

Synopsys[®] VC Formal Tutorial

Formal Testbench Analyzer (FTA)

Version 1.1 | 21-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contents of this document are confidential, privileged, and only for the information of the intended recipient and may not be published or redistributed.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 FTA	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
Switching Apps	13
Debugging	15
Restart FTA and Verify Faults Resolved	19
Exporting Faults to FPV	20
Clustering Faults	21
Appendix	23
<i>Table 1.1. FTA App Functions/commands</i>	24

Introduction

VC Formal uses TCL (Tool Command Language) scripts to tell it what to do. The script can define the app within VC Formal 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 “FTA_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 FTA analysis for us.

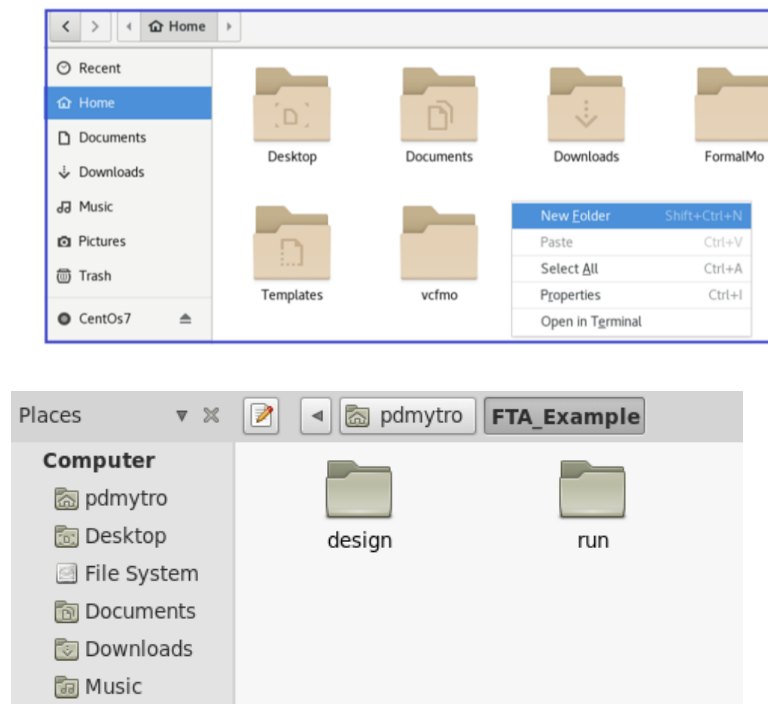
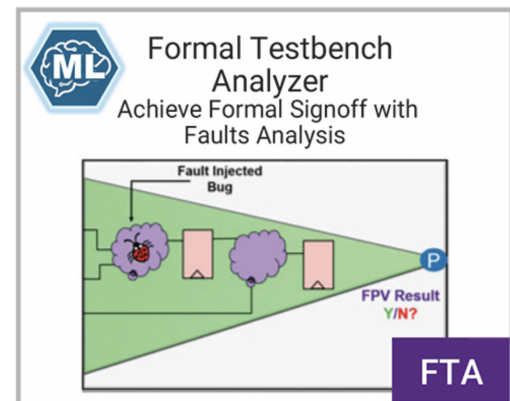


Figure 1. Creating folders to organize design and TCL files.

About and Usage of FTA

The Synopsys VC Formal FTA (Formal Testbench Apps) is an innovative tool designed to enhance the efficiency and effectiveness of functional verification using formal methods. VC Formal FTA offers a comprehensive set of features and capabilities for creating advanced testbenches based on formal techniques. By leveraging its powerful automation capabilities and intelligent analysis, VC Formal FTA enables engineers to rapidly generate high-quality testbenches that thoroughly exercise their designs, ensuring robustness and identifying hard-to-find bugs. This application revolutionizes the traditional testbench creation process, significantly reducing the time and effort required for functional verification.



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.

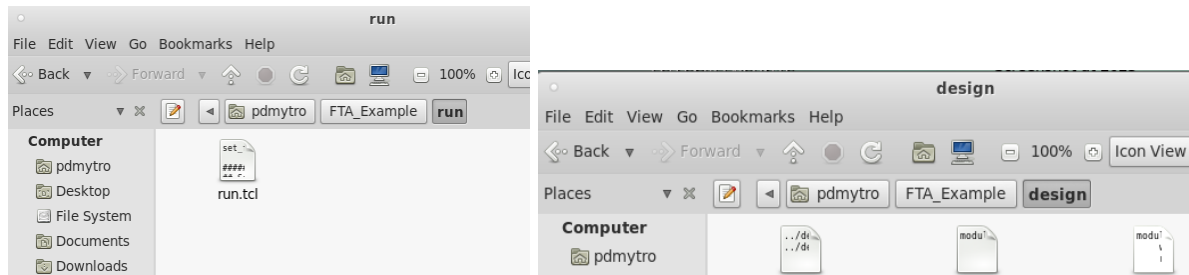


Figure 2. Showing the SystemVerilog and TCL files in the correct folder locations.

