LAB 09

前言: 都只討論比較重要的 MODULE，重複的也不做贅述

第一題

DESIGN SPECIFICATION

(1) INPUT / OUTPUT

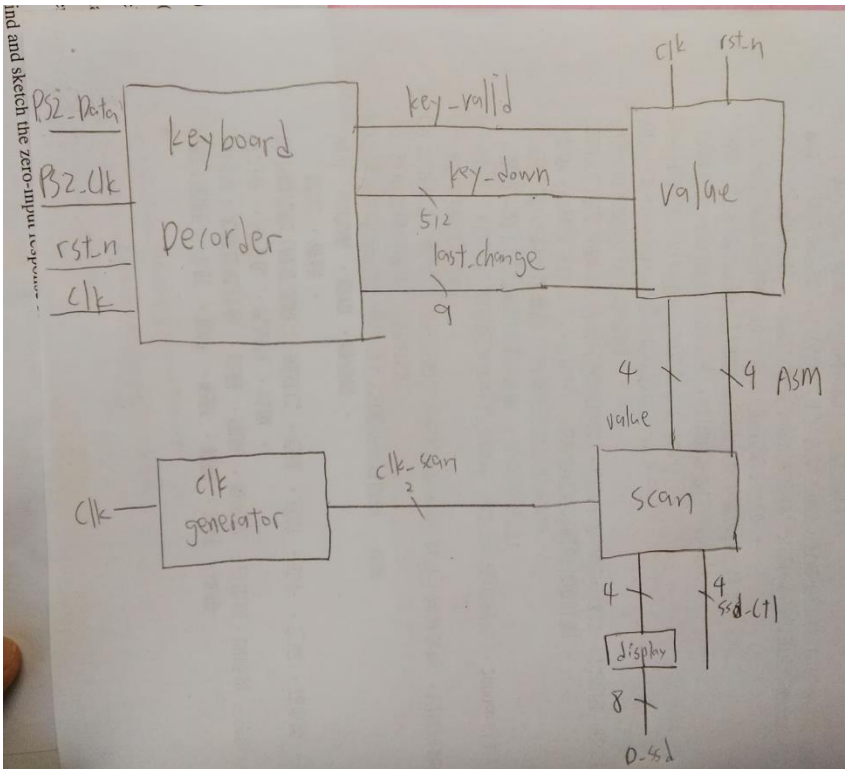
```
module top(clk, rst_n, ssd_ctl, D_ssd, PS2_DATA, PS2_CLK);
    input clk;
    input rst_n;
    inout PS2_DATA;
    inout PS2_CLK;
    output [3:0]ssd_ctl;
   output [7:0]D_ssd;

module KeyboardDecoder
    output reg [511:0] key_down,
    output wire [8:0] last_change,
    output reg key_valid,
    inout wire PS2_DATA,
    inout wire PS2_CLK,
    input wire rst,
    input wire clk
module KeyboardCtrl#
module Ps2Interface#
```

複製老師的範例

```
module value(value, ASM, key_down, last_change, rst_n,
key_valid, clk);
    output reg [3:0]value;
    output reg [3:0]ASM;
    input [511:0]key_down;
    input [8:0]last_change;
    input rst_n;
    input key_valid;
    input clk;
```

reg [3:0]value_temp;
reg [3:0]ASM_TEMP;

(2) BLOCK DIAGRAM



DESIGN IMPLEMENTATION

(1) I / O PINS

D_ssd[7]	OUT		W7
D_ssd[6]	OUT		W6
D_ssd[5]	OUT		U8
D_ssd[4]	OUT		V8
D_ssd[3]	OUT		U5
D_ssd[2]	OUT		V5
D_ssd[1]	OUT		U7
D_ssd[0]	OUT		V7
ssd_ctl (4)	OUT		
ssd_ctl[3]	OUT		W4
ssd_ctl[2]	OUT		V4
ssd_ctl[1]	OUT		U4
ssd_ctl[0]	OUT		U2
Scalar ports (4)			
clk	IN		W5
PS2_CLK	INOUT		C17
PS2_DATA	INOUT		B17
rst_n	IN		V17

