LAB05

第一題

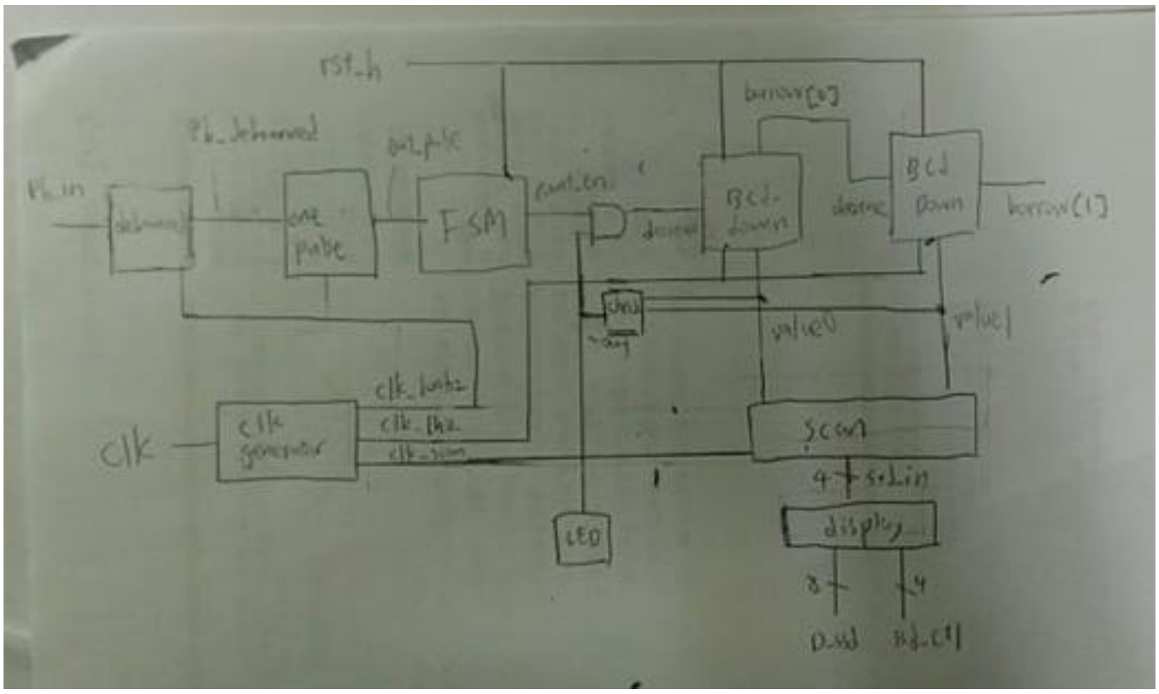
DESIGN SPECIFICATION

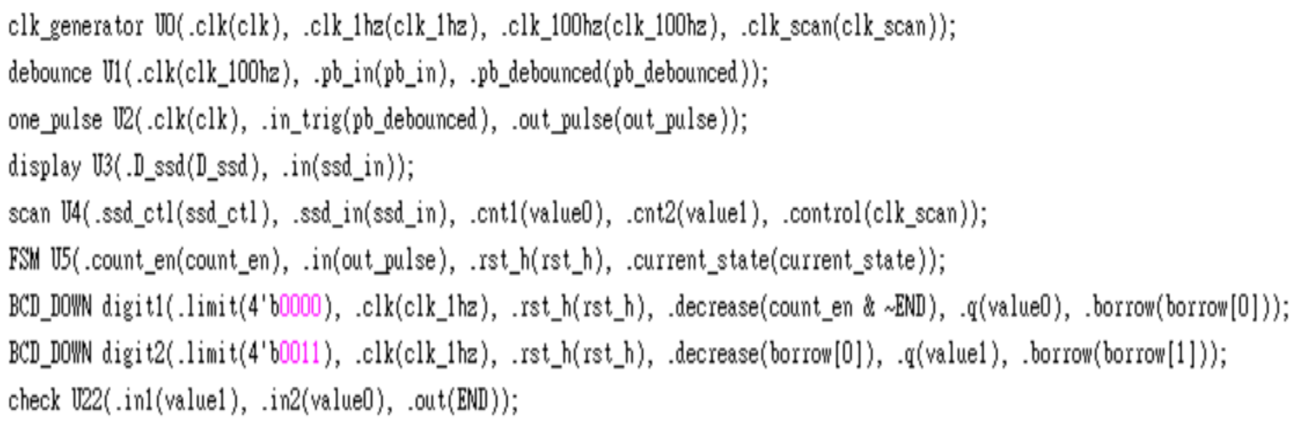
(1) INPUT & OUTPUT

```
module top(clk, rst_h, pb_in, led, D_ssd, ssd_ctl);
 input clk;
 input rst_h;
 input pb_in;
 output reg [15:0] led;
 output [7:0] D_ssd;
 output [3:0] ssd_ctl;

 wire pb_debounced;
 wire clk_1hz,clk_100hz;
 wire [1:0]clk_scan;
 wire out_pulse;
 wire [3:0] ssd_in;
 wire [3:0] value0;
 wire [3:0] value1;
 wire check;
 wire count_en;
 wire END;
 wire [1:0]borrow;
 wire current_state;

module debounce(clk, pb_in, pb_debounced    );
 output reg pb_debounced;
 input clk;
 input pb_in;

module FSM(count_en, in, rst_h, current_state);
 output wire count_en;
 input in;
 input rst_h;
 output current_state;
 reg state;
 reg next_state;
```

```
module BCD_DOWN(limit, clk, rst_h, decrease, q, borrow);
 input [3:0]limit;
 input clk;
 input rst_h;
 input decrease;
 output reg [3:0]q;
 output reg borrow;
 reg [3:0]q_temp;

module scan(ssd_ctl, ssd_in, cnt1, cnt2, control);
 output reg [3:0] ssd_in;
 output reg [3:0] ssd_ctl;
 input [3:0] cnt1;
 input [3:0] cnt2;
 input [1:0] control;

module clk_generator(clk, clk_1hz, clk_100hz, clk_scan);
 input clk;
 output reg clk_1hz;
 output reg clk_100hz;
 output reg [1:0]clk_scan;

module display(D_ssd, in);
 input [3:0] in;
 output reg [7:0] D_ssd;
```

(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));
one_pulse U2(.clk(clk), .in_trig(pb_debounced), .out_pulse(out_pulse));
display U3(.D_ssd(D_ssd), .in(ssd_in));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .cnt1(value0), .cnt2(value1), .control(clk_scan));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h), .current_state(current_state));
BCD_DOWN digit1(.limit(4'b0000), .clk(clk_1hz), .rst_h(rst_h), .decrease(count_en & ~END), .q(value0), .borrow(borrow[0]));
BCD_DOWN digit2(.limit(4'b0011), .clk(clk_1hz), .rst_h(rst_h), .decrease(borrow[0]), .q(value1), .borrow(borrow[1]));
check U22(.in1(value1), .in2(value0), .out(END));
```

DESIGN IMPLEMENTATION

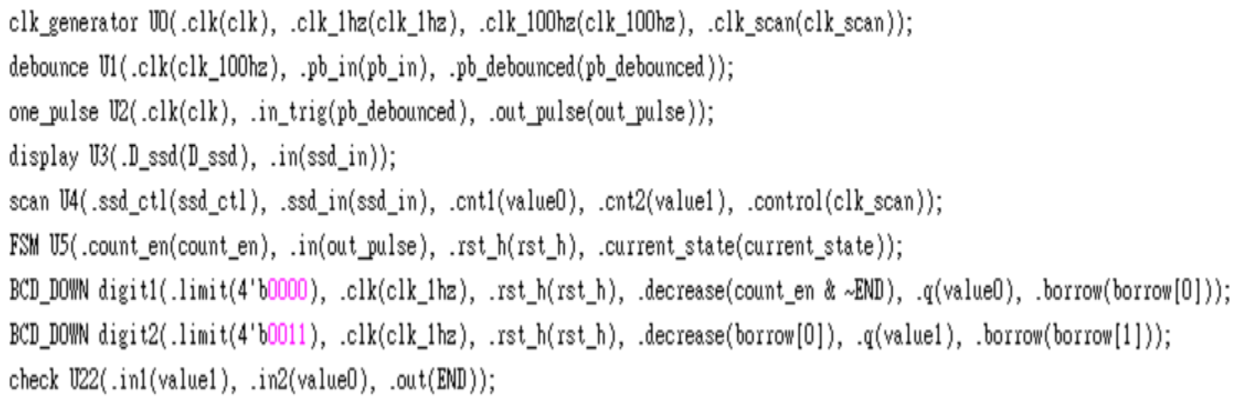
(1) I / O PINS

L1 [get_ports led[15]]
P1 [get_ports led[14]]
N3 [get_ports led[13]]
P3 [get_ports led[12]]
U3 [get_ports led[11]]
W3 [get_ports led[10]]
V3 [get_ports led[9]]
V13 [get_ports led[8]]
V14 [get_ports led[7]]

U14 [get_ports led[6]]
U15 [get_ports led[5]]
W18 [get_ports led[4]]
V19 [get_ports led[3]]
U19 [get_ports led[2]]
E19 [get_ports led[1]]
U16 [get_ports led[0]]
W4 [get_ports ssd_ctl[3]]
V4 [get_ports ssd_ctl[2]]
U4 [get_ports ssd_ctl[1]]
U2 [get_ports ssd_ctl[0]]
W7 [get_ports D_ssd[7]]
W6 [get_ports D_ssd[6]]
U8 [get_ports D_ssd[5]]
V8 [get_ports D_ssd[4]]
U5 [get_ports D_ssd[3]]
V5 [get_ports D_ssd[2]]
U7 [get_ports D_ssd[1]]
V7 [get_ports D_ssd[0]]
U18 [get_ports pb_in]
W5 [get_ports clk]
T18 [get_ports rst_h]

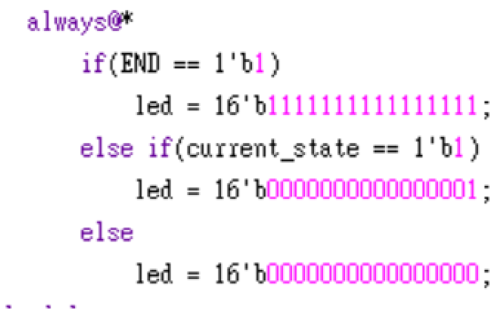
(2) VERILOG CODE

Module top



