2025/10/02
實驗四
序向邏輯練習

姓名：林承羿
學號：01257027
班級：資工 3A
E-mail：IanLin6225@gmail.com

※注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為下周上課前
3. 一人繳交一份
4. 檔名請按照作業檔名格式進行填寫，未依照格式不予批改
5. 檔名範例：學號_姓名_HW4

1、 ReLU 函數

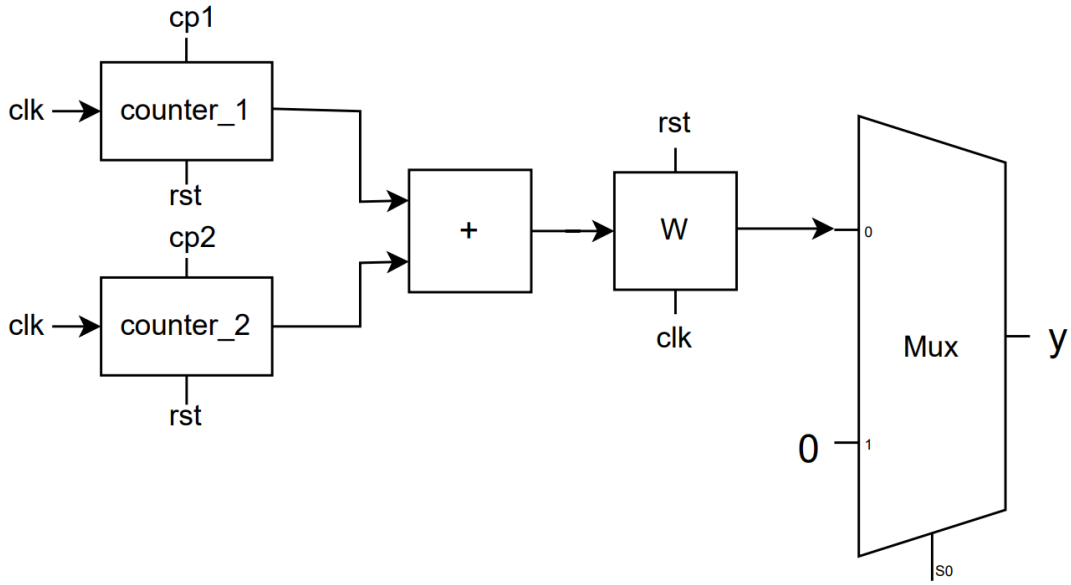
●實驗說明：

設計 1 個 FSM，實現一個 ReLU 函數。

$$ReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

1. Counter_1 從−5、−4、…數到 0 後，接著 counter_2 從 0、1…數到 7
2. 設計 1 個 FSM 控制 2 個[3:0]counters，2 個 counters 相加後，存入暫存器 w，再代入 ReLU 函數(即 mux)，以暫存器 w 之最高位元做為 s0，並輸出最後結果 y。
3. 負數以二進制表示。

●系統硬體架構方塊圖（接線圖）：



●系統架構程式碼、測試資料程式碼與程式碼說明
截圖請善用 **win+shift+S**

relu.sv > relu

```
module relu (
    input logic clk,
    input logic rst,
    output logic [3:0] y
);
    typedef enum {state_0 = 0, state_1 = 1} state_t;
    state_t ps, ns;

    logic cp1, cp2;
    logic [3:0] counter_1, counter_2, w, sum;

    // -5 ~ 0 計數器
    always_ff @(posedge clk) begin
        if (rst) begin
            counter_1 <= -4'sd5;
        end
        else if (cp1) begin
            counter_1 <= counter_1+1;
        end
        else begin
            counter_1 <= counter_1;
        end
    end

    // 0 ~ 7 計數器
    always_ff @(posedge clk) begin
        if (rst) begin
            counter_2 <= 4'd0;
        end
        else if (cp2) begin
            counter_2 <= counter_2+1;
        end
        else begin
            counter_2 <= counter_2;
        end
    end
```

