Synopsys® VC Formal Tutorial

Formal Coverage Analyzer (FCA)

Version 1.0 | 27-Feb-2023

Portland State University

Disclaimer:

Formal Coverage Analyzing in Synopsys VC Formal FCA App Youtube Tutorial

Link:

The video is unlisted, and is not to be shared outside of this class.

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

1. 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 “FCA_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 VCFormal FCA analysis for us.

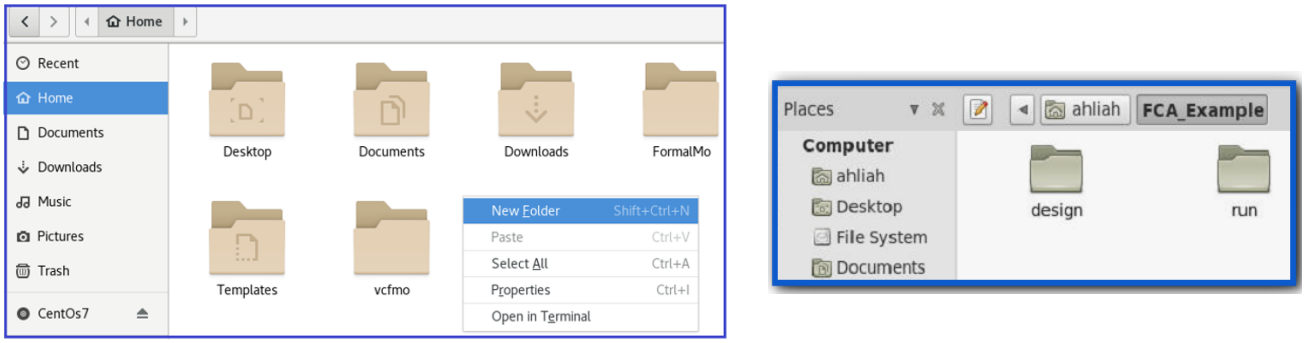
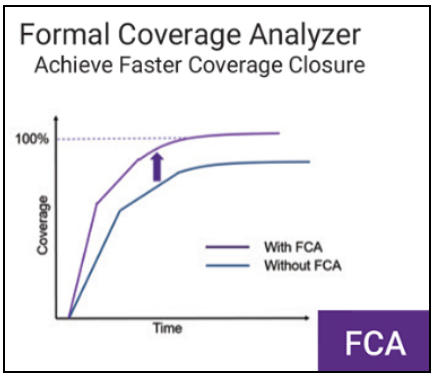


Figure 1.1. Creating folders to organize design and TCL files.

1.1 About and Usage of FCA

Code coverage closure is when a design reaches all of its target goals, and often is very difficult to achieve and time consuming, so coverage metrics are commonly used in simulation to measure progress and reachability analysis, such as providing information on what checks have passed and what has not. This provides proof on uncovered points in coverage goals that are indeed unreachable, allowing them to be removed from further analysis to save manual labor.



VC Formal's Formal Coverage Analyzer application provides:

- ➢ Assisted simulation-based verification coverage signoff
  - ○ Leverage the simulation coverage database
  - ○ Perform unreachability analysis on goals not covered by simulation
  - ○ Provide exclusion list for goals uncoverable for review and waive
- ➢ Formal property verification coverage signoff
  - ○ Over-constraints - unreachable goals due to constraints
  - ○ Property density - structural coverage of the design code in the COI of all properties
  - ○ Bound depth - identify whether the required design code is covered within specified proof depths
  - ○ Formal core - provides minimal design code that is required for the formal engines to reach the full proof or specific proof depths
  - ○ Fault coverages - provides design code covered by the Formal Testbench Analyzer (FTA) application

The full list of specific coverages and their description provided by the FCA application is listed in *Table 1.1.* in the Appendix and it is recommended you go through it to gain a better understanding of what this app truly provides.

1.2 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.

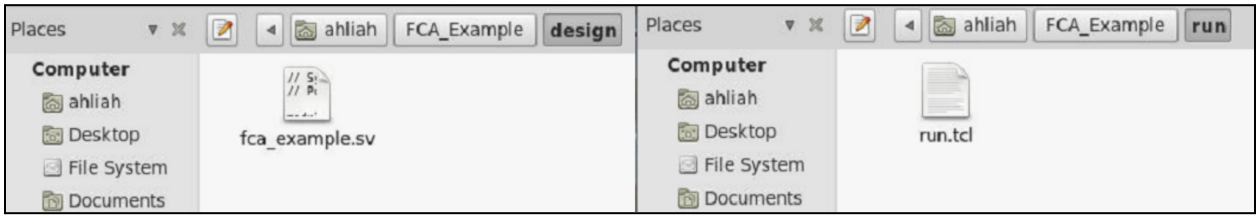


Figure 1.2. Showing the SystemVerilog and TCL files in the correct folder locations.

