**Lab8 Report**



**Ⅰ. Lab8\_1 (Implement Key Board)**

**Design Specification**

IO:

Input: clk, rst.

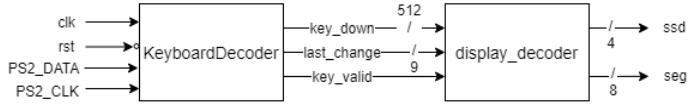
Inout: PS2\_DATA, PS2\_CLK.

Output: [7:0] segs, [3:0] ssd\_ctl.

**Design Implementation**

Block diagram:

本次實驗需要使用以下兩個 module 功能，分別為 KeyboardDecoder 以及display\_decoder。









**KeyboardDecoder**：

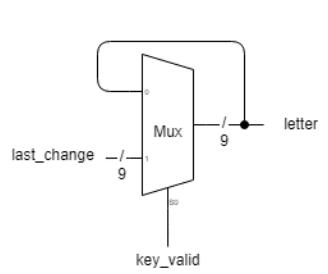
此module將鍵盤訊號轉換成可以得知哪些按鍵目前被按著的訊號(key\_down)、最近一個被操作的按鍵訊號(last\_change)以及按下或放開任一按鍵時的通知訊號(key\_valid)。

**display\_decoder**：

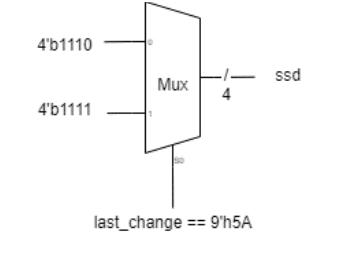
用七段顯示器顯示最近一個被按下的按鍵，包括 1〜9 和A、S、M；當按下 enter 鍵則刷新七段顯示器。

Logic diagram:

display\_decoder: 用七段顯示器顯示最近一個被按下的按鍵，包括1〜9和A、S、M；當按下enter 鍵則刷新七段顯示器。





當出現key\_valid時，代表有按鍵被操作，因此讓letter = last\_change；沒有出現key\_valid，則代表沒有按鍵被操作，讓number維持不變。



當按下 enter 鍵時，也就是 last\_change = 9’h5A 時，關掉七段顯示器，達到重置頁面的效果。

解碼: 將number解碼並顯示在七段顯示器上。

number = 9’h70 → segs = 8’b0000\_0011 (顯示 0)

number = 9’h69 → segs = 8’b1001\_1111 (顯示 1)

number = 9’h72 → segs = 8’b0010\_0101 (顯示 2)

number = 9’h7A → segs = 8’b0000\_1101 (顯示 3)

number = 9’h6B → segs = 8’b1001\_1001 (顯示 4)

number = 9’h73 → segs = 8’b0100\_1001 (顯示 5)

number = 9’h74 → segs = 8’b0100\_0001 (顯示 6)

number = 9’h6C → segs = 8‘b0001\_1111 (顯示 7)

number = 9’h75 → segs = 8’b0000\_0001 (顯示 8)

number = 9’h7D → segs = 8’b0000\_1001 (顯示 9)

number = 9’h1C → segs = 8’b0000\_0101 (顯示 a)

number = 9’h1B → segs = 8’b1111\_1101 (顯示 - )

number = 9’h3A → segs = 8’b1001\_0001 (顯示 \*)

default 為 segs = 8’b 0111\_0001 (顯示 F)

結合兩個 module 功能，可以將鍵盤按下的數字 1〜9 以及 A/S/M 顯示在七段顯示器上。

I/O pin assignment:

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

| ssd[3] | ssd [2] | ssd [1] | ssd [0] | clk | rst | PS2\_CLK | PS2\_DATA |
| --- | --- | --- | --- | --- | --- | --- | --- |
| W4 | V4 | U4 | U2 | W5 | R2 | C17 | B17 |

**Ⅱ. Lab8\_2 (a single digit decimal adder)**

**Design Specification**

IO:

Input: clk, rst.

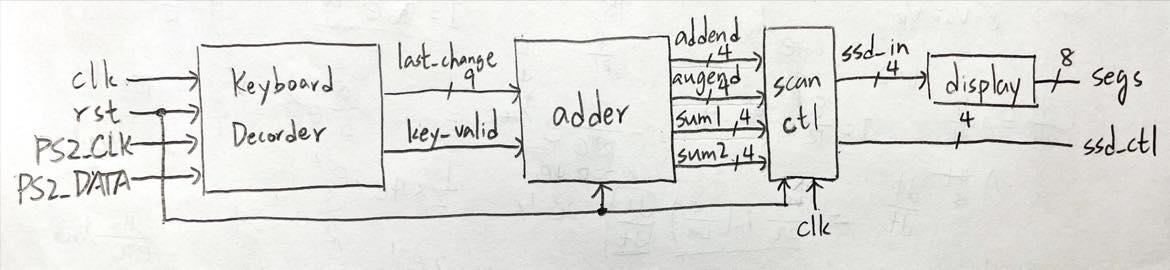
Inout: PS2\_DATA, PS2\_CLK.

Output: [7:0] segs, [3:0] ssd\_ctl.

**Design Implementation**

為了可讀性，我把整個project分成了四個部分：KeyboardDecoder, adder, scan\_control, display和top module，Block diagram如下圖所示。

A simple Block diagram:

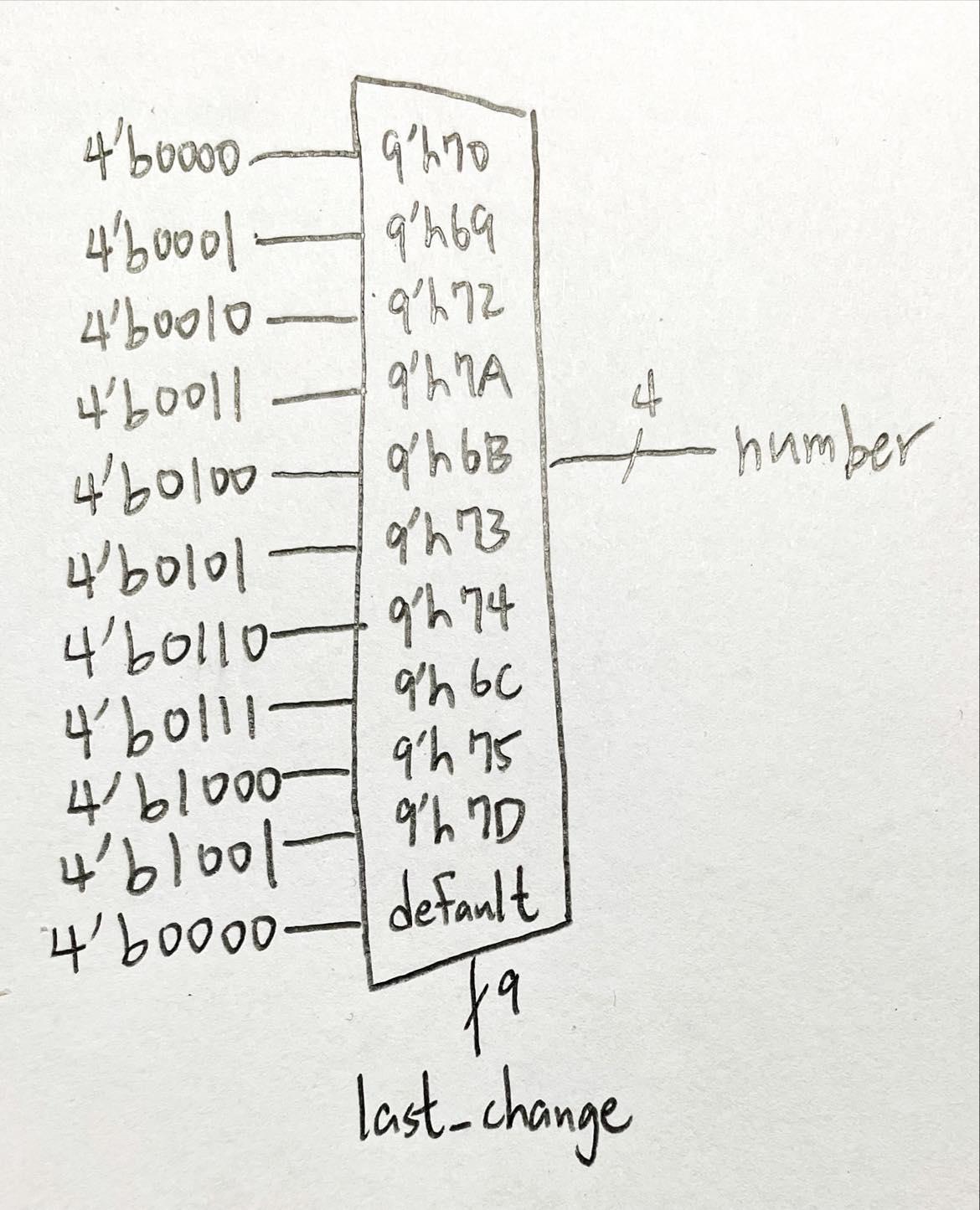


**KeyboardDecoder**：

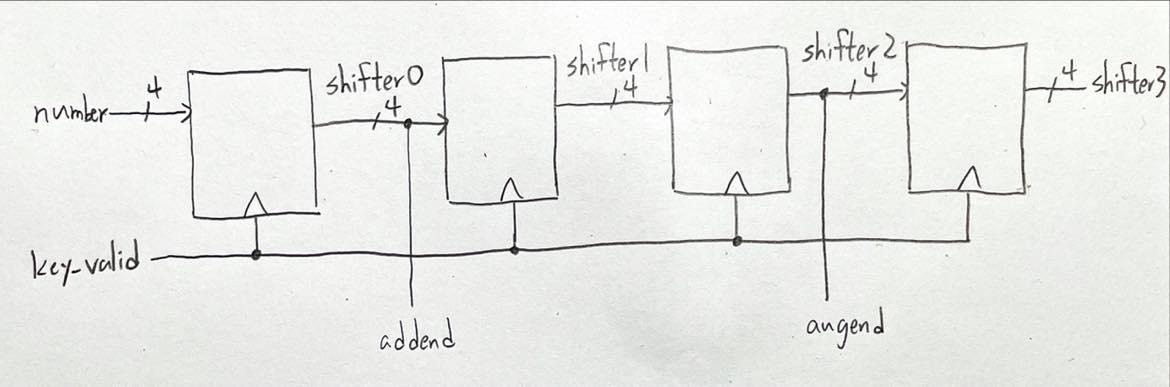
此module將鍵盤訊號轉換成可以得知哪些按鍵目前被按著的訊號(key\_down)、最近一個被操作的按鍵訊號(last\_change)以及按下或放開任一按鍵時的通知訊號(key\_valid)。

**adder**：

首先，我先用一個decoder去把鍵盤的訊號(last\_change)轉變成BCD的形式存入number，logic diagram如下圖所示：



接著，把number作為shifter的input，把key\_valid作為flip-flop的clock，當按下或放開按鍵時，shifter會往右推一位，因為按下和放開按鍵都會引發key\_valid，所以我使用四組flip-flops作為shifter，且輸出第一組flip-flop儲存的值作為addend，第三組flip-flop儲存的值作為augend，以下為shifter的logic diagram：



然後，考慮到兩個BCD的數字相加可能會進位的問題，我先預設會進位和不會進位兩種情況，把addend和augend變成8bits，再把兩種解的個位數都求出來，

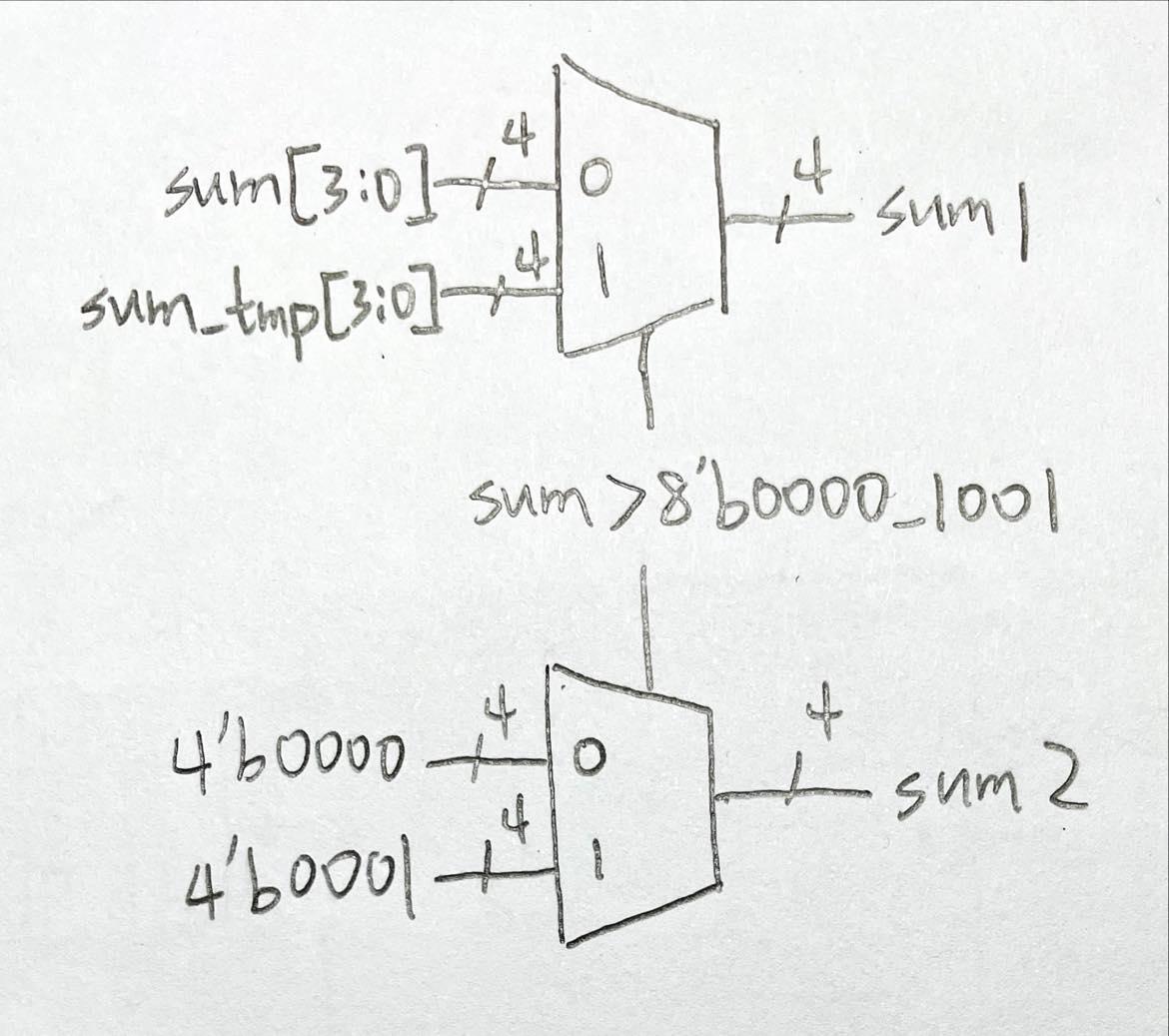
sum = {4’b0000,shifter2} + {4’b0000,shifter0}

