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:

- Generate the FuSa fault database
- Load the FuSa fault database 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_FDB_Flow

Directory Structure	
FuSa_FDB_Flow	Lab main directory
README_VCFormal_FuSa_FDB_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, VC Formal FuSa, ZOIX or VCS licenses to run this flow.

2. Set VC_STATIC_HOME and ZOIXHOME to specify installation paths. Add $VC_STATIC_HOME/bin and $ZOIXHOME/bin to your $PATH

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

NOTE: ZOIXHOME is only required if you compile the design using ZOIX and it is being used to generate the fault database.

3. Set project, storage path, campaign for fault database

```
% setenv SNPS_FDB_PROJECT fifo
% setenv SNPS_FDB_STORAGE_PATH $PWD/fdb_data
% setenv SNPS_FDB_FAULT_CAMPAIGN fc1
```

4. Change your working directory to run

```
% cd run
```

Now you are ready to begin the lab.

Create Makefile for FuSa FDB Flow

5. Open the empty Makefile using an editor

```
% vi Makefile
```

6. Compile the design using ZOIX or VCS
   a. Compile using ZOIX

```
compile_zoix:
    zoix -w -f ../design/rtl.f -portfaults +fault+var \
      -sverilog $* +suppress+cell +nolibcell
```

   b. Compile using VCS

```
compile_vcs:
    vcs -sverilog -design_dump_only \
      -f ../design/rtl.f -full64
```

7. Creating a fault campaign database:

   If the design has been compiled using ZOIX

```
gen_fdb_zoix:
  ${ZOIXHOME}/bin/vc_fcc -full64 -daidir zoix.sim.daidir \
    -fault_test_coverage -sff ../sff/input.sff -dut_path test
```

   If the design has been compiled using VCS

```
  gen_fdb_vcs:
  ${VCS_HOME}/bin/vc_fcc -full64 -daidir simv.daidir \
    -fault_test_coverage -sff ../sff/input.sff -dut_path test
```

8. Run VCF Tcl file:

```
run_vcf_fusa:
  vcf -f run.tcl -verdi &
```

Create run.tcl for VC Formal FuSa

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

```
% vi run.tcl
```

10. Use Tcl variable to switch to FUSA app mode

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

11. Load the fault list

```
fusa_config -fdb_campaign $::env(SNPS_FDB_FAULT_CAMPAIGN) \
  -dut_path test
```

12. Enter the command to compile design

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

13. Enter clock definition

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

14. Enter reset definition and initialize VCF setup

```
create_reset {Reset_} -sense low

sim_run -stable
sim_save_reset
```

15. Generate FuSa properties for faults that have been loaded

```
fusa_generate
```

16. 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}
```

17. Add commands to run structural analysis and analyze report

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

18. Add commands to run controllability analysis and analyze report

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

19. Add commands to run observability analysis and analyze report

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

20. 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.

Run FuSa FDB Flow from Makefile

21. Compile the design:
    a. Compile using ZOIX

```
make compile_zoix
```

    b. Compile using VCS

```
make compile_vcs
```

22. Create fault campaign database
    a. If design has been compiled using ZOIX

```
make gen_fdb_zoix
```

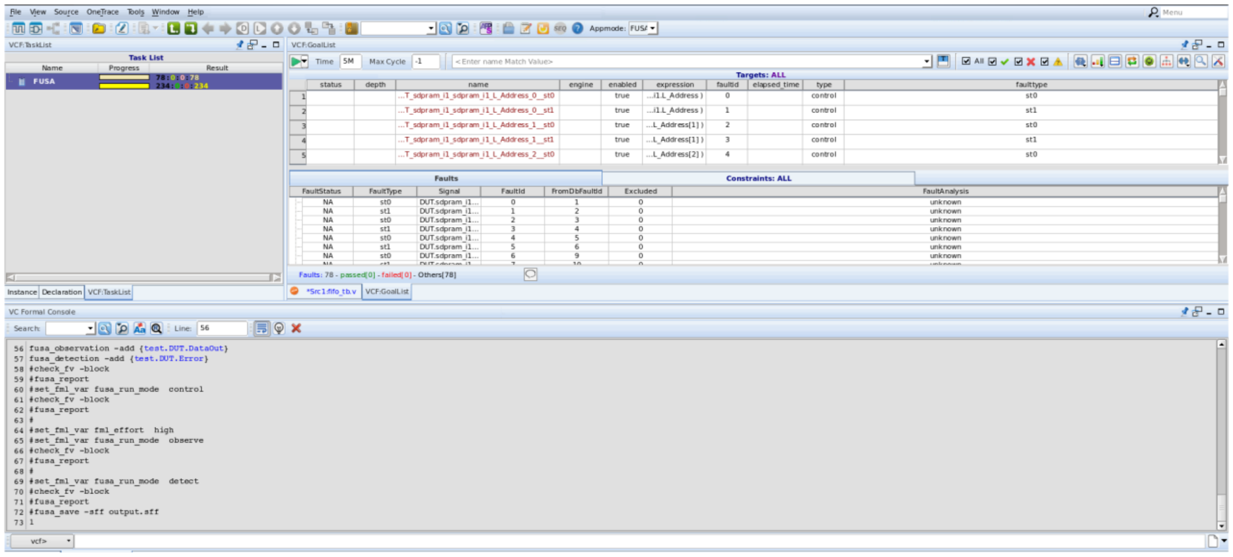
    b. If design has been compiled using VCS

