Lab 08

105060016 謝承儒

一.Lab08-1

1. Design Specification :

A. **Input :** clk //100MHz 的 clk，用來驅動各個 block
rst //打開時，回到最初狀態

B. **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

C. **Inout :** PS2_CLK
PS2_DATE

D. **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

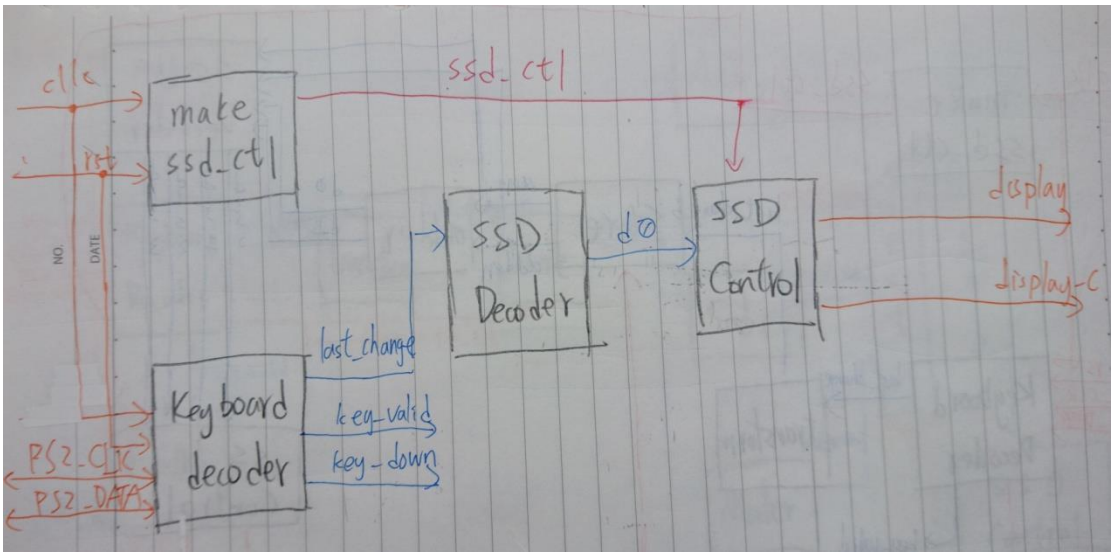

圖 1 Lab08-1 的區塊圖

2. Design Implementation :

A. Code :

```
module make_ssd_ctl(
    input clk,
    input rst,
    output [1:0] ssd_ctl
    );

    reg [26:0] cnt;

    assign ssd_ctl = cnt[15:14];

    always@( posedge clk  )
        cnt <= cnt + 1'b1;

endmodule
```

圖 2 Lab08-1 的 make_ssd_ctl

```
    assign value = last_change[7:0];
    assign refresh_en = (last_change==8'h1B) ||
    always@*
        begin
            case(value)
            8'h45: d0 <= 8'b00000011; //0
            8'h16: d0 <= 8'b10011111; //1
            8'h1E: d0 <= 8'b00100101; //2
            8'h26: d0 <= 8'b00001101; //3
            8'h25: d0 <= 8'b10011001; //4
            8'h2E: d0 <= 8'b01001001; //5
            8'h36: d0 <= 8'b01000001; //6
            8'h3D: d0 <= 8'b00011111; //7
            8'h3E: d0 <= 8'b00000001; //8
            8'h46: d0 <= 8'b00001001; //9
            8'h1C: d0 <= 8'b00010001; //A
            8'h1B: d0 <= 8'b01001001; //S
            9'h3A: d0 <= 8'b11010101; //M
            9'h5A: d0 <= 8'b11111110; //refresh
            default: d0 <= 8'b01110001;
            endcase
        end
```

