

VC Formal Lab

Formal Security Verification (FSV) App Setup and Standard Usage

Learning Objectives

In this VC Formal lab, you will use an AES encryption core from OpenCores to learn to do the following:

- Set up design
- Run interactively with Verdi GUI
- Set up clocks and resets
- Establish initial state for formal
- Write security properties for verification with FSV
- Run checks and report results
- Save session
- Debug failures



Lab Duration:
30 minutes

Familiarity and knowledge of basic formal verification concepts are required for this lab.

Files Location

All files for this VC Formal lab are in directory:
`$VC_STATIC_HOME/doc/vcst/examples/FSV`

Directory Structure	
FSV	Lab main directory
README_VCFormal_FSV.pdf	Lab instructions
design/	Verilog RTL code of the Device Under Test (DUT)
run/	Run directory
solution/	Solution directory

Resources

The following resources are available for in-depth guidance regarding VC Formal usage, commands, and variables.

VC Formal User Guide:

`$VC_STATIC_HOME/doc/vcst/VC_Forma Docs/VC_Forma_UG.pdf`

VC Formal Apps Quick References Guides:

`$VC_STATIC_HOME/doc/vcst/VC_Forma Docs/Quick_Reference_Guides/`

VC Formal Apps Tcl Templates:

`$VC_STATIC_HOME/doc/vcst/VC_Forma Docs/Quick_Reference_Guides/vcf_tcl_templates/`

Prepare your Environment

1. Set environment variable pointing to your VC Formal installation directory:

```
%setenv VC_STATIC_HOME /tools/synopsys/vcstatic
```

2. Add path \$VC_STATIC_HOME/bin to the PATH environment variable.
3. Change your working directory to FSV/run:

```
%cd FSV/run
```

Now you are ready to begin the lab.

Create a run.tcl Setup File

VC Formal has a Tcl-based command interface. It is common to start with a Tcl file to set up and compile a design. In this step, you will create a VC Formal Tcl file for the DUT, an AES core, used in this lab.

4. Open a new file run.tcl (any arbitrary name is ok to use) using an editor.

```
%vi run.tcl
```

5. Enable formal app with FSV mode to on:

```
set_fml_appmode FSV
```

6. Specify DUT top level module name as Tcl variable:

```
set design aes
```

7. Add command to compile DUT:

The DUT files and filelist are located under directory FSV/design.

```
read_file -top $design -format sverilog \  
-vcs {-f ../design/filelist +incdir+../design}
```

Note: To use unified usage model to compile design, use these commands instead of `read_file` to compile design and SVA properties:

```
analyze -format sverilog \  
-vcs {-f ../design/filelist +incdir+../design}  
elaborate $design
```

8. Save run.tcl file and exit editor.

Start VC Formal in Verdi GUI Mode

9. Start the tool in Verdi GUI mode:


```
%vcf -f run.tcl -verdi
```

VC Formal starts in the Verdi GUI mode, with icons, tables, tabs, and windows especially designed for property verification with the FSV App. Look at the top menu bar to check that you are in FSV App mode.

Initial configuration is shown with the “VCF:TaskList” tab on the top left, the “VCF:GoalList” tab on the top right, and the “VC Formal Console” shell at the bottom.

Review Setup and Design Information

10. Review setup and design information:

Click on the Show Complexity icon  on the upper right above the property table. Examine items under “Missing Clock” and “Missing Reset” and trace from the source file to find out the active phase of the reset signal.

Revise Setup and Review Initial State

11. Add the missing clock and reset setup information to the Tcl file:

Click on the Edit Tcl Project File icon  on the upper left and add the following commands to the Tcl file.

```
create_clock clk -period 100
create_reset reset -sense low
```

12. Add design initialization commands:

```
sim_run -stable
sim_save_reset
```

These commands initialize the DUT by holding reset active until sequential elements (latches and flip flops) values are stable.

Generate Security Properties for Secure Registers

The DUT contains registers that are considered secure. FSV properties are generated by specifying source and destination signals to be validated as not interacting with each other e.g., secure signal with non-secure signal.