sum\_tmp = {4’b0000,shifter2} + {4’b0000,shifter0} – 8’b0000\_1010

最後判斷情況為何者(sum>8'b0000\_1001)？

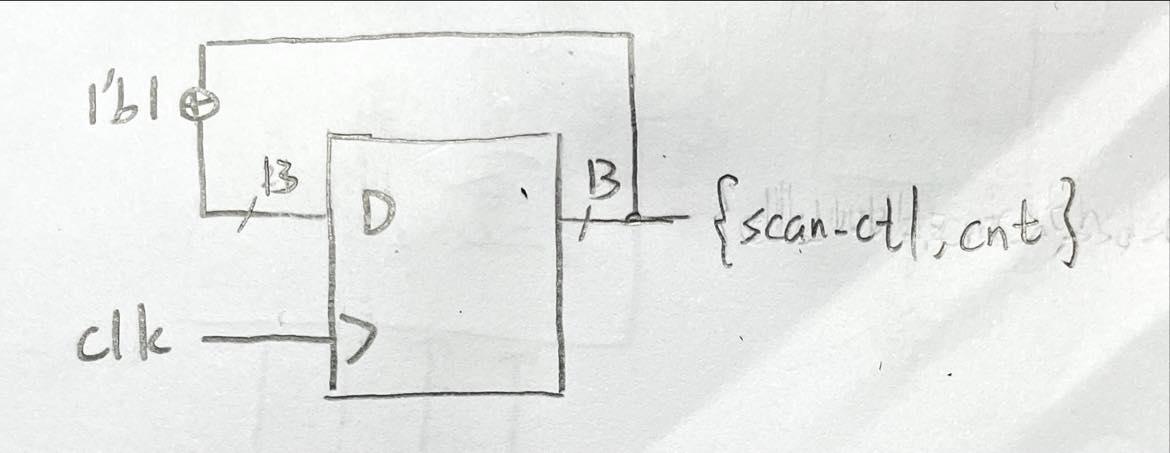
如果和大於9：sum1 = sum\_tmp[3:0], sum2 = 4’b0001

如果和小於9：sum1 = sum[3:0], sum2 = 4’b0000



**scan\_control**：

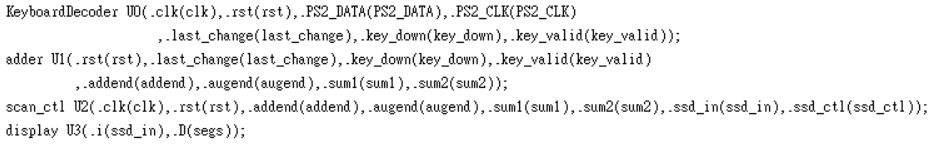
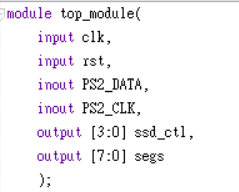
因為這題需要讓四個七段顯示器顯示不同的值，所以需要一個合理的頻率去變動七段顯示器的陽極和陰極，因此，我使用13bits的counter做一個frequency divider且輸出首兩個bits作為切換陽極和陰極的scan\_ctl，如下圖：



使用剛剛做出來的scan\_ctl作為MUX的條件去切換陽極與陰極。

**Display**：是一個decoder，把BCD的數字轉換為七段顯示器顯示該數字應為的陰極。

**Top module**：使用此module去連接各modules, inputs, outputs和signals。



I/O pin assignment:

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

| ssd[3] | ssd [2] | ssd [1] | ssd [0] | clk | rst | PS2\_CLK | PS2\_DATA |
| --- | --- | --- | --- | --- | --- | --- | --- |
| W4 | V4 | U4 | U2 | W5 | R2 | C17 | B17 |

**Ⅱ. Lab8\_3 (a two-digit decimal adder/subtractor/multiplier)**

**Design Specification**

IO:

Input: clk, rst.

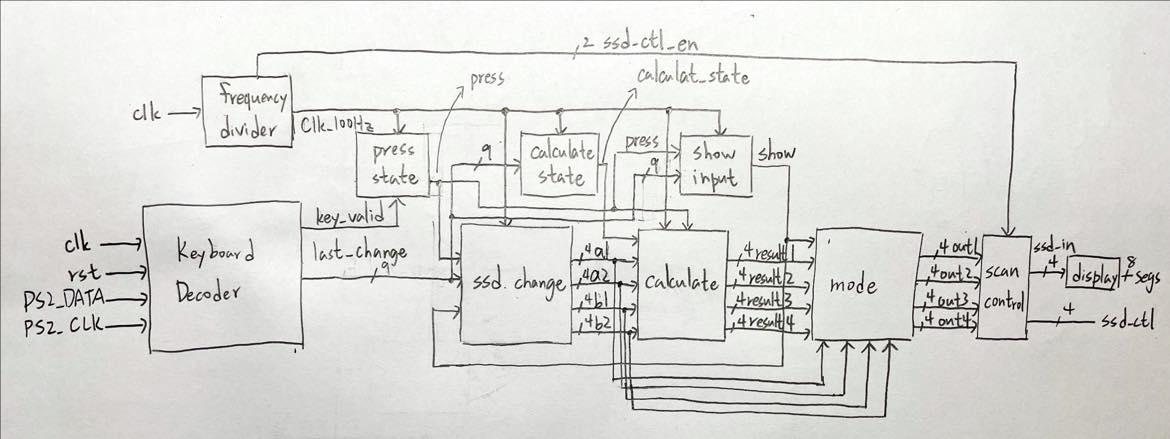
Inout: PS2\_DATA, PS2\_CLK.

