Synopsys® VC Formal Tutorial

Functional Safety Application (FuSa)

Version 1.0 | 13-April-2023

Portland State University

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contents of this document are confidential, privileged, and only for the information of the intended recipient and may not be published or redistributed.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

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 “FuSa_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 FuSa analysis for us.

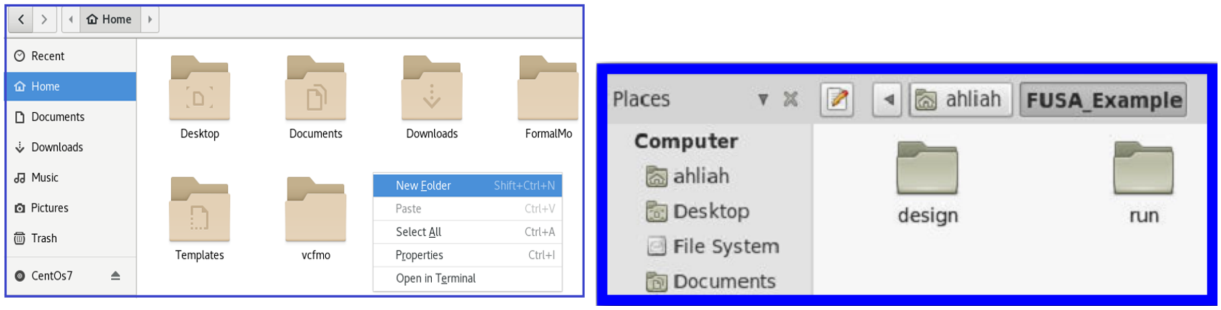
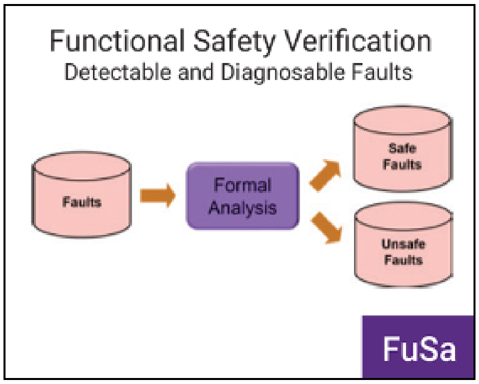


Figure 1.1. Creating folders to organize design and TCL files.

1.1 About and Usage of FuSa


Functional Safety Verification
Detectable and Diagnosable Faults
Faults
Formal Analysis
Safe Faults
Unsafe Faults
FuSa

The Functional Saftey Verification (FuSa) application is used as a safety tool to minimize risk caused by a malfunctioning system. A separate in-house tool that Synopsys uses, called Z01X, is a Verilog software fault injection simulator that provides coverage results. VC Formal takes these results and determines whether the type of faults simulated and identified are "safe" or "dangerous", via the FuSa application. Since the Z01X tool produces many types of faults, VC Formal serves as an extra step to help sift through and reduce the need for manual reviewing.

The FuSa application performs the following types of analysis:

- ❖ Structural Analysis
- ❖ Observability Analysis
- ❖ Controllability Analysis
- ❖ Detection Analysis

Structural Analysis is used to identify whether faults are in or not in the COI of observation/detection points. They are marked if they are not and those that are not are put through further analysis.

Controlability Analysis is used to determine if a signal can transition from one established state to another at the location of the fault. Cover properties are set at fault locations and are marked non-controllable if unreachable or sent for further analysis if properties are covered/inconclusive.

Observability and Detection Analysis is used to observe whether or not a fault is blocking a signal from propagating through to an observation/detection point. Faults that do not propagate are considered safe. Faults that propagate to observation/detection points are marked accessed further.

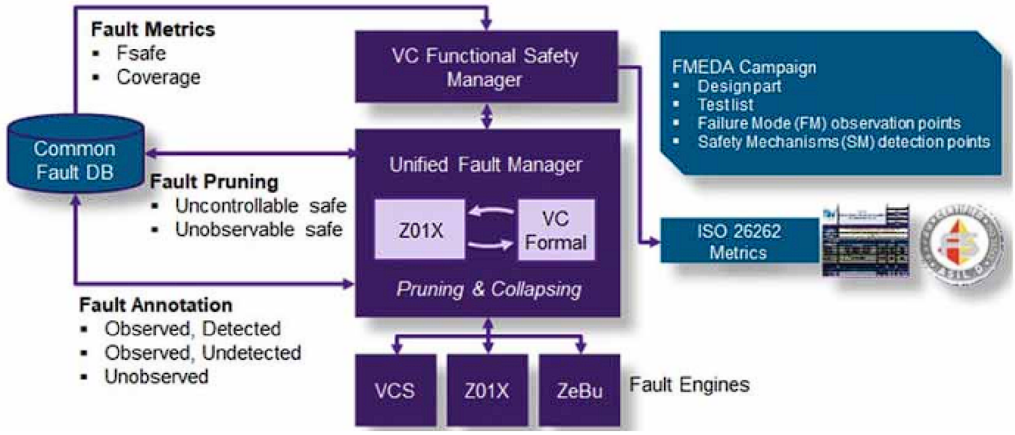


Figure 1.1.1. FuSa Verification Flow Map

1.2 Design Files

In the Design folder, you'll put in your RTL design file(s) and SFF file. Your design file should include all necessary clocks, reset, and constraints needed to function, while also being free from any errors or violations. If your design includes a testbench and/or multiple .sv files, make sure they are also free from any violations, then house them in the design folder. For this tutorial, there will only be one design file (fusa_design.sv).

