LAB07

前言: 只討論重要的 MODULE

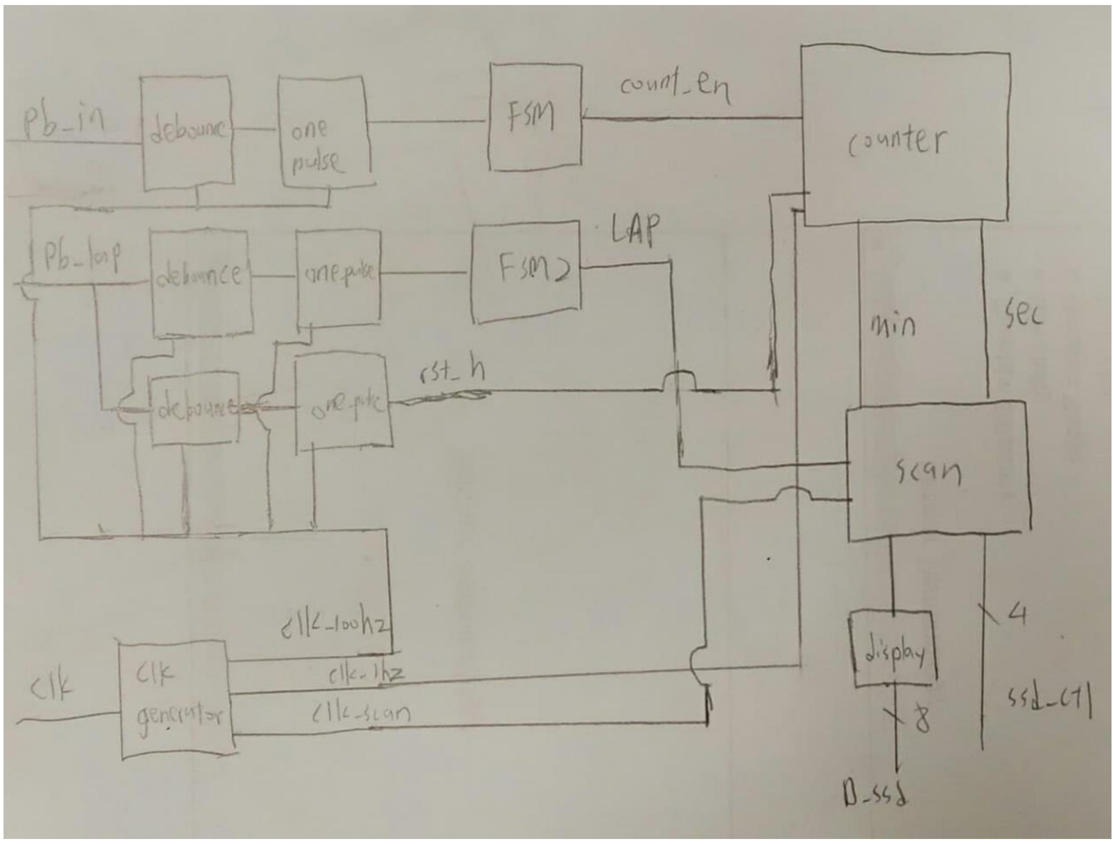
第一題

DESIGN SPECIFICATION

(1) INPUT / OUTPUT

```
module top(clk, pb_in, pb_lap, ssd_ctl, D_ssd);
    input clk;
    input pb_in;
    input pb_lap;
    output [3:0]ssd_ctl;
   output [7:0]D_ssd;

module FSM(count_en, in, rst_h);
    output count_en;
    input in;
    input rst_h;
   reg state;

module FSM2(in, rst_h, state);
    input in;
    input rst_h;
   output reg state;

module BCD_UP(limit, clk, rst_h, add, q, borrow);
    input [3:0]limit;
    input clk;
    input rst_h;
    input add;
    output reg [3:0]q;
    output reg borrow;
   reg [3:0]q_temp;
module scan(ssd_ctl, ssd_in, sec1, sec2, min1, min2, control, enable,
rst_h);
    output reg [3:0] ssd_in;
    output reg [3:0] ssd_ctl;
    input [3:0]sec1;
input [3:0]sec2;
input [3:0]min1;
input [3:0]min2;
input [1:0] control;
input enable;
input rst_h;
```

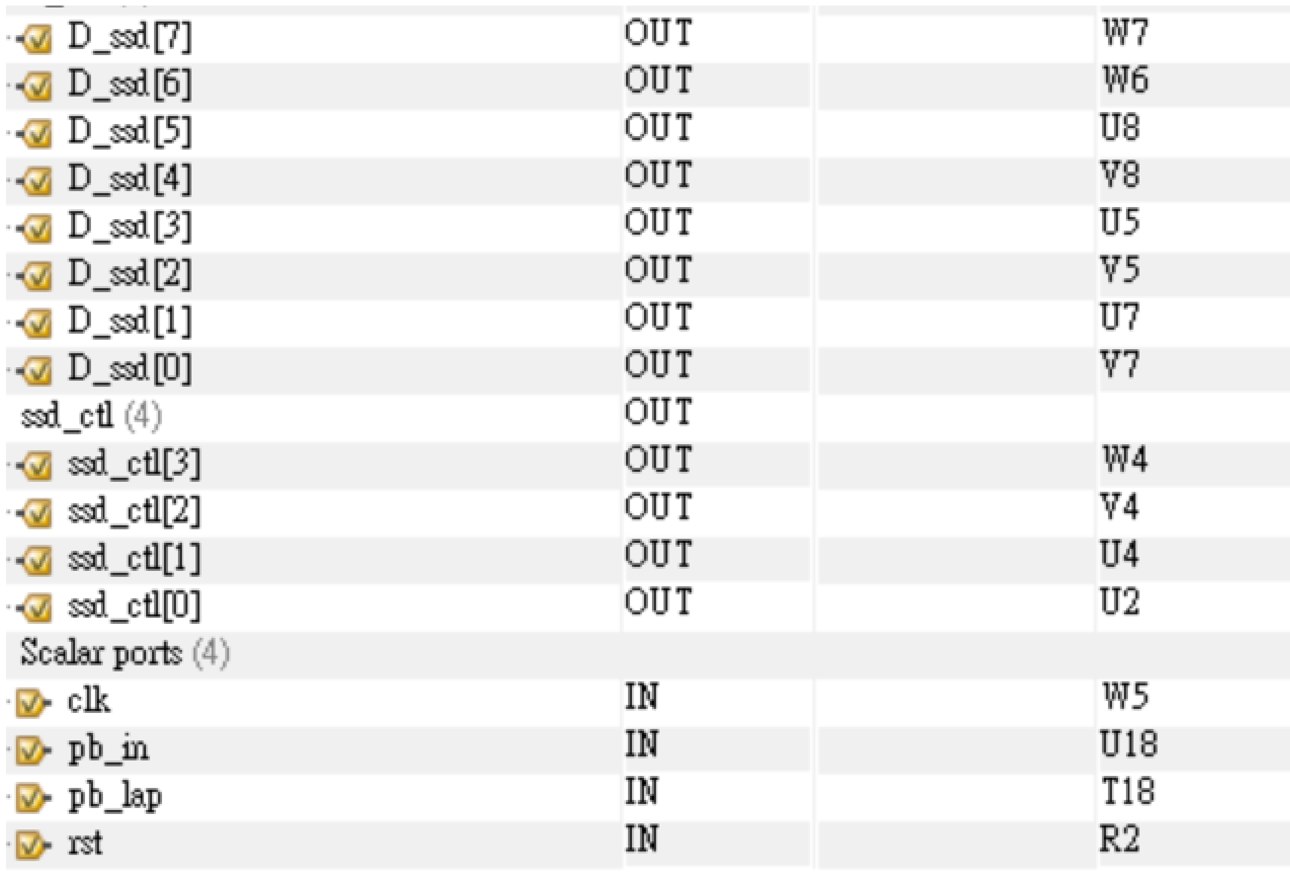
(2) BLOCK DIAGRAM



```
clk_generator U0(.clk(clk), .clk_1hz(clk_1hz), .clk_100hz(clk_100hz), .clk_scan(clk_scan));
debounce U1(.clk(clk_100hz), .pb_in(pb_in), .pb_debounced(pb_debounced), .rst_n(rst));
debounce U21(.clk(clk_1hz), .pb_in(pb_lap), .pb_debounced(pb_debounced_rst), .rst_n(rst));
debounce U8(.clk(clk_100hz), .pb_in(pb_lap), .pb_debounced(pb_debounced_lap), .rst_n(rst));
one_pulse U2(.clk(clk), .in_trig(pb_debounced), .out_pulse(out_pulse), .rst_n(rst));
one_pulse U22(.clk(clk_100hz), .in_trig(pb_debounced_rst), .out_pulse(rst_h), .rst_n(rst));
one_pulse U52(.clk(clk), .in_trig(pb_debounced_lap), .out_pulse(out_pulse_lap), .rst_n(rst));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h));
FSM2 u11(.in(out_pulse_lap), .rst_h(rst_h), .state(LAP));
BCD_UP digit1(.limit(4'b1001), .clk(clk_1hz), .rst_h(rst_h), .add(count_en), .q(sec1), .borrow(borrow[0]));
BCD_UP digit2(.limit(4'b0101), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[0]), .q(sec2), .borrow(borrow[1]));
BCD_UP digit3(.limit(4'b1001), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[1]), .q(min1), .borrow(borrow[2]));
BCD_UP digit4(.limit(4'b0101), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[2]), .q(min2), .borrow(borrow[3]));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .min1(min1), .min2(min2), .sec1(sec1), .sec2(sec2),
.control(clk_scan), .enable(LAP), .rst_h(rst_h));
display(.D_ssd(D_ssd), .in(ssd_in));
```

