Lab5: Push Buttons

107061112 王昊文

Experiments

1 Construct a 30-second down counter 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.

Design Specification

✓ I/O

Input clk // global clock input

Input rst_but // reset push button

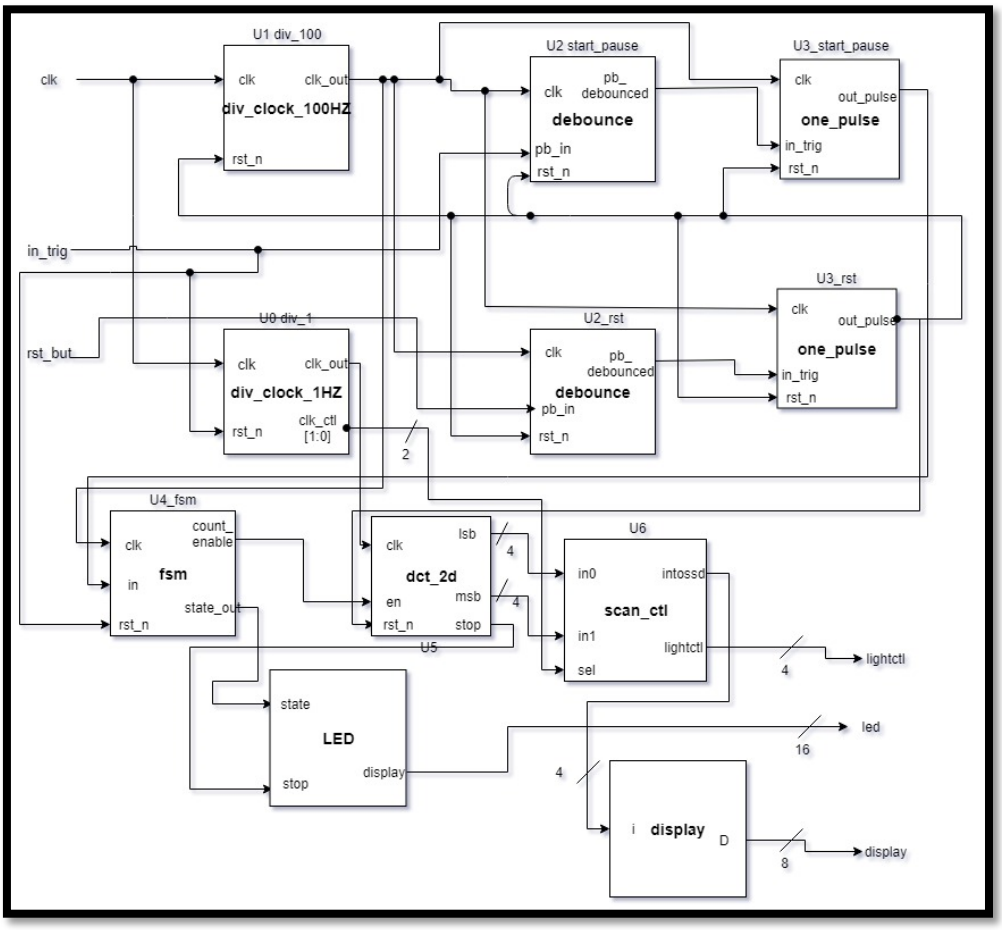
Input in_trig // push button for start / pause

Output [15:0]led // led output

Output [7:0]display // number display ssd

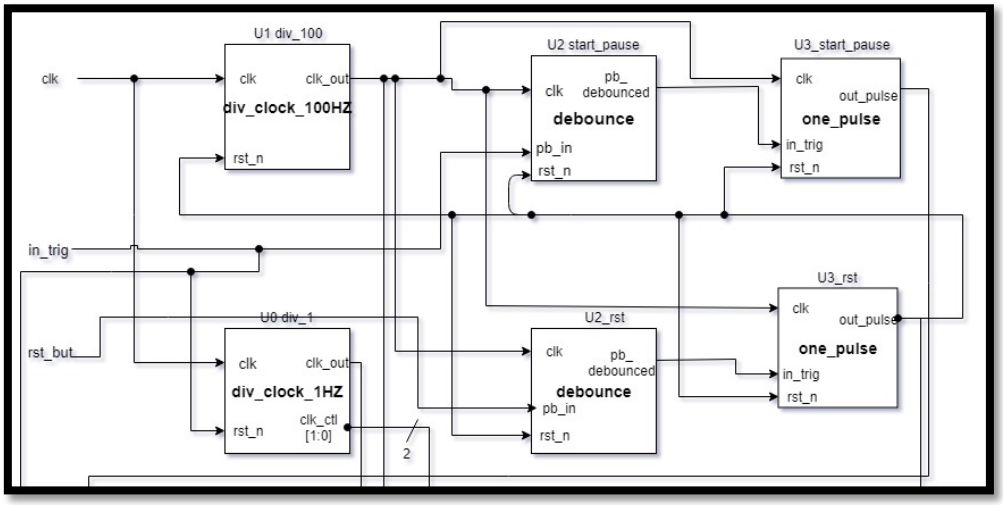
Output [3:0]lightctl // for which light to light

