EEL 3701C - Digital Logic & Computer Systems

Lab 4 Report

Charles Richardson
73112398

Prelab Report

Prelab Design and Implementation

- Designing the prelab involved careful reading of the lab 4 document. Problems 1 and 2 did not take much consideration, but problem 3 required a first principles thinking strategy and took up the bulk of the project. To design this problem, I drew out the state machine and created a Moore Transition Graph to get a grasp of what was necessary in the code.
- Implementing the majority of problem 1 and 2 was simple, some copy and paste of previous labs with minor additions and optimization. On the other hand, implementing problem 3 prelab took countless hours. I required help from PIs for the first time in a lab, and the best piece of advice I got was to section the logic into 3 processes.

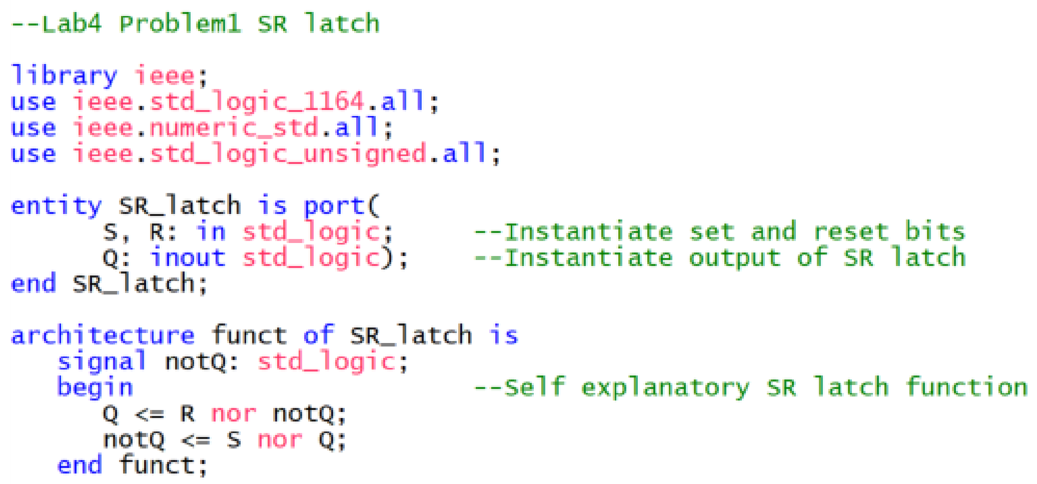
Reflection

- I learned that I need to attend office hours sooner when I run into problems. There is no point in sitting clueless when there are people available to help. These labs are individual assignments, but there is always an opportunity to get help from others. As humans, we depend on each other to survive, being a student makes no difference. I was happy about being able to figure out the third problem by myself after getting under an hour of help from PIs.

Prelab Homework

<u>Problem 1:</u>

1. An SR latch is sufficient to creating a functioning alarm. First, we consider S the door (0 closed, 1 open), R the reset (0 standby, 1 set Q = 0) and Q the alarm (0 standby, 1 active). Under this logic, once S = 1, the signal of Q is fixed at 1 unless R = 1. This means that the alarm cannot be shut off - once the door has been opened - unless the R signal is triggered.



```
--Lab4 Problem1 SR latch

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity SR_latch is port(
      S, R: in std_logic;      --Instantiate set and reset bits
      Q: inout std_logic);     --Instantiate output of SR latch
end SR_latch;

architecture funct of SR_latch is
   signal notQ: std_logic;
   begin                       --Self explanatory SR latch function
      Q <= R nor notQ;
      notQ <= S nor Q;
   end funct;
```

