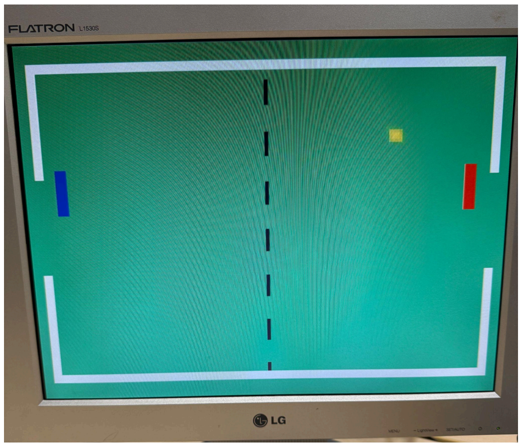

Figure 9. Screen Capture of Game Running

As seen in the figures above, the functionality of a VGA controller was successfully implemented in the Xilinx ISE CAD environment. The Vertical and Horizontal synchronization signals behave as expected and are displayed in figures 8 and 9 respectively. The functionality of the VGA controller was successfully emulated and displayed in figure 9, showing the Pong game running on the VGA monitor.

Conclusions

Ultimately, the objectives of this project were achieved, with the successful implementation of a Simple Video Game Processor using a VGA controller emulated in the Xilinx ISE CAD environment, on the Spartan-3E FPGA. This project served as a wonderful opportunity to further develop our VHDL coding knowledge and techniques, as well as delve into the realm of graphical processing hardware and technologies. Despite the slightly outdated nature of VGA technology, the logic and functionality still remains important to understand as it served as a basis for many of the modern graphical technologies we commonly use today. Overall, this project was very enjoyable and helped to enrich our understanding of VGA technology and graphical processing.

References

VGA connector pinout - basic introduction is here. NextPCB. (n.d.). https://www.nextpcb.com/blog/vga-connector-pinout

Coe-758. Ryerson University - Lev Kirischian - COE758. (n.d.). https://www.ee.torontomu.ca/~lkirisch/coe758/coe758.htm

Appendix with VHDL Code

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

entity lab2 is
        PORT (
        CLK : IN STD_LOGIC;
        RST : IN STD_LOGIC;
        BS : IN STD_LOGIC;
        RS : IN STD_LOGIC;
        HSYNC : inOUT STD_LOGIC;
        VSYNC : inOUT STD_LOGIC;
        DAC_CLK : OUT STD_LOGIC;
        Rout : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Gout : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        Bout : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        SW0    : in STD_LOGIC;
        SW1    : in STD_LOGIC;
        SW2    : in STD_LOGIC;
        SW3    : in STD_LOGIC
        );
end lab2;

architecture Behavioral of lab2 is
        SIGNAL clk25 : STD_LOGIC := '0';
        signal delay : Integer := 0;
        CONSTANT HD : INTEGER := 640; -- Horizontal Display
        CONSTANT HFP : INTEGER := 16; -- Right Border (Front Porch)
        CONSTANT HSP : INTEGER := 96; -- Horizontal Sync Pulse
        CONSTANT HBP : INTEGER := 48; -- Left Border (Back Porch)

        CONSTANT VD : INTEGER := 480; -- Vertical Display
        CONSTANT VFP : INTEGER := 10; -- Right Border (Front Porch)
        CONSTANT VSP : INTEGER := 2; -- Vertical Sync Pulse
        CONSTANT VBP : INTEGER := 33; -- Left Border (Back Porch)
        SIGNAL hPos : INTEGER := 0;
        SIGNAL vPos : INTEGER := 0;

        SIGNAL videoON : STD_LOGIC := '0';
        signal p1, p2 : Integer := 158;
        signal ball_x  : Integer :=312;
        signal ball_y : Integer := 232;
        signal ball_xmove, ball_ymove : Integer := 4;

