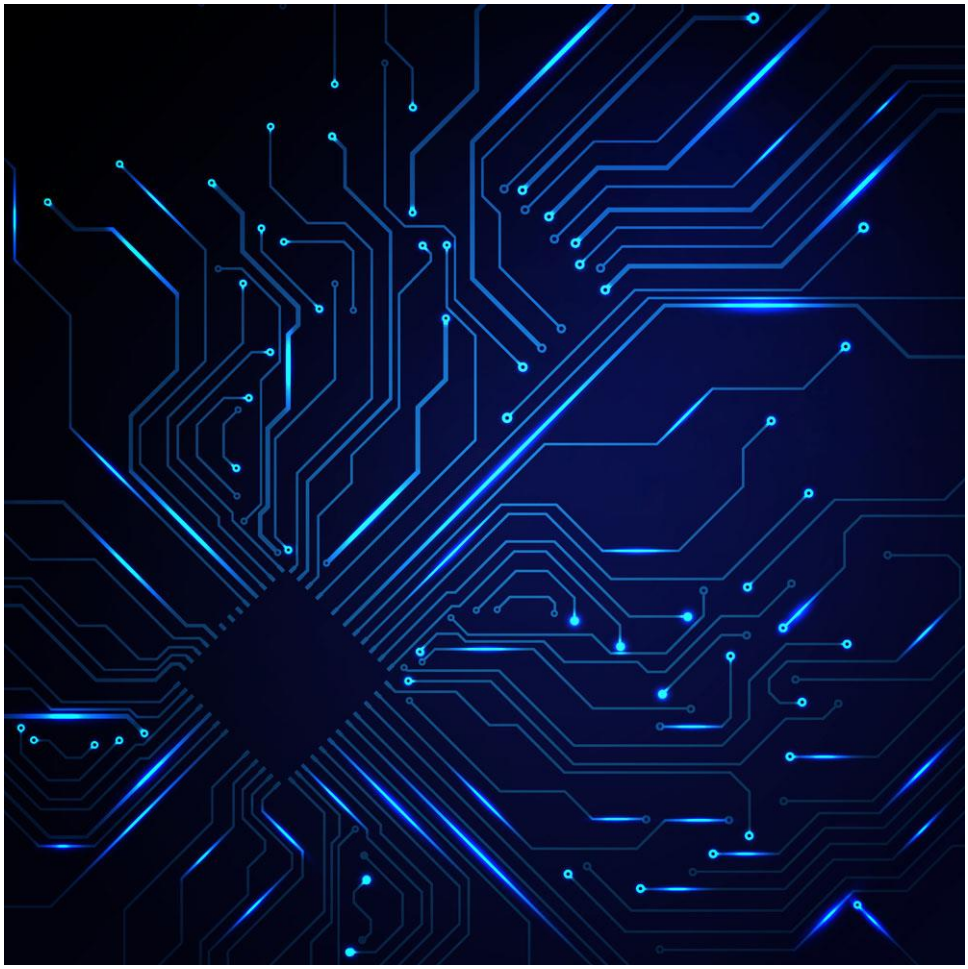


注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為
   隔週三上午九點
3. 一人繳交一份
4. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