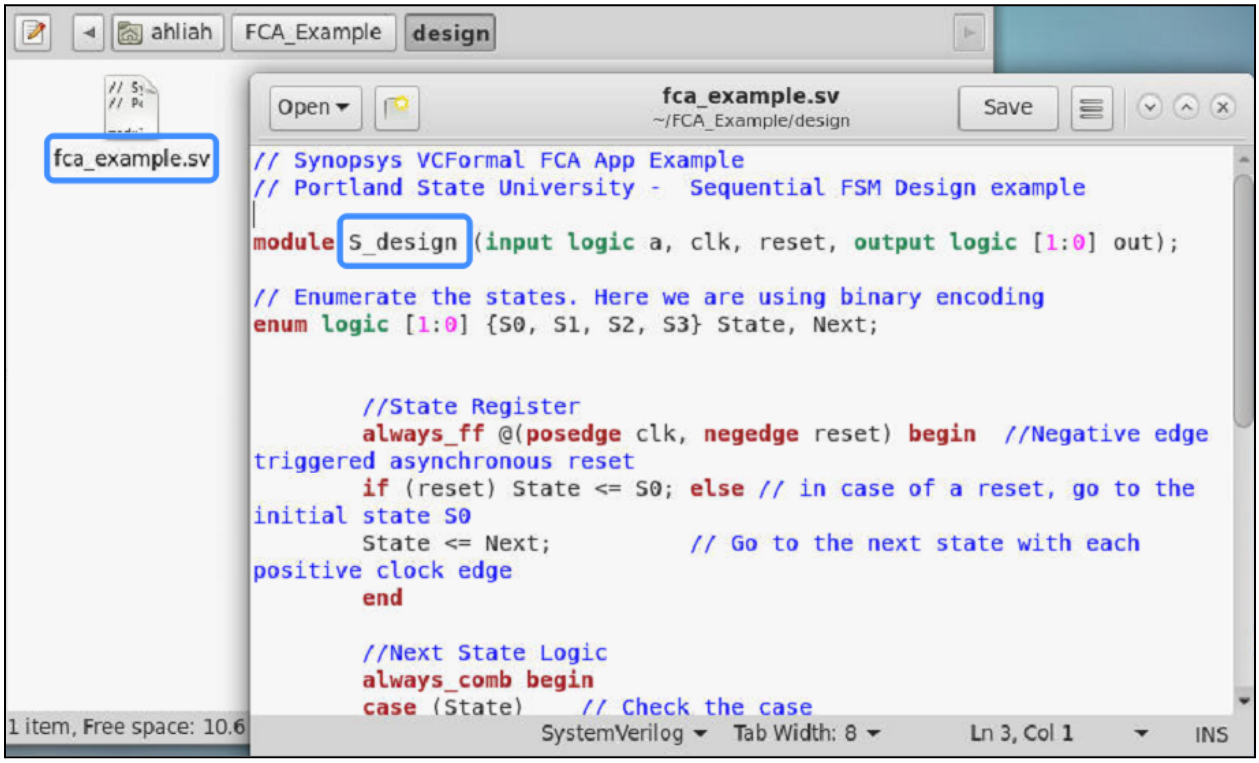


Figure 1.3. Example of the design we are using in this tutorial.

In order for VC Formal to map your design, you need to acknowledge and **keep note of your exact module name AND your file name,** as highlighted in *Figure 1.3* above. This will come in handy when you create your TCL file. Here

1.3 TCL File

Next, we will set up the TCL file. Below is the TCL file (full script in *Figure 1.5)*, which you can use as a template for your functional checks on VC Formal.

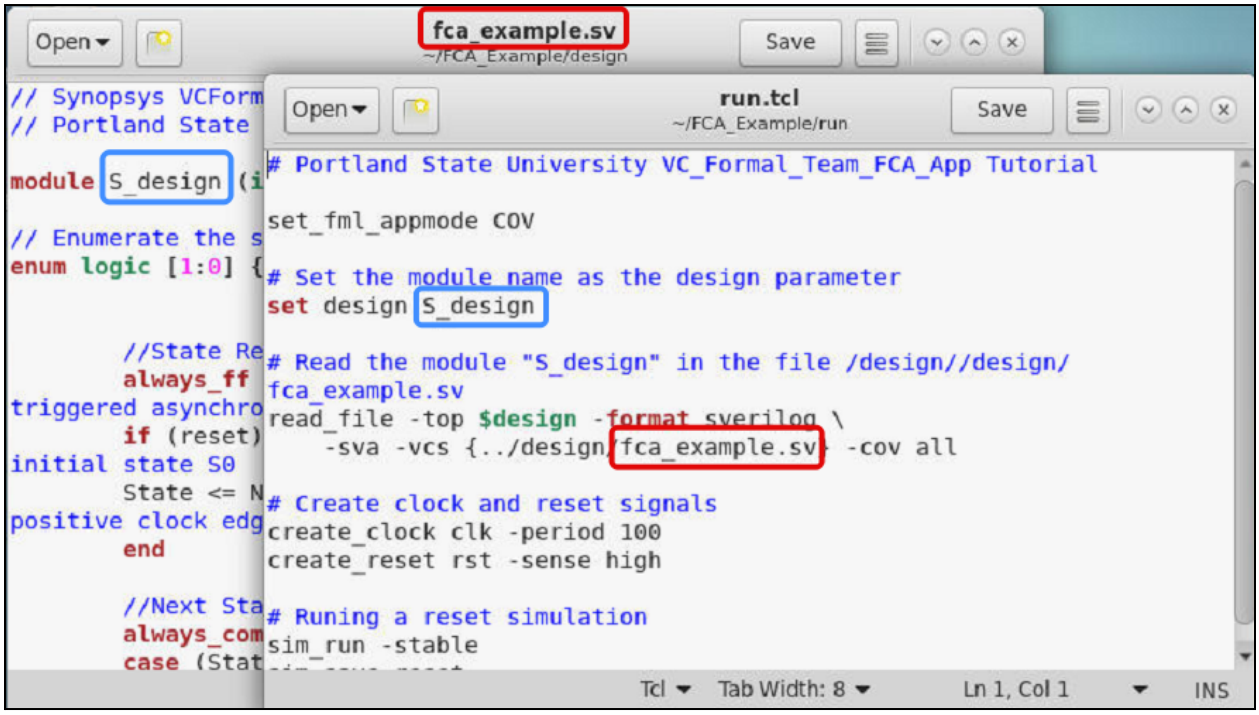


Figure 1.4. Difference between the design file name and module name.

In *Figure 1.4*, both the Design file and TCL files are shown side by side.

- The blue highlights are to demonstrate the **module name**.
- The red highlights are to demonstrate the **design file name**.

> These will **NOT** be the same for every user. You will need to keep track of your own module and design file names in order for your TCL file to work correctly when you try to run in VC Formal.