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.

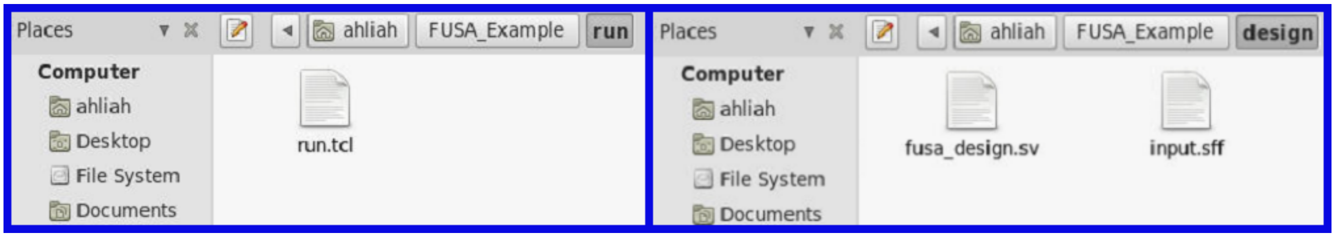


Figure 1.2.1. Showing the (Right) RTL/SFF and (Left) TCL file in the correct folder locations.

1.3 TCL File Syntax

Next, we will create a TCL command script for FuSa setup. This is where we compile our designs using various commands and perform each Fuctional Safety Analysis type respectivley.

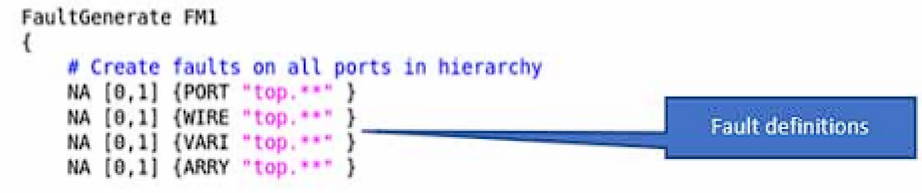
As with the other applications, make sure to set app mode to DPV:

```
set_fml_appmode DPV
```

Then, you need to load the faultlist:

```
Fusa_config -sff <sff filename>
```

Importing faults from SFF menas that faults are defined in Standard Fault Format.



```
FaultGenerate FM1
{
    # Create faults on all ports in hierarchy
    NA [0,1] {PORT "top.**" }
    NA [0,1] {WIRE "top.**" }
    NA [0,1] {VARI "top.**" }
    NA [0,1] {ARRY "top.**" }
```

Figure 1.3.1. Faults defined in SFF, within a .sff file.

Once the faultlist has been imported, compile your error-free design and define “clock” and “reset” as needed for your design

```
read_file -top test -format verilog -vcs <RTL filename>
```

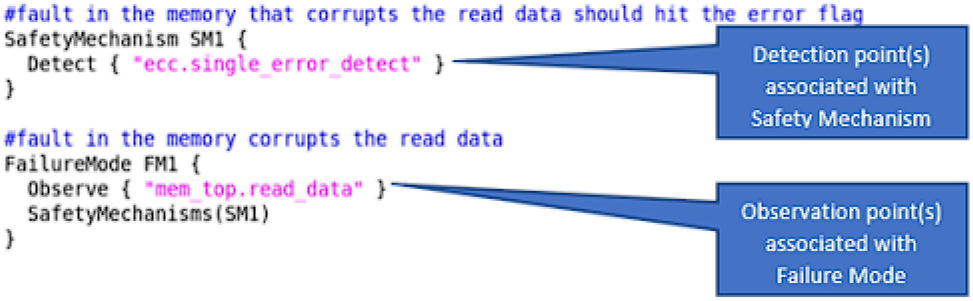
and define “clock” and “reset” as needed for your design following these commands:

`create_clock` and `create_reset`

Then initialize VCF setup:

`sim_run -stable` and `sim_save_reset`

If Observation/Detection points are not already specified in the SFF file like this:



```
#fault in the memory that corrupts the read data should hit the error flag
SafetyMechanism SM1 {
  Detect { "ecc.single_error_detect" }
}

#fault in the memory corrupts the read data
FailureMode FM1 {
  Observe { "mem_top.read_data" }
  SafetyMechanisms(SM1)
}
```

Detection point(s) associated with Safety Mechanism
Observation point(s) associated with Failure Mode

Figure 1.3.2. Observation/Detection points defined within a .sff file.

You can identify them in the TCL file with these commands:

```
fusa_observation -add {<filename>.<list of signal names>}
```

and

```
fusa_detection -add {<filename>.<list of signal names>}
```

To ensure that false proofs are avoided, this command allows data to propagate through a safe black box:

```
fusa_blackbox -all -all_auto_path
```

And now to generate FuSa properties:

```
fusa_generate
```

From here we can run Structual, Controllability, Observability, or Detectability analysis:

```
set_fml_var fusa_run_mode <structural/control/observe/detect>
```

Following this, you want to run the FuSa check and report:

`check_fv` and `fusa_report`

The end of the TCL file will be for saving the results to SFF:

```
fusa_save -sff < sff filename>
```

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 FuSa 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".

