**Lab6 Report**



**Ⅰ. Lab6\_1 (stopwatch 00:00 - 59:59 (lap) )**

**Design Specification**

IO:

Input: clk, rst\_n, button1, button2.

Output: [3:0] ssd\_ctl(anode of seven-segment display),

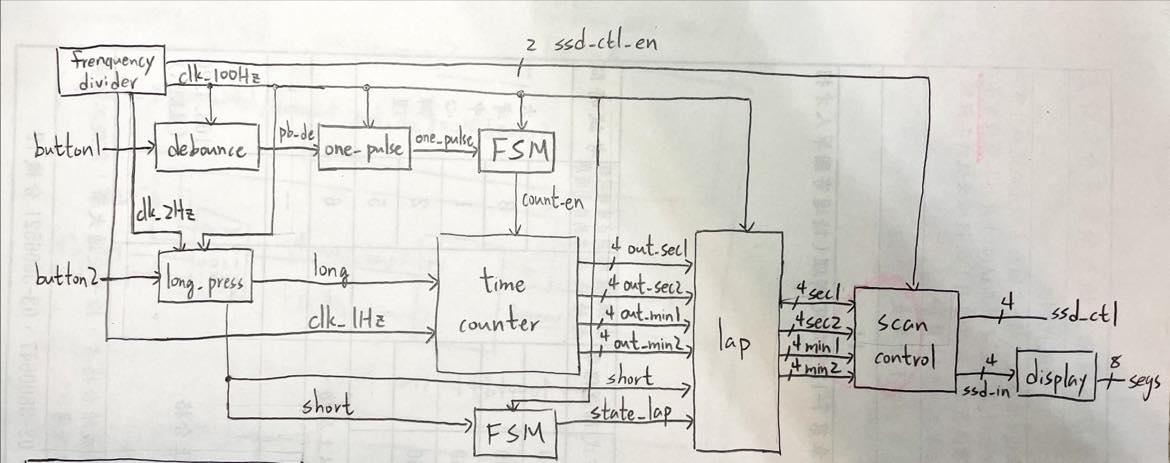
[7:0] segs(cathode of seven-segment display)

[1:0] LED.

**Design Implementation**

In order to readability, I divided this problem into night parts (scan, time counter, frequency divider, debounce, one-pulse, FSM, long press, lap, display) as follows. I will explain the detailed functions of each part below.

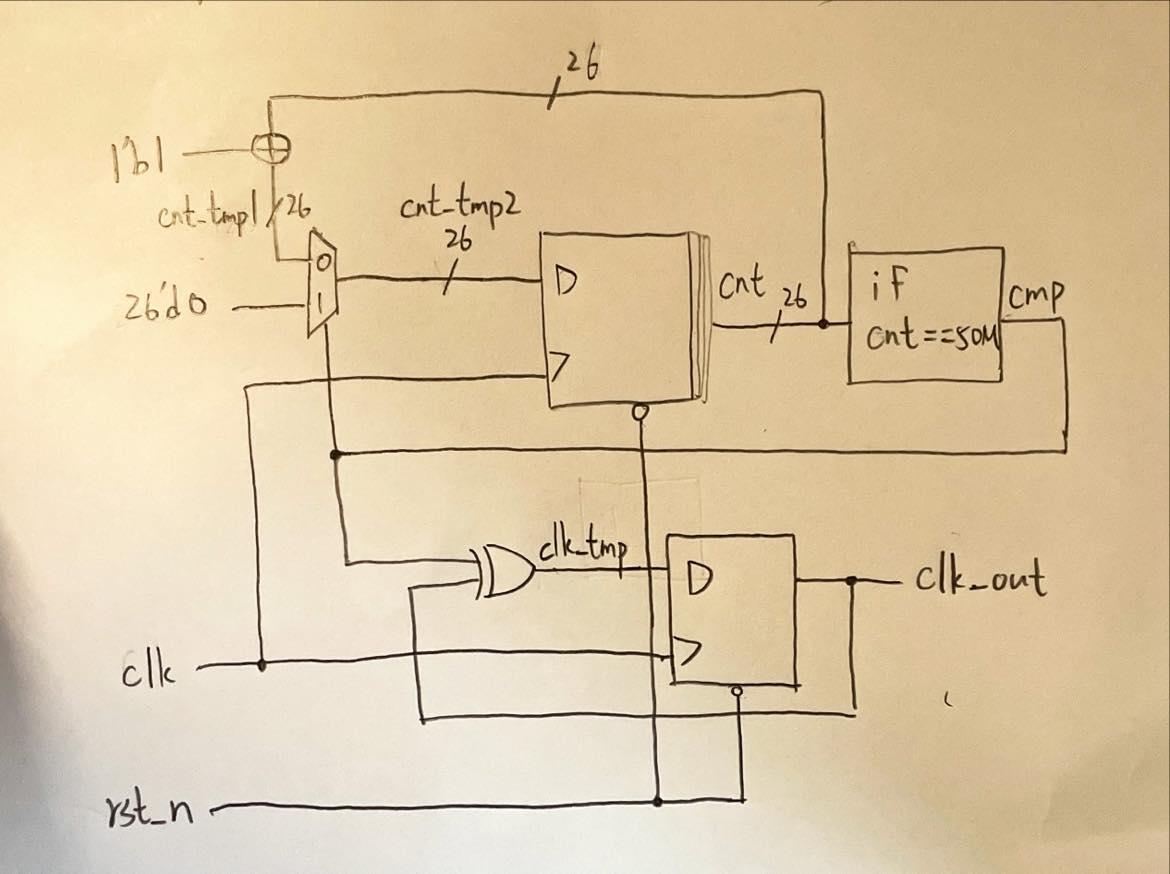
A simple Block diagram:



**Scan control**：Because this project needs four digits, I use an enable from 13th、14th bits of frequency divider to control the seven-segment display. We can see different numbers on four seven-segment displays by anode and cathode rapidly changing before the clock toggle. Furthermore, the left two digits represent the minute and the right two digits represent the second.

**Frequency divider**：This part, I make three clocks to control all modules (time counter, debounce, one-pulse, FSM, long press and lap)：1Hz clock controls time counter; 2Hz clock controls long press(more details in **Long press**); 100Hz clock controls the others(included long press). And I also need an enable to control scan.v, so I produce this enable with 2Hz clock and 100Hz clock .

