ECE 238: Pong Report

Due on Friday, December 5, 2014

Ramiro Jordan 9:30am

Ricardo Piro-Rael & Dillon Thomas

Contents

VHDL Pong Report

Introduction

Within, I have embedded all the VHDL Code. I have sectioned it off.

Top Module

Top VHDL Code

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    10:58:12 11/03/2014
-- Design Name:
-- Module Name:    Top - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.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 Top is
    Port ( Clk,Reset, pause : in  STD_LOGIC;
              Padle: in std_logic_vector (1 downto 0);
                  Hsync, Vsync : out  STD_LOGIC;
                  KeyBoardClock : in std_logic;
                KeyBoardData : in std_logic;
           Ax : out  STD_LOGIC_VECTOR (3 downto 0);
           Seg : out  STD_LOGIC_VECTOR (7 downto 0);
                  rgb : out std_logic_vector(2 downto 0));
end Top;

architecture Behavioral of Top is
component vga_sync is
```

```
45  port(
46            clk, reset : in std_logic;
47            hsync, vsync : out std_logic;
48            video_on, p_tick : out std_logic;
49            pixel_x, pixel_y : out std_logic_vector(9 downto 0) -- Coordinates on screen
50       );
51  end component;
52
53  component paddle is
54  port(
55            clk, reset : in std_logic;
56            btn : in std_logic_vector(1 downto 0);
57            pause : in std_logic;
58            location : in std_logic_vector(9 downto 0);
59            paddle_on : out std_logic;
60            pixel_x, pixel_y : in std_logic_vector(9 downto 0);
61            paddle_rgb : out std_logic_vector(2 downto 0)
62       );
63  end component;
64
65  component KeyboardController is
66      Port ( Clock : in STD_LOGIC;
67               KeyboardClock : in  STD_LOGIC;
68             KeyboardData : in  STD_LOGIC;
69             LeftPaddleDirection : buffer  integer;
70             RightPaddleDirection : buffer  integer
71       );
72  end component;
73
74  component Walls is
75      Port ( Pixel_x, Pixel_Y : in  STD_LOGIC_VECTOR (9 downto 0);
76             Video_on : in std_logic;
77                   Wall_on : out  STD_LOGIC;
78             Wall_RGB : out  STD_LOGIC_VECTOR (2 downto 0));
79  end component;
80
81
82  component ball is
83  port(
84            clk, reset : in std_logic;
85            pause : in std_logic;
86            pedal_on, paddle2_on : in std_logic;
87            pixel_x, pixel_y : in std_logic_vector(9 downto 0);
88            p1score, p2score: out std_logic;
89            ball_on : out std_logic;
90            ball_rgb : out std_logic_vector(2 downto 0)
91       );
92  end component;
93
94
95  component Letter is
96      Port ( X_Sync, Y_Sync : in  STD_LOGIC_VECTOR (9 downto 0);
97             Letter_on : out  STD_LOGIC;
```

```
98            Letter_RGB : out  STD_LOGIC_vector ( 2 downto 0);
99            Video_on,Clk : in  STD_LOGIC);
100 end component;
101 
102 component Koopa is
103     Port ( X_Sync, Y_Sync : in  STD_LOGIC_VECTOR (9 downto 0);
104            Letter_on : out  STD_LOGIC;
105            Letter_RGB : out  STD_LOGIC_vector ( 2 downto 0);
106            Video_on,Clk : in  STD_LOGIC);
107 end component;
108 
109 component Seven_Segment_Disp is
110     Port ( Clk, Reset: in  STD_LOGIC;
111            Ax : out  STD_LOGIC_VECTOR (3 downto 0);
112            Seg : out  STD_LOGIC_VECTOR (7 downto 0);
113            Player_1, Player_2 : in  STD_LOGIC);
114 end component;
115 
116 --signal pause, state_pause : std_logic;
117 
118 signal RGB_Next, rgb_reg, Background_RGB_X, Background_RGB_Y, Background_RGB,Letter_RGB, Koopa_R
119 signal X, Y :  std_logic_vector(9 downto 0):="0000000000";
120 signal video, pixel_tick, Paddle_on, pedal_on, Wall_on, Ball_on, Letter_on, Koopa_on, Wall_Top,
121 
122 signal btn : std_logic_vector(1 downto 0);
123 signal LeftPaddleDirection : integer;
124 signal RightPaddleDirection : integer;
125 signal btn0, btn1 : std_logic_vector(1 downto 0);
126 
127 ----------------------------------------------------------
128 ----------------------------------------------------------
129 begin
130 btn <= std_logic_vector(to_unsigned(LeftPaddleDirection,2));
131 --pause <= state_pause or sys_pause;
132 
133 VGA: vga_sync port map (Clk, Reset, Hsync, Vsync,Video, pixel_tick, X, Y);
134 Peddle: paddle port map (Clk, Reset,Padle ,Pause, "1001011000", Paddle_on, X, Y, Paddle_RGB); --
135 Pedal: paddle port map (Clk, Reset,btn ,Pause, "0000101000", pedal_on, X, Y, Paddle_RGB); --- pe
136 Background: Walls port map (X, Y, Video, Wall_on, Wall_RGB);
137 Back_ball: ball port map (clk, reset, pause, pedal_on, Paddle_on, X, Y,Wall_Top, Wall_Bottom, Ba
138 Letter_V: Letter port map (X,Y, Letter_on,Letter_RGB, Video, Clk);
139 Koopa_V: Koopa port map (X,Y, Koopa_on,Koopa_RGB, Video, Clk);
140 
141 keyboard : KeyboardController
142      port map (
143      Clock => clk,
144      KeyboardClock => KeyboardClock,
145      KeyboardData => KeyboardData,
146      LeftPaddleDirection => LeftPaddleDirection,
147      RightPaddleDirection => RightPaddleDirection
148      );
149 Score: Seven_Segment_Disp port map ( Clk, Reset, AX,Seg,Wall_Top, Wall_Bottom);
150 --Led (0) <= Wall_Top;
```