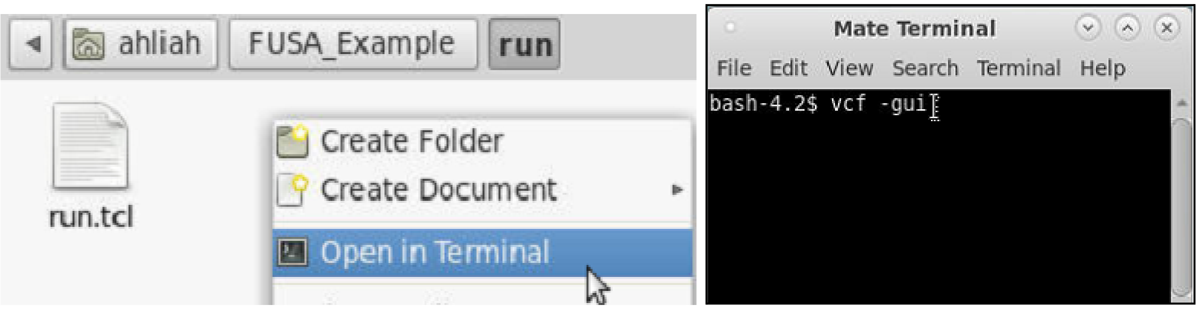
We will go through both of these methods in the following sections.

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
```



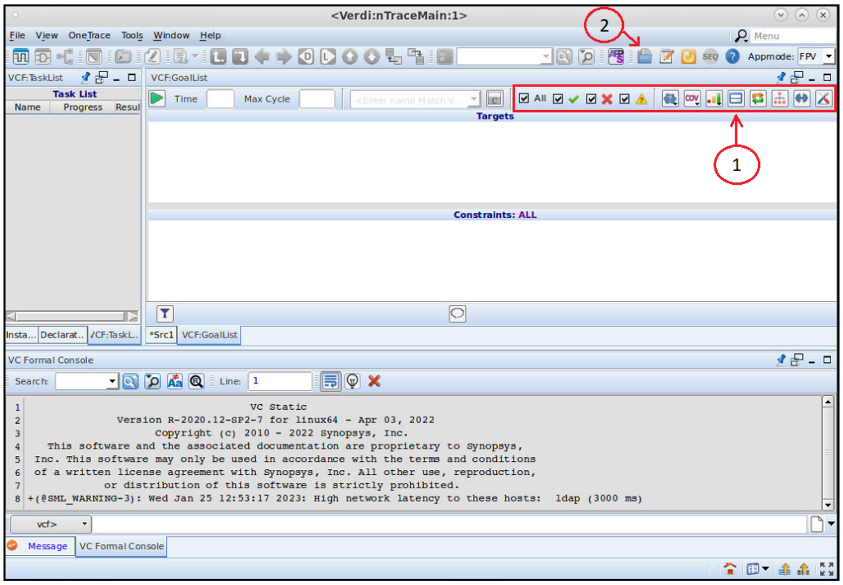
```
bash-4.2$ vcf -gui
```

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: ⊟ ▭ ▢

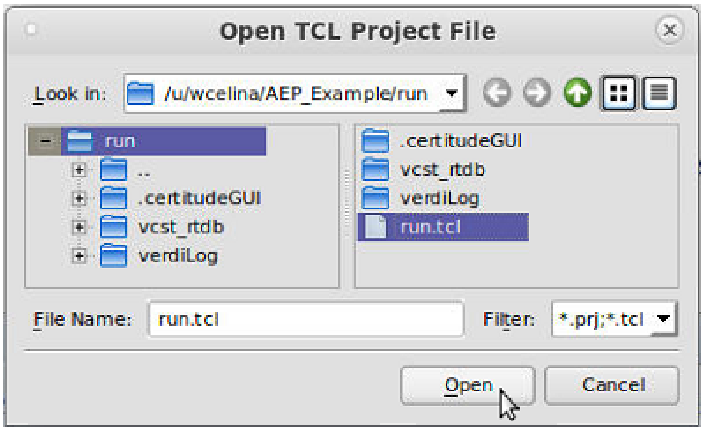


```
                                VC Static
                Version R-2020.12-SP2-7 for linux64 - Apr 03, 2022
                      Copyright (c) 2010 - 2022 Synopsys, Inc.
   This software and the associated documentation are proprietary to Synopsys,
 Inc. This software may only be used in accordance with the terms and conditions
 of a written license agreement with Synopsys, Inc. All other use, reproduction,
           or distribution of this software is strictly prohibited.
+(@SML_WARNING-3): Wed Jan 25 12:53:17 2023: High network latency to these hosts:  ldap (3000 ms)
```

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:



Look in: /u/wcelina/AEP_Example/run

File Name: run.tcl

Filter: *.prj;*.tcl

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
```

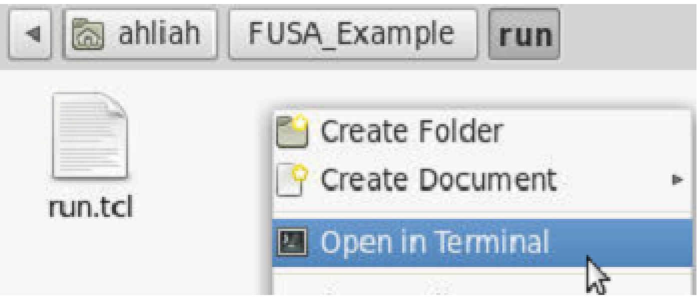
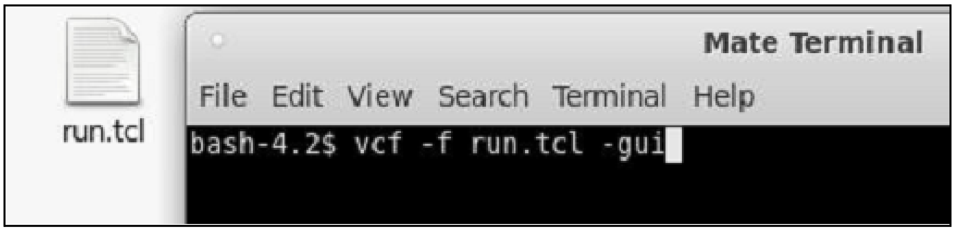


Figure 2.4. Opening terminal in the ‘run’ folder.



