LAB5

第七組

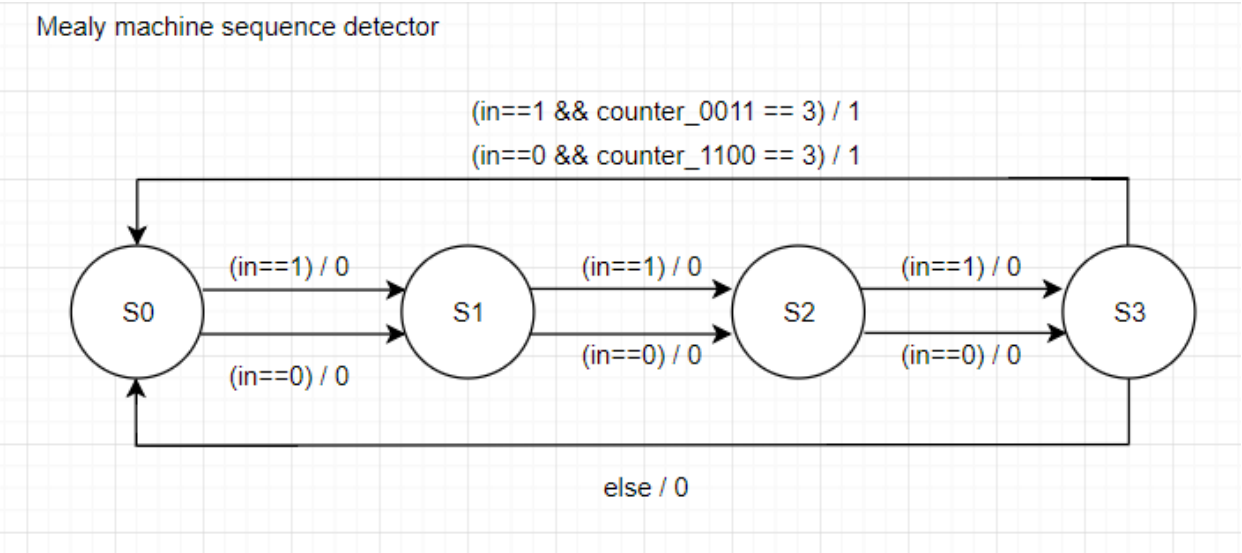
黎佑廷 105030009

郭家瑋 105030015

VERILOG QUESTION 1:

MEALY MACHINE SEQUENCE DETECTOR

✓ State-transition diagram:



當 state == S0 時，不管怎樣都是輸出 0，而且下一個 state 一定是 S1，只是當 in == 1 時，我將 counter_1100 + 1，當 in==0 時，我將 counter_0011 + 1。

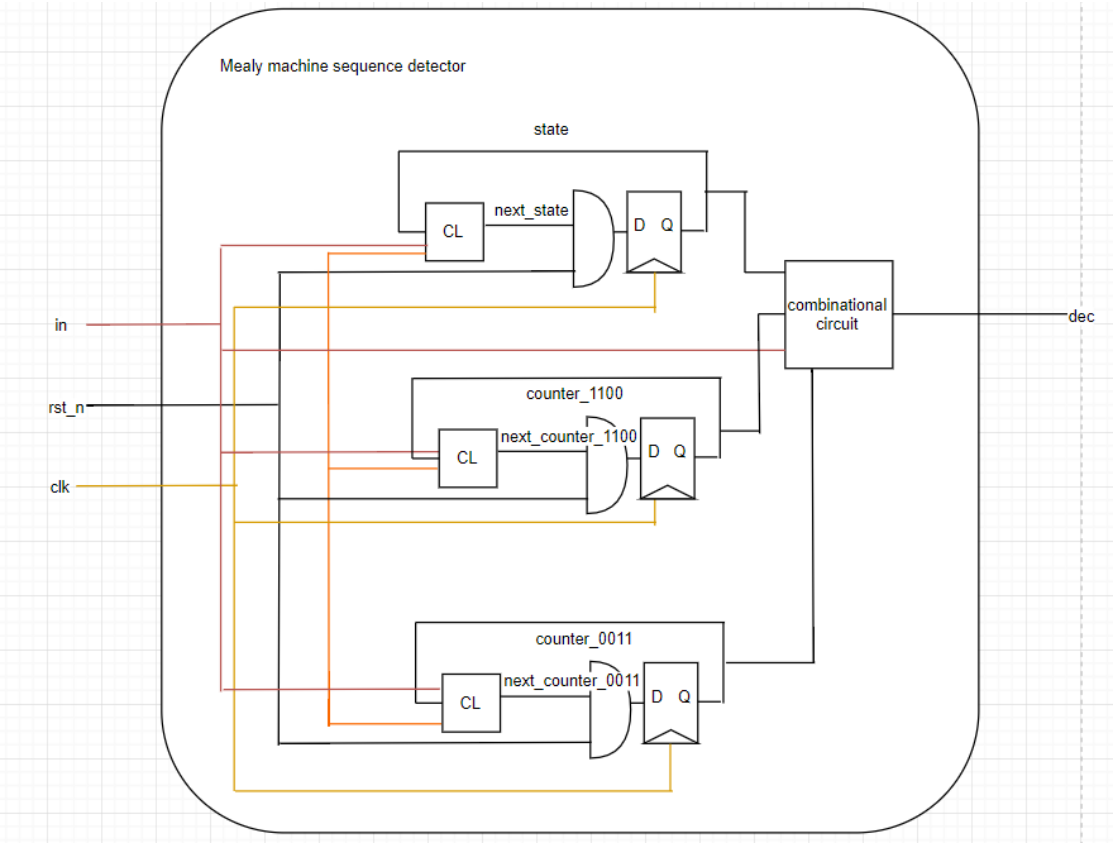
當 state == S1 時，也是不管怎樣都輸出 0，而且下一個 state 一定是 S2，只是當 in == 1 時，我將 counter_1100 + 1，當 in == 0 時，我將 counter_0011 + 1 。

當 state == S2 時，也是不管怎樣都輸出 0，而且下一個 state 一定是 S3，只是當 in == 0 時，我將 counter_1100 + 1 ，當 in == 1 時，我將 counter_0011 + 1。

當 state == S3 時，如果 in == 1 而且 counter_0011 == 3，代表已經偵測到 0011 這個 pattern，所以輸出 1，如果 in==0 而且 counter_1100 == 3 代表已經偵測到 1100 這個 pattern，其他的狀況則是輸出 0，下一個 state 回到 S0 以繼續偵測接下來的 pattern。

因為無論 input 為何，state 的變換都是 S0->S1->S2->S3 所以每四個 bit 都會重新 detect 一次。

✓ Block Diagram



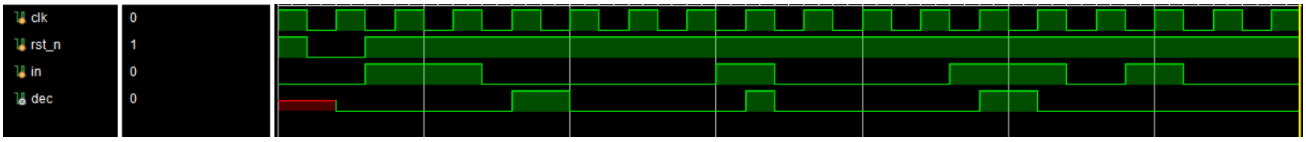
這題我所使用的驗證方法是給予跟老師投影片一樣的 input，看看出來的 waveform 有沒有跟投影片上的一樣。

Code 的部分如下(給予 1001_0010_1001_0100 的 input):

```
initial begin
  @ (negedge clk) rst_n = 1'b0;
  @ (posedge clk)
  @ (negedge clk)
    rst_n = 1'b1;
    in = 1'b1;

  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b1;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) in = 1'b0;
  @ (negedge clk) $finish;
end
```