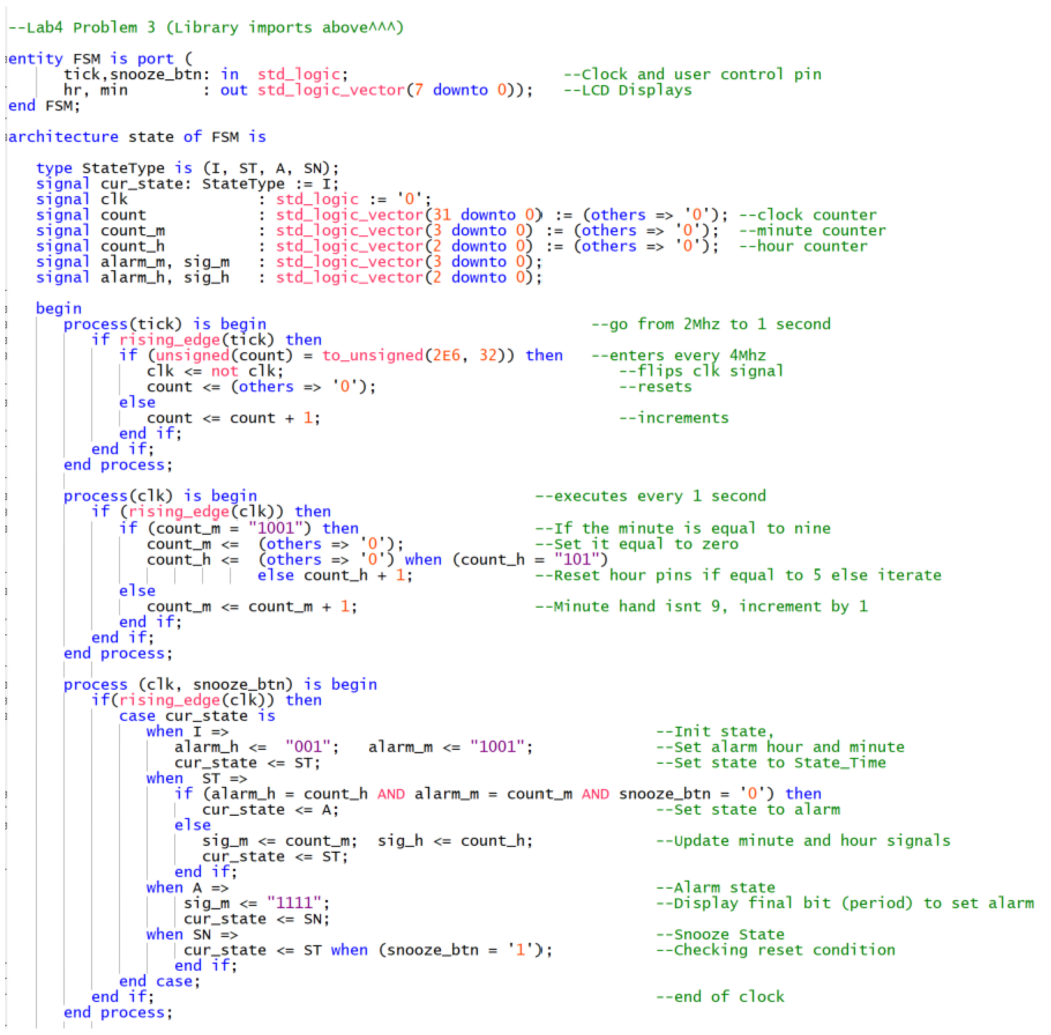
2. This is the code for the SR latch, comments are made to better understand what is happening.

Problem 2:

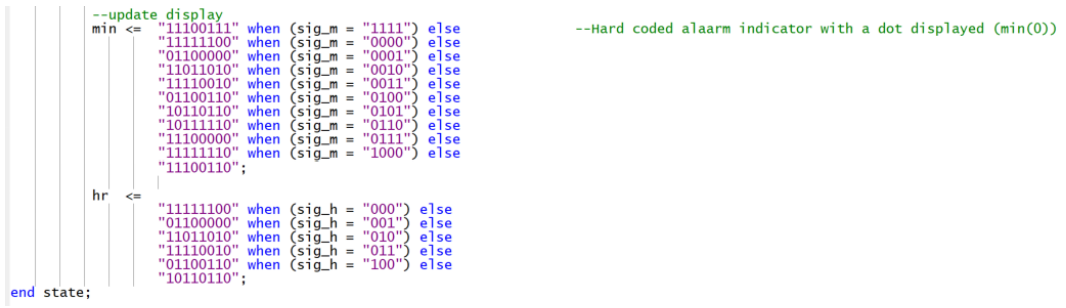
```
--Lab4 Prob 2

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee. math_real.round;
use IEEE.std_logic_unsigned.all;


entity L4_BCD is port(
    tick         : in std_logic;                          --Clock Pin
    minute, hour : out std_logic_vector(7 downto 0)       --LCD Displays
    );
end L4_BCD;


architecture mux of L4_BCD is

   signal count : std_logic_vector(31 downto 0) := (others => '0'); --tick counter
   signal min   : std_logic_vector(4 downto 0) := (others => '0');  --minute counter
   signal hr    : std_logic_vector(2 downto 0) := (others => '0');  --hour counter

 begin
    process(tick) is
    begin
      if rising_edge(tick) then
         if (unsigned(count) = to_unsigned(2E6, 32)) then
            if (min = "1001") then          --If the minute BCD is equal to nine
               min <= (others => '0');      -- Set it equal to zero
               if (hr = "101") then         -- Then check if the hour hand is equal to 5
                  hr <= (others => '0');    --If it is set that to zero as well
               else
                  hr <= hr + 1;             --If not, increment by 1
               end if;
            else
               min <= min + 1;              --Minute hand isnt 9, increment by 1
            end if;
            count <= (others => '0');       --Reset tick
         else
            count <= count + 1;             --Increment tick
         end if;
      end if;

    end process;

    minute <=   "11111100" when (min = "0000") else    --Assign minute pins based on min
                "01100000" when (min = "0001") else
                "11011010" when (min = "0010") else
                "11110010" when (min = "0011") else
                "01100110" when (min = "0100") else
                "10110110" when (min = "0101") else
                "10111110" when (min = "0110") else
                "11100000" when (min = "0111") else
                "11111110" when (min = "1000") else
                "11100110";

    hour    <=  "11111100" when (count_3 = "000") else --Assign hour pins based on hr
                "01100000" when (count_3 = "001") else
                "11011010" when (count_3 = "010") else
                "11110010" when (count_3 = "011") else
                "01100110" when (count_3 = "100") else
                "10110110";

end mux;
```

1. This is the full file of code for the Muxian clock. In short, every time the tick input increments to 4e6 (a 4MHz clock), one second has gone by. Each time this occurs, the counter values of the hour and minute displays increment or reset. The hour signal resets to 0 when it passes 5, and the minute signal resets to 0 when it passes 9. Outside of this logic, the counters are mapped into bit values for the BCD on the FPGA board.