```
151 --Led (1) <= Wall_Bottom;
152
153 process (Video,  Paddle_on, Paddle_RGB, Wall_on, Wall_RGB, Ball_on, Ball_RGB, Letter_on,Letter_R
    is
154 begin
155 if(video='1') then
156
157  if (Paddle_on ='1') then
158   RGB_Next <= Paddle_RGB;
159
160    elsif (Letter_on ='1') then
161   RGB_Next <= Letter_RGB;
162
163    elsif (Koopa_on ='1') then
164   RGB_Next <= Koopa_RGB;
165
166  elsif (Ball_on ='1') then
167   RGB_Next <= Ball_RGB;
168
169  elsif (Wall_on ='1') then
170   RGB_Next <= Wall_RGB;
171
172  elsif (pedal_on ='1') then
173   RGB_Next <= Paddle_RGB;
174
175   else
176
177   RGB_Next <="000";
178  end if;
179
180 else
181
182 RGB_Next <="000";
183
184 end if;
185 end process;
186
187 Process (clk) is
188 begin
189
190  if ( Rising_Edge(Clk)) then
191
192  if (pixel_tick='1') then
193
194  rgb_reg <= RGB_Next;
195
196
197  end if ;
198
199 end if;
200 end process;
201
202
```

```
203
204 RGB <= rgb_reg;
205
206
207 end Behavioral;
```

Walls

Walls VHDL Code

```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    10:12:06 11/10/2014
6  -- Design Name:
7  -- Module Name:    Walls - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity Walls is
33     Port ( Pixel_x, Pixel_Y : in  STD_LOGIC_VECTOR (9 downto 0);
34            Video_on : in std_logic;
35                  Wall_on : out  STD_LOGIC;
36            Wall_RGB : out  STD_LOGIC_VECTOR (2 downto 0));
37 end Walls;
38
39 architecture Behavioral of Walls is
40 Signal Wall_B_On, Wall_T_On, Wall: std_logic:='0'; --Wall_L_On
41 signal Wall_T_RGB, Wall_B_RGB : std_logic_vector ( 2 downto 0):="000";-- Wall_L_RGB,
42 signal X, Y :  std_logic_vector(9 downto 0):="0000000000";
43 constant Letter_L : integer := 20; --Letter Starting Point
```

```
Constant Letter_T : integer := 2;    -- Letter Top Starting Pont

begin
X <= Pixel_x;
Y <= Pixel_Y;

Wall_RGB <="001" when ((Wall = '1')and (video_on='1')) else "000";

Wall <= Wall_T_On or Wall_B_On;-- or Wall_L_On ;
Wall_on <= Wall;
----- Top Wall ----
Wall_T_On <= '1' when ((Y < "0000110010") and (video_on='1')) else '0';

----- Bottom Wall ----
Wall_B_On <= '1' when ((Y > "0110101110") and (Y<="1000001011") and (video_on='1')) else '0';
----- Left Wall ----
--Wall_L_On <= '1' when ((X < "0000010101")  and (video_on='1')) else '0';