✓ Logic Block:

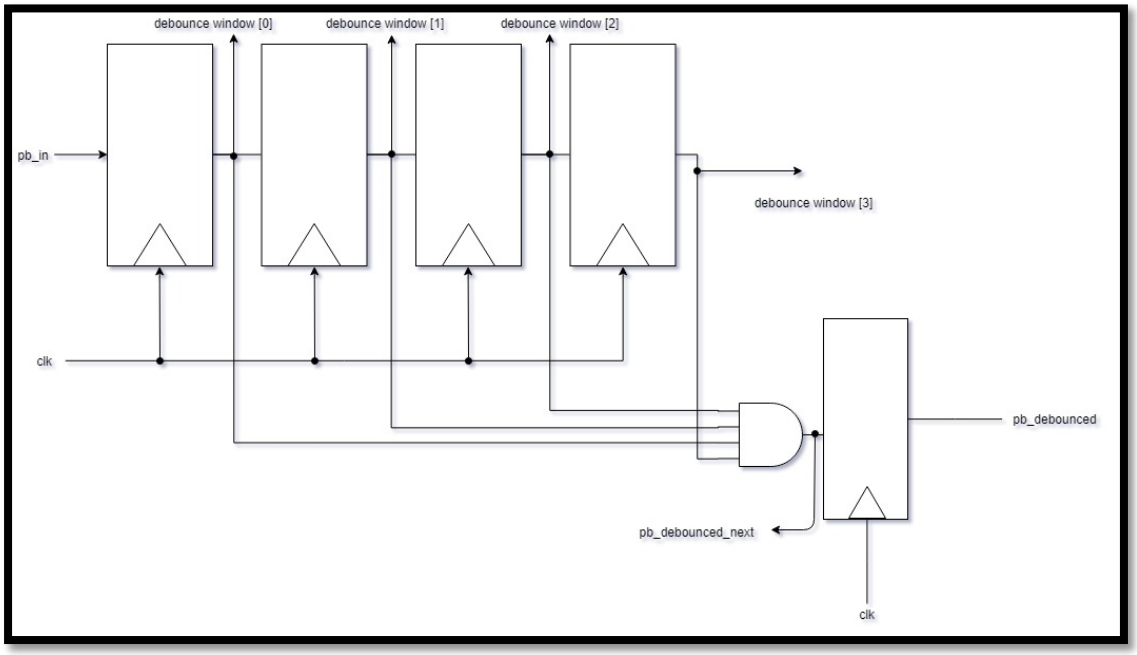


由於這次的lab使用到了push button，在最前面輸入的部分必須經過debounce以及one pulse的處理。In_trig為切換模式push button的input，rst_but為reset push buttton的input。以巨觀角度來看，前半部為訊號處理，後面會通過fsm，做state 轉換，透過fsm的輸出來控制counter的計數。最後面的scan與display與前面的lab相同，想要他同時顯示不同的東西。