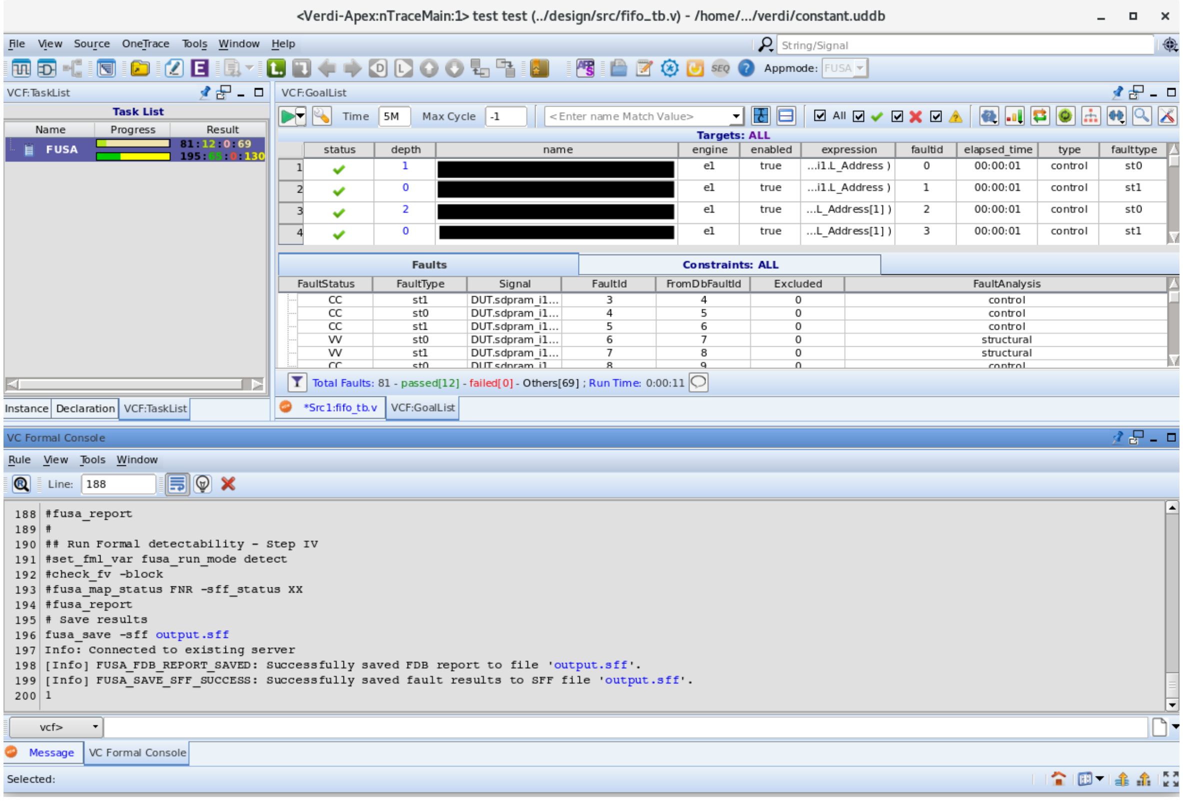
```
bash-4.2$ vcf -f run.tcl -gui
```

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:



Task List

Name	Progress	Result
FUSA		81 : 12 : 0 : 69 195 : 65 : 0 : 130

Targets: ALL

	status	depth	name	engine	enabled	expression	faultid	elapsed_time	type	faulttype
1	✓	1	[redacted]	e1	true	...i1.L_Address)	0	00:00:01	control	st0
2	✓	0	[redacted]	e1	true	...i1.L_Address)	1	00:00:01	control	st1
3	✓	2	[redacted]	e1	true	...L_Address[1])	2	00:00:01	control	st0
4	✓	0	[redacted]	e1	true	...L_Address[1])	3	00:00:01	control	st1

Faults **Constraints: ALL**

FaultStatus	FaultType	Signal	FaultId	FromDbFaultId	Excluded	FaultAnalysis
CC	st1	DUT.sdpram_i1...	3	4	0	control
CC	st0	DUT.sdpram_i1...	4	5	0	control
CC	st1	DUT.sdpram_i1...	5	6	0	control
VV	st0	DUT.sdpram_i1...	6	7	0	structural
VV	st1	DUT.sdpram_i1...	7	8	0	structural
CC	st0	DUT.sdpram_i1...	8	9	0	control

Total Faults: 81 - passed[12] - failed[0] - Others[69] ; Run Time: 0:00:11

```
#fusa_report
#
## Run Formal detectability - Step IV
#set_fml_var fusa_run_mode detect
#check_fv -block
#fusa_map_status FNR -sff_status XX
#fusa_report
# Save results
fusa_save -sff output.sff
Info: Connected to existing server
[Info] FUSA_FDB_REPORT_SAVED: Successfully saved FDB report to file 'output.sff'.
[Info] FUSA_SAVE_SFF_SUCCESS: Successfully saved fault results to SFF file 'output.sff'.
1
```

Figure 3.1. Screen after loading TCL script.

