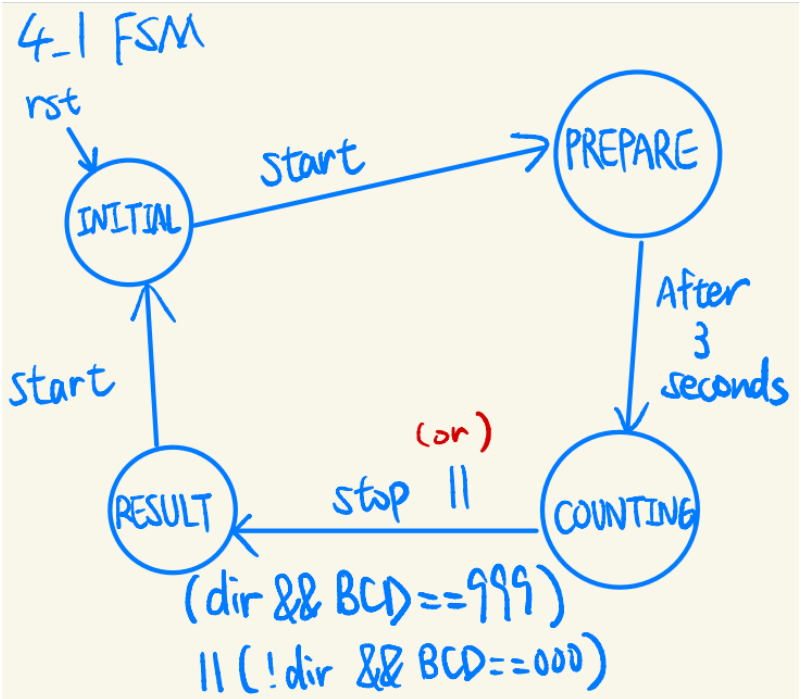
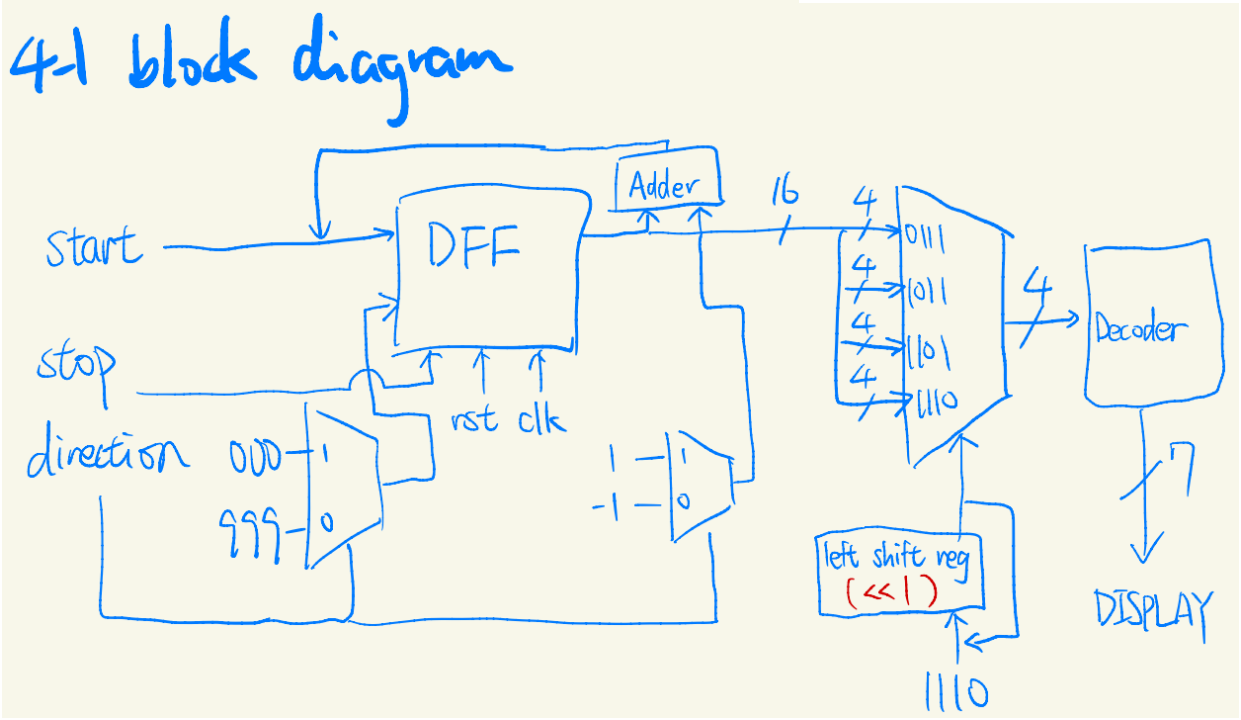
Lab 4