2023/03/08

實驗二

序向邏輯練習

姓名：吳秉宸　　學號：00957202

班級：資工 4A

E-mail：evan20010126@gmail.com

一、 Counter + register

■ 實驗說明：

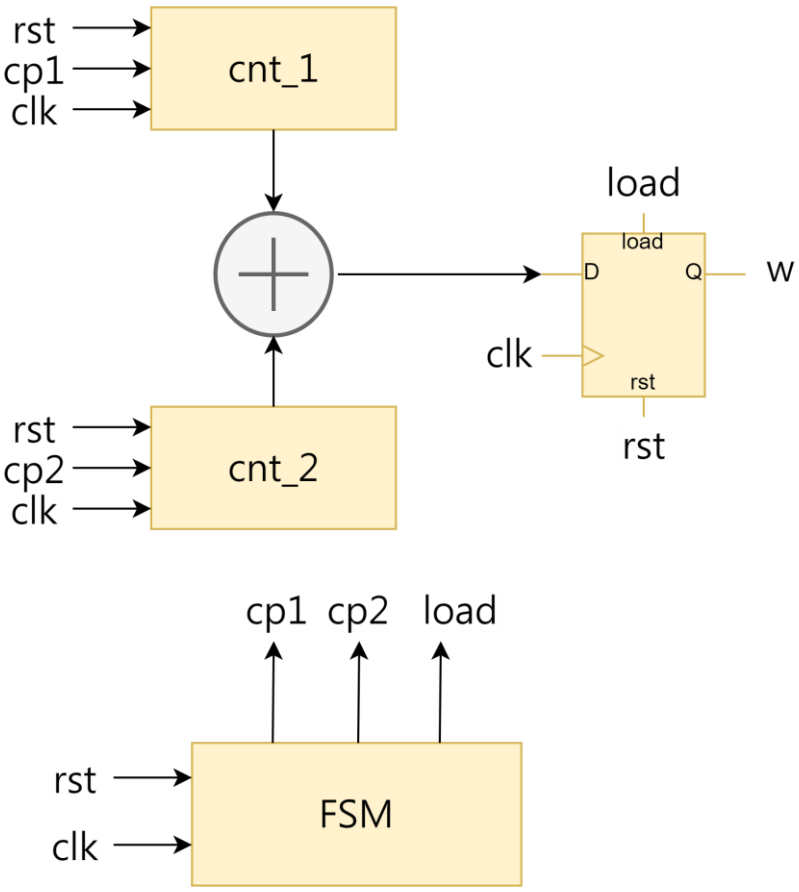
如下圖所示， 2 個 4 bits counter (cnt_1, cnt_2)和 1 個 4 bits 加法器。

- counter 計數器
  - clk 為系統時脈 50 MHz
  - rst = 1 則清除為 0
  - cp = 1 則計數器加 1
- w 暫存器 DFF
  - clk 為系統時脈 50 MHz
  - load_w = 1 則資料載入 w reg

請用 systemverilog 設計此電路，及控制之 FSM ：

1. 將 cnt_1 數到 3，cnt_2 數到 4，然後相加，最後存入暫存器 w (=3+4)。
2. 將 cnt_1 數到 5，cnt_2 數到 7，然後相加，最後存入暫存器 w (=5+7)。

■ 系統硬體架構方塊圖：



■ 系統架構程式碼、測試資料程式碼與程式碼說明(.sv 檔及.do 檔都要截圖)

截圖請善用 win+shift+S

- Counter_wait_plus.sv

```
module counter_wait_plus (
    input clk,
    input reset,
    output logic [3:0] a,
    output logic [3:0] b,
    output logic [3:0] W
);

logic cnt_a;
logic cnt_b;

logic [1:0] load_w;

logic a_ps, a_ns;
logic b_ps, b_ns;

logic [3:0] S;
assign S = a + b;

always_ff @(posedge clk)
begin
    if(reset) a<=#1 0;
    else if(cnt_a) a <= #1 a+1;
end
always_ff @(posedge clk)
begin
    if(reset) b<=#1 0;
    else if(cnt_b) b <= #1 b+1;
end

always_comb
begin
    if(load_w[1] && load_w[0]) W <= #1 S;
end

always_ff @(posedge clk)
// for next state
begin
    if(reset)
        begin
            a_ps<=#1 0;
            b_ps<=#1 0;
        end
    else
```

系統架構的部分，設置 input 為 clk 與 reset，output a,b 為分別為兩個 counter，會同時倒數，a 數到 3 會停，b 數到 4 會停，而當 a,b 都停時，會輸出總和為 W。然後 a 會再數到 5，b 會再數到 7，再輸出總和 W。

設置 control 線 cnt_a 與 cnt_b 分別控制是否要繼續+1，也設置 load_w 兩個 bits，當 a 加到 3 or 5 時 load_w[0]=1，當 b 加到 4 or 7 時，load_w[1]=1。