```
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));
one_pulse U2(.clk(clk), .in_trig(pb_debounced), .out_pulse(out_pulse));
display U3(.D_ssd(D_ssd), .in(ssd_in));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .cnt1(value0), .cnt2(value1), .control(clk_scan));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h), .current_state(current_state));
BCD_DOWN digit1(.limit(4'b0000), .clk(clk_1hz), .rst_h(rst_h), .decrease(count_en & ~END), .q(value0), .borrow(borrow[0]));
BCD_DOWN digit2(.limit(4'b0011), .clk(clk_1hz), .rst_h(rst_h), .decrease(borrow[0]), .q(value1), .borrow(borrow[1]));
check U22(.in1(value1), .in2(value0), .out(END));
```

按照 block diagram 接



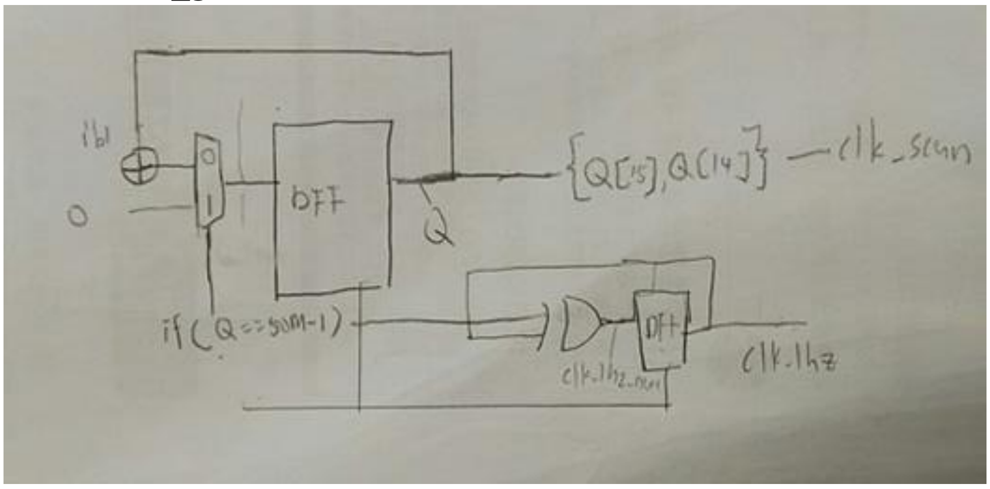
```
always@*
    if(END == 1'b1)
        led = 16'b1111111111111111;
    else if(current_state == 1'b1)
        led = 16'b0000000000000001;
    else
        led = 16'b0000000000000000;
```

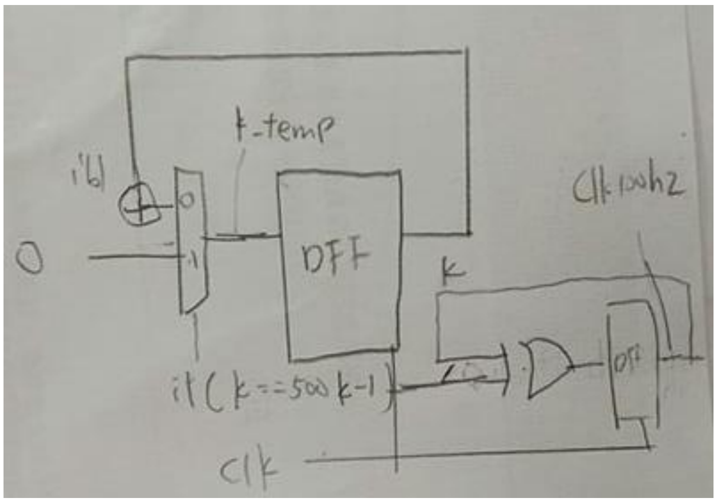
END 是 check MODULE 的 output 只有在 2 位都是 0 時才等於 1;

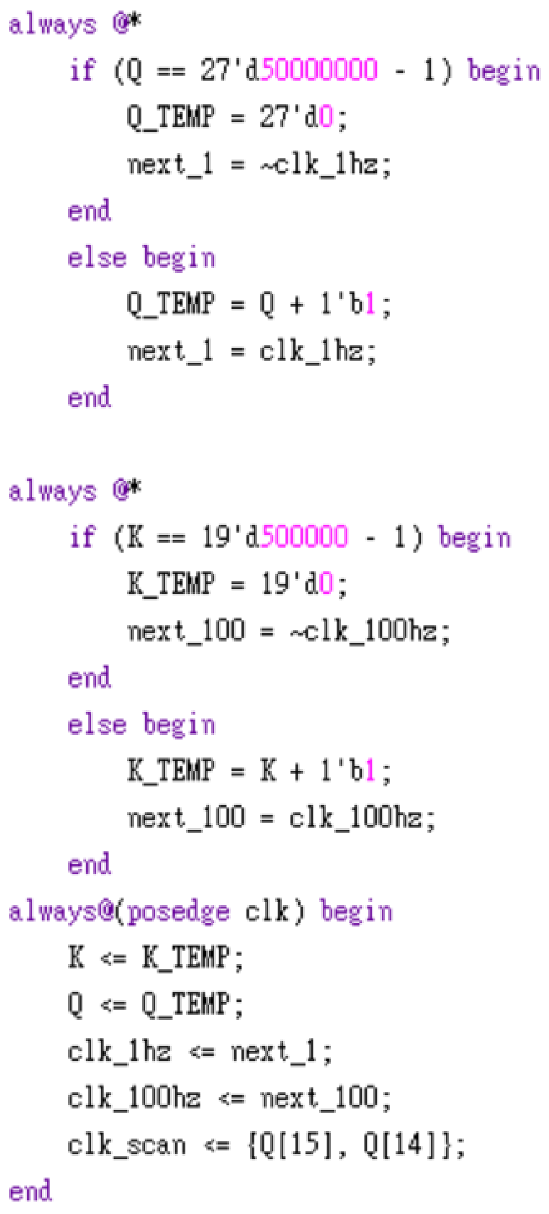
另外，我讓個位的 decrease 在乘上 ~END ，所以到 00 就停

BCD_DOWN digit1(.limit(4'b0000), .clk(clk), .rst(rst), .decrease(count_en & END), .q(value0), .borrow(borrow[0]));

Module clk_generator

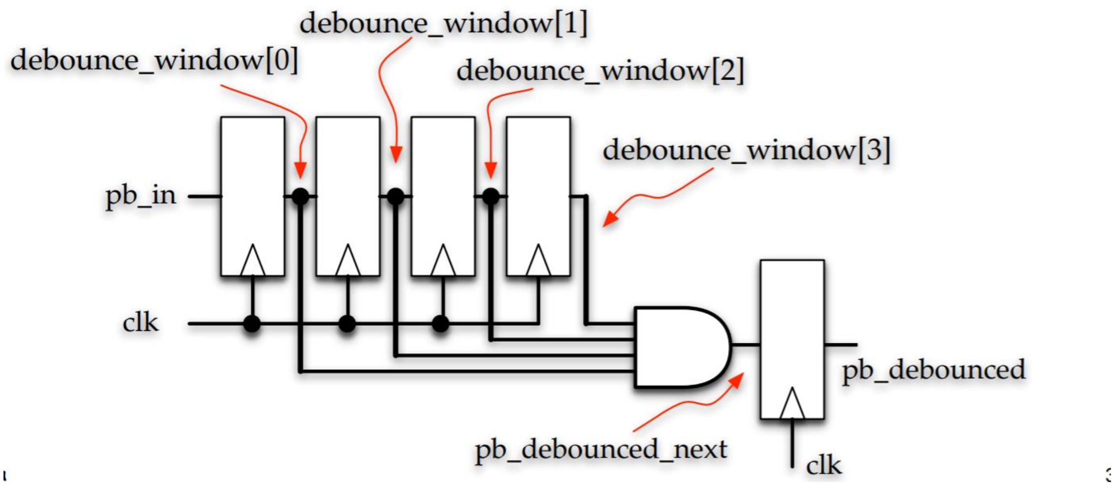






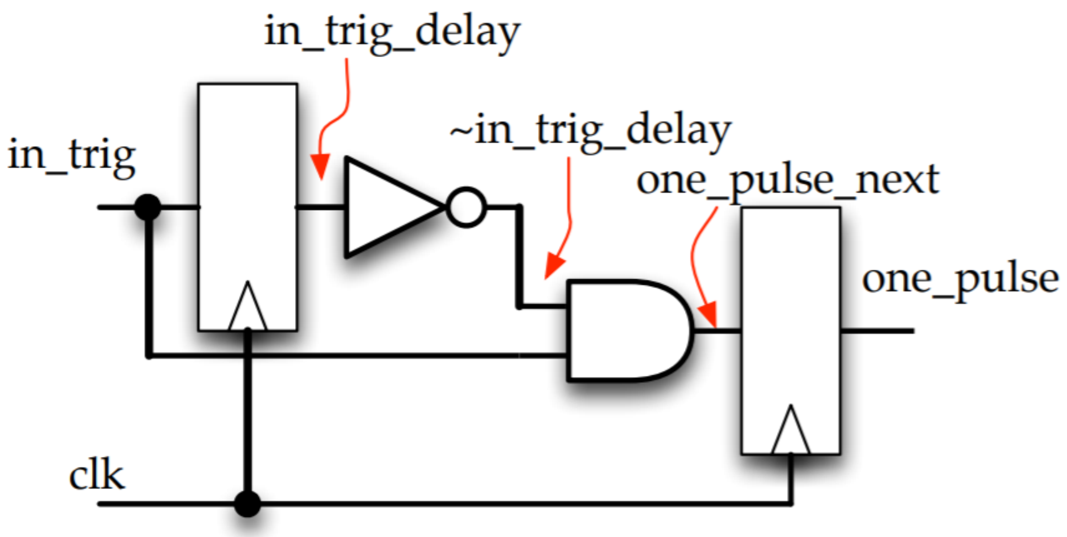
```
always @*
    if (Q == 27'd50000000 - 1) begin
        Q_TEMP = 27'd0;
        next_1 = ~clk_1hz;
    end
    else begin
        Q_TEMP = Q + 1'b1;
        next_1 = clk_1hz;
    end

