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	**Fall 2021**
Instructor	**Dr. Lev Kirischian**

Lab/Tutorial Report NO. 1

Report Title	**Design Project 2**

Section No.	01
Group No.	01
Submission Date	Dec. 6th, 2021
Due Date	Dec. 6th, 2021

Name	Student ID	Signature*
Darien Lee	xxxxx8176	DL
Taimoor Farooqi	xxxxx3873	TF

(Note: remove the first 4 digits from your student ID)

**By signing above you attest that you have contributed to this submission and confirm that all work you have contributed to this submission 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/policies/pol60.pdf.*

Design Project 2

Abstract

In the COE758 final project, we will be building a simple video game processor by using a Video Graphic Adaptor (VGA) as well as the Xilinx Spartan-3E FPGA simulator program. The game which we will be simulating is the classic Pong game in which 2 players will be able to move their paddles in a 2-way vertical direction while trying to prevent a ball from entering their own goal line.

Each time the ball is hit by a paddle, either the horizontal velocity direction is flipped - resulting in the ball going in the opposite direction towards the other player. If a ball strikes a wall of the simulated environment, the ball's vertical velocity direction is flipped - resulting in the ball going towards the opposing wall. Each time the ball is passed through a player's net, it will then begin the game reset process by changing the ball colour to red temporarily and being recentered in the middle of the play area.

The environment is simulated using a VGA display which is updated every time the program sends the RGB, HSYNC and VSYNC values to the display. With every new input, the program draws each individual display and continues to update until the players decide to quit the game.

Introduction

Games have revolutionized the online entertainment industry as a way for people to destress their lives by playing video games and allowing them to live in a virtual world. In Project #2, we will explore how to implement a simple video game processor using a Video Graphic Adaptor (VGA) and the Xilinx Spartan-3E FPGA simulator program. The VGA simulator will act as a display and a tool for students to get practical experience in designing and implementing real-time high-performance signal generators with custom logic programming. The game that students will be creating is the classic **Pong** game using the Xilinx ISE CAD environment. As Pong requires real-time updating of the display, we will also be defining several specifications such as horizontal and vertical parameters of the game, the number of clock cycles for each horizontal parameter and the number of lines for each vertical parameter. The 2 players of the game will be able to move their paddles in a strictly vertical direction and hit the yellow ball in order to prevent the ball from passing the gate at each end of the board. The movement of each paddle is tracked using an input that is completed by the user and the display is then updated with each new input. The other rules remain the same as original Pong.

System Specifications

There are three distinct system specifications outlined for the Simple Video-Game Processor (SVGP):

1. Static Video-Frame: When simulating the program, we need to be able to see all elements of the board including all borders and dynamic components that will impact the flow and functionality of the game. The static elements will help each player recognize the environment and simulate a real-time Pong video game environment.
2. Dynamic Elements: The three dynamic elements of the Pong game will be the (i) yellow ball and (ii) the two distinctly coloured paddles. All of these elements will be required to change their locations as the simulation progresses and will alter the outcome of each subsequent game. The 2 paddles will be able to “hit” the ball which will result in the ball bouncing in the opposite horizontal direction towards the opposing player. As for the 2 paddles/players, they will have a strictly vertical movement that each player is able to alter by providing an input while the program is running. When a player provides the correct input, the program will evaluate the input as either an up or down movement request and reflect the paddle’s location by moving it in the requested direction.
3. Behavior: Now we must define what actually happens when a player successfully hits/misses the ball with their paddle during the game. When the ball is hit by a paddle, the trajectory of the ball will be altered by ±90° depending on the initial direction of the hit. The second scenario is when the player misses the ball and the ball travels past the paddle. When the ball passes the paddle, it will enter the “gate” that is displayed behind the paddle, change its colour to red and disappear after entering the gate. After a brief moment, the ball will reappear in the middle of the board as the standard colour of yellow ready to be played again by the players.

Device Description/Design

VGA Specification

There are 2 sets of VGA specifications for Project #2: Horizontal Parameters and Vertical Parameters. All horizontal (line) time periods are specified in multiples of the VGA pixel clock, which is 25 MHz for a 60 Hz refresh rate. All vertical (frame) time periods are specified in multiples of VGA lines. **Table 1** outlines the horizontal parameters and their respective clock cycles and **Table 2** outlines the vertical parameters and the number of dedicated lines.