✓ Waveform



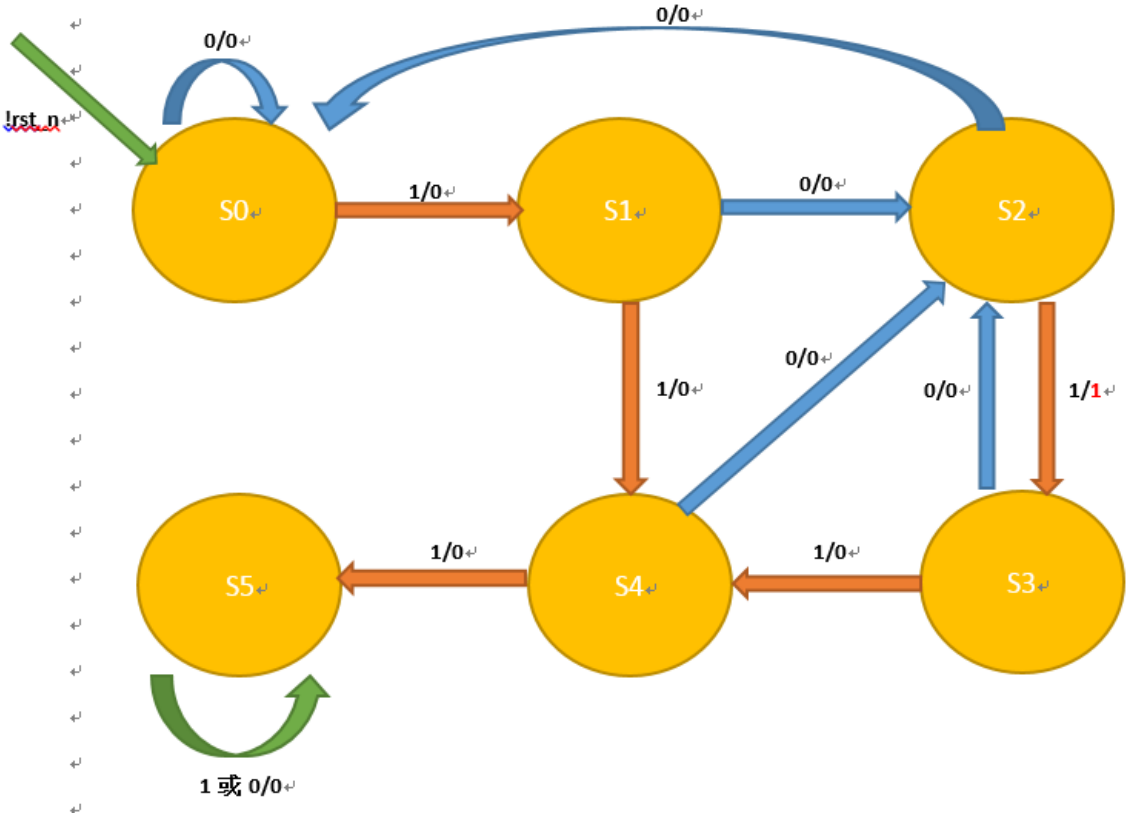
可以發現這個 waveform 跟老師投影片的一模一樣

VERILOG QUESTION 2

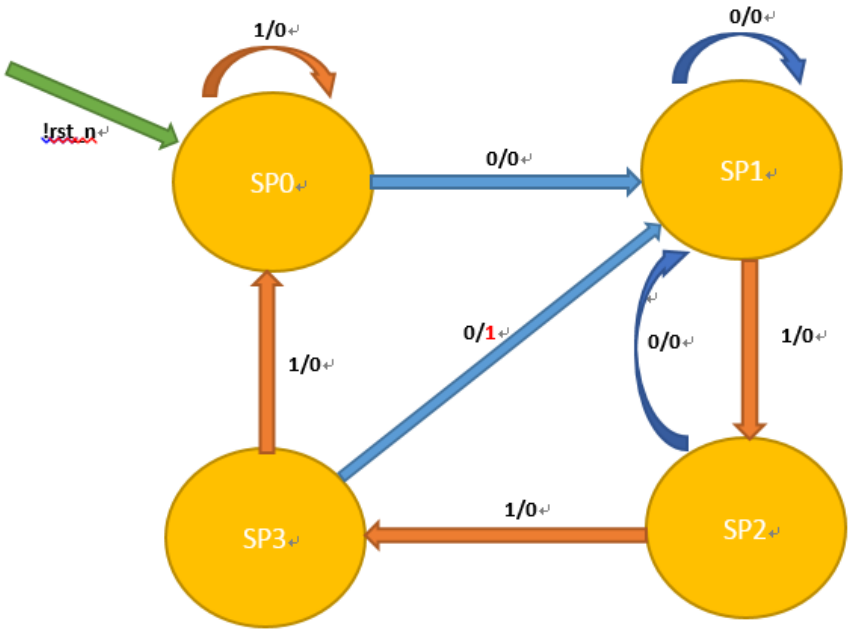
TRAFFIC LIGHT CONTROLLER

✓ State Transition Diagram

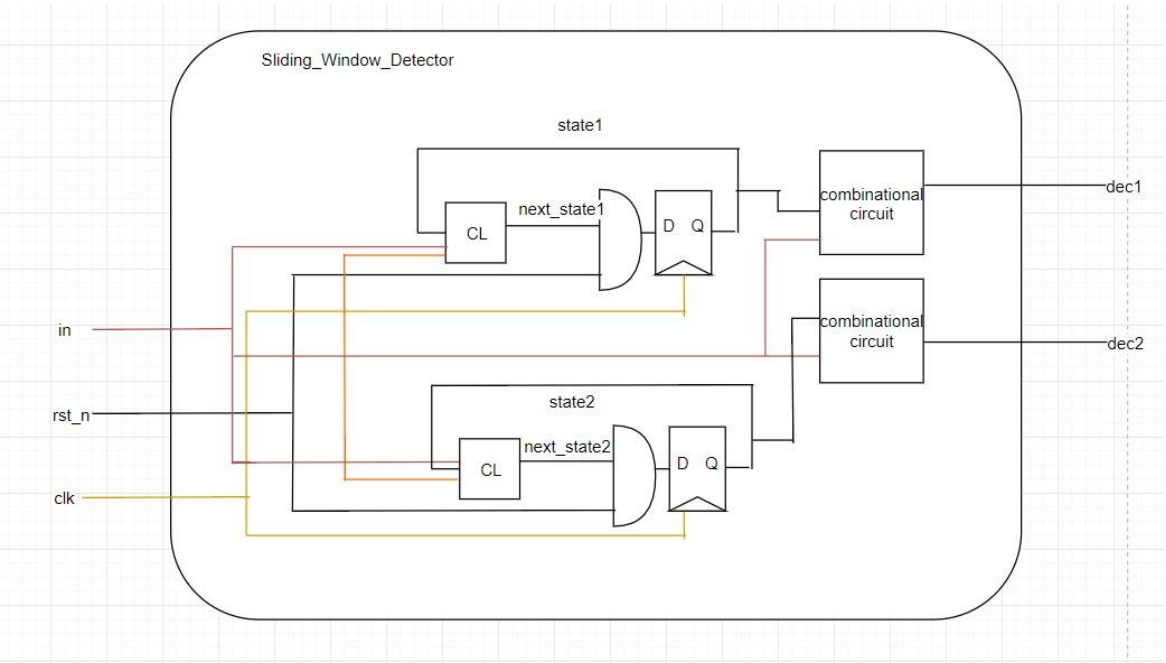
For Dec1



For Dec2



✓ Block_diagram



Dec1：

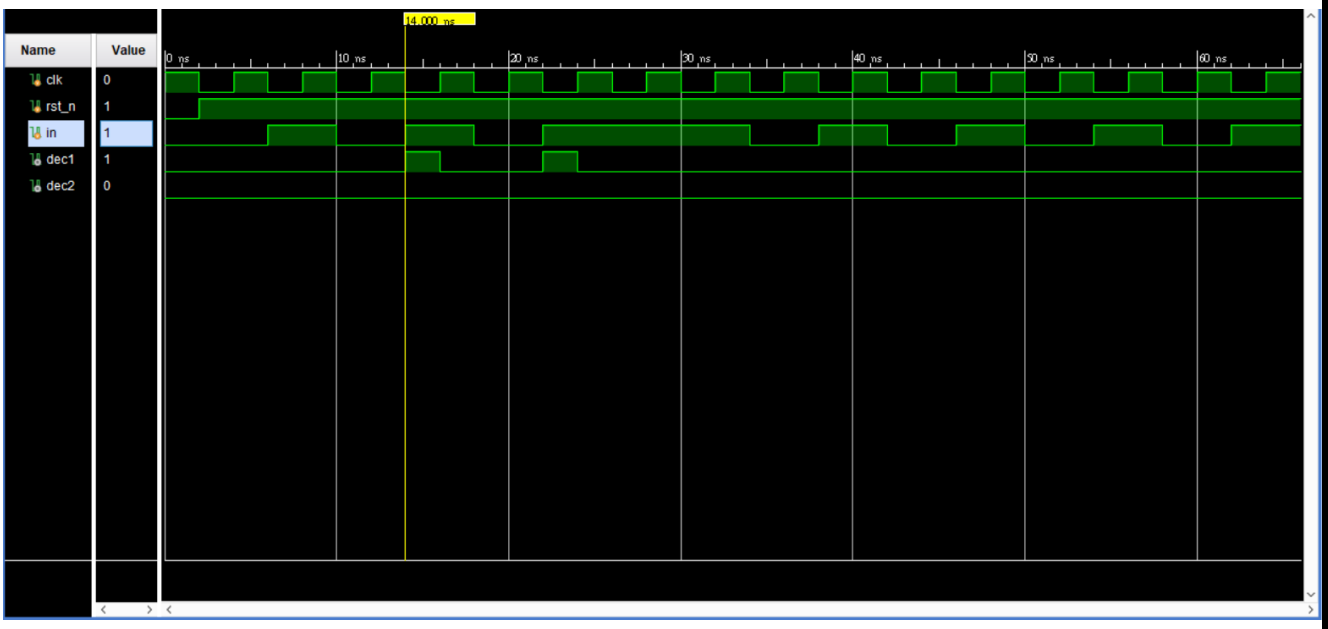
當接收到!RESET 訊號會進入 S0，此時要等待一個 1 才能繼續往下一個 state，因此若 in 是 0，next_state 會是 S0，若 in 是 1，next_state 會是 S1。到達 S1 後，此時等待一個 0 才能往下一個 state 移動，因此若 in 是 1，next_state 會進入 S4，若 in 是 0，next_state 會是 S2。到達 S2 後，此時等待一個 1 即可完成一個 101 的訊號檢測並且 output 1，因此若 in 為 1，next_state 會是 S3，並且 output 1，若 in 為 0，等於出現了「100」的訊號，此時要重新回到 S0 等待一個 1。到達 S3 後，表示剛剛出現了一個「101」的訊號，因為訊號可以接連著上一次的 pattern 做檢測，例如「10101」會有兩次的訊號 pattern 檢測出來。