

# **Cadence JasperGold Tutorial**

## **The Superlint App**

**Version 1.1 | 01-June-2023**



**Portland State University**

**©2023**

**Disclaimer:**

The contents of this document are confidential, privileged, and only for the information of the intended recipient and may not be published or redistributed.

The design example in this tutorial is not supplied. You need to use your own files to follow the tutorial steps and instructions.

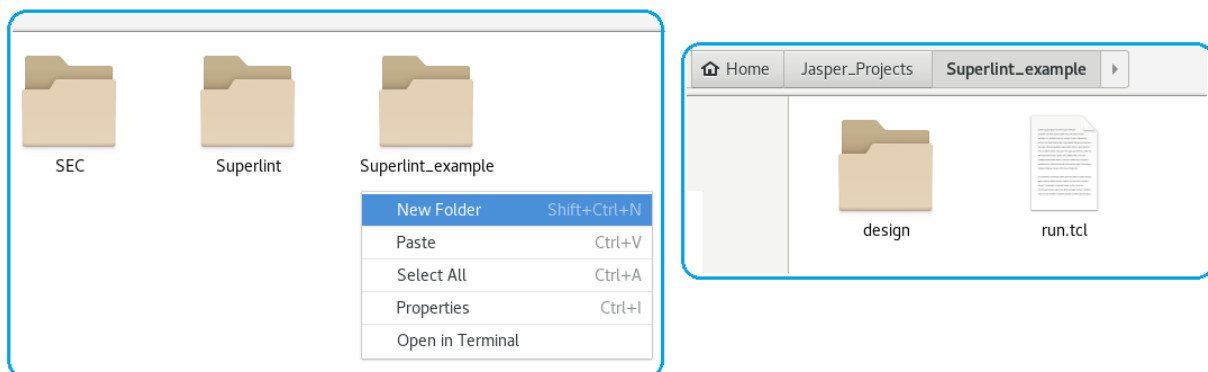
# Table of Contents

Introduction .....	3
About and Usage of the Superlint App.....	4
Design Files .....	5
TCL File .....	6
Application Setup .....	7
Invoking JasperGold .....	8
User Interface Details .....	9
Invoking the JasperGold Superlint GUI .....	12
Loading TCL File .....	11
Invoking Jaspergold Superline Along with TCL File .....	12
Running Files .....	13
Detecting Errors .....	14
Looking into the falsified properties .....	15

## Introduction

While JasperGold (JG) can be used in GUI mode directly without the need for a TCL (Tool Command Language) file, this approach can be time-consuming and inefficient. Therefore, we recommend using TCL scripts to guide JG's actions. These scripts can define various aspects of the analysis, such as the app to be used, the files to be analyzed, and the clock cycles. However, instead of creating TCL scripts from scratch, we can take advantage of pre-made templates. By modifying these templates, we can interact with JG in a way that suits our project's specific requirements.

To get started, we need to set up the VNCserver as described in the previous tutorials and launch the Linux GUI. To keep things organized, we should create a top-level folder for the design we want to analyze. In this example, we'll call it "Superlint\_Example," but it's important to avoid using spaces in file and folder names. Within this top-level folder, we can create an optional subfolder named "design" to store the Verilog or SystemVerilog designs we want to analyze. Additionally, we'll create a TCL file in the top-level folder (not inside the "design" subfolder) to store any TCL commands we want to execute.



*Figure 1. Creating folders to organize design and TCL files.*

## About and Usage of the Superlint App

The Superlint App is a tool that helps engineers verify complex designs and catch potential issues early in the development process. It does this by running different kinds of checks on the design, such as:

Lint checks, which find common coding errors.

DFT (Design for testability) checks, which ensure that the design is testable, and

Automatic formal checks, which verify that the design meets certain specifications.

By using Superlint, engineers can fix issues before validation begins, which saves time and money. Additionally, Superlint can be used as a way to continually check the design as it evolves, helping to catch any new issues that may arise.

Lint	Superlint
Basic linting typically involves analyzing the RTL code for syntax errors, naming conventions, and coding style. This technique can detect simple errors such as incorrect usage of keywords, missing semicolons, and undefined variables. Basic linting can be performed quickly and is a useful tool for ensuring code quality and consistency.	Superlinting goes beyond basic linting and provides more comprehensive analysis of the RTL code. Superlinting includes DFT (Design-for-Testability) checks, which ensure that the design is testable and can be easily validated during the testing phase. It also includes automatic formal checks, which use mathematical methods to verify that the design meets its specification. Superlinting can detect complex errors such as race conditions, deadlocks, and incorrect use of registers.

