1 Construct a 30-second down counter (timer) with pause function. When the counter goes to 0, all the LEDs will be lighted up. You can use one push button for reset and one other for pause/start function.

1.1 Implement a periodic 30-second down counter and demo with the FPGA board.

1.2 Implement Prelab 1.3 and demo with the FPGA board.

1.3 Combine 1.2 and 1.3 to finish the experiment.

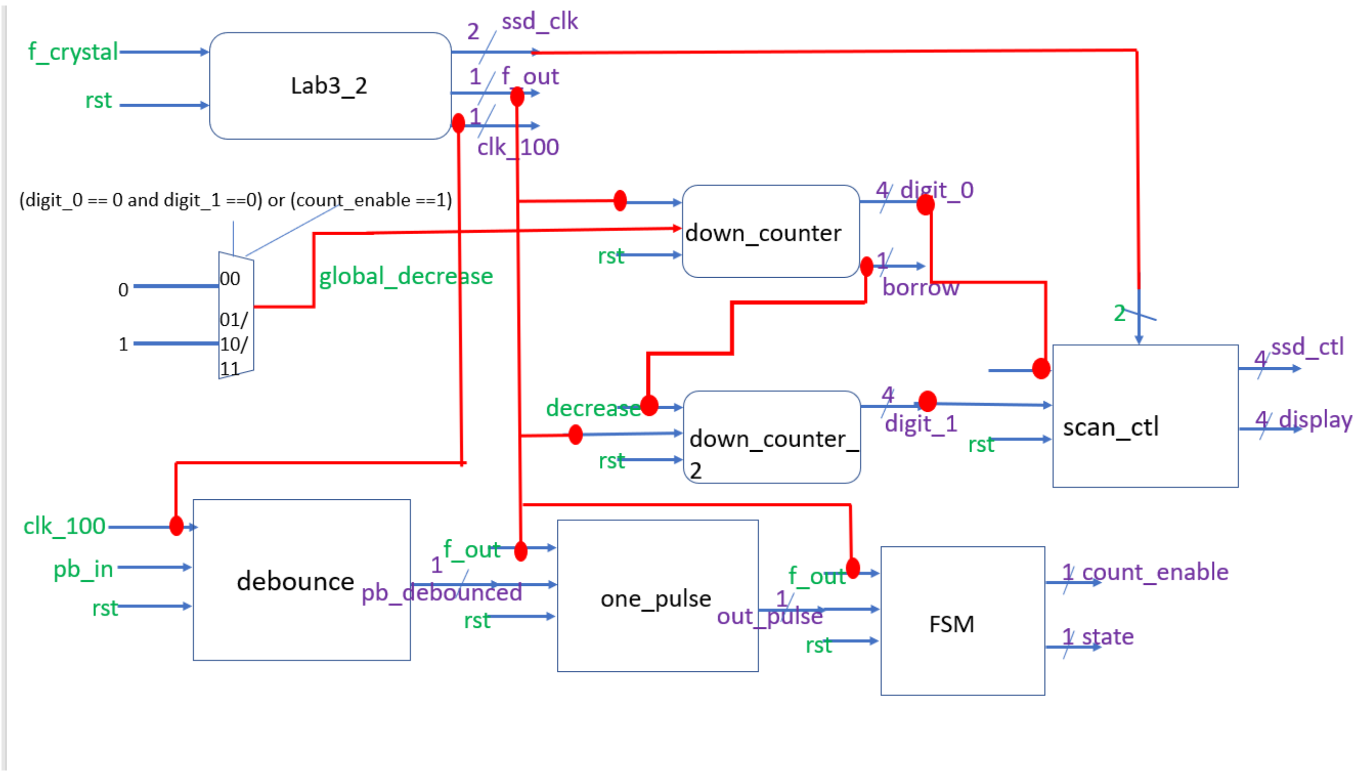
Design Specification:

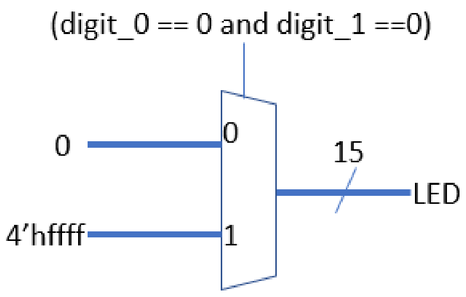
Input: f_crystal, pb_in, rst

Output: [3:0]ssd_ctl, [7:0]D_ssd, [14:0]LED

Output: state

Block diagram:





