

# NYCU-EE IC LAB – FALL 2023

## Formal Verification – Lab11 Exercise

1. Using Assertion Based Verification IP (ABVIP) to verify AXI-4 Lite Master
2. Using Scoreboard & Customized SVA for Verification

### Data Preparation

---

1. Data Extraction: `tar xvf ~iclabTA01/Bonus_formal_verification.tar`
2. The extracted LAB directory contains
  - 00\_TESTBED/
    - Contain TA's **Usertype\_PKG.sv** & **INF.sv**
  - 01\_RTL/
    - Contain **bridge.sv** (bug inserted)
  - 02\_JG/
    - Contain **run\_jg**, **run.tcl**, **top.sv**, **jg.f**, **license.sh**
3. The answer of this lab: `tar xvf ~iclabTA01/Bonus_formal_verification_ans.tar`

### Description

---

In this bonus exercise, you are going to use ABVIP to verify the offered **Customized I/O to AXI4-Lite bridge**. Main focus of this exercise is to check the protocol, and find some bug inside the bridge with build-in scoreboard and customized SVA by Formal Verification tool - **JasperGold**.

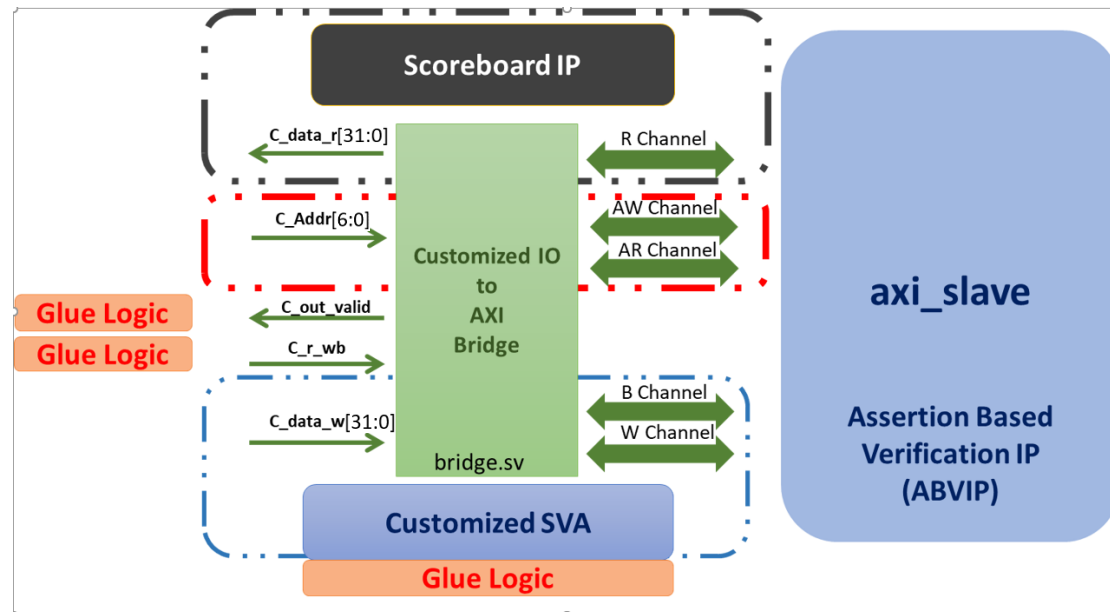
With ABVIP, we could use the build-in SVA property to check our design's AXI4-Lite interface part to make sure our design is reliable to handle any situation on the bus.

As for the interface connected to the design module, it requires customized SVA properties to check the remain of our design.

Different from previous lab (Simulation Based Verification with SVA), this lab demonstrates another perspective of verification – Formal Verification.

Formal Engine would transform our verification environment (including our design, SVA property and auxiliary logic) into another equivalent simplified mathematical problem, such that it uses different algorithms in the backend to finite number of traces to prove whether our design is verified or not.