接下來詳細說明，首先focus前半部input訊號處理的部分。
前半部：



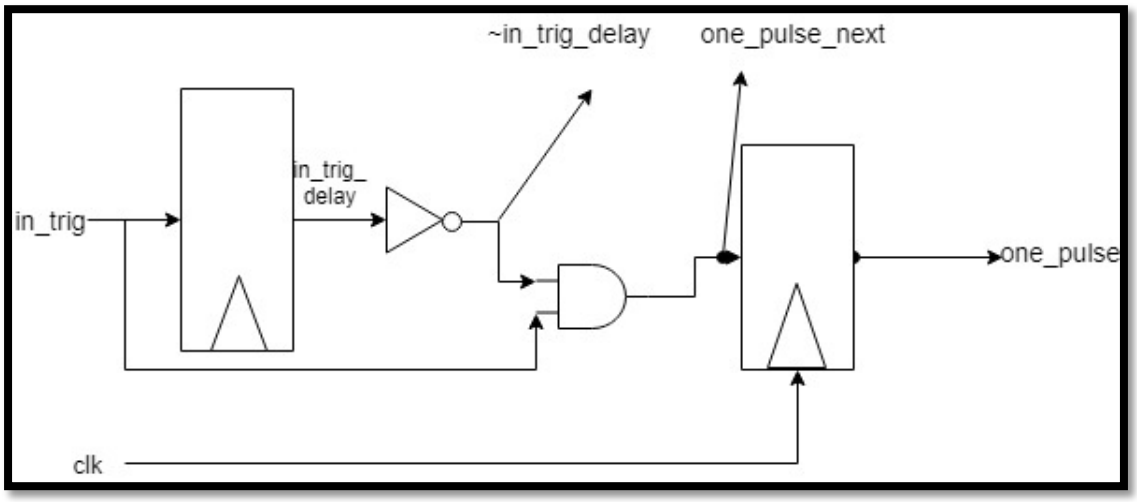
前面設置兩個clock，一個1hz，一個100hz。1hz主要控制counter，100hz則為訊號處理使用。所以後面的debounce以及one pulse，到fsm都是使用100hz的clock。
Debounce結構圖



Debounce在做的是處理訊號不平整的地方。Pb_in為訊號輸入，要處理訊號雜

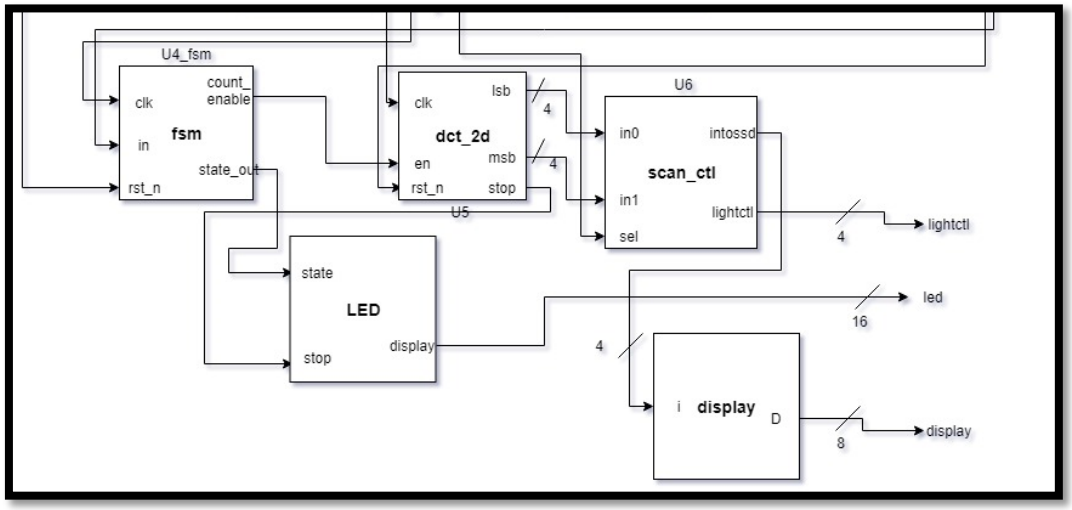
訊，在input送進來的時候檢查debounce window訊號是否一致，如果不一致視為有雜訊，則通過and gate處理，在這裡我們是將之全部處理為0。舉例來說，如果debounce window值為1100，代表訊號並未穩定，通過and gate會把訊號調整成0。最後再使用一個register使得訊號更為穩定。

One_pulse結構圖:

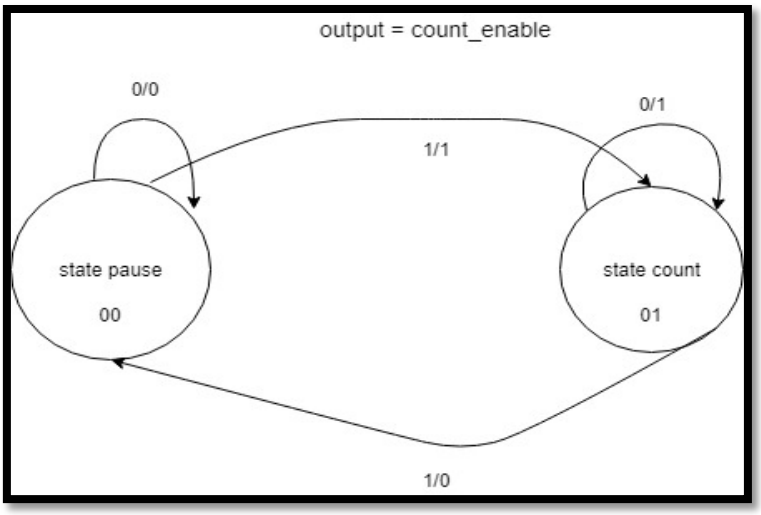


在one pulse的部分，透過一個register使得in_trig與~in_trig產生一個clock cycle的誤差，透過一個and gate即可產生one pulse。最後面再使用一個register來使訊號更穩定。
前面訊號處理要處理start / pause的push button以及reset button的。