學號: 109021115 姓名: 吳嘉濬

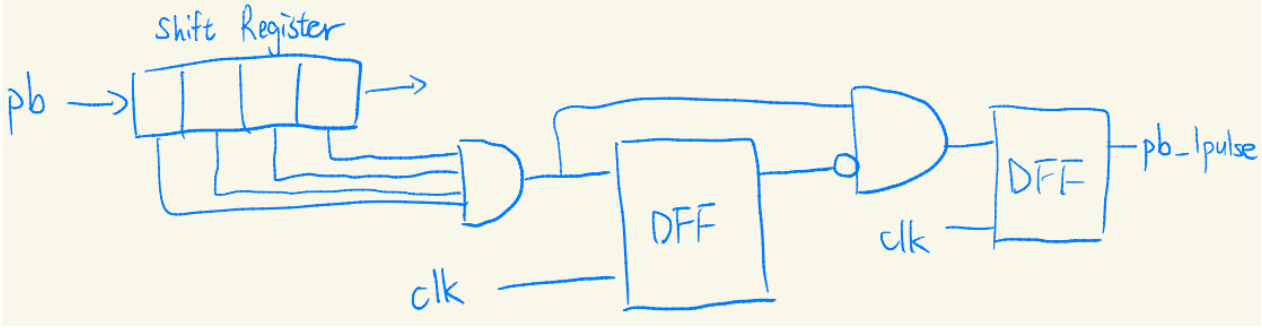
A. Lab Implementation

以下是 lab4_1 的 FSM 和 block diagram：





當 rst==1'b1 時，進入 INITIAL state。按下 start button 時，進入 PREPARE state。這裡要特別留意，因為 press button 時，從低電位到穩定的高電位過程之中會有一段不穩定的訊號，我們可以用 debounce 來把它過濾掉；另外，因為我們一般人在按一次按鈕的時長對高頻率的電路而言其實經過了很多 clock cycle，所以要用 one-pulse 方法來讓訊號只有被 trigger 剛好一個 clock cycle，以下為 debounce 和 one-pulse 的電路：



Debounce 過濾掉沒有連續 4 個 cycle 都接收到高電位的訊號；one-pulse 讓原本的訊號和 delay 過一個 clock cycle 的訊號的相反取交集，如此，可以讓連續的高電位訊號經處理後只剩下第一個 cycle 的高電位訊號還在。

接著我們利用除頻器，做出頻率為 100Hz 的 clock，實作方法如下：

```
module clock_divider(clk,clk_div);
    input clk;
    output clk_div;
    parameter n = 500000;
    reg[29:0]num;
    reg tick=1'b1;
    always @(posedge clk) begin
        if (num == n-1) begin
            num <= 30'b0;
            tick <= ~tick;
        end else begin
            num <= num + 1'b1;
            tick <= tick;
        end
    end

    assign clk_div = tick;

endmodule
```

