**Lab5 Report**



**Ⅰ. Lab5\_1 (use two buttons to control stopwatch from 25s )**

**Design Specification**

IO:

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

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

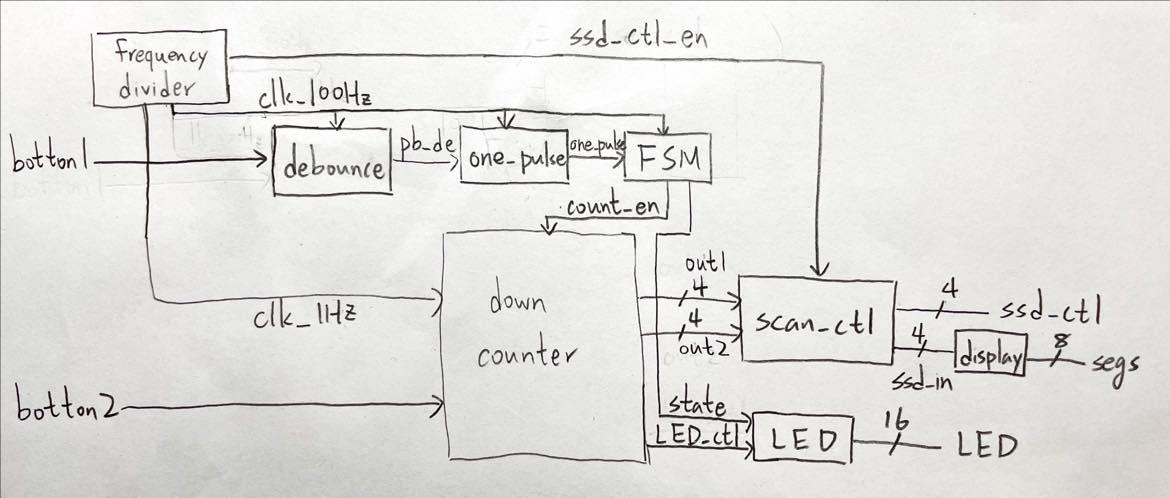
[7:0] segs(cathode of seven-segment display)

[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, display and LED) as follows.

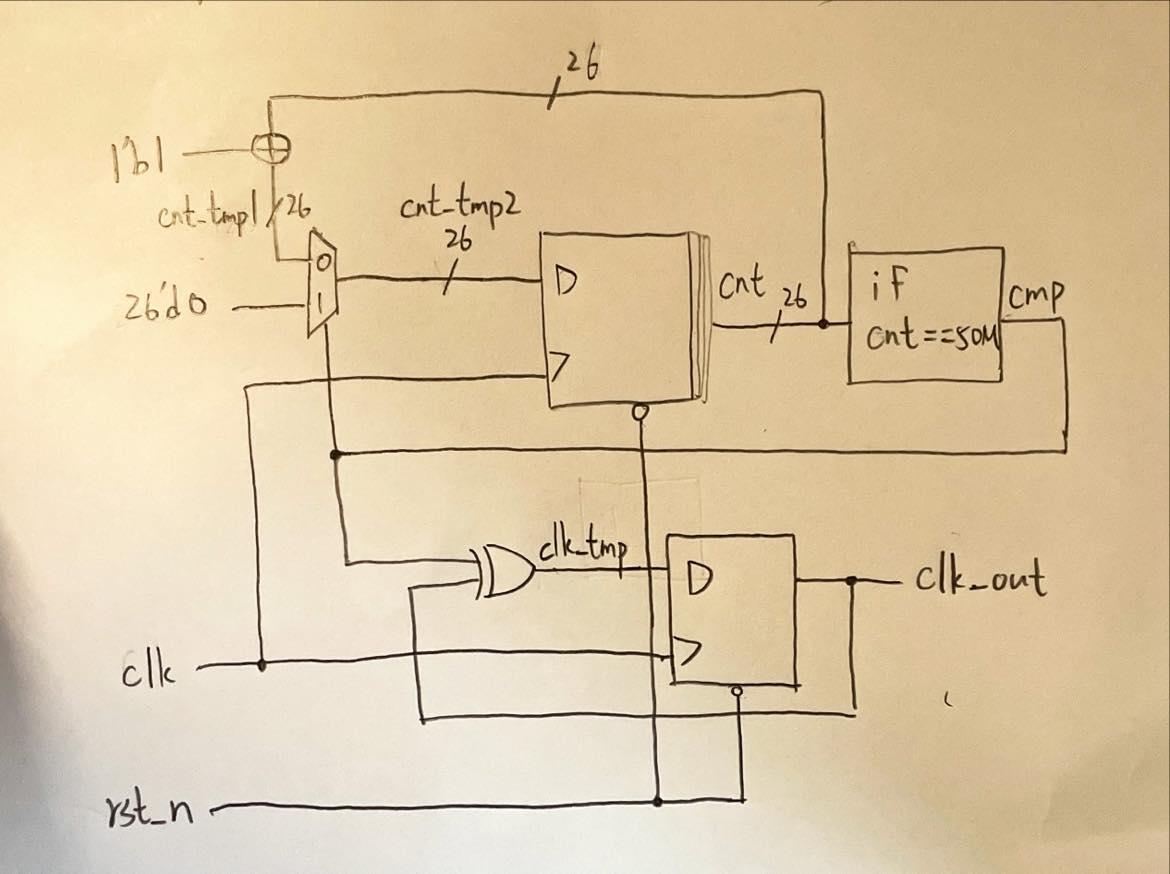
A simple Block diagram:



**Scan control**：Because of only two digits, just use an enable from 14th bit of frequency divider to control the seven-segment display. We can see different numbers on two seven-segment displays by anode and cathode rapidly changing before the clock toggle. Furthermore, left digit’s cathode is connected with the tens of down counter and right digit’s cathode is connected with the units of down counter.

**Frequency divider**：because I need two clocks to control down counter, debounce, one-pulse and FSM：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 and 100Hz clock together.

1Hz clock：



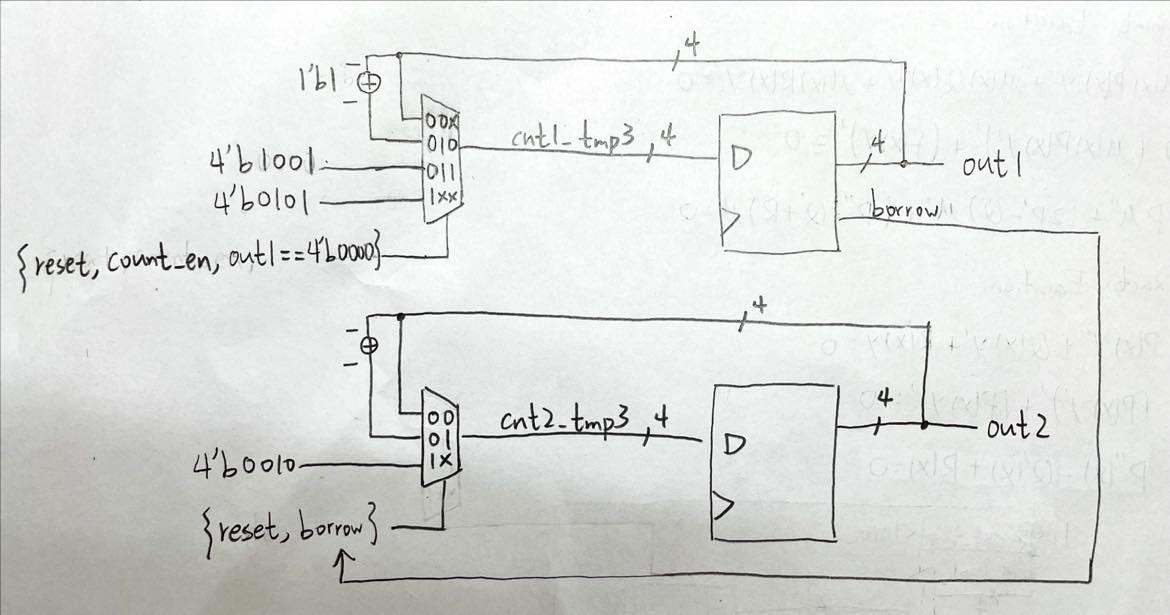
So to construct 1Hz clock, I use a counter to count until 50M, then declare a variable cmp.

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 ( ).

