IC Lab Formal Verification Lab11 Quick Test 2023 Fall

Name: 張理為 **Student ID:** 312510144 **Account:** iclab015

(a) What is Formal verification?

Formal Verification is the process of validating the correctness of a design using mathematical methods. Its tools employ various algorithms to verify the design, but they do not execute any temporal checks (不執行時序檢查). It traverses all possible combinations to ensure that all states are reached, preventing potential failures from going undetected. Formal verification can be categorized into the following three types:

Equivalence Checking: The process of verifying whether two designs are functionally identical.

Model Checking: A state-based verification method that employs efficient search techniques to check if a given system satisfies specifications.

Theorem Proving: Using mathematical reasoning methods to verify whether the implemented system meets the design requirements.

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

And list the pros and cons for each.

Formal Verification:

Pros:

- Employs breadth-first search, prioritizing the exploration of all states to cover potential corner cases.
- Accelerates the IC design flow, allowing verification to commence before testbench creation and simulation.
- Requires less testbench effort.
- Results in higher quality, often uncovering bugs that simulation might not catch.
- Offers more determinism, with little or no reliance on randomization.

Cons:

- Involves more time consumption.

Pattern-Based Verification:

Pros:

- Employs depth-first search, using paths to verify the correctness of the design.
- It can ignore states that are unlikely to occur, thereby reducing the time required.
- Testing is more convenient, and some test data can be written in software and then loaded for use.

Cons:

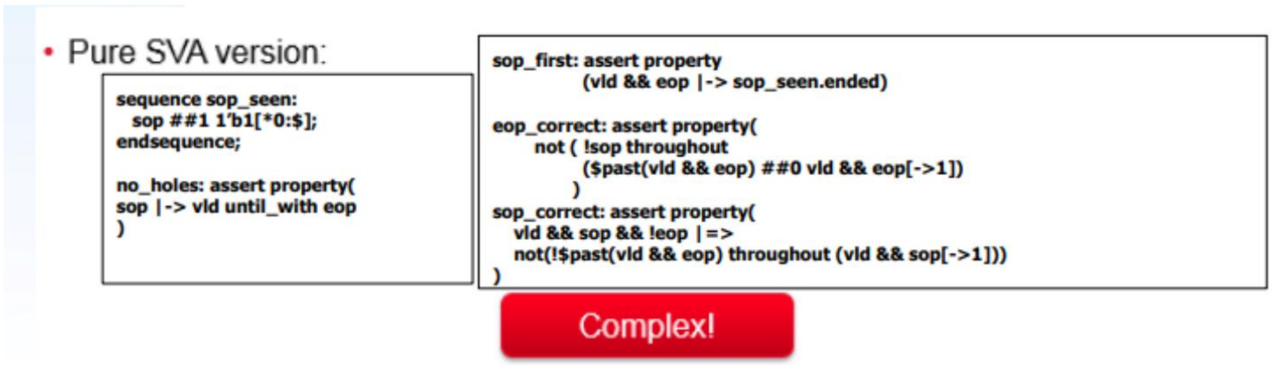
- Requires more testbench effort.
- May miss some errors since it may not test all possibilities.

(b) What is glue logic?

A: When modeling complex behaviors, SVA expressions often become quite intricate. In such cases, glue logic can be employed to observe and track events, which is using additional logical judgments to track the occurrence of events can simplify the writing of SVA.

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

A: Due to the complexity of SVA expressions, which may involve multiple conditions and logical operations, the utilization of glue logic can assist in transforming these intricate SVA expressions into clearer and more identifiable forms. For instance, the assertion appears very complicated without the use of glue logic.



