Lab 5

110060027 朱豐蔚

I. prelab (40s down counter with pause function)

Design Specification

Input: clk, reset, bottom, bottom_reset.
Output: ssd_out[7:0], ssd_ctrl[3:0], led[14:0], led16.

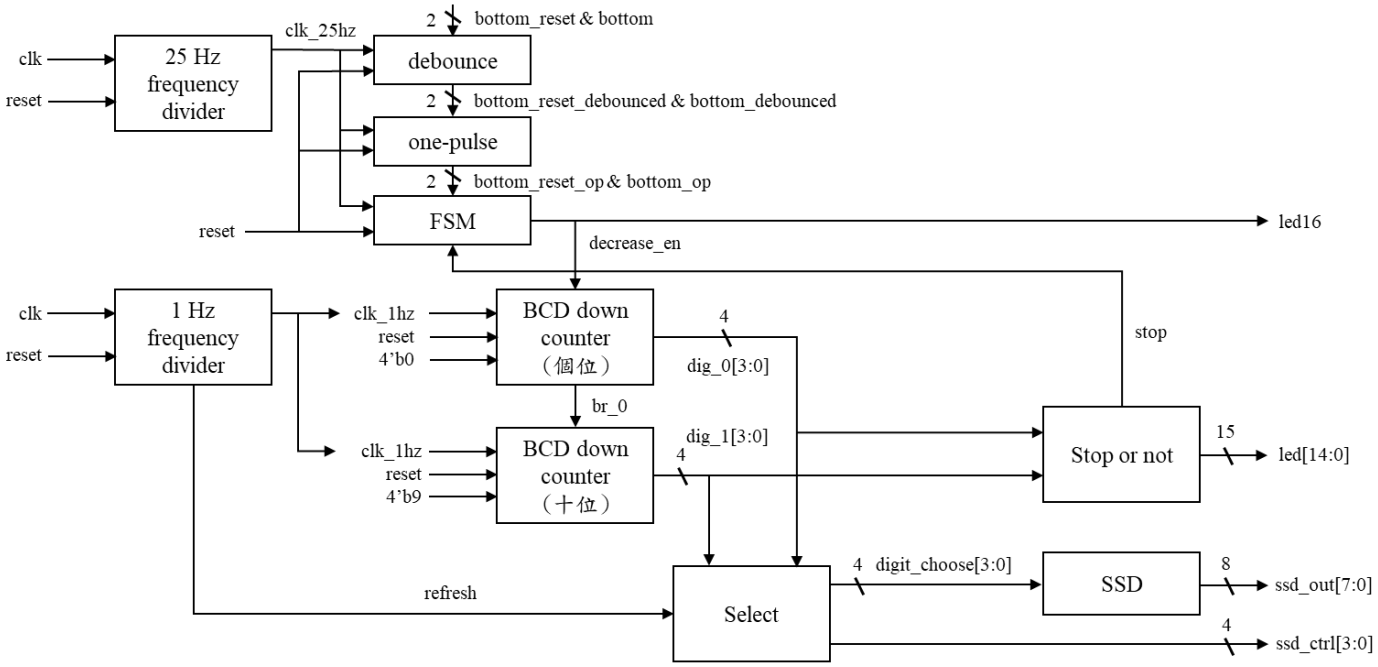
Design Implementation

Specification of the frequency divider:

使用一數 50M 次的 counter 來對原本 100M Hz 的 clk 進行除頻，使輸出為 1 Hz 的訊號 clk_1hz，以提供後續數秒使用。

另使用一數 2M 次的 counter 來對原本 100M Hz 的 clk 進行除頻，使輸出為 25 Hz 的訊號 clk_25hz，以提供 debounce、one-pulse、FSM 使用。

