Faculty of Engineering and Architectural Science

Department of Electrical and Computer Engineering

Course Number	COE 758
Course Title	Digital Systems Engineering
Semester/Year	F2020
Lab No	Project #2
Instructor Name	Lev Kirischian
Section No	03/01

Submission Date	12/07/2020
Due Date	12/07/2020

Name	Student ID	Signature*
Brandon Ho	500727531	B.H.
Vatsal Shreekant	500771363	

**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:*

www.ryerson.ca/senate/current/pol60.pdf

Contents

1. Abstract

Pong was one of the first computer games ever created. The game features two horizontal moving paddles, where each individual paddle is controlled by one of the two players. A player gets a point by reflecting a ball pass the opponent's paddle, where the goal of the game is to defeat your opponent by being the first one to gain 10 points. The game was originally developed by Allan Alcorn and released in 1972 by Atari corporations (Lowood, 2009). The objective for this experiment is to simulate Pong through the Xillinx Spartan-3E FPGA (Kirischian, 2020). To represent the game, horizontal lines are drawn across the screen from the top of the screen to the bottom vertically. Moreover, the VGA pixel clock is set at 25 mHz and the refresh rate at 60 Hz. The VGA Interface Pin Constraints are provided in the lab manual (Kirischian, 2020). The game shows a green background with white borders. The paddle colors are blue and pink with a yellow ball.

2. Introduction

Before implementing the game, it is imperative to understand the process of image formation on the VGA monitor. The color signals on the VGA monitor are to be displayed through a frame of a size 640 by 480 pixels. The display is bifurcated into two components: horizontal axis and vertical axis. Each pixel on the monitor represents one clock cycle. The monitor should display 640 pixels along the horizontal axis and across 480 rows along the vertical axis. "Blanking" is the period in time between each line being displayed (horizontal blanking) and each frame being displayed (vertical blanking). The formula for blanking is as follows (Lagroix, Yanko & Spalek, 2012):

$$\text{Blanking} = \text{Front Porch} + \text{Back Porch} + \text{Sync}$$

The purpose of the horizontal blanking is for the electron beam displaying the image on the VGA monitor to reset the pointer for the next horizontal line. These electrons display a color for each pixel in a horizontal line. Moreover, as the pointer resets, the system will display the next vertical line. The function of vertical blanking is the same as horizontal blanking but with an added step. Instead of displaying a single horizontal line, the vertical blank would display a full frame. Hence, the complete VGA parameters for the monitor would be 525 by 800 clock cycles.

After understanding image formation, the next step is to create the game. The specifications of the game are as follows:

1) Static green background enclosed with a white border.
2) The ball can move within the border after being reflected by either of the two paddles or the border.

To represent the logic behind collisions, 'if-statements' are implemented to simulate ball moving in the opposite direction. The movement of the ball and paddles were split into two components: the horizontal component and the vertical component. As per the logic implemented, the ball is reset in the middle once it reaches the goal.

3. System Specifications

Functional:

The system consists of 3 functional cases. The functional cases are:

1. Ball "flying" in the background
    a. Ball switching colors upon reaching the goal
2. Ball collisions
    a. With paddle
    b. With border/wall
3. Paddles moving up and down

The actions shown in the functional cases will include changing pixels according to each refresh rate, pin assignments with the switches for paddle movement, and pin assignments.

Technical:

The system specifications for the technical portion consists of the following 4 components listed below, which are implemented in one schematic symbol.

1. Sync- Used to refresh a frame after the horizontal/vertical line has been displayed.
2. Refresh- Used to keep the pixel clock in track and reset the pixel clock back to one.
3. VGA Controller- Used to display the different components onto the VGA monitor, these components are static/dynamic background, ball, and paddles.
4. Pong- Used for the incorporation of the other four components and outputs the movement of the ball and paddle as well as the collision scenario.

4. Device description / design

The following are the block symbols of the Pong game, which internally includes the Sync, Refresh, VGA Controller, and Ping Pong components.

