VC Formal Lab
Functional Safety Verification (FuSa) App Setup and Standard Usage

Learning Objectives

In this VC Formal lab, you will use a FIFO example to learn to do the following:

- Load the FuSa fault list (SFF) into VC Formal
- Set up and compile the design
- Run interactively with Verdi GUI
- Set up clocks and resets
- Establish initial state for VC Formal
- Run VC Formal FuSa checks
- Debug VC Formal FuSa 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/FuSa/Fusa_SFF_Flow

Directory Structure	
FuSa_SFF_Flow	Lab main directory
README_VCFormal_FuSa_SFF_Flow.pdf	Lab instructions
design/	Verilog RTL code of the Device Under Test (DUT)
sff/	FuSa input SFF file
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_Formal_Docs/VC_Formal_UG.pdf

VC Formal Apps Quick References Guides:
$VC_STATIC_HOME/doc/vcst/VC_Formal_Docs/Quick_Reference_Guides/

VC Formal Apps Tcl Templates:
$VC_STATIC_HOME/doc/vcst/VC_Formal_Docs/Quick_Reference_Guides/vcf_tcl_templates/

Prepare your Environment

1. Ensure VC Formal is setup with the required licenses. You'll need licenses for VC Formal Base and VC Formal FuSa licenses to run this flow.

2. Set VC_STATIC_HOME to specify installation path. Add $VC_STATIC_HOME/bin to your $PATH

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

3. Change your working directory to run

```
% cd run
```

Now you are ready to begin the lab.

Create run.tcl for VC Formal FuSa

4. Open a new file run.tcl (any arbitrary name will also do) using an editor

```
% vi run.tcl
```

5. Use Tcl variable to switch to FUSA app mode

```
set_fml_var fml_enable_fusa_appmode true -global
set_fml_appmode FUSA
```

6. Load the fault list

```
fusa_config -sff ../sff/input.sff
```

7. Enter the command to compile design

```
read_file -top test -format verilog \
  -vcs "-sverilog -f ../design/rtl.f"
```

8. Enter clock definition

```
create_clock -period 100 {Clock}
```

9. Enter reset definition and initialize VCF setup

```
create_reset {Reset_} -sense low

sim_run -stable
sim_save_reset
```

10. Generate FuSa properties for faults that have been loaded

```
fusa_generate
```

11. Add observation and detection point(s) if not already present in the SFF file

```
fusa_observation -add {test.DUT.DataOut}
fusa_detection -add {test.DUT.Error}
```

12. Add commands to run structural analysis and analyze report

```
set_fml_var fusa_run_mode structural
check_fv -block
fusa_report
```

13. Add commands to run controllability analysis and analyze report

```
set_fml_var fusa_run_mode control
check_fv -block
fusa_report
```

14. Add commands to run observability analysis and analyze report

```
set_fml_var fusa_run_mode observe
check_fv -block
fusa_report
```

15. Add commands to run detectability analysis and analyze report

```
set_fml_var fusa_run_mode  detect
check_fv -block
fusa_report
```

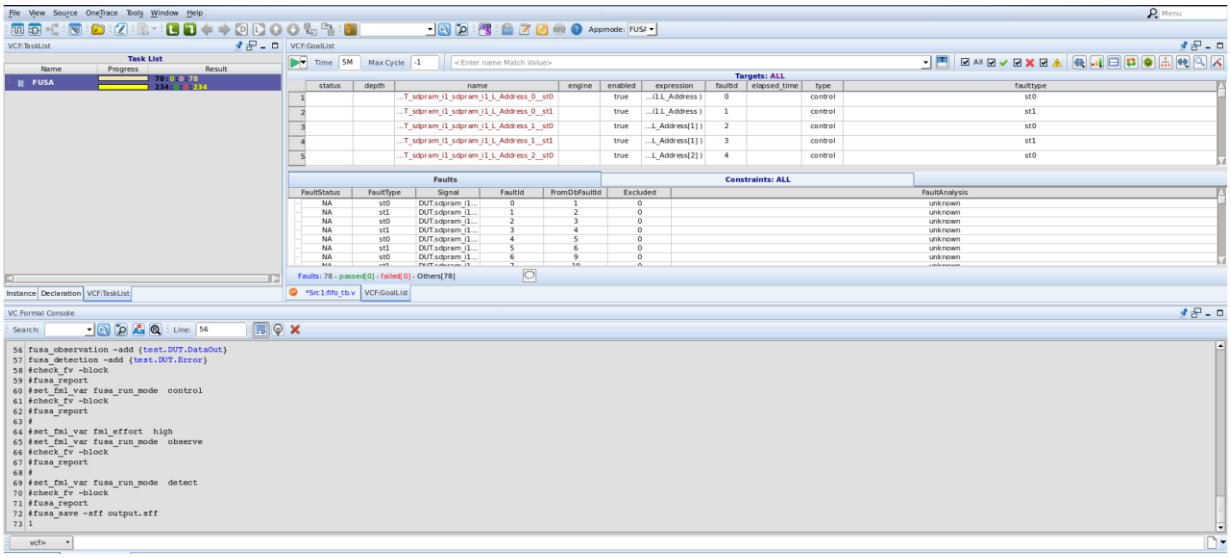
VC Formal can be run in both GUI mode and Shell mode.

