LAB06

第一題

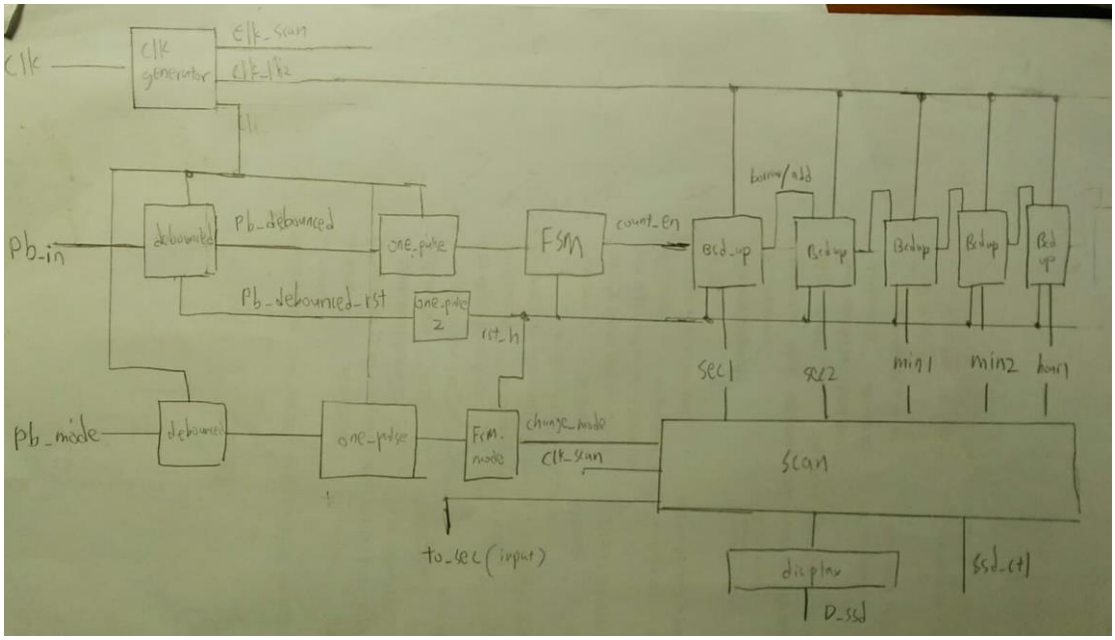
DESIGN SPECIFICATION

(1) INPUT & OUTPUT

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

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

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

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

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

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

module BCD_UP(limit, clk, rst_h, add, q, borrow);
 input [4:0]limit;
 input clk;
 input rst_h;
 input add;
 output reg [4:0]q;
 output reg borrow;

module scan(ssd_ctl, ssd_in, sec1, sec2, min1, min2, hour, control,
 change_mode, to_sec, PM);
 output reg [3:0] ssd_in;
 output reg [3:0] ssd_ctl;
 output reg PM;
 input to_sec;
 input [3:0]sec1;
 input [3:0]sec2;
 input [3:0]min1;
 input [3:0]min2;
 input [4:0]hour;
 input [1:0] control;
 input change_mode;

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));
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));
FSM_mode u11(.in(out_pulse_mode), .rst_h(rst_h), .state(change_mode));
BCD_UP digit1(.limit(4'b0001), .clk(clk_1hz), .rst_h(rst_h), .add(count_en), .q(sec1), .borrow(borrow[0]));
BCD_UP digit2(.limit(4'b0001), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[0]), .q(sec2), .borrow(borrow[1]));
BCD_UP digit3(.limit(4'b0001), .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]));
BCD_UP digit5(.limit(5'b10111), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[3]), .q(hour), .borrow(borrow[4]));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .min1(min1), .min2(min2), .hour(hour), .sec1(sec1), .sec2(sec2),
.control(clk_scan), .change_mode(change_mode), .to_sec(to_sec), .PM(PM));
display(.D_ssd(D_ssd_temp), .in(ssd_in));
```

```
always@*
    if (PM & change_mode) D_ssd = D_ssd_temp - 1'b1;
    else D_ssd = D_ssd_temp;
```

DESIGN IMPLEMENTATION

(1) I / O PINS

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]
V17 [get_ports to_sec]

(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));
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));
FSM_mode u11(.in(out_pulse_mode), .rst_h(rst_h), .state(change_mode));
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]));
BCD_UP digit5(.limit(5'b10111), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[3]), .q(hour), .borrow(borrow[4]));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .min1(min1), .min2(min2), .hour(hour), .sec1(sec1), .sec2(sec2),
.control(clk_scan), .change_mode(change_mode), .to_sec(to_sec), .PM(PM));
display(.D_ssd(D_ssd_temp), .in(ssd_in));
```

