(1) Implement Key Board

1.1 Press 0/1/2/3/4/5/6/7/8/9 and show them in the seven-segment display. When a new number is pressed, the previous number is refreshed and over written.

1.2 Press a/s/m (addition/subtraction/multiplication) and show them in the sevensegment display as your own defined A/S/M pattern. When you press "Enter", refresh (turn off) the seven-segment display.

IO:

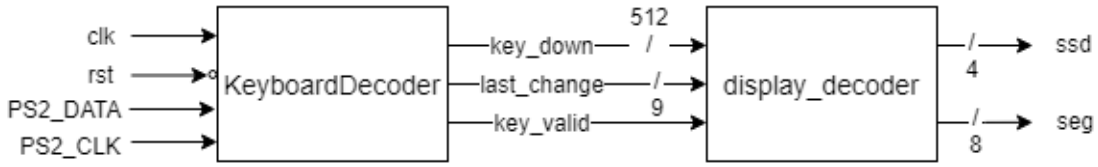
Inout: PS2_DATA, PS2_CLK

Input: rst, clk

Output: [7:0]seg, [3:0]ssd

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。

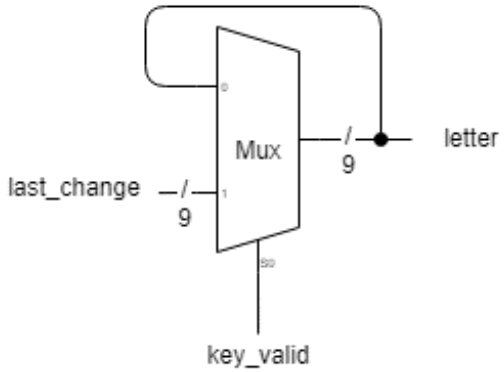


1. KeyboardDecoder: 此 module 將鍵盤訊號轉換成可以得知哪些按鍵目前被按著的訊號(keydown)、最近一個被操作的按鍵訊號(last_change)以及按下或放開任一按鍵時的通知訊號(key_valid)。

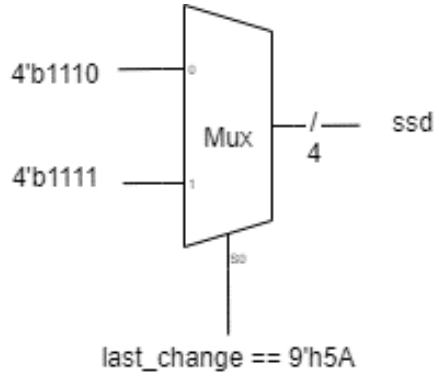
2. display_decoder: 用七段顯示器顯示最近一個被按下的按鍵，包括 1～9 和 A、S、M；當按下 enter 鍵則刷新七段顯示器。

Logic diagram:

<u>display_decoder:</u> 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 letter 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。

<u>解碼:</u> 將 letter 解碼並顯示在七段顯示器上。

letter = 9'h70 → seg = 8'b0000_0011 (顯示 0)
letter = 9'h69 → seg = 8'b1001_1111 (顯示 1)
letter = 9'h72 → seg = 8'b0010_0101 (顯示 2)
letter = 9'h7A → seg = 8'b0000_1101 (顯示 3)
letter = 9'h6B → seg = 8'b1001_1001 (顯示 4)
letter = 9'h73 → seg = 8'b0100_1001 (顯示 5)
letter = 9'h74 → seg = 8'b0100_0001 (顯示 6)
letter = 9'h6C → seg = 8'b0001_1111 (顯示 7)

letter = 9'h75 → seg = 8'b0000_0001 (顯示 8)
letter = 9'h7D → seg = 8'b0000_1001 (顯示 9)
letter = 9'h1C → seg = 8'b 0000_0101 (顯示 a)
letter = 9'h1B → seg = 8'b 1111_1101 (顯示 -)
letter = 9'h3A → seg = 8'b 1001_0001 (顯示 *)
default 為 seg = 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
W4	V4	U4	U2	W5	R2

PS2_CLK	PS2_DATA
C17	B17

(2) Implement a single digit decimal adder using the keyboard as the input and display the results on the 14-segment display (The first two digit are the addend/augend, and the last two digits are the sum).

IO:

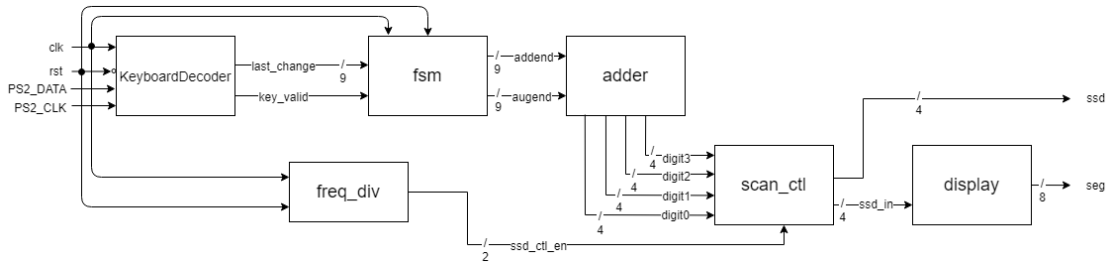
Inout: PS2_DATA, PS2_CLK

Input: rst, clk

Output: [7:0]seg, [3:0]ssd

Block diagram:

本次實驗需要使用以下 module 功能，分別為 KeyboardDecoder、fsm、加法器(adder)、除頻器(freq_div)、scan control(scan_ctl)以及七段顯示解碼器(display)。



1. 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(keydown)、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)。

