Lab9 Report

9-1 One-digit BCD addition

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst, (控制整個功能的開關)

Output:

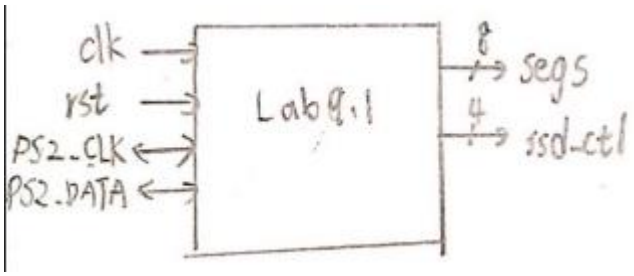
[7:0]segs,(接上七段顯示器的七條燈)

[3:0]ssd_ctl, (接上四個七段顯示器，控制他們亮暗的頻率)

Inout:

PS2_CLK,(讓 PS2, KEYBOARD 彼此溝通用)

PS2_DATA(讓 PS2, KEYBOARD 彼此溝通用)

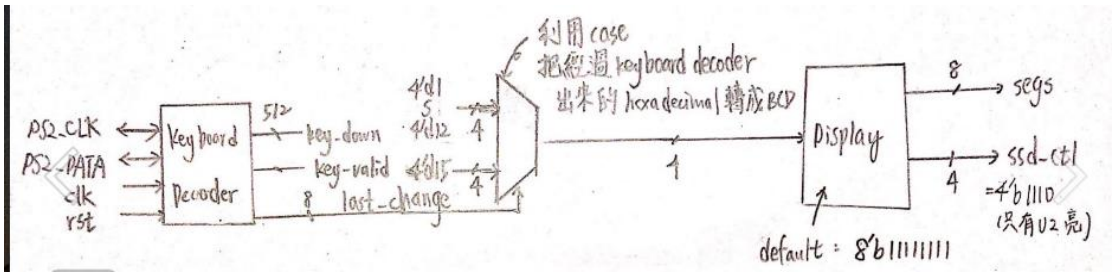


●Design Implementation

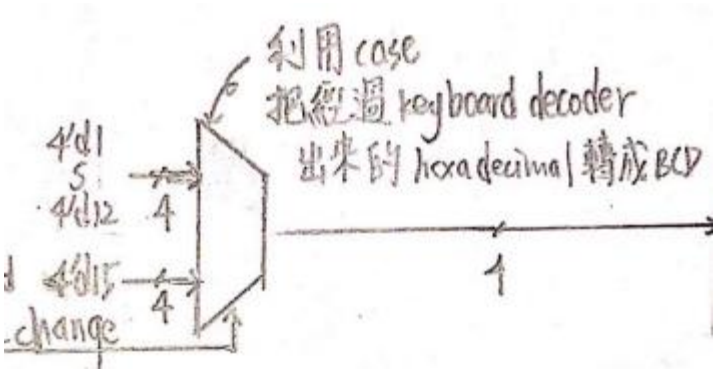
1.Outline

這個實驗主要是要讓我們熟悉接上鍵盤的控制方法，功能是要按下 0~9 還有 A、S、M、ENTER 的時候，七段顯示器上面也會顯示相對應的數字或符號，A、S、M、ENTER 分別利用 10、11、12、15 代替。但是按下 ENTER 的時候要關掉七段顯示器。

2.Logic Diagram



↑第一題使用到的的 module 數目比較少，主要只有加上一個 mux 來當作 decoder 而已



↑上圖的這個 MUX，是利用 CASE 的方法把 16 進位的 last_change 給轉成 BCD，所以判斷條件就是依照作業上的表跟對照的數字一對一對應；另外 A、S、M 就分別利用 10、11、12 代替，ENTER 鍵則是用 15 來代替，為了要達到按下 ENTER 鍵七段顯示器就按掉的效果，我在 Display 中只寫了 1~12 的條件，default 是 8’b11111111，也就是說按下 enter 的時候會讓全部的七段顯示器暗掉。

↓以下是 I/O 接腳

I/O	segs [7]	segs [6]	segs [5]	segs [4]	segs [3]	segs [2]	segs [1]	segs [0]	Clk	Rst
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	V17

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

I/O	PS2_CLK	PS2_DATA
Pin	C17	B17

●Discussion

一開始要理解整個 KEYBOARD 的原理比較花時間，但在理解之後就比較順利了，原本我用我宿舍的鍵盤，結果插上去 BUSY 燈一直亮著，然後完全沒有反應，結果問了同學才知道說可能我的鍵盤太新(?沒有支援，這讓我花了蠻多時間的……，結果最後用實驗室借來的鍵盤才可以順利 DEMO。(免費的東西就是讚)

9-2. Single digit decimal adder

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst, (控制整個功能的開關)

Output:

[7:0]segs,(接上七段顯示器的七條燈)

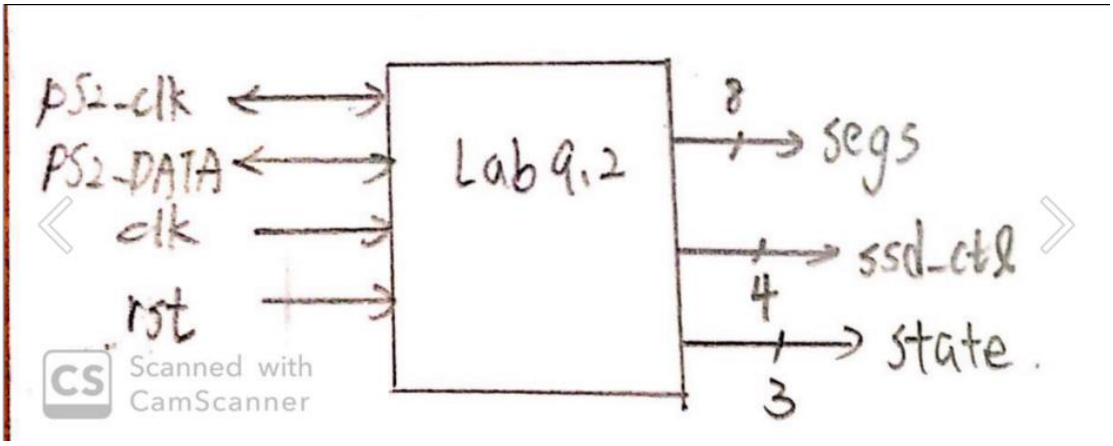
[3:0]ssd_ctl, (接上四個七段顯示器，控制他們亮暗的頻率)