DESIGN IMPLEMENTATION

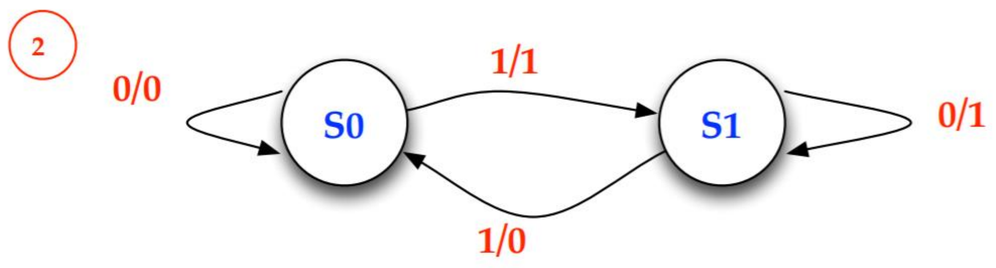
(1) I / O PINS

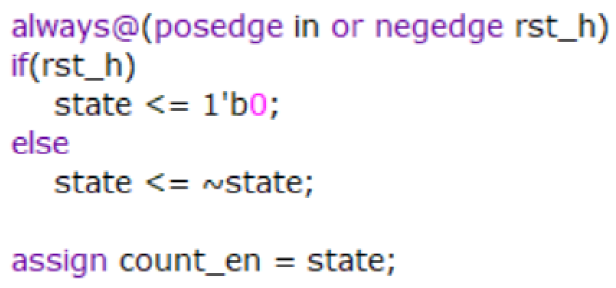


D_ssd[7]	OUT		W7
D_ssd[6]	OUT		W6
D_ssd[5]	OUT		U8
D_ssd[4]	OUT		V8
D_ssd[3]	OUT		U5
D_ssd[2]	OUT		V5
D_ssd[1]	OUT		U7
D_ssd[0]	OUT		V7
ssd_ctl (4)	OUT		
ssd_ctl[3]	OUT		W4
ssd_ctl[2]	OUT		V4
ssd_ctl[1]	OUT		U4
ssd_ctl[0]	OUT		U2
Scalar ports (4)			
clk	IN		W5
pb_in	IN		U18
pb_lap	IN		T18
rst	IN		R2

(2) VERILOGCODE

MODULE FSM



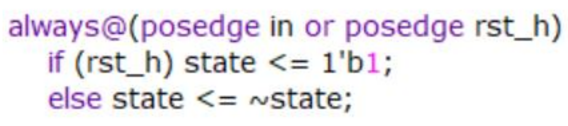


```
always@(posedge in or negedge rst_h)
if(rst_h)
    state <= 1'b0;
else
    state <= ~state;

assign count_en = state;
```

因為只有 2 個 STATE，直接用 in trigger 較為方便

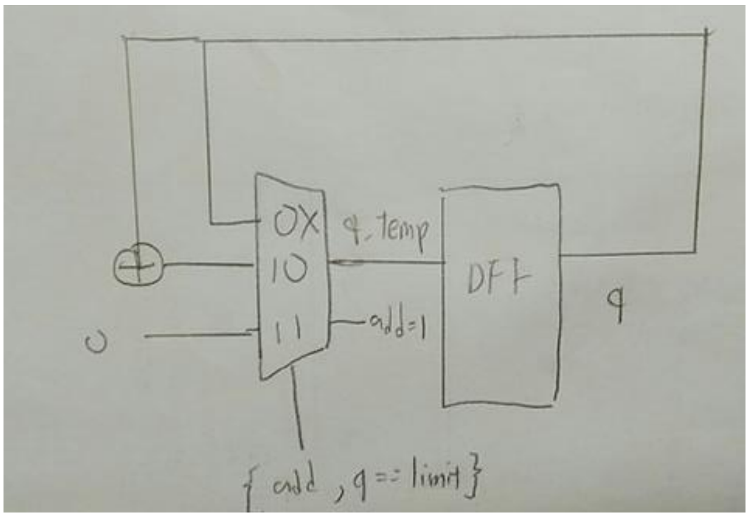
MODULE FSM2

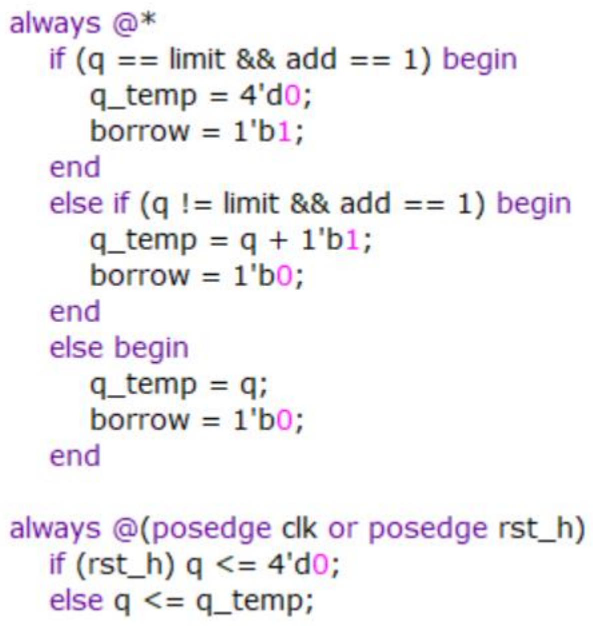
與 FSM 同



```
always@(posedge in or posedge rst_h)
   if (rst_h) state <= 1'b1;
   else state <= ~state;
```

Module BCD_UP

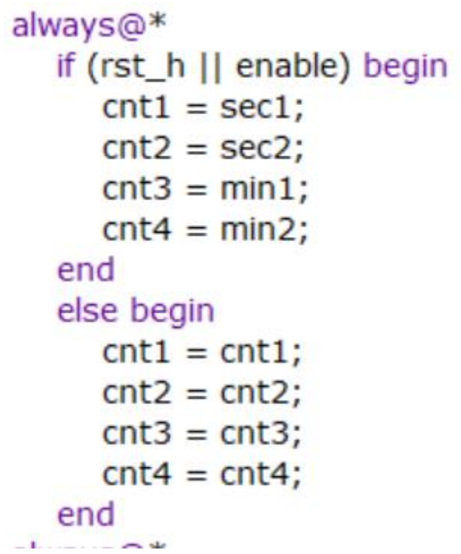




```
always @*
   if (q == limit && add == 1) begin
      q_temp = 4'd0;
      borrow = 1'b1;
   end
   else if (q != limit && add == 1) begin
      q_temp = q + 1'b1;
      borrow = 1'b0;
   end
   else begin
      q_temp = q;
      borrow = 1'b0;
   end

always @(posedge clk or posedge rst_h)
   if (rst_h) q <= 4'd0;
   else q <= q_temp;
```

Module scan

Scan 和之前相同，加了 enable 功能而已，當 enable 或 rst 開，才把新的值傳進去。



```
always@*
   if (rst_h || enable) begin
      cnt1 = sec1;
      cnt2 = sec2;
      cnt3 = min1;
      cnt4 = min2;
   end
   else begin
      cnt1 = cnt1;
      cnt2 = cnt2;
      cnt3 = cnt3;
      cnt4 = cnt4;
   end
```

DISSCUSSION

這次順利，把想法打上去就對了，因為以前 LAB06 就有在 SCAN 處理東西過了，只是這裡我有一個觀念錯了，讓我在 7-2

吃盡了苦頭，那就是 REG 不能存值，所以我在 SCAN MODULE 裡的打法不好，應該要寫一個 DFF。還有我以前的長按是在 ONE_PULSE 做處理，但後來發現觸發的時間好像是不固定的，所以改在 DEBOUNCE 皆較長的 CLK，然後因為長按後觸發的是 RST，所以沒有黨訊號的必要。