2. fsm: 透過當前狀態判斷要將按鍵資料存到哪個變數裡。

3. adder: 先將資料解碼後再將被加數和加數相加後得到總和，之後轉換成 4 個 digit 的 BCD 形式輸出給 scan control 當作輸入。

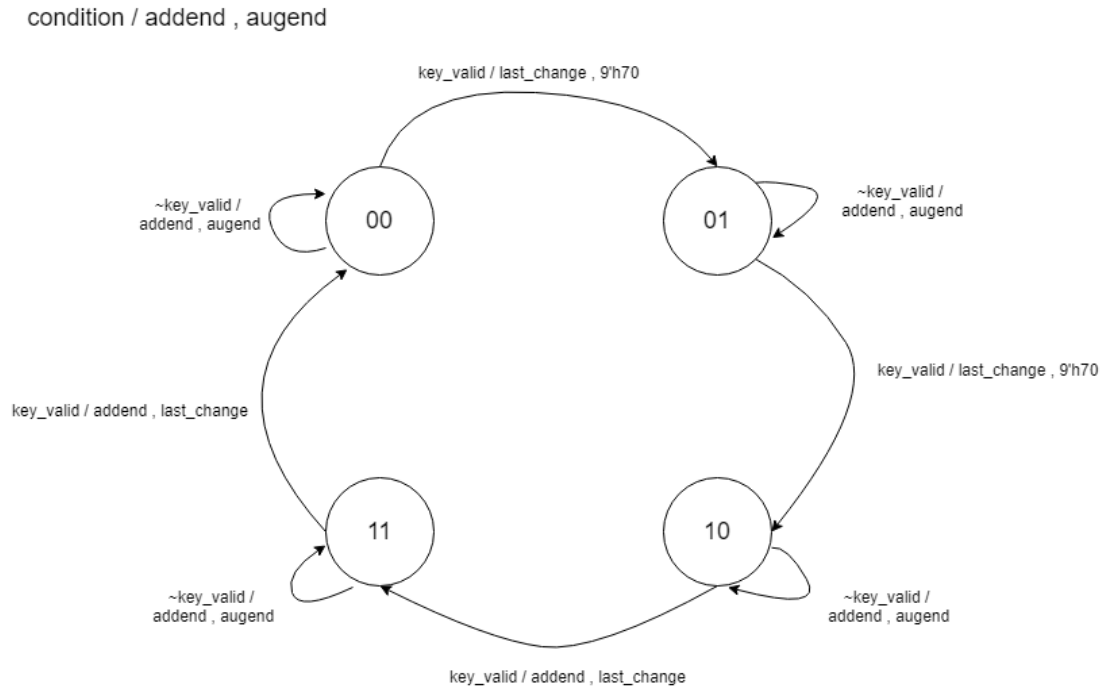
4. 除頻器: 輸出 2bit 的 ssd control enable 作為 scan control 的控制項。

5. scan control: 將 adder 的輸出 digit3～0 以及除頻器的輸出 ssd control enable 作為輸入，透過快速的輪流顯示，可以讓 4 個七段顯示器各自顯示自己要顯示的數字，並讓眼睛看起來是 4 個顯示器同時顯示的。

6. 七段顯示解碼器: 將 scan control 給的輸入值(ssd_in)透過解碼之後顯示在七段顯示器上。

Logic diagram:

fsm: 透過當前狀態判斷要將按鍵資料存到哪個變數裡。



重置狀態為 00。

(1-1) 00→01，有 key_valid 訊號時，代表有按鍵被按下，用 addend 記錄此資料，並將 augend 設作 0。

(1-2) 00→00，沒有 key_valid 訊號，addend、augend 維持原本的值。

(2-1) 01→10，有 key_valid 訊號時，代表有按鍵被放開，資料與剛剛是相同的，用 addend 記錄此資料，並將 augend 設作 0。

(2-2) 01→01，沒有 key_valid 訊號，addend、augend 維持原本的值。

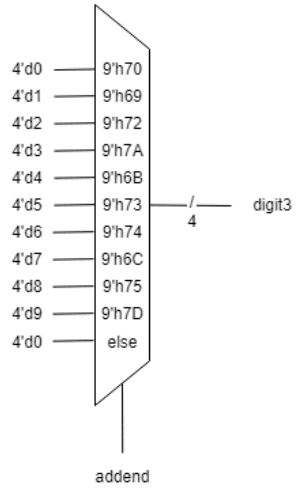
(3-1) 10→11，有 key_valid 訊號時，代表有按鍵被按下，addend 維持原本的值，用 augend 記錄此資料。

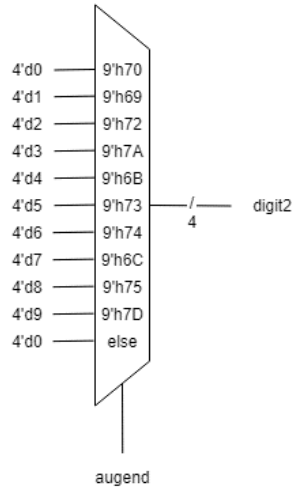
(3-2) 10→10，沒有 key_valid 訊號，addend、augend 維持原本的值。

(4-1) 11→00，有 key_valid 訊號時，代表有按鍵被放開，資料與剛剛是相同的，addend 維持原本的值，用 augend 記錄此資料。

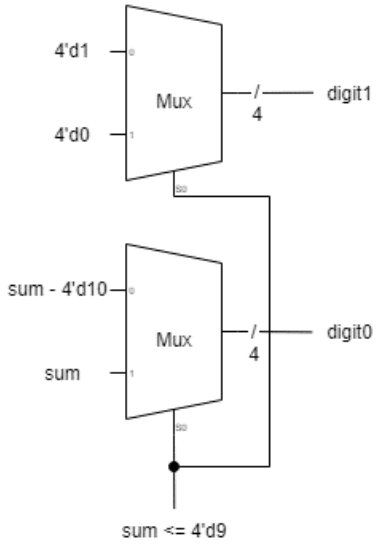
(4-2) 11→11，沒有 key_valid 訊號，addend、augend 維持原本的值。