1Hz clock：



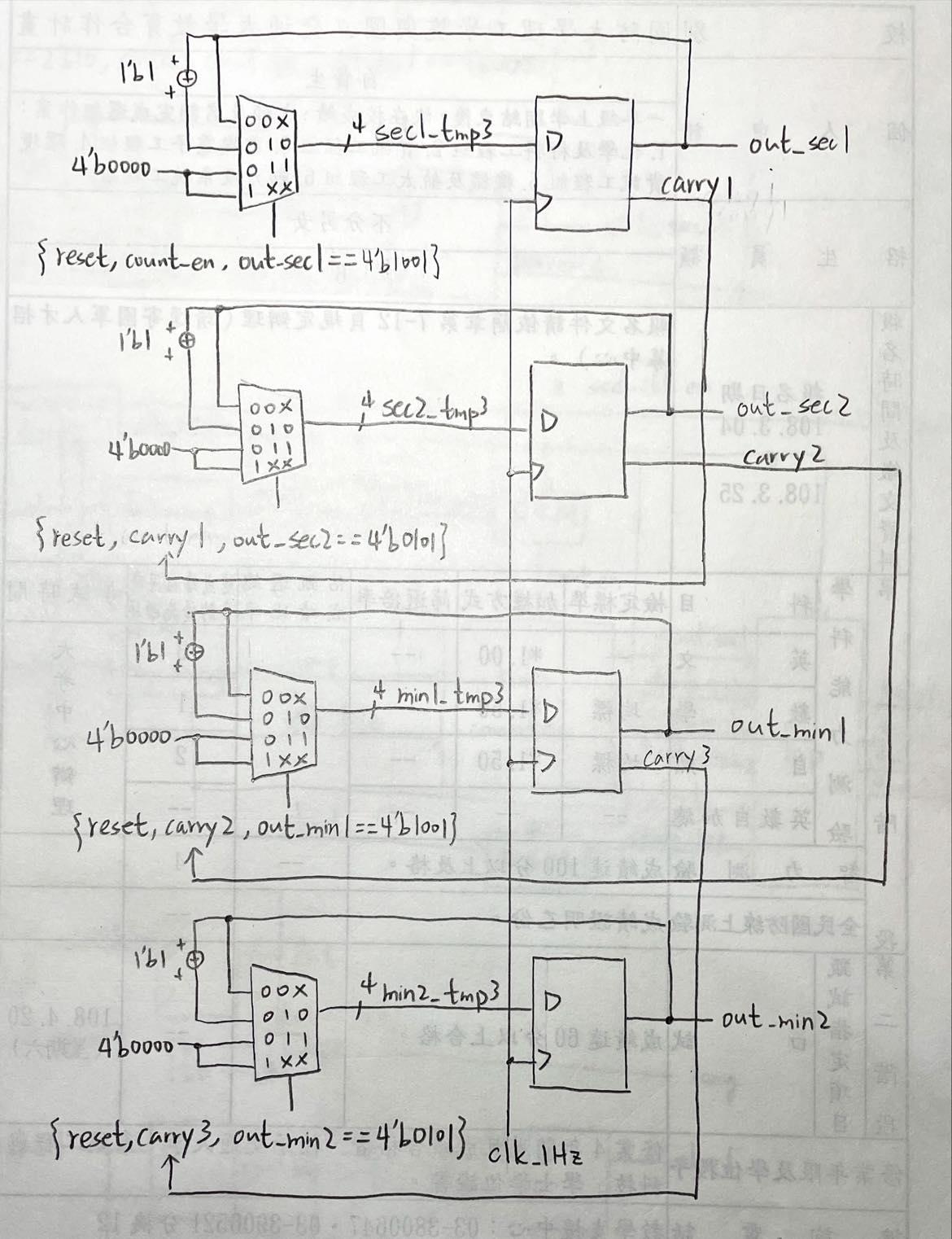
So to construct 1Hz clock, I use a counter to count until 50M, then declare a variable cmp.

If counter count to 50M, cmp = 1 and initialize counter. Therefore, T-flip flop, input cmp and clk\_out, toggle when cmp = 1, so clk\_out will become a 1Hz clock ( ).

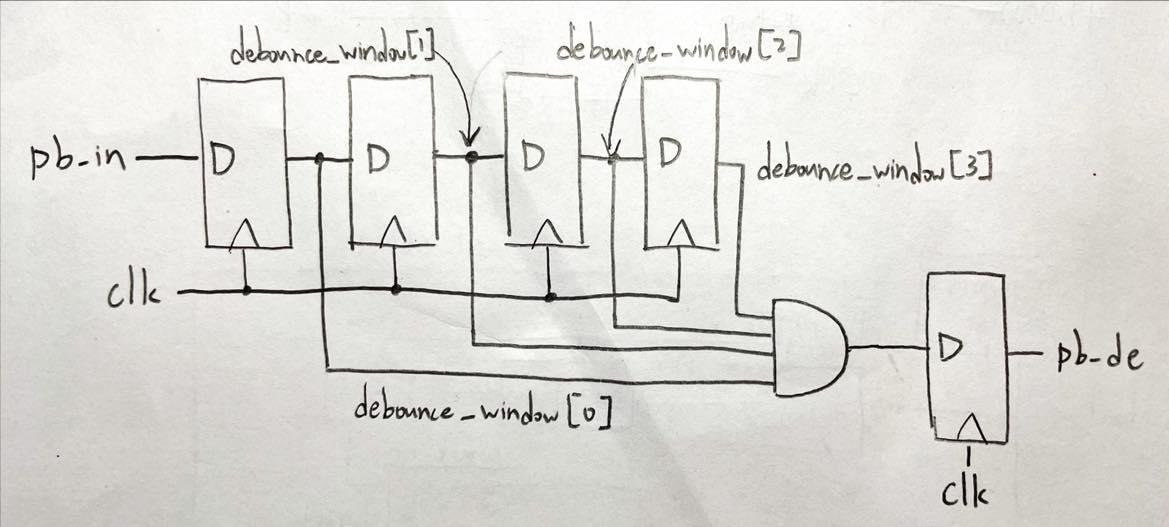
2Hz clock, 100Hz clock and scan enable：

use 26-bit-binary-up counter with initial clock (100MHz) to count and the highest bit is considered 2Hz clock, which is (≒2Hz); 21th bit is considered 100Hz clock, which is (≒100Hz). And output 13th and 14th bits, which are used as an enable for scan control.

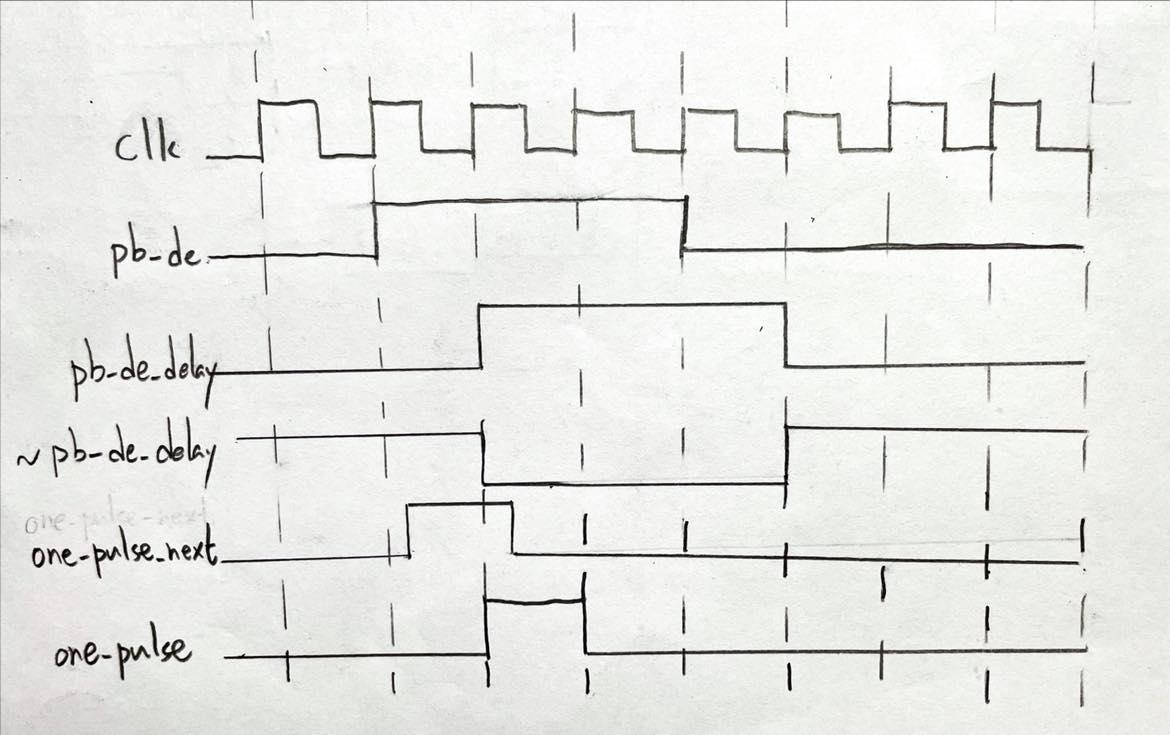
**Time counter**：I use following logic diagram to construct down counter.



