

國立中山大學電機工程系
實用數位系統設計

Homework 3

學生：徐崇皓
學號：C110152334（外校）

一、演算法

i. 演算法介紹：

有兩條交會的道路分別為高速公路（Highway）與農村道路（Farmroad）。高速公路常常有大量的車潮經過，而農村道路則鮮少有車經過。因此，我們在農村道路的紅綠燈前新增了一個感測器(c)。假設要切換成農村綠燈時發現農村路口沒有車(!c)，我們就無需切換成農村的綠燈狀態，增加路口的使用效率。在此路口增加了一個行人的時相，當高速公路執行綠燈 n 次(default 5)後，就會轉為行人時相（若同時發生 Farmroad 有車時，行人權重大於 Farmroad 會優先執行，這種做法可以避免當外部感測器 c 發生錯誤時，在狀態為 S0 與 S2 時測器偵測不一致時所造成的錯誤）。這個時相會讓高速公路與農村道路同時停止供行人行走。切換道路時會產生一些狀態，如黃燈與全部紅燈。黃燈表示該道路時相即將結束，通常在黃燈後會設計一個全部紅燈的時相，用來等待路面淨空供下一個時相運行的汽車使用。由於行人沒有黃燈，所以當綠燈結束後直接進入紅燈。

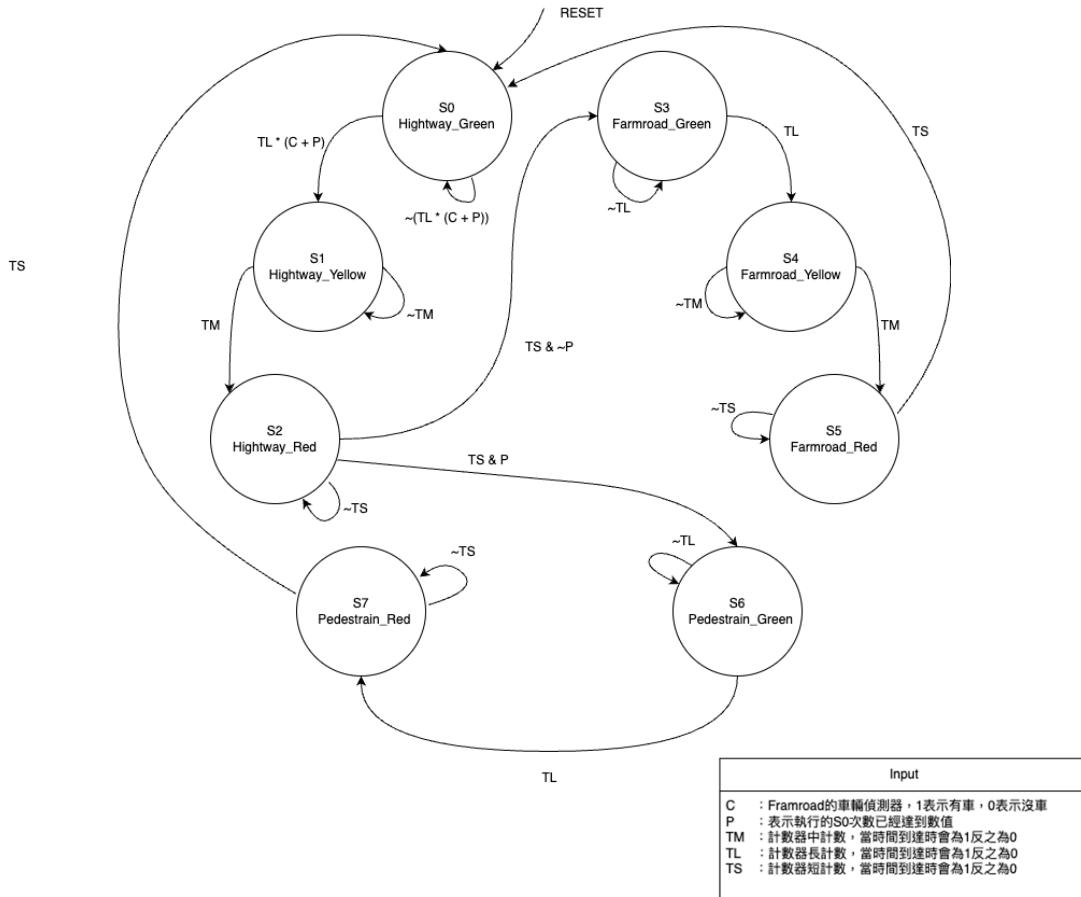
state	描述	Highway	Farmroad	Pedestrian
S0	Highway 綠燈	G	R	R
S1	Highway 黃燈	Y	R	R
S2	Highway 黃燈後開始的全紅燈狀態	R	R	R
S3	Farmroad 綠燈	R	G	R
S4	Farmroad 黃燈	R	Y	R
S5	Farmroad 黃燈後開始的全紅燈狀態	R	R	R
S6	Pedestrian 綠燈	R	R	G
S7	Pedestrian 綠燈後開始的全紅燈狀態	R	R	R

ii. 狀態激勵：

全部狀態接為收到計數器激勵（TS、TM 與 TL）才做狀態轉移（不包含 reset）下表描述不包含計數器激勵，如果未達激勵條件則保持原本狀態。

CN	NS	激勵條件	描述
S0	S1	TL*(C+P)	當偵測到 Farmroad 有車或 S0 執行次數到達 n 次 (default 5) 時，Highway 綠燈轉紅燈
S1	S2	TM	Highway 黃燈轉 Highway 全紅燈
S2	S6	TS*(P)	當偵測到 S0 執行次數到達 n 次 (default 5) 時，Highway 全紅燈狀態改成行人綠燈。
S2	S3	TS*(~P)	當偵測到 Farmroad 有車且 S0 執行次數未達到 n 次 (default 5) 時，Highway 紅燈轉 Farmroad 綠燈。
S3	S4	TL	Farmroad 綠燈轉 Farmroad 黃燈。
S4	S5	TM	Farmroad 黃燈轉 Farmroad 全紅燈。
S5	S0	TS	Farmroad 全紅燈轉 Highway 綠燈。
S6	S7	TL	行人綠燈轉行人紅燈。
S7	S0	TS	行人紅燈轉 Highway 綠燈。

iii. 有限狀態機 (Moore Machine) :



iv. 狀態對應輸出：

state/output	Highway			Farmroad			Pedestrian	
	HG	HY	HR	FG	FY	FR	PG	PR
S0	1	0	0	0	0	1	0	1
S1	0	1	0	0	0	1	0	1
S2	0	0	1	0	0	1	0	1
S3	0	0	1	1	0	0	0	1
S4	0	0	1	0	1	0	0	1
S5	0	0	1	0	0	1	0	1
S6	0	0	1	0	0	1	1	0
S7	0	0	1	0	0	1	0	1

二、電路架構

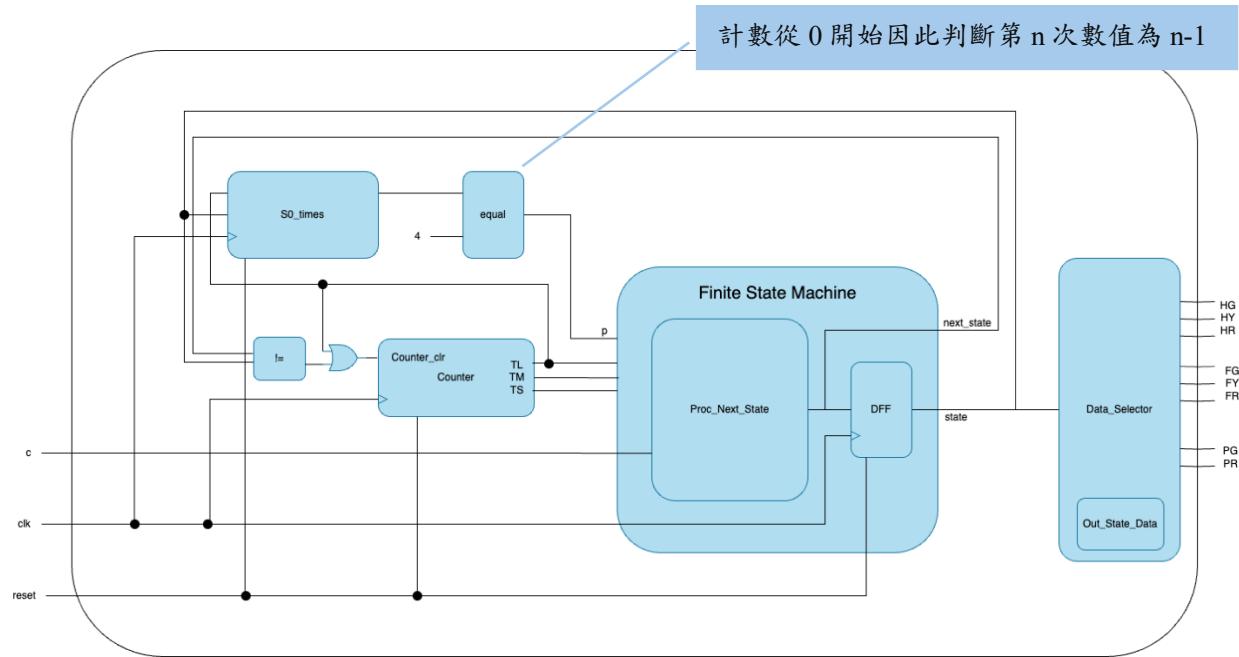
i. I/O SPEC

Port Name	Direction	Description
clk	I	系統時鐘
reset	I	非同步 reset，正緣觸發
c	I	Farmroad 車輛偵測器
HG	O	Highway 綠燈
HY	O	Highway 黃燈
HR	O	Highway 紅燈
FG	O	Farmroad 綠燈
FY	O	Farmroad 黃燈
FR	O	Farmroad 紅燈
PG	O	行人綠燈
PR	O	行人紅燈

ii. Top module

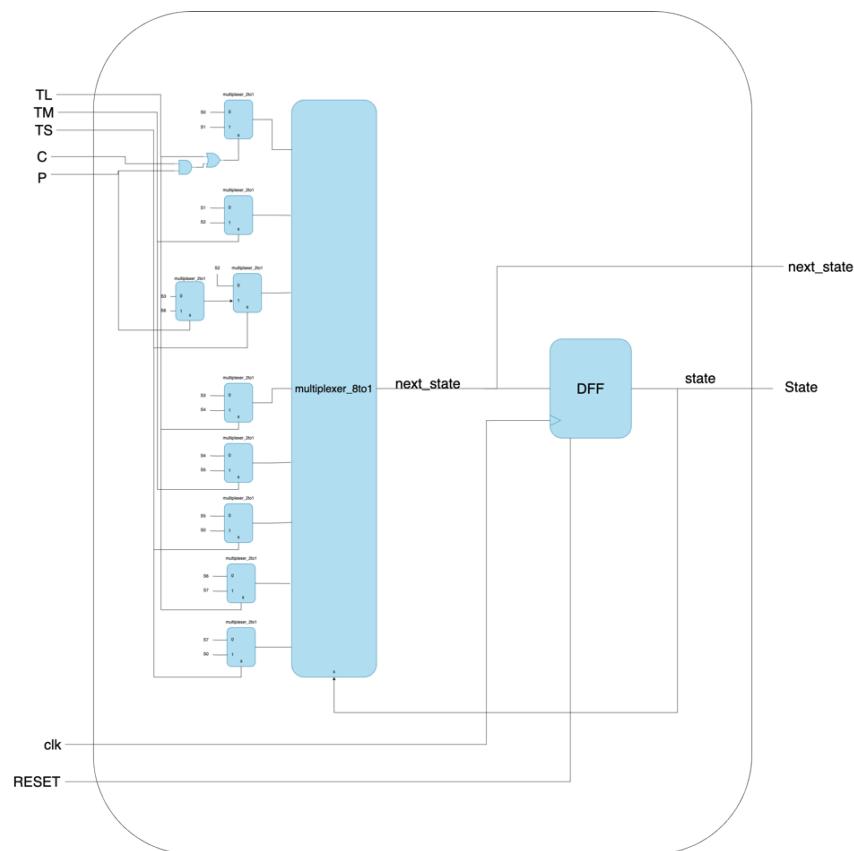
系統組成主要包含四個部分：FSM、Counter、S0_times 和 Data_Selector。

1. FSM 控制了整個系統的狀態並且狀態輸出給 Counter 和 S0_times 以進行歸零。
2. Counter 有兩種情況會導致歸零：一是當狀態準備轉換 ($state \neq next_state$)，另一種是當 S0 到達時間但未轉換狀態 ($state == S0 \& TL$)。由於 TL 只會在 S0、S3 和 S6 上發生，而 S3 和 S6 遇到 TL 後必定會轉換狀態，因此發生 TL Counter 一定會歸零。因此，我們可以將 ($state == S0 \& TL$) 簡化為 TL。將這兩種情況組合起來，可以得到 counter_clr 的布林函數：
 $(state \neq next_state) \& TL$ 。Counter 輸出 TL、TM 和 TS 以輸入 FSM。
3. S0_times 模組用於計數 S0 的發生次數，當 S0 發生至 n 次時，p 將拉高以通知 FSM 切換到行人的時相。
4. Data_Selector 負責將狀態對應的輸出輸出。