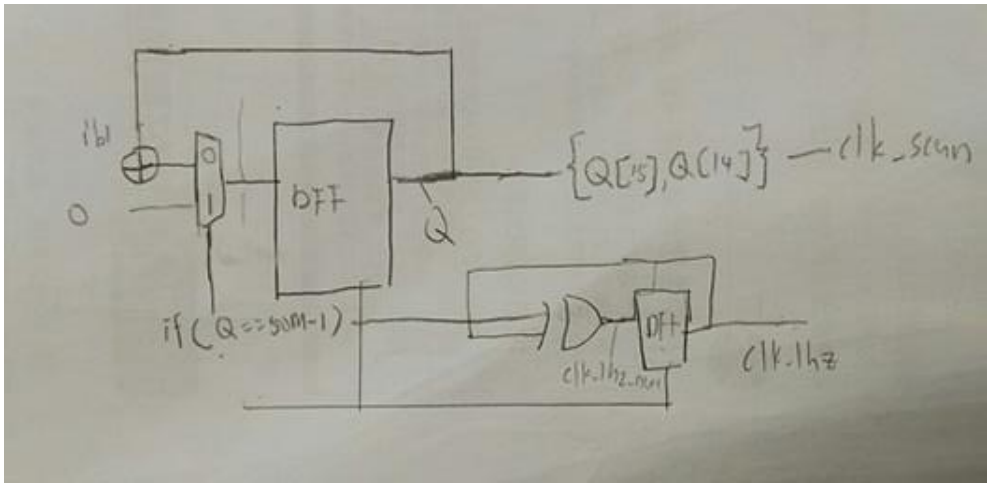
按照 block diagram 接

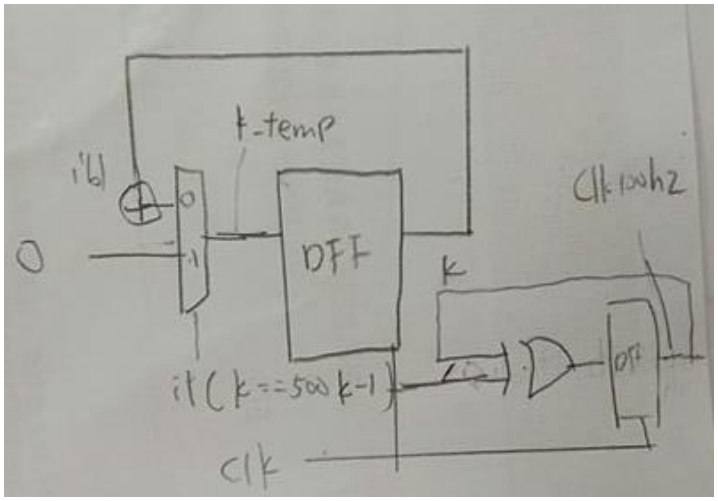
```
always@*
    if (PM & change_mode) D_ssd = D_ssd_temp - 1'b1;
    else D_ssd = D_ssd_temp;
```

Change_mode 是判斷是不是 AM/PM 制

PM 是來判斷是不是 PM 的，如果都是就把 D_ssd 減 1，讓點亮

Module clk_generator

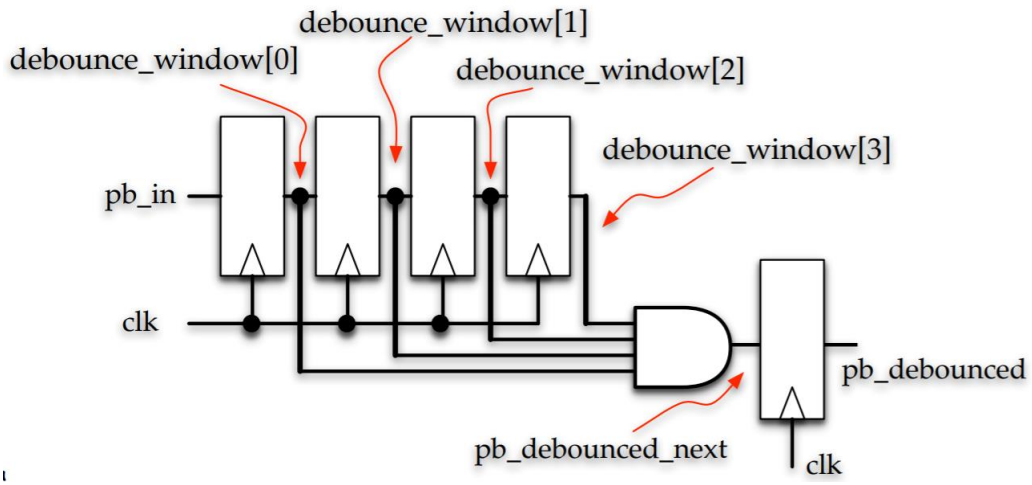




數到 50M 跟 500K 重製，產生 1HZ 跟 100HZ

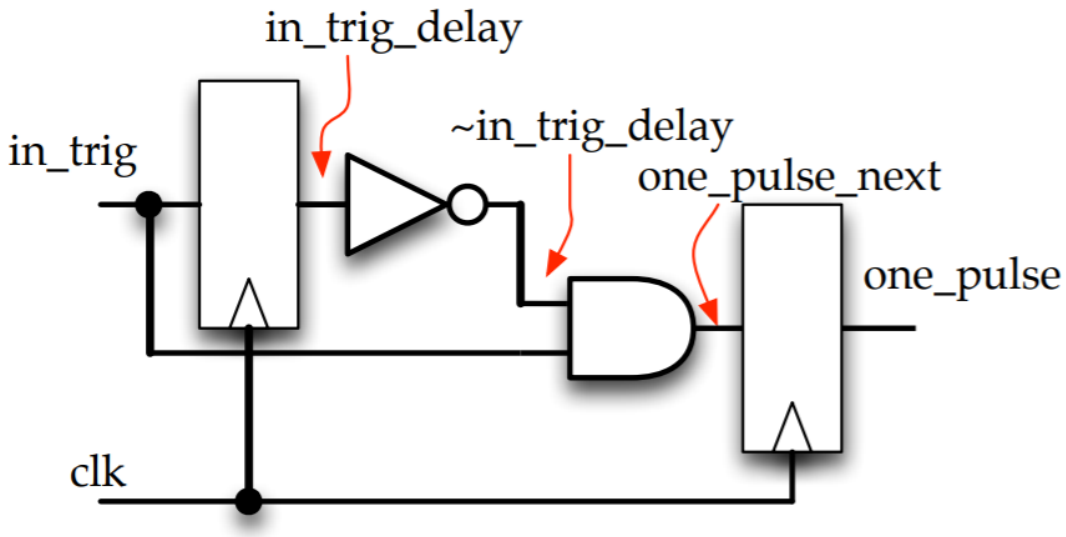
```
always @*
    if (Q == 27'd50000000 - 1) begin
        Q_TEMP = 27'd0;
        next_1 = ~clk_1hz;
    end
    else begin
        Q_TEMP = Q + 1'b1;
        next_1 = clk_1hz;
    end

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;
```