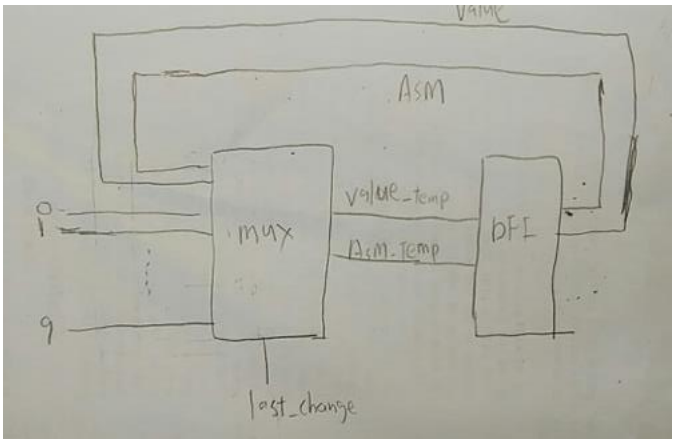
(2) VERILOG CODE

Module top

```
KeyboardDecoder U0(.key_down(key_down), .last_change(last_change), .key_valid(key_valid), .PS2_DATA(PS2_DATA),
                   .PS2_CLK(PS2_CLK), .rst(~rst_n), .clk(clk));
clk_generator U1(.clk_scan(clk_scan), .clk(clk));
value U2(.clk(clk), .value(value), .ASM(ASM), .key_down(key_down), .key_valid(key_valid), .last_change(last_change), .rst_n(rst_n));
scan U3(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .value(value), .ASM(ASM), .control(clk_scan));
display U4(.D_ssd(D_ssd), .in(ssd_in));
```

依 bolck diagram 接

Module value



用 last_change 的值去選擇 value(0~9)跟 ASM(+, - , x)的值

並用 DFF 存值

如果有按[if(key_valid)]就用 mux 選，否則維持原來的值

我把數字設在 7ssd 的第三位，加減乘設在第四位

加是 0, 減是 1, 乘是 2

然後按下 enter 的話就讓他 value, ASM 等於 15，因為 display 中

只要超過 9 就會全暗，rst_n 設成 15 也是同理

DISSCUSSION

第一題沒有很難，只是要先搞懂老師給的 Keyboarddecorder 的 input, output 到底在幹嘛，知道之後就很快完成了。

執行結果就是第三位顯示數字，第四位顯示加減乘，按下去就會一直維持直到下一次按，按下 enter 或 rst_n 就全暗。

CONCLUSION

這次真的花很久時間才搞懂 KEY_VALID, LAST_CHANGE, KEY_DOWN 的功能，因為上課其實沒聽很懂

第二題

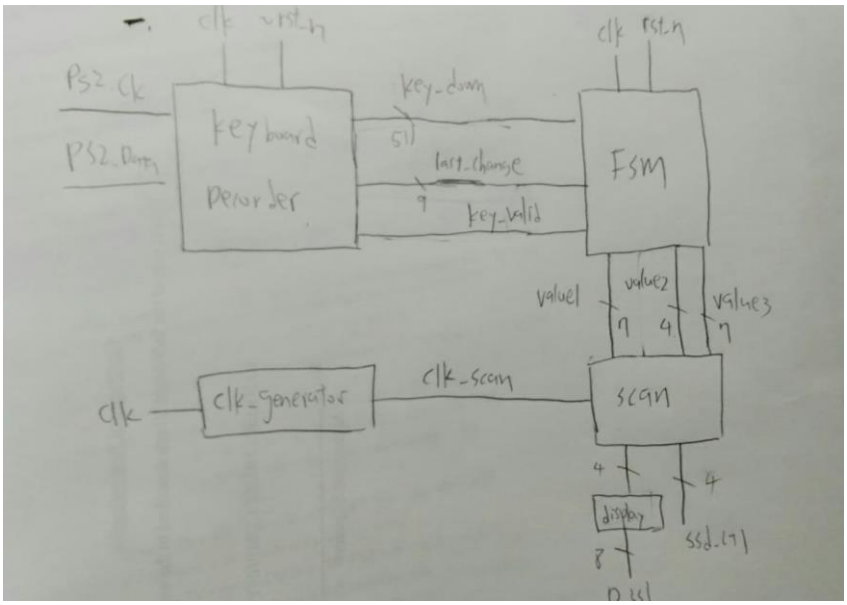
DESIGN SPECIFICATION

(1) INPUT/ OUTPUT

```
module top(clk, rst_n, ssd_ctl, D_ssd, PS2_DATA, PS2_CLK);
    input clk;
    input rst_n;
    inout PS2_DATA;
    inout PS2_CLK;
    output [3:0]ssd_ctl;
   output [7:0]D_ssd;

module FSM(clk, rst_n, key_valid, key_down, last_change, value1,
value2, value3);
    input clk;
    input rst_n;
    input key_valid;
    input [511:0]key_down;
    input [8:0]last_change;
    output reg [6:0]value1;
    output reg [3:0]value2;
```

```
output reg [6:0]value3;

module scan(ssd_ctl, ssd_in, value1, value2, value3, control);
    output reg [6:0] ssd_in;
    output reg [3:0] ssd_ctl;
    input [6:0]value1;
    input [3:0]value2;
    input [6:0]value3;
   input [1:0] control;
```

