110521167 PA4 Report

Case1

由於對 4 bits 比較器進行完整的 simulation 共需要 256 個 test pattern，因此我寫了 case1_tb_generator.cpp，用來產生 PA4_110521167_case1_tb.v，同時在 testbench 中加入用於輸出 result[2..0]是否和預期的功能相符的程式碼，因此在跑 ncverilog 時能夠多產生 Comparator_4bits_simulation.txt，並將有問題的地方標記為「X」，case1_simulation_original.txt 部分結果如下。

case1_simulation_original.txt (only 80 lines)

```
1  a=0000 b=0000    result=(0, 0, 1)
2  a=0000 b=0001    result=(1, 0, 0)
3  a=0000 b=0010    result=(1, 0, 0)
4  a=0000 b=0011    result=(1, 0, 0)
5  a=0000 b=0100    result=(1, 0, 0)
6  a=0000 b=0101    result=(1, 0, 0)
7  a=0000 b=0110    result=(1, 0, 0)
8  a=0000 b=0111    result=(1, 0, 0)
9  a=0000 b=1000    result=(1, 0, 0)
10 a=0000 b=1001    result=(1, 0, 0)
11 a=0000 b=1010    result=(1, 0, 0)
12 a=0000 b=1011    result=(1, 0, 0)
13 a=0000 b=1100    result=(1, 0, 0)
14 a=0000 b=1101    result=(1, 0, 0)
15 a=0000 b=1110    result=(1, 0, 0)
16 a=0000 b=1111    result=(1, 0, 0)
17 a=0001 b=0000    result=(0, 0, 1)X
18 a=0001 b=0001    result=(0, 0, 1)
19 a=0001 b=0010    result=(1, 0, 0)
20 a=0001 b=0011    result=(1, 0, 0)
21 a=0001 b=0100    result=(1, 0, 0)
22 a=0001 b=0101    result=(1, 0, 0)
23 a=0001 b=0110    result=(1, 0, 0)
24 a=0001 b=0111    result=(1, 0, 0)
25 a=0001 b=1000    result=(1, 0, 0)
26 a=0001 b=1001    result=(1, 0, 0)
27 a=0001 b=1010    result=(1, 0, 0)
28 a=0001 b=1011    result=(1, 0, 0)
29 a=0001 b=1100    result=(1, 0, 0)
30 a=0001 b=1101    result=(1, 0, 0)
31 a=0001 b=1110    result=(1, 0, 0)
32 a=0001 b=1111    result=(1, 0, 0)
33 a=0010 b=0000    result=(0, 1, 0)
34 a=0010 b=0001    result=(0, 1, 0)
35 a=0010 b=0010    result=(0, 0, 1)
36 a=0010 b=0011    result=(1, 0, 0)
37 a=0010 b=0100    result=(1, 0, 0)
38 a=0010 b=0101    result=(1, 0, 0)
39 a=0010 b=0110    result=(1, 0, 0)
40 a=0010 b=0111    result=(1, 0, 0)
41 a=0010 b=1000    result=(1, 0, 0)
42 a=0010 b=1001    result=(1, 0, 0)
43 a=0010 b=1010    result=(1, 0, 0)
44 a=0010 b=1011    result=(1, 0, 0)
45 a=0010 b=1100    result=(1, 0, 0)
46 a=0010 b=1101    result=(1, 0, 0)
47 a=0010 b=1110    result=(1, 0, 0)
48 a=0010 b=1111    result=(1, 0, 0)
49 a=0011 b=0000    result=(0, 1, 0)
50 a=0011 b=0001    result=(0, 1, 0)
51 a=0011 b=0010    result=(0, 0, 1)X
52 a=0011 b=0011    result=(0, 0, 1)
53 a=0011 b=0100    result=(1, 0, 0)
54 a=0011 b=0101    result=(1, 0, 0)
55 a=0011 b=0110    result=(1, 0, 0)
56 a=0011 b=0111    result=(1, 0, 0)
57 a=0011 b=1000    result=(1, 0, 0)
58 a=0011 b=1001    result=(1, 0, 0)
59 a=0011 b=1010    result=(1, 0, 0)
60 a=0011 b=1011    result=(1, 0, 0)
61 a=0011 b=1100    result=(1, 0, 0)
62 a=0011 b=1101    result=(1, 0, 0)
63 a=0011 b=1110    result=(1, 0, 0)
64 a=0011 b=1111    result=(1, 0, 0)
65 a=0100 b=0000    result=(0, 1, 0)
66 a=0100 b=0001    result=(0, 1, 0)
67 a=0100 b=0010    result=(0, 1, 0)
68 a=0100 b=0011    result=(0, 1, 0)
69 a=0100 b=0100    result=(0, 0, 1)
70 a=0100 b=0101    result=(1, 0, 0)
71 a=0100 b=0110    result=(1, 0, 0)
72 a=0100 b=0111    result=(1, 0, 0)
73 a=0100 b=1000    result=(1, 0, 0)
74 a=0100 b=1001    result=(1, 0, 0)
75 a=0100 b=1010    result=(1, 0, 0)
76 a=0100 b=1011    result=(1, 0, 0)
77 a=0100 b=1100    result=(1, 0, 0)
78 a=0100 b=1101    result=(1, 0, 0)
79 a=0100 b=1110    result=(1, 0, 0)
80 a=0100 b=1111    result=(1, 0, 0)
```