利用 counter，每一個 clk 的 posegde 會使 num 加一，一直從 0 加到 499999，也就是經過 500000 個 clk 頻率的 clock cycle，就讓 tick 從 0 變成 1 或從 1 變成 0，之後在經過 500000 個 clk 頻率的 clock cycle，再 tick 讓轉變一次。tick 轉變兩次就是 clk_div 一次的 clock period，總共經過 1000000 個 clk 頻率的 clock cycle，相當於 clk_div 的頻率是 clk 的一百萬分之一，而 clk 的頻率是 100Mhz，所以 clk_div 頻率是 100Hz。

如此，我們可以設計出一個每 0.01 秒加一的 counter，進而實作出”進入 PREPARE state 之後，經過 3 秒後進入 COUNTING state”以及”每經過一秒讓 led 燈亮暗亮暗交替”等等的功能。

```
if(offset_cnt==20'd99 || offset_cnt==20'd199 || offset_cnt==20'd299 || offset_cnt==20'd399) next_led=~led;
```

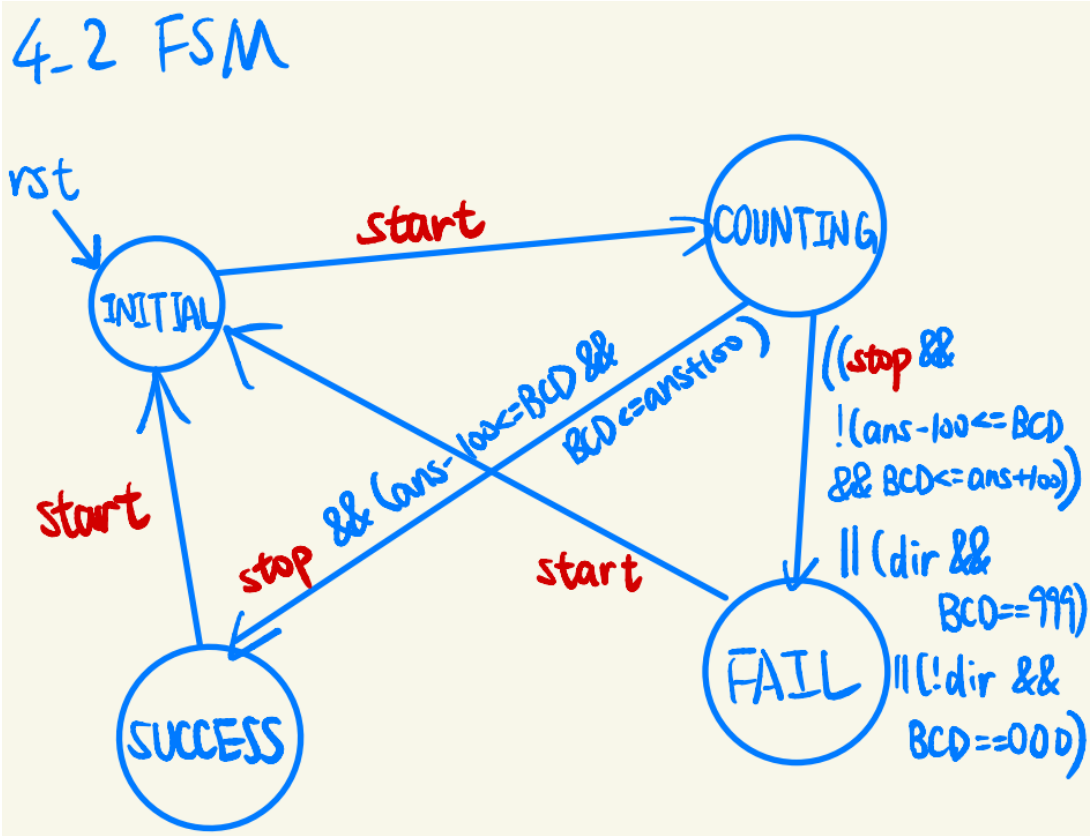
關於 7-seg 的實作原理，我以 2500Hz 的頻率去持續切換每個 cycle 要亮的 7-seg led 燈，在這樣的頻率之下，人類肉眼感受不到 led 燈的交替閃爍感，實作的 kernel code 如下：