In summary, while basic linting is a useful tool for ensuring code quality and consistency, superlinting provides additional features and checks that can identify and fix more complex issues in the RTL code. By using superlinting, engineers can improve the overall quality of their designs and reduce the risk of errors during testing and validation.

We need two inputs for the Superlint app:

- RTL design files in Verilog/SystemVerilog/VHDL formats
- TCL file

## Design File

The SystemVerilog design file should be placed in the Design folder, while the TCL file belongs in the main folder. It's recommended to name the folders with lowercase letters since JG is case-sensitive. Additionally, ensure that there are no spaces in the names of the folders or files.

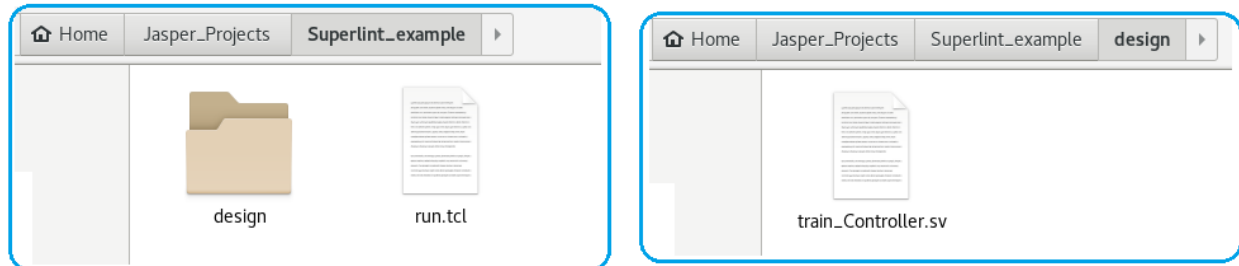


Figure 2. Showing the SystemVerilog and TCL files in the correct folder locations.

The Superlint App runs automated checks, therefore, writing assertions is not required for this app.

```

// Cadence JasperGold Superlint App Example
// Portland State University - Train Controller Example

module train_Controller (input logic clk, reset, s1, s2, s3, s4, s5,
                        output logic sw1, sw2, sw3,
                        output logic [2:0] indicator,
                        output logic [1:0] DA, DB);

logic [2:0] counter;
  
```

Figure 3. Example of the design we are using in this tutorial.

When performing formal analysis in general, it's important to keep track of your top module names. In our example here our module name is train\_Controller

## TCL File

Next, we will set up the TCL file. Below is the TCL file (*Figure 5*), which you can use as a template for your functional checks on VC Formal.

```

# Cadence JasperGold Superlint App Example
# Portland State University - Formal Verification team

# Initialize the Superlint App
check_superlint -init

# Clear any previous analysis, and read the SystemVerilog Design file
analyze -clear
analyze -sv design/train_Controller.sv ①

# Elaborate the design
elaborate -bbox_a 1024

# Setup clocks and reset. In our design, the clock signal is "clk"
# and we have a positive edge-triggered reset signal called "reset"
clock clk ②
reset reset ③

# Extract the superlint checks
check_superlint -extract

# Generate a task specifically for design assumptions and duplicate the assumptions from the embedded task
task -create design_assumptions -copy_assumes -copy_related_covers -source_task <embedded>

# prove the extracted properties
set_max_trace_length 50
check_superlint -prove -task {<SL_*>}

```

*Figure 4. Annotated TCL template file.*

1. Location of the design file should be specified for JG to locate it.
2. Name of the clock signal. If your design uses a clock, you need to type the exact name of your clock. You can also use the `-infer` switch to have JG automatically infer the clock from the design. If you're not using a clock (if it's a combinational logic design), you can use the switch `none`.
3. Name of the reset signal in your design. If your design uses a reset signal, you need to type the exact name of your reset. If you're not using a reset, you can use the switch `none`.

## Application Setup

There are multiple ways to invoke the JasperGold GUI and load the TCL script.

In this tutorial, we will show two methods for doing so. When using either method, it is important that you open the terminal in the Superlint\_example folder.

