Lab 4 code

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

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

-- Class: CPE 3020, Prof. Tippens

-- Coded by: Nick Ray and Julian Duran

-- Module Name: Lab 4 – State Machines

-- The purpose of is this lab is to create a pong game using state machines. The game starts

-- with player one serving, and the serve goes to whoever misses their hit each round. Each

-- time a paddle is hit, the speed of the ball increases. The seven segs update each time a point

-- is scored. The game goes up to five which then the score of the winner will display the middle

-- segment of the corresponding seven segment display.

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

entity lab4 is

port (

clk: in std\_logic;

btnL: in std\_logic;

btnR: in std\_logic;

btnD: in std\_logic;

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

seg: out std\_logic\_vector(6 downto 0);

an: out std\_logic\_vector(3 downto 0)

);

end lab4;

architecture lab4behav of lab4 is

--Constants for seven segs--

constant nothing: std\_logic\_vector(6 downto 0) := "1111111";

constant zero: std\_logic\_vector(6 downto 0) := "1000000";

constant one: std\_logic\_vector(6 downto 0) := "1111001";

constant two: std\_logic\_vector(6 downto 0) := "0100100";

constant three: std\_logic\_vector(6 downto 0) := "0110000";

constant four: std\_logic\_vector(6 downto 0) := "0011001";

constant five: std\_logic\_vector(6 downto 0) := "0010010";

constant center: std\_logic\_vector(6 downto 0) := "0111111";

--Constants for enabling signals and terminal value for onekhz clock--

constant NOTACTIVE: std\_logic := '0';

constant ACTIVE: std\_logic := '1';

constant TERMINAL\_VALUE: integer := 999999;

--Onekhz clock signal--

signal onekhz: std\_logic;

signal SysReset: std\_logic;

--counter for player scores--

signal p1count: integer range 0 to 5;

signal p2count: integer range 0 to 5;

--For seven segs to display scores--

signal p1score: std\_logic\_vector (6 downto 0);

signal p2score: std\_logic\_vector (6 downto 0);

signal p1victory: std\_logic\_vector (6 downto 0);

signal p2victory: std\_logic\_vector (6 downto 0);

--Signal to enable a point for player--

signal p1point: std\_logic;

signal p2point: std\_logic;

signal reset: std\_logic := '0';

signal onehz: std\_logic;

--Btn Debouncing--

signal Rvector: std\_logic\_vector (2 downto 0);

signal Routput: std\_logic;

signal Dvector: std\_logic\_vector (2 downto 0);

signal Doutput: std\_logic;

signal Lvector: std\_logic\_vector (2 downto 0);

signal p2btn: std\_logic;

signal p1btn: std\_logic;

signal p2btnraw: std\_logic;

signal p1btnraw: std\_logic;

--Check for Cheating--

signal p1btnflag: std\_logic;

signal p2btnflag: std\_logic;

signal gamereset: std\_logic;

--Misc. signals--

shared variable nextflag: std\_logic;

signal digitsel: integer range 0 to 3;

signal winnerp1: std\_logic;

signal winnerp2: std\_logic;

--States for State Machine--

type gamestates is (p1serve, p1event, p1miss, MoveR1, MoveR2, MoveR3, MoveR4, MoveR5, MoveR6,

MoveR7, MoveR8, MoveR9, MoveR10, MoveR11, MoveR12, MoveR13, MoveR14, p2serve, p2event, p2miss,

MoveL1, MoveL2, MoveL3, MoveL4, MoveL5, MoveL6, MoveL7, MoveL8, MoveL9, MoveL10, MoveL11,

MoveL12, MoveL13, MoveL14, p1win, p2win);

signal currentstate: gamestates;

signal nextstate: gamestates;

begin

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

--Debouncing of down button and setting output as gamereset

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

D\_button\_debounce\_shift: process (onekhz, btnD, Dvector)

begin

if (rising\_edge(onekhz)) then

Dvector<=Dvector(1 downto 0)&btnD;

end if;

if (Dvector="111") then

Reset <= '1';

else

Reset <= '0';

end if;

end process;

