IC Lab Formal Verification
Lab11 Quick Test
2023 Fall

Name: 林煜睿 **Student ID:** 312510224 **Account:** iclab031

(a) What is Formal verification?

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

And list the pros and cons for each.

Ans:

Formal verification will completely test our RTL code, it will test all possible stimulus per cycle to cover all scenario. Therefore, Formal verification can find some tiny error that Pattern based verification hard to find out.

Formal verification:

- Pros:

  1. Accelerate IC design flow, verification can begin prior to testbench creation and simulation.

  2. Less testbench effort required.

  3. Leads to higher quality, often reveals bugs that simulation would never catch.

  4. More deterministic, none or very little randomization.

- Cons:

  1. Time-consuming (especially for large designs)

  2. Less flexible for directed tests

Pattern Based Verification:

- Pros:

  1. Quick verification of designs at early stage.

  2. Easier to understand and apply, also more readable.

- Cons:

  1. May miss some cases and states, hard to cover all potential errors.

(b) What is glue logic?

Ans:

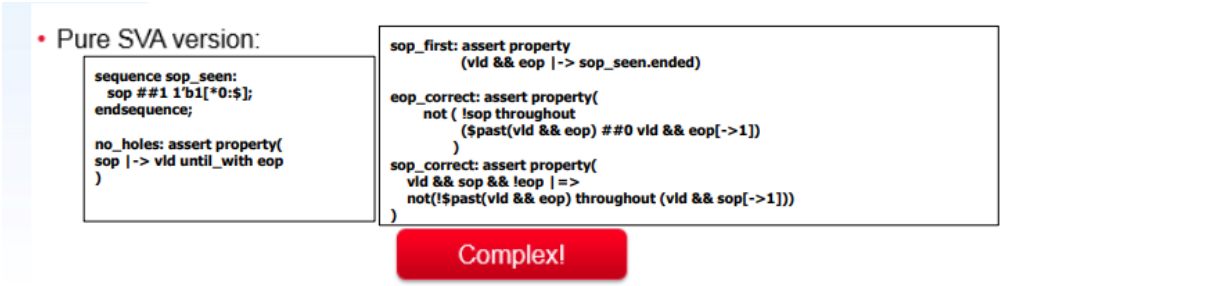
When modeling complex behaviors, SVA expressions usually become quite complex. In such cases, glue logic can be used to observe and track events, helping us simplify coding.

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

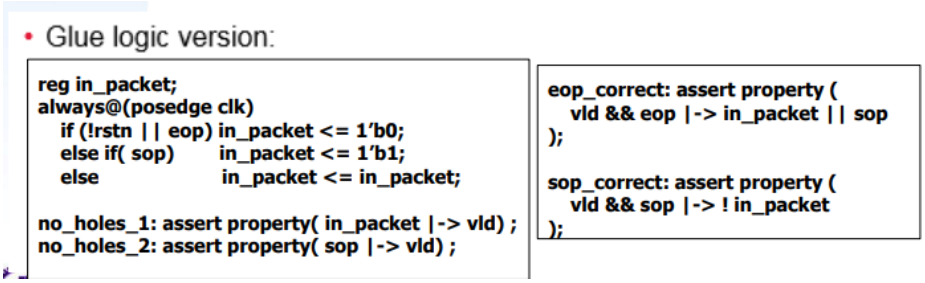
Ans:

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, we can see that the assertion is very complicate without glue logic.



But if we use glue logic to create additional signal in figure below. Then the whole SVA will be simpler than the one without glue logic.



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

Ans:

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. Also, functional coverage is noise-free.

Code coverage measures how much of the "design Code" is exercised. This includes the execution of design blocks, Number of Lines, Conditions, FSM, Toggle and Path. It 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?

Ans:

100% code coverage means our test case covers every line of RTL code (all RTL code are being executed at least once).

Assertion is related to pattern input and design output, if our patterns are not considered enough, then the assertion will not be violated. Therefore, we can't claim that our assertion is well enough 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.

COI coverage:

Each assertion is affected by some cover items , and this region will be called Cone Of Influence (COI). COI coverage finds the union of the all assertion COIs.

The tool analyzes this checker and determines all the inputs, outputs and internal variables of the DUT that influence this checker. The result of this analysis is called the COI. Once the COI is determined, the tool can now determine the full state space of possible values the logic within the cone can take. It then parallelly explores every path in the state space and checks if the assertion can be proven wrong.