## Block Diagram of the Verification Environment



## Customized Signal Behavior

### Input:

name	width	Send to	
C_addr	7-bit	bridge	C_addr Indicates which address we want to access. <b>(Bridge would do Address Conversion)</b>
C_data_w	32-bit	bridge	The data to over right DRAM.
C_in_valid	1-bit	bridge	High when the system is ready to communicate with bridge.
C_r_wb	1-bit	bridge	1'b1: Read axi_slave. 1'b0: Write axi_slave.

### Output:

name	type	Send to	note
C_out_valid	1-bit	design	High when returned data from is ready.
C_data_r	32-bit	design	The returned data from DRAM.

## Bugs

---

- I. There are **4 hardware bugs** TA has insert in the design.
- II. You should finish the blank in the top.sv to check **data integrity**
  - A. Scoreboard Port Connection (inf.AW\_ADDR)
  - B. Customized Assertion (inf.C\_data\_w & W\_DATA)
    - ✓ Method1. Glue Logic + Assertion
    - ✓ Method2. Undetermined Constant

## Note of Submission

---

1. **This lab does not have a second demo opportunity!!!**
2. No need to submit Verilog files, but you need to submit the answer of Quick Test, which include the answer of this lab, before 2023/12/11 12:00
3. Please submit the PDF format on the **E3 platform**, and name the file as **Quick\_Test\_iclabxxx.pdf**, for example, Quick\_Test\_iclab180.pdf.
4. Please name the file according to the file naming convention. If not, it will be considered a **naming error**, and **five points will be deducted** from the score.