```
make gen_fdb_vcs
```

23. Run the VCF Tcl file. This will launch VC Formal in GUI mode

```
make run_vcf_fusa
```

24. 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.



Targets: ALL

	status	depth	name	engine	enabled	expression	faultid	elapsed_time	type	faulttype
1			...T_sdpram_i1_sdpram_i1_L_Address_0__st0		true	...i1.L_Address)	0		control	st0
2			...T_sdpram_i1_sdpram_i1_L_Address_0__st1		true	...i1.L_Address)	1		control	st1
3			...T_sdpram_i1_sdpram_i1_L_Address_1__st0		true	...L_Address[1])	2		control	st0
4			...T_sdpram_i1_sdpram_i1_L_Address_1__st1		true	...L_Address[1])	3		control	st1
5			...T_sdpram_i1_sdpram_i1_L_Address_2__st0		true	...L_Address[2])	4		control	st0

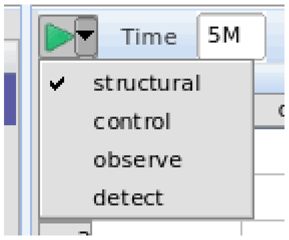
Faults | Constraints: ALL

FaultStatus	FaultType	Signal	FaultId	FromDbFaultId	Excluded	FaultAnalysis
NA	st0	DUT.sdpram_i1...	0	1	0	unknown
NA	st1	DUT.sdpram_i1...	1	2	0	unknown
NA	st0	DUT.sdpram_i1...	2	3	0	unknown
NA	st1	DUT.sdpram_i1...	3	4	0	unknown
NA	st0	DUT.sdpram_i1...	4	5	0	unknown
NA	st1	DUT.sdpram_i1...	5	6	0	unknown
NA	st0	DUT.sdpram_i1...	6	9	0	unknown

Faults: 78 - passed[0] - failed[0] - Others[78]

```
fusa_observation -add {test.DUT.DataOut}
fusa_detection -add {test.DUT.Error}
#check_fv -block
#fusa_report
#set_fml_var fusa_run_mode  control
#check_fv -block
#fusa_report
#
#set_fml_var fml_effort  high
#set_fml_var fusa_run_mode  observe
#check_fv -block
#fusa_report
#
#set_fml_var fusa_run_mode  detect
#check_fv -block
#fusa_report
#fusa_save -sff output.sff
1
```

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

25. Save the result in a SFF file

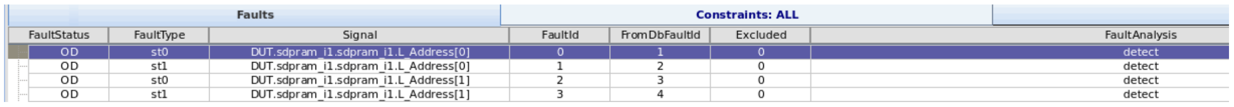
```
fusa_save
```

26. Query the database and dump SFF