adder: 先將資料解碼後再將被加數和加數相加後得到總和，之後轉換成 4 個 digit 的 BCD 形式輸出給 scan control 當作輸入。





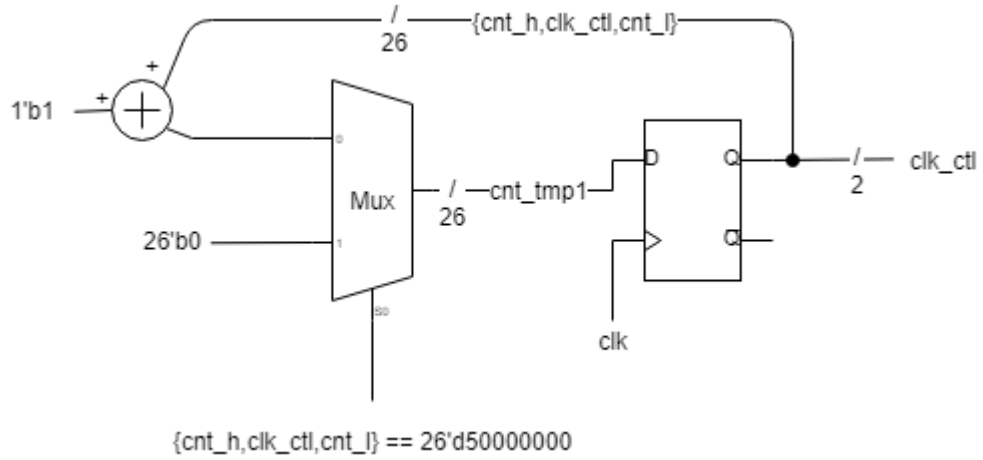
先將按鍵訊號解碼後存取所對應的數字在進行相加 sum = digit3 + digit2，



若總和不超過九，十位數 = 0，個位數 = sum；若總和大於九，十位數 = 1，個

位數 = sum – 10。

除頻器: 輸出 2bit 的 ssd control enable 作為 scan control 的控制項。



將 clk 為 100MHz、上限為 50M 的 counter 中的第 16、17bit 抓出來當作給 scan control 的控制項。

Scan control: 將 adder 的輸出 digit3～0 以及除頻器的輸出 ssd control enable 作為輸入，透過快速的輪流顯示，可以讓 4 個七段顯示器各自顯示自己要顯示的數字，並讓眼睛看起來是 4 個顯示器同時顯示的。

ssd_ctl_en = 00 → ssd_ctl = 0111
ssd_ctl_en = 01 → ssd_ctl = 1011
ssd_ctl_en = 10 → ssd_ctl = 1101
ssd_ctl_en = 11 → ssd_ctl = 1110

四個七段顯示器輪流顯示，ssd_ctl_en = 00 時→顯示第一個七段顯示器(被加數)；ssd_ctl_en = 01 時→顯示第二個七段顯示器(加數)；ssd_ctl_en = 10 時→顯示第三個七段顯示器(總和十位數)；ssd_ctl_en = 11 時→顯示第四個七段顯示器(總和個位數)，以快速輪流顯示的方式達成視覺暫留的效果。

7-segment display decoder: 將 scan control 給的輸入值(ssd_in)透過解碼之後顯示在七段顯示器上。

in = 0 → segs = 8'b0000_0011
in = 1 → segs = 8'b1001_1111
in = 2 → segs = 8'b0010_0101
in = 3 → segs = 8'b0000_1101

in = 4 → segs = 8'b1001_1001
in = 5 → segs = 8'b0100_1001
in = 6 → segs = 8'b0100_0001
in = 7 → segs = 8'b0001_1111
in = 8 → segs = 8'b0000_0001
in = 9 → segs = 8'b0000_1001
default 為 segs = 8’b1111_1111(全暗)

結合以上功能，可以完成一個將鍵盤訊號當作輸入並顯示在七段顯示器上的 single digit 加法器。

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
W4	V4	U4	U2	W5	R2

PS2_CLK	PS2_DATA
C17	B17

(3) Implement a two-digit decimal adder/subtractor/multiplier using the right-hand-side keyboard (inside the red block). You don't need to show all inputs and outputs at the same time in the 7-segment display. You just need to show inputs when they are pressed and show the results after "Enter" is pressed.

IO:

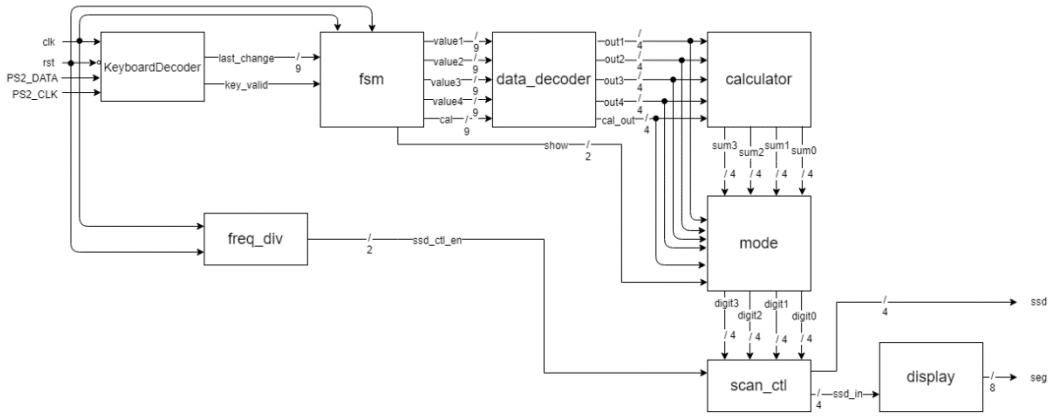
Inout: PS2_DATA, PS2_CLK

Input: rst, clk

Output: [7:0]seg, [3:0]ssd

Block diagram:

本次實驗需要使用以下 module 功能，分別為 KeyboardDecoder、fsm、訊號解碼器(data decoder)、計算器(calculator)、mode、除頻器(freq_div)、scan_control(scan_ctl)以及七段顯示解碼器(display)。



