The Pong Game

The classic Ping-Pong game also a good demonstration of a FPGA design involving VGA interface and Keyboard (PS2) interface as well.

Clock
clk_lc4 LOC=P54

game_title
clk_lc4
clk seven_seg(7:0) an(3:0)
seg_a,seg_b,seg_c,seg_d,seg_e,seg_f,seg_g,seg_dp
an(3:0)

Push Button Reset
btn3 LOC=P41

Four-Digit, Seven-Segment LED Display
AN3 AN2 AN1 AN0
LOC=p34 an(0)
LOC=p33 an(1)
LOC=p32 an(2)
LOC=p26 an(3)
an(3:0)
LOC=p25 seg_a
LOC=p16 seg_b
LOC=p23 seg_c
LOC=p21 seg_d
LOC=p20 seg_e
LOC=p17 seg_f
LOC=p83 seg_g
LOC=p22 seg_dp

Read_PS2_VHDL
clk_lc4, ps2c, ps2d, btn3
Clk, PS2_Clk, PS2_Data, Reset, Serve, Left_dir(1:0), Right_dir(1:0), PS2_Code(7:0)

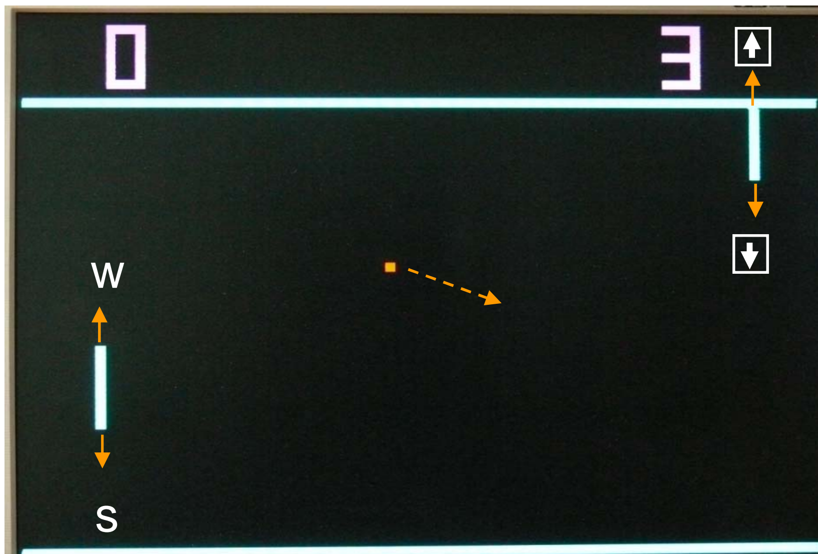
cntrl
clk_lc4, btn3, serve, lf_dir(1:0), rt_dir(1:0), ld(7:0)
CLK, RESET, SERVE, left_dir(1:0), right_dir(1:0), HSYNCH, VSYNCH, COLOR(1:0)

vga_int
clk_lc4, hsync, vsync
CLK, HSYNCH_IN, VSYNCH_IN, COLOR(1:0), RED, BLUE, GREEN, VSYNCH_OUT, HSYNCH_OUT
vga_red0, vga_blue0, vga_green0, vga_vs, vga_hs

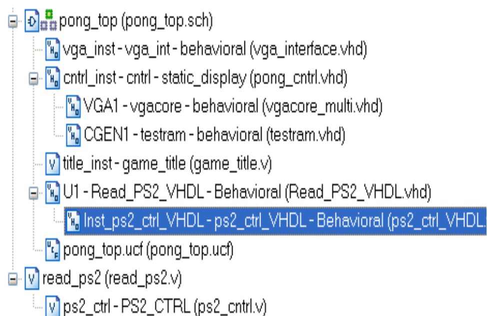
LEDs
ld(7:0)

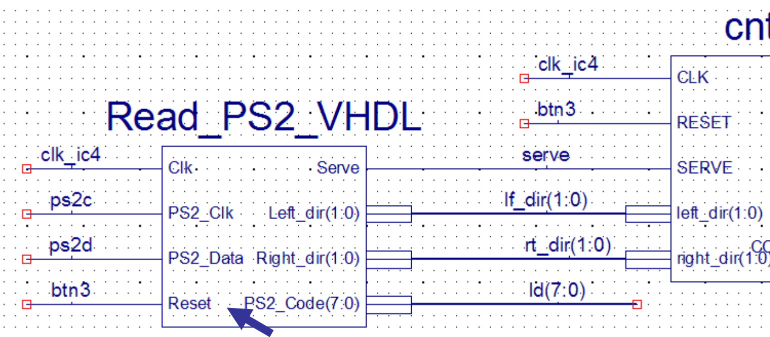
VGA Port
pin5 pin1 pin10 pin6 pin15 pin11
LOC=p70 vga_red0 pin1
LOC=p68 vga_red1
LOC=p67 vga_red2
LOC=p52 vga_green0 pin2
LOC=p51 vga_green1
LOC=p50 vga_green2
LOC=p44 vga_blue0 pin3
LOC=p43 vga_blue1
OBUF
pin13 LOC=p39 vga_hs
pin14 LOC=p35 vga_vs
pins 6,7,8,10,11 => GND
pins 4,5,9,12,15 => OPEN

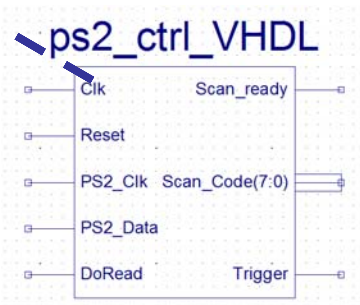
PS2 Port
Pin1 - Data (PS2D)
Pin2 - Reserved
Pin3 - GND
Pin4 - Voltage Supply
Pin5 - CLK (PS2C)
Pin6 - Reserved
ps2d LOC=P97
ps2c LOC=P96



The PS2 interface (1)





