**Lab 08**

105060016 謝承儒

1. **Lab08-1**
2. **Design Specification :**
3. **Input :** clk //100MHz的clk，用來驅動各個block

rst //打開時，回到最初狀態

1. **Output :** [7:0] display //七段顯示器的顯示碼

[3:0] display\_c //控制4個顯示器中哪個會改變

1. **Inout :** PS2\_CLK

PS2\_DATE

1. **Wire :** [511:0] key\_down //記住哪些按鍵被按住

[8:0] last\_change //記住最後被按的按鍵是哪個

key\_valid //是否需要讀值

[1:0] ssd\_ctl //用於作為顯示哪個顯示器的選擇依據

[7:0] d0

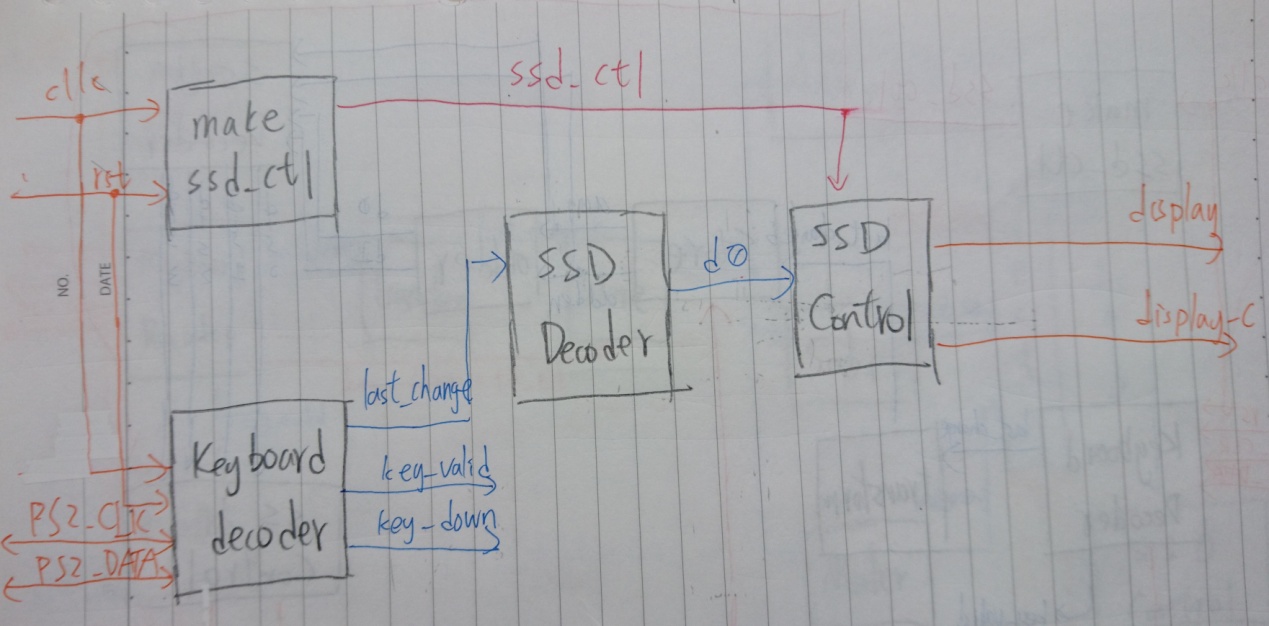


圖1 Lab08-1的區塊圖

1. **Design Implementation :**
2. **Code :**

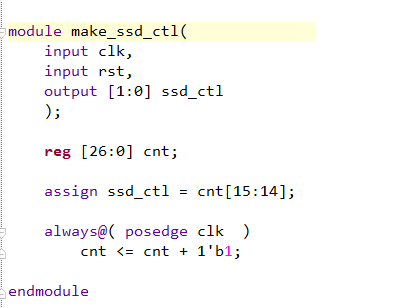


圖2 Lab08-1的make\_ssd\_ctl

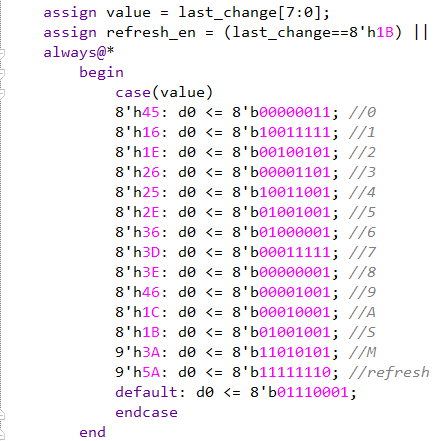


圖3 Lab08-1的SSD Decoder

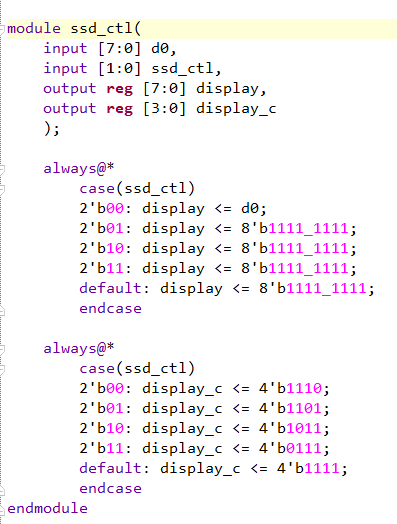


圖4 Lab08-1的SSD Control

1. **Pin assignment :**
2. **Input :**
3. clk = W5
4. rst = R2
5. **Output :**
6. display[0] = V7
7. display[1] = U7
8. display[2] = V5
9. display[3] = U5
10. display[4] = V8
11. display[5] = U8
12. display[6] = W6
13. display[7] = W7
14. display\_c[0] = U2
15. display\_c[1] = U4
16. display\_c[2] = V4
17. display\_c[3] = W4
18. **Inout :**