a) Symbols

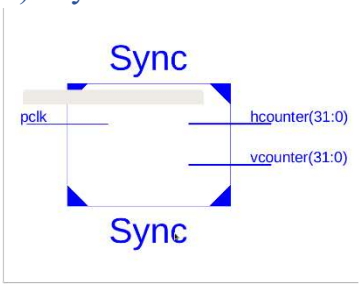

Figure 1: Sync Symbol

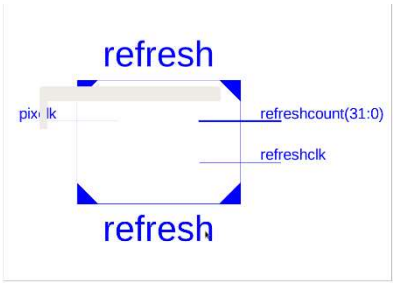

Figure 2: Refresh Symbol

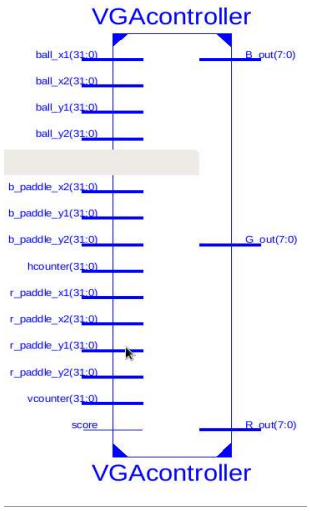

Figure 3: VGA Controller Symbol

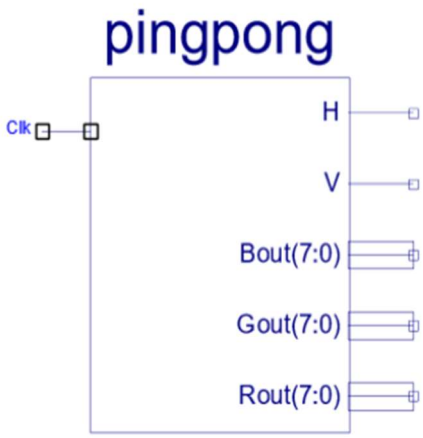

Figure 4: Ping Pong Symbol

The symbols shown above were constructed such that it will take inputs from the top-level entity pong file. The Sync file takes the pixel clock and counts the hcounter and vcounter which the pong file uses. The refresh file counts each frame, which it adds to the counter when 60 frames pass. The VGA Controller file takes the inputs for the ball, red and blue paddles, the score, hcounter and vcounter. In order to display the moving parts which include the ball and paddles, constant updates to the x and y position are needed to display the appropriate colours. The hcounter and vcounter are needed to calculate the width and height of the screen (640x480). The outputs are 8-bit RGB signals which are mapped to the RGB ports in the pong file. The pong file then takes the hcounter, vcounter, RGB and pixel clock signals and maps it to the VGA display.

b) VGA Specification (as used in project)

Table 1: VGA Horizontal/Vertical Parameters

	Horizontal Parameters	**Vertical Parameters**
Parameter	**Clock Cycles**	**Clock Cycles**
Complete Line	800	525
Front Porch	16	10
Sync Pulse	96	2
Back Porch	48	33
Active Image Area	640	480