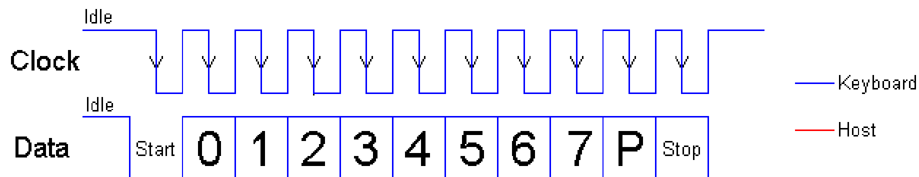



```
 8  entity ps2_ctrl_VHDL is
 9      Port ( Clk :          in   STD_LOGIC;
10             Reset :        in   STD_LOGIC;
11             PS2_Clk :      in   STD_LOGIC;
12             PS2_Data :     in   STD_LOGIC;
13             Scan_Code :    out  STD_LOGIC_VECTOR (7 downto 0);
14             Scan_ready :   out  STD_LOGIC;
15             Trigger :      inout  STD_LOGIC;
16             DoRead :       in STD_LOGIC);
17  end ps2_ctrl_VHDL;
```

```
27
28     This_Process_Synchronize_Data_exchange:
29     process( Clk)
30        Variable Clear_reg:  STD_LOGIC_VECTOR (1 downto 0);
31     begin
32        if rising_edge( clk) then
33           if Reset='1' then
34              Scan_code  <= "00000000";
35              Scan_Ready <= '0';
36              Clear_reg  := "00";
37           else
38              if (trigger='1') and (DoRead='1') then
39                    Scan_Code  <= S_reg;     --Scan_Code gets the shifted in value of PS2_Data
40                    Clear_reg  := "00";
41                    scan_ready <=  '1';      -- New scan code has been received
42              else
43                 clear_reg := clear_reg + 1; -- wait two clock cycles before clearing scan_ready
44                 if (clear_reg >= "10") then
45                    Clear_reg  := "00";
46                    scan_ready <=  '0';
47                 end if;
48              end if;
49           end if;
50        end if;
51     end process This_Process_Synchronize_Data_exchange;
52
```

The PS2 interface (2)

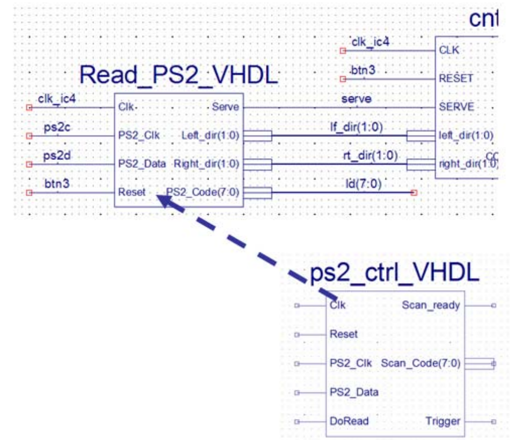
```
19 architecture Behavioral of ps2_ctrl_VHDL is
20    signal S_reg:  STD_LOGIC_VECTOR (7 downto 0);
21    -- After start bit  =  1  0  0  0  0  0  0  0
22    -- After Data bit 0 = d0  1  0  0  0  0  0  0
23    -- After Data bit 1 = d1 d0  1  0  0  0  0  0
24    -- After Data bit 6 = d6 d5 d4 d3 d2 d1 d0  1 <- This 1 bit indicates data almost ready
25    -- After Data bit 7 = d7 d6 d5 d4 d3 d2 d1 d0    Time to change state to Parity check
26 begin
```




```
48    -------------------------------------------------------------------------------
49    --                       start    b0   8 Data-bits     b7  parity   stop
50    -- PS2_Data = -------__________----==================------ ppppppp ssssss ----
51    -- PS2_Clk  = -----------____----____---...........---___----____----___
52    --
53    -------------------------------------------------------------------------------
54    This_Process_Recieve_Data_from_the_PS2:
55    process( PS2_Clk, Reset)
56       type     States is (Start_bit, Data_bits, Parity_bit, Stop_bit);
57       variable Read_state: States;
58    begin
59       if Reset='1' then
60          Read_State := Start_Bit;
61          S_Reg      <= "10000000";    -- The 1-bit acts as a "Counter"
62       elsif falling_edge( PS2_Clk) then
63          Trigger <= '0';
64          case Read_State is
65
66             when Start_bit =>
67                         if PS2_Data='0' then
68                            S_Reg  <= "10000000";
69                            Read_State := Data_Bits;
70                         end if;
71
72             when Data_bits =>
73                         if S_Reg(0)='1' then    -- Change state after 8 data bits
74                            Read_State := Parity_bit;
75                         end if;
76                         S_Reg <= PS2_Data & S_reg(7 downto 1); -- Next data-bit
77
78             when Parity_bit =>                   -- not used or checked
79                         Read_State := Stop_bit;
80
81             when Stop_bit  =>                    -- not checked
82                         Trigger    <= '1';       -- Indicates data ready now
83                         Read_State := Start_bit;  -- One more time
84          end case;
85       end if;
86    end process This_Process_Recieve_Data_from_the_PS2;
```

The Key Encoder (1)

```
8   entity Read_PS2_VHDL is
9        Port ( Clk :       in  STD_LOGIC;
10              PS2_Clk :   in  STD_LOGIC;
11              PS2_Data :  in  STD_LOGIC;
12              Reset :     in  STD_LOGIC;
13              PS2_Code :  inout  STD_LOGIC_VECTOR (7 downto 0);
14              Left_dir :  inout  STD_LOGIC_VECTOR (1 downto 0);
15              Right_dir :inout   STD_LOGIC_VECTOR (1 downto 0);
16              Serve :     inout  STD_LOGIC);
17  end Read_PS2_VHDL;
18
19  architecture Behavioral of Read_PS2_VHDL is
20     signal read:        std_logic;
21     signal Data_Ready:  std_logic;
22     signal stopkey:     std_logic;
23     signal trigger:     std_logic;
24
25     COMPONENT ps2_ctrl_VHDL
26     PORT( Clk  : IN std_logic;
27           Reset : IN std_logic;
28           PS2_Clk : IN std_logic;
29           PS2_Data : IN std_logic;
30           DoRead : IN std_logic;
31           Trigger : INOUT std_logic;
32           Scan_Code : OUT std_logic_vector(7 downto u);
33           Scan_ready : OUT std_logic );
34     END COMPONENT;
35  begin
36
37     Inst_ps2_ctrl_VHDL: ps2_ctrl_VHDL PORT MAP(
38        Clk =>        Clk ,
39        Reset =>      Reset,
40        PS2_Clk =>    PS2_Clk,
41        PS2_Data =>   PS2_Data,
42        Scan_Code =>  PS2_Code,
43        Scan_ready => Data_Ready,
44        Trigger =>    Trigger,
45        DoRead =>     Read);
```



The keyboard sends data to the host in 11-bit words that contain a ‘0’ start bit, followed by 8-bits of scan code (LSB first), followed by an odd parity bit and terminated with a ‘1’ stop bit. The keyboard generates 11 clock transitions (at around 20 - 30KHz) when the data is sent, and data is valid on the falling edge of the clock.