(2) BLOCK DIAGRAM



DESIGN IMPLEMENTATION

(1) I / O PINS

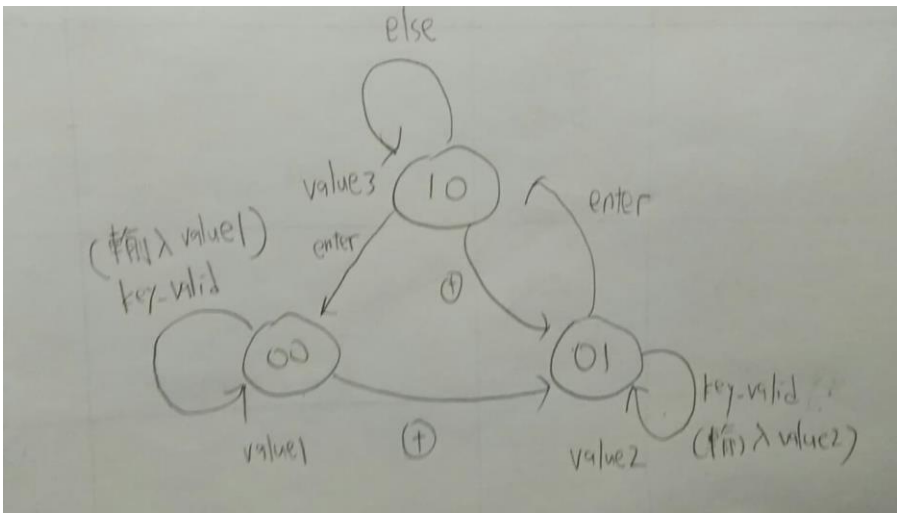
D_ssd[7]	OUT		W7
D_ssd[6]	OUT		W6
D_ssd[5]	OUT		U8
D_ssd[4]	OUT		V8
D_ssd[3]	OUT		U5
D_ssd[2]	OUT		V5
D_ssd[1]	OUT		U7
D_ssd[0]	OUT		V7
ssd_ctl (4)	OUT		
ssd_ctl[3]	OUT		W4
ssd_ctl[2]	OUT		V4
ssd_ctl[1]	OUT		U4
ssd_ctl[0]	OUT		U2
Scalar ports (4)			
clk	IN		W5
PS2_CLK	INOUT		C17
PS2_DATA	INOUT		B17
rst_n	IN		V17

(2) VERILOG CODE

Module top

```
KeyboardDecoder U0(.key_down(key_down), .last_change(last_change), .key_valid(key_valid), .PS2_DATA(PS2_DATA),
                   .PS2_CLK(PS2_CLK), .rst(~rst_n), .clk(clk));
clk_generator U1(.clk_scan(clk_scan), .clk(clk));
FSM U2(.clk(clk), .value1(value1),  .value2(value2),  .value3(value3), .key_down(key_down),
        .key_valid(key_valid), .last_change(last_change), .rst_n(rst_n));
scan U3(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .value1(value1), .value2(value2),  .value3(value3), .control(clk_scan));
display U4(.D_ssd(D_ssd), .in(ssd_in));
```

Module FSM



00 狀態是輸入 value1(第一個加數)，按下+進入 01 state

如果有按數字，則 assign 那個數字給 value1，state 留在 00

01 狀態是輸入是第二個加數(value2)，按下 enter 進入 10 state

如果有按數字，則 assign 那個數字給 value2，state 留在 01

10 狀態是結果(value3)，按下 ＋ 回到 state 01 可做繼續累加，

按下 enter 把全部歸 0 並回到 state 00。

Module scan

```
case(control)
    2'b00:
      begin
        ssd_ctl = 4'b1110;
        ssd_in = value3 % 4'd10;
      end
    2'b01:
      begin
        ssd_ctl = 4'b1101;
        ssd_in = value3 / 4'd10;
      end
    2'b10:
      begin
        ssd_ctl = 4'b1011;
        ssd_in = value2;
      end
    2'b11:
      begin
        ssd_ctl = 4'b0111;
        ssd_in = value1;
      end
    default:
      begin
        ssd_ctl = 4'b1111;
        ssd_in = 4'b1111;
```