end Behavioral;
```

Paddle

Paddle VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity paddle is
port(
          clk, reset : in std_logic;
          btn : in std_logic_vector(1 downto 0);
          pause : in std_logic;
          location : in std_logic_vector(9 downto 0);
          paddle_on : out std_logic;
          pixel_x, pixel_y : in std_logic_vector(9 downto 0);
          paddle_rgb : out std_logic_vector(2 downto 0)
     );
end paddle;

architecture paddle_arch of paddle is

     -- Signal used to control how
     -- often pushbuttons are checked for paddle movement.
     signal refr_tick: std_logic;

     -- x, y coordinates (0,0 to (639, 479)
     signal pix_x, pix_y: unsigned(9 downto 0);

     -- screen dimensions
     constant MAX_X: integer := 640;
     constant MAX_Y_T: integer := 50;
```

```
29      constant MAX_Y_B: integer := 430;
30
31      -- paddle left, right, top, bottom and height left &
32      -- right are constant. top & bottom are signals to
33      -- allow movement. bar_y_t driven by reg below.
34      signal bar_x_l: unsigned(9 downto 0);
35      signal bar_x_r: unsigned(9 downto 0);
36      signal bar_y_t, bar_y_b: unsigned(9 downto 0);
37      constant BAR_Y_SIZE: integer := 72;
38
39      -- reg to track top boundary (x position is fixed)
40      signal bar_y_reg, bar_y_next: unsigned(9 downto 0):= "0011110000";
41
42      -- bar moving velocity when a button is pressed
43      -- the amount the bar is moved.
44      constant BAR_V: integer:= 4;
45
46  begin
47
48      process (clk, reset)
49      begin
50           if (reset = '1') then
51                bar_y_reg <= "0011110000";
52           elsif (clk'event and clk = '1') then
53                bar_y_reg <= bar_y_next;
54           end if;
55      end process;
56
57      bar_x_l <= unsigned(location);
58      bar_x_r <= unsigned(location) + 3;
59
60      pix_x <= unsigned(pixel_x);
61      pix_y <= unsigned(pixel_y);
62
63      -- refr_tick: 1-clock tick asserted at start of v_sync,
64      -- e.g., when the screen is refreshed -- speed is 60 Hz
65      refr_tick <= '1' when (pix_y = 481) and (pix_x = 0) and (pause = '0') else
66                           '0';
67
68      -- pixel within paddle
69      bar_y_t <= bar_y_reg;
70      bar_y_b <= bar_y_t + BAR_Y_SIZE - 1;
71
72      paddle_on <= '1' when (bar_x_l <= pix_x) and (pix_x <= bar_x_r) and
73                                    (bar_y_t <= pix_y) and (pix_y <= bar_y_b) else
74                           '0';
75
76      paddle_rgb <= "010";
77
78      -- Process bar movement requests
79      process( bar_y_reg, bar_y_b, bar_y_t, refr_tick, btn)
80      begin
81           bar_y_next <= bar_y_reg; -- no move
```

```
82
83          if ( refr_tick = '1' ) then
84              -- if btn 0 pressed and paddle not at bottom yet
85              if ( btn(0) = '1' and bar_y_b < (MAX_Y_B - 1 - BAR_V)) then
86                  bar_y_next <= bar_y_reg + BAR_V;
87              -- if btn 1 pressed and bar not at top yet
88              elsif ( btn(1) = '1' and (bar_y_t - 50) > BAR_V) then
89                  bar_y_next <= bar_y_reg - BAR_V;
90              end if;
91          end if;
92      end process;
93
94
95
96  end paddle_arch;
```

Ball

Ball VHDL Code

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity ball is
6   port(
7             clk, reset : in std_logic;
8             pause : in std_logic;
9             pedal_on, paddle2_on : in std_logic;
10            pixel_x, pixel_y : in std_logic_vector(9 downto 0);
11            p1score, p2score: out std_logic;
12          ball_on : out std_logic;
13          ball_rgb : out std_logic_vector(2 downto 0)
14      );
15  end ball;
16
17  architecture ball_arch of ball is
18
19      -- Signal used to control speed of ball and how
20      -- often pushbuttons are checked for paddle movement.
21      signal refr_tick: std_logic;
22
23      -- x, y coordinates (0,0 to (639, 479)
24      signal pix_x, pix_y: unsigned(9 downto 0);
25
26      -- screen dimensions
27      constant MAX_X: integer := 640;
28      constant MAX_Y_T: integer := 50;
29      constant MAX_Y_B: integer := 430;
30      Constant Max_X_L: integer := 20;
31
32      -- square ball -- ball left, right, top and bottom
33      -- all vary. Left and top driven by registers below.
```

```
34      constant BALL_SIZE: integer := 8;
35      signal ball_x_l, ball_x_r: unsigned(9 downto 0):=(others => '0');
36      signal ball_y_t, ball_y_b: unsigned(9 downto 0):=(others => '0');
37
38      -- reg to track left and top boundary
39      signal ball_x_reg, ball_x_next: unsigned(9 downto 0):=(others => '0');
40      signal ball_y_reg, ball_y_next: unsigned(9 downto 0):=(others => '0');
41
42      -- reg to track ball speed
43      signal x_delta_reg, x_delta_next : unsigned(9 downto 0):=(others => '0');
44      signal y_delta_reg, y_delta_next : unsigned(9 downto 0):=(others => '0');
45
46      -- ball movement can be pos or neg
47      constant BALL_V_P: unsigned(9 downto 0) := to_unsigned(2,10);
48      constant BALL_V_N: unsigned(9 downto 0) := unsigned(to_signed(-2,10));
49
50      -- round ball image
51      type rom_type is array(0 to 7) of std_logic_vector(0 to 7);
52      constant BALL_ROM: rom_type:= (
53           "00111100",
54           "01111110",
55           "11111111",
56           "11111111",
57           "11111111",
58           "11111111",
59           "01111110",
60           "00111100"
61      );
62
63      signal rom_addr, rom_col: unsigned(2 downto 0):= "000";
64      signal rom_data: std_logic_vector(7 downto 0):=(others => '0');
65      signal rom_bit: std_logic:='0';
66
67      -- object output signals -- new signal to indicate if
68      -- scan coord is within ball
69      signal sq_ball_on, rd_ball_on : std_logic:='0';
70      -- ================================================
71
72
73      signal scored : std_logic:='0';
74  begin
75
76
77
78      process (clk, reset)
79      begin
80           if (reset = '1') then
81                ball_x_reg <= "0100111011";
82                ball_y_reg <= "0011110000";
83                x_delta_reg <= "0000000010";
84                y_delta_reg <= "0000000010";
85
86           elsif (clk'event and clk = '1') then
```