c) Block diagrams

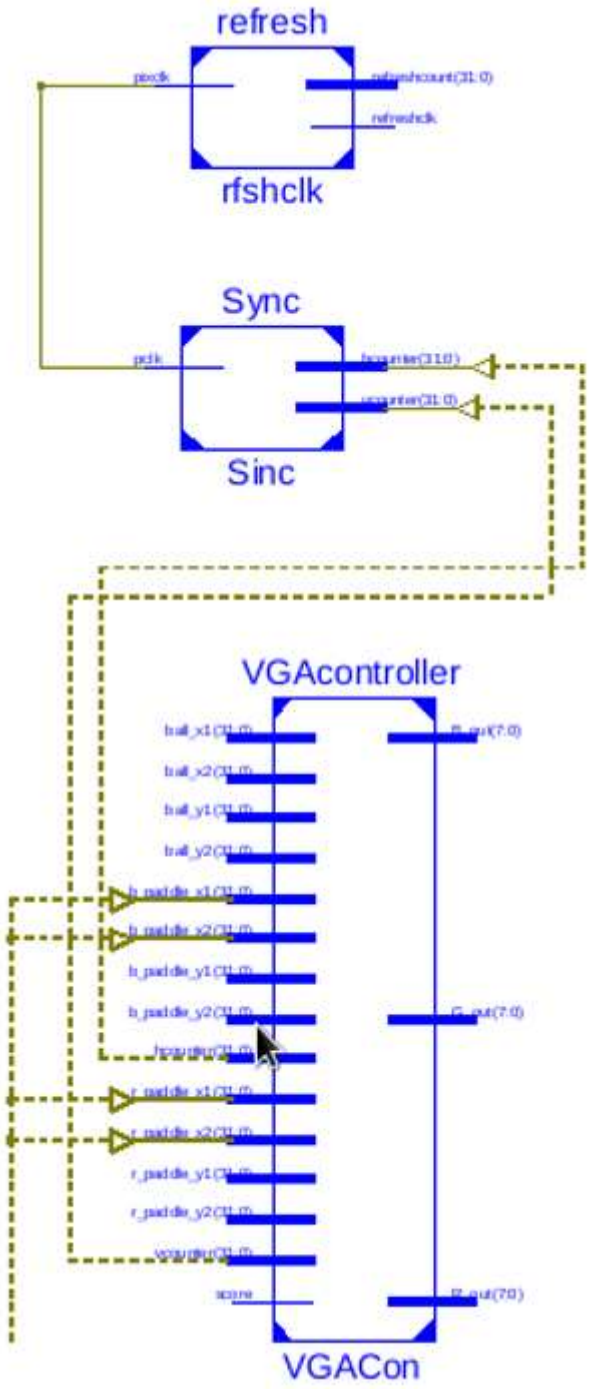

Figure 5: Block Diagram for Pong

d) Process diagram

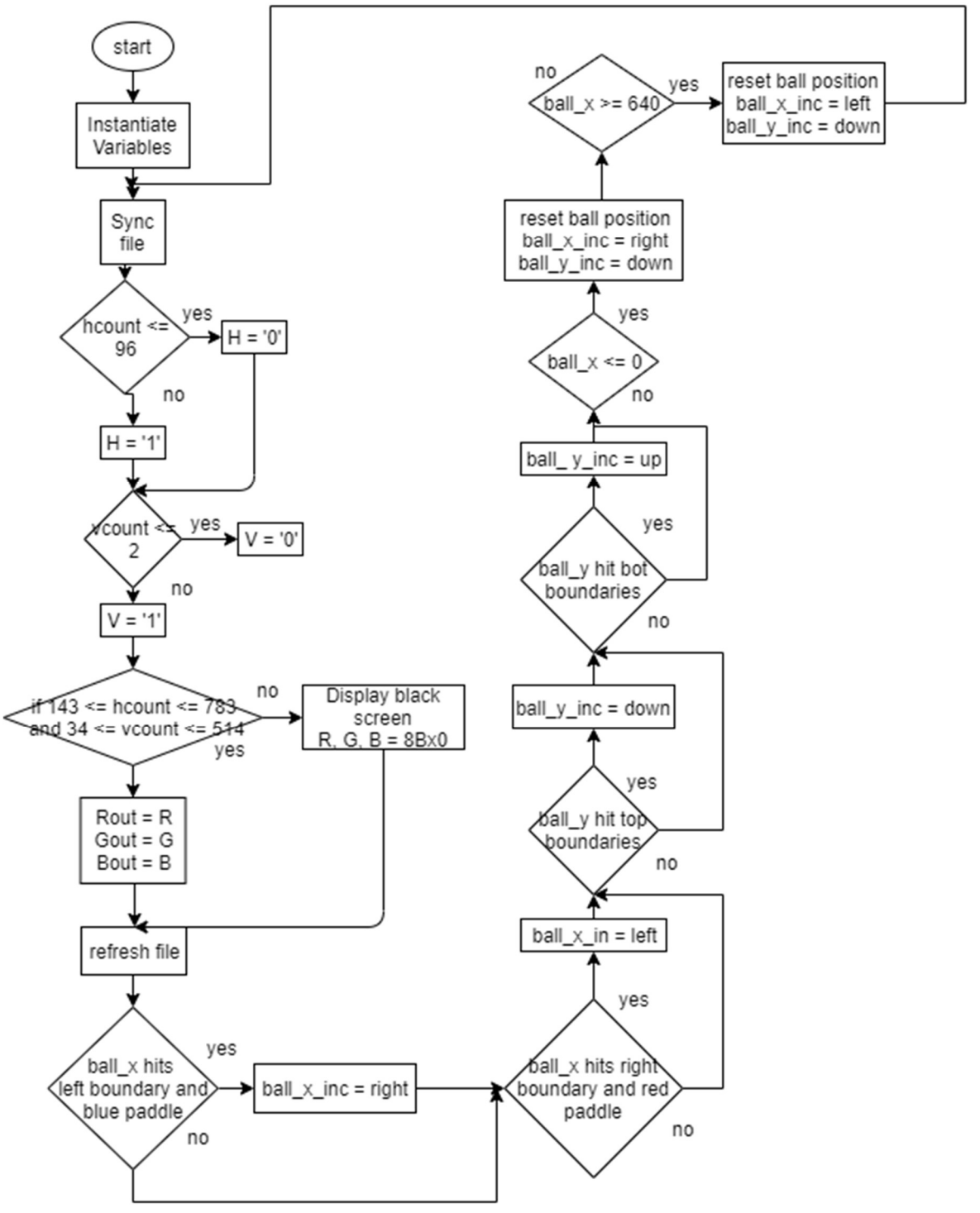

Figure 6: Process Diagram for the main Pong System

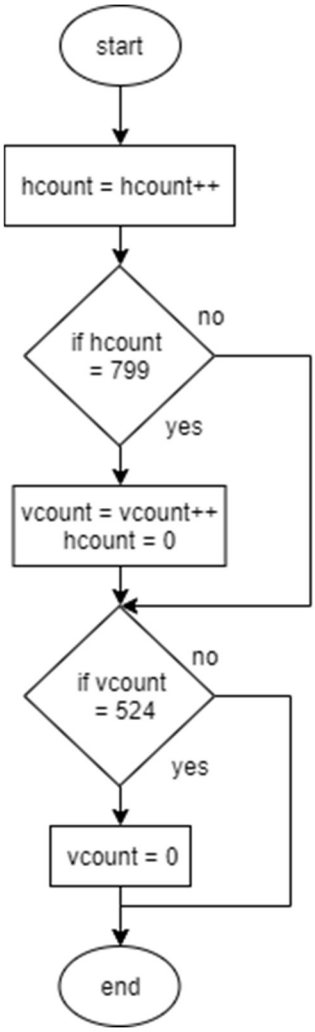

Figure 7: Process diagram for the Sync

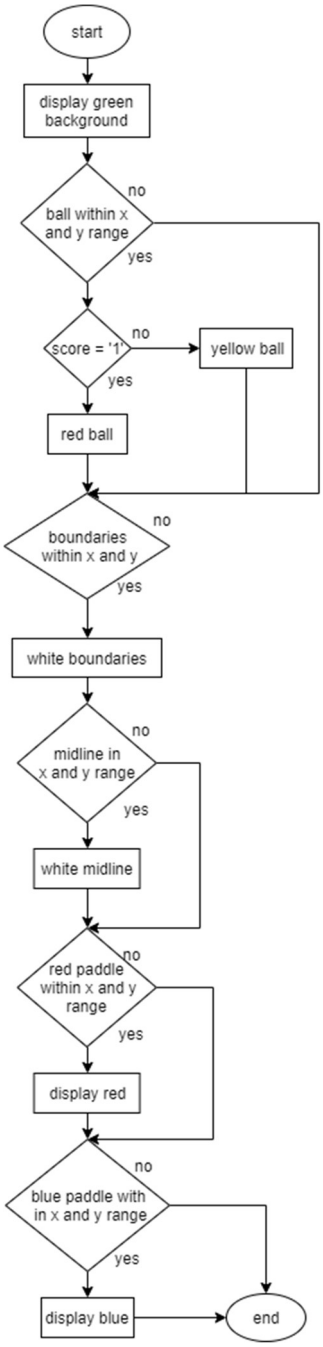

Figure 8: Process diagram for the VGA Controller

The process diagram in Figure 6 shows the overall "Pong" game process that executes in the pong file. The "Pong" file accesses the subsequent components Sync, Refresh, VGA Controller and treats each as a separate process. Each component process diagram has an "end" statement which indicates that it can be accessed by the pong file upon its next cycle. For the process in Figure 7, that process is within the pong file but has an independent parameter in which it acts as an independent cyclical entity.

5. Results

a) Timing diagrams (must show HSync and VSync signals)

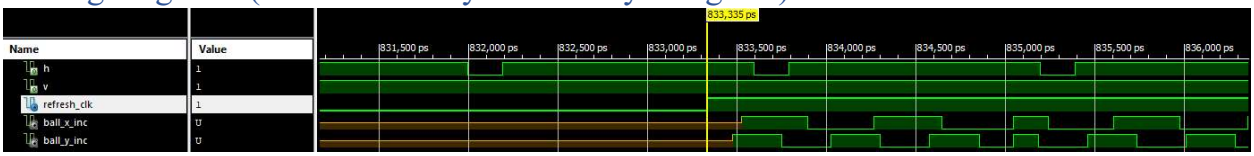

Figure 9: Waveform showing background being loaded after refresh clock switches to High

Figure 9 shows that the game has been initiated as soon as the refresh clock becomes 1. The 'x' and 'y' coordinates of the ball start changing at 833,335 ps.

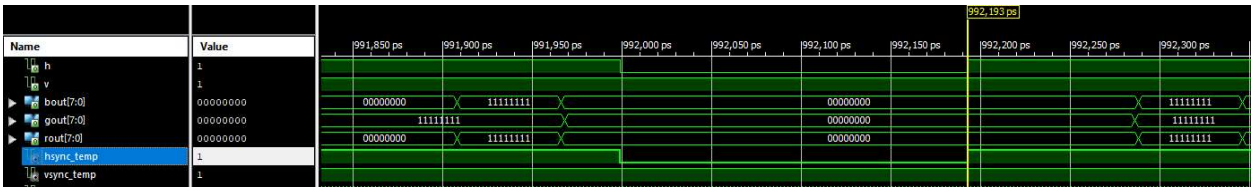

Figure 10: Waveform showing hsync changing to 1 after 96 clock cycles

Figure 10 shows after the hsync (sync pulse = 96 clock cycles) that the VGA monitor would output the next vertical line.

b) Screen captures of video game functioning, showing colors and screen design as per specification

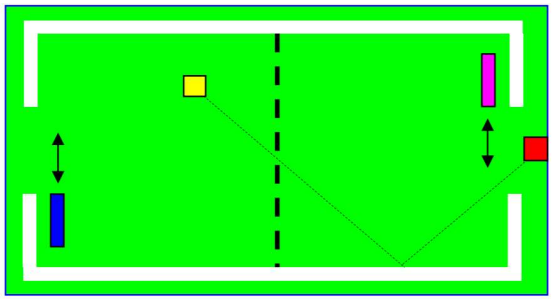

Figure 11: Static SVGP video-frame with dynamic elements– "ball" and "players" (Kirischian, 2020)

c) Brief explanation of results

Figure 9 shows the background loading as soon as the refresh clock hits 1. Figure 10 shows after the hsync (sync pulse = 96 clock cycles) that the VGA monitor would output the next vertical line.