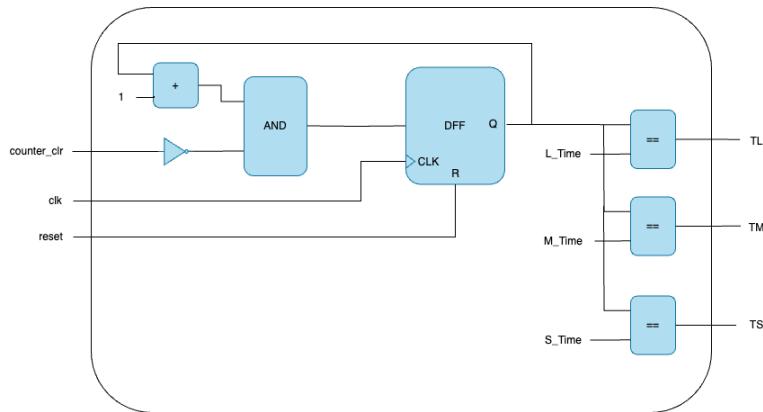
iii. Finite State Machine

由 multiplexer 選擇下 **next_state**，當下一個週期到來時就會將 **state** 更新 **next_state**，使用 Gray Code 作為狀態編碼可以降低 bit 的翻轉率減少動態功耗。



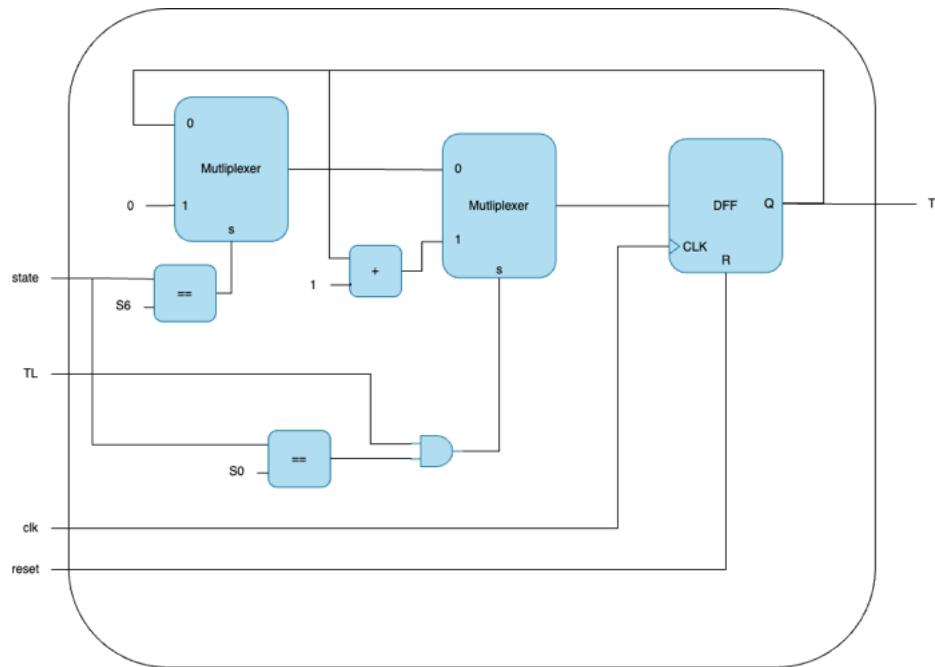
iv. Counter

當 counter_clr 發生會讓計數值歸零， $((\text{counter}+1) \text{ and } \sim \text{counter_clr})$ 作為下一個計數，當 counter_clr 為 1 時可以讓輸出為 0，當 counter_clr 為 0 時可以讓輸出為 $(\text{counter} + 1)$ ，當計數值到達預設的 L_Time、M_Time 或 S_Time 時，對應的輸出 TL、TM 或 TS 會拉為 HIGH。



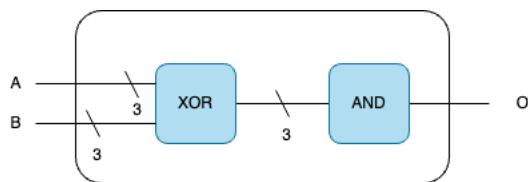
v. S0_times

判斷要不要計數發生在 S0 的最後一個時間，也就是發生在 TL 為 1 時($\text{state} == \text{S0}$ & TL)，當狀態 S6 時表示已到達行人狀態這時候我們將計數值歸零，若以上狀態皆為滿足，計數值將會維持現有狀態。



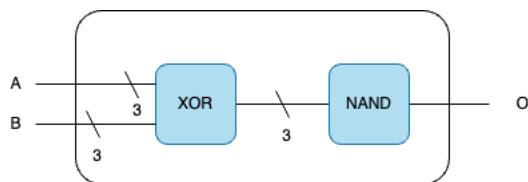
vi. Equal(==)

判斷兩個數值是否相等我們可以使用 XOR 判斷每個 bit 是否相等，從真值表得知 XOR 只會在相等時為 1，相異時為 0，最後再將所有值 AND 起來即可知道是否有值為相異。



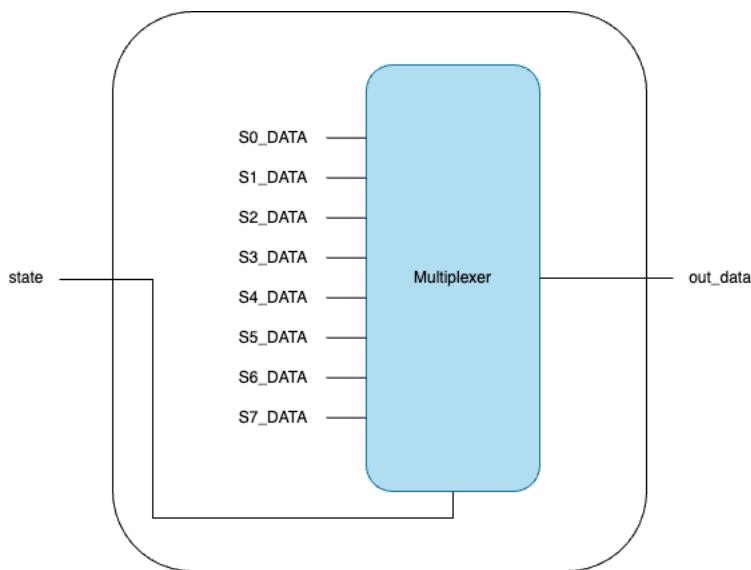
vii. Not_Equal(!=)

與 Equal 同理，只需要在輸出加上 NOT 即可，因此我們將輸出從 AND 改為 NAND。



viii. Date_Selector

使用 Multiplexer 選擇 state 相對應的輸出。



三、合成

i. 介紹

為了提高效率，我決定參考 IC Design Contest 提供的合成 TCL 腳本和 Constraints 檔，以及在網路上找到的資源，來進行本次的合成設計。GUI 的使用效率相對較低，因此我選擇了這種更高效的方式來進行合成。

ii. 合成指令

Command : dc_shell -f dc_syn.tcl

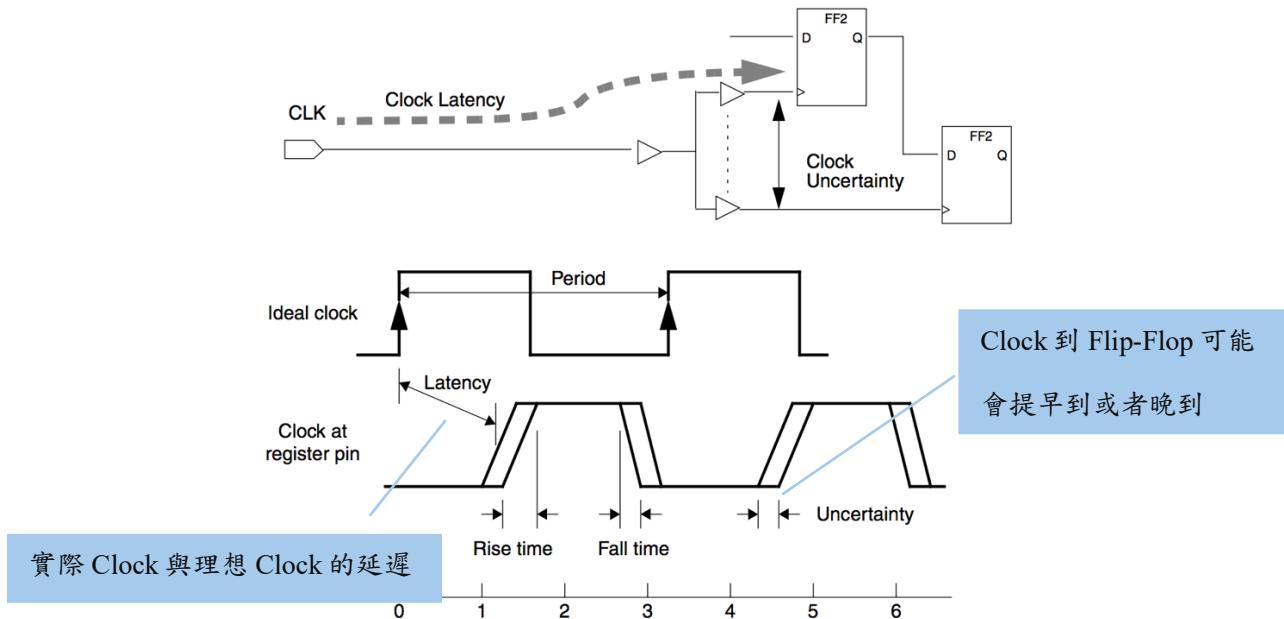
iii. Constraints

1. 設定時中週期為 6 ns。

- set cycle 6
- create_clock -period \$cycle [get_ports clk]

2. 設定 clock 的不穩定性與 clock 延遲。

- set_clock_uncertainty 1 [all_clocks]
- set_clock_latency 1 [all_clocks]



Source : http://web02.gonzaga.edu/faculty/talarico/CP430/LEC/synth_constraints.pdf

3. 讓 Design_Compiler 處理 Hold time Violation 。
 - set_fix_hold [all_clocks]

4. 設定輸入與輸出最大延遲與最小延遲，clock 被單獨拿出來處理所以這部分輸入須去除 clk 。
 - set_input_delay -max 1 -clock clk [remove_from_collection [all_inputs] [get_ports clk]]
 - set_input_delay -min 0 -clock clk [remove_from_collection [all_inputs] [get_ports clk]]
 - set_output_delay -max 1 -clock clk [all_outputs]
 - set_output_delay -min 0 -clock clk [all_outputs]

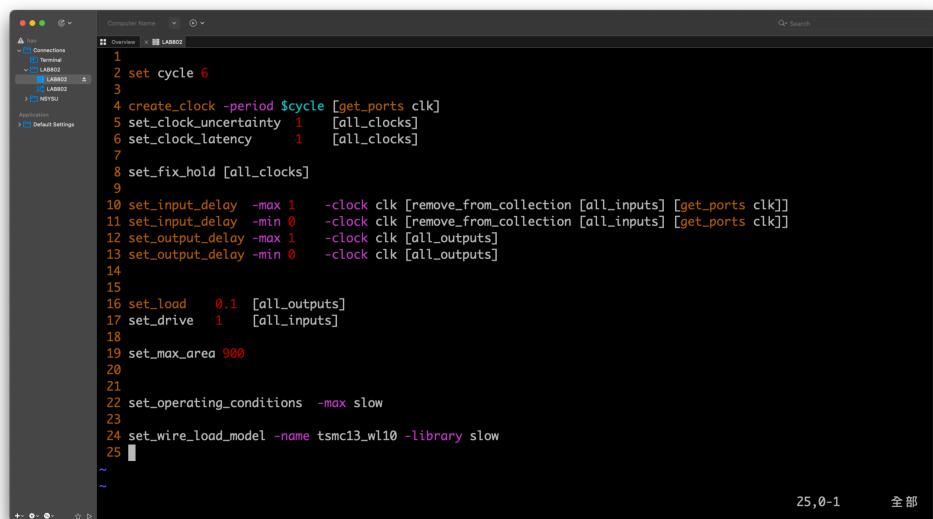
5. 設定輸出 Output load 。
 - set_load 0.1 [all_outputs]

6. 設定 Input driving strength 。
 - set_drive 1 [all_inputs]

7. 設定最大面積 (Design_Compiler 不一定能完成) 。
 - set_max_area 900

8. 設定操作環境為 slow (Temp : 125 Volt : 1.62) 。
 - set_operating_conditions -max slow

9. 設定每個線的種類
 - set_wire_load_model -name tsmc13_wl10 -library slow



```

1
2 set cycle 6
3
4 create_clock -period $cycle [get_ports clk]
5 set_clock_uncertainty 1 [all_clocks]
6 set_clock_latency 1 [all_clocks]
7
8 set_fix_hold [all_clocks]
9
10 set_input_delay -max 1 -clock clk [remove_from_collection [all_inputs] [get_ports clk]]
11 set_input_delay -min 0 -clock clk [remove_from_collection [all_inputs] [get_ports clk]]
12 set_output_delay -max 1 -clock clk [all_outputs]
13 set_output_delay -min 0 -clock clk [all_outputs]
14
15
16 set_load 0.1 [all_outputs]
17 set_drive 1 [all_inputs]
18
19 set_max_area 900
20
21
22 set_operating_conditions -max slow
23
24 set_wire_load_model -name tsmc13_wl10 -library slow
25
~ ~