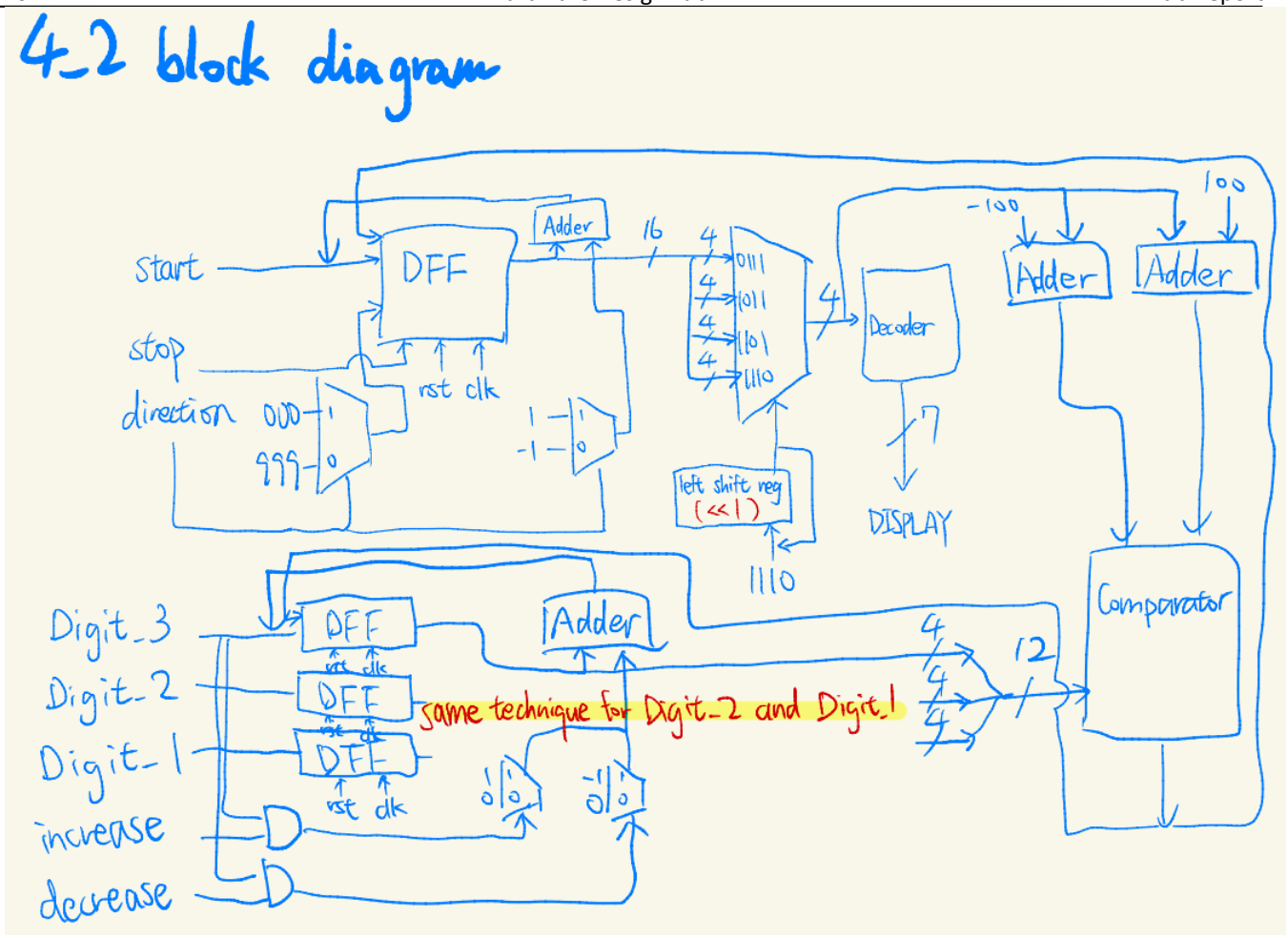
```
always@(posedge clk_div1) begin
    case(DIGIT)
    4'b1110: begin
        value=BCD1;
        DIGIT=4'b1101;
    end
    4'b1101: begin
        value=BCD2;
        DIGIT=4'b1011;
    end
    4'b1011: begin
        value=BCD3;
        DIGIT=4'b0111;
    end
    4'b0111: begin
        value=BCD0;
        DIGIT=4'b1110;
    end
    default: begin
        value=BCD0;
        DIGIT=4'b1110;
    end
    endcase
end
```

```
always@* begin
    case(value)  //GFE_DCBA
        4'd0: DISPLAY=7'b100_0000;
        4'd1: DISPLAY=7'b111_1001;
        4'd2: DISPLAY=7'b010_0100;
        4'd3: DISPLAY=7'b011_0000;
        4'd4: DISPLAY=7'b001_1001;
        4'd5: DISPLAY=7'b001_0010;
        4'd6: DISPLAY=7'b000_0010;
        4'd7: DISPLAY=7'b111_1000;
        4'd8: DISPLAY=7'b000_0000;
        4'd9: DISPLAY=7'b001_0000;
        4'd10: DISPLAY=7'b101_1100;  //up
        4'd11: DISPLAY=7'b110_0011;  //down
        4'd12: DISPLAY=7'b011_1111;  //dash
        4'd13: DISPLAY=7'b000_1100;  //"P"
        4'd14: DISPLAY=7'b111_1111;  //none
        default: DISPLAY=7'b111_1111;
    endcase
end
```

