IC Lab Formal Verification
Bonus Quick Test
2023 Fall

Name: 郭婕榆 **Student ID:** 311510170 **Account:** iclab064

1. Bonus:

(a) What is Formal verification?

A：Formal verification 是一種系統性的驗證方法，它會每個 cycle 去跑過所有可能的 stimulus (包括 inputs, undriven wires at every cycle 以及在第一個 cycle 時未初始化的 register 所有值的組合)，並檢查是否符合我們定義好的 assertion 規則，同時計算 coverage。這樣就可以產生大量不同的 register 和 input 值的組合，以檢查是否存在潛在的 bug。

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

A：Formal verification 比較像是一種 BFS 的演算法，會在每個 cycle 跑過所有可能的情況，進而實現 100%的 coverage，因此幾乎沒有 randomization。而 Pattern based verification 比較像 DFS 演算法，通過 random testing 和 directed testing 的方式在每個 state 形成一條路徑，並透過這條路徑來驗證輸出是否符合預期，但這些路徑可能沒辦法像 Formal verification 一樣測試到 stimulus 和 register's state 的所有可能性，所以可能無法達到 100% coverage。

And list the pros and cons for each.

- ✓ **Formal verification**

  Pros :

  i. Systematic method : 幾乎沒有 randomization

  ii. Less testbench effort required : 不需要另外產生 testbench

  iii. Improves productivity and schedule :

  Verification can begin prior to testbench creation and simulation, can also replace some block- level testbenches

  iv. Leads to higher quality

Cons:

i. 在大型 design 上會有較高的複雜度，會有太多可能的 condition 要做驗證

✓ **Pattern based verification**

Pros :

i. Easier to cope with large designs

Cons :

i. 有可能無法達到 100% coverage，或需要花很大的 effort 找 corner case

ii. Need effort to generate testbench

(b) What is glue logic?

A：在模擬一些較複雜的 behaviors 時，SVA 可能變得過於複雜，可以透過加入 auxiliary logic 來 observe 和 track events，這些 logic 就被稱做 glue logic

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

A：因為 SVA expression 可能變得很複雜，我們可以用 glue logic 讓這些 SVA expression 變得較容易理解，像是用許多簡單的 glue logic 組合來檢查 assertion 以取代複雜的 SVA expression，此外 glue logic 不需要 extra price，因為合成軟體在優化電路時大機率會把它們優化掉。

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

A :

✓ **Functional coverage :**

Functional Coverage 著重驗證設計功能的完整性，會去檢查特定狀態、條件或 sequence 是否得到驗證。透過 property's related 的 covers 和 covergroups 進行 meaningful 的 design functionality and implement the verification plan，需要做 planning, coding and debugging，也可能因為 human error 使其不完整。

✓ **Code coverage :**

會去檢查 code 中的所有可能情況，像是 branch, statement, expression，比起 Functional coverage，它可能無法驗證到所有 meaningful 的 design functionality，它的優點是 easy to generate automatically，但可能因為 coding 上的問題受到影響(ex : unreachable default cases)

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

A：100% code coverage 表示每一行的 RTL code 都有被執行過至少一次，並不表示他的所有 functionality 是都有經過驗證，在某些 corner case 可能會有錯誤，故無法保證 assertion 是 enough for verification。

(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 :**

Finds the union of all assertion COIs (Cone of Influence, the region of cover items which affect each assertion) and the holes in the assertion set (code that is not checked by any asserts).

✓ **Proof coverage :**

Find the region cannot truly influence assertion status and represents the portion of design verified by formal engines.

✓ **Difference :**

i. Proof coverage 是 COI coverage 的子集

ii. COI 不需要證明就能 implement，而 Proof coverage 則需要

iii. Proof coverage 可以比 COI 識別出更多 unchecked code 且需要用到 formal engines 來驗證，因此需要更多的 tool effort

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

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

✓ **ABVIP :**

Definition : The Assertion Based Verification Intellectual Properties (ABVIPs) is a comprehensive set of checkers and RTL that check for protocol compliance.

Objective : ABVIP 的目標是通過提供全面的 set of assertion based on protocols 以提高驗證過程的準確性和效率

Benefit : The benefit of the ABVIP which has been well-designed is to save the designers' time to verify theirs' design.

✓ **Scoreboard :**

Definition : Scoreboard can monitor the input data and output data of the Design Under Verification (DUV). It can also check data packet dropped, duplicated data packets, order of data packets, corrupted data packets.