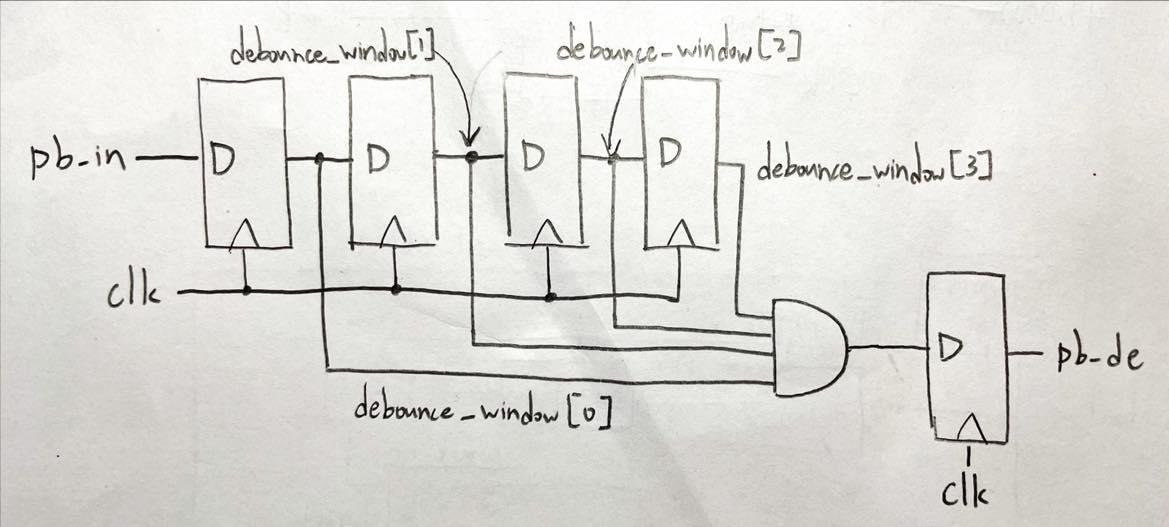
100Hz clock and scan enable：

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 14th bit, which is used as an enable for scan control.

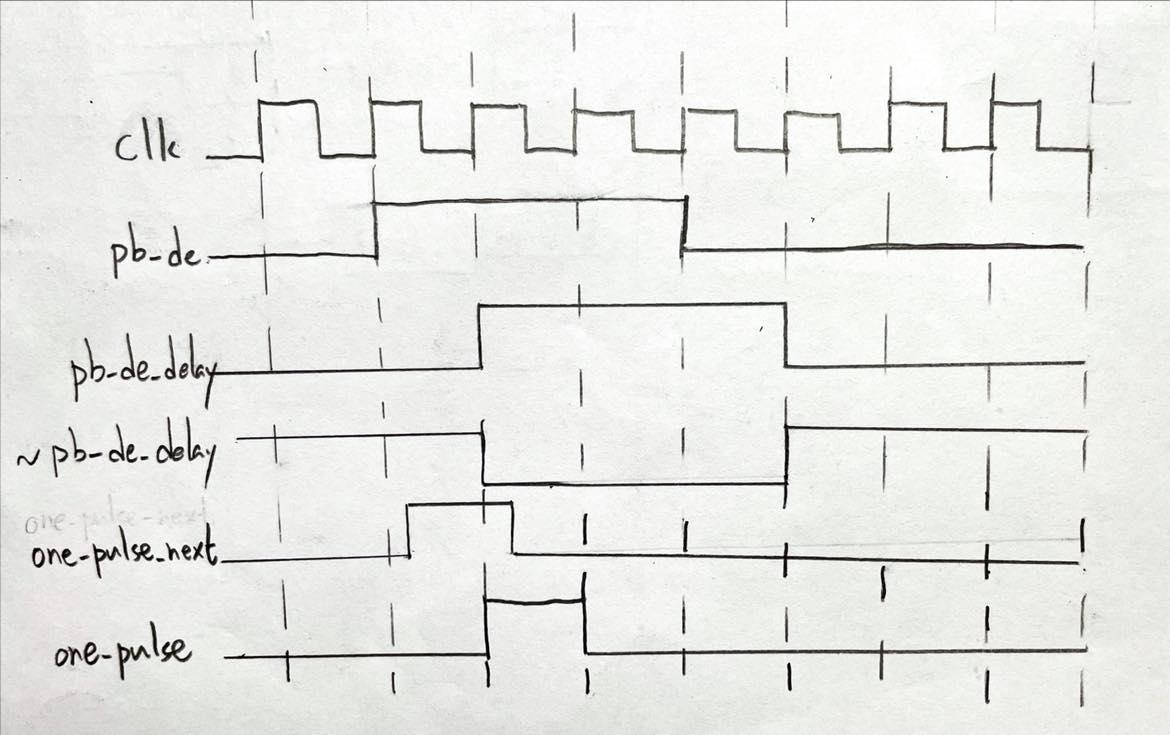
**Down counter**：I not only use following logic diagram to construct down counter but also add some conditions. For example, if out1 and out2 = 0, then keep out1 and out2 = 0; if out1 and out2 = 0, then LED\_ctl = 1 to turn on all LED.



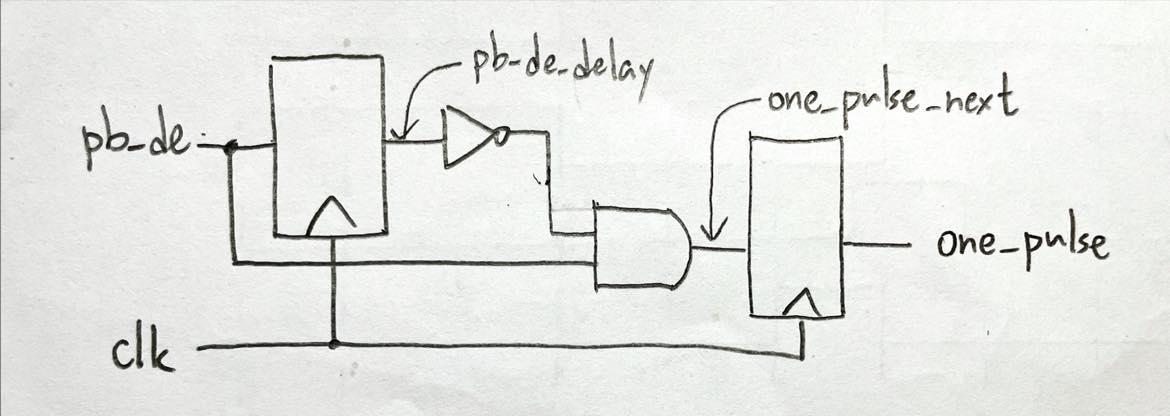
**Debounce**：Because of physical effect, when pressing the button, the spring in th e button will produce the signal of bounce. Therefore, we need to remove the bounce which is called debounce. And the size of bounces is about µs, so we use 100Hz clock and four flip-flops to detect the signal. If four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