圖 3 Lab08-1 的 SSD Decoder

```
module ssd_ctl(
    input [7:0] d0,
    input [1:0] ssd_ctl,
    output reg [7:0] display,
    output reg [3:0] display_c
    );

    always@*
        case(ssd_ctl)
        2'b00: display <= d0;
        2'b01: display <= 8'b1111_1111;
        2'b10: display <= 8'b1111_1111;
        2'b11: display <= 8'b1111_1111;
        default: display <= 8'b1111_1111;
        endcase

    always@*
        case(ssd_ctl)
        2'b00: display_c <= 4'b1110;
        2'b01: display_c <= 4'b1101;
        2'b10: display_c <= 4'b1011;
        2'b11: display_c <= 4'b0111;
        default: display_c <= 4'b1111;
        endcase
endmodule
```

圖 4 Lab08-1 的 SSD Control

B. Pin assignment :

a. Input :

1) clk = W5
2) rst = R2

b. Output :

1) display[0] = V7
2) display[1] = U7
3) display[2] = V5
4) display[3] = U5
5) display[4] = V8
6) display[5] = U8
7) display[6] = W6
8) display[7] = W7
9) display_c[0] = U2
10) display_c[1] = U4

11) display_c[2] = V4

12) display_c[3] = W4

c. Inout :

