# Abstract

In this lab, we set out to create a version of the first video game: Pong. This required the use of a seven-segment display, 8x16 dot matrix, 3 input buttons, and a FPGA programed using a Xilinx ISE. The 8x16 dot matrix comprised of LEDs that are controlled using multiplexing. The game functioned similarly to Pong, except the ball only moved in one dimension and the paddles were stationary. The game winner would be whoever gets to 9 points first. In the end, the system successfully showed a functioning game, albeit a bit barebones. There are bugs where when the Reset button is pressed during a round, the ball moves to the middle left of the screen.

Table of Contents

[Abstract 1](#_Toc90497878)

[The Process 3](#_Toc90497879)

[Conclusion 3](#_Toc90497880)

[Appendix I – Figures 5](#_Toc90497881)

[Appendix II – Device Summary 7](#_Toc90497882)

[Appendix IIIa – The VHDL Code – Lab8\_top\_sch.vhd 8](#_Toc90497883)

[Appendix IIb – VHDL Code – bcd\_to\_7seg.vhd 19](#_Toc90497884)

[Appendix IIIc – VHDL Code – led\_8x16\_driver.vhd 21](#_Toc90497885)

[Appendix IIId – VHDL Code – Seg7Driver.vhd 25](#_Toc90497886)

# The Process

This lab was the most involved of the previous labs as it required multiple finite state machines. The overall game flow was relatively simple (Figure 1). On startup, the game is initialized by pressing the reset button (sw11). Then the ball will appear on the dot matrix on the far right side, (address 15). The game begins when player one presses their paddle button (sw12). The ball will begin to move from right to left at a speed of 2 Hz, or 2 pixels per second. When the ball reaches the far left (address 0), player two must press their paddle button (sw10) the moment the ball reaches that point. If the player holds the button before or doesn’t press it in time, player one will gain a point. If player two successfully hits the ball, the ball speed increases to 3Hz towards player one and the game continues until one player gets nine points. The speed of the ball maxes out at 9Hz. When the game ends, the reset button can be pressed to start a new game. It can also be pressed at any point during the game to restart the game. This required the use of 2 random state machines

The first of the two random state machines the main game control finite state machine (Figure 2). This machine has eleven states. The left and right shift states determine the ball movement, the left and right hit zone states determine if the ball was successfully hit at the right time, the left and right sHit states send out the sHit signal if the ball successfully hit, the left and right point state sends out a signal to the score counter when a player gains a point to update the seven segment display, and lastly the left and right win state ends the game when a player gets nine points. When the game control FSM is the left or right shift state, it sends the shift signal to the second random state machine: the ball movement FSM (Figure 3). The ball movement FSM is far simpler than the game control state machine. It only has five states, the first two are a part of the initialization process. When the reset button is pressed, it resets the address and column. The rest of the states dictate how the ball moves. HoldBall keeps the ball in its current place between pulses from a variable clock. RemoveBall removes the ball from its current address then sends a signal to move to the next address. AddBall sets the new address to be the current ball position to the new address. The signal game\_pulse controls the speed of the ball. Its dictated by the variable clock generator that will output a signal between 2 and 9 Hz depending on how far into the round the game is.

The other important blocks of the design are simple. The first is the score counter block. It sends out the score of the players to the seven-segment display driver to update the score on the seven-segment display. If the score somehow goes above 9, the display will only show an ‘E’ for error. The next block is the variable ‘m’ generator. ‘m’ is the number saved in RAM that is used to calculate the frequency of the game\_pulse signal. It increases the speed after four successful hits. The block game\_f\_gen uses the ‘m’ signal to output the game pulse. The address incrementor block is what determines the address the ball is currently at depending on the signal from the ball movement FSM. The last two blocks are drivers for the seven segment display and the dot matrix display provided by Dr. Larry Aamodt.

# Conclusion

The overall game works, albeit barebones. The ball moves across the dot matrix display at the desired speed with no lag, the hit registration works consistently, and the score will increment with every point gained. The seven-segment display will show the correct score and properly display the colon. We have encountered some bugs, however.