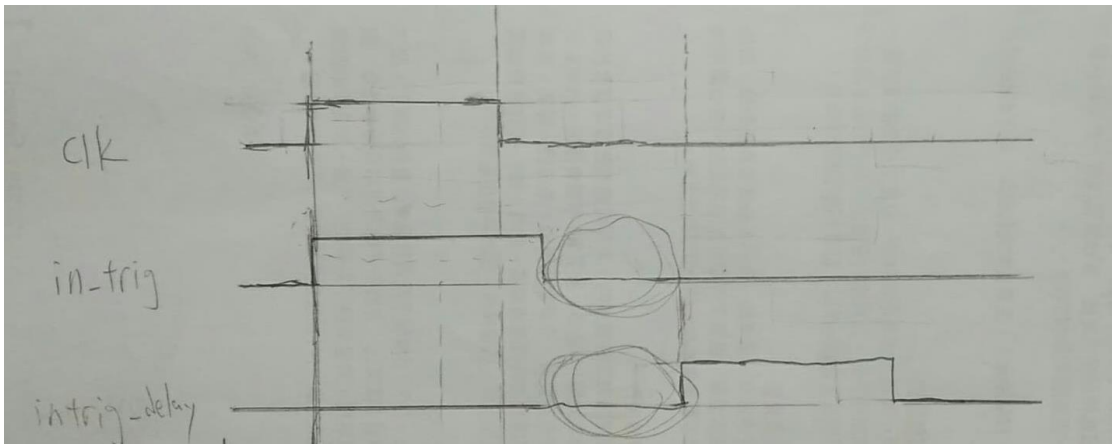
Module one_pulse




```
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 one_pulse2

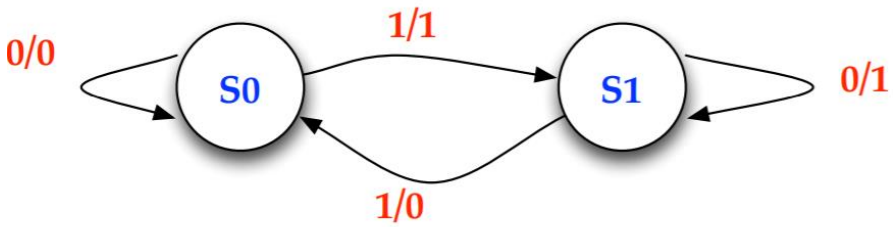
我的想法是把 CLK 延長，然後讓 intrig & intrig_delay，所以如果按的時間小於 CLK 的一周期，out_pulse 不會有訊號，如圖 intrig 跟 intrig_delay 相乘只會是零




```
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;
```

Module FSM

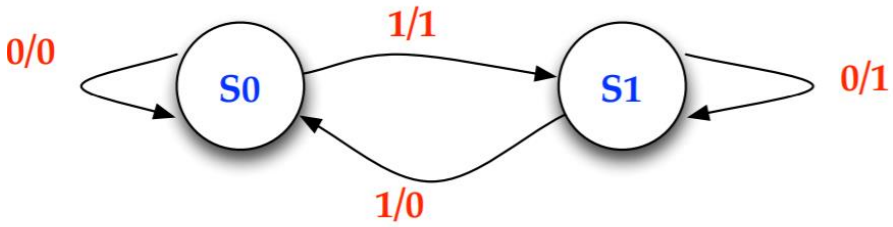
STATE	
1	Count
0	Pause



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

Module FSM_mode

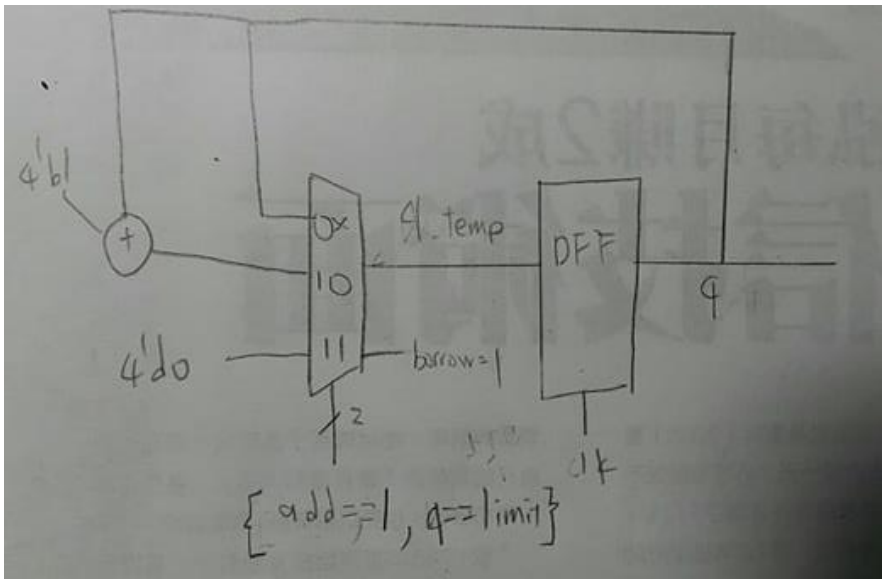
State	
1	AM/PM
0	24HR




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

Module BCD_UP

我的 BCD_UP 不是 BCD_UP，只是普通的 UP COUNTER，但還是有跟下一位借位的功能，改成普通的 UP COUNTER 的原因是 BCD_UP 只有在上限的個位數是 9 的情況下才好用，像 19、59、199 等，但 HOUR(小時)的上限是 23，不太方便，所以改。




```
always @*
    if (q == limit && add == 1) begin
        q_temp = 8'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 <= 8'd0;
    else q <= q_temp;
```

Module scan

Scan 這裡比較複雜，因為我一開始不知道 AM/PM 如何表示，所以就自己打了一個 AM、PM 上去，後來才知道用點，於是又把它加上去。