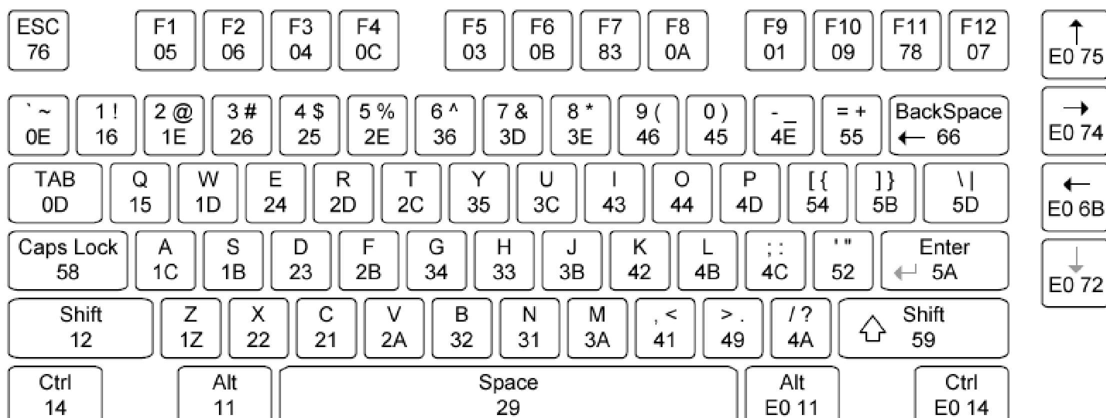


Figure 11. Keyboard scan codes

```
47     This_process_recieves_and_decodes_scan_codes_from_the_keyboard:
48     process( Clk)
49        type      Key_states is (Not_Pressed, Pressed);
50        variable Key_state: Key_states;
51     begin
52        if rising_edge( Clk) then
53           if Reset='1' then
54              right_dir <= "00";              -- No right paddle movement
55              left_dir  <= "00";              -- No left paddle movement
56              serve     <=  '1';              -- Start with ball being served
57              read      <=  '1';              -- Ready to receive scan code
58              stopkey   <=  '0';              -- Stop key code has not been read
59              Key_state :=  Not_Pressed;
60
61           elsif (Data_Ready='1') or (Read='0') then
62              case Key_state is
63                 when Not_Pressed =>
64                    if Stopkey='0' then
65                       case (ps2_code) is
66                          when "01110101" =>          -- up arrow key
67                                right_dir <= "01";    -- right up
68                          when "01110010" =>          -- down arrow key
69                                right_dir <= "10";    -- right down
70                          when "00011101" =>          -- w key
71                                left_dir <= "01";     -- left up
72                          when "00011011" =>          -- s key
73                                left_dir <= "10";     -- left down
74                          when "00101001" =>          -- space bar key
75                                serve <= '1';         -- serve ball
76                          when "11110000" =>          -- A key has been released
77                                stopkey <= '1';       -- set stopkey bit
78                                serve   <= '0';
79                          when others =>
80                       end case;            -- end of case (ps2_code) statement
81                       Key_state := Pressed;
82                    else                   -- new data is telling which key was just released
83                       stopkey <= '0';
84                       case (ps2_code) is
85                          when "01110101" =>               -- up arrow key
86                             if (right_dir = "01") then
87                               right_dir <= "00";          -- stop right paddle up motion
88                             end if;
89                          when "01110010" =>               -- down arrow key
90                             if (right_dir = "10") then
91                                right_dir <= "00";         -- stop right paddle down motion
92                             end if;
93                          when "00011101" =>               -- w key
94                             if (left_dir = "01") then
95                                left_dir <= "00";          -- stop left paddle up motion
96                             end if;
97                          when "00011011" =>               -- s key
98                             if (left_dir = "10") then
99                                left_dir <= "00";          -- stop left paddle down motion
100                            end if;
101                         when "11100000" =>               -- Extended code key
102                               stopkey <= '1';            -- Check next ps2 entry for stop
103                         when "00101001" =>               -- space bar key
104                               serve   <= '0';            -- stop serve ball
105                         when others =>
106                               stopkey <= '0';
107                      end case;                       -- end of case (ps2_code) statement
108                      Key_state := Pressed;
109                   end if;
110                   read <= '0';
111
112                When Pressed =>
113                   if (Trigger='0') then
114                      read <= '1';                 -- resets read bit to enable PS2_Ctrl inp
115                      Key_state := Not_Pressed;
116                   end if;
117             end case;
118          end if;
119       end if;
120    end process This_process_recieves_and_decodes_scan_codes_from_the_keyboard;
```

The VGAcore (1)

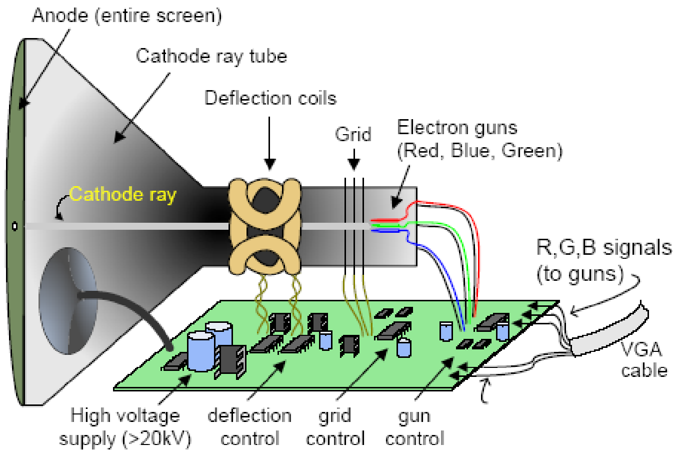
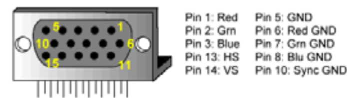


Figure 14. CRT deflection system



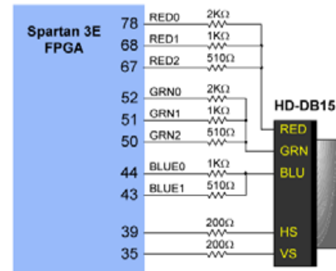
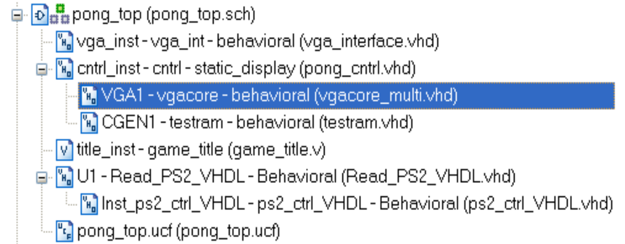
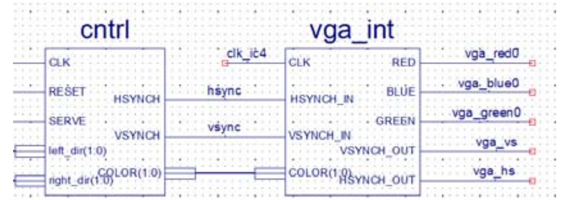