        -- SIZINGS
        signal buttoncounter : Integer range 0 to 50 :=0;
        signal ballmovespeed : Integer := 2;
        signal playermovespeed : Integer := 3;
        signal paddlewidth : Integer := 60 ;
        signal score : Integer range -160 to 160 :=0;
        signal score_x : Integer := 312;

begin
        clk_div : PROCESS (CLK, RST)
        BEGIN
        IF (CLK'EVENT AND CLK = '1') THEN
        clk25 <= NOT clk25;
        DAC_CLK <= clk25;
        END IF;
        END PROCESS;

        Horizontal_position_counter : PROCESS (clk25, RST)
        BEGIN
        IF (RST = '1') THEN
        hPos <= 0;
        ELSIF (clk25'EVENT AND clk25 = '1') THEN
        IF (hPos = (HD + HFP + HSP + HBP)) THEN
        hPos <= 0;
        ELSE
        hPos <= hPos + 1;
        END IF;
        END IF;
        END PROCESS;
-------------------------------------------------------------------------
        horizontal_synchrnozation : PROCESS (clk25, RST, hPos)
        BEGIN
        IF (RST = '1') THEN
        HSYNC <= '0';
        ELSIF (CLK'EVENT AND CLK = '1') THEN
        IF ((hPos <= (HD + HFP)) OR (hPos > HD + HFP + HSP)) THEN
        HSYNC <= '1';
        ELSE
        HSYNC <= '0';
        END IF;
        END IF;
        END PROCESS;
---------------------------------------------------------------------
        Vertical_position_counter : PROCESS (clk25, RST, hPos)
        BEGIN
        IF (RST = '1') THEN
        vPos <= 0;
        ELSIF (clk25'EVENT AND clk25 = '1') THEN
        IF (hPos = (HD + HFP + HSP + HBP)) THEN
        IF (vPos = (VD + VFP + VSP + VBP)) THEN
                vPos <= 0;
        ELSE
                vPos <= vPos + 1;
        END IF;
        END IF;
        END IF;
        END PROCESS;
---------------------------------------------------------------------
        vertical_synchrnozation : PROCESS (clk25, RST, vPos)
        BEGIN
        IF (RST = '1') THEN
        VSYNC <= '0';
        ELSIF (CLK'EVENT AND CLK = '1') THEN
        IF ((vPos <= (VD + VFP)) OR (vPos > VD + VFP + VSP)) THEN
        VSYNC <= '1';
        ELSE
        VSYNC <= '0';
        END IF;
        END IF;
        END PROCESS;
---------------------------------------------------------------------
        video_on : PROCESS (clk25, RST, hPos, vPos)
        BEGIN
        IF (RST = '1') THEN
        videoOn <= '0';
        ELSIF (CLK'EVENT AND CLK = '1') THEN
        IF (hPos <= HD AND vPos <= VD) THEN
        videoOn <= '1';
        ELSE
        videoOn <= '0';
        END IF;
        END IF;
        END PROCESS;
---------------------------------------------------------------------
        draw : PROCESS (clk25, RST, hPos, vPos, videoOn)
        BEGIN
        IF (RST = '1') THEN
        ROut <= "00000000";
        GOut <= "11111111";
        BOut <= "00000000";
        ELSIF (CLK'EVENT AND CLK = '1') THEN
        IF (videoOn = '1') THEN
        --Horizontal Border UP AND DOWN
        IF ((hPos >= 15 AND hPos <= 625 AND ((vPos >= 24 AND vPos <= 35) OR (vPos >= 445
AND vPos <= 456)))) THEN
                ROut <= "11111111";
                GOut <= "11111111";
                BOut <= "11111111";
                --RIGHT AND LEFT BORDER UP
        ELSIF ( ( (hPos >= 615 AND hPos <= 625) OR (hPos >= 15 AND hPos <= 25) ) AND (vPos >=
35 AND vPos <= 170) )THEN
                ROut <= "11111111";
                GOut <= "11111111";
                BOut <= "11111111";