Case1 Error Table	
a=0011 b=0010	result=(0, 0, 1)X
a=0011 b=0010	result=(0, 0, 1)X
a=0101 b=0100	result=(0, 0, 1)X
a=0111 b=0110	result=(0, 0, 1)X
a=1001 b=1000	result=(0, 0, 1)X
a=1011 b=1010	result=(0, 0, 1)X
a=1101 b=1100	result=(0, 0, 1)X
a=1111 b=1110	result=(0, 0, 1)X

根據 Case1 Error Table 可以發現只有在 a > b 的情況下會誤判，且都被判斷為 a = b，因此檢查 eq、ba 兩行程式碼後分別有以下發現。

Hardware Trojan 1 (x[4])

```
or      or1( x[4],  ab_bar[0], x[0] ); // trigger
```

```
// and      eq( result[0], x[3], x[2], x[1], x[4] ); // error, x[4] is payload
and     eq( result[0], x[3], x[2], x[1], x[0] ); // x[4] modified to x[0]
```

檢查 eq 可以發現訊號來源為 x[3]、x[2]、x[1] 、x[4]，從程式碼中可以知道 x[3 : 0]分別代表的是 a[i]、b[i]是否相等，這時 x[4]就顯得很可疑，因此將 x[4]修改為 x[0]較符合判斷相等的邏輯。

Hardware Trojan 2 (x[6])

```
not     n1( x[5], x[4]); // trigger
and    and2( x[6], x[5],ab_bar[0]); // trigger
```

```
// and  and_5( m[1], x[3], x[2], x[1], x[6] ); // x[6] is payload
and     and_5( m[1], x[3], x[2], x[1], ab_bar[0] ); /* x[6] modified to ab_bar[0],
                                                       (a[3...1] == b[3...1]) && (a[0] > b[0])*/
```

檢查 ba 可以發現訊號來源為 ab_bar[3]、m[5]、m[3]、m[1]，從程式碼中可以知道 ab_bar[3]沒有問題，接著檢查 m[5]、m[3]、m[1]，由 m[6 : 2]的程式碼可以知道 m 的功能是，a 和 b 其他權重較高的位元皆相等時下一個位元 a[i] > b[i]或者 a[i] < b[i]，因此除了用 x 來判斷相等以外，也需要 ab_bar 或 ba_bar 來判斷大於或小於，這時 x[6]顯得很突兀，因此將 x[6]修改為 ab_bar[0]較合乎邏輯，此外 x[6]由 x[5]產生故 x[5]也是 HT 電路的一部份。