Figure 13. VGA pin definitions and Basys circuit






```vhdl
23 entity vgacore is Port (
24     CLK :     in  std_logic;
25     RESET:    in  std_logic;
26     HSYNCH:   out std_logic;
27     VSYNCH:   out std_logic;
28     HBLANK:   out std_logic;
29     LINE:     out std_logic_vector(5 downto 0);
30     PIXEL:    out std_logic_vector(6 downto 0));
31 end vgacore;
32
33 architecture behavioral of vgacore is
34     signal TEMP_PIXEL: std_logic_vector( 9 downto 0);
35     signal TEMP_LINE:  std_logic_vector( 8 downto 0);
36
37     signal HCOUNTER :     integer range 1023 downto 0 := 0;
38     signal COUNTER_RESET: std_logic;
39     signal VCLK:          std_logic;
40
41     signal VCOUNTER :     integer range 1023 downto 0 := 0;
42     signal VERTICAL_COUNTER_RESET: std_logic;
43 begin
44
45 -----------------------------------------------------------------------
46 -- State Machine Counter Process
47 -- These counters are to be used as resources for state machine control
48 -----------------------------------------------------------------------
49 horizontal_counter: process ( CLK )
50 begin
51     if CLK='1' and CLK'event then
52         if COUNTER_RESET = '1' then
53             HCOUNTER <= 0;
54         else
55             HCOUNTER <= HCOUNTER + 1;
56         end if;
57     end if;
58 end process;
59
60 vertical_counter: process ( VCLK )
61 begin
62     if VCLK = '1' and VCLK'event then
63         if VERTICAL_COUNTER_RESET = '1' then
64             VCOUNTER <= 0;
65         else
66             VCOUNTER <= VCOUNTER + 1;
67         end if;
68     end if;
69 end process;
```

The VGAcore - Timing of the VGA

pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
pixel 479,0
pixel 479,639
Current through the horizontal deflection coil
Retrace: No information is displayed during this time
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
HS
Horizontal sync signal sets the retrace frequency
"front porch"
"back porch"
UG230_c6_02_021706

Figure 6-2: **CRT Display Timing Example**

Table 6-2: **640x480 Mode VGA Timing**

Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48

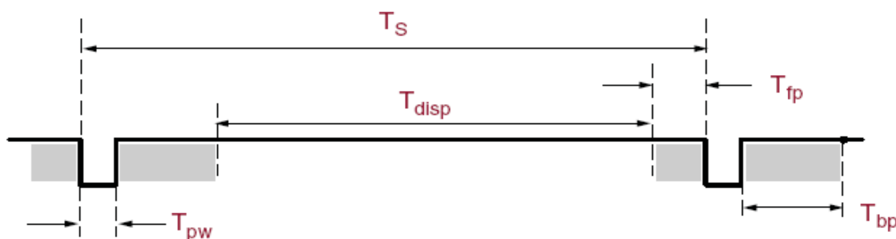
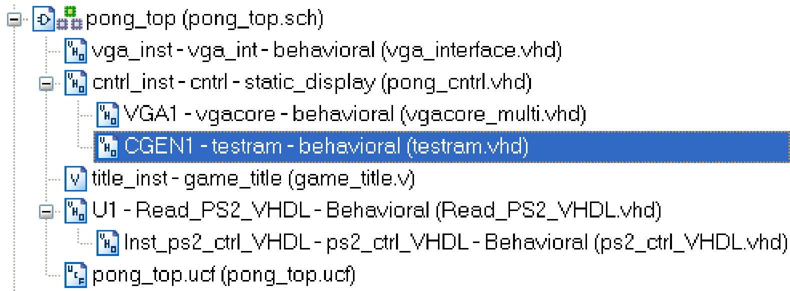