再來看中間部分。



Fsm state diagram



想法很簡單，基本上只要按下按鈕就可以跳到下一個state，沒有輸入就維持原本的state。輸出為count_enable，作為counter的enable。在fsm輸出的部分還有state，這個輸出會連接傲led，告訴目前為何種state。在state count的時候led燈會亮起，state pause會暗掉，以達成題目的需求。

後面down counter以及顯示沿用之前的lab模組，

Design Implementation

Input		
clk	*in_trig*	*rst_but*
W5	T17	W19

Output ssd							
display[0]	***display[1]***	***display[2]***	***display[3]***	***display[4]***	***display[5]***	***display[6]***	***display[7]***
V7	U7	V5	U5	V8	U8	W6	W7

Output lightctl			
lightctl[0]	*lightctl[1]*	*lightctl[2]*	*lightctl[3]*
U2	U4	V4	W4

Discussion

雖然預報中有類似的題目，但是預報中並不需要做訊號input的處理。這一題處理起來複雜很多。這一題中我花了很多時間去理解one pulse 以及debounced的運作原理。以前沒想過這麼多，原來僅僅從dip switch改成button必須經過這麼多的步驟。在第一次製作的時候非常容易接錯clock，後來觀念清楚之後才發現原來後面的debounce one pulse都必須接上頻率高的clock。然後finite state machine的應用也打開了我的眼界。以前學fsm其實並不是很有感覺到底是拿來做什麼的，應用在這邊就相當的清楚，大大的打開了我對於邏輯

設計的應用的眼界。

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

✓ I/O

Input clk // global clock input

Input rst_n // low active reset, to control the clock

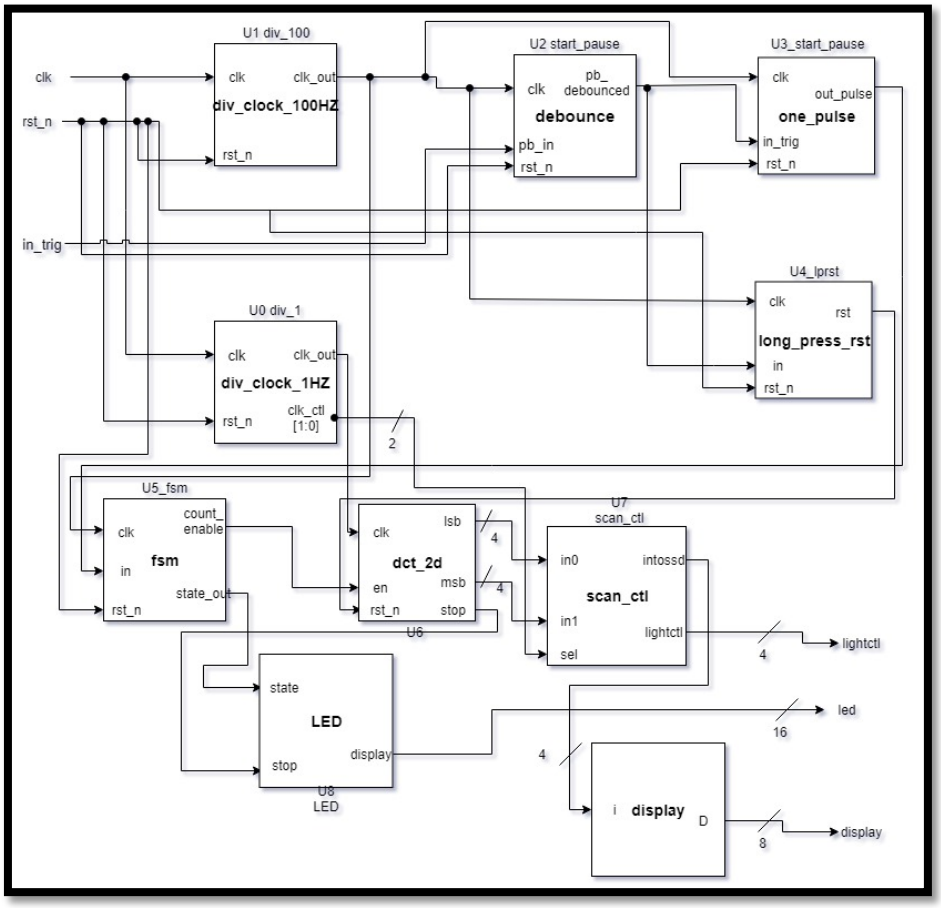
Input in_trig // push button for start / pause / counter reset