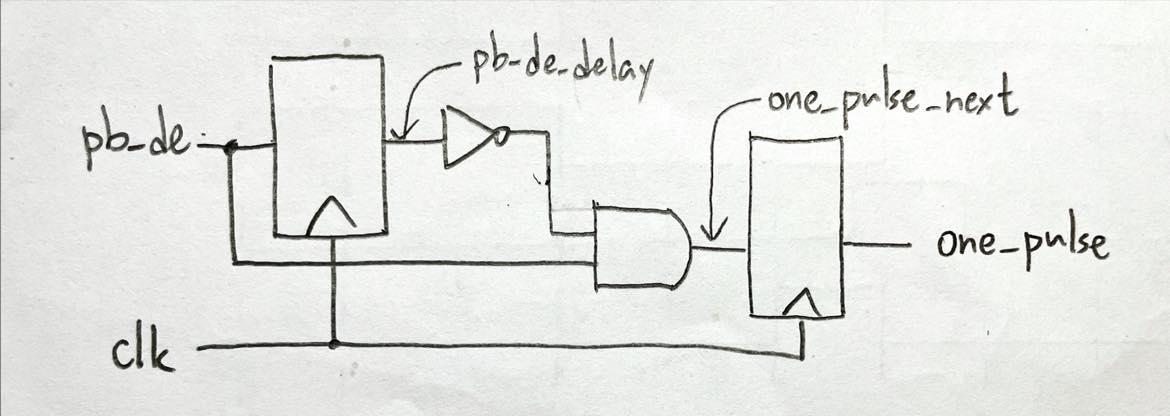
**Debounce**：Because of physical effect, when pressing the button, the spring in the button will produce the extra signal of bounce. Therefore, we need to remove the bounce which is called debounce. And the size of bounces is about µs, so we use 100Hz clock and four flip-flops to detect the signal. If four flip-flops’ value = 1, then signal = 1. Use this method to complete the function of debounce.



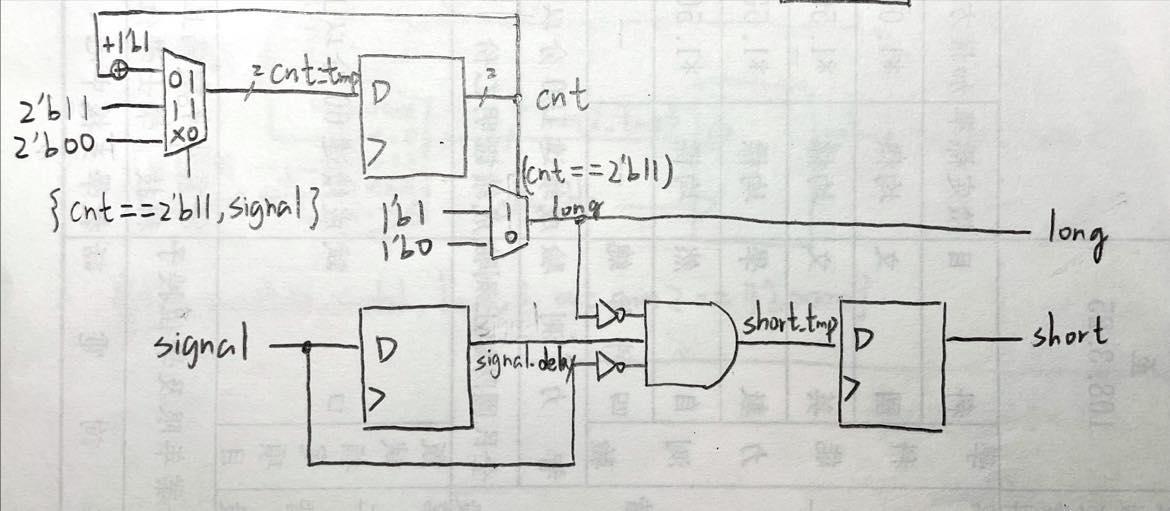
**One-pulse**：After debounce, we use one-pulse to amend the signal. If the time of pressing the button is so long that across more two clocks, then the state produced by FSM will be wrong. So we need to produce one-pulse which across only one clock. Therefore, we use the property of the signal as following timing diagram.



Then, use logic diagram to construct the code.

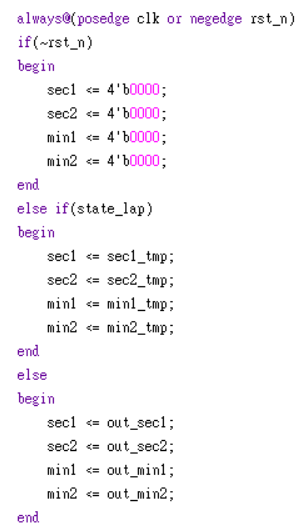
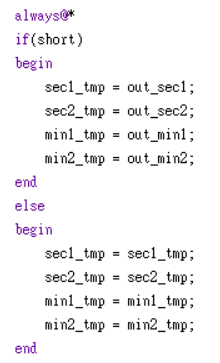


**Long press**：I don’t want the condition occurred, which is that if I attempt to reset all setting then the device will lap before reset, so I am not able to amend the lap button’s signal by one-pulse. I use this part to detect the negedge of signal (~signal & signal\_delay) and the moment of the signal’s negedge whether satisfies the condition of long press. If now it occurs signal’s negedge but not satisfies long press’s condition, then it is short press; else, it satisfies long press’s condition is long press.



**Lap：**This part decides the output is the value of lap or time counter, which is decided by the state (lap/non-lap). If short press = 1, then the value of time counter now store into flip-flop. If now state is lap, then output the value stored in flip-flop. If now state is non-lap, then the value of time counter go through the flip-flop directly.

