IC Lab Formal Verification
Bonus Quick Test
2023 Spring

Name: 欉家新 **Student ID:** 0810923 **Account:** iclab025

1. Bonus:

(a) What is Formal verification?

Formal verification is a process to prove the correctness of the design. It usually involves the mathematical modeling of the design and uses formal language, formal methods such as automated tools to verify whether the design meets specifications.

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

Formal verification confirms the correctness of a design through the assertion of properties and considering all possible stimuli combinations one cycle at a time. While pattern-based verification relies on using hardware design language to generate pattern or stimuli to test if the design gives correct outputs while giving certain inputs. Both of their goal is to find bugs, while formal verification use breadth-first search, pattern based verification use depth-first search.

And list the pros and cons for each.

Formal verification:

Pros:

1. Systematic method
2. Little randomization, more deterministic
3. Less testbench effort.

Cons:

1. Time-consuming (especially for large designs)
2. Less flexible for directed tests

Pattern-based verification:

Pros:

1. Easier to understand and apply
2. More readable

3. Quick verification of designs at early stage.

Cons:

1. May miss some cases and states, cannot cover all possible errors

(b) What is glue logic?

When modeling complex behaviors, SVA expressions can become overly complex. In such cases, some auxiliary logic can be used to observe and track events, helping us simplify the coding. This auxiliary logic is commonly referred to as "glue logic."

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

Because SVA expressions can be complex, involving multiple conditions and logical operations, using glue logic can help transform these complex SVA expressions into clearer and more identifiable forms. For example, instead of using complex SVA expressions to check assertions, we can use many simple glue logics to combine and create an assertion with better readability.

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

Functional coverage is about checking specific states, conditions or sequences to be verified. And it focuses on verifying the completeness of a design's functionality. It is possible to represent all meaningful design functionality, and implements the verification plan, that is, what needs to be verified. It is also noise-free, so nothing is don't care. However, it requires planning, coding, debug and is susceptible to human error. While code coverage will check all the possible cases written in the code, for example, branches, statements and expressions. Unlike functional coverage, code coverage focuses on the extent of code coverage and structural integrity during testing. Therefore, code coverage may not capture all meaningful design functionality, and can be noisy.

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

100% code coverage means that every line of codes has been executed at least once, that is, no dead code exists. However, it doesn't guarantee that the assertion is well enough for verification. Because code coverage only measures the extent to which the code has been tested, but it doesn't ensure that all possible scenarios and corner cases have been adequately addressed. It means that even with 100% code coverage, the functionality coverage may not reach 100% because of lack of some corner cases.

(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 and proof coverage differ in terms of achieving completeness for checkers.

COI is the abbreviation of Cone-Of-Influence. COI coverage will find the cover items affecting assertion. After finding the union of all assertion COIs, the remaining cover items are holes in the assertion set that these codes cannot be checked by any asserts. Because it doesn't need formal engines to run, it is a rather fast measurement. On the other hand, proof coverage identifies the design portions that can influence the outcome of assertions, which is verified by formal engines.

The relationship between the two lies in proof coverage being a subset of COI coverage. In terms of tool effort, proof coverage requires verification using formal engines, thus potentially requiring more resources and time to accomplish.

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

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

ABVIP:

Definition: ABVIP is the abbreviation of “Assertion Based Verification Intellectual Properties”. ABVIP is a comprehensive set of checkers and RTL that check for protocol compliance, for example, ARM, AMBA, AXI and so on.

Objective: The goal of ABVIP is to enhance the accuracy and efficiency of the verification process by providing comprehensive set of assertion based on protocols.

Benefit: It can accelerate verification, enhance debugging and save time for designer.

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: The goal of a scoreboard is to track and observe the data of the design being tested and to confirm its correct operation. It can also reduce the state-space complexity and the barrier for adoption.

Benefit: It helps designer to do error detection and to debug more easily.

(f) List four bugs in Lab Exercise

What is the answer of the Lab Exercise?

Bug 1: The assertion of “AR_VALID must be stable when AR_READY is low” fails.

Answer 1: Correct「if(inf.AR_READY)」 at line 90 to 「if(n_state == AXI_AR)」

Bug 2: The assertion of “AW_VALID must be stable when AW_READY is low” fails

Answer 2: Correct「if(inf.AW_READY)」at line 124 to 「if(n_state == AXI_AW)」

Bug 3: The assertion of “AW_VALID must be stable when AW_READY is low” fails.

Answer 3: Correct「{8’h1000_0000, inf.C_addr, 2’b0}」at line 134 to 「{1’b1, 7’b0, inf.C_addr, 2’b0}」

Bug 4: W_DATA gives wrong value while writing.

Answer 4: Corrrect「if(inf.C_in_valid && inf.C_r_wb)」at line 145 to 「if(inf.C_in_valid && !inf.C_r_wb)」