因此，在 S3，若 in 為 0，要回到 S2，因為到達 S3 代表剛剛出現過一個 in=1，因此只需再等待「01」的出現即可 output=1，若 in 為 1，next_state 為 S4。到達 S4 後，代表連續出現了兩個 1，此時要判斷 in 是否為 1，若 in 為 1，代表連續出現了三個 1，進入 S5 此後不管出現什麼訊號的 pattern 都要 output 0，若是 in=0，代表出現了「110」，此時只需再出現一個 1 即可 output1，因此回到 S2。

Dec2：

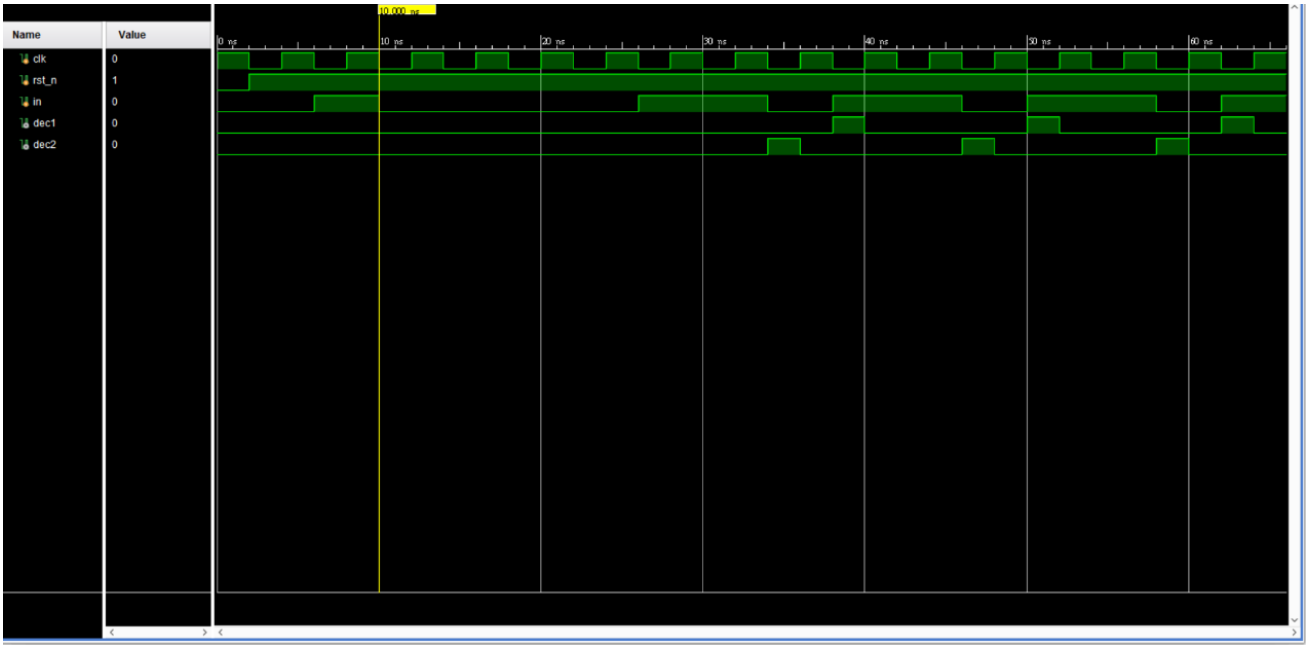
當接收到!RESET 訊號會進入 SP0，此時要等待一個 0 才能繼續往下一個 state，因此若 in=0，next_state 會進入 SP1，若 in=1，next_state 繼續在 SP0。到達 SP1 後，此時需要一個 1 才能進入下一個 state，因此若 in=1，next_state 會是 SP2，若 in=0，next_state 會是 SP1。到達 SP2 後，代表已有「01」，因此，此時需要 1 才能進入下一 state，因此若 in=1，next_state 是 SP3，若 in=0，next_state 會是 SP1，要重新等待「0110」的第一個 1。到達 SP3 後，代表已有「011」，此時只需一個 0 就可 output 1，因此若 in=0，next_state 會是 SP1，已有一個 0 的情況，並且 output 1，若 in=1，要重新等待一個新的 0。

Testbench 寫法是模仿投影片中的 input 並且比較標準答案與我的波形圖是否相等。

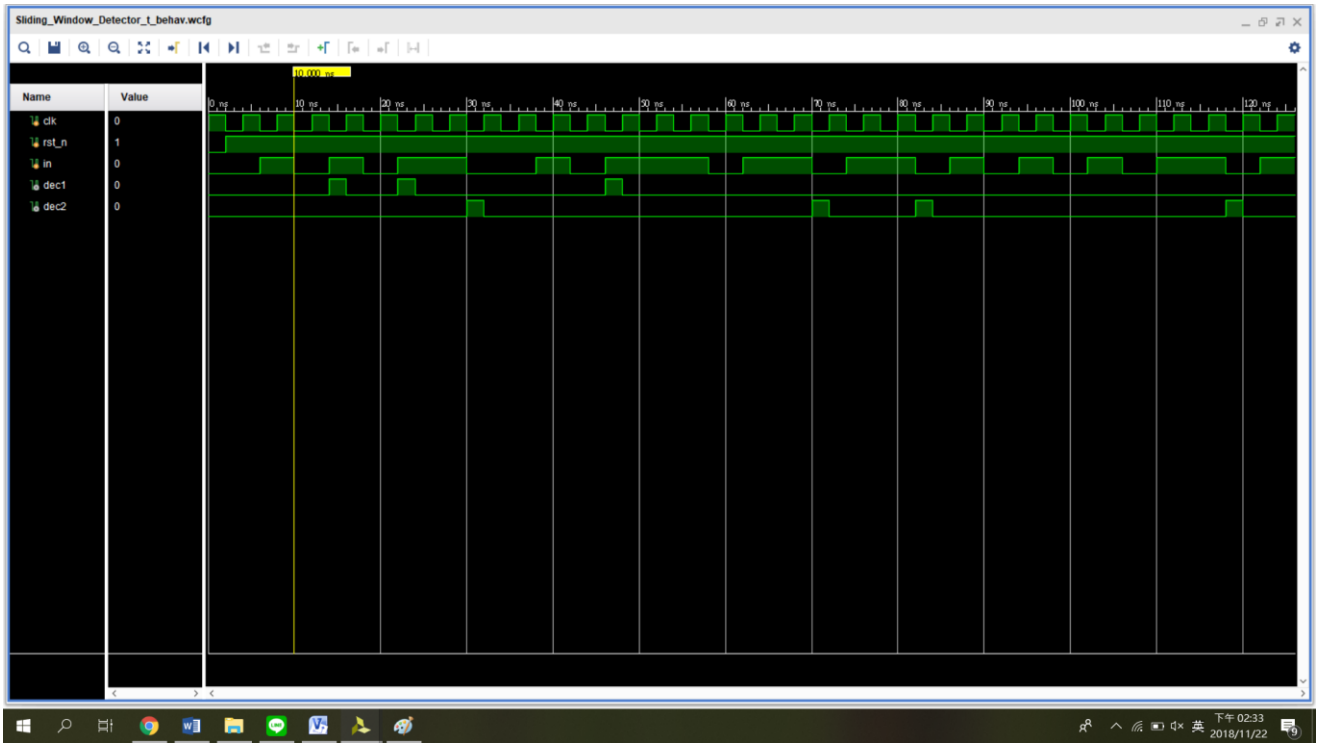
(檢測 dec1)