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

Table 1: VGA Horizontal Parameters

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

Table 2: VGA Vertical Parameters

Symbols and Block Diagrams

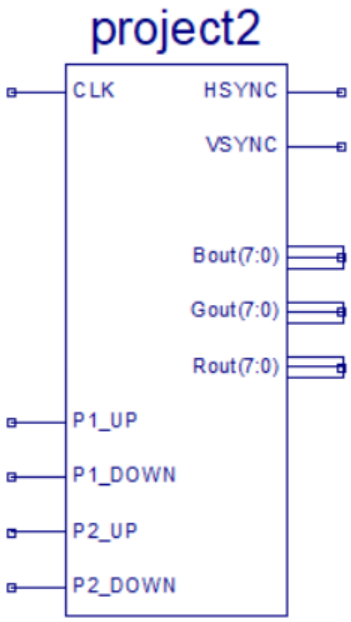

Figure 1

In the diagram shown in **Figure 1**, we can see all the components to the Ping Pong game. On each side of the diagram, we have the input pins shown with P1_UP, P1_DOWN, P2_UP and P2_DOWN being the input pins to control each player's movement in the pong game. When the signal is set to 1 for the respective paddle movement direction (eg. P1_UP= 1), the opposite direction (eg. P1_DOWN) will be set to 0 allowing each player to move up and down according to the inputs without being locked in place by 2 opposing directions (eg. P1_UP= 1 and P1_DOWN= 1). When both directions of up and down are set to 0, the paddle is simply not moving. The CLK input pin is set to 25MHz so that the image of the board is properly displayed as the board gets cut off at higher clock speeds.

The output pins of HSYNC and VSYNC notify the VGA display when new lines or new frames need to be drawn. The R, G, B output pins are used to colour the board with the red, green and blue respectively for visual clarity of the board, players and ball. These colours can be used to create many other unique colours but for our purposes, we only use red(ball), blue(P1 paddle), green(board), yellow(ball) and magenta(P2 paddle). The VGA display receives these outputs (HSYNC, VSYNC, R, G, B) every 20ns to draw the game's interface.