在完成電路修改後得到的 PA4_110521167_case1_simulation.txt 如下，沒有任何「X」出現，電路為一個功能正常的 4bits 比較器。

case1_simulation_modified.txt (only 80 lines)

```
a=0000 b=0000    result=(0, 0, 1)
a=0000 b=0001    result=(1, 0, 0)
a=0000 b=0010    result=(1, 0, 0)
a=0000 b=0011    result=(1, 0, 0)
a=0000 b=0100    result=(1, 0, 0)
a=0000 b=0101    result=(1, 0, 0)
a=0000 b=0110    result=(1, 0, 0)
a=0000 b=0111    result=(1, 0, 0)
a=0000 b=1000    result=(1, 0, 0)
a=0000 b=1001    result=(1, 0, 0)
a=0000 b=1010    result=(1, 0, 0)
a=0000 b=1011    result=(1, 0, 0)
a=0000 b=1100    result=(1, 0, 0)
a=0000 b=1101    result=(1, 0, 0)
a=0000 b=1110    result=(1, 0, 0)
a=0000 b=1111    result=(1, 0, 0)
a=0001 b=0000    result=(0, 1, 0)
a=0001 b=0001    result=(0, 0, 1)
a=0001 b=0010    result=(1, 0, 0)
a=0001 b=0011    result=(1, 0, 0)
a=0001 b=0100    result=(1, 0, 0)
a=0001 b=0101    result=(1, 0, 0)
a=0001 b=0110    result=(1, 0, 0)
a=0001 b=0111    result=(1, 0, 0)
a=0001 b=1000    result=(1, 0, 0)
a=0001 b=1001    result=(1, 0, 0)
a=0001 b=1010    result=(1, 0, 0)
a=0001 b=1011    result=(1, 0, 0)
a=0001 b=1100    result=(1, 0, 0)
a=0001 b=1101    result=(1, 0, 0)
a=0001 b=1110    result=(1, 0, 0)
a=0001 b=1111    result=(1, 0, 0)
a=0010 b=0000    result=(0, 1, 0)
a=0010 b=0001    result=(0, 1, 0)
a=0010 b=0010    result=(0, 0, 1)
a=0010 b=0011    result=(1, 0, 0)
a=0010 b=0100    result=(1, 0, 0)
a=0010 b=0101    result=(1, 0, 0)
a=0010 b=0110    result=(1, 0, 0)
a=0010 b=0111    result=(1, 0, 0)
a=0010 b=1000    result=(1, 0, 0)
a=0010 b=1001    result=(1, 0, 0)
a=0010 b=1010    result=(1, 0, 0)
a=0010 b=1011    result=(1, 0, 0)
a=0010 b=1100    result=(1, 0, 0)
a=0010 b=1101    result=(1, 0, 0)
a=0010 b=1110    result=(1, 0, 0)
a=0010 b=1111    result=(1, 0, 0)
a=0011 b=0000    result=(0, 1, 0)
a=0011 b=0001    result=(0, 1, 0)
a=0011 b=0010    result=(0, 1, 0)
a=0011 b=0011    result=(0, 0, 1)
a=0011 b=0100    result=(1, 0, 0)
a=0011 b=0101    result=(1, 0, 0)
a=0011 b=0110    result=(1, 0, 0)
a=0011 b=0111    result=(1, 0, 0)
a=0011 b=1000    result=(1, 0, 0)
a=0011 b=1001    result=(1, 0, 0)
a=0011 b=1010    result=(1, 0, 0)
a=0011 b=1011    result=(1, 0, 0)
a=0011 b=1100    result=(1, 0, 0)
a=0011 b=1101    result=(1, 0, 0)
a=0011 b=1110    result=(1, 0, 0)
a=0011 b=1111    result=(1, 0, 0)
a=0100 b=0000    result=(0, 1, 0)
a=0100 b=0001    result=(0, 1, 0)
a=0100 b=0010    result=(0, 1, 0)
a=0100 b=0011    result=(0, 1, 0)
a=0100 b=0100    result=(0, 0, 1)
a=0100 b=0101    result=(1, 0, 0)
a=0100 b=0110    result=(1, 0, 0)
a=0100 b=0111    result=(1, 0, 0)
a=0100 b=1000    result=(1, 0, 0)
a=0100 b=1001    result=(1, 0, 0)
a=0100 b=1010    result=(1, 0, 0)
a=0100 b=1011    result=(1, 0, 0)
a=0100 b=1100    result=(1, 0, 0)
a=0100 b=1101    result=(1, 0, 0)
a=0100 b=1110    result=(1, 0, 0)
a=0100 b=1111    result=(1, 0, 0)
```