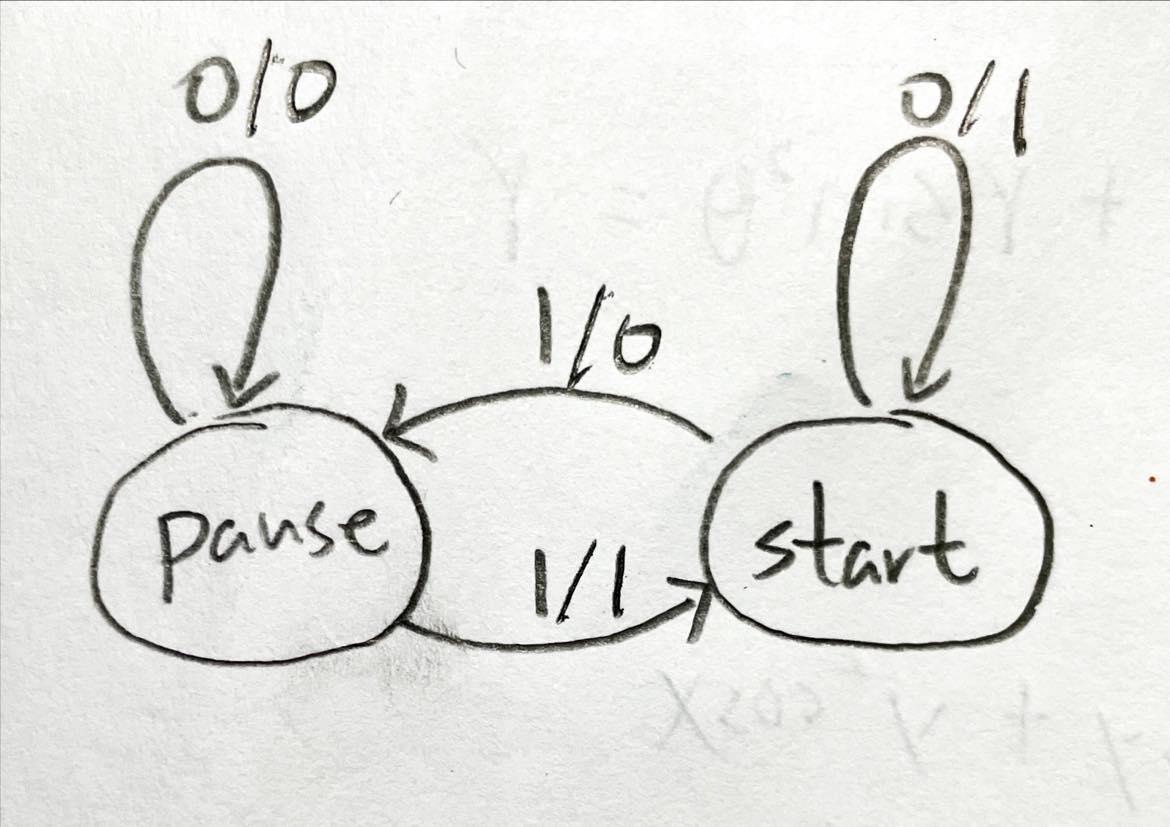
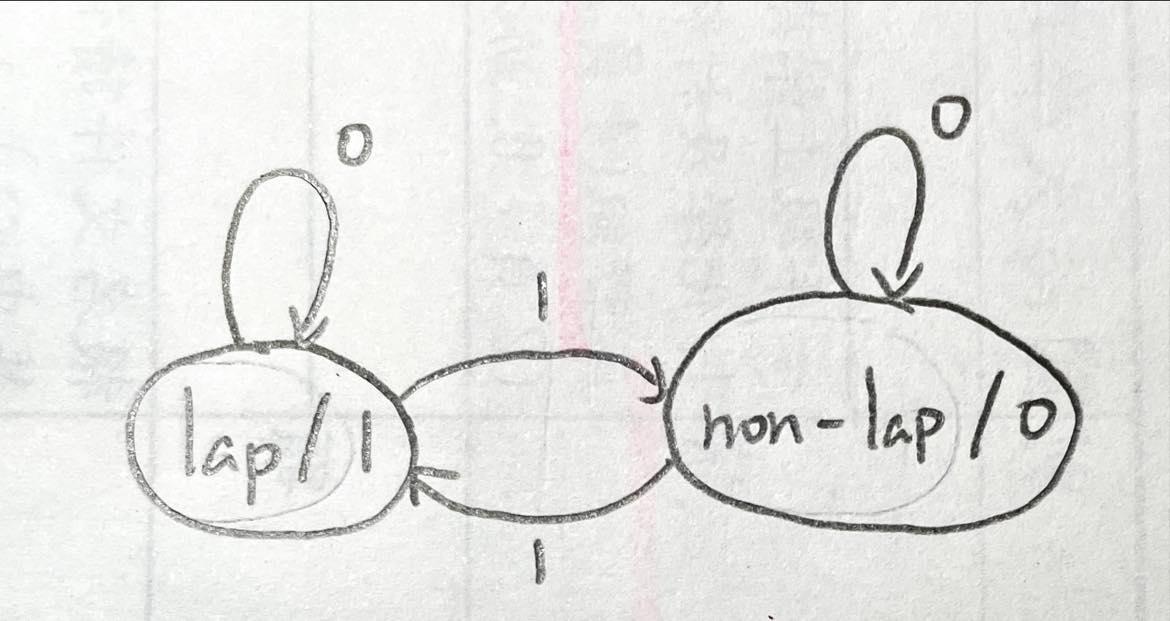
code: out\_sec and out\_min are outputs of time counter



**FSM (Finite State Machine)**：Because this project has two states (stop/start; lap/non-lap) occurred simultaneously but independently, I use the same FSM to control these four states.

State diagram: (mealy machine)

state: (mealy machine) mode: (moore machine)

Mealy machine (stop/start):

1. If state = 0 (pause) and signal is 0,

then keep the state and output (count enable) = 0.

1. If state = 0 (pause) and signal is 1,

then jump to next state (state = 1) (start) and output (count enable) = 1.

1. If state = 1 (start) and signal is 0,

then keep the state and output (count enable) = 1.

1. If state = 1 (start) and signal is 1,

then jump to next state (state = 0) (pause) and output (count enable) = 0.

Moore machine (lap/non-lap):

1. If state = 0 (non-lap), then output (state\_lap) = 0,

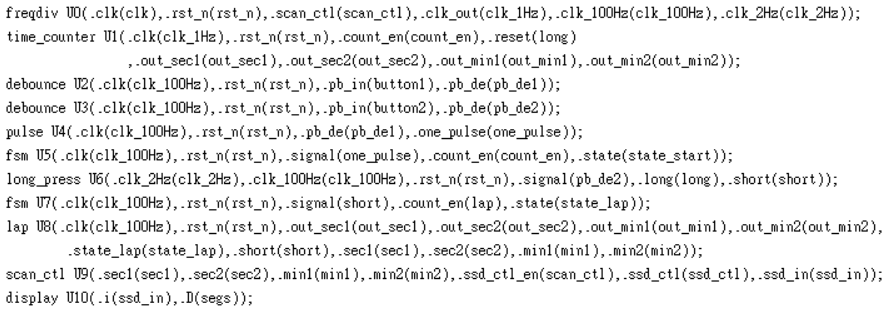
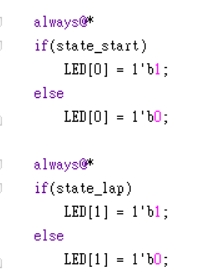
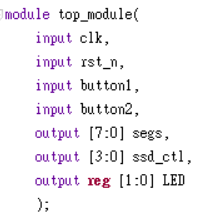
and if signal = 0, keep the state (non-lap); if signal = 1, jump to next state (lap).

