IC Lab Formal Verification
Lab11 Quick Test
2023 Fall

Name: 張宸瑜 **Student ID:** 312510141 **Account:** iclab012

(a) Formal Verification

1. What is Formal verification?

Formal verification 會利用窮舉的方式測試所有可能的 input stimulus

在每個 cycle 去測試再 input port 和 undriven wires 的所有 value 的組合

在第一個 cycle 去測試 uninitialized registers 的所有 value 的組合

最後把被 violated 的 assertion properties 以及被 hit 的 cover properties 輸出成 report

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

And list the pros and cons for each.

Formal verification => 如上面所述，formal verificaiton 會窮舉電路中 (input port、wires……) 所有的 value combination。優點就是可以覆蓋所有 common or corner cases，基本上通過 formal verification 的 design 就可以說是 100% correct。缺點則是因為要窮舉所有可能性，所需的驗證時間則會根據 design 的複雜度成正比增長。

Pattern verification => 此種驗證方式則是通過 designer 自行撰寫 patttern 測試檔，每次只針對一種 input 組合 (內部訊號組合也是一種) 進行測試。因此此種方式的優點很明顯，因為 designer 可以控制所要驗證的 pattern 數量，因此驗證時間相較於 formal verification 較短。缺點則是較難驗到 corner case，像是在前幾次 lab，design 的 input 都是 random 產生，就有可能發生在自己測試電路時功能一切正常，但助教在 demo 時就會出現錯誤的狀況。

(b) Glue Logic

1. What is glue logic?

Glue Logic 是利用額外的邏輯訊號去替代掉常出現的行為訊號，藉此實現簡化邏輯的目的。像講義上所舉的例子，若用一般 SVA 的寫法，每一個 assert property 裡面所寫的邏輯相當複雜。如果使用 glue logic，多宣告一個變數 in_packet，並使用 in_packet 訊號去描述原本複雜的邏輯，進而達到簡化的目的。

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

就如同上面所說，若單純使用 SVA，如果遇到複雜的 aseertion 要進行邏輯描述，會需要寫很長又複雜的邏輯式。但如果使用 Glue Logic，把一些可以重複用到行為描述用一個變數進行 paraphrase，如此一來就可以讓 assertion 裡面的 behavior description 更加簡潔。

(c) Coverage

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

Functional Coverage => 進行設計功能上的 assertion 以及結果 covergroup 的驗證。但需要 desginer 自行去 planning, coding, debugging，因此較為耗時。而且因為是人為進行設計，很有可能會因人為疏失發生驗證不完全等問題。(covergroup hit, sample time 錯誤等問題)。

Code Coverage => 直接檢查 design 裡面每一行 code 是否有被正確執行到。就不需要 designer 自行撰寫檔案，可以用軟體 check automatically。但因為只是檢查每一行 code 是否有被執行到，因此不能保證功能上的完全正確性。而且也可能出現 unmeaningful coding 等問題。

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

不行 / 就像上面所述，code verification 只是檢查 design 裡面的每行 code 是否有被執行到，並不能保證 design 的功能性正確。因此 100%的 code coverage，只能說 deisgn 裡面的每行 code 有被執行到，至於 design 的功能是否正確，還需要使用 functional or checker coverage 去進行驗證才行

(d) Coverage

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:

COI coverage 和 Proof coverage 合起來為 checker coverage，主要用於彌補 Stimuli Coverage 無法確保 design 的 100% correctness 的不足。以下列出 COI 和 Proof 的不同

COI Coverage => 當 designer 在撰寫 assertion 時，COI Coverage 指的是 code 裡面的所有會影響到此 assertion 的訊號

Proof Coverage => Proof coverage 則會進行 formal proof，而只包含 assertion 裡面所實際包含到的訊號

Relationship:

如 meaning 所述，COI 是包含 code 裡面所有影響到此 assertion 的訊號，Proof 則是指 assertion 裡面所實際包含到的訊號。因此，Proof Coverage 為 COI coverage 的 subset

Tool Effort:

同樣如 meaning 所述，因為跑 Proof Coverage 會需要進行 formal proof，因此相較於 COI coverage 會需要更多 running time and resources

(e) ABVIP & Scoreboard

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

Definition:

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

Scoreboard: Scoreboard behaves like a monitor used to observe input data and output data of DUV.