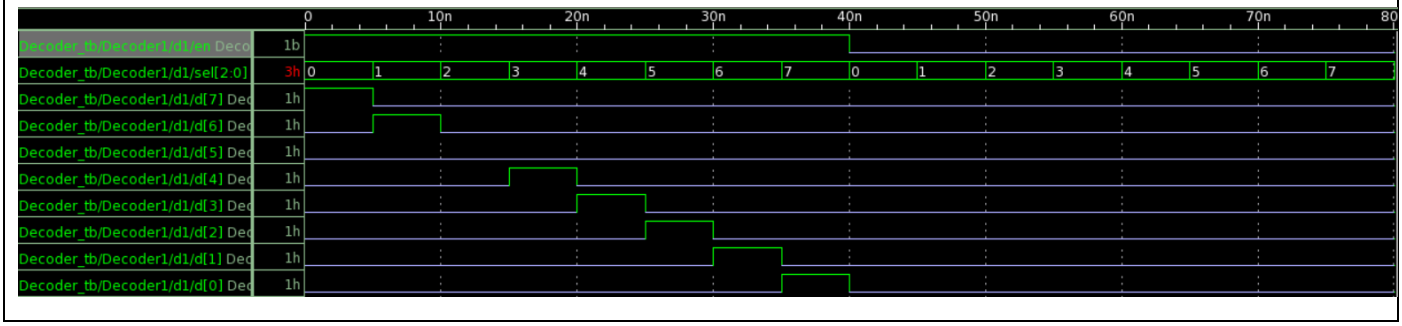
備註**:**

完整的 case1_simulation_original.txt 以及 case1_simulation_modified.txt 繳交再 ee-class 上，可以通過跑 testbench 重複得到這兩個檔案，自動產生的檔名為 Comparator_4bits_simulation.txt。