One of the bugs that we’ve encountered we believe are due to the derived clock speed. There was an issue we encountered where the score would stop at ten instead of nine. We believe this was because the derived clock was faster than the system could check if a player had reached nine points. This allowed the score to go up to ten before the game ended. We did a quick fix by having the system check if the score was at eight instead of nine, thus ending the game at nine. Another bug we encountered involved the reset function. When the reset button is pressed when the ball is in movement or during a round, the ball will reset its position to a little left of the middle of the screen. Pressing the reset button again will reset the ball position to the correct spot on the rightmost pixel of the dot matrix display. The cause of this bug is unknown as of the writing of this report, but it could be due to code in init\_b1 or init\_b2 where the value of DATA is set or with the reset\_addr signal.

There are improvements that could be made to the design. The first, most basic, is the inclusion of visual paddles. The next is resetting the ball after a successful point. We wanted the round to reset after a point is scored the leftmost or rightmost pixel on the dot matrix display depending on who scored the point. If player 2 scored the point, player 1 would have the ball. If player 1 scored the point, player 2 would have the ball. The last would be to increase the maximum speed of the ball. It currently maxes out at 9 Hz, which could be hard for some players, but most people would only find it moderately difficult. Having a much higher maximum speed would provide all players some difficulty as the narrow hit window already provides a decent amount of difficulty. We didn’t include these so that we could make sure that the basic game functions worked, but ran out of time to add them in. With better time management and planning of the project, we could have added these features and possibly other features like having moving paddles and/or having the ball move diagonally.

In conclusion, the final design is a functional game that works as intended, with some minor bugs. The bugs are most likely simple fixes, but we ran out of time to really look into the issue as we discovered it on the last day. Some of the improvements should theoretically not take much effort or coding to implement, like the increased ball speed or the visual paddles. The game is still fun to play regardless of the bugs or missing features. And that, to me, feels like a success.

Appendix I – FiguresDiagram

Description automatically generated

Figure The Block Diagram

Timeline

Description automatically generated

Figure The Game Control FSM

Diagram

Description automatically generated

Figure The Ball Movement FSM

Diagram

Description automatically generated

Figure The Timing Diagram

# Appendix II – Device Summary

|  |  |
| --- | --- |
| Description | Stat |
| Slice F/F | 86/126800 + 4 latches |
| Occupied Slices | 51/15850 |
| Slices Unrelated Logic | N/A |
| Bonded IOBs | 49/300 |
| Number of BUFGs | 2/32 |
| Avg Fanout | 3.33 |

# Appendix IIIa – The VHDL Code – Lab8\_top\_sch.vhd

----------------------------------------------------------------------------------

-- Company: Walla Walla University

-- Engineer: Eric Walsh and Nicholas Zimmerman

--

-- Create Date: 15:50:34 11/16/2021

-- Design Name:

-- Module Name: Lab8\_top\_sch - Behavioral

-- Project Name: Pong\_1d

-- 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;

library UNISIM;

use UNISIM.VComponents.all;

entity Lab8\_top\_sch is

port( mclk : in std\_logic;

btn1, btn2, btn3 : in std\_logic;

led : out std\_logic\_vector(15 downto 0);

--7-segment display

cath : out std\_logic\_vector(7 downto 0);

anode : out std\_logic\_vector(5 downto 1); --We must include unused anodes and drive them to '1'

extout : out std\_logic\_vector(8 downto 0);

tek4 : out std\_logic\_vector(1 downto 0);

tek1 : out std\_logic\_vector(7 downto 0);

tek5 : out std\_logic);

end Lab8\_top\_sch;

architecture SingleFSM of Lab8\_top\_sch is

----signals should be grouped by block of signal origin----

signal R : std\_logic; --An alias for btn2

--next\_game\_f\_gen

--signal c : unsigned(3 downto 0); --Allows 15 frequencies, 1 to 15

signal sHit\_cnt\_next, sHit\_cnt\_reg, sHit\_cnt\_next\_en : unsigned(1 downto 0); --2 bit counter of successful hits, should be 1 bit TFF

