1 Implement a stopwatch function (00:00-59:59) with the FPGA board.

1.1 Use the four (Seven-Segment Displays, SSDs) as the display. The left two digits represent the minute and the right two digits represent the second.

1.2 Use two push buttons to control the function. Use one button to control start/stop and the other to control the lap and reset. When the stopwatch counts, press the 'lap' button will freeze the SSDs but the stopwatch continues counting, and when press the 'lap' button again, the SSDs will start to show current time.

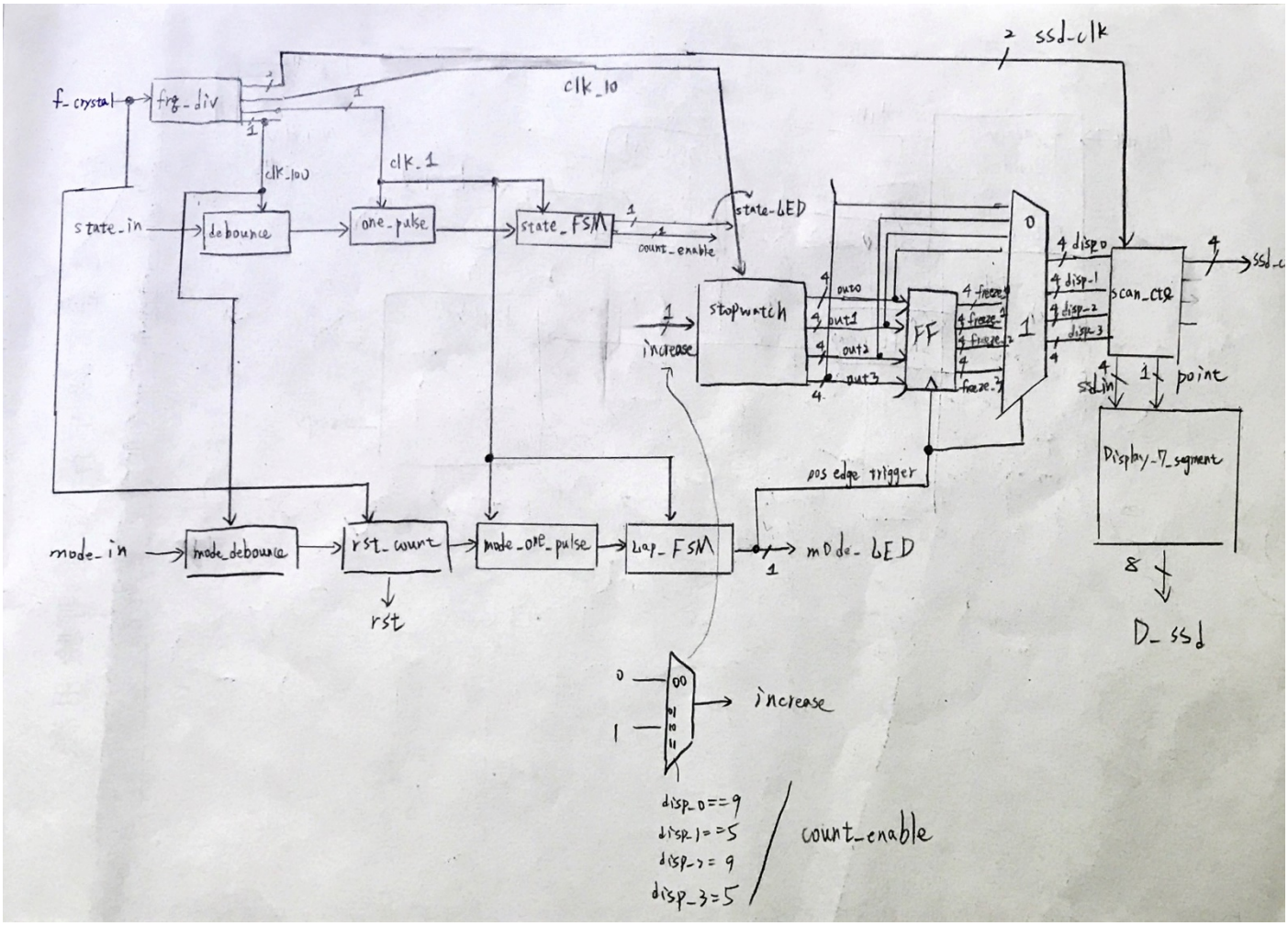
Design Specification

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

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

f_crystal — W5

state_in — W19

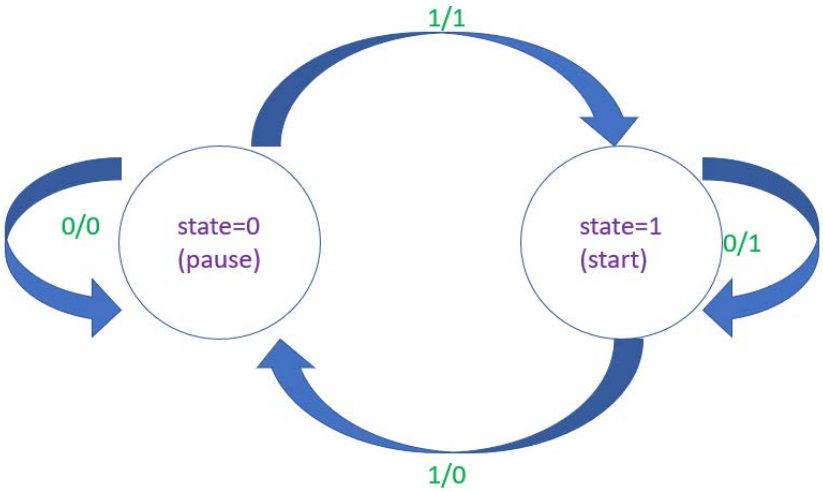
mode_in—T17

state_LED—L1

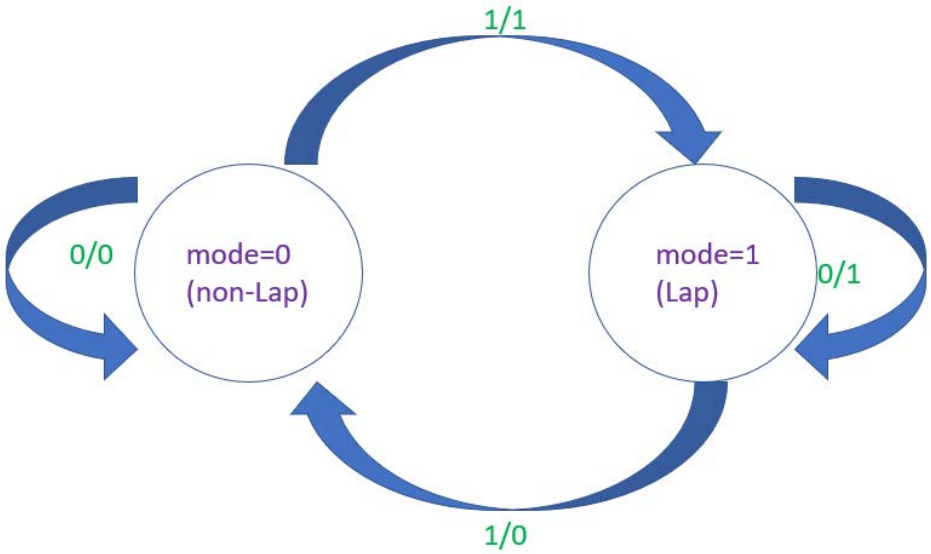
mode_LED—P1

state diagram

state 0/1: pause/start



Mode 0/1: non-Lap/Lap



In this experiment, the function of Pause/Start is the same as previous Lab. So I just need to figure out how to do the function of Lap. As shown in the block diagram, I used mode_LED to serve as clock for FF, so when it's the state of Lap, FF would remember the value of 7-segment display. Then, I used a MUX to choose if I need to output freezing result or the counting one according to mode_LED as well (if it's equal to 1, it shows freezing result. Otherwise, it shows the counting result)

Discussion:

As explained above, I came up with the method to freeze the 7-segment result while still counting. And I applied the module done in previous Lab so that this experiment was completed. For example, frq_div would output different kinds of clock frequency to different modules. I used 10Hz clock for stopwatch module so that it could count quicker and make sure my stopwatch could support 00:00 to 59:59. And the same debounce, one pulse module for each button respectively.

