# EEL 3701 – Digital Logic and Computer Systems Lab 3

# Greg Bolling

# UF ID 64911960

## Problem Statement

The goal of the lab is to build sequential circuits from memory elements. In the first problem, an alarm system is synthesized from an SR-Latch. The design of the alarm system is simple to have a set and reset function. The next problem involved implementing a shift register and changing that shift register to shift left then right. Finally, the last assignment involved building a state machine that counted odd numbers from

## Design

Problem 1 Part 1 starts with creating an alarm system with a panel button that resets the alarm (off state) and a door switch that can turn the alarm on (on-state.) The SR Latch was chosen because it has Set and Reset inputs that align to the function of turning an alarm on and turning an alarm off. The latch was chosen over combinatorial logic because a latch holds the memory of the last event.

SR-Latch

Set is connected to the door switch. When that goes high, the alarm sounds. Connect the switch to a dual position switch (alternatively could use a pulldown resistor and a pushbutton switch to Vcc.)

Reset is connected at the panel to turn the alarm off. It is just connected to Vcc when the button is pushed.

The S-R Latch is created in VHDL using two key combinatorial logic gates:

Q <= R nor notQ;

notQ <= S nor Q;

I will use these pins:

S Pin J7

R Pin F4

Q Pin N10 (Alarm / LED)

The design could have used a JK Latch to handle the case when the door is opened and the alarm is being reset when they both close but this is a simple design so maybe it is not necessary. The design had warnings but no errors in Quartus.

Problem 2

I implemented the code provided in the pre-lab. This is a shift register that clocks all 1’s (LED on) into the register and shifts in a 0 over and over until the entire display is all 0 (LED’s off) and then starts over by setting it to all 1’s (LED on).

I connected the pins as follows:

clock Pin H4

LED[7] Pin N10

LED[6] Pin N12

LED[5] Pin M13

LED[4] Pin L13

LED[3] Pin J13

LED[2] Pin H13

LED[1] Pin G13

LED[0] Pin K7

Reset Pin G4

I reversed this shift direction by inserting a ‘0’ at bit 0 and then shifting everything to bits 7 down to 1. The code for this was modified as :

shift\_reg(7) <= '0';

shift\_reg(6 downto 0) <= shift\_reg(7 downto 1);

I then modified the design to do a left shift followed by a right shift indefinitely. To do this, I set a register to 0 to show that I haven’t started to right shift. I then set the register to a 1 and that tells me to do right shifts as opposed to left shifts. The system works the same otherwise. I used the same pin assignments from the shift registers above.

Problem 3

I modified the provided VHDL code to add the state machine transitions in VHDL as follows:

P1: process(present\_state, a\_g)

begin

case present\_state is

when S0 =>

if(a\_g = "0000110") then

next\_state <= S1;

else

next\_state <= S0;

end if;

a\_g<= "0000110";

display <= "0001";

when S1 =>

if(a\_g = "0100100") then

next\_state <= S2;

else

next\_state <= S1;

end if;

a\_g<= "0100100";

display <= "0010";

when S2 =>

if(a\_g = "0000000") then

next\_state <= S3;

else

next\_state <= S2;

end if;

a\_g<= "0000000";

display <= "0100";

when S3 =>

if(a\_g = "0001100") then

next\_state <= S0;

else

next\_state <= S3;

end if;

a\_g<= "0001100";

display <= "1000";

end case;

end process;

Pin Assignments for the seven-segment display (I renamed LED to SevenSeg because I used LED above and it became a problem keeping this aligned.)

clock Pin H4

reset Pin G4

SevenSeg[6] Pin D11

SevenSeg[5] Pin M12

SevenSeg[4] Pin M11

SevenSeg[3] Pin M10

SevenSeg[2] Pin M9

SevenSeg[1] Pin K12

SevenSeg[0] Pin K8

Display[3] Pin H3

Display[2] Pin H2

Display[1] Pin K5

Display[0] Pin J2

Problem 4

The code for counting 0,1,3,5,7,9,0 is below in this file. It will use the same pin assignments as in Problem 3 last part above.

clock Pin H4

reset Pin G4

SevenSeg[6] Pin D11

SevenSeg[5] Pin M12

SevenSeg[4] Pin M11

SevenSeg[3] Pin M10

SevenSeg[2] Pin M9

SevenSeg[1] Pin K12

SevenSeg[0] Pin K8

Display[3] Pin H3

Display[2] Pin H2

Display[1] Pin K5

Display[0] Pin J2

## Implementation

All of the designs were implemented using VHDL and similar to previous labs. Most of the designs were clocked with a clock except the first which was just to experiment and build an SR-Latch. All designs were compiled and downloaded to the Out of The Box board. This last problem in this lab brought together parts of the previous labs to build a state machine that transitioned from 0,1,3,5,7,9,0 and around again while displaying the results on a seven segment display and using registers to latch the value form a slow clock ( a clock that divided down a fast clock to be slower.)

The design all added multiple VHDL files to the project with one file being the top file and the rest were below it and it utilized VHDL components to build up a design from other parts.

## Conclusions

VHDL’s ability to add in components from other components is interesting. It makes it possible to build large designs from smaller designs that are compartmentalized. This is analogous to the C++ constructs of objects and classes. With this method, complex parts can be separated from other parts of the design to make the design easier to update and maintain in the future. It also allows for building circuits in different clock domains using the mechanism of port maps to different signals at the higher level of the design.

The OOTB design is interesting, its easy to see from this that a more complex design can be created the adds in multiple features.

Appendix

VHDL Files

Problem 4

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

use ieee.std\_logic\_unsigned.all;

--------------fastclock for sevensegment display-------------

entity lab3part4 is

Port (clock, reset : in STD\_LOGIC;

display: out std\_logic\_vector(3 DOWNTO 0);

Seven\_Seg : out std\_logic\_vector(6 DOWNTO 0));

end lab3part4;

architecture Behavioral of lab3part4 is

signal a\_g : std\_logic\_vector(6 downto 0); -- declare the (state-machine) enumerated type

signal count : std\_logic\_vector(3 downto 0); -- counter

signal slow\_clk : std\_logic;

component slow\_clock is

port(

clock, reset : in STD\_LOGIC;

slow\_clk : out STD\_LOGIC); end component;

begin

U0: slow\_clock

port map(

slow\_clk=>slow\_clk,

reset=>reset,

clock=>clock);

U1: SevenSeg

port map(

binary => count,

a\_to\_g => a\_g);

P0: process(slow\_clk, reset)

begin

if(reset = '1') then

count <= "0000";

elsif (slow\_clk ='1' and slow\_clk'event) then

count(3) <= count(2) and count(1);

count(2) <= ((not count(3) and count(0)) and not count(2)) or

((not count(3) and count(0)) and not count(1));

count(1) <= not count(3) and not count(1);

count(0) <= not count(3);

end if;

end process;

display <= "0001";

Seven\_Seg <= a\_g;

end Behavioral;

Picture(s)