                -- RIGHT AND LEFT BORDER DOWN
        ELSIF (((hpos >= 615 AND hpos <= 625) OR (hpos >= 15 AND hpos <= 25)) AND vpos >= 300
AND vpos <= 445) THEN
        Rout <= "11111111";
        Gout <= "11111111";
        Bout <= "11111111";
                -- Create Lines Down the middle
        ELSIF ((hPos > 317 AND hPos < 323) AND vPos >= 30 AND vPos < 455) AND (((vPos - 15)
MOD 64) > 32) THEN
                ROUT <= "00101010";
                GOUT <= "00100000";
                BOUT <= "00101010";
        ELSIF (hPos >= 0 AND hPos <= 640 AND vPos >= 0 AND vPos <= 480) THEN
                ROut <= "00110100";
                GOut <= "10101110";
                BOut <= "01001110";
        END IF;
        --(Blue paddle)
        if(hPos > 40 AND hPos <=55 AND vPos> p1 AND vPos<= p1 + paddlewidth) THEN
                ROUT <= "00000000";
                GOUT <= "00000000";
                BOUT <= "10101101";
        -- (Pink Paddle)
        elsif(hPos > 580 AND hPos <=595 AND vPos> p2 AND vPos<= p2 + paddlewidth) THEN
                ROUT <= "11110111";
                GOUT <= "00000000";
                BOUT <= "00000000";
        elsif(hPos > ball_x AND hPos <= ball_x+16 AND vPos>ball_y AND vPos<=(ball_y+16)) THEN
                if(ball_x <= 12 OR ball_x+16 >= 628) THEN
                --Ball turn red when in goal
                ROUT <= "10101101";
                GOUT <= "00000000";
                BOUT <= "00000000";
                if(delay = 6000) THEN
                        ball_x <= 312;
                        delay <= 0;
                        ball_xmove <= -ball_xmove;
                        ball_ymove <= -ball_ymove;
                else
                        delay <= delay + 1;
                end if;
                else
                ROUT <= "11111111";
                GOUT <= "11011100";
                BOUT <= "00000000";
                end if;


        END IF;

        if(hPos = 639 AND vPos = 479)THEN
                -- BALL BOUNCE
                -- Parameters for wall
                if(ball_x <= 36 OR ball_x+16 >= 605) THEN
                if(ball_y > 180 AND ball_y < 290) THEN

                else
                        if(ball_x <= 36) THEN
                        ball_xmove <= ballmovespeed;
                        else
                        ball_xmove <= -ballmovespeed;
                        end if;
                end if;
                end if;

                -- Bottom part of net
                if(ball_y <= 36) THEN
                ball_ymove <= ballmovespeed;
                end if;
                --Top part
                if(ball_y+16 >= 435) THEN
                ball_ymove <= -ballmovespeed;
                end if;

                --Player1
                if((ball_x <= 60 AND ball_x+16 >=44) AND (ball_y+16 > p1 AND ball_y <
p1+paddlewidth)) THEN
```

```
                ball_xmove <= ballmovespeed;
                end if;

                -- Player2
                if((ball_x<=597 AND ball_x+16 >= 581) AND(ball_y+16 > p2 AND ball_y <
p2+paddlewidth)) THEN
                ball_xmove <= -ballmovespeed;
                end if;

                -- move player 1 (blue actual board pin is sw2 and 3)
                --sw1 up and down
                -- sw0 stop and go
                if ((SW0 = '1' AND SW1 = '0') AND p1 >= 40) then
                        p1 <= p1 - playermovespeed; --move up
                elsif ((SW0 = '0' AND SW1 = '1') AND p1 + paddlewidth <= 440) then
                        p1 <= p1 + playermovespeed;
                end if;

                -- Move Player 2
                if ((SW2 = '1' AND SW3 = '0') AND p2 >= 40) then
                        p2 <= p2 - playermovespeed;
                elsif ((SW2 = '0' AND SW3 = '1') AND p2 + paddlewidth <= 440) then
                        p2 <= p2 + playermovespeed;
                end if;

                -- Move Ball
                if(NOT(ball_x <= 10 OR ball_x+16 >= 630)) THEN
                ball_x <= ball_x + ball_xmove;
                ball_y <= ball_y + ball_ymove;
                end if;

        END IF;
        ELSE
        ROut <= "00000000";
        GOut <= "00000000";
        BOut <= "00000000";
        END IF;
        END IF;
        END PROCESS;


end Behavioral;
```