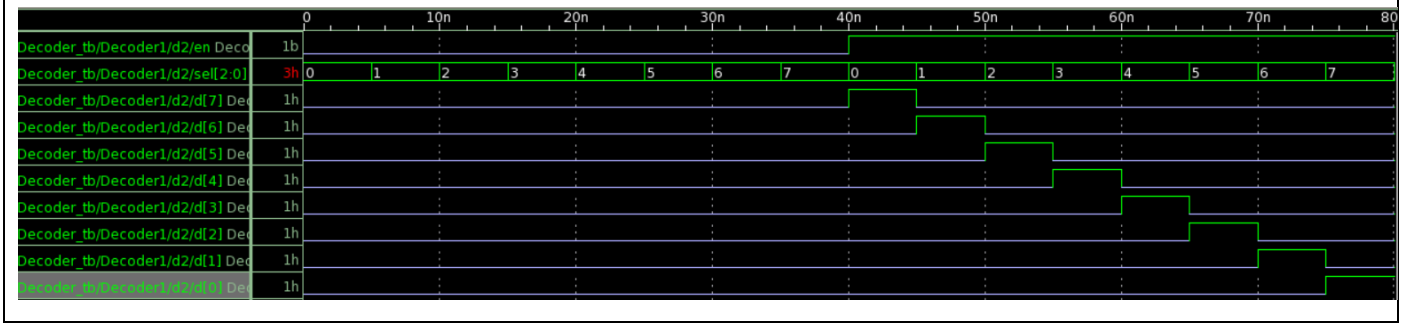
Case2

由於對 Decoder 進行完整的 simulation 所需要的 test pattern 僅需 16 項，因此我直接透過觀察 waveform 來檢測電路正確性。

Decoder1 Waveform (original)



Decoder2 Waveform



觀察 Decoder2 的波型可以知道 Decoder2 沒有被加入 hardware trojan，是一個功能正常的電路，但觀察 Decoder1 的波型會發現當 sel[2:0]等於 2 時 d[5]沒有正常動作，發現 Decoder1 受到感染。

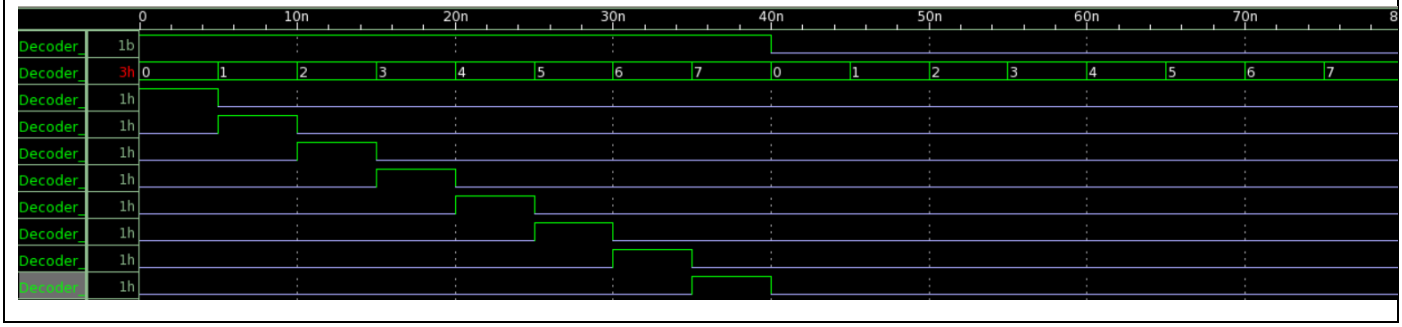
Hardware Trojan (s_bar[3])

```
not     n0( s_bar[3], en ); // trigger

// and     a1( d[5], s_bar[2], s_bar[3], s_bar[0], en ); // error, s_bar[3] is payload
and     a1( d[5], s_bar[2], sel[1], s_bar[0], en ); // s_bar[3] modified to sel[1]
```

檢查 Decoder1.v 中的 d[5]，會發現 sel 只有 3 bits 但是 s_bar 竟然有 4 bits，這就顯得很可疑了，果然一看就知道邏輯上應該將 s_bar[3]修改成 sel[1]才符合 Decoder 的功能。

Decoder1 Waveform (modified)