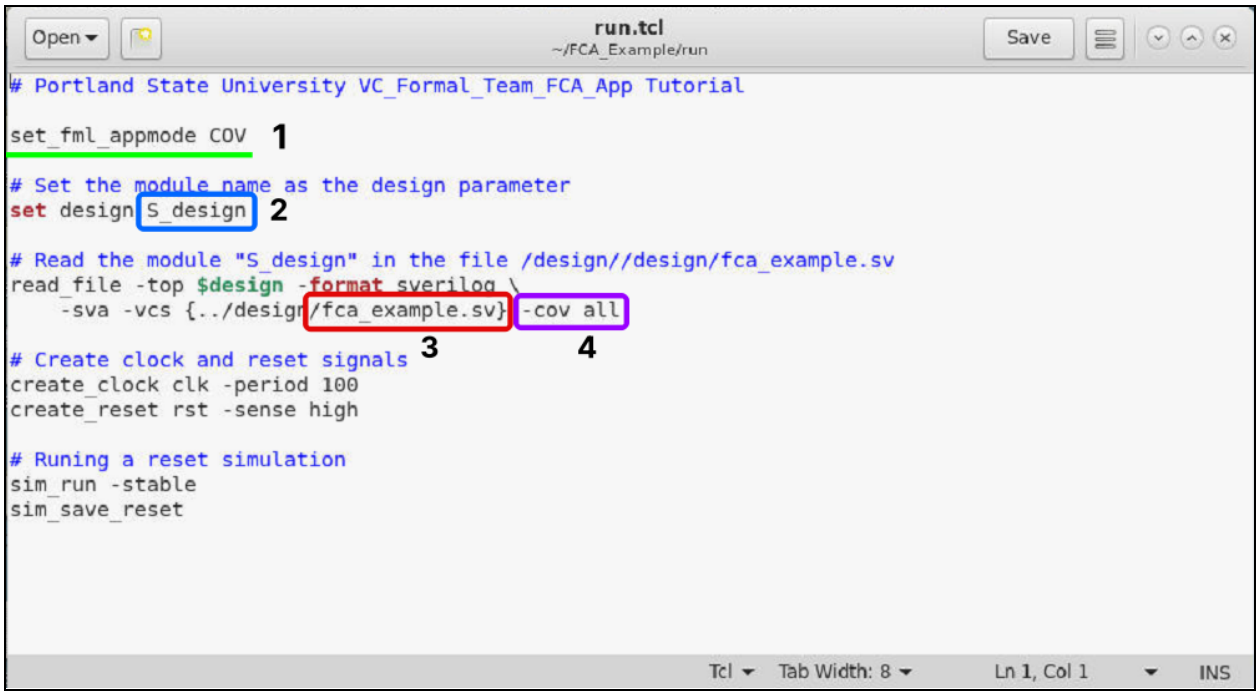


Figure 1.5. Annotated TCL template file.

(1) Instruction that sets the appmode to FCA in VC Formal.
(2) Name of the main module as established in the Design file.
(3) Design file location so VC Formal knows where to find the file.
(4) Identifier for FCA coverage metrics. Here we will enable all.

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 1.3*.

The file name (fca_example.sv) can be named however you want.

2. 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 FCA 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.

2.1 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

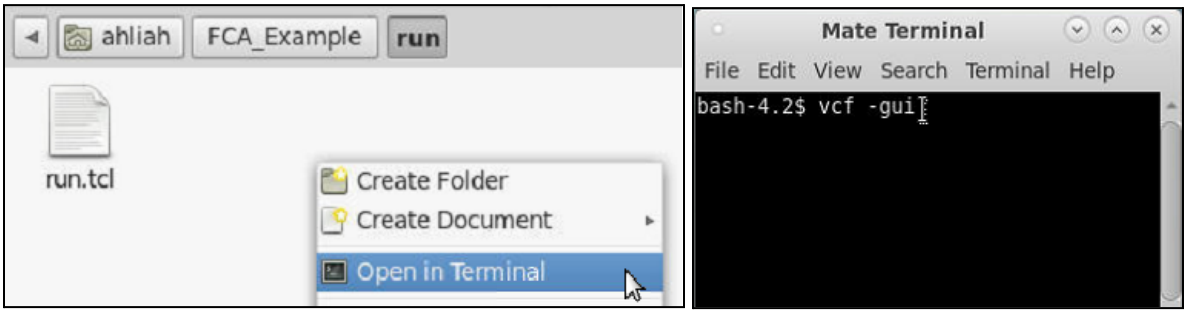


Figure 2.1. Invoking VC Formal in the terminal.

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 2.2* 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: 

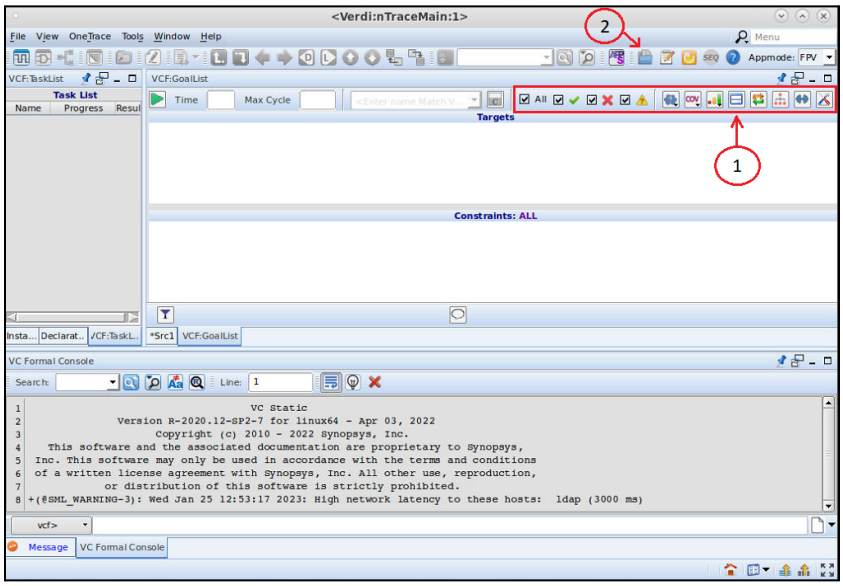


Figure 2.2. VC Formal GUI introductory screen.

Then load a TCL script by clicking on the  icon **(2)** as shown in *Figure 2.2*.

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

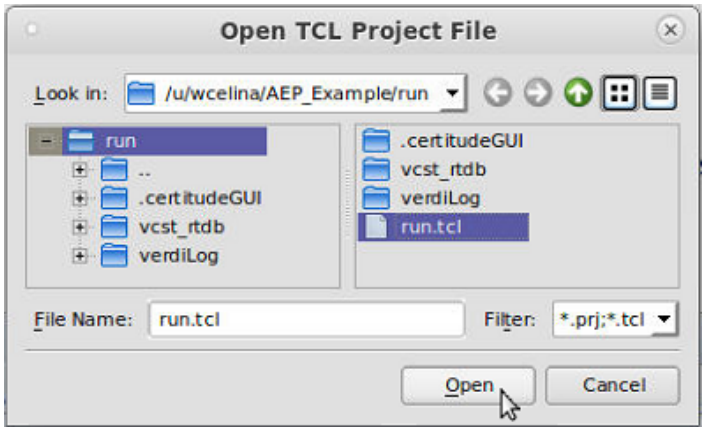


Figure 2.3. Selecting TCL file.