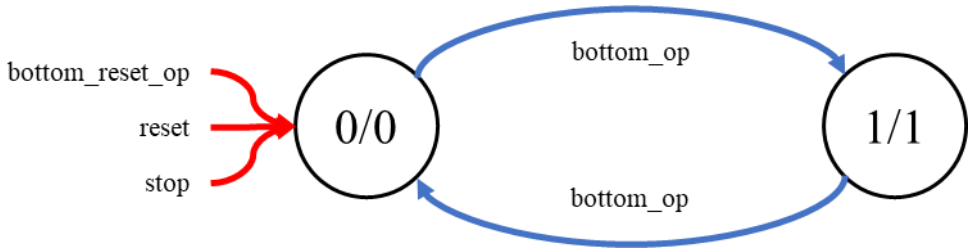
Block diagram:



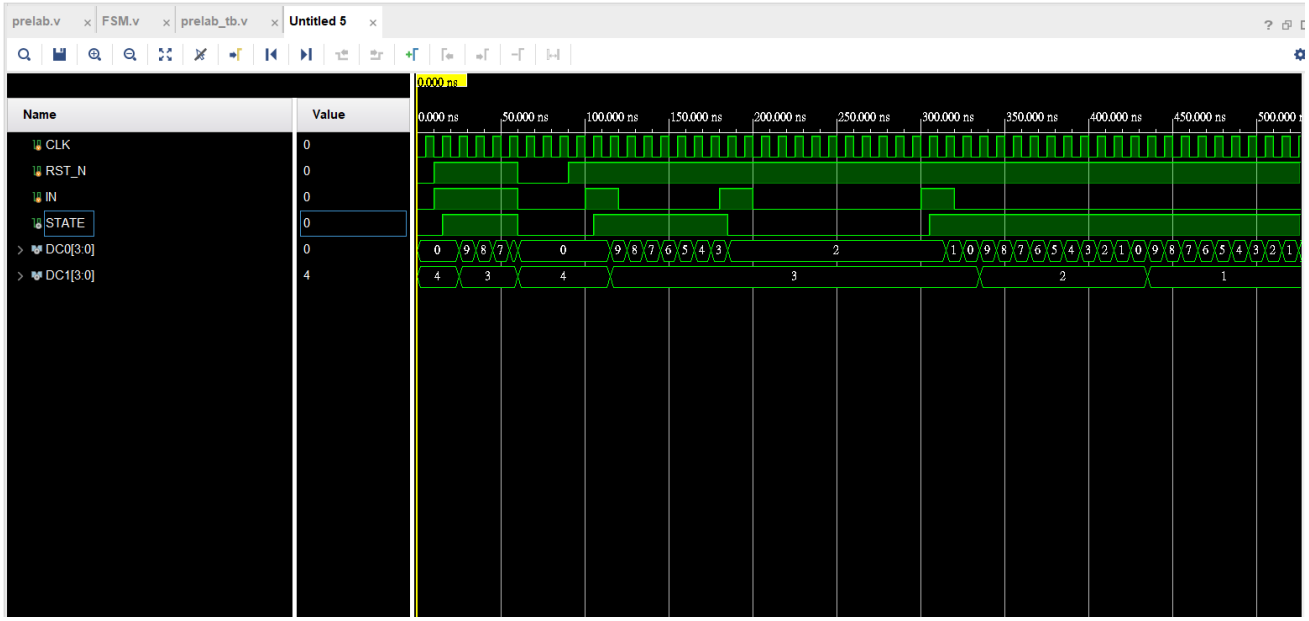
FSM design:

考慮此題，僅有停止數、繼續數兩種狀態，故只需要一位元即可表示所有 state。

使用與 debounce、one-pulse 相同的 clk_25hz 作為判斷速度，每當收到 bottom_op 訊號時切換 state，當收到 reset 或 bottom_reset_op 或 stop 時重置 FSM，使 FSM 回到停止數的狀態，而最後輸出 decrese_en 作為 state 的判斷。



simulation waveform:



Discussion:

跟著 prelab 的設計走，第一次接觸按鈕設計的我們，很快就可以上手並找到 debounce、one-pulse 模組中的錯誤，prelab 中要求的 state 也不多，很方便我們進行 FSM 的撰寫，我覺得這次 prelab 的難度恰到好處。