Proof Coverage:

Proof coverage is a subset of the COI. It's the region that can't truly influence assertion status. COI is the maximum potential of proof coverage. COI doesn't require a proof to take place, while proof coverage does. Identify more unchecked code than COI measurement, it needs greater tool effort. It's slower measurement than COI.

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

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

Ans:

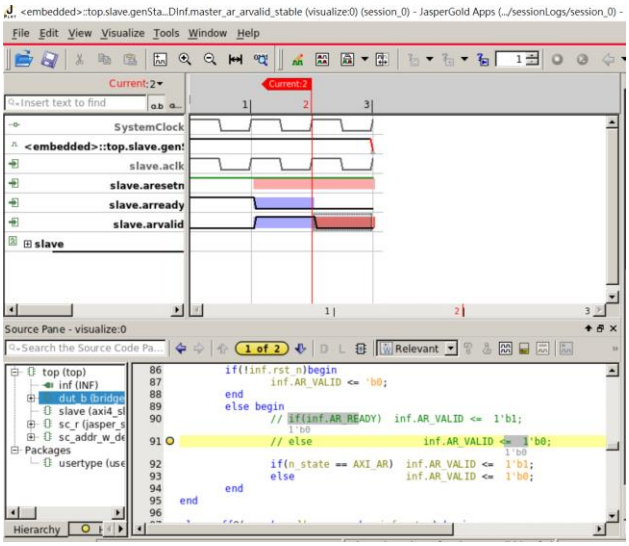
ABVIP are comprehensive set of checkers and RTL that check for protocol compliance. ABVIP would exhaustively verify the compliance of a design under test (DUT) to a given protocol. The benefit of ABVIP is that ABVIP help to categorize the design intent and excepted operations so that can find critical bugs earlier and shorten the overall verification schedule.

Scoreboard checks output from the design with expected behavior, it behaves like a monitor, observing input data and output data of DUV. The Scoreboard can have a reference model that reflects the expected behavior of the DUT. Therefore, if DUT has a functional problem, output from the DUT will not match the output from our reference model. The benefit of Scoreboard is that formal optimized to reduce state-space complexity, which reduces the barrier for adoption.

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

What is the answer of the Lab Exercise?

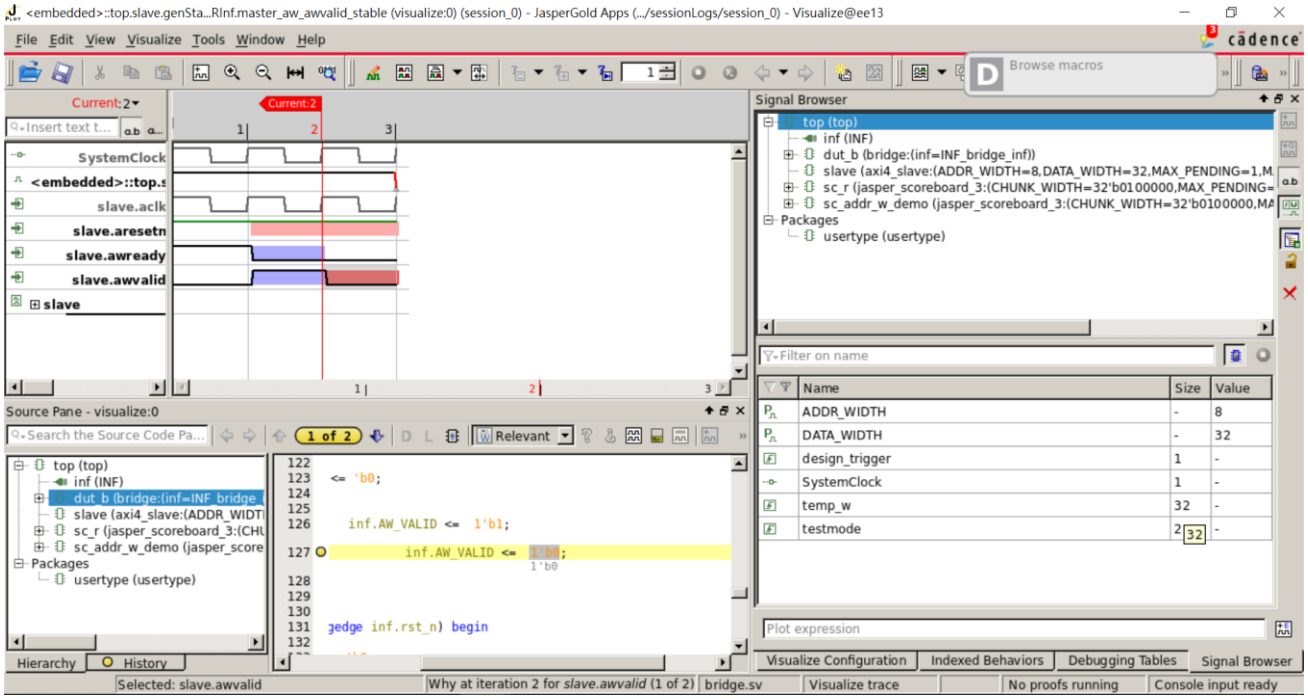
1st bug: arvalid should remain stable if (arvalid and !arready).



