Name: 吳承蒼 **Student ID:** 312510158 **Account:** iclab008

(a) What is Formal Verification?

Formal Verification 是使用了一種透過數學運算來驗證設計正確性的一種驗證方式。進行 Formal Analysis 時，tool 會把所有的 input/wire 組合產生出來，以檢查其對應的輸出是否都有達到應滿足的條件。

What's the difference between Formal and Pattern based verification? and list the pros and cons for each.

相較於 Formal Verification，Pattern based Verification 就是我們熟悉的流程，也就是透過 pattern.v 來產生 input pattern 給 DUV，接著接收 DUV 產生出來的 output 來與正確答案比對，驗證 DUV 是否有功能上的不正確。以下為兩種驗證方式的優缺點：

Formal Verification

Pros:

1. 因為 Formal verification 會產生所有 input/wire 的組合，原本在 Pattern based 很難驗證的 corner case 反而在此驗證方式下會較容易被找到。
2. 由於產生了很多 input/wire 組合，以及受到了嚴格的數學推導及驗證，可以有較高的信心水準確保系統在任何的條件下都可以正常的運作，驗證上的品質較好。
3. 相較於 Pattern based，formal 可以不用寫 testbench，而是只要把 assertion 的條件寫好，就可以進行驗證。

Cons:

1. 此驗證方式的延展性非常受限，原因正是要產生所有 input/wire 的組合。小電路還可以使用窮舉法來驗證，但是當電路規模變大時，驗證的時間複雜度是指數型的上升，因此 Formal Verification 對大型且複雜的電路會是一個很大的負擔。
2. 只要電路愈大，需要產生的組合數量會急遽上升，是非常占用電腦資源的驗證方式。

Pattern-Based Verification

Pros:

1. 針對已知的問題所需的耗費時間較少。
2. 如果要驗證某些特殊的情境，使用 Pattern base verification 反而可以更有效率的驗證到系統的正確性。

Cons:

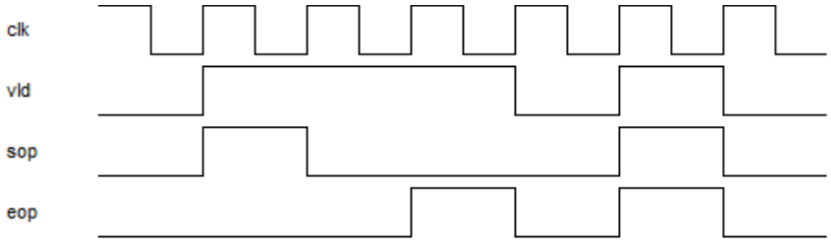
1. 驗證上的完整性無法如 Formal Verification 來的高，因為沒有將所有的組合產生出來，以驗證系統在當下的組合下能否成功運作。

2. 驗證的品質會受到 input pattern 品質的影響，假如所有的 input pattern 都只能驗證到簡單的 case，無法測到 corner case，則即使驗證通過了，仍可能會碰到沒驗證到的 corner case，導致電路出錯。

(b) What is glue logic? Why will we use glue logic to simplify our SVA expression?

當要 Model 一個複雜的電路行為時，SVA expression 可能會很複雜，因此與其直接 model 整個電路行為，借助一點額外的邏輯運算來輔助 SVA 的驗證反而會比較輕鬆，而這個專門用來輔助 SVA 的邏輯計算就是 glue logic。

Example:



• Pure SVA version:

```
sequence sop_seen:
  sop ##1 1'b1[*0:$];
endsequence;

no_holes: assert property(
sop |-> vld until_with eop
)
```

```
sop_first: assert property
          (vld && eop |-> sop_seen.ended)

eop_correct: assert property(
      not ( !sop throughout
            ($past(vld && eop) ##0 vld && eop[->1])
          )
sop_correct: assert property(
   vld && sop && !eop |=>
   not(!$past(vld && eop) throughout (vld && sop[->1]))
)
```

• Glue logic version:

```
reg in_packet;
always@(posedge clk)
   if (!rstn || eop) in_packet <= 1'b0;
   else if( sop)     in_packet <= 1'b1;
   else              in_packet <= in_packet;

no_holes_1: assert property( in_packet |-> vld) ;
no_holes_2: assert property( sop |-> vld) ;
```

```
eop_correct: assert property (
   vld && eop |-> in_packet || sop
);

sop_correct: assert property (
   vld && sop |-> ! in_packet
);
```

由以上例子可以看到，如果沒有加上 glue logic，SVA 為了要 model 電路行為就需要寫很多條 assertion，以達到驗證上的完整性。而有加上 glue logic 的這方則是額外增加了 in_pocket 的邏輯運算，讓 in_pocket 在 sop 與 eop 之間要維持 high，並透過這個 in_pocket 來進一步做 SVA，可以看見右方 assertion 的部分判斷的方式明顯簡單許多。

(c) What is the difference between Functional coverage and Code coverage?

