IC Lab Formal Verification
Lab11 Quick Test
2023 Fall

Name : 林恆霈 **Student ID : 312551078** **Account : iclab131**

(a) What is Formal verification ? What's the difference between **Formal** and **Pattern** based verification ? And list the pros and cons for each.

Formal verification : 為一種用來驗證 design 是否有符合一些特定規範的方法，Formal verification 會使用窮舉法來驗證每一個 state 是否都正確，可靠度較高但是需要更多的運算，也較為費時。而 Pattern verification 則是使用隨機的方式來檢查，執行速度較快，但是有可能會有 corner case 無法測出來的情況，可靠度較低。

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

Glue logic 指在 design 當中為了簡化驗證而加入的 auxiliary logic ，通常與結果輸出無關所以並不會被合成、也不會造成面積上升，但是有了 glue logic 可以讓我們在驗證時有更簡單明瞭的規則，並且可以簡化表達式、提高可讀性等等。

(c) What is the difference between **Functional coverage** and **Code coverage**? What's the meaning of 100% code coverage, could we claim that our assertion is well enough for verification? Why?

Functional coverage 通常用於檢查 design functionality ，且須由設計者自行定義 coverage 的規則，而 Code coverage 通常為 tool 自動產生，可以檢查我們的 RTL code 裡面是否都有成功被執行。 100 % code coverage 的意思為整個 RTL code 都有被成功的執行，達到了完全覆蓋，但是即使有 100 % code coverage 也不代表我們的 assertion 已經完備，因為我們只能確保所有 RTL cod 都有被執行，但是執行結果的正確性還需要進一步的確認。

(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 會找出所有可能跟 assertion 有關的 cover items ，而 proof coverage 則只會找出真正跟 assertion 有關的 cover items ，所以可以得知 proof coverage 會是 COI coverage 的子集合 ， 此外 COI 不需要執行 formal 而 proof 需要 ，所以 proof coverage 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 : Assertion Based Verification Intellectual Properties ，為 checker 組成的 IP，常用於 protocol 的檢查，由於一個 protocol 可能需要大量的 assertion 來檢查，而且相同 protocol 也可以重複使用相同的 assertion，所以就產生了 ABVIP 可以加速設計者驗證自己的電路是否有違反 protocol 。 另外 scoreboard 就像是一個 monitor，而它可以即時檢查輸入、輸出的訊號是否正確，可以加速設計者驗證自己的電路功能是否有誤。

(f) List four **bugs** in Lab Exercise

What is the answer of the Lab Exercise?

```
83  ////////////////////////////////////////////////////////////////////        AR
84
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          /* bug 1
91          if(inf.AR_READY)  inf.AR_VALID <=  1'b1;
92          else              inf.AR_VALID <=  1'b0;
93          */
94          if( n_state == AXI_AR ) inf.AR_VALID <=  1'b1;
95          else                    inf.AR_VALID <=  1'b0;
96      end
97  end
```

```
122 ////////////////////////////////////////////////////////////////////        AW
123 always_ff@(posedge clk or negedge inf.rst_n) begin
124     if(!inf.rst_n)begin
125         inf.AW_VALID <= 'b0;
126     end
127     else begin
128         /* bugs 2
129         if(inf.AW_READY)    inf.AW_VALID <=  1'b1;
130         else                inf.AW_VALID <=  1'b0;
131         */
132         if(n_state == AXI_AW)   inf.AW_VALID <=  1'b1;
133         else                    inf.AW_VALID <=  1'b0;
134     end
135 end
```

Bug 1 為 AR_VALID 的控制訊號，Bug 2 為 AW_VALID 的控制訊號，原本的寫法可能會有無法 handshake 的情況。而因助教已經有宣告 FSM 的 n_state ，可以改成當 n_state == AXI_AR 以及 AXI_AW 時為 1 即可。

```
137 always_ff@(posedge clk or negedge inf.rst_n) begin
138     if(!inf.rst_n)begin
139         inf.AW_ADDR <= 'b0;
140     end
141     else begin                                  //inf.AW_ADDR <= {8'h1000_0000, inf.C_addr, 2'b0}; // bug 3
142         if(n_state == AXI_AW && c_state != AXI_AW)   inf.AW_ADDR <= {8'b1000_0000, inf.C_addr, 2'b0};
143         else                                        inf.AW_ADDR <= inf.AW_ADDR ;
144     end
145 end
146
```

Bug 3 為單純進制符號打錯(8’h10000000 → 8’b10000000)。

```
147  ////////////////////////////////////////////////////////////      W      ////////////
148  always_ff@(posedge clk or negedge inf.rst_n) begin
149      if(!inf.rst_n)begin
150          inf.W_DATA  <= 'b0;
151      end
152      else begin
153          /*  bug 4
154          if(inf.C_in_valid && inf.C_r_wb)      inf.W_DATA  <= inf.C_data_w;
155          else                                  inf.W_DATA  <= inf.W_DATA  ;
156          */
157          if(inf.C_in_valid && !inf.C_r_wb )
158              inf.W_DATA  <= inf.C_data_w;
159      end
160  end
```

Bug 4 為 C_r_wb 判斷式寫錯，W_DATA 應該為寫入(C_r_wb == 0)時才更新。

(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.

在這 4 個主題當中最令我印象最深刻的就是 Lab07 的 Jasper CDC ，而這個 tool 適用於當 design 需要跨越多個時域時，可以利用窮舉法檢查來檢查跨越時域的訊號有沒有正常運作，以及檢查有沒有符合 setup hold time 等規則。當初看到題目時以為只是簡單的 CDC，只要正常使用 NDFF 就可以避免掉 convergence 的問題發生，不過在寫完之後的第一次模擬當中，發現還是發生了 convergence 的問題，並同時發生了 POP_ON_EMTY 、 PSH_ON_FULL 、RPT_NO_GRAY、 WPT_NO_GRAY 等 functional 的問題，為了找出是哪一個訊號發生了 convergence，我先分別註解 clk1 , clk2 ,clk3 這三個 module 來觀察，最後發現是 clk2 – FIFO 之間的一個 flag 所造成的，而 flag 主要是從 FIFO 裡用組合電路直接連接到 clk2 裡，最後參考了同學的建議在 flag 前面多檔一層 reg 並且要使用與 clk2 相同的時脈控制 reg 之後就成功解決掉了 convergence，但是也因為組合電路跟循序電路會差一個 cycle，所以我也對整個 design 重新進行設計。 而剩下的 funtional 部分，一開始在看完助教給的 JG_Guide 之後成功的讓 functional 的紅色叉叉消失，不過我同時也發現我的 JG 右下角會一直在旋轉，也就是代表 JG 一直無法判斷我的 RPT_NO_GRAY、 WPT_NO_GRAY 是否正確，而詢問助教之後助教告知我使用更簡單的邏輯判斷來寫也許會有所幫助，所以我就把 FSM 裡的兩個訊號(rptr wptr)拉出來單獨做判斷，並盡量簡化判斷式，最後終於成功地縮短了 JG 檢查的時間，也通過了所有 functional 以及 convergence 的檢查。