Answer:

```
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;
    if(n_state == AXI_AR)  inf.AR_VALID <=  1'b1;
    else                   inf.AR_VALID <=  1'b0;
  end
end
```

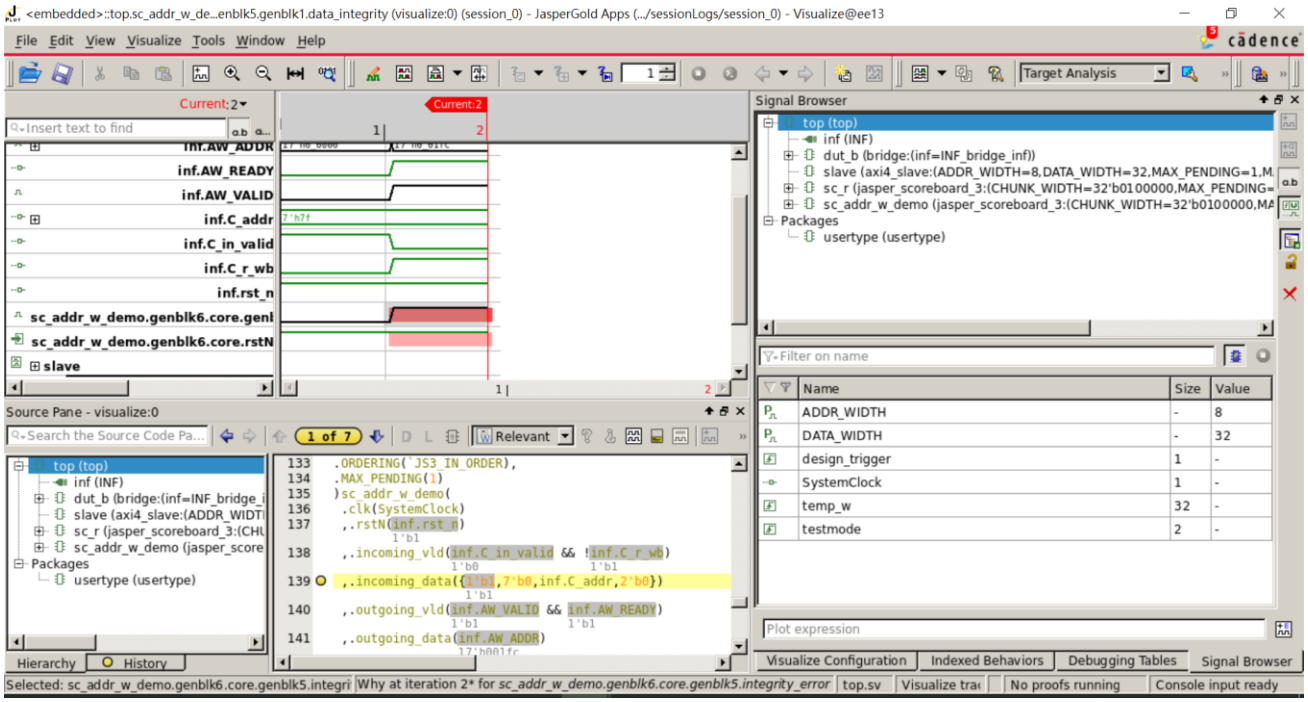
2nd bug: awvalid should remain stable if awvalid and !awready



Answer:

```
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;
      if(n_state==AXI_AW) inf.AW_VALID <=  1'b1;
      else                inf.AW_VALID <=  1'b0;
  end
end
```