1) PS2_ CLK = C17
2) PS2_ DATA = B17

3. Discussion :

A. 整體運作過程 :

b. 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。

c. 將 last_change 輸入到 SSD_Decoder，經過解碼後得到顯示碼 (d0)，把它輸出。

d. 把 d0 輸入到 SSD_Control，再加上 ssd_ctl，就可以把 d0 顯示出來。

B. 各 block 的構想 :

a. 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。

b. 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。

c. 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。

C. 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。

二.Lab08-2：加法器

1. Design Specification :

A. Input : clk　　//輸入的頻率(100MHz) 。
rst　　//當=1 時，回到最初狀態

B. 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。

C. Inout : PS2_CLK
PS2_DATA

D. 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

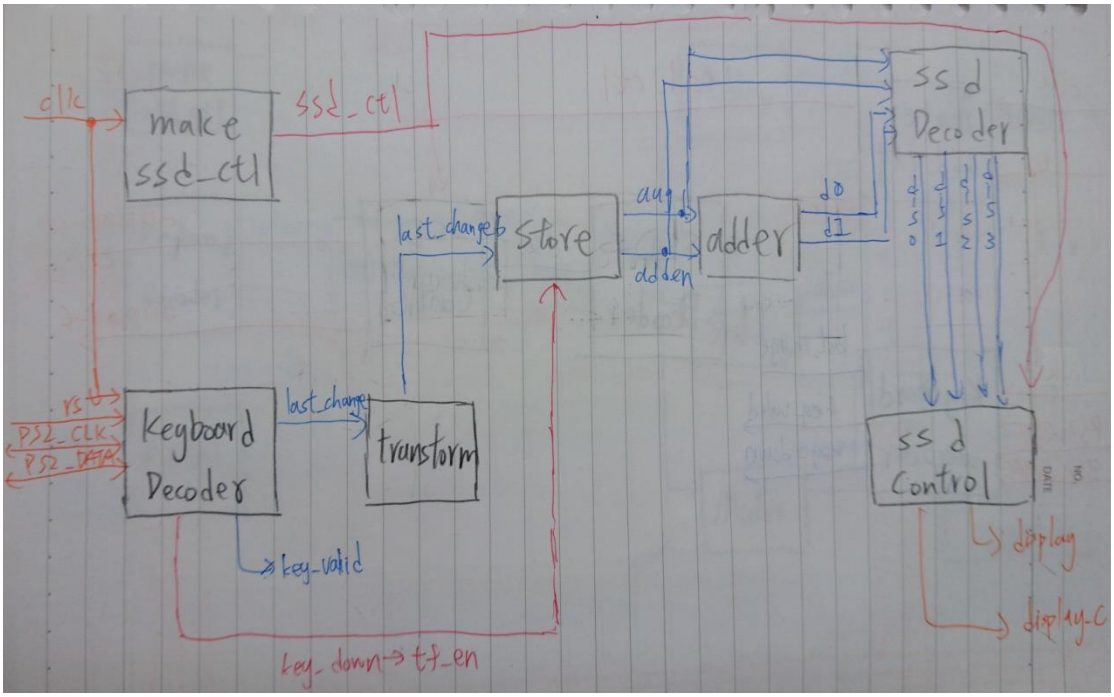

圖 5 Lab08-2 的區塊圖

2. Design Implementation :

A. 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]

B. Code :

```
assign value = last_change[7:0];

always@*
    case(value)
    8'h45: last_change_b <= 4'b0000; //0
    8'h16: last_change_b <= 4'b0001; //1
    8'h1E: last_change_b <= 4'b0010; //2
    8'h26: last_change_b <= 4'b0011; //3
    8'h25: last_change_b <= 4'b0100; //4
    8'h2E: last_change_b <= 4'b0101; //5
    8'h36: last_change_b <= 4'b0110; //6
    8'h3D: last_change_b <= 4'b0111; //7
    8'h3E: last_change_b <= 4'b1000; //8
    8'h46: last_change_b <= 4'b1001; //9
    default: last_change_b <= 4'b1111;
    endcase
```

圖 6 Lab08-2 的 transform

```
always@(posedge tf_en)
    if(choose_add)
    begin
        adden = last_change_b;
        choose_add = 1'b0;
    end
    else
    begin
        aug = last_change_b;
        choose_add = 1'b1;
    end
```

圖 7 Lab08-2 的 store

```
assign value = aug + adden;


always@(posedge clk or posedge rst)
    if(rst)
    begin
        d0 <= 4'd0;
        d1 <= 4'd0;
    end
    else
    begin
        d0 <= d0_tmp;
        d1 <= d1_tmp;
    end

always@*
    if(value < 10)
    begin
        d0_tmp <= value[3:0];
        d1_tmp <= 4'd0;
    end
    else
    begin
        d0_tmp <= value - 10;
        d1_tmp <= 4'd1;
    end
```

