Nick Ray

Julian Duran

October 11, 2018

VHDL Design with FPGAs

Lab 3 – Stop Watch

CPE 3020 (01)

Lab 3 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 3 - StopWatch

-- The purpose of is this lab is to create a stop watch that will count up to 20 and

-- set off an LED sequence to notify the user that we have reached the end of the count.

-- The user will also have the ability to stop/start the count by pressing the center button

-- and reset the count by pressing the down button. The lab also allows the user to change

-- the mode of the counter from the default counter, which count to 20, to an input mode

-- which allows the user to input a value using the LED’s.

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

entity lab2 is

port(

sw: in std\_logic\_vector(7 downto 0); --switches to enter max value to

an: out std\_logic\_vector(3 downto 0); --Anode displays of 7 segments

seg: out std\_logic\_vector(6 downto 0); --The 7 segment LED's to display numbers

led: out std\_logic\_vector(0 to 15); --LED's for the sequence to display when count has finished

clk: in std\_logic; -- 100MHz

btnC: in std\_logic; -- Start/Stop the counter

btnL: in std\_logic; -- Reset the counter

btnD: in std\_logic; -- Set the count mode to input a value or default to count to 20.

dp: out std\_logic -- To display that in input mode

);

end lab2;

architecture lab2\_arch of lab2 is

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

-- Constants for the 7-seg Display

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

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 six: std\_logic\_vector(6 downto 0) := "0000010";

constant seven: std\_logic\_vector(6 downto 0) := "1111000";

constant eight: std\_logic\_vector(6 downto 0) := "0000000";

constant nine: std\_logic\_vector(6 downto 0) := "0011000";

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

-- Defining signals and constants for use in processes

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

constant NOTACTIVE: std\_logic := '0'; -- Off with LOW

constant ACTIVE: std\_logic := '1'; --On with HIGH

constant TERMINAL\_VALUE: integer := 999999; --Making OneKHz Clock

constant SMALL\_TERMINAL\_VALUE: integer := 38; --Making OneHz Clock

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

signal enable: std\_logic := '0'; --Enable counter

signal digitsel: integer range 0 to 3; --Select Anodes

signal onekhz: std\_logic; --OneKHz for 7-seg toggle

signal onehz: std\_logic; --OneHz for seconds increment

signal seg\_low: std\_logic\_vector (6 downto 0); --Low digit for 7-seg

signal seg\_high: std\_logic\_vector (6 downto 0); --High digit for 7-seg

signal count\_low: integer range 0 to 9; --Count for 7-seg low

signal count\_high: integer range 0 to 9; --Count for 7-seg high

signal desired\_low: integer; --Value of counting to in seconds digit

signal desired\_high: integer; --Value of counting to in tens digit

signal sequence: std\_logic\_vector(0 to 15); --Which LED's to turn on for sequence

signal finished: std\_logic; --For the LED's to know when to turn on sequence

signal pause: std\_logic :='0'; --To stop/start the count

signal Cvector: std\_logic\_vector (9 downto 0); --For Center Button Debouncing

signal Coutput: std\_logic; --Output of debouncing

signal Dvector: std\_logic\_vector (9 downto 0); --For Down Button Debouncing

signal Doutput: std\_logic; --Output of debouncing

signal Lvector: std\_logic\_vector (9 downto 0); --For Left Button Debouncing

signal Loutput: std\_logic; --Output of debouncing

signal toggleInput: std\_logic := '0'; --Selecting Mode between inputed value or default value

signal lowBCD: integer range 0 to 9; --Value of inputed seconds number

signal highBCD : integer range 0 to 9; --Value of inputed Tens number

begin

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

-- Shift Debounce for Center Button (Stop/Start)

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

C\_button\_debounce\_shift: process (onekhz, btnC, Cvector, Coutput)

begin

if (rising\_edge(onekhz)) then

Cvector<=Cvector(8 downto 0)&btnC;

end if;

if (Cvector="1111111111") then

Coutput <= '1';

else

Coutput <= '0';

end if;

end process;

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

-- Shift Debounce for Down Button (Reset)

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

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

begin

if (rising\_edge(onekhz)) then

Dvector<=Dvector(8 downto 0)&btnD;

end if;

if (Dvector="1111111111") then

Doutput <= '1';

else

Doutput <= '0';

end if;

end process;

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

-- Shift Debounce for Left Button (Mode)

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

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

begin

if (rising\_edge(onekhz)) then

Lvector<=Lvector(8 downto 0)&btnL;

end if;

if (Lvector="1111111111") then

Loutput <= '1';

else

Loutput <= '0';

end if;

end process;

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

-- Toggle for Stop/Start

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

pause\_process: process(Coutput)

begin

if (rising\_edge(Coutput)) then

pause <= not pause;

end if;

end process;

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

-- Toggle for Mode

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

toggle\_process: process(Loutput)

begin

if (rising\_edge(Loutput)) then

toggleInput <= not toggleInput;

end if;

end process;