- [Invoke JasperGold GUI](#), then manually choose the app and load the TCL script in the application:

```
$jg
```

**OR**

- [Invoke JasperGold GUI in the superlint app](#), then manually load the TCL script in the application:

```
$jg -superlint
```

**OR**

- [Invoke JasperGold GUI in the superlint app and TCL script](#) in one command:

```
$jg -superlint run.tcl
```

'run.tcl' is the name of the TCL file we are using. If your file name differs from this, you will need to change it accordingly in this command.

We will go through both of these methods in the following sections.

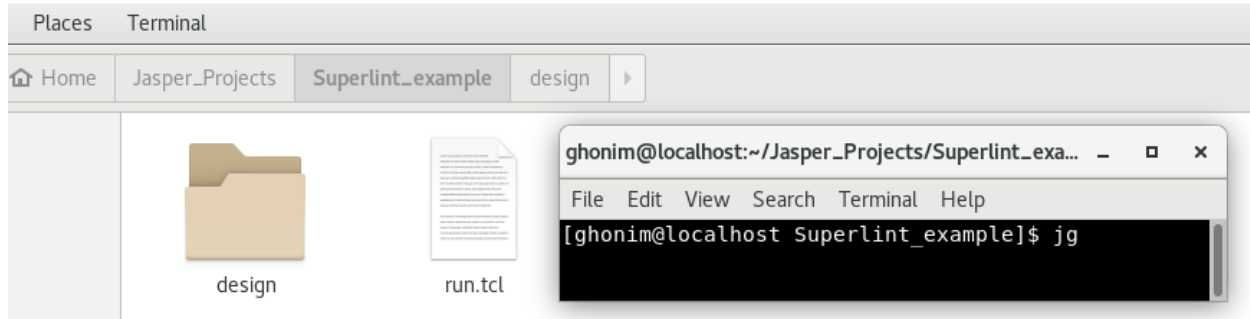


## Invoking JasperGold GUI

To proceed with invoking VC Formal:


- 1) Inside the “Superlint\_example” folder, right-click the whitespace and choose “Open in Terminal”
- 2) In the terminal, type in the command:

`jg`



*Figure 5. Invoking JasperGold in the terminal.*

Note: You may have to wait a few seconds for the program to start up.

You should then see the JasperGold GUI as shown in *Figure 8* below. JasperGold will always open in the Formal Property Verification app unless instructed otherwise in the terminal or the tcl file. To change/add other modes or apps, we can click on the + sign, the one boxed in blue in the figure below. . The GUI interface will look different depending on the app being used.