圖 8 Lab08-2 的 adder

C. Logic diagram

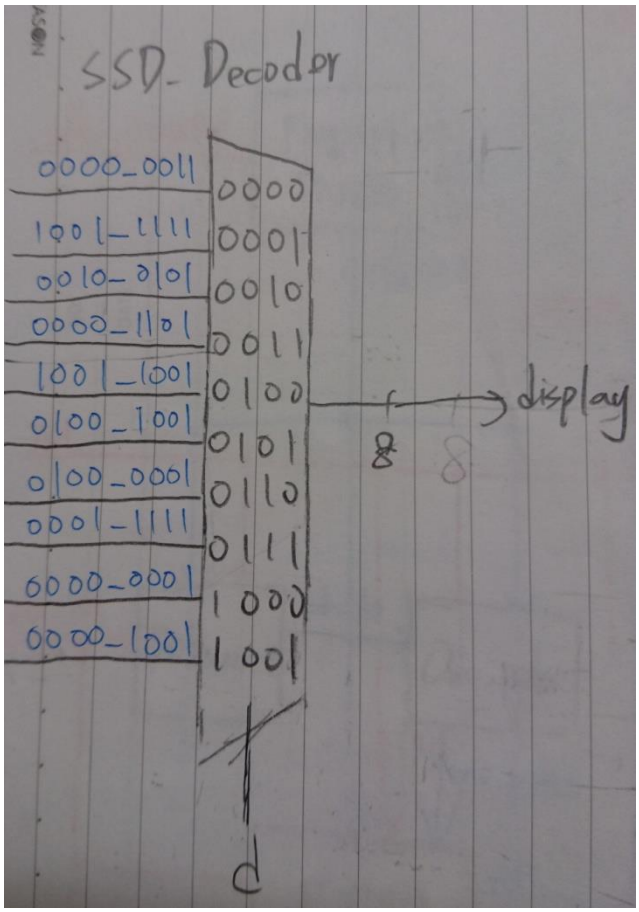

圖 9 Lab08-2 的 SSD_Decoder

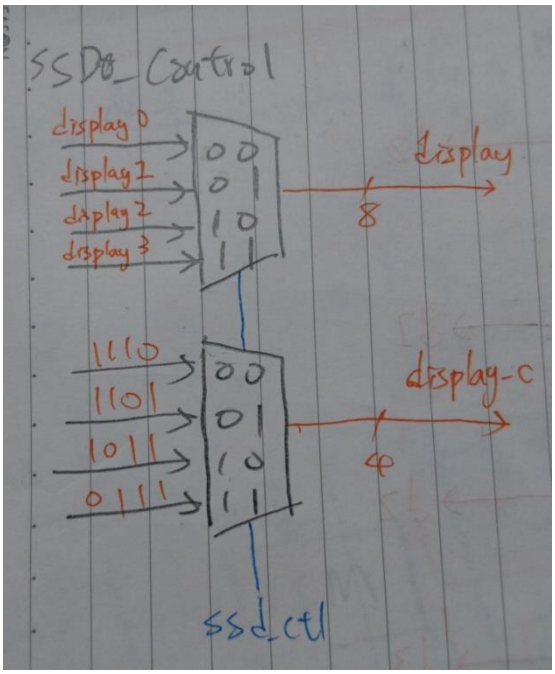

圖 10 Lab08-2 的 SSD_Control

D. Pin assignment :

a. Input :

1) clk : W5
2) rst : V17

b. Output :

1) display[0] = V7
2) display[1] = U7
3) display[2] = V5
4) display[3] = U5
5) display[4] = V8
6) display[5] = U8
7) display[6] = W6
8) display[7] = W7
9) display_c[0] = U2
10) display_c[1] = U4
11) display_c[2] = V4
12) display_c[3] = W4

c. Inout :