PS2\_ CLK = C17

PS2\_ DATA = B17

1. **Discussion :**
2. **整體運作過程 :** 
   * + - 1. 將clk輸入進製造ssd\_ctl的bolck，輸出ssd\_ctl來做為待會SSD\_Control的其中一個Input。
         2. 將last\_change輸入到SSD\_Decoder，經過解碼後得到顯示碼(d0)，把它輸出。
         3. 把d0輸入到SSD\_Control，再加上ssd\_ctl，就可以把d0顯示出來。
3. **各block的構想 :**
4. **make\_ssd\_ctl:**

本來這是在除頻器裡，不過這是不需用除頻，就額外做一個block來製造ssd\_ctl，協助SSD Control運作。

1. **SSD Decoder :** 將last\_change轉換成顯示碼(d0)

把last\_change輸入進來，因為這次要顯示的只有左邊的0~9、A、S、M，沒有左右的差別，所以只看後8-bit就好。利用case在last\_change不同的值，輸出不同的d0。

1. **SSD Control :** 決定哪塊板子的值改變

把在除頻器得到的2-bit的ssd\_ctl輸入，作為選擇SSD四個顯示器的依據。

當ssd\_ctl\_en=00時，便將display0輸出(display)，並將display\_c輸出為1110。如此顯示器上便只有最後一位可以改變，保留其他三個的字母，其他ssd\_ctl的情況也是一樣。

1. **過程中的Bug :**

一開始完全不懂從KeyboardDecoder輸出的東西代表甚麼，不過經過助教解釋後就懂了。

1. **Lab08-2 : 加法器**
   1. **Design Specification :**
      1. **Input :** clk //輸入的頻率(100MHz) 。

rst //當=1時，回到最初狀態

* + 1. **Output :** [7:0] display //七段顯示器的顯示碼。

[3:0] display\_c //決定哪個顯示器改變。

* + 1. **Inout :** PS2\_CLK

PS2\_DATA

* + 1. **Wire :** [511:0] key\_down //記住哪些按鍵被按住

[8:0] last\_change //記住最後被按的按鍵是哪個

key\_valid //是否需要讀值

[1:0] ssd\_ctl //用於作為顯示哪個顯示器的選擇依據

tf\_en //改變輸入的值是要給被加數或是加數

[3:0] last\_change\_b//把last\_change解碼成BCD

[3:0] aug、adden //被加數、加數的值

[3:0] d0、d1 //和的個位數、十位數

[7:0] dis0,dis1,dis2,dis3

//將d0、d1、adden、aug轉換出來的顯示碼

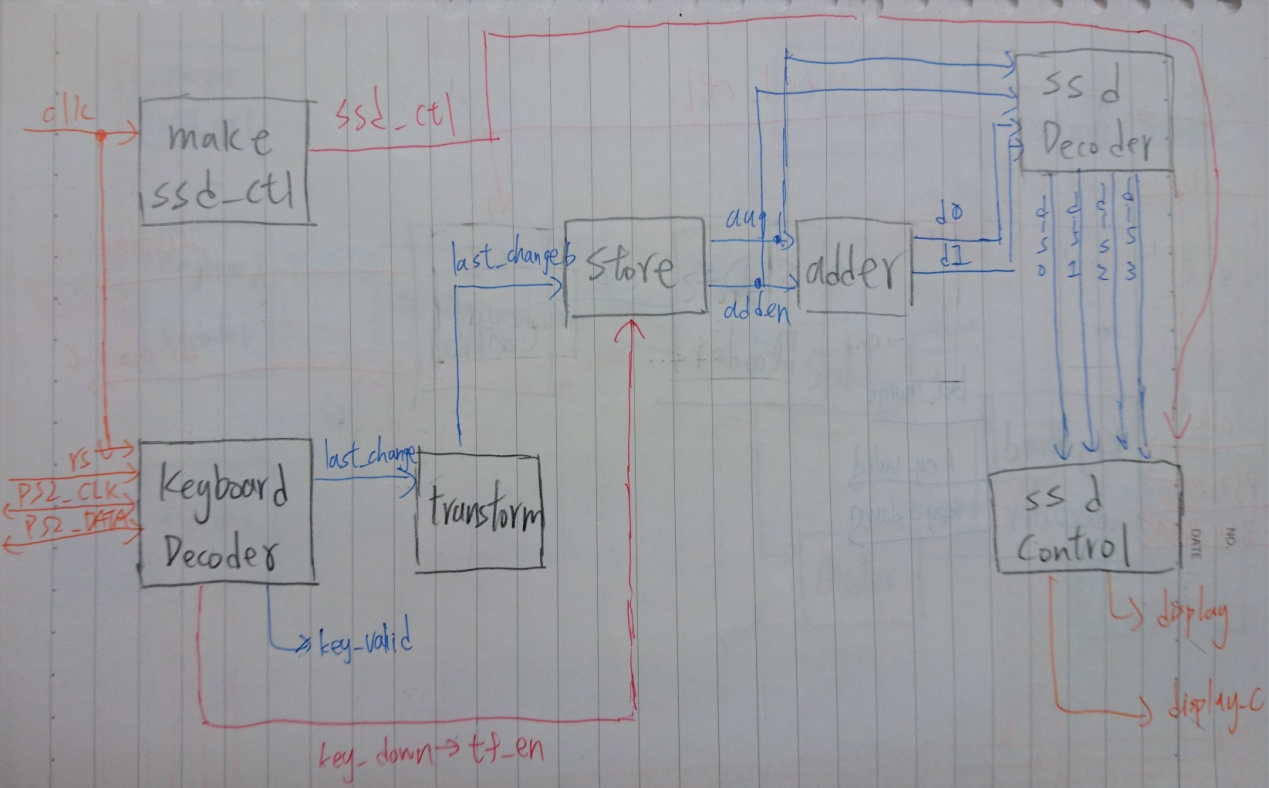


圖5 Lab08-2的區塊圖

* 1. **Design Implementation :** 
     1. **Logic function :**

tf\_en = key\_down[8'h45] || key\_down[8'h16] || key\_down[8'h1E] || key\_down[8'h26] ||key\_down[8'h25] || key\_down[8'h2E] || key\_down[8'h36] || key\_down[8'h3D] ||key\_down[8'h3E] || key\_down[8'h46]