signal select\_f\_next, select\_f\_reg, select\_f\_notMax : unsigned(2 downto 0); --:= (others=>'0'); --attempting RAM of 8 predetermined frequencies

signal m : unsigned(25 downto 0); --:= to\_unsigned(50000000,26); --26 bit unsigned value,

signal select\_f\_forOutput\_next, select\_f\_forOutput\_reg : std\_logic\_vector(7 downto 0); --:= (others=>'0');

--game\_f\_gen --generates constant sys\_clk and variable game\_pulse which counts in units of sys\_clk

signal clk\_reg, clk\_next, clk\_next\_check : unsigned(25 downto 0); --2^26 = 67,108,864 > 50E6; 32 bits to compare with m

--signal clk\_overflow : std\_logic;

--signal t\_reg, t\_next: std\_logic;

--signal game\_clk : std\_logic;

signal game\_pulse : std\_logic;

--control\_FSM

type control\_state\_type is (init\_c, lSh,lHit\_zone,rPt,rWin,l\_sHit, --Seems to initialize to 1st state in list

rSh,rHit\_zone,lPt,lWin,r\_sHit );

signal control\_state\_reg, control\_state\_next : control\_state\_type;

signal rHit,lHit : std\_logic; --Logical high of btn3,btn1 respectively

signal sHit : std\_logic; --Successful hit by either player

signal p1\_pt, p2\_pt : std\_logic; --Increment score counter

signal dir,shift : std\_logic; --dir: 0 shifts left, 1 shifts right

signal DATA : std\_logic\_vector(7 downto 0);

signal WE : std\_logic;

--score\_counter

signal p1\_score\_reg, p1\_score\_next, p2\_score\_reg, p2\_score\_next: unsigned(3 downto 0); --Seems to initialize to 0

signal rW, lW : std\_logic; --Win condition met

--ball\_movement\_FSM

type ball\_state\_type is (init\_b1, init\_b2, holdBall, removeBall, addBall);

signal ball\_state\_reg, ball\_state\_next : ball\_state\_type;

signal incr\_addr : std\_logic;

signal reset\_addr : std\_logic;

--address\_incrementer

signal addr\_reg : unsigned(3 downto 0); --:= to\_unsigned(15,4); --Point to the rightmost part of the screen

signal addr\_next, enabled\_addr\_next, overwritten\_addr\_next : unsigned(3 downto 0);

signal ADDR : std\_logic\_vector(3 downto 0); --std\_logic\_vector data type

---Components---

--seg7Driver

component Seg7Driver

port( mclk : in std\_logic;

p1\_score, p2\_score : in unsigned(3 downto 0);

to\_anode2, to\_anode4 : out std\_logic;

to\_cathode : out std\_logic\_vector(7 downto 0) );

end component;

--dot\_matrix\_driver

component led\_8x16\_driver

port( mclk: in STD\_LOGIC; -- 50 Mhz clock

sys\_clk: in STD\_LOGIC; -- clock from game

we: in STD\_LOGIC; -- write enable

data\_in: in STD\_LOGIC\_VECTOR(7 downto 0);

wrt\_addr: in STD\_LOGIC\_VECTOR(3 downto 0);

out\_to\_display: out STD\_LOGIC\_VECTOR(7 downto 0) );

end component;

begin

-----------------------------------------------------------------------------

--Next\_game\_f\_gen (50 MHz to 1 Hz) (not yet resettable)

-----------------------------------------------------------------------------

--Successful Hit counter

process(mclk)

begin

if (mclk'event and mclk='1') then

if(R='1') then

select\_f\_reg <= (others=>'0');

select\_f\_forOutput\_reg <= (others=>'0');

else

-- sHit\_cnt\_reg <= sHit\_cnt\_next; --For counting number of successful hits

select\_f\_reg <= select\_f\_next;

select\_f\_forOutput\_reg <= select\_f\_forOutput\_next;

end if;

end if;

end process;