6. Conclusions

Overall, the goal of the lab was to understand the use of pin assignments to display a "Pong" game on the VGA monitor using the Xillinx Spartan 3E-FPGA board. To reiterate, the sync, refresh, and VGA Controller components were used to create the "Pong" game.

The "Pong" game has been made through the understanding of how the VGA monitor functions in which the display creates an image by outputting a series of pixels horizontally across the screen and looping line by line vertically from top to bottom until the screen until the frame is complete. By using a clock, we can output a horizontal line of 640 pixels, with a guard band of 160 which includes the front porch, back porch and horizontal sync pulse. After a horizontal line of 800 pixels have been created, the pointer of the horizontal component would be reset, but the vertical component would be incremented by 1. After a display of 480 lines is complete, with a guard band of 45 which includes the front porch, back porch and vertical sync pulse, the complete 525 vertical lines would display one frame (Based on the pixel clock of 25mHz and a refresh rate of 60Hz). Therefore, the project of the *pong* game had been completed properly and thoroughly.

7. References (no references are considered academic misconduct)

Kirischian, L. (2020, September 09). Project #2 – Simple Video Game Processor for VGA. Retrieved from https://www.ee.ryerson.ca/~lkirisch/ele758/labs/SimpleVideoGame[11-11-11].pdf

Lagroix, H. E. P., Yanko, M. R., & Spalek, T. M. (2012). LCDs are better: Psychophysical and photometric estimates of the temporal characteristics of CRT and LCD monitors. Attention, Perception & Psychophysics, 74(5), 1033-1041.

Lowood, H. (2009). Videogames in computer space: The complex history of pong. IEEE Annals of the History of Computing, 31(3), 5-19.