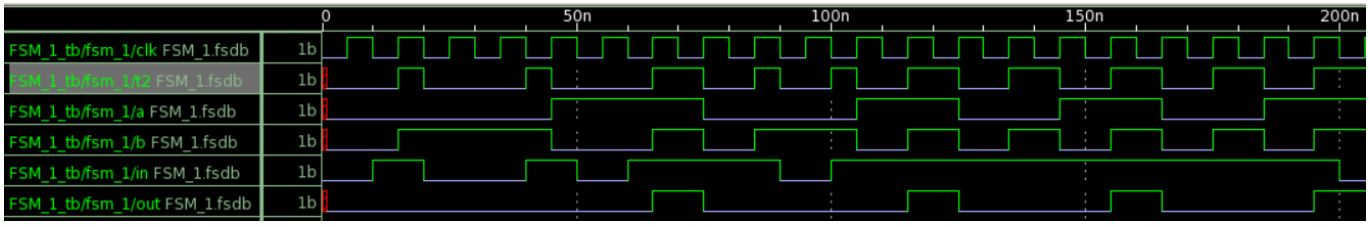


修改完後 Decoder1 波型也正常了，Case2 完成。

Case3

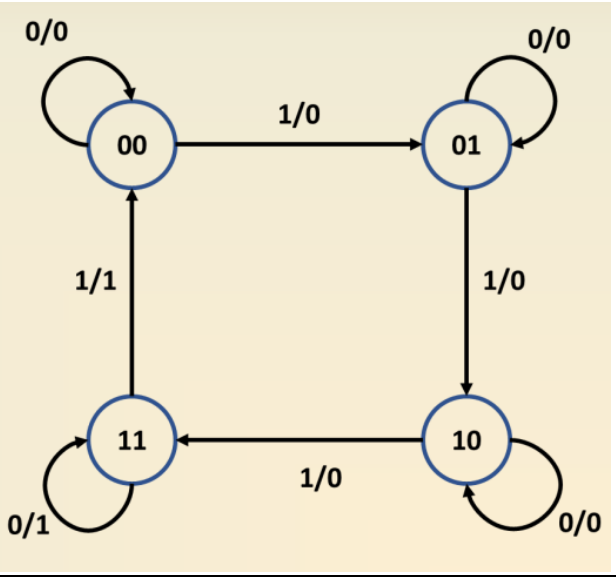
首先對電路進行 simulation，經過觀察 waveform 可以確定 T 型正反器的功能是正確的，排除 hardware trojan 存在於 TFF 的可能性，但是僅憑波型圖難以再進一步推測剩下的三個邏輯閘究竟是哪一個導致功能錯誤。

FSM_1 Waveform (original)



Boolean function Simplification

Current State		Input	Next State		Output
A	B	in	A	B	out
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1



RTL 的電路圖以布林函數進行表示式為

$A\_next = in \oplus A\_current$

$B\_next = [(in)(A\_current) + (A\_current)(B\_current)] \oplus B\_current$

透過真值表化簡後得到的布林函數進行表示式為

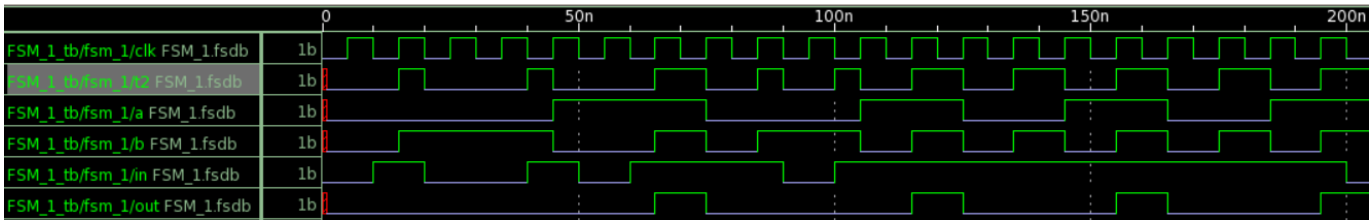
$A\_next = A\_current \oplus (in)(B\_current)$

