IC Lab Formal Verification Lab10 Quick Test 2024 Spring

Name: 郭銘宸 **Student ID:** 311511082 **Account:** iclab048

(a) What is Formal verification?

Formal verification tests each possible stimulus, one cycle at a time. This means testing all combinations of values for inputs and undriven wires at every cycle. And it tests all combinations of values for uninitialized registers at first cycle.

Formal verification checks properties when it is walking through stimuli. It will report when assert properties are violated or cover properties are hit.

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

And list the pros and cons for each.

Formal verification tests all possible value combinations of input, wire, etc. It is easy to consider the corner case. But the design is too big, it will spend more simulation time. Pattern verification is a verification method designed based on the DUT. It only tests one input combination at a time to verify the correctness of the design circuit. It need human to design corner case to ensure coverage. So it may be have human error.

As mentioned above, the main difference between Formal verification and Pattern verification is in their verification methods.

Formal verification:

- Pros:
  1. Systematic method: None or very little randomization, so it is more deterministic.
  2. Less testbench effort required: Formal testbench is much simpler than sim testbench.
  3. Leads to higher quality: Often reveals bugs that simulation would never catch.
  4. Improves productivity: it can replace some block-level testbench, so it can begin prior to testbench creation and simulation.
- Cons:
  1. If the design is too complex, it will lead to an exponential increase in the number of verification required, resulting in the consumption of significant resources and time.

Pattern verification:

- Pros:

  (1) Because it tests behavior only under specific patterns, it consumes fewer resources and requires less time for verification of large circuits.

- Cons:

  (1) If the designer fails to consider all possible scenarios (aka. Corner case). It can leads to incomplete pattern coverage, reducing the accuracy of verification. Additionally, it increased time spent for designing simulation pattern because it is lower automation levels.

(b) What is glue logic?

Glue logic is additional logic used to reduce the complexity of SVA (aka. System Verilog Assertion) when modeling complex behaviors. In other Words, we can use glue logic to simplify the representation of SVA.

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

Using glue logic can simplify the way SVA is written, increasing its readability, and making maintenance easier, all without affecting functionality of the design circuit. Because Design Compiler ignores glue logic, so it won't impact design area, timing, etc.

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

Functional coverage: it is possible to represent all meaningful design functionality, implements the verification plan, and noise-free (it means that nothing is don't-care). However, it is designed by human, it requires planning, coding, and debugging, which can be consuming more time. Additionally, there is a possibility of incomplete verification due to human error.

Code coverage: it is easy to generate automatically by EDA tool, directly checks if each line of code in the design has been executed correctly. It can guarantees to be structurally complete, it mean no human error. But it may have noisy, like don't-care statement. So we need to manual waive these situations.

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

NO!!! Code verification simply verifies whether each line of code in the design has been executed, but it does not ensure the functional correctness of design. For example, if the design needs certain computations module that are not present in your RTL code, achieving 100% code coverage but does not guarantee the correctness of the functionality.

(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.

- Meaning

COI coverage: Each assertion is affected by some cover items, forming region that will be called COI(Cone-Of-Influence). It finds the union of the all assertion COIs.

Proof coverage: It will find the region cannot truly influence assertion status. However, proof coverage uses formal engines to execute formal proof. In the other words, it represents the portion of the design verified by formal engines. Additionally, it is a subset of the COI coverage.

- Relationship

  Proof coverage is a subset of COI coverage. COI is the maximum potential of proof coverage.

- Tool effort

  Proof coverage identify more unchecked code than COI coverage. So proof coverage verify design by formal engines, and COI coverage is fast measurement that is no formal engines are run. In this reason, COI coverage requires more running time and resources.

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

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

- Definition

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

  (2) Scoreboard: Scoreboard behaves like a monitor that use to o observe input data and output data of DUV.

- Objective

  (1) ABVIP: Verify a protocol and analyzing its completeness is a key challenge for engineers these days. So ABVIP provides a comprehensive and pre-built IP for engineers to verify the existing protocol. It can save designing time and ensure the correctness.