8. Appendix with VHDL code (all blocks shall be included)

pingpong.vhd Mon Dec 07 14:47:39 2020

```
1   ------------------------Pong-----------------------------------------
2   ----------------------------------------------------------------------------------
3   -- Company:
4   -- Engineer: Brandon Ho & Vatsal Shreekant
5   --
6   -- Create Date:    17:10:28 11/22/2020
7   -- Design Name:
8   -- Module Name:    pingpong - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23  use IEEE.STD_LOGIC_ARITH.ALL;
24  use IEEE.STD_LOGIC_UNSIGNED.ALL;
25  use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if using
28  -- arithmetic functions with Signed or Unsigned values
29  --use IEEE.NUMERIC_STD.ALL;
30
31  -- Uncomment the following library declaration if instantiating
32  -- any Xilinx primitives in this code.
33  --library UNISIM;
34  --use UNISIM.VComponents.all;
35
36  entity pingpong is
37  port (
38            H : out  STD_LOGIC;
39            V : out  STD_LOGIC;
40            Bout : out  STD_LOGIC_VECTOR (7 downto 0);
41            Gout : out  STD_LOGIC_VECTOR (7 downto 0);
42            Rout : out  STD_LOGIC_VECTOR (7 downto 0)
43  );
44  end pingpong;
45
46  architecture Behavioral of pingpong is
47
48  --Signals
49
50  signal hsync_temp, vsync_temp : std_logic;
51
52  --Horizontal and Vertical Counters
53  signal hcounter : integer range 0 to 799;
54  signal vcounter : integer range 0 to 524;
55  --Pixel Clock and Refresh Clock
```