Output [15:0]led // led output

Output [7:0]display // number display ssd

Output [3:0]lightctl // which light to light up

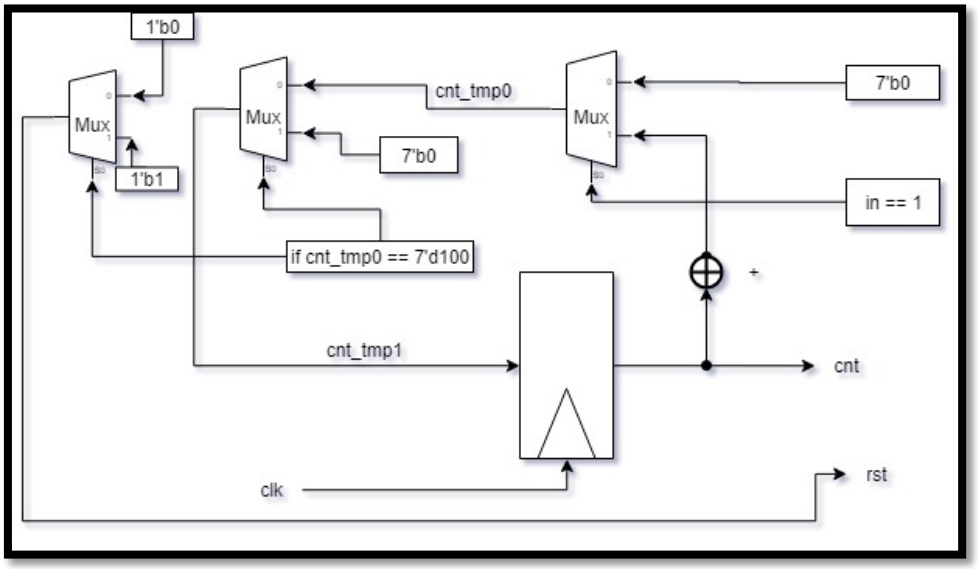
✓ Logic Block:



跟上一題不一樣的地方，必須將reset與mode切換的push button合而為一。在這一題中我多使用了一個rst_n輸入來控制clock及其他訊號處理module的reset，因為在長按的時候會產生時間差，必須考慮posedge與negedge等等問題，不想大幅修改之前使用過的module，因此我選擇了多一個rst_n來控制clock及其他訊號處理module的reset。就把他想成總開關的概念即可，counter的reset還是使用push button控制。

此次設計最大的不同，是我設計了long_press_reset。大致來說裡面會多一個counter來判斷壓的時間是否達到標準，然後輸出reset訊號，接下來會詳細說明。後面的state，ssd等設定皆與前一題相同。

Long_press_reset:



輸入in，判斷是否為1，進行counter向上數。若沒有輸入，counter保持在0。最後有一個判斷條件，如果counter數到7‘d100，代表使用者按了1秒，則counter歸零，rst輸出為1，反之代表按的時間不夠，counter將會繼續數。這個long_press_reset是不會經過one pulse處理的，因此輸入in即為使用者按下的時間，因此可以使用這重判斷方式。

Design Implementation

Input		
clk	*in_trig*	*rst_n*
W5	U18	R2

Output ssd							
display[0]	***display[1]***	***display[2]***	***display[3]***	***display[4]***	***display[5]***	***display[6]***	***display[7]***
V7	U7	V5	U5	V8	U8	W6	W7

Output lightctl			
lightctl[0]	*lightctl[1]*	*lightctl[2]*	*lightctl[3]*
U2	U4	V4	W4

Discussion

這一題處理起來與上一題相當類似，但是這一題中我花費了非常多的時間，處理那個long press reset。設計reset時必須非常小心posedge，negedge，還有究竟是low active還是high active trigger。由於push button與dip運作方式不同，必須很清楚之間的關係才能設計出能夠正持運作的機器。尤其是設定low active high active，這部分的bug真的不容易看出來，必須很清楚自己的設計才是。