State/Process Diagrams

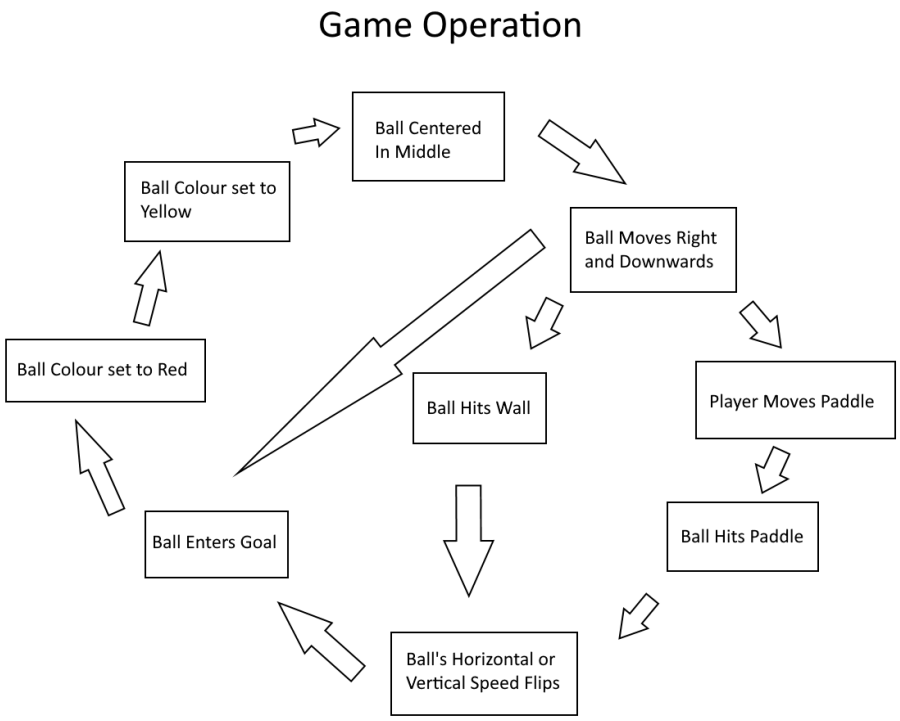

Figure 2: Process Diagram for Game operation

In **Figure 2** above, the process diagram of the game is shown. The game begins with a yellow ball starting in the centre of the display between the two player paddles. We programmed the ball to always start from the centre and move towards player 2 on the right with a down-right vector of travel. The velocity of the ball is constant but when it collides with either a player paddle or wall, the ball will change directions depending on the object hit. If the ball collides with a player, the ball's x vector will be flipped while the y vector remains the same. If the ball collides with a wall, the ball's x vector will remain the same while the y vector is flipped. If the ball ever touches the goal/net of a player, the ball will turn red briefly before resetting to the centre position to restart the loop.

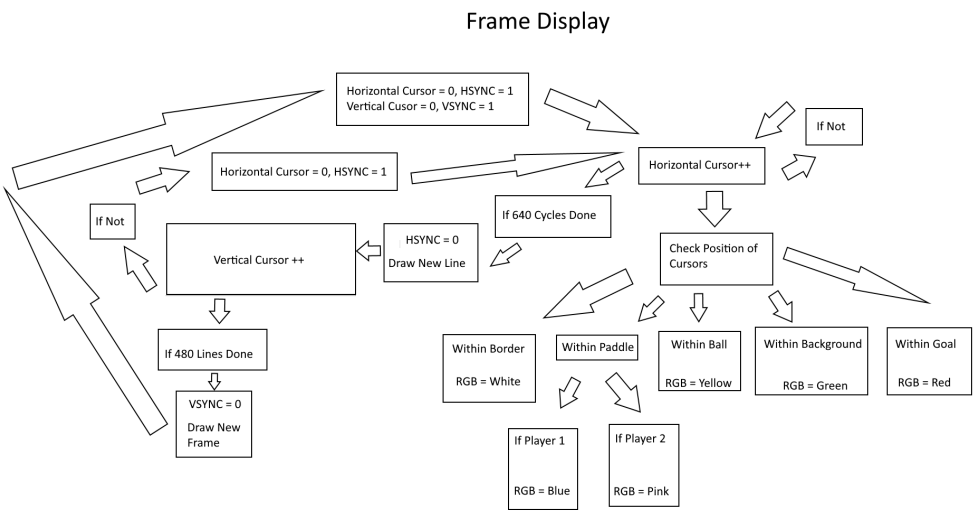

Figure 3: Process Diagram for Frame Display

In **Figure 3** above, the process diagram of how frames are displayed is shown. Horizontal and vertical cursors are used in the VGA to determine the position of the pixel in the frame that needs to be drawn. These are controlled by the horizontal and vertical position counter processes (which are the x and y vectors that form the “cursor”) respectively and are initialized at 0. HSYNC and VSYNC are both set to 1 as the frame-drawing process is initiated. Within each cycle of 20ns, the horizontal positional counter is incremented to draw the next pixel. When the horizontal counter reaches its max (the determined frame horizontal size of 640 pixels and thus 640 cycles), the vertical counter will be incremented, HSYNC is set back to 0 to indicate completion of the line and the horizontal counter is reset back to 0. This cycle continues until the vertical counter reaches its max size (480 pixels) which will take 307,200(640*480) cycles in total. Upon completion of both these counters, the HSYNC and VSYNC are both set to 0 indicating that the drawing of the frame is complete. The horizontal counter and the vertical counter will both be reset to 0 while the HSYNC and VSYNC are both set to 1 to begin the process of drawing a new frame. This process will loop until the program is stopped. Any movement of the ball and players only occur once per full frame cycle so that there is no screen tearing.