2 Implement a timer (can support as long as 23:59) with the following functions.

2.1 Use one DIP switch as the 'setting' control. When the 'setting' is ON, you can use two buttons to set the minute and second.

2.2 Use other two buttons to control the timer operation. One button for start/stop and the other button for pause/resume.

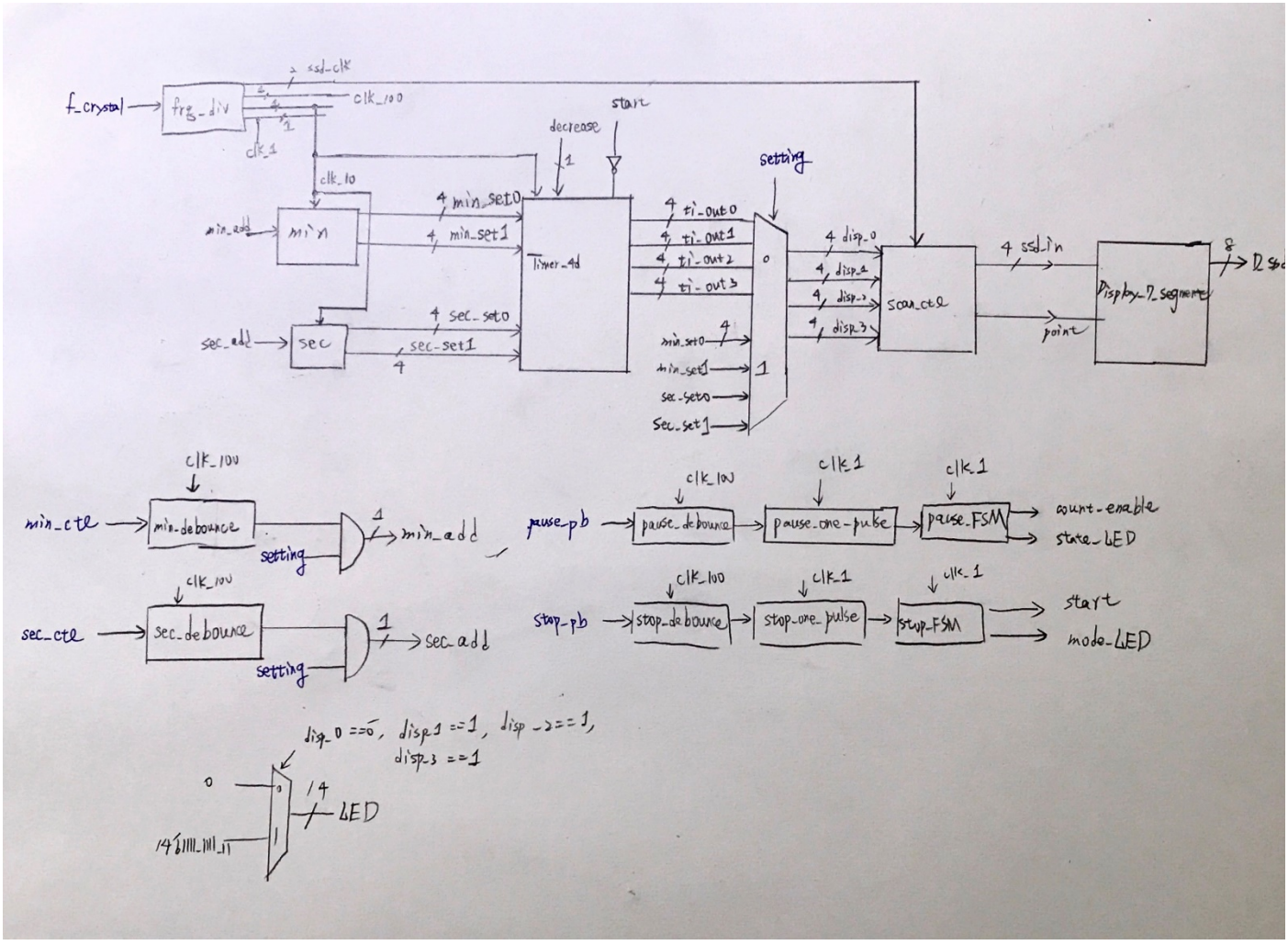
2.3 When the time goes to 0, light up all the LEDs.

