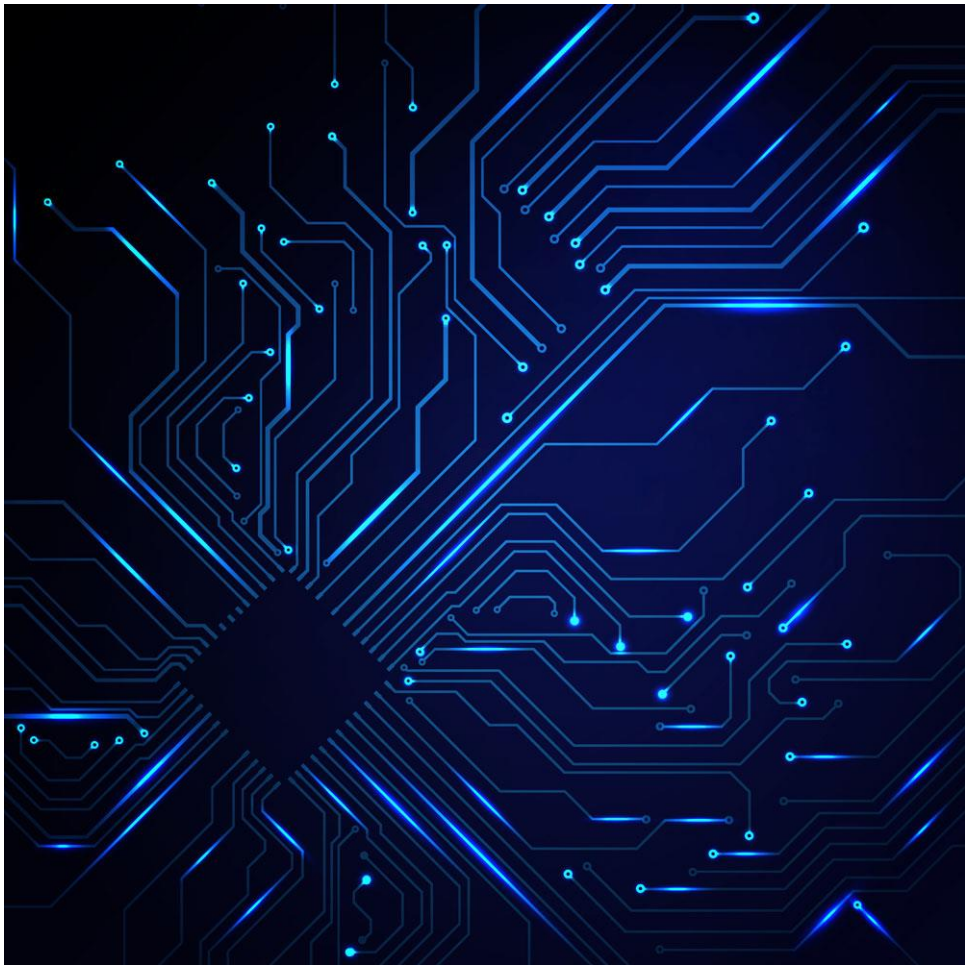


2022/10/05

實驗四

序向邏輯練習

姓名：王嘉羽　　學號：00957116

班級：資工 3B

E-mail：vayne20011125@gmail.com

注意

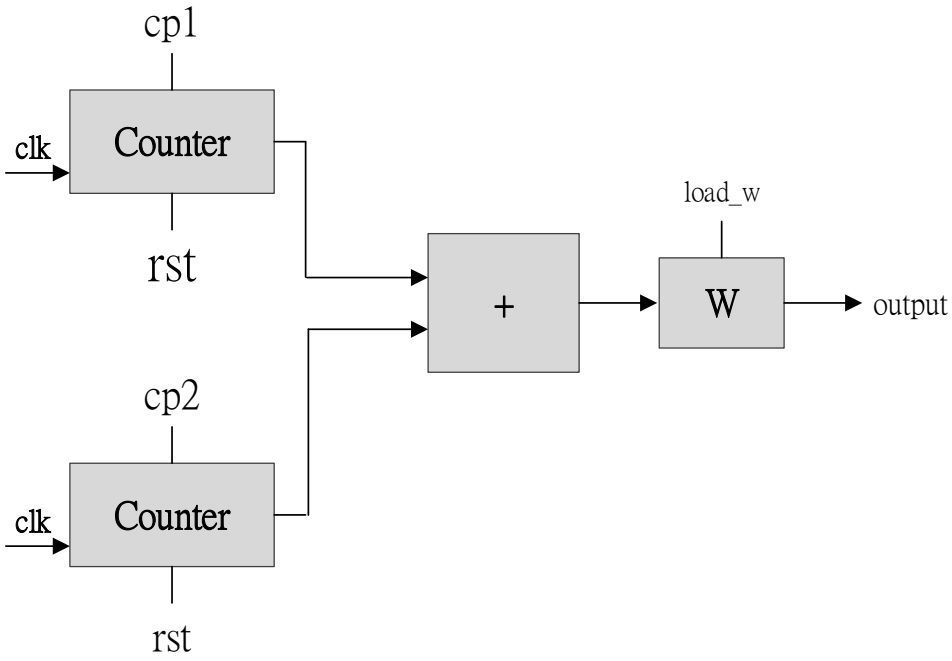
1. 繳交時一律轉 PDF 檔
2. 繳交期限為
   上完課後
   當週五晚上 12 點前
3. 一人繳交一份
4. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

一、 Counter + register

● 實驗說明：

設計 2 個 controller 和 1 個加法器，將其中一個 counter 數到 9，另一個 counter 數到 4，然後相加，最後存入一個暫存器 w 中，輸出暫存器 w 的值

● 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(.sv 檔及.do 檔都要截圖)

截圖請善用 win+shift+S

➢ 硬體程式碼

```
`timescale 1ns/10ps
module counter_reg(
    input clk,                  //時脈
    input reset,                //reset
    input load_w,               //載入控制
    output logic[3:0] q         //輸出
);

    logic cp1,cp2;              //是否計數
    logic [3:0] a,b;            //暫存值
    logic [1:0] ps,ns;          //現在狀態 下一個狀態

    always_ff @(posedge clk)        //fsm
    begin
       if(reset)
          ps <= #1 0;
       else
          ps <= #1 ns;
    end

    always_ff @(posedge clk)       //counter1
    begin
       if(reset) a <= #1 0;
       else if(cp1) a <= #1 a + 1;
    end

     always_ff @(posedge clk)       //counter2
     begin
        if(reset) b <= #1 0;
        else if(cp2) b <= #1 b + 1;
     end

     parameter STATE_TOGTHER = 0;   //一起計數
     parameter STATE_CNT1 = 1;      //只數cnt1
     parameter STATE_STOP = 2;      //暫停

    always_comb
    begin
       cp1 = 0; cp2 = 0;
       case(ps)
          STATE_TOGTHER:           //一起計數
          begin
          if(b == 4'd4)            //當cnt2數到4 ，就只數cnt1
                begin
                   cp1 = 1;
                   ns = STATE_CNT1;
                end
          else                     //不然就一起數
                begin
                   ns = STATE_TOGTHER;
                   cp1 = 1;
                   cp2 = 1;
                end
          end

          STATE_CNT1:               //只數cnt1
          begin
             if(a == 4'd9) ns = STATE_STOP;
             else
                begin
                   ns = STATE_CNT1;
                   cp1 = 1;
                end
          end

           STATE_STOP:              //不數了
           ns = STATE_STOP;

       endcase
    end

    always_ff @(posedge clk)
    begin
       if(reset)
          q <= 0;
       else if(load_w == 1)
          q <= a+b;
       else
          q <= q;
    end
 endmodule
```