  (2) Scoreboard: It is used to check whether the input and output match the expected value, serving as high-level monitor.

- Benefit

  (1) ABVIP: Using ABVIP provided by the vendor can ensure the correctness and save time in designing assertion.

  (2) Scoreboard: Formal optimized to reduce state-space complexity. It can reduce barrier for adoption.

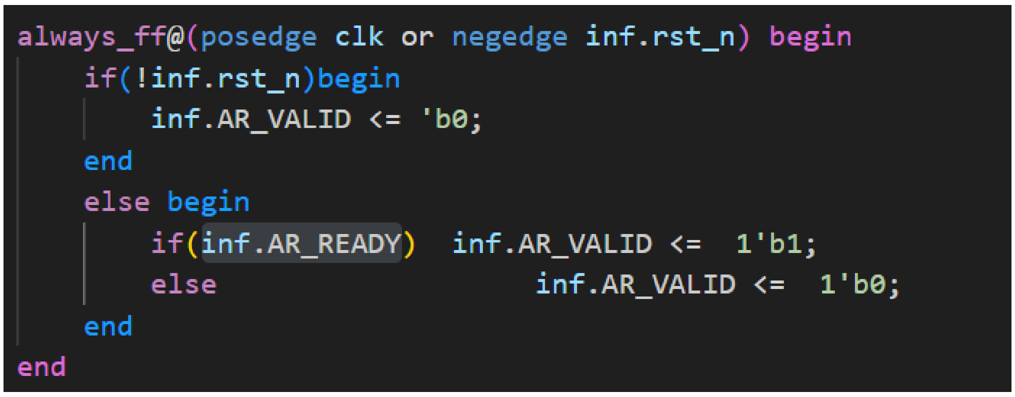
(f) List four bugs in Lab Exercise

What is the answer of the Lab Exercise?

(3) Bug1:

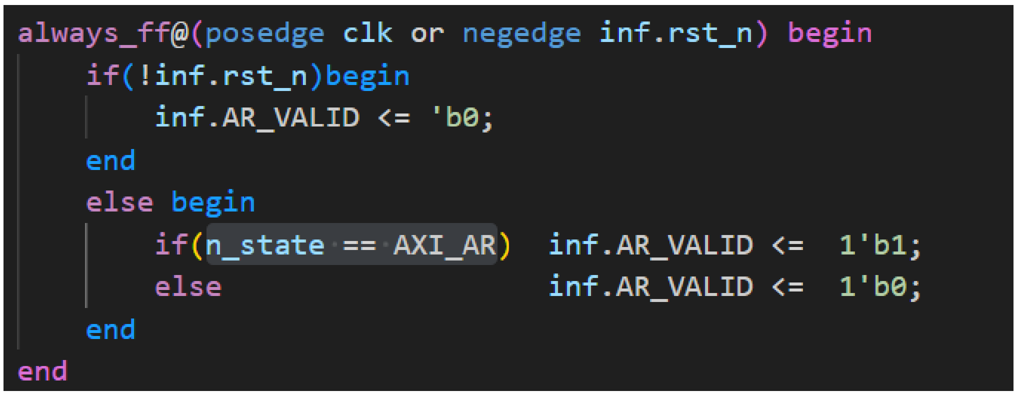
In AXI protocol, the valid signal of one AXI component must be independent on the ready signal of the other component in the transaction. (prevent deadlock)