```
always@*
    if (change_mode) begin
        if (hour > 5'd12) begin
            PM = 1'b1;
            cnt4 = (hour - 4'd12) / 6'd10;
            cnt3 = (hour - 4'd12) % 6'd10;
            cnt2 = min2;
            cnt1 = min1;
        end
        else if (hour == 5'd0) begin
            cnt4 = 4'd1;
            cnt3 = 4'd2;
            PM = 1'b0;
            cnt2 = min2;
            cnt1 = min1;
        end
         else if (hour == 5'd12) begin
             cnt4 = 4'd1;
             cnt3 = 4'd2;
             PM = 1'b1;
             cnt2 = min2;
             cnt1 = min1;
        end
        else begin
            PM = 1'b0;
            cnt4 = hour / 4'd10;
            cnt3 = hour % 4'd10;
            cnt2 = min2;
            cnt1 = min1;
        end
    end
```

如果 change_mode == 1，表示用 AM/PM 表示

PM == 1 表示是是下午，PM 是 OUTPUT，丟回主程式判斷是否讓

點亮

HOUR(24)	AM/PM	PM
HOUR > 12	HOUR - 12	1
HOUR == 12	不變	1
HOUR == 0	HOUR = 12	0
ELSE	不變	0

如果 Change_mode == 0 就不變

```
else begin
    cnt4 = hour / 4'd10;
    cnt3 = hour % 4'd10;
    cnt2 = min2;
    cnt1 = min1;
end
```

然後我把顯示秒跟顯示 AM/PM 寫在 CONTROL 這裡

4’b1010 是 ‘A’; 4’b1011 是 ‘P’; 4’b1100 是 ‘m’

To_sec 是判斷要不要換乘顯示秒

在前兩位，如果 to_sec == 1，就換成秒

```
2'b00:
    begin
        ssd_ctl = 4'b0111;
        if (to_sec) ssd_in = sec2;
        else ssd_in = cnt4;
    end
2'b01:
    begin
        ssd_ctl = 4'b1011;
        if (to_sec) ssd_in = sec1;
        else ssd_in = cnt3;
    end
```

在後兩位比較複雜

第三位

```
2'b10:
    begin
        ssd_ctl = 4'b1101;
        if (change_mode & to_sec & ~PM) ssd_in = 4'b1010;
        else if (change_mode & to_sec & PM) ssd_in = 4'b1011;
        else if (~change_mode & to_sec) ssd_in = 4'b1111;
        else ssd_in = cnt2;
    end
```

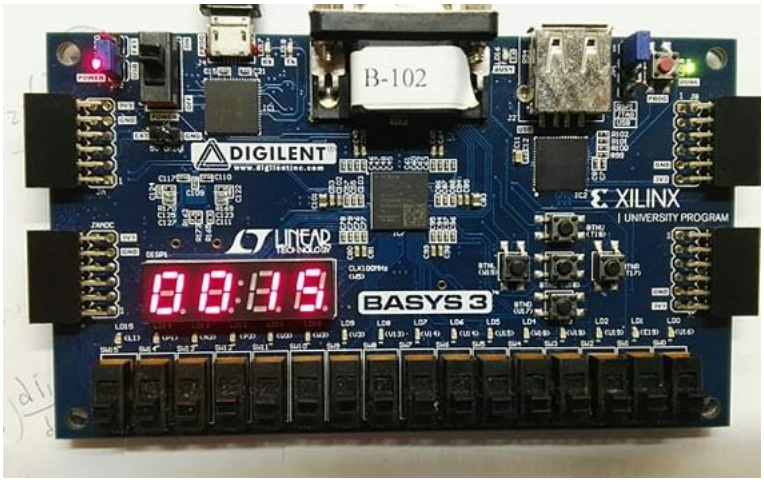
如果要換成 AM/PM 制，然後 PM == 1 就顯示 P，PM == 0 則顯示 A

那如果沒有要換，但 to_sec == 1，就顯示一個點

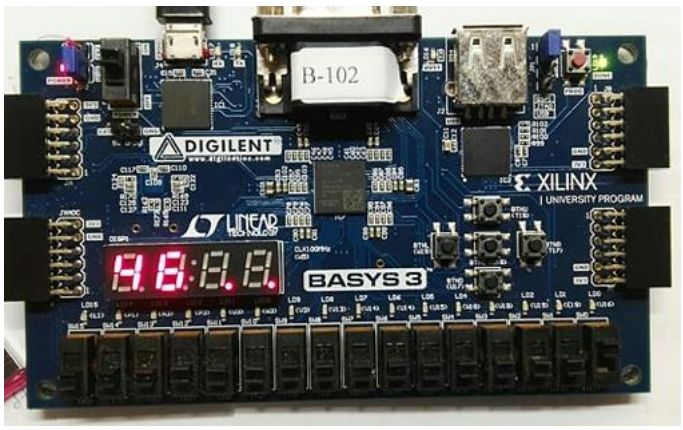
第四位跟第三位略同，附上 default