Functional coverage

Functional coverage 著重在某些特定狀態、條件、或是 sequence 是否有被成功打到，換句話說，就是針對電路的特定功能來做 coverage 的計算。在計算 coverage 前，可能要先一一列出所有需要被驗證的功能，如 Lab10 所使用的 covergroup，因此 Functional coverage 可以當成是電路在功能性上的評估分數。

Code coverage

Code coverage 則是在檢查所有 code 的條件是否都有被執行到，比如說所有的 if/else statement、case 等等，是以 RTL code 的角度去規劃的 coverage。相較於 Functional coverage，code coverage 的觸發條件簡單許多，通常可以直接透過 tool 自動化的方式來列舉需要達到的所有 covergroup，以檢查是否每一行程式都有正確地被執行到。

What's the meaning of 100% code coverage, could we claim that our assertion is well enough for verification? Why?

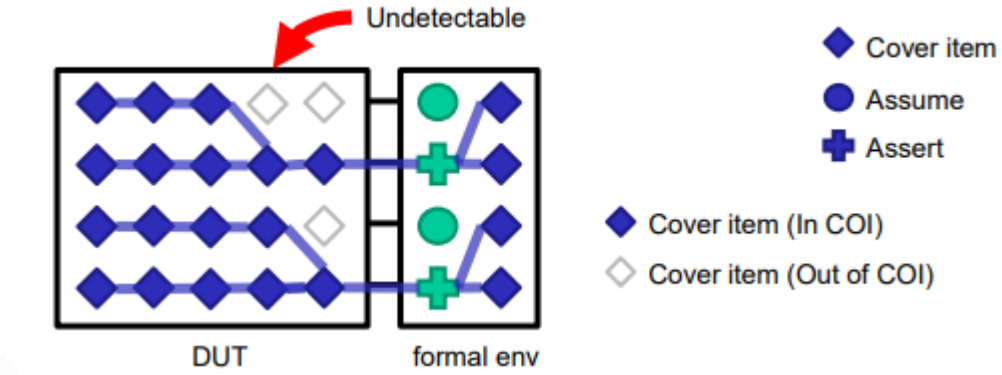
NO!

當達到了 100%的 code coverage，表示 design code 裡面的所有 branch、case 的執行次數都有達到標準，但嚴格來說這也只能代表所有 branch、case 有被執行到，卻沒辦法直接代表功能上的完整性。就以(b)裡 vld 要在 sop 到 eop 之間維持 1 的情況為例，這種電路行為的驗證就很難單純靠 code coverage 檢驗出來，但這種功能性驗證反而才是電路是否在功能上有正確運作的重要指標。

(d) What is the difference between COI coverage and proof coverage for realizing checker's completeness? Try to explain from the meaning, relationship, and tool effort perspective.

COI coverage

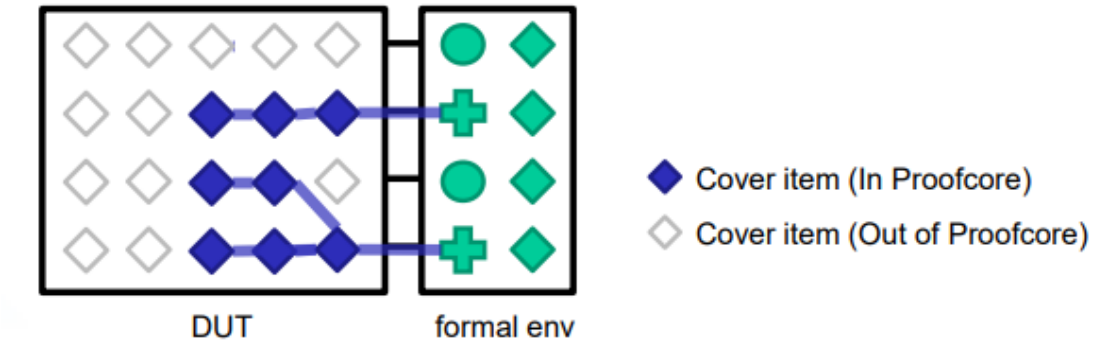
又稱 Cone-Of-Influence coverage，COI coverage 會從一個 register 開始往前回朔，只要會影響到此 register 結果的 item 都會被列入到 COI coverage 裡，這也意味著在這個 coverage 裡的所有 cover item 如果有變動，都有機會影響到 assertion 的結果，換句話說，如果 cover item 沒有出現在 COI coverage 裡，就可以間接說明此 cover item 是無法被驗證到的。COI coverage 是一個不需要使用 Formal engine 的方法，它只需要往前 retrace 就好，因此是一個相對快速的方法。