always @*
    if (K == 19'd500000 - 1) begin
        K_TEMP = 19'd0;
        next_100 = ~clk_100hz;
    end
    else begin
        K_TEMP = K + 1'b1;
        next_100 = clk_100hz;
    end
always@(posedge clk) begin
    K <= K_TEMP;
    Q <= Q_TEMP;
    clk_1hz <= next_1;
    clk_100hz <= next_100;
    clk_scan <= {Q[15], Q[14]};
end
```

Module pb



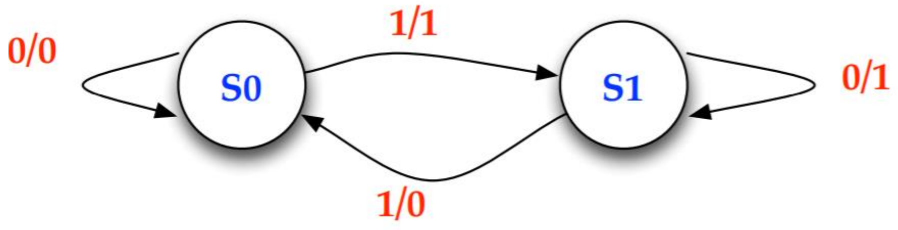
```
always@(posedge clk)
    debounce_window <= {debounce_window[2:0],pb_in};

assign pb_debounced_next = debounce_window[3] & debounce_window[2] & debounce_window[1] & debounce_window[0];

always@(posedge clk)
    pb_debounced  <= pb_debounced_next;
```

Module one_pulse



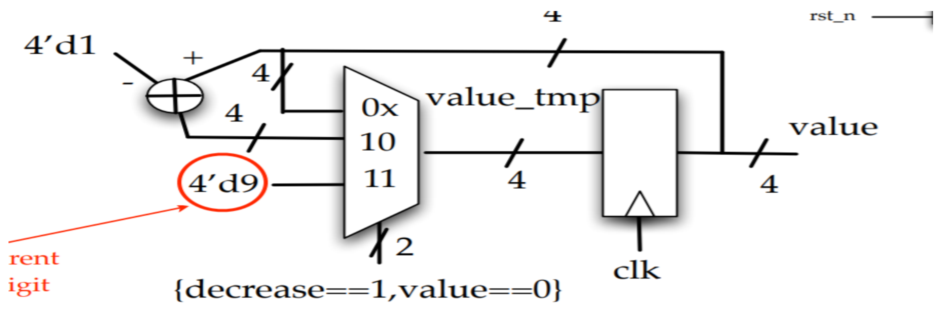
```
always@(posedge clk)
 in_trig_delay <= in_trig;

assign one_pulse_next = in_trig & (~in_trig_delay);

always@(posedge clk)
    out_pulse <= one_pulse_next;
```

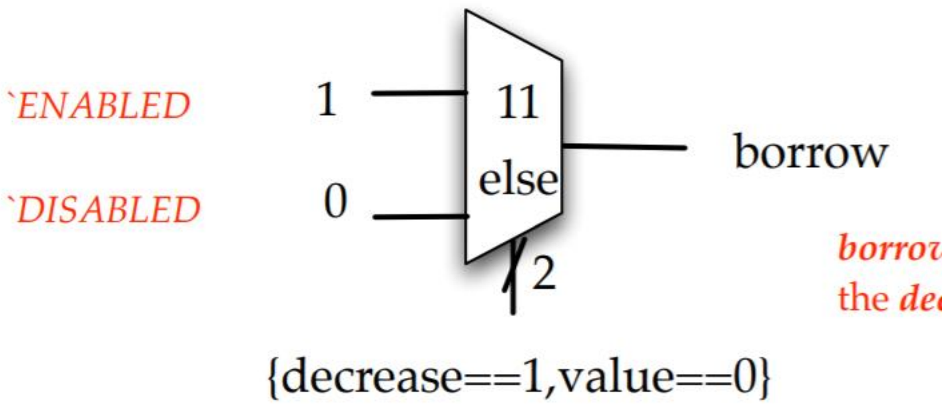
Module FSM

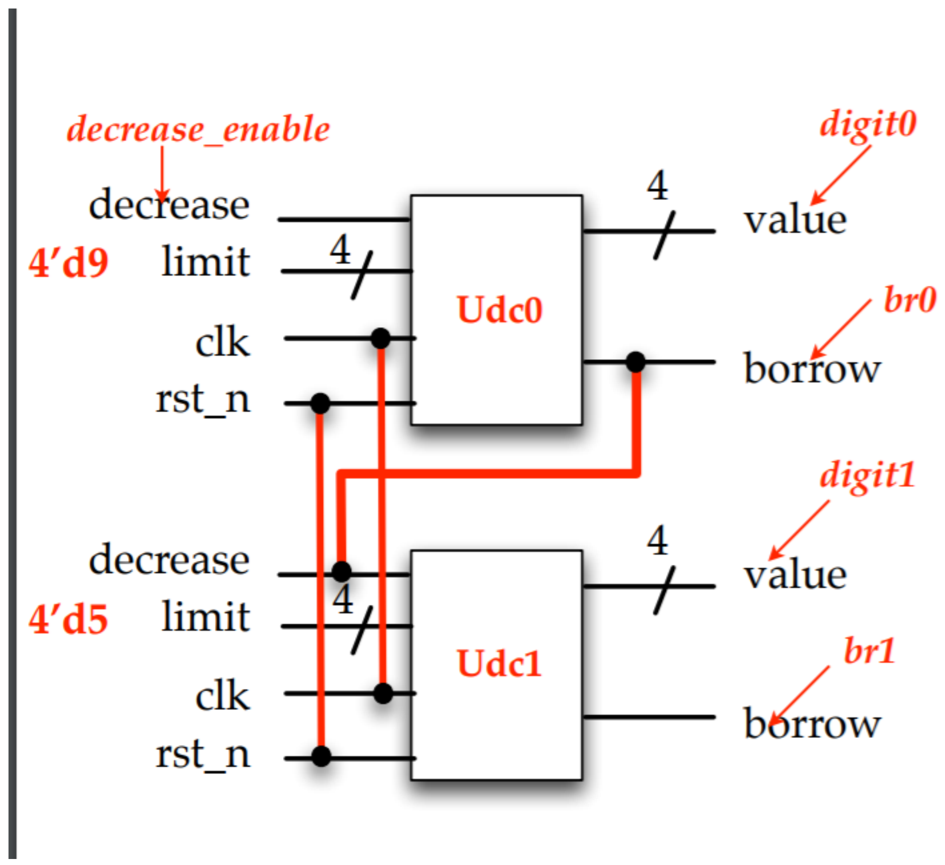


```
always@(posedge in or posedge rst_h)
if(rst_h)
    state <= 1'b0;
else
    state <= ~state;