1. If state = 1 (lap), then output (state\_lap) = 1,

and if signal = 0, keep the state (lap); if signal = 1, jump to next state (non-lap).

**Display**：is a decoder from BCD to cathode of seven-segment display.

**Top module**：to connect all modules and input, output the signal and use state from FSM to control 2 LEDs which show two state now occurred.



**IO pin assignment：**

| LED[1] | LED[0] |
| --- | --- |
| E19 | U16 |

| segs[7] | segs[6] | segs[5] | segs[4] | segs[3] | segs[2] | segs[1] | segs[0] |
| --- | --- | --- | --- | --- | --- | --- | --- |
| W7 | W6 | U8 | V8 | U5 | V5 | U7 | V7 |

| ssd\_ctl[3] | ssd\_ctl[2] | ssd\_ctl[1] | ssd\_ctl[0] | rst\_n | clk | button1 | button2 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| W7 | W6 | U8 | V8 | V17 | W5 | W19 | T17 |

**Ⅱ. Lab6\_2 (stopwatch with setting control)**

**Design Specification**

IO:

Input: clk, rst\_n, button1, button2.

Output: [3:0] ssd\_ctl(anode of seven-segment display),

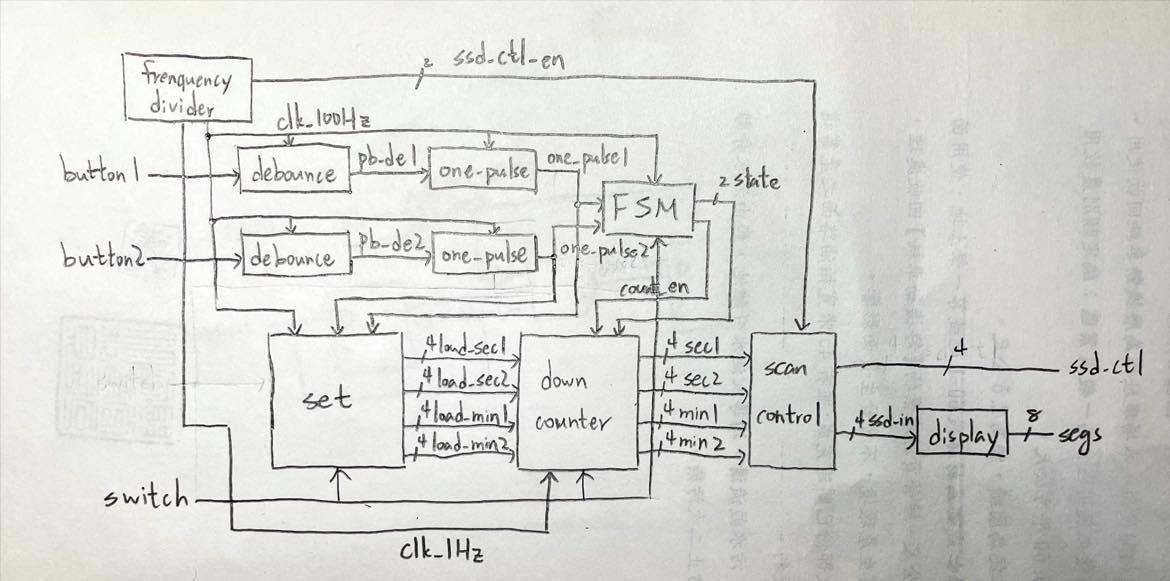
[7:0] segs(cathode of seven-segment display)

[15:0] LED.

**Design Implementation**

In order to readability, I divided this problem into eight parts (scan, down counter, frequency divider, debounce, one-pulse, FSM, set and display) as follows.

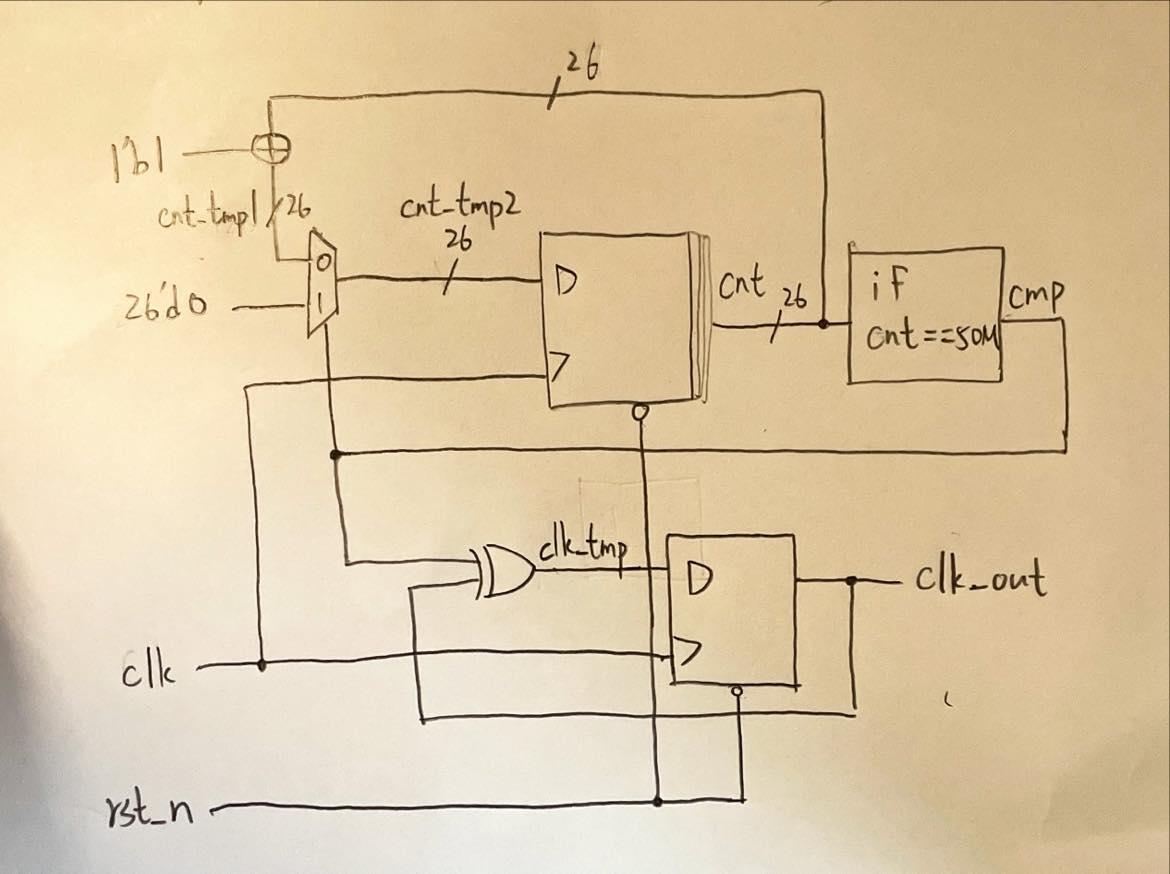
A simple Block diagram:

****

**Scan control (the same as lab6\_1)**：Because this project needs four digits, I use an enable from 13th、14th bits of frequency divider to control the seven-segment display. We can see different numbers on four seven-segment displays by anode and cathode rapidly changing before the clock toggle. Furthermore, the left two digits represent the minute and the right two digits represent the second.