值得注意的是，因為實際電路設計的關係，7-seg 是 negative enable，即 bit 0 才會使對應的 led 燈亮，不是一般直覺的 bit 1。

以下是 lab4_2 的 FSM 和 block diagram：





基本上實作的原理和 lab4_1 差不多，我針對 lab4_1 沒有出現的內容作說明。

第一，如何實作在 COUNTINTG state3 秒鐘之後讓 7-seg 的顯示從 counting 的數值轉變成”-“。

```
always@(posedge clk_div1) begin
    case(DIGIT)
    4'b1110: begin
        value = (show) ? BCD1 : 4'd12;  //dash
        DIGIT=4'b1101;
    end
    4'b1101: begin
        value = (show) ? BCD2 : 4'd12;
        DIGIT=4'b1011;
    end
    4'b1011: begin
        value = (show) ? BCD3 : BCD3;  //doesn't hide
        DIGIT=4'b0111;
    end
    4'b0111: begin
        value = (show) ? BCD0 : 4'd12;
        DIGIT=4'b1110;
    end
    default: begin
        value = (show) ? BCD0 : 4'd12;
        DIGIT=4'b1110;
    end
    endcase
end
```

```
if(offset_cnt==20'd299) begin
    next_show=1'b0;
    next_led=16'b0000000000000000;
end
```

我的作法是額外設計一個 signal 並命名為 show。在利用 counter 判定進入 COUNTING state3 秒鐘之後，讓 show 從 1’b1 變成 1’b0，如此，配合”xx?xx:xx;“的語法可以選擇想要在 7-seg 顯示的內容。