1) PS2_ CLK = C17
2) PS2_ DATA = B17

3. Discussion :

A. 整體運作過程 :

a. 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。

b. 把 last_change 輸入到 transform block 解碼成 4-bit BCD 形式的 last_change_b。

c. 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。

d. 把 aug、adden 輸入到 adder bolck，得到和的個位數和十位數 (d0、d1)。

e. 把 aug、adden、d0、d1 輸入到 SSD_Decoder，解碼成 8-bit 的顯示碼(dis3~0)。

f. 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)。

B. 新 block 的構想 ：

a. 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。

b. 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。

c. 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)。

d. 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。

C. 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。

三.Lab08-3： 加/減/乘法器

1. Design Specification :

A. Input : clk //100MHz 的 clk，用來驅動各個 block
rst //當=1 時，回到最初狀態

B. 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

C. Inout : PS2_CLK
PS2_DATA

D. 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

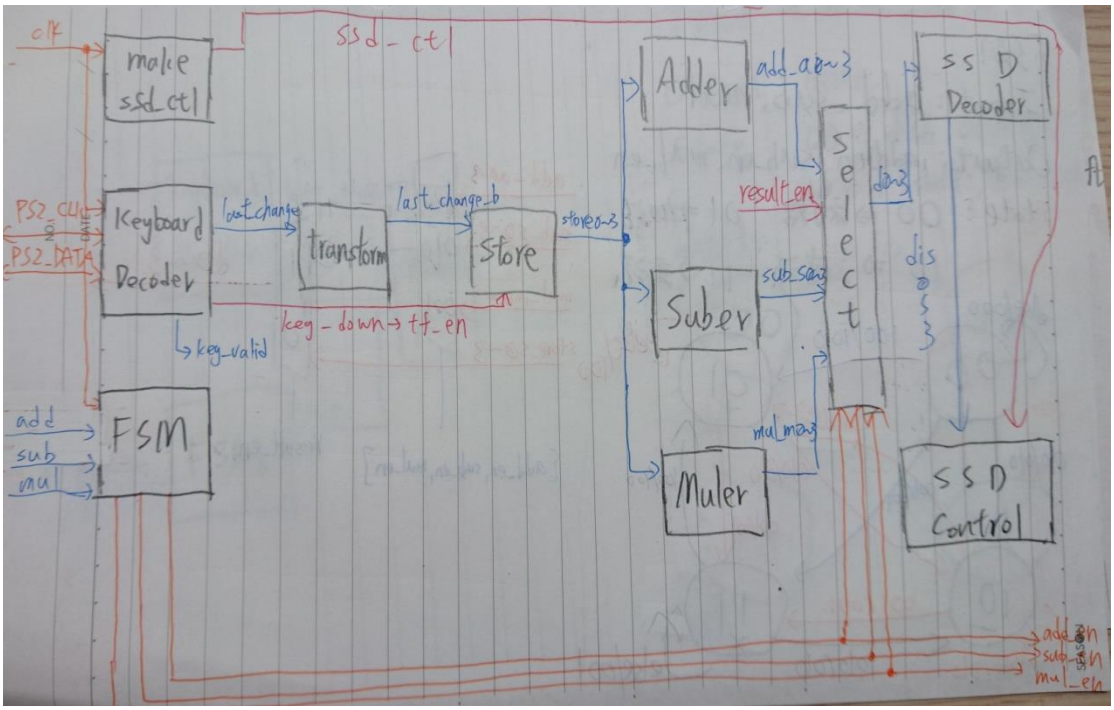

圖 11 Lab08-3 的區塊圖

2. Design Implementation :

A. Logic function :

a. **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]

b. **add =** (key_down[8'h79])? 1'b1 : 1'b0

c. **sub =** (key_down[8'h7B])? 1'b1 : 1'b0

d. **mul =** (key_down[8'h7C])? 1'b1 : 1'b0

e. **result_en =** (key_down[8'h5A])? 1'b1 : 1'b0

B. Code :

```
assign negative = (store_s3 < store_s1) | (store_s3 == store_s1) && (store_s2 < store_s0);
always@*
    if(negative)
    begin
        if(store_s2 > store_s0)
        begin
            sub_s0_tmp = store_s0 + 4'd10 - store_s2;
            br_ten = 1'b1;
        end
        else
        begin
            sub_s0_tmp = store_s0 - store_s2;
            br_ten = 1'b0;
        end
    end
    else if(store_s2 < store_s0)
    begin
        sub_s0_tmp = store_s2 + 4'd10 - store_s0;
        br_ten = 1'b1;
    end
    else
    begin
        sub_s0_tmp = store_s2 - store_s0;
        br_ten = 1'b0;
    end
```

圖 12 Lab08-3 的 suber(個位數)

```
always@*
    if(negative)
    begin
        sub_s1_tmp = store_s1 - store_s3 - br_ten;
    end
    else
    begin
        sub_s1_tmp = store_s3 - store_s1 - br_ten;
    end

always@*
    sub_s2_tmp = 4'd0;

always@*
    if(negative)
        sub_s3_tmp = 4'd15; //-
    else
        sub_s3_tmp = 4'd0;
```

圖 13 Lab08-3 的 suber(十位數)

```
assign value = (4'd10*store_s3 + store_s2) * (4'd10*store_s1 + store_s0);
assign mul_m0_tmp = value % 4'd10;
assign mul_m1_tmp = ((value - mul_m0_tmp) % 7'd100) / 4'd10;
assign mul_m2_tmp = ((value - mul_m1_tmp*4'd10 - mul_m0_tmp) % 11'd1000) / 7'd100;
assign mul_m3_tmp = (value - mul_m2_tmp*7'd100 - mul_m1_tmp*4'd10 - mul_m0_tmp) / 11'd1000;


always@(posedge clk or posedge rst)
    if(rst)
    begin
        mul_m0 = 4'd0;
        mul_m1 = 4'd0;
        mul_m2 = 4'd0;
        mul_m3 = 4'd0;
    end
    else
    begin
        mul_m0 = mul_m0_tmp;
        mul_m1 = mul_m1_tmp;
        mul_m2 = mul_m2_tmp;
        mul_m3 = mul_m3_tmp;
    end
```