Page 1

```
 56   signal pixel_clk : std_logic;
 57   signal refresh_clk : std_logic;
 58   signal refresh_cntr : integer := 0;
 59   --R, G and B signals
 60   signal R, G, B : std_logic_vector (7 downto 0);
 61
 62   --Vertical bar 1
 63   signal top_border_x1: integer := 0;
 64   signal top_border_x2: integer := 30;
 65   signal top_border_y1: integer := 10;
 66   signal top_border_y2: integer := 160;
 67
 68   --Horizontal bar 1
 69   signal top_border_x3: integer := 0;
 70   signal top_border_x4: integer := 640;
 71   signal top_border_y3: integer := 0;
 72   signal top_border_y4: integer := 30;
 73
 74   --Vertical bar 2
 75   signal top_border_x5: integer := 610;
 76   signal top_border_x6: integer := 640;
 77   signal top_border_y5: integer := 0;
 78   signal top_border_y6: integer := 160;
 79
 80   --Vertical bar 3
 81   signal b_border_x1  : integer := 0;
 82   signal b_border_x2  : integer := 30;
 83   signal b_border_y1  : integer := 320;
 84   signal b_border_y2  : integer := 480;
 85
 86   --Horizontal bar 2
 87   signal b_border_x3  : integer := 0;
 88   signal b_border_x4  : integer := 640;
 89   signal b_border_y3  : integer := 450;
 90   signal b_border_y4  : integer := 480;
 91
 92   --Vertical bar 4
 93   signal b_border_x5  : integer := 610;
 94   signal b_border_x6  : integer := 640;
 95   signal b_border_y5  : integer := 320;
 96   signal b_border_y6  : integer := 480;
 97
 98   --Mid-field line
 99   signal m_line_x1    : integer := 318;
100   signal m_line_x2    : integer := 322;
101   signal m_line_y1    : integer := 30;
102   signal m_line_y2    : integer := 450;
103
104   --Center field white
105   signal c_border_x1: integer := 300;
106   signal c_border_x2: integer := 340;
107   signal c_border_y1: integer := 220;
108   signal c_border_y2: integer := 260;
109
110   --Center field green
```

```
111   signal cc_border_x1: integer := 305;
112   signal cc_border_x2: integer := 335;
113   signal cc_border_y1: integer := 225;
114   signal cc_border_y2: integer := 255;
115
116   --Paddle dimensions for the red paddle
117   signal r_paddle_x1   : integer := 590;
118   signal r_paddle_x2   : integer := 605;
119   signal r_paddle_y1   : integer := 200;
120   signal r_paddle_y2   : integer := 275;
121   signal r_paddle_x_inc: integer := 0;
122   signal r_paddle_y_inc: integer := 0;
123
124   --Paddle dimensions for the blue paddle
125   signal b_paddle_x1   : integer := 35;
126   signal b_paddle_x2   : integer := 50;
127   signal b_paddle_y1   : integer := 200;
128   signal b_paddle_y2   : integer := 275;
129   signal b_paddle_x_inc: integer := 0;
130   signal b_paddle_y_inc: integer := 0;
131
132   --Dimensions for the ball
133   signal ball_x1       : integer := 310;
134   signal ball_x2       : integer := 325;
135   signal ball_y1       : integer := 230;
136   signal ball_y2       : integer := 245;
137
138   --Goal lines for the red and blue sides
139   signal r_goal_x      : integer := 620;
140   signal b_goal_x      : integer := 20;
141   signal goal_y1       : integer := 160;
142   signal goal_y2       : integer := 320;
143
144   --Flags for score detection and reset
145   signal score         : std_logic;
146
147   --These Are to tell the ball to move left or right, depending if a boudnary is
      reached.
148   signal ball_x_inc    : std_logic;
149   signal ball_y_inc    : std_logic;
150
151   signal clk : std_logic:='0';
152
153   component Sync
154   Port (
155      pclk : out std_logic;
156      hcounter, vcounter : out integer);
157   end component;
158
159   component VGAcontroller
160   Port (
161      hcounter, vcounter : in integer;
162      r_paddle_x1, r_paddle_x2, r_paddle_y1, r_paddle_y2: in integer;
163      b_paddle_x1, b_paddle_x2, b_paddle_y2, b_paddle_y1: in integer;
164      ball_x1, ball_x2, ball_y1, ball_y2: in integer;
```

```
165       score : in std_logic;
166       R_out : out std_logic_vector(7 downto 0);
167       G_out : out std_logic_vector(7 downto 0);
168       B_out : out std_logic_vector(7 downto 0)
169
170    );
171    end component;
172
173    component refresh
174    Port (
175       pixclk : out std_logic;
176       refreshcount: out integer;
177       refreshclk : out std_logic);
178    end component;
179
180
181
182    begin
183
184
185    Sinc : Sync
186    PORT MAP (
187       pclk => pixel_clk,
188       hcounter => hcounter,
189       vcounter => vcounter
190       );
191
192    hsync_temp <= '0' when hcounter <= 96 else '1';
193    vsync_temp <= '0' when vcounter <= 2 else '1';
194
195    H <= hsync_temp;
196    v <= vsync_temp;
197
198    process(clk)
199    begin
200    clk <= not clk after 1ps;
201    --
202    --
203
204       if(hcounter >= 143 and hcounter <= 783 and vcounter >= 34 and vcounter <= 514)
       then
205          Rout <= R;
206          Gout <= G;
207          Bout <= B;
208
209       else
210          Rout <= (others => '0');
211          Gout <= (others => '0');
212          Bout <= (others => '0');
213       end if;
214    end process;
215
216    rfshclk : refresh
217    PORT MAP (
218       pixclk => pixel_clk,
```

```
219      refreshcount => refresh_cntr,
220      refreshclk => refresh_clk
221      );
222
223   process(clk)
224      begin
225         if clk'event and clk = '1' and refresh_clk ='1' then
226   --Check if Ball Hits Wall
227
228   --Hits left wall
229            if (ball_x1 <= top_border_x2 and (ball_y1 >= top_border_y4 and ball_y2<=
      top_border_y6)) then
230   --Ball hits top-left
231               ball_x_inc <= '1';
232
233            elsif(ball_x1 <= b_border_x2 and (ball_y1 >= b_border_y5 and ball_y2<=
      b_border_y6)) then
234   --Ball hits bottom-left
235               ball_x_inc <= '1';
236
237
238   --Hits right wall
239            elsif (ball_x2 >= top_border_x5 and (ball_y2 >= top_border_y4 and ball_y1
      <= top_border_y6)) then
240   --Ball hits top-right
241               ball_x_inc <= '0';
242
243            elsif (ball_x2 >= b_border_x5 and (ball_y2 >= b_border_y1 and ball_y1<=
      b_border_y6)) then
244   --Ball hits bottom-right
245               ball_x_inc <= '0';
246
247            end if;
248
249   --Ball hits paddle
250            if (ball_x1 <= b_paddle_x2 and (ball_y1 >= b_paddle_y1 and ball_y2 <=
      b_paddle_y2)) then
251   --Ball hits left paddle
252                  ball_x_inc <= '1';
253
254            elsif (ball_x2 >= r_paddle_x1 and (ball_y1 >= r_paddle_y1 and ball_y2 <=
      r_paddle_y2)) then
255   --Ball hits right paddle
256               ball_x_inc <= '0';
257
258            end if;
259
260            if (ball_y1 <= top_border_y4) then
261   --Ball hits top wall
262               ball_y_inc <= '0';
263
264            elsif (ball_y2 >= b_border_y3) then
265   --Ball hits bottom wall
266               ball_y_inc <= '1';
267
```

```
268             end if;
269
270     --Goals
271             if (ball_x1 < b_goal_x and ball_y1 >= goal_y1 and ball_y2 <= goal_y2) then
272     --Ball scores in left goal
273                 score <= '1';
274
275             elsif (ball_x2 > r_goal_x and ball_y1 >= goal_y1 and ball_y2 <= goal_y2)
        then
276     --Ball scored in right goal
277                 score <= '1';
278
279             else
280     --No Goal
281                 score <= '0';
282             end if;
283
284             if (ball_x1 <= 0) then
285     --Reset Ball after goal on left
286                 ball_x1 <= 310;
287                 ball_x2 <= 325;
288                 ball_y1 <= 230;
289                 ball_y2 <= 245;
290                 ball_x_inc <= '1';
291                 ball_y_inc <= '1';
292             elsif (ball_x2 >= 640) then
293     --Reset Ball after goal on right
294                 ball_x1 <= 310;
295                 ball_x2 <= 325;
296                 ball_y1 <= 230;
297                 ball_y2 <= 245;
298                 ball_x_inc <= '0';
299                 ball_y_inc <= '1';
300             else
301     --Ball movement
302                 if (ball_x_inc = '1') then
303     --Ball in positive x direction
304                     ball_x1 <= ball_x1 + 3;
305                     ball_x2 <= ball_x2 + 3;
306                 else
307     --Ball in negative x direction
308                     ball_x1 <= ball_x1 - 3;
309                     ball_x2 <= ball_x2 - 3;
310
311                  end if;
312                 if ( ball_y_inc = '1') then
313     --Ball in positive y direction
314                     ball_y1 <= ball_y1 - 3;
315                     ball_y2 <= ball_y2 - 3;
316                 else
317     --Ball in negative y direction
318                     ball_y1 <= ball_y1 + 3;
319                     ball_y2 <= ball_y2 + 3;
320                 end if;
321             end if;
```

```
322         end if;
323     end process;
324
325     VGACon : VGAcontroller
326     PORT MAP (
327        hcounter => hcounter,
328        vcounter => vcounter,
329        r_paddle_x1 => r_paddle_x1,
330        r_paddle_x2 => r_paddle_x2,
331        r_paddle_y1 => r_paddle_y1,
332        r_paddle_y2 => r_paddle_y2,
333        b_paddle_x1 => b_paddle_x1,
334        b_paddle_x2 => b_paddle_x2,
335        b_paddle_y2 => b_paddle_y2,
336        b_paddle_y1 => b_paddle_y1,
337        ball_x1 => ball_x1,
338        ball_x2 => ball_x2,
339        ball_y1 => ball_y1,
340        ball_y2 => ball_y2,
341        score => score,
342        R_out => R,
343        G_out => G,
344        B_out => B
345     );
346
347     --hcount_temp <= to_unsigned(hcounter, hcount_temp);
348     --vcount_temp <=
349
350     end Behavioral;
```

```
1   ------------------------Sync-----------------------------------------------------------
    ----
2   ----------------------------------------------------------------------------------
3   -- Company:
4   -- Engineer: Brandon Ho & Vatsal Shreekant
5   --
6   -- Create Date:    17:10:28 11/22/2020
7   -- Design Name:
8   -- Module Name:    Sync - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23  use IEEE.STD_LOGIC_ARITH.ALL;
24  use IEEE.STD_LOGIC_UNSIGNED.ALL;
25  use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if using
28  -- arithmetic functions with Signed or Unsigned values
29  --use IEEE.NUMERIC_STD.ALL;
30
31  -- Uncomment the following library declaration if instantiating
32  -- any Xilinx primitives in this code.
33  --library UNISIM;
34  --use UNISIM.VComponents.all;
35
36  entity Sync is
37  port (
38  pclk : out std_logic;
39  hcounter, vcounter : out integer);
40  end Sync;
41
42  architecture Behavioral of Sync is
43
44  signal hcount : integer:=0;
45  signal vcount : integer:=0;
46
47  signal clk: std_logic:='0';
48
49  begin
50  process (clk)
51     begin
52        clk <= not clk after 1 ps;
53        if clk'event and clk = '1' then
54  -- horizontal counts from 0 to 799
```

```
55            hcount <= hcount+1;
56            if (hcount = 799) then
57               vcount <= vcount+1;
58               hcount <= 0;
59            end if;
60    -- vertical counts from 0 to 524
61            if (vcount = 524) then
62               vcount <= 0;
63            end if;
64         end if;
65
66    end process;
67    hcounter <= hcount;
68    vcounter <= vcount;
69    pclk <= clk;
70    end Behavioral;
```