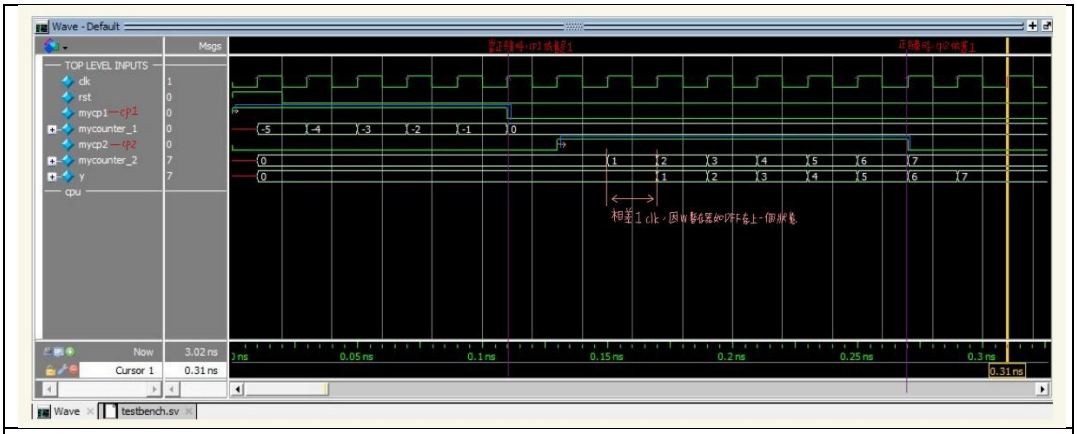
```
37
38      // 狀態初始化，讓進入點是正確的(state_0)
39      always_ff @(posedge clk) begin
40          if (rst)
41              ps <= state_0;
42          else
43              ps <= ns;
44      end
45
46      // FSM controller: s0(狀態零)為counter_1遞減至0; s1(狀態一)為counter_2地增至7
47      always_comb begin
48          cp1 = 0; cp2 = 0; ns = ps;
49          case (ps)
50              state_0: begin
51                  if (counter_1 == 4'd0) begin
52                      ns = state_1;
53                  end
54                  else begin
55                      cp1 = 1;
56                  end
57              end
58
59              state_1: begin
60                  if (counter_2 == 4'd7) begin
61                      ns = state_0;
62                  end
63                  else begin
64                      cp2 = 1;
65                  end
66              end
67          endcase
68      end
```

```
69
70      // 加法器
71      assign sum = counter_1 + counter_2;
72
73      // W 暫存器，正緣給出總和，否則維持
74      always_ff @(posedge clk) begin
75          if (rst) begin
76              w <= 4'd0;
77          end
78          else begin
79              w <= sum;
80          end
81      end
82
83      // 令 s0 為 select 線，為 w 之 most-significant-bit
84      assign s0 = w[3];
85
86      // Relu(x) 多功器
87      always_comb begin
88          case (s0)
89              1'b0: y = w;
90              1'b1: y = 4'd0;
91          endcase
92      end
93  endmodule
```

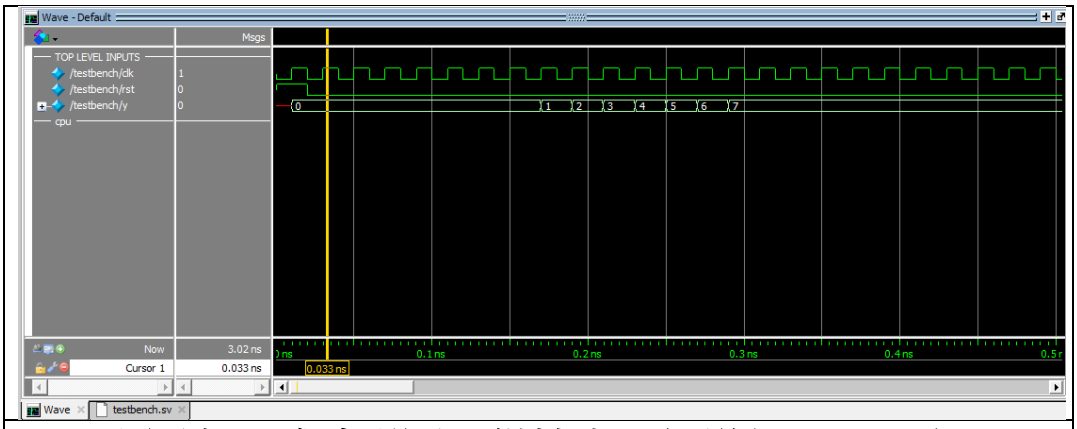
從上圖可看到程式區塊、變數命名基本上依照給定方塊圖施工，其中以 FSM controller 控制兩個 counter，並在 reset 時確認 counter 初值、狀態機的進入點、暫存器 W，而這次學到的是-5 在初始化時要給-4sd’5(負號寫前面，單位為 signed decimal)。