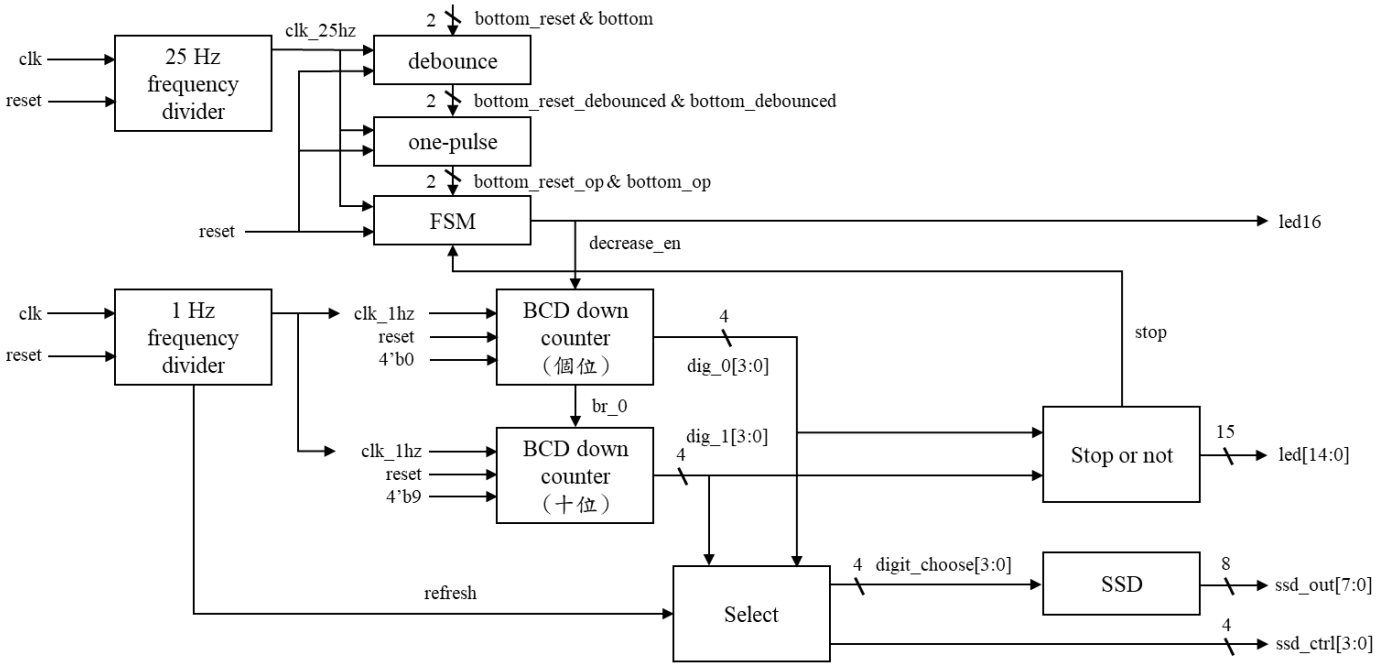
II. exp_1 (40s down counter with pause function)

Design Specification

Input: clk, reset, bottom, bottom_reset.
Output: ssd_out[7:0], ssd_ctrl[3:0], led[14:0], led16.

Design Implementation

Block diagram:



Design flow:

將 prelab 中測試完全的 FSM 模組套入進來，並加上帶有 enable 輸入的 BCD down counter，使用 decrease_en 作為此 counter 的 enable。此外，與 Lab4-4 相似，透過兩個 counter 分別去紀錄十位數與個位數，其中將 br_0 作為串連兩個 counter 的進位引導，最後引入 select、SSD 模組以進行數字的獨立顯示。

在題目另外有要求當數到 0 時，停止記時並點亮所有 led，我使用 if 的寫法，當 dig_1 跟 dig_0 都為 4'b0 時，使 stop 變成 1，led 全亮即可達成題目所求。

