IC Lab Formal Verification Bonus Quick Test
2022 Spring

Name: 李裕祥 **Student ID:** 0810727 **Account:** iclab126

(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 is a method to check whether the design trying to test satisfy some property that specified by user, the property is specified by “assume” , ”assert”, “covergroup” etc, these verifications don’t need to use test case, they just use mathematical methods to prove the correctness of the design.
The different between formal verification and pattern based verification is that pattern based verification need a lot of test case to test, it lead to more time to check the functionality of the design, it just use random test case as input of the design, and check whether the output of the design as expected.

	formal verification	pattern based
props	1.Less time to verify 2.higher quality due to systematic method	1.easier to implement. 2.can test some corner case by just adding some test case.
cons	1.need some background knowledge. 2.cost is higher than pattern based.	1.may omit some corner case. 2.need more time to test the design.

(b) What is glue logic?

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

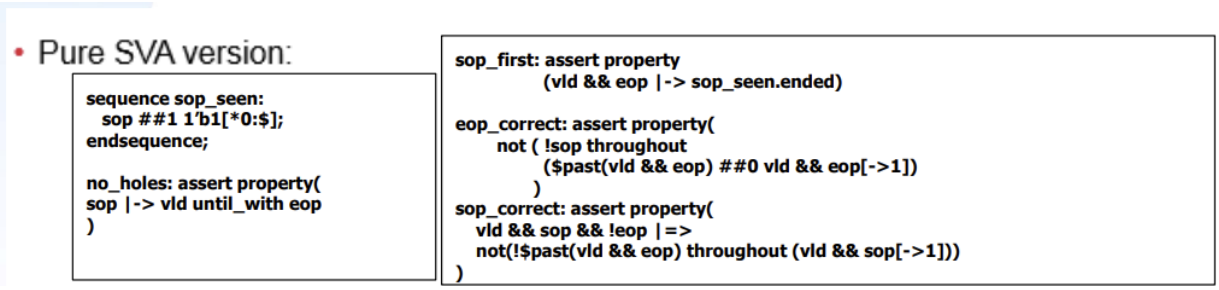
Glue logic is a circuit that used to combine different circuit, it uses auxiliary logic to track the behavior of two logic, so it can easily track the event, and debugging.

When using glue logic, the logic can be simplify, for example:

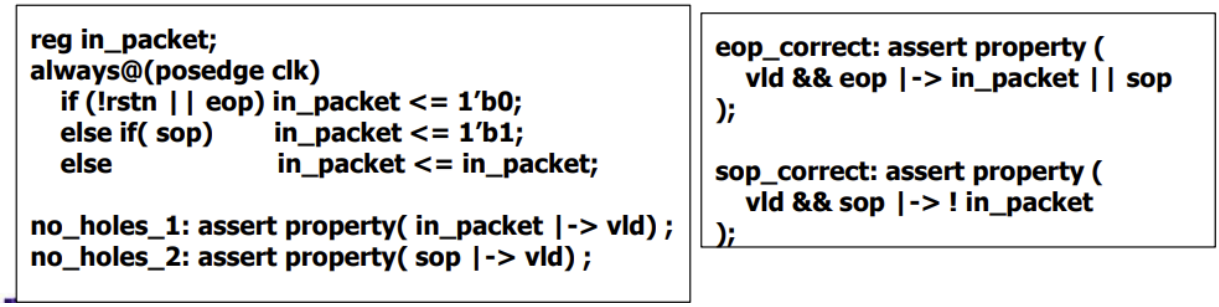
To meet the following spec:

- No overlapping packets (SOP-EOP always in pairs)
- Single-cycle packets allowed (SOP and EOP at the same cycle)
- Continuous packet transfer (no vld holes between SOP-EOP)

Without glue:



With glue:



It's easy to find that with glue logic, the circuit is more simple.

(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 is defined by a collection of cover items, cover groups and crosses.it test all required functionality required by the specifications, in other words, it test all function by user's point of view, if all functionality has been tested, it means functional coverage metric is 100%.

Code coverage is a method to measure how many lines of code has been executed by the test pattern, if each line of code has been executed while testing, it means the code coverage metric is 100%, but even though the code coverage is 100%, it still can't cover each transition of state, so it may has some corner case that can't reach by the test.

(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.

The definition of COI coverage is each assertion affected by some cover items, which is maximum potential of proof coverage, and it declared in part of checker coverage.
And proof coverage is a subset of COI, which require a proof to take place, so it requires more time to complete with formal engine, in other words, it requires more tool effort.

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

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

Because more and more company decide to use protocol to transmit data, so verify the completeness gradually became a challenge, that is the reason why more and more company use assertion to check the designed IP, and that is called ABVIP (Assertion Based Verification Intellectual Properties).

With ABVIP, we can find bugs early, because it doesn't require any test stimulus, and it is possible to check all type of access policy.

The benefit of ABVIP is it enable the automated and plug-and-play capabilities, it is easier to use with context -sensitive IP, its performance also higher for both Incisive formal and simulation engines.

(f) List four bugs in Lab Exercise

What is the answer of the Lab Exercise?

1.

Wrong:

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

Correct:

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

2.

Wrong:

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

Correct:

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

3.

Wrong:

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

Correct:

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

4.

Wrong:

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

Correct:

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