```

25,0-1 全部

iv. Design Compiler 腳本

1. 統一將路徑與名稱設定成變數供之後腳本使用（方便之後修改路徑）。
 - set Design_Source ../src
 - set Project_Name traffic_light
2. 讀取電路檔案。
 - read_file -format verilog \$Design_Source/\$Project_Name.v
 - current_design \$Project_Name
 - link
3. 讀取 Constraint 檔。
 - source -echo -verbose \$Design_Source/\$Project_Name.sdc
 - check_design
4. 將直接輸出插入 Buffer Gate。
 - set_fix_multiple_port_nets -all -buffer_constants [get_designs *]
5. 合成。
 - compile
6. 設定輸出資料夾。
 - set Project_Name_syn [append Project_Name "_syn"]
 - file mkdir Results
 - file mkdir Reports
7. 儲存 SDF 延遲檔。
 - write_sdf -version 1.0 "./Results/\$Project_Name_syn.sdf"
8. 輸出合成後 netlist 檔。
 - write -format verilog -hierarchy -output "./Results/\$Project_Name_syn.v"
9. 儲存面積報告。
 - report_area > ./Reports/area.log
10. 儲存時序報告。
 - report_timing -delay_type min > ./Reports/timing_min.log
 - report_timing -delay_type max > ./Reports/timing_max.log
11. 儲存功耗報告。
 - report_power > ./Reports/power.log
12. 打開 GUI（若有需要可以開啟）。
 - gui_start
13. 結束。
 - quit

```
# File Name : dc_syn.tcl #
# Author   : ChongHs_Xu
# Description :
#
# Desing_Source#
# Circuit   :Circuit_Name.vc(.sv)
# Constraints :Constraint.sdc
#
#####
# Design_Source ./src
set Project_Name traffic.light
#####

##### Read Files #####
read_file -format verilog $Design_Source/$Project_Name.v
current_design $Project_Name
link

##### Setting Constraints #####
source -echo -verbose $Design_Source/$Project_Name.sdc
check_design
set_fix_multiple_port_nets -all -buffer_constants [get_designs]

##### Synthesis all design #####
compile

#####
# Write Out #####
set Project_Name_syn [append Project_Name "_syn"]
file mkdir Results
file mkdir Reports
write_sdf -version 1.0 "$Results/$Project_Name_syn.sdf"
write -format verilog -hierarchy -output "$Results/$Project_Name_syn.v"
report_area > ./Reports/area.log
report_timing -delay_type min > ./Reports/timing_min.log
report_timing -delay_type max > ./Reports/timing_max.log
report_power > ./Reports/power.log
```

Annotations:

- Red box highlights the first section of code: `read_file -format verilog $Design_Source/$Project_Name.v`. To its right is the label "讀取 Verilog 檔案" (Read Verilog file).
- Red box highlights the second section of code: `source -echo -verbose $Design_Source/$Project_Name.sdc`. To its right is the label "讀取 電路 Constraints" (Read circuit Constraints).
- Red box highlights the third section of code: `compile`. To its right is the label "讀取 電路 Constraints" (Read circuit Constraints).
- Red box highlights the final section of code: `report_area > ./Reports/area.log`, `report_timing -delay_type min > ./Reports/timing_min.log`, `report_timing -delay_type max > ./Reports/timing_max.log`, and `report_power > ./Reports/power.log`. To its right is the label "儲存合成結果" (Save synthesis results).

四、電路分析

i. Area Report

1. Total Cell Area : $813.054588 \mu m^2$
2. Gate Count. : 160 $(\text{int}(813.054588 / 5.0922) \cong 160)$

```
dc_shell> report_area
*****
Report : area
Design : traffic_light
Version: Q-2019.12
Date   : Sun May 12 20:52:43 2024
*****

Library(s) Used:
    slow (File: /usr/cad/designkit/CBDK_IC_Contest_v2.1/SynopsysDC/db/slow.db)

Number of ports:          11
Number of nets:           81
Number of cells:          69
Number of combinational cells: 58
Number of sequential cells: 11
Number of macros/black boxes: 0
Number of buf/inv:         15
Number of references:      33

Combinational area:        453.205799
Buf/Inv area:              71.290798
Noncombinational area:     359.848789
Macro/Black Box area:      0.000000
Net Interconnect area:     8213.374573

Total cell area:           813.054588
Total area:                9026.429161
1
```

ii. Timing Report

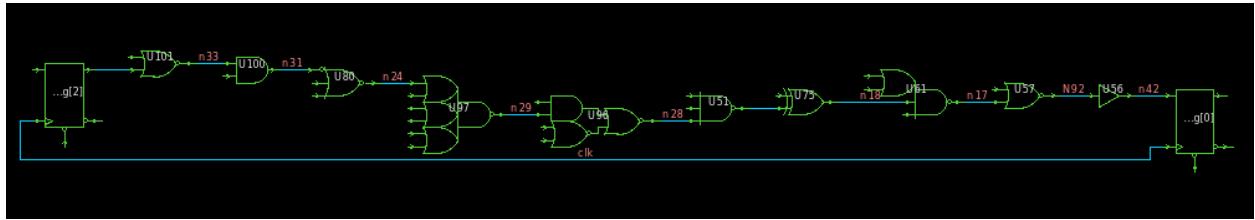
1. Setup timing report

Report : timing -path full -delay max -max_paths 1			
Design : traffic_light Version: Q-2019.12 Date : Sun May 12 20:58:32 2024 *****			
Operating Conditions: slow Library: slow Wire Load Model Mode: top			
Startpoint: counter_reg[2] (rising edge-triggered flip-flop clocked by clk) Endpoint: counter_reg[0] (rising edge-triggered flip-flop clocked by clk) Path Group: clk Path Type: max			
Des/Clust/Port	Wire Load Model	Library	
-----	-----	-----	
traffic_light	tsmc13_wl10	slow	Constraint 產生的 clock 延遲
Point	Incr	Path	
-----	-----	-----	
clock_clk (rise edge)	0.00	0.00	
clock network delay (ideal)	1.00	1.00	
counter_reg[2]/CK (DFFRX1)	0.00	1.00 r	
counter_reg[2]/Q (DFFRX1)	0.63	1.63 f	
U101/Y (NOR2X1)	0.23	1.86 r	
U100/Y (AND2X2)	0.28	2.15 r	
U80/Y (NOR3BX1)	0.74	2.88 r	
U97/Y (OAI222XL)	0.42	3.30 f	
U96/Y (AOI2BB2X1)	0.35	3.65 r	
U51/Y (NAND3XL)	0.41	4.06 f	
U75/Y (XOR2X1)	0.37	4.43 r	
U61/Y (OAI211X1)	0.59	5	Constraint 產生的 clock 延遲
U57/Y (NOR2XL)	0.52	5	與不穩定性
U56/Y (BUFX2)	0.22	5	
counter_reg[0]/D (DFFRX1)	0.00	5.76	
data arrival time			
-----	-----	-----	
clock_clk (rise edge)	6.00	6.00	
clock network delay (ideal)	1.00	7.00	
clock uncertainty	-1.00	6.00	
counter_reg[0]/CK (DFFRX1)	0.00	6.00 r	
library setup time	-0.23	5.77	
data required time		5.77	
-----	-----	-----	
data required time		5.77	
data arrival time		-5.76	
-----	-----	-----	
slack (MET)	0.00		

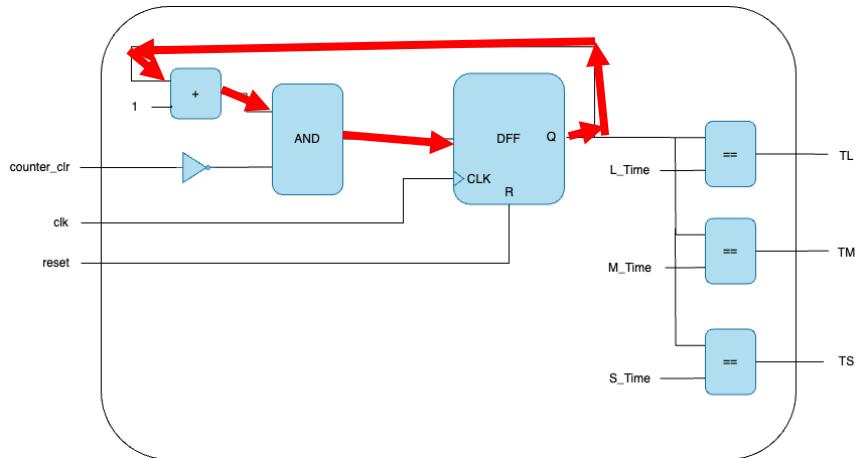
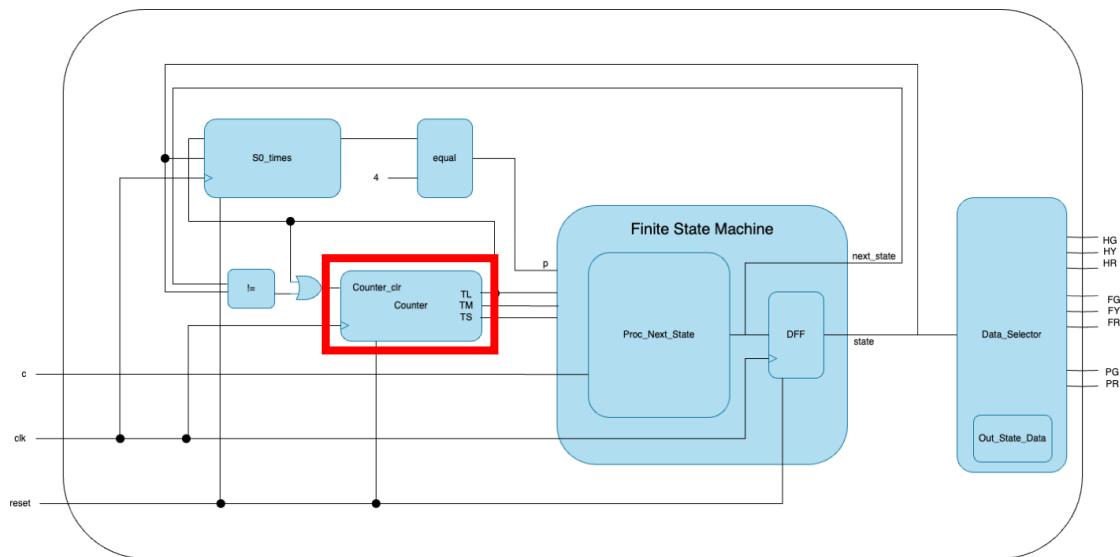
Slack 為 0 代表最大路徑延遲剛好符合 setup time 的最低要求

2. Critical Path (Setup)

合成後最大延遲路徑如下圖。



根據路徑推斷合成前最大延遲路徑如下圖。



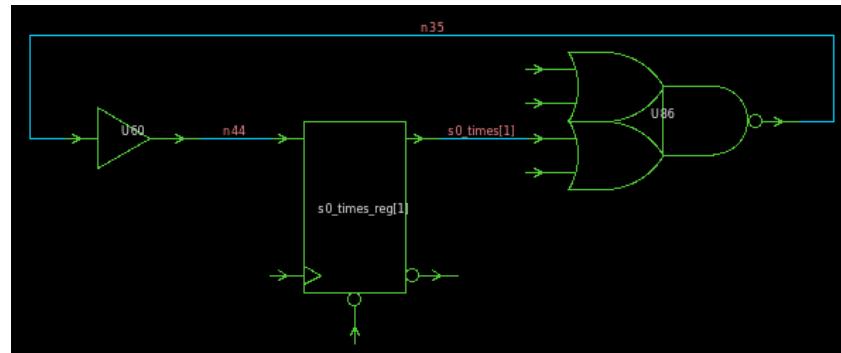
3. Hold timing report

Report : timing -path full -delay min -max_paths 1 -sort_by group			
Design : traffic_light Version: Q-2019.12 Date : Sun May 12 21:26:38 2024 *****			
Operating Conditions: slow Library: slow Wire Load Model Mode: top			
Startpoint: s0_times_reg[1] (rising edge-triggered flip-flop clocked by clk)			
Endpoint: s0_times_reg[1] (rising edge-triggered flip-flop clocked by clk)			
Path Group: clk Path Type: min			
Des/Clust/Port	Wire Load Model	Library	Constraint 產生的 clock 延遲
traffic_light	tsmc13_wl10	slow	
Point	Incr	Path	
clock_clk (rise edge)	0.00	0.00	
clock network delay (ideal)	1.00	1.00	
s0_times_reg[1]/CK (DFFRX1)	0.00	1.00 r	
s0_times_reg[1]/QN (DFFRX1)	0.51	1.51 r	
U86/Y (OAI22XL)	0.25	1.76 f	
U60/Y (BUFX2)			
s0_times_reg[1]/D (DFFRX1)			
data arrival time	1.99		
clock_clk (rise edge)	0.00	0.00	
clock network delay (ideal)	1.00	1.00	
clock uncertainty	1.00	2.00	
s0_times_reg[1]/CK (DFFRX1)	0.00	2.00 r	
library hold time	-0.03	1.97	
data required time		1.97	
data required time	1.97		
data arrival time	-1.99		
slack (MET)	0.01		