* + 1. **Code :**

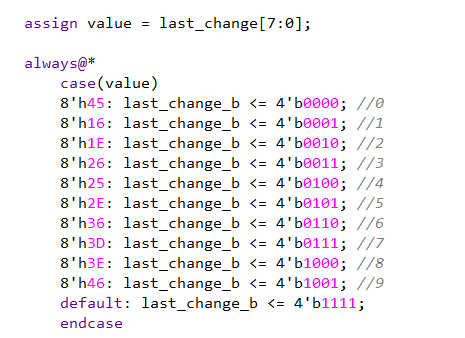


圖6 Lab08-2的transform

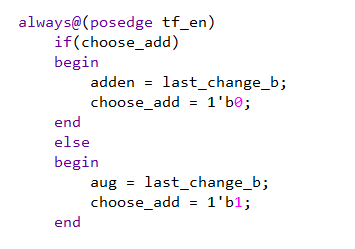


圖7 Lab08-2的store

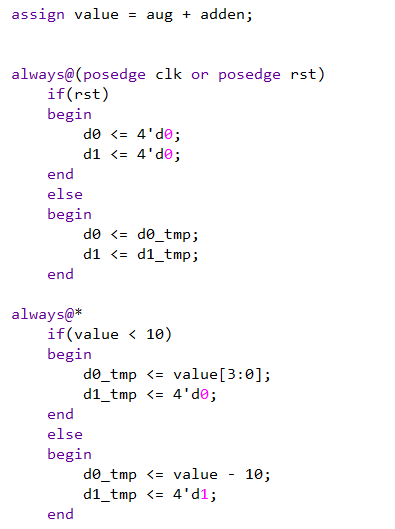


圖8 Lab08-2的adder

* + 1. **Logic diagram**

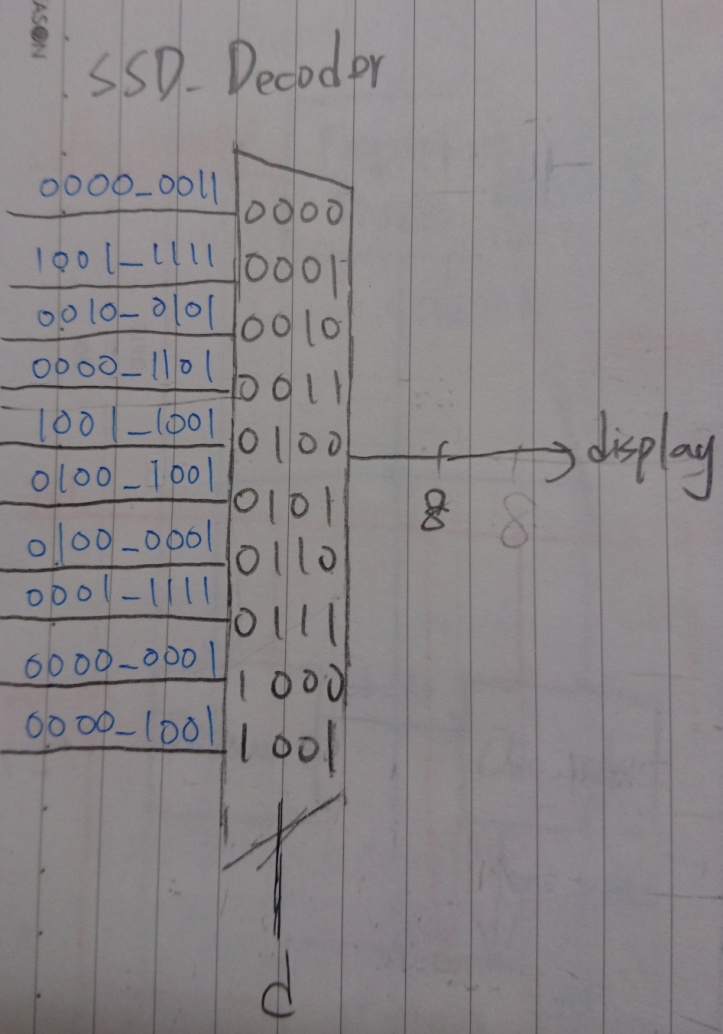


圖9 Lab08-2的 SSD\_Decoder

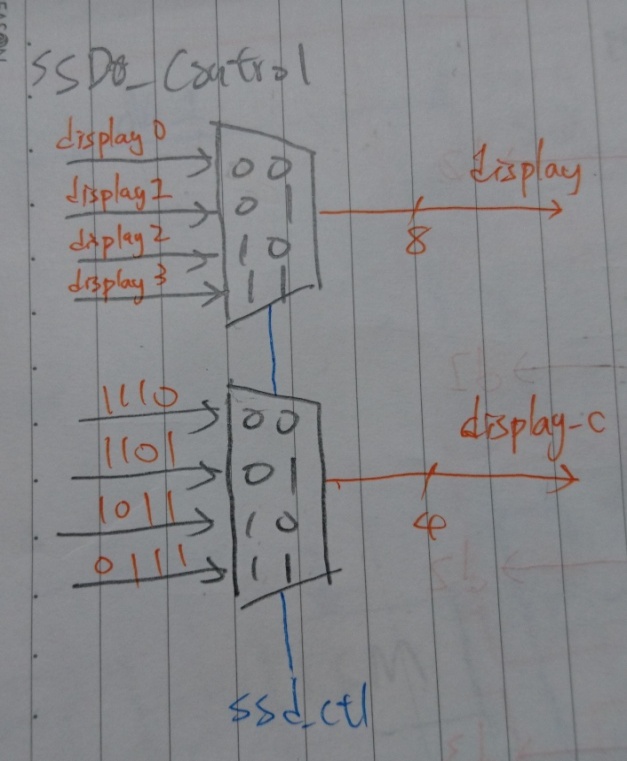


圖10 Lab08-2的 SSD\_Control

* + 1. **Pin assignment :**
       - 1. **Input :**

clk : W5

rst : V17

* + - * 1. **Output :**

display[0] = V7

display[1] = U7

display[2] = V5

display[3] = U5

display[4] = V8

display[5] = U8

display[6] = W6

display[7] = W7

display\_c[0] = U2

display\_c[1] = U4

display\_c[2] = V4

display\_c[3] = W4

* + - * 1. **Inout :**

PS2\_ CLK = C17

PS2\_ DATA = B17

* 1. **Discussion :** 
     1. **整體運作過程 :** 
        + 1. 將clk輸入進製造ssd\_ctl的bolck，輸出ssd\_ctl來做為待會SSD\_Control的其中一個Input。
          2. 把last\_change輸入到transform block解碼成4-bit BCD形式的last\_change\_b。
          3. 把key\_down裡代表0~9的bit用or連起來(tf\_en)，作為改變last\_change\_b儲存位置的trigger。把tf\_en和last\_change\_b輸入到store block儲存到被加數、加數(aug、adden)裡面。
          4. 把aug、adden輸入到adder bolck，得到和的個位數和十位數(d0、d1)。
          5. 把aug、adden、d0、d1輸入到SSD\_Decoder，解碼成8-bit的顯示碼(dis3~0)。
          6. 把dis3~0和ssd\_ctl輸入到SSD\_Control，輸出要顯示的東西(display)以及它會顯示在哪一個顯示器(display\_c)。
     2. **新block的構想 :** 
        + 1. **transform :** 把9-bit的last\_change轉成4-bit BCD的last\_change\_b

取出last\_change後8碼來知道是按下甚麼數字，就可以轉換成BCD的last\_change\_b形式。

* + - * 1. **tf\_en :** 改變last\_change\_b存放的位置

把key\_down裡代表0~9的bit用or連起來，如此在按下且放開後，就只會有1次posedge，可以作為是否已經輸入好數字的依據。

* + - * 1. **store :** 將last\_change\_b儲存到aug、adden

開一個choose\_add來決定下次存的是aug或是adden。

每次tf\_en posedge時，就將last\_change\_b儲存起來，並改變下次儲存的位置。如果這次存在aug(choose\_add = 0)，那posedge時，last\_change\_b就會存到aug裡，並且將儲存位置改為adden(choose\_add = 1)。

* + - * 1. **adder :** 將aug和adden相加得到和

把aug、adden輸入進來，先開一個5-bit的value儲存兩者相加的和。

若value小於10，就讓d0 = value，d1 = 0。

若value沒有小於10，就讓d0 = value – 10，d1 = 1。

* + 1. **過程中的Bug :**

本來打算使用key\_valid來做為trigger，來改變輸入的值存放的位置，但當按下時會有1次posedge，放開時也會有1次，如此按下後放開，輸入的值仍會存在被加數，而不會存到加數的位置。

後來同學建議說使用key\_down來作為trigger，把0~9的key\_down都用or gate接在一起，也就是tf\_en，這樣當按下任一數字鍵時，tf\_en就會變為1，放開時就會變成0，如此就不會有第二次的posedge，能夠順利的改變存放位置。

1. **Lab08-3 : 加/減/乘法器**
   1. **Design Specification :**
      1. **Input :** clk //100MHz的clk，用來驅動各個block