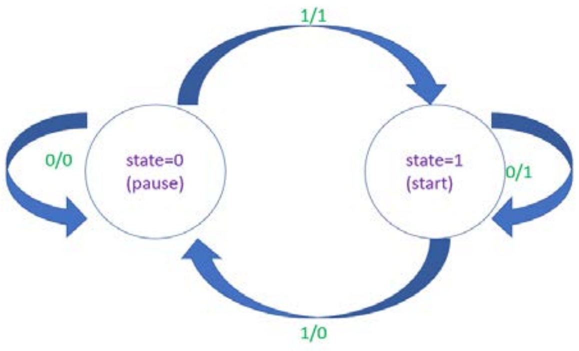
Design Implementation:

I/O pin assignment:

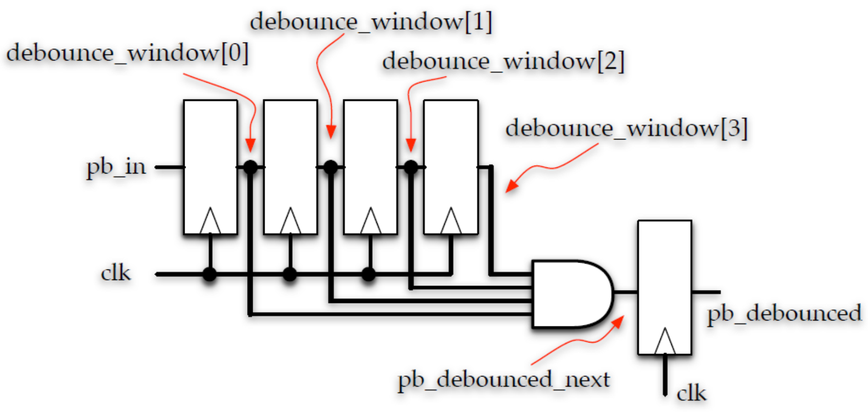
f_crystal—W5
rst—W19
pb_in—U17
ssd_ctl[3]—W4
ssd_ctl[2]—V4
ssd_ctl[1]—U4
ssd_ctl[0]—U2
D_ssd[7]—W7
D_ssd[6] — W6
D_ssd[5] — U8
D_ssd[4] — V8
D_ssd[3] — U5
D_ssd[2] — V5
D_ssd[1] — U7
D_ssd[0] — V7
state—L1
LED[14]—P1
LED[13]—N3
LED[12]—P3
LED[11]—U3
LED[10]—W3
LED[9]—V3
LED[8]—V13
LED[7]—V14
LED[6]—U14
LED[5]—U15
LED[4]—W18
LED[3]—V19
LED[2]—U19
LED[1]—E19
LED[0]—U16

Since we need to process the pulse signal from push button to produce perfect one pulse signal for FSM. Therefore, we need to make use of debounce and one-pulse circuits to do so. And as debounce circuit needs 100Hz clock, I output 100Hz by adding another counter in Lab3_2 (module). Then FSM will change - the state according to the state to decide if the timer is counting or not. Also, the result is shown on the 7-segment display.

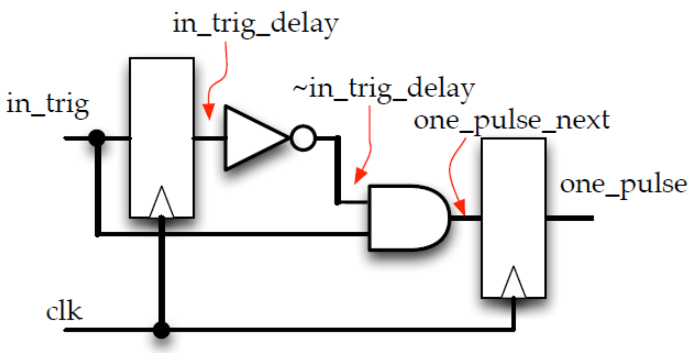
state diagram:



Debounce circuit:



One Pulse circuit:



Discussion

The key in this experiment is to control start/stop of timer as we have done other part of function in previous Labs. So, I used FSM to control the state of the timer. And after I drew out the state diagram, it's much easier to describe in Verilog.