Output: [7:0] segs, [3:0] ssd\_ctl, [3:0] LED.

**Design Implementation**

為了可讀性，我把整個project分成了十一個部分：KeyboardDecoder, frequency divider, press state, calculate state, show input, ssd change, calculate, mode, scan\_control, display和top module，Block diagram如下圖所示。

A simple Block diagram:

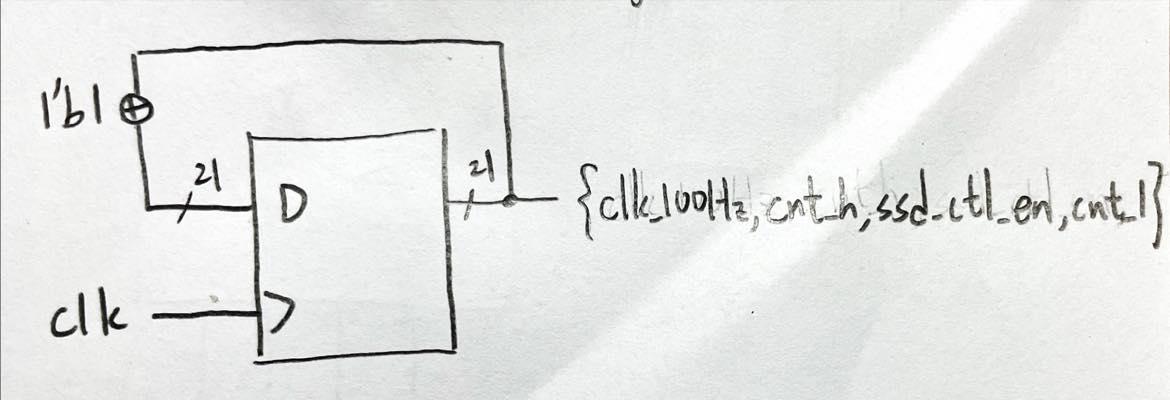


**KeyboardDecoder**：

此module將鍵盤訊號轉換成可以得知哪些按鍵目前被按著的訊號(key\_down)、最近一個被操作的按鍵訊號(last\_change)以及按下或放開任一按鍵時的通知訊號(key\_valid)。

**Frequency divider**：

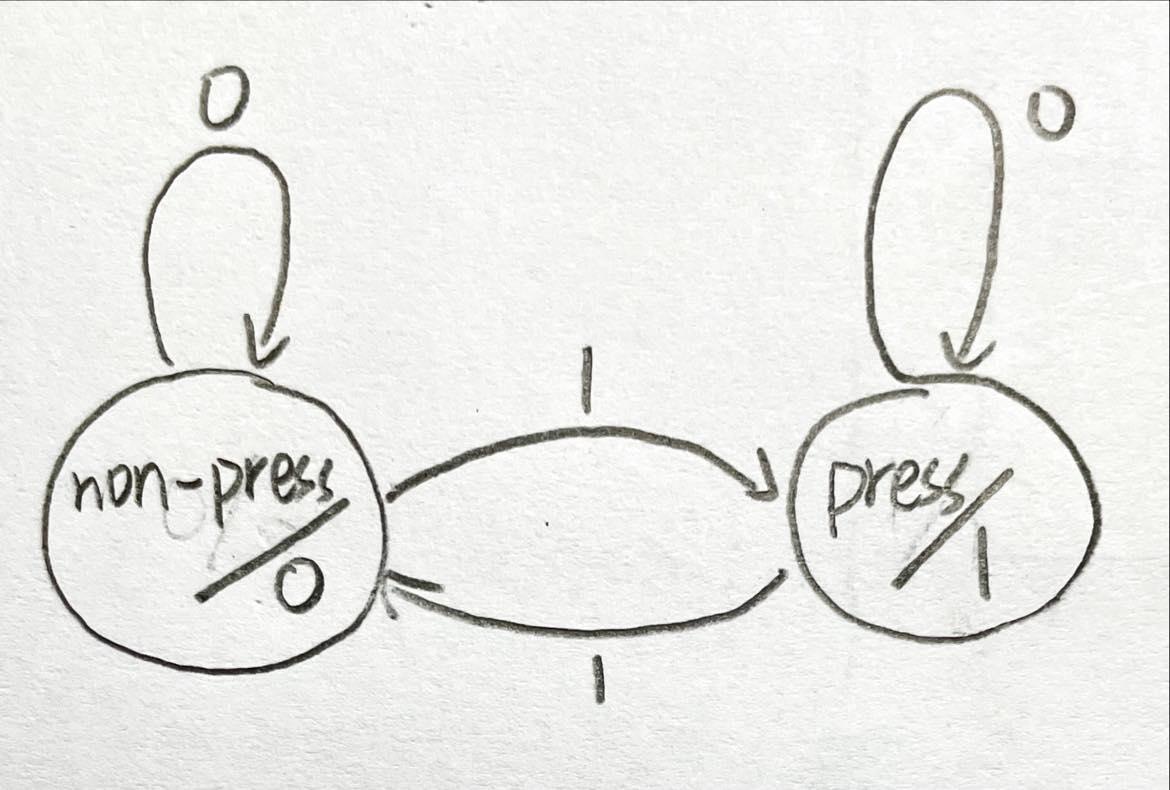
因為這題會需要許多種FSM去決定現在的狀態，所以需要除出FSM需要的頻率(100Hz)，另外，因為要在四個七段顯示器上顯示不同的數字，所以還需要一個enable去控制scan control。因此，我建構一個counter用原本的clock(100MHz)去數，取 bit為100Hz的clock()，取作為scan control的enable。