[2:0]state(為了更好掌握 state 的變化，利用三個 Led 顯示)

Inout:

PS2_CLK,(讓 PS2, KEYBOARD 彼此溝通用)

PS2_DATA(讓 PS2, KEYBOARD 彼此溝通用)



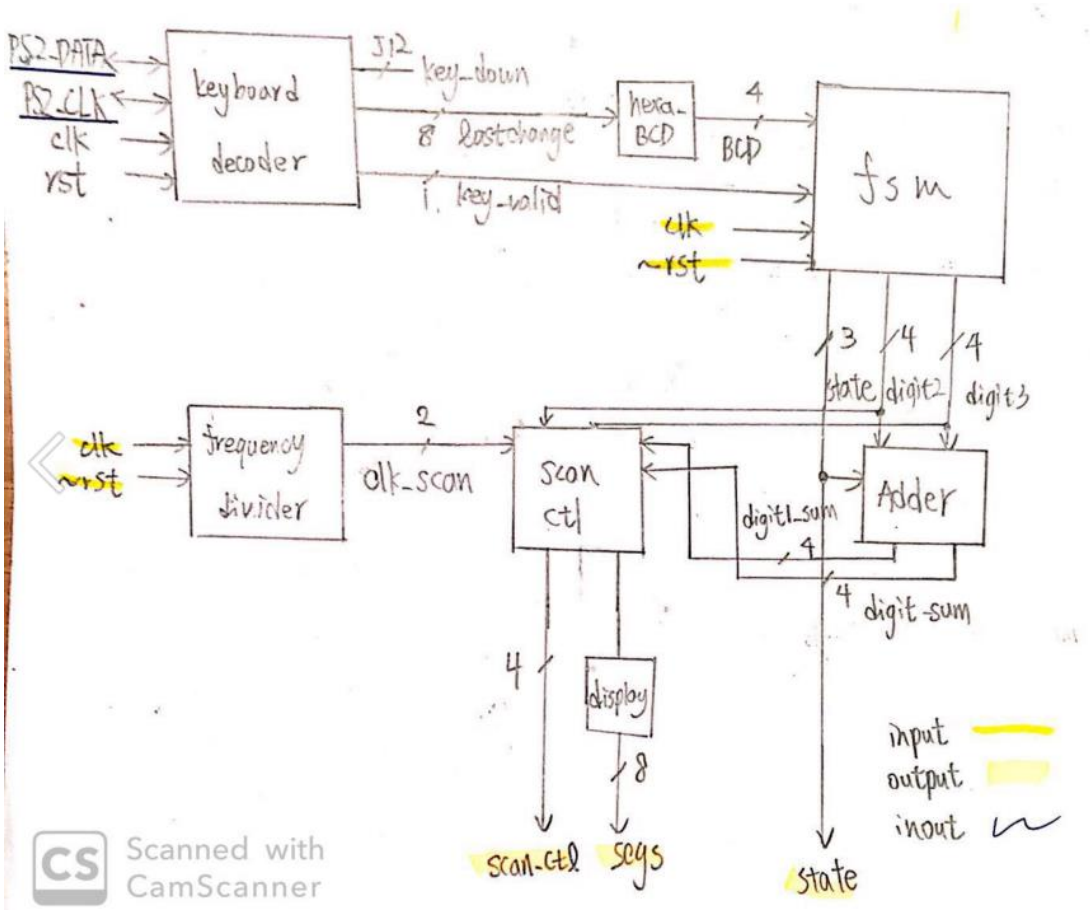
●Design Implementation

1.Outline

這個實驗主要是要做出 1bit 的 BCD 加法器，七段顯示器的前面兩位分別代表兩個 BCD 的數字，右邊兩位則代表出來的結果。

這邊的 FSM 我並沒有打得非常好，在 9.2 這題不會有什麼影響，但是之後的題目就會有影響了，會在之後的題目中說明。

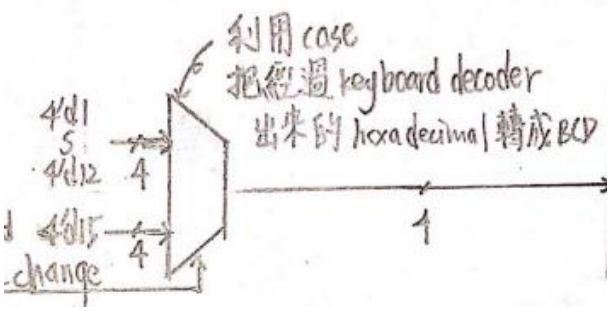
2.Logic Diagram



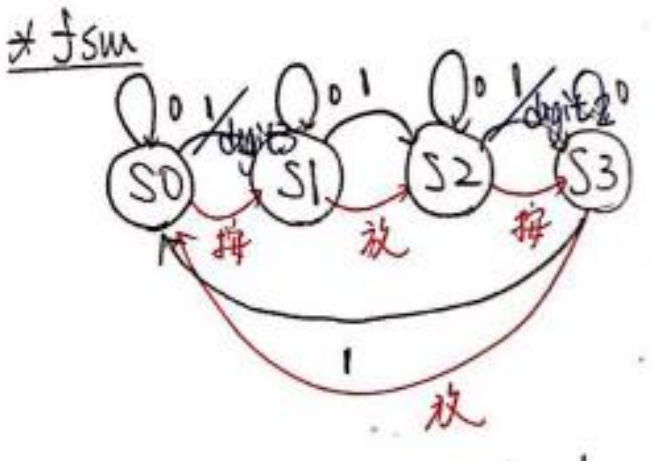
↑ 上圖是整個系統的邏輯圖，新的 module 在下面介紹

▲Hexa_BCD

這個 MODULE 其實就是上面 9.1 的那個把 keyboard 十六進位轉成 BCD，因為感覺會時常用到，所以乾脆寫了一個 module 給他。不過把 enter 鍵改回一般的功能了。