Example:

```
fsv_generate -name property_name \
  -src source_signal_name -dest dest_signal_name
```

13. Generate security property for secure register “state” in module “subbytes”:

```
fsv_generate -name sub1state \
  -src [get_attribute [get_ports - word \
  -filter {direction==in}] full_name] \
  -dest sub1.state
```

This security property checks that input ports of the DUT cannot modify secure register “state” in module “subbytes” (instance “sub1”). Note that while there are multiple input ports, a single security property is generated.

14. Generate security property for secure register “key_reg” in module “keysched”:

```
fsv_generate -name key_reg \
  -src aes.ks1.key_reg -dest data_o
```

This security property checks that secure register “key_reg” in module “keysched” (instance “aes.ks1”) cannot reach output port “data_o” of the DUT.

15. Check the properties that are generated using Tcl command `fsv_report`:

```
fsv_report -list
```

16. Save edited run.tcl Tcl file:

Click on the Save icon .


17. Restart VC Formal:

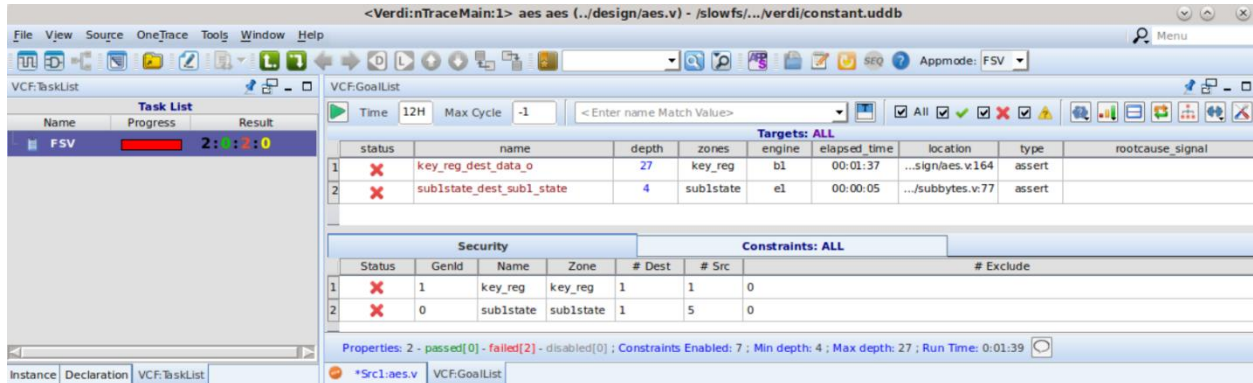
Click on the Restart VCST icon .

Run FSV Formal Proofs and Review Results

Now that you have a correct setup, check the target security properties with FSV once the DUT initialization completes.

18. Start property verification:

Click on the Start Check icon .



Observe that the two security properties are falsified and marked with **✗**. Debugging falsified security properties helps determine which inputs and outputs are involved.

19. When check completes, report results:

```
vcf> report_fv -list
```

Save Session

20. Save session using default name from the VC Formal Shell:

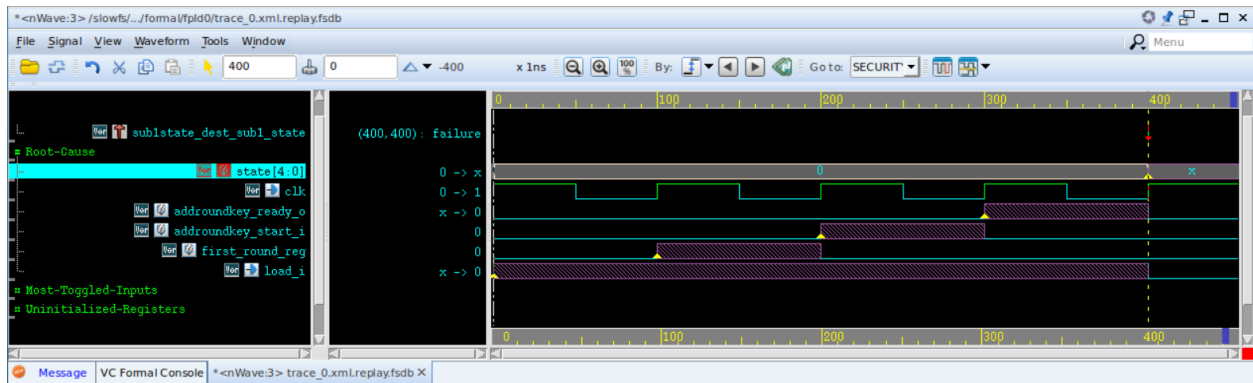
```
vcf> save_session
```

Alternatively, click on File → Save Session... to open the “Save Session” dialog window.