且確認 counter_1 + counter_2 在 bits 為 4 的前提下不可能溢位，邊界為 -5 => 1011，7 => 0111，故不需要用 5 bits 接住總合。

●模擬結果與結果說明：



這是說明用的波形圖，主要應證 cp1、cp2 都應該向右移一小段距離來看待結果才會正確。且 y 與相加的總合一定差一個 clock。



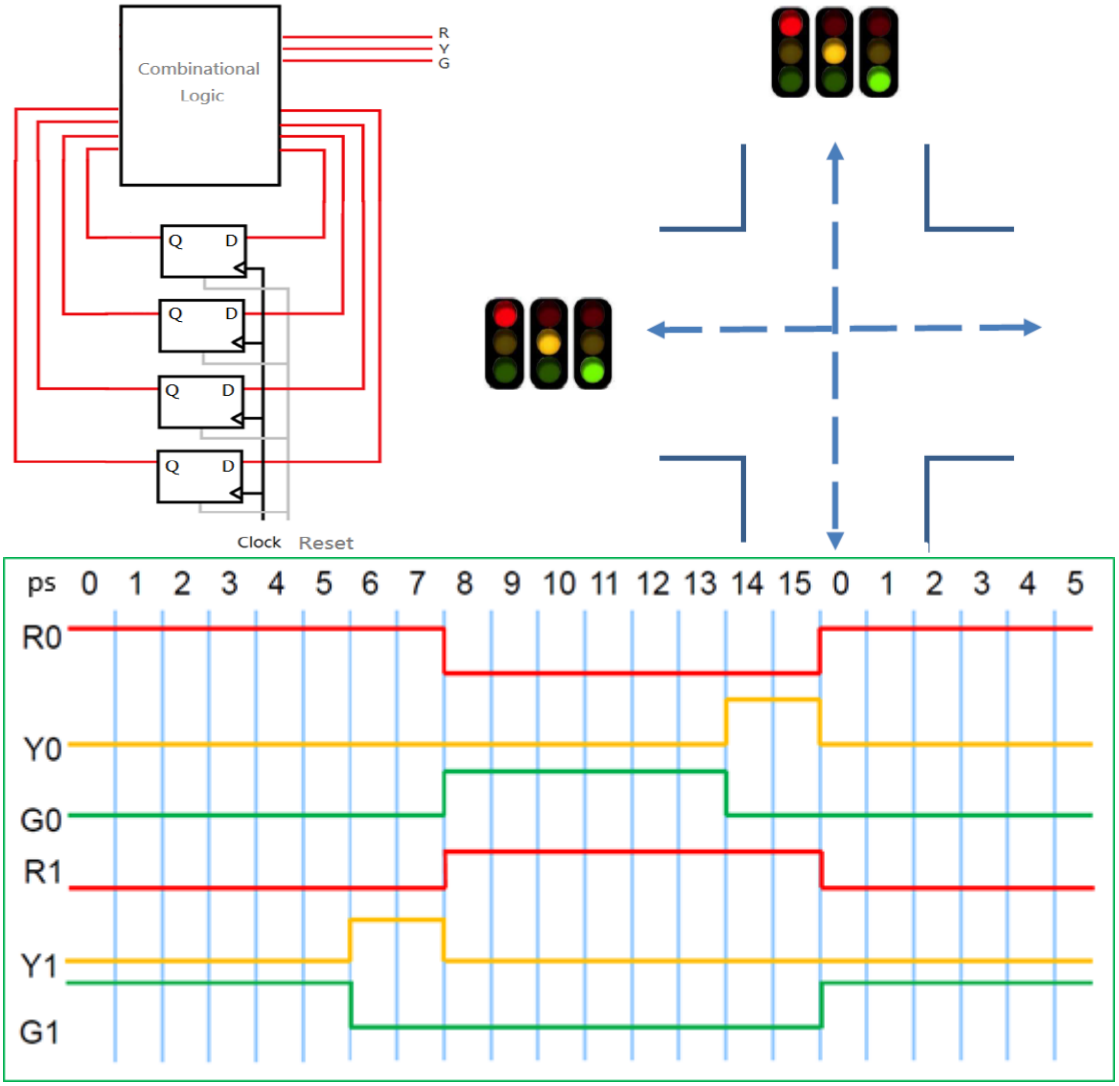
上圖為正式波形圖，從輸出可知道如 Relu(x)之 mux 所示，在 counter_1 的-5~0 區間結果 y 都是 0，在 counter_2 的 0~7 區間結果呈現 counter_1 + counter_2 之總合(因 counter_2 有變化時，counter_1 為 0，故 y 即 counter_2 之值)。

