IC Lab Formal Verification
Bonus Report 2024 Fall

Name: 陳孟頡 **Student ID:** 110511118 **Account:** iclab113

(a)

1. **What is Formal verification?**

Formal verification 是一種驗證所設計電路功能正確性的方法，以窮舉輸入訊號組合來達成檢驗，確保電路下線前具備完整功能。

Formal verification 以一個 cycle 為單位，在第一個 cycle 確認 uninitialized registers 的所有可能數值，同時會在每一個 cycle 檢查 input 及 undriven wires 的所有數值組合。

測試完所有組合後，會將違反 specifications 的 assertion properties，以及 cover properties 輸出為 report 檔案，以便進行除錯。

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

Formal verification:

- 利用窮舉測試資料驗證電路
- 能確保電路對所有可能輸入資料、系統狀態都能正確運作
- 測試時間長、覆蓋率高

Pattern verification:

- 利用設計者預先定義的測資進行驗證
- 以已知資料組合進行驗證，可能並未包含所有輸入可能，
- 測試時間較短、覆蓋率可能較低

	Formal verification	Pattern verification
Stimulus	窮舉輸入組合	設計者定義測資
Coverage	基本上 100%	<100%，涵蓋特定範圍
Time Cost	耗時較長	耗時較短

3. **And list the pros and cons for each.**

Formal verification:

- Pros:
  - 確保電路在所有情況皆能正常運作
  - 能偵測細微錯誤、corner case 正確性

- Cons:
  - 相當耗費計算資源及測試時間
  - 通常需要額外的 EDA 工具協助進行
  - 可能難以在大型系統設計中使用

Pattern verification:

- Pros:
  - 驗證速度較快、靈活度高
  - 易於使用及實作

- Cons:
  - 可能遺漏 corner case，難以達到 100% 覆蓋率
  - 需設計者詳細撰寫，才能達到嚴謹的驗證

(b)

1. **Explain SVA (SystemVerilog Assertions) and the roles of Assertion, Cover, and Assumption.**

SystemVerilog Assertion 建立於 SystemVerilog 硬體描述語言上，用來表示 assertions, covers 以及 assumptions 此些 properties，並可與 Verilog、VHDL 混用。其主要功能，為檢驗電路在特定條件狀況下的行為，是否符合設計。

- Assertion:

  定義電路運作時必須時刻保持的 properties。若違反 assertion，表示電路行為有錯，設計中有錯務須進行調整。

- Cover:

  著重於 coverage metric，用以確定驗證電路的測試資料涵蓋各種不同情境及條件。些覆蓋率過低，可能存在特定測試資料組合，會使電路行為錯誤。

- Assumption:

  設定電路的輸入或環境 constraints 或是特定行為，用來限制驗證時的 state space。

2. **What is glue logic?**

撰寫 SVA 時，時常出現重複的行為訊號，若是每次皆直接撰寫，容易使程式邏輯變得相當複雜。而 glue logic 便是用以解決此問題的解方，其運用簡單的額外邏輯訊號，取代這些經常性重複出現的行為訊號，讓其他 module、function 可以直接使用，達到簡化邏輯的目的。

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

以下列出幾點用 glue logic 簡化 SVA 之原因:

- Modularity:
  使用 glue logic 能將原先複雜的 assertion 簡單化，並增加在其他 module 使用的便利性。

- Clarity:
  使用 glue logic 能增加程式的可讀性，對於獨立電路開發或是大型系統合作皆有所幫助。

- Efficiency:
  將複雜的邏輯簡化後，不只增加使用效率，還能減少模擬及驗證時的 computation cost。

(c)

1. **What is the difference between Functional coverage and Code coverage?**

Functional coverage:
- 驗證電路的設計功能及行為，檢查設計者定義的 assertion 以及 covergruop。
- 著重於 high-level 電路功能及執行狀況是否符合設計要求。

Code coverage:
- 檢查 HDL 被執行的狀況，程式層面的設計正確性。
- 確認程式的 branch、statement、expression 是否無誤，並不直接檢查電路功能正確性。

	Functional coverage	Code coverage
Verify	驗證設計功能、covergroup	每一行程式是否被正確執行
Conduction	設計者定義並執行	軟體自動執行
Time Cost	較長	較短
Problem	可能有人為疏失，產生驗證不完全	只保證程式正確，可能存在電路功能錯誤

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

即使 code coverage 達到 100%，還是無法表示 assertion properties 也被完整驗證。

100% 的 code coverage 代表程式中每一行、每個 branch、每種 condition 皆被正確執行，確保設計的 HDL 正確無誤，但無法保證電路功能的正確性，以下為幾點原因:

- Code behavior:
  程式覆蓋率只代表程式的正確執行，與電路功能完整性並不相等。

- Uncovered function scenario:
  可能存在並未涵蓋在程式中的電路行為及狀態，並不會被 code coverage 檢查到。

- Property scopes:
  Assumptions 及 constraints 可能沒有考慮到 corner case，並未包含完整功能檢驗。

(d)

1. **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.**

- Meaning:
  在 formal testbench analysis 中，分為 stimulus coverage 以及 checker coverage。Stimulus coverage 主要檢測 input space，而 checker coverage 則著重於 property space。

  Checker coverage 中，又分為 COI coverage 以及 proof coverage，COI coverage 偵測所有可能會影響到 property 或 assertion 的電路訊號，proof coverage 使用 formal engine 進行檢驗，只包含 assertion 內實際的訊號。

- Relationship:
  COI coverage 涵蓋所有可能訊號，因此 proof coverage 的實際訊號也在其中。也就是說，proof coverage 為 COI coverage 的一個子集合，若電路中所有 assertion 相關訊號都有實際影響，則 proof coverage 會等於 COI coverage。

- Tool effort:
  由於 proof coverage 代表真正影響的訊號，因此需要以 formal engine 進行驗證，會消耗較 COI coverage 多的資源，需要的驗證時間也較長。

(e)

1. **What are the roles of ABVIP and scoreboard separately? Try to explain the definition, objective, and the benefit.**

- ABVIP
  - Definition:
    Assertion based verification IP 為用來驗證 protocol 或 interface 的 checker 及 RTL 集合。
  - Objective:
    與 pattern 相似，模仿與 DUT 互動的介面，檢查 protocol 或 interface 的正確性。
  - Benefit:
    方便使用，簡化 testbench 的開發複雜度及成本，同時確認 protocol 的正確性，降低錯誤發生率。
- Scoreboard:
  - Definition:
    驗證設計的監視器環境，追蹤並觀察輸入數值、預期輸出數值及真實輸出數值的關係。
  - Objective:
    檢查 DUT 的輸出訊號與 reference model 或 golden output 的數值是否一致。
  - Benefit:
    以 high-level 層級來進行除錯，減少 state-space 複雜度，增加檢驗便利性。

(f)

1. **Among the JasperGold tools (Formal Verification, SuperLint, Jasper CDC, IMC Coverage), which one do you think is the most effective based on its functionality and typical application scenarios? Please explain your reasoning by describing a hypothetical scenario where this tool would be particularly beneficial, and discuss any potential challenges or limitations that might arise when using it.**

- Tool of choice: **Jasper CDC**
- Reasoning:
  隨著製程技術、電路優化不斷演進，現代的數位電路設計往往使用多個 clock domain 來達到效能以及功耗的最佳化，因此，如何確保跨時域(clock domain crossing, CDC)電路的功能正確性，是時下完成複雜電路設計的必要技能之一。

- Hypothetical Scenario:
  假設我們需要實現一個系統電路的整合，其中包含兩個不同速度的子電路，則所需考慮並進行驗證的項目有下:

  1. Metastability:
     確保電路取值正常，沒有發生訊號不穩的情形(沒有 unknown 的產生)

  2. Data Loss or Duplication:
     檢測兩邊的取值情形，確認無訊號掉失或是重複等問題。

  3. Protocol Violations:
     確保兩個子電路正確進行 handshake，沒有 protocol 的錯誤。

- Benefits:
  針對上述問題，Jasper CDC 能提供的幫助及益處有:

  1. Detect Clock Domain Issues:
     Jasper CDC 能自動偵測電路中的 CDC 路徑並進行驗證，且因為 Jasper CDC 是專門為 CDC 問題所設計之工具，相較其他驗證方法會來的有效率，同時具備可靠性。

  2. Check Protocol Adherence:
     Jasper CDC 能檢查電路中的 CDC 同步機制是否符合規範，避免資料丟失或是 metastability 情形的發生。

  3. Provide Debugging Insights:
     針對出錯的地方提供簡單的說明，方便使用者進行除錯。

- Potential Challenges or Limitations:
  1. Detailed Debugging Insight Needed:
     Jasper CDC 所提供的除錯資訊有限，時常令人摸不著頭緒，只能隨機嘗試修改電路，試著出錯誤之處。

  2. Complexity in Mixed Domains:
     若電路系痛過於龐大，且橫跨多的時域，可能需要而外的 EDA tool 或是人為輔助進行完整的分析檢驗。