2 The same function as Exp. 1, instead of using two push buttons for reset/pause/start, try to use just one push button to finish the design. (Hint: You can press the push button longer to represent the reset)

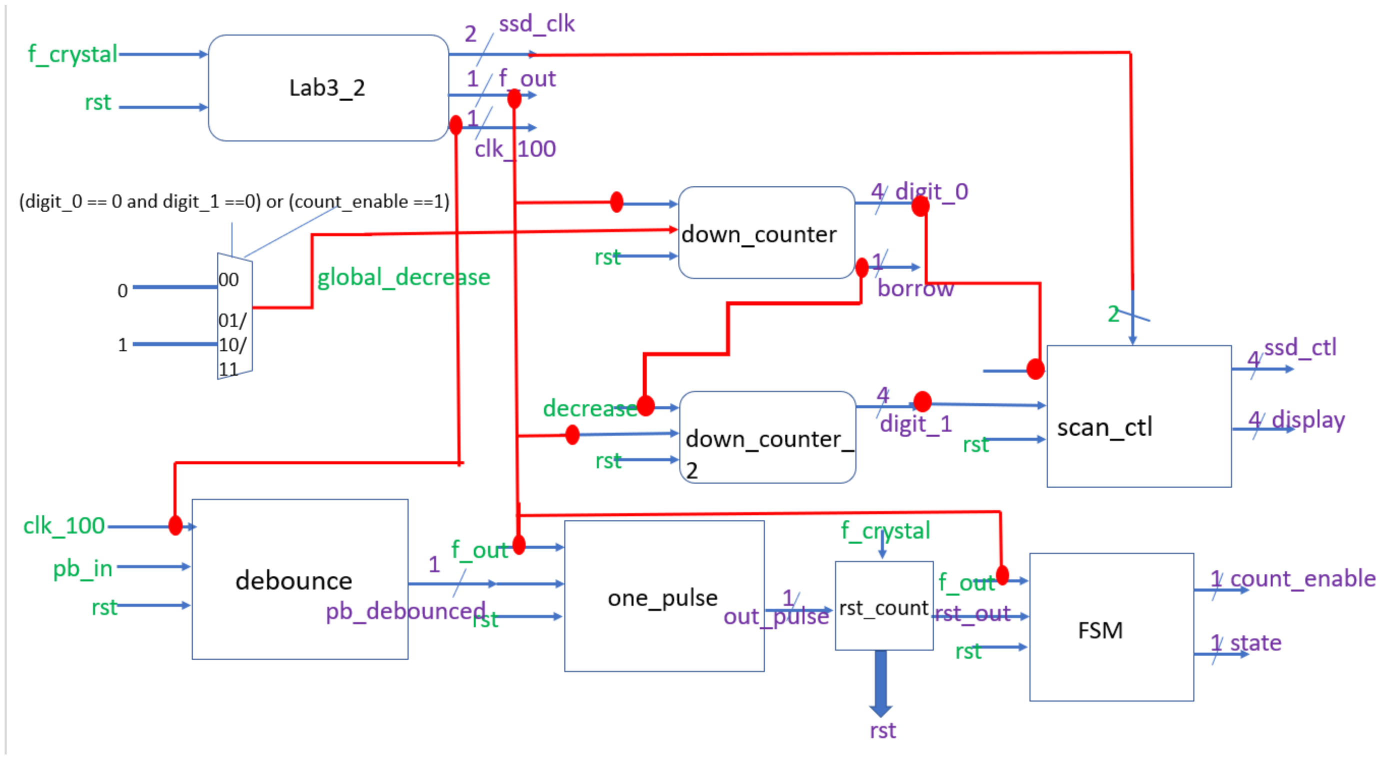
Design Specification:

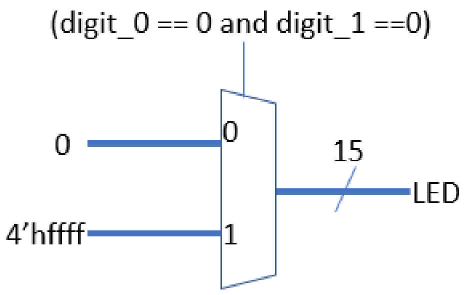
Input: f_crystal, pb_in

Output: [3:0]ssd_ctl, [7:0]D_ssd, [14:0]LED

Output: state

Block diagram:



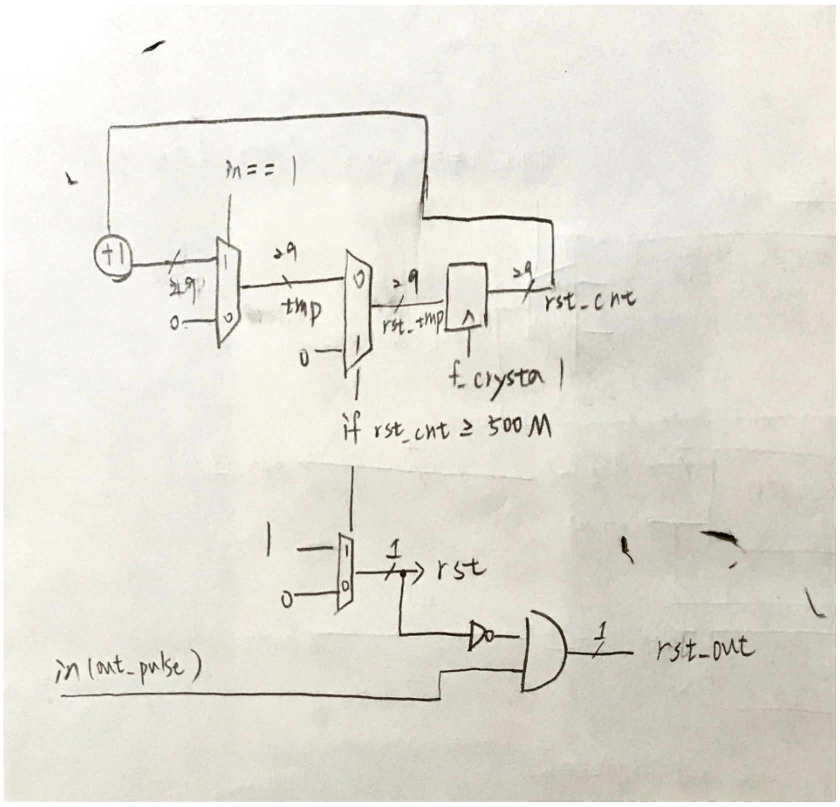


Design Implementation:

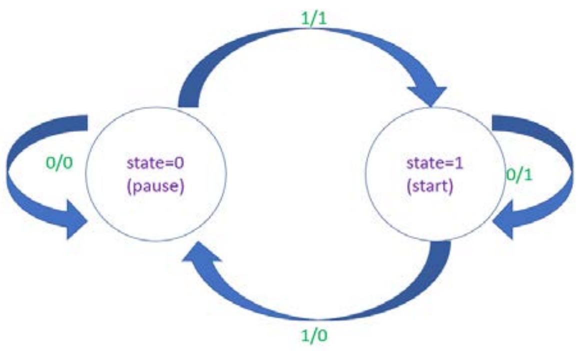
I/O pin assignment:

f_crystal—W5
pb_in—U17
ssd_ctl[3]—W4
ssd_ctl[2]—V4
ssd_ctl[1]—U4
ssd_ctl[0]—U2
D_ssd[7]—W7
D_ssd[6] — W6
D_ssd[5] — U8
D_ssd[4] — V8
D_ssd[3] — U5
D_ssd[2] — V5
D_ssd[1] — U7
D_ssd[0] — V7
state—L1
LED[14]—P1
LED[13]—N3
LED[12]—P3
LED[11]—U3
LED[10]—W3
LED[9]—V3
LED[8]—V13
LED[7]—V14
LED[6]—U14
LED[5]—U15
LED[4]—W18
LED[3]—V19
LED[2]—U19
LED[1]—E19
LED[0]—U16

rst_count circuit:



State diagram:



This experiment is similar to Exp5_1. Instead of using another push button for reset, I have to use only one push button to implement reset/start/pause. Hence, I used how much long the button is pushed to check if it's reset signal (i.e. long press means reset.). I implemented this function in the rst_count module in the block diagram, which is just a counter. If the button is pushed, it started to count. When the counter value reaches 500M (using 100MHz clock from FPGA board directly for convenience), which means about 5 seconds, it will send reset signal. And it output 0 to FSM to stay pause.

Discussion:

The key to this experiment is to decide what the signal represents, reset/start/pause. So the easiest way is to count how much time the button has been pushed. By doing so, I think it can reduce the use of buttons as the # of buttons are limited. With one button performing more functions, FPGA could perform more complicated tasks.