1. 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(keydown)、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)。

2. fsm: 透過當前狀態判斷要將按鍵資料存到哪個變數裡，並按照狀態告知(mode)現在要顯示哪些資料。

3. 訊號解碼器: 將資料解碼成 BCD 的形式讓 calculator 可以進行運算。

4. 計算器: 將資料進行加或減或乘的運算後，將結果輸出給 mode 當作輸入。

5. mode: 依據 fsm 給的通知訊號(show)選擇現在要在七段顯示器上顯示的資料，並將選擇好的資料輸出給 scan control 當作輸入。

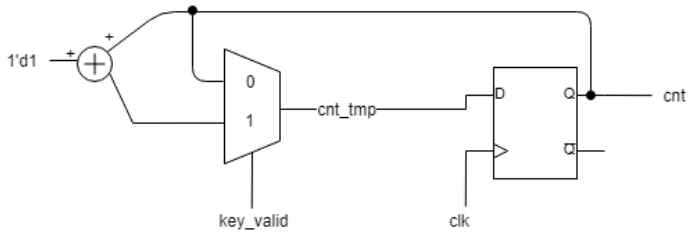
6. 除頻器: 輸出 2bit 的 ssd control enable 作為 scan control 的控制項。

7. scan control: 將 mode 的輸出 digit3～0 以及除頻器的輸出 ssd control enable 作為輸入，透過快速的輪流顯示，可以讓 4 個七段顯示器各自顯示自己要顯示的數字，並讓眼睛看起來是 4 個顯示器同時顯示的。

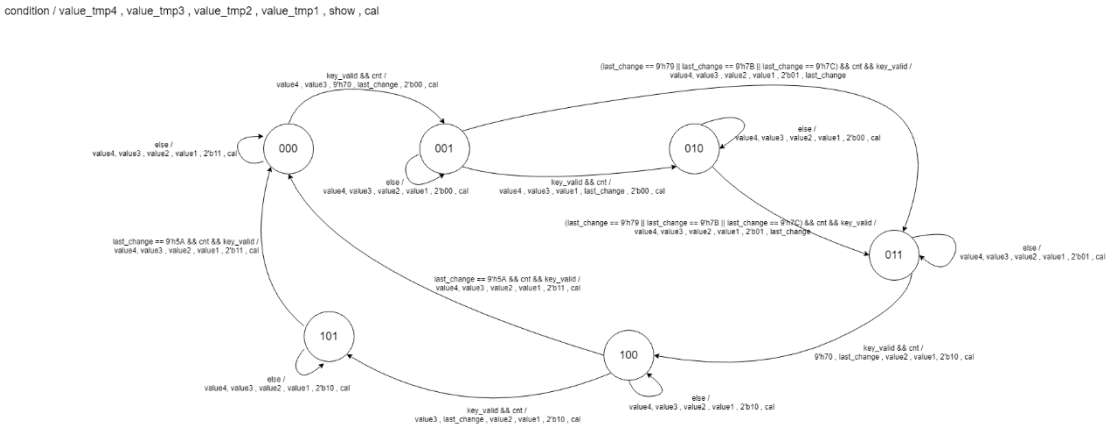
8. 七段顯示解碼器: 將 scan control 給的輸入值(ssd_in)透過解碼之後顯示在七段顯示器上。

Logic diagram:

<u>fsm</u>**:** 透過當前狀態判斷要將按鍵資料存到哪個變數裡，並按照狀態告知(mode)現在要顯示哪些資料。



由於每按一次鍵盤會有兩次的 key_valid 訊號，因此設計一個 one-bit 的 counter，每當有一次 key_valid 的訊號時，讓 cnt 加 1，透過 cnt 的限制，可以讓 fsm 在按下一次鍵盤時只會越過一個 state。



重置狀態為 000。

(1) 000→001，輸入第一組數字的第一個數字，用 value_tmp1 記錄此資料，並將 value_tmp2 設作 0；show = 2’b00(顯示第一組數字)。

(2-1) 001→011，輸入訊號為運算符號，用 cal 記錄此運算符號；show = 2’b01(顯示此時輸入的運算符號)，其餘資料不變。

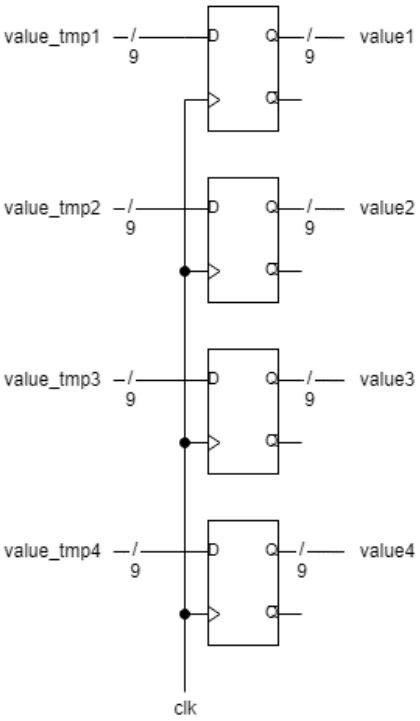
(2-2) 001→010，輸入第一組數字的第二個數字，用 value_tmp1 記錄此資料，用 value_tmp2 紀錄原本 value1 的值，將其挪至十位數；show = 2’b00(顯示第一組數字)。

(3) 010→011，輸入訊號為運算符號，用 cal 記錄此運算符號；show = 2’b01(顯示此時輸入的運算符號)，其餘資料不變。

(4) 011→100，輸入第二組數字的第一個數字，用 value_tmp3 記錄此資料，並將 value_tmp4 設作 0；show = 2’b 10(顯示第二組數字)。