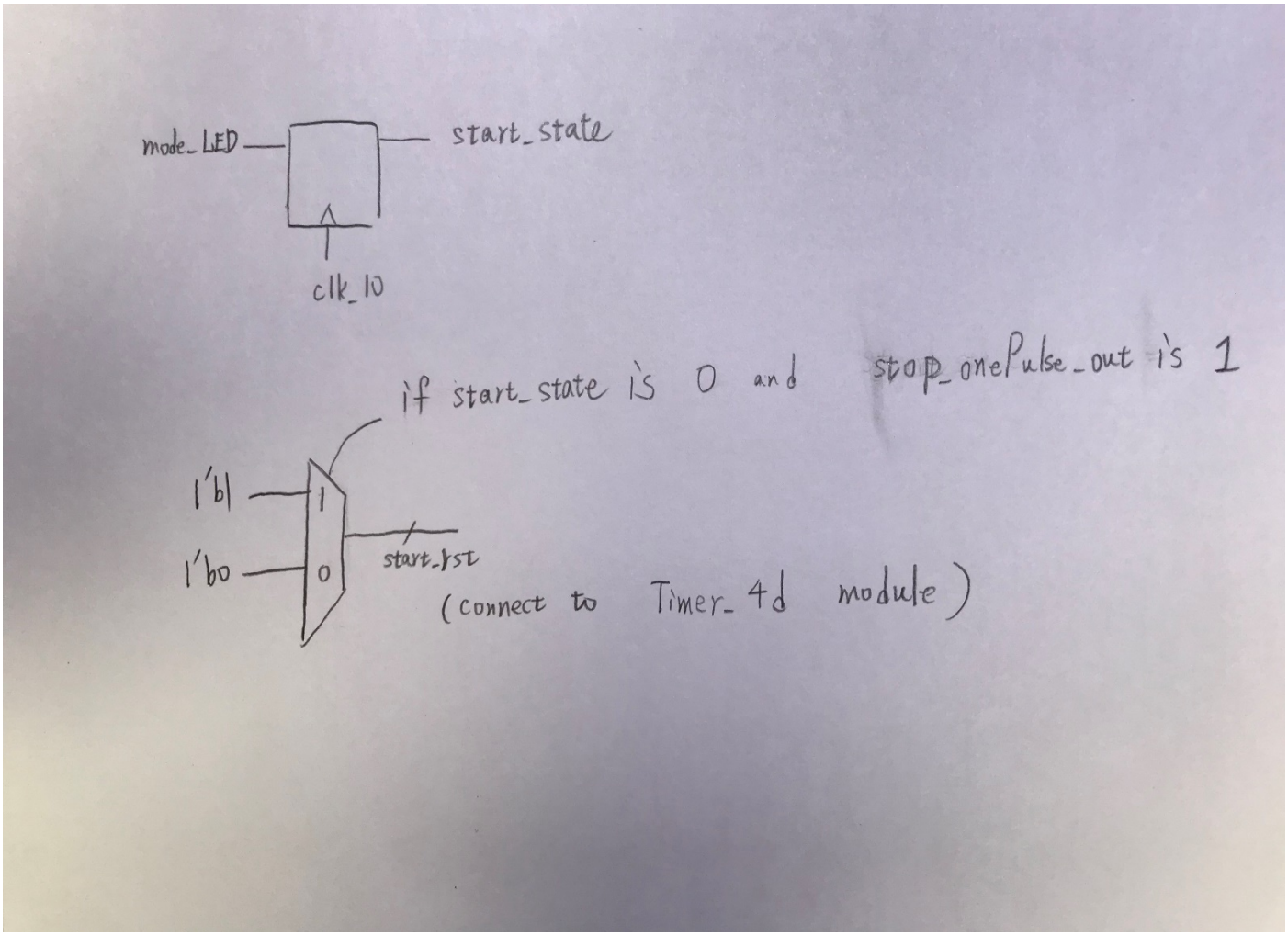
Design specification:

Input: f_crystal, rst, setting, pause_pb, stop_pb, sec_ctl, min_ctl

Output: state_LED, mode_LED, [3:0]ssd_ctl, [7:0]D_ssd, [13:0]LED

Block diagram:





Design Implementation:

I/O pin assignment

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

f_crystal — W5

pause_pb — W19

stop_pb — T17

min_ctl—U17

sec_ctl — T18

setting — R2

rst — T1

state_LED—L1

mode_LED—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

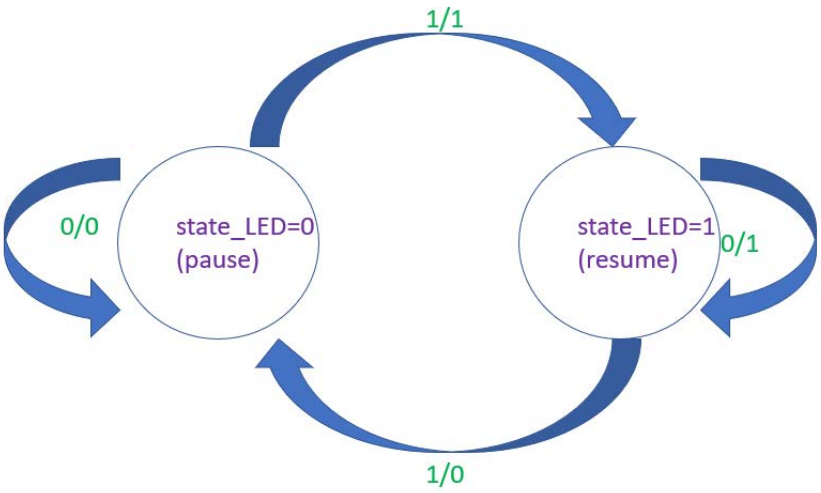
LED[2]—U19

LED[1]—E19

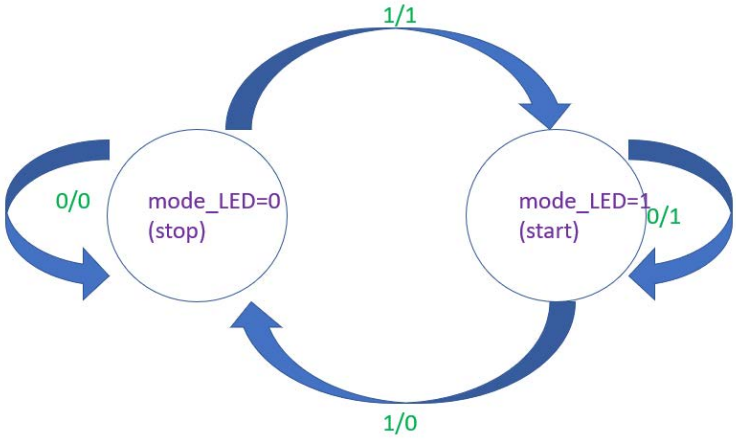
LED[0]—U16

State diagram:

State for pause/resume: state=0/1



State for start/stop:



In this experiment, pause and resume is the same function we did in previous Lab. So the difficulty here in this experiment is how to set minute/second to any value ranging from 00:00 to 23:59 and implement start/stop function.

Since there is a DIP switch to control if the minute and second could be set or not, I need to AND the input from this DIP switch and push button signal. Note that push button signal doesn't need to go through on pulse processing as the longer the push button is pressed, the higher the value is going to be set.

As for start/stop function, I use another Flip Flop to remember the current state. If it's stop state, then the 4-digit counter would just stop. When pressing again this same push button (one pulse would output 1), I use a MUX to check the state remembered by the Flip Flop (stop state), the counter would go back to the initially set value and start count. The Flip Flop is used because I want to make sure the MUX could take the right value of state. Otherwise, state would change very quickly as soon as the push button is pressed so that the MUX would be possibly too late to take the state value.

Also, according to the value of setting, the 7-segment display shows different result. If setting == 1, the show the result of set value, which would change along the time, if keeping pressing push button. Otherwise, it shows the down counting result.

Discussion:

After understanding what this experiment wants I do, I analyze which module I haven't done before. For example, although I have done timer before, I combine 4 single digit timer into a big module with 4 digits timer. Similar function for sec & min, I combine single digit counter to complete it. Then, I combine each module together in a top module.