- Pure SVA version:

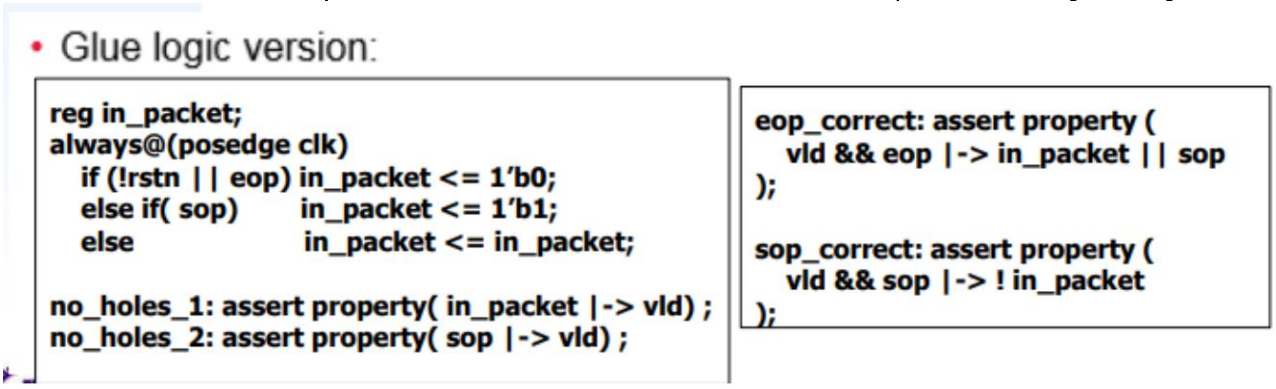
```
sequence sop_seen:
  sop ##1 1'b1[*0:$];
endsequence;

no_holes: assert property(
sop |-> vld until_with eop
)
```

```
sop_first: assert property
            (vld && eop |-> sop_seen.ended)

eop_correct: assert property(
      not ( !sop throughout
            ($past(vld && eop) ##0 vld && eop[->1])
           )
sop_correct: assert property(
   vld && sop && !eop |=>
   not(!$past(vld && eop) throughout (vld && sop[->1]))
)
```

However, by employing glue logic to introduce additional signals in the figure below, the entire SVA will be simpler than it would be without the incorporation of glue logic.



- Glue logic version:

```
reg in_packet;
always@(posedge clk)
   if (!rstn || eop) in_packet <= 1'b0;
   else if( sop)      in_packet <= 1'b1;
   else               in_packet <= in_packet;

no_holes_1: assert property( in_packet |-> vld) ;
no_holes_2: assert property( sop |-> vld) ;
```

```
eop_correct: assert property (
   vld && eop |-> in_packet || sop
);

sop_correct: assert property (
   vld && sop |-> ! in_packet
);
```

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

A:

Functional coverage needs to be manually written, while Code coverage is automatically generated.

Functional coverage requires setting conditions by yourself and check if the design's behavior matches the expected. On the other hand, code coverage examines whether each line has been executed. code coverage’s advantage is detecting areas that might have been overlooked, but the disadvantage is it will detect some cases that may never occur, and the code doesn't need to be executed in those cases.

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

A:

Even if we achieve 100% code coverage, we cannot claim that our verification is sufficient. Code coverage only checks whether we "execute" the code or not, but it does not assess the correctness of the code. As a result, it is impossible to infer whether the assertion is sufficient for comprehensive 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.

A:

COI Coverage: COI Coverage stands for Cone-Of-Influence Coverage. Every assertion is influenced by some cover items, and the path of these cover items resembles a cone. Measuring it takes a little time, and there is no need to run formal engines.

Proof Coverage: Proof Coverage is a subset of COI, representing the region that cannot truly influence the assertion status. COI constitutes the maximum potential of proof coverage. Unlike COI, proof coverage requires an actual proof to take place , and some of the circuit must run on formal engines. It identifies more unchecked code compared to COI measurement, demanding greater tool effort. Moreover, proof coverage is a slower measurement than COI.

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

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

A:

ABVIP:

Definition: Assertion Based Verification IP (ABVIP) constitute a comprehensive set of checkers and RTL designed to ensure protocol compliance. It will be used in verification and contains many well-written assertions and checkers that we can use directly, significantly simplifying the time required for our verification.

Objective: The primary goal is to verify a protocol and analyze its completeness.

Benefit: ABVIP aids designers in verifying new designs more easily and quickly.