--NS Logic

-- sHit\_cnt\_next <= sHit\_cnt\_next\_en when (sHit='1') else sHit\_cnt\_reg; --enable

-- sHit\_cnt\_next\_en <= (others => '0') when (sHit\_cnt\_reg=3) else (sHit\_cnt\_reg+1);

--Update the frequency every 4 non-consecutive successful hits

-- select\_f\_next <= (select\_f\_reg+1) when (sHit='1' and sHit\_cnt\_reg=3) else select\_f\_reg;

select\_f\_next <= select\_f\_reg when (select\_f\_reg=7) else select\_f\_notMax; --Hold value instead of cycling back to 0

select\_f\_notMax <= (select\_f\_reg+1) when (sHit='1') else select\_f\_reg;

--Output Logic

--RAM for m --tried a case statement earlier

with select\_f\_reg select

m <= to\_unsigned(25000000,26) when "000", --2Hz

to\_unsigned(16666666,26) when "001", --3Hz

to\_unsigned(12500000,26) when "010", --4Hz

to\_unsigned(10000000,26) when "011", --5Hz

to\_unsigned(8333333,26) when "100", --6Hz

to\_unsigned(7142857,26) when "101", --7Hz

to\_unsigned(6250000,26) when "110", --8Hz

to\_unsigned(5555555,26) when others; --111 --9Hz

--c generation --unused

--c <= to\_unsigned(1,4); --Magic number for now

select\_f\_forOutput\_next <= ('1' & select\_f\_forOutput\_reg(7 downto 1)) when (sHit='1') else select\_f\_forOutput\_reg;

-----------------------------------------------------------------------------

--Game\_f\_gen (50 MHz to 1 Hz) (not yet resettable)

-----------------------------------------------------------------------------

process(mclk)

begin

if (mclk'event and mclk='1') then

clk\_reg <= clk\_next;

end if;

end process;

--NS Logic

--clk\_overflow <= '1' when (clk\_reg >= 24999999) else '0';

--clk\_next <= (others=>'0') when (clk\_overflow='1') else (clk\_reg + c); --Possible timing hazard with c on overflow, so we won't use it

--t\_reg <= (not t\_reg) when (clk\_overflow='1') else t\_reg;

clk\_next\_check <= (others=>'0') when (clk\_reg = 49999999) else clk\_next; --make sure clk\_reg is reset when m is changed

clk\_next <= (others=>'0') when (clk\_reg = (m-1)) else (clk\_reg + 1);

--Output Logic

--Clk\_Buffer: BUFG -- Put t\_reg on a buffered clock line

-- port map ( I => t\_reg, O => game\_clk);

--game\_pulse <= '1' when (clk\_overflow='1' and t\_reg='1') else '0';

game\_pulse <= '1' when (clk\_reg = (m-1)) else '0'; --comparison of 26 bit unsigned values

--It would be a good idea to buffer the above signal signal from glitches

--Note: There will need a comparator for variable c but not for variable m

-----------------------------------------------------------------------------

--Score\_counter

-----------------------------------------------------------------------------

process(mclk)

begin

if (mclk'event and mclk='1') then

if(R='1') then

p1\_score\_reg <= (others=>'0');

p2\_score\_reg <= (others=>'0');

else

p1\_score\_reg <= p1\_score\_next;

p2\_score\_reg <= p2\_score\_next;

end if;

end if;

end process;

--NS Logic

p1\_score\_next <= p1\_score\_reg+1 when (p1\_pt='1') else p1\_score\_reg;

p2\_score\_next <= p2\_score\_reg+1 when (p2\_pt='1') else p2\_score\_reg;

--Out Logic

rW <= '1' when (p1\_score\_reg=8) else '0'; --right player wins, should win on 9... but timing

lW <= '1' when (p2\_score\_reg=8) else '0'; --left player wins, should win on 9... but timing

-----------------------------------------------------------------------------

--Control\_FSM

-----------------------------------------------------------------------------

process(mclk)

begin

--if (R='1') then --asynchronous reset?