21~31 行分別做 a，b 的 counter，會依照控制線信號來判斷是否+1。

33~36 行設定為 combination，讓 load = “11”的當下就加總，若設置 flipflop 會延遲一個 clk 才加總。

38~51 行，會在 clk 發生時，將 present_state(ps)設為 next_state(ns)的值。

```
        begin
            a_ps <= #1 a_ns;
            b_ps <= #1 b_ns;
        end
end

parameter T0 = 0; // keep going
parameter T1 = 1; // stop

always_comb
begin
    cnt_a = 0;
    cnt_b = 0;
    a_ns = 0;
    b_ns = 0;
    load_w = 0;

    case(a_ps)
        T0:
        begin
            cnt_a = 1;
            if (a== 2 || a == 4) a_ns = T1;
            else a_ns = T0;
        end
        T1:
        begin
            a_ns = T1;
            load_w[0] = 1;
        end
    endcase

    case(b_ps)
        T0:
        begin
            cnt_b = 1;
            if (b==3 || b == 6) b_ns = T1;
            else b_ns = T0;
        end
        T1:
        begin
            load_w[1] = 1;
            if (b == 7) b_ns = T1;
            else begin
                a_ns = T0;
                b_ns = T0;
            end
        end
    endcase
end
endmodule
```

設置狀態 To(0)當作繼續累加的狀態，T1(1)為暫停累加的狀態。

下方即為 FSM，

先為 a 的 state 轉換與依照狀態設置 control 線的值，值得注意的是當 a==2 時此回合會加到 3，下一輪狀態應設為 T1，而到 T1 時狀態值回到 T1 且 load 線設為 1。

後為 b 的 state 轉換與依照狀態設置 control 線的值，值得注意的是當 b==3 時此回合會加到 4，下一輪狀態應設為 T1，且讓 a、b counter 繼續數，數到 5 與 7 時再回到 T1 且讓狀態維持在 T1。

- complie.do

```
#vlib work



# ----------------------------------------------------------------
vlog ../tb/testbench.sv
vlog ../../design/sub_4bit.sv
vlog ../../design/DE0_CV.sv
vlog ../../design/counter_wait_plus.sv









# ----------------------------------------------------------------
```

Compile testbench 與 counter_wait_plus 檔案。

- Wave.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -divider {TOP LEVEL INPUTS}
add wave -noupdate /testbench/clk
add wave -noupdate /testbench/rst
add wave -noupdate -divider cpu
add wave -noupdate -radix hexadecimal /testbench/a
add wave -noupdate -radix hexadecimal /testbench/b
add wave -noupdate -radix hexadecimal /testbench/W
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {79 ps} 0}
quietly wave cursor active 1
configure wave -namecolwidth 150
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ps
update
WaveRestoreZoom {0 ps} {1 ns}
```

上半部將 clk 與 reset 呈現在模擬圖。

下方將 a、b 與加總後的 W 數值以 hex 表示。

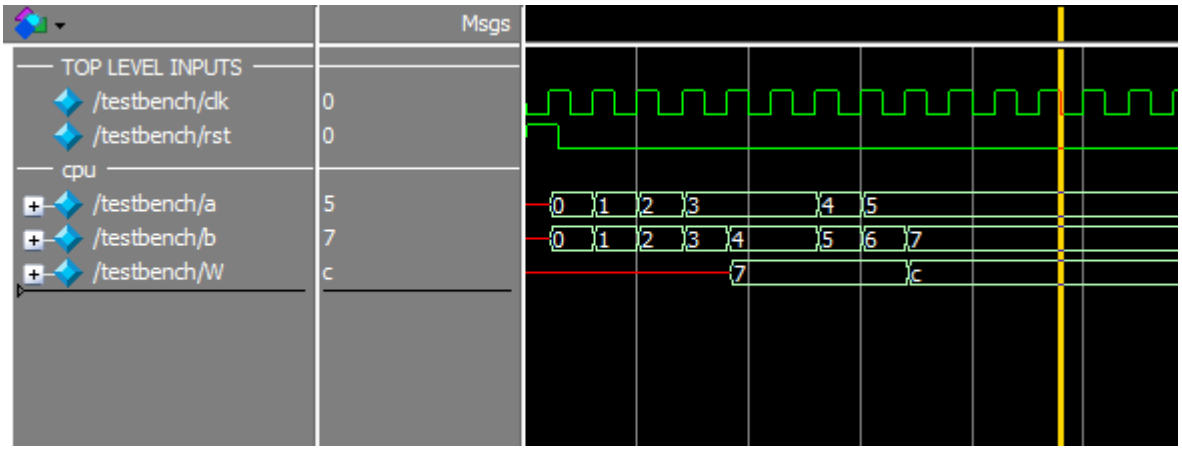
- Testbench.sv