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

--Debouncing of Right button and assigning output to p2btn

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

R\_button\_debounce\_shift: process (clk, btnR, Rvector)

begin

if (rising\_edge(clk)) then

Rvector<=Rvector(1 downto 0)&btnR;

end if;

if (Rvector="111") then

p2btnraw <= '1';

else

p2btnraw <= '0';

end if;

end process;

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

--Debouncing of left button and assigning output to p1btn

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

L\_button\_debounce\_shift: process (clk, btnL, Lvector)

begin

if (rising\_edge(clk)) then

Lvector<=Lvector(1 downto 0)&btnL;

end if;

if (Lvector="111") then

p1btnraw <= '1';

else

p1btnraw <= '0';

end if;

end process;

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

--Processes the button inputs for P1 and P2

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

Button\_processing: process(p1btnraw, p2btnraw, clk)

begin

if(rising\_edge(clk)) then

if(nextflag='0') then

if(p1btnraw = '1') then

p1btn <= '1';

end if;

if (p2btnraw = '1') then

p2btn <= '1';

end if;

elsif(nextflag='1') then

p1btn <= '0';

p2btn <= '0';

end if;

end if;

end process;

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

--Controls the speed of the ball (leds)

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

Ballspeed\_process: process(Reset, clk)

variable Ballspeed: integer :=50000000;

variable counter3: integer range 0 to 50000000;

begin

onehz <= not ACTIVE;

if (Reset = ACTIVE) then

counter3 := 0;

Ballspeed := 50000000;

elsif (rising\_edge(clk)) then

if (counter3 = Ballspeed) then

onehz <= ACTIVE;

counter3 := 0;

if (currentstate = p2event) then

Ballspeed := Ballspeed - 4000000;

elsif (currentstate = p1event) then

Ballspeed := Ballspeed - 4000000;

elsif (currentstate = p1serve) then

Ballspeed := 50000000;

elsif (currentstate = p2serve) then

Ballspeed := 50000000;

else

Ballspeed := Ballspeed;

end if;

else

onehz <= not ACTIVE;

counter3 := counter3 + 1;

end if;

end if;

end process;

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

--Displays middle segment of 7 seg to display player one has won

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

process (clk,Reset)

begin

if (rising\_edge(clk)) then

if (winnerp1=ACTIVE) then

p1victory <= center;

end if;

end if;

end process;

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

--Displays middle segment of 7 seg to display player two has won

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

process (clk,Reset)

begin

if (rising\_edge(clk)) then

if (winnerp2=ACTIVE) then

p2victory <= center;

end if;

end if;

end process;

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

--Transition Register telling current state what next will be

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

Pong\_State\_register: process(Reset, clk)

begin

nextflag := '0';

if (Reset = ACTIVE) then

currentstate <= p1serve;

--nextflag <= '0';

elsif (rising\_edge(clk)) then

if (onehz = ACTIVE) then

currentstate <= nextstate;

nextflag := '1';

end if;

end if;

--nextflag <= '0';

end process;

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

--Transition Register telling current state what next will be

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

Pong\_State\_Transition: process(currentstate)

begin

case currentstate is

when p1serve =>

winnerp1 <= not ACTIVE;

winnerp2 <= not ACTIVE;

led <= "1000000000000000";

p1point <= NOTACTIVE;

p2point <= NOTACTIVE;

if (p1btn= ACTIVE) then

nextstate <= MoveR1;

else

nextstate <= p1serve;

end if;

when p1miss =>

led <= "0000000000000000";

if (onehz = ACTIVE) then

nextstate <= p1serve;

p2point <= ACTIVE;

if (p2count=4) then

nextstate <= p2win;

end if;

else

nextstate <= p1miss;

end if;

when p1event =>

led <= "1000000000000000";

if (onehz=ACTIVE) then

if (p1btnflag = not ACTIVE) then

if (p1btn=ACTIVE) then

nextstate <= MoveR1;

else

nextstate <= p1miss;

end if;