- testbench

simulation > tb > testbench.sv

```
1  `timescale 1ns/10ps
2  module testbench;
3
4      logic reset;                    //重置
5      logic clk;                      //時脈
6      logic [3:0] q;                  //輸出
7      logic load_w;
8
9      counter_reg counter_reg1(
10         .reset(reset), //()內的變數為tb的變數，"."後面為counter_reg.sv的變數
11         .clk(clk),
12         .q(q),
13         .load_w(load_w)
14     );
15
16     always #10 clk = ~clk;
17
18     initial begin
19         reset = 1;clk = 0;load_w = 1; //一開始先reset，將時脈歸0
20         #15 reset = 0;
21         #1000 $stop;
22     end
23 endmodule
```

- compile.do

simulation > modelsim > compile.do

```
1   #vlib work
2
3
4
5   # ----------------------------------------------------------
6   vlog ../tb/testbench.sv
7   vlog ../../design/counter_reg.sv
8
9
10
11
12
13  # ----------------------------------------------------------
14
```

- sim.do

simulation > modelsim > sim.do

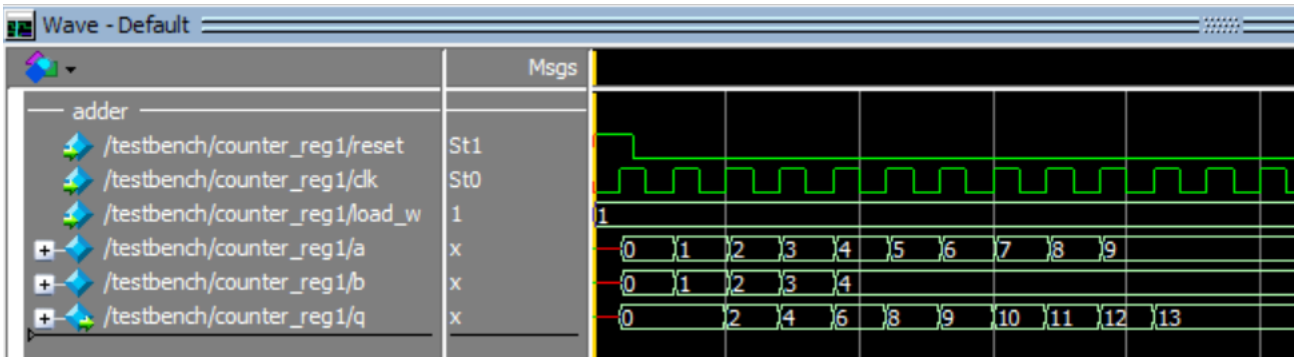
```
1   vsim -voptargs=+acc work.testbench
2   view structure wave signals
3
4   do wave.do
5
6   log -r *
7   run -all
8
```

➢ wave.do

simulation > modelsim > wave.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0

#add wave -noupdate -divider {TOP LEVEL INPUTS}

#add wave -noupdate -format Logic /testbench/clk
#add wave -noupdate -format Logic /testbench/rst