執行結果是按下 LAP 鎖住螢幕，長按 RST。

CONCLUSION

這題我打很快，30min 以內，但是關於 reg 的觀念是錯的。

第二題

DESIGN SPECIFICATION

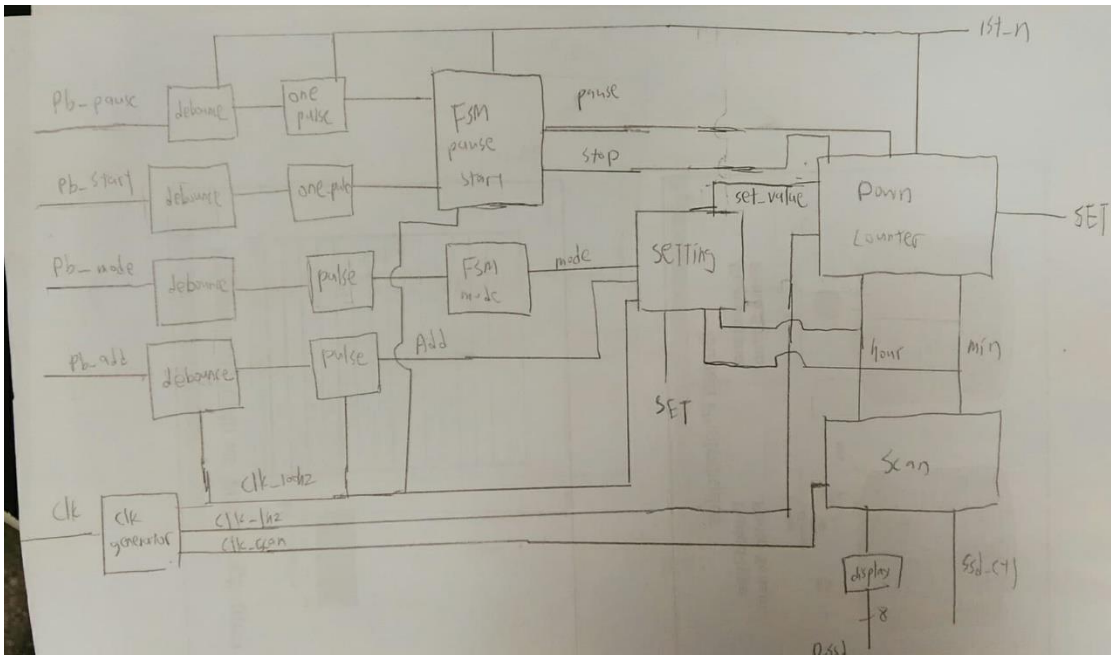
(1) INPUT / OUTPUT

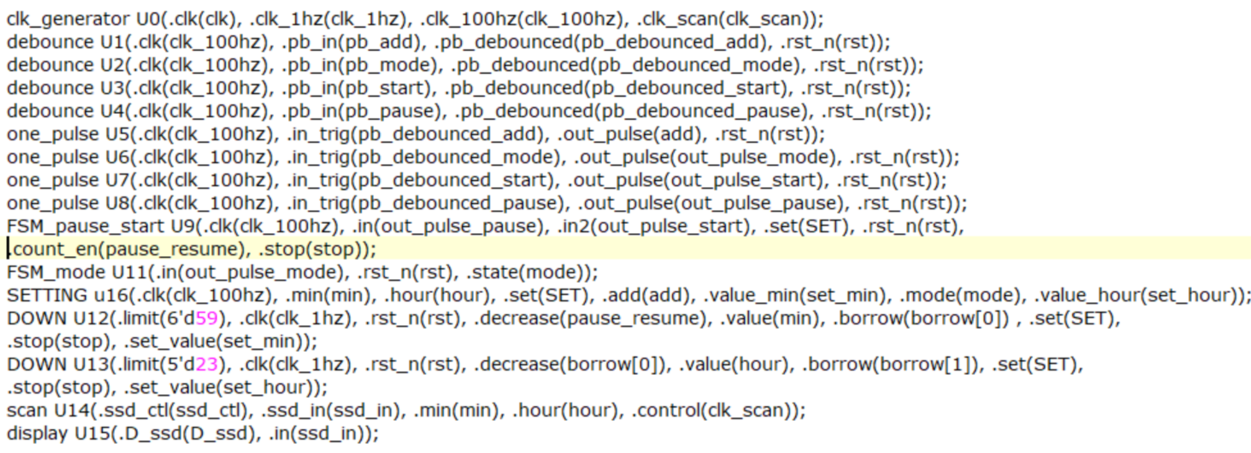
```
module top(clk, SET, pb_start, rst, pb_pause, pb_mode, pb_add,
ssd_ctl, D_ssd, led);
    input SET;
    input clk;
    input pb_pause;
    input pb_start;
    input pb_mode;
    input pb_add;
    input rst;
    output [3:0]ssd_ctl;
    output [7:0]D_ssd;
    output reg [15:0]led;
```

```
module FSM_pause_start(in, rst_n, count_en, set, in2, stop, clk);
        input set;
        input in;
        input rst_n;
        input in2;
        input clk;
        output reg count_en;
        output reg stop;
        reg [1:0]state;
       reg [1:0]next_state;

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

module SETTING(min, hour, set, add, value_min, mode, value_hour,
clk);
        input [5:0]min;
        input [4:0]hour;
        input set;
        input add;
        input mode;
        input clk;
        output reg [5:0]value_min;
        output reg [4:0]value_hour;
        reg [5:0]value_min_temp;
         reg [4:0]value_hour_temp;

module DOWN(limit, clk, rst_n, decrease, borrow, set, value, stop,
set_value);
        input [5:0]limit;
        input clk;
        input rst_n;
        input decrease;
        input set;
        input stop;
        input [5:0]set_value;
```

```
output reg [5:0]value;
output reg borrow;
reg [5:0]LIMIT;
reg [5:0]LIMIT_TEMP;
reg [5:0]q;
reg [5:0]q_temp;
 reg [5:0]in;
```