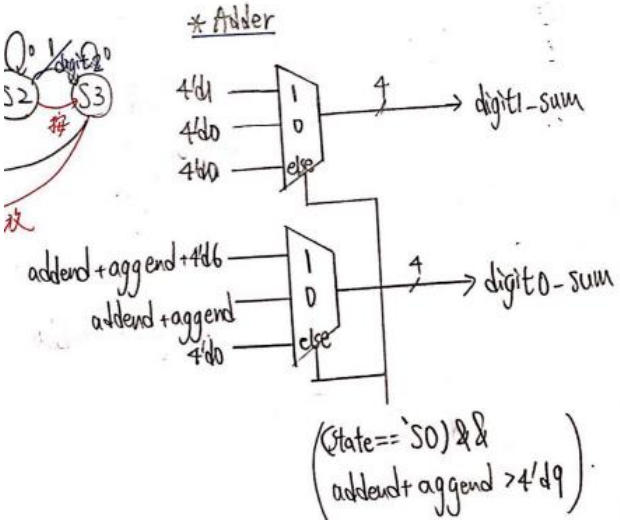
▲fsm



↑我用 state 的圖片來解釋 fsm 的功能，一開始是默認在 S0，之後利用 key_vaild 當作 input，所以在按下任何鍵的時候，會跑到下一個 state，然後因為 key_valid 的特性，再放開按鍵的時候又會產生下一個 input，所以等於我完整按下按鈕一次，會跑兩個 state。利用這種特性在 S0 轉換到 S1 的時候的 last_change 輸入到 digit3 裡面；S2 轉換到 S3 的時候把 last_change 輸入到 digit2 裡面，在 S3 放開按鍵的時候就會顯示出 SUM 的結果出來。

另外值得一提的是，這次的 fsm 我是輸出 next_state，而不是像是之前的 state，因為這次需要瞬間變化，如果用 state 還要等那個 Dflipflop 再經過一個 clk 才會有結果，所以比較特殊。

▲Adder



↑這個是負責輸出 sum 的 module，利用兩個 mux，兩個的 output 分別為 digit1_sum 和 digit0_sum，控制的條件是要當 state == S0 且加數與被加數是否大於 9 來決定。大於 9 的話還要加上六才會顯示正確的 BCD 結果。

↓以下是 I/O 接腳

I/O	segs [7]	segs [6]	segs [5]	segs [4]	segs [3]	segs [2]	segs [1]	segs [0]	Clk	Rst
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	V17

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

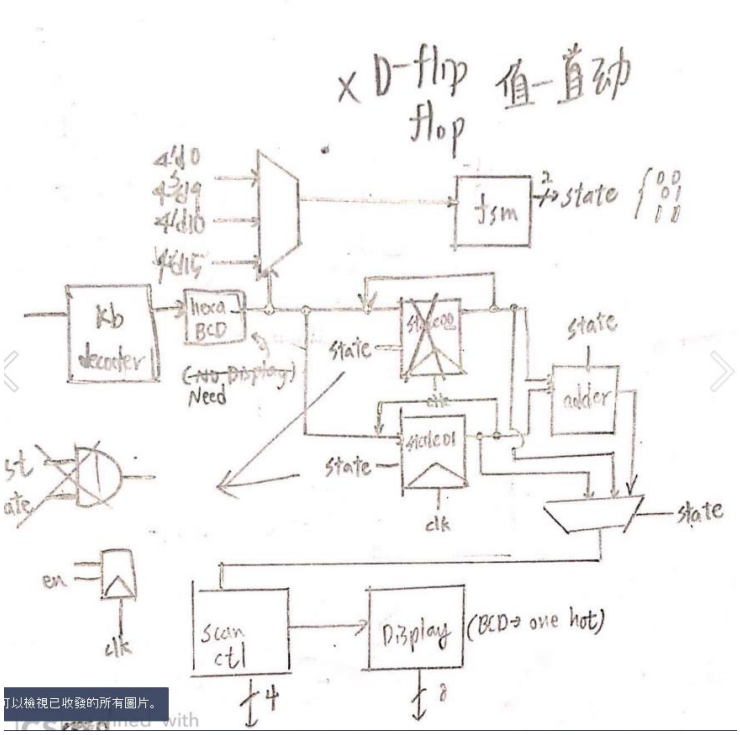
I/O	PS2_CLK	PS2_DATA	State[2]	State[1]	State[0]
Pin	C17	B17	L1	P1	N3

●Discussion

我在這個實驗一開始遇到蠻大的困難的，因為還沒有相當了解鍵盤的控制方法。

1. 我一開始把 FSM 的 INPUT 設置為 last_change，我一開始覺得十分合理，就是按下去的時候剛好可以把 last_change 輸入進去，但是結果我用來檢測 state 的三個 LED 燈會會全部都亮的(比較暗一點)，仔細想想之後原來是 last_change 是按下去那個按鍵就會一直輸出，也就代表說會讓 fsm 一直以瘋狂的速度跟換不同的 state，燈比較按的原因是他人眼沒辦法捕捉到這麼快頻率的變化，但是會感受到整體亮度變暗。

2. 第二個是我一開始想說先利用兩個 D_flipflop 來儲存加數與被加數，然後設定在 S2 的時候輸入 last_change 到相對應的位數，但是出來的結果是在按第一次按鈕的時候，digit3&digit2 都會跑出當時的 BCD，原因是 STATE 的效果設置不當，會讓第二個數字在不該接收到數字的時候接受到。



↑一開始設計的邏輯圖

3. 忘記在 FSM 的每個變數都要設定一個 Dflipflop 給他，因為我一開始並沒有設定，這樣會導致切換到下一個 state 的時候，前面的變數就跑掉了。Ex: 設定好第一個 digit 的時候，切換到 S3 的時候，結果第一個 digit 就變成跟第二個 digit 一樣了。

9-3.Two digit adder/subtractor/multiplier

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst, (控制整個功能的開關)

Output:

[7:0]segs,（接上七段顯示器的七條燈)