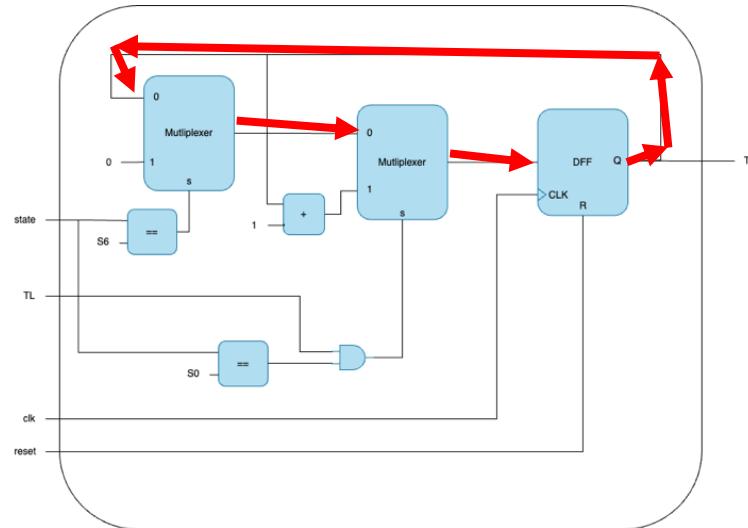
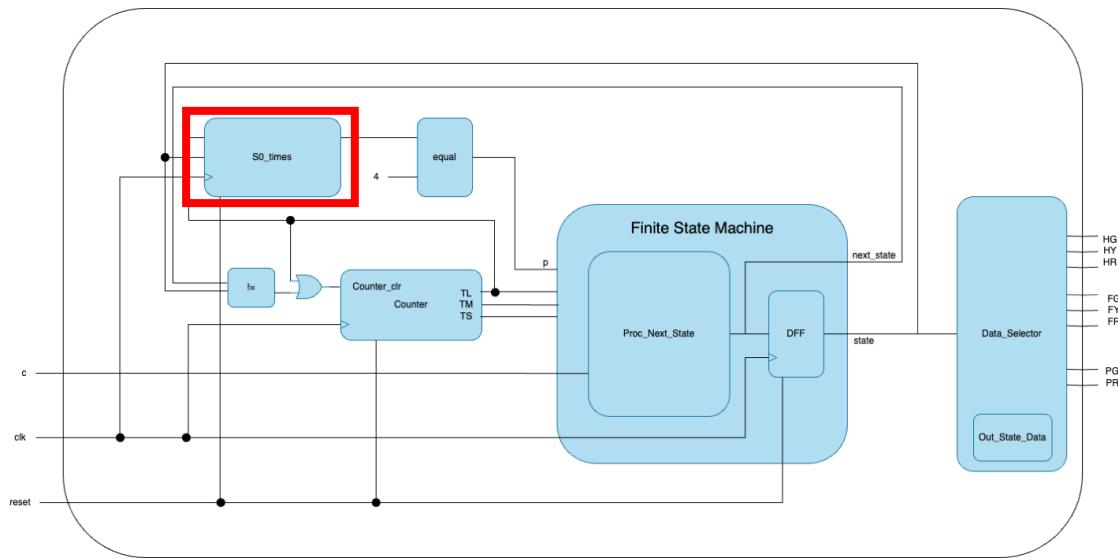
Slack ≥ 0 代表最小路徑延遲符合 Hold time 的要求

4. Critical Path (Hold)

合成後最小延遲路徑如下圖。



根據路徑推斷合成前最小延遲路徑如下圖。



iii. Power Report

Dynamic Power : 53.5948 uW
Static Power : 647.4128 nW

```
*****
Report : power
-analyses_effort low
Design : traffic_light
Version: Q-2019.12
Date   : Sun May 12 21:37:14 2024
*****  
  
Library(s) Used:  
slow (File: /usr/cad/designkit/CBDK_IC_Contest_v2.1/SynopsysDC/db/slow.db)  
  
Operating Conditions: slow Library: slow  
Wire Load Model Mode: top  
  
Design      Wire Load Model      Library  
-----  
traffic_light      tsmc13_wl10      slow  
  
Global Operating Voltage = 1.08  
Power-specific unit information :  
  Voltage Units = 1V  
  Capacitance Units = 1.000000pf  
  Time Units = 1ns  
  Dynamic Power Units = 1mW    (derived from V,C,T units)  
  Leakage Power Units = 1pW  
  
Cell Internal Power = 42.6093 uW (80%)  
Net Switching Power = 10.9856 uW (20%)  
  
Total Dynamic Power = 53.5948 uW (100%)  
Cell Leakage Power = 647.4128 nW  
  
Dynamic Power  
  
Static Power  
  
Power Group      Internal Power      Switching Power      Leakage Power      Total Power ( % ) Atts  
-----  
io_pad          0.0000            0.0000            0.0000            0.0000 ( 0.00%)  
memory          0.0000            0.0000            0.0000            0.0000 ( 0.00%)  
black_box        0.0000            0.0000            0.0000            0.0000 ( 0.00%)  
clock_network    0.0000            0.0000            0.0000            0.0000 ( 0.00%)  
register         4.0238e-02       3.8880e-03       3.2711e+05       4.4453e-02 ( 81.95%)  
sequential        0.0000            0.0000            0.0000            0.0000 ( 0.00%)  
combinational    2.3711e-03       7.0976e-03       3.2030e+05       9.7890e-03 ( 18.05%)  
-----  
Total           4.2609e-02 mW     1.0986e-02 mW     6.4741e+05 pW     5.4242e-02 mW  
1
```

五、模擬

i. 參考指令

本次設計的 testbench 將許多功能分開撰寫（如 SDF 讀取與生成波形檔），讓測試更有彈性如有需要生成波形在指令後加入+define+FSDB，若使用 Gate level 模擬請先確定 SDF 檔案位置是否與 testbench 一致，並且在指令後新增+define+SDF，此外可以調整執行週指令後新增+define+CYCLE=6（預設為 10）。

1. RTL 模擬：

```
ncverilog tb.v traffic_light.v +define+FSDB +access+r
```

2. Gate Level 模擬：

```
ncverilog tb.v traffic_light_syn.v -v tsmc13.v +define+SDF+FSDB+CYCLE=6  
+access+r
```

SDF 檔案路徑

```
1 `timescale 1ns/1ps
2
3 `define CYCLE_10
4 `define SDF_PATH "./syn/Results/traffic_light_syn.sdf"
5
6
7
8 module tb;
9
10
11 parameter LONG_TIME = 20;
12 parameter MEDIUM_TIME = 4;
13 parameter SHORT_TIME = 2;
14 parameter PEDESTRAIN_CHANSFER = 5;
15
16
17 // Target Answer
18 parameter S0_Target = 8'b100_001_01;
19 parameter S1_Target = 8'b010_001_01;
20 parameter S2_Target = 8'b001_001_01;
21 parameter S3_Target = 8'b001_100_01;
22 parameter S4_Target = 8'b001_010_01;
23 parameter S5_Target = 8'b001_001_01;
24 parameter S6_Target = 8'b001_001_10;
25 parameter S7_Target = 8'b001_001_01;
26
27
28 endmodule
```

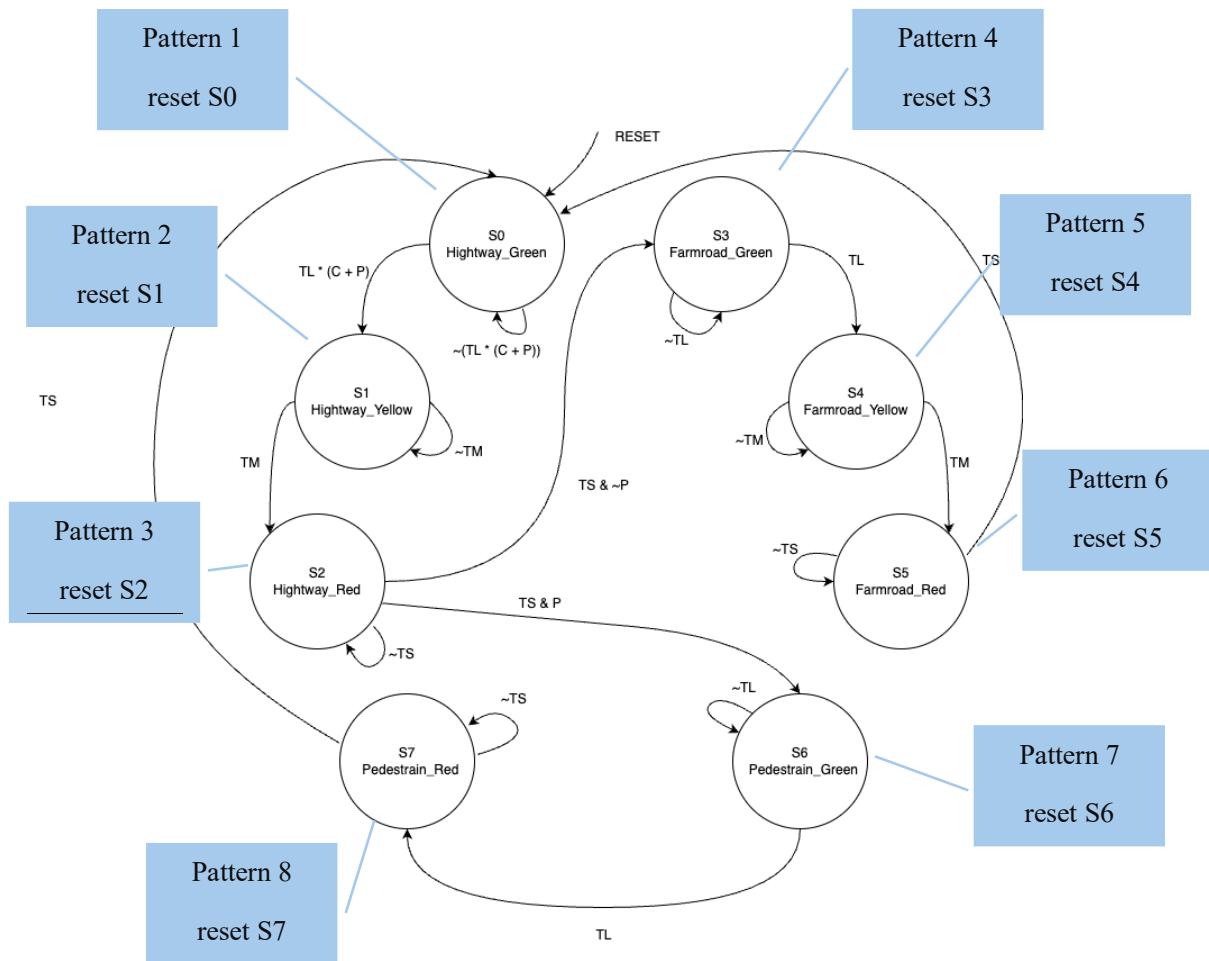
測試成功後會呈現一隻吉伊卡哇的圖案

ii. 測資設計 (RTL 模擬)

此次的設計目標為所有狀態接執行過一次並且每種轉換都跑過至少一次。因此我們可以將轉換分為兩種第一種為 RESET，另一種為狀態間的切換。

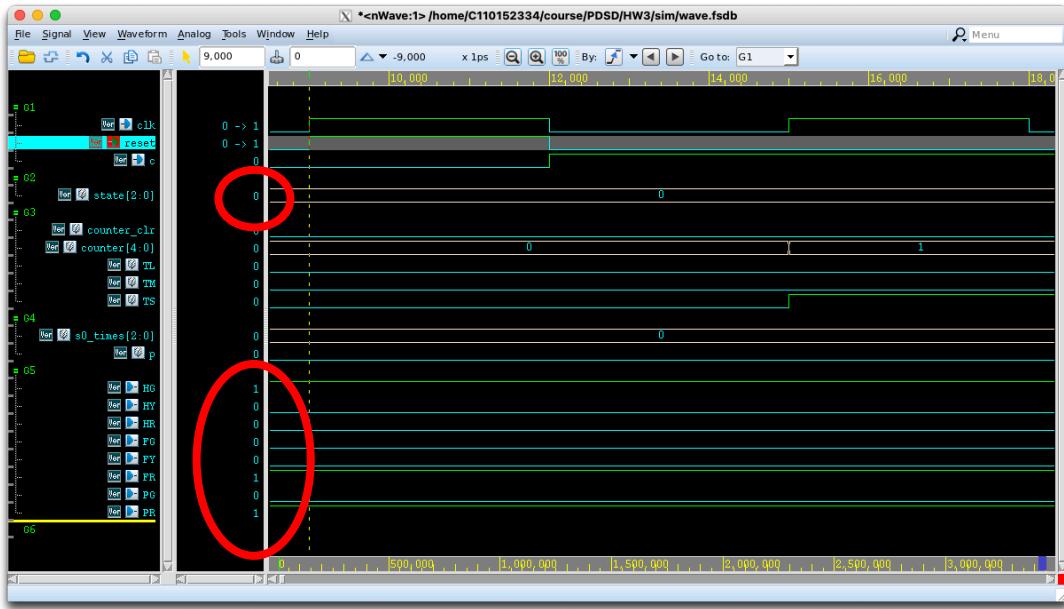
1. RESET (Pattern 1 ~ 8) :

將狀態從 S0~S7 做 RESET 判斷是否會切回 S0 狀態，並且檢查狀態是否輸出對應輸出訊號。



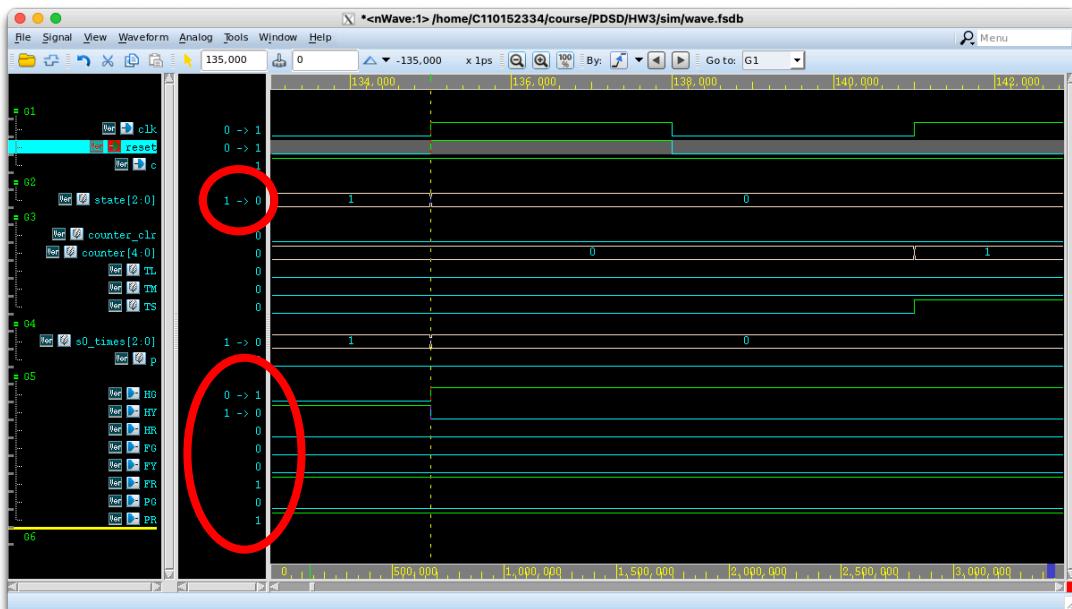
Pattern 1 (S0 -> S0)