- Wrong Code



```
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
```

- Correct Code



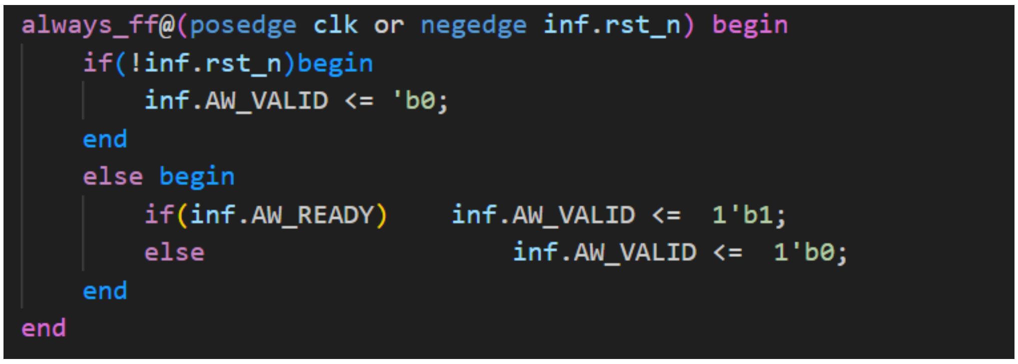
```
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AR_VALID <= 'b0;
    end
    else begin
        if(n_state == AXI_AR)  inf.AR_VALID <=  1'b1;
        else                   inf.AR_VALID <=  1'b0;
    end
end
```

(4) Bug2:

Same reason with Bug1.

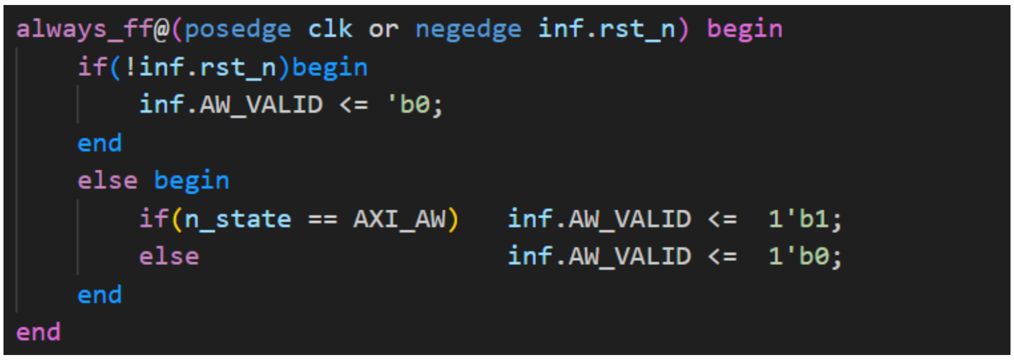
In AXI protocol, the valid signal of one AXI component must be independent on the ready signal of the other component in the transaction. (prevent deadlock)

- Wrong Code



```
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
```

- Correct Code



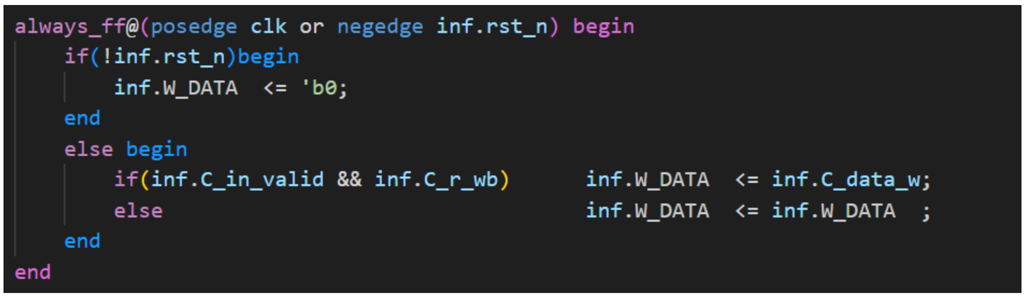
```
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AW_VALID <= 'b0;
    end
    else begin
        if(n_state == AXI_AW)   inf.AW_VALID <=  1'b1;
        else                    inf.AW_VALID <=  1'b0;
    end
end
```

(5) Bug3:

Read: inf.C_r_wb = 1;