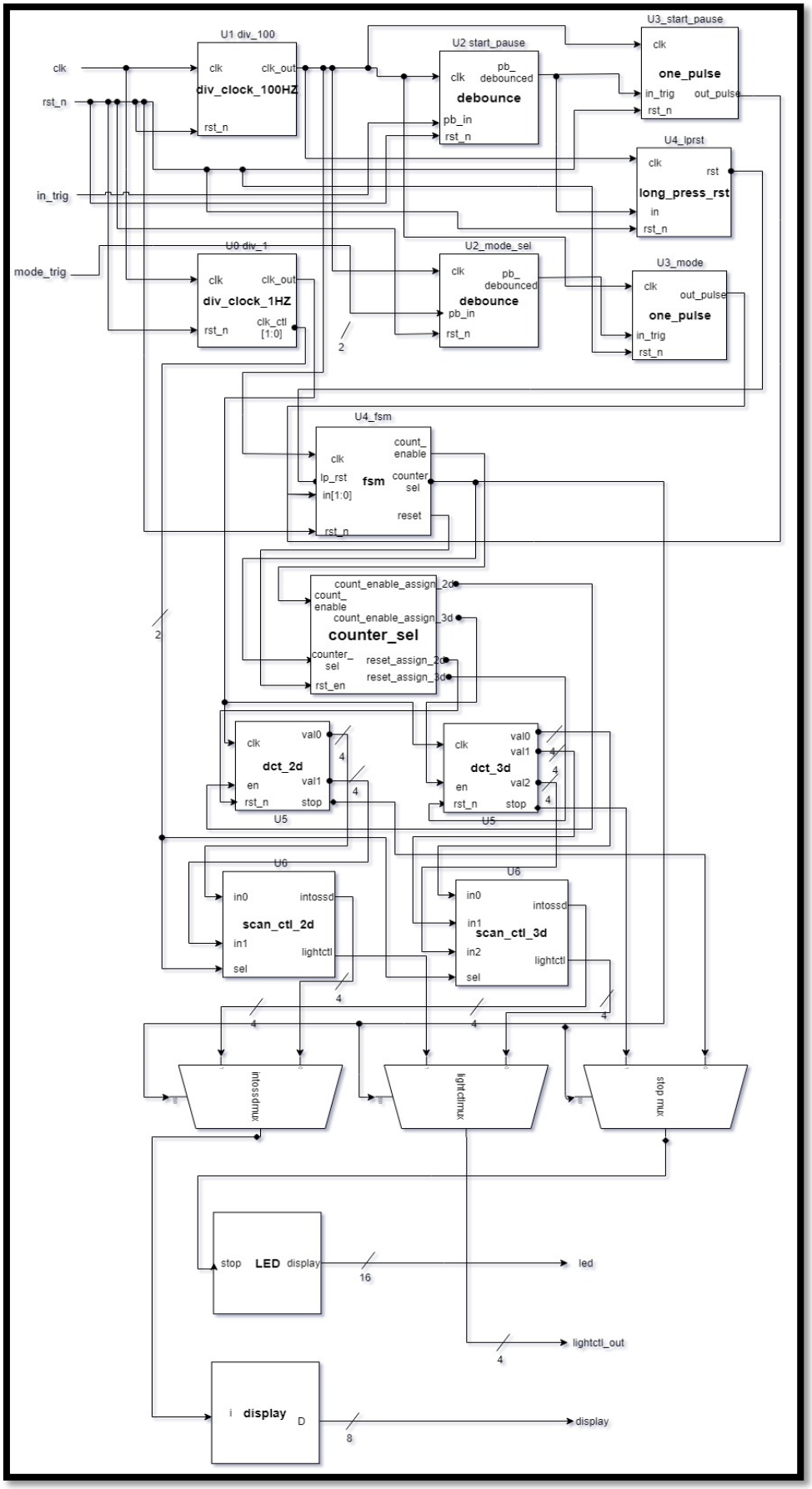
3 (Bonus) Use two push buttons to control a multi-function timer (mode selection, reset, start, stop). The stop timer has two modes: 30-second/1-minute countdown. When being reset, the seven-segment display shows the digits 30/1:00. When the timer counts to 0, it will stop.