**Frequency divider**：Because I need two clocks to control down counter, debounce, one-pulse, FSM and set：1Hz clock controls down counter and 100Hz clock controls the others. And I also need an enable to control scan.v, so I produce this enable with 100Hz clock.

1Hz clock：



(the same as lab6\_1)

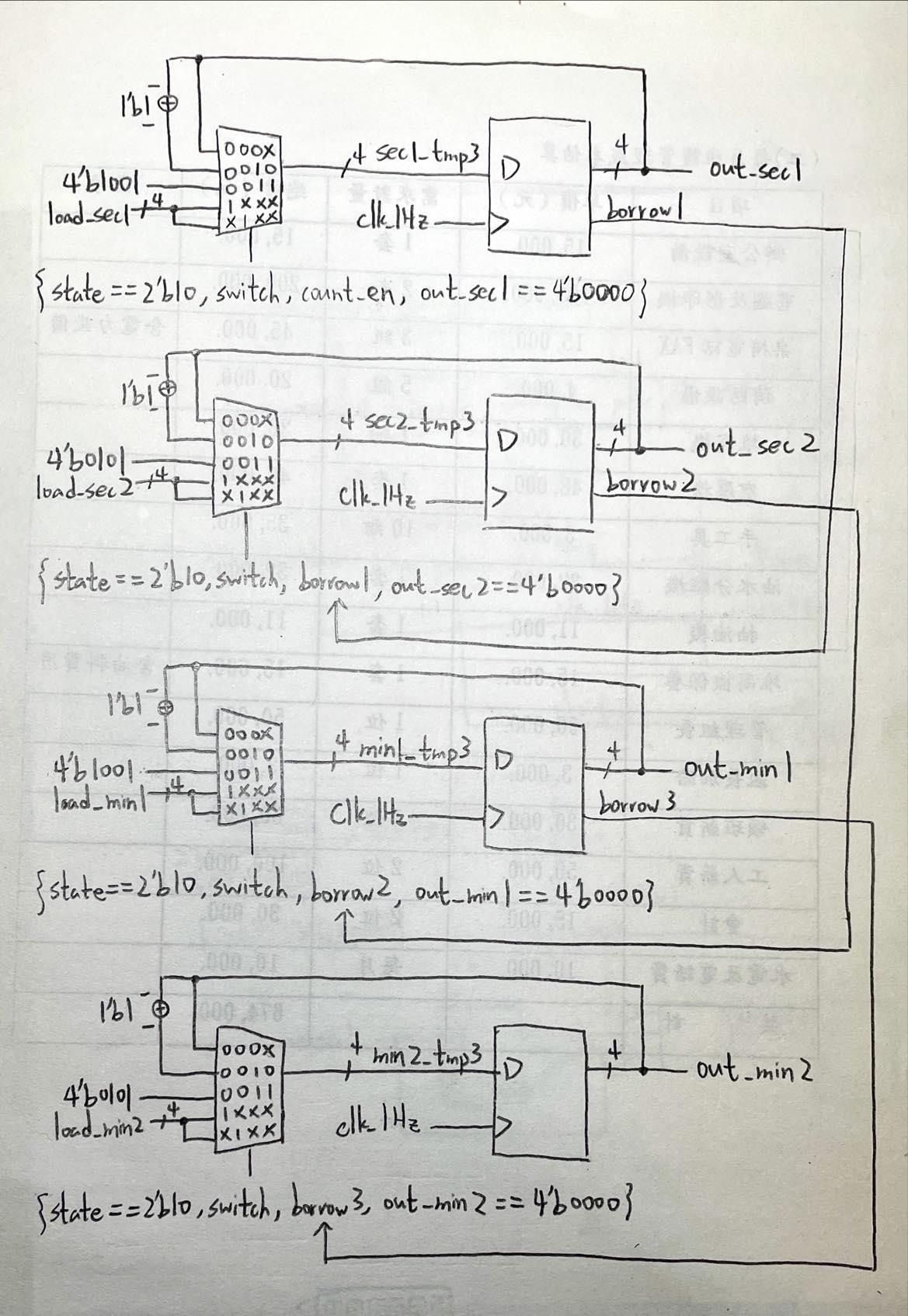
To construct 1Hz clock, I use a counter to count until 50M, then declare a variable cmp.

If counter count to 50M, cmp = 1 and initialize counter. Therefore, T-flip flop, input cmp and clk\_out, toggle when cmp = 1, so clk\_out will become a 1Hz clock ( ).

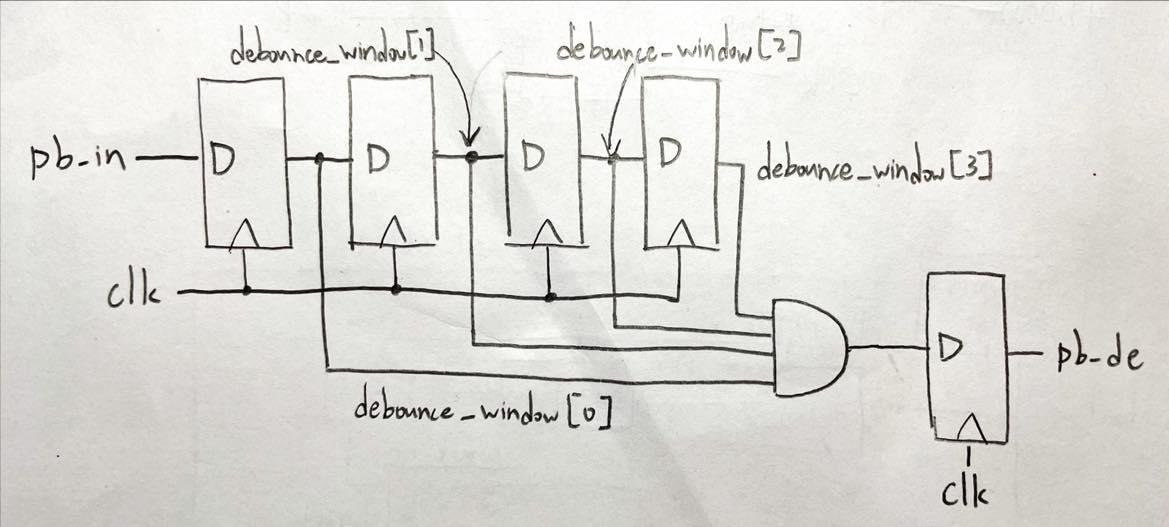
100Hz clock and scan enable：

use 21-bit-binary-up counter with initial clock to count and the highest bit is considered new clock (100Hz), which is (≒100Hz). And output 13th and 14th bits, which are used as an enable for scan control.