I/O pins assignment:

I/O	clk	reset	bottom	bottom _reset
LOC	W5	V17	T17	W19

I/O	ssd_ ctrl[3]	ssd_ ctrl[2]	ssd_ ctrl[1]	ssd_ ctrl[0]
LOC	W4	V4	U4	U2

I/O	s_o[7]	s_o[6]	s_o[5]	s_o[4]	s_o[3]	s_o[2]	s_o[1]	s_o[0]
LOC	W7	W6	U8	V8	U5	V5	U7	V7

I/O	led[14]	led[13]	led[12]	led[11]	led[10]	led[9]	led[8]	led[7]
LOC	L1	P1	N3	P3	U3	W3	V3	V13

I/O	led[6]	led[5]	led[4]	led[3]	led[2]	led[1]	led[0]	led16
LOC	V14	U14	U15	W18	V19	U19	E19	U16

Discussion:

我覺得這題是承上啟下的，承接著 Lab4-4 的兩位元 counter，而往下延伸出 FSM 與按鈕的 debounce、one-pulse，但不一樣的地方是此題的 counter 為 down counter，FSM 也較為簡單，我認為這題的難度是恰到好處的。

在此題的撰寫當中，我遇到最大的困難就是在 debug 上，因為我是一次性把所有模組寫完再進行 implement 的，所以當我按按鈕沒有反應時，我完全不知道是哪個環節出了問題，花了很長的時間才找到是因為我在 one-pulse 模組中多打了一個 ~。這次經驗也告訴了我，一次做太多反而不是一件好事，應該要逐步慢慢前進，做多少先看多少，才能夠提早發現錯誤，及時修正。

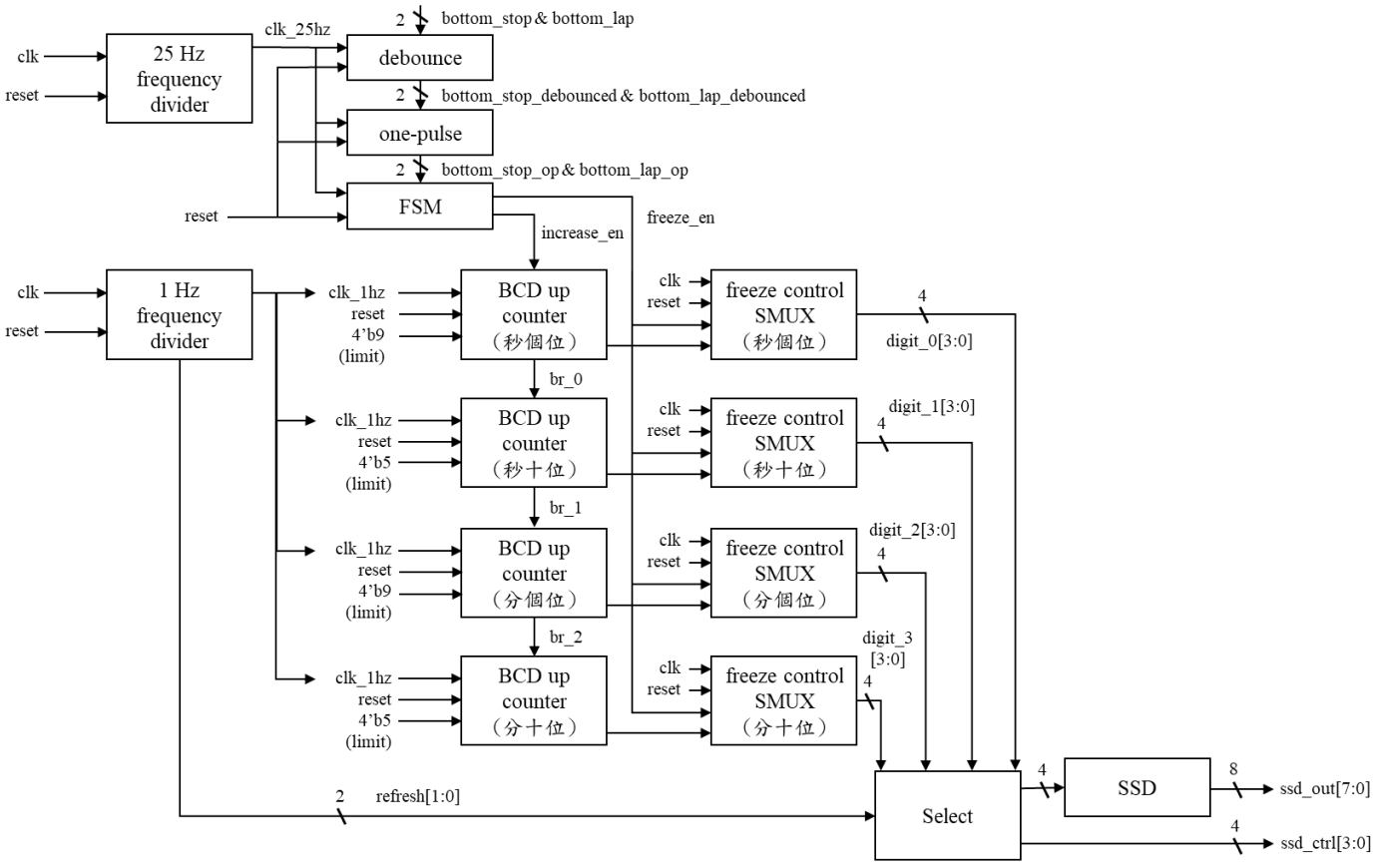
III. exp_2 (stopwatch)