圖 14 Lab08-3 的 muler

C. Logic diagram :

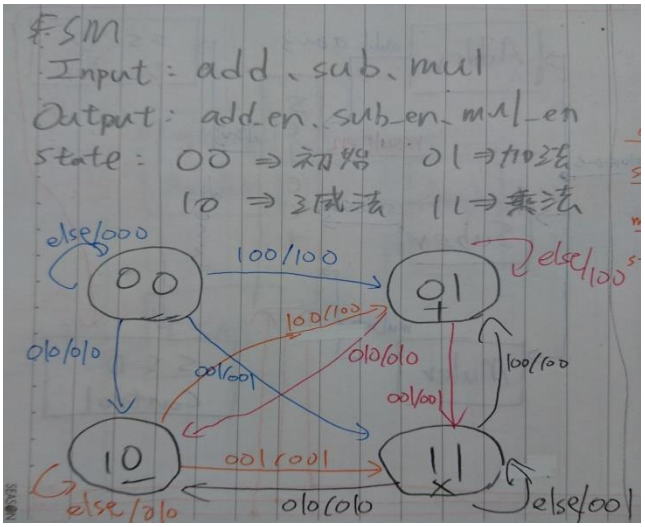

圖 15 Lab08-3 的 FSM

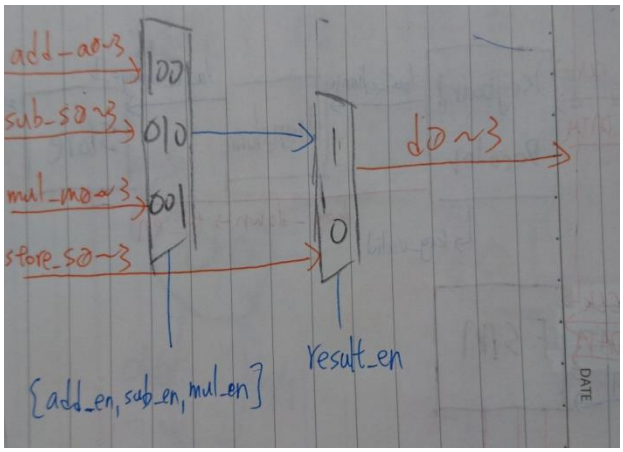

圖 16 Lab08-3 的 select_value

D. Pin assignment :

a. Input :

1) clk : W5
2) rst : R2

b. Output :

1) add_en = L1
2) sub_en = P1
3) mul_en = N3
4) display[0] = V7
5) display[1] = U7
6) display[2] = V5
7) display[3] = U5
8) display[4] = V8
9) display[5] = U8
10) display[6] = W6
11) display[7] = W7
12) display_c[0] = U2
13) display_c[1] = U4
14) display_c[2] = V4
15) display_c[3] = W4

c. Inout :

1) PS2_ CLK = C17
2) PS2_ DATA = B17

3. Discussion :

A. 整體運作過程 :

a. 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。

b. 把 last_change 輸入到 transform block 解碼成 4-bit BCD 形式的 last_change_b。

c. 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。

d. 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)。

e. 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。

f. 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)。

g. 把 d3~0 輸入到 SSD_Decoder 解碼成 8-bit 的顯示碼(dis3~0)。

h. 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)。

B. 新 block 的構想 ：

a. 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。

b. 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。

c. 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。

d. 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。

C. **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。

四.Lab08-4 : 字母大小寫

1. Design Specification :

A. **Input :** clk //100MHz 的 clk，用來驅動各個 block
rst //當=1 時，回到最初狀態
B. **Output :** mini //代表現在是否為小寫
[6:0]LEDs //用來顯示字母的編號
C. **Inout :** PS2_CLK
PS2_DATA
D. **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