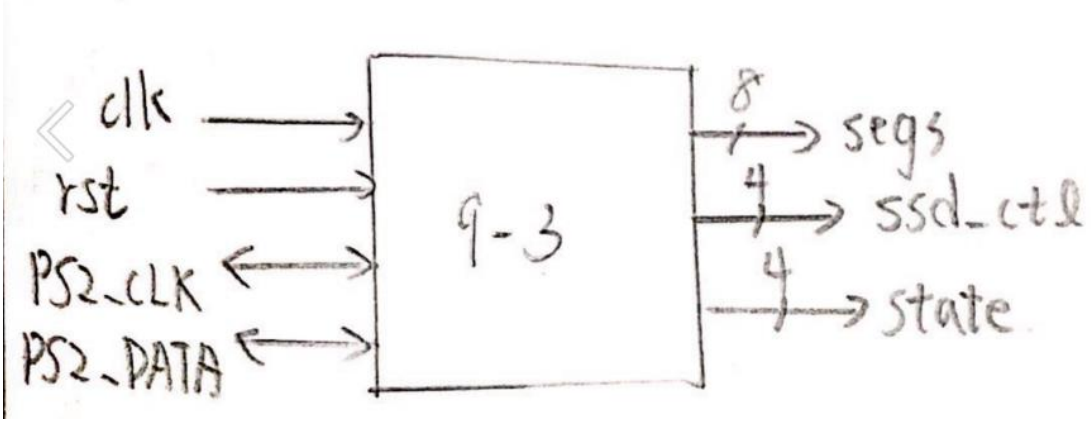
[3:0]ssd_ctl, (接上四個七段顯示器，控制他們亮暗的頻率)

[3:0]state(為了更好掌握 state 的變化，利用三個 Led 顯示）

Inout:

PS2_CLK,(讓 PS2, KEYBOARD 彼此溝通用)

PS2_DATA(讓 PS2, KEYBOARD 彼此溝通用)