--control\_state\_reg <= init;

--end if?? else?

if (mclk'event and mclk='1') then

if (R='1') then --synchronous reset

control\_state\_reg <= init\_c;

else

control\_state\_reg <= control\_state\_next;

end if;

end if;

--end if??

end process;

--NS Logic

process(control\_state\_reg)

begin

case control\_state\_reg is

when init\_c =>

if (rHit='1') then

control\_state\_next <= lSh;

else

control\_state\_next <= init\_c;

end if;

when lSh =>

if (addr\_reg /= 0) then

control\_state\_next <= lSh;

else --addr\_reg=0

if (lHit='1') then

control\_state\_next <= rPt;

else

control\_state\_next <= lHit\_zone;

end if;

end if;

when lHit\_zone =>

if (lHit='1') then

control\_state\_next <= l\_sHit;

else --lHit='0'

if (game\_pulse='1') then

control\_state\_next <= rPt;

else

control\_state\_next <= lHit\_zone;

end if;

end if;

when rPt =>

if (rW='1') then

control\_state\_next <= rWin;

else

control\_state\_next <= rSh;

end if;

when rWin =>

when l\_sHit =>

control\_state\_next <= rSh;

when rSh =>

if (addr\_reg /= 15) then

control\_state\_next <= rSh;

else --addr\_reg=15

if (rHit='1') then

control\_state\_next <= lPt;

else

control\_state\_next <= rHit\_zone;

end if;

end if;

when rHit\_zone =>

if (rHit='1') then

control\_state\_next <= r\_sHit;

else --lHit='0'

if (game\_pulse='1') then

control\_state\_next <= lPt;

else

control\_state\_next <= rHit\_zone;

end if;

end if;

when lPt =>

if (lW='1') then

control\_state\_next <= lWin;

else

control\_state\_next <= lSh;

end if;

when lWin =>

when r\_sHit =>

control\_state\_next <= lSh;

end case;

end process;

--Out Logic

process(control\_state\_reg)

begin

shift <= '0';

dir <= '0';

p1\_pt <= '0';

p2\_pt <= '0';

sHit <= '0'; --Leaving this out was a beginner's mistake! It broke the clock.

case control\_state\_reg is

when init\_c =>

when lSh =>

shift <= '1';

--dir <= '0';

when lHit\_zone =>

when rPt =>

p1\_pt <= '1';

when rWin =>

when l\_sHit =>

sHit <= '1';

when rSh =>

shift <= '1';

dir <= '1';

when rHit\_zone =>

when lPt =>

p2\_pt <= '1';

when lWin =>

when r\_sHit =>

sHit <= '1';

end case;

end process;

-----------------------------------------------------------------------------

--Ball\_Movement\_FSM

-----------------------------------------------------------------------------

process(mclk)

begin

if (mclk'event and mclk='1') then

if (R='1') then --synchronous reset

ball\_state\_reg <= init\_b1;

else

ball\_state\_reg <= ball\_state\_next;

end if;

end if;

end process;

--NS Logic

process(ball\_state\_reg) --We're choosing not to have game\_pulse in the sensitivity list because glitches

begin

case ball\_state\_reg is

when init\_b1 =>

ball\_state\_next <= init\_b2;

when init\_b2 =>

ball\_state\_next <= holdBall;

when holdBall =>

if (shift='1' and game\_pulse='1') then

ball\_state\_next <= removeBall;

else

ball\_state\_next <= holdBall;

end if;

when removeBall =>

ball\_state\_next <= addBall;

when addBall =>

ball\_state\_next <= holdBall;

end case;

end process;

--Out Logic

process(ball\_state\_reg)

begin

incr\_addr <= '0';

DATA <= "10101010";

WE <= '0';

reset\_addr <= '0';

case ball\_state\_reg is

when init\_b1 =>

--Change the next address

reset\_addr <= '1'; --Change the next address

--Write in the the current addressed location

DATA <= (others=>'0'); --Set the currently addressed column

WE <= '1';

when init\_b2 =>

DATA <= "00000001";

WE <= '1';

when holdBall =>

when removeBall =>

incr\_addr <= '1';

DATA <= (others=>'0'); --Redundant statement?