Figure 6-3: **VGA Control Timing**

```
71  ----------------------------------------------------------------------
72  -- Horizontal State Machine
73  ----------------------------------------------------------------------
74  Horizontal: process ( CLK, RESET )
75      type HSTATE is (  HRESET,
76                        FRONT_PORCH,
77                        SYNCH,              --  96 pixels (clk-pulses) wide
78                        BACK_PORCH,
79                        LEFT_BORDER,
80                        ACTIVE_VIDEO,       -- 640 pixels
81                        RIGHT_BORDER );
82      variable Horizontal_State : HSTATE := HRESET;
83      variable PIXEL_COUNT : integer := 0;
84  begin
85      if ( RESET = '1' ) then
86          Horizontal_State := HRESET;
87          COUNTER_RESET  <= '1';
88          HSYNCH         <= '1';
89          VCLK           <= '0';
90          HBLANK         <= '1';
91      elsif ( CLK = '1' and CLK'EVENT ) then
92          VCLK           <= '0';
93          HBLANK         <= '1';
94          PIXEL          <= ( others => '0' );
95          COUNTER_RESET  <= '0';
96          HSYNCH         <= '1';
97
98          case ( Horizontal_State ) is
99              when HRESET =>
100                 Horizontal_State := LEFT_BORDER;
101                 COUNTER_RESET   <= '0';
102                 HSYNCH          <= '1';
103             --------------------------- 5 pixels----- Want Left Border
104             when LEFT_BORDER   =>
105                 if (HCOUNTER = 4) then
106                     Horizontal_State := ACTIVE_VIDEO;
107                     COUNTER_RESET     <= '1';
108                 end if;
109             ------------------------------------- Want 640 Active Pixels
110             when ACTIVE_VIDEO =>
111                 HBLANK <= '0';
112                 TEMP_PIXEL <= CONV_STD_LOGIC_VECTOR(HCOUNTER,10);
113                 PIXEL( 6 downto 0 ) <= TEMP_PIXEL( 9 downto 3 );
114                 if (HCOUNTER = 639) then
115                     Horizontal_State := RIGHT_BORDER;
116                     COUNTER_RESET     <= '1';
117                 end if;
118             ------------------------------ 6 ---- Want Right Border Pixels
119             when RIGHT_BORDER =>
120                 if (HCOUNTER = 6) then
121                     Horizontal_State := FRONT_PORCH;
122                     COUNTER_RESET     <= '1';
123                 end if;
124             --------------------------- 7+9=16 -- Want Front porch Pixels
125             when FRONT_PORCH =>
126                 if (HCOUNTER = 8) then
127                     Horizontal_State := SYNCH;
128                     COUNTER_RESET     <= '1';
129                     VCLK              <= '1';         -- Generate a VCLK __---
130                 end if;
131             -------------------------------- 96 pixels -- Wanted Synch puls
132             when SYNCH =>
133                 HSYNCH <= '0';
134                 if (HCOUNTER = 95) then
135                     Horizontal_State := BACK_PORCH;
136                     COUNTER_RESET     <= '1';
137                 end if;
138             ----------------------------- 47 pixels ----- Wanted Back Porch
139             -- This value will move the sceen image vertically
140             when BACK_PORCH =>
141                 if (HCOUNTER = 47) then
142                     Horizontal_State := LEFT_BORDER;
143                     COUNTER_RESET     <= '1';
144                 end if;
145             when others =>
146                 Horizontal_State := HRESET;
147                 COUNTER_RESET     <= '1';
148                 HSYNCH            <= '1';
149         end case;
150     end if;
151 end process;
152
```

```
153 ----------------------------------------------------------------------
154 -- Vertical State Machine
155 ----------------------------------------------------------------------
156 Vertical: process ( VCLK, RESET )
157    type VSTATE is ( VRESET,
158                     FRONT_PORCH,
159                     SYNCH,          --   2 lines
160                     BACK_PORCH,
161                     TOP_BORDER,
162                     ACTIVE_VIDEO,   -- 480 lines
163                     BOTTOM_BORDER );
164    variable Vertical_State : VSTATE := VRESET;
165    variable LINE_COUNT : integer := 0;
166 begin
167    if (RESET = '1' ) then
168       Vertical_State          := VRESET;
169       VERTICAL_COUNTER_RESET  <= '1';
170       VSYNCH                  <= '1';
171    elsif ( VCLK = '1' and VCLK'EVENT ) then
172       VERTICAL_COUNTER_RESET  <= '0';
173       VSYNCH                  <= '1';
174       LINE                    <= ( others => '0' );
175 
176       case ( Vertical_State ) is
177          when VRESET =>
178             Vertical_State := TOP_BORDER;
179             VERTICAL_COUNTER_RESET <= '1';
180             VSYNCH <= '1';
181          --------------------6+2=8 lines-- Want a Vertical Front porch
182          when FRONT_PORCH =>
183             if (VCOUNTER = 1) then
184                Vertical_State := SYNCH;
185                VERTICAL_COUNTER_RESET <= '1';
186             end if;
187          ----------------------------2 lines--- Want a Vertical Synch
188          when SYNCH =>
189             VSYNCH <= '0';
190             if (VCOUNTER = 1) then
191                Vertical_State := BACK_PORCH;
192                VERTICAL_COUNTER_RESET <= '1';
193             end if;
194          ---------------------------24----- Want a Vertical Back Porch
195          when BACK_PORCH =>
196             if (VCOUNTER = 23) then
197                Vertical_State := TOP_BORDER;
198                VERTICAL_COUNTER_RESET <= '1';
199             end if;
200          ---------------------24+5=29 lines-- Want a Vertical Top border
201          when TOP_BORDER   =>
202             if (VCOUNTER = 4) then
203                Vertical_State := ACTIVE_VIDEO;
204                VERTICAL_COUNTER_RESET <= '1';
205             end if;
206          --------------------------------------------- Want 480 lines
207          when ACTIVE_VIDEO =>
208             TEMP_LINE <= CONV_STD_LOGIC_VECTOR(VCOUNTER,9);
209             LINE      <= TEMP_LINE(8 downto 3);
210             if (VCOUNTER = 479) then
211                Vertical_State := BOTTOM_BORDER;
212                VERTICAL_COUNTER_RESET <= '1';
213             end if;
214          ------------------------------6------ Want a Vertical border
215          when BOTTOM_BORDER =>
216             if (VCOUNTER = 5) then
217                Vertical_State := FRONT_PORCH;
218                VERTICAL_COUNTER_RESET <= '1';
219             end if;
220          when others =>
221             Vertical_State := VRESET;
222             VERTICAL_COUNTER_RESET <= '1';
223             VSYNCH <= '1';
224          end case;
225    end if;
226 end process;
227 
228 end behavioral;
```

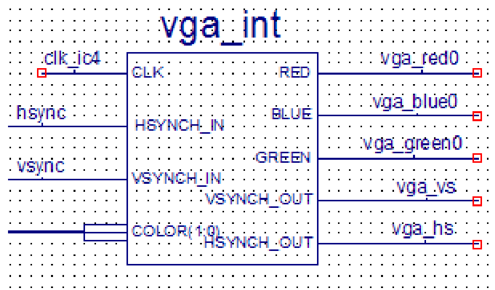
The Digit ROM

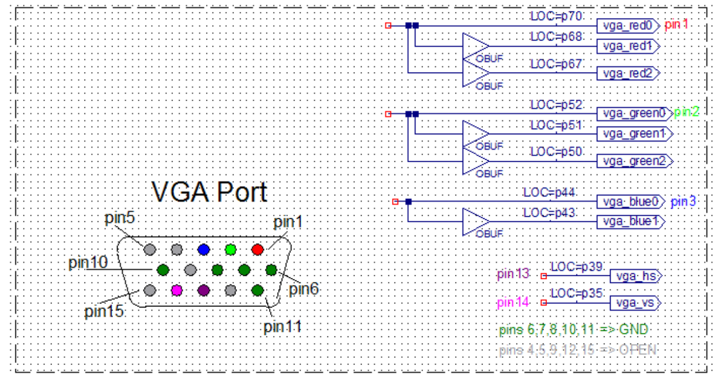



```
18  library IEEE;
19  use IEEE.STD_LOGIC_1164.ALL;
20  use IEEE.STD_LOGIC_ARITH.ALL;
21  use IEEE.STD_LOGIC_UNSIGNED.ALL;
22
23  entity testram is Port (
24      address: in std_logic_vector(6 downto 0);
25      data:    out std_logic_vector(3 downto 0)
26      );
27  end testram;
28
29  architecture behavioral of testram is
30
31  type mem_array is array (0 to 79) of
32      std_logic_vector(3 downto 0);
33  constant characters: mem_array := (
34
35      -- 0
36      "0000",
37      "1111",
38      "1001",
39      "1001",
40      "1001",
41      "1001",
42      "1001",
43      "1111",
44
45      -- 1
46      "0000",
47      "0001",
48      "0001",
49      "0001",
50      "0001",
51      "0001",
52      "0001",
53      "0001",
54
55      -- 2
56      "0000",
57      "1111",
58      "0001",
59      "0001",
60      "1111",
61      "1000",
62      "1000",
63      "1111",
65      -- 3
66      "0000",
67      "1111",
68      "0001",
69      "0001",
70      "1111",
71      "0001",
72      "0001",
73      "1111",
74
75      -- 4
```

```
75      -- 4
76      "0000",
77      "1001",
78      "1001",
79      "1001",
80      "1111",
81      "0001",
82      "0001",
83      "0001",
84
85      -- 5
86      "0000",
87      "1111",
88      "1000",
89      "1000",
90      "1111",
91      "0001",
92      "0001",
93      "1111",
94
95      -- 6
96      "0000",
97      "1111",
98      "1000",
99      "1000",
100     "1111",
101     "1001",
102     "1001",
103     "1111",
104
105     -- 7
106     "0000",
107     "1111",
108     "0001",
109     "0001",
110     "0001",
111     "0001",
112     "0001",
113     "0001",
114
115     -- 8
116     "0000",
117     "1111",
118     "1001",
119     "1001",
120     "1111",
121     "1001",
122     "1001",
123     "1111",
124
125     -- 9
126     "0000",
127     "1111",
128     "1001",
129     "1001",
130     "1111",
131     "0001",
132     "0001",
133     "0001"
134     );
135
136 begin
137
138 process (address )
139 begin
140         data <= characters(
141         conv_integer(address));
142 end process;
143
144 end behavioral;
```