add wave -noupdate -divider {adder}

add wave -noupdate -format logic           /testbench/counter_reg1/reset
add wave -noupdate -format logic           /testbench/counter_reg1/clk
add wave -noupdate -format Literal -radix Unsigned  /testbench/counter_reg1/load_w
add wave -noupdate -format Literal -radix Unsigned  /testbench/counter_reg1/a
add wave -noupdate -format Literal -radix Unsigned  /testbench/counter_reg1/b
add wave -noupdate -format Literal -radix Unsigned  /testbench/counter_reg1/q
```

● **模擬結果與結果說明：**



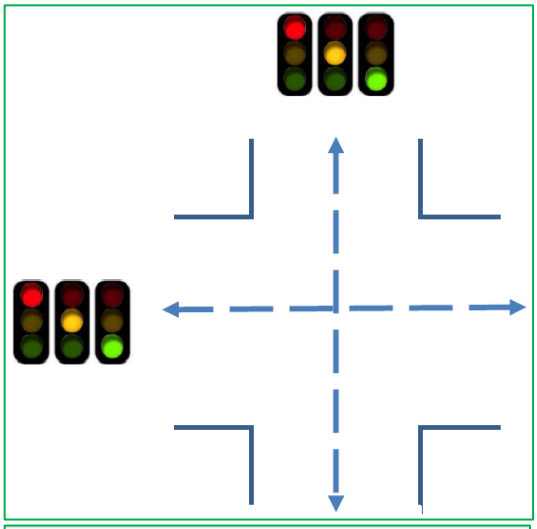
a(cnt1)從 0~9，b(cnt2)從 0~4 然後加起來，當 load_w = 1 時，會將上一次的結果載入

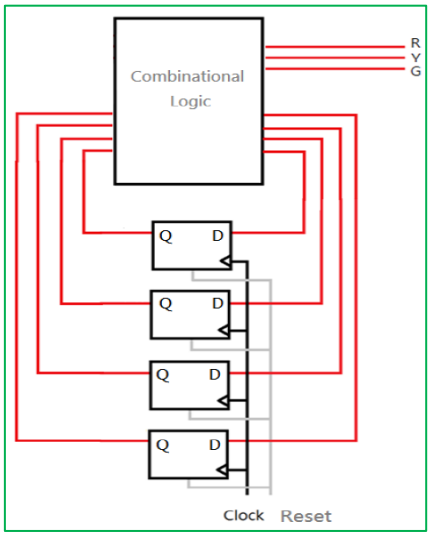
二、 紅綠燈

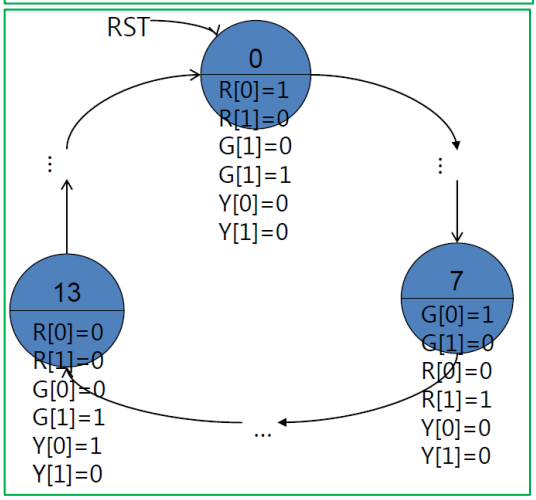
● 實驗說明：

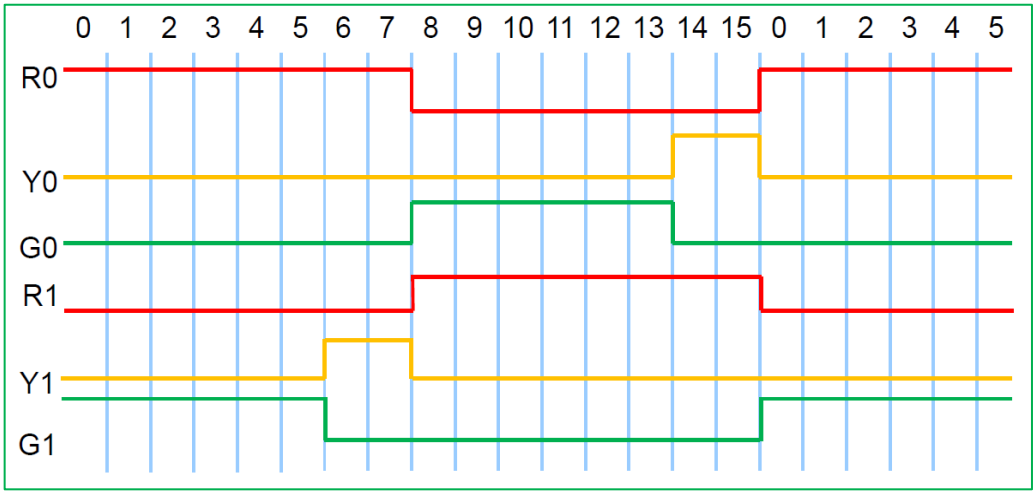
1. 用 FSM(Finite State Machine)實作紅綠燈
2. 第一組紅綠燈(R[0], Y[0], G[0]) 由紅燈為起點依序變換為 綠燈→黃燈..
3. 第二組紅綠燈(R[1], Y[1], G[1]) 根據地一組紅綠燈的狀態顯示 綠燈→黃燈→紅燈..
4. 以 R 表示紅燈、Y 表示黃燈、G 表示綠燈。
5. 1 表示燈亮，0 表示燈滅，最後輸出[1:0]R、[1:0]Y、[1:0]G。

● 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(.sv 檔及.do 檔都要截圖)

截圖請善用 win+shift+S

✧硬體程式碼