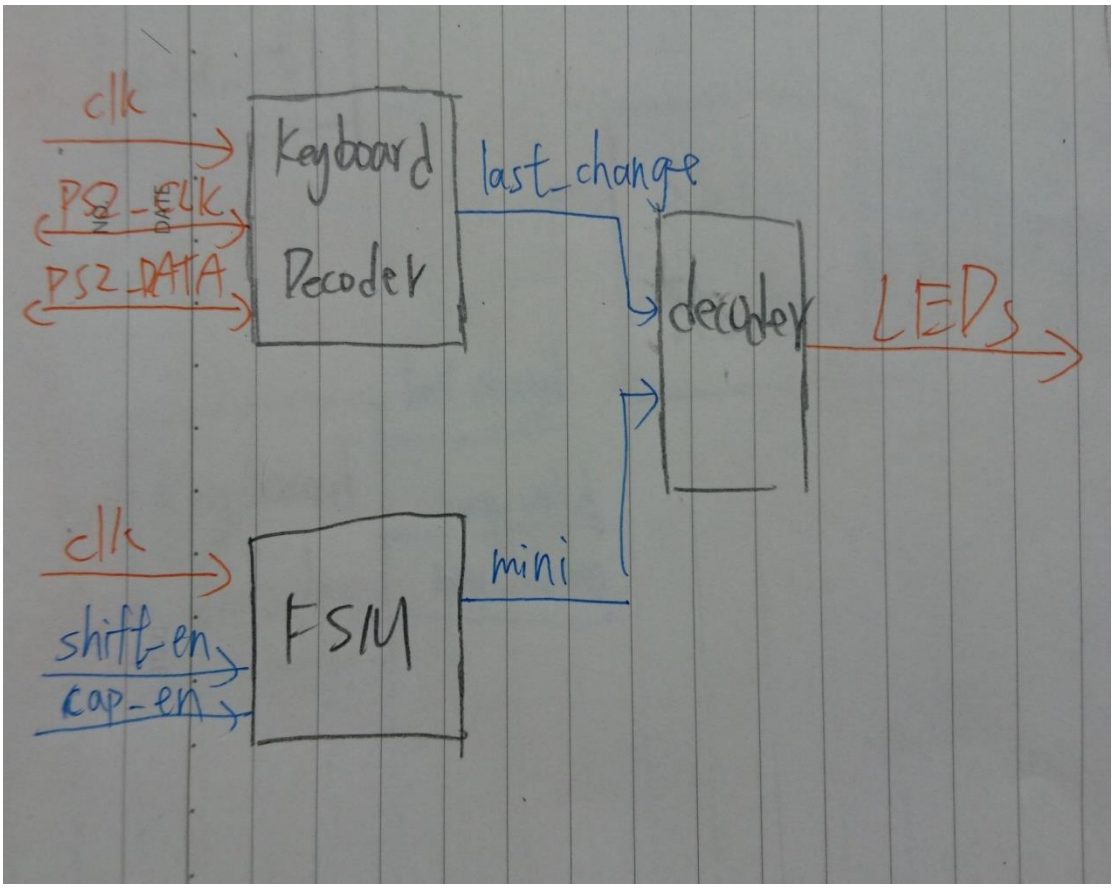

圖 17 Lab08-4 的區塊圖

2. Design Implementation :

A. Logic function :

1) **shift_en =** (key_down[8'h12] | key_down[8'h59])? 1'b1 : 1'b0
2) **cap_en =** (key_down[8'h58)? 1'b1 : 1'b0