3 (Bonus) Use two push buttons to control a multi-function timer (mode selection, reset, start, stop). The stop timer has two modes: 30-second/1-minute countdown. When being reset, the seven-segment display shows the digits 30/1:00. When the timer counts to 0, it will stop.

3.1 List the specification of the detector.

3.2 Design the FSM used in this design.

3.3 Draw the block diagram/logic schematic.

3.4 Implement the stop timer with FPGA demo board.

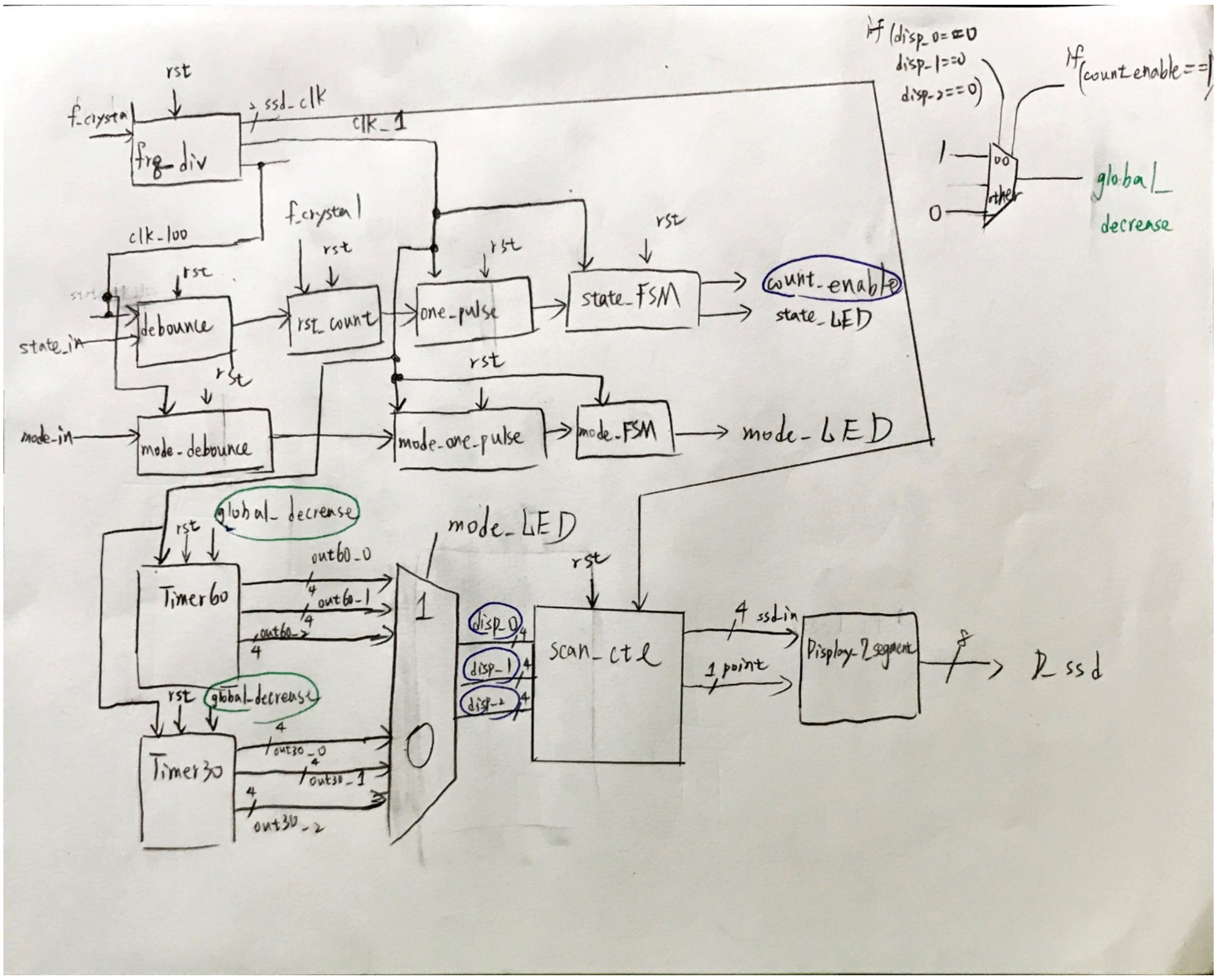
Design Specification:

Input: f_crystal, mode_in, state_in

Output: state_LED, mode_LED

Output: [3:0]ssd_ctl, [7:0]D_ssd

Block diagram:

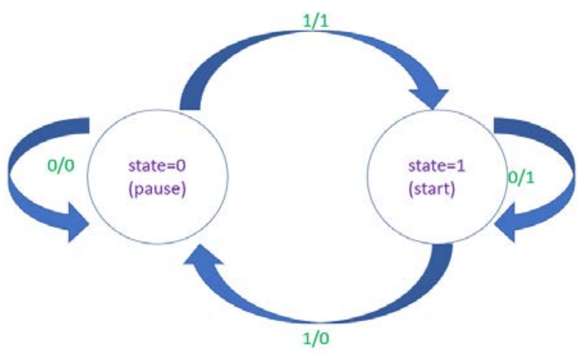


Design Implementation:

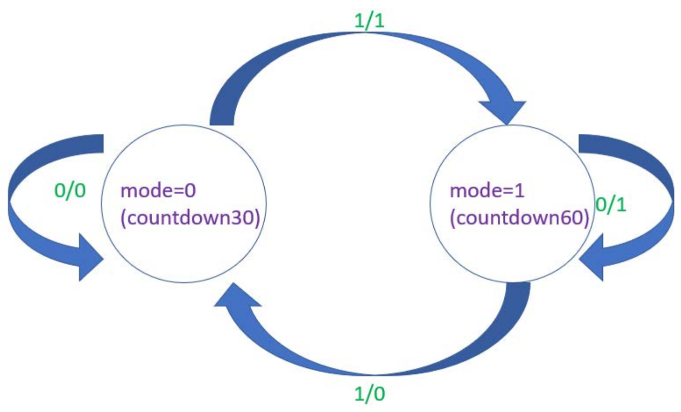
I/O pin assignment:

f_crystal—W5
state_in—W19
mode_in—T17
state_LED—L1
mode_LED—P1
ssd_ctl[3]—W4
ssd_ctl[2]—V4
ssd_ctl[1]—U4
ssd_ctl[0]—U2
D_ssd[7]—W7
D_ssd[6] — W6
D_ssd[5] — U8
D_ssd[4] — V8
D_ssd[3] — U5
D_ssd[2] — V5
D_ssd[1] — U7
D_ssd[0] — V7

State diagram for pause/start:



State diagram for mode (coundown30/countdown60)



There are two modes (countdown30/60) and two states (start/stop) for each mode. So it needs two push button to implement. And both push buttons are both require debounce and one pulse to deal with. If it's mode0 (countdown30), then I show the timer of 30 seconds on 7-segment display. Otherwise, I show the timer of 60 seconds on 7-segment display. And note that the decrease signal of Timer30 and Timer60 are connected together, which means they will countdown at the same time. By the choice of mode, the result on 7-segment display is different.

Discussion:

It seems a very complex experiment at first. Although there are two state diagram, one is just for mode choice and the other one is start/stop/reset control. So depending on the current mode, I switched to its corresponded timer. Overall, it's indeed complicated but not very difficult to implement it. What I do is to decide what each module is responsible for and connect them together in a top module.

Conclusion for Lab5:

In this lab, I know the importance of FSM, which is widely used in our daily life. And it's important to draw the state diagram first before write FSM in Verilog, which would be easier and clear. Also, I have also learned how to process signal from push button, which is debounce and one pulse. Last but not least, I made an serious error in the lab, which took me several hours to solve. I connect a output from combinational logic directly to its input, which then cause a error when generating the bitstream. I google what the error message meant and I found it. I started to check which part of code has the problem Google said. And I did again find it and solve it of course. From this experience, I know more about how Vivado works and how to avoid the same mistake next time.

And after completing this Lab, I found some unnecessary design in **rst_count** module used in Lab5_2 and Lab5_3. I thought I need to invert the input when rst signal is equal to 1 as reset signal is not supposed to mean input is 1. But I found it's unnecessary and would not affect the correctness of experiment. As I used long press to implement reset, at the same button, it will send 1'b1 signal to FSM anyway (as I am pressing that button). Therefore, although after the reset is equal to 1 later, FSM has already its state. FSM will not change its state until next 1'b1 signal sent to FSM. Hence, that is unnecessary design.

Reference for Lab5:

Handouts given by professor. From the handout, I know the circuit for debounce, one pulse and why we need them on push button. Also, I know how to describe FSM in Verilog