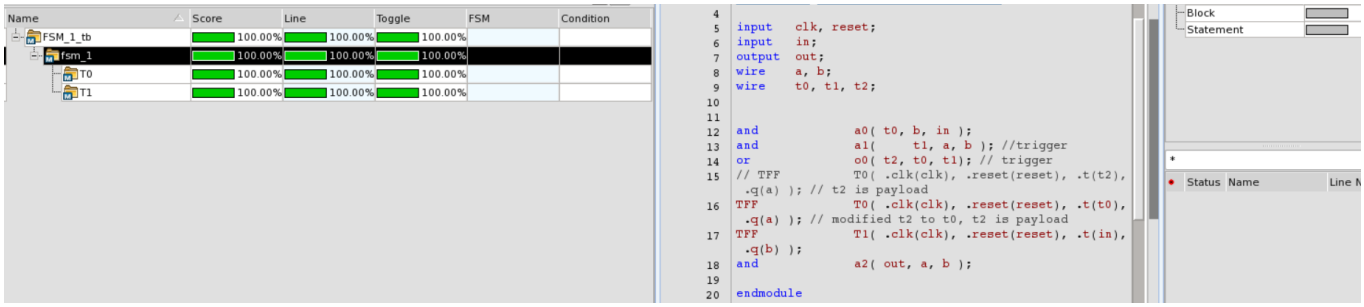
$B\_next = in \oplus B\_current$

比較兩者之間的差異會發現 t1、t2 都是非必要的電路十分可疑，因此嘗試將 t2 替換為其他訊號。

Hardware Trojan (t2) & FSM_1 Waveform & FSM_1 Verdi (modified)

```
and     a1( t1, a, b ); //trigger
or      o0( t2, t0, t1); // trigger
// TFF      T0( .clk(clk), .reset(reset), .t(t2), .q(a) ); // t2 is payload
TFF     T0( .clk(clk), .reset(reset), .t(t0), .q(a) ); // modified t2 to t0, t2 is payload
```



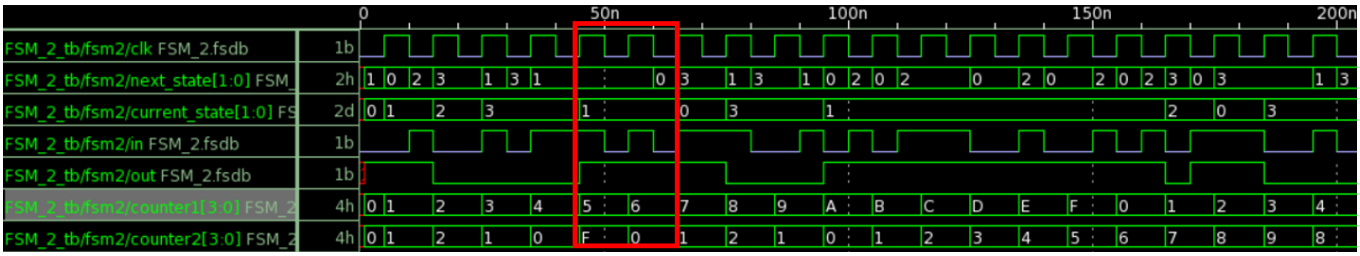


將 t2 修正為 t0 後再次比對波型與真值表確認電路功能正確無誤，同時各項 coverage 也都達到了 100%，因此確認 test pattern 完整的驗過這個電路，case3 完成。

Case4

我找尋這一個電路的 hardware trojan 的方法是先進行 simulation 再直接觀察轉態情形。

FSM_2 Waveform (original)



Hardware Trojan (counter2)