(檢測 dec2)



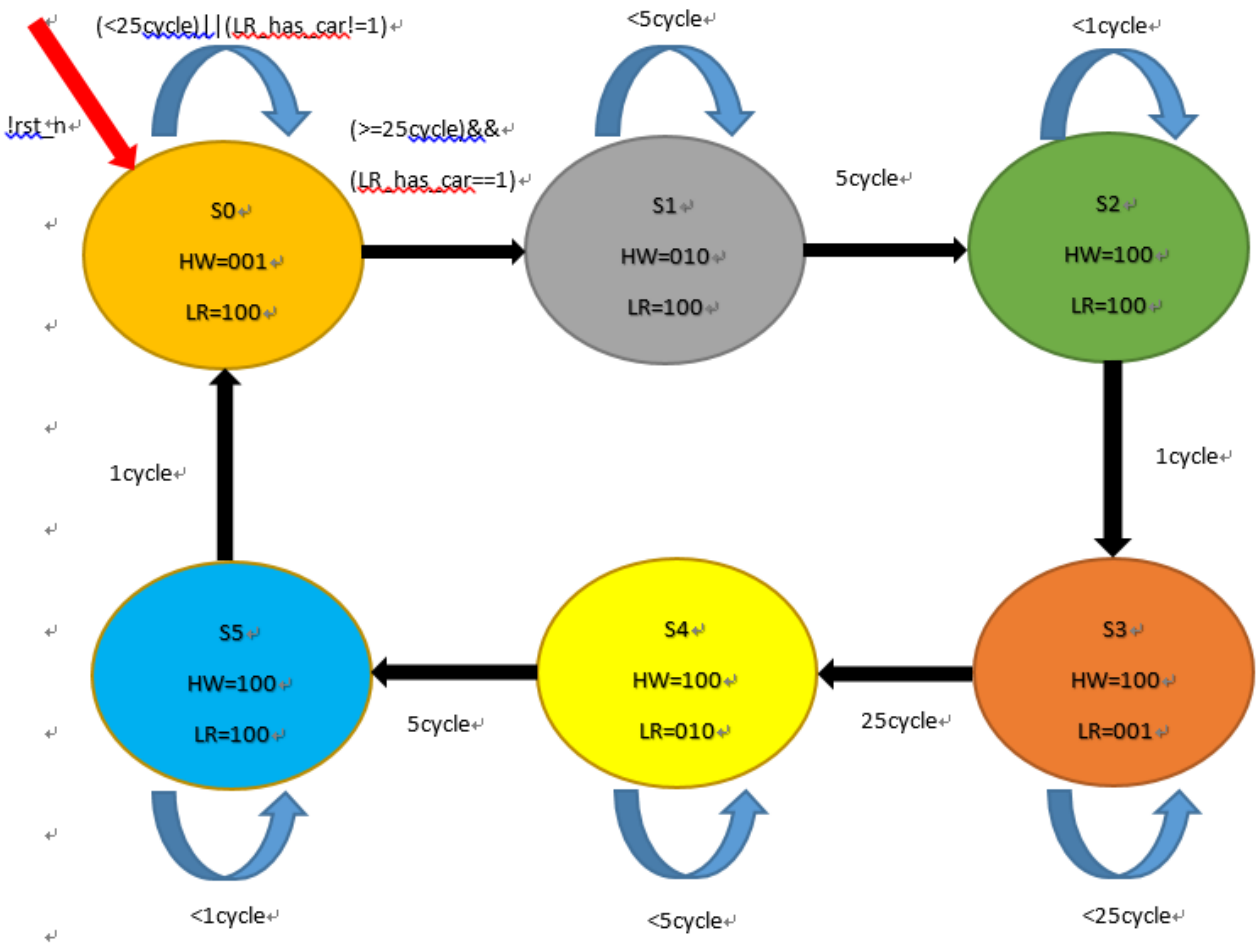
(general test)