第二，如何存取使用者設定的欲猜測秒數。

```
reg [3:0] ans[2:0], next_ans[2:0];
```

我的作法很簡單，就是額外設計 6 個 4-bit 大小 register 去儲存和更新欲猜測秒數。

```
if(inc_one) begin
    if(Digit_1) next_BCD0 = (BCD0==4'd9) ? 4'd0 : BCD0 + 1'b1;
    if(Digit_2) next_BCD1 = (BCD1==4'd9) ? 4'd0 : BCD1 + 1'b1;
    if(Digit_3) next_BCD2 = (BCD2==4'd9) ? 4'd0 : BCD2 + 1'b1;
end
if(dec_one) begin
    if(Digit_1) next_BCD0 = (BCD0==4'd0) ? 4'd9 : BCD0 - 1'b1;
    if(Digit_2) next_BCD1 = (BCD1==4'd0) ? 4'd9 : BCD1 - 1'b1;
    if(Digit_3) next_BCD2 = (BCD2==4'd0) ? 4'd9 : BCD2 - 1'b1;
```

設計可以調整百位數、十位數和個位數欲猜測秒數的機制。

```
if(start_one) begin
    next_state=COUNTING;
    next_led=16'b1111111111111111;
    next_offset_cnt = 20'd0;
    next_ans[2]=BCD2;
    next_ans[1]=BCD1;
    next_ans[0]=BCD0;
```

並在按下 start button 進入 COUNTING state 前，讓欲猜測秒數存進 register ans。

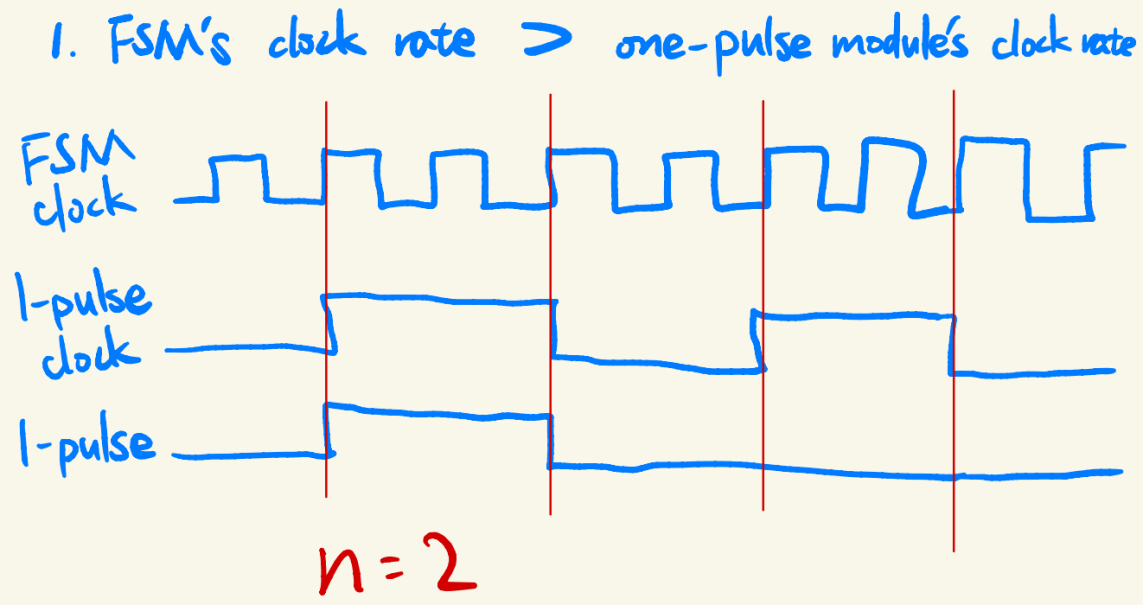
利用 lab4_1 的實作原理加上以上兩點重要設計觀念，能夠順利成功完成 lab4_2。