Design Specification

✓ I/O

Input clk // global clock input
Input rst_n // low active reset, to control the clock
Input in_trig // push button for start / pause / counter reset
Input mode_trig // 30 60 select
Output [15:0]led // led output, all lightened up when stop
Output [7:0]display // number display ssd
Output [3:0]lightctl _out // which light to light up

✓ Logic Block:

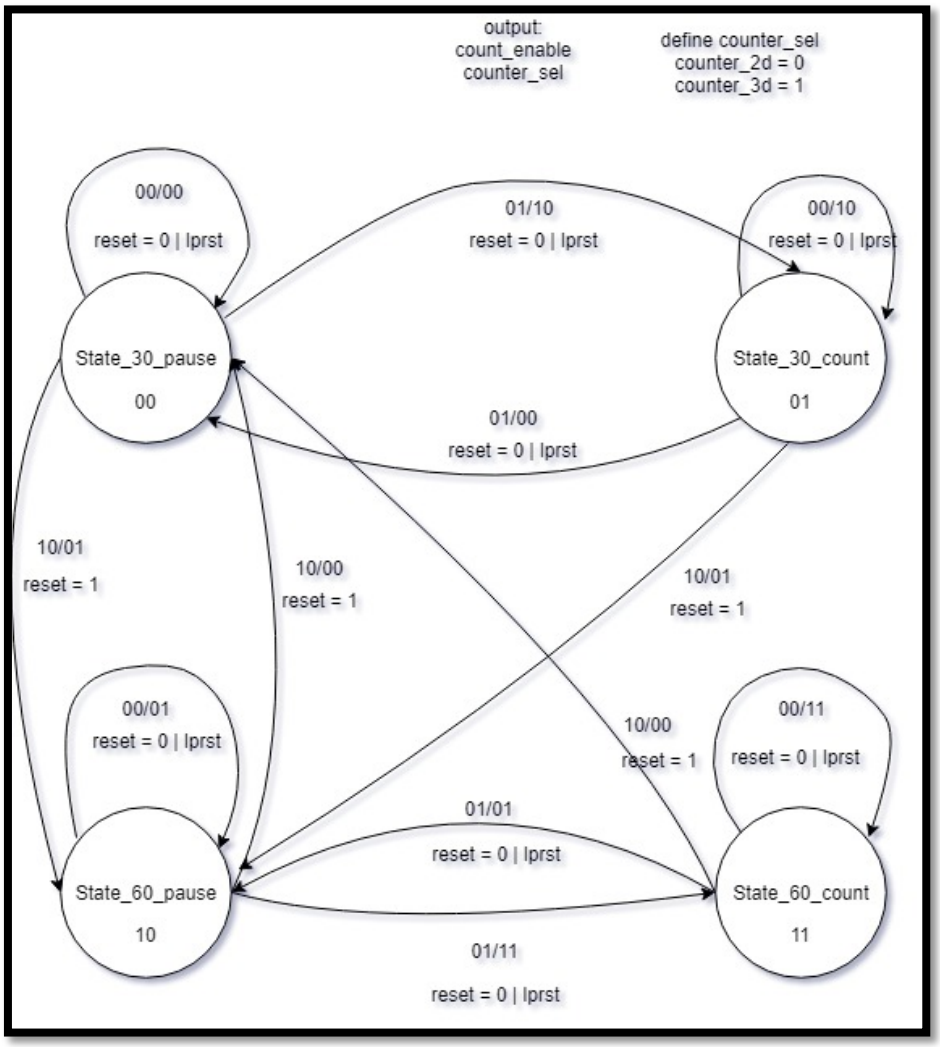

clk
rst_n
in_trig
mode_trig
U1 div_100
div_clock_100HZ
U2 start_pause
debounce
U3_start_pause
one_pulse
U4_lprst
long_press_rst
U0 div_1
div_clock_1HZ
U2_mode_sel
U3_mode
U4_fsm
fsm
counter_sel
dct_2d
dct_3d
U5
U6
scan_ctl_2d
scan_ctl_3d
intossdmux
lightctlmux
stop mux
LED
led
lightctl_out
display
8
16

這一題可以說是非常複雜。先從I/O來說，我設計了兩個push button，一個來控制暫停開始，一個用來切換30/60秒。然後想要reset counter，長按start/pause按鈕即可reset。從大略來說，前面一樣做訊號處理，還有long press reset處理。之後進到fsm中，state稍微複雜，後面會做詳細說明。Fsm output有counter_enable做為counter的指標，counter_sel作為後面MUX選擇要顯示的counter，還有一個reset_assign作為state切換時，指示counter做reset。

這一題我的想法是使用兩個counter，然後最後使用Mux來根據使用者設定來顯示需要顯示的counter。然後後面的led，display與前面相同。

前面的input，start/pause button，以及30/60 push button，還有long press reset，訊號處理與前面相同，就不多加描述。