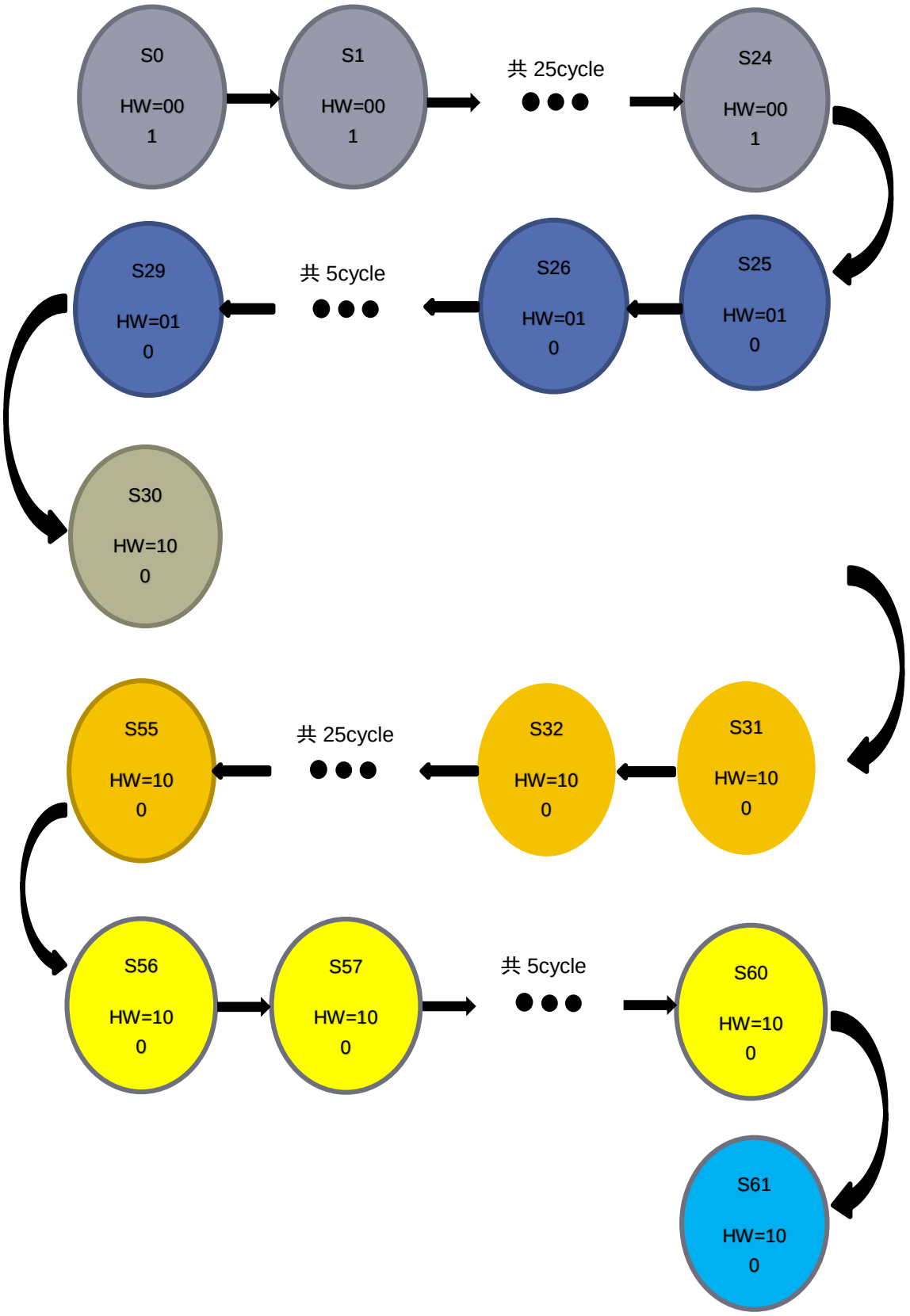
VERILOG QUESTION 3

TRAFFIC LIGHT CONTROLLER

✓ State transition graph

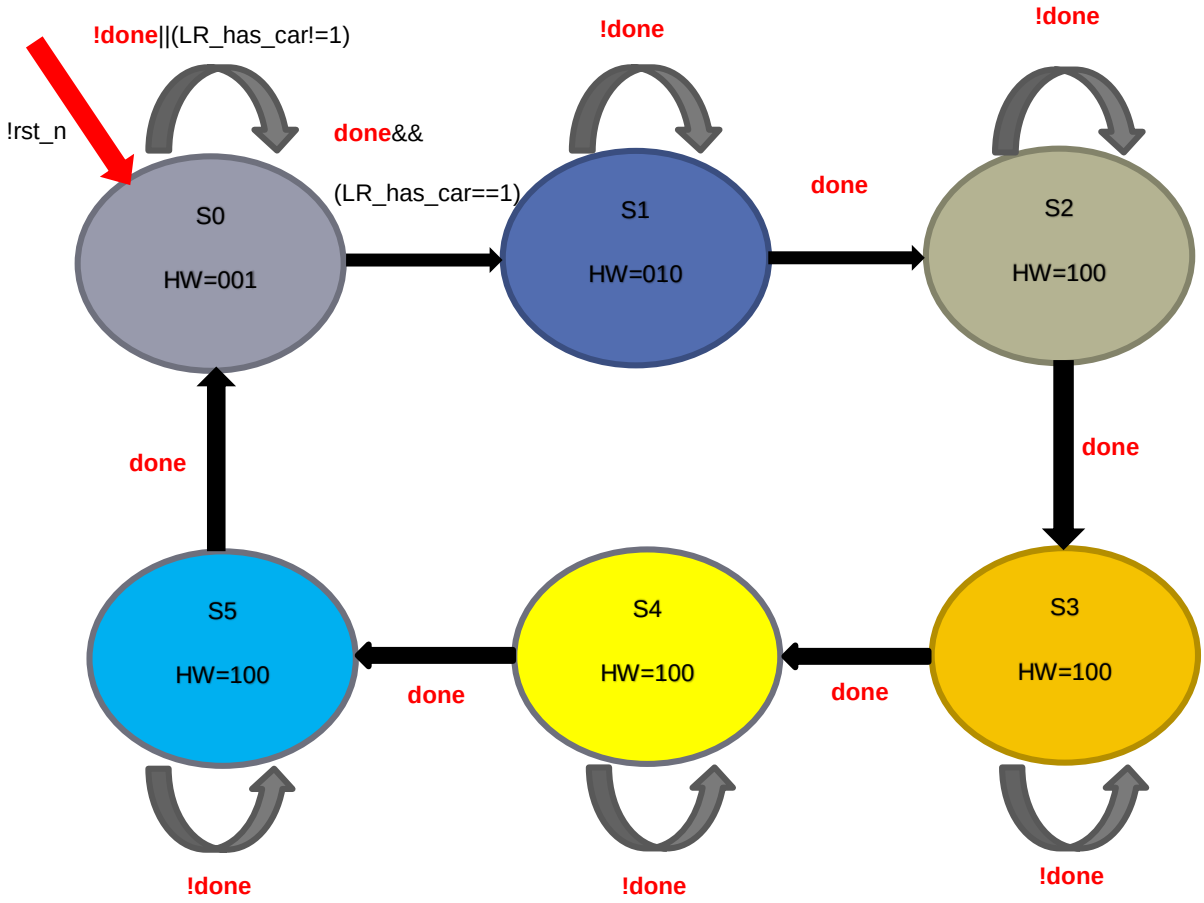


這題實際的做法，若是像下方的方式實作，就會顯得太多 state 而造成不易看懂全貌：

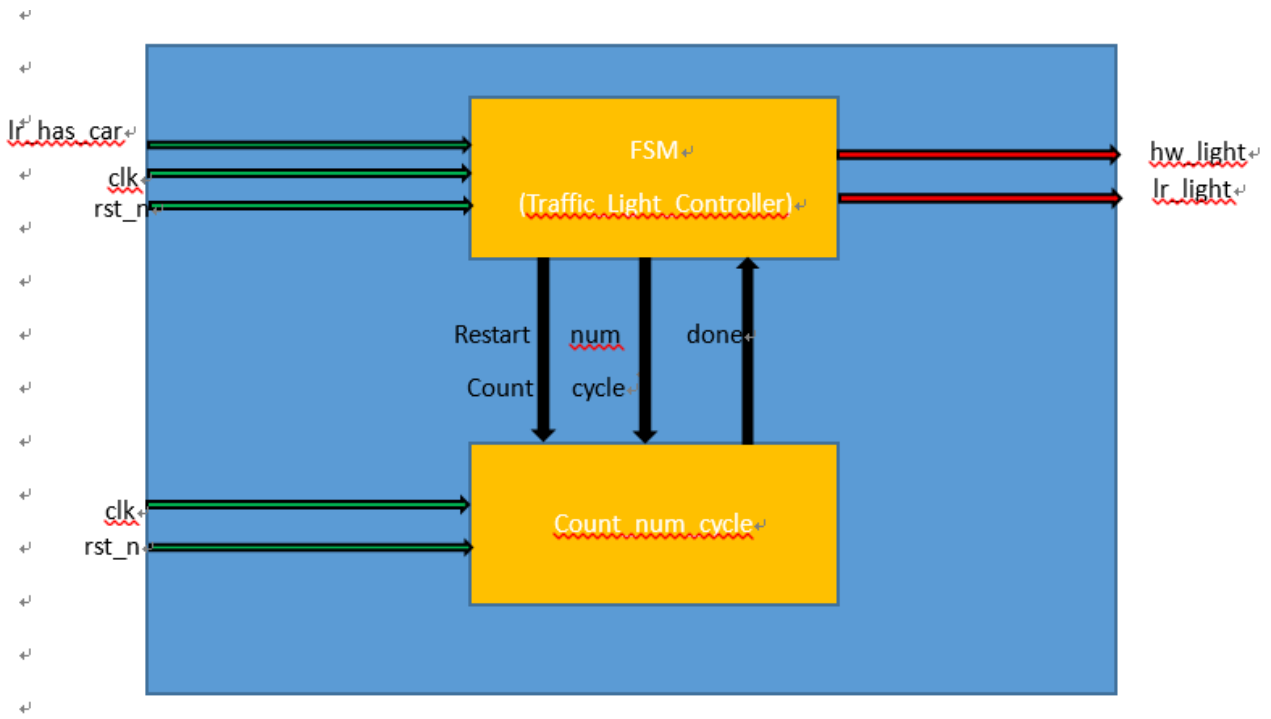


這樣的做法會用到共 61 個 state，不但寫起來很複雜難寫，並且也不易看清全貌。因為同樣的計數周期的功能運用多次，因此我把這樣的功能做成一個 module，若沒有達到所要求的 cycle 次數，就讓 done=0，繼續在同一個 state。若達到所要求的 cycle 次數，就讓 done=1，可以進入下一個 state。這樣的實作方式，只需共 6 個 state。

Transition graph 如下：



Block diagram 如下：



關於計數 cycle 的 module 實作如下

```
55  module count_num_cycle(clk, rst_n, restart_count, num_cycle, done);
56      input clk;
57      input rst_n;
58      input [4:0]num_cycle;
59      input restart_count;
60      output done;
61
62      reg done;
63      reg [4:0]count, next_count;
64
65      always@(posedge clk) begin
66          if(!rst_n||restart_count)  count<=5'b1;
67          else                       count<=next_count;
68      end
69
70      always@(*) begin
71          next_count=(count==num_cycle)? count:count+5'b1;
72          done=(count==num_cycle)? 1'b1:1'b0;
73      end
74
75  endmodule
```