```
`timescale 1ns/10ps
module RYG_light(
    input clk,                  //時脈
    input reset,                //reset
    output logic[1:0] R,        //紅燈
    output logic[1:0] Y,        //黃燈
    output logic[1:0] G         //綠燈
);

    logic [3:0] ps,ns;          //現在狀態 下一個狀態
    logic [5:0] tp;             //燈的變化

    always_ff @(posedge clk)        //fsm
    begin
       if(reset)
          ps <= #1 0;
       else
          ps <= #1 ns;
    end

    always_comb
    begin
    case(ps)            //每一種狀態對應到不同燈
          0:
          begin
             ns = 1;
             tp = 6'b100001; //對應到R[0]Y[0]G[0]R[1]Y[1]G[1]
          end

          1:
          begin
             ns = 2;
             tp = 6'b100001;
          end

          2:
          begin
             ns = 3;
             tp = 6'b100001;
          end

          3:
          begin
             ns = 4;
             tp = 6'b100001;
          end
```

```
47  
48          4:
49          begin
50              ns = 5;
51              tp = 6'b100001;
52          end
53  
54          5:
55          begin
56              ns = 6;
57              tp = 6'b100001;
58          end
59  
60          6:
61          begin
62              ns = 7;
63              tp = 6'b100010;
64          end
65  
66          7:
67          begin
68              ns = 8;
69              tp = 6'b100010;
70          end
71  
72          8:
73          begin
74              ns = 9;
75              tp = 6'b001100;
76          end
77  
78          9:
79          begin
80              ns = 10;
81              tp = 6'b001100;
82          end
83  
84          10:
85          begin
86              ns = 11;
87              tp = 6'b001100;
88          end
89  
90          11:
91          begin
92              ns = 12;
93              tp = 6'b001100;
94          end
95  
96          12:
97          begin
98              ns = 13;
99              tp = 6'b001100;
100         end
101 
102         13:
103         begin
104             ns = 14;
105             tp = 6'b001100;
106         end
107 
108         14:
109         begin
110             ns = 15;
111             tp = 6'b010100;
112         end
113 
114         15:
115         begin
116             ns = 0;
117             tp = 6'b010100;
118         end
119     endcase
120 end
121 
122 assign R[0] = tp[5];   //將tp的值給output
123 assign Y[0] = tp[4];
124 assign G[0] = tp[3];
125 assign R[1] = tp[2];
126 assign Y[1] = tp[1];
127 assign G[1] = tp[0];
128 
129 endmodule
```

✧testbench

simulation > tb > testbench.sv

```
`timescale 1ns/10ps
module testbench;

    logic reset;                    //重置
    logic clk;                      //時脈
    logic [1:0] R,Y,G;              //輸出