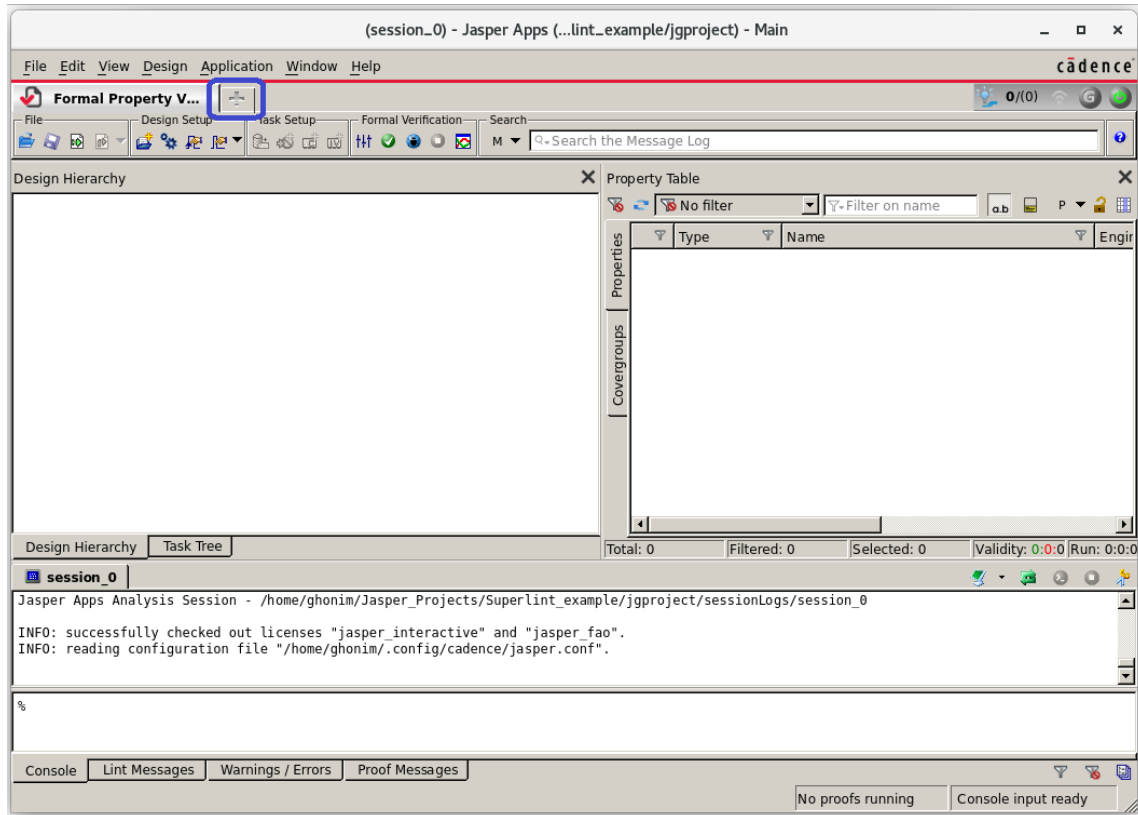


Figure 6. JasperGold introductory screen in the Formal Property Verification app.

After clicking on the + icon, we can then choose the JG app in which we want to work. Here we will choose the Superlint App as shown below.

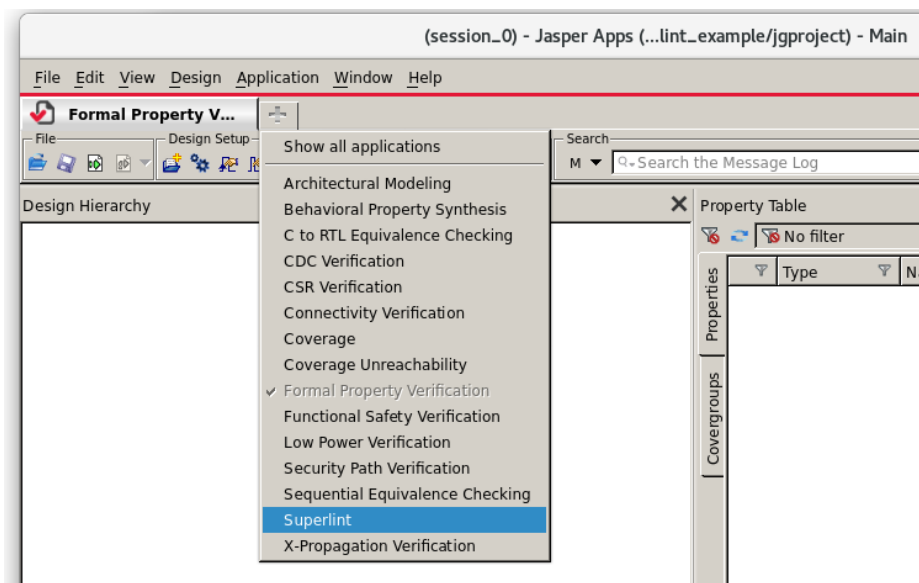
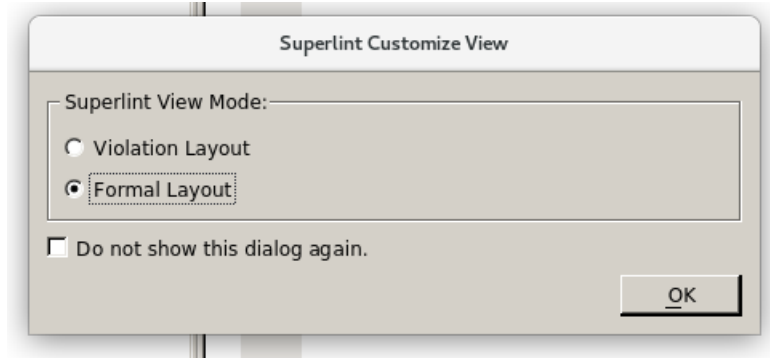
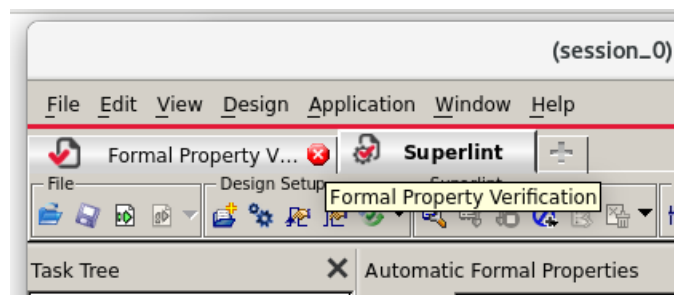