(2) BLOCK DIAGRAM

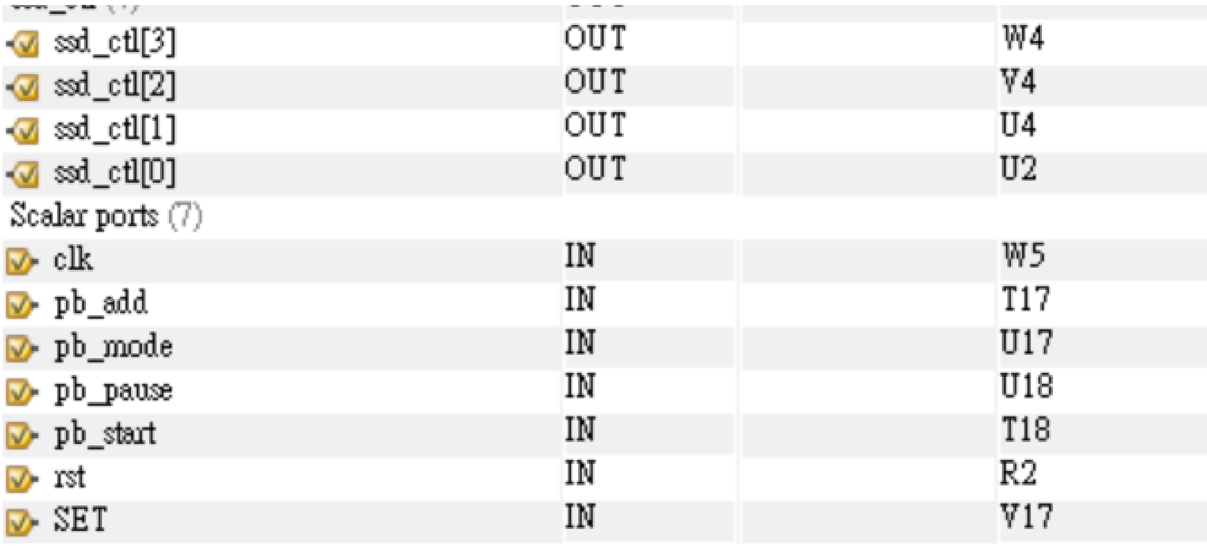




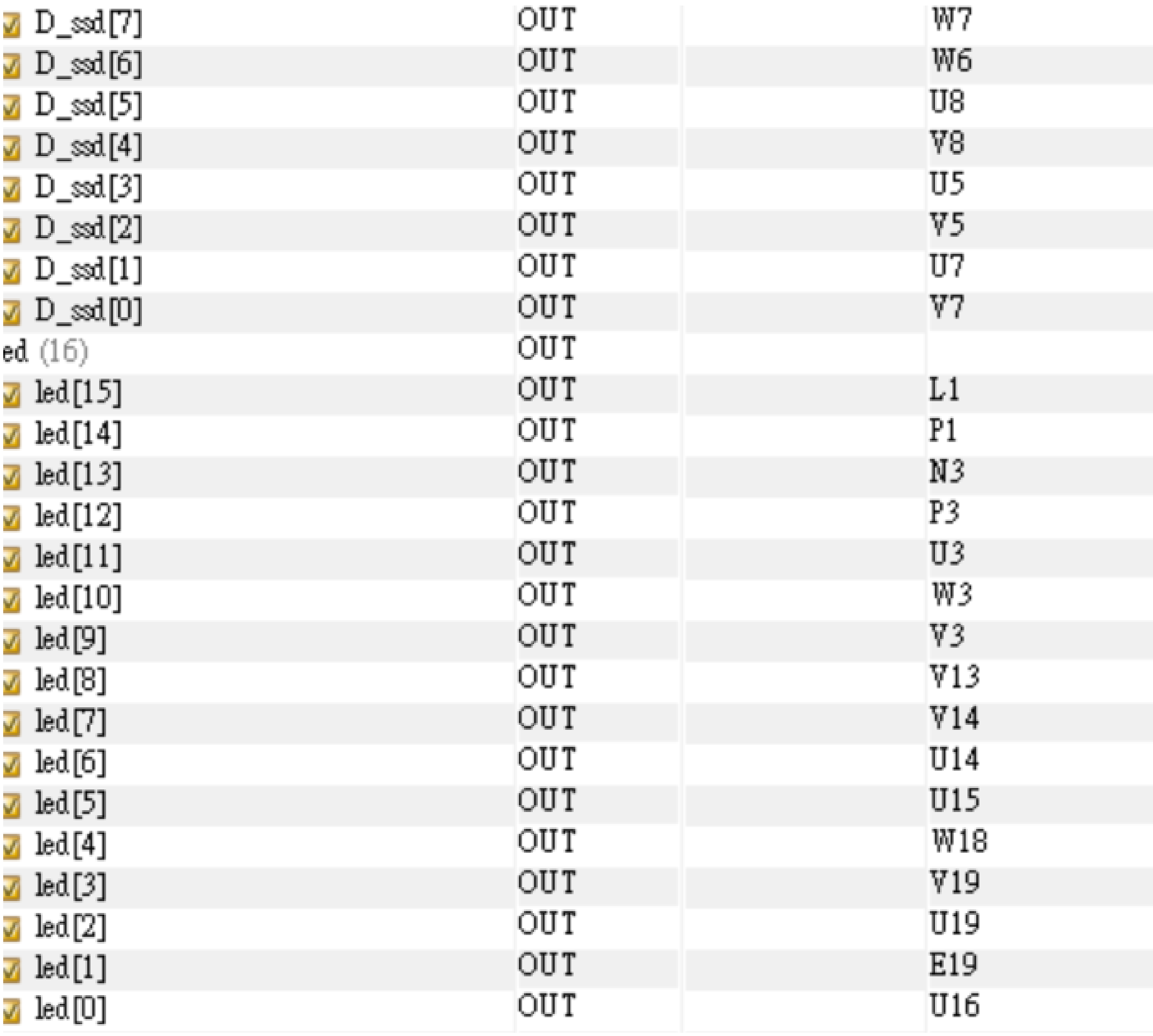
```
clk_generator U0(.clk(clk), .clk_1hz(clk_1hz), .clk_100hz(clk_100hz), .clk_scan(clk_scan));
debounce U1(.clk(clk_100hz), .pb_in(pb_add), .pb_debounced(pb_debounced_add), .rst_n(rst));
debounce U2(.clk(clk_100hz), .pb_in(pb_mode), .pb_debounced(pb_debounced_mode), .rst_n(rst));
debounce U3(.clk(clk_100hz), .pb_in(pb_start), .pb_debounced(pb_debounced_start), .rst_n(rst));
debounce U4(.clk(clk_100hz), .pb_in(pb_pause), .pb_debounced(pb_debounced_pause), .rst_n(rst));
one_pulse U5(.clk(clk_100hz), .in_trig(pb_debounced_add), .out_pulse(add), .rst_n(rst));
one_pulse U6(.clk(clk_100hz), .in_trig(pb_debounced_mode), .out_pulse(out_pulse_mode), .rst_n(rst));
one_pulse U7(.clk(clk_100hz), .in_trig(pb_debounced_start), .out_pulse(out_pulse_start), .rst_n(rst));
one_pulse U8(.clk(clk_100hz), .in_trig(pb_debounced_pause), .out_pulse(out_pulse_pause), .rst_n(rst));
FSM_pause_start U9(.clk(clk_100hz), .in(out_pulse_pause), .in2(out_pulse_start), .set(SET), .rst_n(rst),
.count_en(pause_resume), .stop(stop));
FSM_mode U11(.in(out_pulse_mode), .rst_n(rst), .state(mode));
SETTING u16(.clk(clk_100hz), .min(min), .hour(hour), .set(SET), .add(add), .value_min(set_min), .mode(mode), .value_hour(set_hour));
DOWN U12(.limit(6'd59), .clk(clk_1hz), .rst_n(rst), .decrease(pause_resume), .value(min), .borrow(borrow[0]) , .set(SET),
.stop(stop), .set_value(set_min));
DOWN U13(.limit(5'd23), .clk(clk_1hz), .rst_n(rst), .decrease(borrow[0]), .value(hour), .borrow(borrow[1]), .set(SET),
.stop(stop), .set_value(set_hour));
scan U14(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .min(min), .hour(hour), .control(clk_scan));
display U15(.D_ssd(D_ssd), .in(ssd_in));
```

DESIGN IMPLEMENTATION

(1) I / O PINS



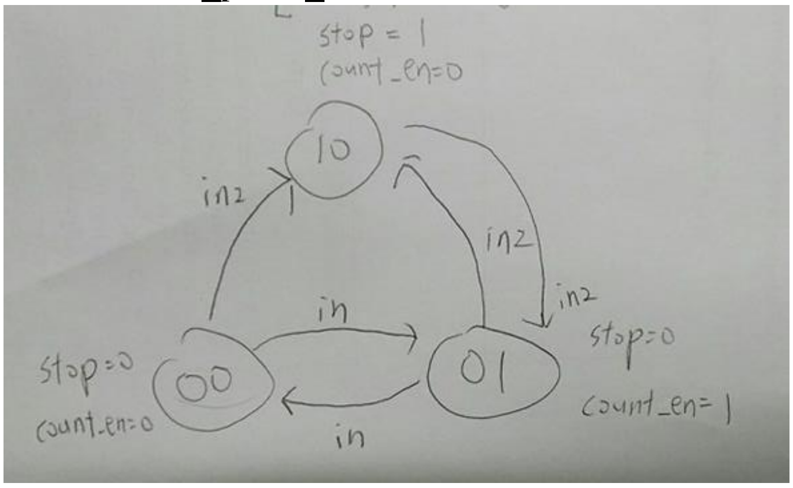
ssd_ctl[3]	OUT		W4
ssd_ctl[2]	OUT		V4
ssd_ctl[1]	OUT		U4
ssd_ctl[0]	OUT		U2
Scalar ports (7)			
clk	IN		W5
pb_add	IN		T17
pb_mode	IN		U17
pb_pause	IN		U18
pb_start	IN		T18
rst	IN		R2
SET	IN		V17



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
ed (16)	OUT		
led[15]	OUT		L1
led[14]	OUT		P1
led[13]	OUT		N3
led[12]	OUT		P3
led[11]	OUT		U3
led[10]	OUT		W3
led[9]	OUT		V3
led[8]	OUT		V13
led[7]	OUT		V14
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

