

# **Synopsys<sup>®</sup> VC Formal Tutorial**

## **Formal Security Verification (FSV)**

**Version 1.1 | 19-May-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contents of this document are confidential, privileged, and only for the information of the intended recipient and may not be published or redistributed.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 FSV .....	4
Design Files .....	5
TCL File .....	6
Application Setup .....	7
Invoking VC Formal GUI .....	8
User Interface Details .....	8
Loading TCL File .....	9
Invoking VC Formal Along with TCL File .....	11
Running Files .....	12
Detecting Errors .....	13
Source Tracing .....	14
Debugging .....	15
Appendix .....	19
<i>Table 1.1. FSV App Functions/commands</i> .....	20

## Introduction

VC Formal uses TCL (Tool Command Language) scripts to tell it what to do. The script can define the app within VCFormal that we want to use, the files we are working on, how we are mapping them, the clock cycles, and more. Instead of diving into the details of TCL scripts, we will use templates. One would then simply need to modify those templates to interact with VC Formal as needed in their project.

To begin, you need to set up the VNCserver as shown in the previous tutorials and open the Linux GUI. For better practices and to keep everything organized, we create a main folder for the design we want to analyze, and in this case, I called mine “FSV\_Example”. Don’t use spaces when naming the files and folders.

Inside that folder, we create two folders: one is Design, where you put the actual Verilog or SystemVerilog designs we want to analyze, and the other is Run, where we put the TCL that will run the VC Formal FSV analysis for us.

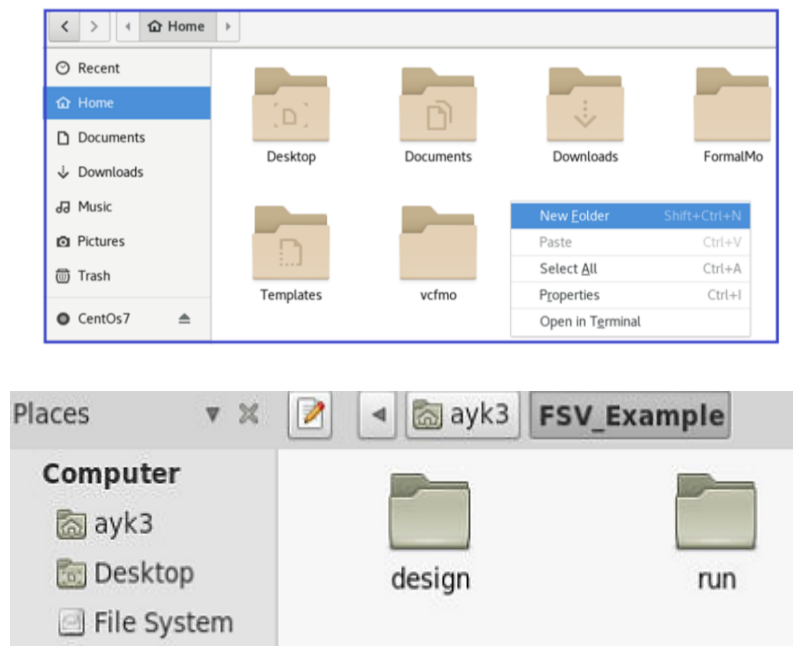
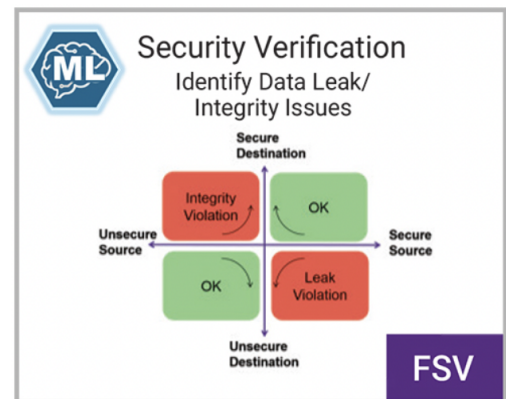


Figure 1. Creating folders to organize design and TCL files.

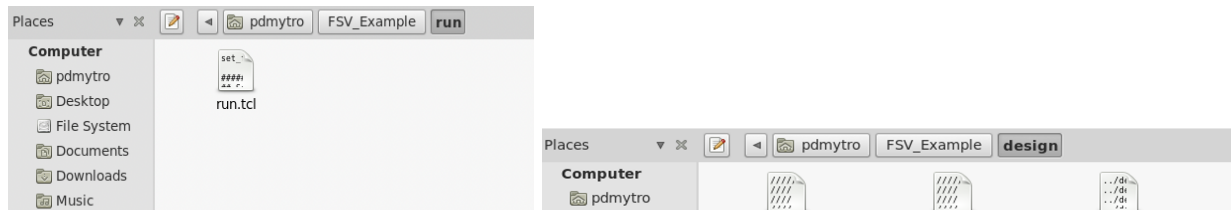
## About and Usage of FSV

The “FSV” app in VC Formal is used as a verification application to ensure no illegal data propagations are in your design. VC Formal FSV app streamlines the verification process, ensuring the completeness and correctness of designs. By leveraging formal verification techniques, this application empowers designers to identify and resolve complex functional issues efficiently, minimizing the risk of bugs and improving overall design quality.