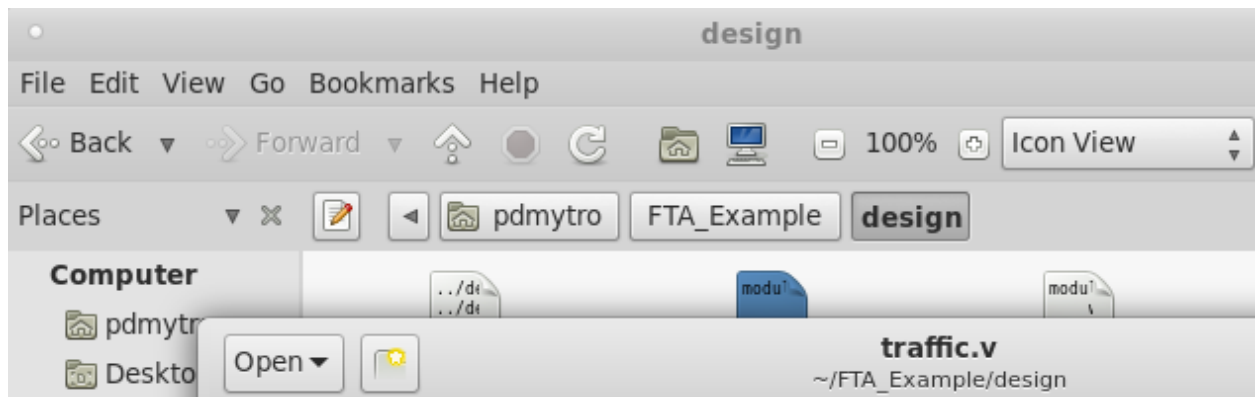


Figure 3. Example of the design we are using in this tutorial.

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.

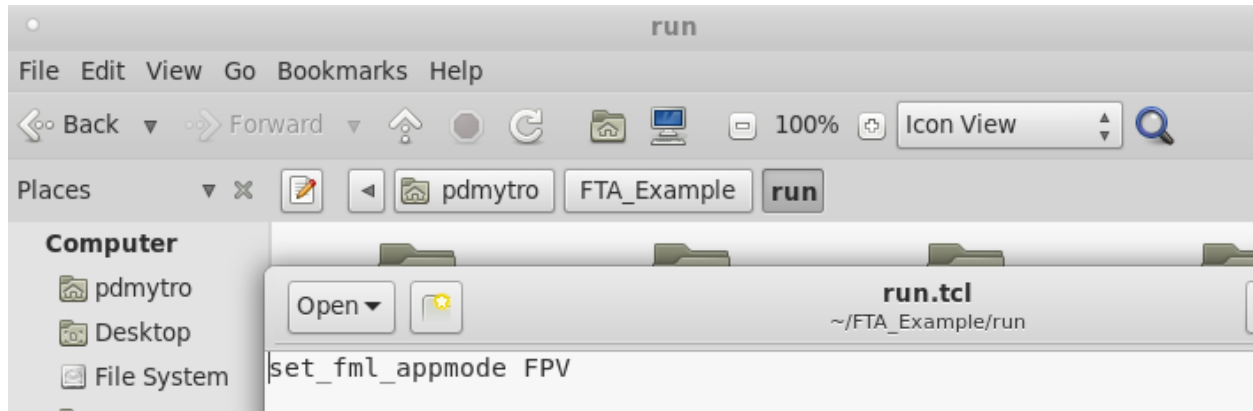


Figure 4. Annotated TCL template file.