```
87                ball_x_reg <= ball_x_next;
88                ball_y_reg <= ball_y_next;
89                x_delta_reg <= x_delta_next;
90                y_delta_reg <= y_delta_next;
91
92            end if;
93        end process;
94
95        pix_x <= unsigned(pixel_x);
96        pix_y <= unsigned(pixel_y);
97
98        -- refr_tick: 1-clock tick asserted at start of v_sync,
99        -- e.g., when the screen is refreshed -- speed is 60 Hz
100       refr_tick <= '1' when (pix_y = 481) and (pix_x = 0) and (pause = '0')else
101                           '0';
102
103       -- set coordinates of square ball.
104       ball_x_l <= ball_x_reg;
105       ball_y_t <= ball_y_reg;
106       ball_x_r <= ball_x_l + BALL_SIZE - 1;
107       ball_y_b <= ball_y_t + BALL_SIZE - 1;
108
109       -- pixel within square ball
110       sq_ball_on <= '1' when (ball_x_l <= pix_x) and (pix_x <= ball_x_r) and
111                                              (ball_y_t <= pix_y) and (pix_y <= ball_y_b) else
112                             '0';
113
114       -- map scan coord to ROM addr/col -- use low order three
115       -- bits of pixel and ball positions.
116       -- ROM row
117       rom_addr <= pix_y(2 downto 0) - ball_y_t(2 downto 0);
118       -- ROM column
119       rom_col <= pix_x(2 downto 0) - ball_x_l(2 downto 0);
120       -- Get row data
121       rom_data <= BALL_ROM(to_integer(rom_addr));
122       -- Get column bit
123       rom_bit <= rom_data(to_integer(rom_col));
124
125       -- Turn ball on only if within square and ROM bit is 1.
126       rd_ball_on <= '1' when (sq_ball_on = '1') and (rom_bit = '1') else
127                             '0';
128       ball_rgb <= "100"; -- red
129       ball_on <= rd_ball_on;
130
131       -- Update the ball position 60 times per second.
132       ball_x_next <= "0100111011" when scored = '1' else
133                                 ball_x_reg + x_delta_reg when refr_tick = '1' else
134                                 ball_x_reg;
135       ball_y_next <= ball_y_reg + y_delta_reg when refr_tick = '1' else
136                                 ball_y_reg;
137
138       --player_scored <= scored;
139
```

```
140     -- Set the value of the next ball position according to
141     -- the boundaries.
142     process(x_delta_reg, y_delta_reg, ball_y_t, ball_x_l, ball_x_r,
143                 ball_y_t, ball_y_b, pedal_on, paddle2_on, rd_ball_on)
144     begin
145          x_delta_next <= x_delta_reg;
146          y_delta_next <= y_delta_reg;
147          scored <= '0';
148               p1score <= '0';
149               p2score <= '0';
150          -- ball reached top, make offset positive
151          if ( ball_y_t < MAX_Y_T - 1 ) then
152               y_delta_next <= BALL_V_P;
153          -- reached bottom, make negative
154          elsif (ball_y_b > (MAX_Y_B - 1)) then
155               y_delta_next <= BALL_V_N;
156          -- Hit left paddle
157          elsif(pedal_on = '1' and rd_ball_on = '1') then
158               x_delta_next <= BALL_V_P;
159          -- Hit right paddle
160          elsif(paddle2_on = '1' and rd_ball_on = '1') then
161               x_delta_next <= BALL_V_N;
162          elsif(ball_x_r > (MAX_X - 8)) then
163              scored <= '1';
164                p1score <= '1';
165          elsif(ball_x_l <  MAX_X_L) then
166              scored <= '1';
167                p2score <= '1';
168 --                 x_delta_next <= BALL_V_N;
169          end if;
170     end process;
171
172
173
174 end ball_arch;
```

Koopa Image

Koopa Image VHDL Code

```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    11:42:49 11/11/2014
6  -- Design Name:
7  -- Module Name:    Letter - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
```

```vhdl
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

-- Uncomment the following library declaration if usingw
-- 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 Koopa is
    Port ( X_Sync, Y_Sync : in  STD_LOGIC_VECTOR (9 downto 0);
           Letter_on : out  STD_LOGIC;
           Letter_RGB : out  STD_LOGIC_vector ( 2 downto 0);
           Video_on, Clk : in  STD_LOGIC);
end Koopa;

architecture Behavioral of Koopa is
Constant Rom_Row: integer := 6; --- Row_Columns can be represented by how many bits
 (2^(6+1) = 128 >102
Constant Rom_Column: integer := 102; --- Number of Row in your Rom
Constant Heigh_Rom : integer := 35;--- Number of Column in your Rom

type rom_type is array (0 to 35) of std_logic_vector ( Rom_Column downto 0);
constant Letter_ROM: rom_type:= (
"0000000000000011111111000000000000000000000000000000000000000000000000000000000000000000000000
"0000000000000111111111000000000000000000000000000000000000000000000000000000000000000000000000
"0000000000001111111111111000000000000000000000000000000000000000000000000000000000000000000000
"0000000000011111111111111000000000000000000000000000000000000000000000000000000000000000000000
"0000000001011111111111111100000000000000000000000000000000000000000000000000000000000000000000
"0000000001111111111111111110000000000000000000000000000000000000000000000000000000000000000000
"0000000001111111111111111110000000000000000000000000000000000111111111111000000000000000011111 1
"0000000111111111111111111111100000000000000000000000000000111111111111110000000000001111111111
"0000000111111111111111111111110000000000000000000000000001111111111111110000000000111111111111
"0000111111111111111111111111111000000000000000000000000011111000000011110000000001111111000000
"0000011111111111111111111111111000000000000000000000000111110000000011110000000011111000000000
"0011111110011111111111110011111110000000000000000000000111100000000011110000000011110000000000
"0001111110011111111111110011111110000000000000000000000111100000000011110000000111110000000000
"0011111110011111111111110011111110000000000000000000000111100000000011110000000111100000000000
"0011111110011111111111110011111110000000000000000000000111100000000011110000000111100000000000
"0011111110011111111111110011111110000000000000000000000111100000000011110000001111000000000000
"1111111110011000111000110011111111100000000000000000000011110000000011110000001111000000000000
"1111111110011000111001110011111111100000000000000000000011111000000011110000001111000000000000
"1111111110000000111000000011111111100000000000000000000011111111111111110000001111000000000000
```