Design Specification

Input: clk, reset, bottom_stop, bottom_lap.
Output: ssd_out[7:0], ssd_ctrl[3:0].

Design Implementation

Block diagram:

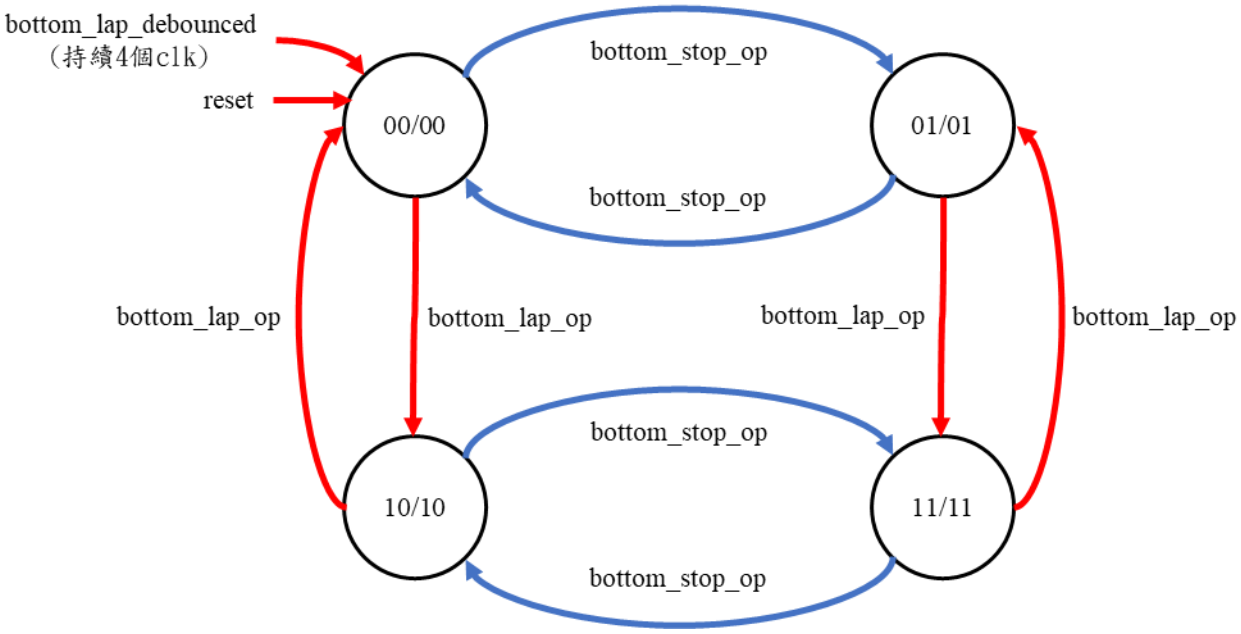


Design flow:

相比第一題，這題的複雜處在於 state 變多了，考慮是否 count、是否 freeze，總計就有 4 種 state，因此需要兩位元來表示。

在 FSM 的設計中，如下圖，設定 00 為不數且不 freeze、01 為數且不 freeze、11 為數且 freeze、10 為不數且 freeze，因此，令 output 中的第一項為 freeze_en，第二項為 count_en，即可達成題目的要求。

另外，freeze 控制，我使用了一個 SMUX 來做控制，若 freeze_en = 1，就讓輸入等於輸出，在下一個 clk 後，就會維持不變；而當 freeze_en = 0，就讓輸入等於 counter 中的值，即可正常顯示出來。



I/O pins assignment:

I/O	clk	reset	bottom_stop	bottom_lap
LOC	W5	V17	T17	W19

I/O	ssd_ctrl[3]	ssd_ctrl[2]	ssd_ctrl[1]	ssd_ctrl[0]
LOC	W4	V4	U4	U2

I/O	s_o[7]	s_o[6]	s_o[5]	s_o[4]	s_o[3]	s_o[2]	s_o[1]	s_o[0]
LOC	W7	W6	U8	V8	U5	V5	U7	V7

Discussion:

在這題的撰寫當中，我遇到最大的問題就是長按 reset 的設計，因為原先我一直是使用 bottom_lap 的 one-pulse 訊號作為 window 的判斷輸入，但一直發現沒反應，最後我另外增加 4 個燈號作為判斷 window 狀態的輸出，才發現 window 要使用 bottom_lap 的 debounced 訊號才可成功。