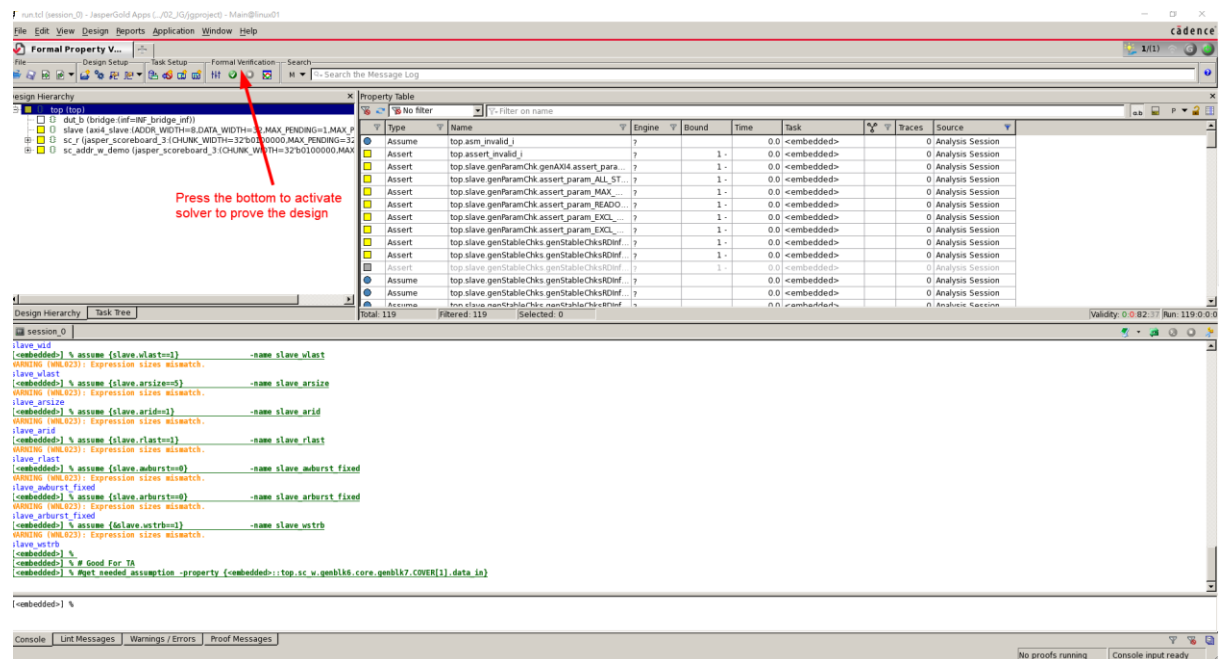
**Enjoy Bug Hunting!!!**



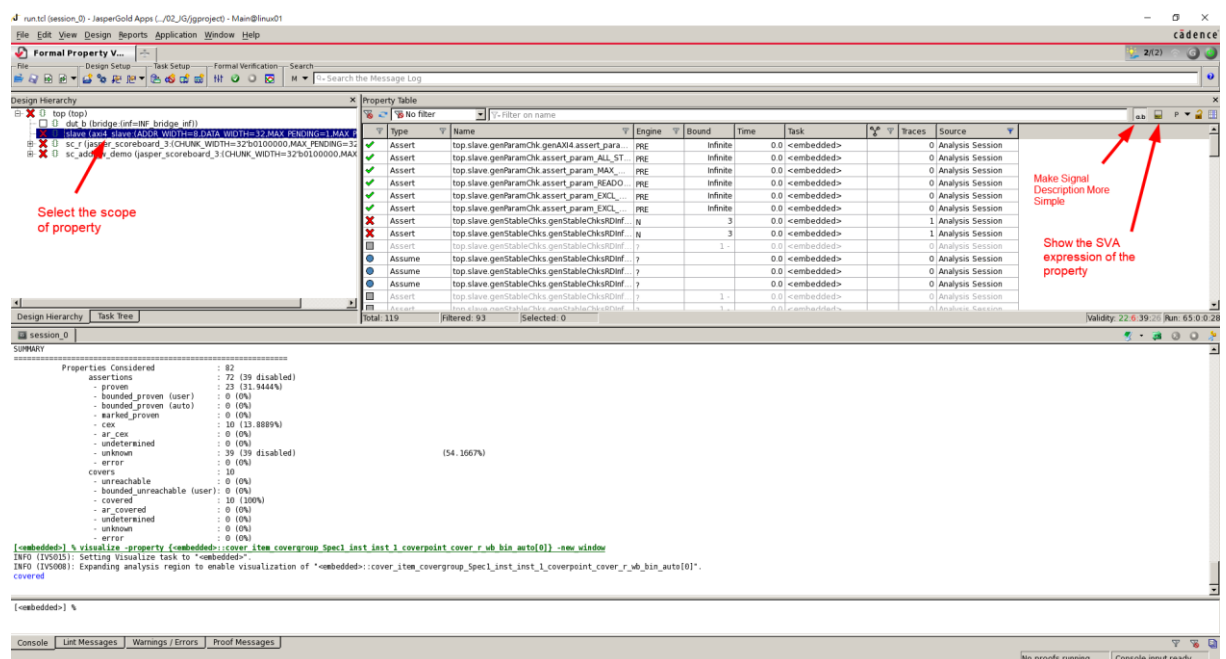
## Step to run

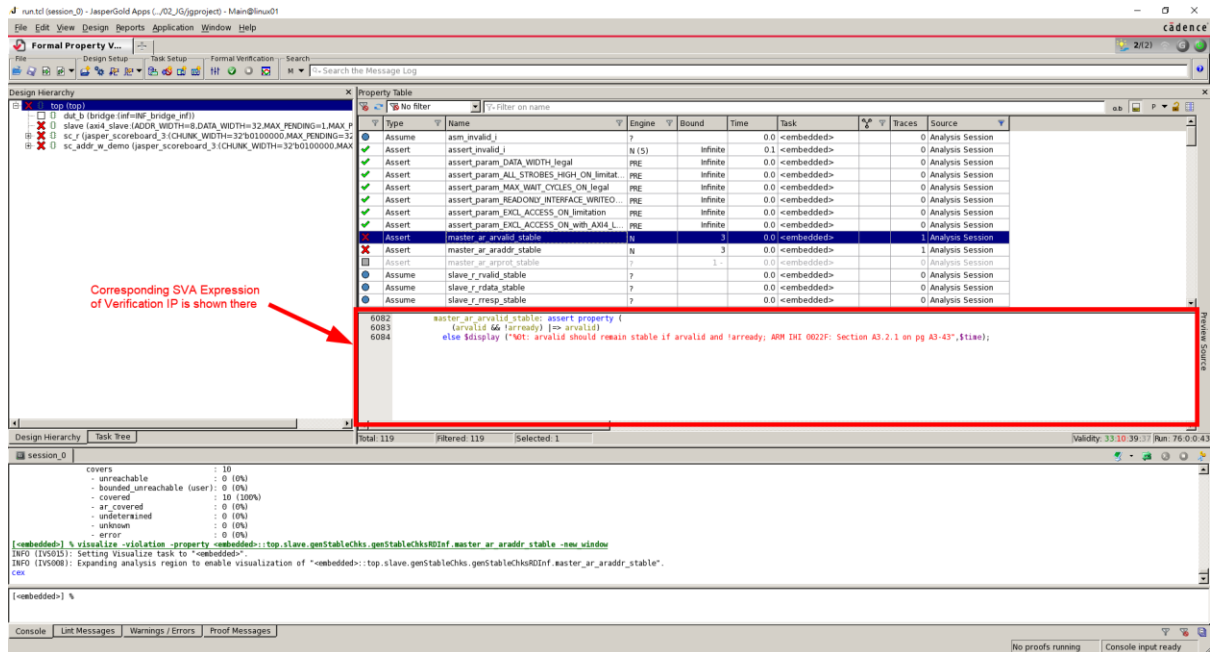
1. Go to Directory  
~/.Bonus\_formal\_verification/Exercise/02\_JG/
2. execute the command: `source license.sh` (Important!!!)
3. execute the command: `./run_jg`

JasperGold would automatically start reading files (01\_RTL/bridge.sv , 02\_JG/top.sv)