讓第一二位分別當加數，三四位當結果

DICUSSION

這題一開始本來沒甚麼想法，結果寫個 FSM 好像就出來

了，這題碰到的問題是我又把 VERILOG 當軟體寫，像在 00

STATE 只有對 value1 做輸入，所以我並沒有寫

value2_temp = 4'd0;
value3_temp = 4'd0;

我想說 value2, 3 沒有要動，就不寫，導致他們的值會亂跳，

可是都寫上去行數會很多。

執行的結果就是輸入第一個數，按下加，輸入第二個，按下

+就累加，或按下 enter 重製

CONCLUSION

前 2 題我都用 key_vaid 的訊號來判斷有沒有按下去，但其

實我誤解他的訊號長怎樣，導致我第 3 題遇到了問題。

第三題

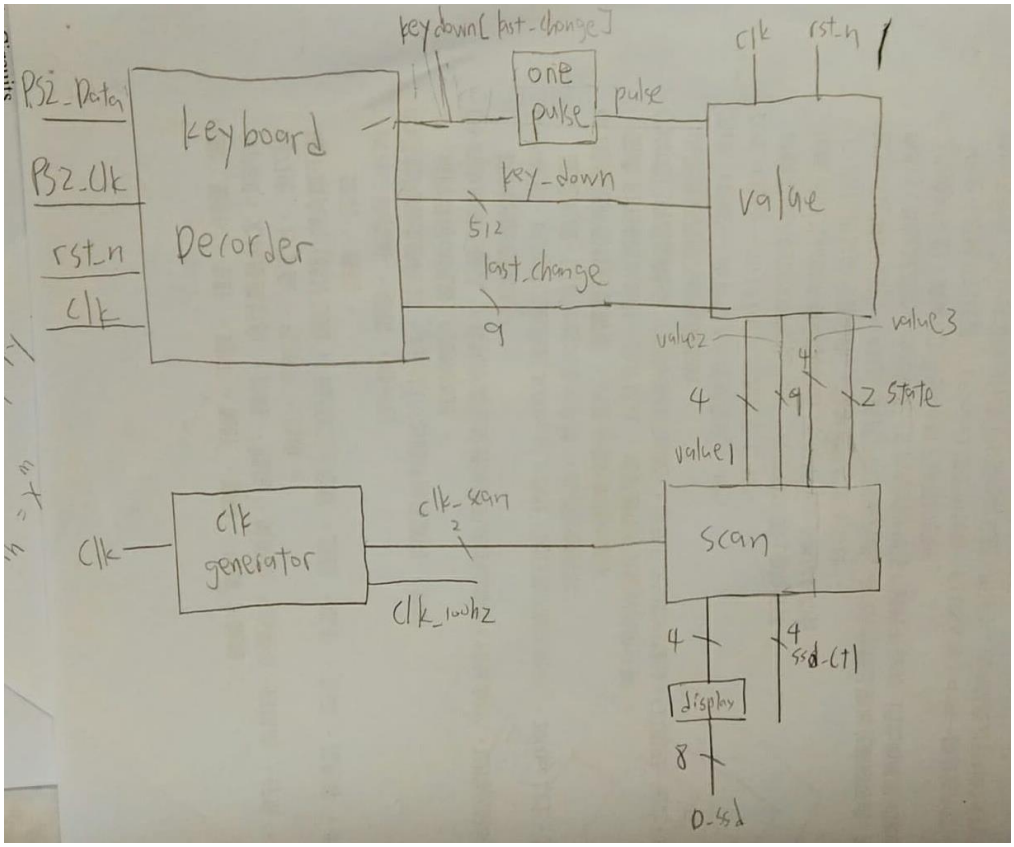
DESIGN SPECIFICATION

(1) INPUT / OUTPUT

```
module top(clk, rst_n, ssd_ctl, D_ssd, PS2_DATA, PS2_CLK);
 input clk;
 input rst_n;
 inout PS2_DATA;
 inout PS2_CLK;
 output [3:0]ssd_ctl;
 output [7:0]D_ssd;

module FSM(clk, rst_n, press, key_down, last_change, value1,
value2, value3, state);
 input clk;
 input rst_n;
 input press;
```

```
input [511:0]key_down;
input [8:0]last_change;
output reg [13:0]value1;
output reg [6:0]value2;
output reg [13:0]value3;
output reg [1:0]state;

module scan(ssd_ctl, ssd_in, value1, value2, value3, control,
state, clk, rst_n);
output reg [3:0] ssd_in;
output reg [3:0] ssd_ctl;
input [13:0]value1;
input [6:0]value2;
input [13:0]value3;
input [1:0] control;
input [1:0]state;
input clk;
input rst_n;
```

(2) BLOCK DIAGRAM



DESIGN IMPLEMENTATION

(1) I / O PINS

D_ssd[7]	OUT		W7
D_ssd[6]	OUT		W6
D_ssd[5]	OUT		U8
D_ssd[4]	OUT		V8