```
"11111111100000001111000000011111111100000000000000000000000011111111111100000011110000000000
"11111111111110001111111000111111111110000000000000000000000011111111111100000011110000000000
"11111111111110001111111000111111111110000000000000000000000011111100001111000000111100000000
"10111111111010001010101001111111111000000000000000000000000011110000001111000000111100000000
"00111111111000000000000001111111110000000000000000000000000111100000001111000000011100000000
"00011111111000000000000001111111110000000000000000000000000111100000001111000000011100000000
"00000000000000000000000000000000000000000000000000000000000011110000000011110000000111100000
"00000000000000000000000000000000000000000000000000000000000011111000000000111100000001111000
"00000111100000000000000000000000000000000000000000000000000011110000000001111000000000111100
"00001111100000000000000000000000000000000000000000000000000011110000000001111000000000111110
"00111111111110000000000011110000000000000000000000000011111100000000000111100000000000111111
"00011111111110000000000011110000000000000000000000000011111000000000000111100000000000111111
"00111111111111010000001011110000000000000000000000000011111000000000000111100000000000001011
"00111111111111110000001111110000000000000000000000000000000000000000000000000000000000000000
"00111111111111110000001111110000000000000000000000000000000000000000000000000000000000000000
"00000111111111110000111111100000000000000000000000000000000000000000000000000000000000000000
"00000111111111110000111111100000000000000000000000000000000000000000000000000000000000000000
```

```
--   constant Letter_ROM: rom_type:= (
--   "00000000000000000000000000",
--   "01111100000000000001111000",
--   "01111100000000000011111000",
--   "01111100000000000011111000",
--   "01111100000000000011110000",
--   "00111110000000000111110000",
--   "00111110000000000111110000",
--   "00111110000000000111100000",
--   "00011111000000000111100000",
--   "00011111000000000111100000",
--   "00011111000000000111000000",
--   "00001111100000001111000000",
--   "00001111100000001111000000",
--   "00001111100000001110000000",
--   "00000111110000111110000000",
--   "00000111110000111110000000",
--   "00000111110000111100000000",
--   "00000011111001111100000000",
--   "00000011111001111000000000",
--   "00000011111001111000000000",
--   "00000001111111111000000000",
--   "00000001111111110000000000",
--   "00000001111111110000000000",
--   "00000000111111110000000000",
--   "00000000111111110000000000");
```

```
119 constant Letter_L : integer := 500;      --Letter Starting Point
120 Constant Letter_T : integer := 10;    -- Letter Top Starting Pont
121 signal Let_RGB : std_logic_vector ( 2 downto 0):="000";
122 signal X, Y :  unsigned(9 downto 0):="0000000000";
123      signal rom_addr, rom_col, V_y_Reg,V_X_Reg, V_y_Next,V_X_Next : unsigned(Rom_Row downto 0):=
124      signal rom_data: std_logic_vector(Rom_Column downto 0):=(others => '0');
125      signal Screen_on,  L_on, L_C: std_logic :='0';
126
127 begin
128
129 --- Screen_on is signal every time screen refreshes.
130 Screen_on <= '1' when (Y <= 480) else '0';
131
132
133 ---- Input pixel is converted into unsigned bits
134 X <= unsigned(X_Sync);
135 Y <= unsigned(Y_Sync);
136
137 --Once the pixel reach the location user want to display the letter
138 -- V_X_Next and V_Y_Next start incrementing to collect the bits from the Rom
139 V_X_Next <= (X (Rom_Row downto 0) - Letter_L) when L_C ='1' else V_X_Reg(Rom_Row downto 0) ;
140 V_Y_Next <= (Y (Rom_Row downto 0) - Letter_T ) when L_C ='1' else V_Y_Reg(Rom_Row downto 0);
141
142
143 rom_col <= V_X_Reg;
144 Rom_addr <= V_Y_Reg;
145
146 --- Transfer the V_x_Next to V_x_Reg every time Video is on
147  process (Clk, Video_on)
148  begin
149  if ( Video_on ='0') then
150    V_X_Reg <= (others => '0');
151  elsif (rising_edge(Clk))then
152  V_X_Reg <= V_X_Next;
153  end if;
154  end process;
155
156  --- Transfer the V_y_Next to V_Y_Reg every time Screen Refreshes
157   process (Video_on, Screen_on, Clk)
158  begin
159  if ( Screen_on ='0') then
160    V_Y_Reg <= (others => '0');
161  elsif (rising_edge(Clk))then
162  V_Y_Reg <= V_Y_Next;
163  end if;
164  end process;
165
166
167
168 --- L_C will only turn on when the pixels is in between where
169 --- user want to display the letter Letter_L = Position in X direction from left
170 ---- Letter_T = Position wrt to top
171  L_C <= '1' when ( (X >= Letter_L) and
```