Proof coverage

Proof coverage 相較於 COI coverage，就會使用 formal engine 來進行驗證，以課堂講義上的例子為例：

在 COI coverage 裡，所有會影響到 assertion 的因子都會被列入到 COI coverage 裡，而在 proof coverage 中，由於會使用到 formal engine，在這個階段下是可以直接找出什麼因子會影響到 assertion，因此 proof coverage 的會比 COI coverage 的深度來的小，更準確地說，proof coverage 會是 COI coverage 的 subset，因為同樣是在尋找會影響 assertion 的因子，但 proof coverage 會在找到直接影響 assertion 的因子就停止，不會繼續 retrace 回去。最後由於會使用到 formal engine，proof coverage 的速度就會比 COI coverage 來的慢。



(e) What are the roles of ABVIP and scoreboard separately? Try to explain the definition, objective, and the benefit.

ABVIP

全名 Assertion Based Verification Intellectual Property。ABVIP 主要目的是在驗證 design 是否能與某個特定的 protocol 進行溝通，如 dram 所使用的 AXI Interface。ABVIP 主要是由一系列的 checker 所組成，裡面也裝下了許許多多的 assertion，使用 ABVIP 的主要目的就是讓設計者在面對複雜的 protocol 時，可以不需再自行寫一份 checker，而是拿現成且完整性較高的 ABVIP 來驗證是否能成功透過此 protocol 進行溝通，讓設計者可以更輕鬆地完成設計的步驟，並且優點是驗證的正確性較高。

Scoreboard

Scoreboard 的角色類似一個 monitor，當 formal engine 給 DUV 輸入的同時也會給 scoreboard 一樣的輸入，接著當 DUV 計算完輸出後，會把結果再一次傳給 formal engine 和 scoreboard，scoreboard 這時就會將輸入與輸出作互相做比對。除了訊號之正確性外，scoreboard 還可以檢查：

1. Data Packet Dropped (資料是否不見)
2. Duplicate Data Packet (重複的數據)
3. Order of Data Packet (是否照順序)
4. Corrupted Data Packet (是否符合預期)

綜上所述，scoreboard 主要目的為檢察訊號的正確性，另外優點則是降低了 state-space 的複雜度。

(f) List four bugs in Lab Exercise. What is the answer of the Lab Exercise?

1. bridge.sv @ line 90
   X If (inf.AR_READY) inf.AR_VALID <= 1'b1;
   V If (n_state == AXI_AR) inf.AR_VALID <= 1'b1;
2. bridge.sv @ line 124
   X If (inf.AW_READY) inf.AW_VALID <= 1'b1;
   V If (n_state == AXI_AW) inf.AW_VALID <= 1'b1;
3. bridge.sv @ line 134
   X inf.AW_ADDR <= {8'h1000_0000, inf.C_addr, 2'b0};
   V inf.AW_ADDR <= {1'b1, 7'b0, inf.C_addr, 2'b0};
4. bridge.sv @ line 145
   X if (inf.C_in_valid && inf.C_r_wb)
   V if (inf.C_in_valid && ! inf.C_r_wb)

(g) Among the JasperGold tools (Formal Verification, SuperLint, Jasper CDC, IMC Coverage), which one have you found to be the most effective in your verification process? Please describe a specific scenario where you applied this tool, detailing how it benefited your workflow and any challenge you encountered while using it.

我認為目前感到最實用的，是這次的 Lab 所使用的 Formal Verification，尤其是這次使用 AXI 的 ABVIP，讓原本複雜的驗證過程變成只需要按下一個按鈕就可以進行驗證，不需要特地去製作一個 pattern 來驗證 AXI 的運作是否合法，真的輕鬆許多。不過在 Lab07 的 FIFO 所使用的 CDC，我認為 tool 在判定 convergence 的部分不太直觀，當時從錯誤訊息欄裡打開電路圖找到哪些訊號出現了 convergence，於是嘗試對這些訊號進行修改，沒料到修改完後卻變成另一個訊號產生了 convergence 的問題，多次來回修正後反而又回到了一開始發生的問題，當時在解決這方面的問題花了不少時間。因此關於 convergence 的部分，是否能有機會更清楚點出發生 convergence 問題的根源，或是列出可能的解決方法，這樣在 CDC 電路的設計上也許能夠更加順暢。