Debug Failures

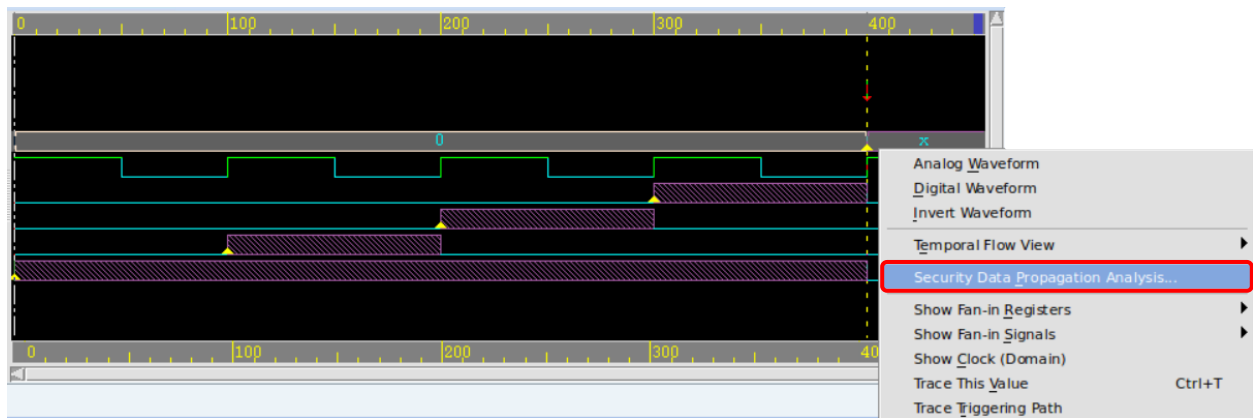
21. Debug failure for secure register “state”:

Double click on **✗** under the status of property “sub1state_dest_sub1_state” to open a counter-example waveform.
(Note that a double click on the “name” of a property will open the source code window instead.)