WE <= '1';

--Wait longer to write value? (use a counter)

when addBall =>

DATA <= "00000001";

WE <= '1';

--Wait longer to write value? (use a counter)

end case;

end process;

-----------------------------------------------------------------------------

--Address\_incrementer

-----------------------------------------------------------------------------

process(mclk)

begin

if (mclk'event and mclk='1') then

addr\_reg <= addr\_next;

end if;

end process;

--NS Logic

overwritten\_addr\_next <= to\_unsigned(15,4) when (reset\_addr ='1') else addr\_next; --override

addr\_next <= (enabled\_addr\_next) when (incr\_addr='1') else addr\_reg; --enable

enabled\_addr\_next <= (addr\_reg+1) when (dir='1') else (addr\_reg-1);

--Output Logic

ADDR <= std\_logic\_vector(addr\_reg);

-----------------------------------------------------------------------------

--Top level code

-----------------------------------------------------------------------------

rHit <= not btn3; --player 1

lHit <= not btn1; --player 2

R <= not btn2; --Initialize/Reset

Seg7Driver\_1: Seg7Driver port map( mclk => mclk,

p1\_score => p1\_score\_reg,

p2\_score => p2\_score\_reg,

to\_anode2 => anode(2),

to\_anode4 => anode(4),

to\_cathode => cath );

anode(1) <= '1'; --unused

anode(3) <= '1'; --unused

anode(5) <= '1'; --unused

dotMatrixDriver : led\_8x16\_driver

port map ( mclk => mclk,

sys\_clk => mclk,

we => WE,

data\_in => DATA,

wrt\_addr => ADDR,

out\_to\_display => extout(7 downto 0) );

--Score

led(7 downto 0) <= std\_logic\_vector(p2\_score\_reg) & std\_logic\_vector(p1\_score\_reg);

--led(15 downto 8) <= std\_logic\_vector(m(15 downto 8));

led(15 downto 8) <= lW & "000000" & rW;

--led(15 downto 8) <= select\_f\_forOutput\_reg;

tek4(0) <= WE;

tek1(0) <= WE;

tek4(1) <= game\_pulse; --was DATA(0)

tek1(1) <= game\_pulse; --was DATA(0)

tek1(2) <= game\_pulse;

extout(8) <= mclk;

tek1(7) <= mclk;

tek5 <= mclk;

end SingleFSM;

# Appendix IIb – VHDL Code – bcd\_to\_7seg.vhd

----------------------------------------------------------------------------------

-- Company: Walla Walla University

-- Engineer: L.Aamodt

--

-- Create Date: 16:42:06 10/13/2020

-- Design Name: Binary to 7-segment decoder

-- Module Name: bcd2\_7seg - Behavioral

-- Project Name:

--

-- Description: Converts 4-bit binary representing one BCD digit

-- to 7-segment display encoding

-- Dependencies: WWU FPGA3 board

--

-- Revision:

-- Revision 1.0 - File Created

-- Additional Comments:

-- Cathode signals asserted low to turn on the segment.

-- The cathode that controls the decimal point is always turned off.

-- Segment a of the display is cath\_out(0) and segment g is cath\_out(6).