```
module testbench;

    logic clk;
    logic rst;
    logic [3:0] a;
    logic [3:0] b;
    logic [3:0] W;

    counter_wait_plus counter_wait_plus_test(
        .clk(clk),
        .reset(rst),
        .a(a),
        .b(b),
        .W(W)
    );

    always #10 clk = ~clk;
    initial begin
        rst = 1; clk = 0;
        #15 rst = 0;
        #1000 $stop;
    end;

endmodule
```

說明：

設置 clk 每 10 個單位轉向，而一開始設 reset 為 1，clk 為 0 來做初始化。經過 15 秒後 reset 設回 0 開始計數。

■ 模擬結果與結果說明：



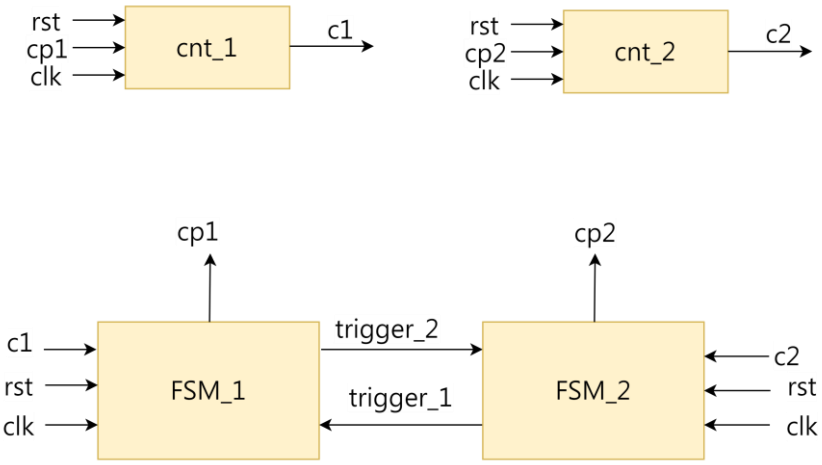
可看 a 從 0~3 後停止，

B 從 0~4 後輸出 7 在 W。

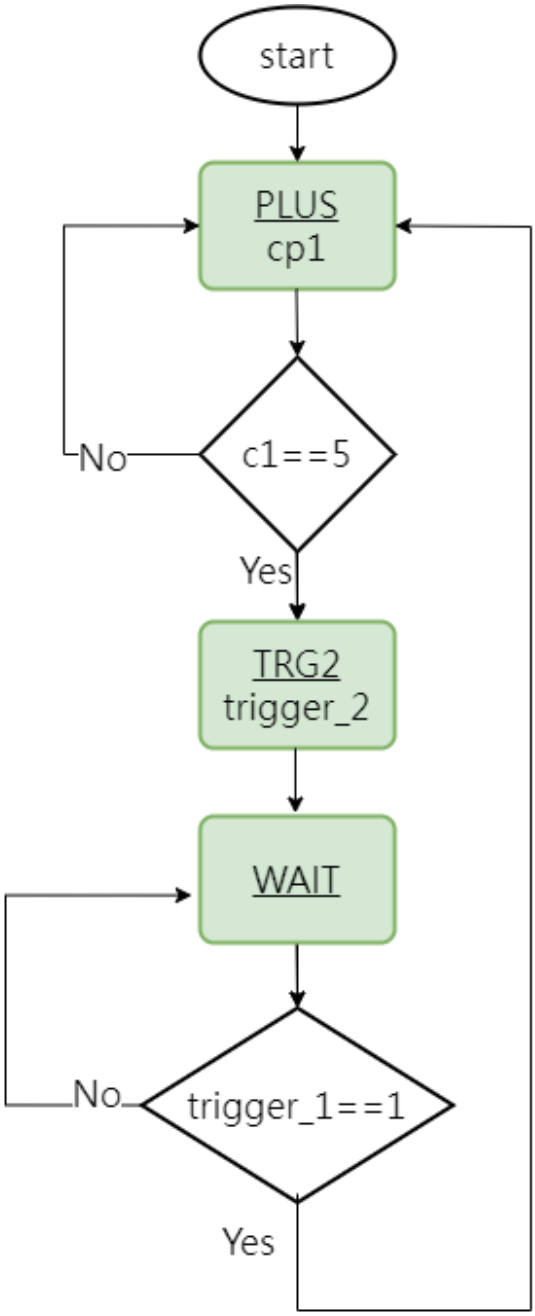
後續再繼續數 a、b counter，a 數 5 停止、b 數到 7 停止，最後輸出 12 在 W。

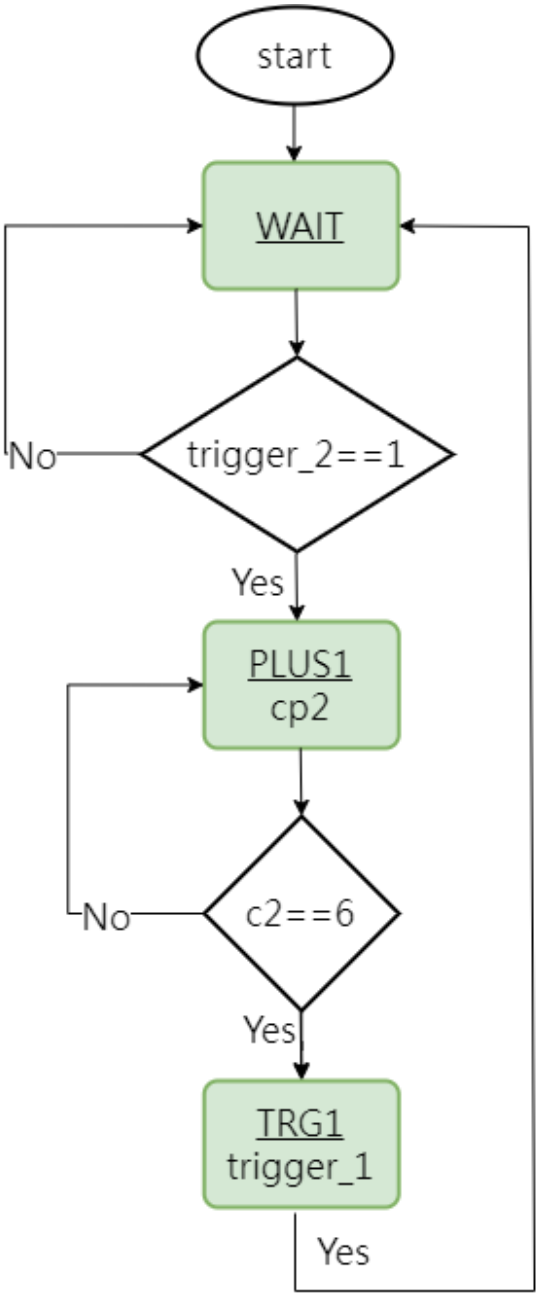
二、 Handshaking

- 如下圖所示， 2 個 4 bits counter (cnt_1, cnt_2)和 2 個 FSM。