Figure 7. Selecting the Superlint app



*Figure 8. Selecting the superlint view mode the first time you run it*

You'll then be asked to choose the Superlint View mode, you can choose either of them depending on your focus of the analysis. You can easily switch between them later. Here we recommend choosing the Formal Layout as a start.



*Figure 9. Choose the right mode you want to work with "Superlint" in our case. Close any other apps you don't need.*

If not needed anymore, you can hover over the Formal Property Verification tab and click on “x” to close it.

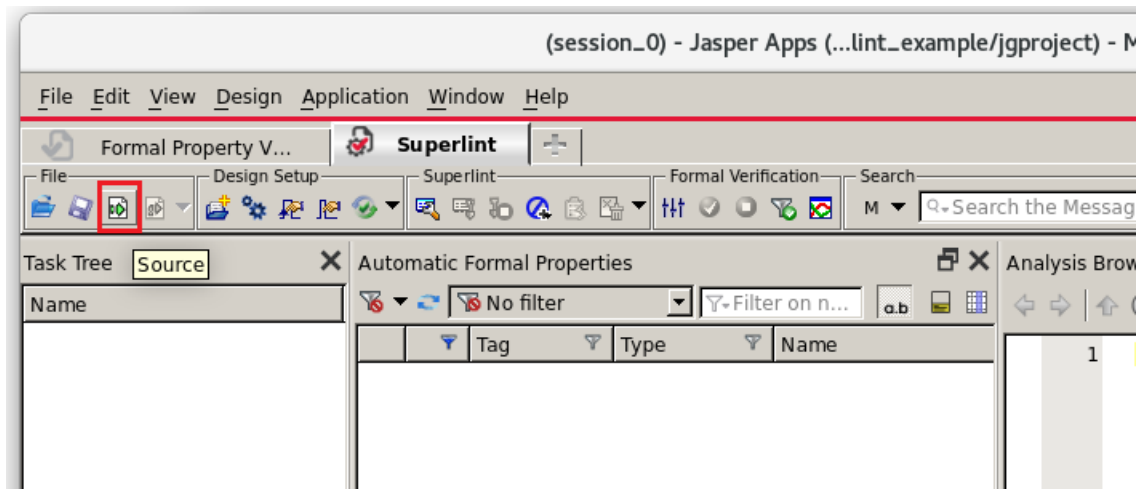


Figure 10. Superlint App interface. We can click on “Source” to choose and load the TCL file.

To load the TCL file, click on the source icon, which can be found in the File section in the top left, the one boxed in red in the figure above.

## Invoking JasperGold in the Superlint App:

- 1) Inside the “Superlint\_example” folder, right-click the whitespace and choose “Open in Terminal”
- 2) In the terminal, type in the command:

```
jg -superlint
```

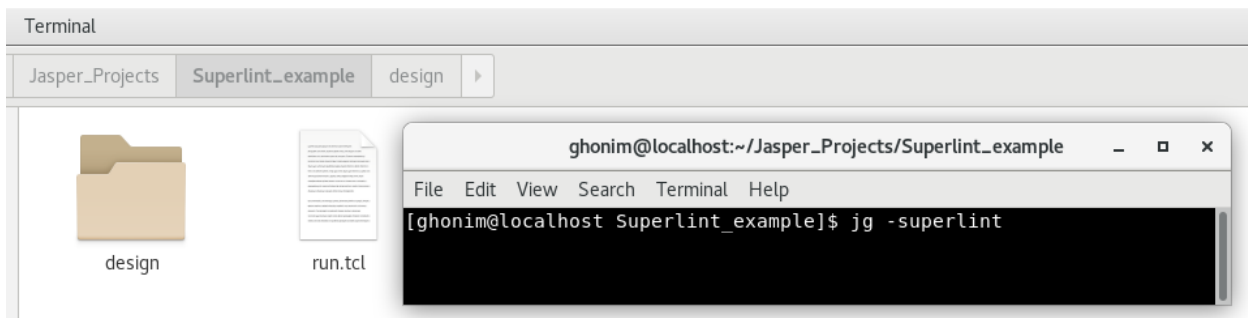


Figure 11. Invoking JG and choosing the superlint app from the terminal

The last, and most efficient option is to load JasperGold in the app you want “superlint in our case” and load the TCL file in the same step as shown below:

```
jg -superlint run.tcl
```