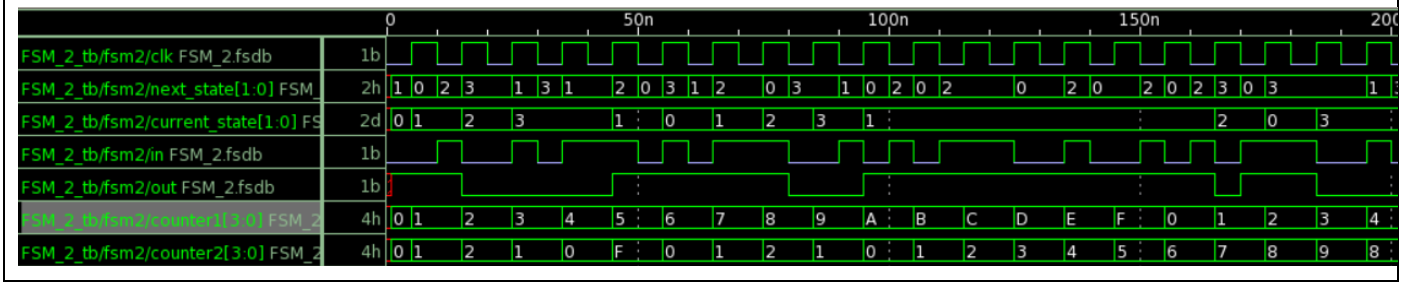
```
// trigger
if(out)
  counter2 <= counter2 + 1'b1;
else
  counter2 <= counter2 - 1'b1;
```

```
S1: begin
  if(!in) begin // correct
    if(counter2 < 4'd10/*1*/) begin // counter2 is payload
      next_state<=S0;
      out<=1'b1;
    end
    else begin
      next_state<=S1;
      out<=1'b1;
    end
  end
  else begin // correct
    if(counter2 < 4'd10/*1*/) begin // counter2 is payload
      next_state<=S2;
      out<=1'b1;
    end
    else begin
      next_state<=S1;
      out<=1'b1;
    end
  end
end//end S1
```

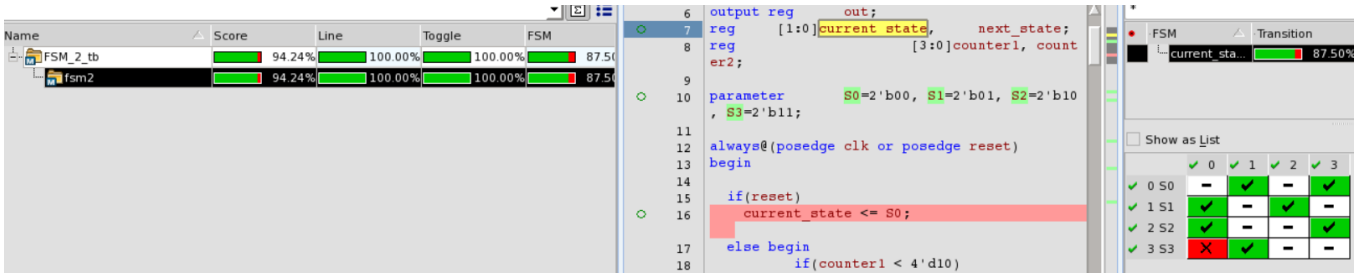
和真值表比對過後發現所有的錯誤都只會發生在 S1，特徵是除了 Idel Mode 以外不論 in = 0、1，next_state 都不可能是 S1，因此 S1 持續超過兩個週期很明顯指出 S1

受到感染。接著我進一步檢查 S1 的程式碼發現與 S0、S2、S3 不同的地方是 S1 多包了一層 if (counter2 < 4'd10) … else，正好也只有 S1 出問題，因此 counter2 極有可能就是 hardware trojan。

FSM_2 Waveform (modified)



FSM_2 Verdi (modified)



在修改 S1 中的判斷式以後，可以從 waveform 發現 S1 的問題已經解決，再次比對轉態圖以及真值表可以確認電路功能已正確無誤，最後再透過 verdi 確認除了一開始 reset 使 S3 轉態至 S0 以外其餘狀態都有跑到，line coverage 也達到了 100%，因此可以確信這份 testbench 確實完成了這個電路的功能驗證。

Hardness of this assignment

我覺得這次作業四個 case 分別適合使用不同的方法去思考，在剛開始著手一個 case 時需要花費一些時間去觀察題目、波型，接著才能想出要怎麼去找，另外這次我只寫出單一 case 的 testbench 產生器，將產生器通用化將會是未來的目標。