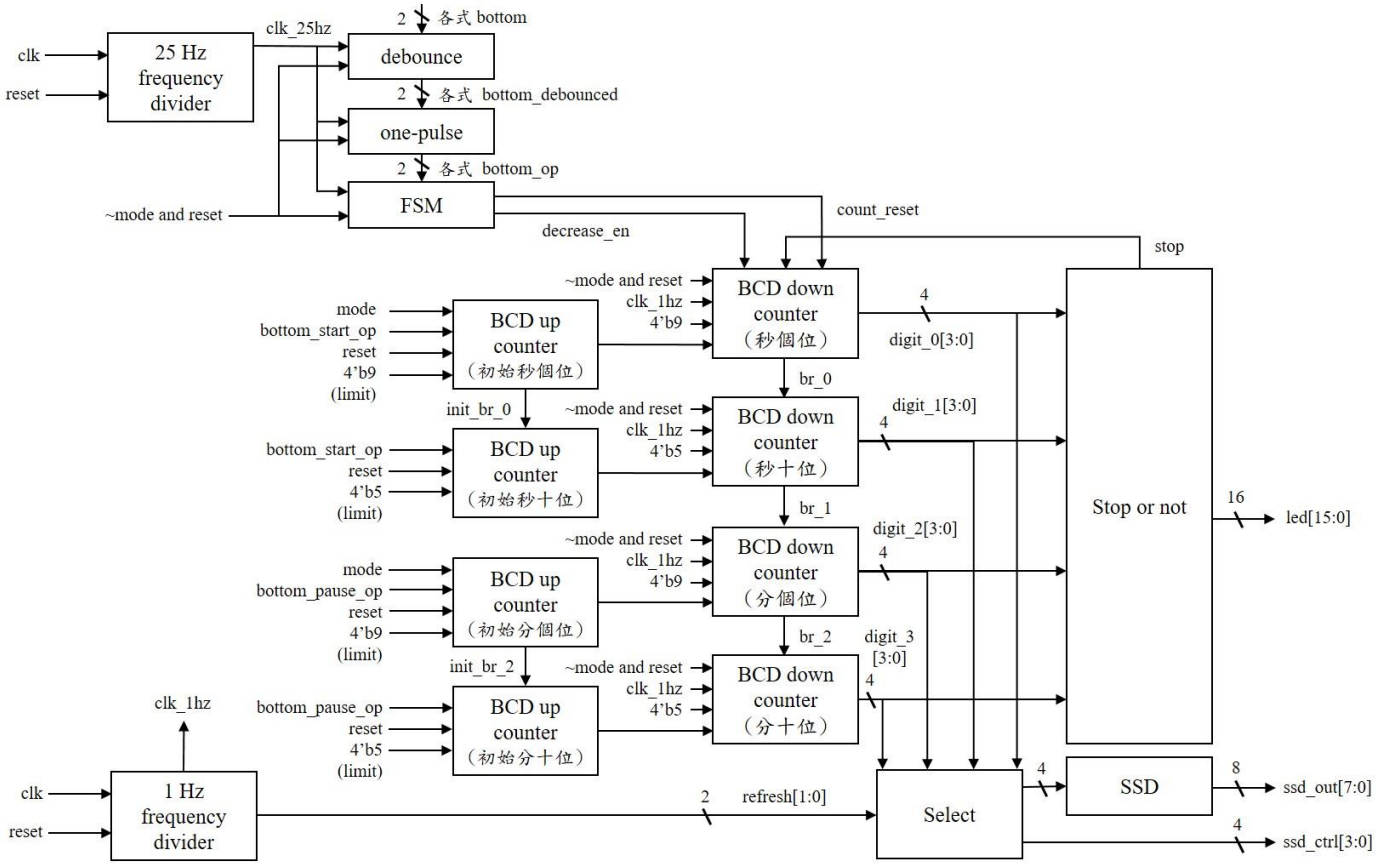
IV. exp_3 (timer with setting function)

Design Specification

Input: clk, reset, mode, bottom_start, bottom_pause.
Output: ssd_out[7:0], ssd_ctrl[3:0], led[15:0].

Design Implementation

Block diagram:



Design flow:

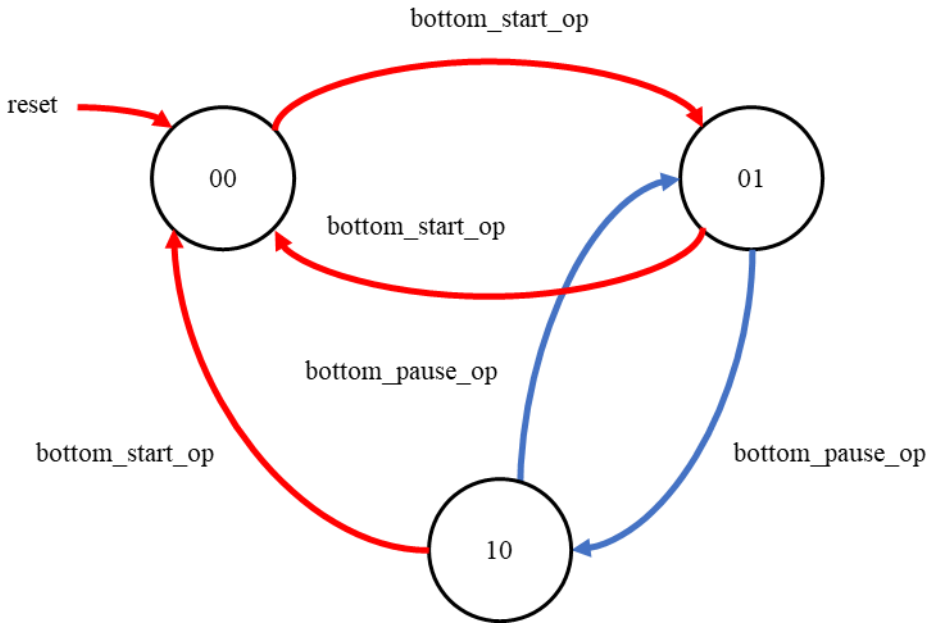
首先先考慮此題所需要的 state，有 reset 後的初始態，訂為 00，在來就是可以數的狀態，訂為 01，最後是暫停態，訂為 10。