**One-pulse**：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.

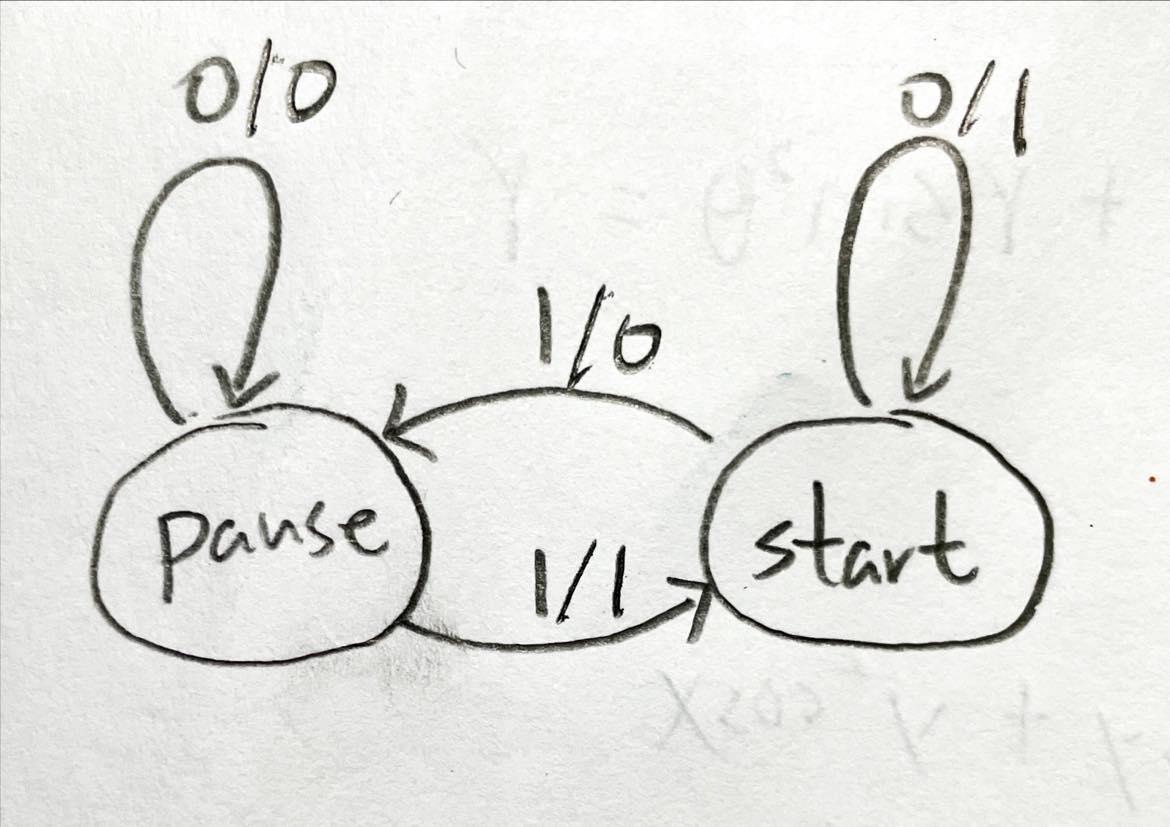


Then, use logic diagram to construct the code.



**FSM (Finite State Machine)**：

State diagram: (mealy machine)



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.

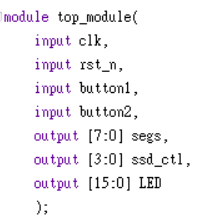
1. If state = 1 (start) and signal is 1,

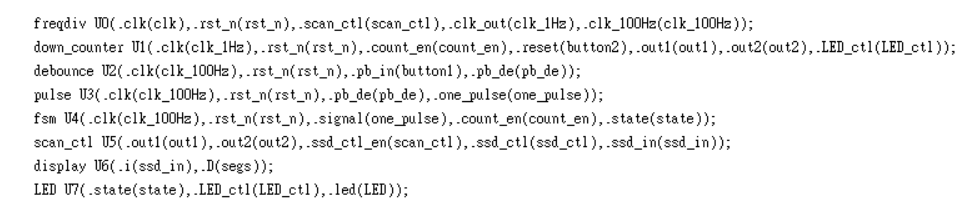
then jump to next state (state = 0) (pause) and output (count enable) = 0.

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

**LED**：use LED\_ctl from down counter and state from FSM to control 16 LEDs.

**Top module**：to connect all modules and input, output the signal





**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 |

**Ⅱ. Lab5\_2 (use one buttons to control stopwatch from 25s )**

**Design Specification**

IO:

Input: clk, rst\_n, button.

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

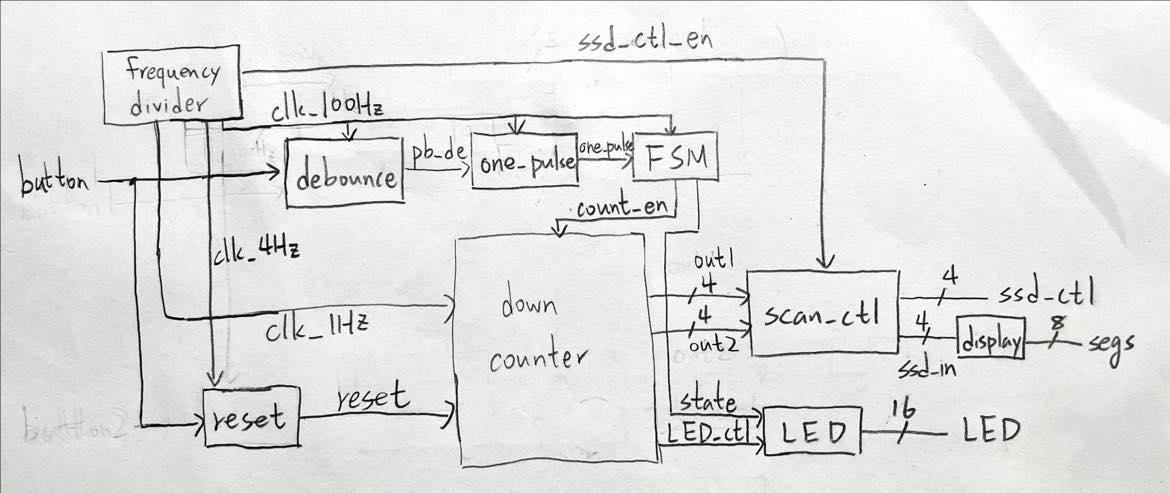
[7:0] segs(cathode of seven-segment display)

[15:0] LED.

**Design Implementation**

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

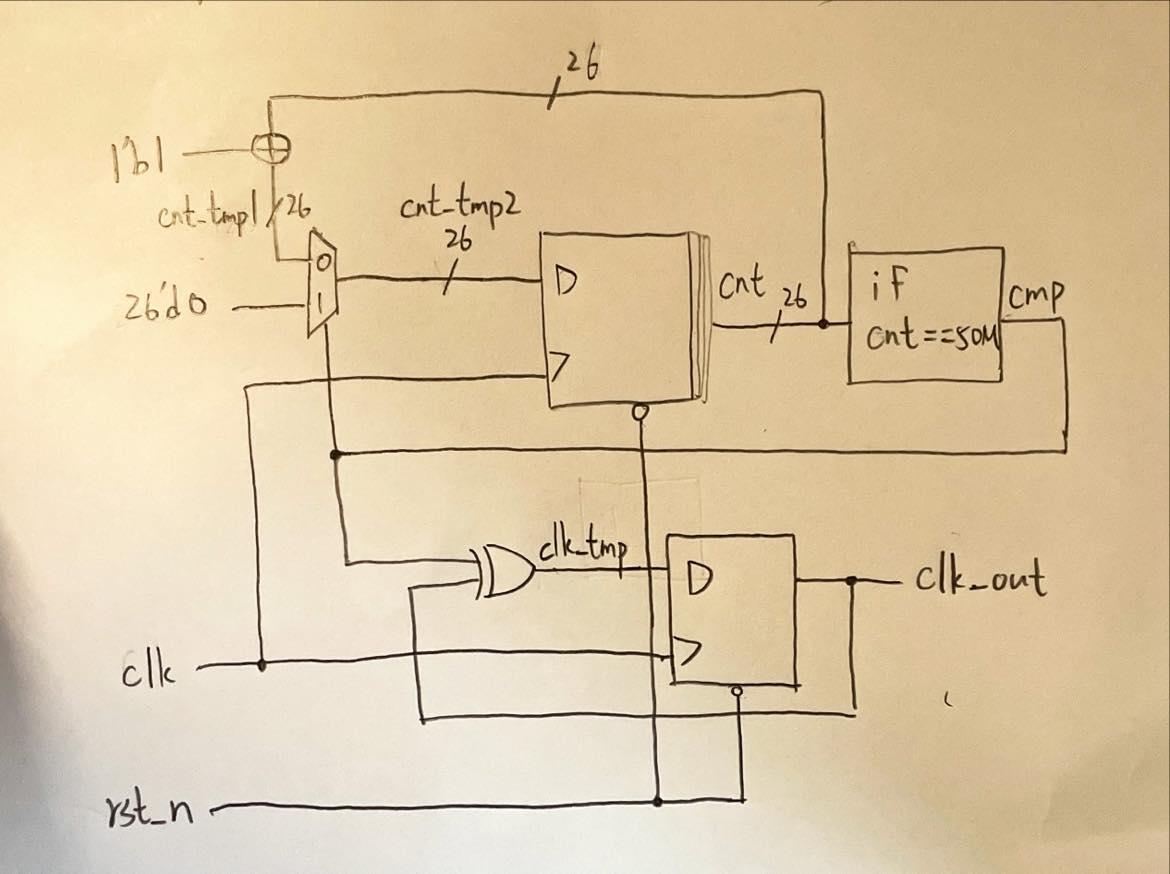
A simple Block diagram:

****

**Scan control (the same as lab5\_1)**：Because of only two digits, just use an enable from 14th bit of frequency divider to control the seven-segment display. We can see different numbers on two seven-segment displays by anode and cathode rapidly changing before the clock toggle. Furthermore, left digit’s cathode is connected with the tens of down counter and right digit’s cathode is connected with the units of down counter.

**Frequency divider**：Because I need three clocks to control down counter, debounce, reset, one-pulse and FSM：1Hz clock controls down counter, 4Hz clock controls reset and 100Hz clock controls the others. And I also need an enable to control scan.v, so I produce this enable, 4Hz and 100Hz clock simultaneously.