2.2 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
```

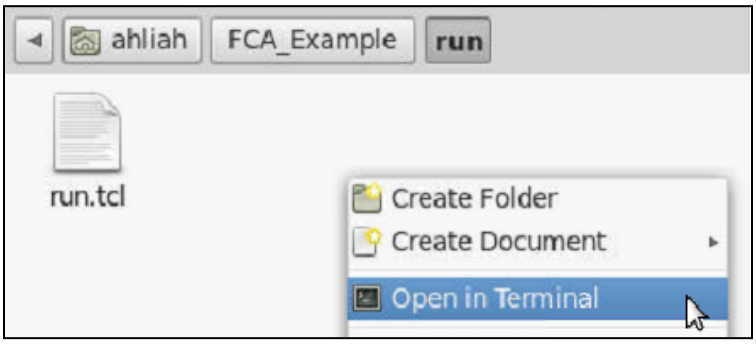


Figure 2.4. Opening terminal in the ‘run’ folder.

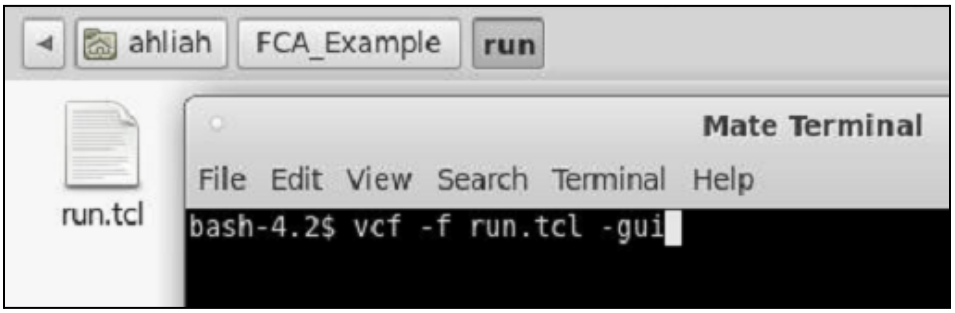


Figure 2.5. Invoking VC Formal and TCL script in the terminal.

And that’s it!

3 Application Usage

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:

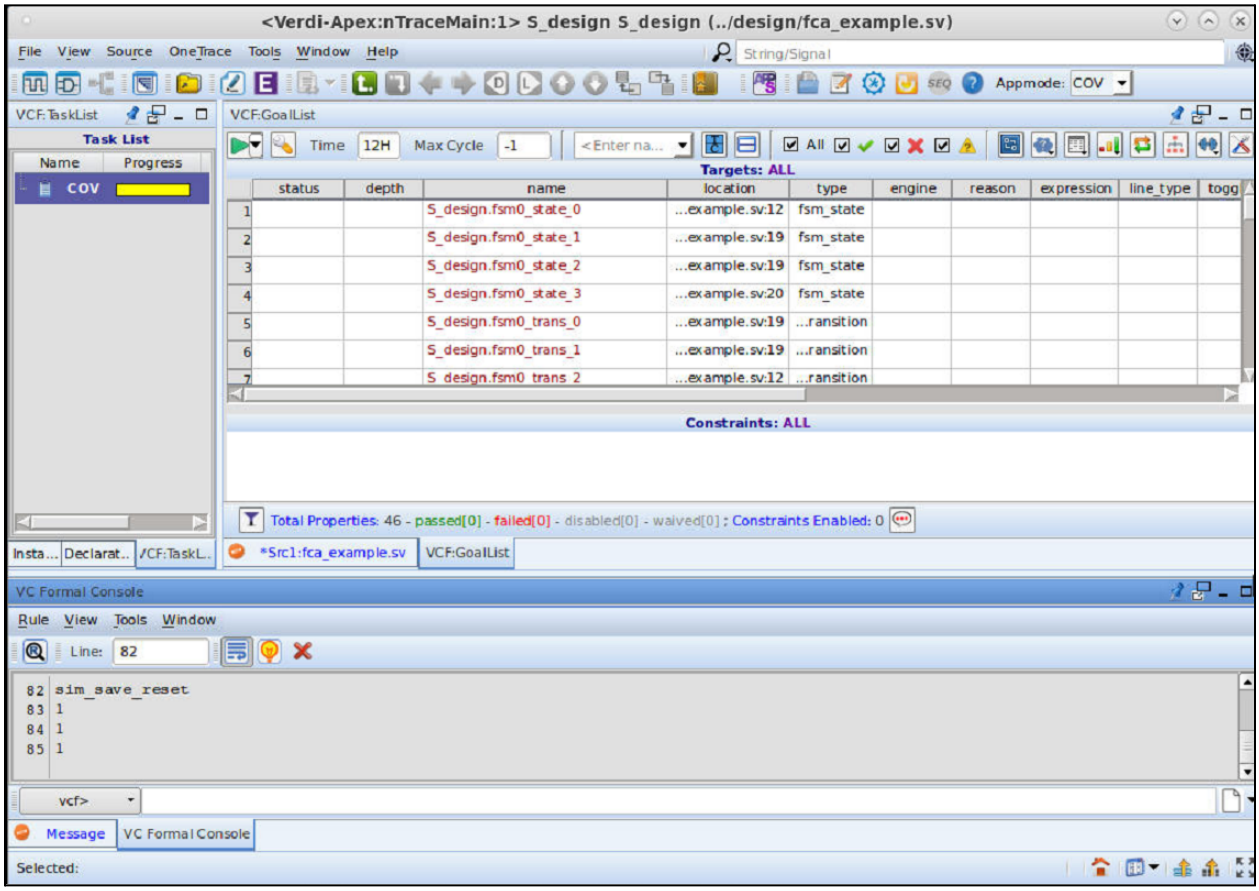


Figure 3.1. Screen after loading TCL script.

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

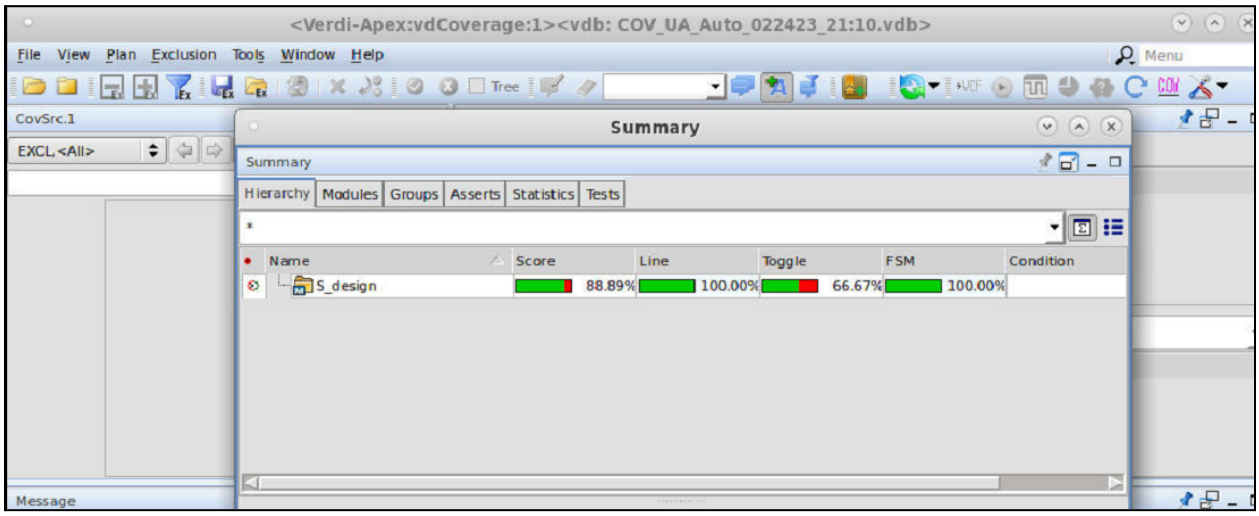


Figure 3.2: Coverage window pop-up after clicking play button.

When you run your design, a separate window will appear. Here we want to take notice to the Hierarchy tab within the Summary window. This will tell us our FCA score for our design; in *Figure 3.2* above, we got 88.99% for our fca_example design file. Our score is made up of the following checks: Line, Toggle, and FSM.

For the rest of the tutorial, we will not be using this window so go ahead and minimize it. The following steps will take place in the main VC Formal window.