**Press stste**：

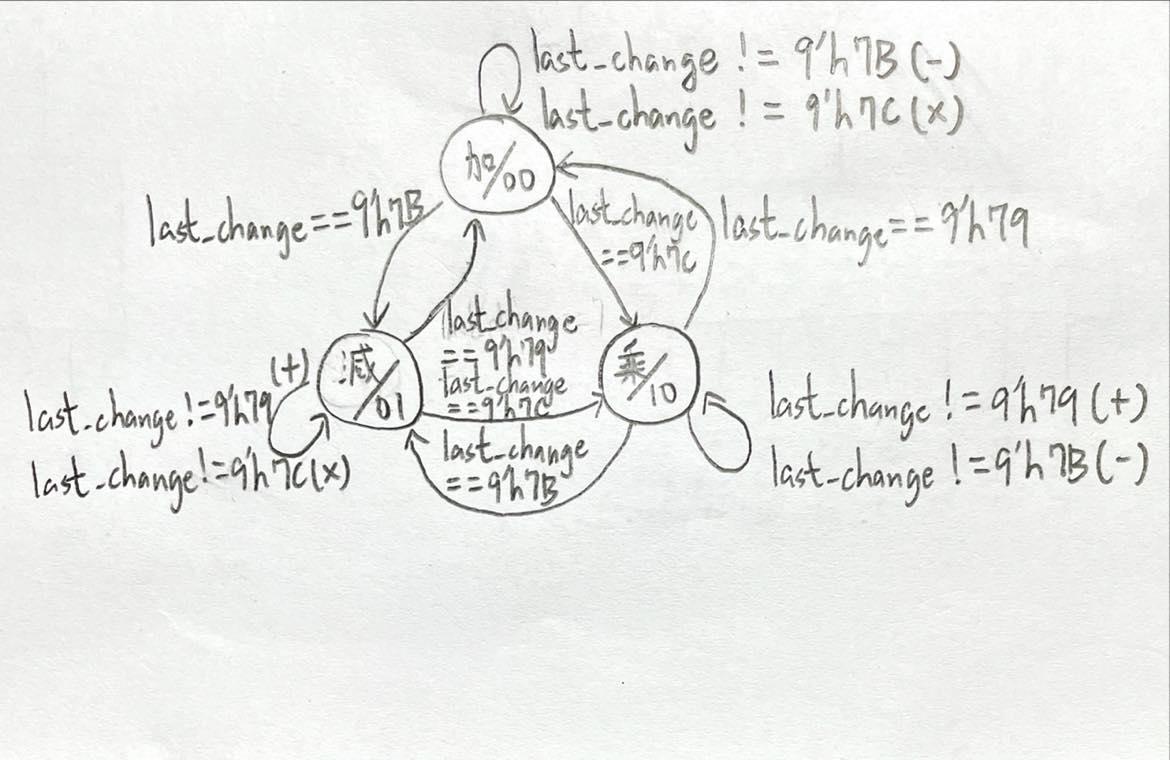
因為我覺得key\_valid在操作上不是那麼方便，所以我利用一個FSM創造出另一個新的訊號，當出現key\_valid時會轉變一次state，且原本的state設為0，因此，我只要確保每次都只按下一個按鍵，如果這個訊號是0代表我現在沒按下任何按鍵，如果訊號是1代表我按著某個按鍵，而且我只需要判斷這個訊號的posedge和negedge就可以知道現在我是放開按鍵還是按下按鍵，所以我寫了這個module去產生這個叫press的訊號。

Moore machine:



**Calculate state**：

這個部分是判斷目前的運算是加、減或乘，我一樣利用了FSM(Moore machine)去實現這個module，我先用一個MUX去判斷我按下的按鍵是加、減還是乘(last\_change)，我設加的state為00、減為01、乘為10，用MUX讓next\_state等於目前的運算符號。下方是state diagram：

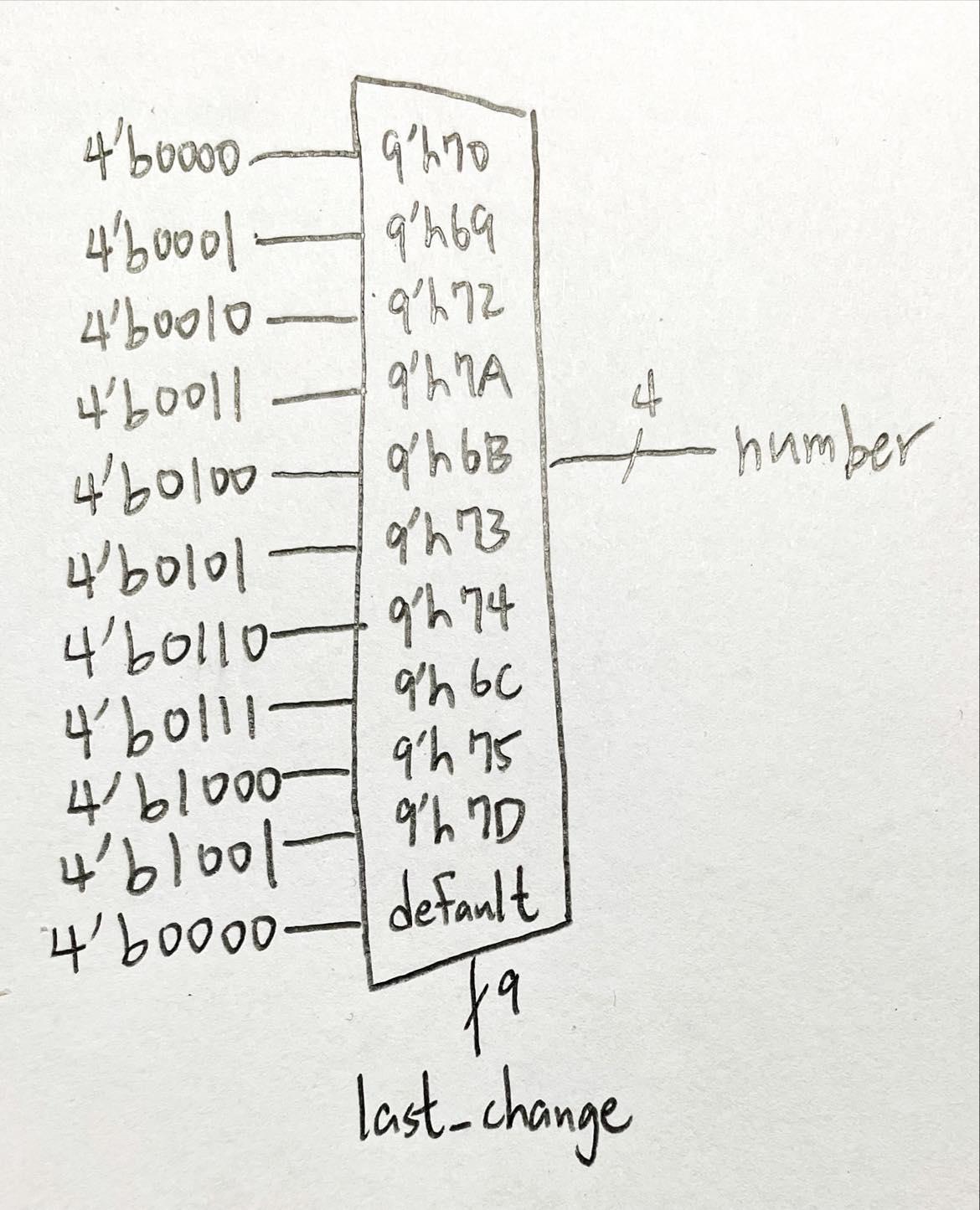


**Show input**：

因為在這題要求下，我們按Enter需要顯示計算出來的答案，而在此之前我們要先輸入兩個數字並顯示在顯示器上，所以我就把Enter當作是一個切換這兩種模式的開關，我做一個Moore machine當按下Enter切換一次state且輸出叫做show，如果show = 0時，代表現在是輸入數字的時候，可以任意的更改輸入值；如果show = 1時，代表現在要顯示目前的答案，而按下其他數字鍵不會更改到內部的輸入值。