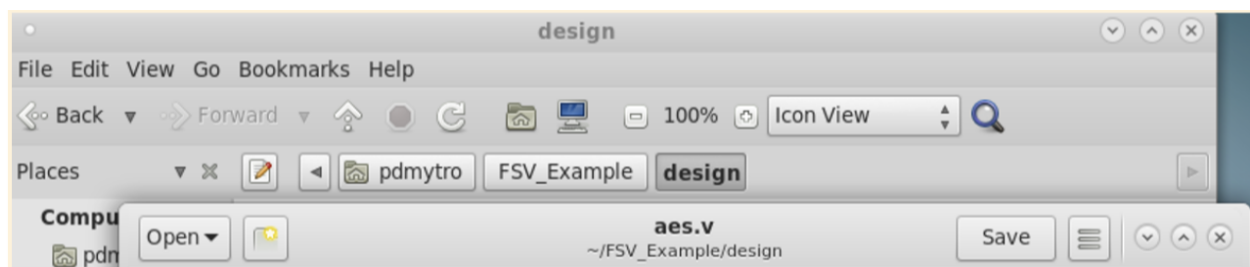


## Design Files

In the Design folder, you'll put in the SystemVerilog design file. The TCL file will be put in the Run folder. We suggest you name your folders with lowercase letters, as VC Formal is case-sensitive.



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

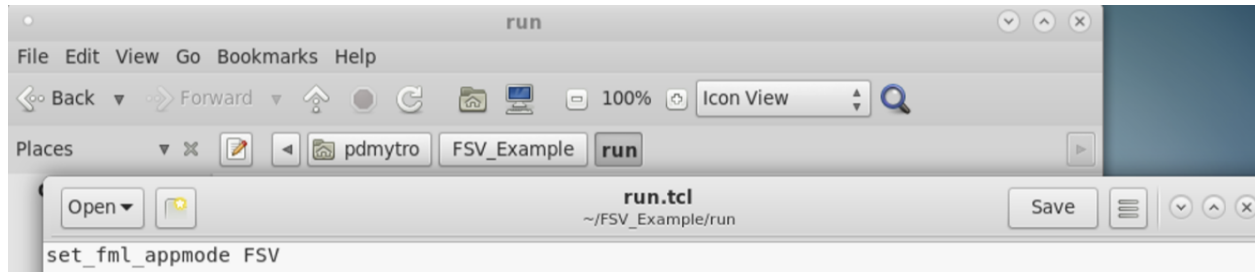


*Figure 3. Example of the design we are using in this tutorial.*

In order for VC Formal to map your design, you need to acknowledge and keep note of your exact module name, as highlighted in *Figure 3* above. This will come in handy when you create your TCL file. To make things simple, we suggest naming your design file after its corresponding module name (more on this in the TCL File section below).

## TCL File

Next, we will set up the TCL file. Below is the TCL file (*Figure 4*), which you can use as a template for your functional checks on VC Formal.



*Figure 4. Annotated TCL template file.*

- (1) Instruction that sets the appmode to FSV in VC Formal.
- (2) Name of the main module as established in the Design file.
- (3) The property type identifier switch name for desired FSV analysis.
- (4) Design file location so VC Formal knows where to find the file.

As mentioned before, VC Formal is case sensitive, and in order for it to map your designs, you will need to use the same module name in the TCL script **(2)** and the module name in the design file. For example, we can see that our module name in the TCL script is the same as the module name in the design file in *Figure 3*.

The file name (aes.v) can be named however you want.

## Application Setup

There are multiple ways to invoke the VC Formal 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 within the appropriate “Run” folder associated with the AEP app.

- [Invoke VC Formal GUI](#), then manually load the TCL script in the application:

```
$vcf -gui
```

**OR**

- [Invoke VC Formal GUI and TCL script](#) in one command:

```
$vcf -f run.tcl -gui    or    $vcf -f run.tcl -verdi
```

‘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.

The “-gui” switch opens VC Formal in the GUI, and it’s equivalent to the switch “-verdi”.

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



## Invoking VC Formal GUI

To proceed with invoking VC Formal:

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

```
vcf -gui
```

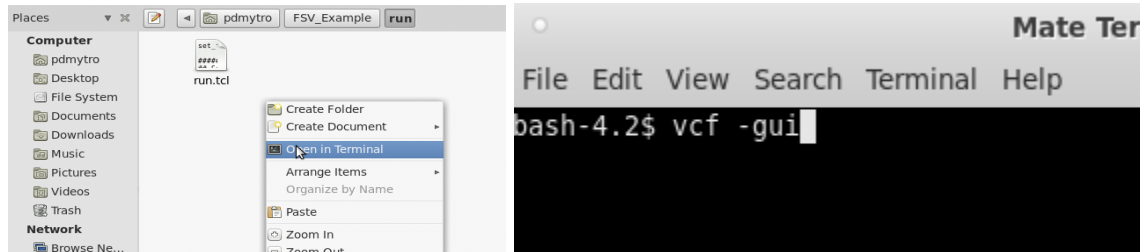



Figure 5. Invoking VC Formal in the terminal.