else

nextstate <= p1miss;

end if;

else

nextstate <= p1event;

end if;

when MoveR1 =>

led <= "0100000000000000";

if (onehz=ACTIVE) then

nextstate <= MoveR2;

else

nextstate <= MoveR1;

end if;

when MoveR2 =>

led <= "0010000000000000";

if (onehz = ACTIVE) then

nextstate <= MoveR3;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR2;

end if;

when MoveR3 =>

led <= "0001000000000000";

if (onehz = ACTIVE) then

nextstate <= MoveR4;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR3;

end if;

when MoveR4 =>

led <= "0000100000000000";

if (onehz = ACTIVE) then

nextstate <= MoveR5;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR4;

end if;

when MoveR5 =>

led <= "0000010000000000";

if (onehz = ACTIVE) then

nextstate <= MoveR6;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR5;

end if;

when MoveR6 =>

led <= "0000001000000000";

if (onehz = ACTIVE) then

nextstate <= MoveR7;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR6;

end if;

when MoveR7 =>

led <= "0000000100000000";

if (onehz = ACTIVE) then

nextstate <= MoveR8;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR7;

end if;

when MoveR8 =>

led <= "0000000010000000";

if (onehz = ACTIVE) then

nextstate <= MoveR9;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR8;

end if;

when MoveR9 =>

led <= "0000000001000000";

if (onehz = ACTIVE) then

nextstate <= MoveR10;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR9;

end if;

when MoveR10 =>

led <= "0000000000100000";

if (onehz = ACTIVE) then

nextstate <= MoveR11;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR10;

end if;

when MoveR11 =>

led <= "0000000000010000";

if (onehz = ACTIVE) then

nextstate <= MoveR12;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR11;

end if;

when MoveR12 =>

led <= "0000000000001000";

if (onehz = ACTIVE) then

nextstate <= MoveR13;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR12;

end if;

when MoveR13 =>

led <= "0000000000000100";

if (onehz = ACTIVE) then

nextstate <= MoveR14;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR13;

end if;

when MoveR14 =>

led <= "0000000000000010";

if (p2btn = ACTIVE) then

p2btnflag <= '1';

elsif (p2btn = not ACTIVE) then

p2btnflag <= '0';

end if;

if (onehz = ACTIVE) then

nextstate <= p2event;

elsif (onehz = not ACTIVE) then

nextstate <= MoveR14;

end if;

when p2event =>

led <= "0000000000000001";

if (onehz=ACTIVE) then

if (p2btnflag = not ACTIVE) then

if (p2btn=ACTIVE) then

nextstate <= MoveL1;

else

nextstate <= p2miss;

end if;

else

nextstate <= p2miss;

end if;

else

nextstate<=p2event;

end if;

when p2miss =>

led <= "0000000000000000";

if (onehz = ACTIVE) then

nextstate <= p2serve;

p1point <= ACTIVE;

if (p1count=4) then

nextstate <= p1win;

end if;

else

nextstate <=p2miss;

end if;

when p2serve =>

led <= "0000000000000001";

p1point <= NOTACTIVE;

p2point <= NOTACTIVE;

if (p2btn=ACTIVE) then

nextstate <= MoveL1;

else

nextstate <= p2serve;

end if;

when MoveL1 =>

led <= "0000000000000010";

if (onehz = ACTIVE) then

nextstate <= MoveL2;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL1;

end if;

when MoveL2 =>

led <= "0000000000000100";

if (onehz = ACTIVE) then

nextstate <= MoveL3;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL2;

end if;

when MoveL3 =>

led <= "0000000000001000";

if (onehz = ACTIVE) then

nextstate <= MoveL4;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL3;

end if;

when MoveL4 =>

led <= "0000000000010000";

if (onehz = ACTIVE) then

nextstate <= MoveL5;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL4;

end if;

when MoveL5 =>

led <= "0000000000100000";

if (onehz = ACTIVE) then

nextstate <= MoveL6;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL5;

end if;

when MoveL6 =>

led <= "0000000001000000";