refresh.vhd Sun Dec 06 21:41:42 2020

```
1   ----------------------------------------------------Refresh-------------------------
    -------------------------
2   ----------------------------------------------------------------------------------
3   -- Company:
4   -- Engineer: Brandon Ho & Vatsal Shreekant
5   --
6   -- Create Date:    17:10:28 11/22/2020
7   -- Design Name:
8   -- Module Name:    refresh - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity refresh is
34  port (
35     pixclk : out std_logic;
36     refreshcount: out integer;
37     refreshclk : out std_logic);
38
39  end refresh;
40
41  architecture Behavioral of refresh is
42  signal refresh_clk : std_logic :='0';
43  signal refreshcnt : integer:= 0;
44
45  signal clk: std_logic:='0';
46
47  begin
48  process(clk)
49     begin
50     clk <= not clk after 1ps;
51        if clk'event and clk='1' then
52           if (refreshcnt >= 416667) then
53              refresh_clk <= not(refresh_clk);
54              refreshcnt <= 0;
```

Page 1

```
55            else
56               refreshcnt <= refreshcnt + 1;
57            end if;
58
59         end if;
60   end process;
61
62   refreshcount <= refreshcnt;
63   refreshclk <= refresh_clk;
64   pixclk <= clk;
65   end Behavioral;
66
67
```