assign current_state = state;
assign count_en = state;
```

MODULE bcd_down

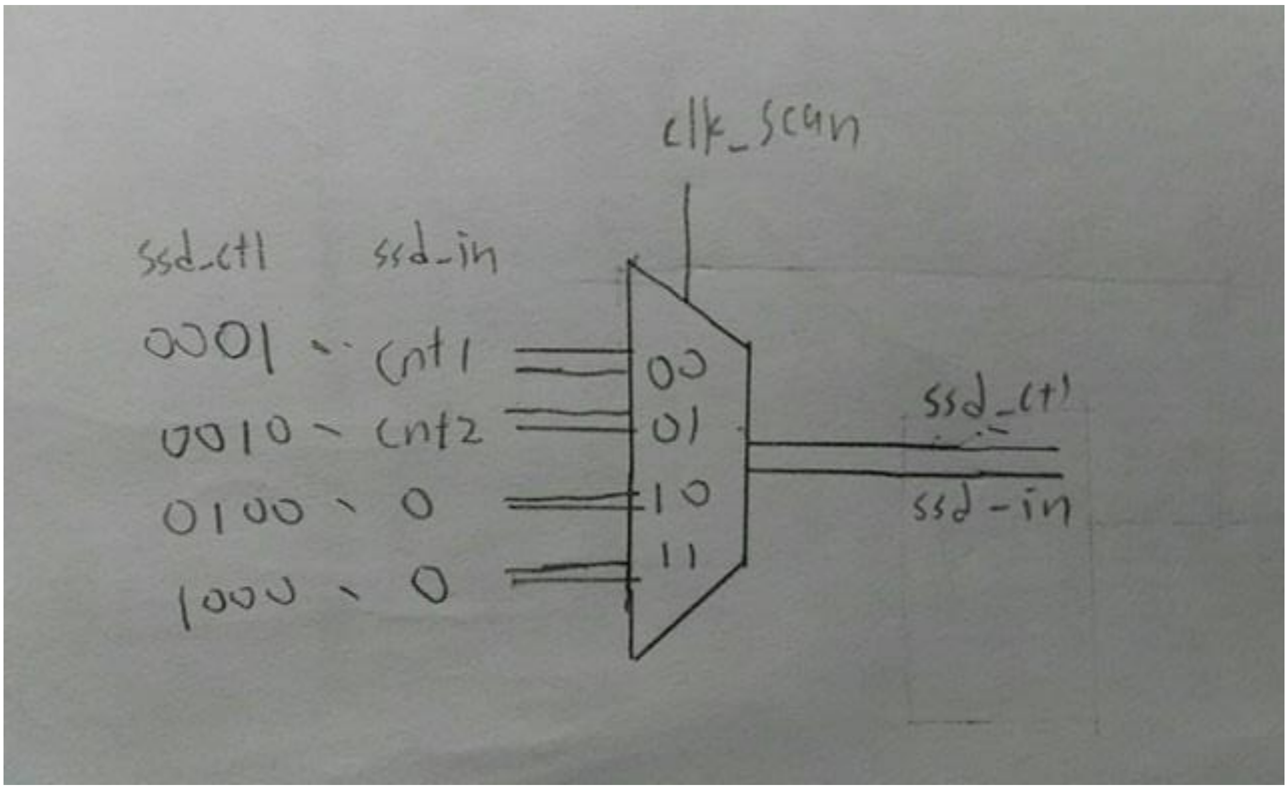


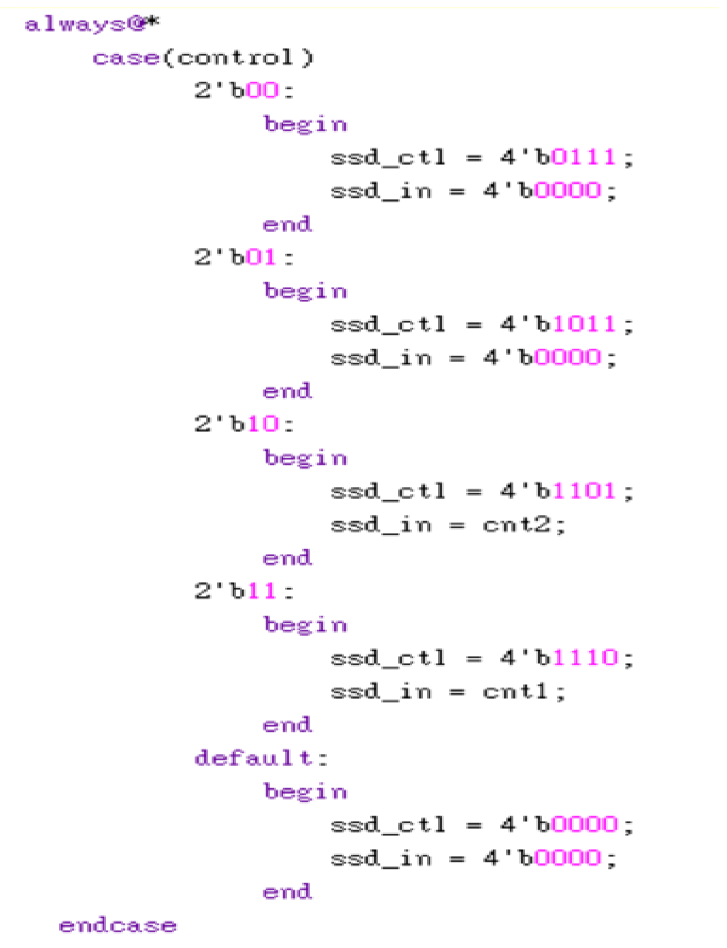

`ENABLED
`DISABLED
1
0
11
else
2
borrow
borrow
the de
{decrease==1,value==0}


decrease_enable
decrease
4'd9
limit
4
clk
rst_n
Udc0
4
value
digit0
br0
borrow
digit1
decrease
4'd5
limit
4
clk
rst_n
Udc1
4
value
br1
borrow

```
always @*
    if (q == 4'b0000 && decrease == 1) begin
        q_temp = 4'b1001;
        borrow = 1'b1;
    end
    else if (q != 4'b0000 && decrease == 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 <= limit;
    else q <= q_temp;
```

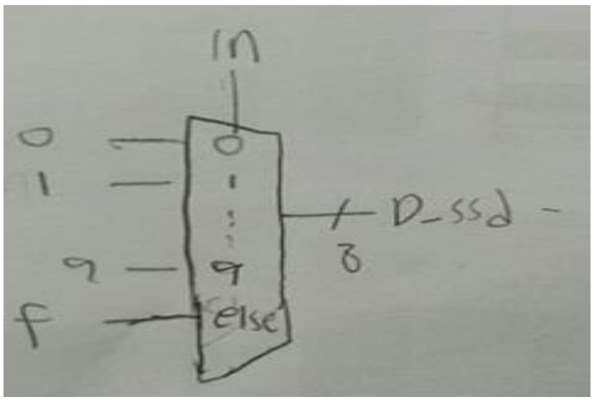
Module scan

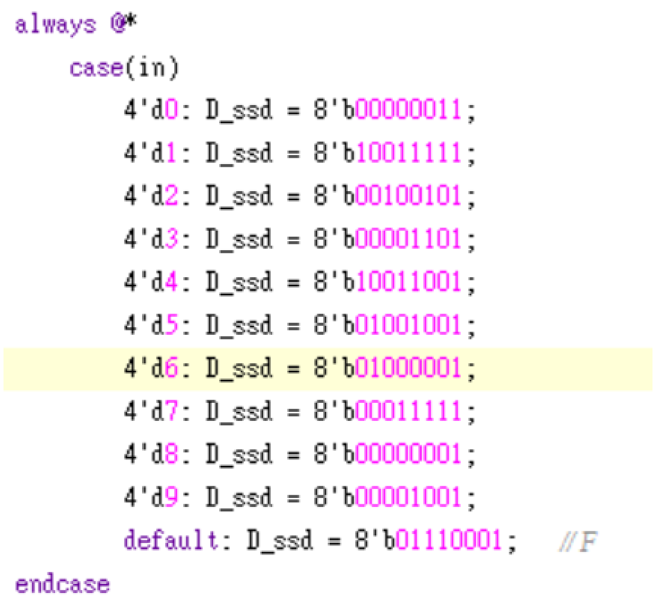




```
always@*
    case(control)
          2'b00:
              begin
                  ssd_ctl = 4'b0111;
                  ssd_in = 4'b0000;
              end
          2'b01:
              begin
                  ssd_ctl = 4'b1011;
                  ssd_in = 4'b0000;
              end
          2'b10:
              begin
                  ssd_ctl = 4'b1101;
                  ssd_in = cnt2;
              end
          2'b11:
              begin
                  ssd_ctl = 4'b1110;
                  ssd_in = cnt1;
              end
          default:
              begin
                  ssd_ctl = 4'b0000;
                  ssd_in = 4'b0000;
              end
  endcase
```

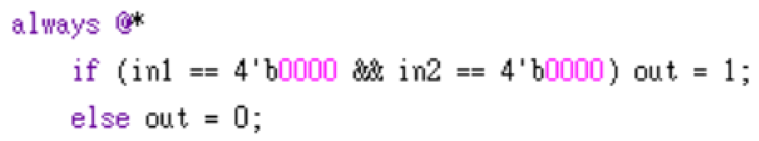
Module display





```
always @*
    case(in)
        4'd0: D_ssd = 8'b00000011;
        4'd1: D_ssd = 8'b10011111;
        4'd2: D_ssd = 8'b00100101;
        4'd3: D_ssd = 8'b00001101;
        4'd4: D_ssd = 8'b10011001;
        4'd5: D_ssd = 8'b01001001;
        4'd6: D_ssd = 8'b01000001;
        4'd7: D_ssd = 8'b00011111;
        4'd8: D_ssd = 8'b00000001;
        4'd9: D_ssd = 8'b00001001;
        default: D_ssd = 8'b01110001;   //F
endcase
```

Module check