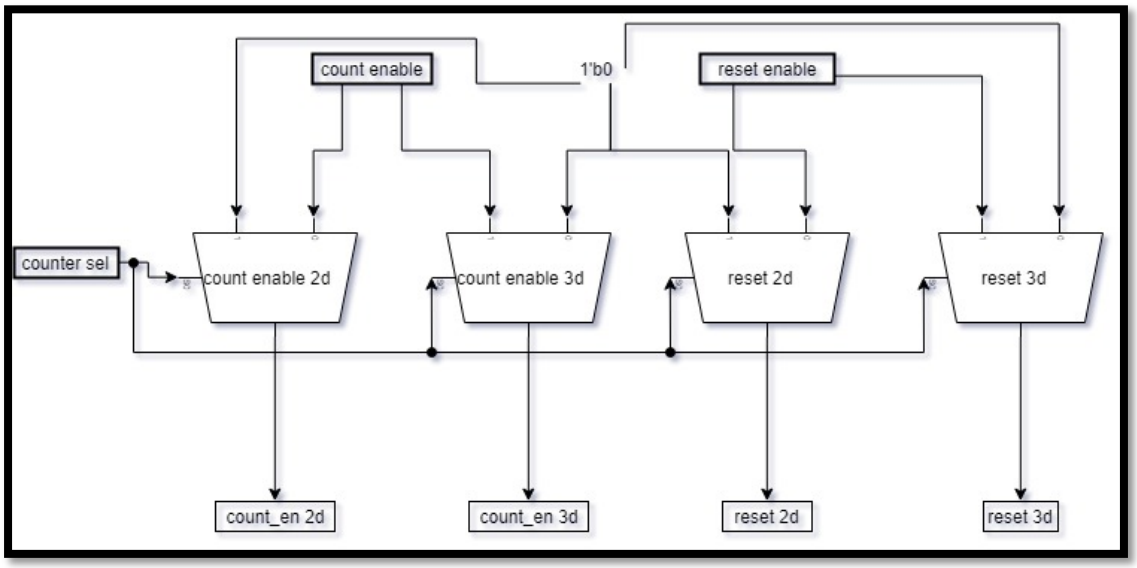
Fsm:



Fsm input共有兩個按鈕，第一個為mode select，選擇30/60。第二個input為

start pause的input。Output共有三個，第一個為count enable，讓counter開始計數。第二個為counter sel，這個output會輸出到後面的mux，可以用來選擇想要顯示的counter。第三個為reset訊號。當我在30秒計數時突然按下切換mode按鈕，這時我會切入到60秒，但是是reset過60秒，亦即我不會只是切到60秒然後仍然繼續在數。由於我的設計牽涉到強制reset，所以我在輸出的地方連上了reset，在每一個輸出reset的地方再跟long press reset的訊號or在一起，如果這時候long press reset有訊號進來也會reset counter。

Counter sel module



Counter sel，counter enable，reset enable輸入都來自fsm的output。當counter sel = 0會選擇2d counter，也就是30秒counter，反之counter sel = 1時會選擇3dcounter，也就是60秒counter。

設計圖後半部份的mux一樣透過counter sel做選擇，選擇目前要顯示的counter。

Design Implementation

Input		
clk	*in_trig*	*rst_n*
W5	U18	R2

Output ssd							
display[0]	***display[1]***	***display[2]***	***display[3]***	***display[4]***	***display[5]***	***display[6]***	***display[7]***
V7	U7	V5	U5	V8	U8	W6	W7

Output lightctl			
lightctl[0]	*lightctl[1]*	*lightctl[2]*	*lightctl[3]*
U2	U4	V4	W4

Discussion

這一題可說是相當有挑戰性的一題，起初真的沒什麼好的想法，後來用了比較暴力的解法，使用兩個counter同時計數。當然這一題非常不好設計，跟很多同學討論一直沒有好的想法，看很多同學是改動initial的值，覺得很容易跑出bug不好偵錯。最後用了很多mux拼拼湊湊，總算是把這一題很複雜的bonus給做出來了。

Conclusion

這一次的lab難度明顯提升很多，module數量突然增加很多，debug起來真的是蠻痛苦的，似乎不像前面的lab這麼輕鬆了。但是經過這次的lab，體驗一次簡易碼表的製作，可以利用fsm製作可以切換功能的手錶，實用性無窮，更是讓我期待後面的lab的到來，做出來的時候會帶來更大成就感。