## After running proof

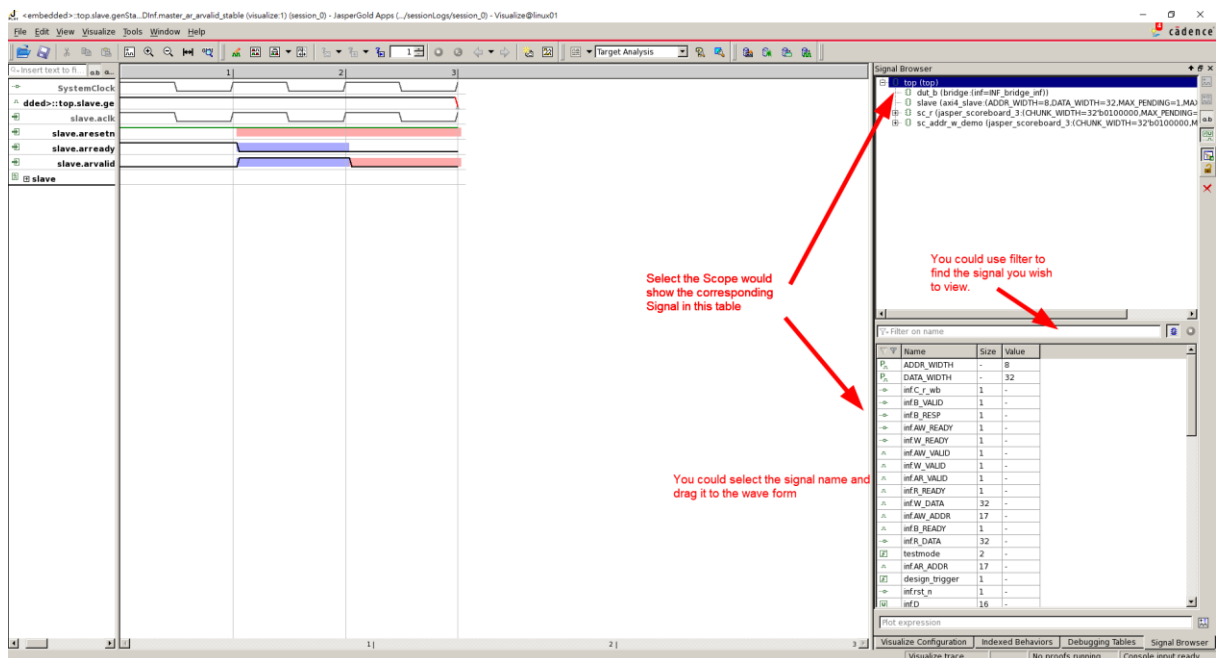




You could double click the **red check** to see the waveform which follow the property

If there is assertion violation in your design, open the waveform and try your best to analyze the root cause of this violation and debug it.

## Basic Tutorial to view waveform



<embedded>-top.slave.gen5ta...\_Dinf.master\_ar\_valid\_stable (visualize:1) (session:0) - JasperGold Apps (...) - Visualize@linux01

File Edit View Visualize Tools Window Help

Target Analysis

Signal Browser

Reduce distractions in waveform

1 2 3

SystemClock

er\_ar\_valid\_stable

slave.ack

slave.resetn

slave.arready

slave.arvalid

slave

inf.ar\_valid

You could double click the waveform to see the corresponding source code

After Drag the Signal Name

inf.ar\_valid (inf=inf\_bridge\_inf)

inf.slave (axi4\_slave (ADDR\_WIDTH=8, DATA\_WIDTH=32, MAX\_PENDING=1, MA...))

sc.r (jasper\_scoreboard\_3 (CHUNK\_WIDTH=32, b01000000, MAX\_PENDING=...))

sc\_addr\_w\_demo (jasper\_scoreboard\_3 (CHUNK\_WIDTH=32, b01000000, M...))

Filter on name

Name	Size	Value
ADDR_WIDTH	8	-
DATA_WIDTH	32	-
inf.C_r_wb	1	-
inf.B_VALID	1	-
inf.B_RESP	1	-
inf.AW_READY	1	-
inf.W_READY	1	-
inf.AW_VALID	1	-
inf.W_VALID	1	-
inf.ar_valid	1	-
inf.R_READY	1	-
inf.W_DATA	32	-
inf.AW_ADDR	17	-
inf.B_READY	1	-
inf.R_DATA	32	-
testmode	2	-
inf.ar_addr	17	-
design_trigger	1	-
inf.rst_n	1	-
inf.D	16	-

Visualize Configuration Indexed Behaviors Debugging Tables Signal Browser

Visualize trace No proofs running Console input ready

<embedded>-top.slave.gen5ta...\_Dinf.master\_ar\_valid\_stable (visualize:1) (session:0) - JasperGold Apps (...) - Visualize@linux01