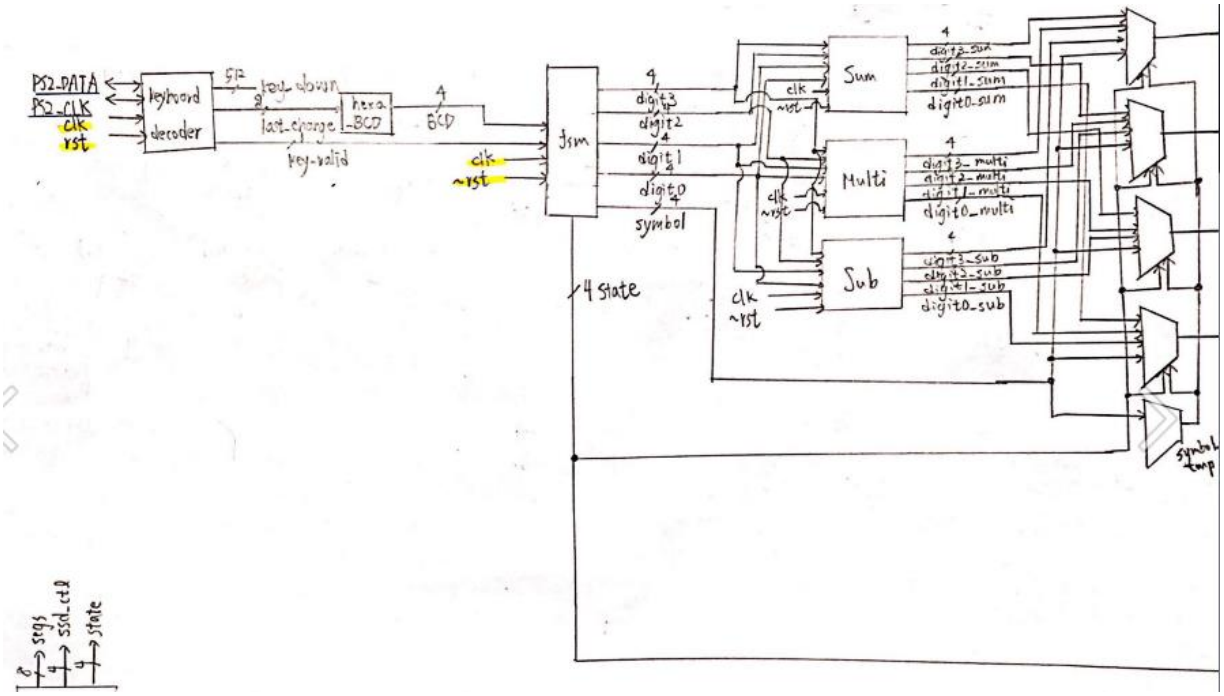


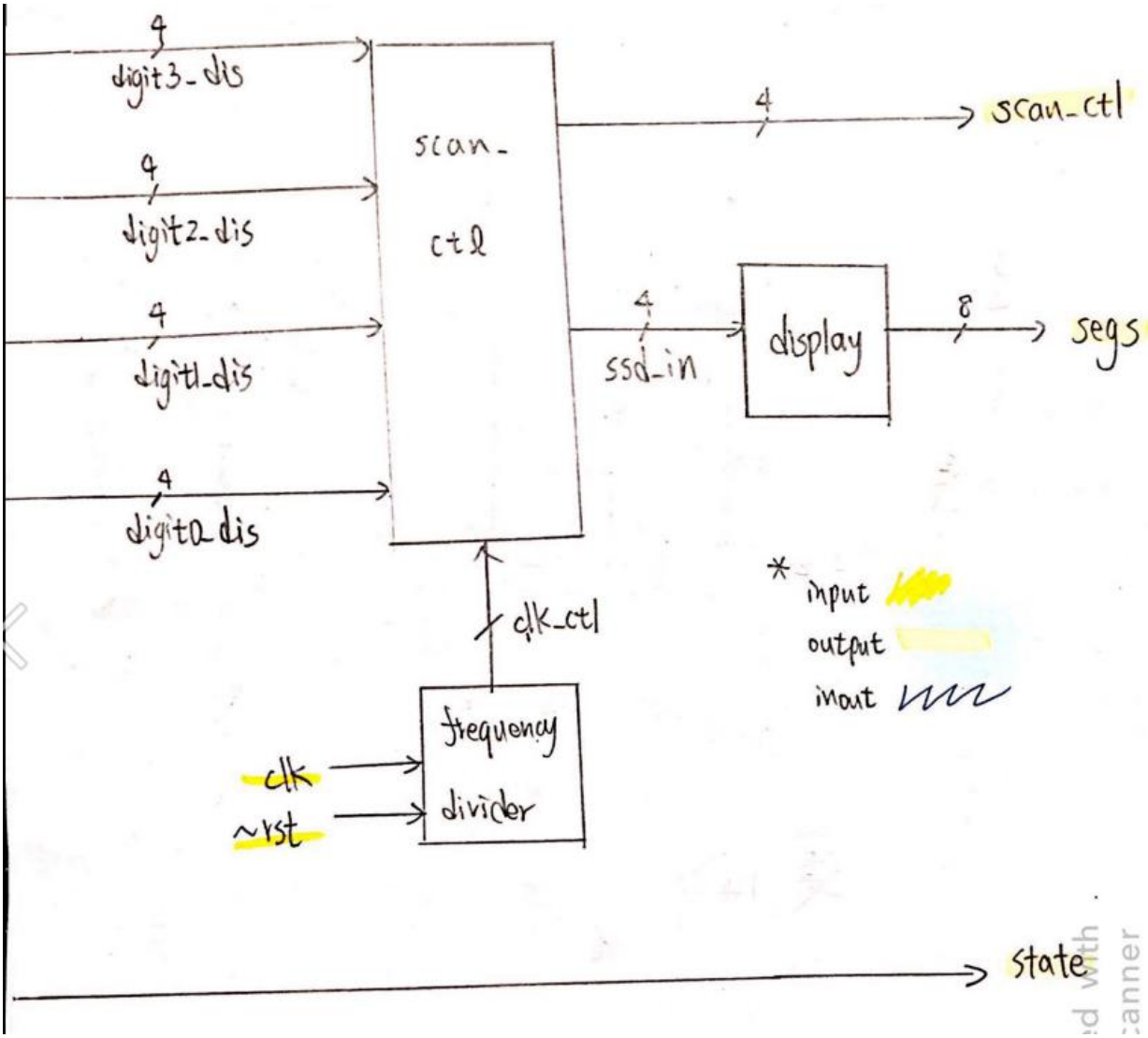
●Design Implementation

1.Outline

這題是這次作業裡面最難的，因為需要考慮到非常多的情況，還有要非常理解 keyboard 的 last_change、key_valid、key_down 的原理應用。要執行 2 個兩 digit 的加法、減法、乘法，要在不同的 state 顯示出當時的情況。還需要記得在計算完之後還要變回 BCD，並設定在正確的位數上，才可以正確顯示。

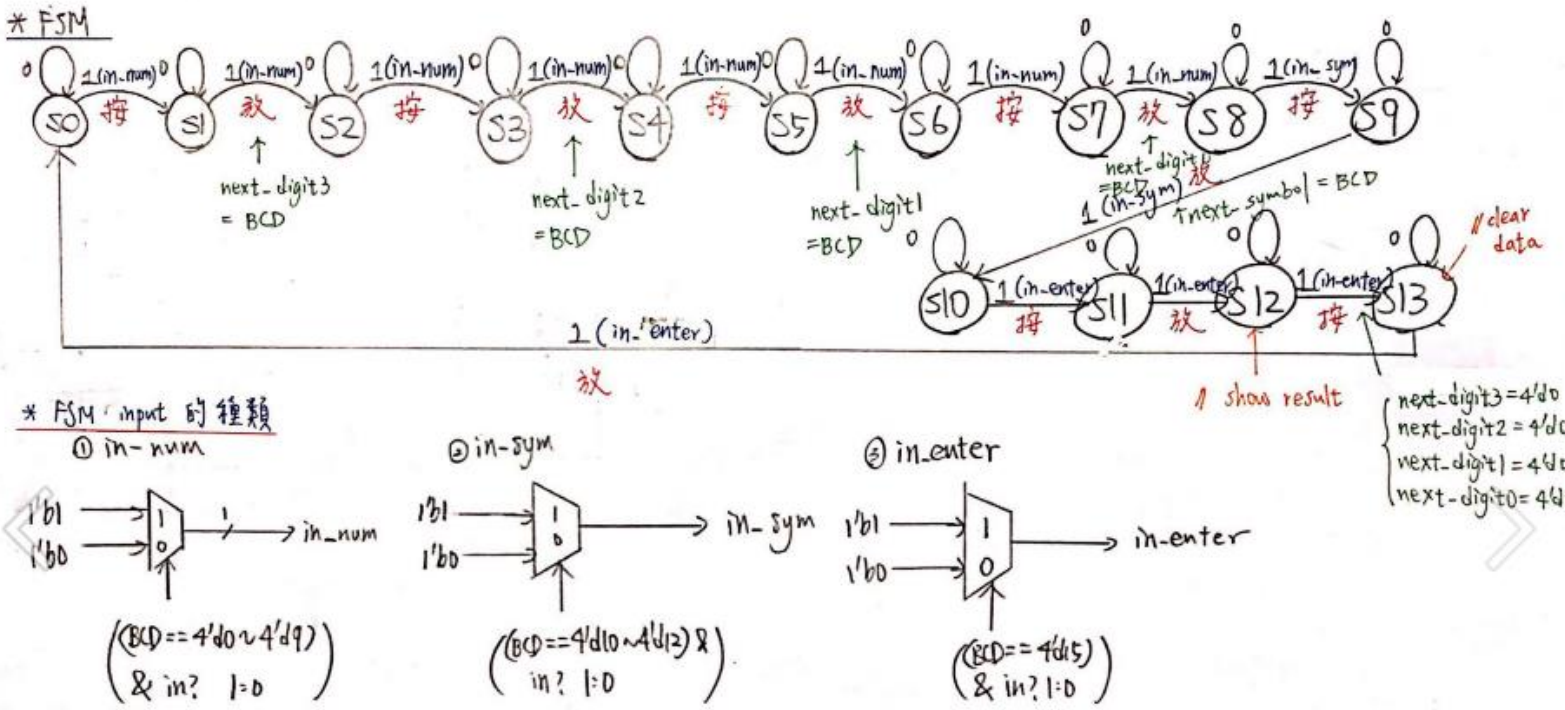
2.Logic Diagram(抱歉畫到兩張，沒辦法放一起，上下兩張是合在一起的)