　　　rst //當=1時，回到最初狀態

* + 1. **Output :** [7:0] display //七段顯示器的顯示碼

[3:0] display\_c //決定哪個顯示器改變

add\_en, sub\_en, mul\_en

//判斷現在是在加/減/乘狀態

* + 1. **Inout :** PS2\_CLK

PS2\_DATA

* + 1. **Wire :** [511:0] key\_down //記住哪些按鍵被按住

[8:0] last\_change //記住最後被按的按鍵是哪個

key\_valid //是否需要讀值

[1:0] ssd\_ctl //用於作為顯示哪個顯示器的選擇依據

[3:0] last\_change\_b//把last\_change解碼成BCD

add, sub, mul, result\_en

//判斷是否按下加、減、乘、Enter

tf\_en //改變輸入的值存放的位置

[3:0] store\_s0,store\_s1,store\_s2,store\_s3

//輸入的值

[3:0] add\_a0,add\_a1,add\_a2,add\_a3

//經過加法後得到的值

[3:0] sub\_s0,sub\_s1,sub\_s2,sub\_s3

//經過減法後得到的值

[3:0] mul\_m0,mul\_m1,mul\_m2,mul\_m3

//經過乘法後得到的值

[3:0] d0,d1,d2,d3 //要顯示出來的值

[7:0] dis0,dis1,dis2,dis3

//由d0,d1,d2,d3轉換出來的顯示碼

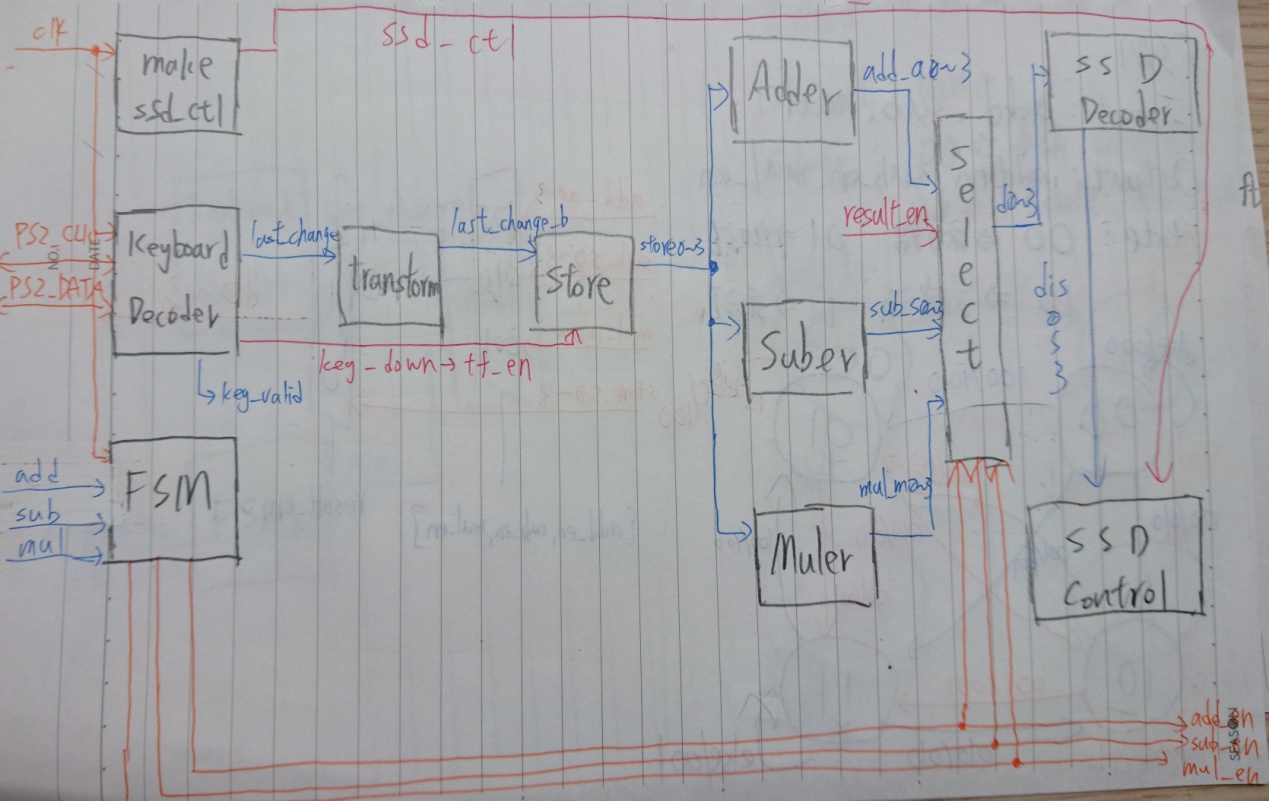


圖11 Lab08-3的區塊圖

* 1. **Design Implementation :**
     1. **Logic function :** 
        + 1. **tf\_en =** key\_down[8'h70] || key\_down[8'h69] || key\_down[8'h72] || key\_down[8'h7A] || key\_down[8'h6B] || key\_down[8'h73] || key\_down[8'h74] || key\_down[8'h6C] || key\_down[8'h75] || key\_down[8'h7D]
          2. **add =** (key\_down[8'h79])? 1'b1 : 1'b0
          3. **sub =** (key\_down[8'h7B])? 1'b1 : 1'b0
          4. **mul =** (key\_down[8'h7C])? 1'b1 : 1'b0
          5. **result\_en =** (key\_down[8'h5A])? 1'b1 : 1'b0
     2. **Code :**

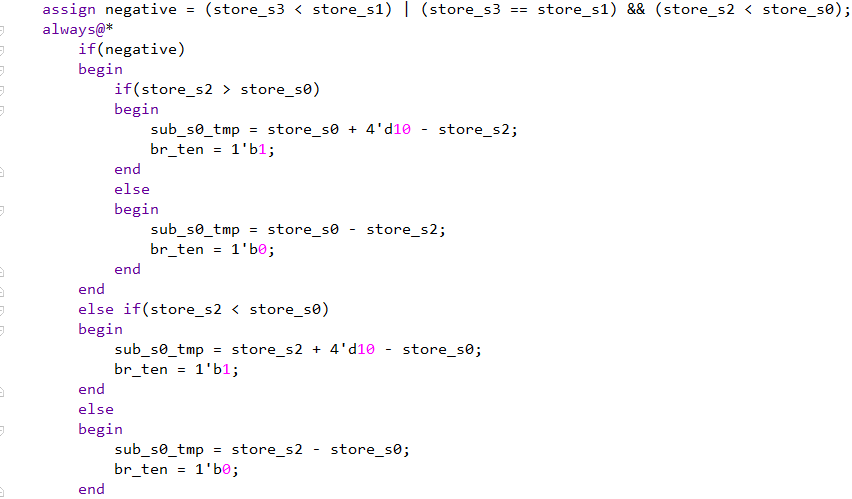


圖12 Lab08-3的suber(個位數)

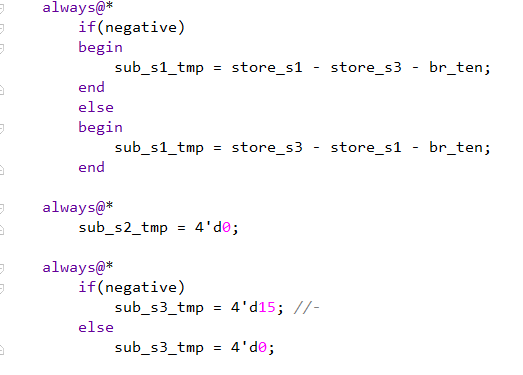


圖13 Lab08-3的suber(十位數)