D_ssd[3]	OUT		U5
D_ssd[2]	OUT		V5
D_ssd[1]	OUT		U7
D_ssd[0]	OUT		V7
ssd_ctl (4)	OUT		
ssd_ctl[3]	OUT		W4
ssd_ctl[2]	OUT		V4
ssd_ctl[1]	OUT		U4
ssd_ctl[0]	OUT		U2
Scalar ports (4)			
clk	IN		W5
PS2_CLK	INOUT		C17
PS2_DATA	INOUT		B17
rst_n	IN		V17

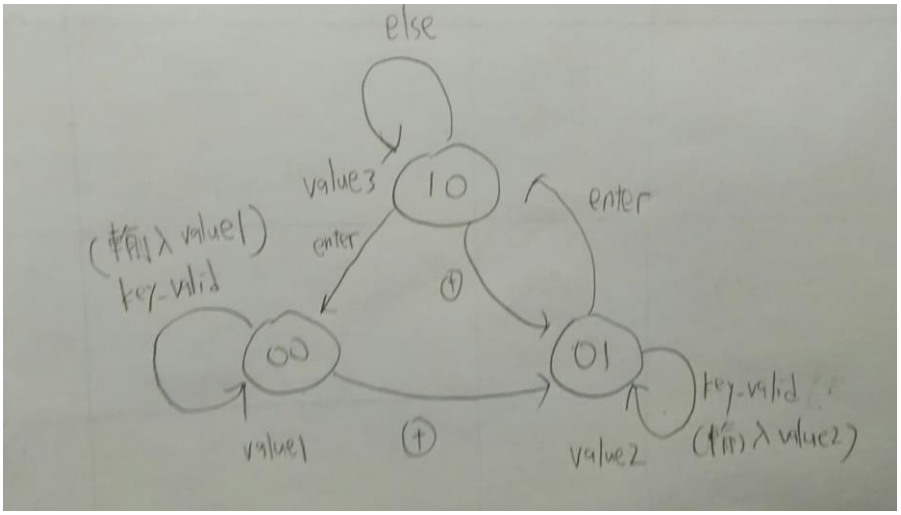
(2) VERILOG CODE

Module top

```
KeyboardDecoder U0(.key_down(key_down), .last_change(last_change), .key_valid(), .PS2_DATA(PS2_DAT
                   .PS2_CLK(PS2_CLK), .rst(~rst_n), .clk(clk));
clk_generator U1(.clk_scan(clk_scan), .clk(clk));
one_pulse U2(.clk(clk), .in_trig(key_down[last_change]), .out_pulse(pulse), .rst_n(rst_n));
FSM U3(.clk(clk), .value1(value1),  .value2(value2),  .value3(value3), .key_down(key_down), .press(pulse),
        .last_change(last_change), .rst_n(rst_n), .state(state), .neg(neg));
scan U4(.clk(clk), .rst_n(rst_n), .ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .value1(value1), .value2(value2),
        .value3(value3), .control(clk_scan), .state(state));
display U5(.D_ssd(D_ssd_temp), .in(ssd_in));

always@*
   if (neg && state == 2'd2) D_ssd = D_ssd_temp - 1'b1;
   else D_ssd = D_ssd_temp;
```

上面的就是按照 block diagram 接，下面的就用 neg(判斷數否為負數)來做處理，我的負數表示法是讓 7ssd 右下的那個點亮，所以如果是負數，然後 state 是 2(output 結果的 state)，就讓點亮