2、 紅綠燈

●實驗說明：

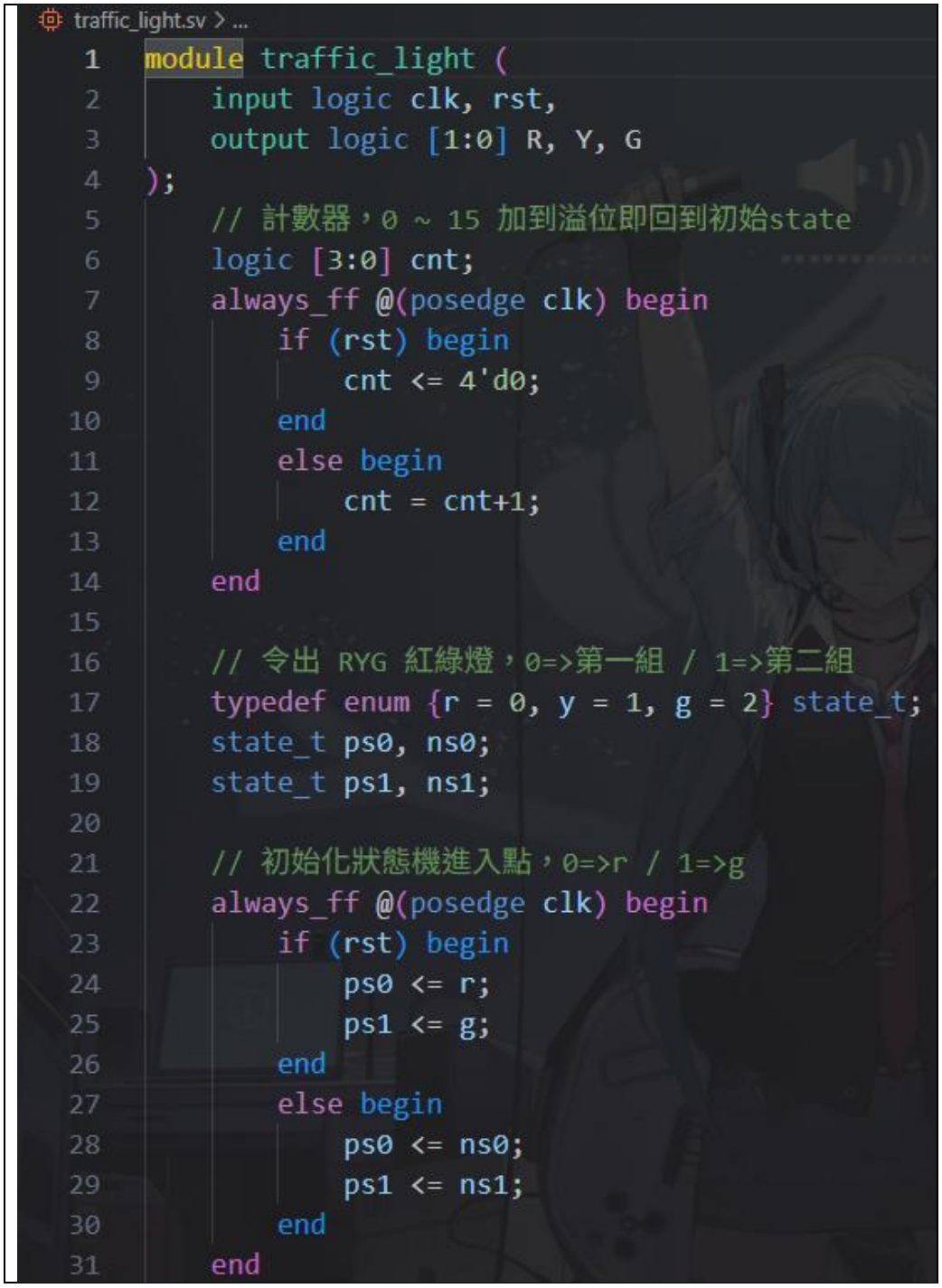
4. 用 FSM 實作紅綠燈
5. 第一組紅綠燈(R[0], Y[0], G[0]) 由紅燈為起點依序變換為 綠燈→黃燈..
6. 第二組紅綠燈(R[1], Y[1], G[1]) 根據地一組紅綠燈的狀態顯示 綠燈→黃燈→紅燈..
7. 以 R 表示紅燈、Y 表示黃燈、G 表示綠燈。
8. 1 表示燈亮，0 表示燈滅，最後輸出[1:0]R、[1:0]Y、[1:0]G。