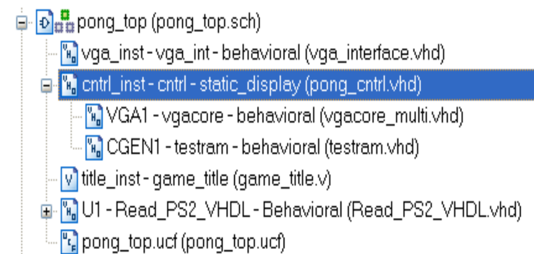
The VGA int






```
23  entity vga_int is Port (
24      CLK :        in  std_logic;
25      COLOR:       in  std_logic_vector ( 1 downto 0 );
26      VSYNCH_IN:   in  std_logic;
27      HSYNCH_IN:   in  std_logic;
28      RED:         out std_logic;
29      BLUE:        out std_logic;
30      GREEN:       out std_logic;
31      VSYNCH_OUT:  out std_logic;
32      HSYNCH_OUT:  out std_logic);
33  end vga_int;
34
35  architecture behavioral of vga_int is
36      signal VSYNCH_PIPE: std_logic;
37      signal HSYNCH_PIPE: std_logic;
38
39      signal    RGB:       Std_logic_vector( 2 downto 0);
40      constant xBLACK:     Std_logic_vector( 2 downto 0) := "000";
41      constant xRED:       Std_logic_vector( 2 downto 0) := "100";
42      constant xGREEN:     Std_logic_vector( 2 downto 0) := "010";
43      constant xBLUE:      Std_logic_vector( 2 downto 0) := "001";
44      constant xWHITE:     Std_logic_vector( 2 downto 0) := "111";
45      constant xMAGENTA:   Std_logic_vector( 2 downto 0) := "101";
46      constant xCYAN:      Std_logic_vector( 2 downto 0) := "011";
47      constant xYELLOW:    Std_logic_vector( 2 downto 0) := "110";
48  begin
49      PIPELINE: process  (CLK)
50      begin
51          if ( CLK = '1' and CLK'event ) then
52  --          VSYNCH_OUT  <= VSYNCH_PIPE; HSYNCH_OUT  <= HSYNCH_PIPE;
53  --          HSYNCH_PIPE <= HSYNCH_IN;   VSYNCH_PIPE <= VSYNCH_IN;
54              VSYNCH_OUT  <= VSYNCH_IN;
55              HSYNCH_OUT  <= HSYNCH_IN;
56          end if;
57      end process;
58
59      COLOR_LUT: process  (CLK)
60      begin
61          if ( CLK = '1' and CLK'event ) then
62              RGB <= xBLACK;
63              case COLOR is
64                  when "00" =>        -- Black (only ???)
65                      RGB <= xBLACK;
66                  when "01" =>
67                      RGB <= xRED;
68                  when "10" =>
69                      RGB <= xMAGENTA;
70                  when "11" =>
71                      RGB <= xCYAN;
72                  when others => NULL;
73              end case;
74          end if;
75      end process;
76
77      RED   <= RGB(2);
78      GREEN <= RGB(1);
79      BLUE  <= RGB(0);
80  end behavioral;
```

```vhdl
23  entity cntrl is Port (
24      CLK :       in std_logic;
25      RESET:      in std_logic;
26      left_dir:     in std_logic_vector(1 downto 0);
27      right_dir:    in std_logic_vector(1 downto 0);
28      SERVE:      in std_logic;
29      HSYNCH:     out std_logic;
30      VSYNCH:     out std_logic;
31      COLOR:      out std_logic_vector( 1 downto 0 ));
32  end cntrl;
33
34  architecture static_display of cntrl is
35
36      component VGAcore is Port (
37          CLK :       in std_logic;
38          RESET:      in std_logic;
39          HSYNCH:     out std_logic;
40          VSYNCH:     out std_logic;
41          HBLANK:     out std_logic;
42          LINE:       out std_logic_vector(5 downto 0);
43          PIXEL:      out std_logic_vector(6 downto 0));
44      end component;
45
46      component testram is Port (
47          address: in std_logic_vector( 6 downto 0 );
48          data:    out std_logic_vector( 3 downto 0 ));
49      end component;
50
51      signal   number_data:     std_logic_vector( 3 downto 0 );
52      signal   number_address:  std_logic_vector( 6 downto 0 );
53      signal   enable:          std_logic;
54
55      signal   LINE:     std_logic_vector( 5 downto 0 );
56      signal   PIXEL:    std_logic_vector( 6 downto 0 );
57      signal   HBLANK:   std_logic;
58
59      signal   next_HSYNCH:    std_logic;
60      signal   next_VSYNCH:    std_logic;
61      signal   VCLK:           std_logic;
62      signal   next_COLOR:     std_logic_vector( 1 downto 0 );
63
64      constant ball_height:    integer := 1;
65      constant ball_width:     integer := 2;
66
67      constant paddle_height:   integer := 8;
68      constant paddle_heighta:  integer := 4;  -- defines top 1/4 of paddle
69      constant paddle_heightb:  integer := 6;  -- defines middle 1/2 of paddle
70      constant paddle_heightc:  integer := 8;  -- defines bottom 1/4 of paddle
71      constant paddle_width:    integer := 2;  -- defines width of paddle
72
73      constant wall_height:    integer := 1;
74      constant wall_top:       integer := 9;
75      constant wall_bottom:    integer := 58;
76      constant right_wall:     integer := 77;
77      constant left_wall:      integer := 2;
78
79      constant right_score_x:  integer := 64;
80      constant right_score_y:  integer := 4;
81      constant left_score_x:   integer := 16;
82      constant left_score_y:   integer := 4;
83      constant score_width:    integer := 4;
84      constant score_height:   integer := 4;
85
86      constant right_x: integer := 72;
87      constant left_x:  integer := 8;
88
89      signal ball_xdir, ball_ydir: std_logic;
90      signal ball_x:      integer range 0 to 80;
91      signal ball_y:      integer range 0 to 60 := 30;
92      signal ball_yrate:  integer range 0 to 3;
93
94      signal left_y:        std_logic_vector(5 downto 0);
95      signal right_y:       std_logic_vector(5 downto 0);