An initial Verification analysis will run automatically after invoking VC Formal. However, you can choose one of the Functional Safety Analysis types by clicking on the down arrow next to the play icon in the upper left corner of the *VCF:GoalList* tab.

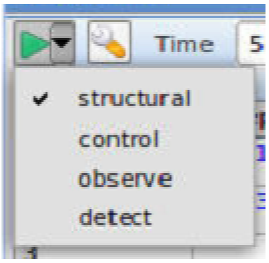
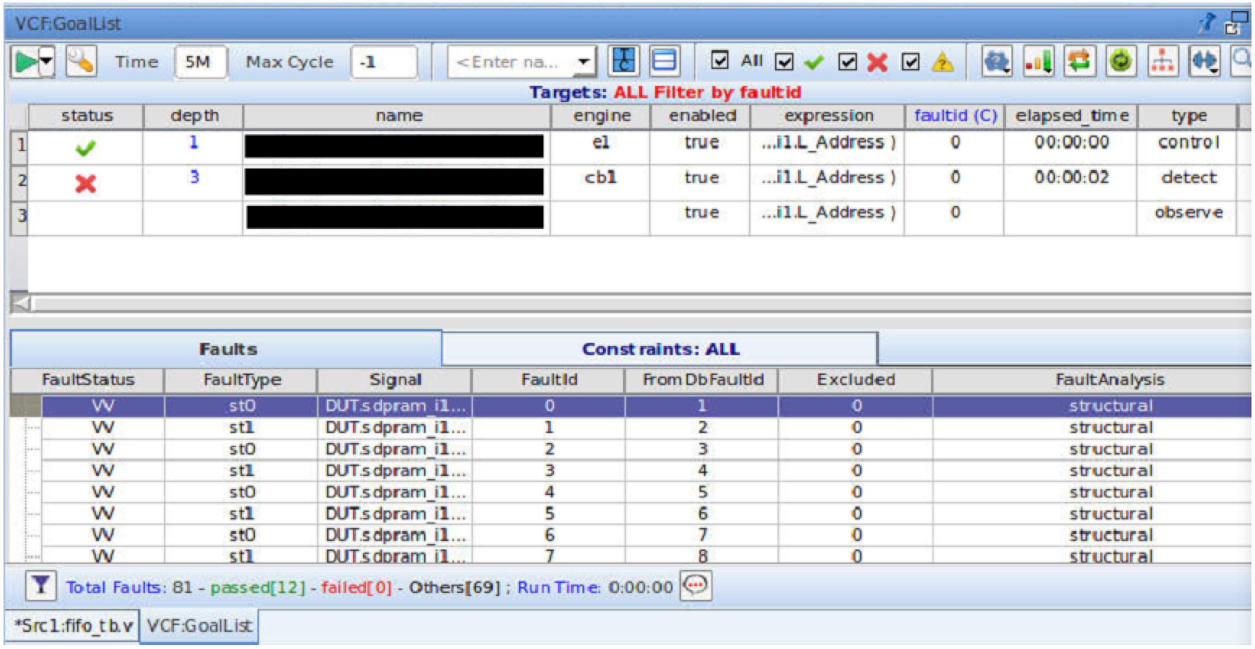


Figure 3.2. Functional Safety Analysis types from drop arrow menu.

3.1 Detecting Errors

When you run your design through one of these FuSa verification types, you will see the associates faults with that analysis. Here we want to take notice of the Faults tab within the VCF:GoalList window.

Double-click a fault Signal and it will populate the properties associated. Here we have 1 passed status and 1 failed status.



Targets: ALL Filter by faultid

	status	depth	name	engine	enabled	expression	faultid (C)	elapsed_time	type
1	✓	1	[redacted]	e1	true	...i1.L_Address)	0	00:00:00	control
2	✗	3	[redacted]	cb1	true	...i1.L_Address)	0	00:00:02	detect
3			[redacted]		true	...i1.L_Address)	0		observe

FaultStatus	FaultType	Signal	FaultId	From DbFaultId	Excluded	FaultAnalysis
VV	st0	DUT.sdpram_i1...	0	1	0	structural
VV	st1	DUT.sdpram_i1...	1	2	0	structural
VV	st0	DUT.sdpram_i1...	2	3	0	structural
VV	st1	DUT.sdpram_i1...	3	4	0	structural
VV	st0	DUT.sdpram_i1...	4	5	0	structural
VV	st1	DUT.sdpram_i1...	5	6	0	structural
VV	st0	DUT.sdpram_i1...	6	7	0	structural
VV	st1	DUT.sdpram_i1...	7	8	0	structural

Total Faults: 81 - passed[12] - failed[0] - Others[69] ; Run Time: 0:00:00

Figure 3.3. Functional Safety Analysis types from drop arrow menu.

In *Figure 3.3* above, we see one ☑ icon. VC Formal gives this "Passed" status icon for the part of our design that was "covered" by FuSa.