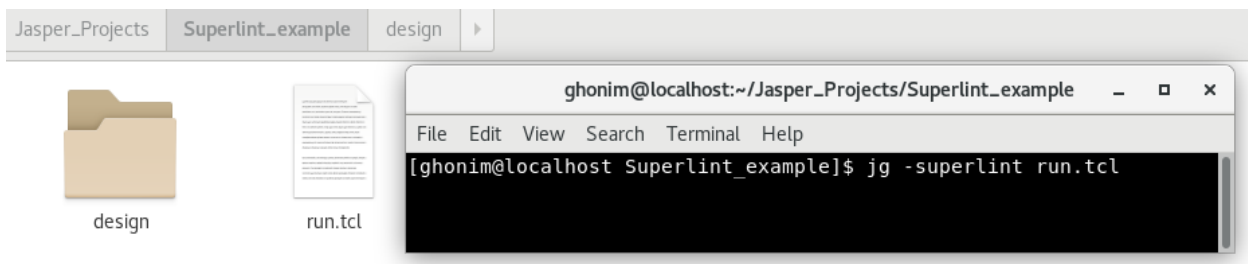


Figure 12. Invoking JG, Choosing the superlint app, and loading the TCL file from the terminal

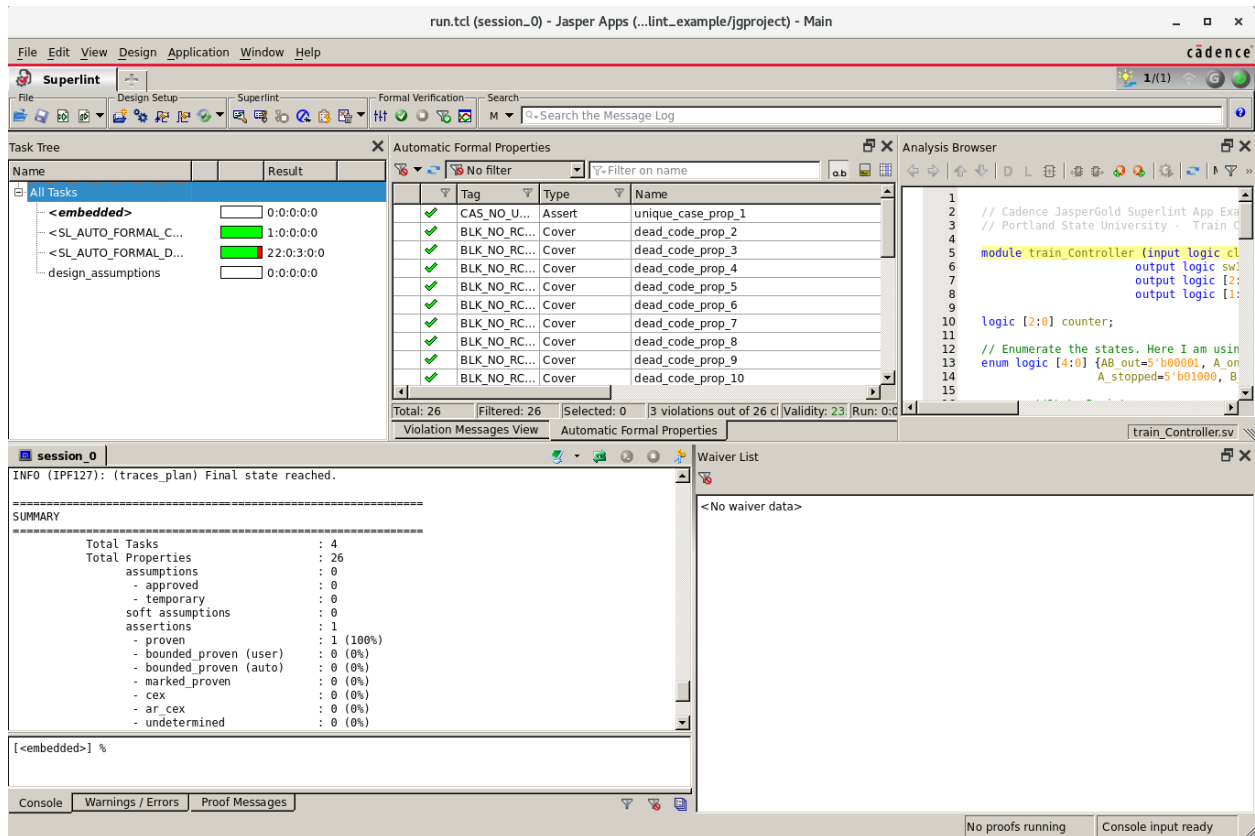


Figure 13. GUI of the JasperGold Superlint App

This is the interface we get after running JG in the Superlint app and loading the TCL file. Some information and analysis will run automatically given our TCL file, and to run the formal engines, we can click on the green tick icon under formal verification.

```

session_0
INFO (IPF127): (traces_plan) Final state reached.

=====
SUMMARY
=====
Total Tasks                : 4
Total Properties           : 26
assumptions                : 0
  - approved               : 0
  - temporary              : 0
soft assumptions           : 0
assertions                 : 1
  - proven                 : 1 (100%)
  - bounded_proven (user)  : 0 (0%)
  - bounded_proven (auto) : 0 (0%)
  - marked_proven         : 0 (0%)
  - cex                   : 0 (0%)
  - ar_cex                : 0 (0%)
  - undetermined          : 0 (0%)

[<embedded>] %

```

Figure 14. JG Superlint Session Summary information

We also have a session window of the screen with a summary of the analysis. This summary gives the tutorial number of properties checked, 26 in our case, how many assertions were placed, and so on.

Task Tree			
Name			Result
All Tasks			
<embedded>			0:0:0:0:0
<SL_AUTO_FORMAL_C...			1:0:0:0:0
<SL_AUTO_FORMAL_D...			22:0:3:0:0
design_assumptions			0:0:0:0:0

Figure 15. JG Superlint Task tree progress and results