Objective:

ABVIP: provide a well-developed tool for designers to check those existed protocol.

Scoreboard: Check the consistency of input and output at design high level.

Benefit:

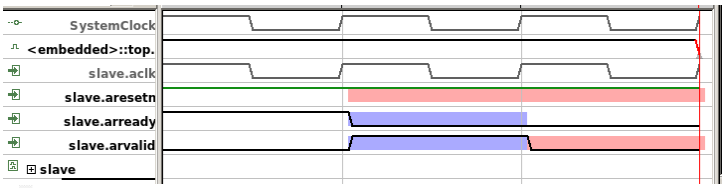
ABVIP: Save time for designing assertion and ensure the correctness.

Scoreboard: Formal optimized to reduce state-space complexity (to reduce barrier for adoption)

(f) Bug Hunting

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

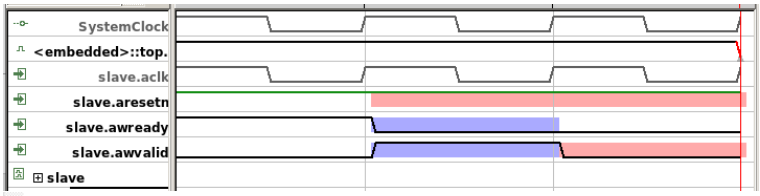
1. inf.arvalid 和 inf.arready 應該要在 inf.arvalid 變 0 的同時變 0




```
//////////////////////////////////////////////////////////////
//                  Before correct
//////////////////////////////////////////////////////////////
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AR_VALID <= 'b0;
    end
    else begin
        if(inf.AR_READY)  inf.AR_VALID <=  1'b1;
        else              inf.AR_VALID <=  1'b0;
    end
end

//////////////////////////////////////////////////////////////
//                  After correct
//////////////////////////////////////////////////////////////
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AR_VALID <= 'b0;
    end
    else begin
        if(n_state == AXI_AR)  inf.AR_VALID <=  1'b1;
        else                   inf.AR_VALID <=  1'b0;
    end
end
```

【Fig.1】Bug1 & Correct

2. inf.awvalid 和 inf.awready 應該要在 inf.awvalid 變 0 的同時變 0




```
//////////////////////////////////////////////////////////////
//                  Before correct
//////////////////////////////////////////////////////////////
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AW_VALID <= 'b0;
    end
    else begin
        if(inf.AW_READY)    inf.AW_VALID <=  1'b1;
        else                inf.AW_VALID <=  1'b0;
    end
end

//////////////////////////////////////////////////////////////
//                  After correct
//////////////////////////////////////////////////////////////
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AW_VALID <= 'b0;
    end
    else begin
        if(n_state == AXI_AW)   inf.AW_VALID <=  1'b1;
        else                    inf.AW_VALID <=  1'b0;
    end
end
```

【Fig.2】Bug2 & Correct

3. inf.C_r_wb 為 1 時是 read，inf.C_r_wb 為 0 時才是 write (inf.W_DATA wrong)

```
logic[31:0] temp_w;
always_ff @(posedge SystemClock)begin
    if(inf.C_in_valid && !inf.C_r_wb) begin
        temp_w <= inf.C_data_w;
    end
end

asm_data_w_and_w_data: assert property ( @(posedge SystemClock) (inf.W_VALID && inf.W_READY)  |->  (inf.W_DATA == temp_w));
```