(2) VERILOG CODE

Module FSM_pause_start

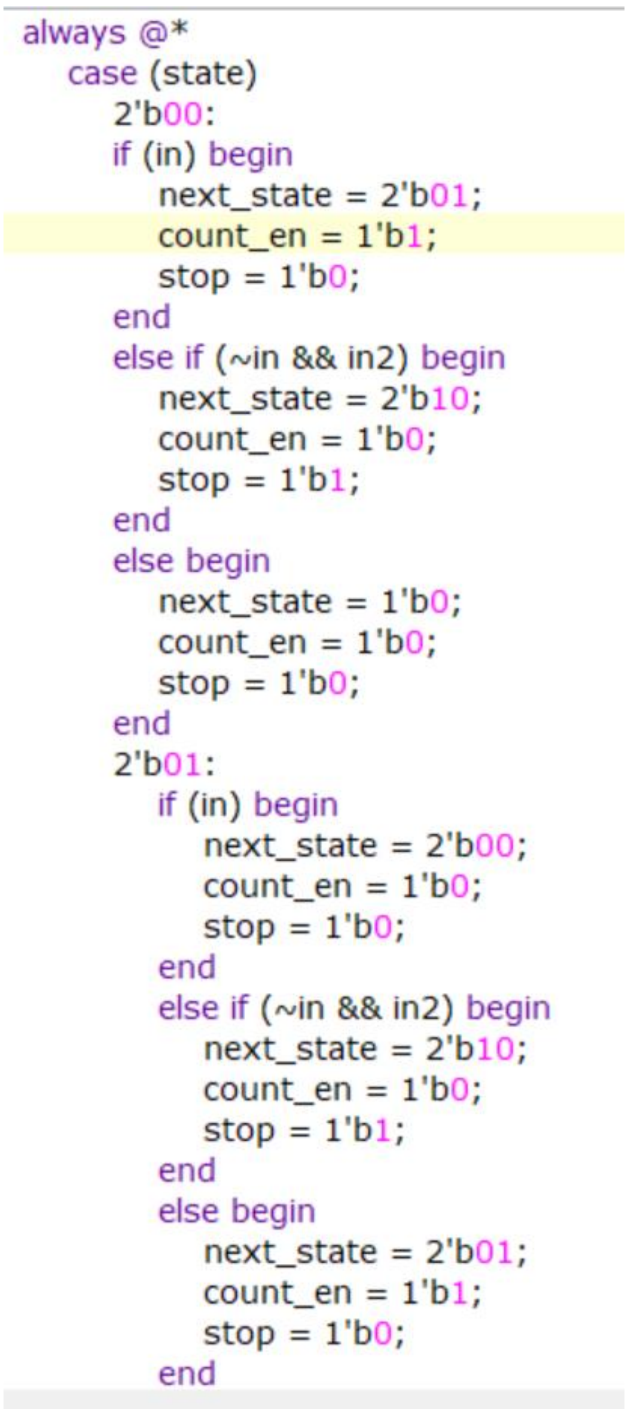


In 是 pause/resume 之鈕

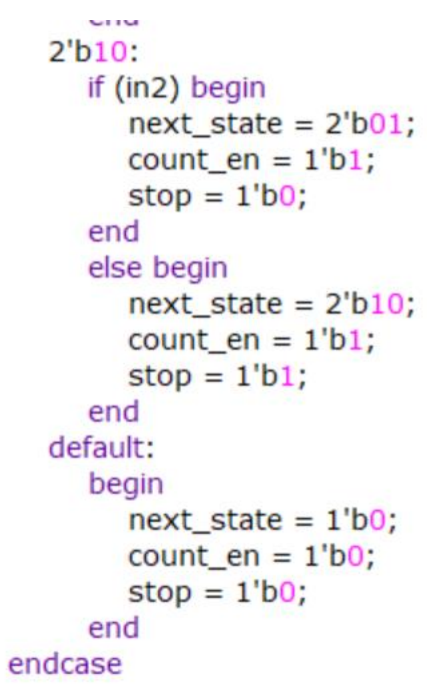
In2 是 stop/start 之鈕

Count_en 是 count enable，接到分的 add

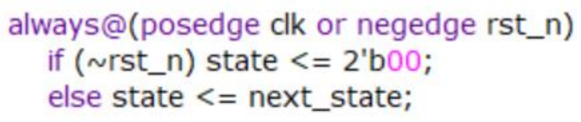
Stop 是要不要回到設定之時間



```
always @*
   case (state)
      2'b00:
      if (in) begin
         next_state = 2'b01;
         count_en = 1'b1;
         stop = 1'b0;
      end
      else if (~in && in2) begin
         next_state = 2'b10;
         count_en = 1'b0;
         stop = 1'b1;
      end
      else begin
         next_state = 1'b0;
         count_en = 1'b0;
         stop = 1'b0;
      end
      2'b01:
         if (in) begin
            next_state = 2'b00;
            count_en = 1'b0;
            stop = 1'b0;
         end
         else if (~in && in2) begin
            next_state = 2'b10;
            count_en = 1'b0;
            stop = 1'b1;
         end
         else begin
            next_state = 2'b01;
            count_en = 1'b1;
            stop = 1'b0;
         end
```



```
        end
    2'b10:
        if (in2) begin
            next_state = 2'b01;
            count_en = 1'b1;
            stop = 1'b0;
        end
        else begin
            next_state = 2'b10;
            count_en = 1'b1;
            stop = 1'b1;
        end
    default:
        begin
            next_state = 1'b0;
            count_en = 1'b0;
            stop = 1'b0;
        end
endcase
```

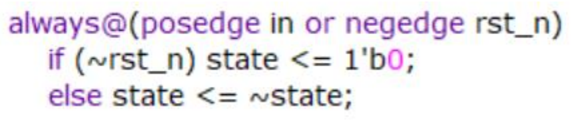


```
always@(posedge clk or negedge rst_n)
    if (~rst_n) state <= 2'b00;
    else state <= next_state;
```

Module FSM_mode

這是控制要加時還是要加分

In 是 mode 鈕



```
always@(posedge in or negedge rst_n)
    if (~rst_n) state <= 1'b0;
    else state <= ~state;
```