```
exec vc_fdb_report \
-campaign $::env(SNPS_FDB_FAULT_CAMPAIGN)  -report output.sff
```

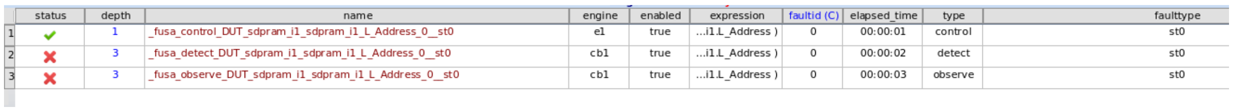
Debug Failures

27. 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

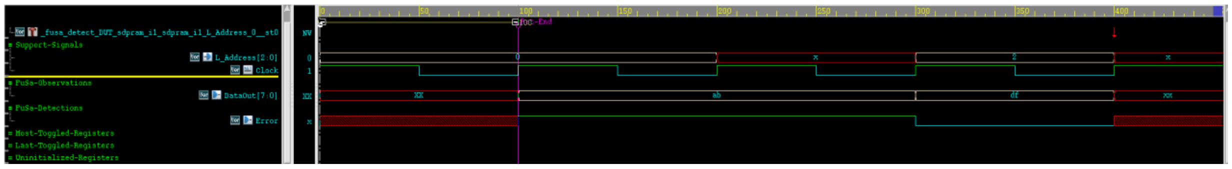


	status	depth	name	engine	enabled	expression	faultid (C)	elapsed_time	type	faulttype
1	✓	1	_fusa_control_DUT_sdpram_i1_sdpram_i1_L_Address_0__st0	e1	true	...i1.L_Address)	0	00:00:01	control	st0
2	✗	3	_fusa_detect_DUT_sdpram_i1_sdpram_i1_L_Address_0__st0	cb1	true	...i1.L_Address)	0	00:00:02	detect	st0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

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

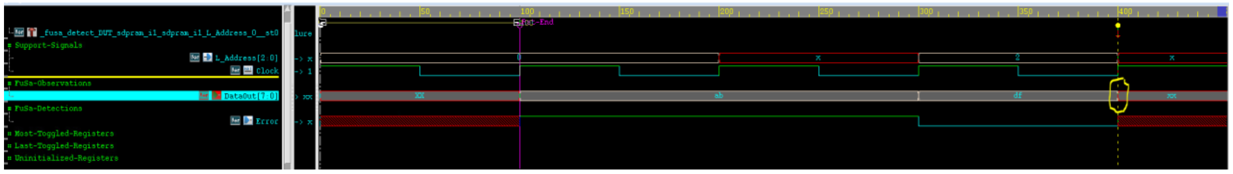
    For e.g., double-click in the status of "_fusa_detect_DUT_sdpram_i1_sdpram_i1_L_Address_0__st0"

    This would open a CEX as shown below:

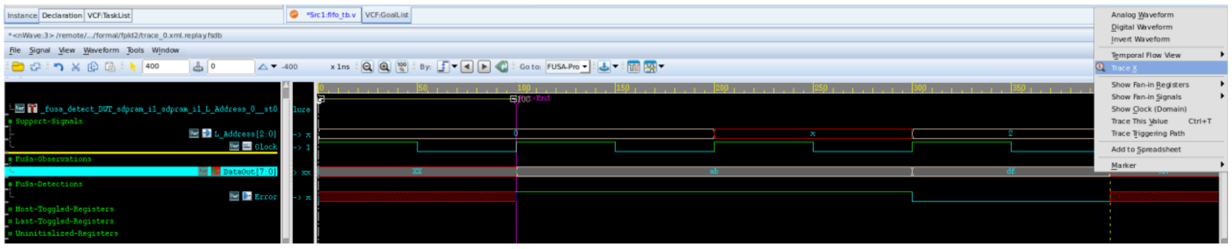


29. Since this fault was Observed and Detected (OD), we can trace either the observation or the detection points to see how this fault got observed/detected.