1Hz clock：



(the same as lab5\_1)

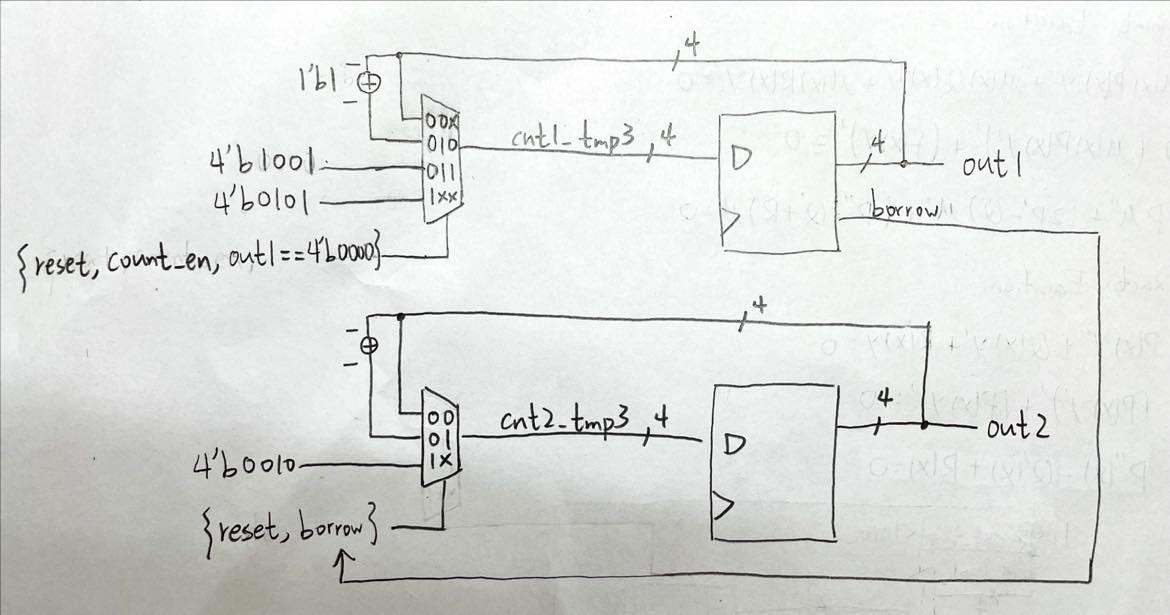
To construct 1Hz clock, I use a counter to count until 50M, then declare a variable cmp.

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 ( ).

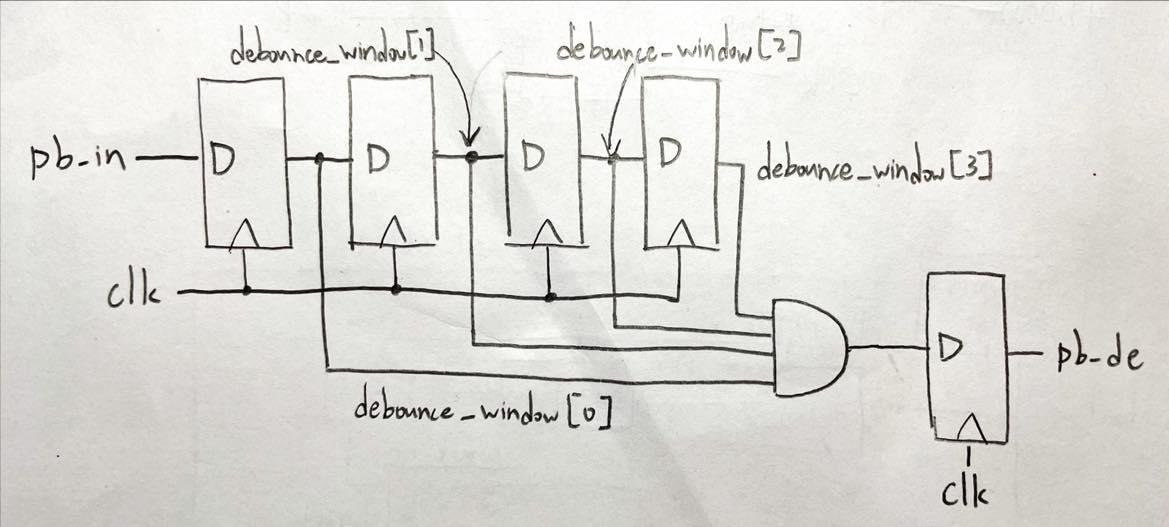
100Hz clock, 4Hz clock and scan enable：

use 25-bit-binary-up counter with initial clock to count and the highest bit is considered new clock (4Hz), which is (≒4Hz); 21th bit is considered new clock (100Hz), which is (≒100Hz). And output 14th bit, which is used as an enable for scan control.

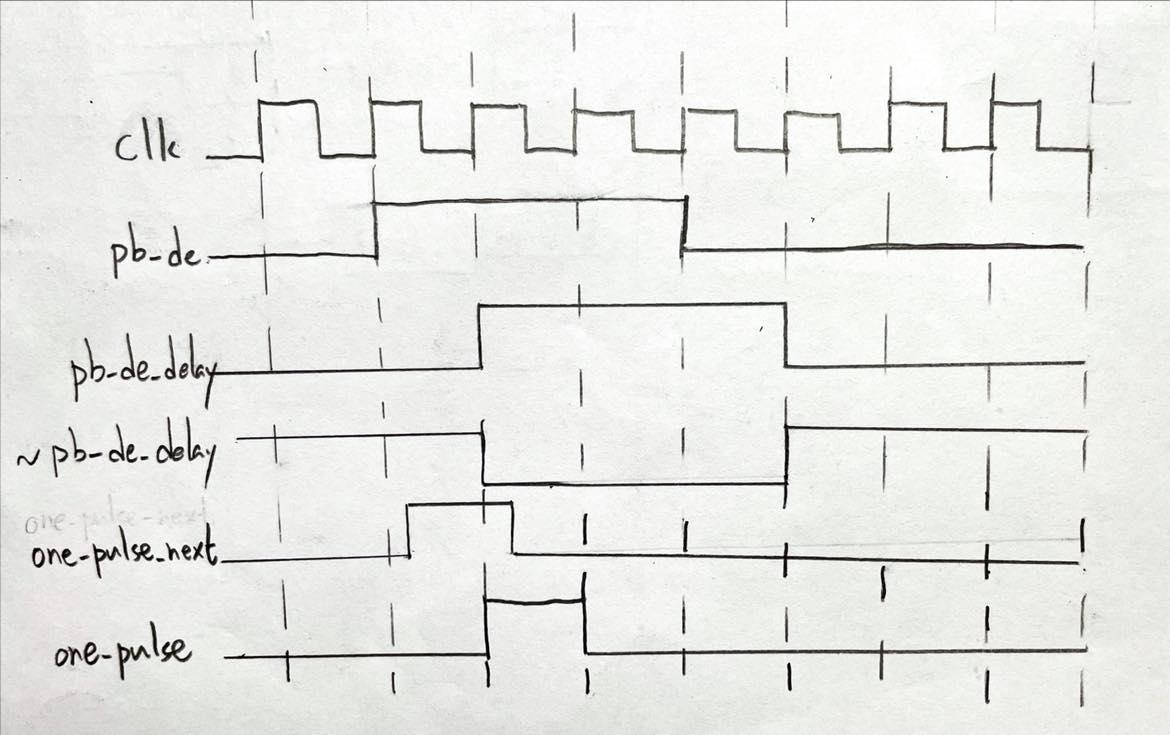
**Down counter (the same as lab5\_1)**：I not only use following logic diagram to construct down counter but also add some conditions. For example, if out1 and out2 = 0, then keep out1 and out2 = 0; if out1 and out2 = 0, then LED\_ctl = 1 to turn on all LED.