96      signal nextleft_y:    std_logic_vector(5 downto 0);
97      signal nextright_y:   std_logic_vector(5 downto 0);
98
99      -- Delay vector is used to slow down the speed of the ball
100     signal delay: std_logic_vector(2 downto 0);
101     signal lscore, rscore: integer range 0 to 9 := 0;
102 begin
```



```
102 begin
103     -- VGA CORE Instantiation
104     VGA1: vgacore port map (
105         CLK     => CLK,
106         RESET   => RESET,
107         HSYNCH  => next_HSYNCH,
108         VSYNCH  => next_VSYNCH,
109         HBLANK  => HBLANK,
110         LINE    => LINE,
111         PIXEL   => PIXEL
112         );
113 
114     -- Character generator memory instantiation
115     CGEN1: testram port map (
116         address    => number_address ,
117         data       => number_data
118         );
119 
120 -- Pipeline the control signals to account for Game Delay
121 pipeline: process ( clk, pixel, line)
122 begin
123     if ( clk = '1' and clk'event) then
124         VSYNCH <= next_VSYNCH;
125         VCLK   <= next_VSYNCH;
126         HSYNCH <= next_HSYNCH;
127         if ( HBLANK = '1' ) then
128             color <= "00";
129         else
130             color <= next_COLOR;
131         end if;
132     end if;
133 end process;
134 
135 -- Code to display the ball and paddles
136 display: process (clk, line, pixel, left_y, right_y,
137                   ball_x, ball_y, lscore, number_data, rscore)
138 begin
139     number_address(2 downto 0) <= line(2 downto 0);
140     ------------------------------- Display Background Color ---------------
141     next_COLOR <= "00";
142     ------------------------------- Display the playing field top bar ------
143     if ( line = wall_top  ) then
144         next_COLOR <= "11";
145     end if;
146 
147     ------------------------------- Display the playing field bottom bar ----
148     if ( line = wall_bottom ) then
149         next_COLOR <= "11";
150     end if;
151 
152     ------------------------------- Display the left Paddle -----------------
153     if (( pixel = left_x -1)  ) then
154         if ( (line >= left_y ) and (line <= (left_y + paddle_height)) ) then
155         next_COLOR <= "11";
156         end if;
157     end if;
158 
159     ------------------------------ Display the right Paddle -----------------
160     if ( pixel = right_x + 1 ) then
161         if ( (line >= right_y) and (line <= (right_y + paddle_height)) ) then
162         next_COLOR <= "11";
163         end if;
164     end if;
165 
```

```
166     ---------------------------------- Display the Ball --------------------------
167     if ( (pixel = ball_x) ) then
168         if ( line = ball_y ) then
169             next_COLOR <= "01";
170         end if;
171     end if;
172
173     -- Display the Left Score ( Using Std_Logic_Vectors instead of integers )
174     if ( clk = '1' and clk'event) then
175         if ((pixel >= "0001000" ) and ( pixel <= "0001011" )) and
176                 ((line >= "000000" ) and (line <= "000111" )) then
177                 number_address(6 downto 3) <= CONV_STD_LOGIC_VECTOR(lscore,4);
178         elsif ((pixel >= "01000000" ) and ( pixel <= "01000011" )) and
179                 ((line >= "000000" ) and (line <= "000111" )) then
180                 number_address(6 downto 3) <= CONV_STD_LOGIC_VECTOR(rscore,4);
181         else
182             number_address(6 downto 3) <= "0001";
183         end if;
184     end if;
185
186     ------------------------------------- Display the Left Score ------------------
187     if ((pixel >= "0001000") and (pixel <= "0001011" )) and
188         ((line >= "000000") and (line <= "000111" ) ) then
189             case pixel( 1 downto 0 ) is
190                 when "00" =>
191                     if ( number_data(3) = '1' ) then
192                         next_COLOR <= "10";
193                     end if;
194                 when "01" =>
195                     if ( number_data(2) = '1' ) then
196                         next_COLOR <= "10";
197                     end if;
198                 when "10" =>
199                     if ( number_data(1) = '1' ) then
200                         next_COLOR <= "10";
201                     end if;
202                 when "11" =>
203                     if ( number_data(0) = '1' ) then
204                         next_COLOR <= "10";
205                     end if;
206                 when others => NULL;
207             end case;
208     end if;
209
210     ---------------------------- Display the Right Score ---note alternative version------
211     if ((pixel >= "01000000") and (pixel <= "01000011" )) and
212         ((line >= "000000") and (line <= "000111")) then
213             if number_data( 3-conv_integer( pixel(1 downto 0)))='1' then
214                 next_COLOR <= "10";
215             end if;
216     end if;
217
218 end process;
219
220 ----------------------------- Game play logic ---------------------------------
221 moving_paddles: process (VCLK, reset)
222 begin
223     if (reset = '1') then
224         left_y          <= "001001";
225         right_y         <= "001001";
```

