IC Lab Formal Verification
Lab11 Quick Test
2024 Spring

Name: 鄒振邦 **Student ID:** 312510120 **Account:** iclab109

(a) What is Formal verification?

Formal verification is a method of validation that uses mathematical techniques to ensure the correctness of a design. During formal analysis, the tool generates all possible combinations of inputs and connections, and checks the resulting outputs to validate the correctness of the system.

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

The main difference between formal verification and pattern based verification is their verification methods. Formal verification uses mathematical and logical techniques to prove the correctness of a system, while pattern based verification generates test patterns and feeds them as inputs to the DUV to detect issues or errors.

And list the pros and cons for each.

Formal Verification

Pros:

(1) High accuracy:

Testing all possible inputs using mathematical methods, none or very little randomization, more deterministic.

(2) High quality:

Find bugs from a different angle : breadth-first search (formal) vs. depth-first search (sim), often reveals bugs that simulation would never catch.

(3) Less testbench effort required:

Formal testbench tends to be much simpler than sim testbench.

(4) Improves productivity and schedule:

Can replace some block-level testbenches, verification can begin prior to testbench creation and simulation.

Cons:

(1) Consume a large amount of computational resources:

As the circuit scale increases, the complexity of verification rises exponentially. Covering all input combinations requires substantial computational resources and time.

Pattern Based Verification

Pros:

(1) Less time consumption:

Because verification is conducted by setting patterns corresponding to common issues and errors, the designer can control the number of patterns and reduce verification time.

Cons:

(1) Low coverage:

If the designer doesn't consider all input combinations, it can easily result in a low coverage rate.

(b) What is glue logic?

Using additional computational logic to replace commonly occurring behavioral signals, thus simplifying the logic.

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

When encountering the need for complex assertions to describe logic, we need to write lengthy and intricate logical expressions. In such cases, we can utilize glue logic to simplify the complex signal relationships into new logic, facilitating the composition of SVA. This approach enhances readability and aids in inter-module verification.

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

Functional coverage

Functional coverage primarily checks whether the functionalities in the design meet expectations, ensuring that our design operates correctly under various conditions. Because it's manually designed, there's a significant risk of human error leading to incomplete verification, and the design process could be time-consuming.

Code coverage

Code coverage primarily checks whether branches, statements, and expressions in the code have been executed. Although code coverage is generated by EDA tools, sometimes it detects cases that will never occur, requiring manual waiver.

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

A code coverage of 100% indicates that every line of code has been executed. However, code coverage does not perform functional validation and testing, so the functionality of our design may still be incorrect. Therefore, achieving 100% code coverage alone is not sufficient 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.

Meaning:

COI coverage verifies all cover items that affect the logic checked by assertions by backtracking all cover items. Therefore, if a cover item does not appear in COI coverage, it means that this cover item will not be tested. Proof coverage, on the other hand, only verifies cover items that directly impact the assertion.

Relationship:

Proof coverage is the subset of COI coverage.

Tool effort:

COI coverage only requires backtracking all cover items that influence assertions, hence, it doesn't necessitate a formal engine, resulting in lower tool effort. Proof coverage, on the other hand, requires identifying the actual paths that affect assertions, necessitating a formal engine and thus higher tool effort.

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

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

Definition:

ABVIP are a comprehensive set of checkers and RTL that check for protocol compliance.

Scoreboard behaves like a monitor which observe input data and output data of DUV and it can verify the result of our design.

Objective:

ABVIP provides pre-designed assertion IP to facilitate designers in checking whether the protocol conforms to the expected functionality and behavior.

Scoreboard is to verify whether the actual output matches the expected output.

Benefit:

When dealing with complex protocols, ABVIP avoids the need for manually writing checkers by directly utilizing pre-designed ABVIPs, thereby expediting the verification process. Since ABVIPs have undergone verification, the verification results are more rigorous and reliable.

In addition to comparing signal correctness, a scoreboard can also check for Data packet dropped, Duplicated Data Packets, Order of Data Packets, and Corrupted Data Packets.

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

What is the answer of the Lab Exercise?

1. AR_VALID and AR_READY should be low simultaneously


```
90          if(inf.AR_READY)  inf.AR_VALID <=  1'b1;
```

Answer

```
90          if(n_state == AXI_AR)  inf.AR_VALID <=  1'b1;
```

2. AW_READY and AW_VALID should be low simultaneously


```
124         if(inf.AW_READY)    inf.AW_VALID <=  1'b1;
```

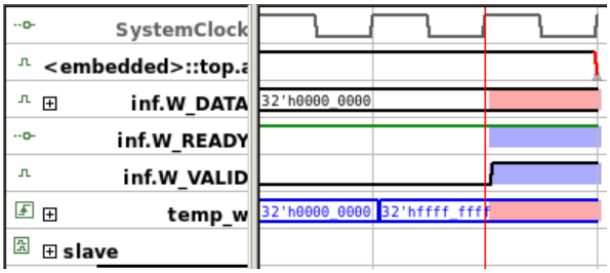
Answer

```
124         if(n_state == AXI_AW)   inf.AW_VALID <=  1'b1;
```

3. C_r_wb is read when it is high. C_r_wb is write when it is low.

Add glue logic

```
157  logic[31:0] temp_w;
158  always@(posedge SystemClock)begin
159      if(inf.C_in_valid && !inf.C_r_wb) begin
160          temp_w <= inf.C_data_w;
161      end
162
163      else begin
164          temp_w <= temp_w;
165      end
166  end
167
168  asm_data_w_and_w_data: assert property ( @(posedge SystemClock) (inf.W_VALID && inf.W_READY)  |->  (inf.W_DATA == temp_w));
```




```
145          if(inf.C_in_valid && inf.C_r_wb)       inf.W_DATA  <= inf.C_data_w;
```

Answer

```
145          if(inf.C_in_valid && !inf.C_r_wb)       inf.W_DATA  <= inf.C_data_w;
```

4. Wrong AW_ADDR

Add Scoreboard IP

```
141   jasper_scoreboard_3 #(
142   .CHUNK_WIDTH(32),
143   .SINGLE_CLOCK(1),
144   .ORDERING(`JS3_IN_ORDER),
145   .MAX_PENDING(1)
146   )sc_addr_w_demo(
147    .clk(SystemClock)
148    ,.rstN(inf.rst_n)
149    ,.incoming_vld(inf.C_in_valid && !inf.C_r_wb)
150    ,.incoming_data({1'b1,7'b0,inf.C_addr,2'b0})
151    ,.outgoing_vld(inf.AW_VALID && inf.AW_READY)
152    ,.outgoing_data(inf.AW_ADDR)
153    );
```


```
134          if(n_state == AXI_AW && c_state != AXI_AW)   inf.AW_ADDR <= {8'h1000_0000, inf.C_addr, 2'b0};
```

Answer

```
134          if(n_state == AXI_AW && c_state != AXI_AW)   inf.AW_ADDR <= {1'b1, 7'b0, inf.C_addr, 2'b0};
```

(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.

I find formal verification to be the most convenient and practical in this Lab. We don't need to spend time writing checkers for verification; we just need to master the tool's operation to complete the verification of AXI4 functionality. This approach not only saves a considerable amount of time but also results in more rigorous and reliable verification outcomes.