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

You should then see the VC Formal GUI as shown in *Figure 6* below. You can toggle between showing Targets, Constraints, or both Targets and Constraints window by clicking the blue box icon in the upper right-hand of the application **(1)**.

It may look like any of these three icons: 

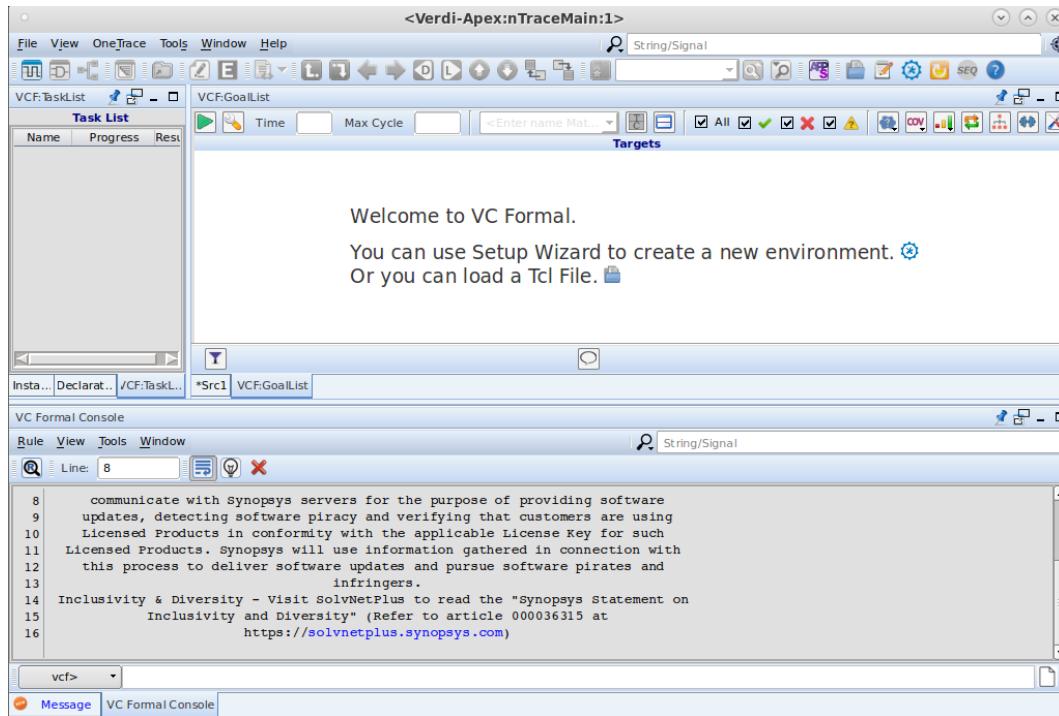



Figure 6. VC Formal GUI introductory screen.

Then load a TCL script by clicking on the  icon **(2)** as shown in Figure 6.

Next, select the “run.tcl” file we have in the “run” folder:

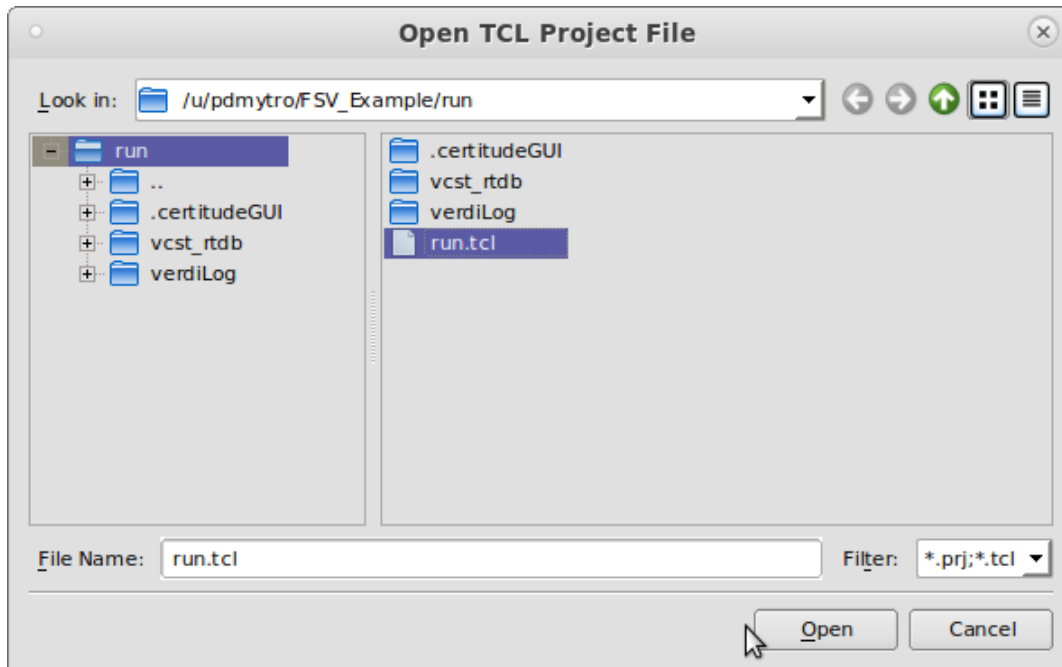


Figure 7. Selecting TCL file.