File Edit View Visualize Tools Window Help

Target Analysis

Signal Browser

Reduce distractions in waveform

1 2 3

SystemClock

edded>:top.slave.g

slave.ack

slave.resetn

slave.arready

slave.arvalid

slave

inf.ar\_valid

Second Cycle of the Waveform and the corresponding value of the

Source Pane - visualize:1

Search the Source Code

Why at iteration 2 for inf.ar\_valid (1 of 2)

```

80 end
81
82
83 /////////////////////////////////////////////////////////////////// AR
84
85 always_ff@(posedge clk or negedge inf.rst_n) begin
86   if(!inf.rst_n)begin
87     inf.ar_valid <= 'b0;
88   end
89   else begin
90     if(!inf.ar_ready) inf.ar_valid <= 'b1;
91     '1'0; inf.ar_valid <= 'b0;
92   end
93 end
94
95 always_ff@(posedge clk or negedge inf.rst_n) begin
96   if(!inf.rst_n)begin
97     inf.ar_addr <= 'b0;
98   end
99   else begin
100     if(in_state == AXI_AR_66 c_state != AXI_AR) inf.ar_addr <= {1'b1, 6'b0, inf.C_addr, 2'b0};
101     inf.ar_addr <= inf.ar_addr;
102   end
103 end
104
105 /////////////////////////////////////////////////////////////////// R
106
107 always_ff@(posedge clk or negedge inf.rst_n) begin
108   if(!inf.rst_n)begin
109     inf.r_ready <= 'b0;
110   end
111   else begin
112     if(inf.r_valid) inf.r_ready <= 'b1;
113     else inf.r_ready <= 'b0;
114   end
115 end
116
117

```

inf.ar\_valid (inf=inf\_bridge\_inf)

inf.slave (axi4\_slave (ADDR\_WIDTH=8, DATA\_WIDTH=32, MAX\_PENDING=1, MA...))

sc.r (jasper\_scoreboard\_3 (CHUNK\_WIDTH=32, b01000000, MAX\_PENDING=...))

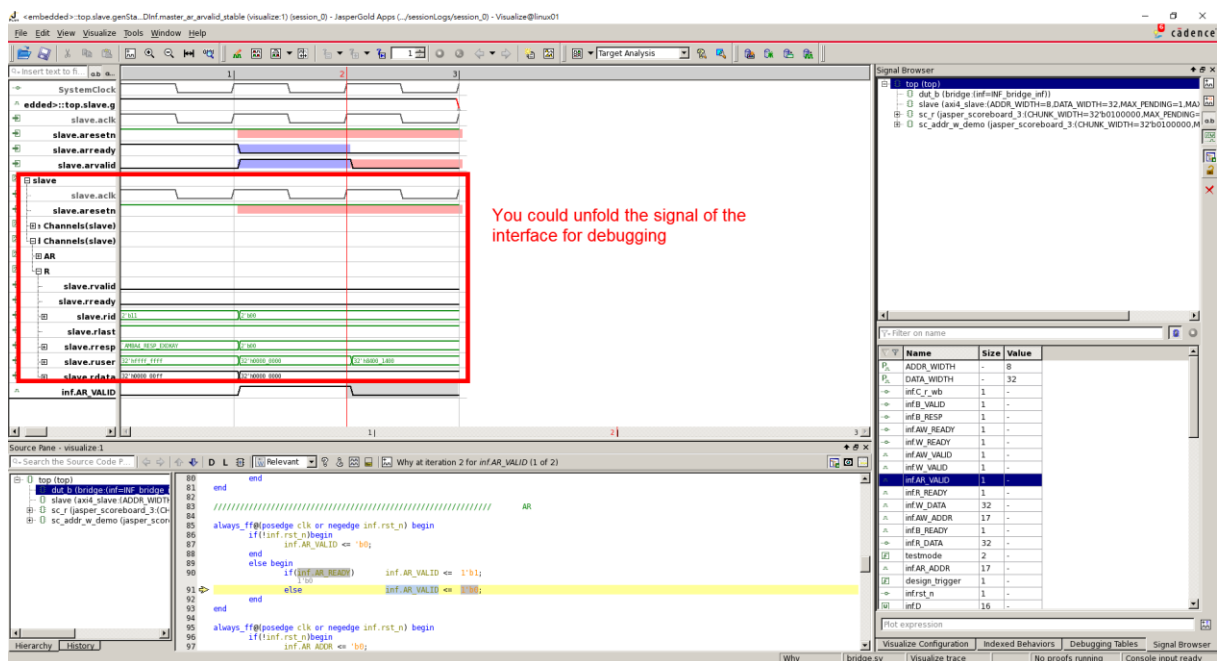
sc\_addr\_w\_demo (jasper\_scoreboard\_3 (CHUNK\_WIDTH=32, b01000000, M...))