3.1 Detecting Errors

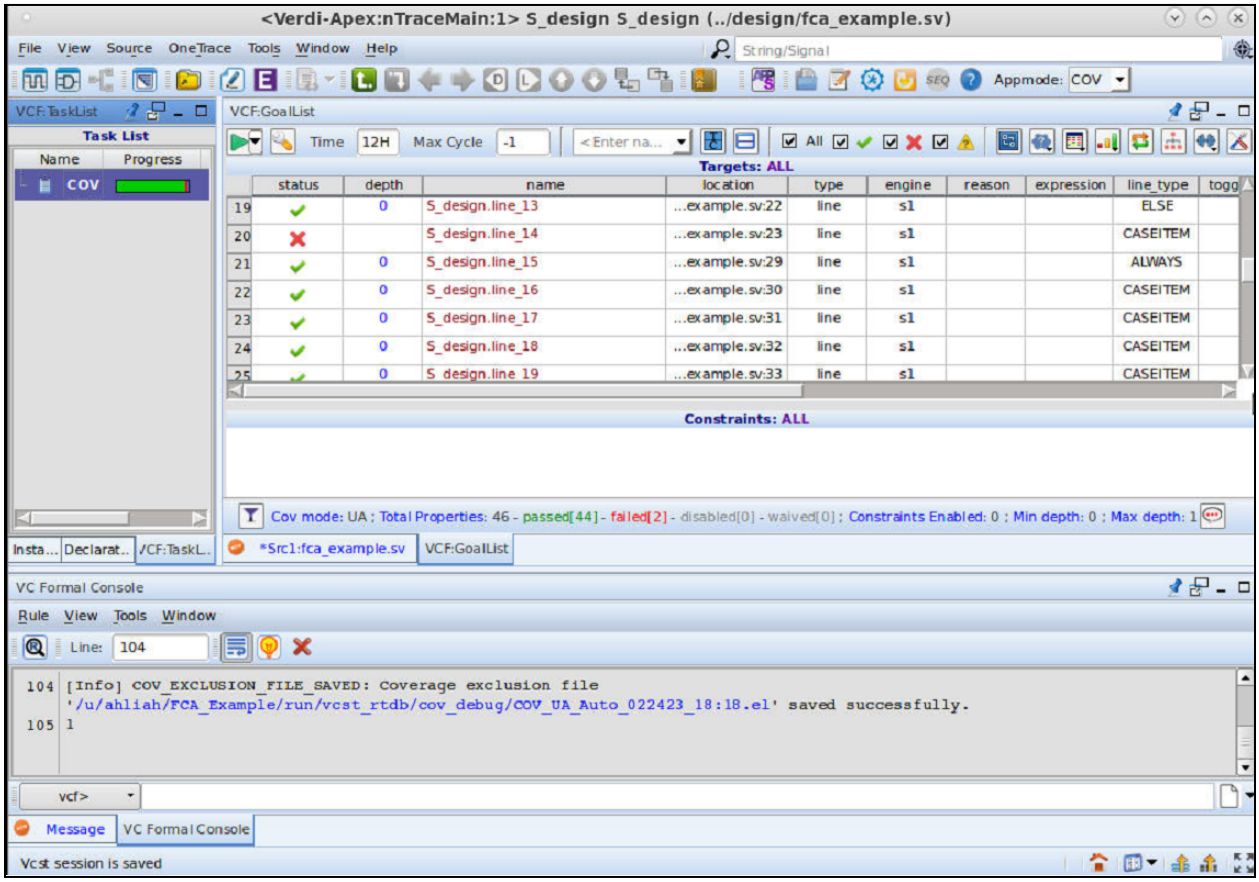


Figure 3.3. Screen after running TCL script and the status given = ✖.

In *Figure 3.3* above, we see multiple ✔ icons. VC Formal gives this “Passed” status icon for the part of our design that was “covered” by FCA.

We also see one ✖ icon; VC Formal gives this “Fail” status icon for the part of our design that was either “uncoverable” or “inconclusive”.

For example, in the figure above, the ✖ sign means that there is a CASEITEM that did not pass the “Line” coverage check on line 23 (see location column in Targets tab).

On the left under *Task List*, we can see that VC Formal was given one task by the FCA 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:

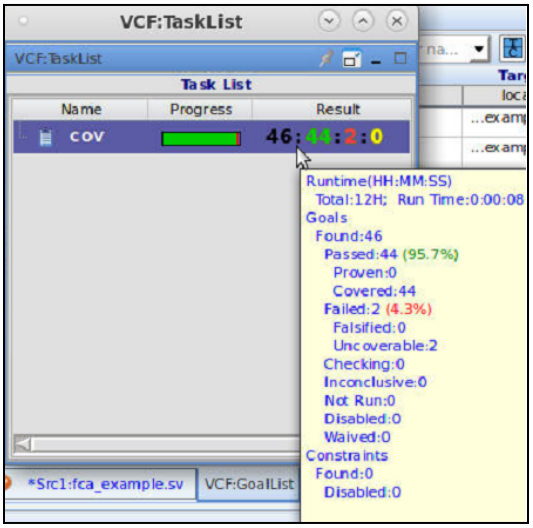


Figure 3.4. Results of the analyzed script.

This does not align exactly with our FCA score (as shown in *Figure 3.2*) because these results are specifically VC Formal tasks.

We can start by looking at the Summary Report. Use the command below in the VC Formal Console window to see the report:

```
report_fv
```

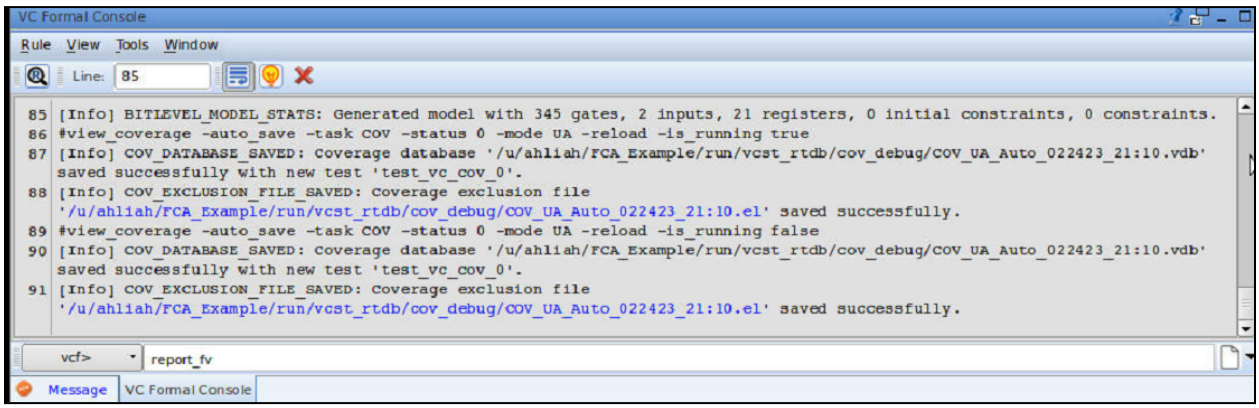


Figure 3.5. VC Formal Console window.

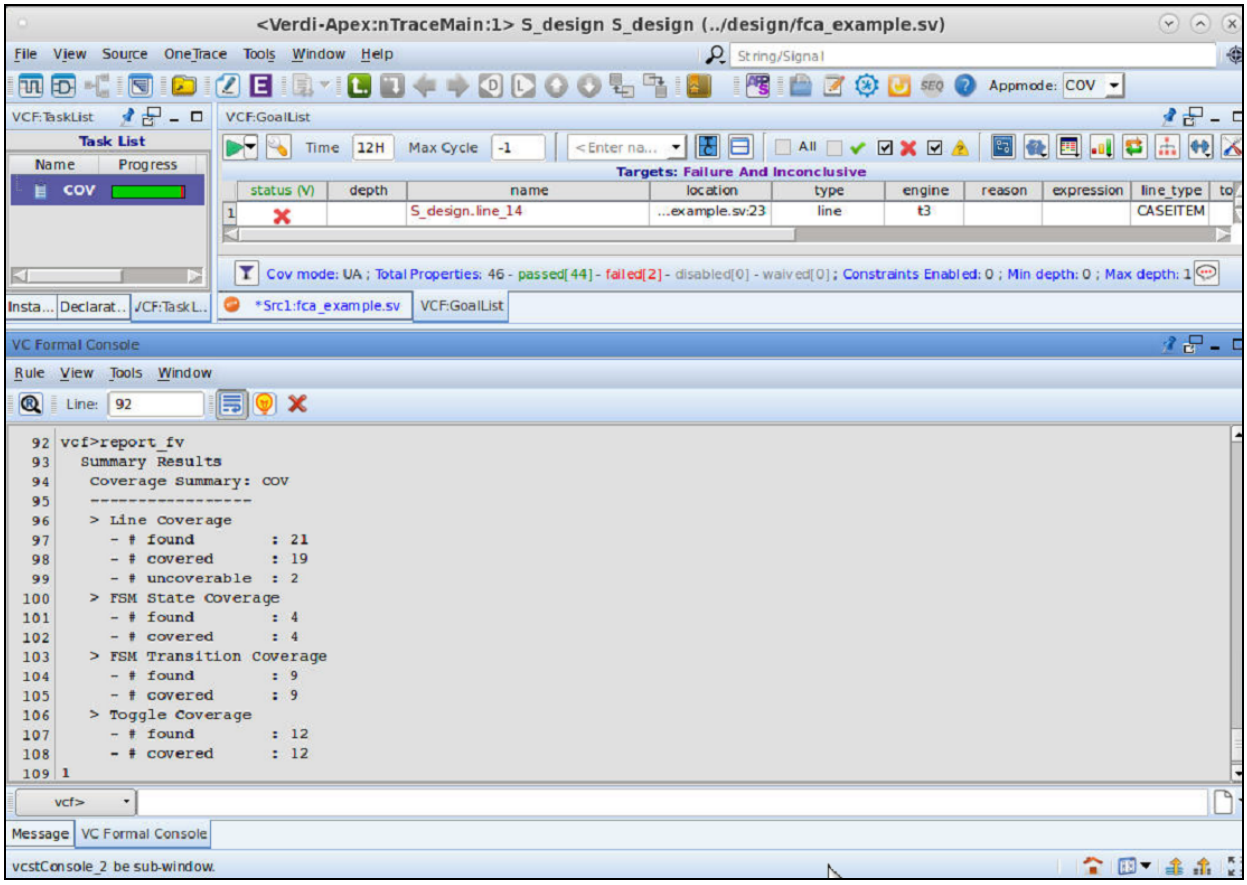


Figure 3.6. VC Formal Console showing Summary Results.

The Summary Results will show our Coverage Summary for the types of checks applicable to our design: Line, FSM Stage, FSM Transition, and Toggle. Ultimately, we want the number of *found* statuses to match the *covered* statuses.

Observe in Figure 3.6, the only time where this does not happen for our design is when FCA checks for Line Coverage, where it states that we have 2 uncoverable areas.

3.2 Generating Waveforms

In previous tutorials, we generated waveforms to examine and source trace our errors. In FCA, or COV mode, we will not be using this method, and waveforms can only be generated for covered conditions - the items with a green check  . The overall purpose of this is for the user to be able to review how a condition was covered.

To enable this feature, add this line to your tcl file:

set_fml_var fml_cov_gen_trace on

Make sure to save, restart/reload VC Formal after any changes to the script, and when you double-click on a check  , the waveform should be generated for that covered task.