```
2'b11:
    begin
        ssd_ctl = 4'b1110;
        if (change_mode & to_sec) ssd_in = 4'b1100;
        else if (~change_mode & to_sec) ssd_in = 4'b1111;
        else ssd_in = cnt1;
    end
default:
    begin
        ssd_ctl = 4'b0000;
        ssd_in = 4'b0000;
```

24 小時制 時/分



秒



AM/PM 制 時/分

秒

Module display (decorder)

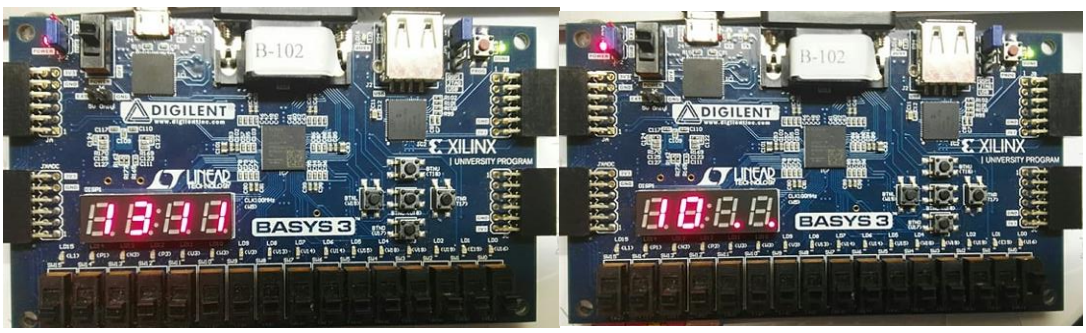
```
always @*
    case(in)
        4'd0: D_ssd = 8'b00000011;
        4'd1: D_ssd = 8'b10011111;
        4'd2: D_ssd = 8'b00100101;
        4'd3: D_ssd = 8'b00001101;
        4'd4: D_ssd = 8'b10011001;
        4'd5: D_ssd = 8'b01001001;
        4'd6: D_ssd = 8'b01000001;
        4'd7: D_ssd = 8'b00011111;
        4'd8: D_ssd = 8'b00000001;
        4'd9: D_ssd = 8'b00001001;
        4'd10: D_ssd = 8'b00010001; //A
        4'd11: D_ssd = 8'b00110001;  //P
        4'd12: D_ssd = 8'b11010101;  //m
        default: D_ssd = 8'b11111110;  //
endcase
```

DISSCUSSION

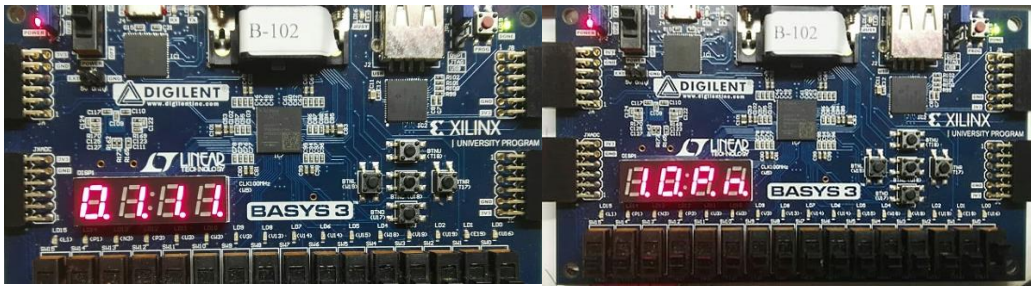
這次我原本全部用 BCD UP COUNTER 來做，24 小時也分個位和十位，過程中 DE 了很多 BUG，像是我把小時這樣寫{hour2, hour1}，以為這樣就可以一個十位，一個當個位，像是如果 hour2 = 1, hour1 = 2，我以為{hour2, hour1}是 12，但不是，因為 2 個都是 4bit，所以這樣寫是 8'b00010010，還有我把個位數的 limit 設 3，十位的設 2，想說他這樣數到 23 又會跳回 00，但這是有問題的，因為我數到 3 會直接跳 10，數到 13 會跳 20，所以才改成普通的 up counter，然後好險在 verilog 裡面 c 語言的除(/) 跟餘數 (%)在 verilog 可以用，不然我沒想到如何把一個二位數的十位跟個位拆開。

執行的結果就是秒數到 59 後回到 00，分加 1，分數到 59 後回到 00，小時加 1，小時數到 23 回到 00，然後按下 dip switch 從時/分換到秒，按下 change_mode 的 push button，就換成 AM/PM。

24hr 制



AM/PM 制



CONCLUSION

這次我又再次嘗試了老師在地 5 章範例的 FSM，但還是沒有成功，很失落，其他部分其實不難，幾乎都引用以前的，除了把 BCD_DOWN 改成 UP COUNTER 而已，原本用 BCD_UP 很麻煩，但換成 UP COUNTER 後就成功了

第二題

DESIGN SPECIFICATION

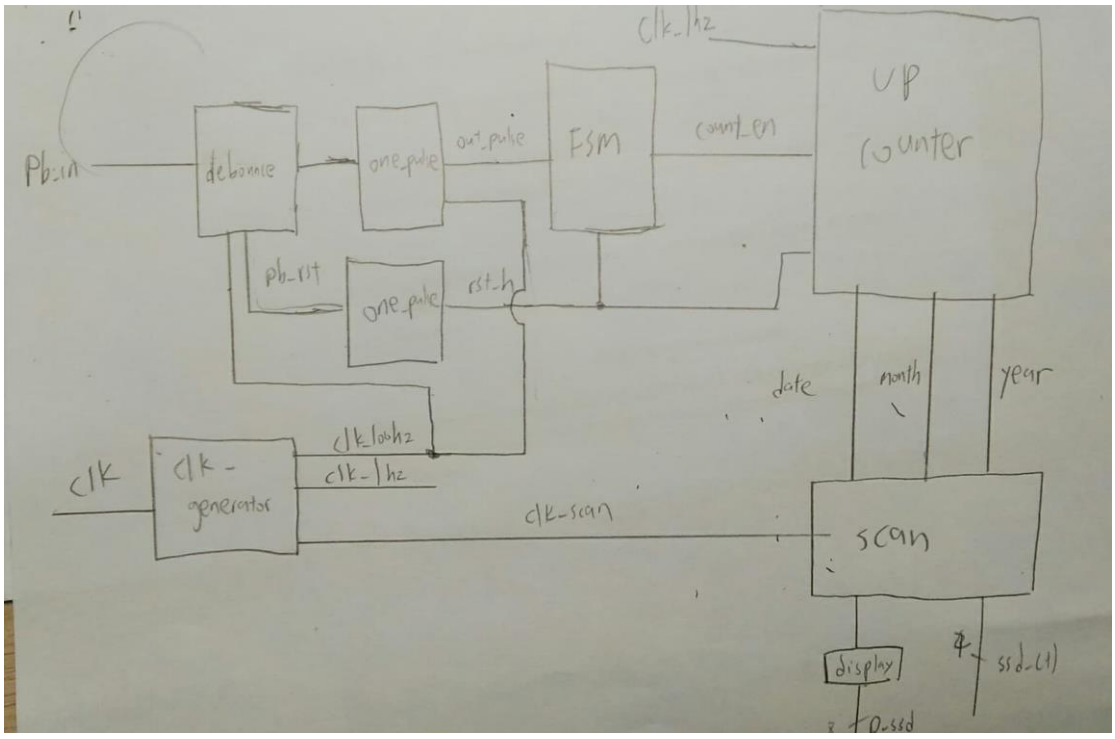
前言: 因為 Module clk_generator, debounce, one_pulse, one_pulse2, FSM(count, stop, rst), display 在每題都一樣，故不重複討論

(1) INPUT & OUTPUT

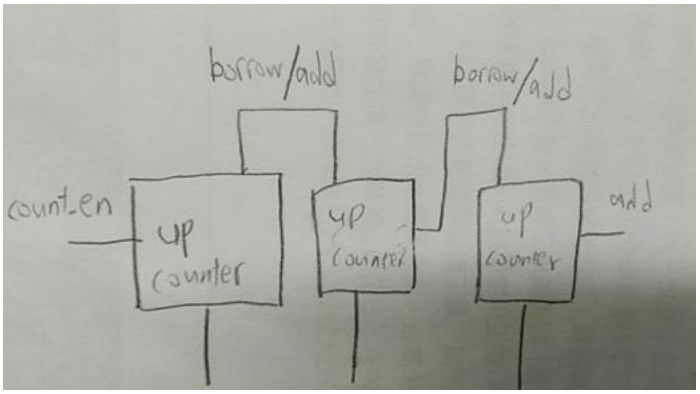
```
module top(clk, pb_in, change, ssd_ctl, D_ssd);
    input clk;
    input pb_in;
    input change;
    output [3:0]ssd_ctl;
    output [7:0]D_ssd;