```
                    (Y >= Letter_T )  and
                                    (X < (Rom_Column +Letter_L + 1) )and
                                    (Y < (Letter_T + Heigh_Rom + 1 ))) else '0';

 --- Whole row is being selected from the rom and stored in Rom_data
Rom_data <= Letter_ROM(to_integer(rom_addr));
 ---- From the row stored in Rom_data single is collected one by one to L_on
 process (L_C, rom_col, rom_data)
 begin
 if (L_C = '1') then
 L_on <= rom_data(to_integer(rom_col)) ;
 else
 L_on <='0';

 end if;
 end process;


Letter_RGB <= "010" when L_on = '1' else "000";
Letter_on <= L_on;




end Behavioral;
```

Goomba Image

Goomba Image VHDL Code

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:42:49 11/11/2014
-- Design Name:
-- Module Name:    Letter - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

```

```
-- Uncomment the following library declaration if usingw
-- 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 Letter is
    Port ( X_Sync, Y_Sync : in  STD_LOGIC_VECTOR (9 downto 0);
           Letter_on : out  STD_LOGIC;
           Letter_RGB : out  STD_LOGIC_vector ( 2 downto 0);
           Video_on, Clk : in  STD_LOGIC);
end Letter;

architecture Behavioral of Letter is
Constant Rom_Row: integer := 6; --- Row_Columns can be represented by how many bits (2^(6+1) = 128 >102
Constant Rom_Column: integer := 102; --- Number of Row in your Rom
Constant Heigh_Rom : integer := 35;--- Number of Column in your Rom

type rom_type is array (0 to 35) of std_logic_vector ( Rom_Column downto 0);
constant Letter_ROM: rom_type:= (

"00000000000000000000000000000000000011111111000000000000000000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011111111001011011011010000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011000011001111111111110000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011000011001100000000100000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011001111111100000000111000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011001000000000000000011000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011001111111111111111111000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011111111111111111111111000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011111111101100011111110000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011111111111110001111111100000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011111000110000011101111000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011111011110000111110111000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011100001110000011111011100000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011111111111000011111111100000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011111111111000000111111000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000011111111111000000111111000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000001111000000000000011011000000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000000111111111111111111111111000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000000111111111111111111111111000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000000011101111110111111111111000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000000011100111111111111111111000000000000000000000000000000000000000000000000000000
"00000000000000000000000000000000000000011100011011111100111111111111111000000000000000000000000000000000000000000000
"00000000000000000000000000000000000000011100011011111101111111111111111000000000000000000000000000000000000000000000
"00000000000000000000000000000000000000011111111000000000111111111111000110000000000000000000000000000000000000000000
"00000000000000000000000000000000000000011111111100011001011101011111000110000000000000000000000000000000000000000000
"00000000000000000000000000000000000000011111001110001110001110001111000110000000000000000000000000000000000000000000
"00000000000000000000000000000000000000011111001110001100001110001111011110000000000000000000000000000000000000000000
"00000000000000000000000000000000000000011111000000000000000000000111101111000000000000000000000000000000000000000000
```

```
"00000000000000000000000000000000111110000000000000000011111111100000000000000000000000000000
"00000000000000000000000000000000111110000000000000000011111111100000000000000000000000000000
"00000000000000000000000000000000111111111111000000000011111111000000000000000000000000000000
"00000000000000000000000000000000111111111111000000000011111111000000000000000000000000000000
"00000000000000000000000000000000000000000000111111110000001111111000000000000000000000000000
"00000000000000000000000000000000000000000000111111110000001111111000000000000000000000000000
"00000000000000000000000000000000000000000000111111111110011110000000000000000000000000000000
"00000000000000000000000000000000000000000000111111111110011110000000000000000000000000000000



--    constant Letter_ROM: rom_type:= (
--    "00000000000000000000000000",
--    "01111100000000000001111000",
--    "01111100000000000011111000",
--    "01111100000000000011111000",
--    "01111110000000000011110000",
--    "00111110000000000111110000",
--    "00111110000000000111110000",
--    "00111110000000000111100000",
--    "00011111000000001111100000",
--    "00011111000000001111100000",
--    "00011111000000001111000000",
--    "00001111100000011111000000",
--    "00001111100000011111000000",
--    "00001111100000011110000000",
--    "00001111110000111110000000",
--    "00001111110000111110000000",
--    "00001111110000111100000000",
--    "00000111111001111100000000",
--    "00000111111001111000000000",
--    "00000111111001111000000000",
--    "00000011111111111000000000",
--    "00000011111111110000000000",
--    "00000011111111110000000000",
--    "00000001111111110000000000",
--    "00000001111111100000000000",
--    "00000001111111100000000000");


constant Letter_L : integer := 50; --Letter Starting Point
Constant Letter_T : integer := 10;    -- Letter Top Starting Pont
signal Let_RGB : std_logic_vector ( 2 downto 0):="000";
signal X, Y :  unsigned(9 downto 0):="0000000000";
     signal rom_addr, rom_col, V_y_Reg,V_X_Reg, V_y_Next,V_X_Next : unsigned(Rom_Row downto 0):=
     signal rom_data: std_logic_vector(Rom_Column downto 0):=(others => '0');
     signal Screen_on,  L_on, L_C: std_logic :='0';

begin
```

```
129
130 --- Screen_on is signal every time screen refreshes.
131 Screen_on <= '1' when (Y <= 480) else '0';
132
133
134 ---- Input pixel is converted into unsigned bits
135 X <= unsigned(X_Sync);
136 Y <= unsigned(Y_Sync);
137
138 --Once the pixel reach the location user want to display the letter
139 -- V_X_Next and V_Y_Next start incrementing to collect the bits from the Rom
140 V_X_Next <= (X (Rom_Row downto 0) - Letter_L) when L_C ='1' else V_X_Reg(Rom_Row downto 0) ;
141 V_Y_Next <= (Y (Rom_Row downto 0) - Letter_T ) when L_C ='1' else V_Y_Reg(Rom_Row downto 0);
142
143
144 rom_col <= V_X_Reg;
145 Rom_addr <= V_Y_Reg;
146
147 --- Transfer the V_x_Next to V_x_Reg every time Video is on
148  process (Clk, Video_on)
149  begin
150  if ( Video_on ='0') then
151    V_X_Reg <= (others => '0');
152  elsif (rising_edge(Clk))then
153  V_X_Reg <= V_X_Next;
154  end if;
155  end process;
156
157  --- Transfer the V_y_Next to V_Y_Reg every time Screen Refreshes
158   process (Video_on, Screen_on, Clk)
159  begin
160  if ( Screen_on ='0') then
161    V_Y_Reg <= (others => '0');
162  elsif (rising_edge(Clk))then
163  V_Y_Reg <= V_Y_Next;
164  end if;
165  end process;
166
167
168
169 --- L_C will only turn on when the pixels is in between where
170 --- user want to display the letter Letter_L = Position in X direction from left
171 ---- Letter_T = Position wrt to top
172  L_C <= '1' when ( (X >= Letter_L) and
173                    (Y >= Letter_T )  and
174                                 (X < (Rom_Column +Letter_L + 1) )and
175                                 (Y < (Letter_T + Heigh_Rom + 1 ))) else '0';
176
177  --- Whole row is being selected from the rom and stored in Rom_data
178 Rom_data <= Letter_ROM(to_integer(rom_addr));
179  ---- From the row stored in Rom_data single is collected one by one to L_on
180  process (L_C, rom_col, rom_data)
181  begin
```

```
 if (L_C = '1') then
 L_on <= rom_data(to_integer(rom_col)) ;
 else
 L_on <='0';

 end if;
 end process;


Letter_RGB <= "100" when L_on = '1' else "000";
Letter_on <= L_on;




end Behavioral;
```