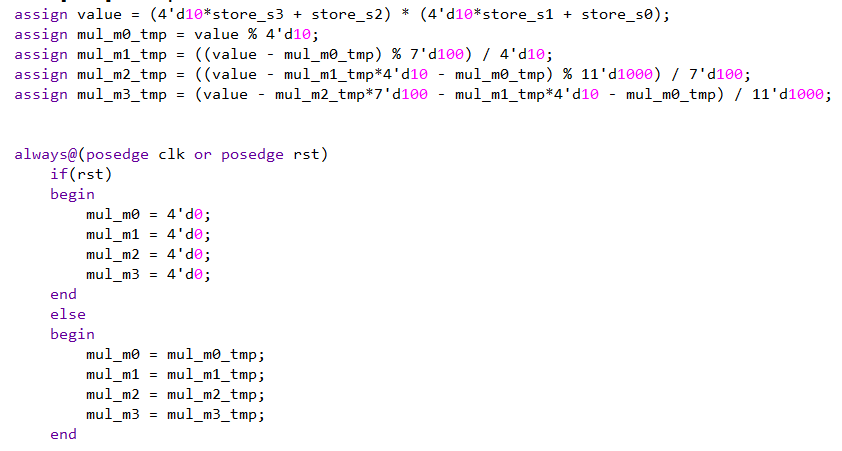


圖14 Lab08-3的muler

* + 1. **Logic diagram :**

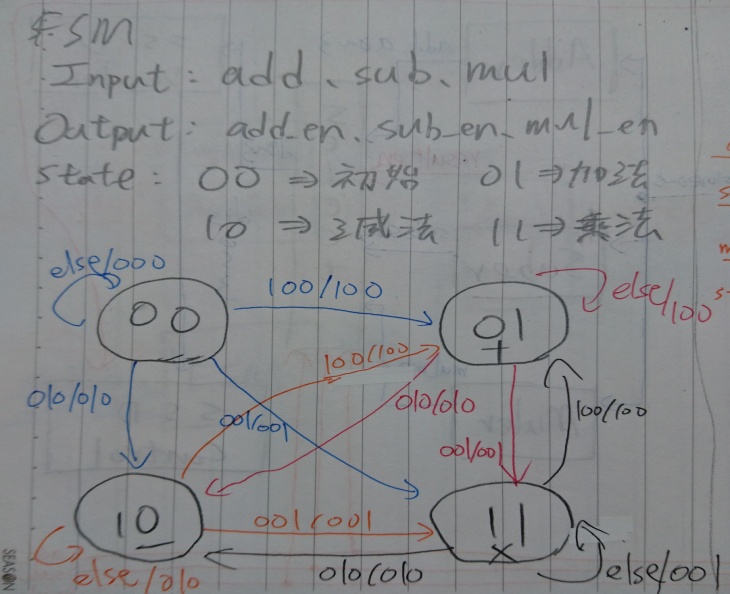


圖15 Lab08-3的FSM

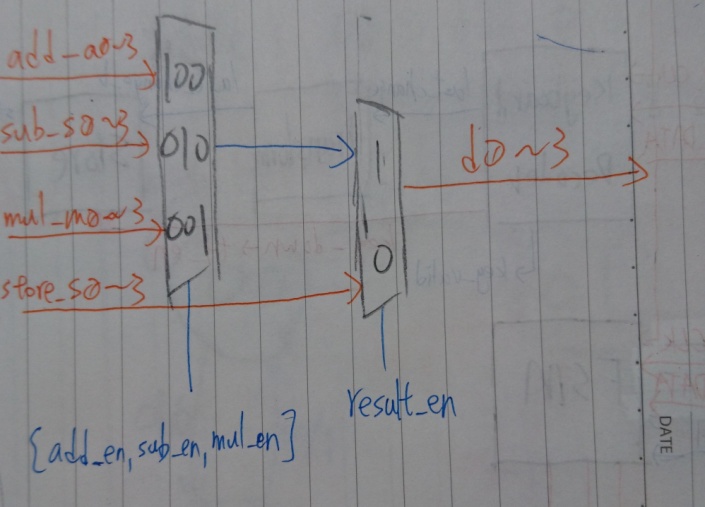


圖16 Lab08-3的select\_value

* + 1. **Pin assignment :** 
       - 1. **Input :**

clk : W5

rst : R2

* + - * 1. **Output :**

add\_en = L1

sub\_en = P1

mul\_en = N3

display[0] = V7

display[1] = U7

display[2] = V5

display[3] = U5

display[4] = V8

display[5] = U8

display[6] = W6

display[7] = W7

display\_c[0] = U2

display\_c[1] = U4

display\_c[2] = V4

display\_c[3] = W4

* + - * 1. **Inout :**

PS2\_ CLK = C17

PS2\_ DATA = B17

* 1. **Discussion :** 
     1. **整體運作過程 :** 
        + 1. 將clk輸入進製造ssd\_ctl的bolck，輸出ssd\_ctl來做為待會SSD\_Control的其中一個Input。
          2. 把last\_change輸入到transform block解碼成4-bit BCD形式的last\_change\_b。
          3. 把key\_down裡代表0~9的bit用or連起來(tf\_en)，作為改變last\_change\_b儲存位置的trigger。把tf\_en和last\_change\_b輸入到store block儲存到store\_s3~0裡面。
          4. 把store\_s3~0輸入到加法器(adder)，減法器(suber)、乘法器(muler)，分別得到和(add\_a3~0)、差(sub\_s3~0)、積(mul\_m3~0)。
          5. 按下**+**、**-**、**\***，讓add、sub、mul其中之一為1，讓FSM的狀態變到加、減、乘其中之一，輸出add\_en、sub\_en、mul\_en。
          6. 把add\_en、sub\_en、mul\_en、result\_en和add\_a3~0、sub\_s3~0、mul\_m3~0輸入到select\_value block。按下Enter時，再依據現在是處於哪種運算，從和、差、積其一輸出(d3~0)。
          7. 把d3~0輸入到SSD\_Decoder解碼成8-bit的顯示碼(dis3~0)。
          8. 把dis3~0和ssd\_ctl輸入到SSD\_Control，輸出要顯示的東西(display)以及它會顯示在哪一個顯示器(display\_c)。
     2. **新block的構想 :** 
        + 1. **fsm :** 決定現在處於初始、加、減、乘何種狀況

最一開始並沒有處於任何運算，add\_en, sub\_en, mul\_en皆為0。若按下**+**、**-**、**\***後，便會跳到相應的運算狀態，並把add\_en, sub\_en, mul\_en其一設為1，來代表現在處於何種運算。

* + - * 1. **suber :** 把store\_s3~0做減法處理

把store\_s3、store\_s2作為被減數的十、個位數，store\_s1、store\_s0作為減數的十、個位數。

首先，先把兩數比較，若被減數較大或相等，則差為不為負(negative = 0)；反之，若被減數較小，則差為負(negative=1)。

若negative=0，判斷store\_s2、store\_s0何者誰大，若store\_s0較大，則就需要借位(br\_ten=1)；反之，就不需要借位，直接相減即可。十位數的值就是store\_s3- store\_s1- br\_ten。