↑ 上圖是整個系統的邏輯圖，上面那張的右邊接到下面那張的左邊，新的 module 將在下面介紹。

▲FSM



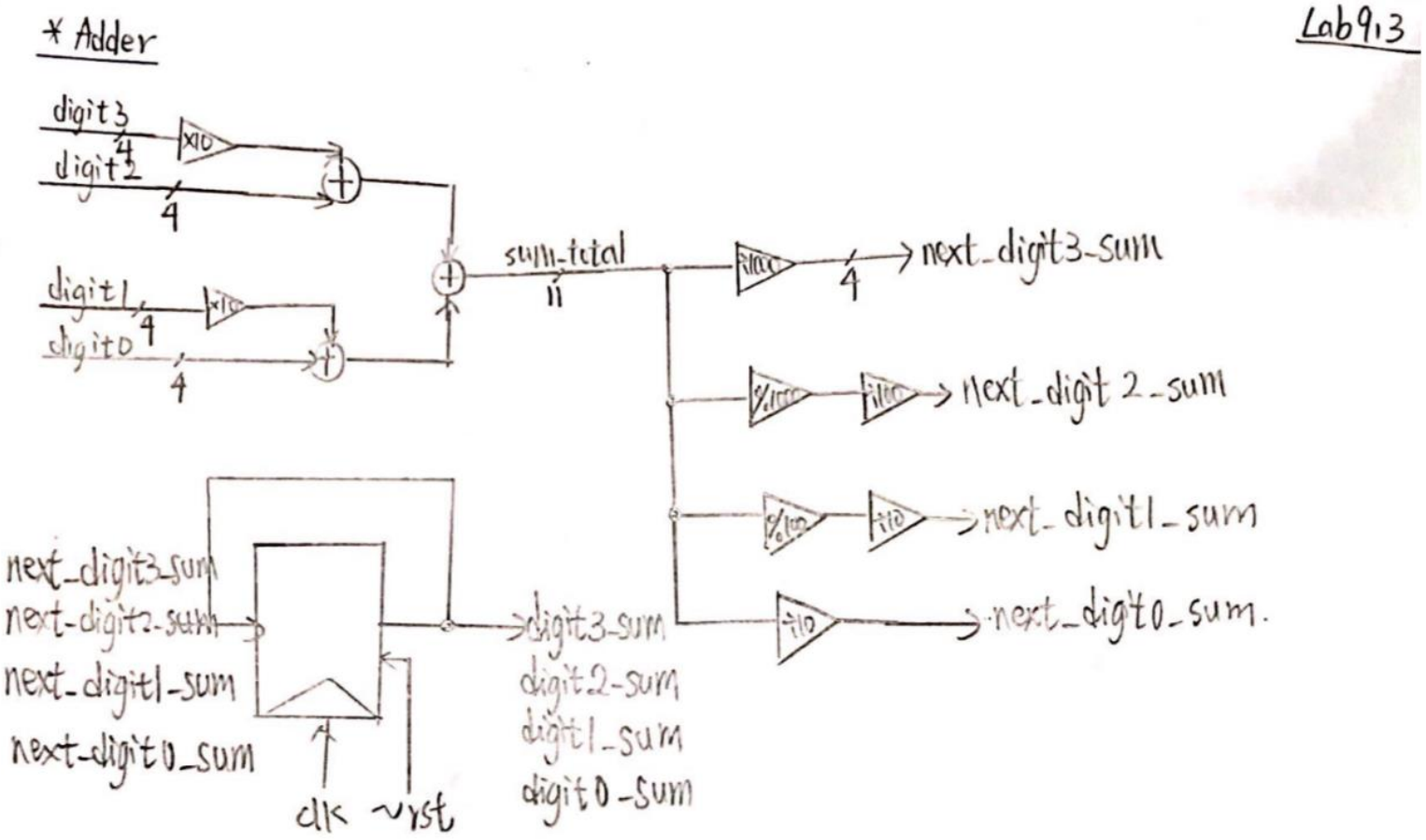
↑ 這次的 FSM 我用到 13 個 STATE，可能還可以簡化，但是我覺得這樣比較直接。

因為每一個 STATE 的功能不一樣，所以所需要的 input 訊號也不一樣，為了確保能夠正確輸入，所以我設計了一個防呆的設計，把 key_valid 分別和 num(數字)、sym(加減乘法)、enter and 在一起，總共三種不同的 input 訊號，確保在該輸入數字的時候只有按到數字的按鍵才有效果，其他的沒有，可以有效避免誤觸，邏輯圖在上圖下方的 input 的種類。每個 state 需要的 input trigger 也都標示在邏輯圖上。

另外還有一點蠻重要的是，跟 9.2 不一樣的地方。我把輸入 BCD 到 next_digit 的時刻放在放開按鈕的時候。因為如果像之前一樣，把 BCD 輸入到 next_digit 的時刻放在持續比較久的 S2、S4……state 的話，會讓 BCD 一直瘋狂輸入到 next_digit 裡面，並且當你在輸入該位數字的時候，雖然會成功輸入，但是在你輸入下一個數字的時候，原本那位數字也會被改變。

