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 ) | |
|  |  |

|  |
| --- |
| 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] ) |
|  |
| 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] ) |
|  |
| 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 ) | |
|  |  |

**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] ) |
|  |
| 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 |
|  |
| RTL的電路圖以布林函數進行表示式為  A\_next = in ⊕ A\_current  B\_next = [(in)(A\_current) + (A\_current)(B\_current)] ⊕ 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 ⊕ (in)(B\_current)  B\_next = in ⊕ 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 ) |
|  |
|  |
|  |

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 ) |
|  |
| 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。