B. Questions and Discussions

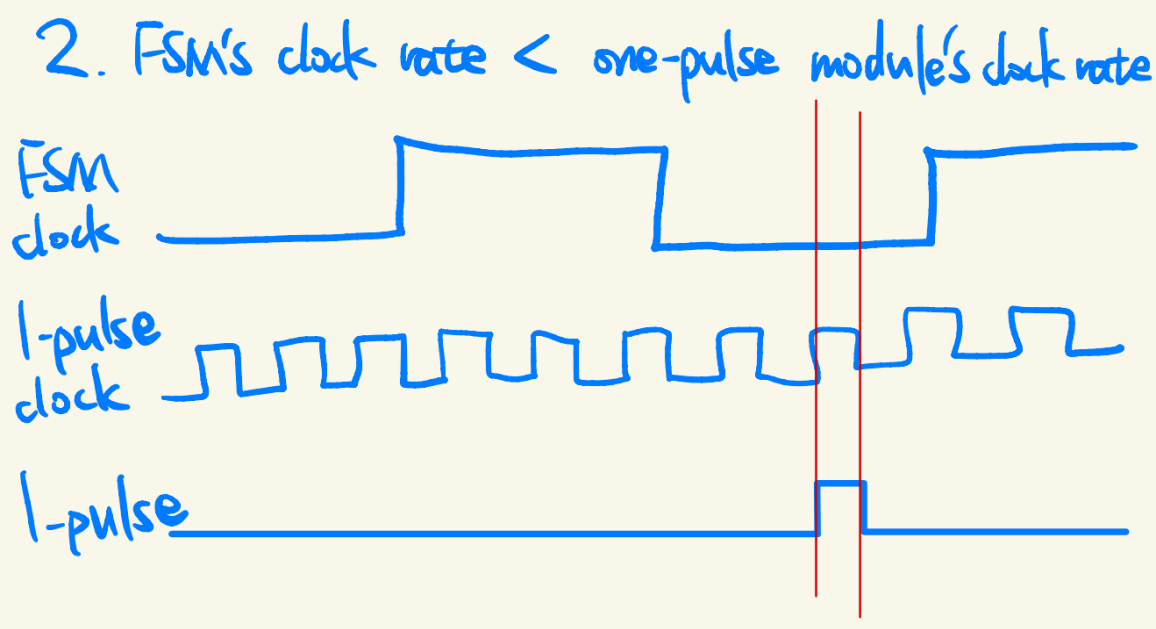
A. Why do we need the debounce and one-pulse modules? What is the relation between the FSM's clock rate and the one-pulse module's clock rate? Explain your reason. And what will happen when we implement with a wrong clock rate? (Please draw waveforms to explain)

使用者在按按鈕時，訊號從低電位到高電位的過渡過程當中，會產生許多不穩定、忽高忽低的訊號，此時如果沒有做過濾的話，除了可能會讓訊號的高電位被多次讀取之外，每一次得到的訊號也都會不一，所以 debounce module 幫助我們在按按鈕時能過濾掉前期不穩定的訊號。使用者在按按鈕時，對使用者而言雖然可能只是一瞬間的事情，但對高頻率的電路而言，其實已經產生超過一個 clock 周期的連續高電位訊號，而為了避免持續許多週期的高電位訊號被重複讀取(被判定成按許多次按鈕)，我們使用 one-pulse module 以確保每次使用者在按一次的按鈕時，只產生剛好一個 clock cycle 的高電位訊號。

FSM's clock rate 需要和 one-pulse module's clock rate 一致，如此一來，按按鈕所產生的 one clock cycle 訊號才會被 FSM 讀取到且剛好作用一次。如果 FSM's clock rate 大於 one-pulse module's clock rate，因為按按鈕產生一個 clock cycle 的訊號是 depend on one-pulse module's clock rate，所以如果 FSM's clock rate 是 one-pulse module's clock rate 的 n 倍，這樣 FSM 會認為他連續接收到 n 個 clock cycle 的高電位訊號，進而作用 n 次。waveform 示意圖如下：



而如果 FSM's clock rate 小於 one-pulse module's clock rate，因為 FSM 是在每個 clock cycle 的 posedge 時更新狀態，所以有可能 one-pulse module 產生一個 clock cycle 的高電位訊號時，FSM 沒有經歷到它自己 clock 的 posedge，導致按按鈕的 one-pulse 訊號沒有被 FSM 判斷到。waveform 示意圖如下：