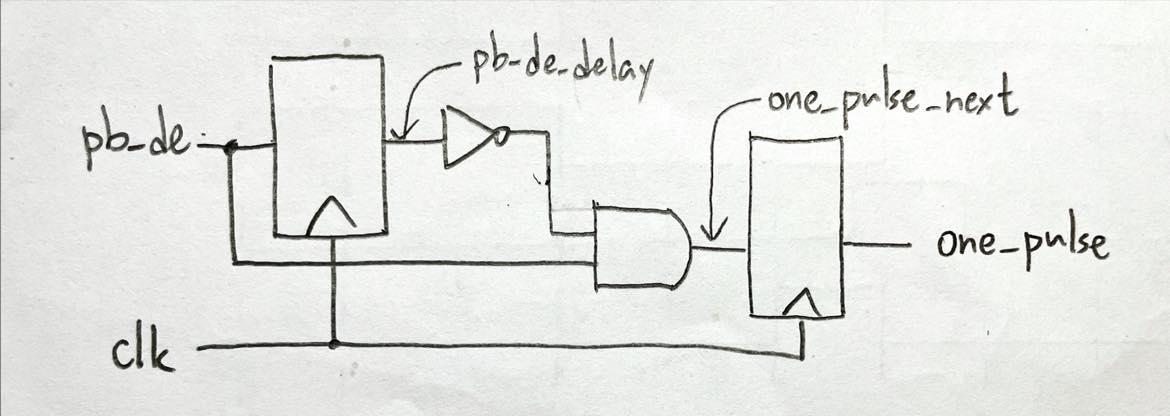
**Debounce (the same as lab5\_1)**：Because of physical effect, when pressing the button, the spring in the button will produce the signal of bounce. Therefore, we need to remove the bounce which is called debounce. And the size of bounces is about µs, so we use 100Hz clock and four flip-flops to detect 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 lab5\_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.

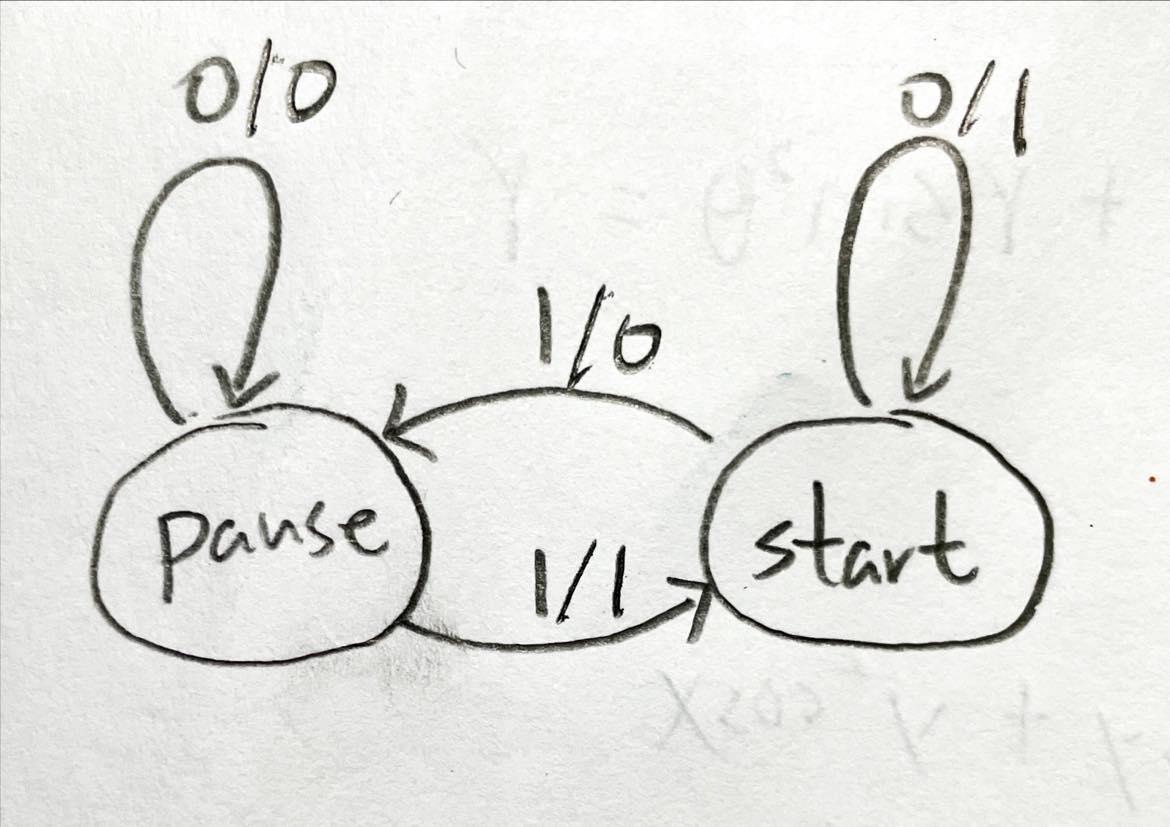


Then, use logic diagram to construct the code.



**FSM (Finite State Machine) (the same as lab5\_1)**：

State diagram: (mealy machine)



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.

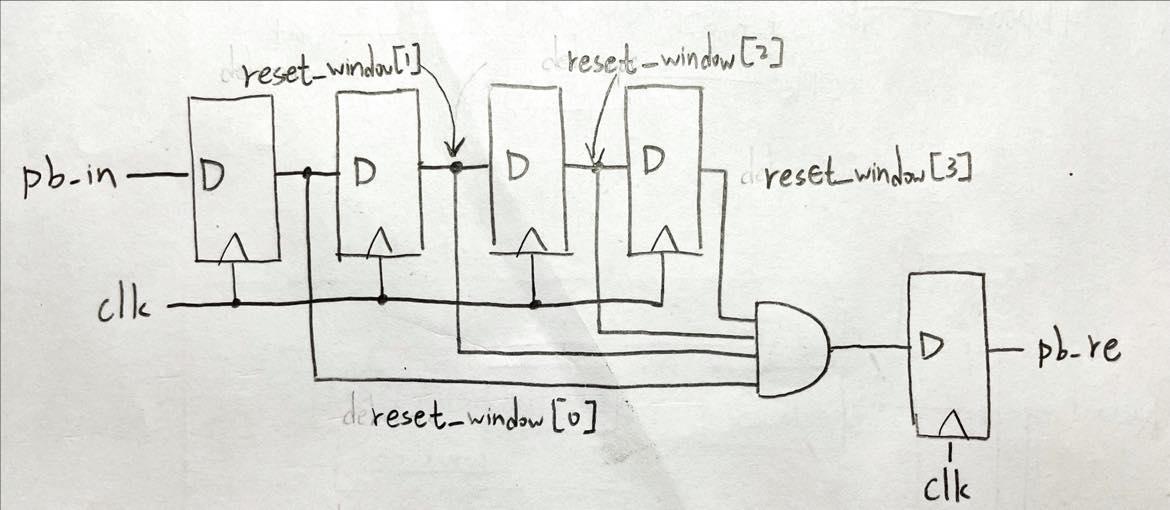
1. If state = 1 (start) and signal is 0,

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.

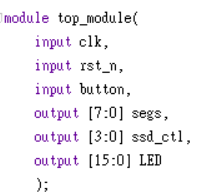
**Reset**：this part is similar with debounce, I control the clock of four flip-flops to make signal go through four flip-flops needed at least .

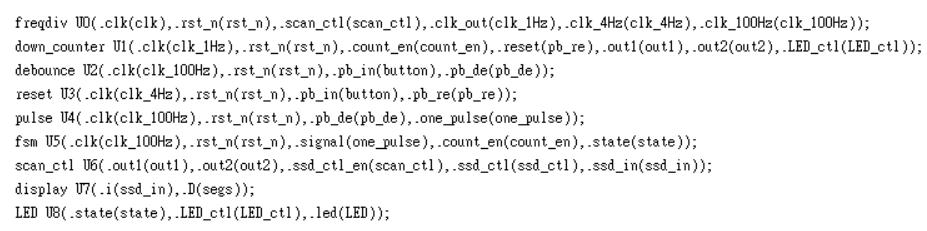


**Display (the same as lab5\_1)**：is a decoder from BCD to cathode of seven-segment display.

**LED**：use LED\_ctl from down counter and state from FSM to control 16 LEDs.

**Top module**：to connect all modules and input, output the signal

****

****

**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 | button |
| --- | --- | --- | --- | --- | --- | --- |
| W7 | W6 | U8 | V8 | V17 | W5 | W19 |