**Ssd change**：

首先，我先用一個decoder去把鍵盤的訊號(last\_change)轉變成BCD的形式存入number，logic diagram如下圖所示：



接著，因為有四個七段顯示器的值需要輸入，所以我把它們都各別視為一個state，從左到右分別是00、01、10、11，只要我按下任何數字鍵，七段顯示器的state就加一，最後我只需要判斷目前的state，把剛剛decoder解碼出來的number輸進當前state所代表的七段顯示器裡，而其餘的七段顯示器只須保持原來的值，這樣就可做出輸入變數進四個七段顯示器的效果。但是，因為我要去切換四個七段顯示器又要去給他們賦值，我的想法是先賦值再切換顯示器，所以我切換顯示器的時機是press state所產生的訊號的negedge，這樣就不會發生賦值在我不預期的顯示器上了。

**calculate**：

這個mode是利用calculate state所生成的state(運算符號)去執行ssd change所得的兩個兩位數的數字(a1、a2、b1、b2)加、減或乘的運算，下方是個運算的詳細過程：

加(+)：

因為兩位數的數字相加不可能會加到千位，所以我千位數就直接設成4’b0000；而百位數則是把 因為verilog裡的除法只會去到整數位，所以這樣就得到了百位數；十位數也是一樣的道理把，因為除法只會留下整數，所以減掉剛剛的百位數乘100就可以去除百位的數字，再除10就可以得到十位數了；而個位數也是用相同的方法，把兩數相加減掉百位數乘100再減掉十位數乘10就是個位數了。

減(-)：

減法我是先判斷兩數a、b的大小，如果a >= b，那就把a-b；如果a < b，那就把b-a，但因為還要考慮借位的問題，要去觀察a2與b2的大小關係，再決定要不要借位，而且因為是減法所以千位和百位數一定是4'b0000，但如果a < b，那我就在原本百位數的地方加一個負號，我把負號寫成4'b1010，最後在display時再把它decode成負號顯示