Keyboard Controller

Keyboard Controller VHDL Code

```
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 instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity KeyboardController is
    Port ( Clock : in STD_LOGIC;
              KeyboardClock : in  STD_LOGIC;
           KeyboardData : in  STD_LOGIC;
           LeftPaddleDirection : buffer  integer;
           RightPaddleDirection : buffer  integer
     );
end KeyboardController;

architecture Behavioral of KeyboardController is

signal bitCount : integer range 0 to 100 := 0;
signal scancodeReady : STD_LOGIC := '0';
signal scancode : STD_LOGIC_VECTOR(7 downto 0);
signal breakReceived : STD_LOGIC := '0';

constant keyboardA : STD_LOGIC_VECTOR(7 downto 0) := "00011100";
constant keyboardY : STD_LOGIC_VECTOR(7 downto 0) := "00011010";
constant keyboardK : STD_LOGIC_VECTOR(7 downto 0) := "01000010";
constant keyboardM : STD_LOGIC_VECTOR(7 downto 0) := "00111010";

begin
```

```
33
34      keksfabrik : process(KeyboardClock)
35      begin
36           if falling_edge(KeyboardClock) then
37                if bitCount = 0 and KeyboardData = '0' then --keyboard wants to send data
38                     scancodeReady <= '0';
39                     bitCount <= bitCount + 1;
40                elsif bitCount > 0 and bitCount < 9 then -- shift one bit into the scancode from
41                     scancode <= KeyboardData & scancode(7 downto 1);
42                     bitCount <= bitCount + 1;
43                elsif bitCount = 9 then -- parity bit
44                     bitCount <= bitCount + 1;
45                elsif bitCount = 10 then -- end of message
46                     scancodeReady <= '1';
47                     bitCount <= 0;
48                end if;
49           end if;
50      end process keksfabrik;
51
52      kruemelfabrik : process(scancodeReady, scancode)
53      begin
54           if scancodeReady'event and scancodeReady = '1' then
55                -- breakcode breaks the current scancode
56                if breakReceived = '1' then
57                     breakReceived <= '0';
58                     if scancode = keyboardA or scancode = keyboardY then
59                          LeftPaddleDirection <= 0;
60                     elsif scancode = keyboardK or scancode = keyboardM then
61                          RightPaddleDirection <= 0;
62                     end if;
63                elsif breakReceived = '0' then
64                     -- scancode processing
65                     if scancode = "11110000" then -- mark break for next scancode
66                          breakReceived <= '1';
67                     end if;
68
69                     if scancode = keyboardA then
70                          LeftPaddleDirection <= 2; -- -1
71                     elsif scancode = keyboardY then
72                          LeftPaddleDirection <= 1;
73                     elsif scancode = keyboardK then
74                          RightPaddleDirection <= 2; -- -1
75                     elsif scancode = keyboardM then
76                          RightPaddleDirection <= 1;
77                     end if;
78                end if;
79           end if;
80      end process kruemelfabrik;
81 end Behavioral;
```