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity bcd2\_7seg is

port ( data : in STD\_LOGIC\_VECTOR (3 downto 0);

cath\_out : out STD\_LOGIC\_VECTOR (7 downto 0));

end bcd2\_7seg;

architecture Behavioral of bcd2\_7seg is

begin

process(data)

begin

case data is

when "0000" => cath\_out <= "11000000";

when "0001" => cath\_out <= "11111001";

when "0010" => cath\_out <= "10100100";

when "0011" => cath\_out <= "10110000";

when "0100" => cath\_out <= "10011001";

when "0101" => cath\_out <= "10010010";

when "0110" => cath\_out <= "10000010";

when "0111" => cath\_out <= "11111000";

when "1000" => cath\_out <= "10000000";

when "1001" => cath\_out <= "10011000";

when others => cath\_out <= "10000110";

end case;

end process;

end Behavioral;

# Appendix IIIc – VHDL Code – led\_8x16\_driver.vhd

----------------------------------------------------------------------------------

-- Company: Walla Walla University

-- Engineer: L.Aamodt

--

-- Create Date: 21:14:50 12/06/2015

-- Design Name: led\_8x16\_driver

-- Module Name: led\_8x16\_driver - Behavioral

-- Project Name: led\_8x16\_driver

-- Target Devices: Spartan 3e

-- Tool versions: ISE 14.7

-- Description: Driver to send data to a 8x16 LED array arranged as

-- 8 rows of 16 bits (i.e. 16 columns). Data is read

-- asynchronously as bytes from a 16 byte RAM, bits in

-- the byte are selected sequentially with a multiplexer,

-- and together with an address are used to select one

-- LED at a time for potential turn on. All 128 bits are

-- scanned about 79 times per second.

--

-- Dependencies:

--

-- Revision:

-- Revision 2.0 12/7/16

-- Additional Comments:

--

----------------------------------------------------------------------------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity led\_8x16\_driver is

Port (mclk: in STD\_LOGIC; -- 50 Mhz clock

sys\_clk: in STD\_LOGIC; -- clock from game

we: in STD\_LOGIC; -- write enable

data\_in: in STD\_LOGIC\_VECTOR(7 downto 0);

wrt\_addr: in STD\_LOGIC\_VECTOR(3 downto 0);

out\_to\_display: out STD\_LOGIC\_VECTOR(7 downto 0));

end led\_8x16\_driver;

architecture Behavioral of led\_8x16\_driver is

signal r\_reg1: unsigned(11 downto 0);

signal r\_next1: unsigned(11 downto 0);

signal led\_addr: unsigned(6 downto 0);

signal led\_next: unsigned(6 downto 0);

signal ram\_data: STD\_LOGIC\_VECTOR(7 downto 0);

signal clk\_20k: STD\_LOGIC;

signal clk\_10k: STD\_LOGIC;

signal Bclk\_10k: STD\_LOGIC;

signal led\_data: STD\_LOGIC;

type ram\_type is array (0 to 15) of std\_logic\_vector(7 downto 0);

signal RAM: ram\_type := (

-- X"81", X"82", X"84", X"88", -- initial data, pattern 1

-- X"44", X"42", X"41", x"40",

-- X"81", X"82", X"84", X"88",

-- X"44", X"42", X"41", x"40"

-- X"01", X"02", X"04", X"08", -- initial data, pattern 2

-- X"10", X"20", X"40", x"80",

-- X"80", X"40", X"20", x"10",

-- X"08", X"04", X"02", x"01"

X"00", X"00", X"00", X"00", -- initial data, pattern 3

X"00", X"00", X"00", x"00",

X"00", X"00", X"00", x"00",

X"00", X"00", X"00", x"00"

);

begin

------------------------------------------------------------

-- Scan clock generator - 10khz output to run LED update

------------------------------------------------------------

process(mclk,r\_reg1) -- create 20000hz signal

begin

if (mclk'event and mclk='1') then

if (r\_reg1 = 2499) then

r\_reg1 <= (others=>'0');

else

r\_reg1 <= r\_next1;

end if;

end if;

end process;

r\_next1 <= r\_reg1 + 1;

clk\_20k <= '1' when r\_reg1 = 2499 else '0';

process(mclk,clk\_20k) -- create square 10khz clock

begin

if (mclk'event and mclk='1') then

if (clk\_20k = '1') then

clk\_10k <= NOT clk\_10k;

end if;

end if;

end process;