## Invoking VC Formal Along with TCL File:

To proceed with invoking VC Formal:

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

```
vcf -f run.tcl -gui
```

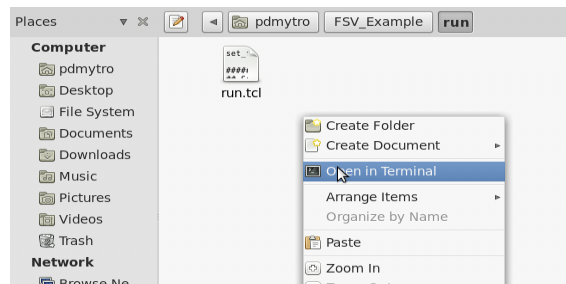


Figure 8. Opening terminal in the ‘run’ folder.

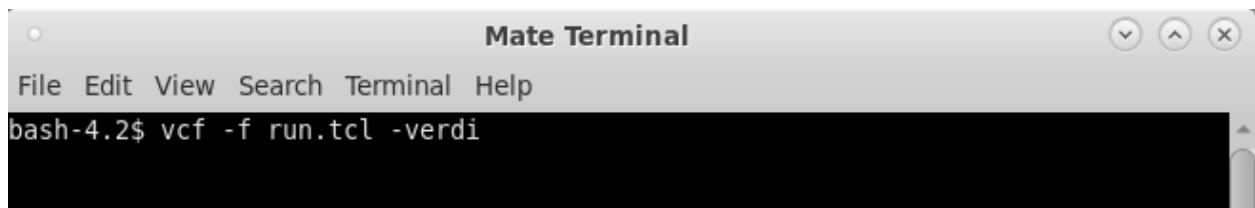


Figure 9. Invoking VC Formal and TCL script in the terminal.

And that’s it!

## Running Files

After successfully invoking VC Formal GUI and loading the TCL script in the above methods, your screen should have contents in the *VCF:GoalList* tab and look something like this:

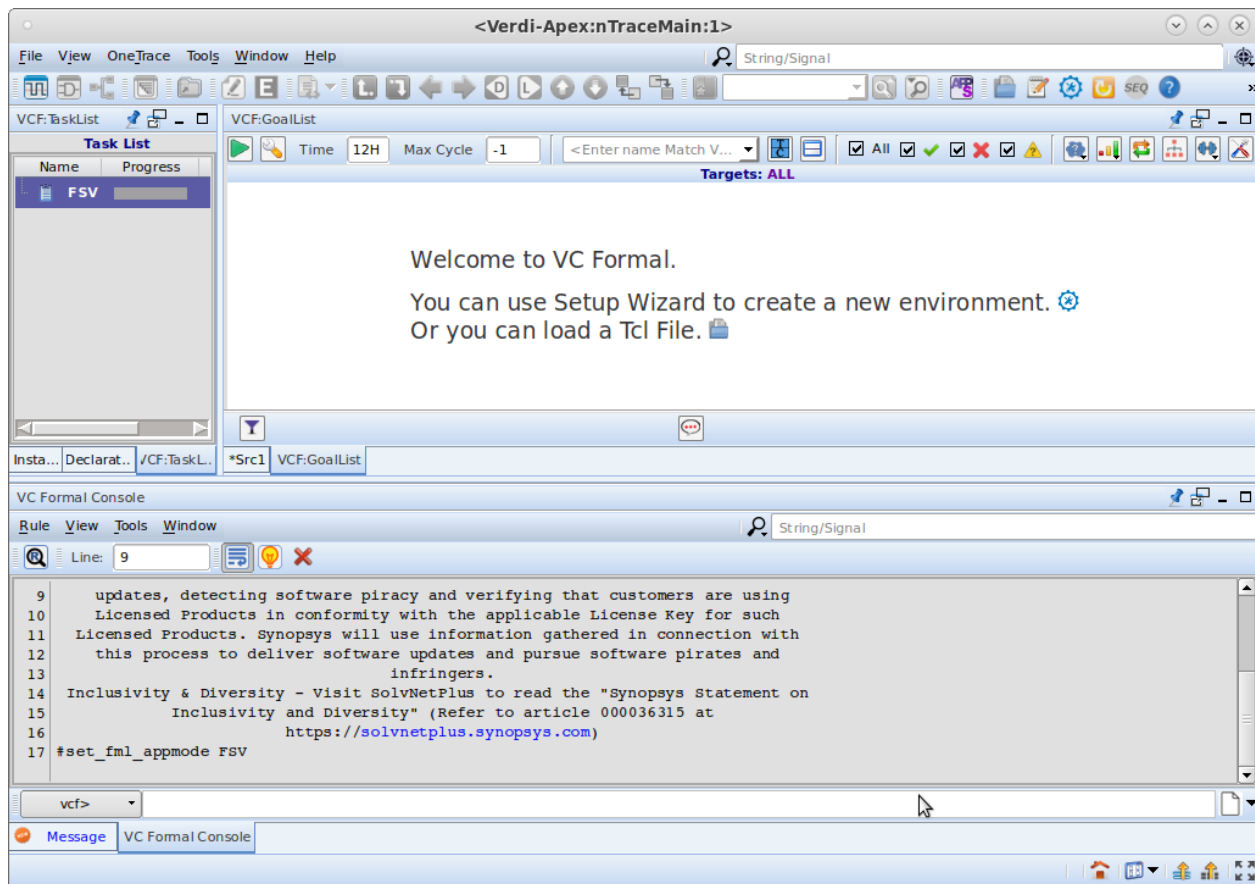



Figure 10. Screen after loading TCL script.

Now, go ahead and run the verification analysis by clicking on the play  icon in the upper left corner of the *VCF:GoalList* tab.

## Detecting Errors

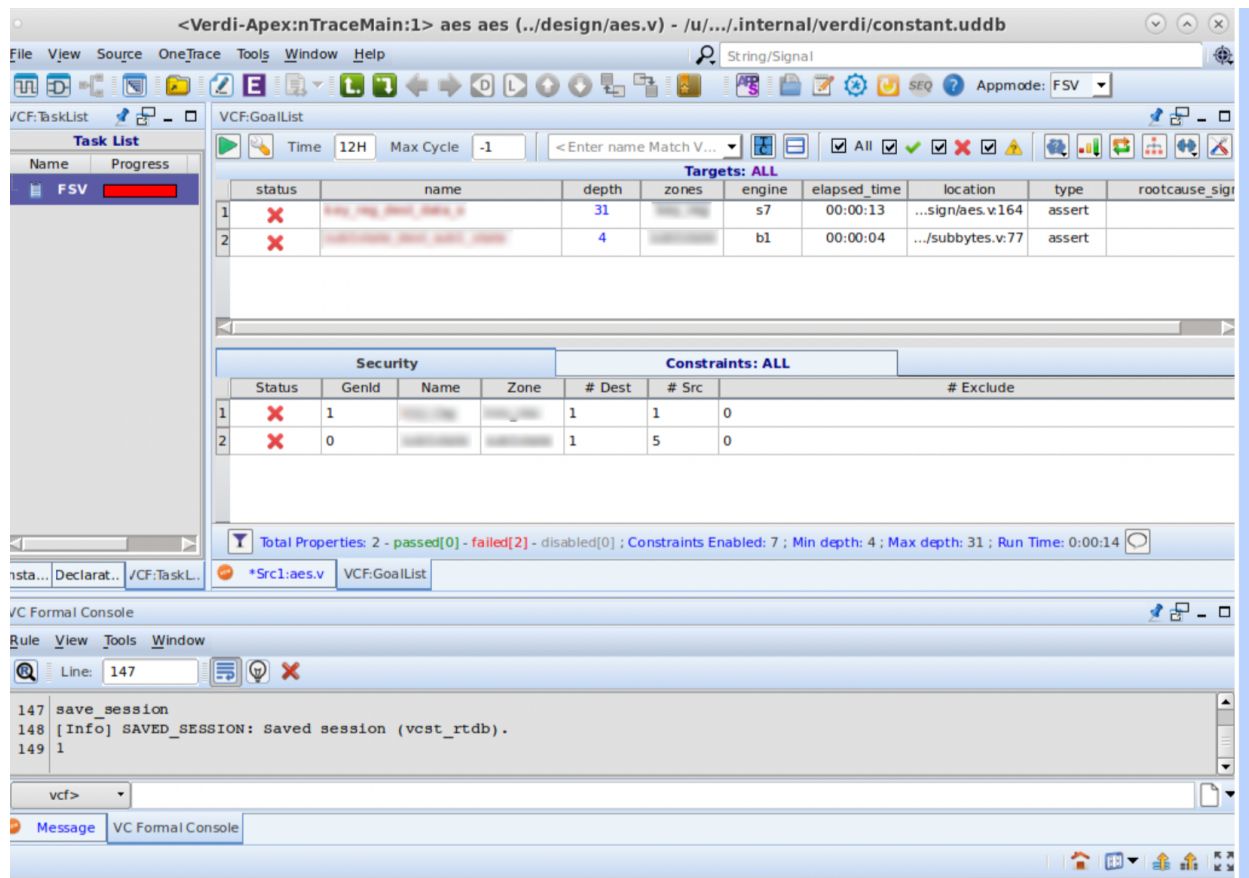


Figure 11. Screen after running TCL script and the status given = ✖.