Start VC Formal in Verdi GUI mode

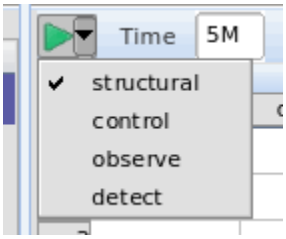
16. Run the VCF Tcl file in Verdi GUI mode

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

17. The GUI starts in VC Formal tailored for FuSa. The App mode is set to FPV by default. When the FUSA app mode variable is set, it will start the GUI in FuSa mode.



The check_fv commands specified in 17, 18, 19 & 20 can also be accessed in the GUI drop-down menu as shown below.



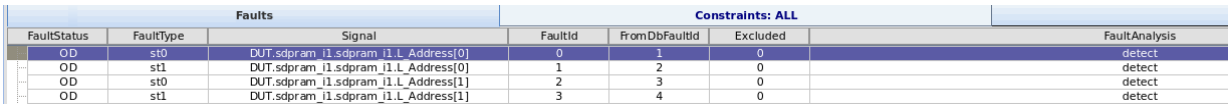
Save result and query the database to dump SFF

18. Save the result in a SFF file

```
fusa_save -sff output.sff
```

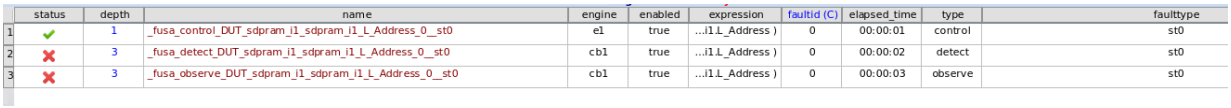
Debug Failures

19. Double-click on signal name of interest in the fault table



Faults			Constraints: ALL			
FaultStatus	FaultType	Signal	FaultId	FromDbFaultId	Excluded	FaultAnalysis
OD	st0	DUT.sdpram_i1.sdpram_i1.L_Address[0]	0	1	0	detect
OD	st1	DUT.sdpram_i1.sdpram_i1.L_Address[0]	1	2	0	detect
OD	st0	DUT.sdpram_i1.sdpram_i1.L_Address[1]	2	3	0	detect
OD	st1	DUT.sdpram_i1.sdpram_i1.L_Address[1]	3	4	0	detect

This would show the properties corresponding to the fault

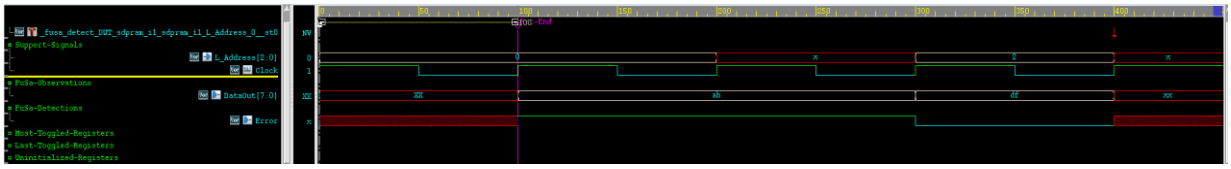


	status	depth	name	engine	enabled	expression	faultid (C)	elapsed_time	type	faulttype
1	✔	1	_fusa_control_DUT_sdpram_i1_sdpram_i1_L_Address_0__st0	e1	true	...i1.L_Address)	0	00:00:01	control	st0
2	✘	3	_fusa_detect_DUT_sdpram_i1_sdpram_i1_L_Address_0__st0	cb1	true	...i1.L_Address)	0	00:00:02	detect	st0
3	✘	3	_fusa_observe_DUT_sdpram_i1_sdpram_i1_L_Address_0__st0	cb1	true	...i1.L_Address)	0	00:00:03	observe	st0