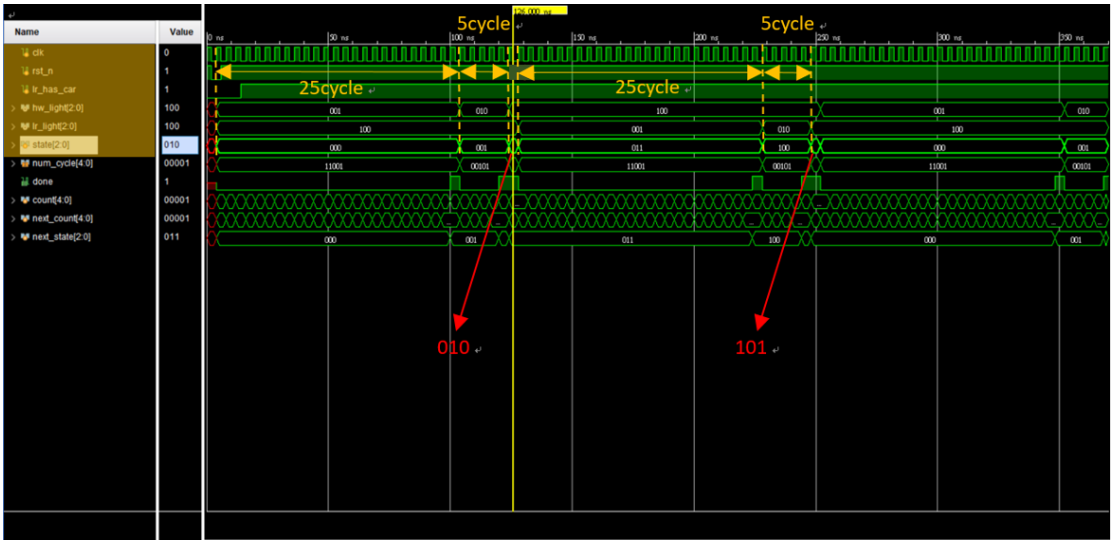
因此，state transition 長的會像這樣。

```
27      // instantiate count_num_cycle
28      count_num_cycle c0(clk, rst_n, restart_count, num_cycle, done);
29
30      always@(posedge clk) begin
31          if(!rst_n) state<=S0;
32          else       state<=next_state;
33      end
34
35      always@(*) begin
36          case(state)
37              S0: {hw_light, lr_light, num_cycle, next_state, restart_count}=
38                  {GRE, RED, 5'd25, (done==1'b1&&lr_has_car==1'b1)? S1:S0, (done==1'b1&&lr_has_car==1'b1)? 1'b1:1'b0};
39              S1: {hw_light, lr_light, num_cycle, next_state, restart_count}=
40                  {YEL, RED, 5'd5, (done==1'b1)? S2:S1, (done==1'b1)? 1'b1:1'b0};
41              S2: {hw_light, lr_light, num_cycle, next_state, restart_count}=
42                  {RED, RED, 5'b1, (done==1'b1)? S3:S2, (done==1'b1)? 1'b1:1'b0};
43              S3: {hw_light, lr_light, num_cycle, next_state, restart_count}=
44                  {RED, GRE, 5'd25, (done==1'b1)? S4:S3, (done==1'b1)? 1'b1:1'b0};
45              S4: {hw_light, lr_light, num_cycle, next_state, restart_count}=
46                  {RED, YEL, 5'd5, (done==1'b1)? S5:S4, (done==1'b1)? 1'b1:1'b0};
47              S5: {hw_light, lr_light, num_cycle, next_state, restart_count}=
48                  {RED, RED, 5'd1, (done==1'b1)? S0:S5, (done==1'b1)? 1'b1:1'b0};
49          endcase
50      end
51  endmodule
```

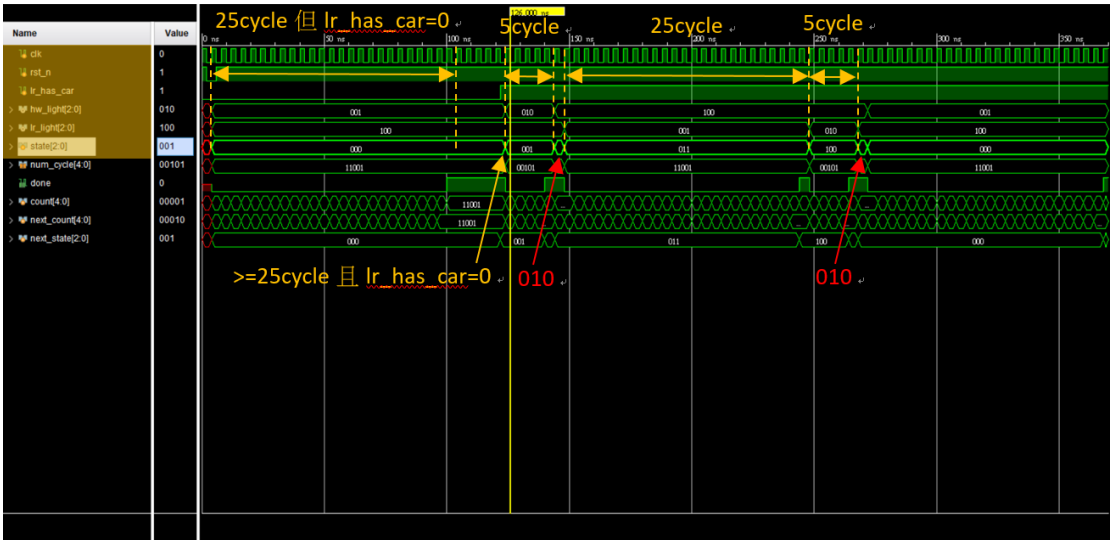
因為是 Moore’ s Machine，故 output 只決定於 state，在不同的 state 就會給出該 state 的 output (hw_light、lr_light)。因為在不同 state 需要的 cycle 數不同，因此會需要帶入 num_cycle 給 count_num_cycle。Next_state 會取決於是否 done，在 state S0 中，除了要 done 外，還需要 lr_has_car，才能決定 next_state。最後，因 count_num_cycle 在不同的 state 中計數完必須要歸零，才能在下一個 state 中重新計算，所以要給出 restart_count 給 count_num_cycle。restart_count 的判斷依與 next_state 的判斷依據相同，若會進入下一個不同的 state，就要將 restart_count 設為 1；若無，就設 restart_count=0。