module BCD_UP(limit, clk, rst_h, add, q, borrow, RETURN,
INITIAL);
     input INITIAL;
```

```
input [6:0]limit;
input clk;
input rst_h;
input add;
input [3:0]RETURN;
output reg [6:0]q;
output reg borrow;

module scan(ssd_ctl, ssd_in, date, month, year, control, change);
output reg [3:0] ssd_in;
output reg [3:0] ssd_ctl;
input change;
input [4:0]date;
input [3:0]month;
input [6:0]year;
input [1:0] control;
```

(2) Block diagram



Up counter 裡是一連串的 ripple up counter，我在程式裡把它命名為 BCD_UP




```
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));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h));
BCD_UP U44(.limit(5'd31), .clk(clk_1hz), .rst_h(rst_h), .add(count_en), .q(date), .borrow(borrow[0]), .RETURN(month), .INITIAL(1'b1))
BCD_UP U45(.limit(4'd12), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[0]), .q(month), .borrow(borrow[1]), .RETURN(4'd0), .INITIAL(1'b1)
BCD_UP U46(.limit(7'd99), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[1]), .q(year), .borrow(borrow[2]), .RETURN(4'd0), .INITIAL(1'b0))
scan V4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .date(date), .month(month), .year(year), .control(clk_scan), .change(change));
display UU(.D_ssd(D_ssd), .in(ssd_in));
```

DESIGN IMPLEMENTATION

(1) I / O PINS

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]
V17 [get_ports change]

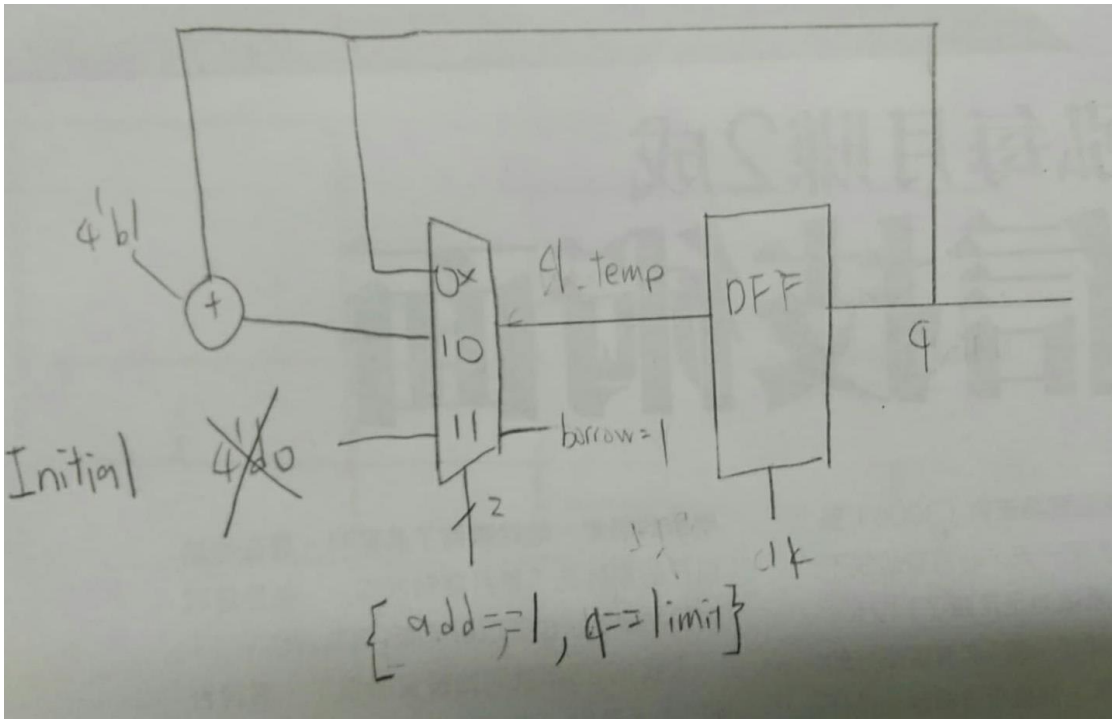
(2) Verilog code

Module top

按照 block diagram 接

```
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));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h));
BCD_UP U44(.limit(5'd31), .clk(clk_1hz), .rst_h(rst_h), .add(count_en), .q(date), .borrow(borrow[0]), .RETURN(month), .INITIAL(1'b1));
BCD_UP U45(.limit(4'd12), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[0]), .q(month), .borrow(borrow[1]), .RETURN(4'd0), .INITIAL(1'b1));
BCD_UP U46(.limit(7'd99), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[1]), .q(year), .borrow(borrow[2]), .RETURN(4'd0), .INITIAL(1'b0));
scan V4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .date(date), .month(month), .year(year), .control(clk_scan), .change(change));
display UU(.D_ssd(D_ssd), .in(ssd_in));
```

Module BCD_UP