We see two ✖ icons; indicating that there are security properties that are falsified. Debugging falsified properties can help determine which input and outputs are involved.

On the left, under *Task List*, we can see that VC Formal was given one task by the FSV app. You can hover over the numbers under *Result* to see the results in detail. You may need to alter the tab size by dragging the side panels or scrolling to the left:

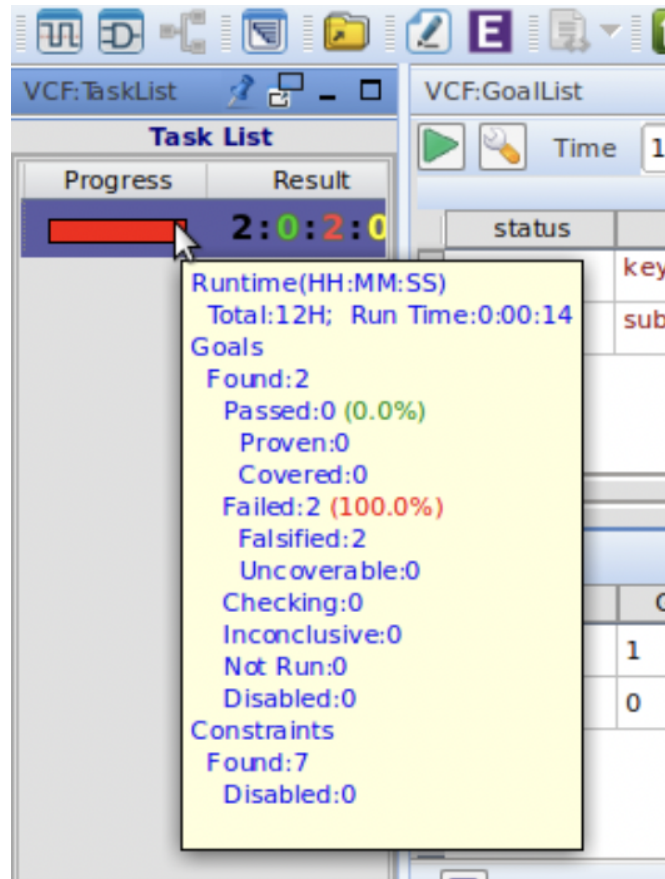




Figure 12. Results of the analyzed script.

## Debugging Failures

Source tracing is a great way to determine where exactly our errors lie and to get a deeper look into what the issue is. To start, go ahead and double-click on an  icon.

We are going to right-click on the first  (line 1) from *Figure 11*. Select View Trace and then Property.