- 請用 systemverilog 設計此電路，及控制之 FSM ，FSM_1, FSM_2 之流程圖如下所示：


start
PLUS
cp1
c1==5
No
Yes
TRG2
trigger_2
WAIT
trigger_1==1
No
Yes


start
WAIT
trigger_2==1
No
Yes
PLUS1
cp2
c2==6
No
Yes
TRG1
trigger_1
Yes

STATE
OUT

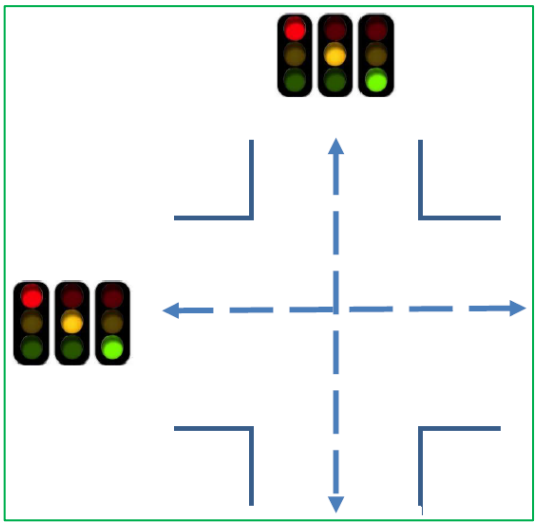
- **系統架構程式碼、測試資料程式碼與程式碼說明(.sv 檔及.do 檔都要截圖)**
  截圖請善用 win+shift+S

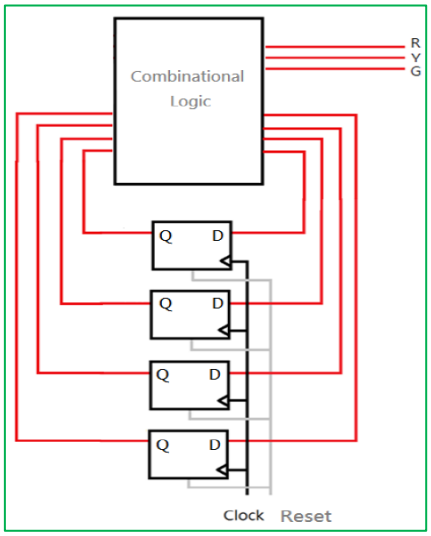
三、 紅綠燈

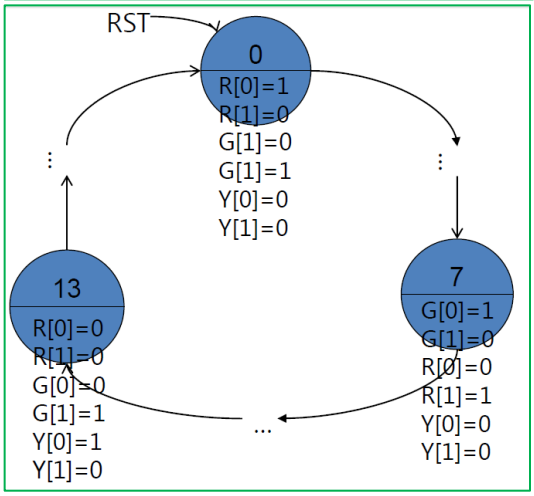
■ 實驗說明：

1. 用 FSM(Finite State Machine)實作紅綠燈
2. 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..
3. 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..
4. 以 R 表示紅燈、Y 表示黃燈、G 表示綠燈。
5. 1 表示燈亮，0 表示燈滅，最後輸出[1:0]R、[1:0]Y、[1:0]G。
6. 紅燈持續 8 個 clk ，綠燈持續 6 個 clk ，黃燈持續 2 個 clk ，16 個 clk 一次循環。

■ 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.sv