Results

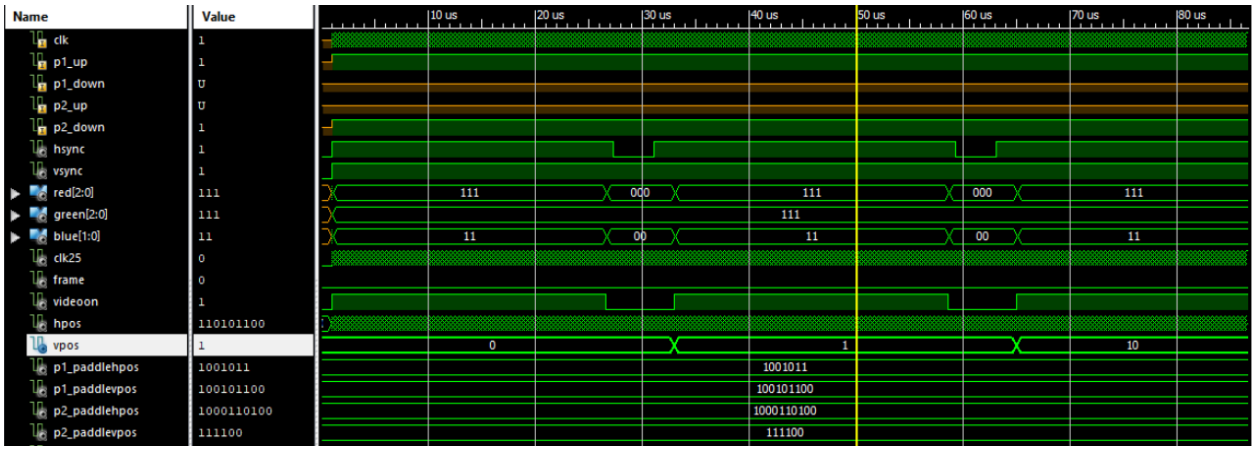

Figure 4: Timing Diagram for when *hsync* = 1 and *vsync* = 1

In **Figure 4**, a snippet of the timing diagram for when the *hsync* and *vsync* are both 1 is shown, indicating that the cursor consisting of the horizontal and vertical are still drawing within the board size parameters (640*480). Here, the RGB (‘111’, ’111’, ’11’) signals indicate that the pixels it is drawing are white. This tells us that it is currently drawing the borders of the game board.

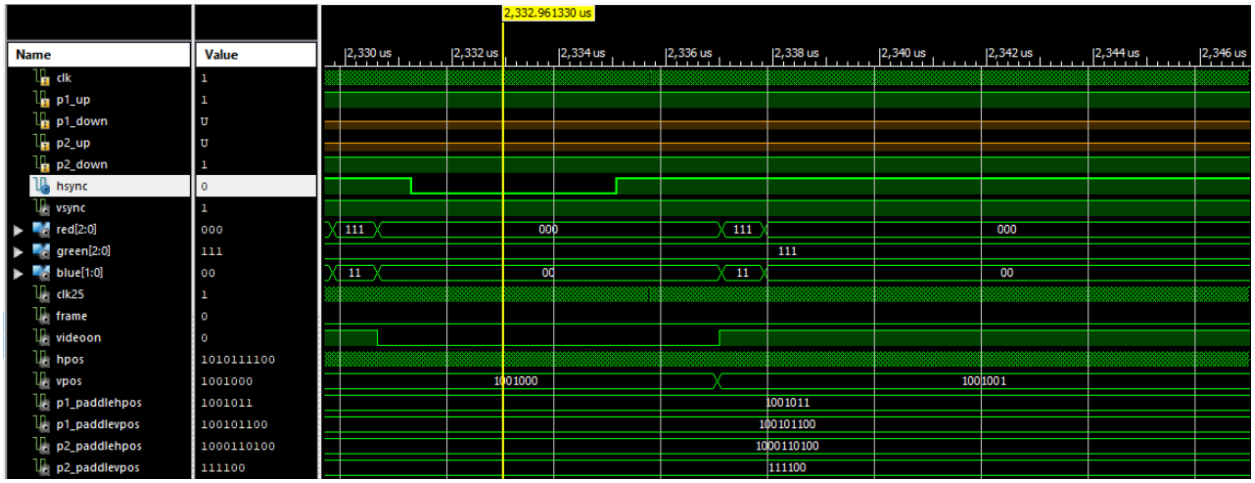

Figure 5: Timing Diagram for when *hsync* = 0 and *vsync* = 1.