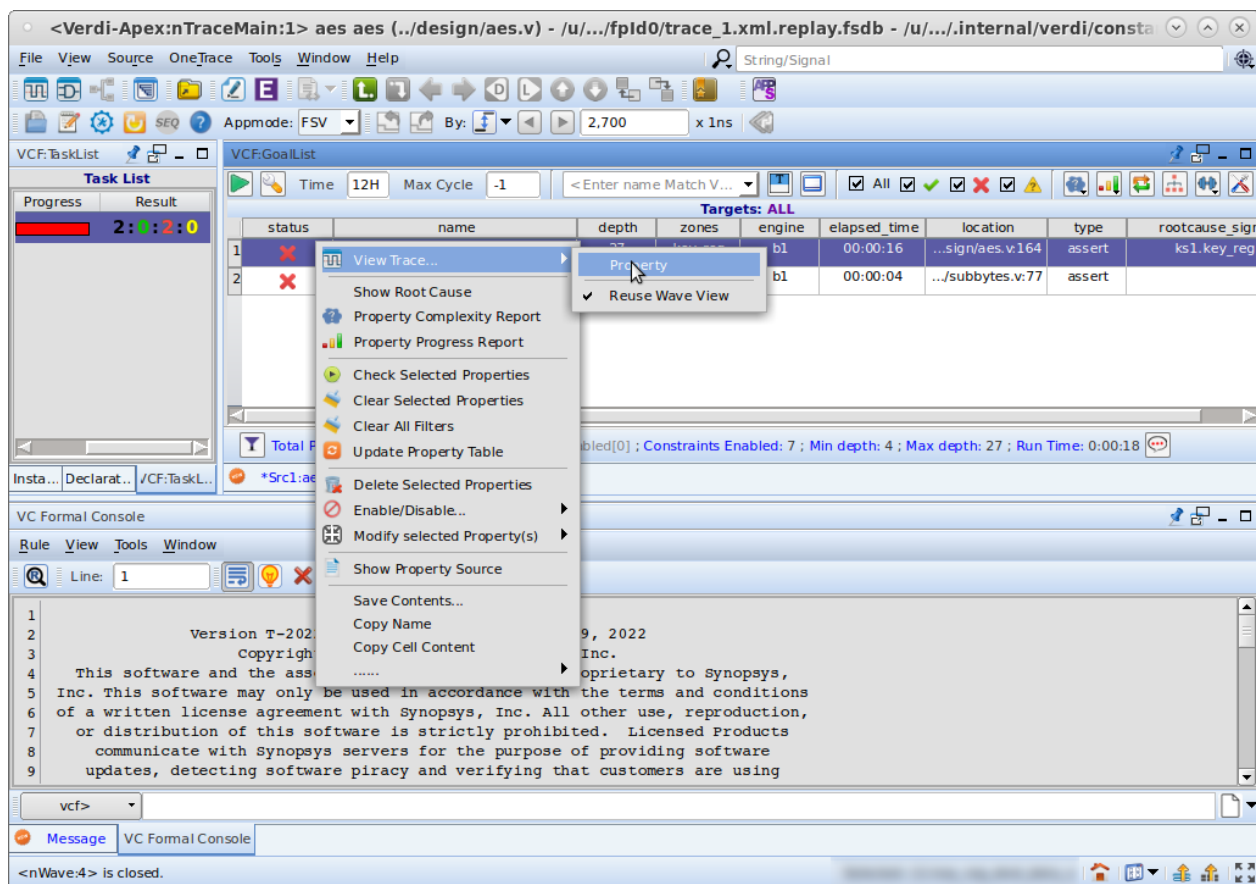


Figure 13. Examining the falsified security properties in our design.

You should then see a generated counter-example waveform as shown below from Figure 14:



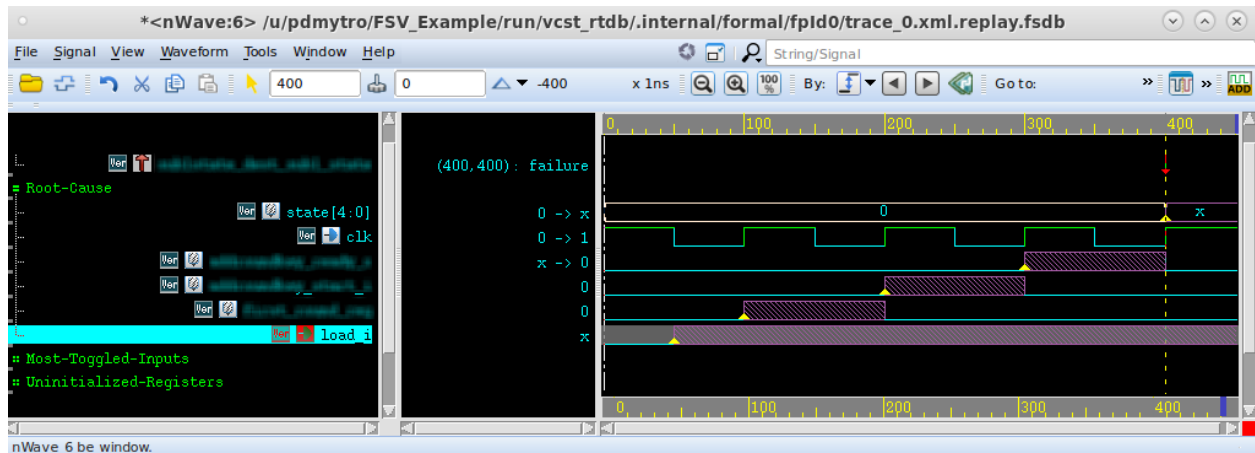


Figure 14. Examining the waveform

In Figure 14 on the left side we see all of the registers in the propagation path from the source to the destination.

As seen above the waveform shows that the given input "load\_i" can be affecting the value of the register state which is a potential security issue within the DUT.