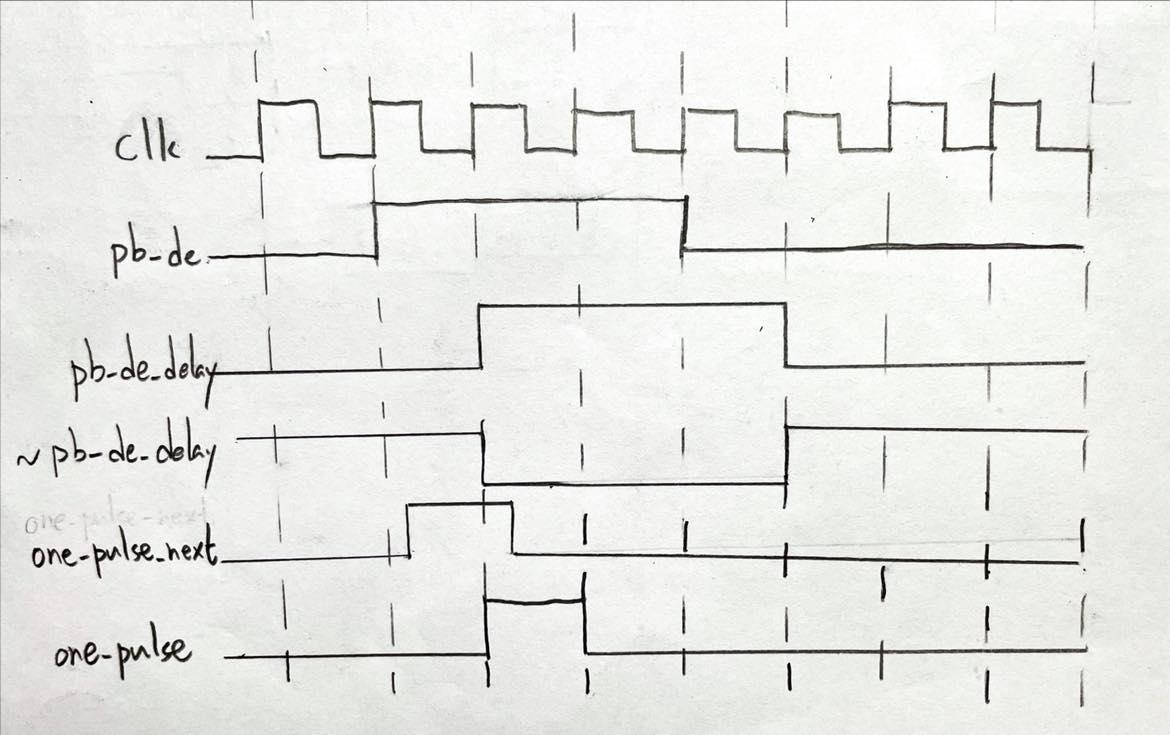
**Down counter**：I not only use following logic diagram to construct down counter but also add some conditions. For example, if out\_sec1, out\_sec2, out\_min1 and out\_min2 = 0, then keep out\_sec1, out\_sec2, out\_min1 and out\_min2 = 0 and LED\_ctl = 1 to turn on all LED. When the setting switch turns up, counter will stop and out\_sec1, out\_sec2, out\_min1 and out\_min2 will connect directly with the outputs of set.



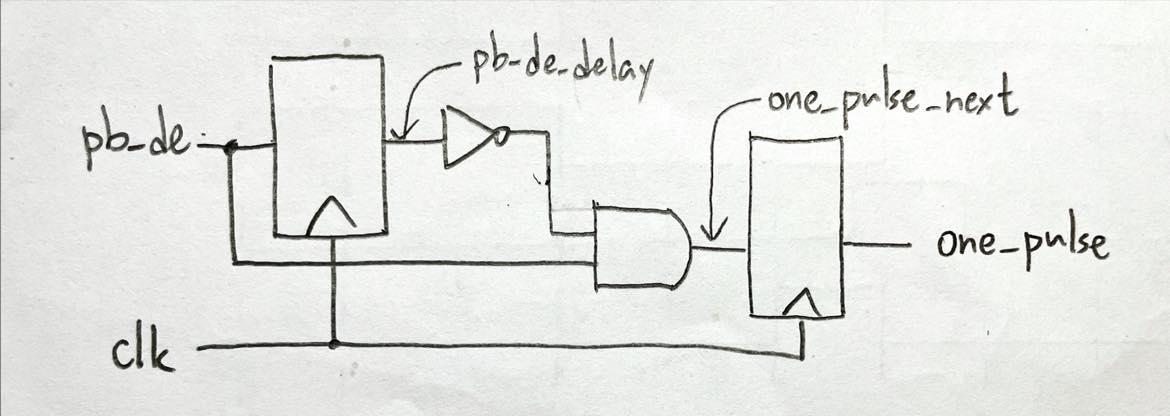
**Debounce (the same as lab6\_1)**：Because of physical effect, when pressing the button, the spring in the button will produce the signal of bounce. Therefore, we need to remove the bounce which is called debounce. And the size of bounces is about µs, so we use 100Hz clock and four flip-flops to detect the signal. If four flip-flops’ value = 1, then signal = 1. Use this method to complete the function of debounce.



**One-pulse (the same as lab6\_1)**：After debounce, we use one-pulse to amend the signal. If the time of pressing the button is so long that across more two clocks, then the state produced by FSM will be wrong. So we need to produce one-pulse which across only one clock. Therefore, we use the property of the signal as following clock diagram.