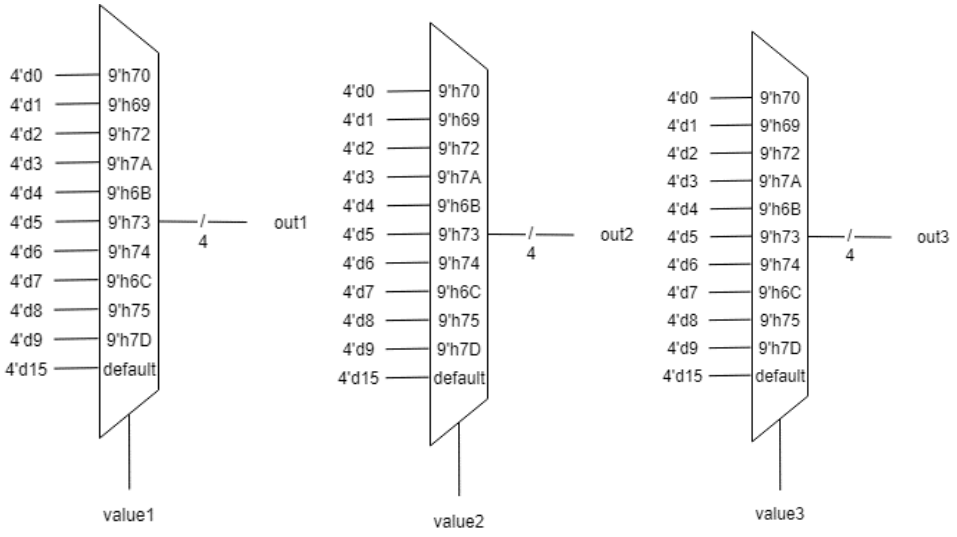
(5-1) 100→000，輸入訊號為 enter 鍵，show = 2’b11(顯示運算結果)，其餘所有

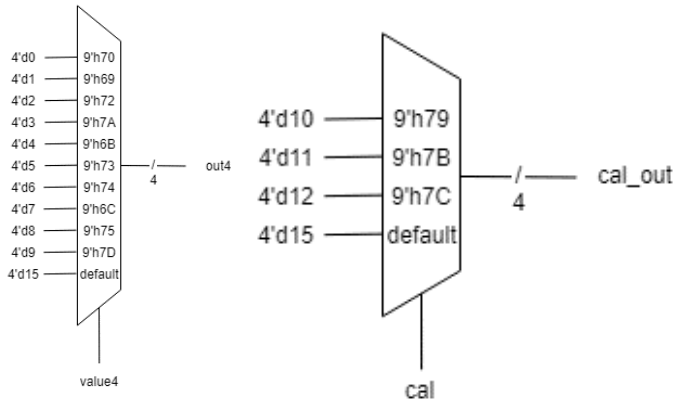
資料都不變。
(5-2) 100→101，輸入第二組數字的第二個數字，用 value_tmp3 記錄此資料，用 value_tmp4 紀錄原本 value3 的值，將其挪至十位數；show = 2’b10(顯示第二組數字)。
(6) 101→000，輸入訊號為 enter 鍵，show = 2’b11(顯示運算結果)，其餘所有資料都不變。