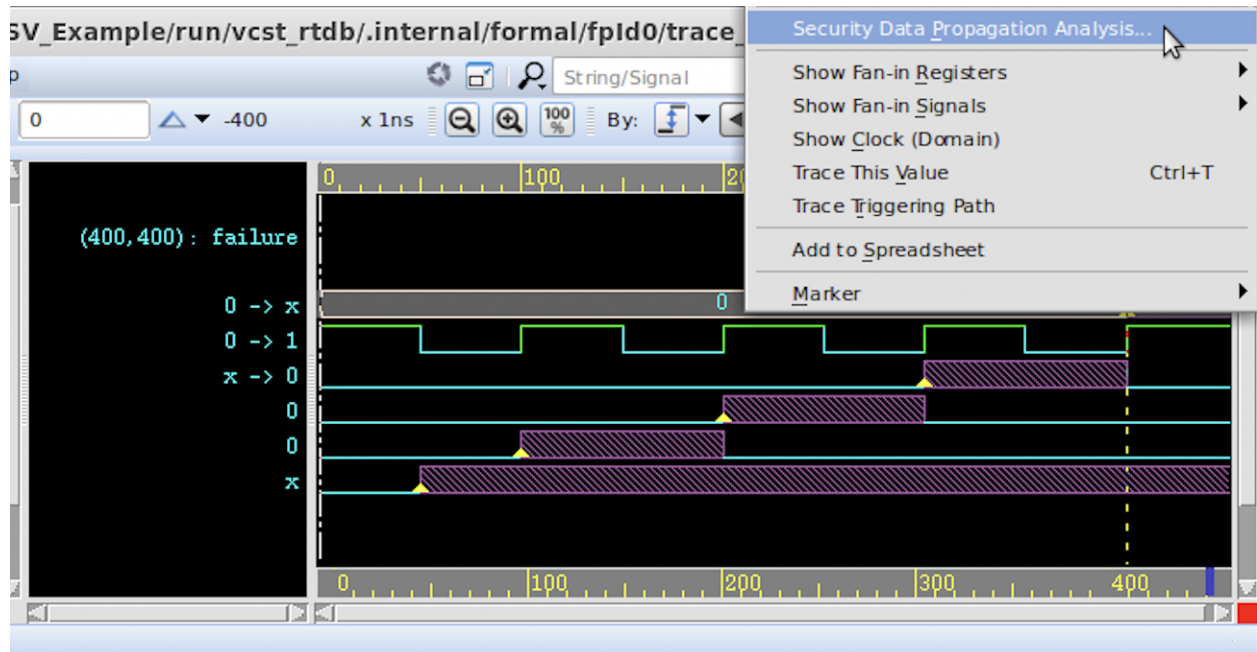


Figure 15. Further examining the propagation between the input and the secure register

For further analysis right click on the location of the failure and select “Security Data Propagation Analysis”. (see Figure 15 above)

This will highlight how the propagation happens between the input and the secure register. (see Figure 16 below)

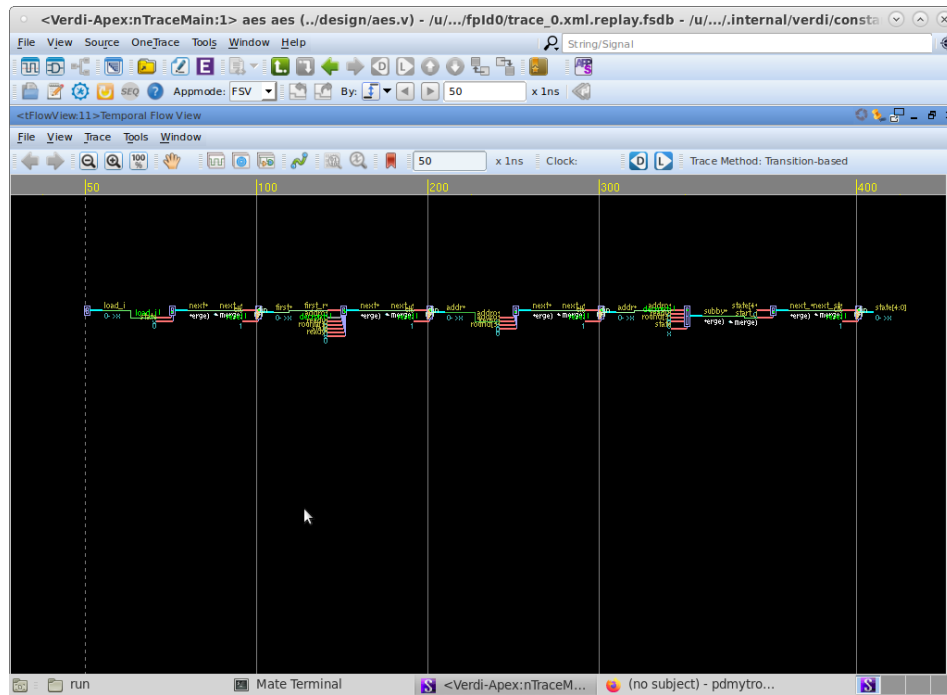


Figure 16. Data propagation of the source & destination of the security in temporal flow view

Repeat the debugging step for each falsified property.

## Appendix

---

Function	Code	Description
Set App mode	set_fml_appmode FSV	VC Formal App mode command for FSV
Format	-format <format>	The format of the file to be loaded. The two formats are csv or table. The format option is optional, and default is csv.
Results Summary	-quiet	Option to hide the progress message and summary results.
CSV File Path	<filename>	Path of the properly formatted csv file to read.
Formal Compile	formal_compile	This command is used to compile the design and the formal properties specified in the input files, generating a formal model for verification.
Formal Verify	formal_verify	Use this command to initiate the formal verification process. It checks the specified properties against the formal model generated during compilation.
Formal Debug	formal_debug	Use this command to initiate the formal debugging process. It provides interactive debugging capabilities to help analyze and resolve issues found during verification.
Report of Warnings	-warning	Specify this option to get a report of warnings. You can view the warnings file-wise. Appending the -warning switch with -list and -verbose, displays all warnings from the list and verbose.
Summary OFF State	-no_summary	Specify this option if you do not want to see the summary information for any other switch.
Detailed Report Information	-verbose	Displays the summary of results and then provides detailed information about the report such as the status, connection name, line number and message of each property file-wise.
Save initial state	sim_save_reset	Saves initial state
Check setup	check_fv_setup	Checks setup for errors

*Table 1.1. FSV App important functions and commands*