```
always @*
    if (in1 == 4'b0000 && in2 == 4'b0000) out = 1;
    else out = 0;
```

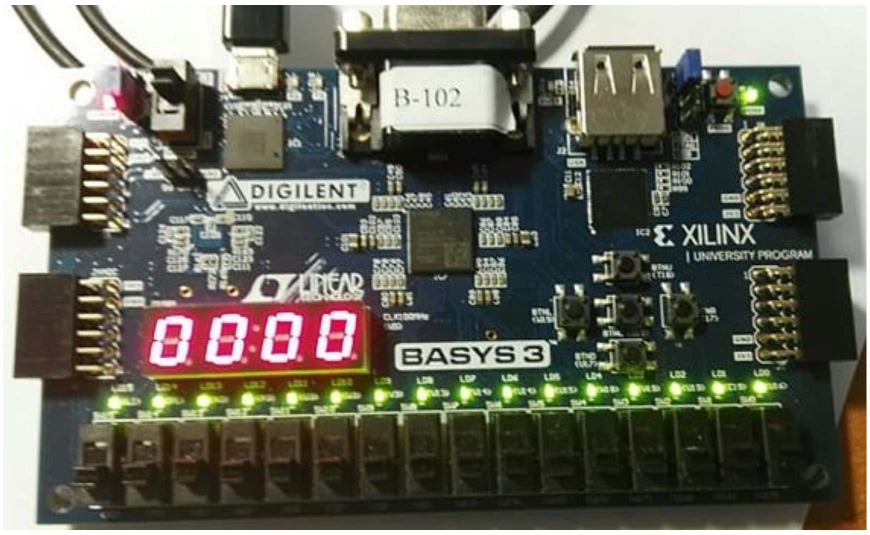
DISCUSSION

這題的想法就是原本 4-4 的倒數計時器再加上 DEBOUNCED、ONE_PULSE 跟 FSM，基本上和老師上課說得一樣，原本畫出圖以後覺得不難，但打出來後出現 2 個問題，第一個是按 PAUSE 鈕根本沒反應，於是我先讓 DECREASE = 1; 讓她必定會往下數，然後發現我的倒數也有問題，是一次數 6，後來發現是因為我數道 50M 時直接讓 CLK_1HZ = ~CLK_1HZ，沒有設一個 CLK_1HZ_NEXT，然後在 DFF 裡再讓 CLK_1HZ <= CLK_1HZ_NEXT，但我不知為何少了這個步驟就會一次數 6。

再來，我按下 RST 發現是可以運作的，所以知道問題出在 FSM，

我試過很多方法還是不行，後來在想會不會是 OUT_PULSE 出來的訊號沒有與 FSM 的 CLK 對到，所以乾脆 FSM 就不接 CLK 了，直接拿 IN 來當 CLK { always@(posedge in or posedge rst_h) }，結果就成功了，但我覺得很奇怪的一點就是，老師講義中的 FSM，count_enable 不受 FSM 中的 CLK 影響，照理來說 IN = 1 時，他就會馬上改變，所以可能也不是 CLK 沒對到的問題，而我站時也想不出來是什麼問題。

結果就是按 RST 回到 30 然後暫停，按 PB_IN 就是暫停 OR 繼續數，數到 00 時 LED 全亮。



CONCLUSION

這次本來以為很簡單，想說 PRE_LAB05 都打出來了，在接幾個 MODULE 就好，結果 DEBUG DE 到快往生，不知道花了幾天才搞出來，而且同學都很忙，我認識的人又不多，所以只能孤軍奮戰，很無助，FSM 也是嘗試過好幾種打法，接上每一種 CLK，都無法運作，後來用這種怪怪的方法用出來，也還想不透問題所在。

第二題

前言: 因為第 2 題與第一題過於相似，所以只討論不同的部分

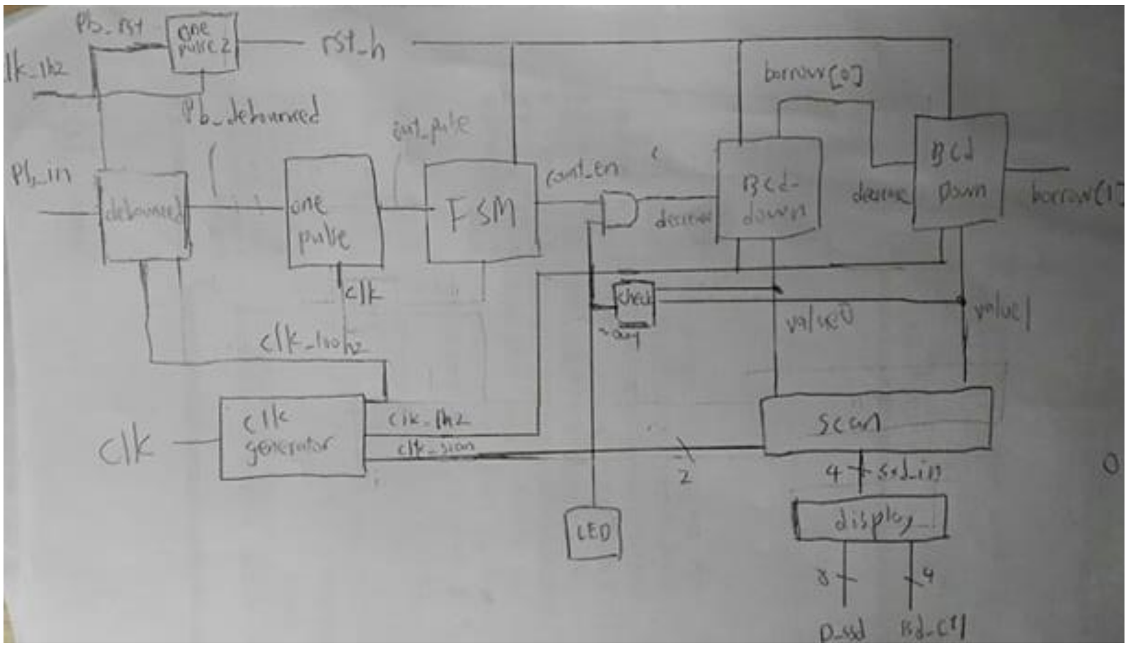
DESIGN SPECIFICATION

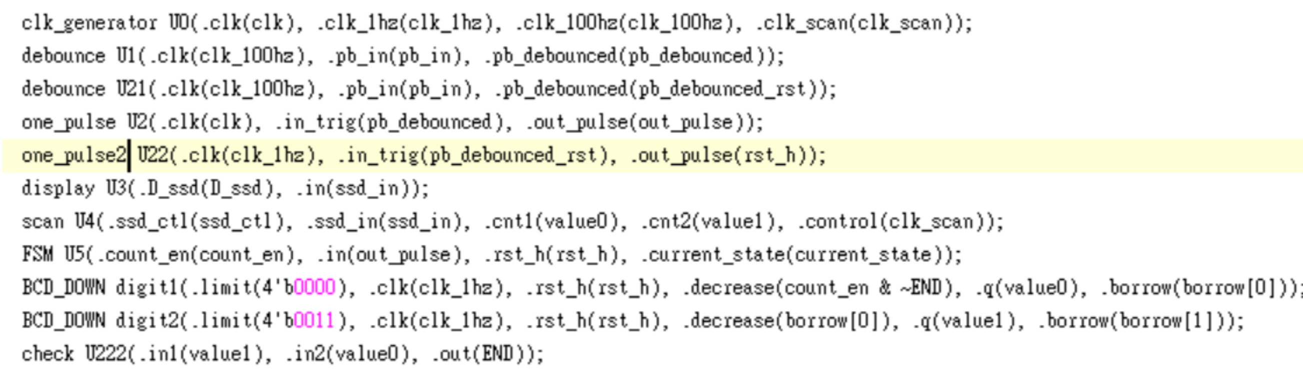
(1) INPUT & OUTPUT

```
module main(clk, pb_in, led, D_ssd, ssd_ctl);
    input clk;
    input pb_in;
    output reg [15:0] led;
    output [7:0] D_ssd;
    output [3:0] ssd_ctl;

    wire rst_h;
    wire pb_debounced_rst;
    wire pb_debounced;
    wire clk_1hz,clk_100hz;
    wire [1:0]clk_scan;
    wire out_pulse;
    wire [3:0] ssd_in;
    wire [3:0] value0;
    wire [3:0] value1;
    wire check;
    wire count_en;
    wire END;
    wire [1:0]borrow;
    wire current_state;

module one_pulse2(out_pulse, clk,    in_trig);
    output reg out_pulse;
    input clk;
    input in_trig;

    reg in_trig_delay;
    wire one_pulse_next;
```

其餘同

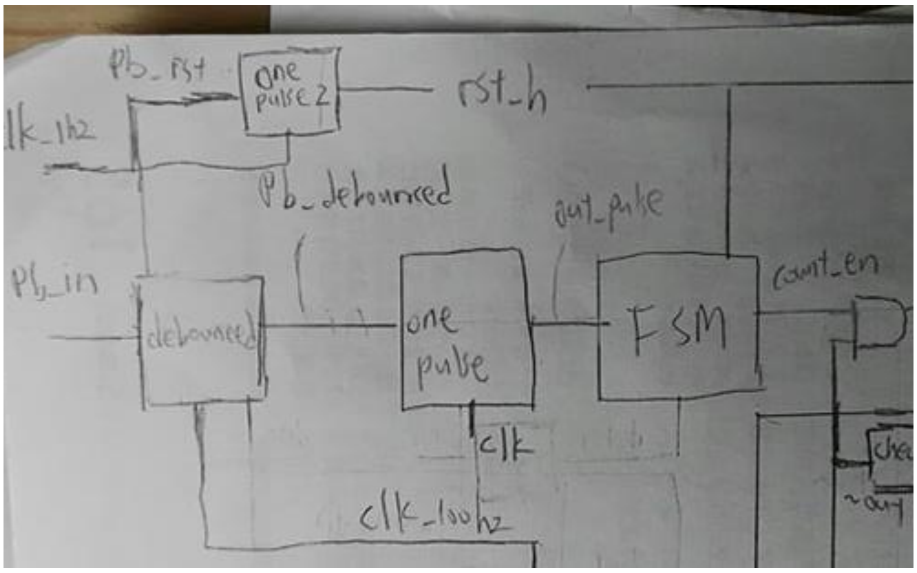
(2) BLOCK diagram





```
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_in), .pb_debounced(pb_debounced));
debounce U21(.clk(clk_100hz), .pb_in(pb_in), .pb_debounced(pb_debounced_rst));
one_pulse U2(.clk(clk), .in_trig(pb_debounced), .out_pulse(out_pulse));
one_pulse2 U22(.clk(clk_1hz), .in_trig(pb_debounced_rst), .out_pulse(rst_h));
display U3(.D_ssd(D_ssd), .in(ssd_in));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .cnt1(value0), .cnt2(value1), .control(clk_scan));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h), .current_state(current_state));
BCD_DOWN digit1(.limit(4'b0000), .clk(clk_1hz), .rst_h(rst_h), .decrease(count_en & ~END), .q(value0), .borrow(borrow[0]))
BCD_DOWN digit2(.limit(4'b0011), .clk(clk_1hz), .rst_h(rst_h), .decrease(borrow[0]), .q(value1), .borrow(borrow[1]));
check U222(.in1(value1), .in2(value0), .out(END));
```

主要不同的是



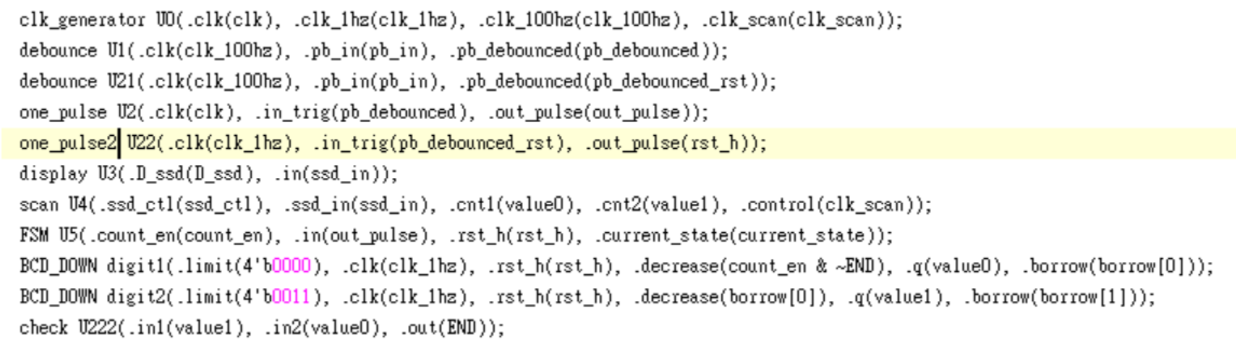
DESIGN IMPLEMENTATION

(1) I / O PIN