Scoreboard:

Definition: A Scoreboard functions as a monitor, observing input and output data of the DUV.

Objective: It is often employed to check for (1) dropped data packets, (2) duplicated data packets, (3) order of data packets, and (4) corrupted data packets.

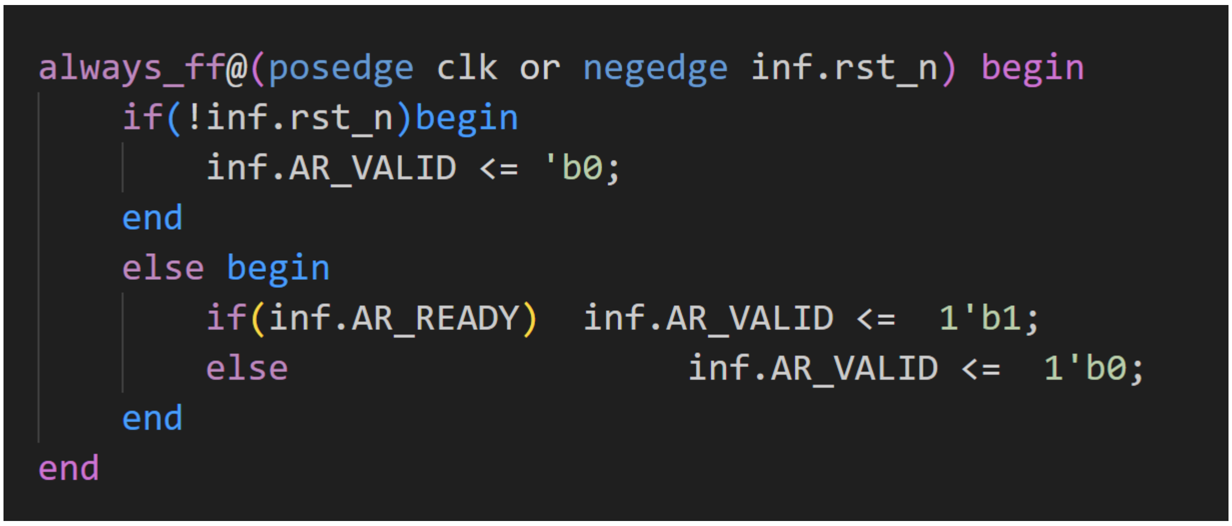
Benefit: Formally optimized to reduce state-space complexity, it also lowers the barrier for adoption.

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

What is the answer of the Lab Exercise?

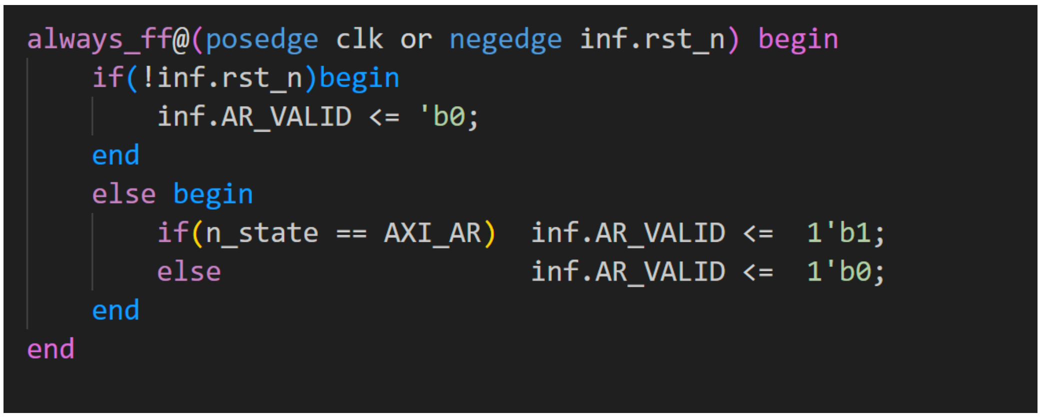
Bug 1: arvalid should remain stable if (arvalid and !arready).