```
module traffic(
    input logic clk,
    input logic rst,
    output logic [1:0]r,
    output logic [1:0]y,
    output logic [1:0]g
    );

    logic [3:0] ps, ns;

    always_comb
    begin
        ns = ps + 1;
    end

    always_ff @(posedge clk)
    begin
        if (rst) begin
            ps <= 0;
        end else begin
            ps <= ns;
        end
    end

    always_comb
    begin
        case (ps)
            0,1,2,3,4,5: begin
                r[0] = 1;
                y[0] = 0;
                g[0] = 0;
                r[1] = 0;
                y[1] = 0;
                g[1] = 1;
            end
            6,7: begin
                r[0] = 1;
                y[0] = 0;
                g[0] = 0;
                r[1] = 0;
                y[1] = 1;
                g[1] = 0;
            end
```

```
            8,9,10,11,12,13: begin
                r[0] = 0;
                y[0] = 0;
                g[0] = 1;
                r[1] = 1;
                y[1] = 0;
                g[1] = 0;
            end
            14,15: begin
                r[0] = 0;
                y[0] = 1;
                g[0] = 0;
                r[1] = 1;
                y[1] = 0;
                g[1] = 0;
            end
            default: begin
            end
        endcase
    end

endmodule
```

系統架構的部分，輸入有 clk 與 reset 控制 state，RYG 分別宣告[1:0]代表兩個紅綠燈。

設定好 state，使 state 與題目紅綠燈閃爍方式吻合。

- testbench.sv

