IC Lab Formal Verification
Bonus Quick Test
2025 Fall

Name: 余俊瑋 Student ID: 314511034 Account: iclab160

(a)What is Formal verification?

Formal Verification 是使用數學的方式證明，對 DUT 的 State 進行窮舉，驗證其功能正確性。

What's the difference between Formal and Pattern based verification?

And list the pros and cons for each.

Formal Verification:

1. Tool 會自動探索所有可能的輸入。
2. 不需要大量的測試向量。
3. 較容易抓到 Corner Case。
4. 只是可能遇到 State 爆炸，導致時間過久，所以可以搭配 Assumption。

Pattern-Based:

1. 需要人工撰寫 Testbench + Pattern。
2. 較容易理解，且較適合複雜的系統。
3. 需要特別思考 Corner Case 的情況，需要更加細心。

Method	Pros	Cons
Formal Verification	自動探索所有可達的狀態，較容易抓到 Corner Case。	較容易遇到 state-space-explosion, 尤其是當設計過於複雜，窮舉出所有 state 需要花很多時間。
Pattern Based Verification	較易於理解且直覺。	需要花時間思考 Corner Case, 要達到高 Coverage, 需要大量 Test Pattern。

(b)What is glue logic?

Why will we use glue logic to simplify our SVA expression?

Glue Logic 是在 DUT 旁額外加入輔助邏輯(通常是暫存器、flag, counter 等等)，用來追蹤某些事件或是時序狀態。它本身並不會改變 DUT 的行為，只是協助更容易描述複雜的驗證條件。當某些行為用 SVA 描述很複雜時，使用 Glue Logic 來觀察、紀錄能大幅減少 SVA 的複雜度。

我們使用 Glue Logic 是因為它能把複雜的 SVA 拆解成易懂的狀態，使 Property 更精準、更短，因此是寫複雜行為時好用的技巧。

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

Functional Coverage 是依照 Spec, 由工程師自己定義的 Coverage: 可以是 property 相關的，也可以是 covergroup, 針對變數值、轉換、cross 等組合做 bin。其特色是可以直接對應規格上有哪些情境需要被驗證，但是需要人為設計，需仰賴時間與經驗。

Code Coverage 是由 EDA Tool 自動從 RTL 中萃取的結構性 Coverage, 可以分成以下幾種:

(i) Statement Coverage: 哪幾行 assignment, statement 被執行過。

(ii) Branch Coverage: if/else 、case、?: 的分支是否都被走過。

(iii) Expression Coverage: 複雜的 Boolean Expression 中是否每個 Operand 都與輸出相關。

兩者的差異: Functional Coverage 可以測出: 我想驗證的 SPEC 是否有符合; 而 Code Coverage 則是在乎是否所有可達的 statement / branch / expression 都曾被觸發過。

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

不能，理由如下:

(i) Proof Coverage 只會找出真正影響 assertion 結果的 proof core, 是 COI 的子集合，要跑 formal engine 才能知道。如果某段 code 在 COI 外，或是 proof core 外，即使 code coverage hit, 也代表沒有實際被 formal 嚴格檢查。

(ii) 100% code coverage 只確保「跑過某行 code」, 並不表示所有的 Corner Case 都有真正的被檢驗到。

(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.

1.Meaning

COI 是所有可能「影響某條 assertion」的邏輯組合。它是純結構性分析，不跑 formal, 如果某段邏輯在 COI 之外，表示即使該邏輯發生錯誤，也永遠不會被 assertion 到。

Proof Coverage 是在 formal 引擎實際證明 assertion 的過程中，那些邏輯真的被使用，是 COI 的子集合。

2. Relationship

$$\textit{Proof coverage} \subset \textit{COI coverage}$$

COI 可以告訴我們 assertion 最廣可以檢查到的區域，而 Proof Coverage 告訴我們實際工具檢查到哪裡。

3. Tool Effort

COI Coverage: 快速、靜態分析。僅做 dataflow 以及相依性分析，不需要 formal state-space exploration, 適合早期用來找 assertion 是否寫太少。

Proof Coverage: 需要嘗試找 assertion failure, cover witness, 或走道 full proof, 計算量大，時間可能很長，是真正反映 formal 檢查到什麼的 coverage。

(e)What are the roles of ABVIP and scoreboard separately?

Try to explain the definition, objective, and the benefit.

ABVIP:

1. Definition:

是一組現成的檢查器，針對某個 protocol 準備好 assertion & cover, 只要將 bus signal 連接好，就可以直接做 protocol compliance check。

2. Objective:

確認我們的電路設計有滿足通訊協定規範, 例如 AXI 的 handshake 順序, burst 長度等等。

3. Benefit

由 Tool 或 IP 廠商多年整理、修正的版本，可信度高，且導入的效率高，不用自己寫驗證，覆蓋也較自己寫的完整。

Scoreboard:

1. Definition:

觀察 Design 的 input /output data & control, 並建立一個預期的資料 sequence, 與實際的輸出進行比對。

2. Objective:

檢查資料層級的正確性，而不只是 protocol 是否符合規範。

3.Benefit

自動化 Data integrity check, 不需要手寫 input / output 配對的 assertion，且支援不同的 ordering (in-order / out-of-order), 另外，也與 formal 引擎結合，Tool 會嘗試所有可能情境，找出 data 相關的 bug。