State 直接當 output，在只有 2 個 state 0/1 時我都這樣打

Module SETTING

這裡是 SETTING 打開之後設定時間的 MODULE

SET 沒打開時，讓 VALUE 跟著當下的時間

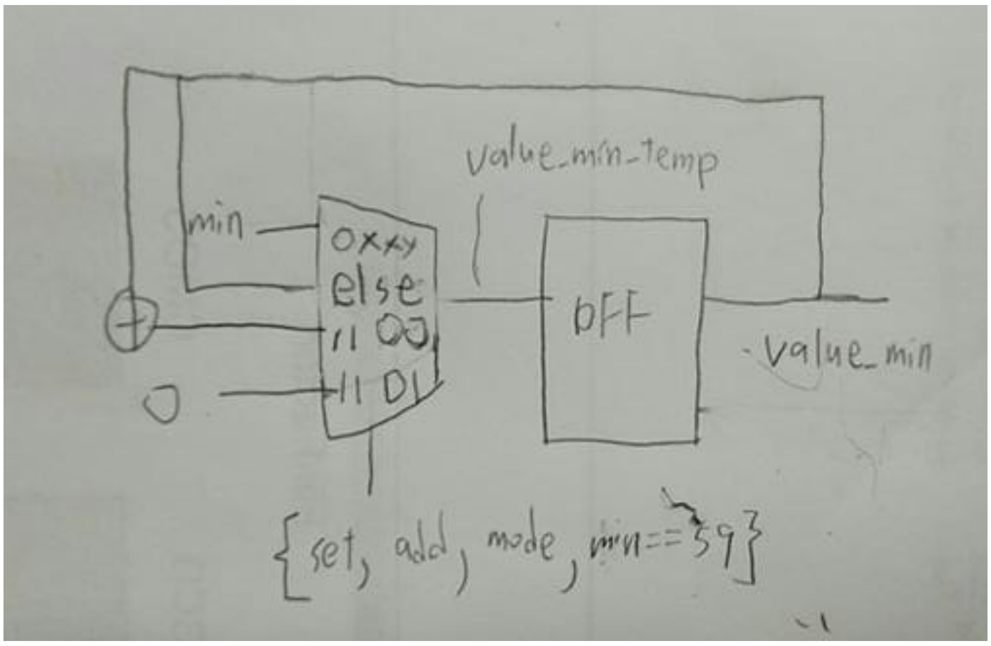
MODE == 0 加分

MODE == 1 加時

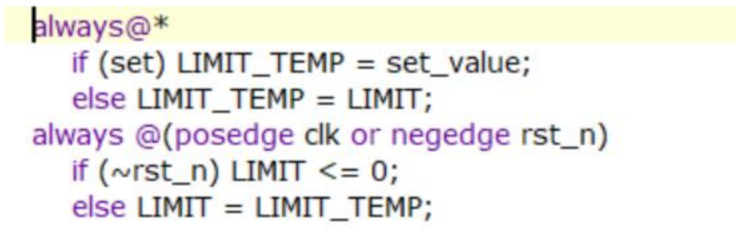
ADD == 1 讓他加一

若分達到 59 或時達到 24，就回到 0

下圖只有畫分圖，因為時的跟分的差不多，只是 MODE 倒過來跟上限的不同

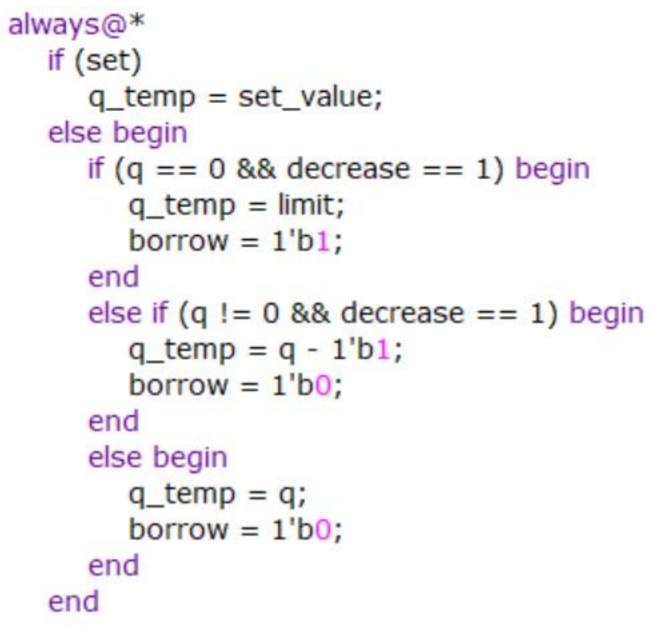


Module DOWN



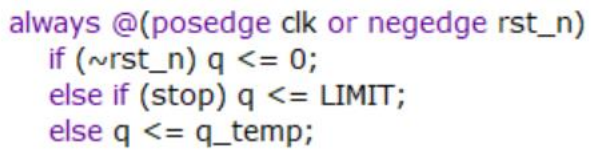
```
always@*
    if (set) LIMIT_TEMP = set_value;
    else LIMIT_TEMP = LIMIT;
always @(posedge clk or negedge rst_n)
    if (~rst_n) LIMIT <= 0;
    else LIMIT = LIMIT_TEMP;
```

LIMIT 設定之後的上限，set_vaue 是 SETTING MODULE 加完之結果，當 SET 打開才把 set_vaue，否則維持原本的值



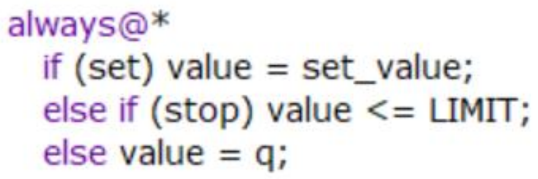
```
always@*
   if (set)
      q_temp = set_value;
   else begin
      if (q == 0 && decrease == 1) begin
         q_temp = limit;
         borrow = 1'b1;
      end
      else if (q != 0 && decrease == 1) begin
         q_temp = q - 1'b1;
         borrow = 1'b0;
      end
      else begin
         q_temp = q;
         borrow = 1'b0;
      end
   end
```

q 是計數器正在數的數字，set 打開就傳 set_value，否則正常運作



```
always @(posedge clk or negedge rst_n)
   if (~rst_n) q <= 0;
   else if (stop) q <= LIMIT;
   else q <= q_temp;
```

如果 reset 就回到 0，stop 回到 LIMIT，其餘正常



```
always@*
   if (set) value = set_value;
   else if (stop) value <= LIMIT;
   else value = q;
```

我實際的 OUTPUT 是 value，特地在寫一個 always 的原因是當按下 stop 或 add 時，q 的值因為 clk 是 1hz 的關係無法及時改變，所以加了一個 always，但這只能讓螢幕上的數字即時改變，真正要讓 q 改變需要等至多 0.5 秒。

DISCUSSION

7-2 真的是惡夢，花我超久時間 DEBUG，這次因為 FSM 比較複雜，不能再用以前用 in trigger FilpFlop 的方法，因為當我寫 always@(posedge in or posedge in2 or negedge rst_n) 他就顯示 ambiguous clock condition，所以我只能回去研究以前我用老師的為何不行，結果居然是因為 one_pulse 跟 debounce 沒有接 reset，但我不知為何沒加個 reset 就跑不動。

還有這次我覺得我很快把邏輯打出來了，但我都用 always@* 我不知道存值要用 FF，害我曾一度不知所措，不知道該怎麼做，後來把每個東西都加上一個 FF，就成功了，但整個過程應該花我有 8 小時，有夠累的。

執行結果就是開啟 SETTING，按下 ADD 分加一，在按 MODE 換加時，關掉 SETTING 開始數，按下 STOP 回到設定時間，在按 START 才開始數。

CONCLUSION

每個語言都有他的規範，真的要熟知他的規定再開始打，這次真的吃了大虧，但我也銘記於心，一路下來我摸索許多規定，雖然辛苦，但也算頗有成就感。

第三題

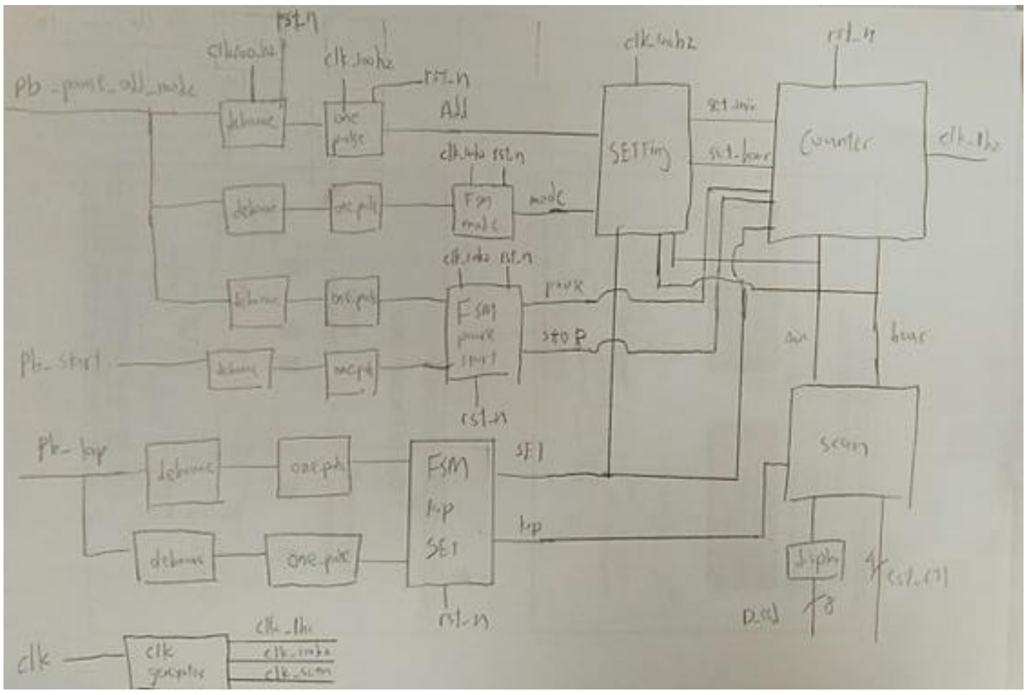
DESIGN SPECIFICATION

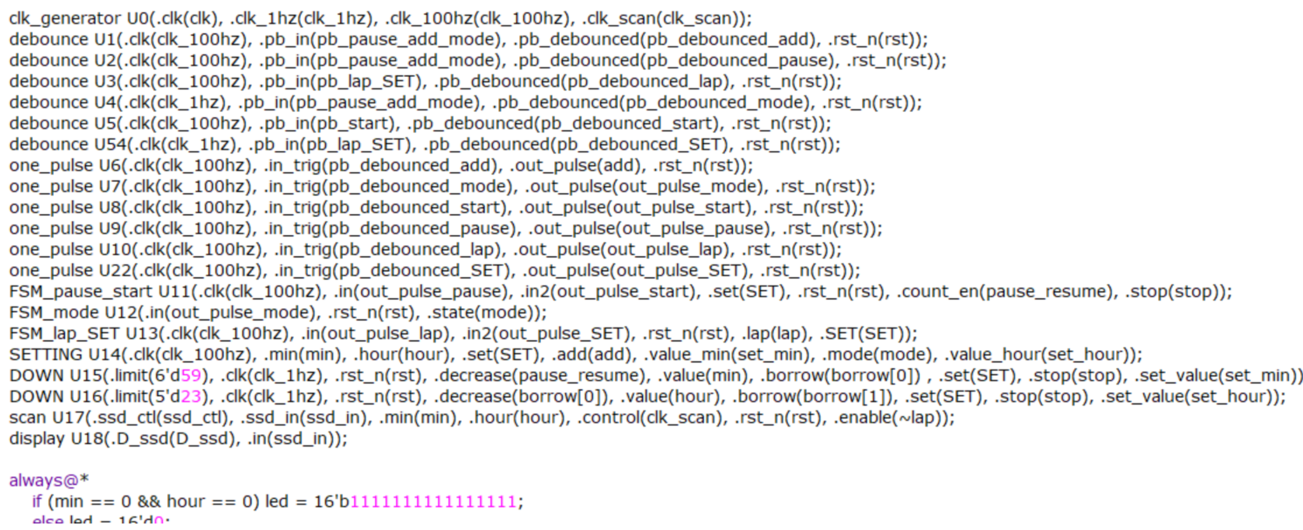
(1) INPUT / OUTPUT

```
module top(clk, pb_start, rst, pb_pause_add_mode, pb_lap_SET,
ssd_ctl, D_ssd, led);
        input clk;
        input pb_pause_add_mode;
        input pb_start;
        input pb_lap_SET;
        input rst;
        output [3:0]ssd_ctl;
        output [7:0]D_ssd;
        output reg [15:0]led;

module FSM_lap_SET(in, rst_n, lap, in2, SET, clk);
        input in;
        input rst_n;
        input in2;
        input clk;
        output reg lap;
        output reg SET;
        reg [1:0]state;
        reg [1:0]next_state;
```