Wrong: if(inf.AR_READY)



```
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;
    end
end
```

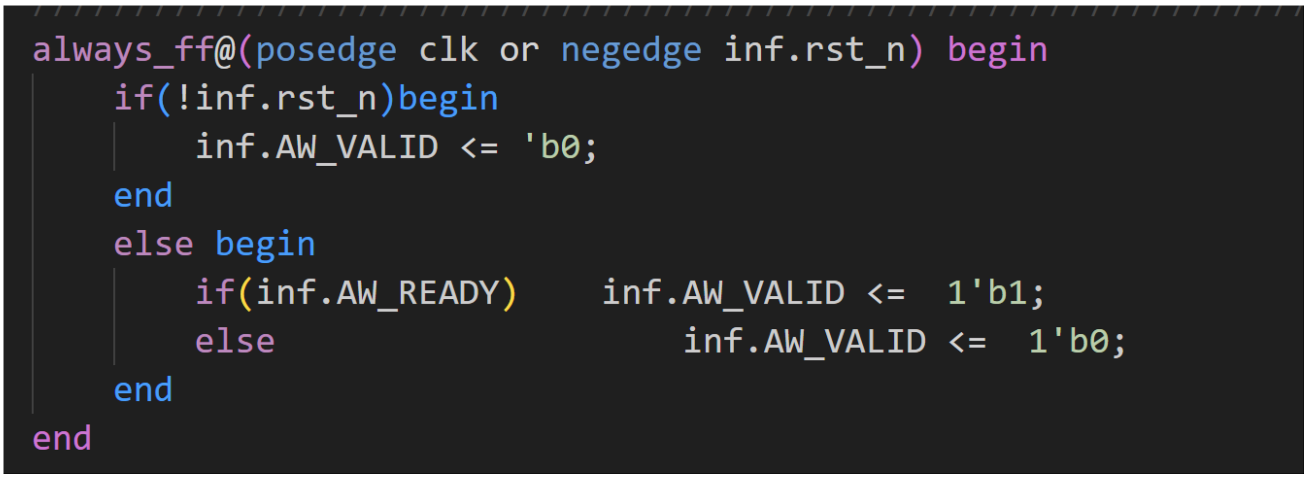
Correct: if(n_state == AXI_AR)



```
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AR_VALID <= 'b0;
    end
    else begin
        if(n_state == AXI_AR)  inf.AR_VALID <=  1'b1;
        else                   inf.AR_VALID <=  1'b0;
    end
end
```

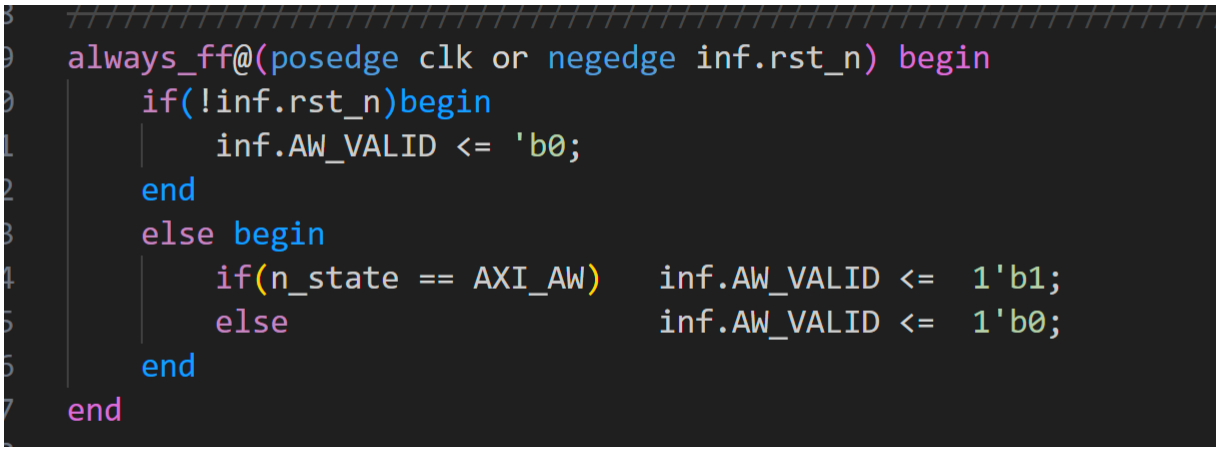
Bug 2: awvalid should remain stable if awvalid and !awready.