在 clk 來時將各個 value_tmp 的值輸入到各個 value 裡。

<u>訊號解碼器:</u> 將資料解碼成 BCD 的形式讓 calculator 可以進行運算。

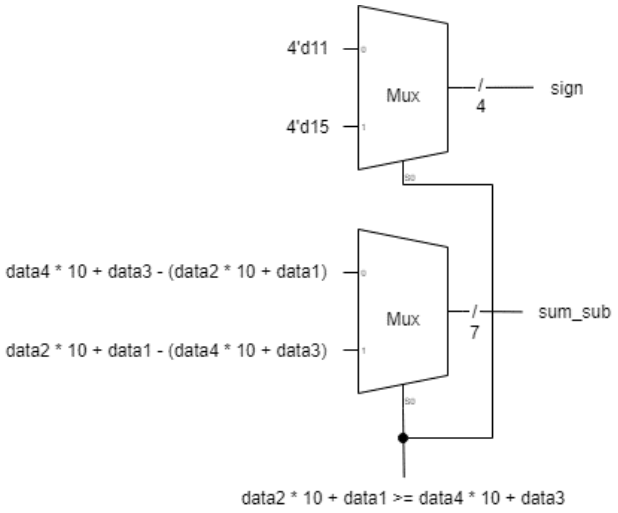




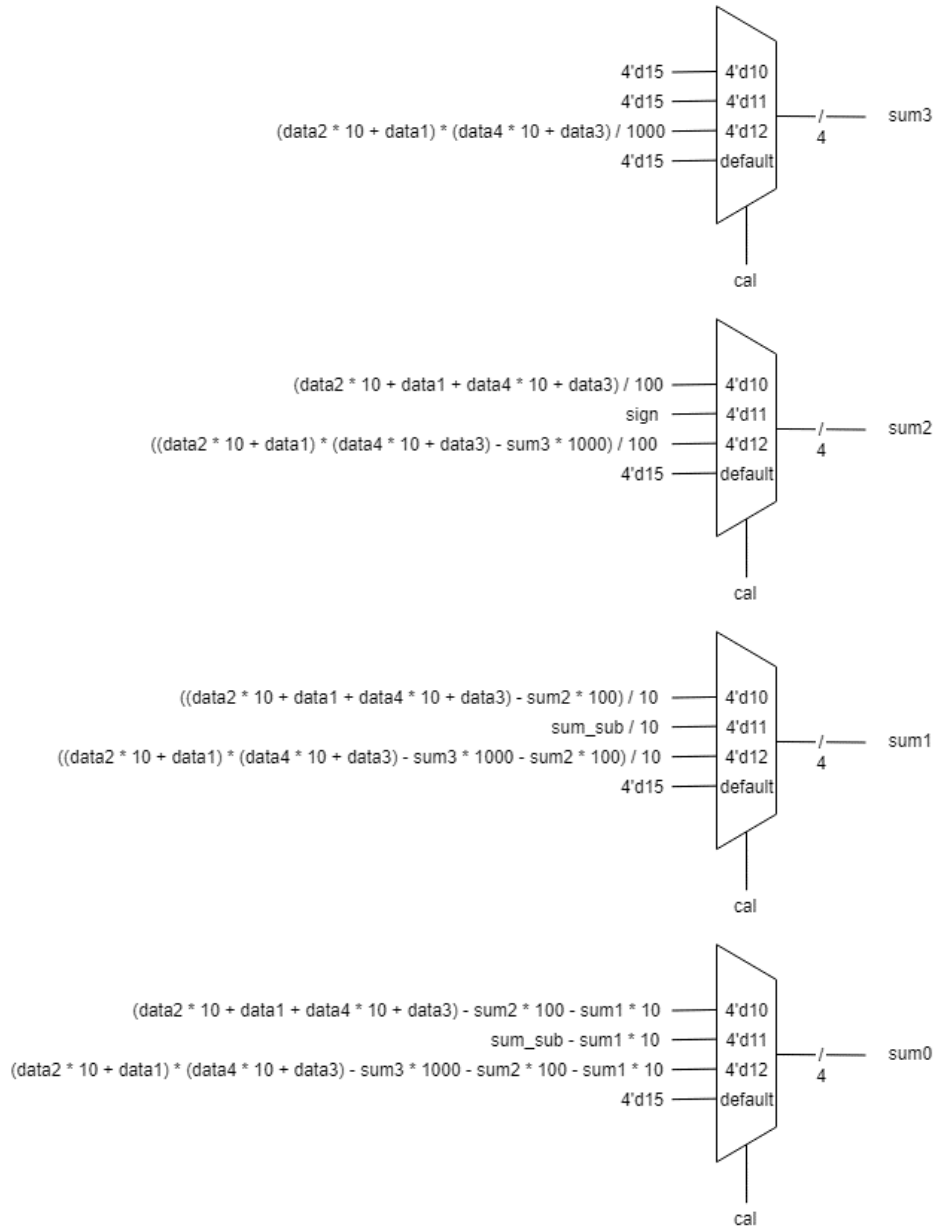
將各個鍵盤訊號解碼成所對應到的數字或運算符號，再將結果輸出到 calculator 進行運算。

<u>計算器:</u> 將資料進行加或減或乘的運算後，將結果輸出給 mode 當作輸入。

(sign 判斷是否要顯示負號)
(sum_sub 為減法的結果的絕對值)



先判斷兩組數字的大小關係，若第一組數字較大，sign = 4’d15(不顯示負號)，sum_sub = 第一組數字減掉第二組數字；若第二組數字較大，sign = 4’d11(顯示負號)，sum_sub = 第二組數字減掉第一組數字。



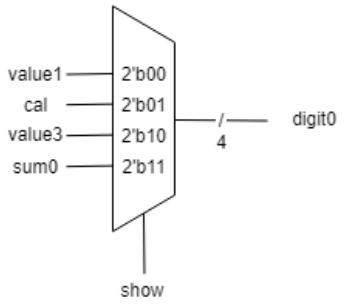
透過運算符號(cal)選擇 sum 要輸入加法、減法或是乘法的結果。

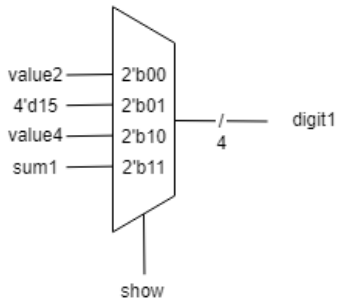
cal = 10 → 輸入加法結果；

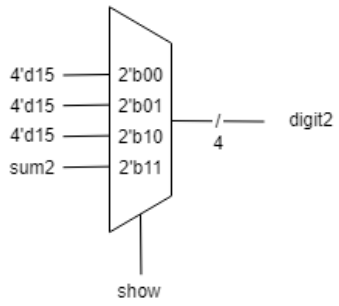
cal = 11 → 輸入減法結果；

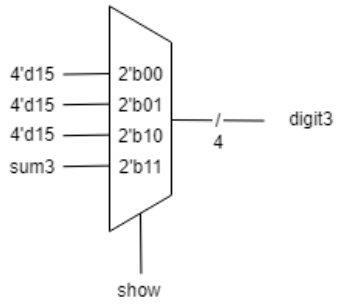
cal = 12 → 輸入乘法結果。

mode: 依據 fsm 給的通知訊號(show)選擇現在要在七段顯示器上顯示的資料，並將選擇好的資料輸出給 scan control 當作輸入。









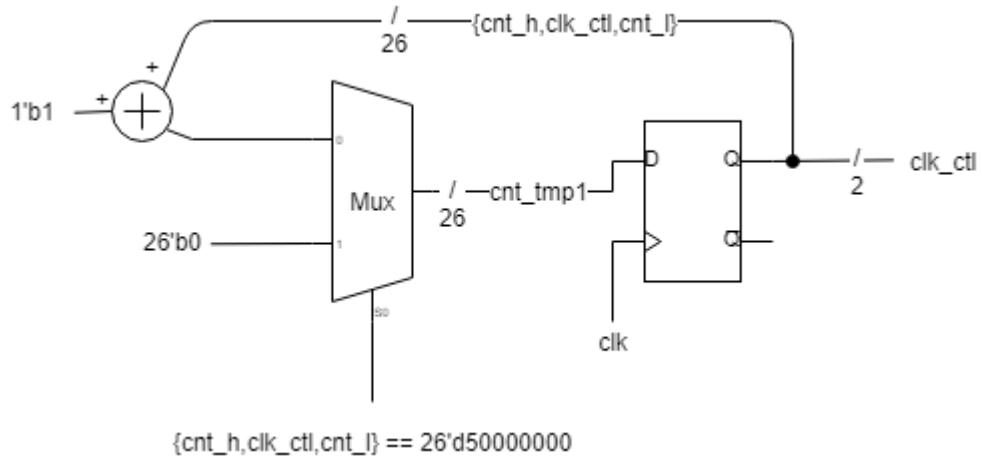
show = 2'b00 → 顯示第一組數字

show = 2'b01 → 顯示輸入的運算符號

show = 2'b10 → 顯示第二組數字

show = 2'b11 → 顯示運算結果

除頻器: 輸出 2bit 的 ssd control enable 作為 scan control 的控制項。



將 clk 為 100MHz、上限為 50M 的 counter 中的第 16、17bit 抓出來當作給 scan control 的控制項。

Scan control: 將 mode 的輸出 digit3～0 以及除頻器的輸出 ssd control enable 作為輸入，透過快速的輪流顯示，可以讓 4 個七段顯示器各自顯示自己要顯示的數字，並讓眼睛看起來是 4 個顯示器同時顯示的。