- (1) Instruction that sets the appmode to FPV 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
```

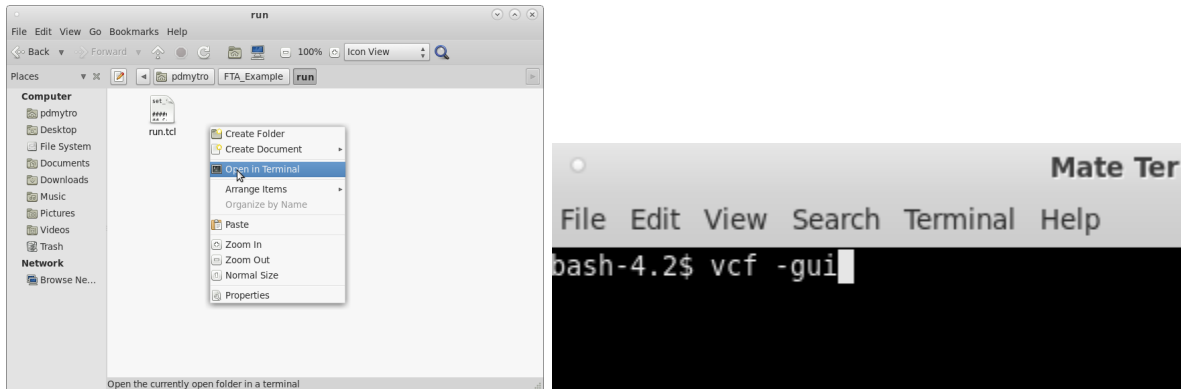



Figure 5. Invoking VC Formal in the terminal.

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: 

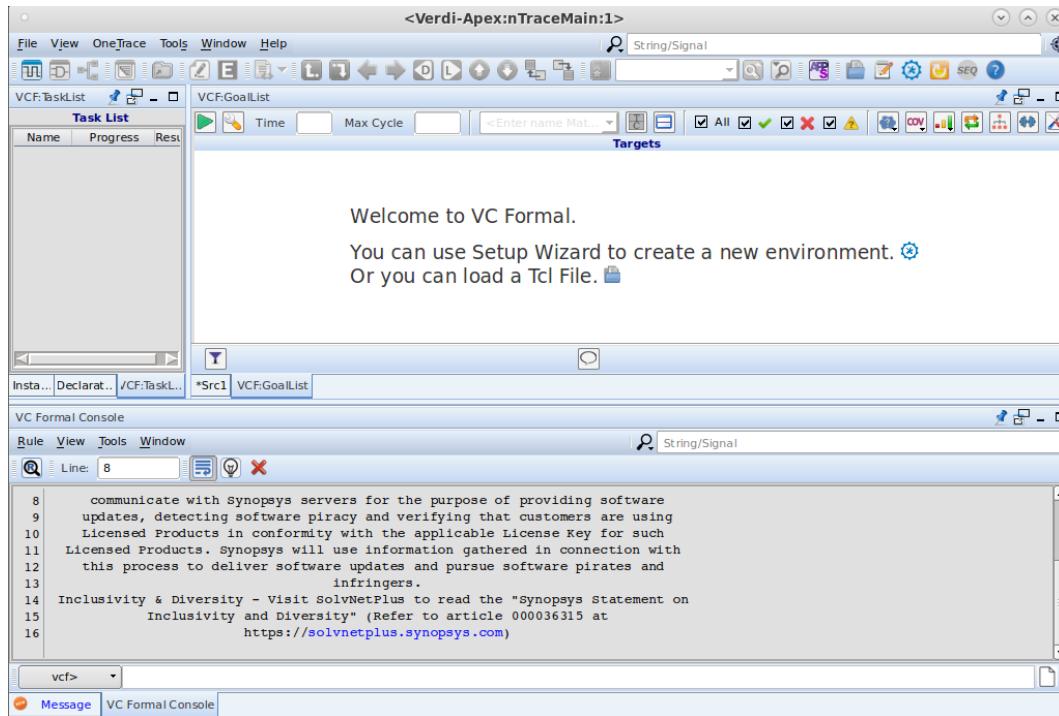



Figure 6. VC Formal GUI introductory screen.

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:

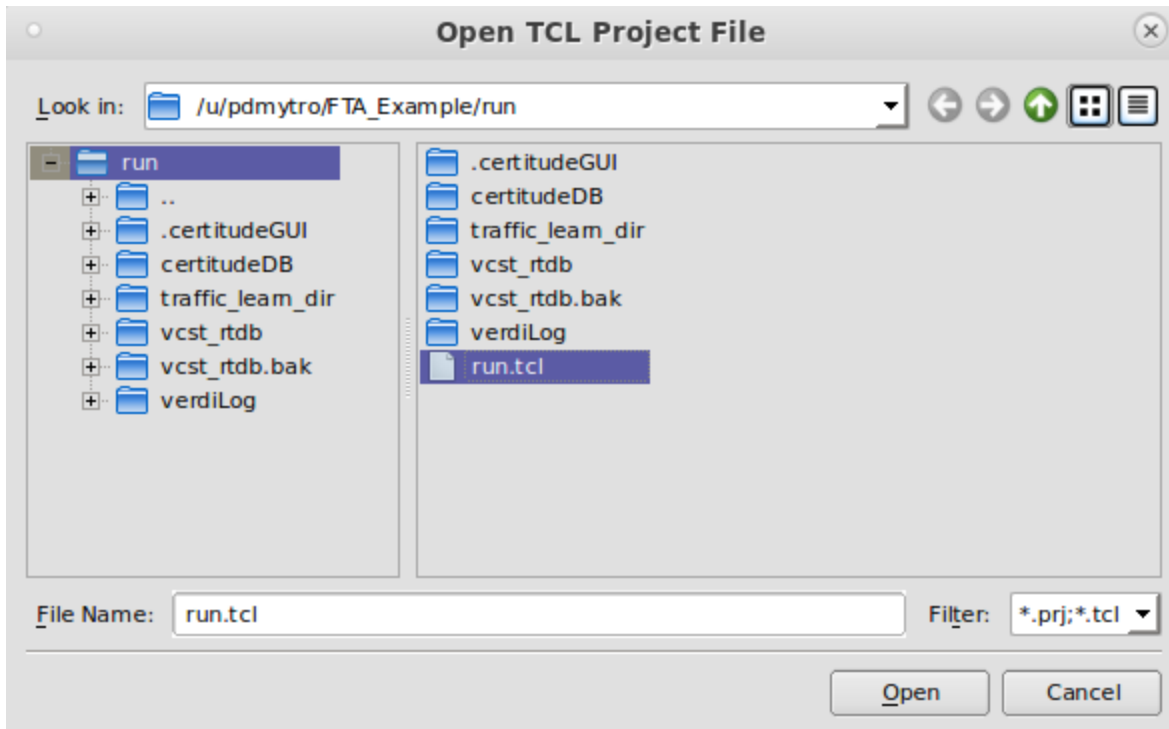


Figure 7. Selecting TCL file.

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
```