若negative=1，只需要把被減數和減數的位置顛倒就可以，變成store\_s1、store\_s0減掉store\_s3、store\_s2，只是需要在最左邊的顯示器顯示負號。

* + - * 1. **muler :** 把store\_s3~0做乘法處理

把store\_s3、store\_s2作為被乘數的十、個位數，store\_s1、store\_s0作為乘數的十、個位數。

設定積為value= (10\*store\_s3+store\_s2) \* (10\*store\_s1+store\_s0)

這部分我利用除號**/**和取餘數**%**來做。

取個位數時，對value%10。

取十位數時，減掉個位數，再%100，最後/10。

取百位數時，減掉個、十位數，再%1000，最後/100。

取千位數時，減掉個、十、百位數，再/1000。

* + - * 1. **select\_value :** 依據現在是何種運算，選擇不同的結果顯示

把得到的和(add\_a3~0)、差(sub\_s3~0)、積(mul\_m3~0)和FSM的add\_en、sub\_en、mul\_en輸入。當按下Enter代表result\_en=1，就會把運算結果存入要顯示的東西(d3~0)，若是在初始狀態，因為沒有在運算，所以就算按Enter也只會顯示原樣。

* + 1. **過程中的Bug :**

在乘法器取千、百、十位數時，把後面位數清成0後，就直接把值放進mul\_m3~1，忘了將它們除以1000、100、10。像是如果積是2182，要取千位數時，我將它清為2000後就直接儲存起來。應該把2000除以1000後得到2在把它儲存。

1. **Lab08-4 : 字母大小寫**
   1. **Design Specification :**
      1. **Input :** clk //100MHz的clk，用來驅動各個block

