IC Lab Formal Verification Bonus Quick Test
2022 Spring

Name: 呂承哲 **Student ID:** 310551145 **Account:** iclab039

(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 refers to the process of establishing functional equivalence of designs generally represented as HDL models without running simulations.

The advantage of formal verification is as followed:

1. Formal verification is more deterministic which requires none or very little randomization.

2. Formal verification requires less testbench effort compared to simulation-based verification.

3. Formal verification is complementary with simulation-based verification. Checking DUTs using two methods could lead to higher quality.

4. Just as 2. mentioned, formal verification can begin prior to testbench creation and simulation. It can replace some block-level testbenches and improve productivity and schedule.

Formal verification, however, has its disadvantage too.

The module verified by formal verification should not be too large in terms of the number of states, otherwise the runtime may be unacceptable.

(b) What is glue logic?

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

Glue logic refers to auxiliary logic used in SVA which could help observe and track events.

The main reason glue logic is used to simplify SVA expression is because it requires almost no extra price. Formal verification tools (e.g. JasperGold) do not care whether the property is composed of SVA only or SVA + glue logic. Also, glue logic could be used to express the property more directly, which makes the expression easier to understand for both formal verification tools and verification engineers.

(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 covers the property-related bins. It describes specific states, conditions, or sequences which should be verified. Normally it is determined by user.

Code coverage covers the code-related bins. Those bins are based on the branch and statement in HDL code. Normally it is determined by tool itself.

We could not ensure the assertion has no problem even though the code coverage is 100%. If the functionality of design is implemented wrong, the code coverage is meaningless. Both functional coverage and code coverage should be considered when verifying designs.

(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.

Cone-of-Influence (COI) coverage refers to the maximum potential cover items that will be affected by an assertion. It identifies holes in the set of assertions. COI coverage analysis is done before proof coverage analysis and is the superset of it. It does not require formal engine to calculate the scope of it. Instead, it could be calculated just by tracing the relation between assertions and cover items.

Proof coverage refers to the scope of cover items that are truly affected by an assertion. It identifies areas of the design which is not verified by full or bounded proofs. Proof coverage analysis is done after COI coverage analysis and is the subset of it. It requires formal engine to iteratively identify the scope of it, which takes greater tool effort.

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

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

ABVIP is a comprehensive set of checkers and RTL that check for protocol compliance. It could be used to check the functionality of DUTs, by instantiating a master (slave) instance to interact with slave (master) DUTs, or a monitor instance to check both. The benefit of ABVIP is that it is built upon assertions and high-level logic, which could be used to check DUTs repeatedly.

Scoreboard, on the other hand, behaves like a monitor that observe the input and output data of DUTs. The objective and benefit of scoreboard is similar with ABVIP. Scoreboard verifies the functionality of DUTs, and it is separated from DUTs so that it could be repeated used across designs.

(f) List four bugs in Lab Exercise

What is the answer of the Lab Exercise?

1. Violating master_ar_arvalid_stable. The answer should be:

```
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 <= 0;
                else if (c_state == AXI_AR)     inf.AR_VALID <= 1;
                else                            inf.AR_VALID <= inf.AR_VALID;
        end
end
```

2. Violating master_aw_awvalid_stable. The answer should be:

```
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 <= 0;
                else if (c_state == AXI_AW)     inf.AW_VALID <= 1;
                else                            inf.AW_VALID <= inf.AW_VALID;
        end
end
```

3. Violating AW_ADDR data_integrity. The answer should be:

```
always_ff@(posedge clk or negedge inf.rst_n) begin
	if(!inf.rst_n)begin
		inf.AW_ADDR <= 'b0;
	end
	else begin
		if (inf.C_in_valid && !inf.C_r_wb)	inf.AW_ADDR <= {1'b1, 7'b0, inf.C_addr, 2'b0};
		else					inf.AW_ADDR <= inf.AW_ADDR;
	end
end
```

4. Violating assert_w_data (customized assertion in top.sv). The answer should be:

```
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
```