B. Please propose two different design methods to manipulate two or more clocks in one module.

第一個方法，也就是我在實作這次 lab 時使用的方法，如下圖所示：

```
module clock_divider(clk,clk_div);
    input clk;
    output clk_div;
    parameter n = 500000;
    reg[29:0]num;
    reg tick=1'b1;
    always @(posedge clk) begin
        if (num == n-1) begin
            num <= 30'b0;
            tick <= ~tick;
        end else begin
            num <= num + 1'b1;
            tick <= tick;
        end
    end

    assign clk_div = tick;

endmodule
```

```
clock_divider #(20000) div1(.clk(clk),.clk_div(clk_div1));
clock_divider div2(.clk(clk),.clk_div(clk_div2));
```

我的做法是設計一個頻率為原本頻率除以 2n 的除頻器 module，且讓 n 為 parameter，方便我們在 instantiate 兩個不同頻率的 clock 時，可以利用#(xxxx)的語法去設定不同頻率的除頻器。另外，我讓 n 預設為 500000，產生頻率為 100Hz 的 clock(週期為 0.01 秒)。

第二個方法是 instantiate 完一個 clock 之後，利用 always block 設計一個 counter，當它從 0 count 到 m-1 時，clk_div 從 0 變成 1 或從 1 變成 0，如此一來就能設計頻率為第一個 clock 除以 2m 的除頻器。

C. Problem Encountered

起初在設計除頻器時，我把 FSM 的 clock 頻率設為原本未除頻過的 clk，結果發現 bug 百出，像是按按鈕有時有反應有時沒反應，7-seg 出現閃爍、每個 digit 亮度不一或是 digit 的 led 顯示殘影跑到隔壁 digit 的問題。

之後看 spec 時發現 B.A 的問題討論，於是往這個方向去思考，覺得可能就是因為 FSM 和 digit 切換的頻率出錯才導致出現這些問題。最後，我把 FSM 的 clock 頻率改成和 one-pulse module 的 clock 頻率一樣，並且調整 digit 切換的頻率之後順利地把問題都解決了。

以下為相對應的 code segment：

```
always@(posedge clk_div2 or posedge rst)begin
    if(rst) begin
        state<=INITIAL;
        dir<=1'b1;  //up
        led<=16'b1111111111111111;
        ans[2]<=4'd0;
        ans[1]<=4'd0;
        ans[0]<=4'd0;
    end else begin
        state<=next_state;
        dir<=next_dir;
        led<=next_led;
        ans[2]<=next_ans[2];
        ans[1]<=next_ans[1];
        ans[0]<=next_ans[0];
    end
```

```
always@(posedge clk_div1) begin
    case(DIGIT)
    4'b1110: begin
        value = (show) ? BCD1 : 4'd12;  //dash
        DIGIT=4'b1101;
    end
    4'b1101: begin
        value = (show) ? BCD2 : 4'd12;
        DIGIT=4'b1011;
    end
    4'b1011: begin
        value = (show) ? BCD3 : BCD3;  //doesn't hide
        DIGIT=4'b0111;
    end
    4'b0111: begin
        value = (show) ? BCD0 : 4'd12;
        DIGIT=4'b1110;
    end
    default: begin
        value = (show) ? BCD0 : 4'd12;
        DIGIT=4'b1110;
    end
    endcase
end
```

其中 clk_div1 頻率為 2500Hz；clk_div2 頻率為 100Hz。

D. Suggestions

這次的 lab 還是和以往一樣很 time consuming，但不同的是老師這次有在課堂中講解 lab，幫助我們快速了解 lab 要我們實作什麼，進而增加我們自己看 spec 時的效率，希望以後也能繼續在課堂中講解 lab。