　 　 rst //當=1時，回到最初狀態

* + 1. **Output :** mini //代表現在是否為小寫

[6:0]LEDs //用來顯示字母的編號

* + 1. **Inout :** PS2\_CLK

PS2\_DATA

* + 1. **Wire :** [511:0] key\_down //記住哪些按鍵被按住

[8:0] last\_change //記住最後被按的按鍵是哪個

key\_valid //是否需要讀值

shift\_en //是否按下shift

cap\_en //是否按下Cap

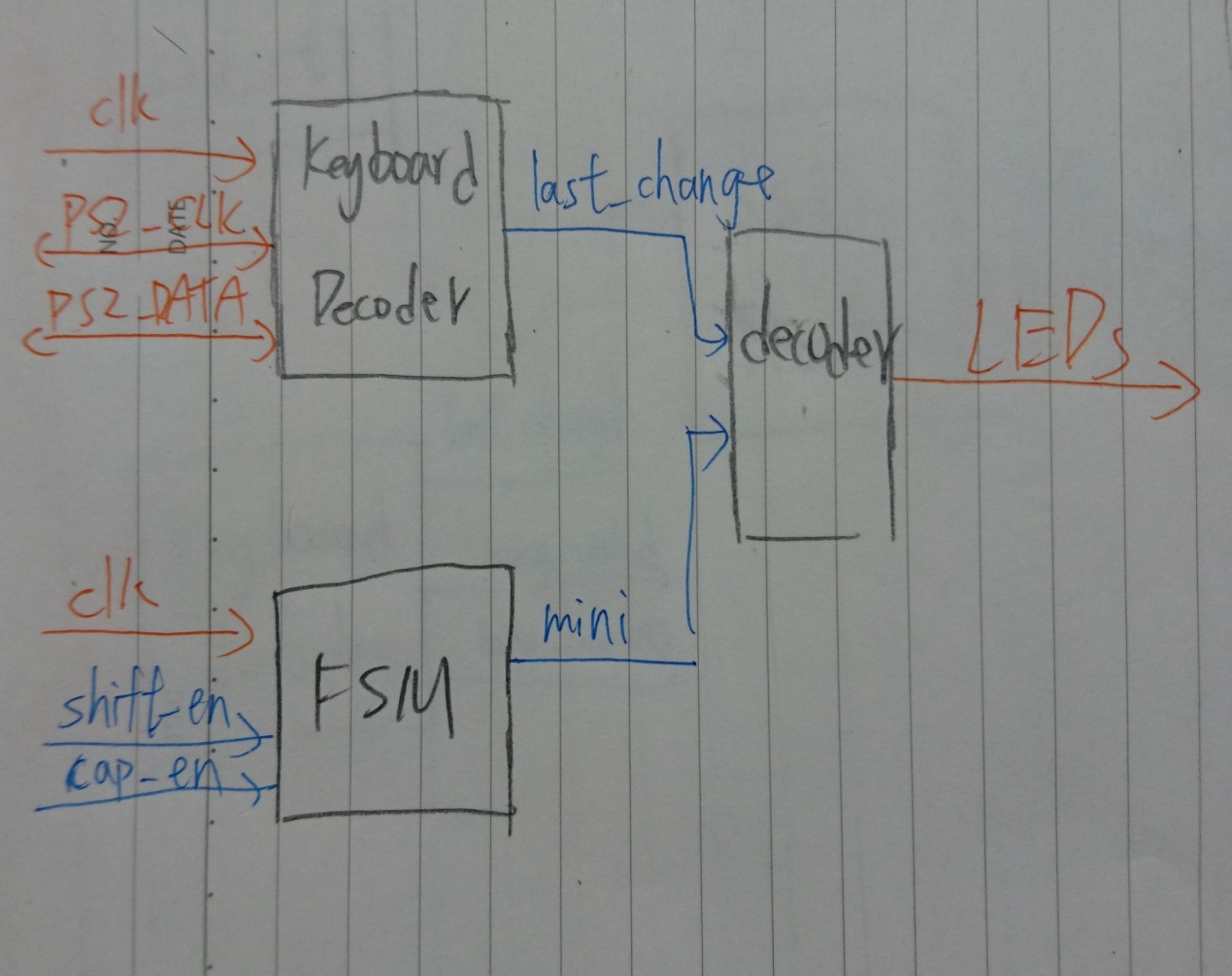
****

圖17 Lab08-4的區塊圖

* 1. **Design Implementation :**
     1. **Logic function :**

**shift\_en =** (key\_down[8'h12] | key\_down[8'h59])? 1'b1 : 1'b0

**cap\_en =** (key\_down[8'h58)? 1'b1 : 1'b0

* + 1. **Code :**

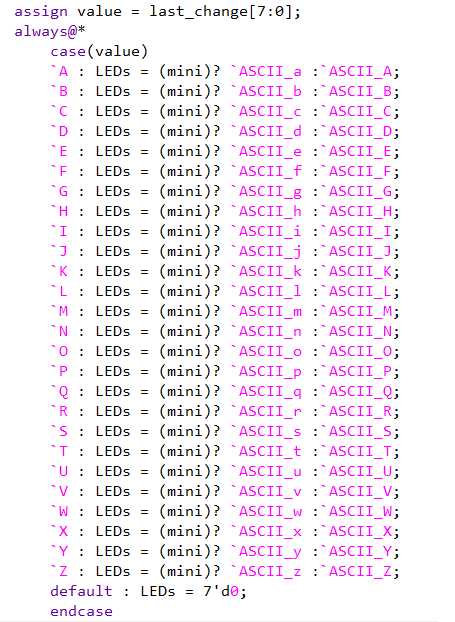


圖18 Lab08-4的decoder

* + 1. **Logic diagram :**

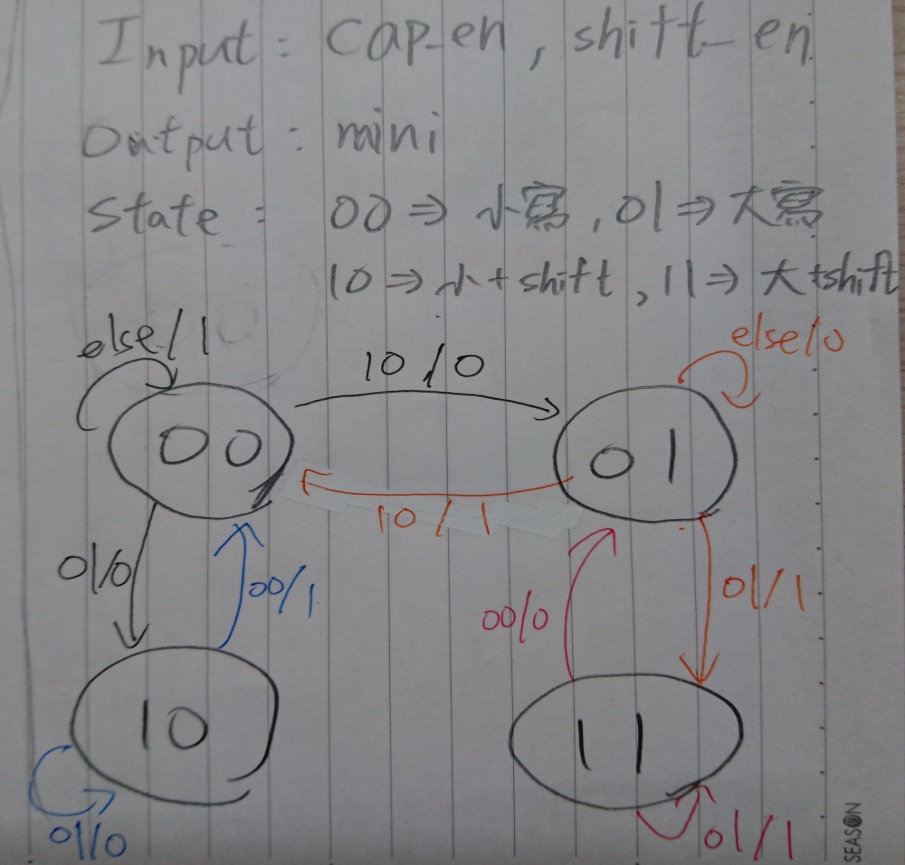


圖19 Lab08-4的FSM

* + 1. **Pin assignment :** 
       - 1. **Input :**

clk : W5

rst : R2

* + - * 1. **Output :**

mini : U16

LEDs[6] : L1

LEDs[5] : P1

LEDs[4] : P3