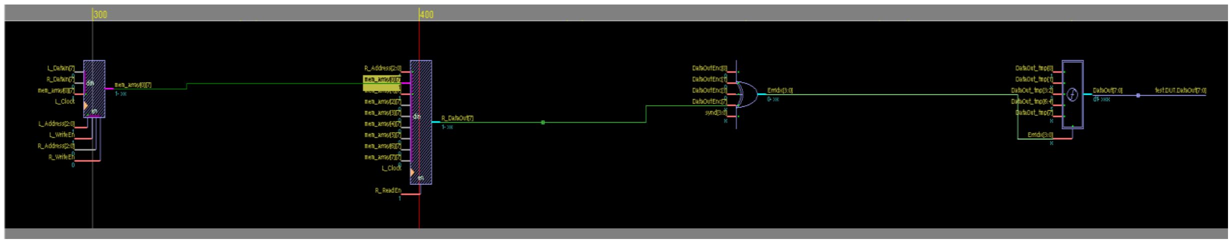
    Select DataOut[7:0] signal in the waveform. Click on the waveform to move the cursor to the point of where "X" is seen.



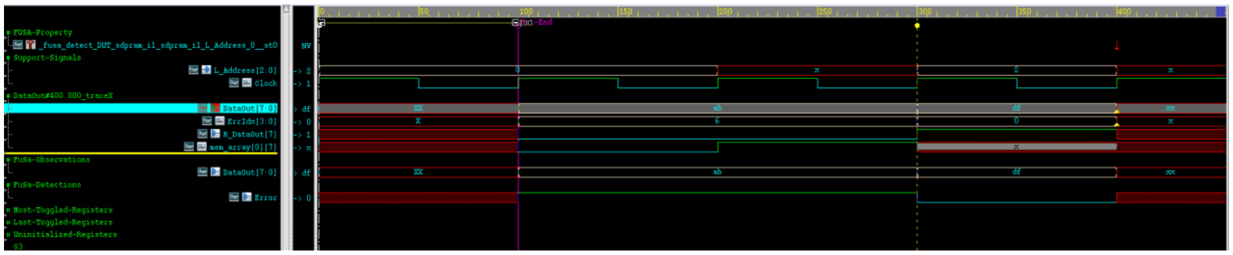
    Right click on the waveform and select "Trace X"



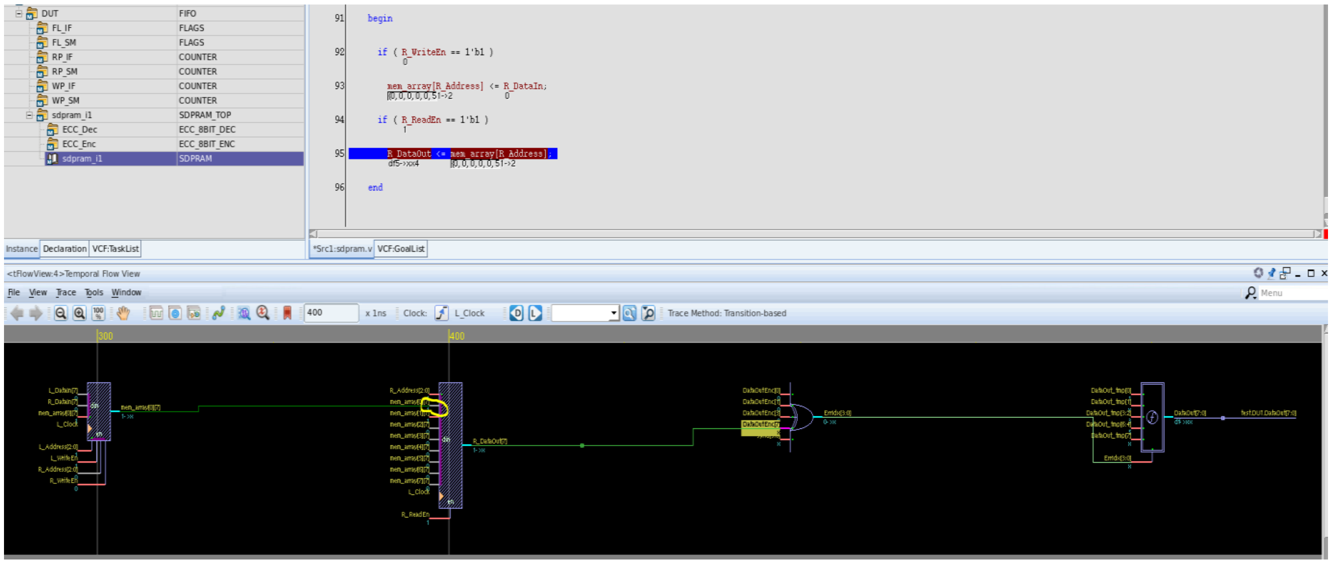
30. The tool will trace the source and display how the fault got observed/detected at the observation/detection point using a temporal flow view as shown below in the schematic.



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



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



DUT	FIFO
FL_IF	FLAGS
FL_SM	FLAGS
RP_IF	COUNTER
RP_SM	COUNTER
WP_IF	COUNTER
WP_SM	COUNTER
sdpram_i1	SDPRAM_TOP
ECC_Dec	ECC_8BIT_DEC
ECC_Enc	ECC_8BIT_ENC
sdpram_i1	SDPRAM

```
begin

  if ( R_WriteEn == 1'b1 )

    mem_array[R_Address] <= R_DataIn;

  if ( R_ReadEn == 1'b1 )

    R_DataOut <= mem_array[R_Address];

end
```