testBench 的寫法為模仿 spec 示範的波形圖來檢測 state 轉換與 output

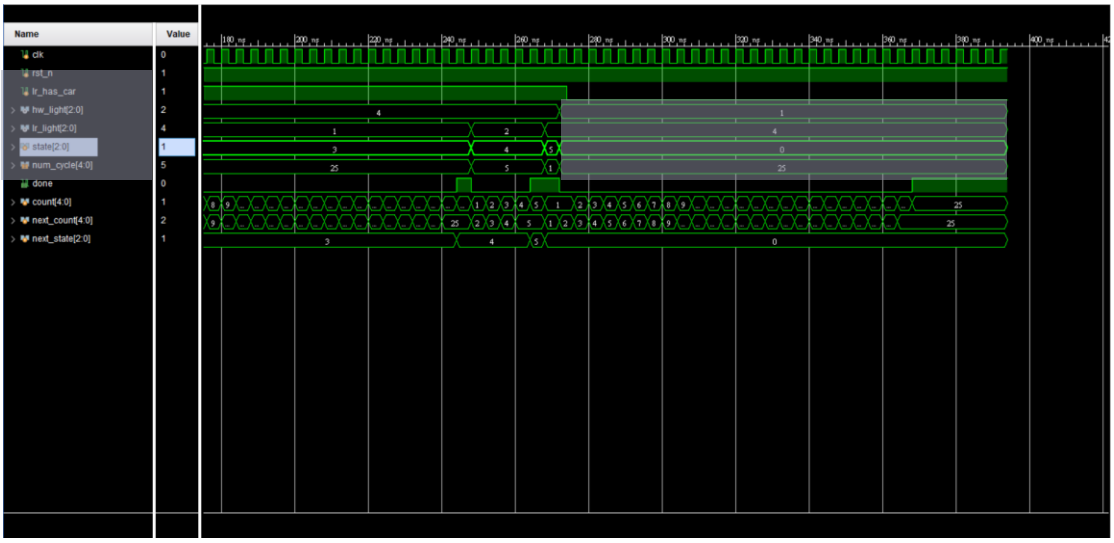
(test1) 一般情況



(test2) state0 等了 25cycle 但 lr_has_car=0 情況



(test3) state0 跑到 state5 後，回到 state0 但 lr_has_car 一直是 0

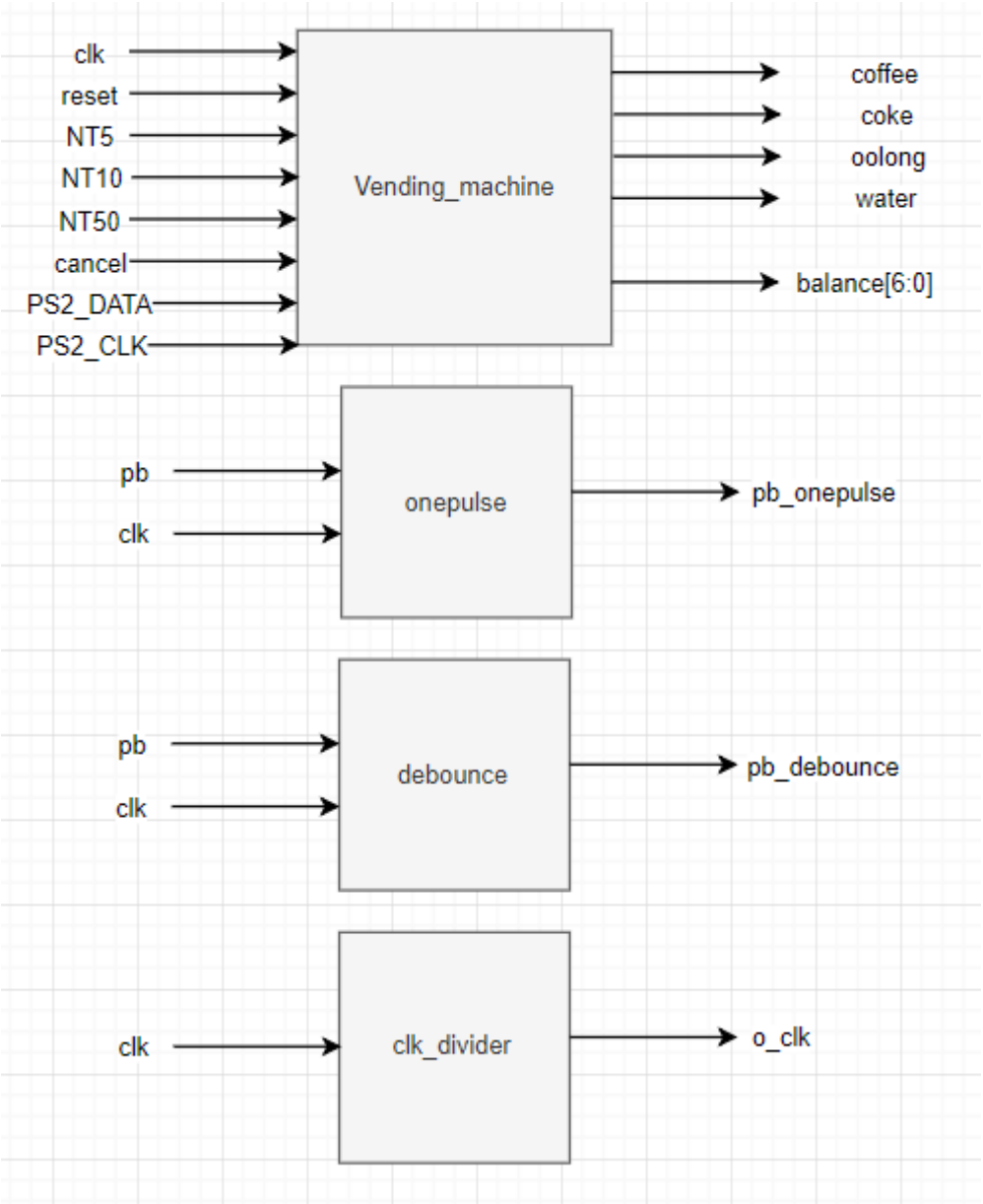


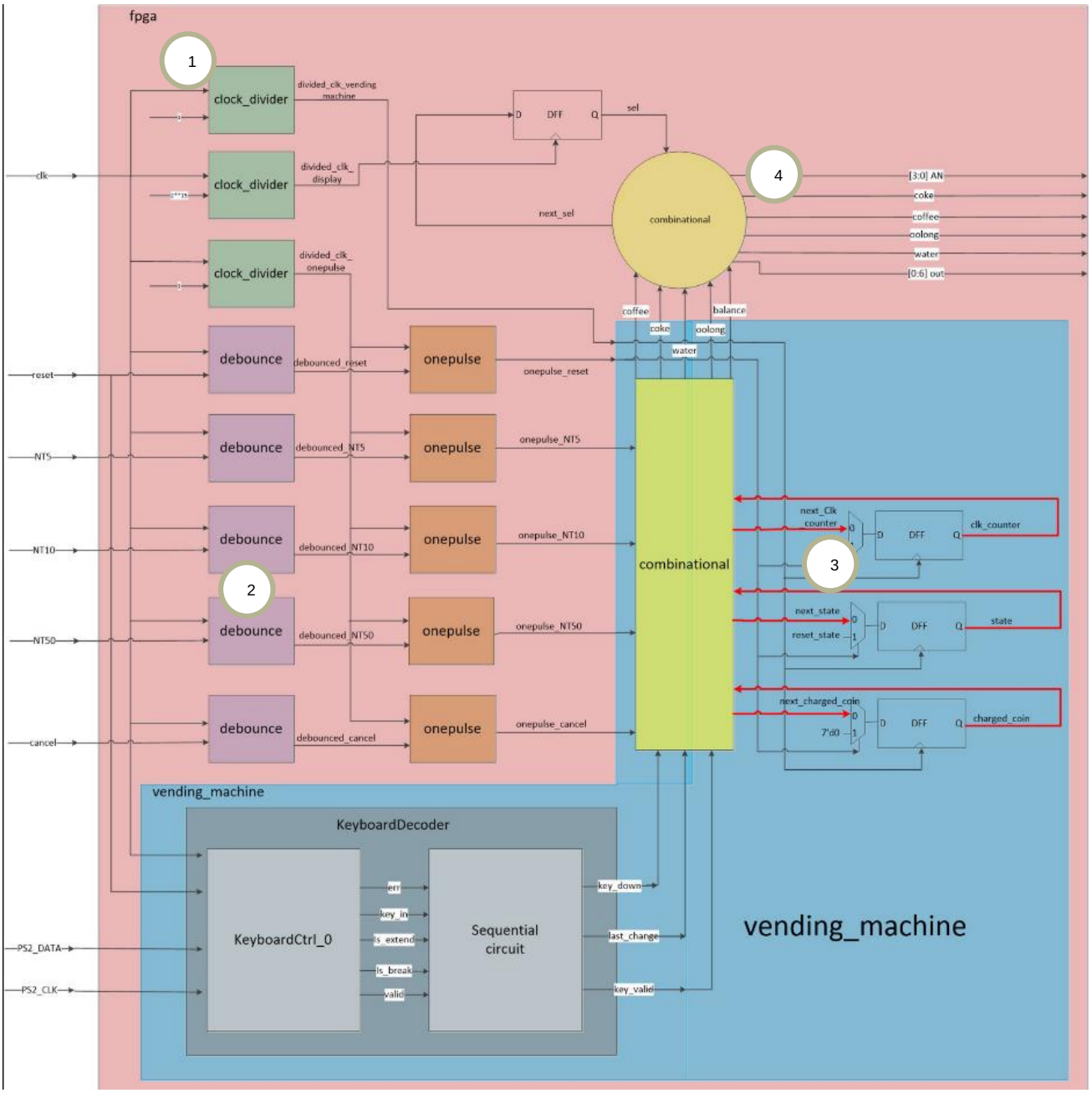
VERILOG QUESTION 4

FPGA

✓ Block diagram

在 top module：fpga 中包含了以下小 module，為了簡化整體的 block diagram，先放上小 module 本身的 port，在整體的 block diagram 就省略小 module 的 port。





✓ 說明

①

關於 clock 由於不同情況下會需要不一樣的 clock 來進行運算，所以我用 clock_divider 做出了三個不一樣的 clock:

divided_clk_vending_machine : 用於 vending_machine 這個 module，因為需要接鍵盤，所子我將它除 1，其實就是板子的 clock，100MHZ(原本我為了能讓找錢的速度是一秒五塊，所以把頻率調成 1HZ，但結果發現 1HZ 時按鍵盤沒反應，所以後來用了 100MHZ，至於怎麼讓它一秒找一次錢，這個我在下面會詳述)。

divided_clk_onepulse : 用於 onepulse 這個 module 上，它的頻率必須跟 divided_clk_vending_machine 一樣，這樣才不會發生在 vending_machine 在遇到 posedge clk 前 onepulse 的訊號就變回 0 的尷尬情形。

divided_clk_display : 用於讓四個七段顯示器分別通電的時間週期。我將內建 clock 乘以 2^{15}。若四個七段顯示器通電時間太短則來不及通電，無法顯示正確的數字，若太長則無法造成明顯的視覺暫留。

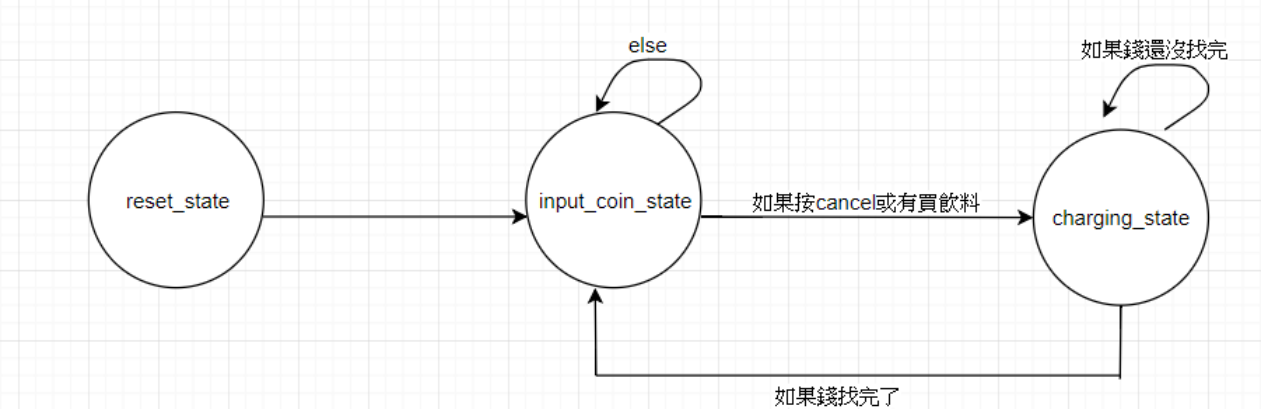
②

clk 設定好後，處裡按鈕的去除雜訊，按鈕包括 reset、cancel、NT5、NT10、NT50。debounce module 包含四個 DFF 相連，DFF 的 clk 都使用 BASYS 內建 clk，即 100MHz。當四個 DFF output 都是 1 代表準確的按下，輸出 debounced_reset、debounced_cancel、NT5、NT10、NT50。