In **Figure 5**, a snippet of the timing diagram for when the *hsync* = 0 and *vsync* = 1 is shown, indicating that the program has just finished drawing the horizontal line that it is currently on and will be incremented shortly to begin a new horizontal line. Here, the RGB (‘000’, ‘111’, ‘00’) signals indicate that it has just finished drawing a green pixel. The *vsync* = 1 indicates that the vertical position is still within the drawing field. However, if *hsync* = 1 and *vsync* = 0, we would have the cases swapped whereby the drawing of the vertical line would be finished drawing and wait to be incremented to the next line.

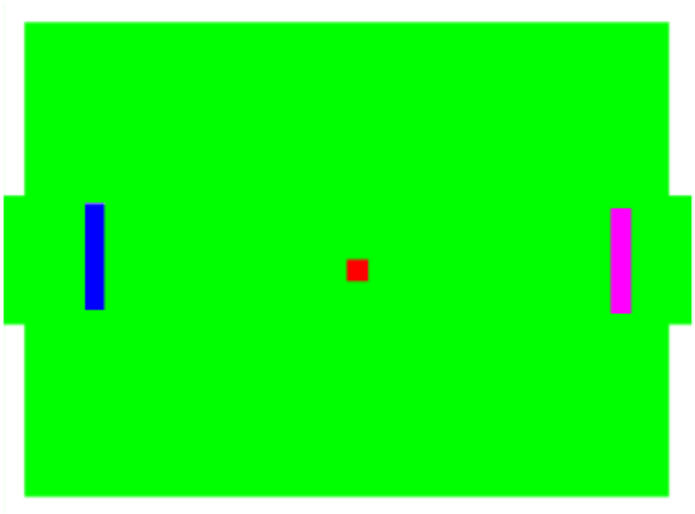

Figure 6a

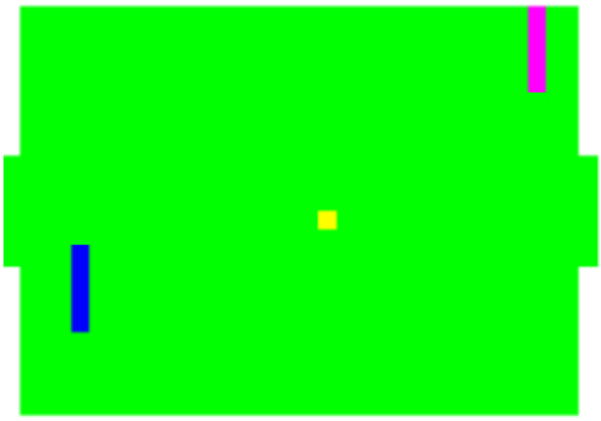

Figure 6b

In the **Figure 6a** above, we see that the ball had just been scored. This is also the starting position of the ball, indicated by its red colour and center position after a reset. When in play, the ball's colour will change back to the original yellow colour (**Figure 6b**) and the game will resume.

Conclusions

In this project, we learn the importance and intricacies of digital systems regarding displays in our devices. The design choices of these display frames greatly contributes to the functionality and flow of the entire game. We often overlook the background processes and systems that make up our everyday devices. This project showed us not only how to fully build but also simulate a fully functioning Pong game without the need of physical equipment or hardware in front of us.

References

Eastwood, Eric. “VGA Simulator: Getting Started.” *Eric Eastwood Web Developer*, 14 Jan. 2018, ericeastwood.com/blog/8/vga-simulator-getting-started.

Kirischian, Lev. “Project #2 – Simply Video Game Processor for VGA” COE758, https://www.ee.ryerson.ca/~lkirisch/ele758/labs/SimpleVideoGame[11-11-11].pdf

FPGAKey. “A VGA Interface in Verilog”, https://www.fpgakey.com/tutorial/section892