```
# Portland State University VC_Formal_Team_FCA_App Tutorial

set_fml_appmode COV

# Set the module name as the design parameter
set design S_design

# Read the module "S_design" in the file /design/fca_example.sv
read_file -top $design -format sverilog \
        -sva -vcs {../design/fca_example.sv} -cov all

# Create clock and reset signals
create_clock clk -period 100
create_reset rst -sense high

# Running a reset simulation
sim_run -stable
sim_save_reset

# Enable trace for covered goals
set_fml_var fml_cov_gen_trace on
```

Figure 3.7. Adding command to tcl script to enable trace for covered goals.

3.3 Resolving Errors

To see the code/design where this fault is resulting, we go to the *type* column in the *VCF:GoalList* tab and double-click on *CASEITEM*.

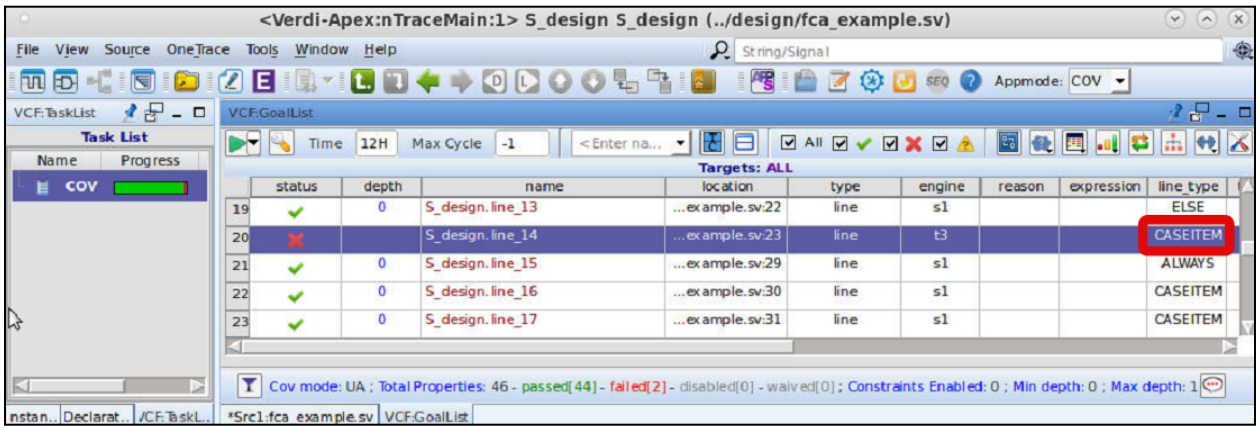


Figure 3.8. Under "line_type", double-click on "CASEITEM".

We are then taken to the part of the code that is causing this fault (CASEITEM), with the operation highlighted in blue, and the signal highlighted in red.

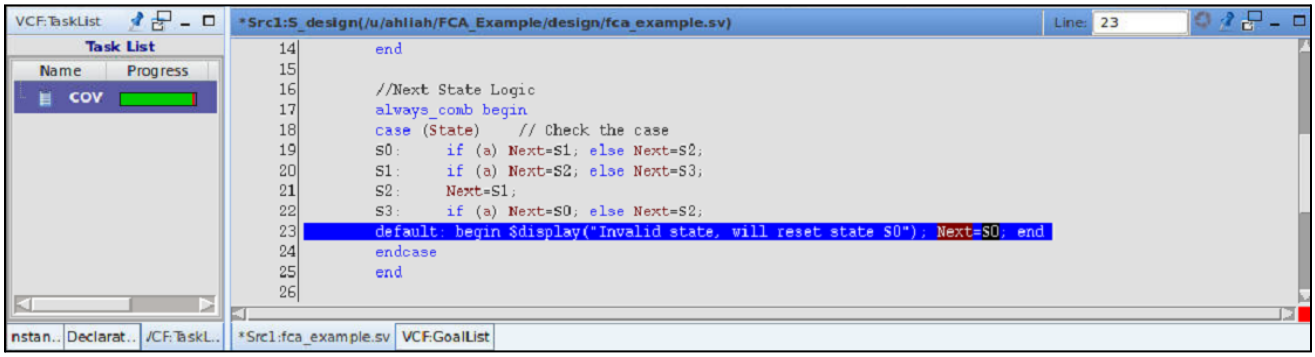


Figure 3.9. Identified errors within the Design file.

Looking at the FCA app Coverage Metrics table in the *Appendix*, the Line Coverage check looks at which lines of code are not exercised. In my code, the default case was not utilized during the simulation.

To fix this issue, we need to go alter our design file and make the following changes:

→ Comment out line 23

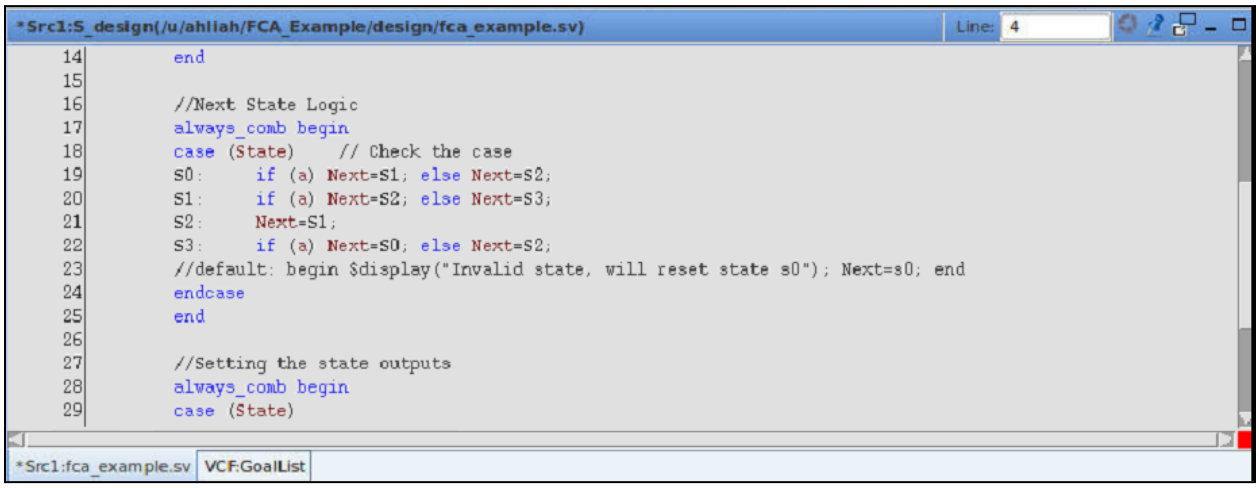


Figure 3.10. Altered design file to fix the error on line 23 (compare with Figure 18).

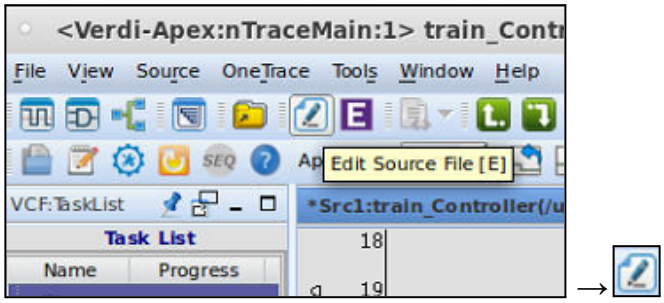


Figure 3.11. To make changes to the design file in VC Formal, click on "Edit Source File". Don't forget to save your changes.

Next, to complete our changes, follow the steps below to restart VC Formal.

3.4 Restarting VC Formal

We can restart VC Formal by clicking on  . If you invoked VC Formal along with the TCL script in the one-line command, then VC Formal will automatically load the same TCL file again. If you invoked VC Formal without the TCL script, then you will have to manually load the script again.

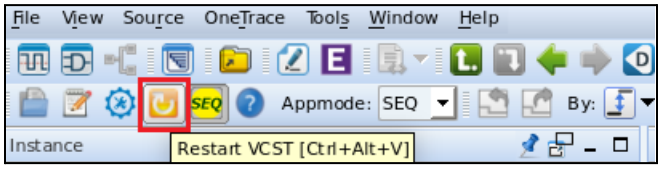