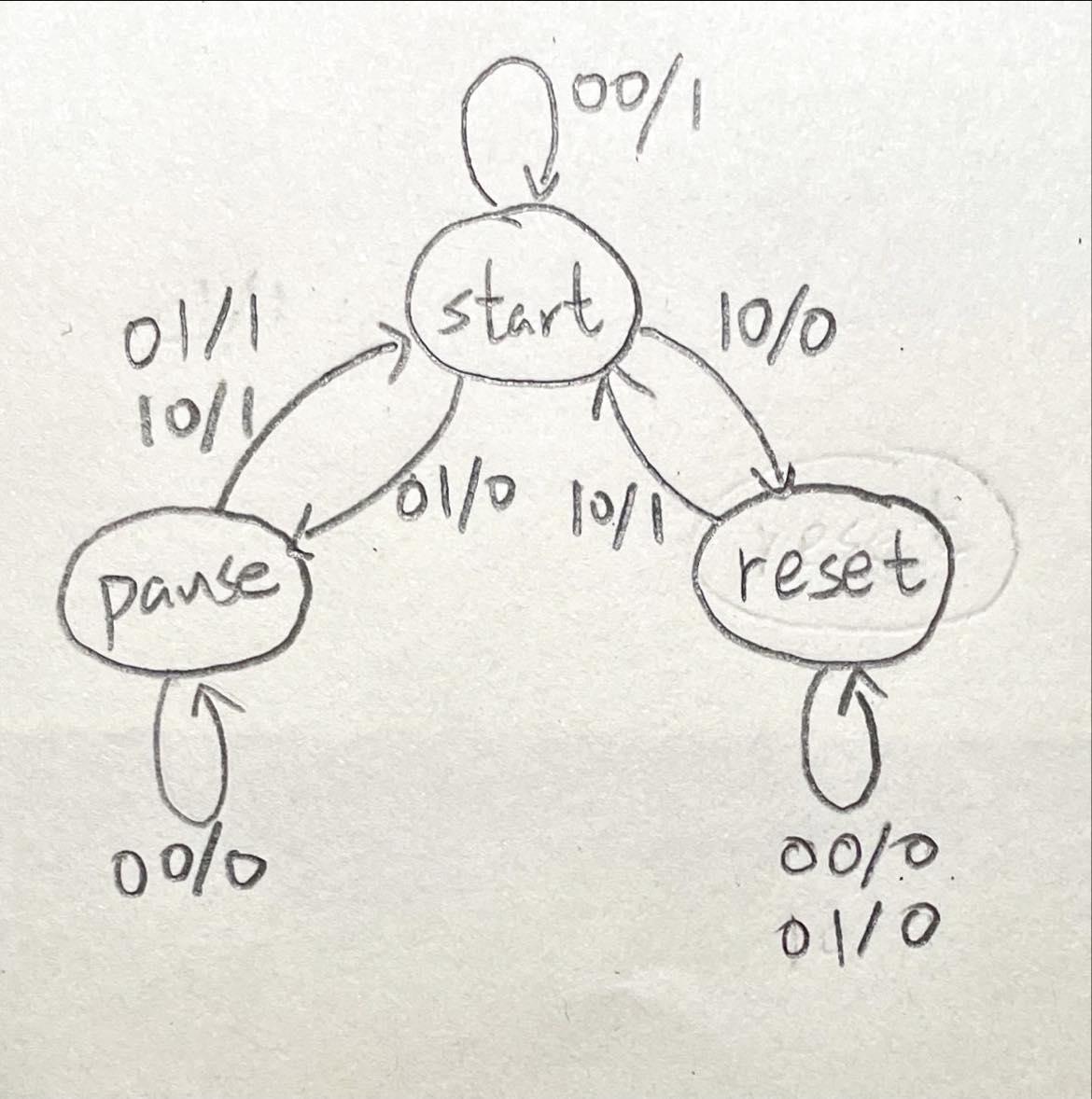


Then, use logic diagram to construct the code.



**FSM (Finite State Machine)**：(signal is {button1, button2})

State diagram:

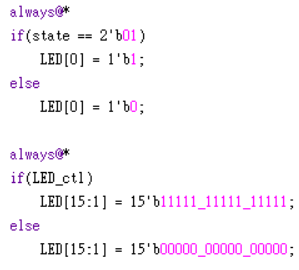
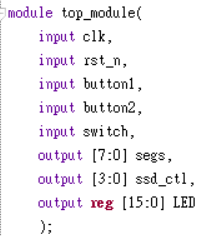


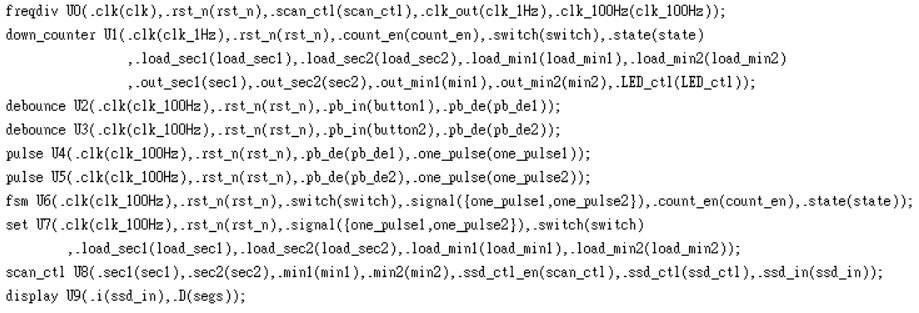
1. If

**Set**：The function of this part is set the initial value for down counter. I produce the signal of switch’s posedge (switch & ~switch\_delay) to reset the setting value (00:00). And use one button to set second (+1 when press once), use another button to set minute (+1 when press once). So I construct a up counter to complete this function, two buttons are considered count enable for second and minute counter.

**Display (the same as lab6\_1)**：is a decoder from BCD to cathode of seven-segment display.

**Top module**：to connect all modules and input, output the signal and use LED\_ctl from down counter and state from FSM to control 16 LEDs

****

****

**IO pin assignment：**

| LED[15] | LED[14] | LED[13] | LED[12] | LED[11] | LED[10] | LED[9] | LED[8] |
| --- | --- | --- | --- | --- | --- | --- | --- |
| L1 | P1 | N3 | P3 | U3 | W3 | V3 | V13 |

| LED[7] | LED[6] | LED[5] | LED[4] | LED[3] | LED[2] | LED[1] | LED[0] |
| --- | --- | --- | --- | --- | --- | --- | --- |
| V14 | U14 | U15 | W18 | V19 | U19 | E19 | U16 |

| segs[7] | segs[6] | segs[5] | segs[4] | segs[3] | segs[2] | segs[1] | segs[0] |
| --- | --- | --- | --- | --- | --- | --- | --- |
| W7 | W6 | U8 | V8 | U5 | V5 | U7 | V7 |

| ssd\_ctl[3] | ssd\_ctl[2] | ssd\_ctl[1] | ssd\_ctl[0] | rst\_n | clk | button1 | button2 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| W7 | W6 | U8 | V8 | V17 | W5 | W19 | T17 |

**Discussion**

這次實驗的第一題，我幾乎沒花多久時間就做出來了，結果第二題因為counter判斷式的問題，花了我三、四個小時在debug，而且我遇到了從沒見過的error message，所以我對問題出在哪完全沒有頭緒，只能一行一行慢慢找，在demo石助教有問我說一、二題做得那麼好，為什麼不做bonus？其實第二題我不是照題意一樣要使用四個按鈕來完成lab，我只使用了兩個按鈕，就是想連接bonus只需三個按鈕的條件，結果人算不如天算，我第二題花了太多時間，而且又遇到了期中考週，造成我不得不放棄bonus。

**Conclusion**

這次實驗當中我花了最久時間的就是畫出 block diagram、debug 以及做這份報告，由於實驗越趨複雜，block diagram 要愈連愈多，module 也愈來愈多個，各個 module 之間的線路與關係也愈來愈雜亂，而我雖然有先把每個小的功能都一個一個先確認過沒問題之後再連接訊號，但有時還是會出現 bug，像是FSM中我有單獨拉出來測試 count enable 和 reset 的輸出是沒問題的，但是接上下數器時卻不能正常的pause/resume 或是 stop/start，我花了很多時間思考這個問題，最後我換一個邏輯一樣但是寫法不同的方式試試看，功能就變正常了，因此 debug 的時間遠遠比預想的還要花得更久，甚至是比設計過程還要久的時間，再來則是由於功能越來越多，報告中要討論的東西也越來越多，因此寫報告的時間也跟著拉長了，之後的實驗只會愈來愈複雜，還是希望我可以盡快熟悉 verilog 的格式等問題，才不會讓像是上述 FSM 中的 bug 花費掉我太多的時間，因為只要一個小地方出錯，可能整個大的功能都會出問題，因此還是老話一句，希望我盡量不要在非必要的地方出錯，並且在以後的實驗也能夠更加的順利。