**Ⅲ. Lab5\_bonus (stopwatch with two modes：1min 30sec and 25sec)**

**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, display and LED) as follows.

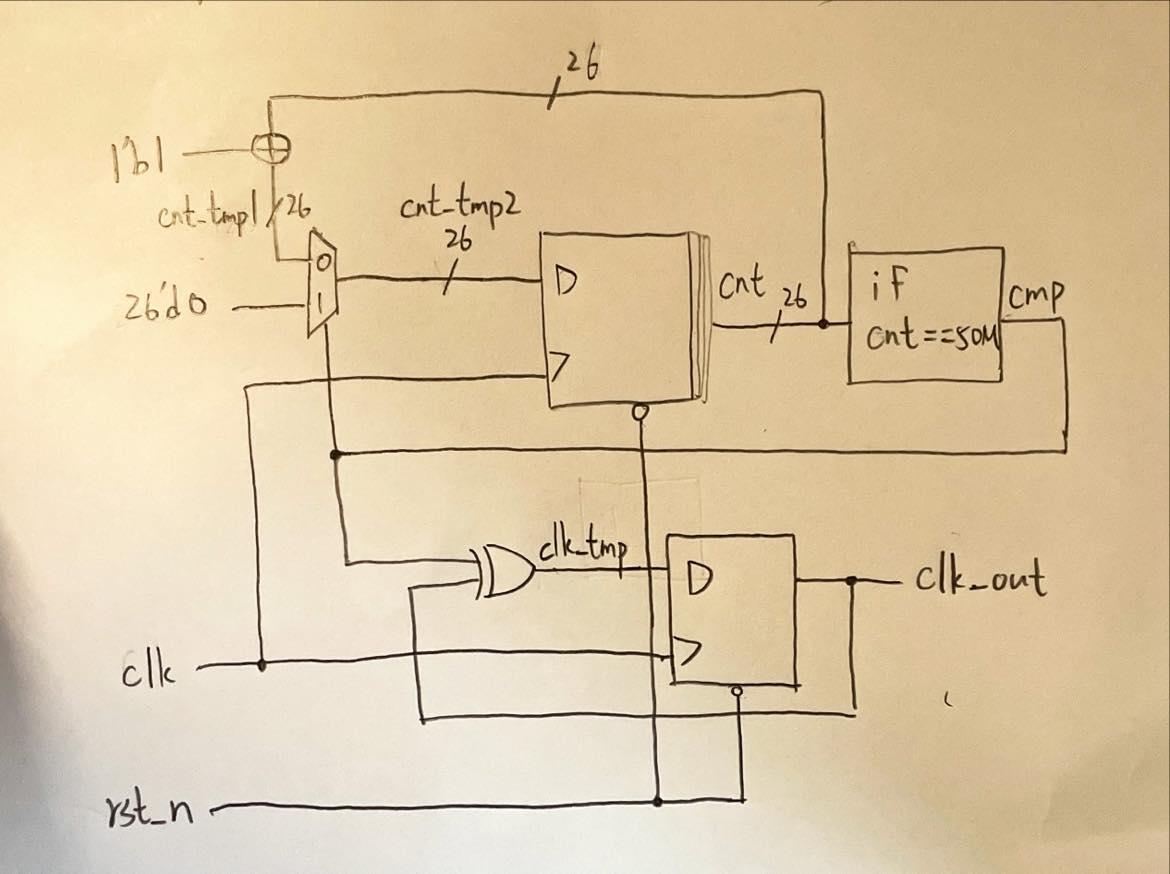
A simple Block diagram:



**Scan control**：Because of three digits, just use an enable from 13th and 14th bits of frequency divider to control the seven-segment display. We can see different numbers on three seven-segment displays by anode and cathode rapidly changing before the clock toggle. Furthermore, left digit’s cathode is connected with the unit minute of down counter, middle digit’s cathode is connected with the tens second of down counter and right digit’s cathode is connected with the unit second of down counter.

**Frequency divider**：because I need four clocks to control down counter, debounce, one-pulse, reset and FSM：1Hz clock controls down counter, 16Hz clock controls button1’s reset, 64Hz controls button2’s reset and 100Hz clock controls the others. And I also need an enable to control scan.v, so I produce this enable other clocks without 1Hz simultaneously.

1Hz clock：

#

(the same as lab5\_1)

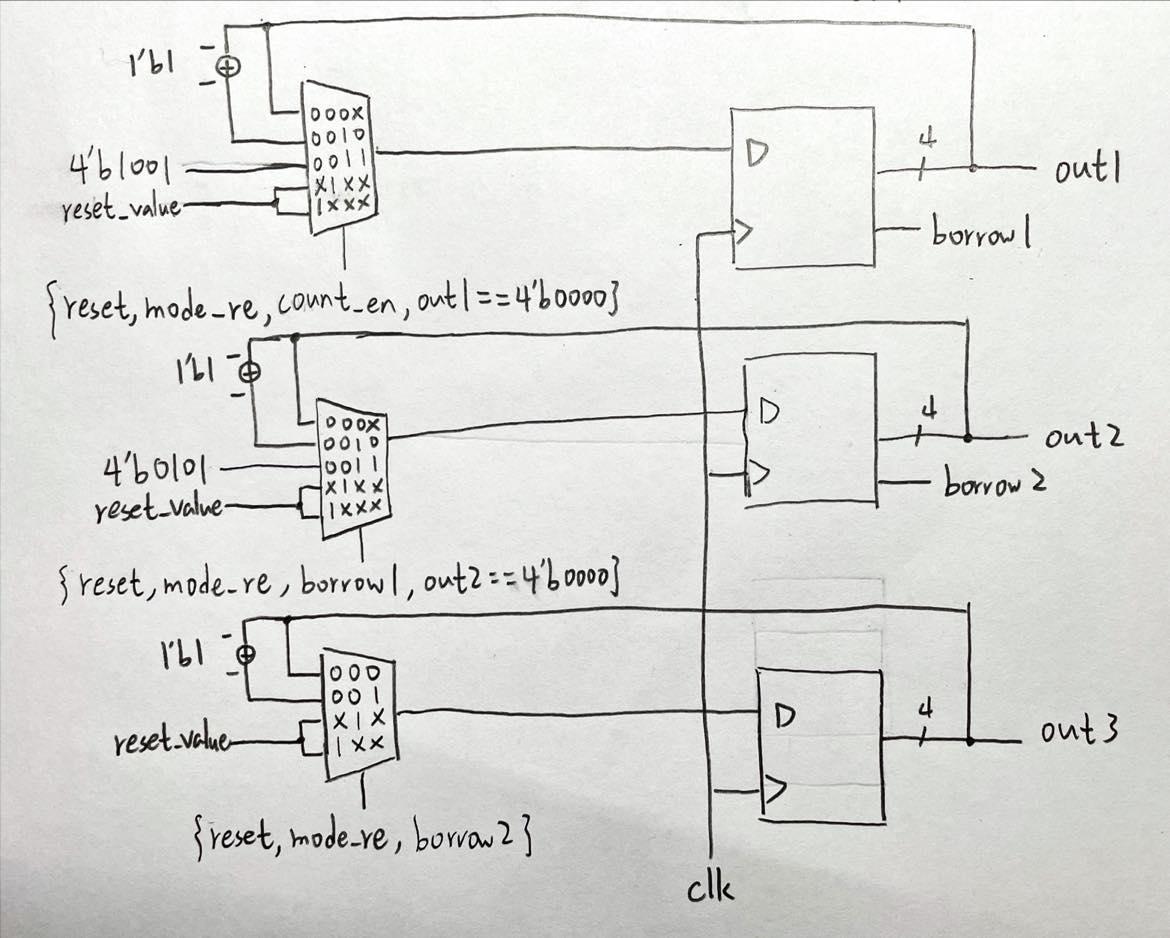
So to construct 1Hz clock, I use a counter to count until 50M, then declare a variable cmp.

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 ( ).

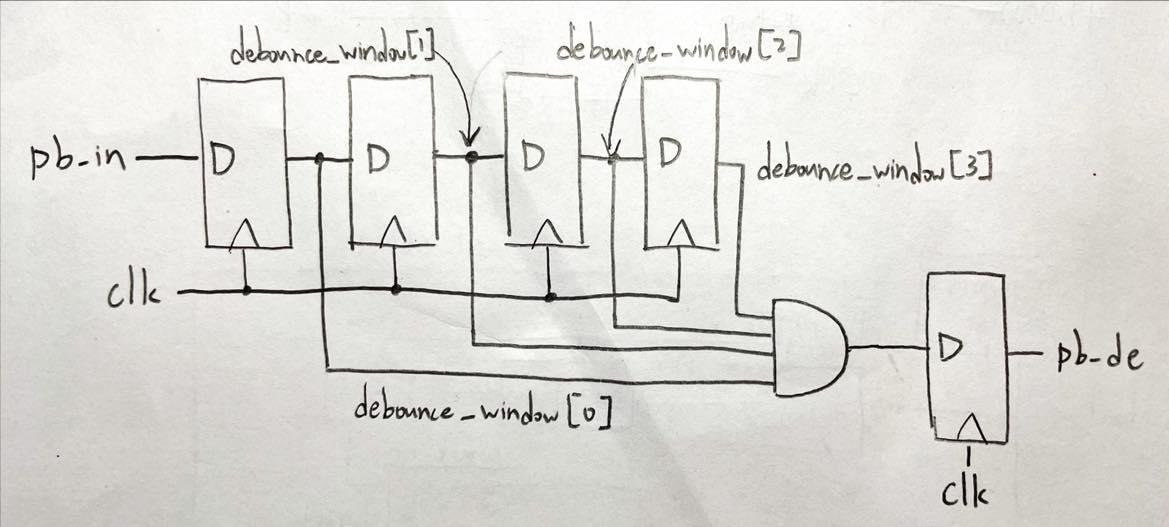
100Hz, 16Hz, 64Hz clock and scan enable：

use 23-bit-binary-up counter with initial clock to count and the highest bit is considered new clock (16Hz), which is (≒16Hz); 21th bit is considered new clock (64Hz), which is (≒64Hz); 20th bit is considered new clock (100Hz), which is (≒100Hz). And output 13th and 14th bits, which are used as an enable for scan control.