Module FSM



這題的原理和上一題基本上一樣，00, 01 state 是第一二個加數，10 是結果，只是因為要做 2 位數、加減乘還有負數，所以多了 count(控制個位十位)，ASD(控制加減乘)還有 neg(判斷是否為負數)來操控。另外，我本來用 key_valid 是否按下去，現在改用 press，是把 key_down[last_change]經過 one_pulse 處理後的訊號，原因稍後說明。

在 00 state 時，若按下+, -, *，就進到 01 state，然後如果按

加 ASD = 0, 減 ASD = 1, 乘 ASD = 2。

到 01 STATE 然後按下 ENTER 輸出結果時，根據 ASD 的值決定要加,減,乘

控制數入二位數是用 count，一開始 count = 0

當第一次按下數字(count == 0)，value 就等於那個數字，並讓 count = 1;

當第二次按下去(count == 1)，就讓原先 value 的值乘 10，再加上按的數字，然後再把 count 重製為 0，每當進入到下一個 state，count 也會重製為 0。

然後因為還要考慮負數，所以在 2 個地方有做額外處理

第一個是在 01 state 按下 enter，準備 output 出結果那裏

當 value1 被減數小於減數 value2，就變一次號(藍色部分)

並讓結果等於 value2 減 value1

```
if (ASD == 0) begin
    value3_temp = value1 + value2;
    neg_temp = neg;
end
else if (ASD == 1) begin
    if (value1 < value2) begin
        value3_temp = value2 - value1;
        neg_temp = ~neg;
    end
    else begin
        value3_temp = value1 - value2;
        neg_temp = neg;
    end
end
else begin
    value3_temp = value1 * value2;
    neg_temp = neg;
end
```

第二個地方是 10 state 按下加減乘準備做累積計算

因為如果是在負數狀態，加的反而對負數是減，減的反而是加，乘不變

```
else if (press & (key_down[9'h079] || key_down[9'h07B] || key_down[9'h07C])) begin
    value1_temp = value3;
    value2_temp = 4'd0;
    value3_temp = 4'd0;
    if (key_down[9'h079])
        if (neg) ASD_TEMP = 2'd1;
        else ASD_TEMP = 2'd0;
    else if (key_down[9'h07B])
        if (neg) ASD_TEMP = 2'd0;
        else ASD_TEMP = 2'd1;
    else ASD_TEMP = 2'd2;
    count_next = 2'b0;
    neg_temp = neg;
    next_state = 2'b01;
end
```

Module scan

我把 state 接到 scan，以控制螢幕當下要顯示什麼

Value3 做了偏向軟體語言的處理，幸好 verilog 可以接受

```
case(state)
    2'b00: begin
        cnt4_temp = 4'd0;
        cnt3_temp = 4'd0;
        cnt2_temp = value1 / 4'd10;
        cnt1_temp = value1 % 4'd10;
    end
    2'b01: begin
        cnt4_temp = 4'd0;
        cnt3_temp = 4'd0;
        cnt2_temp = value2 / 4'd10;
        cnt1_temp = value2 % 4'd10;
    end
    2'b10: begin
      cnt4_temp = value3 / 10'd1000;
      cnt3_temp = (value3 % 10'd1000) / 10'd100;
      cnt2_temp = (value3 % 10'd100) / 4'd10;
      cnt1_temp = value3 % 4'd10;
    end
    default: begin
        cnt1_temp = 4'd15;
        cnt2_temp = 4'd15;
        cnt3_temp = 4'd15;
        cnt4_temp = 4'd15;
    end
```

DISSCUSSION

這次遇到了 3 個問題，第一個是我按下某個數字，他的十位跟個位都會變一樣或只有個位變，假如我按 1，他就會變 11 或 1，這是隨機的，我就知道這是因為訊號重複觸發，於是我引用以前的 one_pulse，把 key_valid 整理乘 one_pulse，結果沒用，後來才發現 key_valid 是以 clk 的周期上下震盪的訊號，所以沒用，所以我就把按下去的期間一直為 1 的 key_down[last_change]整理成 one_pulse，這樣才會壓一次，訊號只收到一次。