B. Code :

```
assign value = last_change[7:0];
always@*
    case(value)
    `A : LEDs = (mini)? `ASCII_a :`ASCII_A;
    `B : LEDs = (mini)? `ASCII_b :`ASCII_B;
    `C : LEDs = (mini)? `ASCII_c :`ASCII_C;
    `D : LEDs = (mini)? `ASCII_d :`ASCII_D;
    `E : LEDs = (mini)? `ASCII_e :`ASCII_E;
    `F : LEDs = (mini)? `ASCII_f :`ASCII_F;
    `G : LEDs = (mini)? `ASCII_g :`ASCII_G;
    `H : LEDs = (mini)? `ASCII_h :`ASCII_H;
    `I : LEDs = (mini)? `ASCII_i :`ASCII_I;
    `J : LEDs = (mini)? `ASCII_j :`ASCII_J;
    `K : LEDs = (mini)? `ASCII_k :`ASCII_K;
    `L : LEDs = (mini)? `ASCII_l :`ASCII_L;
    `M : LEDs = (mini)? `ASCII_m :`ASCII_M;
    `N : LEDs = (mini)? `ASCII_n :`ASCII_N;
    `O : LEDs = (mini)? `ASCII_o :`ASCII_O;
    `P : LEDs = (mini)? `ASCII_p :`ASCII_P;
    `Q : LEDs = (mini)? `ASCII_q :`ASCII_Q;
    `R : LEDs = (mini)? `ASCII_r :`ASCII_R;
    `S : LEDs = (mini)? `ASCII_s :`ASCII_S;
    `T : LEDs = (mini)? `ASCII_t :`ASCII_T;
    `U : LEDs = (mini)? `ASCII_u :`ASCII_U;
    `V : LEDs = (mini)? `ASCII_v :`ASCII_V;
    `W : LEDs = (mini)? `ASCII_w :`ASCII_W;
    `X : LEDs = (mini)? `ASCII_x :`ASCII_X;
    `Y : LEDs = (mini)? `ASCII_y :`ASCII_Y;
    `Z : LEDs = (mini)? `ASCII_z :`ASCII_Z;
    default : LEDs = 7'd0;
    endcase
```

圖 18 Lab08-4 的 decoder

C. Logic diagram :

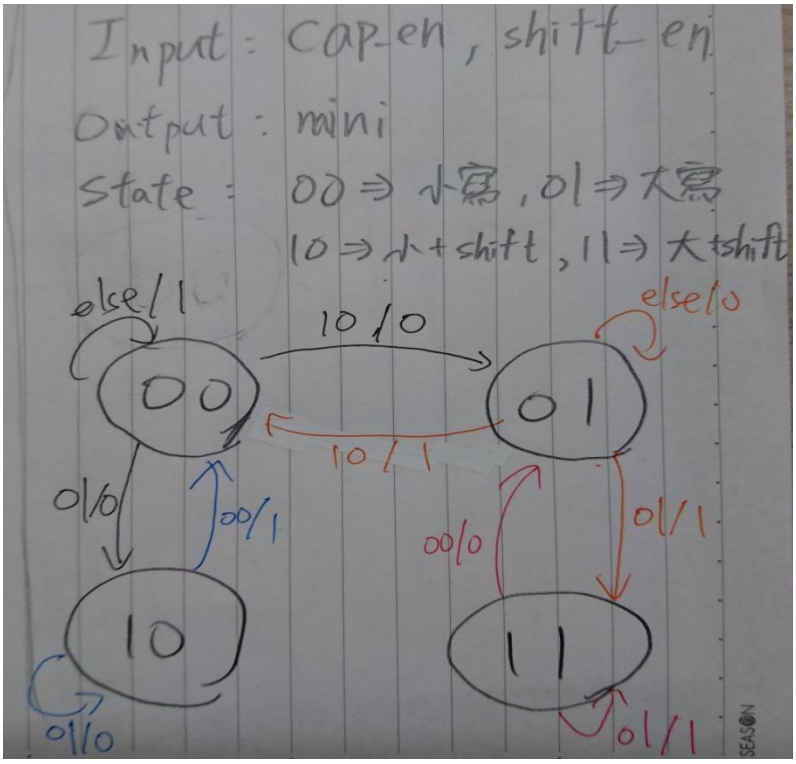

圖 19 Lab08-4 的 FSM

D. Pin assignment :

a. Input :

1) clk : W5
2) rst : R2

b. Output :

1) mini : U16
2) LEDs[6] : L1
3) LEDs[5] : P1
4) LEDs[4] : P3
5) LEDs[3] : U3
6) LEDs[2] : W3
7) LEDs[1] : V3
8) LEDs[0] : V13

c. Inout :

1) PS2_CLK : C17
2) PS2_DATA : B17

3. Discussion :

A. 整體運作過程 :

a. 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。

b. 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。

B. 各 block 的構想 :

a. 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。

b. 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。

C. 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。

五.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。

六.Reference :

1. 老師給的實驗講義
   讓我知道大概的架構是時麼。