Appendix

Project2.vhd

```
28  LIBRARY ieee;
29  USE ieee.std_logic_1164.ALL;
30  USE ieee.std_logic_textio.ALL;
31  USE std.textio.all;
32  use ieee.numeric_std.all;
33
34
35  -- Uncomment the following library declaration if using
36  -- arithmetic functions with Signed or Unsigned values
37  --USE ieee.numeric_std.ALL;
38
39  ENTITY Project2 IS
40      PORT(
41           clk : IN  std_logic;
42           P1_UP : IN  std_logic;
43           P1_DOWN : IN  std_logic;
44           P2_UP : IN  std_logic;
45           P2_DOWN : IN  std_logic
46
47           HSYNC : OUT  std_logic;
48           VSYNC : OUT  std_logic;
49           RED : OUT  std_logic_vector(2 downto 0);
50           GREEN : OUT  std_logic_vector(2 downto 0);
51           BLUE : OUT  std_logic_vector(1 downto 0)
52          );
53  END Project2;
54
55  ARCHITECTURE Behavorial OF Project2 IS
56
57     --Outputs
58     signal HSYNC : std_logic:='0';
59     signal VSYNC : std_logic:='0';
60     signal RED : std_logic_vector(2 downto 0);
61     signal GREEN : std_logic_vector(2 downto 0);
62     signal BLUE : std_logic_vector(1 downto 0);
```

```
66    --constants
67    constant HArea : integer:= 639;
68    constant HFront : integer:= 16;
69    constant HPulse : integer:= 96;
70    constant HBack : integer:= 48;
71    constant VArea : integer:= 479;
72    constant VFront : integer:= 10;
73    constant VPulse : integer:= 2;
74    constant VBack : integer:= 33;
75
76    --ball size
77    constant BallSize : integer:=20;
78
79    --borders
80    constant Top : integer:= 0;
81    constant TopLength : integer:= 20;
82    constant Bot : integer := 479;
83    constant BotLength: integer:= 20;
84    constant Left : integer := 0;
85    constant LeftLength : integer := 20;
86    constant Right : integer := 639;
87    constant RightLength : integer := 20;
88
89    --goals
90    constant GoalTop :integer := 180;
91    constant GoalBottom : integer := 300;
92
93    --clock and other funct
94    signal clk25 : std_logic := '0';
95    signal Frame : std_logic :='0';
96    signal VideoOn : std_logic :='0';
97
98    --cursor
99    signal HPos : integer :=0;
100   signal VPos : integer :=0;
```

```
101
102     --p1 paddle
103     signal P1_PaddleHPos : integer:=75;
104     signal P1_PaddleVPos : integer:=300;
105     --p2 paddle
106     signal P2_PaddleHPos: integer:=564;
107     signal P2_PaddleVPos: integer:=60;
108
109     --paddle configs
110     signal PaddleWidth :integer :=20;
111     signal PaddleLength : integer := 120;
112
113     --ball configs
114     signal BallHPos :integer:= 320;
115     signal BallVPos :integer:= 240;
116     signal BallHSpeed :integer:= 20;
117     signal BallVSpeed :integer:= 20;
118     signal BallColor : std_logic:='0';
119
120     constant clk_period : time := 10 ns;
121
122  BEGIN
123     clk_div:process(CLK)
124
125     begin
126        if(CLK'event and CLK='1') then
127           clk25 <= not clk25;
128        end if;
129     end process;
130
131     Horizontal_position_counter:process(clk25)
132
133     begin
134        if(clk25'event and clk25='1') then
135        --reset h-position
136        if (HPos = (HArea + HFront + HPulse + HBack)) then
```

```
136         if (HPos = (HArea + HFront + HPulse + HBack)) then
137            Frame <= '0';
138            HPos <= 0;
139
140            --draw next frame after reaching max h and v positional
141            if (VPos = (VArea + VFront + VPulse + VBack)) then
142               Frame <= '1';
143            end if;
144            else
145               Frame <= '0';
146               HPos <= HPos + 1;
147            end if;
148         end if;
149      end process;
150
151      Vertical_position_counter:process(clk25, HPos)
152
153      begin
154         if(clk25'event and clk25='1') then
155            --reset v-position
156            if (HPos = (HArea + HFront + HPulse + HBack)) then
157               if(VPos = (VArea + VFront + VPulse + VBack)) then
158                  VPos <= 0;
159               else
160                  --increment vpos to draw next line
161                  VPos <= VPos + 1;
162               end if;
163            end if;
164         end if;
165      end process;
166