```
always @*
    if (q == LIMIT && add == 1) begin
        q_temp = INITIAL;
        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 <= INITIAL;
    else q <= q_temp;
```

跟 6-1 不同的是，我把數到 limit 之後便會的值，跟一開始 rst_h 給定的初始值交由 INITIAL 控制，因應月份跟日期都是從 1 開始數，不是 0。

另外，我還新設了一個 RETURN，這個 RETURN 是把月份(month)回傳到日期(date)的 MODULE，去控制哪個月要幾天

```
BCD_UP U44(.limit(5'd31), .clk(clk_1hz), .rst_h(rst_h), .add(count_en), .q(date), .borrow(borrow[0]), .RETURN(month), .INITIAL(1'b1)
```

然後初始的上限是 31，在根據是幾月去判斷 limit 要減多少

```
always @*
    case(RETURN)
        4'd2: LIMIT = limit - 4'd3;
        4'd4: LIMIT = limit - 4'd1;
        4'd6: LIMIT = limit - 4'd1;
        4'd9: LIMIT = limit - 4'd1;
        4'd11: LIMIT = limit - 4'd1;
        default: LIMIT = limit;
    endcase
```

Module scan

這次的 scan 很簡單，引入 change 判斷是否要由年轉到月/日

```
always@*
    case(control)
        2'b00:
            begin
                ssd_ctl = 4'b0111;
                if (change) ssd_in = month / 4'd10;
                else ssd_in = 4'b0000;
            end
        2'b01:
            begin
                ssd_ctl = 4'b1011;
                if (change) ssd_in = month % 4'd10;
                else ssd_in = 4'b0000;
            end
        2'b10:
            begin
                ssd_ctl = 4'b1101;
                if (change) ssd_in = date / 4'd10;
                else ssd_in = year / 4'd10;
            end
        2'b11:
            begin
                ssd_ctl = 4'b1110;
                if (change) ssd_in = date % 4'd10;
                else ssd_in = year % 4'd10;
            end
        default:
            begin
                ssd_ctl = 4'b0000;
                ssd_in = 4'b0000;
            end
```

DISSCUSSION

這題其實比第一題容易，主要問題只有如何控制每月天數不同，還有比起使值設成一，依照蟻的經驗，遇到問題就多加變數，朝這個方向思考，很快就解決了，唯一遇到的問題只有一開始只有第一位的 0 跟 1 在跑，後來發現原來是我新設的變數 LIMIT 沒設成

8bit，我已經犯這種錯很多次了，以後會格外小心。

執行的結果就是數到月的極限之後，年就加一。

Conclusion

這次還頗順利，因為在做第一題實際有想過以前把值傳回去這個概念了。

第三題

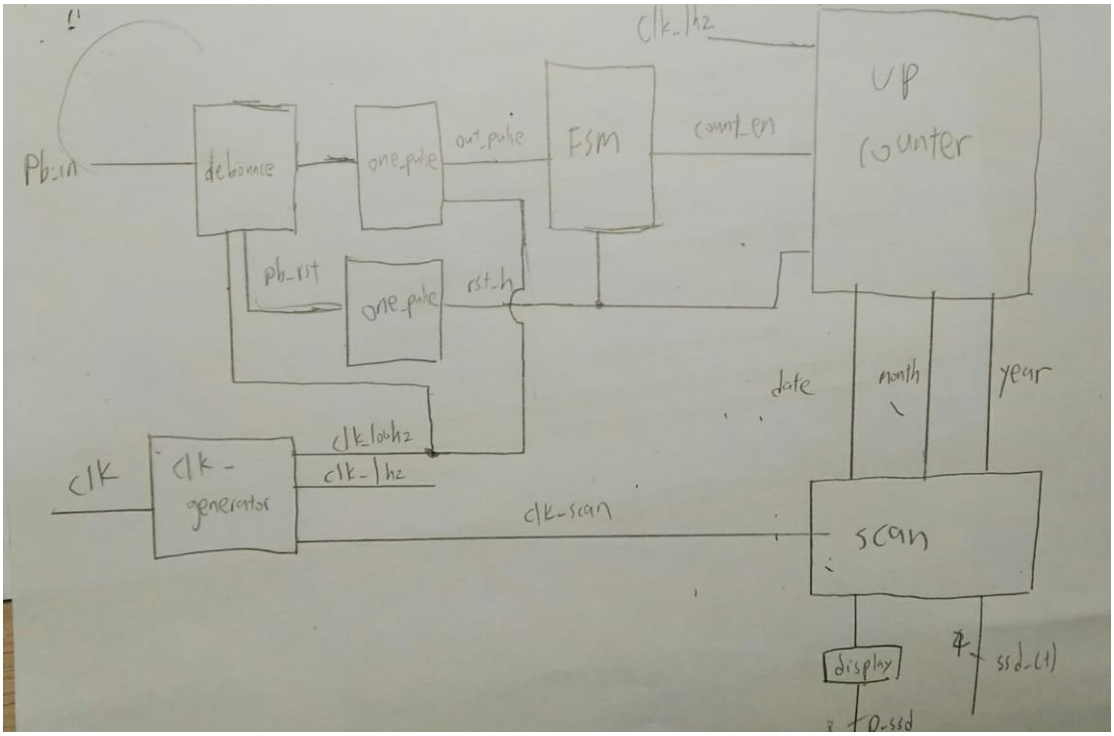
前言: 只討論不同的

DESIGN SPECIFICATION

(1) INPUT / OUTPUT

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

module BCD_UP2(limit, clk, rst_h, add, q, borrow, RETURN,
INITIAL, RETURN2);
    input INITIAL;
    input [7:0]limit;
    input clk;
    input rst_h;
    input add;
    input [3:0]RETURN;
    input [7:0]RETURN2;
    output reg [7:0]q;
    output reg borrow
```