State	HG	HY	HR	FG	FY	FR	PG	PR
S0	1	0	0	0	0	1	0	1



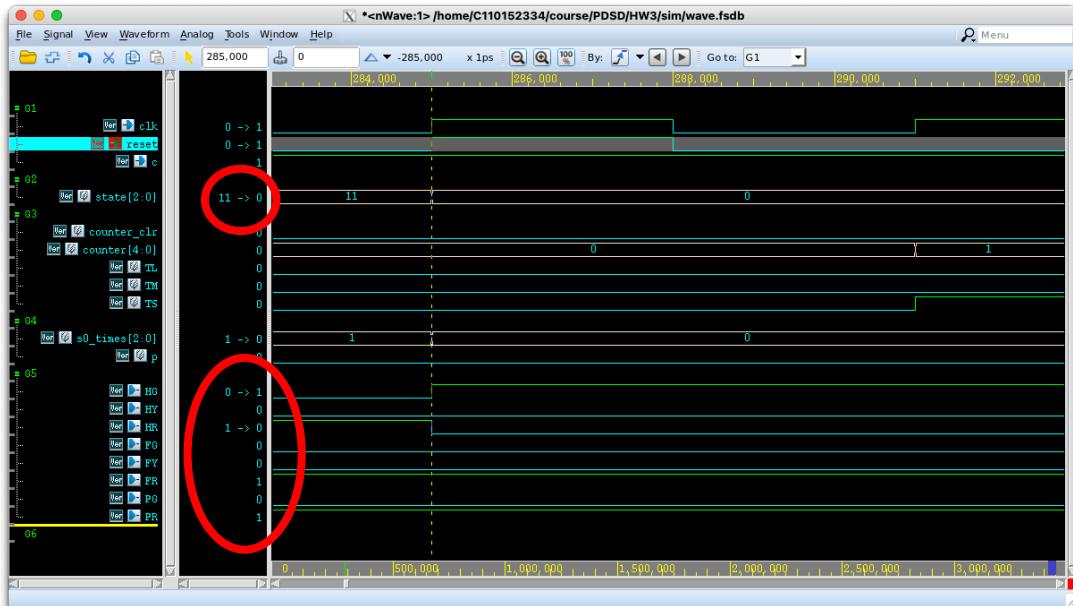
Pattern 2 (S1 -> S0)

State	HG	HY	HR	FG	FY	FR	PG	PR
S1	0	1	0	0	0	1	0	1



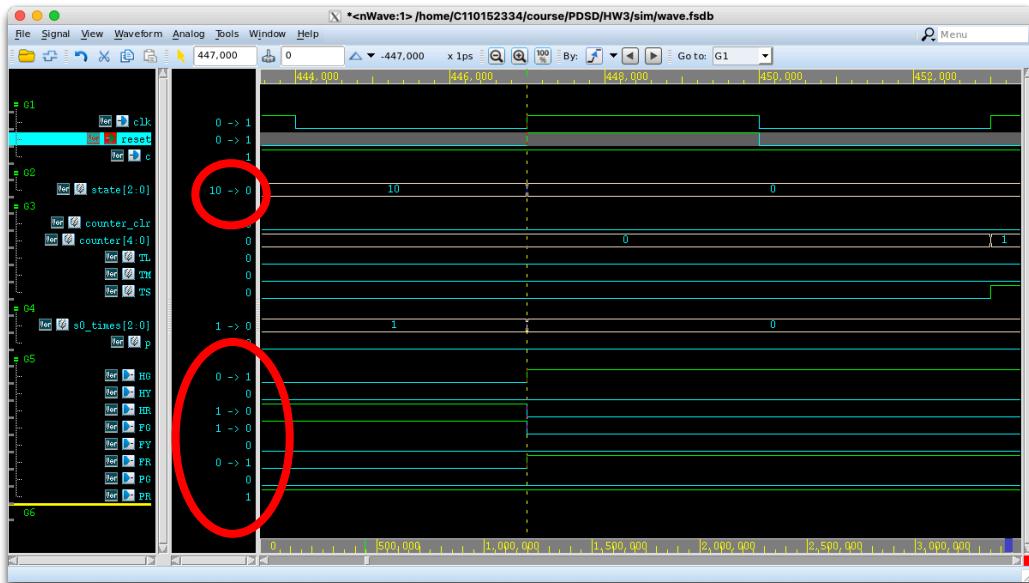
Pattern 3 (S2 -> S0)

State	HG	HY	HR	FG	FY	FR	PG	PR
S2	0	0	1	0	0	1	0	1



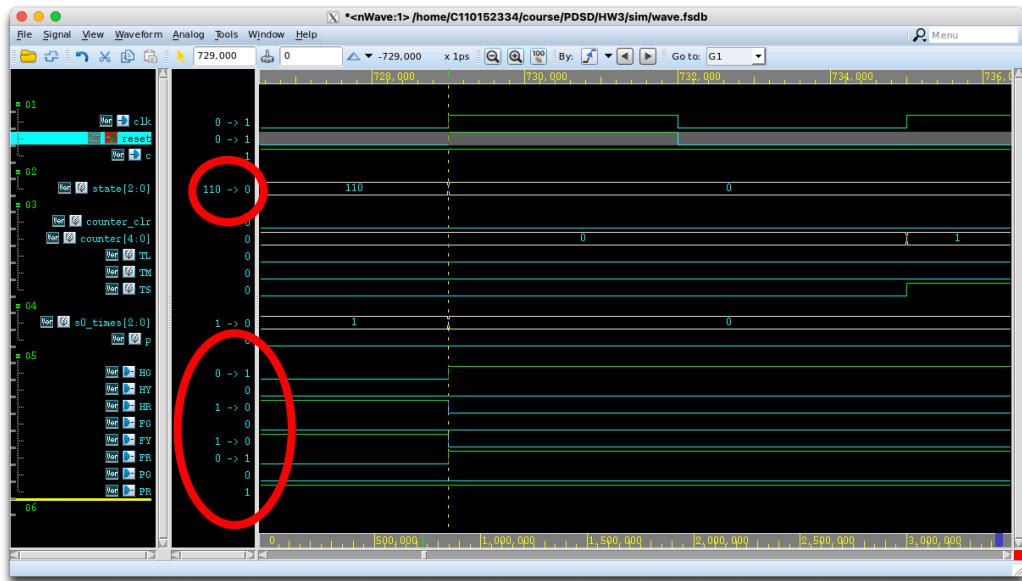
Pattern 4 (S3 -> S0)

State	HG	HY	HR	FG	FY	FR	PG	PR
S3	0	0	1	1	0	0	0	1



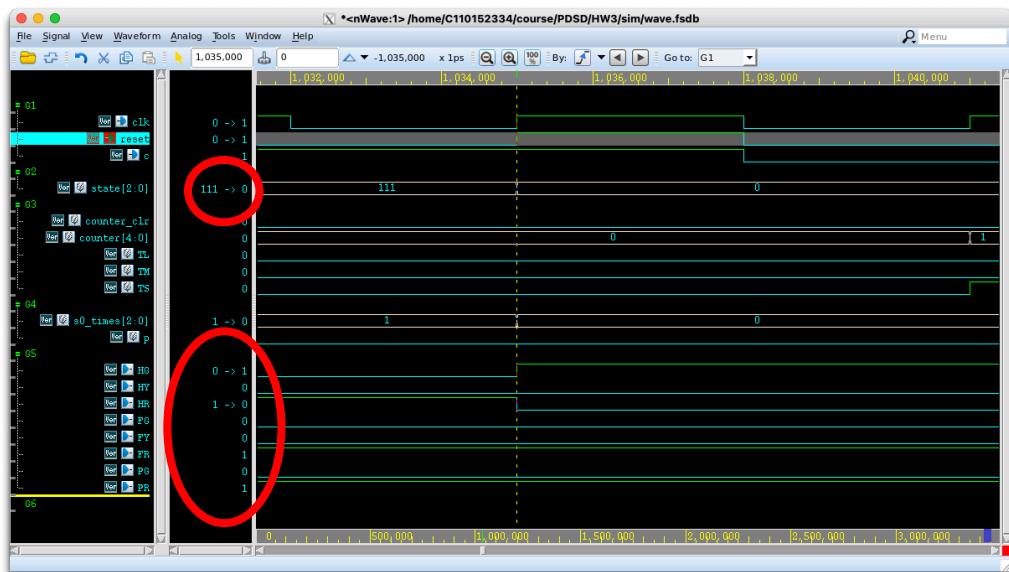
Pattern 5 (S4 -> S0)

State	HG	HY	HR	FG	FY	FR	PG	PR
S4	0	0	1	0	1	0	0	1



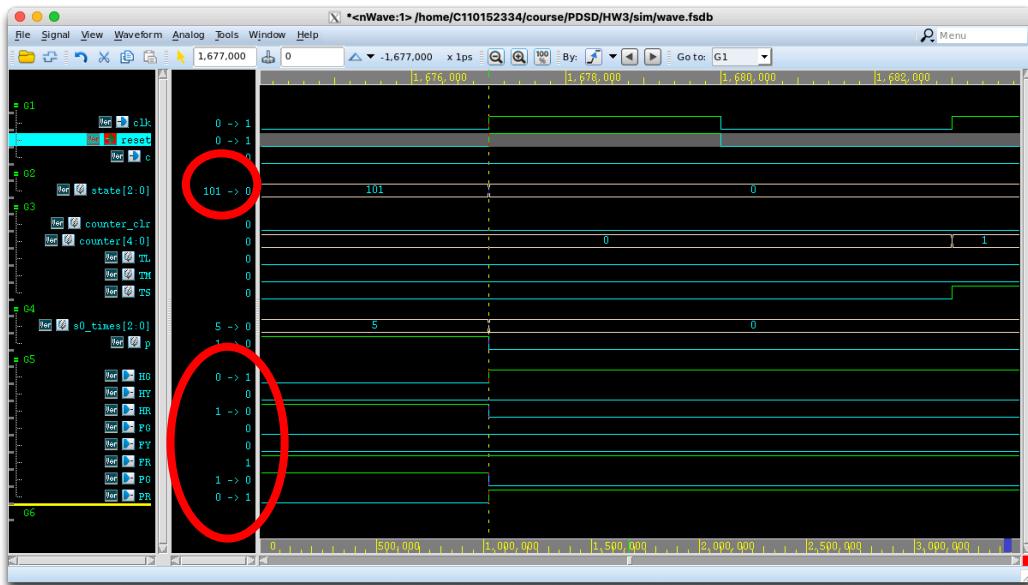
Pattern 6 (S5 -> S0)

State	HG	HY	HR	FG	FY	FR	PG	PR
S4	0	0	1	0	0	1	0	1



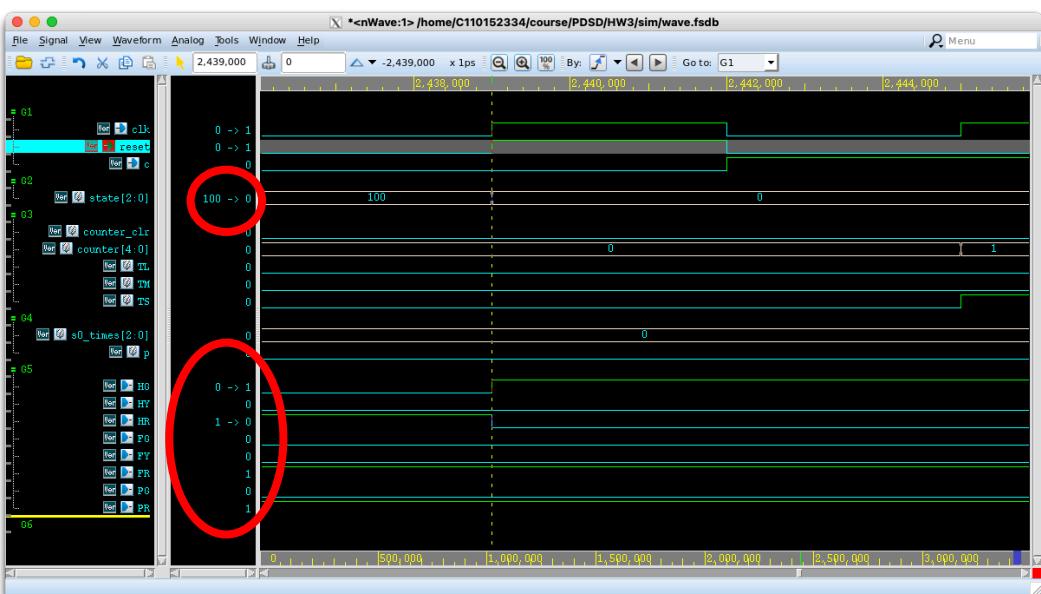
Pattern 7 (S6 -> S0)