Problem 3

1. 

```
--Lab4 Problem 3 (Library imports above^^^)

entity FSM is port (
    tick,snooze_btn: in  std_logic;                    --Clock and user control pin
    hr, min        : out std_logic_vector(7 downto 0));   --LCD Displays
end FSM;

architecture state of FSM is

  type StateType is (I, ST, A, SN);
  signal cur_state: StateType := I;
  signal clk            : std_logic := '0';
  signal count          : std_logic_vector(31 downto 0) := (others => '0'); --clock counter
  signal count_m        : std_logic_vector(3 downto 0)  := (others => '0');  --minute counter
  signal count_h        : std_logic_vector(2 downto 0)  := (others => '0');  --hour counter
  signal alarm_m, sig_m : std_logic_vector(3 downto 0);
  signal alarm_h, sig_h : std_logic_vector(2 downto 0);

  begin
    process(tick) is begin                                  --go from 2Mhz to 1 second
      if rising_edge(tick) then
        if (unsigned(count) = to_unsigned(2E6, 32)) then   --enters every 4Mhz
          clk <= not clk;                                     --flips clk signal
          count <= (others => '0');                           --resets
        else
          count <= count + 1;                                 --increments
        end if;
      end if;
    end process;

    process(clk) is begin                            --executes every 1 second
      if (rising_edge(clk)) then
        if (count_m = "1001") then                   --If the minute is equal to nine
          count_m <=  (others => '0');               --Set it equal to zero
          count_h <=  (others => '0') when (count_h = "101")
                      else count_h + 1;              --Reset hour pins if equal to 5 else iterate
        else
          count_m <= count_m + 1;                    --Minute hand isnt 9, increment by 1
        end if;
      end if;
    end process;

    process (clk, snooze_btn) is begin
      if(rising_edge(clk)) then
        case cur_state is
          when I =>                                                --Init state,
            alarm_h <=  "001";   alarm_m <= "1001";                --Set alarm hour and minute
            cur_state <= ST;                                       --Set state to State_Time
          when  ST =>
            if (alarm_h = count_h AND alarm_m = count_m AND snooze_btn = '0') then
              cur_state <= A;                                      --Set state to alarm
            else
              sig_m <= count_m;  sig_h <= count_h;                 --Update minute and hour signals
              cur_state <= ST;
            end if;
          when A =>                                                --Alarm state
           sig_m <= "1111";                                        --Display final bit (period) to set alarm
           cur_state <= SN;
          when SN =>                                               --Snooze State
           cur_state <= ST when (snooze_btn = '1');                --Checking reset condition
          end if;
        end case;
      end if;                                                      --end of clock
    end process;
```



```
      --update display
      min <=   "11100111" when (sig_m = "1111") else           --Hard coded alaarm indicator with a dot displayed (min(0))
               "11111100" when (sig_m = "0000") else
               "01100000" when (sig_m = "0001") else
               "11011010" when (sig_m = "0010") else
               "11110010" when (sig_m = "0011") else
               "01100110" when (sig_m = "0100") else
               "10110110" when (sig_m = "0101") else
               "10111110" when (sig_m = "0110") else
               "11100000" when (sig_m = "0111") else
               "11111110" when (sig_m = "1000") else
               "11100110";

      hr  <=
               "11111100" when (sig_h = "000") else
               "01100000" when (sig_h = "001") else
               "11011010" when (sig_h = "010") else
               "11110010" when (sig_h = "011") else
               "01100110" when (sig_h = "100") else
               "10110110";
end state;
```

This is the full file for the Muxian alarm. The clock feature works identical to the Muxian clock, however, it has been segmented into 2 processes for better readability and functionality. The third process takes care of switching between the states of the machine and handling the functionality of the inputs. At the end of the file, the displays are mapped identical to the clock in problem 2, with the exception of an additional when else loop in the min output to indicate the alarm has been triggered.

Postlab Report

Problem Statement:

This lab was meant to give students an idea of how to work with finite state machines (FSM) and get more comfortable with coding circuits that are practical in real life, such as a memory block and an alarm clock. Using very simple inputs, the lab was able to accomplish plenty of "real life" applications of digital logic, and shine a light on the attainable potential of the software that is used for the class.

Design:

The most important design decisions that was made was to first get the program working, then simplify it. I made a handful of extra layers to make sure that the code was operating as expected before I abstracted the system to work without guard rails. Doing this took a large chunk of time, but it gave me more confidence in the work I was doing. In the end, adding extra helper layers got me to understand it better, and ultimately explain the functionality in the demonstration video better. (Code and comments can be found in prelab section)

Implementation:

Implementing problems 1 and 2 were relatively straightforward, as mentioned in the prelab report. Problem 3 is what took most of my time, approximately 5 hours to be specific. The complexity of the problem overwhelmed me at first, but after attending office hours and clarifying my understanding of the system, the project worked much better. In the end, about 2 hours of my time were wasted by forgetting a simple rising_edge() function. I will certainly not be making that mistake again.