    RYG_light RYG_light1(
        .reset(reset), //()內的變數為tb的變數，"."後面為RYG_light.sv的變數，將2者對應起來
        .clk(clk),
        .R(R),
        .G(G),
        .Y(Y)
    );

    always #10 clk = ~clk;

    initial begin
        reset = 1;clk = 0; //一開始先reset，將時脈歸0
        #15 reset = 0;
        #1000 $stop;
    end
endmodule
```

✧wave.do

simulation > modelsim > wave.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0

#add wave -noupdate -divider {TOP LEVEL INPUTS}

#add wave -noupdate -format Logic /testbench/clk
#add wave -noupdate -format Logic /testbench/rst


add wave -noupdate -divider {adder}

add wave -noupdate -format logic    /testbench/RYG_light1/reset
add wave -noupdate -format logic    /testbench/RYG_light1/clk
add wave -noupdate -format logic    /testbench/RYG_light1/R\[0\]
add wave -noupdate -format logic    /testbench/RYG_light1/Y\[0\]
add wave -noupdate -format logic    /testbench/RYG_light1/G\[0\]
add wave -noupdate -format logic    /testbench/RYG_light1/R\[1\]
add wave -noupdate -format logic    /testbench/RYG_light1/Y\[1\]
add wave -noupdate -format logic    /testbench/RYG_light1/G\[1\]
```

✧Sim.do

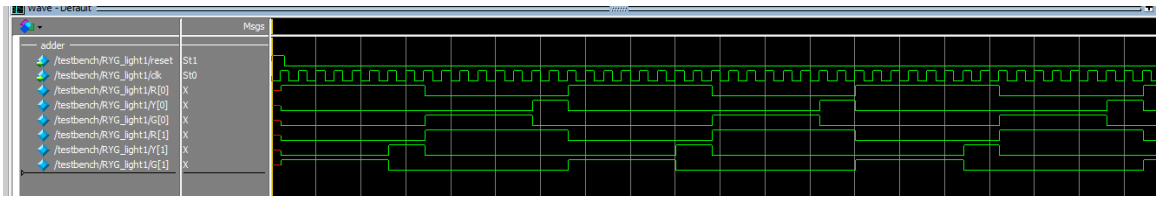
simulation > modelsim > sim.do

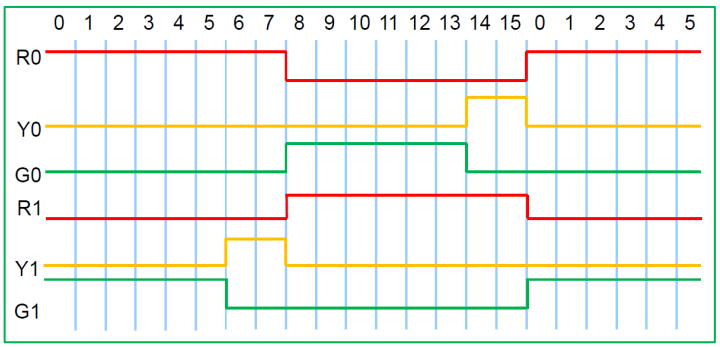
```
vsim -voptargs=+acc work.testbench
view structure wave signals

do wave.do

log -r *
run -all
```

✧Compile.do

```
simulation > modelsim > compile.do
1   #vlib work
2
3
4
5   # ----------------------------------------------------------
6   vlog ../tb/testbench.sv
7   vlog ../../design/RYG_light.sv
8
```

● 模擬結果與結果說明：





和模擬圖長得一模一樣!!!

● 結論與心得：

寫了紅綠燈後更加了解 fsm 的強大之處，如果沒有用 fsm，code 一定會很複雜。Fsm 很單純的就是把狀態寫出來，然後照著狀態給值，就成功了!這個單元也是我最喜歡的單元，我覺得很好玩!!