其餘和 7-1, 7-2 同

(2) BLOCK DIAGRAM

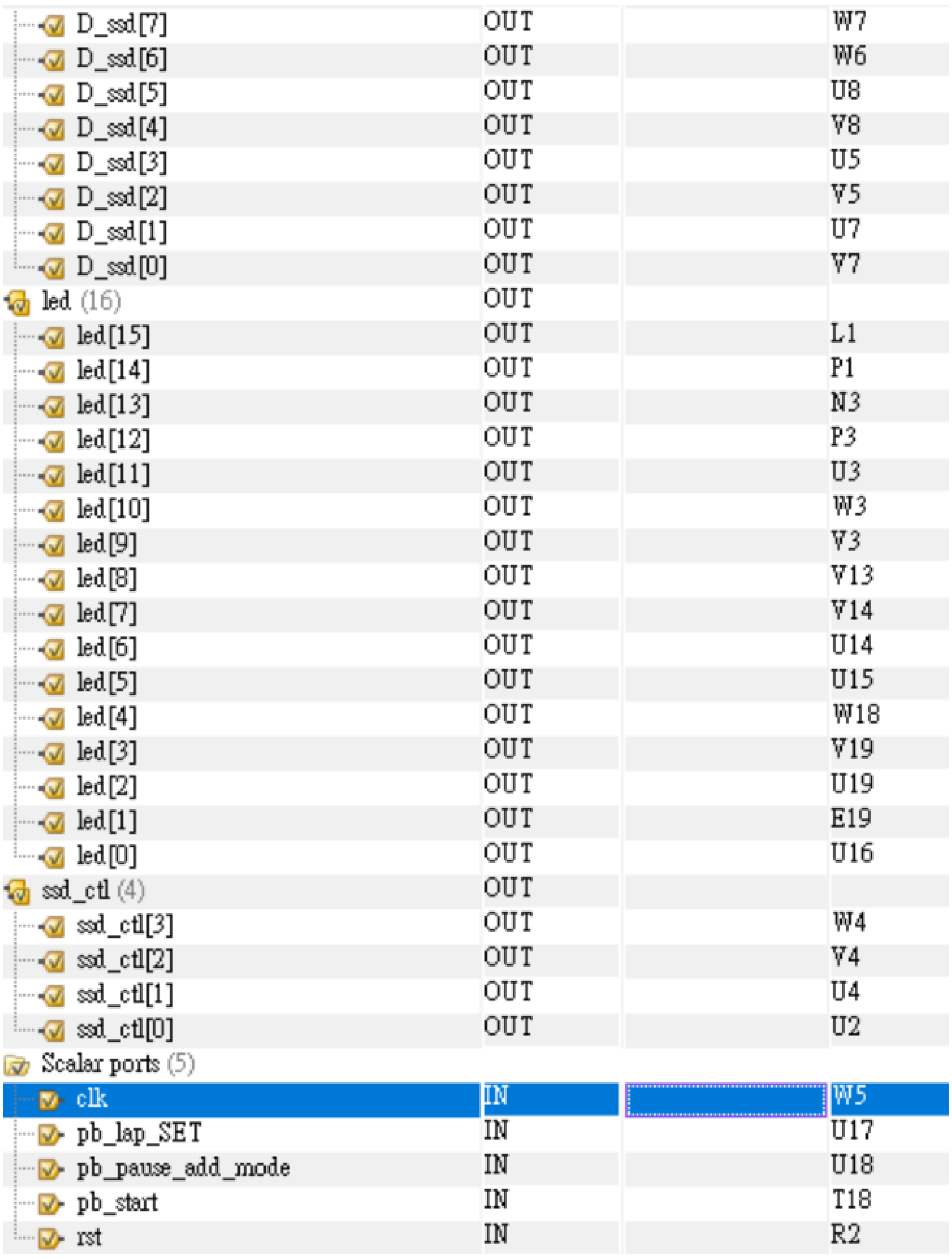




```
clk_generator U0(.clk(clk), .clk_1hz(clk_1hz), .clk_100hz(clk_100hz), .clk_scan(clk_scan));
debounce U1(.clk(clk_100hz), .pb_in(pb_pause_add_mode), .pb_debounced(pb_debounced_add), .rst_n(rst));
debounce U2(.clk(clk_100hz), .pb_in(pb_pause_add_mode), .pb_debounced(pb_debounced_pause), .rst_n(rst));
debounce U3(.clk(clk_100hz), .pb_in(pb_lap_SET), .pb_debounced(pb_debounced_lap), .rst_n(rst));
debounce U4(.clk(clk_1hz), .pb_in(pb_pause_add_mode), .pb_debounced(pb_debounced_mode), .rst_n(rst));
debounce U5(.clk(clk_100hz), .pb_in(pb_start), .pb_debounced(pb_debounced_start), .rst_n(rst));
debounce U54(.clk(clk_1hz), .pb_in(pb_lap_SET), .pb_debounced(pb_debounced_SET), .rst_n(rst));
one_pulse U6(.clk(clk_100hz), .in_trig(pb_debounced_add), .out_pulse(add), .rst_n(rst));
one_pulse U7(.clk(clk_100hz), .in_trig(pb_debounced_mode), .out_pulse(out_pulse_mode), .rst_n(rst));
one_pulse U8(.clk(clk_100hz), .in_trig(pb_debounced_start), .out_pulse(out_pulse_start), .rst_n(rst));
one_pulse U9(.clk(clk_100hz), .in_trig(pb_debounced_pause), .out_pulse(out_pulse_pause), .rst_n(rst));
one_pulse U10(.clk(clk_100hz), .in_trig(pb_debounced_lap), .out_pulse(out_pulse_lap), .rst_n(rst));
one_pulse U22(.clk(clk_100hz), .in_trig(pb_debounced_SET), .out_pulse(out_pulse_SET), .rst_n(rst));
FSM_pause_start U11(.clk(clk_100hz), .in(out_pulse_pause), .in2(out_pulse_start), .set(SET), .rst_n(rst), .count_en(pause_resume), .stop(stop));
FSM_mode U12(.in(out_pulse_mode), .rst_n(rst), .state(mode));
FSM_lap_SET U13(.clk(clk_100hz), .in(out_pulse_lap), .in2(out_pulse_SET), .rst_n(rst), .lap(lap), .SET(SET));
SETTING U14(.clk(clk_100hz), .min(min), .hour(hour), .set(SET), .add(add), .value_min(set_min), .mode(mode), .value_hour(set_hour));
DOWN U15(.limit(6'd59), .clk(clk_1hz), .rst_n(rst), .decrease(pause_resume), .value(min), .borrow(borrow[0]) , .set(SET), .stop(stop), .set_value(set_min))
DOWN U16(.limit(5'd23), .clk(clk_1hz), .rst_n(rst), .decrease(borrow[0]), .value(hour), .borrow(borrow[1]), .set(SET), .stop(stop), .set_value(set_hour));
scan U17(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .min(min), .hour(hour), .control(clk_scan), .rst_n(rst), .enable(~lap));
display U18(.D_ssd(D_ssd), .in(ssd_in));

always@*
   if (min == 0 && hour == 0) led = 16'b1111111111111111;
   else led = 16'd0;
```