When we run the analysis, we see that the

	Tag	Type	Name
✓	CAS_NO U...	Assert	unique_case_prop_1
✓	BLK_NO_RC...	Cover	dead_code_prop_2
✓	BLK_NO_RC...	Cover	dead_code_prop_3
✓	BLK_NO_RC...	Cover	dead_code_prop_4
✓	BLK_NO_RC...	Cover	dead_code_prop_5
✓	BLK_NO_RC...	Cover	dead_code_prop_6
✓	BLK_NO_RC...	Cover	dead_code_prop_7
✓	BLK_NO_RC...	Cover	dead_code_prop_8
✓	BLK_NO_RC...	Cover	dead_code_prop_9
✓	BLK_NO_RC...	Cover	dead_code_prop_10

Total: 26   Filtered: 26   Selected: 0   3 violations out of 26 c   Validity: 23   Run: 0:0

Figure 16. JG Superlint Automatic Formal Properties view

We can also see in the screen the automatic formal properties that were generated and are being analyzed in this code. Those covers, assertions and properties in general can be verified or falsified.

	Tag	Type	Name	Engin	Bound	Time	Task
✓	BLK_NO_RC...	Cover	dead_code_prop_11	Ht	2	0.0	<SL_AUTO_FORMAL_DE...
✓	BLK_NO_RC...	Cover	dead_code_prop_12	Ht	3	0.0	<SL_AUTO_FORMAL_DE...
✓	BLK_NO_RC...	Cover	dead_code_prop_13	Ht	3	0.0	<SL_AUTO_FORMAL_DE...
✓	BLK_NO_RC...	Cover	dead_code_prop_14	Ht	3	0.0	<SL_AUTO_FORMAL_DE...
✓	BLK_NO_RC...	Cover	dead_code_prop_15	Ht	3	0.0	<SL_AUTO_FORMAL_DE...
✗	BLK_NO_RC...	Cover	dead_code_prop_16	Ht (1)	Infinite	0.0	<SL_AUTO_FORMAL_DE...
✓	BLK_NO_RC...	Cover	dead_code_prop_17	Ht	2	0.0	<SL_AUTO_FORMAL_DE...
✓	BLK_NO_RC...	Cover	dead_code_prop_18	Ht	2	0.0	<SL_AUTO_FORMAL_DE...
✓	BLK_NO_RC...	Cover	dead_code_prop_19	Ht	3	0.0	<SL_AUTO_FORMAL_DE...
✓	BLK_NO_RC...	Cover	dead_code_prop_20	Ht	3	0.0	<SL_AUTO_FORMAL_DE...