Testing:

I tested this lab by first using a VWF (Waveform) simulation. Once I tuned the system on the waveform (following the prelab sections) I created a simple DIP switch on the breadboard and programmed a SOF file to my FPGA to see it work off my computer screen and get a demonstration video. The only impediment I ran into during testing was with the SR latch, this was required to be demonstrated through the VWF simulation (as per PIs orders)

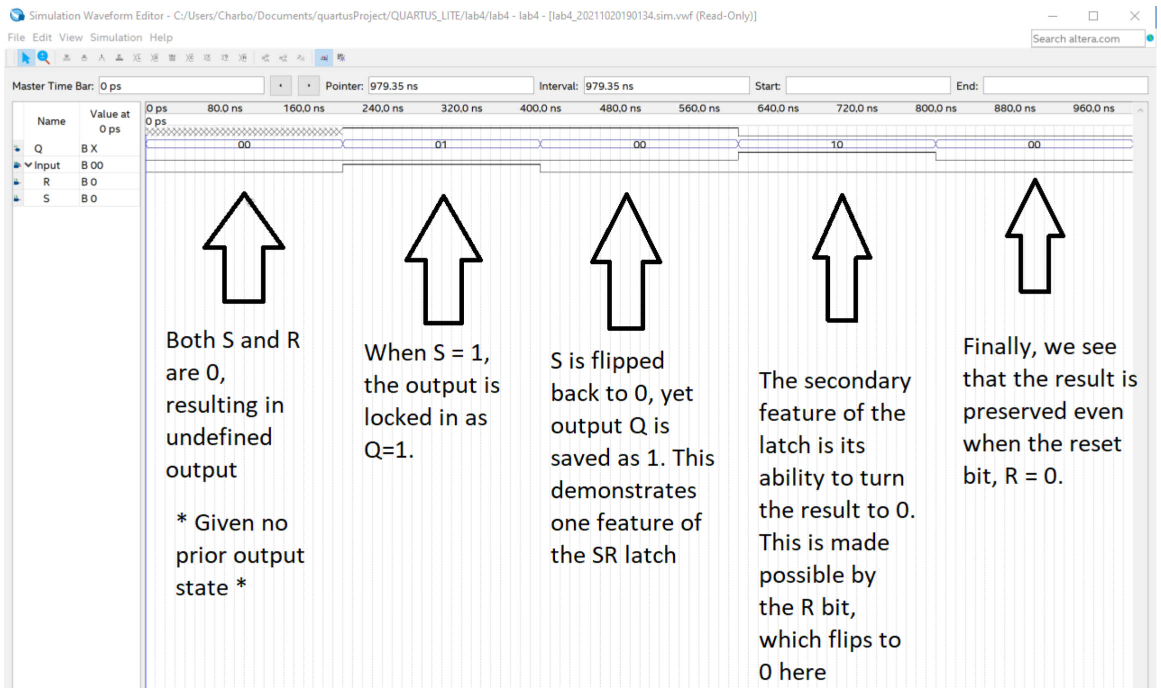
Conclusions:

I learned a lot this lab. First, I learned that the prelabs are meant to be SUBMITTED before the lab, not just finished… sorry Daniel. Second, I learned that getting help from office hours helps more than I thought it would. Third, I learned that the people who make this class sound so hard are just trying to make themselves feel better by getting affirmation in groupchats, so Ill be steering clear of those for the most part. Finally, I learned how to make an alarm clock, super cool!

Appendix

Problem 1 Implementation

1.


Simulation Waveform Editor - C:/Users/Charbo/Documents/quartusProject/QUARTUS_LITE/lab4/lab4 - lab4 - [lab4_20211020190134.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
880.0 ns
960.0 ns
00
01
00
10
00

Master Time Bar: 0 ps
Pointer: 979.35 ns
Interval: 979.35 ns
Start:
End:

Name	Value at 0 ps
Q	B X
Input	B 00
R	B 0
S	B 0

Both S and R are 0, resulting in undefined output

* Given no prior output state *

When S = 1, the output is locked in as Q=1.

S is flipped back to 0, yet output Q is saved as 1. This demonstrates one feature of the SR latch

The secondary feature of the latch is its ability to turn the result to 0. This is made possible by the R bit, which flips to 0 here

Finally, we see that the result is preserved even when the reset bit, R = 0.

Problem 2 Implementation

1. Attached as video

Problem 3 Implementation

1. Attached as video