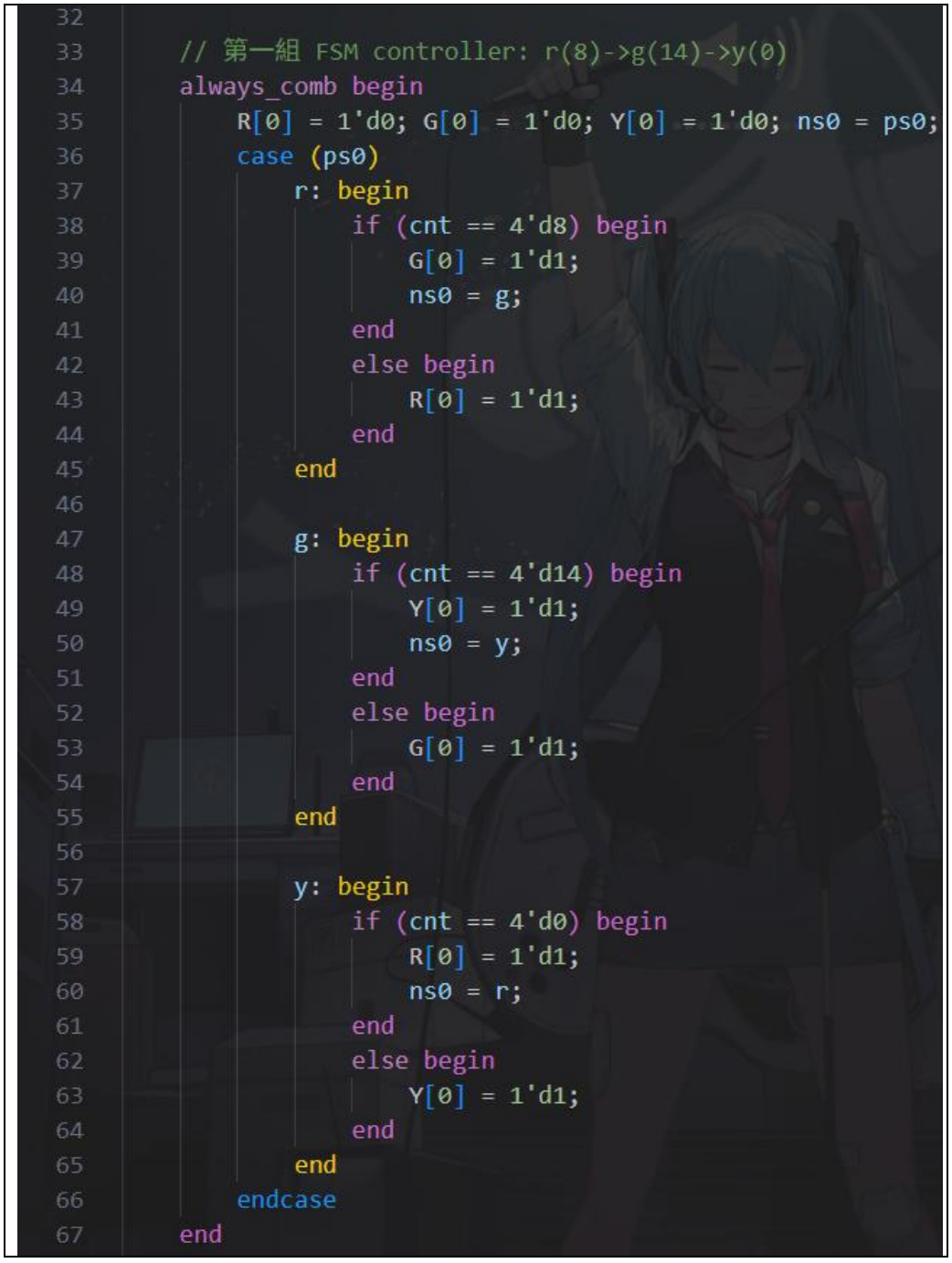
●系統硬體架構方塊圖（接線圖）：



●系統架構程式碼、測試資料程式碼與程式碼說明

截圖請善用 **win+shift+S**


traffic_light.sv > ...
1 module traffic_light (
2 input logic clk, rst,
3 output logic [1:0] R, Y, G
4);
5 // 計數器，0 ~ 15 加到溢位即回到初始state
6 logic [3:0] cnt;
7 always_ff @(posedge clk) begin
8 if (rst) begin
9 cnt <= 4'd0;
10 end
11 else begin
12 cnt = cnt+1;
13 end
14 end
15
16 // 令出 RYG 紅綠燈，0=>第一組 / 1=>第二組
17 typedef enum {r = 0, y = 1, g = 2} state_t;
18 state_t ps0, ns0;
19 state_t ps1, ns1;
20
21 // 初始化狀態機進入點，0=>r / 1=>g
22 always_ff @(posedge clk) begin
23 if (rst) begin
24 ps0 <= r;
25 ps1 <= g;
26 end
27 else begin
28 ps0 <= ns0;
29 ps1 <= ns1;
30 end
31 end



```
32
33      // 第一組 FSM controller: r(8)->g(14)->y(0)
34      always_comb begin
35          R[0] = 1'd0; G[0] = 1'd0; Y[0] = 1'd0; ns0 = ps0;
36          case (ps0)
37              r: begin
38                  if (cnt == 4'd8) begin
39                      G[0] = 1'd1;
40                      ns0 = g;
41                  end
42                  else begin
43                      R[0] = 1'd1;
44                  end
45              end
46
47              g: begin
48                  if (cnt == 4'd14) begin
49                      Y[0] = 1'd1;
50                      ns0 = y;
51                  end
52                  else begin
53                      G[0] = 1'd1;
54                  end
55              end
56
57              y: begin
58                  if (cnt == 4'd0) begin
59                      R[0] = 1'd1;
60                      ns0 = r;
61                  end
62                  else begin
63                      Y[0] = 1'd1;
64                  end
65              end
66          endcase
67      end
```