20. Double-click on the status of the failing property corresponding to this fault.

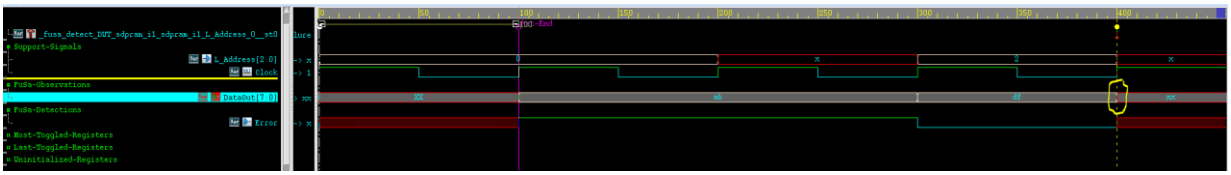
    For e.g., double-click in the status of
    "_fusa_detect_DUT_sdpram_i1_sdpram_i1_L_Address_0__st0"

    This would open a CEX as shown below:

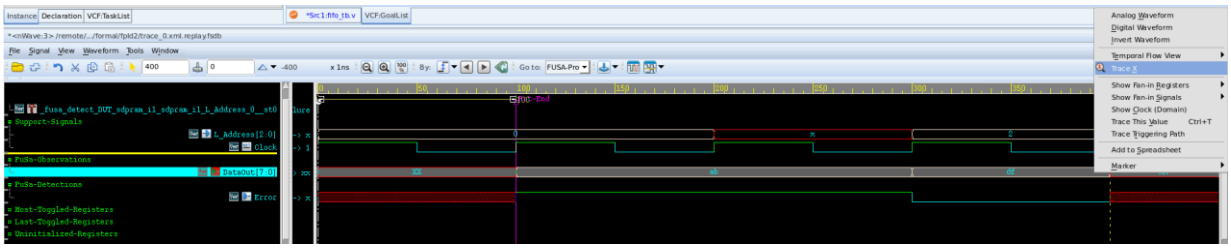


21. Since this fault was Observed and Detected (OD), we can trace either the observation or the detection points to see how this fault got observed/detected.

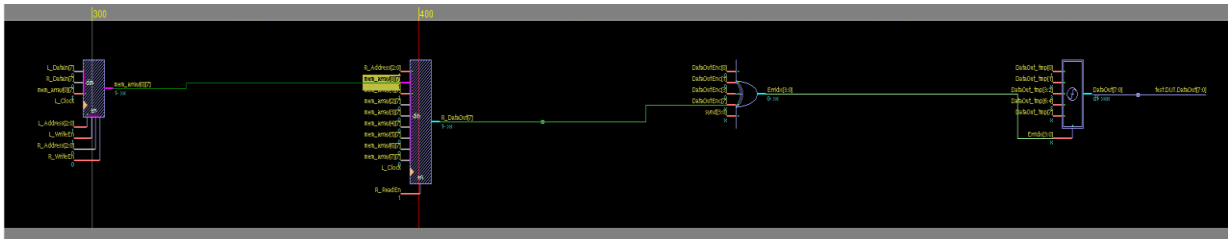
    Select DataOut[7:0] signal in the waveform. Click on the waveform to move the cursor to the point of where "X" is seen.



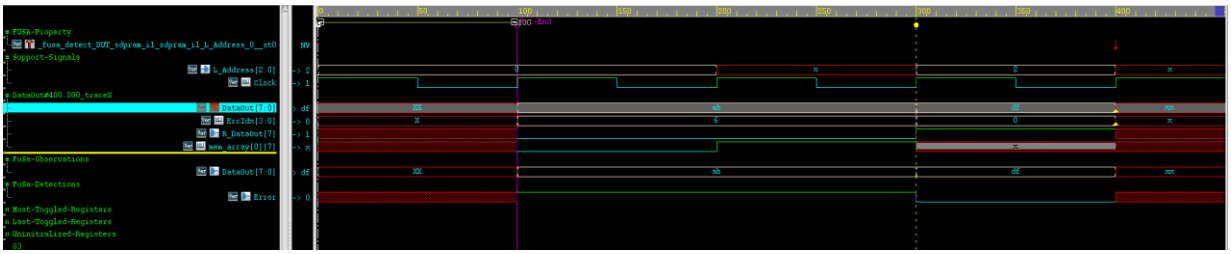
Right click on the waveform and select “Trace X”



22. The tool will trace the source and display how the fault got observed/detected at the observation/detection point using a temporal flow view as shown below in the schematic.



You can also root-cause the same driving logic in the waveform as shown below:



23. Double-click on the connection in the temporal flow view to see its root-cause in the RTL