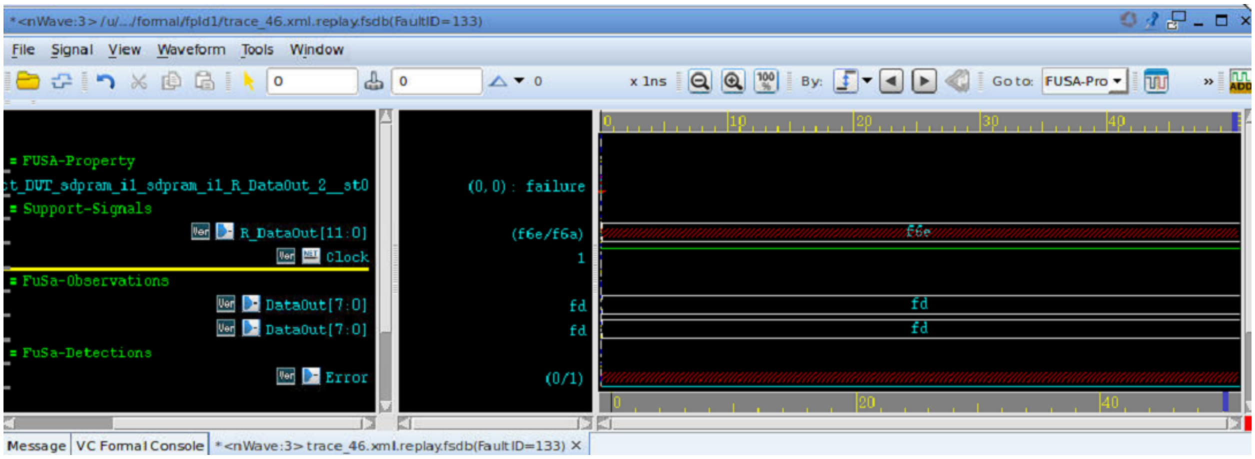
We also see one ☒ icon; VC Formal gives this "Fail" status icon for the part of our design that was either "uncoverable" or "inconclusive".

Double-click on the failed status icon to generate a waveform.

3.2 CEX Waveform and Source Tracing

VC Formal uses an CEX waveform as a debugging tool for the FuSa app. It is used to show a signal value in a “good machine” and a “faulty machine”. The analysis will fail if there is an instance where these values differ.

To start, go ahead and double-click on an ☒ icon. You should then see a generated waveform similarly shown below:


*<nWave:3> /u/.../formal/fpld1/trace_46.xml.replay.fsdb(FaultID=133)
File
Signal
View
Waveform
Tools
Window
0
0
0
x 1ns
By:
Go to:
FUSA-Pro
0
10
20
30
40
f6e
fd
fd
0
20
40
Message
VC Formal Console
*<nWave:3> trace_46.xml.replay.fsdb(FaultID=133)

FUSA-Property	
t_DUT_sdpram_i1_sdpram_i1_R_DataOut_2__st0	(0, 0) : failure
Support-Signals	
R_DataOut[11:0]	(f6e/f6a)
Clock	1
FuSa-Observations	
DataOut[7:0]	fd
DataOut[7:0]	fd
FuSa-Detections	
Error	(0/1)

Figure 3.4. Examining the failed FuSa fault in our design.

On the left in *Figure 3.4* above, we see *Support-Signals, FuSa-Observations,* and *FuSa-Detections* in green text. You can choose to look at where this fault was observed and/or detected by selecting a signal with a shaded portion. Right-click on the chosen signal in the waveform and select “Add Fault Waveform”; shown below.

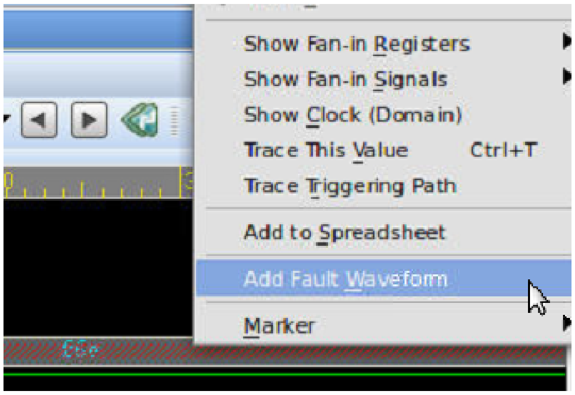


Figure 3.5. Pop-up from right-clicking on a signal.

This will add a faulty signal in order to exhibit the value causing the fault:

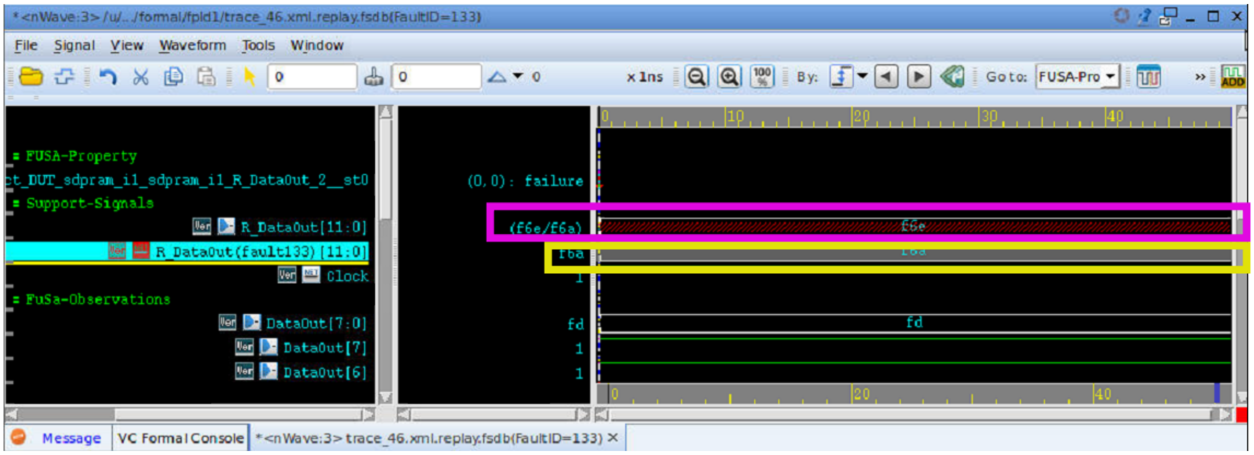

*<nWave:3> /u/.../formal/fpld1/trace_46.xml.replay.fsdb(FaultID=133)
File Signal View Waveform Tools Window
x 1ns
By:
Go to: FUSA-Pro
= FUSA-Property
t_DUT_sdpram_i1_sdpram_i1_R_DataOut_2__st0
= Support-Signals
R_DataOut[11:0]
R_DataOut(fault133)[11:0]
Clock
= FuSa-Observations
DataOut[7:0]
DataOut[7]
DataOut[6]
(0,0): failure
(f6e/f6a)
f6a
1
fd
1
1
f6e
f6a
fd
0
10
20
30
40
Message
VC Formal Console
*<nWave:3> trace_46.xml.replay.fsdb(FaultID=133)

Figure 3.6. Faulty signal added to waveform.

By tracing the source and adding this faulty signal, we are essentially showing the faulty machine signal values and comparing them to the good machine signal values. The red shaded signals are an indication of a deviation from the good machine; the initial generated signals are all good machine values (one highlighted in pink). When you add a "fault waveform" signal (highlighted in yellow), it shows the values associated with the faulty machine.

Another way to trace the source of a fault and exhibit where it was observed/detected is though a temporal flow view.

Right-click on the chosen signal in the waveform and select "Auto Trace" under "Temporal Flow View"; shown below.

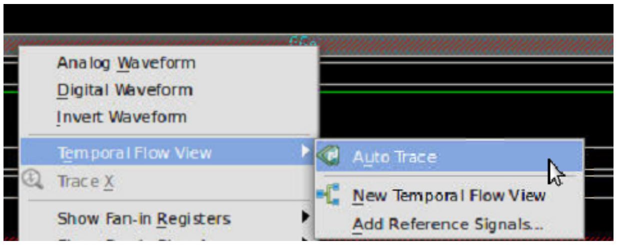
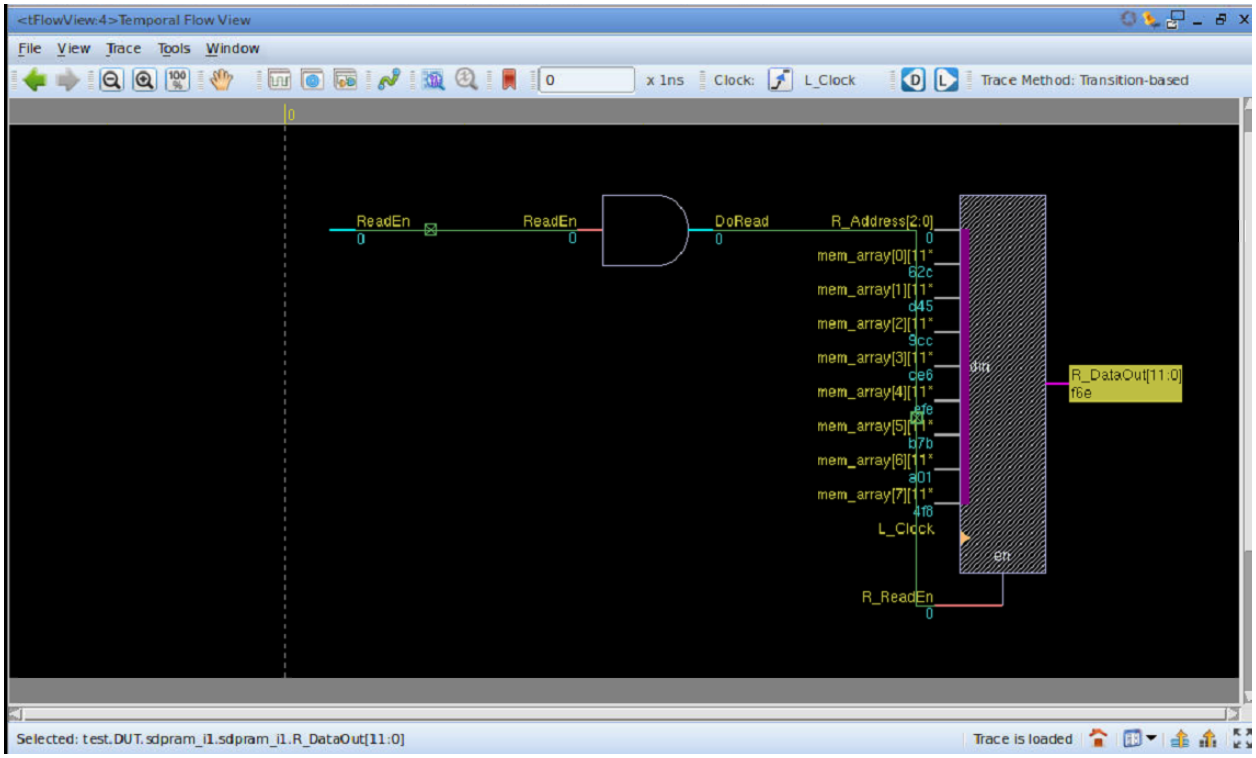


Figure 3.7. Pop-up from right-clicking on a signal.