ssd_ctl_en = 00 → ssd_ctl = 0111
ssd_ctl_en = 01 → ssd_ctl = 1011
ssd_ctl_en = 10 → ssd_ctl = 1101
ssd_ctl_en = 11 → ssd_ctl = 1110

(依據模式顯示 1. 第一組數字 2. 輸入的運算符號 3. 第二組數字 4. 運算結果)
四個七段顯示器輪流顯示，ssd_ctl_en = 00 時→顯示第一個七段顯示器；ssd_ctl_en = 01 時→顯示第二個七段顯示器；ssd_ctl_en = 10 時→顯示第三個七段顯示器；ssd_ctl_en = 11 時→顯示第四個七段顯示器，以快速輪流顯示的方式達成視覺暫留的效果。

7-segment display decoder: 將 scan control 給的輸入值(ssd_in)透過解碼之後顯示在七段顯示器上。

in = 0 → segs = 8'b0000_0011
in = 1 → segs = 8'b1001_1111
in = 2 → segs = 8'b0010_0101
in = 3 → segs = 8'b0000_1101
in = 4 → segs = 8'b1001_1001

in = 5 → segs = 8'b0100_1001
in = 6 → segs = 8'b0100_0001
in = 7 → segs = 8'b0001_1111
in = 8 → segs = 8'b0000_0001
in = 9 → segs = 8'b0000_1001
in = 10 → segs = 8'b 0000_0101(加號)
in = 11 → segs = 8'b 1111_1101(減號)
in = 12 → segs = 8'b 1001_0001(乘號)
default 為 segs = 8'b1111_1111(全暗)

結合以上功能，可以完成一個有加法、減法、乘法功能的 two-digit decimal adder/subtractor/multiplier。

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
W4	V4	U4	U2	W5	R2

PS2_CLK	PS2_DATA
C17	B17

(4) Implement the “Caps” control in the keyboard. When you press A-Z and a-z in the keyboard, the ASCII code of the pressed key (letter) is shown on 7-bit LEDs.

4.1 Press “Caps Lock” key to change the status of capital/lower case on the keyboard. Use a led to indicate the status of capital/lowercase in the keyboard and show the ASSCII code of the pressed key one 7-bit LEDS.

4.2 Implement the combinational keys. When you press “Shift” and the letter keys at the same time. The 7-bit LEDs will show the ASCII code of the uppercase/lowercase of the pressed letter when the “Caps Lock” is at the lowercase/uppercase status.

IO:

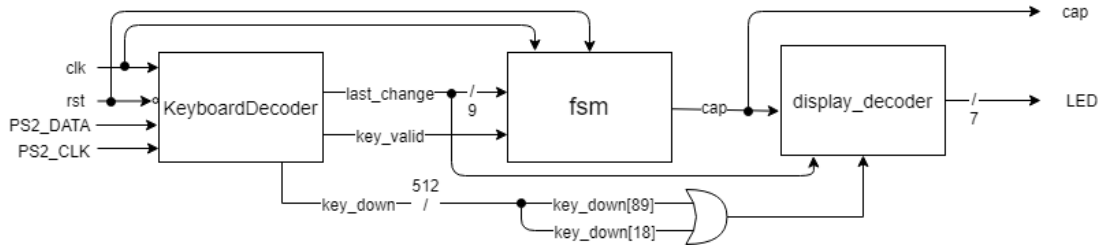
Inout: PS2_DATA, PS2_CLK

Input: rst, clk

Output: [6:0]LED, cap

Block diagram:

本次實驗需要使用以下 module 功能，分別為 KeyboardDecoder、fsm 以及 ASCII code 解碼器(display_decoder)。



1. 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(keydown)、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)。