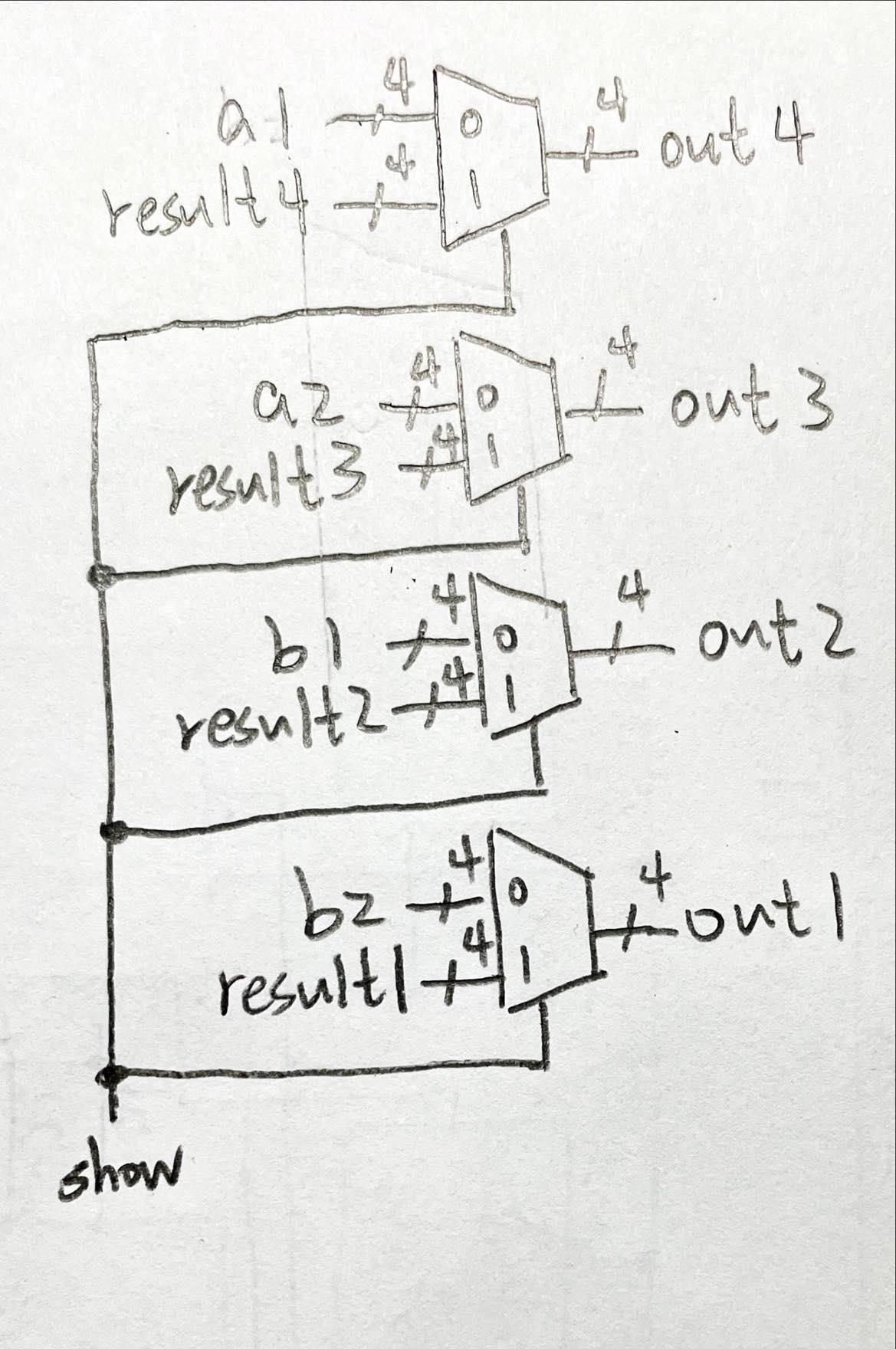
在七段顯示器上。

乘(\*)：

在乘法時，我利用類似加法的方法，千位數等於；百位數等於；十位數等於；個位數等於

**mode**：

決定最後是輸出計算出的答案還是輸入的兩個變數，以show判斷用MUX實現。

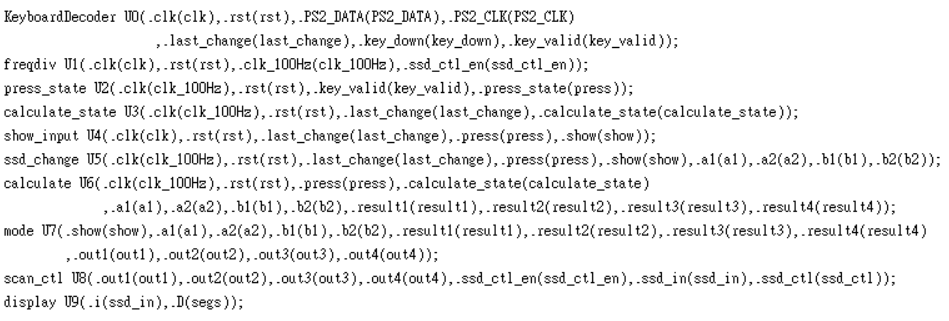
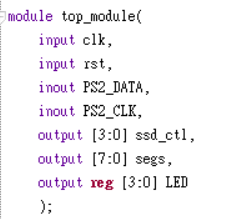


**scan\_control**：

我利用frequency divider做出兩個bits的ssd\_ctl\_en作為切換陽極和陰極的enable， 使用做出來的scan\_ctl\_en作為MUX的條件去切換陽極與陰極。

**Display**：是一個decoder，把BCD的數字轉換為七段顯示器顯示該數字或符號應為的陰極。

**Top module**：使用此module去連接各modules, inputs, outputs和signals，利用LED來顯示現在是加法、減法還是乘法，還有現在是顯示答案還是顯示輸入值。



**IO pin assignment：**

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

| ssd[3] | ssd [2] | ssd [1] | ssd [0] | clk | rst | PS2\_CLK | PS2\_DATA |
| --- | --- | --- | --- | --- | --- | --- | --- |
| W4 | V4 | U4 | U2 | W5 | R2 | C17 | B17 |

| LED[3] | LED[2] | LED[1] | LED[0] |
| --- | --- | --- | --- |
| V19 | U19 | E19 | U16 |

**Ⅱ. Lab8\_4 (Implement the “Caps” control in the keyboard)**

**Design Specification**

IO:

Input: clk, rst.

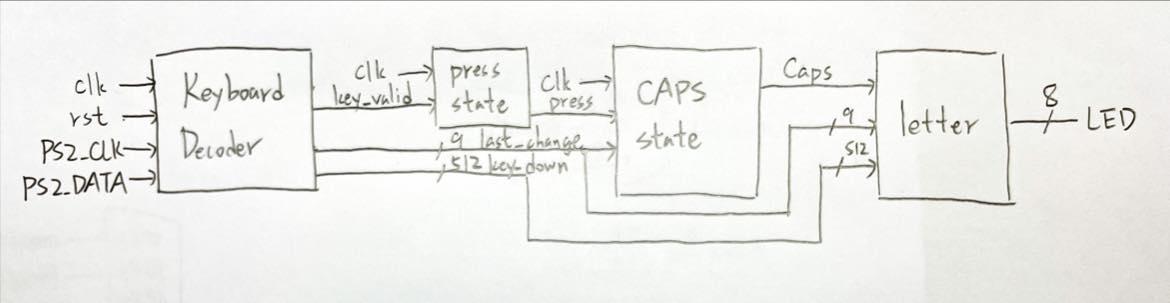
Inout: PS2\_DATA, PS2\_CLK.

Output: [7:0] LED.

**Design Implementation**

為了可讀性，我把整個project分成了五個部分：KeyboardDecoder, press state, CAPS state, letter和top module，Block diagram如下圖所示。

A simple Block diagram:



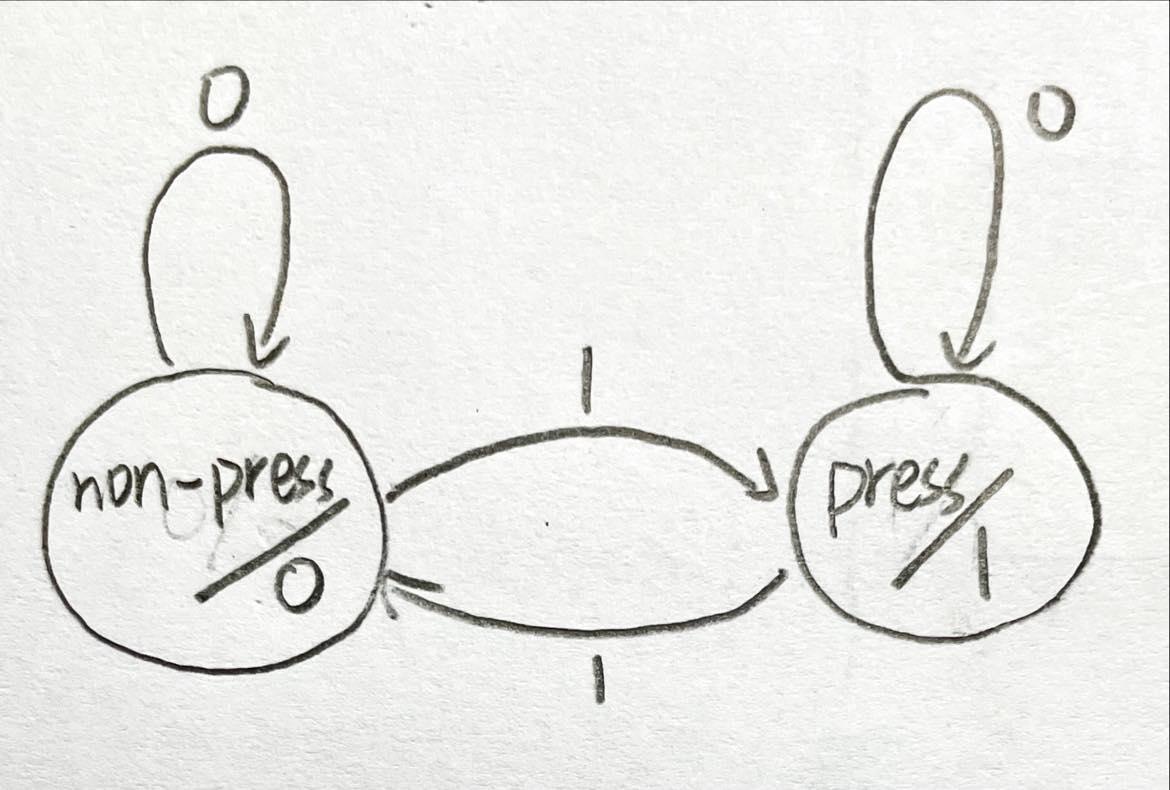
**KeyboardDecoder**：

此module將鍵盤訊號轉換成可以得知哪些按鍵目前被按著的訊號(key\_down)、最近一個被操作的按鍵訊號(last\_change)以及按下或放開任一按鍵時的通知訊號(key\_valid)。