Game play logic

The code below will define the logic of the game. Would it be possible to implement ”World of warcraft” this way?

```
220  ---------------------------- Game play logic ----------------------------------
221  moving_paddles: process (VCLK, reset)
222  begin
223     if (reset = '1') then
224        left_y       <= "001001";
225        right_y      <= "001001";
226        nextleft_y   <= "001001";
227        nextright_y  <= "001001";
228     elsif (VCLK = '1' and VCLK'event) then
229
230        if (left_dir = "10") then
231           nextleft_y <= nextleft_y + 1;  -- move up
232        elsif (left_dir = "01") then
233           nextleft_y <= nextleft_y - 1;  -- move down
234        else
235           nextleft_y <= nextleft_y;      -- don't move
236        end if;
237
238        if (right_dir = "10") then
239           nextright_y <= nextright_y + 1;  -- move up
240        elsif (right_dir = "01") then
241           nextright_y <= nextright_y - 1;  -- move down
242        else
243           nextright_y <= nextright_y;      -- don't move
244        end if;
245
246        if (nextleft_y < 9) then
247           left_y      <= "001001";         -- stop at top of screen
248           nextleft_y  <= "001001";
249        elsif(nextleft_y > 50) then
250           left_y      <= "110010";         -- stop at bottom of screen
251           nextleft_y  <= "110010";
252        else
253           left_y <= nextleft_y;
254        end if;
255
256        if ( nextright_y < 9 ) then
257           right_y <= "001001";             -- stop at top of screen
258           nextright_y <= "001001";
259        elsif( nextright_y > 50 ) then
260           right_y <= "110010";             -- stop at bottom of screen
261           nextright_y <= "110010";
262        else
263           right_y <= nextright_y;
264        end if;
265      end if;
266  end process;
267
268  moving_ball: process (VCLK, ball_xdir, ball_ydir, ball_x, ball_y, reset)
269  begin
270     if (reset = '1') then
271        ball_x <= left_wall;
272        ball_y <= 32;
273        ball_yrate <= 1;
274        lscore <= 0;
275        rscore <= 0;
276        enable <= '0';
277        delay <= "000";
278     elsif (VCLK = '1' and VCLK'event) then
279        if (delay >= "100") then   --This value may be increased or decreased to
```

```
268 moving_ball: process (VCLK, ball_xdir, ball_ydir, ball_x, ball_y, reset)
269 begin
270     if (reset = '1') then
271         ball_x <= left_wall;
272         ball_y <= 32;
273         ball_yrate <= 1;
274         lscore <= 0;
275         rscore <= 0;
276         enable <= '0';
277         delay <= "000";
278     elsif (VCLK = '1' and VCLK'event) then
279         if (delay >= "100") then    --This value may be increased or decreased to
280                                     -- adjust the speed of the ball
281             delay <= "000";
282             if (SERVE = '1') then
283                 if (enable = '0' ) then
284                     ball_yrate <= 0;
285                     ball_y     <= 16;
286                 end if;
287                 ball_y <= 32;
288                 enable <= '1';
289             end if;
290             ----------------------------->>>>>>>>>>>> O >>>>>>>>>>>>>  Ball going Right
291             if (ball_xdir = '1') then  -- Horizontal Movement ( 1 = right )
292                 if (enable = '1') then
293                     ball_x <= ball_x + 1;
294                 end if;
295 
296                 -- check for hit on upper 1/4 of right paddle
297                 if  ((ball_x = right_x) and ( ball_y >= right_y - ball_height ) and
298                      (ball_y < right_y + paddle_heighta) ) then
299                     ball_xdir <= '0';
300                     -- if ball is going down
301                     if ( ball_ydir = '0' ) then
302                         if ( ball_yrate > 0 ) then
303                             ball_yrate <= ball_yrate - 1;
304                         else
305                             ball_yrate <= 1;
306                             ball_ydir <= '1';
307                         end if;
308 
309                     -- if ball is going up
310                     else
311                         if ( ball_yrate < 2 ) then
312                             ball_yrate <= ball_yrate + 1;
313                         else
314                             ball_yrate <= 2;
315                         end if;
316                     end if;
317 
318                 elsif  ((ball_x = right_x) and ( ball_y >= right_y + paddle_heighta) and
319                         (ball_y < right_y + paddle_heightb) ) then
320                     ball_xdir <= '0';
321 
322                 -- check for hit on lower half of right paddle
323                 elsif  ((ball_x = right_x) and ( ball_y >= right_y + paddle_heightb) and
324                         (ball_y <= right_y + paddle_heightc) ) then
325                     ball_xdir <= '0';
326                     -- if ball is going down
327                     if (ball_ydir = '0') then
328                         if (ball_yrate < 2) then
329                             ball_yrate <= ball_yrate + 1;
330                         else
331                             ball_yrate <= 2;
332                         end if;
333                     -- if ball is going up
334                     else
335                         if (ball_yrate > 0) then
336                             ball_yrate <= ball_yrate - 1;
337                         else
338                             ball_ydir <= '0';
339                             ball_yrate <= 1;
340                         end if;
341                     end if;
342 
```

```
343                -- Score for left team
344                else
345                   if (ball_x = right_wall) then
346                      ball_xdir <= '0';
347                      if (enable = '1') then
348                         if (lscore = 9) then
349                            lscore <= 0;
350                         else
351                            lscore <= lscore + 1;
352                         end if;
353                      end if;
354                      enable <= '0';
355                   end if;
356                end if;
357             -----------------------------<<<<<<<<<<<<< O <<<<<<<<<<<<  Ball going left
358             else
359                -- in middle of playing field
360                if ( enable = '1' ) then
361                   ball_x <= ball_x - 1;
362                end if;
363
364                if ( ball_x = left_x ) then
365                -- upper portion of paddle
366                   if (( ball_y >= left_y ) and ( ball_y < left_y + paddle_heighta )) then
367                      ball_xdir <= '1';
368                      -- if ball is going down
369                      if ( ball_ydir = '0' ) then
370                         if ( ball_yrate > 0 ) then
371                            ball_yrate <= ball_yrate - 1;
372                         else
373                            ball_ydir <= '1';
374                            ball_yrate <= 1;
375                         end if;
376                      -- if ball is going up
377                      else
378                         if ( ball_yrate < 2 ) then
379                            ball_yrate <= ball_yrate + 1;
380                         else
381                            ball_yrate <= 2;
382                         end if;
383                      end if;
384
385                   -- lower portion of paddle
386                   elsif  ( ( ball_y >= left_y + paddle_heightb) and
387                            ( ball_y <= left_y + paddle_heightc) ) then
388                      ball_xdir <= '1';
389                      -- if ball is going down
390                      if ( ball_ydir = '0' ) then
391                         if ( ball_yrate < 2 ) then
392                            ball_yrate <= ball_yrate + 1;
393                         else
394                            ball_yrate <= 2;
395                         end if;
396                      -- if ball is going up
397                      else
398                         if ( ball_yrate > 0 ) then
399                            ball_yrate <= ball_yrate - 1;
400                         else
401                            ball_ydir <= '0';
402                            ball_yrate <= 1;
403                         end if;
404                      end if;
405                   elsif (( ball_y >= left_y + paddle_heighta) and
406                            ( ball_y < left_y + paddle_heightb)) then
407                      ball_xdir <= '1';
408                   end if;
```

```
409
410             -- Score for right team
411             else
412                 if ( ball_x = left_wall ) then
413                     ball_xdir <= '1';
414                     if ( enable = '1' ) then
415                         if ( rscore = 9 ) then
416                             rscore <= 0;
417                         else
418                             rscore <= rscore + 1;
419                         end if;
420                     end if;
421                     enable <= '0';
422                 end if;
423             end if;
424         end if;
425
426         -- Vertical Movement ( 1 = up )
427         if ( ball_ydir = '1' ) then
428             if (ball_y <= wall_top) then
429                 ball_ydir <= '0';
430             else
431                 ball_y <= ball_y - ball_yrate;
432             end if;
433         else
434             if (ball_y >= wall_bottom) then
435                 ball_ydir <= '1';
436             else
437                 ball_y <= ball_y + ball_yrate;
438             end if;
439         end if;
440       else
441         delay       <= delay + '1';
442         ball_y      <= ball_y;
443         ball_yrate  <= ball_yrate;
444         ball_x      <= ball_x;
445         ball_ydir   <= ball_ydir;
446         ball_xdir   <= ball_xdir;
447     end if;
448   end if;
449 end process;
450
451 end static_display;
```