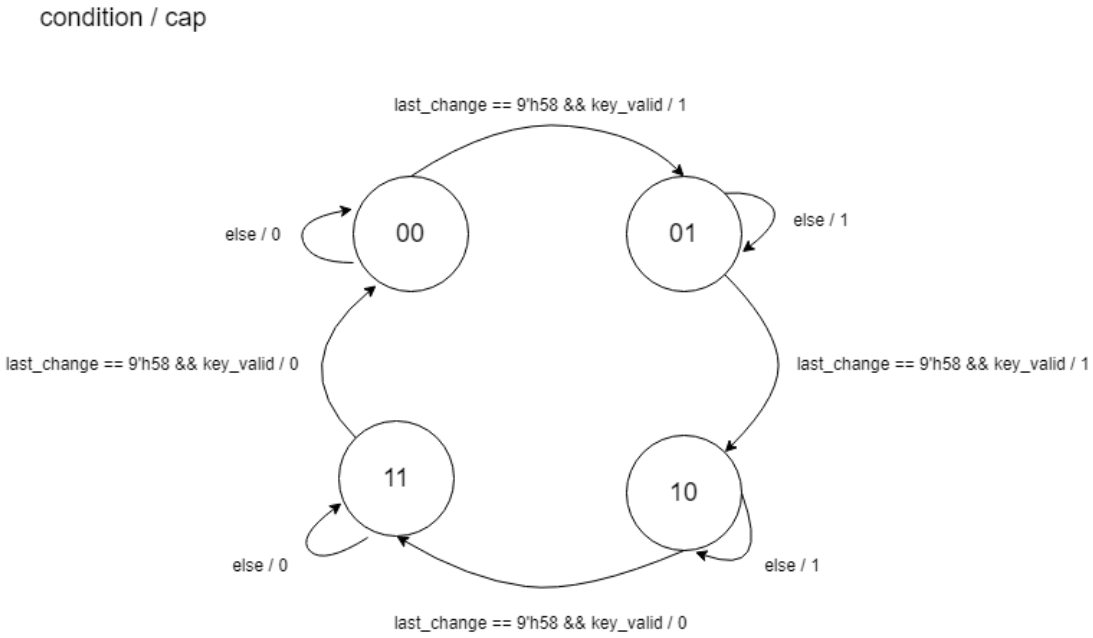
2. fsm: 透過當前狀態判斷 caps_lock 按鍵是否被啟動。

3. 將 key_down[89]和 key_down[18]通過 or gate 之後連接到 display_decoder 當作鍵盤上左右兩個 shift 按鍵的訊號輸入。

4. ASCII code 解碼器: 將各個大寫、小寫的英文字母轉換成所對應到的 ASCII code，並顯示在 LED 上。

Logic diagram:

fsm: 透過當前狀態判斷 caps_lock 按鍵是否被啟動。



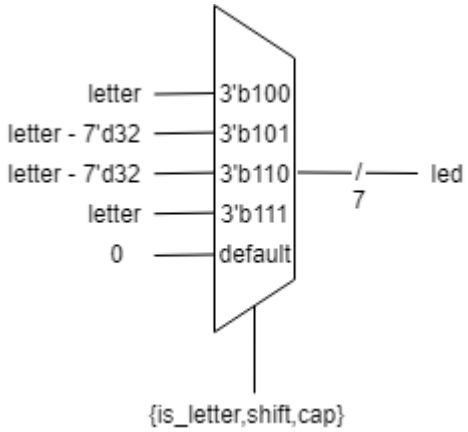
(1) 00→01，caps_lock 被按下，啟動 caps_lock，cap = 1。
(2) 01→10，caps_lock 按鍵被放開，cap = 1。
(3) 10→11，caps_lock 被按下，關閉 caps_lock，cap = 0。
(4) 11→00，caps_lock 按鍵被放開，cap = 0。
將 cap 訊號連接一個 LED 燈來表示目前 caps_lock 的狀態。

ASCII code 解碼器: 將各個大寫、小寫的英文字母轉換成所對應到的 ASCII code，並顯示在 LED 上。

last_change = 9'h1c(小寫 a) → letter = 97 , is_letter = 1
last_change = 9'h32(小寫 b) → letter = 98 , is_letter = 1
last_change = 9'h21(小寫 c) → letter = 99 , is_letter = 1
last_change = 9'h23(小寫 d) → letter = 100 , is_letter = 1
last_change = 9'h24(小寫 e) → letter = 101 , is_letter = 1
last_change = 9'h2b(小寫 f) → letter = 102 , is_letter = 1
last_change = 9'h34(小寫 g) → letter = 103 , is_letter = 1
last_change = 9'h33(小寫 h) → letter = 104 , is_letter = 1
last_change = 9'h43(小寫 i) → letter = 105 , is_letter = 1
last_change = 9'h3b(小寫 j) → letter = 106 , is_letter = 1
last_change = 9'h42(小寫 k) → letter = 107 , is_letter = 1
last_change = 9'h4b(小寫 l) → letter = 108 , is_letter = 1

last_change = 9'h3a(小寫 m) → letter = 109 , is_letter = 1
last_change = 9'h31(小寫 n) → letter = 110 , is_letter = 1
last_change = 9'h44(小寫 o) → letter = 111 , is_letter = 1
last_change = 9'h4d(小寫 p) → letter = 112 , is_letter = 1
last_change = 9'h15(小寫 q) → letter = 113 , is_letter = 1
last_change = 9'h2d(小寫 r) → letter = 114 , is_letter = 1
last_change = 9'h1b(小寫 s) → letter = 115 , is_letter = 1
last_change = 9'h2c(小寫 t) → letter = 116 , is_letter = 1
last_change = 9'h3c(小寫 u) → letter = 117 , is_letter = 1
last_change = 9'h2a(小寫 v) → letter = 118 , is_letter = 1
last_change = 9'h1d(小寫 w) → letter = 119 , is_letter = 1
last_change = 9'h22(小寫 x) → letter = 120 , is_letter = 1
last_change = 9'h35(小寫 y) → letter = 121 , is_letter = 1
last_change = 9'h1a(小寫 z) → letter = 122 , is_letter = 1
default: letter = 0 , is_letter = 0

先將各英文字母按鍵對應到小寫的 ASCII code，再透過下方的多工器選擇是否要轉換成大寫的 ASCII code；is_letter 用來判斷按鍵是否為其中一個英文字母按鍵，若不是，則 is_letter = 0(LED 全暗)。



(shift 為按下鍵盤上 shift 按鍵的 key_down 訊號)
當{is_letter,shift,cap} = 3’b100 → 不需要改成大寫，led = letter。
當{is_letter,shift,cap} = 3’b101 → 需要改成大寫，led = letter - 32。
當{is_letter,shift,cap} = 3’b110 → 需要改成大寫，led = letter - 32。
當{is_letter,shift,cap} = 3’b111 → 不需要改成大寫，led = letter。
其餘狀況為 is_letter = 0，也就是按到不是英文字母的按鍵 → led = 0(全暗)。

結合以上模組，可以完成一個同時具有 caps_lock 和 shift 控制英文字母大小寫，並將各英文字母對應到的 ASCII code 顯示在 LED 上的功能。

I/O pin assignment:

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

cap	PS2_CLK	PS2_DATA	clk	rst
L1	C17	B17	W5	R2

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。