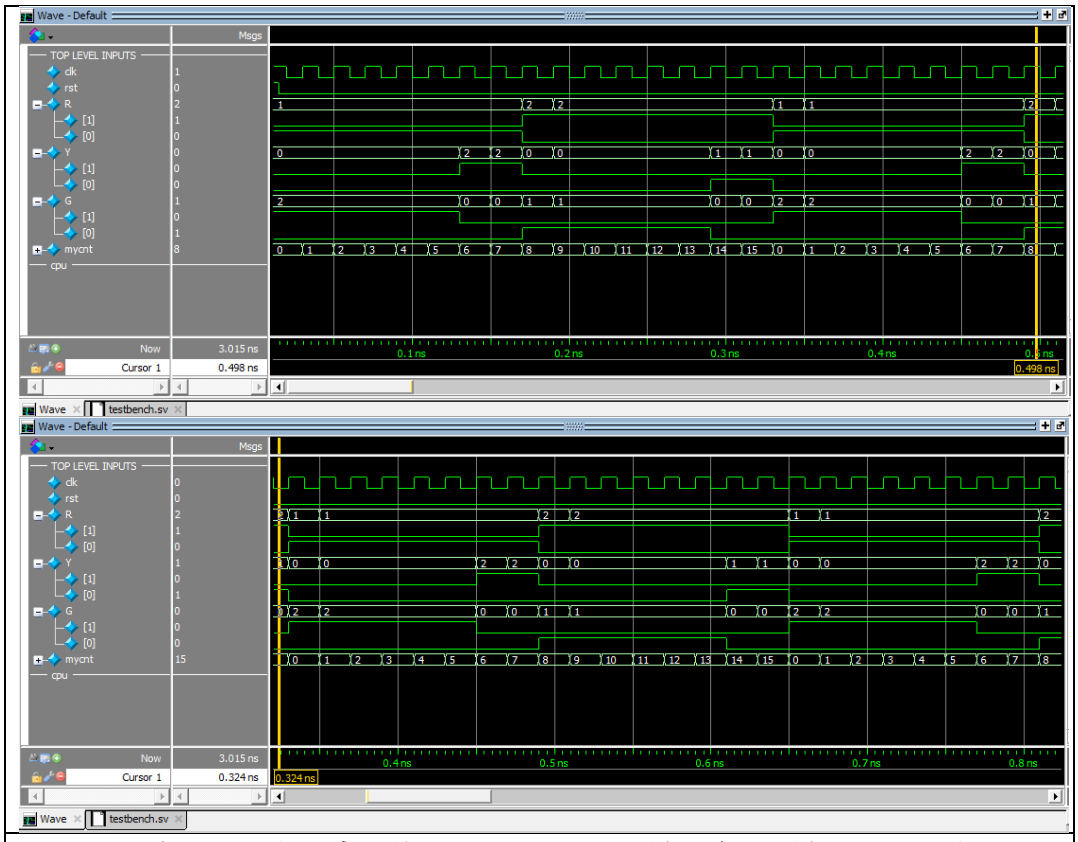
```
68
69      // 第二組 FSM controller: g(6)->y(8)->r(0)
70      always_comb begin
71          G[1] = 1'd0;  Y[1] = 1'd0; R[1] = 1'd0; ns1 = ps1;
72          case (ps1)
73              g: begin
74                  if (cnt == 4'd6) begin
75                      Y[1] = 1'd1;
76                      ns1 = y;
77                  end
78                  else begin
79                      G[1] = 1'd1;
80                  end
81              end
82
83              y: begin
84                  if (cnt == 4'd8) begin
85                      R[1] = 1'd1;
86                      ns1 = r;
87                  end
88                  else begin
89                      Y[1] = 1'd1;
90                  end
91              end
92
93              r: begin;
94                  if (cnt == 4'd0) begin
95                      G[1] = 1'd1;
96                      ns1 = g;
97                  end
98                  else begin
99                      R[1] = 1'd1;
100                 end
101             end
102         endcase
103     end
104 endmodule
```