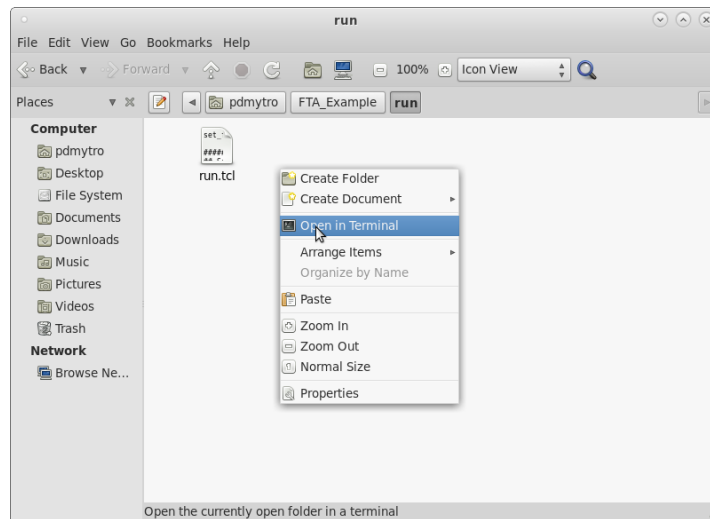


Figure 8. Opening terminal in the ‘run’ folder.

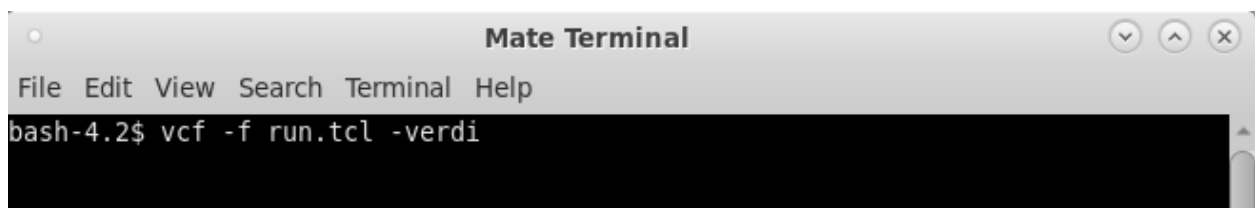


Figure 9. Invoking VC Formal and TCL script in the terminal.

And that’s it!

Running Formal Property Verification

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:

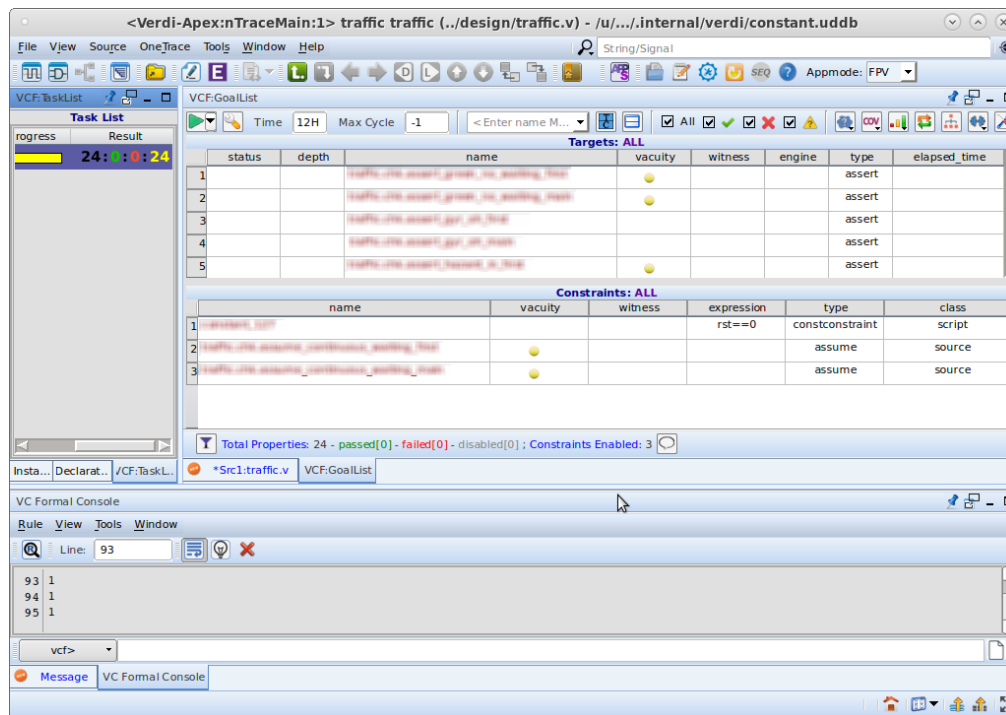



Figure 10. Screen after loading TCL script.