還有在最後的 S12、S13 功能是要讓所有變數歸零，可以重複使用這個功能。

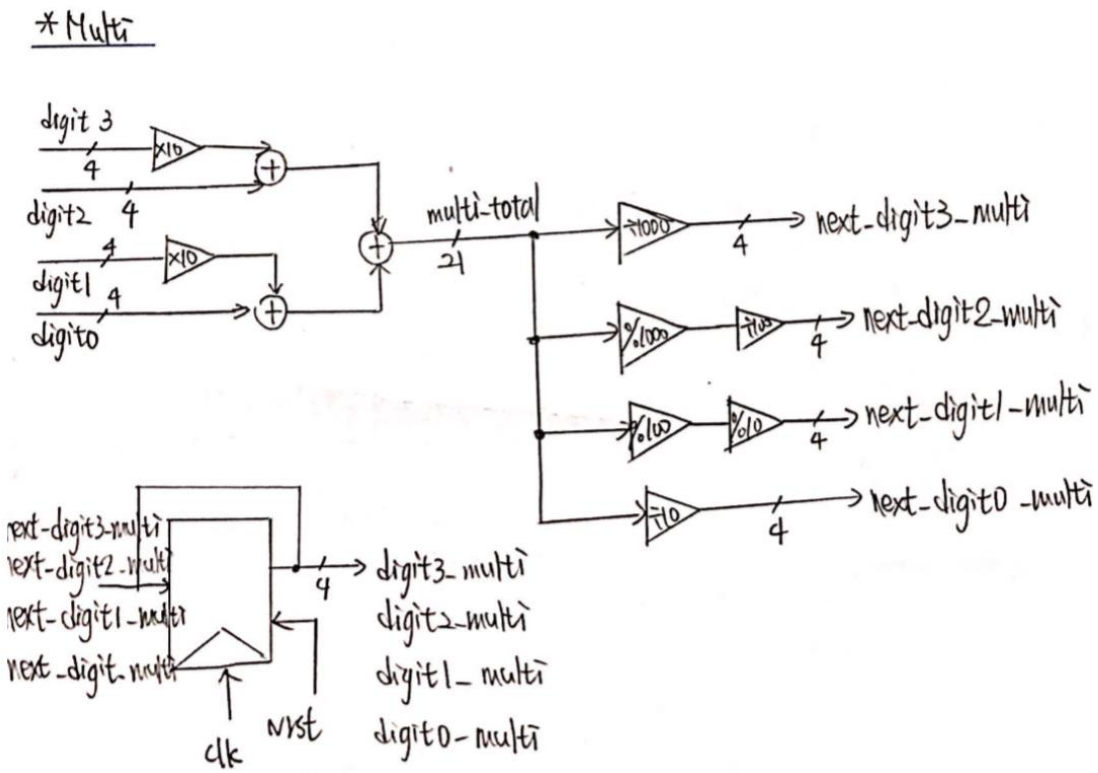
▲Adder



↑ 這次的 adder 需要計算兩位數加上兩位數

主要是利用一些簡單的乘法跟取於數的計算來取需要的結果，並用四個 D flip flop 儲存結果。

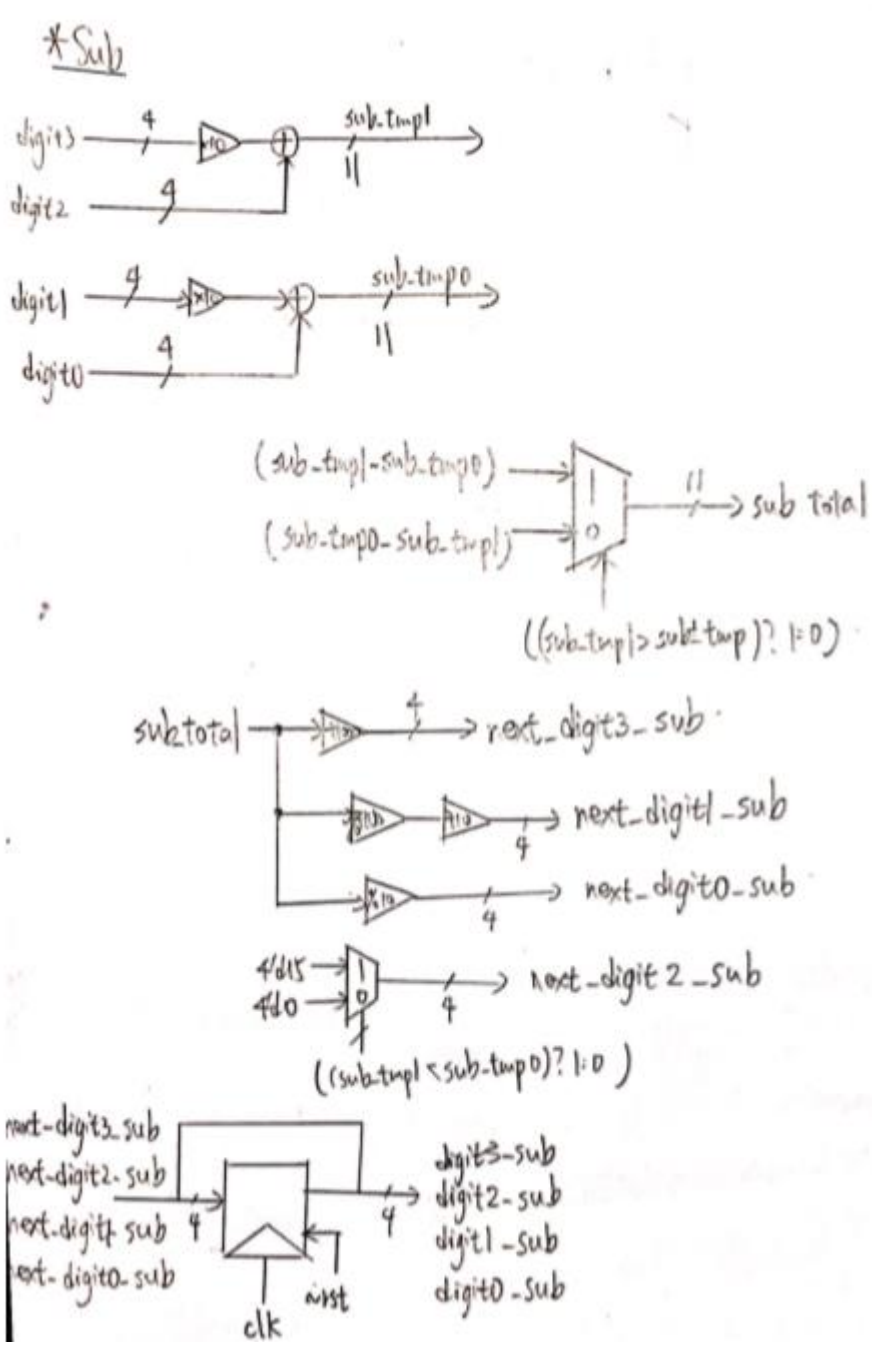
▲Multi



↑ 這次的 Multi 需要計算兩位數乘上兩位數

主要是利用一些簡單的乘法跟取於數的計算來取需要的結果，並用四個 D flip flop 儲存結果。

▲Sub



↑ Sub 需要計算兩位數減上兩位數

跟上面兩個 module 不同的地方是減法會有負數的問題，所以我另外設了一個 mux 來確保減出來的結果不會是負數，在最後面 next_digit2_sub 的地方再利用一個判別式來判斷結果為正為負，如果是負數的話就輸出 4’d15，經過 Display，最後會顯示出 “-“。

↓以下是 I/O 接腳

I/O	segs [7]	segs [6]	segs [5]	segs [4]	segs [3]	segs [2]	segs [1]	segs [0]	Clk	Rst
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	R2

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

I/O	PS2_CLK	PS2_DATA	State[3]	State[2]	State[1]	State[0]
Pin	C17	B17	L1	P1	N3	P3

●Discussion

這次的 lab 我花了比較大的心力在 fsm 上面，因為要非常清楚每個 state 的功能是什麼，如果放錯地方，顯示的結果就會是錯的，所以看到錯誤的 demo 結果要回推出自己程式碼哪裡出了問題也是非常重要的能力。

另外，我在 demo 的時候有聽到助教說盡量不要用除法當作計算符號，因為好像會用到很大量的資源，在之後如果更大的程式裡面會有問題，我想到比較好的方法應該是利用 loop 持續減除數，直到被除數被減到小於除數，loop 幾次就是原本/的結果，最後剩下的數字就是%的結果。