Objective : Scoreboard 的目標為 track and observe 正在進行測試的 design data，確保其正確性

Benefit : 可以降低 state-space complexity、協助 designer 更容易 detect error 和 debug

(f) List four bugs in Lab Exercise

What is the answer of the Lab Exercise?

i. W_DATA 在寫入的時候給值錯誤(145 行)

Wrong :

```
140  always_ff@(posedge clk or negedge inf.rst_n) begin
141      if(!inf.rst_n)begin
142          inf.W_DATA  <= 'b0;
143      end
144      else begin
145          if(inf.C_in_valid && inf.C_r_wb)      inf.W_DATA  <= inf.C_data_w;
146          else                                  inf.W_DATA  <= inf.W_DATA  ;
147      end
148  end
```

Correct :

```
140  always_ff@(posedge clk or negedge inf.rst_n) begin
141      if(!inf.rst_n)begin
142          inf.W_DATA  <= 'b0;
143      end
144      else begin
145          if(inf.C_in_valid && !inf.C_r_wb)      inf.W_DATA  <= inf.C_data_w;
146          else                                  inf.W_DATA  <= inf.W_DATA  ;
147      end
148  end
```

ii. AR_VALID 拉起的判斷錯誤，應該是 AR_VALID 先拉起為 1 輸出給 dram 要讀的 address 後 dram 才會輸出 AR_READY 為 1 (90 行)

Wrong:

```
85  always_ff@(posedge clk or negedge inf.rst_n) begin
86      if(!inf.rst_n)begin
87          inf.AR_VALID <= 'b0;
88      end
89      else begin
90          if(inf.AR_READY)  inf.AR_VALID <=  1'b1;
91          else                  inf.AR_VALID <=  1'b0;
92      end
93  end
```

Correct:

```
85  always_ff@(posedge clk or negedge inf.rst_n) begin
86      if(!inf.rst_n)begin
87          inf.AR_VALID <= 'b0;
88      end
89      else begin
90          if(n_state == AXI_AR)  inf.AR_VALID <=  1'b1;
91          else                   inf.AR_VALID <=  1'b0;
92      end
93  end
```

iii. AW_VALID 拉起的判斷錯誤，應該是 AW_VALID 先拉起為 1 輸出給 dram 要寫入的 address 後 dram 才會輸出 AW_READY 為 1 (124 行)

Wrong:

```
119  always_ff@(posedge clk or negedge inf.rst_n) begin
120      if(!inf.rst_n)begin
121          inf.AW_VALID <= 'b0;
122      end
123      else begin
124          if(inf.AW_READY)    inf.AW_VALID <=  1'b1;
125          else                    inf.AW_VALID <=  1'b0;
126      end
127  end
```

Correct:

```
119  always_ff@(posedge clk or negedge inf.rst_n) begin
120      if(!inf.rst_n)begin
121          inf.AW_VALID <= 'b0;
122      end
123      else begin
124          if(n_state == AXI_AW)   inf.AW_VALID <=  1'b1;
125          else                    inf.AW_VALID <=  1'b0;
126      end
127  end
```

iv. 用 16 進位表示 8bit 應該只有 2 個數(1 個數可以有 4bit)，但在這邊卻有 8 個數(134 行)

Wrong:

```
129  always_ff@(posedge clk or negedge inf.rst_n) begin
130      if(!inf.rst_n)begin
131          inf.AW_ADDR <= 'b0;
132      end
133      else begin
134          if(n_state == AXI_AW && c_state != AXI_AW)   inf.AW_ADDR <= {8'h1000_0000, inf.C_addr, 2'b0};
135          else                                        inf.AW_ADDR <= inf.AW_ADDR ;
136      end
137  end
```

Correct:

```
129  always_ff@(posedge clk or negedge inf.rst_n) begin
130      if(!inf.rst_n)begin
131          inf.AW_ADDR <= 'b0;
132      end
133      else begin
134          if(n_state == AXI_AW && c_state != AXI_AW)   inf.AW_ADDR <= {1'b1, 7'b0, inf.C_addr, 2'b0};
135          else                                        inf.AW_ADDR <= inf.AW_ADDR ;
136      end
137  end
```

(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.

我覺得是 Jasper CDC，在 lab7 的時候進行 FIFO、handshake 設計，利用 JG 可以很容易幫我們找出 CDC 中可能會有的問題，包括 convergence, signal not stable for 3 edge 等問題，這些在設計後跑 pattern 時可能無法發現的問題，使用 JG 可以另外幫我們檢查到，減少設計失誤的可能性。但在解決 convergence 的問題時因為較不理解 JG 檢查判斷的標準也因為是第一次做 CDC 的設計，所以這部分嘗試了很多種不同的寫法。