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

Once the property verification analysis is completed your screen should look like this:

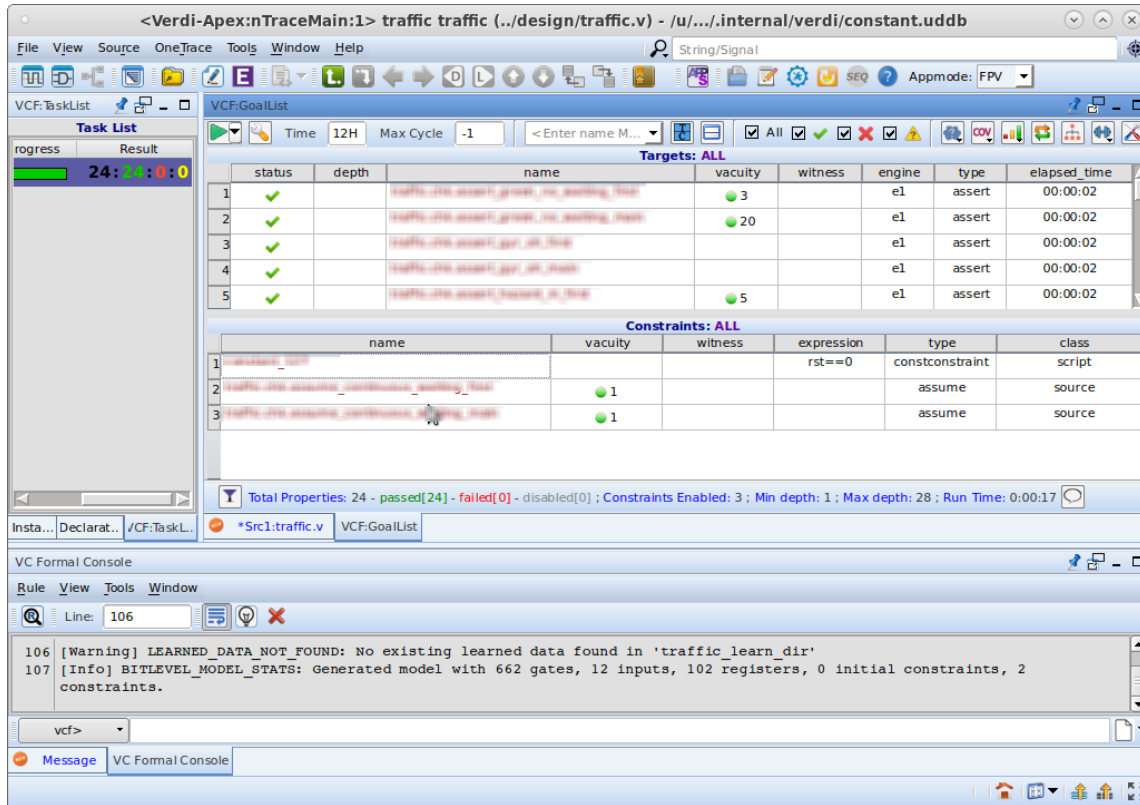


Figure 11. After running property verification

This step is necessary because the proven properties from the FPV run will be further used in the FTA flow to detect faults.

Switching from FPV app mode to FTA app mode

After running property verification, switch to the FTA app mode.

This is done by finding the appmode icon and clicking on the drop down menu:

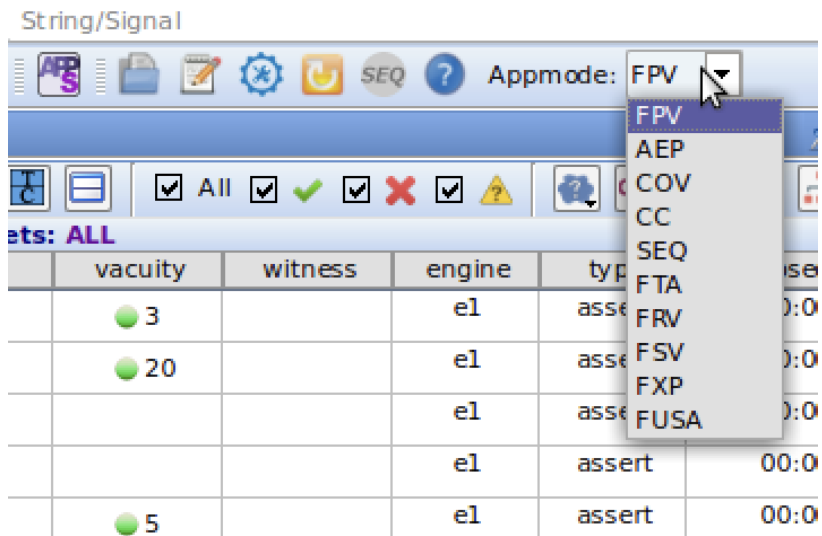


Figure 12. Switching between applications

Once the switch from the FPV app to the FTA app has been invoked your screen should like this:

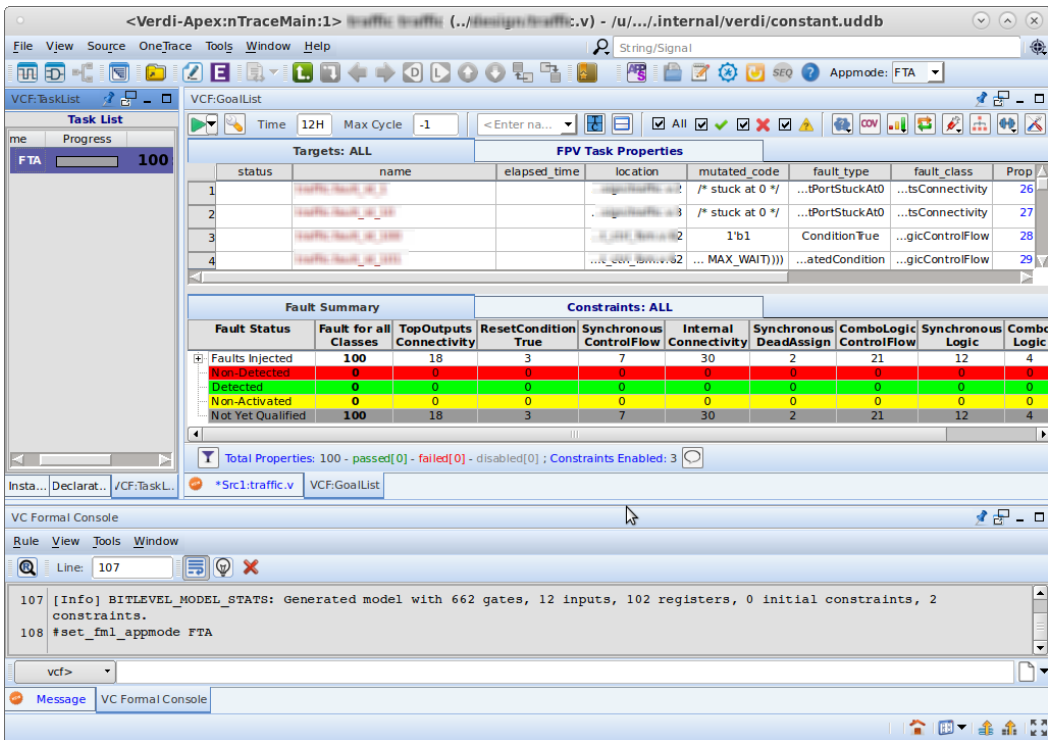




Figure 13. Loaded into FTA appmode after running property verification

Start property verification in the FTA appmode by clicking on the  play icon in the upper left corner.

The play  command will create an FPV_FTA task and start the verification process.

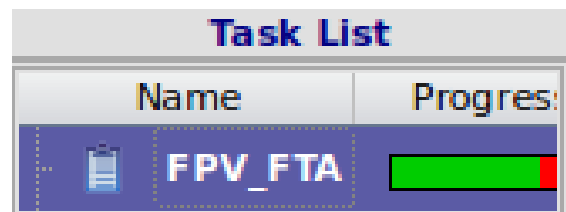


Figure 14: New task created once the start icon is clicked in the FTA appmode

Once the check is complete a fault summary will be displayed.

Fault Summary			Constraints: ALL						
Fault Status	Fault for all Classes	TopOutputs Connectivity	ResetCondition True	Synchronous ControlFlow	Internal Connectivity	Synchronous DeadAssign	ComboLogic ControlFlow	Synchronous Logic	Combo Logic
Faults Injected	100	18	3	7	30	2	21	12	4
Non-Detected	22	2	0	1	6	0	10	2	1
Detected	75	16	2	6	24	1	11	10	3
Non-Activated	3	0	1	0	0	1	0	0	0
Not Yet Qualified	0	0	0	0	0	0	0	0	0

Total Properties: 100 - passed[75] - failed[22] - disabled[3] ; Constraints Enabled: 3 ; Min depth: 2 ; Max depth: 22 ; Run Time: 0:06:18

Figure 15: Fault Summary display once FTA check is complete

Debugging Non-Detected Faults

Debugging non-detected faults is done through the fault summary that is generated when the FTA appmode is done running.

The non-detected faults will be displayed in the faults summary and they will be highlighted in red labeled “Non-detected”.

Non-Detected	22	2	0	1	6	0	10	2	1
--------------	----	---	---	---	---	---	----	---	---

Figure 16: Non-Detected faults tab in the faults summary

Double click on the TopOutputsConnectivity cell that corresponds to the Non-Detected (the cell with the number 2) cell to filter the “Targets: All” table to show the “Targets: Failure” table.

Fault Summary		
Fault Status	Fault for all Classes	TopOutputs Connectivity
Faults Injected	100	18
Non-Detected	22	2

Figure 17: TopOutputsConnectivity X Non-Detected Faults (2)

The displayed “Targets: Failure” table will look like the following table:

Targets: Failure Filter by fault_class, status			FPV Task Properties							
status (V)	name	elapsed_time	location	mutated_code	fault_type	fault_class (V)	Prop ID			
1	✗	...tPortStuckAt0	00:05:23	...tPortStuckAt0	/* stuck at 0 */	...tPortStuckAt0	...tsConnectivity	43		
2	✗	...tPortStuckAt0	00:05:36	...tPortStuckAt0	/* stuck at 0 */	...tPortStuckAt0	...tsConnectivity	72		

Fault Summary			Constraints: ALL							
Fault Status	Fault for all Classes	TopOutputs Connectivity	ResetCondition True	Synchronous ControlFlow	Internal Connectivity	Synchronous DeadAssign	ComboLogic ControlFlow	Synchronous Logic	ComboLogic	
Faults Injected	100	18	3	7	30	2	21	12	4	
Non-Detected	22	2	0	1	6	0	10	2	1	
Detected	75	16	2	6	24	1	11	10	3	
Non-Activated	3	0	1	0	0	1	0	0	0	
Not Yet Qualified	0	0	0	0	0	0	0	0	0	

Total Properties: 100 - passed[75] - failed[22] - disabled[3] ; Constraints Enabled: 3 ; Min depth: 2 ; Max depth: 22 ; Run Time: 0:06:18

Figure 18: Targets:Failure table showing the undetected faults


The isolated undetected faults will be labeled with an  in the status cell. Once the non-detected faults are identified and isolated, double click on the fault in the name cell to open up a highlighted source code window. This window will open the source code and identify the undetected faults with the color red.