if (onehz = ACTIVE) then

nextstate <= MoveL7;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL6;

end if;

when MoveL7 =>

led <= "0000000010000000";

if (onehz = ACTIVE) then

nextstate <= MoveL8;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL7;

end if;

when MoveL8 =>

led <= "0000000100000000";

if (onehz = ACTIVE) then

nextstate <= MoveL9;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL8;

end if;

when MoveL9 =>

led <= "0000001000000000";

if (onehz = ACTIVE) then

nextstate <= MoveL10;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL9;

end if;

when MoveL10 =>

led <= "0000010000000000";

if (onehz = ACTIVE) then

nextstate <= MoveL11;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL10;

end if;

when MoveL11 =>

led <= "0000100000000000";

if (onehz = ACTIVE) then

nextstate <= MoveL12;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL11;

end if;

when MoveL12 =>

led <= "0001000000000000";

if (onehz = ACTIVE) then

nextstate <= MoveL13;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL12;

end if;

when MoveL13 =>

led <= "0010000000000000";

if (onehz = ACTIVE) then

nextstate <= MoveL14;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL13;

end if;

when MoveL14 =>

led <= "0100000000000000";

if (p1btn = ACTIVE) then

p1btnflag <= '1';

elsif (p1btn = not ACTIVE) then

p1btnflag <= '0';

end if;

if (onehz = ACTIVE) then

nextstate <= p1event;

elsif (onehz = not ACTIVE) then

nextstate <= MoveL14;

end if;

when p1win =>

winnerp1 <= ACTIVE;

nextstate <= p1win;

when p2win =>

winnerp2 <= ACTIVE;

nextstate <= p2win;

end case;

end process;

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

--Counts the number of points player one has when player two misses, counts when p1point is active

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

p1count\_score: process(Reset,clk)

begin

if (Reset=ACTIVE) then

p1count <= 0;

elsif (rising\_edge(clk)) then

if (p1point=ACTIVE) then

p1count <= p1count+1;

else

p1count <= p1count;

end if;

end if;

end process;

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

--Counts the number of points player two has when player one misses, counts when p2point is active

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

p2count\_score: process(Reset,clk)

begin

if (Reset=ACTIVE) then

p2count <= 0;

elsif (rising\_edge(clk)) then

if (p2point=ACTIVE) then

p2count <= p2count + 1;

else

p2count <= p2count;

end if;

end if;

end process;

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

-- OneKHz Clock Divider

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

onekhz\_process: process(clk, reset)

variable counter: integer range 0 to TERMINAL\_VALUE;

begin

if (SysReset=ACTIVE) then

counter := 0;

elsif (rising\_edge(clk)) then

counter := counter + 1;

if (counter = TERMINAL\_VALUE) then

counter := 0;

onekhz <= not onekhz;

end if;

end if;

end process;

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

--Used for selecting what seven segs will display and which anodes to use

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

digit\_select: process (digitsel, onekhz)

variable count: integer range 0 to 1;

begin

if(onekhz = NOTACTIVE) then

count :=0;

elsif(onekhz = ACTIVE) then

count := 1;

end if;

digitsel <=count;

end process;

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

--Tells seven segs what to display for player one's score is depending on what p1count is

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

with p1count select

p1score <= zero when 0,

one when 1,

two when 2,

three when 3,

four when 4,

p1victory when others;

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

--Tells seven segs what to display for player two's score is depending on what p2count is

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

with p2count select

p2score <= zero when 0,

one when 1,

two when 2,

three when 3,

four when 4,

p2victory when others;

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

--Chooses what the seven seg will display depending on the count of digitsel,

--shows the scores of player one and player two.

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

with digitsel select

seg <= p1score when 0,

p2score when 1,

nothing when others;

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

--Chooses which seven seg to display depending on the count of digitsel,

--player one's score is the left most anode and player two's score is the right most anode.

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

with digitsel select

an <= "0111" when 0,

"1110" when 1,

"1011" when 2,

"1101" when others;

end lab4behav;

Flow Diagram