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

-- OneKHz Clock Divider

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

onekhz\_process: process(clk, reset)

variable counter: integer range 0 to TERMINAL\_VALUE;

begin

if (reset=ACTIVE) then

counter := 0;

elsif (rising\_edge(clk)) then

counter := counter + 1;

if (counter = TERMINAL\_VALUE) then

counter := 0;

onekhz <= not onekhz;

enable <= not enable;

end if;

end if;

end process;

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

--OneHz clock Divider

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

onehz\_process: process(clk, reset, enable)

variable counter2: integer range 0 to SMALL\_TERMINAL\_VALUE;

begin

if (reset=ACTIVE) then

counter2 := 0;

elsif (rising\_edge(clk)) then

if (enable = ACTIVE) then

counter2 := counter2 + 1;

elsif (counter2 = SMALL\_TERMINAL\_VALUE) then

counter2 := 0;

onehz <= not onehz;

end if;

end if;

end process;

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

-- OneHz Counter for Stop Watch with Restart Button, Start/Stop Button, and the desired value for the counter

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

onehz\_count: process(Doutput, pause, reset, onehz)

begin

if (reset=ACTIVE) then

count\_high <= 0;

count\_low <= 0;

elsif (Doutput = '1') then

count\_low <= 0;

count\_high <= 0;

finished <= '0';

elsif (pause = '0') then

count\_low <= count\_low;

count\_high <= count\_high;

elsif (rising\_edge(onehz)) then

if(count\_high < desired\_high) then

if (count\_low < 9) then

count\_low <= count\_low+1;

elsif (count\_low = 9) then

count\_high <= count\_high+1;

count\_low <= 0;

end if;

end if;

if (count\_high = desired\_high) then

if (desired\_low = 0) then

finished <= '1';

elsif (count\_low < desired\_low) then

count\_low <= count\_low+1;

finished <= '0';

elsif (count\_low = desired\_low) then

finished <= '1';

end if;

elsif (count\_high < desired\_high or count\_low < desired\_low) then

finished <= '0';

end if;

end if;

end process;

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

-- LED Sequence set off when Count hits Desired Value

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

finishedLEDs:process (onehz, finished)

begin

if (onehz = ACTIVE) then

if (finished = ACTIVE) then

sequence<="1100110011001100";

end if;

end if;

if (onehz = NOTACTIVE) then

if (finished = ACTIVE) then

sequence<="0011001100110011";

end if;

end if;

end process;

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

-- Toggle of decimal point for the mode in. Decimal is on if in input mode

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

decimal\_point\_select: process(digitsel, toggleInput)

begin

if ((digitsel=1)and(toggleInput='1')) then

dp<=ACTIVE;

else

dp<=NOTACTIVE;

end if;

end process;

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

-- Digit Select process follows the one Hz clock

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

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;

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

--Reads what is the inputed from the switches for the desired value of the first 7-seg display

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

with sw(3 downto 0) select

lowBCD <= 0 when "0000",

1 when "0001",

2 when "0010",

3 when "0011",

4 when "0100",

5 when "0101",

6 when "0110",

7 when "0111",

8 when "1000",

9 when "1001",

9 when others;

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

--Reads what is the inputed from the switches for the desired value of the second 7-seg display

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

with sw(7 downto 4) select

highBCD <= 0 when "0000",

1 when "0001",

2 when "0010",

3 when "0011",

4 when "0100",

5 when "0101",

6 when "0110",

7 when "0111",

8 when "1000",

9 when "1001",

9 when others;

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

--Reads what the count\_low is and sends info to count\_low for first 7-seg display

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

with count\_low select

seg\_low <= zero when 0,

one when 1,

two when 2,

three when 3,

four when 4,

five when 5,

six when 6,

seven when 7,

eight when 8,

nine when 9,

nothing when others;

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

-- Reads what the count\_high is and sends info to count\_high for second 7-seg display

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

with count\_high select

seg\_high <= one when 1,

two when 2,

three when 3,

four when 4,

five when 5,

six when 6,

seven when 7,

eight when 8,

nine when 9,

zero when others;

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

-- Telling system to set off LED sequence when the finished signal goes high

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

with finished select

LED(0 to 15) <= sequence when '1',

"0000000000000000" when others;

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

-- Selecting the 7 Segments to display

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

with digitsel select

seg <= seg\_low when 0,

seg\_high when 1,

seg\_high when others;

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

-- Selecting which anodes to use

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

with digitsel select

an <= "1110" when 0,

"1101" when 1,

"1111" when others;

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

-- Toggle of desired\_low between default value and inputed value depening on Left button

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

with toggleInput select

desired\_low <= 0 when '0',

lowBCD when '1';

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

-- Toggle of desired\_high between default value and inputed value depending on Left Button

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

with toggleInput select

desired\_high <= 2 when '0',

highBCD when '1';

end lab2\_arch;