DESIGN IMPLEMENTATION

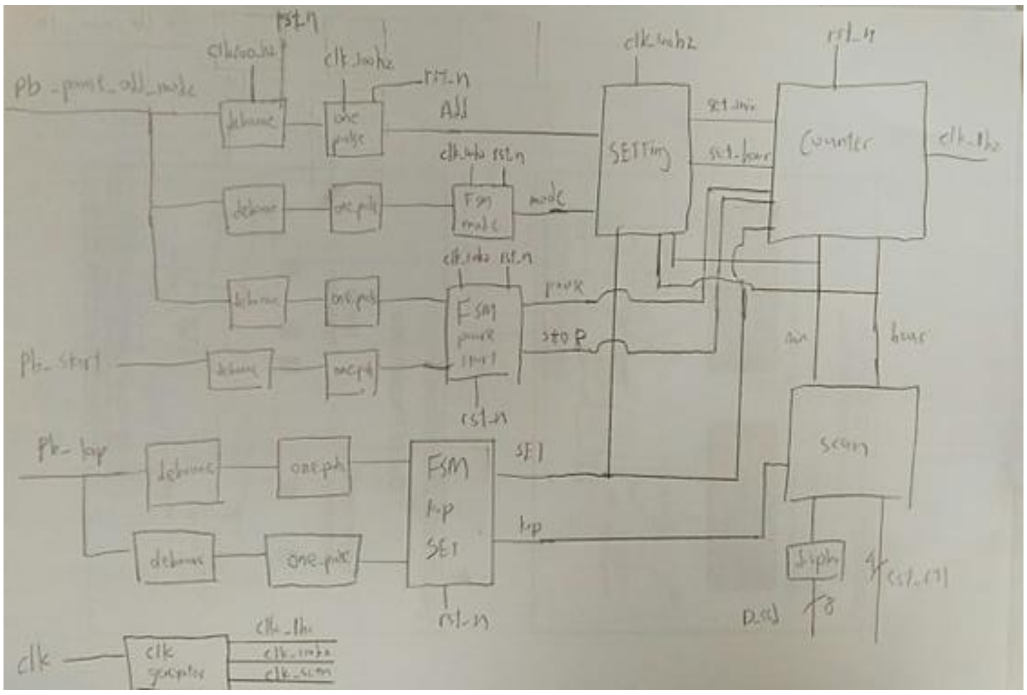
(1) I / O PINS



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
led (16)	OUT		
led[15]	OUT		L1
led[14]	OUT		P1
led[13]	OUT		N3
led[12]	OUT		P3
led[11]	OUT		U3
led[10]	OUT		W3
led[9]	OUT		V3
led[8]	OUT		V13
led[7]	OUT		V14
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
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 (5)			
clk	IN		W5
pb_lap_SET	IN		U17
pb_pause_add_mode	IN		U18
pb_start	IN		T18
rst	IN		R2

(2) Verilog code

Module top



我把 pause/resume 跟 add / mode(決定要加時還加分) 設在同

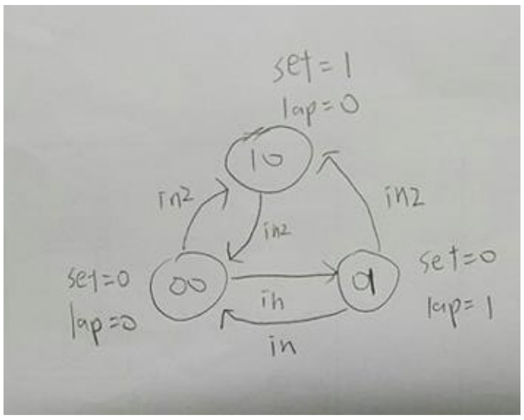
一個按鈕，因為他們一個是在 set 打開時才運作，一個是在 set

關起時才運作，互不衝突，然後 mode 是用長按

然後 stop/start 一個鈕 lap 跟 SETTING 一個鈕

Module FSM_lap_SET

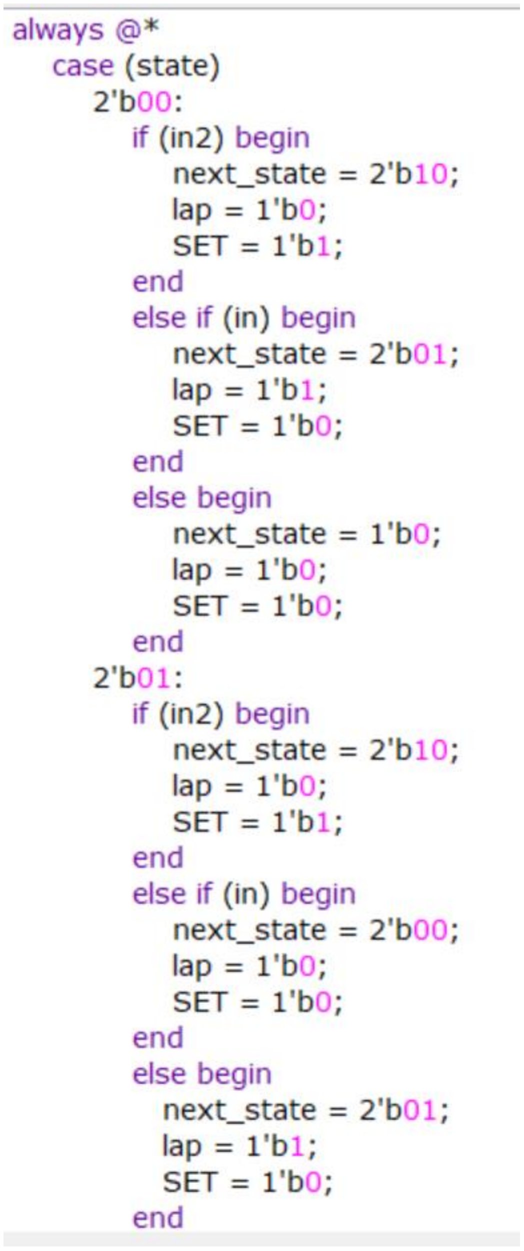
因為 SET 長按時會觸發 LAP，所以寫個 FSM 來擋訊號



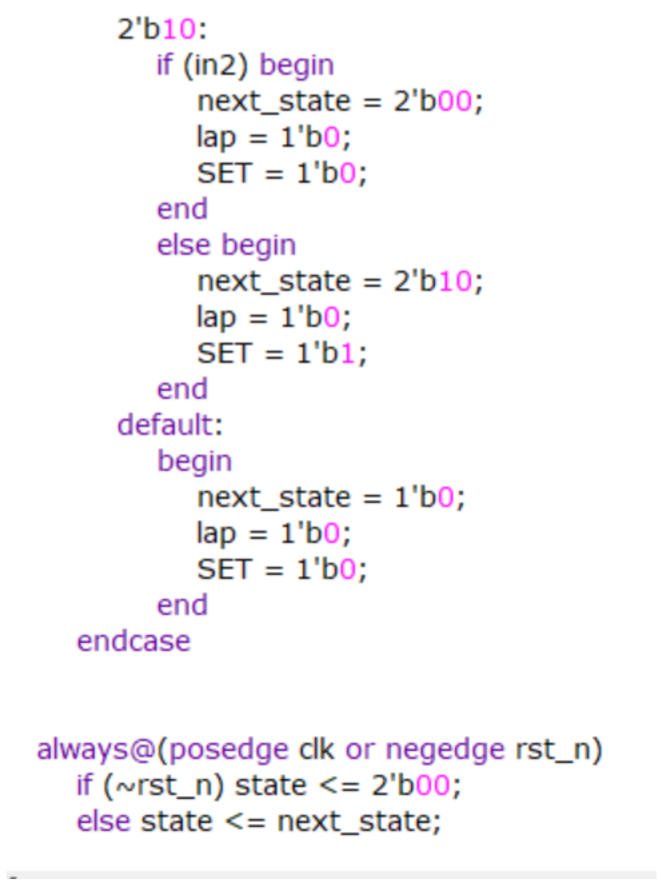
In 是 lap 訊號， in2 是 set 訊號(長按)，只要一有 set 訊號，

就把 lap 解除。

Lap = 0 代表不把螢幕鎖住



```
always @*
   case (state)
      2'b00:
         if (in2) begin
            next_state = 2'b10;
            lap = 1'b0;
            SET = 1'b1;
         end
         else if (in) begin
            next_state = 2'b01;
            lap = 1'b1;
            SET = 1'b0;
         end
         else begin
            next_state = 1'b0;
            lap = 1'b0;
            SET = 1'b0;
         end
      2'b01:
         if (in2) begin
            next_state = 2'b10;
            lap = 1'b0;
            SET = 1'b1;
         end
         else if (in) begin
            next_state = 2'b00;
            lap = 1'b0;
            SET = 1'b0;
         end
         else begin
           next_state = 2'b01;
           lap = 1'b1;
           SET = 1'b0;
         end
```



```
        2'b10:
            if (in2) begin
                next_state = 2'b00;
                lap = 1'b0;
                SET = 1'b0;
            end
            else begin
                next_state = 2'b10;
                lap = 1'b0;
                SET = 1'b1;
            end
        default:
            begin
                next_state = 1'b0;
                lap = 1'b0;
                SET = 1'b0;
            end
    endcase

always@(posedge clk or negedge rst_n)
    if (~rst_n) state <= 2'b00;
    else state <= next_state;
```

DISSCUSSION

這題很快就完成，基本上把 7-1, 7-2 接再一起再加一個 FSM 就好了，而且 7-2 已經把很多問題就解決了，可謂苦盡甘來。

執行結果 LAP 長按進入 SET，然後用 pause/resume 鈕去加，長按控制加時還加分。

CONCLUSION

這題只能用 3 個按鈕，看似很難，但因我想到 pause/resume 與 setting 時的加還有 mode 不衝突，一顆鈕直接解決 4 個功能，所以剩下就容易了。