167      Horizontal_Synchronization:process(clk25, HPos)
168
169      begin
170         if(clk25'event and clk25='1') then
171            --h sync when outside sync pulse region
```

```
172            if((HPos <= (HArea + HFront)) or (HPos > HArea + HFront +HPulse)) then
173               HSYNC <= '1';
174            else HSYNC <= '0';
175            end if;
176         end if;
177      end process;
178
179      Vertical_Synchronization:process(clk25, VPos)
180
181      begin
182         if(clk25'event and clk25='1')then
183            --v sync when outside sync pulse region
184            if((VPos <= (VArea + VFront)) or (VPos > VArea + VFront + VPulse)) then
185               VSYNC <= '1';
186            else VSYNC <= '0';
187            end if;
188         end if;
189      end process;
190
191      video_on:process(clk25, HPos, VPos)
192
193      begin
194         if(clk25'event and clk25='1')then
195            --enable video output when in active image area
196            if(HPos <= HArea and VPos <= VArea)then
197               VideoOn <= '1';
198            else
199               VideoOn <= '0';
200            end if;
201         end if;
202      end process;
203
204      ball_move:process(clk25, Frame, BallVPos, BallHPos, P1_PaddleHPos, P1_PaddleVPos,PaddleLength,P2_PaddleHPos, P2_PaddleVPos, Ba
205
206      begin
207         --when new frame drawn
208         if(clk25'event and clk25='1' and Frame='1')then
```

```
209         --reset ball color to yellow
210         BallColor <= '0';
211      end if;
212      --when ball touches top border, flips vspeed downwards
213      if(BallVPos - Ballsize <= Top + Toplength) then
214         BallVSpeed <= 20;
215      end if;
216
217      --when ball touches bottom border, flips vspeed upwards
218      if(BallVPos + Ballsize >= Bot - BotLength) then
219         BallHSpeed <=-20;
220      end if;
221
222      --when ball touches right border, changes h direction to left
223      if(BallHPos + BallSize >= Right + RightLength) then
224         BallHSpeed<=-20;
225      end if;
226
227      --when ball touches left border, changes h direction to left
228      if(BallHPos - BallSize >= Left + LeftLength) then
229         BallHSpeed<=20;
230      end if;
231
232      --when ball touches p1 paddle, h direction is right
233      if(BallHPos - BallSize <= P1_PaddleHPos) and (BallVPos >= P1_PaddleVPos and BallVPos < P1_PaddleVPos + PaddleLength) then
234       BallHSpeed <= 20;
235      end if;
236
237      --when ball touches p2 paddle, h direction is left
238      if(BallHPos + BallSize >= P2_PaddleHPos) and (BallVPos >= P2_PaddleVPos and BallVPos < P2_PaddleVPos + PaddleLength) then
239       BallHSpeed <= -20;
240      end if;
241
242      --updates ball's position based on current position, v and h speed
243      BallHPos <= BallHpos + BallHSpeed;
244      BallVPos <= BallVPos + BallVSpeed;
245
```

```
246       --resets ball's position to center upon reaching a goal, sets ball color to red
247       if ((BallHPos - BallSize <= Left + LeftLength) or (BallHPos + BallSize >= Right - RightLength)) then
248          if((BallHPos - BallSize <= Left + LeftLength) or (BallHPos + BallSize >= Right - RightLength)) then
249             if(BallVPos >= GoalTop and BallVPos <= GoalBottom) then
250                BallHPos <= 320;
251                BallVpos <= 240;
252                BallColor <= '1';
253             end if;
254          end if;
255       end if;
256    end process;
257
258    paddle_move:process(clk25, Frame)
259
260    begin
261       --on new frame
262       if(clk25'event and clk25='1' and Frame='1')then
263          --if paddle is under top border, move paddle up when p1_up is high
264          if(P1_UP='1' and P1_PaddleVPos >= Top + TopLength) then
265             P1_PaddleVPos <=P1_PaddleVPos - 50;
266          end if;
267
268          --if padle above the bottom border, move the paddle down when p1_down is high
269          if(P1_DOWN ='1' and P1_PaddleVPos + PaddleLength <= Bot - BotLength) then
270             P1_PaddleVPos <= P1_PaddleVPos + 50;
271          end if;
272
273          --if paddle is under top border, move paddle up when p2_up is high
274          if(P2_UP='1' and P2_PaddleVPos >= Top + TopLength) then