9-4. Display ASCII Code With Caps and shiftf

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst, (控制整個功能的開關)

Output:

[7:0]segs,（接上七段顯示器的七條燈)

[3:0]ssd_ctl, (接上四個七段顯示器，控制他們亮暗的頻率)

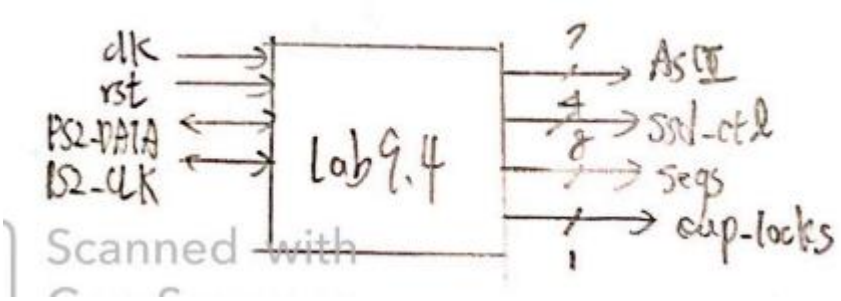
[6:0]ASCII (顯示 ASCII code 的結果在 LED 燈上面）

Caps_lock(用 LED 表示 caps 的狀態)

Inout:

PS2_CLK,(讓 PS2, KEYBOARD 彼此溝通用)

PS2_DATA(讓 PS2, KEYBOARD 彼此溝通用)

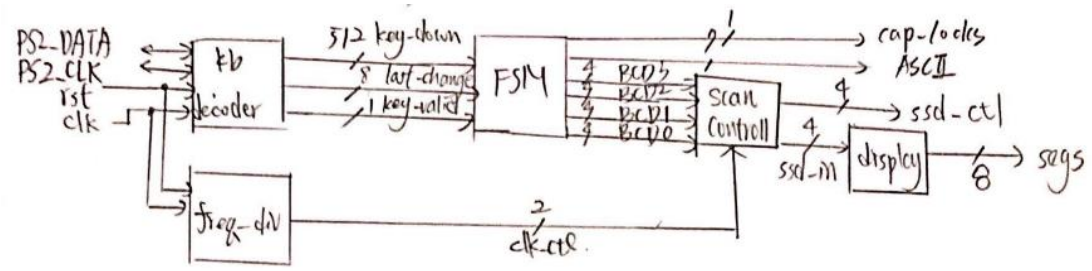


●Design Implementation

1.Outline

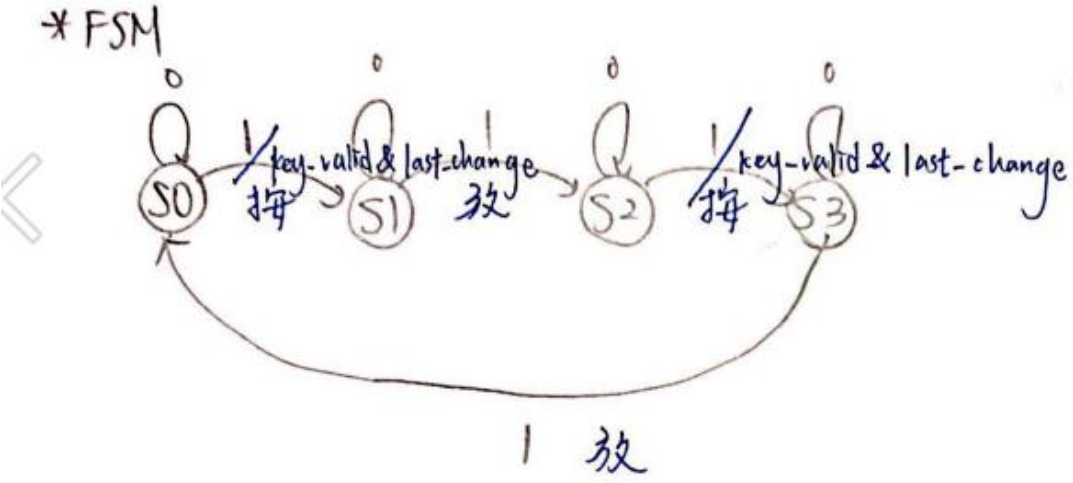
這個 lab 主要是要利用到 key_down 做出組合鍵的功能，還有 caps 的大小寫功能，按著 shift 再去按其他鍵的時候要顯示出另外一種狀態的 ASCII code。

2.Logic Diagram



↑ 上圖是整個系統的邏輯圖，新的 module 將在下面介紹。

▲FSM



↑ 這次 FSM 的 input 變成 key_valid&&(last_change == caps)，也就是說要按到 caps 鍵才會改變狀態

表示狀態的 led 燈亮的時候代表小寫，反之大寫。值得一提的有 shift 多功能按鍵，我是利用一個 XOR 來當作判斷式，如下。

if(caps_lock ^ key_down [18] == 1'b1) ←key_down[18]正是 shift 鍵的位置。

如果兩個異號的時候會顯示出小寫，反之則是大寫。

以下是 I/O 接腳

I/O	segs [7]	segs [6]	segs [5]	segs [4]	segs [3]	segs [2]	segs [1]	segs [0]	Clk	Rst
Pin	V14	U14	U15	W18	V19	U19	E19	U16	W5	R2

I/O	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
Pin	W4	V4	U4	U2

I/O	PS2_CLK	PS2_DATA	Caps
Pin	C17	B17	L1

I/O	ASCII[6]	ASCII[5]	ASCII[4]	ASCII[3]	ASCII[2]	ASCII[1]	ASCII[0]
Pin	U14	U15	W18	V19	U19	E19	U16

●Discussion

最後一題因為經過前面的洗禮，比較知道怎麼去運用那些變數，所以花的時間比較少，另外為了 debug 方便，我直接顯示 ASCII code 在 Led 上面，比較容易找出問題。

●Conclusion

做完這次的實驗，讓我學會如何使用 Keyboard 的使用，可以拿一隻鍵盤回宿舍真的有莫名的爽快感(? 不過我有好奇看了一下 KEYBOARD 的 MODULE，結果發現真的超複雜真的都看不懂，難怪各種實驗或是發明都要經過非常久的研究，立足在前人的貢獻上面，一代一代再往前進一點點，突然有一種自己很苗小的感覺……