③

關於 vending_machine 這個 module，它的主要任務就是負責販賣機的相關運算(如找錢、算錢等)，並將 output(coffee、coke、oolong、water、balance)傳給 top module，詳細解釋如下:

✓ State transition diagram



在偵測到 reset 訊號時，我將 charged_coin(這個變數是指買家投入的金額)設成 0、state 設給 reset_state，如果沒有偵測到 reset 訊號，那 state、charged_coin、clk_counter 就會分別等於 conbinational block 計算出來的 next_state、next_charged_coin、next_clk_counter(關於 clk_counter 我在下面會詳述)，code 如下圖所示:

```
always@(posedge clk)begin
    if(reset)begin
        state <= reset_state;
        charged_coin <= 7'd0;
    end
    else begin
        state <= next_state;
        charged_coin <= next_charged_coin;
        clk_counter <= next_clk_counter;
    end
end
```

在 reset_state 時，下一個 state 設給 input_coin_state 以等待 start 訊號，賣家投入金額(charged_coin)保持不動，clk_counter 初始成 0，code 如下圖所示:

```
reset_state:begin
    next_state = input_coin_state;
    next_charged_coin = charged_coin;
    next_clk_counter = 27'd0;
end
```

在 input_coin_state 時，主要的任務就是計算買家投入的金額，以及買飲料時所需要的運算(像是扣錢)，如果按下 cancel 按鍵，那下一個 state 就會進入 charging_state 以進行找錢相關的運算，code 如下圖所示:

```
if(cancel == 1'b1)begin
    next_state = charging_state;
end
else begin
```

如果 cancel 鍵沒有被按下，那我就計算買家目前投入了多少金額，並看看目前的金額是否足以買飲料，如果足夠就讓相對應的 output 變成 1 使 LED 燈發亮(例如買家投入 55 元，這時他可以買 coffee、coke、oolong、water，所以這四個變數都會被設成 1，以便讓 LED 燈發亮)，此外，input_coin_state 也會對買家所買的飲料進行餘額運算(例如買家投入 80 元，買了 55 元的咖啡剩 25 元)，code 如下所示:

```
input_coin_state:begin
    if(cancel == 1'b1)begin
        next_state = charging_state;
    end
    else begin
        if(NT5 == 1'b1)begin
            if(charged_coin > 7'd75)begin
                next_charged_coin = 7'd80;
            end
            else begin
                next_charged_coin = charged_coin + 7'd5;
            end
        end
        else begin
            if(NT10 == 1'b1)begin
                if(charged_coin > 7'd70)begin
                    next_charged_coin = 7'd80;
                end
                else begin
                    next_charged_coin = charged_coin + 7'd10;
                end
            end
            else begin
                if(NT50 == 1'b1)begin
                    if(charged_coin > 7'd30)begin
                        next_charged_coin = 7'd80;
                    end
                    else begin
                        next_charged_coin = charged_coin + 7'd50;
                    end
                end
                else begin
                    next_charged_coin = charged_coin;
                end
            end
        end
```

如果投入五元，買家投入的金額(charged_coin)就加 5，且總金額不會超過 80 元

如果投入十元，買家投入的金額(charged_coin)就加 10，且總金額不會超過 80 元

如果投入五十元，買家投入的金額(charged_coin)就加 50，且總金額不會超過 80 元

如果沒投錢，charged_coin 就保持不變

```
if(charged_coin < 7'd20)begin
    coke = 0;
    coffee = 0;
    oolong = 0;
    water = 0;
end
else if(charged_coin >= 7'd20 && charged_coin < 7'd25)begin
    coke = 1;
    coffee = 0;
    oolong = 0;
    water = 0;
end
else if (charged_coin >= 7'd25 && charged_coin < 7'd30) begin
    coke = 1;
    oolong = 1;
    water = 0;
    coffee = 0;
end
else if(charged_coin >= 7'd30 && charged_coin < 7'd55)begin
    coke = 1;
    oolong = 1;
    water = 1;
    coffee = 0;
end
else begin
    coke = 1;
    oolong = 1;
    water = 1;
    coffee = 1;
end
```

如果投入金額小於 20，什麼飲料都沒辦法買

如果投入金額在 20~24 之間，可以買可樂!

如果投入金額在 25~29 之間，可以買可樂以及烏龍茶!

如果投入金額在 30~54 之間，可以買可樂、烏龍茶和水!

如果投入金額在大於 55，那所有飲料都能買!

```
if (been_ready && key_down[last_change] == 1'b1)begin
        if(key_s == 1'b1)begin
            if(coke == 1'b1)begin
                next_charged_coin = charged_coin - 7'd20;
                next_state = charging_state;
            end
            else begin
                next_charged_coin = charged_coin;
                next_state = input_coin_state;
            end
        end

        if(key_d == 1'b1)begin
            if(oolong == 1'b1)begin
                next_charged_coin = charged_coin - 7'd25;
                next_state = charging_state;
            end
            else begin
                next_charged_coin = charged_coin;
                next_state = input_coin_state;
            end
        end

        if(key_f == 1'b1)begin
            if(water == 1'b1)begin
                next_charged_coin = charged_coin - 7'd30;
                next_state = charging_state;
            end
            else begin
                next_state = input_coin_state;
                next_charged_coin = charged_coin;
            end
        end

        if(key_a == 1'b1)begin
```

如果買家按下 S 買可樂，先檢查他所投入金額是否足夠，如果夠的話就將 charged_coin 減去可樂價錢，並且讓下一個 state 是 charging_state，以便進行找錢的相關運算; 如果錢不夠，那按 S 就什麼都不會發生

買烏龍茶的相關運算

買水的相關運算

```
if(key_a == 1'b1)begin
    if(coffee == 1'b1)begin
        next_charged_coin = charged_coin - 7'd55;
        next_state = charging_state;
    end
    else begin
        next_state = input_coin_state;
        next_charged_coin = charged_coin;
    end
end
```

買咖啡的相關運算

Charging_state 的主要任務就是找錢，但因為一秒只找一次，而 clk 的頻率是 100MHZ，所以我設了一個 clk_counter，它會在 posedge 時加 1，如果加到 10**8，那就代表已經經過了一秒，此時賣販機就會找五元出去，所剩的金額也就扣五元，如果錢找完了那就會回到 reset_state 重新開始，code 如下圖所示:

```
charging_state:begin
    if(clk_counter == 10**8)begin
        next_clk_counter = 27'd0;

        if(charged_coin > 0)begin
            next_charged_coin = charged_coin - 7'd5;
            next_state = charging_state;
        end
        else begin
            next_charged_coin = charged_coin;
            next_state = reset_state;
        end

    end
    else begin
        next_clk_counter = clk_counter + 1;
        next_state = charging_state;
        next_charged_coin = charged_coin;
    end
```

④

這個部分主要是處理如何將買家所投入的金額以及 coffee、coke、oolong、water 四個 LED 燈正確的顯示在 FPGA 上，

✓ 心得(BY 郭家瑋)

這次 lab 的 pattern 檢測不是很難，主要就是 FSM。紅綠燈的問題也不算難，主要就是要應用 factored FSM，否則會跑出一堆 state。這次 lab 學系到比較多的主要在 basic，學會了如何裝鍵盤與喇叭，雖然蠻複雜的，debug 也因 module 眾多而困難，但聲音跑出來非常有成就感。

✓ 心得(BY 黎佑廷)

我覺得這次 lab 的第一題不難，跟 lab4 的第一題很相似，所以寫起來比較沒那麼陌生，但 debug 花了許多的時間，因為當時也在想圖片裡的問題:

想了很久覺得很奇怪，後來去 ilms 上看時發現有人問了一樣的問題，下次在打之前應該先看看討論區才不會花太多時間 debug，至於，fpga 的部分，由於對鍵盤的使用還不太熟悉，所以花了很多時間 debug。

成員:

黎佑廷: 負責第 1 題以及 fpga

郭家瑋: 負責第 2 題以及第 3 題