275             P2_PaddleVPos <=P2_PaddleVPos - 50;
276          end if;
277
278          --if padle above the bottom border, move the paddle down when p2_down is high
279          if(P2_DOWN ='1' and P2_PaddleVPos + PaddleLength <= Bot - BotLength) then
280             P2_PaddleVPos <= P2_PaddleVPos + 50;
281          end if;
282       end if;
```

```
283     end process;
284
285     draw:process(clk25, HPos, VPos, VideoOn)
286
287     begin
288        if(clk25'event and clk25='1') then
289           --when within active image area
290           if(VideoOn ='1') then
291              --set color to draw as white (top & bottom borders)
292              if((VPos >= Top and VPos <= Top + TopLength) or (VPos >= Bot - BotLength and VPos <= Bot)) then
293                  RED <= "111";
294                  GREEN <= "111";
295                  BLUE <= "11";
296              elsif((HPos >= Left and HPos <= Left + LeftLength) or (HPos <= Right and HPos >= Right - RightLength)) then
297                 --set color as white (left a& right borders)
298                    if((VPos >= Top and VPos <= GoalTop) or (VPos >= GoalBottom and VPos <= Bot)) then
299                       RED <= "111";
300                       GREEN <= "111";
301                       BLUE <= "11";
302                    end if;
303                 --colors p1 paddle as blue
304                 elsif((HPos > P1_PaddleHPos and HPos < P1_PaddleHPos + PaddleWidth) and (VPos > P1_PaddleVPos and VPos < P1_PaddleV
305                       RED <= "000";
306                       GREEN <= "000";
307                       BLUE <= "11";
308                 --colors p2 paddle as magenta
309                 elsif((HPos > P2_PaddleHPos and HPos < P2_PaddleHPos + PaddleWidth) and (VPos > P2_PaddleVPos and VPos < P2_PaddleV
310                       RED <= "111";
311                       GREEN <= "000";
312                       BLUE <= "11";
313                 elsif((HPos >= BallHPos and HPos < BallHPos + BallSize) and (VPos >= BallVPos and VPos < BallVpos + BallSize)) then
314                 --colors ball yellow
315                    if(BallColor = '0') then
316                       RED <= "111";
317                       GREEN <= "111";
318                       BLUE <= "00";
319                 --colors ball red
```

```
319                 --colors ball red
320                    else
321                       RED <= "111";
322                       GREEN <= "000";
323                       BLUE <= "00";
324                    end if;
325              --colors backgroud green
326              else
327                 RED <= "000";
328                 GREEN <= "111";
329                 BLUE <= "00";
330              end if;
331              --green when not in active image region
332              else
333                 RED <= "000";
334                 GREEN <= "111";
335                 BLUE <= "00";
336              end if;
337           end if;
338        end process;
339
340
341     -- Clock process definitions
342     process(clk)
343        file file_pointer: text is out "write.txt";
344        variable line_el: line;
345     begin
346        if rising_edge(clk) then
347           write(line_el, now);
348           write(line_el, ":");
349
350           write(line_el, " ");
351           write(line_el, HSYNC);
352
353
354           write(line_el, " ");
355           write(line_el, VSYNC);
```

```
345     begin
346        if rising_edge(clk) then
347           write(line_el, now);
348           write(line_el, ":");
349
350           write(line_el, " ");
351           write(line_el, HSYNC);
352
353
354           write(line_el, " ");
355           write(line_el, VSYNC);
356
357           write(line_el, " ");
358           write(line_el, RED);
359           write(line_el, " ");
360           write(line_el, GREEN);
361           write(line_el, " ");
362           write(line_el, BLUE);
363           writeline(file_pointer, line_el);
364        end if;
365     end process;
366
367  -- clk_process :process
368  --   begin
369  --    clk <= '0';
370  --    wait for clk_period/2;
371  --    clk <= '1';
372  --    wait for clk_period/2;
373  --   end process;
374  --
375  END Behavorial;
```