State	HG	HY	HR	FG	FY	FR	PG	PR
S4	0	0	1	0	0	1	1	0



Pattern 8 (S7 -> S0)

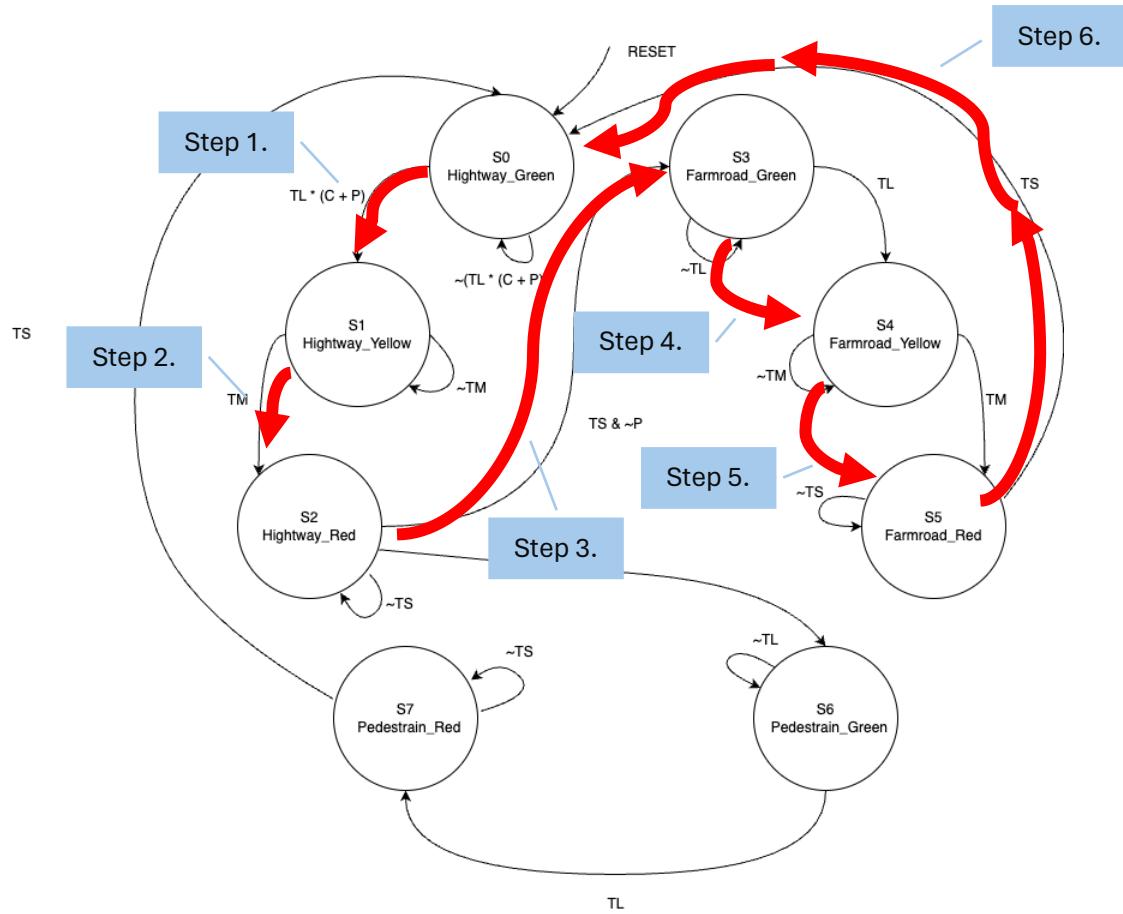
State	HG	HY	HR	FG	FY	FR	PG	PR
S4	0	0	1	0	0	1	0	1

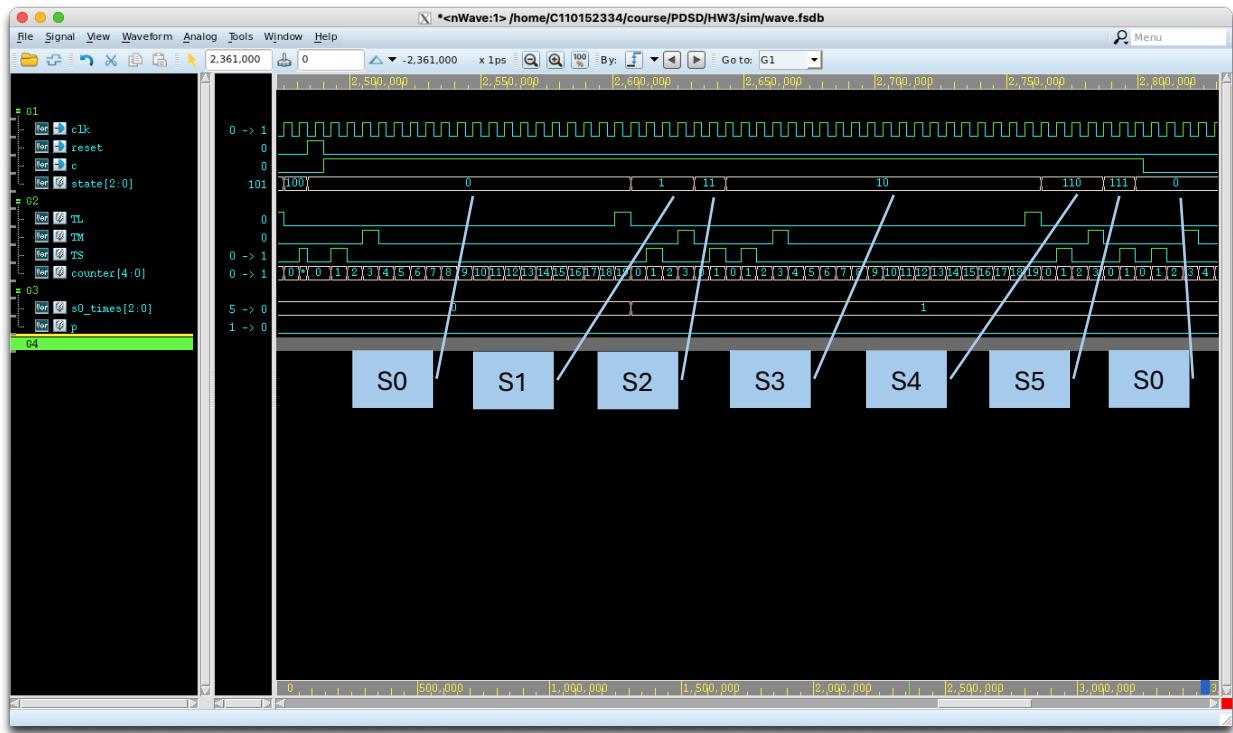


2. 狀態切換：

檢查這個路徑間的跳轉狀態，無須刻意檢查維持的跳轉狀態（如 S0 -> S0，激勵條件為 $\sim(TL * (C + P))$ ），因為在當發生狀態轉換之前就已經發生數次的維持，因此我們只要檢測在對應時間點是否跳轉至某狀態即可。

Pattern 9 : S0 -> S1 -> S2 -> S3 -> S4 -> S5 -> S0





Step 1. 當 TL 觸發時並且符合 $TL^*(C+P)$ ，跳轉至 S1。

Step 2. 當 TM 觸發時跳轉至 S2。

Step 3. 當 TS 觸發時並且符合 $TS^*(\sim P)$ ，跳轉至 S3。

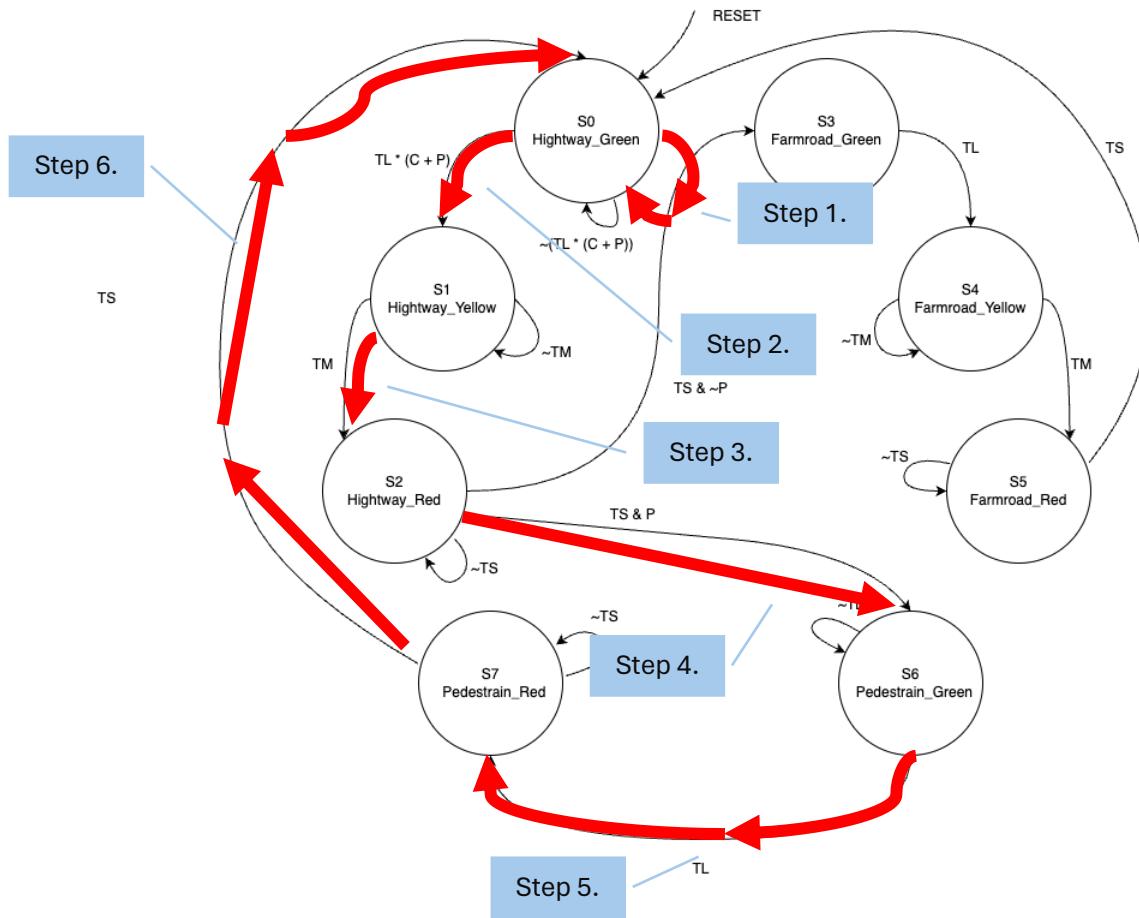
Step 4. 當 TL 觸發時跳轉至 S4。

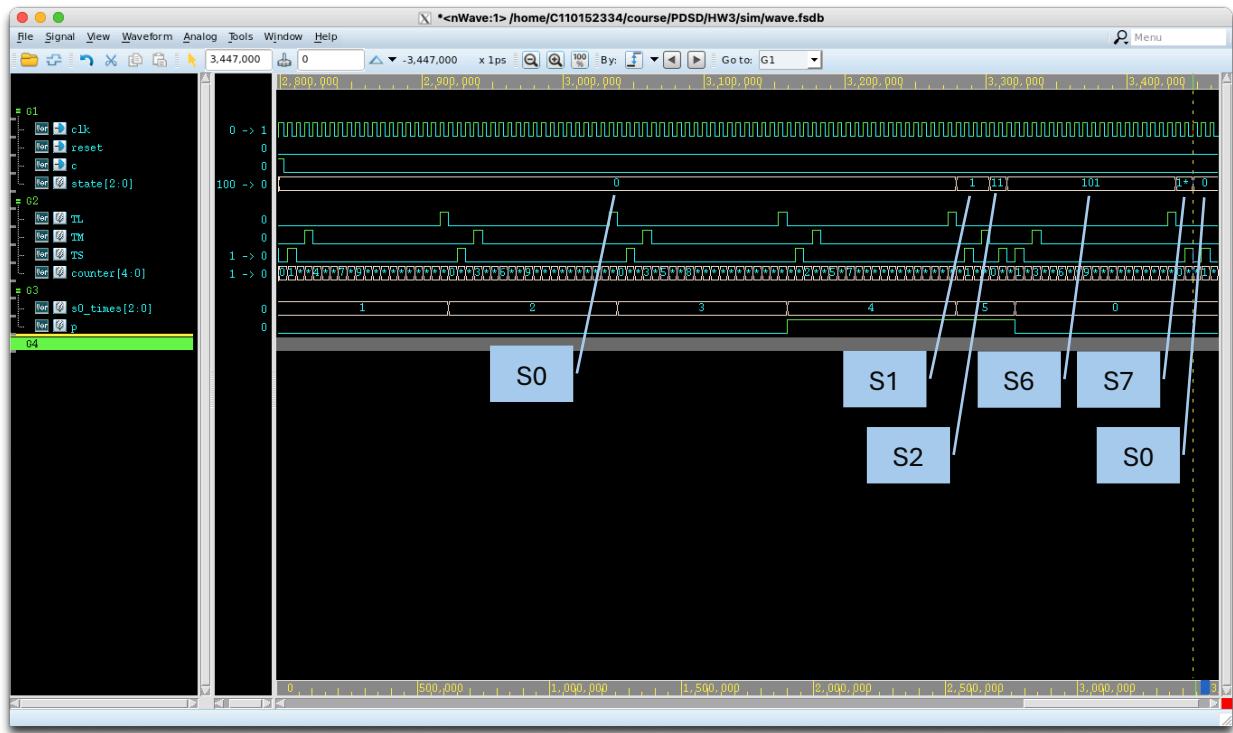
Step 5. 當 TM 觸發時跳轉至 S5。

Step 6. 當 TS 觸發時跳轉至 S0。

Pattern 10 : $S_0^*3 \rightarrow S_1 \rightarrow S_2 \rightarrow S_6 \rightarrow S_7 \rightarrow S_0$