Selected: test.DUT.sdpram_i1.sdpram_i1.R_DataOut[11:0]

Trace is loaded

Figure 3.8. Temporal Flow View of design schematic.

By tracing the source, we are essentially backtracking it to the driving signal of the output in order to find the discrepancy. Here you can examine the root-cause by double-clicking on a connection on the RTL and It will take you to the part of your code where the fault was observed/detected.

3.3 Resolving Errors

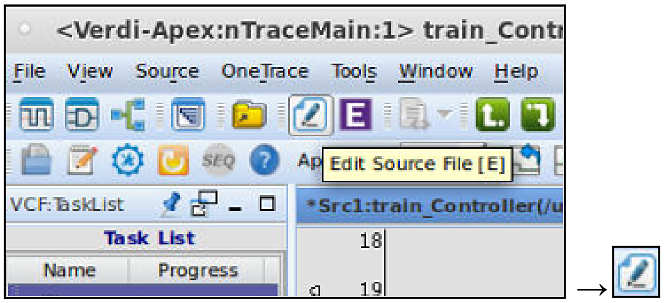

<Verdi-Apex:nTraceMain:1> train_Contr
File View Source OneTrace Tools Window Help
Ap
Edit Source File [E]
VCF:TaskList
*Src1:train_Controller(/u
Task List
Name
Progress
18
19
→

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 finalize 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.

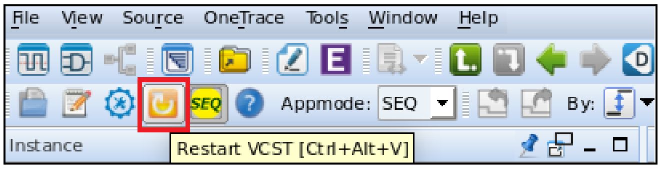
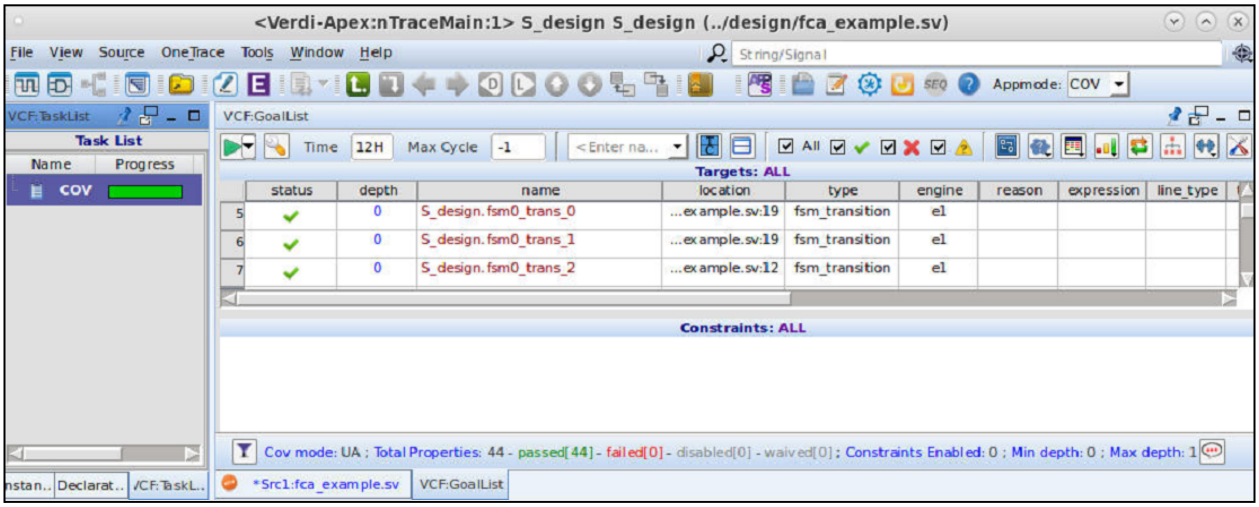


Figure 4.1. Location of the restart button in VC Formal window.

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



<Verdi-Apex:nTraceMain:1> S_design S_design (../design/fca_example.sv)

	status	depth	name	location	type	engine	reason	expression	line_type
5	✓	0	S_design.fsm0_trans_0	...example.sv:19	fsm_transition	e1			
6	✓	0	S_design.fsm0_trans_1	...example.sv:19	fsm_transition	e1			
7	✓	0	S_design.fsm0_trans_2	...example.sv:12	fsm_transition	e1			

Cov mode: UA ; Total Properties: 44 - passed[44] - failed[0] - disabled[0] - waived[0] ; Constraints Enabled: 0 ; Min depth: 0 ; Max depth: 1

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

Appendix

Description	Internal Formal Status	Status Codes
Formal No Result	FNR	NA
Structural Inconclusive	SIN	VV
Controllabiliuty Inconclusive	CIN	VV
Controllability Controllable	CCC	CC
Observability Inclonclusive	OIN	XX
Detectability Inclonclusive	DIN	XX
Observability Observed	OOB	OX
Detectability Not Detected	DND	XF
Detectability Detected	DDT	XD
Observable and Not-Detected	OND	OF
Structurally Not Observed Not Detected	SSF	UU
Structurally Not Observed	SNO	IV
Structurally Not Detected	SND	VI
Controllability Non-Controllable	CNC	UT
Observability Non-Observable	ONO	UB
Non-Observable and Detected	NOD	UB
Observable and Detected	ODT	OD
Non-Observable and Not Detected	NON	UB