Figure 19: Non-detected faults identified in the source code highlighted in red

Modify the property by clicking on the “edit source file” icon



Once modified save the new source code.

Debugging Non-Activated Faults

Debugging non-activated faults is also done through the fault summary that is generated when the FTA appmode is done running.


The non-activated faults will be displayed in the faults summary and they will be highlighted in yellow labeled “Non-activated”.

Non-Activated	3	0	1	0	0	1	0	0	0
---------------	---	---	---	---	---	---	---	---	---

Figure 20: Non-activated faults tab in the faults summary (highlighted in yellow)


Double click on the ResetCondition True cell that corresponds to the Non-activated (the cell with the number 1) cell to filter the “Targets: All” table to show the “Targets: Failure” table.

The displayed “Targets: Failure” table will look like the following table:


Targets: Failure Filter by fault_class, status			FPV Task Properties						
status (V)	name	elapsed_time	location	mutated_code	fault_type	fault_class (V)	Prop ID		
1 	traffic.fault_id_50		...ign/traffic.v:45	1'b1	ConditionTrue	...tConditionTrue	83		

Fault Summary			Constraints: ALL						
Fault Status	Fault for all Classes	TopOutputs Connectivity	ResetCondition True	Synchronous ControlFlow	Internal Connectivity	Synchronous DeadAssign	ComboLogic ControlFlow	Synchronous Logic	Combo Logic
Faults Injected	100	18	3	7	30	2	21	12	4
Non-Detected	22	2	0	1	6	0	10	2	1
Detected	75	16	2	6	24	1	11	10	3
Non-Activated	3	0	1	0	0	1	0	0	0
Not Yet Qualified	0	0	0	0	0	0	0	0	0

Figure 21: Targets:Failure table showing the non-activated faults


The isolated non-activated faults will be labeled with an  in the status cell. Once the non-activated faults are identified and isolated, double click on the fault in the name cell to open up a highlighted source code window. This window will open the source code and identify the non-activated faults with the color yellow.

If the non-activated faults indicate source code that is outside of the current set properties of the current used FTA flow, these properties will need to be added. Adding properties that cover those signals will eliminate non-activated faults. Since these properties are set in the .tcl script, the current script will need to be edited.

To edit and add properties to the Tcl script find the “Edit TCL Project File” icon. () This can also be done by opening up the Tcl script from the run folder in your project with a text editor application.

Add the commands to cover the signals outside your script's current cone of influence and save the edited Tcl script by pressing the save button.

Restart FTA and Verify Faults Resolved

Restart VC Formal by clicking on the Restart VCST icon. ()

Once VC Formal is restarted after making changes to the .tcl/source files & adding modified assertions, we can go back in and verify whether or not the faults have been resolved.

Once VC formal is restarted with the updated parameters, observe that the non-activated and non-detected faults are now detected.

Fault Summary			Constraints: ALL						
Fault Status	Fault for all Classes	TopOutputs Connectivity	ResetCondition True	Synchronous ControlFlow	Internal Connectivity	Synchronous DeadAssign	ComboLogic ControlFlow	Synchronous Logic	Combo Logic
Faults Injected	100	18	3	7	30	2	21	12	4
Non-Detected	20	0	0	1	6	0	10	2	1
Detected	80	18	3	6	24	2	11	10	3
Non-Activated	0	0	0	0	0	0	0	0	0
Not Yet Qualified	0	0	0	0	0	0	0	0	0

Total Properties: 100 - passed[80] - failed[20] - disabled[0] ; Constraints Enabled: 3 ; Min depth: 2 ; Max depth: 22 ; Run Time: 0:03:05

Figure 22: Observe the non-activated faults and non-detected faults are now detected

Exporting Faults to FPV

The faults can be exported as an FPV task.