```
module testbench;

    logic clk;
    logic rst;
    logic [1:0] r,y,g;


    traffic t1(
    .clk(clk),
    .rst(rst),
    .r(r),
    .y(y),
    .g(g)
    );

    always #10 clk = ~clk;

    initial begin
        rst = 1; clk = 0;
        #15 rst = 0;
        #1000 $stop;
```

```
    end
endmodule
```

- compile.do

```
#vlib work



# ----------------------------------------------------------------
vlog ../tb/testbench.sv
vlog ../../design/sub_4bit.sv
vlog ../../design/DE0_CV.sv
vlog ../../design/traffic.sv









# ----------------------------------------------------------------

```

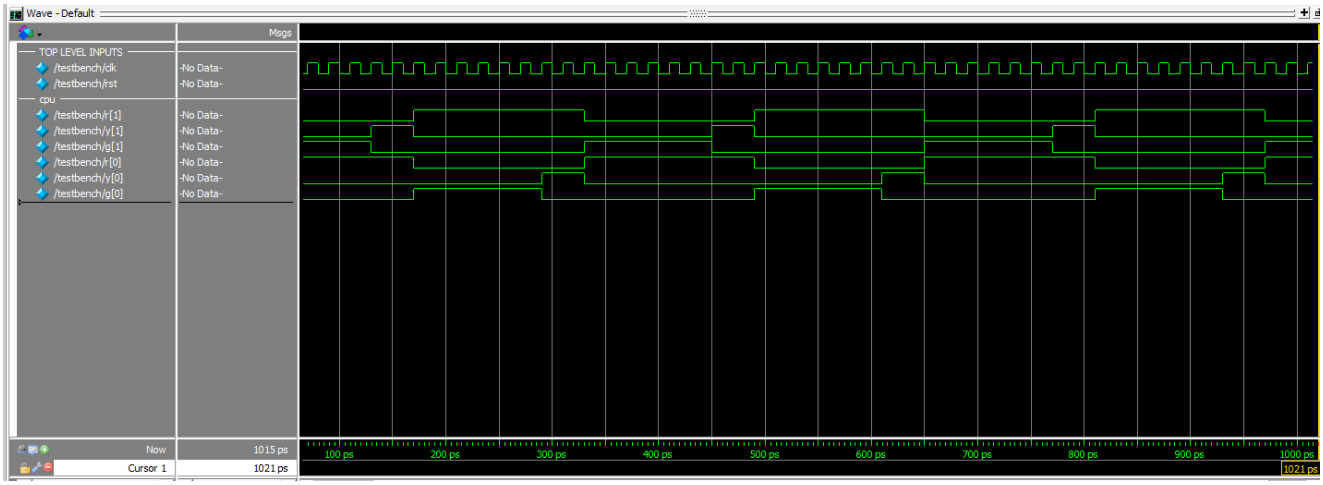
說明：compile traffic.sv。

- Wave.do

```
add wave -noupdate -divider {TOP LEVEL INPUTS}
add wave -noupdate /testbench/clk
add wave -noupdate /testbench/rst
add wave -noupdate -divider cpu
add wave -noupdate {/testbench/r[1]}
add wave -noupdate {/testbench/y[1]}
add wave -noupdate {/testbench/g[1]}
add wave -noupdate {/testbench/r[0]}
add wave -noupdate {/testbench/y[0]}
add wave -noupdate {/testbench/g[0]}

```

說明：將 clk 與 rst 與第一盞紅綠、第二盞紅綠燈呈現在模擬圖中。

■ 模擬結果與結果說明：



上方為第一盞紅綠燈的閃爍情況，下方為第二盞紅綠燈的閃爍情況。

■ 結論與心得：

這次的實驗中我做了紅綠燈的應用，與利用 FSM 去控制兩個 counter 是否要相加，是否要停止，我覺得實驗很有趣，希望下次的實驗也能這麼有趣。