Wrong: if(inf.AW_READY)



```
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;
    end
end
```

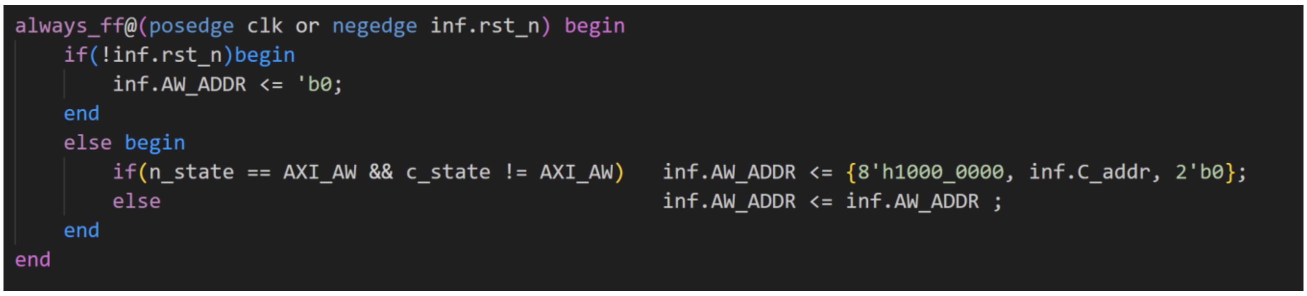
Correct: if(n_state == AXI_AW)



```
always_ff@(posedge clk or negedge inf.rst_n) begin
    if(!inf.rst_n)begin
        inf.AW_VALID <= 'b0;
    end
    else begin
        if(n_state == AXI_AW)   inf.AW_VALID <=  1'b1;
        else                    inf.AW_VALID <=  1'b0;
    end
end
```

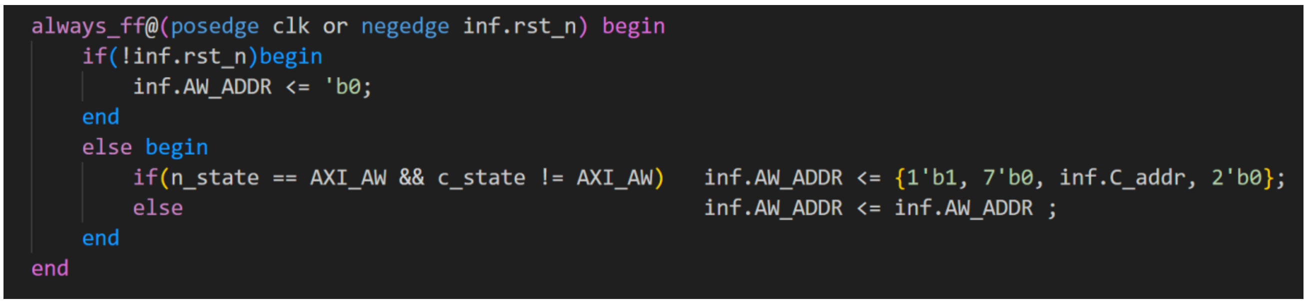
Bug 3: when (inf.AW_VALID && inf.AW_READY), inf.AW_ADDR isn't equal to {1'b1,7'b0,inf.C_addr,2'b0}

Wrong: inf.AW_ADDR <= {8'h1000_0000, inf.C_addr, 2'b0};



```
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};
        else                                         inf.AW_ADDR <= inf.AW_ADDR ;
    end
end
```