與第一題不同的是，我這題採用在 FSM controller 中不將個別狀態要做的事以 flag 分開，而是寫在裡面，以下說明原因。

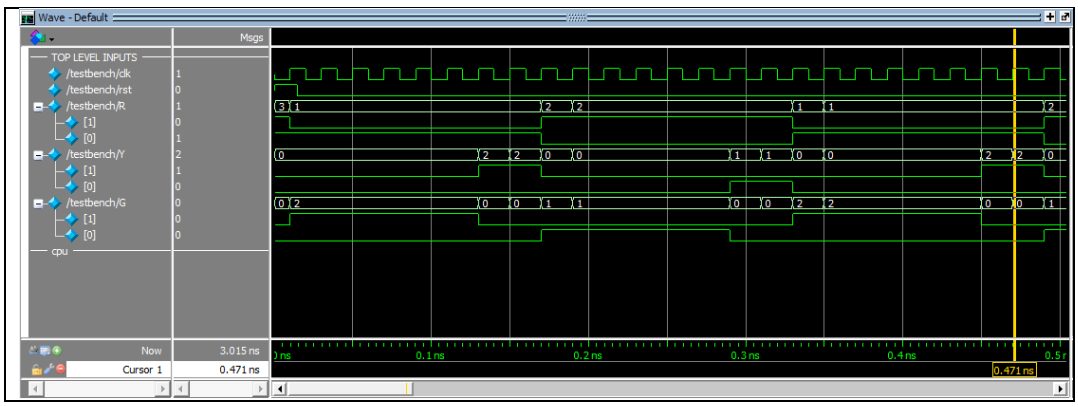
從提供的波形圖已知 RYG 其中一者為 low，必有另一者馬上為 high，如果只在 else 寫出要 output 會慢一個 clock，意思是我需要在 if 中提前一個 clock 坐下個狀態的結果以保證必有一者為 high，因此如果合在一起寫可能會多寫另外兩個 always_comb，沒啥必要。

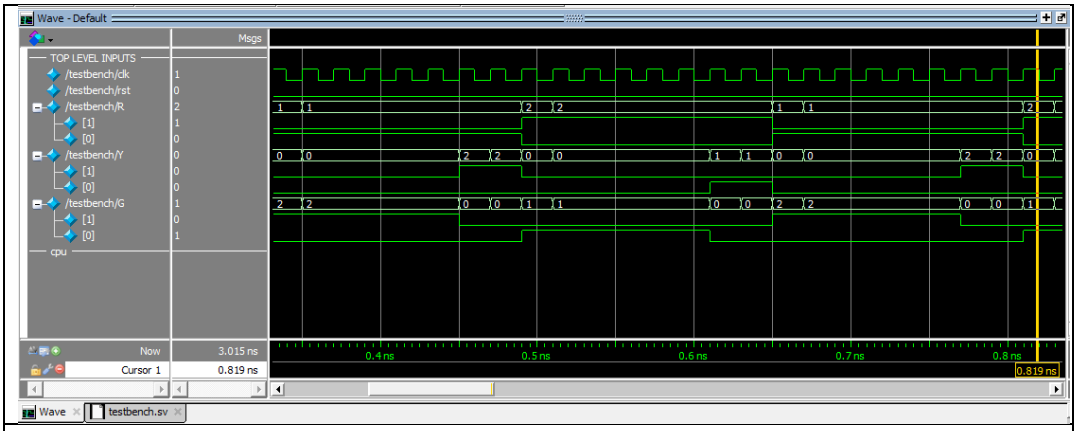
另外可以注意到的是計數器為 0~15，為 4bits 溢位後當歸零，即從頭開始。因此在 reset 後沒有將計數器歸零的必要。

●模擬結果與結果說明：



此為說明用波形圖，可以看到最底下的 mycnt 是一個計數器，方便用來快速確認 RYG 是否正確，本人放上兩段的波形圖以證明可以循環相同的結果。





此為正式波型圖，具體差別只是拿掉計數器，其餘皆沒有改變。

●結論與心得：

經過此次的實驗，我更熟悉如何撰寫 system verilog 中的狀態機，並更深刻的了解波形圖所呈現的結果。第一題印象深刻的是學會如何初始化負數。從第二題中我學習到如何撰寫多個狀態基於同一份 sv 檔案中。除此之外最多的是透過擴充 tb 查看自己想要的輸出，讓除錯更有邏輯與循序漸進的感覺而非一味通靈毫無頭緒的憑猜想嘗試。