UCF File

UCF File

```
## Clock signal
NET "clk" LOC = "V10";

## 7 segment display
NET "Seg<0>"    LOC = "T17";
NET "Seg<1>"    LOC = "T18";
NET "Seg<2>"    LOC = "U17";
NET "Seg<3>"    LOC = "U18";
NET "Seg<4>"    LOC = "M14";
NET "Seg<5>"    LOC = "N14";
NET "Seg<6>"    LOC = "L14";
NET "Seg<7>"    LOC = "M13";

NET "AX<0>"     LOC = "N16";
NET "AX<1>"     LOC = "N15";
NET "AX<2>"     LOC = "P18";
NET "AX<3>"     LOC = "P17";


## Leds
#NET "Led<0>"        LOC = "U16" ;#| IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L2P_CMPC
Sch name = LD0
#NET "Led<1>"          LOC = "V16" ;#| IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L2N_CM
Sch name = LD1
#NET "Led<2>"          LOC = "U15" ;#|| IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L5P,
Sch name = LD2
#NET "Led<3>"          LOC = "V15" ;#|| IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L5N,
Sch name = LD3
#NET "Led<4>"          LOC = "M11" | IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L15P,
Sch name = LD4
#NET "Led<5>"          LOC = "N11" | IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L15N,
Sch name = LD5
#NET "Led<6>"          LOC = "R11" | IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L16P,
Sch name = LD6
#NET "Led<7>"          LOC = "T11" | IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L16N_VRE
Sch name = LD7


## Switches
NET "reset"          LOC = "T10";
#NET "color<1>"     LOC = "T9";
#NET "color<2>"     LOC = "V9";
#NET "sw<3>"          LOC = "M8"  | IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L40P,
Sch name = SW3
#NET "sw<4>"          LOC = "N8"  | IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L40N,
Sch name = SW4
#NET "sw<5>"          LOC = "U8"  | IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L41P,
Sch name = SW5
#NET "sw<6>"          LOC = "V8"  | IOSTANDARD = "LVCMOS33";    #Bank = 2, Pin name = IO_L41N_VRE
Sch name = SW6
#NET "sys_pause"          LOC = "T5"  | IOSTANDARD = "LVCMOS33";    #Bank = MISC, Pin name = IO_L
Sch name = SW7
```

```
40
41
42  ## Buttons
43  #NET "player1_controls<0>"    LOC = "C4";
44  NET "Padle<1>" LOC = "A8";
45  #
46  NET "Padle<0>"      LOC = "C9";
47  #NET "player2_controls<1>"    LOC = "D9";
48
49  ##NET "continue" LOC = "B8";
50
51  ## VGA Connector
52  NET "rgb<2>"    LOC = "U7";
53  NET "rgb<2>"    LOC = "V7";
54  NET "rgb<2>"    LOC = "N7";
55  NET "rgb<1>"    LOC = "P8";
56  NET "rgb<1>"    LOC = "T6";
57  NET "rgb<1>"    LOC = "V6";
58  NET "rgb<0>"    LOC = "R7";
59  NET "rgb<0>"    LOC = "T7";
60
61  NET "hsync"         LOC = "N6";
62  NET "vsync"    LOC = "P7";
63
64  NET "KeyboardData" LOC = "J13" | PULLUP;
65  NET "KeyboardClock" LOC = "L12" | PULLUP;
```

Summary

In this lab, we used an existing VGA driver and a bare-bones pong game and added the following features:

1. A second paddle
2. Deleted left wall
3. Added Keyboard Control
4. Added Player Decals
5. Added Player Initials
6. Fixed scoring so both players score correctly

Wall

These were all put together with components in the top module. To remove the wall, we simply commented the code that inserted the left wall.

Paddles

To add the second, we called the same paddle, remapped the left wall signal to the new paddle, and positioned that paddle on the left side, 40 pixels from the wall.

Keyboard Control

To add keyboard control, we used a keyboard controller built by ress on Github: https://github.com/ress/VHDL-Pong/blob/master/KeyboardController.vhd. This keyboard controller took the place of the buttons. We then mapped the keyboard clock and keyboard data in our UCF with the `PULLUP` option.

Player Decals and Initials

To add player decals and initials, we used GIMP and stored an ASCII PBM. We wrote a small Ruby function to process the PBM into the correct dimensions and reverse it, as well as add the quotes and commas.

Ruby to parse PBM file for the VHDL Pong Game

```ruby
image=File.read("image.pbm")
image.scan(/.{1,103}/m).each do |line|
  print "\"", line.reverse, "\"", ",", "\n"
end; nil
```

Scoring

To get the scoring working, we added a small bit of magic to keep the ball from going outside the allowable buffer, then setting the minimum left value to at or above that value: `ball_x_l < MAX_X_L`. After this addition, the scoring worked perfectly.

Links

All the code is hosted on a public repository on Github: https://github.com/rickpr/vhdlpong
The Keyboard controller is thanks to ress: https://github.com/ress/VHDL-Pong/blob/master/KeyboardController.vhd
-Ricardo Piro-Rael & Dillon Thomas fdisk@fdisk.co