VGAcontroller.vhd Sun Dec 06 21:42:16 2020

```
1   ----------------------------------------------VGA
    Controller-----------------------------------------------------
2   ----------------------------------------------------------------------------------
3   -- Company:
4   -- Engineer: Brandon Ho & Vatsal Shreekant
5   --
6   -- Create Date:    17:10:28 11/22/2020
7   -- Design Name:
8   -- Module Name:    VGAcontroller - Behavioral
9   -- Project Name:
10  -- Target Devices:
11  -- Tool versions:
12  -- Description:
13  --
14  -- Dependencies:
15  --
16  -- Revision:
17  -- Revision 0.01 - File Created
18  -- Additional Comments:
19  --
20  ----------------------------------------------------------------------------------
21  library IEEE;
22  use IEEE.STD_LOGIC_1164.ALL;
23  use IEEE.STD_LOGIC_ARITH.ALL;
24  use IEEE.STD_LOGIC_UNSIGNED.ALL;
25  use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if using
28  -- arithmetic functions with Signed or Unsigned values
29  --use IEEE.NUMERIC_STD.ALL;
30
31  -- Uncomment the following library declaration if instantiating
32  -- any Xilinx primitives in this code.
33  --library UNISIM;
34  --use UNISIM.VComponents.all;
35
36  entity VGAcontroller is
37  port (
38     --refclk: in std_logic;
39     hcounter, vcounter : in integer;
40     r_paddle_x1, r_paddle_x2, r_paddle_y1, r_paddle_y2: in integer;
41     b_paddle_x1, b_paddle_x2, b_paddle_y2, b_paddle_y1: in integer;
42     ball_x1, ball_x2, ball_y1, ball_y2: in integer;
43     score : in std_logic;
44     R_out : out std_logic_vector(7 downto 0);
45     G_out : out std_logic_vector(7 downto 0);
46     B_out : out std_logic_vector(7 downto 0)
47
48  );
49  end VGAcontroller;
50
51  architecture Behavioral of VGAcontroller is
52
53  --Vertical bar 1
54  signal top_border_x1: integer := 0;
```

Page 1

```
55    signal top_border_x2: integer := 30;
56    signal top_border_y1: integer := 10;
57    signal top_border_y2: integer := 160;
58
59    --Horizontal bar 1
60    signal top_border_x3: integer := 0;
61    signal top_border_x4: integer := 640;
62    signal top_border_y3: integer := 0;
63    signal top_border_y4: integer := 30;
64
65    --Vertical bar 2
66    signal top_border_x5: integer := 610;
67    signal top_border_x6: integer := 640;
68    signal top_border_y5: integer := 0;
69    signal top_border_y6: integer := 160;
70
71    --Vertical bar 3
72    signal b_border_x1  : integer := 0;
73    signal b_border_x2  : integer := 30;
74    signal b_border_y1  : integer := 320;
75    signal b_border_y2  : integer := 480;
76
77    --Horizontal bar 2
78    signal b_border_x3  : integer := 0;
79    signal b_border_x4  : integer := 640;
80    signal b_border_y3  : integer := 450;
81    signal b_border_y4  : integer := 480;
82
83    --Vertical bar 4
84    signal b_border_x5  : integer := 610;
85    signal b_border_x6  : integer := 640;
86    signal b_border_y5  : integer := 320;
87    signal b_border_y6  : integer := 480;
88
89    --Mid-field line
90    signal m_line_x1    : integer := 320;
91    signal m_line_x2    : integer := 320;
92    signal m_line_y1    : integer := 30;
93    signal m_line_y2    : integer := 450;
94    begin
95
96    process(hcounter, vcounter)
97
98       variable x: integer range 0 to 639;
99       variable y: integer range 0 to 479;
100   begin
101
102   --To isolate the active region, we subtract the number of cycles it takes
103   --for H-sync and V-sync to reach their respective active regions and place
104   --the values into x and y coordinates.This helps to intuitively determine
105   --the placement of objects on the physical screen
106      x := hcounter - 143;
107      y := vcounter - 34;
108
109   --Every pixel that isn't an object on the screen is set to display green
```

```
110       R_out <= "00000000";
111       G_out <= "11111111";
112       B_out <= "00000000";
113
114    --Ball
115       if (x > ball_x1 and x < ball_x2 and y > ball_y1 and y < ball_y2) then
116             --Changing the ball colour to red when either side has scored
117          if (score = '1') then
118             R_out <= "11111111";
119             G_out <= "00000000";
120             B_out <= "00000000";
121          else
122             R_out <= "11111111";
123             G_out <= "11111111";
124             B_out <= "00000000";
125          end if;
126    --Boundaries of the field
127       elsif ( (x > top_border_x1 and x < top_border_x2 and y > top_border_y1 and y <
       top_border_y2) or (x > top_border_x3 and x < top_border_x4 and y > top_border_y3
       and y < top_border_y4) or (x > top_border_x5 and x < top_border_x6 and y >
       top_border_y5 and y < top_border_y6) or (x > b_border_x1 and x < b_border_x2 and y
        > b_border_y1 and y < b_border_y2) or (x > b_border_x3 and x < b_border_x4 and y
       > b_border_y3 and y < b_border_y4) or (x > b_border_x5 and x < b_border_x6 and y >
        b_border_y5 and y < b_border_y6)) then
128          R_out <= "11111111";
129          G_out <= "11111111";
130          B_out <= "11111111";
131
132       elsif (x > m_line_x1 and x < m_line_x2 and y > m_line_y1 and y < m_line_y2) then
133          R_out <= "11111111";
134          G_out <= "11111111";
135          B_out <= "11111111";
136
137
138    --Paddles
139    --Purple paddle
140       elsif (x > r_paddle_x1 and x < r_paddle_x2 and y > r_paddle_y1 and y <
       r_paddle_y2) then
141          R_out <= "11111111";
142          G_out <= "00000000";
143          B_out <= "11111111";
144    --Blue paddle
145       elsif (x > b_paddle_x1 and x <b_paddle_x2 and y > b_paddle_y1 and y <
       b_paddle_y2) then
146          R_out <= "00000000";
147          G_out <= "00000000";
148          B_out <= "11111111";
149       end if;
150
151    end process;
152
153    end Behavioral;
```