I think another key point is how to implement the start/stop function correctly. I use Flip Flop to remember the state and detect the onePulse output to check if the button is pressed. This makes me think of STOP button (正方形的按鈕) on the radio, which is fun and realistic stuff.

When doing this experiment, I found some errors on 7-segment display. But due to the lack of experience in debugging Verilog code, it took me several hours to find the simple bug, which is I forgot to declare a point (i.e. : in 7:36) in Display_7_segment module. I thought it was problems from other modules like Timer_4d or min/sec. None of them are the problem here. After this experience, I found it's not that easy to debug Verilog. But I guess more experience to get better at this.

3 (Bonus) Integrate the above two functions together with only three buttons.

- Note: You can use one addition button for reset in this lab.

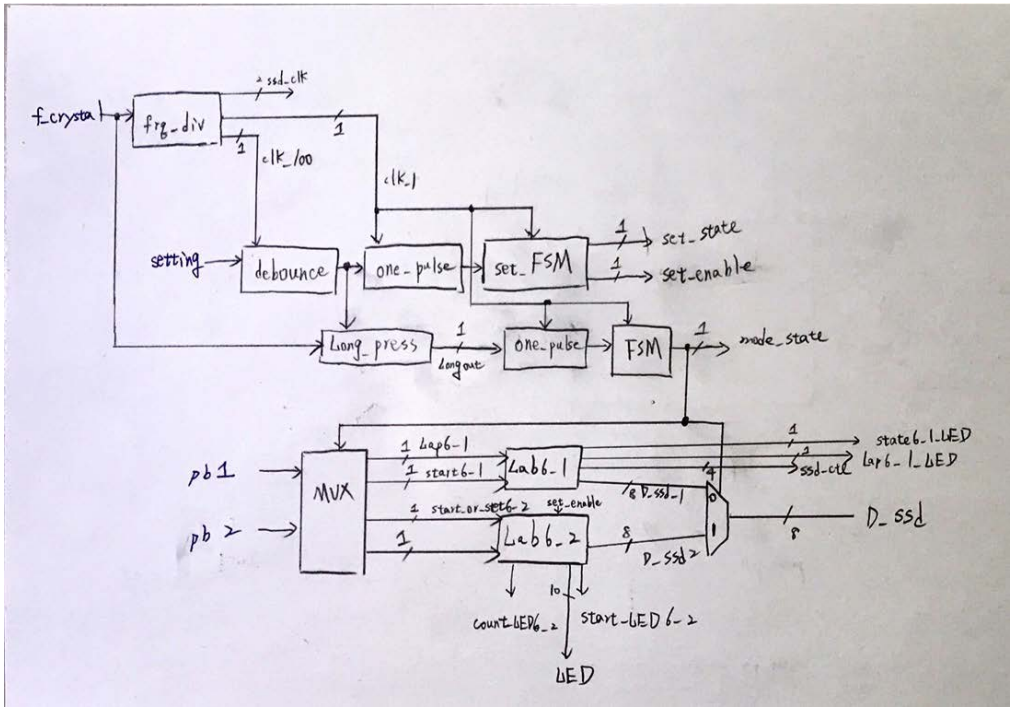
Design Specification

Input: pb1, pb2, reset, setting, f_crystal

Output: set_state, mode_state, Lap6_1_LED, state6_1_LED, start_LED6_2, count_LED6_2

Output: [9:0]LED, [3:0]ssd_ctl, [7:0]D_ssd

Block diagram:



Design Implementation:

I/O pin assignment

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

f_crystal — W5

setting — T18

reset — U18

pb1 — W19

pb2 — T17

set_state—L1

mode _state—P1

state6_1_LED—N3

Lap6_1_LED—P3

count_LED6_2—U3

start_LED6_2—W3

LED[9]—V3

LED[8]—V13

LED[7]—V14

LED[6]—U14

LED[5]—U15

LED[4]—W18

LED[3]—V19

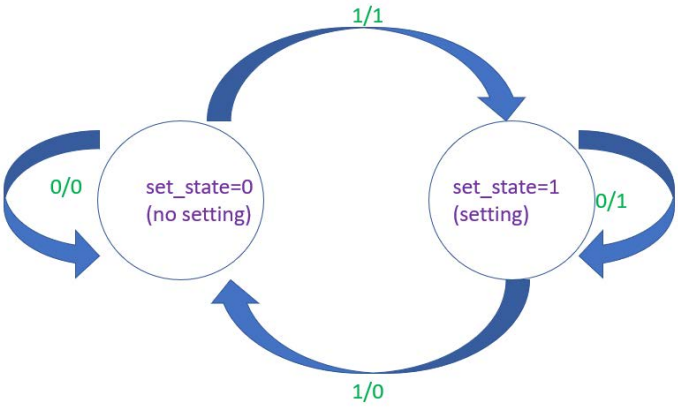
LED[2]—U19

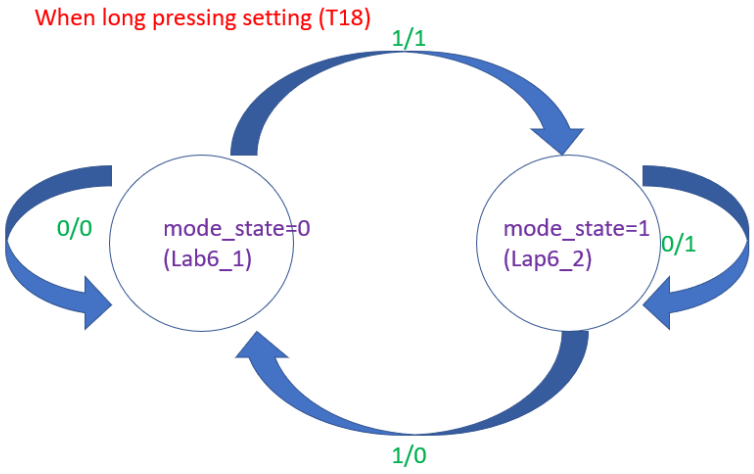
LED[1]—E19

LED[0]—U16

State diagram

Setting (T18) state diagram:





Other state diagram are the same diagram in Lab6_1 and Lab6_2!

In this experiment, the most important thing is to come up with a way to use just 3+1 (reset) push buttons to combine all the function of Lab6_1 and Lab6_2.

First of all, use setting button (T18) to output **set_enable** signal to decide if minute and second can be set. And at this same push button, when long pressing it, it could switch between different modes (Lab6_1 or Lab6_2)

If it's Lab6_1, I can use the other two buttons (W19, T17) to preform 'lap' and 'start/pause' function. If it's Lab6_2, I can use the signal set_enable to decide two buttons (W19, T17) are able to set minute and second or not. If set_enable == 1, then two buttons can be used to set minute and second. Otherwise, it is used to perform start/stop and pause/resume function.

Discussion:

As there are many functions to show but only 3+1 push buttons are allowed to use. So, I initially guess some functions are required long press button to implement it. So, I used long press to switch between Lab6_1 and Lab6_2 mode. At this very button, I also add setting function in Lab6_2 on it.

Although there are many similar modules in Lab6_1 and Lab6_2 at the same time, such as debounce, one pulse, FSM and so on. I didn't separate them from Lab6_1 and Lab6_2. Because it would be easier to debug when viewing Lab6_1 and Lab6_2 as a big module after making sure Lab6_1 and Lab6_2 are correct. Also, there would be less modification on Lab6_1 and Lab6_2 when doing so.

Conclusion for Lab6:

In this lab, there are more FSM applications and more modes to switch in a single experiment, which is more complicated compared to Lab5.

Also, in this Lab, I have learned FSM can not only preform pause/resume but also Lap/start/stop and shows the result on 7-segment display. All of these functions are common in our daily life, such as timer and stopwatch function in our smartphones.

Through this Lab, I find it's useful when thinking about how the experiment is being used in our daily life, such as timer, stopwatch and switch the mode between them. Sometimes, it's easy to think of their functions but hard to express them in Verilog. It's probably because I haven't think about details thoroughly.

And I have known that I can use the same module repeatedly. (For example, write debounce U0(…) and debounce U1(…) in Verilog as U0 and U1 can be viewed as different module with same function). Therefore, there will not be too many .v files with the same code, which looks way more clean.