(2) BLOCK DIAGRAM



Up COUNTER 裡是 8 個 ripple counter(with add and borrow)

```
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));
debounce U9(.clk(clk_100hz), .pb_in(pb_APM), .pb_debounced(pb_debounced_APM));
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));
one_pulse U53(.clk(clk), .in_trig(pb_debounced_APM), .out_pulse(out_pulse_APM));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h));
FSM_mode u11(.in(out_pulse_mode), .rst_h(rst_h), .state(mode));
FSM_APM u12(.in(out_pulse_APM), .rst_h(rst_h), .state(APM));
BCD_UP digit1(.limit(4'd9), .clk(clk_1hz), .rst_h(rst_h), .add(count_en), .q(sec1), .borrow(borrow[0]), .INITIAL(1'b0));
BCD_UP digit2(.limit(4'd5), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[0]), .q(sec2), .borrow(borrow[1]), .INITIAL(1'b0));
BCD_UP digit3(.limit(4'd9), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[1]), .q(min1), .borrow(borrow[2]), .INITIAL(1'b0));
BCD_UP digit4(.limit(4'd5), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[2]), .q(min2), .borrow(borrow[3]), .INITIAL(1'b0));
BCD_UP digit5(.limit(5'd23), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[3]), .q(hour), .borrow(borrow[4]), .INITIAL(1'b0));
BCD_UP2 digit6(.limit(5'd31), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[4]), .q(date), .borrow(borrow[5]), .INITIAL(1'b1), .RETURN(month), .RETURN2(year));
BCD_UP digit7(.limit(4'd12), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[5]), .q(month), .borrow(borrow[6]), .INITIAL(1'b1));
BCD_UP digit8(.limit(8'd200), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[6]), .q(year), .borrow(borrow[7]), .INITIAL(8'd0));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .min1(min1), .min2(min2), .hour(hour), .sec1(sec1), .sec2(sec2), .date(date), .month(month), .year(year),
.control(clk_scan), .mode(mode), .APM(APM), .PM(PM));
display U99(.D_ssd(D_ssd_temp), .in(ssd_in));

always@*
    if (PM) D_ssd = D_ssd_temp - 1'b1;
    else D_ssd = D_ssd_temp;
```

DESIGN IMPLEMENTATION

(1) I / O PINS

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]
T17 [get_ports pb_APM]

(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));
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));
debounce U9(.clk(clk_100hz), .pb_in(pb_APM), .pb_debounced(pb_debounced_APM));
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));
one_pulse U53(.clk(clk), .in_trig(pb_debounced_APM), .out_pulse(out_pulse_APM));
FSM U5(.count_en(count_en), .in(out_pulse), .rst_h(rst_h));
FSM_mode u11(.in(out_pulse_mode), .rst_h(rst_h), .state(mode));
FSM_APM u12(.in(out_pulse_APM), .rst_h(rst_h), .state(APM));
BCD_UP digit1(.limit(4'd9), .clk(clk_1hz), .rst_h(rst_h), .add(count_en), .q(sec1), .borrow(borrow[0]), .INITIAL(1'b0));
BCD_UP digit2(.limit(4'd5), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[0]), .q(sec2), .borrow(borrow[1]), .INITIAL(1'b0));
BCD_UP digit3(.limit(4'd9), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[1]), .q(min1), .borrow(borrow[2]), .INITIAL(1'b0));
BCD_UP digit4(.limit(4'd5), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[2]), .q(min2), .borrow(borrow[3]), .INITIAL(1'b0));
BCD_UP digit5(.limit(5'd23), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[3]), .q(hour), .borrow(borrow[4]), .INITIAL(1'b0));
BCD_UP2 digit6(.limit(5'd31), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[4]), .q(date), .borrow(borrow[5]), .INITIAL(1'b1), .RETURN(month), .RETURN2(year));
BCD_UP digit7(.limit(4'd12), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[5]), .q(month), .borrow(borrow[6]), .INITIAL(1'b1));
BCD_UP digit8(.limit(8'd200), .clk(clk_1hz), .rst_h(rst_h), .add(borrow[6]), .q(year), .borrow(borrow[7]), .INITIAL(8'd0));
scan U4(.ssd_ctl(ssd_ctl), .ssd_in(ssd_in), .min1(min1), .min2(min2), .hour(hour), .sec1(sec1), .sec2(sec2), .date(date), .month(month), .year(year),
.control(clk_scan), .mode(mode), .APM(APM), .PM(PM));
display U99(.D_ssd(D_ssd_temp), .in(ssd_in));

always@*
    if (PM) D_ssd = D_ssd_temp - 1'b1;
    else D_ssd = D_ssd_temp;
```

MODULE BCD_UP2

```
always @*
    case(RETURN)
        4'd2: begin
            if (RETURN2 == 8'd0) LIMIT = limit - 4'd2;
            else if (RETURN2 % 3'd4 == 0 && RETURN2 % 7'd100 != 1'd0) LIMIT = limit - 4'd2;
            else LIMIT = limit - 4'd3;
        end
        4'd4: LIMIT = limit - 4'd1;
        4'd6: LIMIT = limit - 4'd1;
        4'd9: LIMIT = limit - 4'd1;
        4'd11: LIMIT = limit - 4'd1;
        default: LIMIT = limit;
    endcase
```

在日期(date)裡的 counter 跟別人不一樣，她會多了一個 return 跟 restur2，分別代表月和年，然後因為此次只有 2000~2200，除了 2000 年之外沒有 400 年的問題，所以我就只打了如果年是 00，那就 29 天，然後 4 的倍數但不是 100 的倍數也是 29 天。

RETURN 值的概念與 6-2 一樣

DISSCUSSION

這題只是沿用第二題 RETURN 的概念而已，打起來無大礙。

結果就是在 2000 年時 2/29

2004 年 2/29

2100 年 2/28

CONCLUSION

這題不難，但是很大，接了 8 個 COUNTER，變數也很多，真的很容易不小心出錯，，過程中 DE 了很多小 BUG，但自己實際完成一個日曆也是很有成就感。