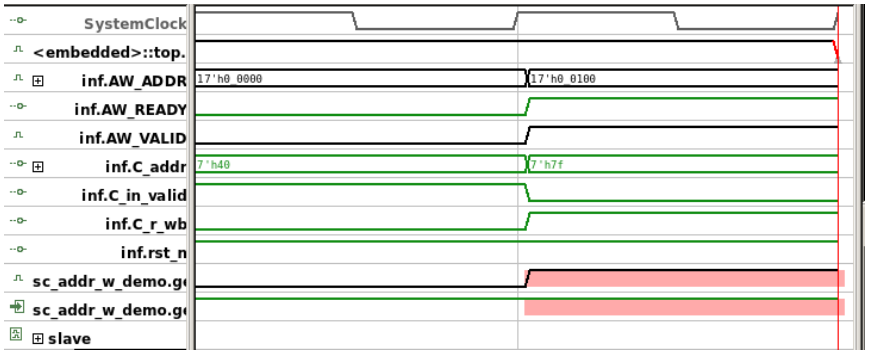

```
//////////////////////////////////////////////////////////////
//                  Before correct
//////////////////////////////////////////////////////////////
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.W_DATA  <= 'b0;
    end
    else begin
        if(inf.C_in_valid && inf.C_r_wb)      inf.W_DATA  <= inf.C_data_w;
        else                                  inf.W_DATA  <= inf.W_DATA  ;
    end
end

//////////////////////////////////////////////////////////////
//                  After correct
//////////////////////////////////////////////////////////////
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.W_DATA  <= 'b0;
    end
    else begin
        if(inf.C_in_valid && !inf.C_r_wb)      inf.W_DATA  <= inf.C_data_w;
        else                                  inf.W_DATA  <= inf.W_DATA  ;
    end
end
```

【Fig.3】Bug3 & Correct & Glue Logic

4. inf.AW_ADDR 應為 8’b1000_000，而不是 8’h1000_0000

```
jasper_scoreboard_3 #(
.CHUNK_WIDTH(32),
.SINGLE_CLOCK(1),
.ORDERING(`JS3_IN_ORDER),
.MAX_PENDING(1)
)sc_addr_w_demo(
 .clk(SystemClock)
 ,.rstN(inf.rst_n)
 ,.incoming_vld(inf.C_in_valid && !inf.C_r_wb)
 ,.incoming_data({1'b1,7'b0,inf.C_addr,2'b0})
 ,.outgoing_vld(inf.AW_VALID && inf.AW_READY)
 ,.outgoing_data(inf.AW_ADDR)
 );
```

【Fig.4】Scoreboard find bug




```
//////////////////////////////////////////////////////////////
//                  Before correct
//////////////////////////////////////////////////////////////
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AW_ADDR <= 'b0;
    end
    else begin
        if(n_state == AXI_AW && c_state != AXI_AW)   inf.AW_ADDR <= {8'h1000_0000, inf.C_addr, 2'b0};
        else                                        inf.AW_ADDR <= inf.AW_ADDR ;
    end
end

//////////////////////////////////////////////////////////////
//                  After correct
//////////////////////////////////////////////////////////////
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AW_ADDR <= 'b0;
    end
    else begin
        if(n_state == AXI_AW && c_state != AXI_AW)   inf.AW_ADDR <= {8'b1000_0000, inf.C_addr, 2'b0};
        else                                        inf.AW_ADDR <= inf.AW_ADDR ;
    end
end
```

【Fig.5】Bug4 & Correct

(g) Feedback

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.

從 Lab07 開始所使用到的 Jasper CDC，一直到 Lab10 所使用到的 IMC Coverage。其實這些 tool 對於我在完成作業的 design 都扮演著很重要的角色。但如果真的要選擇一個對我來說最 effective 的 JasperGold Tools 的話，我會選擇 IMC Coverage。

使用 IMC Coverage 這個 Tool，讓我可以很清楚的知道目前我打到了幾%的 coverage，因為只使用助教所給的 ./02_detail，裡面所顯示的打到的 bin 若不是 100%就不會顯示出來，但有時候可能就只差 1、2 次就可以 100%，就會需要利用 display 的方式去計算到底打到了幾次，但如果使用 IMC Coverage 去查看，整個過程就變得相當快速。

接著講一下所遇到的 Challenge，其實使用這個 tool 沒有遇到像 CDC 那種一直都解不掉的 convergence problem，唯一遇到的狀況就是，IMC Coverage 只能顯示 bins 被打到的次數，並沒辦法細節的顯示出每筆 pattern 所給的 input，以及中間的轉換狀況。因此，我還是會需要利用 dislpay 的功能去進行 debug。但整體來講，這個 tool 真的對於設計 100% coverage pattern 非常好用，所以讓我印象特別深刻。