Clk\_Buffer: BUFG -- create 10khz buffered clock

port map ( I => clk\_10k, O => Bclk\_10k);

------------------------------------------------------------

-- Dual ported RAM that holds an image of what is to be displayed

-- synchronous write and asynchronous read

------------------------------------------------------------

process(sys\_clk)

begin

if (sys\_clk'event and sys\_clk = '1') then

if (we = '1') then

RAM(to\_integer(unsigned(wrt\_addr))) <= data\_in;

end if;

end if;

end process;

ram\_data <= RAM(to\_integer(unsigned(led\_addr(6 downto 3))));

------------------------------------------------------------

-- LED address generator, a 7 bit binary counter

------------------------------------------------------------

process(Bclk\_10k)

begin

if (Bclk\_10k'event and Bclk\_10k='1') then

if (led\_addr = 127) then

led\_addr <= (others=>'0');

else

led\_addr <= led\_next;

end if;

end if;

end process;

led\_next <= led\_addr + 1;

out\_to\_display(7 downto 1) <= std\_logic\_vector(led\_addr);

-------------------------------------------------------------

-- Multiplexer to get one led data bit from a data byte

-------------------------------------------------------------

process(led\_addr, ram\_data)

begin

case led\_addr(2 downto 0) is

when "000" =>

led\_data <= ram\_data(7);

when "001" =>

led\_data <= ram\_data(6);

when "010" =>

led\_data <= ram\_data(5);

when "011" =>

led\_data <= ram\_data(4);

when "100" =>

led\_data <= ram\_data(3);

when "101" =>

led\_data <= ram\_data(2);

when "110" =>

led\_data <= ram\_data(1);

when others =>

led\_data <= ram\_data(0);

end case;

end process;

out\_to\_display(0) <= led\_data;

end Behavioral;

# Appendix IIId – VHDL Code – Seg7Driver.vhd

----------------------------------------------------------------------------------

-- Company:

-- Engineer:

--

-- Create Date: 15:35:14 12/12/2021

-- Design Name:

-- Module Name: Seg7Driver - BasedOnLab5

-- 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 instantiating

-- any Xilinx primitives in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Seg7Driver is

port( mclk : in std\_logic;

p1\_score, p2\_score : in unsigned(3 downto 0);

to\_anode2, to\_anode4 : out std\_logic;

to\_cathode : out std\_logic\_vector(7 downto 0) );

end Seg7Driver;

architecture BasedOnLab5 of Seg7Driver is

--10kHz TFF used to toggle between 2 anodes

signal cnt\_reg, cnt\_next : unsigned(12 downto 0); --2^13=8192

signal t\_reg, t\_next : std\_logic;

--Routing

signal binary\_under\_1001 : std\_logic\_vector(3 downto 0);

--Components

component bcd2\_7seg

port ( data : in STD\_LOGIC\_VECTOR (3 downto 0);

cath\_out : out STD\_LOGIC\_VECTOR (7 downto 0) );

end component;

begin

--1 bit counter

--Register

process(mclk)

begin

if (mclk'event and mclk='1') then

cnt\_reg <= cnt\_next;

t\_reg <= t\_next;

end if;

end process;

--NS Logic

cnt\_next <= (others=>'0') when (cnt\_reg=2499) else (cnt\_reg+1);

t\_next <= (not t\_reg) when (cnt\_reg=2499) else t\_reg; --10kHz

--Generate to\_cathode

binary\_under\_1001 <= std\_logic\_vector(p1\_score) when (t\_reg='1') else std\_logic\_vector(p2\_score);

bcdTo7SegDecoder: bcd2\_7seg port map (binary\_under\_1001,to\_cathode);

--Generate to\_anode

to\_anode2 <= t\_reg; --p2, pulled down to 0(activated) when cnt\_reg=0which selects p2\_score

to\_anode4 <= not t\_reg; --p1, pulled down to 0(activated) when cnt\_reg=1 which selects p1\_score

end BasedOnLab5;