**Press stste**：

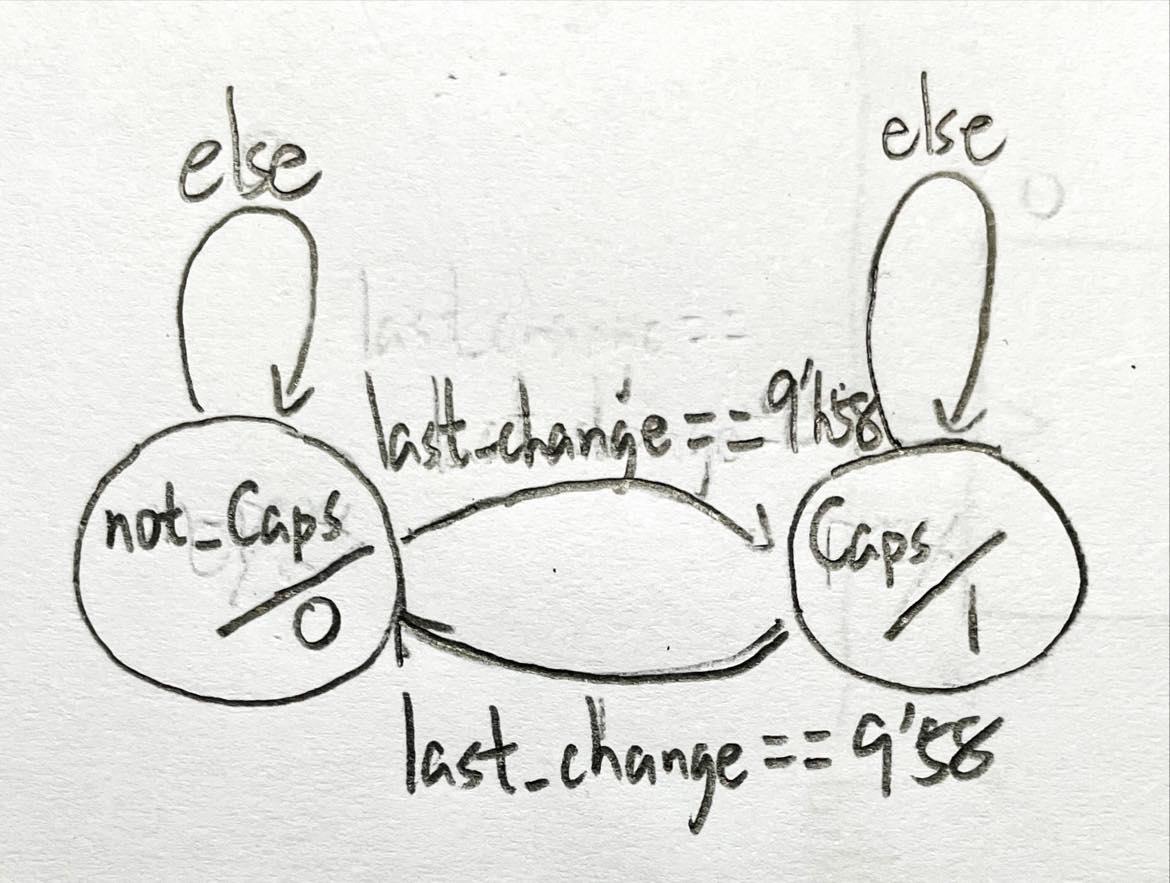
因為我覺得key\_valid在操作上不是那麼方便，所以我利用一個FSM創造出另一個新的訊號，當出現key\_valid時會轉變一次state，且原本的state設為0，因此，我只要確保每次都只按下一個按鍵，如果這個訊號是0代表我現在沒按下任何按鍵，如果訊號是1代表我按著某個按鍵，而且我只需要判斷這個訊號的posedge和negedge就可以知道現在我是放開按鍵還是按下按鍵，所以我寫了這個module去產生這個叫press的訊號。

Moore machine:



**CAPS state**：

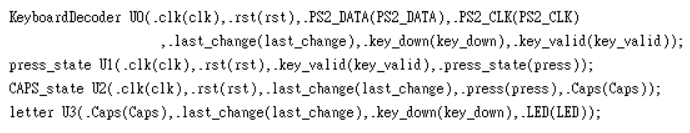
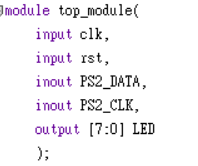
這個部分是一個FSM(moore machine)去產出Caps的訊號，如果Caps = 0，代表現在按的字母都是小寫；如果Caps = 1，代表現在按的字母是大寫。首先判斷當時的Caps，如果按下了鍵盤上的CAPS按鍵就轉變一次Caps的值，因為只需轉變一次，所以我去偵測negedge press，因此，如果現在偵測到press的negedge，按下的按鍵又是CAPS(last\_change = 9’h58)，那就轉換一次Caps的值。



**letter**：

這主要是一個decoder去把鍵盤的last\_change轉為ASCII code的編碼，在按下字母按鍵時，會去檢查shift有沒有被按下(key\_down[9'h12], key\_down[9'h59])還有現在Caps的狀態，原本decoder出來的值是大寫字母的ASCII code，所以如果判斷出來為小寫要再把原本的ASCII code加32轉為小寫的ASCII code。最後直接把結果賦值給LED，顯示在FPGA板上。

**Top module**：使用此module去連接各modules, inputs, outputs和signals。



**IO pin assignment：**

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

| Caps | PS2\_CLK | PS2\_DATA | clk | rst |
| --- | --- | --- | --- | --- |
| L1 | C17 | B17 | W5 | R2 |

**Discussion**

這次實驗我花了很多時間在研究老師的鍵盤訊號的真正用法，另外，我在第三題的時候出現了時間的問題，我輸入的值會在ssd 轉換了之後才賦值，這件事原本讓我很困擾，但我最後利用了posedge和negedge的時間差去解決了這個問題，第一次遇到時間的問題，讓我想起教授上課時常提醒clock對齊的問題，也了解到控制各module時間的重要。

**Conclusion**

這次實驗第一次接上鍵盤當作輸入訊號，雖然老師有提供鍵盤訊號的解碼器，但因為還不熟悉解碼後的訊號的概念，因此我也是一邊作實驗一邊摸索，所以在有些小地方還是花了不少時間，不過題目本身除了第三題的 FSM 比較複雜之外，其他都還算沒那麼難處理。這次實驗學到了怎麼把鍵盤的訊號當作輸入訊號來處理，並輸出結果到有顯示功能的版面上，這對我之後要做的 final project 也有相當大的幫助。