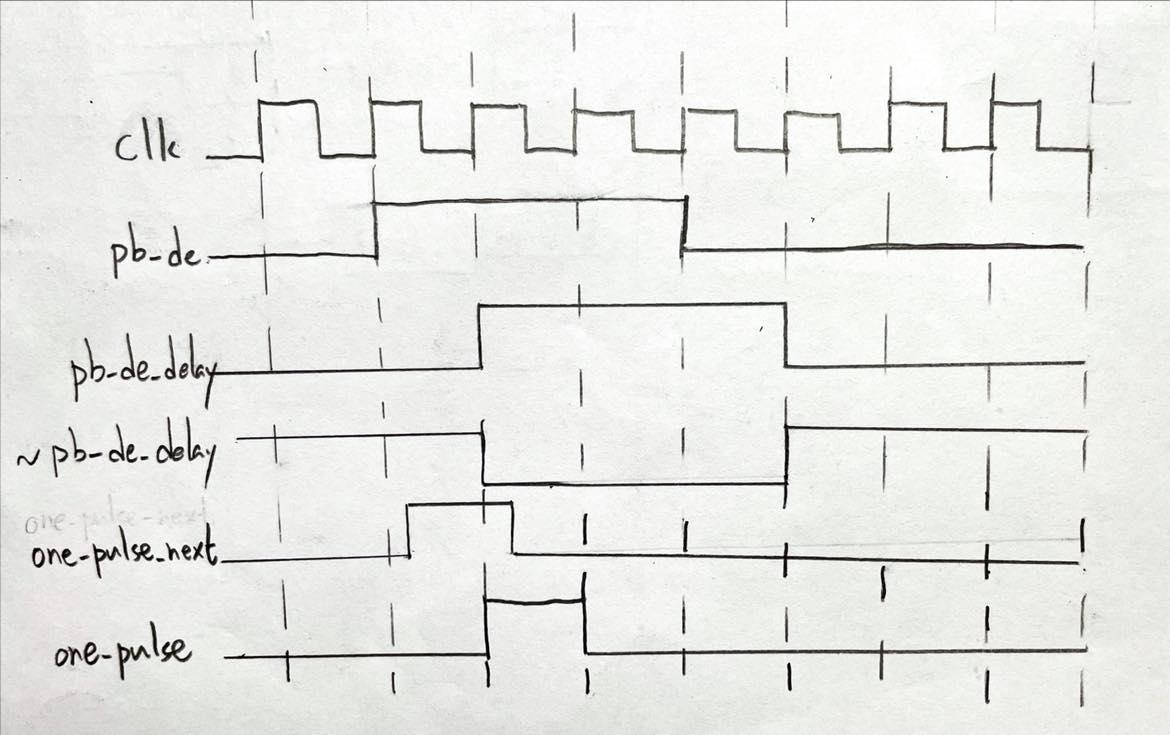
**Down counter**：I not only use following logic diagram to construct down counter but also add some conditions. For example, if out1, out2 and out3 = 0, then keep out1, out2 and out3 = 0; if out1, out2 and out3 = 0, then LED\_ctl = 1 to turn on all LED.



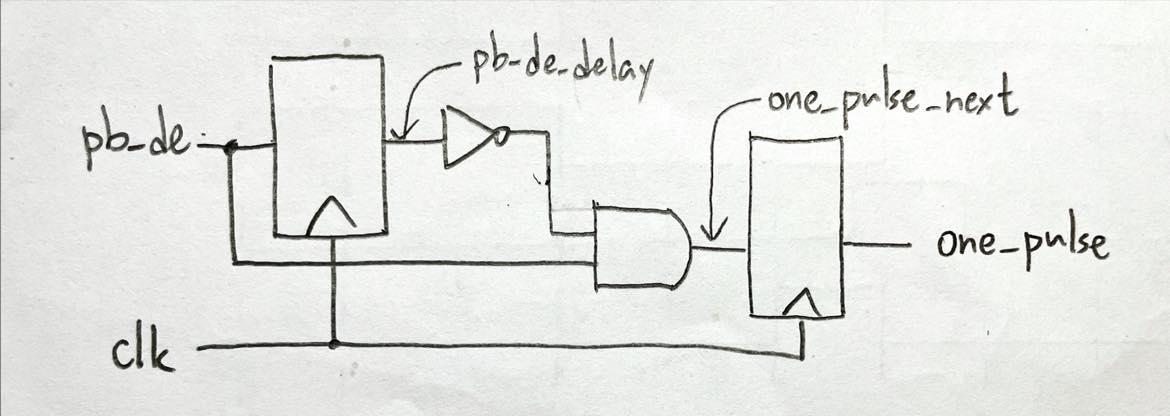
**Debounce (the same as lab5\_1)**：Because of physical effect, when pressing the button, the spring in the button will produce the signal of bounce. Therefore, we need to remove the bounce which is called debounce. And the size of bounces is about µs, so we use 100Hz clock and four flip-flops to detect 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 lab5\_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.