要測試 S_6 與 S_7 必須先執行五次的 S_0 才會啟動，所以這筆 Pattern 放在最後值行，由於先前已經執行過 1 次，所已只需再執行 4 次即可進入此模式。





Step 1. 當 TL 觸發時並且符合 $TL^*(C+P)$ ，跳轉至 S1。

Step 2. 當 TM 觸發時跳轉至 S2。

Step 3. 當 TS 觸發時並且符合 $TS^*(P)$ ，跳轉至 S6。

Step 4. 當 TL 觸發時跳轉至 S7。

Step 5. 當 TS 觸發時跳轉至 S0。

iii. 合成後模擬 (Gate level 模擬)

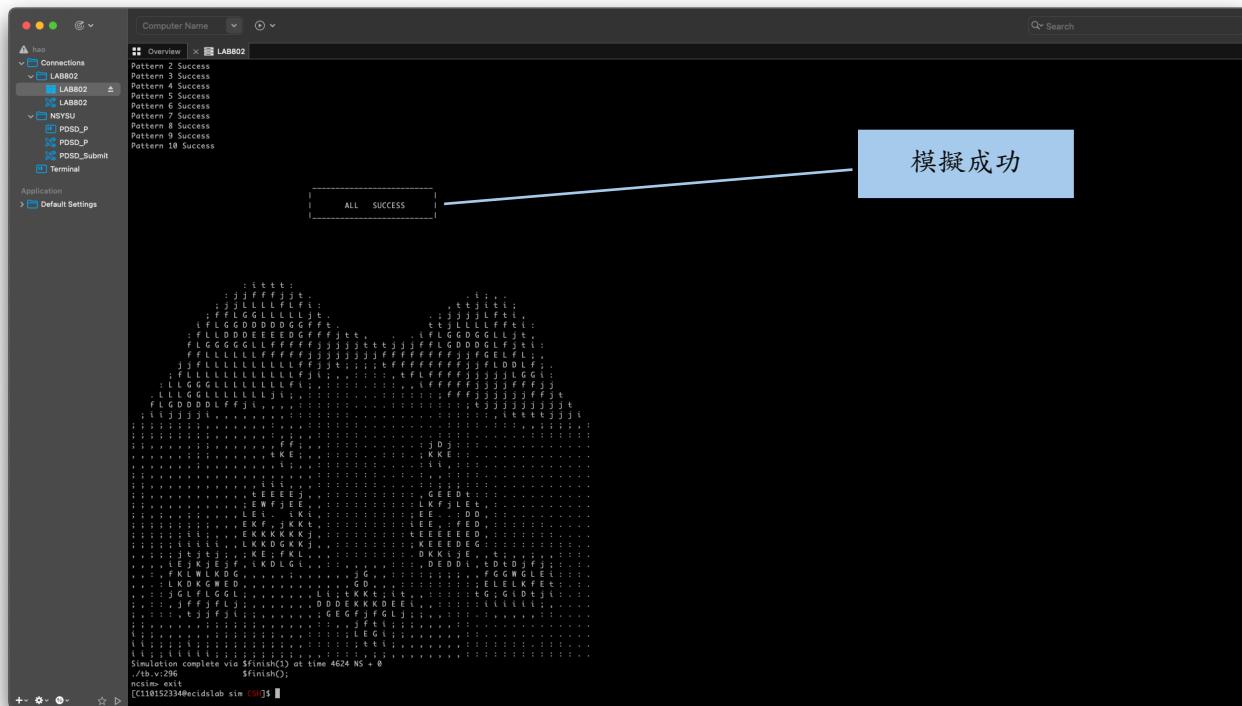
1. 合成後，開始產生延遲相關資訊。模擬應該符合 clock constraint 的頻率，否則可能會發生 Timing violation，就像下圖所示。因此，我們應該調整成與合成時 constraint 的頻率。

```
Warning! Timing violation
$setuphold<hold>< posedge CK && (flag == 1):6 NS, negedge D:6 NS, 1.000 : 1 NS, 0.500 : 500 PS );
  File: ./tsmc13_neg.v, line = 18348
  Scope: tb.dut.\s0_times_reg[2]
  Time: 6 NS
```

2. 模擬：

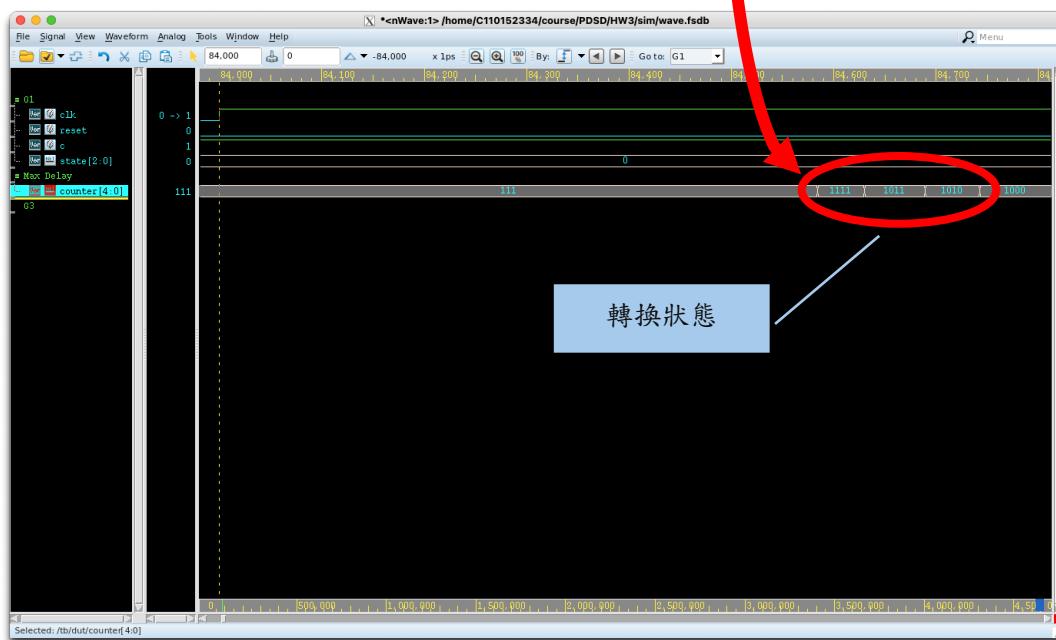
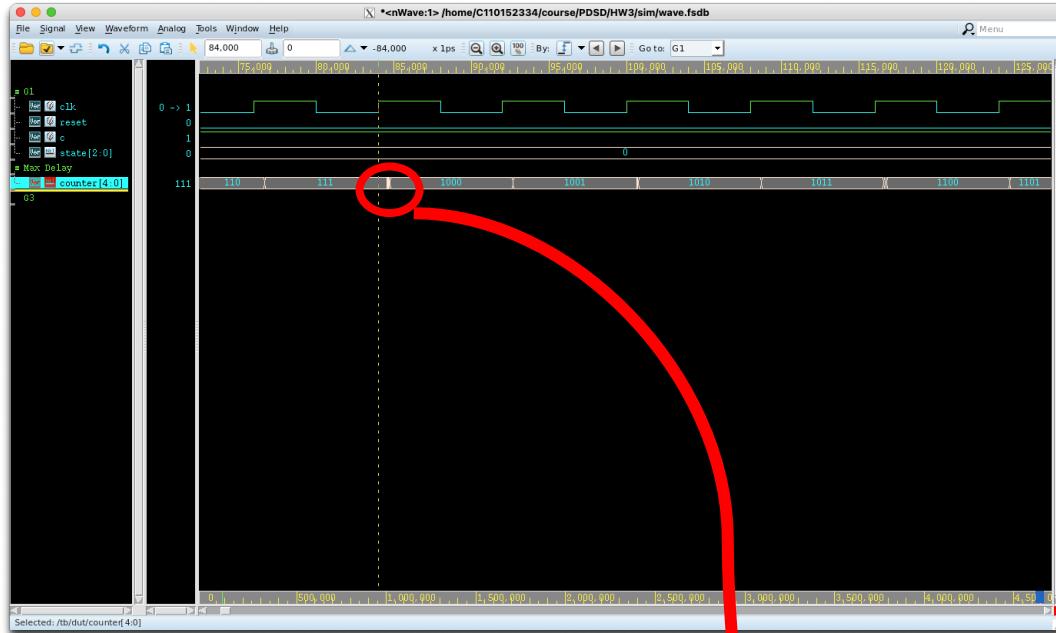
```
Reading SDF file from location ".../syn/Results/traffic_light_syn.sdf"
Writing compiled SDF file to "traffic_light_syn.sdf.X".
Annotating SDF timing data:
  Compiled SDF file: traffic_light_syn.sdf.X
  Log file:
  Backannotation scope: tb.dut
  Configuration file:
  MTM control:
  Scale factors:
  Scale type:
Annotation completed successfully...
SDF statistics: No. of Pathdelays = 197 Annotated = 100.00% -- No. of Tchecks = 66 Annotated = 100.00%
      Total      Annotated      Percentage
Path Delays          197          197        100.00
      $width           33           33        100.00
      $setuphold       33           33        100.00
```

成功讀取延遲檔



3. 波形：

合成完成後，電路會產生每個邏輯閘的延遲。因此，在訊號轉換的過程中，常常會出現許多個暫態，直到最終達到穩態。



iv. 測試覆蓋率

我使用 Synopsys 所提供的 vcs 以及 dve 來觀察測試覆蓋率，我觀察了共五種的測試覆蓋率 Line、Toggle、FSM，Condition 與 Branch。

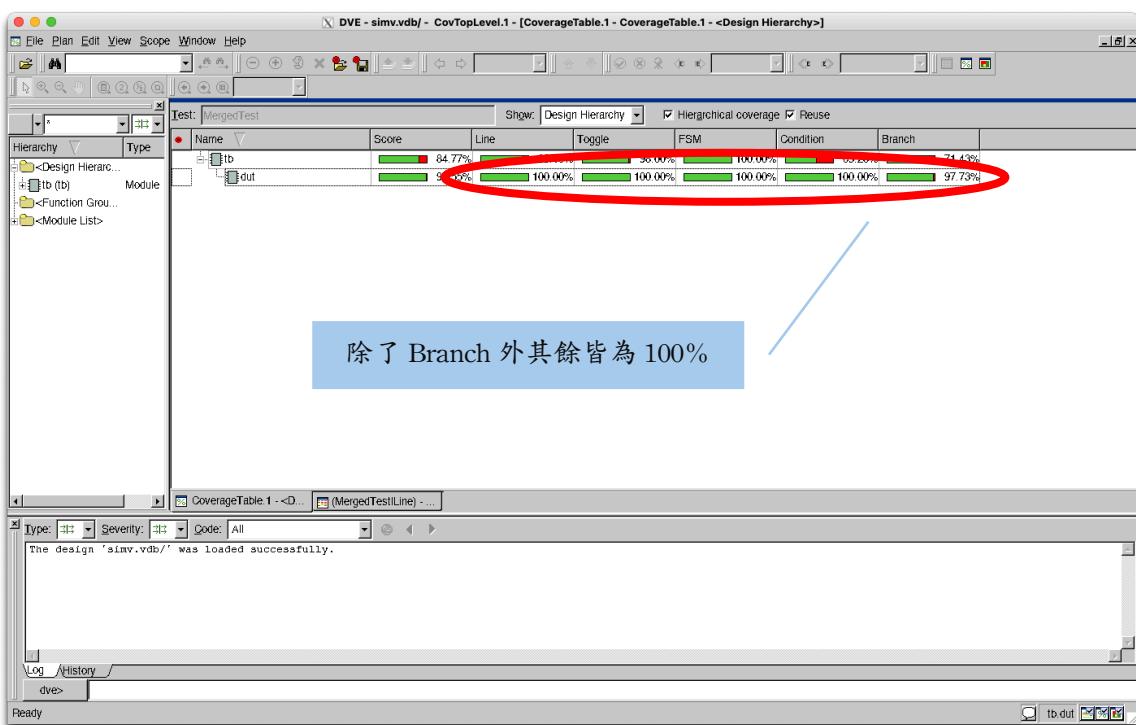
VCS 就如同 ncverilog 一樣次一個模擬工具而 DVE 是一個 Debug 工具，以下是他的使用指令。

Command :

```
vcs -R -full64 tb.v traffic_light.v -cm line+toggle+fsm+condition+branch  
+debug_access
```