Filter on name

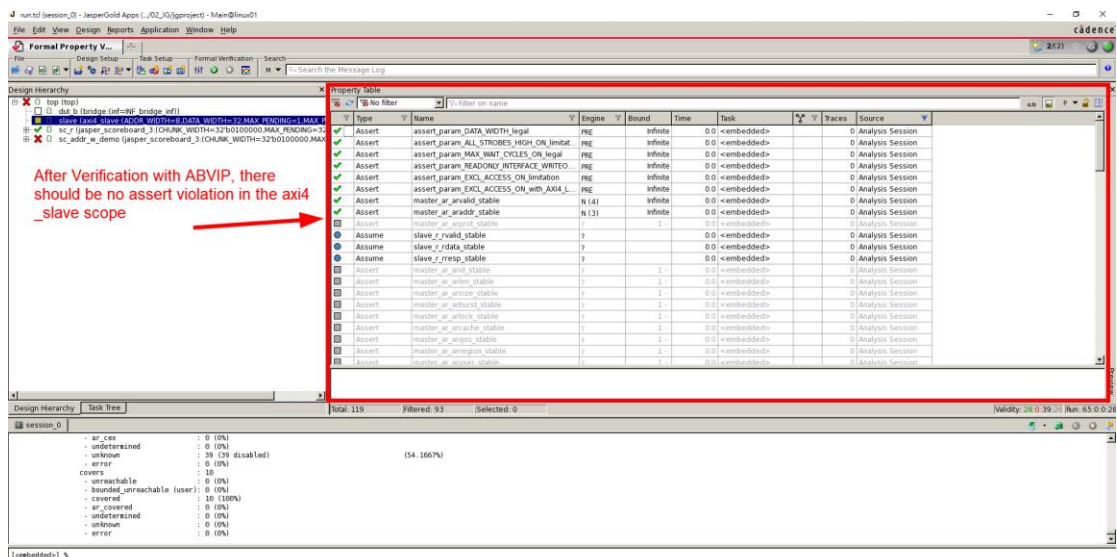
Name	Size	Value
ADDR_WIDTH	8	-
DATA_WIDTH	32	-
inf.C_r_wb	1	-
inf.B_VALID	1	-
inf.B_RESP	1	-
inf.AW_READY	1	-
inf.W_READY	1	-
inf.AW_VALID	1	-
inf.W_VALID	1	-
inf.ar_valid	1	-
inf.R_READY	1	-
inf.W_DATA	32	-
inf.AW_ADDR	17	-
inf.B_READY	1	-
inf.R_DATA	32	-
testmode	2	-
inf.ar_addr	17	-
design_trigger	1	-
inf.rst_n	1	-
inf.D	16	-

Visualize Configuration Indexed Behaviors Debugging Tables Signal Browser

Visualize trace No proofs running Console input ready



The figure of full proof of your assertion property



If the AXI-Lite Interface is already been proven, you could start debugging in the other scope such as scoreboards and or your customized SVA property in the top

The assertion property should not be violated, you could double click to check the waveform to see whether it match your property. You could scrutinize your **assumption, assertion or glue logic** to find the bugs with your SVA.

## Development Log:

Li-Wei, Liu @OASIS LAB

ICLAB2019Fall

Kai-Jyun, Hung @OASIS LAB	ICLAB2020Spring
Kai-Jyun, Hung @OASIS LAB	ICLAB2020FALL
Lin-Hung, Lai @Si2 LAB	ICLAB2021Spring
Wen-Yue, Lin @Si2 LAB.	ICLAB2021Fall
Po-Kang, Chang @Si2 LAB.	ICLAB2022Fall
Chan-Pin, Hung@Si2 LAB	ICLAB2023Spring