```
L1 [get_ports led[15]]
P1 [get_ports led[14]]
N3 [get_ports led[13]]
P3 [get_ports led[12]]
U3 [get_ports led[11]]
W3 [get_ports led[10]]
V3 [get_ports led[9]]
V13 [get_ports led[8]]
V14 [get_ports led[7]]
U14 [get_ports led[6]]
U15 [get_ports led[5]]
W18 [get_ports led[4]]
V19 [get_ports led[3]]
U19 [get_ports led[2]]
E19 [get_ports led[1]]
U16 [get_ports led[0]]
W4 [get_ports ssd_ctl[3]]
V4 [get_ports ssd_ctl[2]]
U4 [get_ports ssd_ctl[1]]
U2 [get_ports ssd_ctl[0]]
W7 [get_ports D_ssd[7]]
W6 [get_ports D_ssd[6]]
U8 [get_ports D_ssd[5]]
V8 [get_ports D_ssd[4]]
U5 [get_ports D_ssd[3]]
V5 [get_ports D_ssd[2]]
U7 [get_ports D_ssd[1]]
V7 [get_ports D_ssd[0]]
U18 [get_ports pb_in]
W5 [get_ports clk]
T18 [get_ports rst_h]
```

(2) Verilog code

Module main



```
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));
debounce U21(.clk(clk_100hz), .pb_in(pb_in), .pb_debounced(pb_debounced_rst));
one_pulse U2(.clk(clk), .in_trig(pb_debounced), .out_pulse(out_pulse));
one_pulse2 U22(.clk(clk_1hz), .in_trig(pb_debounced_rst), .out_pulse(rst_h));
display U3(.D_ssd(D_ssd), .in(ssd_in));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .cnt1(value0), .cnt2(value1), .control(clk_scan));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h), .current_state(current_state));
BCD_DOWN digit1(.limit(4'b0000), .clk(clk_1hz), .rst_h(rst_h), .decrease(count_en & ~END), .q(value0), .borrow(borrow[0]));
BCD_DOWN digit2(.limit(4'b0011), .clk(clk_1hz), .rst_h(rst_h), .decrease(borrow[0]), .q(value1), .borrow(borrow[1]));
check U222(.in1(value1), .in2(value0), .out(END));
```

相較第一題　多了

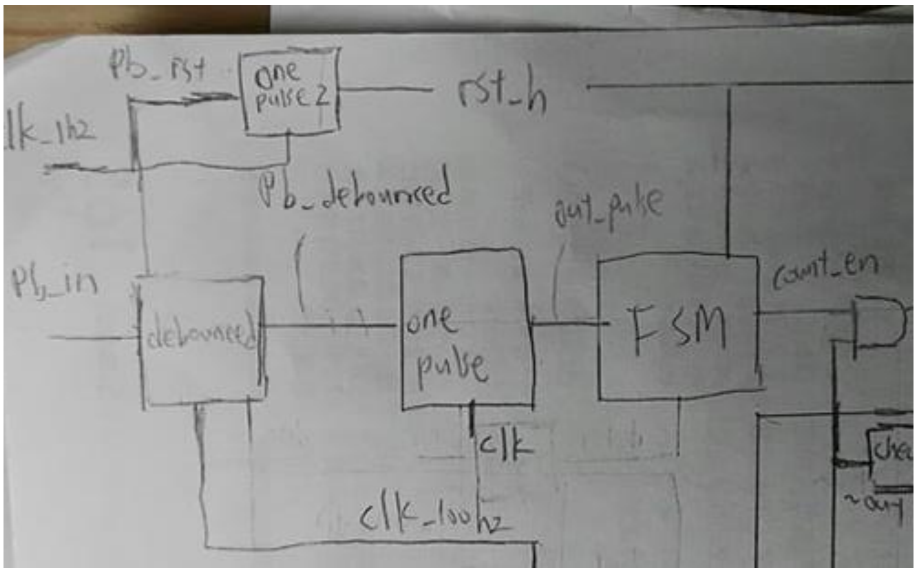
Debounce

U21(.clk(clk_100hz), .pb_in(pb_in), .pb_debounced(pb_debounced_rst));

one_pulse2 U22(.clk(clk_1hz), .in_trig(pb_debounced_rst), .out_pulse(rst_h));

讓 pb_in 在多控制一個 wire(pb_debounced_rst)，並經由不同的

one_pulse 生成 rst_h

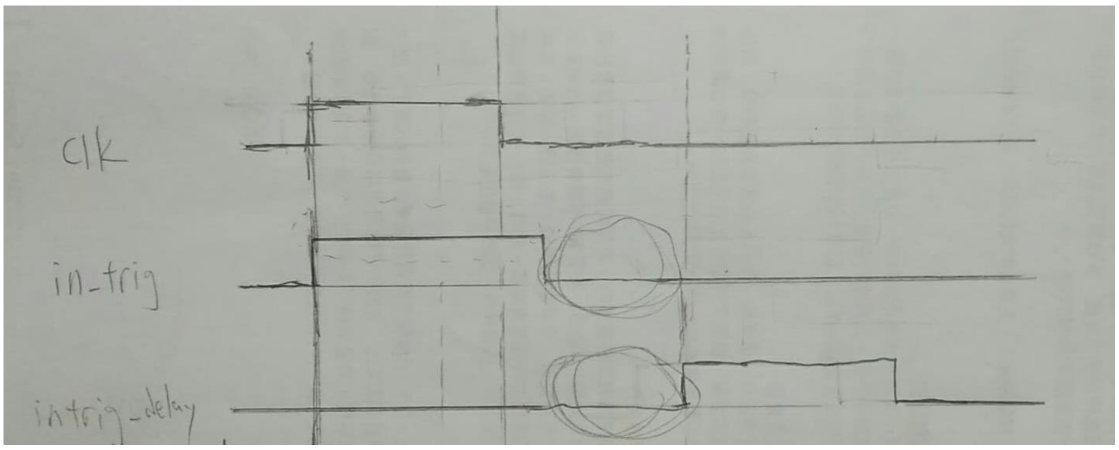


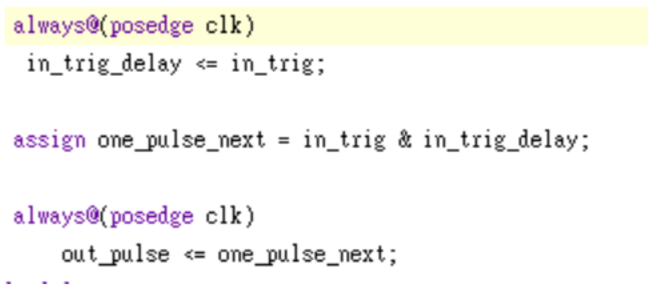
Module one_pulse2

我的想法是把 CLK 延長，然後讓 intrig & intrig_delay，所以如果按

的時間小於 CLK 的一周期，out_pulse 不會有訊號，如圖　intrig

跟　intrig_delay 相乘只會是零



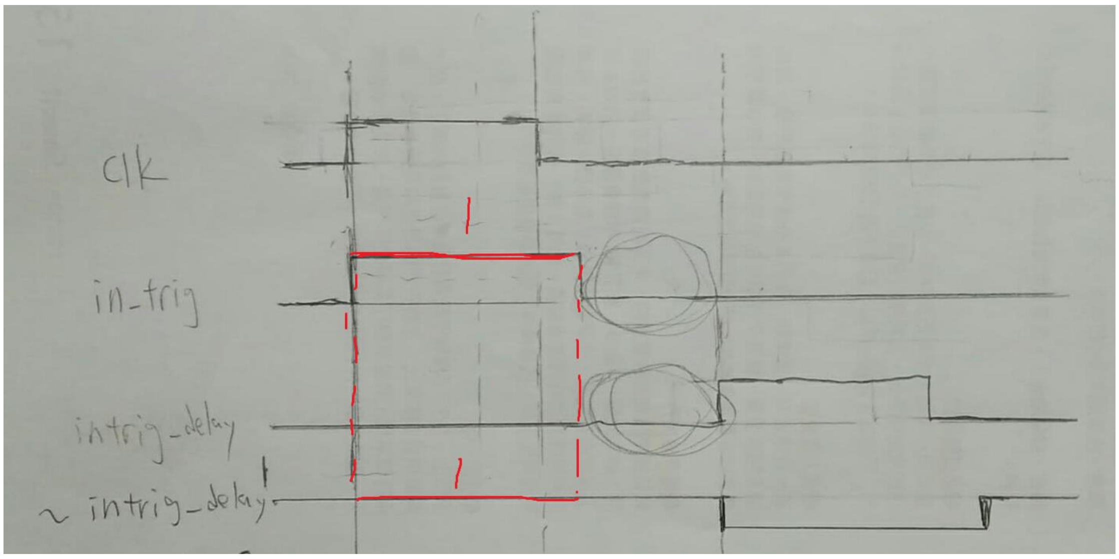