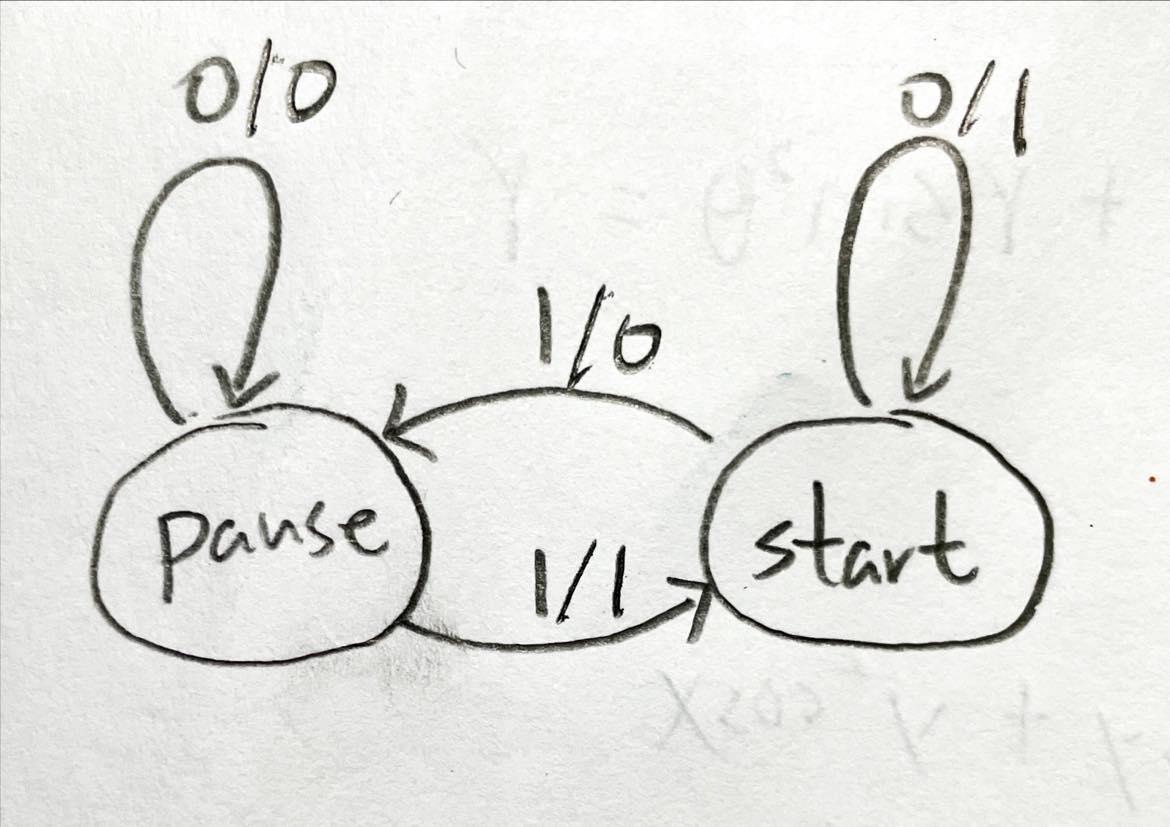
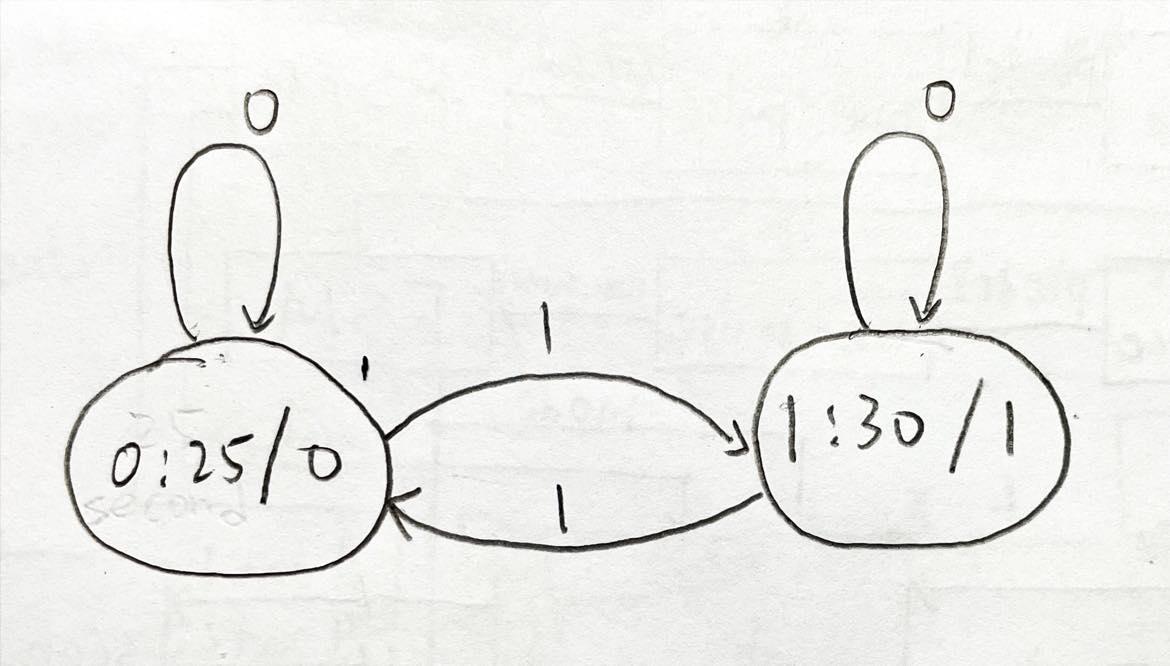


**FSM (Finite State Machine)**：because I didn’t want to construct a new FSM, I use two FSMs before to represent two modes and two states.

button1 controls state and button2 controls mode.

State diagram:

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

State:

1. If state = 0 (pause) and button1’s signal is 0,

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

1. If state = 0 (pause) and button1’s signal is 1,

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

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

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

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

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

Mode:

1. If state = 0 (0:25) then output (mode) = 0

and if button2’s signal is 0 then keep the state = 0 (0:25).

1. If state = 0 (0:25) then output (mode) = 0

and if button2’s signal is 1 then jump to next state = 1 (1:30).

1. If state = 1 (1:30) then output (mode) = 1

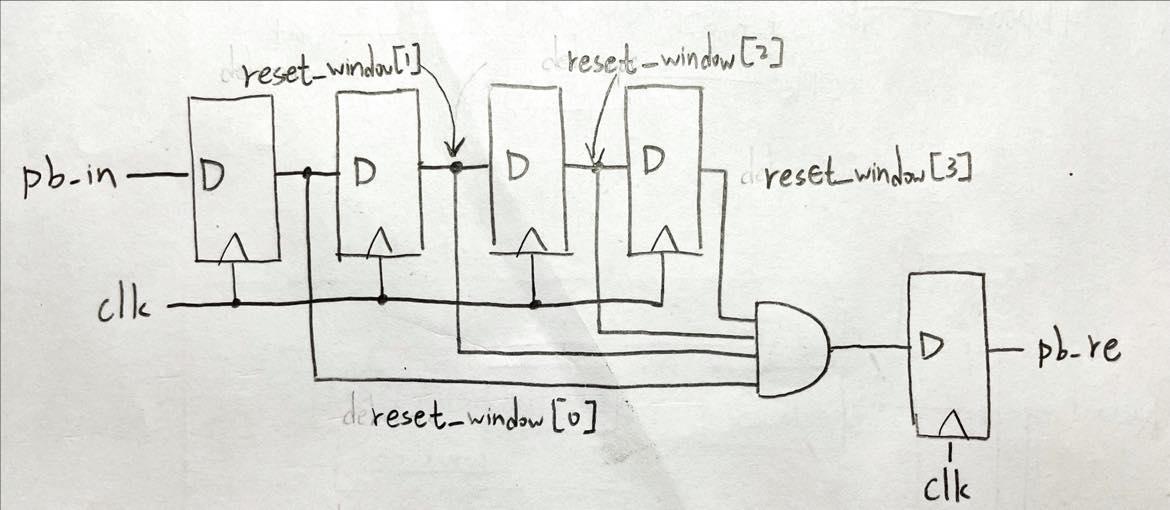
and if button2’s signal is 0 then keep the state = 1 (1:30).

1. If state = 1 (1:30) then output (mode) = 1

and if button2’s signal is 1 then jump to next state = 0 (0:25).

**Reset**：this part is similar with debounce, I control the clock of four flip-flops to make signal go through four flip-flops needed longer time. I want to reset the counter when I change mode and I don’t expect that the time of pressing button2 is too long , so I use 64Hz clock to control four flip-flops. For common reset, I expect that the time of pressing button1 is longer than mode changing reset, so I use 16Hz clock to control four flip-flops.

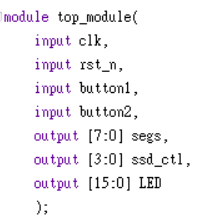
reset logic diagram:

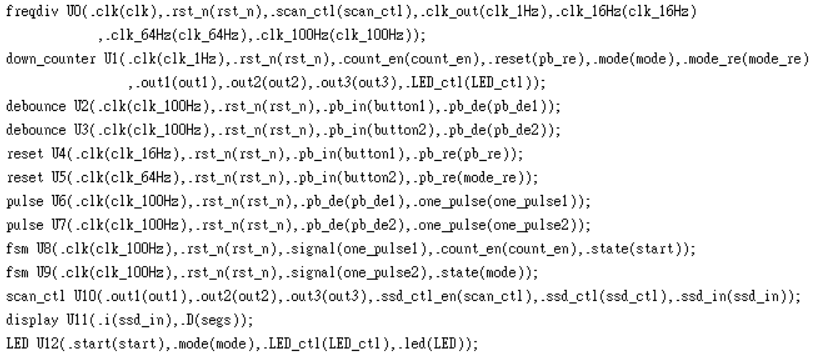


**Display**：is a decoder from BCD to cathode of seven-segment display.

**LED**：use LED\_ctl from down counter, state and mode from FSM to control 16 LEDs.

**Top module**：to connect all modules and input, output the signal





**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**

在第二題與第三題需要長按按鈕的題目，我沒使用教授上課教的做法(用counter去數秒數)，而是利用改變clock去控制訊號通過的時間，進而實現長按的功能，這個方法重新利用了debounce的code讓我整體實作的時間縮短了一些。而在實作時我也發現，我常在一些判斷式的地方粗心，像是reset = 0時應該是跑正常的功能，但我卻因為判斷的地方寫錯，而造成我把整個counter都reset了，之後我會更注意一些小細節，才不會之後在debug時花費太多時間。

**Conclusion**

這次實驗其實把第一題做完，剩下的兩題也差不多結束了，例如第二題只是在第一題增加一個長按的功能，剩下的只要複製第一題的module就完成了。但就算是這樣，這次實驗所耗費的時間也沒比較短，因為我第一次接觸到FSM在verilog上的實作，而且也是第一次接觸到FPGA板上的按鈕，按鈕因為一些物理性質而會造成一些雜訊，我覺得我們處理的方式真的很神奇，我也花了一段時間去理解，總而言之，我在這次實驗學到很多關於FSM的實體運用。