LEDs[3] : U3

LEDs[2] : W3

LEDs[1] : V3

LEDs[0] : V13

* + - * 1. **Inout :**

PS2\_CLK : C17

PS2\_DATA : B17

* 1. **Discussion :** 
     1. **整體運作過程 :** 
        + 1. 把shift\_en、cap\_en輸入到FSM，在不同的情況下，按下cap或shift都會讓改變大小寫，輸出mini來代表是否為小寫。
          2. 把last\_change和mini一起輸入到decoder block，先判斷是按到哪個字母，在判斷是小寫或是大寫，最後輸出LEDs來代表編號。
     2. **各block的構想 :** 
        + 1. **fsm :**

一開始預設為小寫狀態，若是按下Cap則會轉到大寫，再按一次就會轉回小寫。

當按下shift時，若此時小寫，就會轉到一個暫時為大寫的狀態；若此時大寫，就會轉到一個暫時為小寫的狀態。這和上面利用Cap轉換是不同的轉態。

而在該為小寫的狀態時，就把mini輸出1，反之，則輸出0。

* + - * 1. **decoder :**

把last\_change和mini一起輸入到decoder block，先判斷是按到哪個字母，在判斷是小寫或是大寫，最後輸出LEDs來代表編號。

* + 1. **過程中的Bug :**

因為cap\_en沒有經過one\_pulse化，當按下去時，會經過好幾次posedge，會讓FSM的狀態從小寫跳到大寫，又跳回小寫，所以有時會無法轉換大小寫。

1. **Conclusion :**

雖然很高興能夠外接鍵盤，不是只單單在使用板子，不過果然鍵盤沒有那麼好懂。

1. **Reference :**
   1. 老師給的實驗講義

讓我知道大概的架構是時麼。