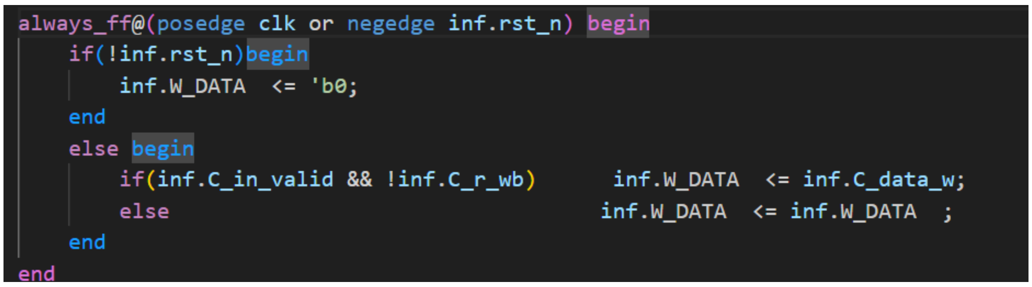
Write: inf.C_r_wb = 0;

- Wrong Code



```
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
```

- Correct Code

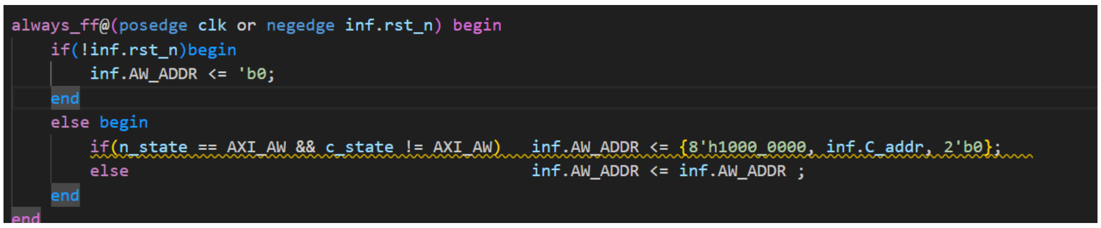


```
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.W_DATA  <= 'b0;
    end
    else begin
        if(inf.C_in_valid && !inf.C_r_wb)      inf.W_DATA  <= inf.C_data_w;
        else                                  inf.W_DATA  <= inf.W_DATA  ;
    end
end
```

(6) Bug4:

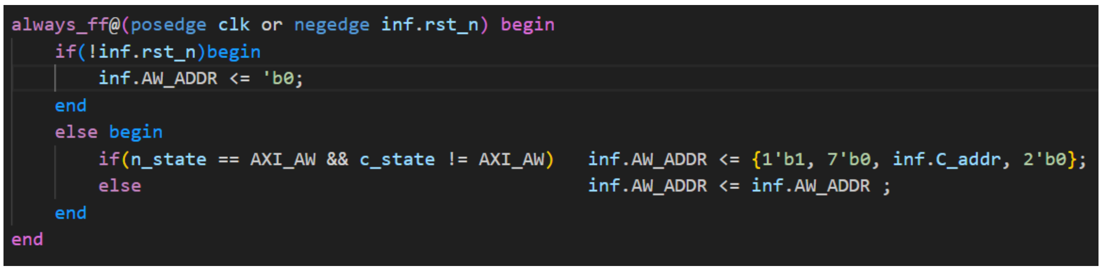
inf.AW_ADD should be {1'b1, 7'b0, inf.C_addr, 2'b0}. 8'h1000_0000 is out of bit range(8 bits).

- Wrong Code



```
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
```

- Correct Code



```
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 <= {1'b1, 7'b0, inf.C_addr, 2'b0};
        else                                          inf.AW_ADDR <= inf.AW_ADDR ;
    end
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.**

For me, I think Jasper CDC is the most helpful tool in the verification process. Although I haven't encountered CDC much in school, but I know it's a common issue in the company to deal with cross clock domain problems. Also, I think CDC is quite complex because it needs to ensure data can be safety transferred to another clock domain and it requires special attention in design.

In Lab7, initially I used patterns to verify the functionality of the design. Both my patterns and waveforms showed everything correctly. However, when verifying with Jasper CDC, error occurred. Jasper CDC identified test data that was not covered by the patterns. And Jasper CDC indicated where the issues were. This left a deep impression on me because using patterns to check design correctness is a commonly used method, but pattern also have some loopholes. If we can use software like Jasper CDC to assist in verification, especially for more complex designs like CDC, it would be extremely helpful.