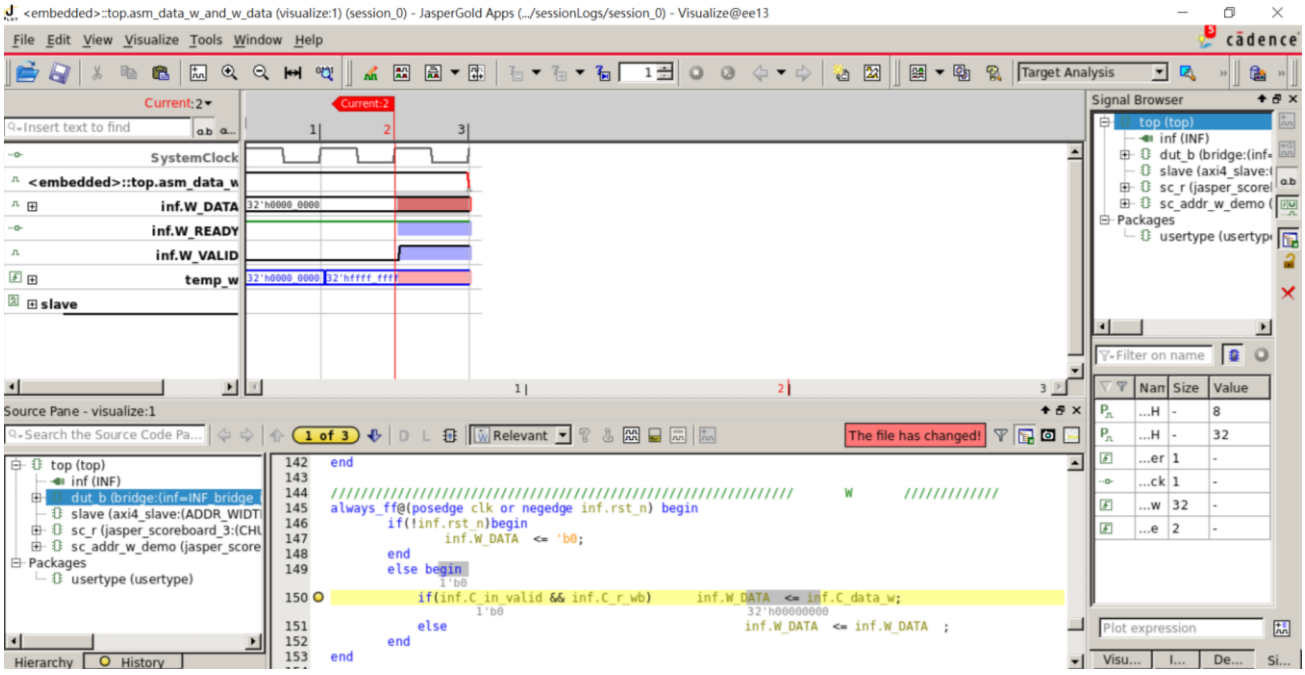
3rd bug: when (inf.AW_VALID && inf.AW_READY), inf.AW_ADDR isn't equal to {1'b1,7'b0,inf.C_addr,2'b0}



Answer:

```
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};
      if(n_state == AXI_AW && c_state != AXI_AW)    inf.AW_ADDR <= {8'b1000_0000, inf.C_addr, 2'b0};
      else                                          inf.AW_ADDR <= inf.AW_ADDR ;
  end
end
```

4th bug: when (inf.W_VALID && inf.W_READY), W_DATA != inf.C_data_w.



Answer:

```
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;
      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
```

(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.

Ans:

以上四種我使用得比較順手的是 IMC Coverage，他很好地協助我檢查我自己設計的 checker 及 pattern 於何處的 coverage 未達到 100%，而且我自己也覺得 IMC 的介面比較好看。

另外讓我印象最深刻的 tool 是 Jasper SEC，當時在進行 clock gating 時 JG 一直報錯誤，我百思不得其解，找了很久之後才發現有一個地方的 clock gating 所設的範圍真的是錯誤的(雖然不會影響答案)，一解掉 JG 就瞬間通過。令我很佩服製作這個工具的開發者們。

其實自己在使用上還沒有遇到太大的困難，感覺主要是因為助教們都把 script 寫好了，十分謝謝 Cadence 以及辛苦的助教們給我們使用這些強大的工具!