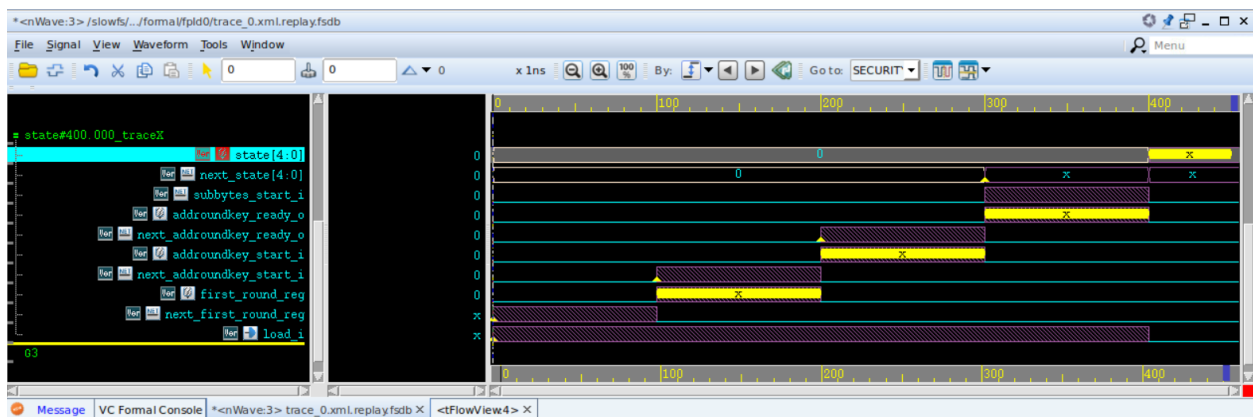


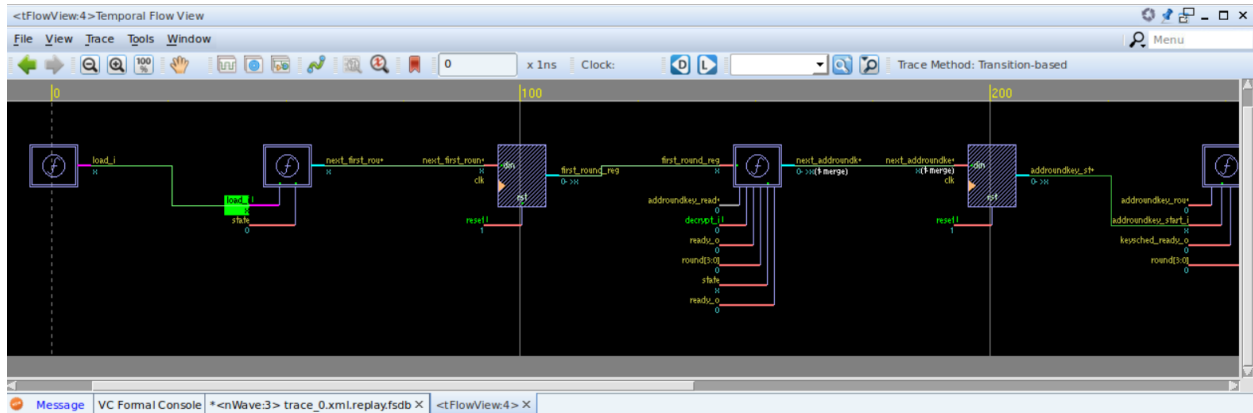
The waveform shows that input “load_i” can affect the value of secure register “state” thus identifying a potential security issue in the DUT.

Right click on the location of the failure and select “Security Data Propagation Analysis” to highlight how the propagation happens between input “load_i” and secure register “state”.



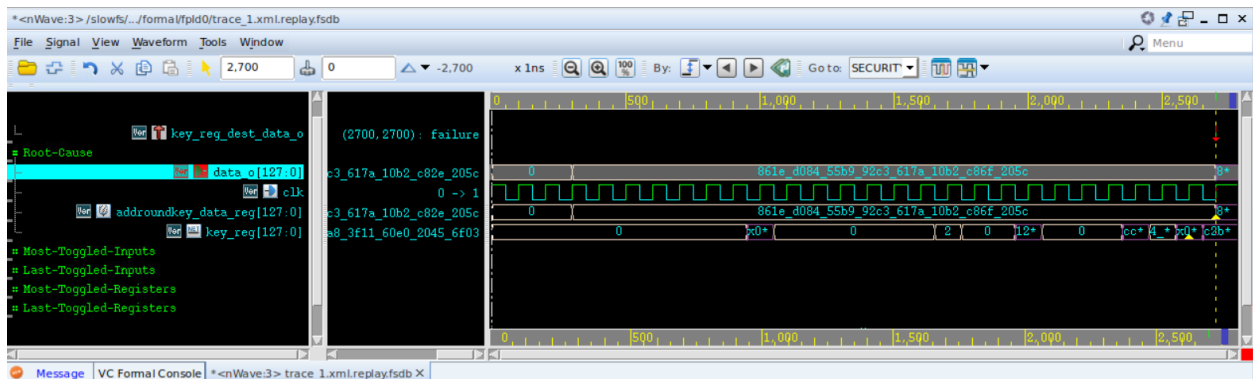
The data propagation between source and destination of the security property is shown in both the waveform view and temporal flow view.





22. Debug failure for secure register “key_reg”:

Double click on  under the status of property “key_reg_dest_data_o” to open a counter-example waveform.



The waveform shows that secure register “key_reg” can affect the value of output “data_o” thus identifying a potential security issue where secure information propagates to an output in the DUT.

Use “Security Data Propagation Analysis” in this waveform to identify how the propagation to output “data_o” happens.