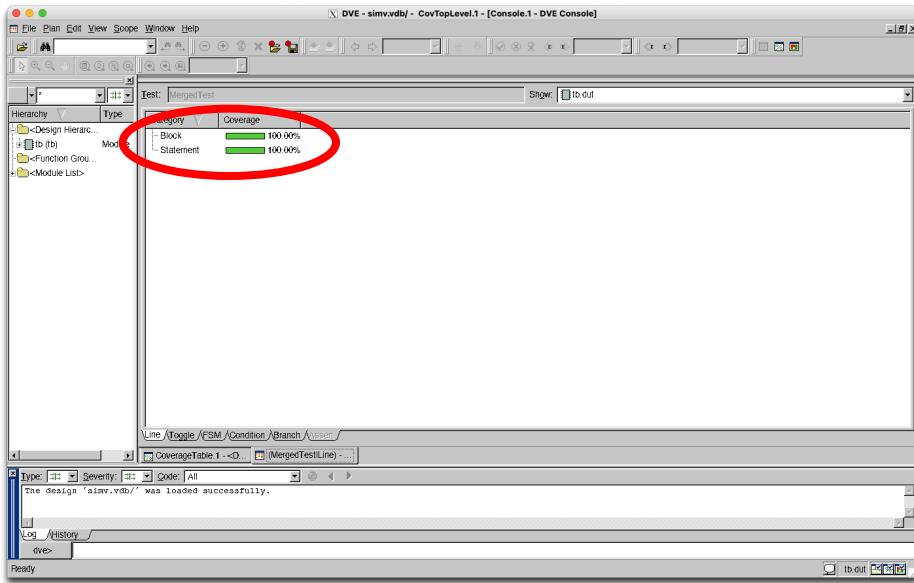
```
dve -full64 -dir simv.vdb
```

Total Coverage



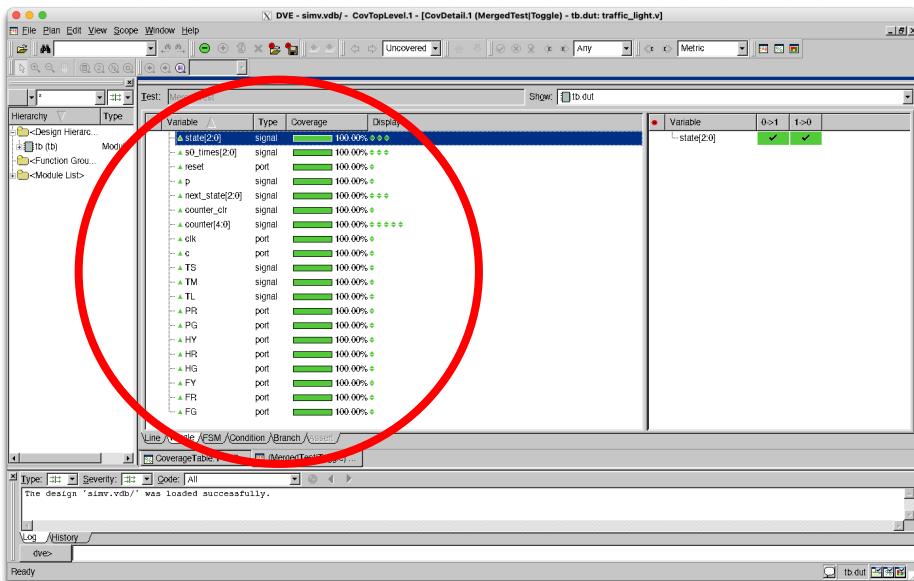
Line Coverage

執行的行數佔全部的多少，這次測試執行過所有行電路檔覆蓋率為 100%。



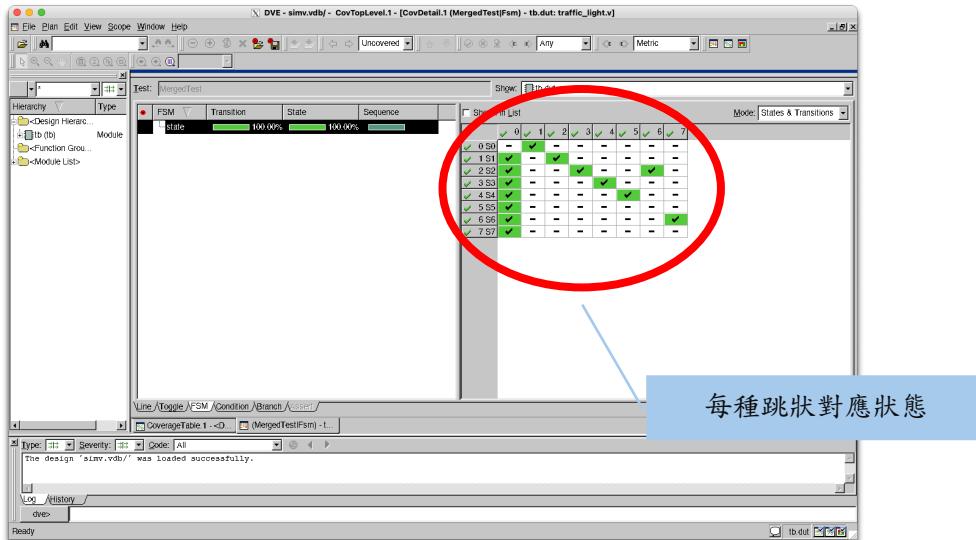
Toggle Coverage

計算電路中每個訊號的翻轉佔所有訊號的比例，這裡會觀測每個訊號在執行 testbench 時是否皆有翻轉包含 0 變 1 與 1 變 0，這個 Testbench 讓所有訊號都有翻轉過至少一次覆蓋率為 100%。



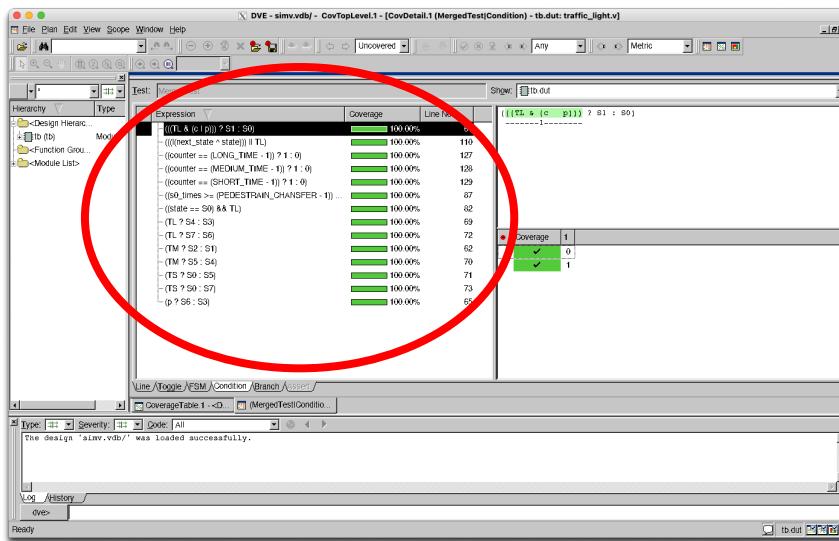
FSM Coverage

計算 FSM 跳轉狀態的項目佔所有跳轉項目的比例，這次測試中執行了所有種的跳轉狀態覆蓋率為 100%。



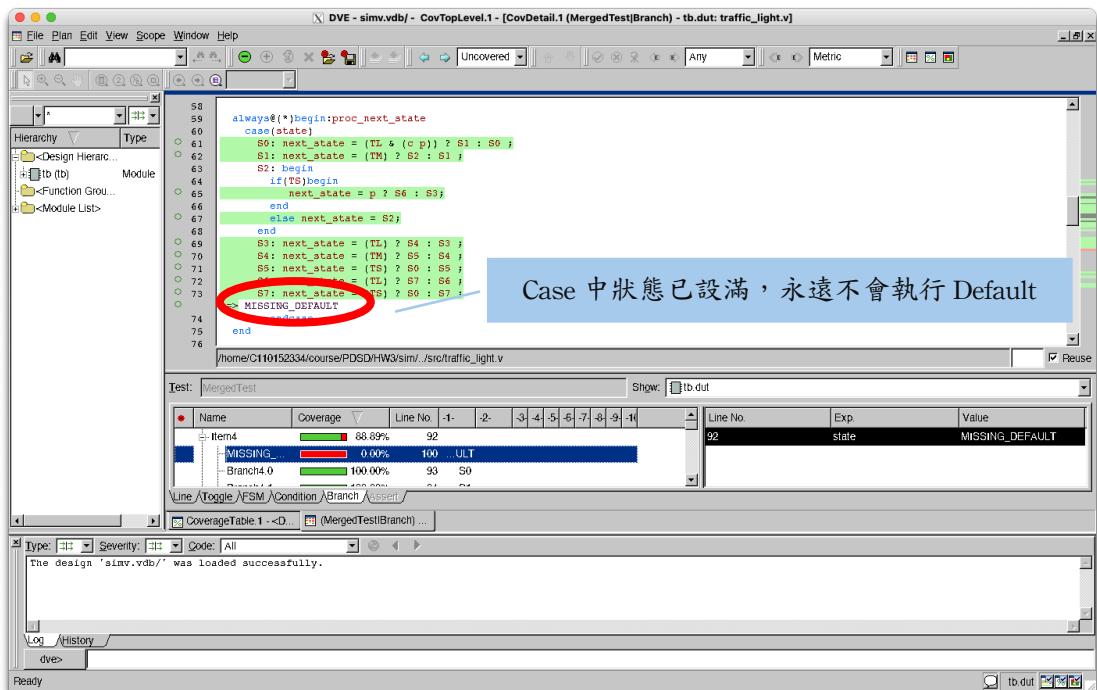
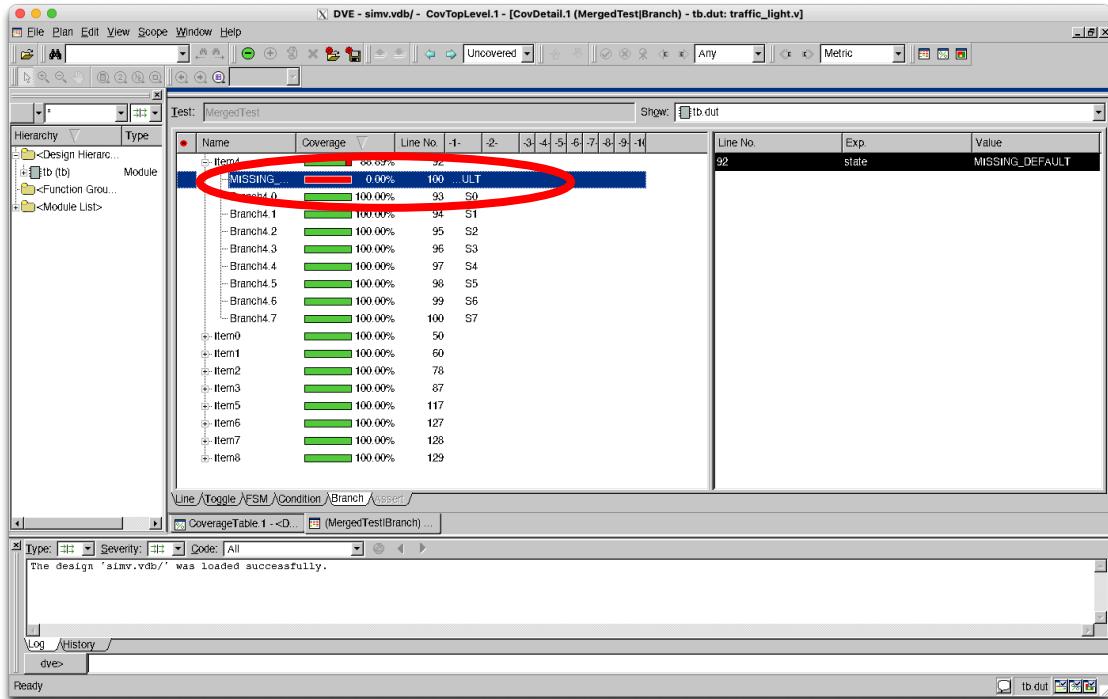
Condition Coverage

計算執行過的 if else 狀態（包含三元運算子），這次測試每種條件都執行過覆蓋率為 100%。



Branch Coverage

計算執行過的 case 狀態佔所有 case 狀態的比例，這裡產生了一個未覆蓋到的分支為如下，由於狀態是設滿的所以永遠不會執行到 default，因此達不到滿分。



六、 檔案介紹

	File Names	備註
1	traffic_light.v	電路設計檔
2	tb.v	Testbench
3	traffic_light_syn.v	合成後電路設計檔
4	traffic_light_syn.sdf	合成後電路延遲檔案
5	C110152334.pdf	報告

七、 心得

這次的任務對我來說是一段相當寶貴的學習旅程。特別是在探索 Design Compiler 和設計 Testbench 的過程中，我深刻體會到了許多新的知識和技能。過去，我主要使用 Quartus 這個工具進行合成，合成過程相對簡單，只需點擊一下 "run" 按鈕即可。然而，當我開始轉向使用 Design Compiler 時，我意識到合成的過程變得更加複雜。需要設定大量的約束條件和各種參數，而合成的結果可能與預期不符，需要進行額外的調整。這讓我更深入地了解了合成工具的運作原理，以及如何優化設計以達到更好的結果。儘管在過程中可能會遇到挑戰，但這些挑戰也是成長的機會。我期待著在未來的項目中能夠更加熟練地應用這些工具和技術。

過去，在設計有狀態機電路的 Testbench 時，我往往只是隨意輸入一些數據，然後觀察是否符合預期的結果。然而，在設計的過程中，我想到了老師在課堂上曾經提及的設計覆蓋率。於是開始從這方面入手，開始研究如何使用工具來觀察測試覆蓋率。起初，我的 Testbench 第一個版本仍然存在許多未能覆蓋到的地方。通過工具生成的報告，我發現了許多未覆蓋的點，並將它們添加到了設計中。當所有點都得到覆蓋時，我感到非常有成就感（儘管最後仍然有一個永遠不會被覆蓋的點）。這個過程讓我更加深入地了解了測試覆蓋率的重要性，並學到了如何通過調整 Testbench 來提高覆蓋率，從而提高設計的測試品質。