```
always@(posedge clk)
  in_trig_delay <= in_trig;

assign one_pulse_next = in_trig & in_trig_delay;

always@(posedge clk)
     out_pulse <= one_pulse_next;
```

Disscussion

這題我把我的想法打上去就成功了，意外的順利，唯一遇到的麻煩點是在，我直接把 5-1 的 FILE 加進來，結果我在修改時竟然把 5-1 的內容改掉了，後來只好再開一個 FILE，並用複製貼上的方式。

這題還有一個問題，就是我使用沒有修改果的 ONE_PULSE 竟然也成功了(one_pulse_next = in_trig & ~in_trig_delay)， 可是我覺得這不符合邏輯阿，如果要乘上~in_trig_delay，就算短於 CLK 還是會有訊號(如圖)，感覺我對這部分理解還不是很清楚。



結果就是，快速的按下 BUTTON 是 PAUSE/START，按住約莫 1sec 就回到 30 並暫停。

CONCLUSION

5-1 花的時間讓我對 push button 心生恐懼，結果 5-2 居然一次成功，真是爽爆了，唯一的遺憾是對 one_pulse 那邊沒搞很清楚，所以感覺是矇到的。

第三題

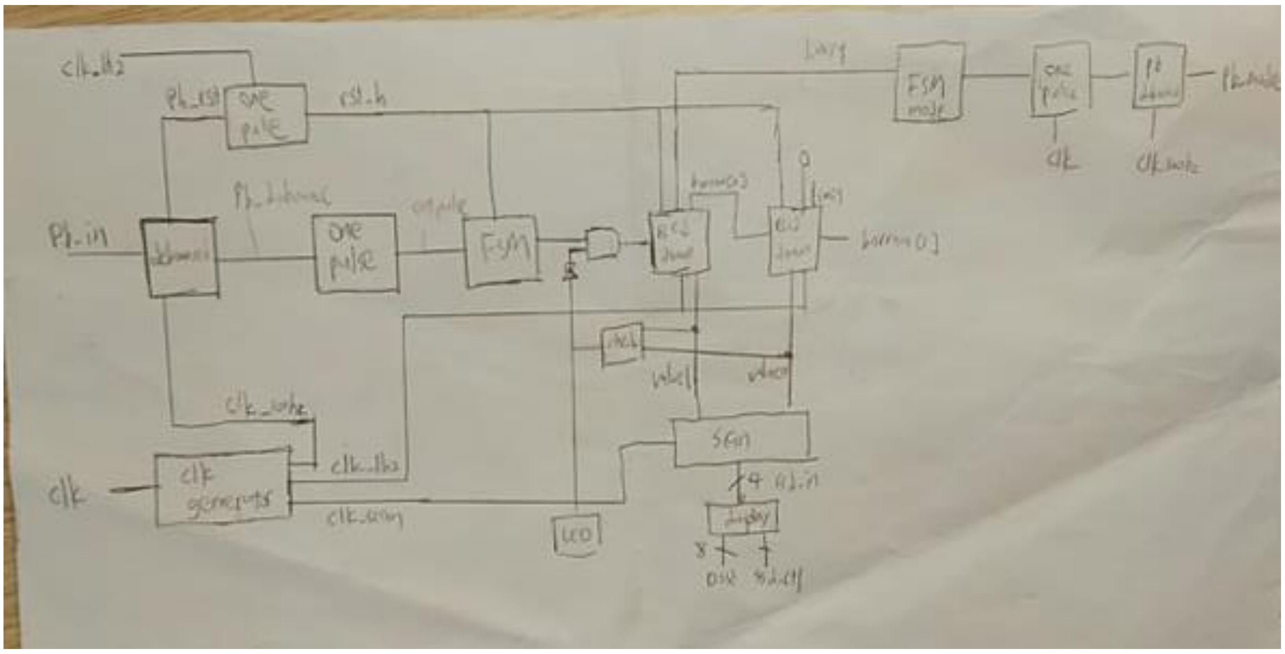
前言: 這題由於與第二題相似，於是也只討論不同的部分

Design specification

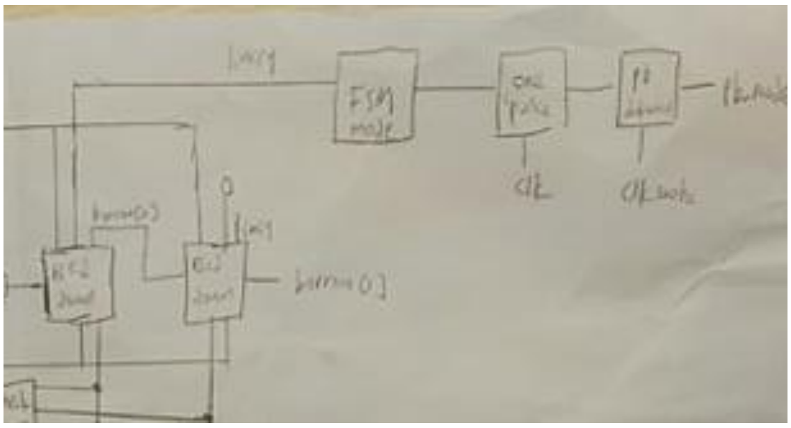
(1) Input / output

```
module main(clk, pb_in, pb_mode, led, D_ssd, ssd_ctl);
	input clk;
	input pb_in;
	input pb_mode;
	output reg [15:0]led;
	output [7:0]D_ssd;
	output [3:0]ssd_ctl;

	wire rst_h;
	wire pb_debounced_rst;
	wire pb_debounced;
	wire pb_debounced_mode;
	wire out_pulse_mode;
	wire clk_1hz;
	wire clk_100hz;
	wire [1:0]clk_scan;
	wire out_pulse;
	wire [3:0]limit;
	wire [3:0] ssd_in;
	wire [3:0] value0;
	wire [3:0] value1;
	wire check;
	wire count_en;
	wire END;
	wire [1:0]borrow;
	wire current_state;

module FSM_mode(in, rst, out);
	input in;
	input rst;
	output reg [3:0]out;
	reg state;
	reg state_next;
```

(2) Block diagram



與 5-2 主要不同的地方是這裡



DESIGN IMPLEMENTATION

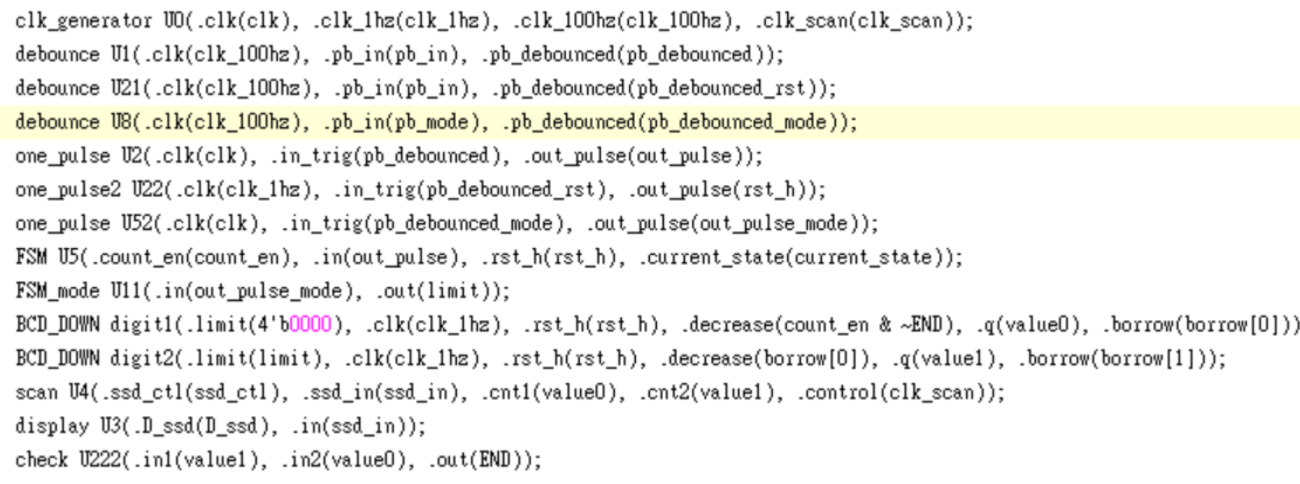
(1) I / O PINS

L1 [get_ports led[15]]
P1 [get_ports led[14]]
N3 [get_ports led[13]]
P3 [get_ports led[12]]
U3 [get_ports led[11]]
W3 [get_ports led[10]]
V3 [get_ports led[9]]
V13 [get_ports led[8]]
V14 [get_ports led[7]]

U14 [get_ports led[6]]
U15 [get_ports led[5]]
W18 [get_ports led[4]]
V19 [get_ports led[3]]
U19 [get_ports led[2]]
E19 [get_ports led[1]]
U16 [get_ports led[0]]
W4 [get_ports ssd_ctl[3]]
V4 [get_ports ssd_ctl[2]]
U4 [get_ports ssd_ctl[1]]
U2 [get_ports ssd_ctl[0]]
W7 [get_ports D_ssd[7]]
W6 [get_ports D_ssd[6]]
U8 [get_ports D_ssd[5]]
V8 [get_ports D_ssd[4]]
U5 [get_ports D_ssd[3]]
V5 [get_ports D_ssd[2]]
U7 [get_ports D_ssd[1]]
V7 [get_ports D_ssd[0]]
U18 [get_ports pb_in]
W5 [get_ports clk]
T18 [get_ports pb_mode]

(2) Verilog code

Module main



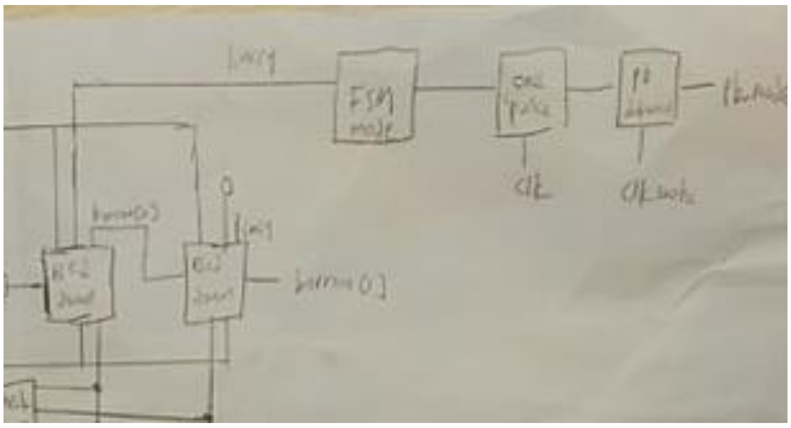
```
clk_generator U0(.clk(clk), .clk_1hz(clk_1hz), .clk_100hz(clk_100hz), .clk_scan(clk_scan));
debounce U1(.clk(clk_100hz), .pb_in(pb_in), .pb_debounced(pb_debounced));
debounce U21(.clk(clk_100hz), .pb_in(pb_in), .pb_debounced(pb_debounced_rst));
debounce U8(.clk(clk_100hz), .pb_in(pb_mode), .pb_debounced(pb_debounced_mode));
one_pulse U2(.clk(clk), .in_trig(pb_debounced), .out_pulse(out_pulse));
one_pulse2 U22(.clk(clk_1hz), .in_trig(pb_debounced_rst), .out_pulse(rst_h));
one_pulse U52(.clk(clk), .in_trig(pb_debounced_mode), .out_pulse(out_pulse_mode));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h), .current_state(current_state));
FSM_mode U11(.in(out_pulse_mode), .out(limit));
BCD_DOWN digit1(.limit(4'b0000), .clk(clk_1hz), .rst_h(rst_h), .decrease(count_en & ~END), .q(value0), .borrow(borrow[0]))
BCD_DOWN digit2(.limit(limit), .clk(clk_1hz), .rst_h(rst_h), .decrease(borrow[0]), .q(value1), .borrow(borrow[1]));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .cnt1(value0), .cnt2(value1), .control(clk_scan));
display U3(.D_ssd(D_ssd), .in(ssd_in));
check U222(.in1(value1), .in2(value0), .out(END));
```

這題多設了一個 push button(pb_mode) 來控制 change mode 之 input

我還寫了一個 FSM(FSM_mode) 去控制 mode，然後她的 output

結果就只有 3 跟 6，並接到 BCD_DOWN 的 limit(初始值)