Figure 4.1. Location of the restart button in VC Formal window.

After the application and TCL file is loaded, click the green play button and we should see no errors:

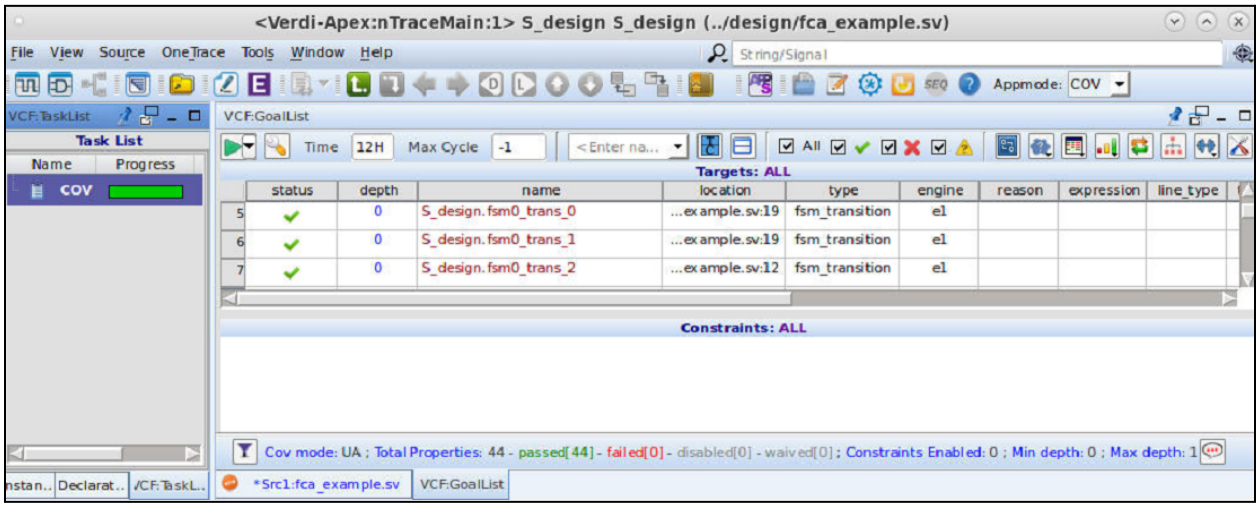


Figure 4.2. Results after fixing errors and restarting VC Formal and reloading TCL script.

Appendix

Coverage Type	Description
Line Coverage	Line coverage is applied to signal and variable assignments in HDL code. Shows which lines of code are exercised and which ones are not, by your testbench during a simulation run. A zero execution count, pin points a line of code that has not been exercised. This could be the source of a potential design error.
Condition Coverage	Monitors whether certain expressions and sub-expressions in your code evaluate to true or false. *Note: By condition coverage is not supported inside functions and tasks. You must use the -cm_cond tf option in VCS to enable the same.*
Toggle Coverage	Monitors value changes on signal bits in the design. When toggle coverage reaches 100%, that means every bit of monitored signals has changed its value from 0 to 1 and from 1 to 0. Missing transitions of values provide definitive conclusions about inactive elements and unexercised portions of the design. *Note: Toggle coverage is not supported inside functions and tasks.*
Branch Coverage	Monitors the execution of conditional statements such as if/else and case statements, and the ternary operator ?: in a design. By default, branch coverage analyzes which alternative of these conditional statements is covered. *Note: This is needed either -cov branch of set_fml_var fml_cov_enable_branch_cov true or with -cov all.*
FSM Coverage	Identifies a group of statements in the source code to be a finite state machine (FSM) and tracks the states and transitions that occur in the FSM during simulation.
SV Covergroups Coverage	Monitors values, transitions, and crosses for variables and signals. There are a list of limitations regarding SV covergroups; refer to the VC Formal User Guide on Formal Coverage Analyzer Application for more details (pg267).

Table 1.1. FCA App Coverage Metrics. Identifiers and descriptions of all coverages offered.