當在初始態按下 start 後就會進到數與暫停的循環，此時 pause 鍵會讓 state 在 01、10 之間互相切換，此時控制 counter 是否數的訊號即為 count_en。而不管在 01 還是 10，再次按下 start 就會回到 00 態，此時讓 count_reset 變為 1，讓 counter 重置，回到設定的初始值。

接著要設計模式的切換與設定初始值方式，我想到的方法是另外創立 4 個 counter 去作為 timer 的初始值紀錄，於是設定只會影響到初始值 counter，再由其值去做為 timer 的 initial 輸入，即可在不干擾 timer 的情況下達成設定效果。

同 exp_1，一樣會需要設立當所有數值歸 0 時的 led 輸出，我一樣選用 if 的寫法，當所有數值都為 0 時，產生 stop 訊號，讓 timer 不再倒數，且讓所有 led 燈發亮。

最後的數值呈現，由於沒有 freeze 功能，所以不需要如 exp_2 一樣加入 SMUX 去選擇，可以直接將數值輸入到 select 當中，再藉由 refresh 訊號去做數值的獨立顯示。



I/O pins assignment:

I/O	clk	mode	reset	bottom _start	bottom _pause
LOC	W5	V17	R2	T18	U18

I/O	ssd_ ctrl[3]	ssd_ ctrl[2]	ssd_ ctrl[1]	ssd_ ctrl[0]
LOC	W4	V4	U4	U2

I/O	s_o[7]	s_o[6]	s_o[5]	s_o[4]	s_o[3]	s_o[2]	s_o[1]	s_o[0]
LOC	W7	W6	U8	V8	U5	V5	U7	V7

I/O	led[15]	led[14]	led[13]	led[12]	led[11]	led[10]	led[9]	led[8]
LOC	L1	P1	N3	P3	U3	W3	V3	V13

I/O	led[7]	led[6]	led[5]	led[4]	led[3]	led[2]	led[1]	led[0]
LOC	V14	U14	U15	W18	V19	U19	E19	U16

Discussion:

我認為 exp_3 相比於 exp_2，在 FSM 的撰寫上較為簡單，因為 state 數僅有 3 個，且彼此的切換簡單易懂，但更困難的地方是在設定模式的創建，需要引入 switch 作為模式的切換，還需要考慮如何加入設定的功能。

我首先設想了兩種設定方式，第一種如上述，將初始值 counter 獨立出來，而第二種則是合在一起，但合在一起的方式第一是難寫，第二是輸入太多太雜容易互相干擾，因此我選擇了第一種方式作為我的設定寫法。

Conclusion

在這次 Lab 中，我最大的收穫就是學會了按鈕的寫法，透過 debounce 消除雜訊，在利用 one-pulse 使訊號單一不重複，這兩個模組缺一不可。尤其是在 exp_2 中的長按功能，更讓我了解到了 debounced 訊號與 one-pulse 訊號個有其優缺點與應用之處，one-pulse 雖然不會重複切換 state，但同時卻無法處理長按這種需要紀錄時間長度的 function，因此，如何靈活運用這兩種訊號的型態應該會是今後最大的課題之一。

此外，我也想到 one-pulse 的這種寫法並不侷限於按鈕，應該試用任何長訊號來源，因此，更可以透過 one-pulse 讓 switch 也有按鈕的效果，這點可以應用於 Lab6 的撰寫當中，豐富了我們處理題目的方法與手段。

References

FSM 編撰方式: (from 第五週上課講義)
Freeze 功能編撰方式: (from 第五週上課講義)