```
FSM_mode U11(.in(out_pulse_mode), .out(limit));
BCD_DOWN
digit2(.limit(limit), .clk(clk_1hz), .rst_h(rst_h), .decrease(borrow[0]), .q
(value1), .borrow(borrow[1]));
```



所以 BCD_DOWN 的初始值就只會在 30 跟 60 之間切換

其餘線路都與 5-2 同

MODULE FSM_mode

我只接用 in 來當 trigger，不接 clk 了

如果 state = 1 是 60， state = 0 是 30 ， in = 1 就切換

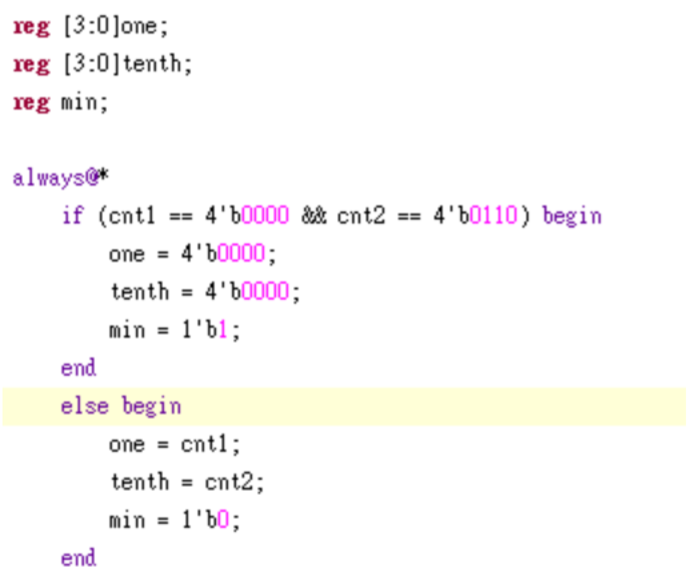
```
always@* state = state_next;
always@(posedge in) state_next <= ~state;

always@*
     if (state) out = 4'b0110;
     else out = 4'b0011;
```

Module scan

Module scan 多加了這些，為了顯示出 1.00

one 是個位、tenth 是十位、第三位是‘.’、第四位是分鐘

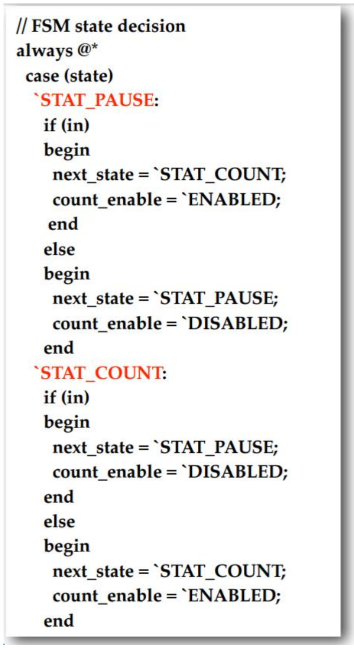
這裡是判斷如果 input == 60 則顯示 1.00



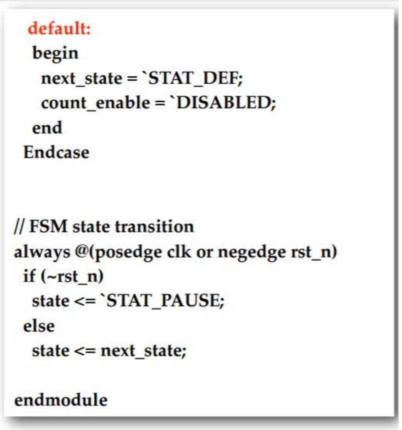
```
reg [3:0]one;
reg [3:0]tenth;
reg min;

always@*
    if (cnt1 == 4'b0000 && cnt2 == 4'b0110) begin
        one = 4'b0000;
        tenth = 4'b0000;
        min = 1'b1;
    end
    else begin
        one = cnt1;
        tenth = cnt2;
        min = 1'b0;
    end
```

DISCUSSION

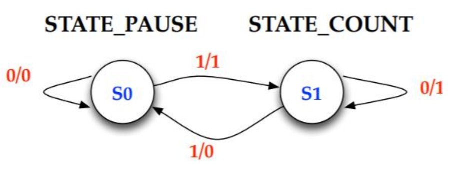
這次我 FSM 再試試老師的方法



```
// FSM state decision
always @*
  case (state)
   `STAT_PAUSE:
     if (in)
     begin
       next_state = `STAT_COUNT;
       count_enable = `ENABLED;
      end
     else
     begin
       next_state = `STAT_PAUSE;
       count_enable = `DISABLED;
     end
   `STAT_COUNT:
     if (in)
     begin
       next_state = `STAT_PAUSE;
       count_enable = `DISABLED;
     end
     else
     begin
       next_state = `STAT_COUNT;
       count_enable = `ENABLED;
     end
```

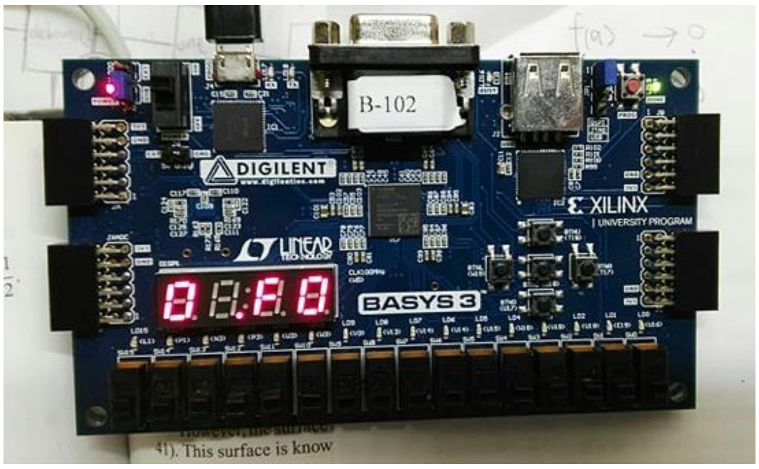


```
 default:
  begin
    next_state = `STAT_DEF;
    count_enable = `DISABLED;
  end
 Endcase


// FSM state transition
always @(posedge clk or negedge rst_n)
  if (~rst_n)
    state <= `STAT_PAUSE;
  else
    state <= next_state;

endmodule
```



然後 one_pule 跟 FSM 的 CLK 都接 100hz，可是還是沒有用，於是我還是採用比較奇怪的打法。

途中我遇到的問題有按下 change_mode 顯示的卻是 0.f0



後來發現我把 4’b0110 打成 4’d0110

還有一個問題我還沒解決，那就是 program device 完的初始值是 0.20，可是我明明完全沒有寫到 2 這個數字，我懷疑是某些東西沒有初始值，所以加上 rst 或用 initial 給值，但都沒用，所以又刪掉了。

這題其實只是 5-2 再多接一個 push button 跟 FSM。想法上沒很難，但還是要 DE 一堆 BUG。

CONCLUSION

打完 LAB05 這幾題，讓我覺得 VERILOG 真的是一個很難的東西，有時候你知道自己有錯，卻很難 DEBUG，感覺自己邏輯沒有打錯，可是還是會錯，CLK 的問題也是很麻煩，時間差也會影響

很大，但是卻很難思考到問題所在，像 FSM 的問題我也不知道出在哪裡。