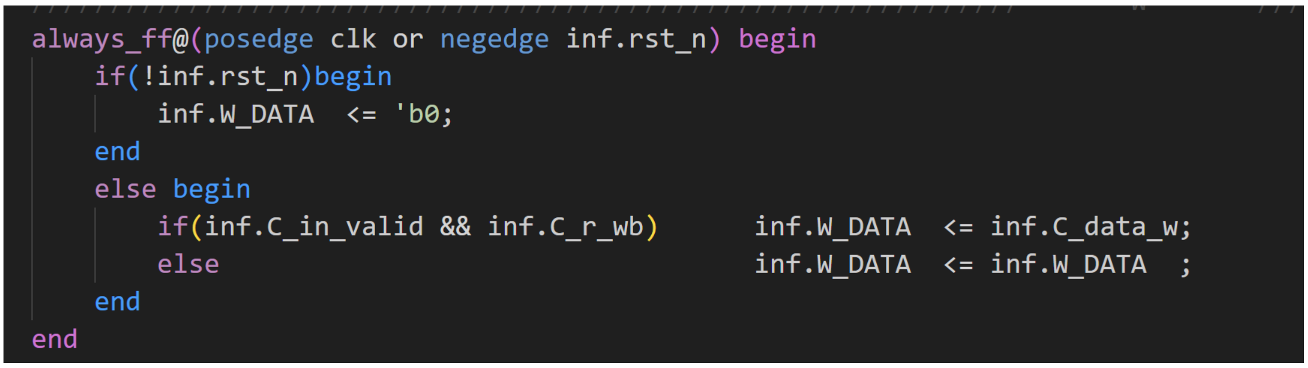
Correct: inf.AW_ADDR <= {1'b1, 7'b0, inf.C_addr, 2'b0};



```
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 <= {1'b1, 7'b0, inf.C_addr, 2'b0};
        else                                         inf.AW_ADDR <= inf.AW_ADDR ;
    end
end
```

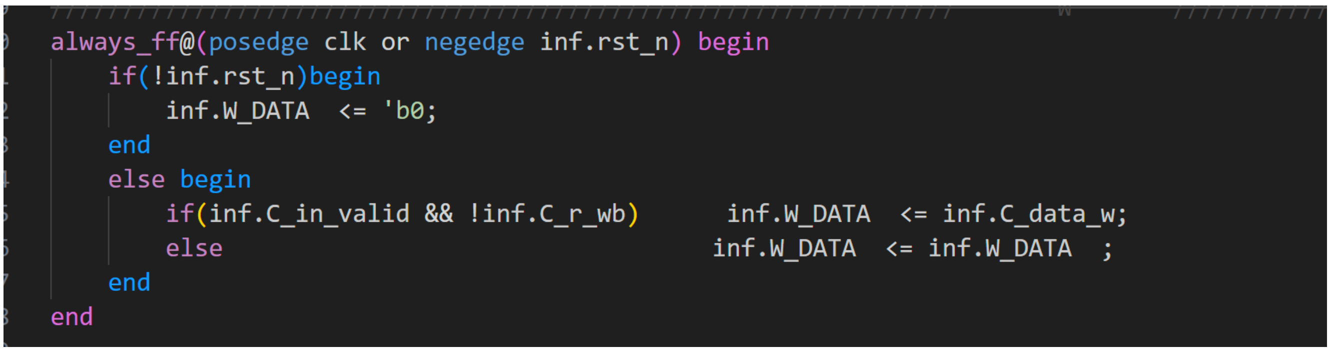
Bug 4: when (inf.W_VALID && inf.W_READY), W_DATA != inf.C_data_w

Wrong: if(inf.C_in_valid && inf.C_r_wb)



```
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
```

Correct: if(inf.C_in_valid && !inf.C_r_wb)



```
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
```

(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.

在這次修課使用的 JasperGold tools 中，我印象最深刻的是 Jasper CDC。在 Lab7 時練習 CDC 時需要跑 Jasper CDC，當時有一個地方忘記插 double DFF，而 Jasper CDC 馬上就將這個 bug 找出來，讓我省去大量的 debug 時間。在使用 tools 的時候，因為助教都已經將 script 都寫完了，所以在使用上非常方便!