第二個是個低級錯誤，但卻花了我很久因為我一直以為是 FSM 有問題，結果只是我在主程式忘記把 state 接出來，導致結果錯誤。

第三個在做負數的部分時，在負數狀態的時候加要變減，減要變加的那裏寫錯地方了，應該要寫在 10 state，我當時腦袋昏昏的，寫在 00 state。

執行結果就是 2 位數的加減乘，並可累積繼續計算，包含負數。

CONCLUSION

我覺得我犯那種錯誤不少次了，真的很費時，以後直接把 input/output 那一行複製到主程式，應該能減少此種錯誤。

這次原本想加減乘各做一個 state，但後來發現這樣行數會爆多，所以想了加變數 ASD 存值這個辦法。

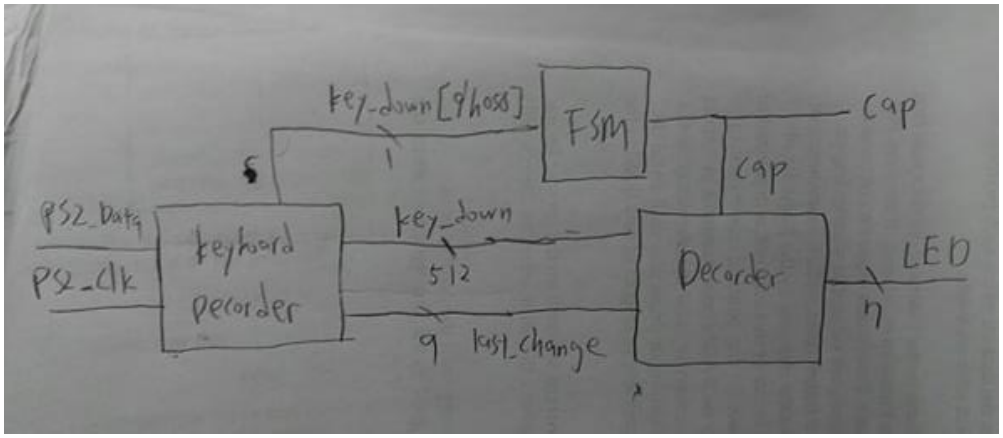
第四題

DESIGN SPECIFICATION

(1) INPUT / OUTPUT

```
module top(clk, rst_n, LED, LED_CAP, PS2_DATA, PS2_CLK);
        input clk;
        input rst_n;
        inout PS2_DATA;
        inout PS2_CLK;
        output [6:0]LED;
       output LED_CAP;

module FSM_CAP(in, rst_n, state);
        input in;
        input rst_n;
       output reg state;

module DECORDER(LED, last_change, key_down, CAP, clk,
rst_n);
        input clk;
        input rst_n;;
        input [8:0]last_change;
        input [511:0]key_down;
        input CAP;
        output reg [6:0]LED;
       reg [6:0]LED_TEMP;
```

(2) BLOCK DIAGRAM



DESIGN IMPLEMENTATION

(1) I / O PINS

LED[6]	OUT	U14
LED[5]	OUT	U15
LED[4]	OUT	W18
LED[3]	OUT	V19
LED[2]	OUT	U19
LED[1]	OUT	E19
LED[0]	OUT	U16
Scalar ports (5)		
clk	IN	W5
LED_CAP	OUT	L1
PS2_CLK	INOUT	C17
PS2_DATA	INOUT	B17
rst_n	IN	V17

(2) VERILOG CODE

Module top

```
KeyboardDecoder U0(.key_down(key_down), .last_change(last_change), .key_valid(), .PS2_DATA(PS2_DATA),
                   .PS2_CLK(PS2_CLK), .rst(~rst_n), .clk(clk));
FSM_CAP U1(.in(key_down[9'h058]), .rst_n(rst_n), .state(CAP));
DECORDER U2(.clk(clk), .rst_n(rst_n), .LED(LED), .last_change(last_change), .key_down(key_down), .CAP(CAP));
assign LED_CAP = CAP;
```

Module FSM_CAP

這次我懶得用 one_pulse，加上 CAP 只有 1 跟 0 兩個 STATE

，就直接拿 in 當 clk 了

```
always@(posedge in or negedge rst_n)
    if (~rst_n) state <= 0;
    else state <= ~state;
```

Module DECORDER

這個就是普通的 DECORDER，按下相應的字母給代碼，然後分

四種情形

	CAP	~CAP
shift	小寫	大寫
~shift	大寫	小寫

DISSCUSSION

執行結果就是按下字母，顯示代碼，根據 SHIFT 和 CAP 決定大小寫

CONCLUSION

這題不難，只是那個 DECORDER 真的要做有夠久阿!