Total: 26   Filtered: 26   Selected: 1   3 violations out of 26 checks   Validity: 23:3:0:0   Run: 0:0:0:26

```

30 AB_out: begin if (s)
31 A_on2: begin if (s)
32 B_on2: begin if (s)
33 A_stopped: begin if (s)
34 B_stopped: begin if (s)
35 default: begin $display('
36 Next=AB out;
37 end
38 endcase
39
40
41 ng the state outputs
42 comb begin
43 case (State)
44 begin sw1=0: sw2=0;

```

Figure 17. JG Superlint Automatic Formal Properties and code line of the falsified cover

In case of a falsified property, we can click on the x sign, or that whole row in general to see the code in the Analysis Brower on the right, with the line causing this cover to be falsified highlighted in red. We can also see the tasks under which these covers are generated, which engines were used for the analysis, and how long it took those engines to complete the analysis.



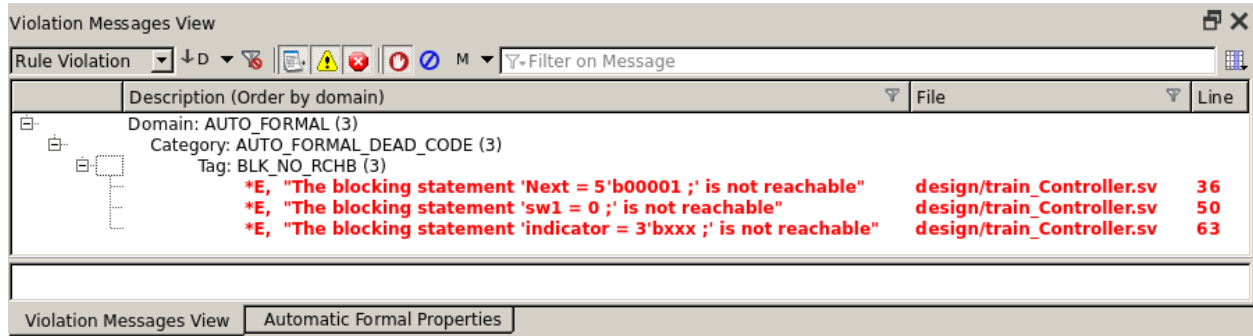


Figure 18. Violation messages window

In the bottom part of the screen, we also have the Violation messages view, which gives us more insight on the issues causing this specific cover to be falsified. In our case here we have 3 violation messages. There's also a reference to the sv file and line where the issue is found. This is essential for debugging and fixing the code.

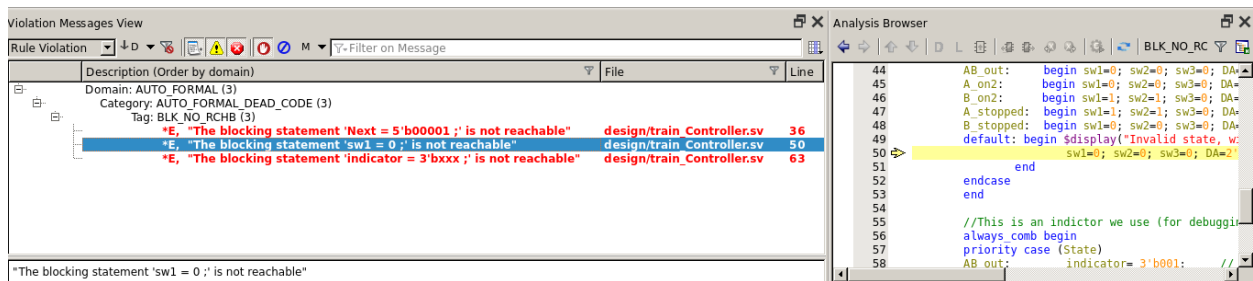


Figure 19. Violation messages window with reference to the code line

Once the issues are fixed, we can rerun the analysis again to confirm that we are getting any covers, or assertions falsified. Please note that sometimes the falsified properties may not be very meaningful to our design. Remember, these properties are all generated automatically. For example, some falsified covers may not be truly errors or issues in the design.