Right-click on a fault status category and choose “Export Fault”

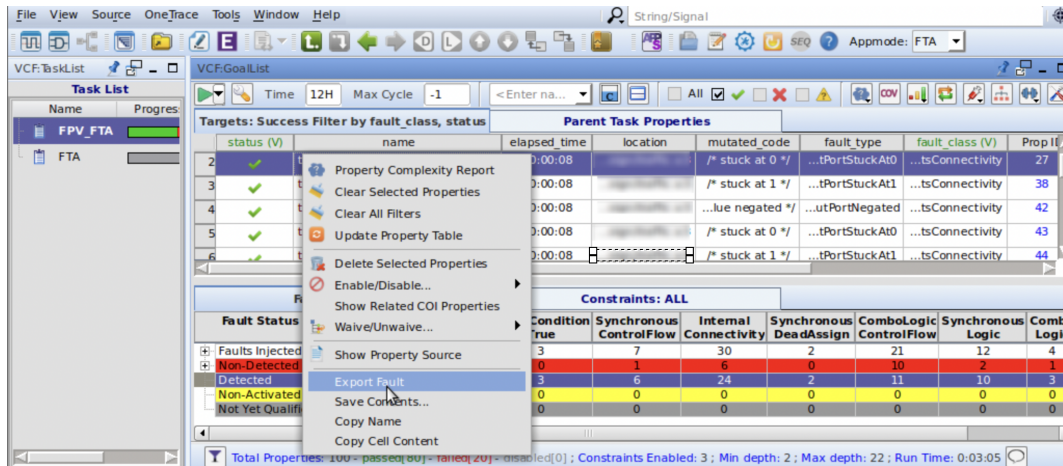



Figure 23: Exporting Faults to FPV


The target fault is now exported to the FPV task where its impact on proven properties can be checked by running the proof on the design with the fault inserted.

Once faults are exported to FPV, we can start the property verification by pressing the green play button/Start check icon. ()

Once completed check that the properties are falsified due to the exported fault.

Clustering Faults

Clustering faults can be a tool used for easier and more efficient analysis. Clusters are faults grouped together based on their classification and their locations in the DUT.

Cluster the faults by clicking the “FTA Property and Fault Selection” icon () and selecting the “non-detected” fault status cluster option.


Return to the fault summary.

Expand the non-detected faults highlighted in red, by clicking the “+”, to see the faults clustered in the same fault class.

Fault Summary			Constraints: ALL						
Fault Status	Fault for all Classes	TopOutputs Connectivity	ResetCondition True	Synchronous ControlFlow	Internal Connectivity	Synchronous DeadAssign	ComboLogic ControlFlow	Synchronous Logic	Com Log
Faults Injected	100	18	3	7	30	2	21	12	4
Non-Detected	20	0	0	1	6	0	10	2	1
Cluster 0	1	0	0	0	0	0	0	0	1
Cluster 1	3	0	0	0	0	0	3	0	0
Cluster 2	2	0	0	0	2	0	0	0	0
Cluster 3	1	0	0	1	0	0	0	0	0
Cluster 4	2	0	0	0	2	0	0	0	0
Cluster 5	7	0	0	0	0	0	7	0	0
Cluster 6	1	0	0	0	1	0	0	0	0
Cluster 7	2	0	0	0	0	0	0	2	0
Cluster 8	1	0	0	0	1	0	0	0	0

Total Properties: 100 - passed[80] - failed[20] - disabled[0]; Constraints Enabled: 3; Min depth: 2; Max depth: 22; Run Time: 0:03:05

Figure 24: Clustered faults by all of the faults in the same fault classification

Analyze the faults by double clicking on the cluster number. The clustered faults will appear in the “Targets:Failure” table. They will be labeled with a  status.

Targets: Failure Filter by fault_class, status

FPV Task Properties

status (V)	name	elapsed_time	location	mutated_code	fault_type	fault_class (V)	Prop ID
1	tsConnectivity	00:05:23	tsConnectivity	/* stuck at 0 */	...tPortStuckAt0	...tsConnectivity	43
2	tsConnectivity	00:05:36	tsConnectivity	/* stuck at 0 */	...tPortStuckAt0	...tsConnectivity	72

Fault Summary

Constraints: ALL

Fault Status	Fault for all Classes	TopOutputs Connectivity	ResetCondition True	Synchronous ControlFlow	Internal Connectivity	Synchronous DeadAssign	ComboLogic ControlFlow	Synchronous Logic	Combo Logic
Faults Injected	100	18	3	7	30	2	21	12	4
Non-Detected	22	2	0	1	6	0	10	2	1
Detected	75	16	2	6	24	1	11	10	3
Non-Activated	3	0	1	0	0	1	0	0	0
Not Yet Qualified	0	0	0	0	0	0	0	0	0

Total Properties: 100 - passed[75] - failed[22] - disabled[3]; Constraints Enabled: 3; Min depth: 2; Max depth: 22; Run Time: 0:06:18

Figure 25: Clustered Faults Table

Double click on the name of the clustered property to open a highlighted source code file showing the clustered non-detected faults highlighted in the color red.

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.
FTA create	fta_create	Creates a new formal testbench project. It initializes the project structure and sets up the necessary files and directories.
FTA Add Property	fta_add_property	This command allows you to add properties to the formal testbench. You can specify assertions, cover properties, or assume-guarantee properties to capture the desired behavior of the design.
FTA Generate Testbench	fta_generate_testbench	Automatically generate a formal testbench based on the properties specified. The tool will synthesize test vectors and stimulus patterns to exercise the design and verify the properties.
FTA Analyze Coverage	fta_analyze_coverage	Enables you to analyze the coverage achieved by the formal testbench. It provides insights into the completeness of the testbench and identifies areas that require additional coverage.
FTA Debug	fta_debug	Initiate the formal debugging process for the testbench. It provides interactive capabilities to help analyze the behavior of the design and investigate any issues encountered during verification.
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. FTA App important functions and commands