VC Formal Lab
Formal Property Verification (FPV) App
Spec to Signoff

Learning Objectives

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

Step 1: FPV

- Read the spec provided in this document
- Write SVA as per suggestions provided in checker file
- Setup environment for FPV and verify the design using assertions you have created
- Determine root cause and fix CEXs if any

Step 2: Formal Signoff

- Re-visit your FPV setup and instrument Coverage & Faults for analysis
- Re-run FPV and compute Formal Core to assess code coverage of proven assertions
- Root cause Formal Core holes and add assertions if needed
- Exclude and save manual exclusions
- Invoke and Run FTA
- Root cause Non-Activated, Non-Formalcore and Non-Detected faults and add assertions if needed

Lab Duration: 120minutes

PRE-REQUISITES:
Familiarity with the SystemVerilog Assertion (SVA) language 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/FPV/Spec_to_Signoff/

Directory Structure	
Spec_to_Signoff	Lab main directory
README_VCFormal_Spec_to_Signoff.pdf	Lab instructions
design/	Verilog RTL code of the Device Under Test (DUT)
sva/	SVA properties to check functionality of the DUT
run/	Run directory
solution/	Solution directory

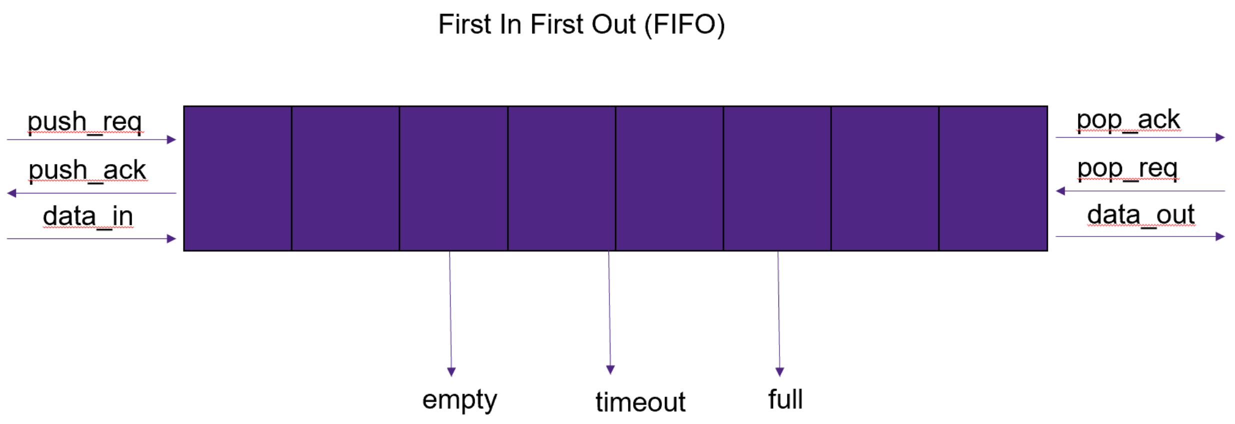
Resources

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/



FIFO Spec

- Design should write data and read data in a First In First Out order and handle up to 16 transfers.
- When the design has no more free space the full flag should be raised
- When the full flag is high, push_ack must be low.
- When push_req is high and push_ack low, push_req must be kept high and data_in stable.
- When the design has no data inside the empty flag should be raised
- When the empty flag is high, **pop_ack** must be low.
- When **pop_req** is high and **pop_ack** low, **pop_req** must be kept high.
- Data comes out of the pop interface 1 cycle after **pop_req** and **pop_ack** are high together.
- When a gap of 5 or more cycles appears between **push_req**s timeout should be set

Step 1: FPV

Prepare your Formal Environment

1. Load VC Formal and Verdi to set up the tool and required licenses

```
% module load vcstatic
% module load verdi
```

2. Change your working directory to sva

```
$ cd sva
```

Now you are ready to begin the lab.

Create FIFO Checker from Spec

1. Open the “fifo_sva.sv” file. There are five questions embedded as comments, e.g.

```
// Q1. Complete the assumes described in the labels below
//
// In absence of push_ack, data in and push request
// is held stable
 am_push_req_stable_when_no_ack:
   assume property (`clk_rst (push_req && !push_ack) |=>

 am_data_in_stable_when_no_ack:
   assume property (`clk_rst

// If pop_req is asserted and pop_ack is low,
// pop_req should   be held stable
 am_pop_req_stable_when_no_ack:
   assume property (`clk_rst
```

Complete questions Q1, Q2 & Q3 in the file “fifo_sva.sv” by writing assertion and assume expressions as **instructed in the comments above**.

Verify Design using VC Formal

2. Verify the FIFO design using the checker you have written.

```
% cd ../run
% vcf -f fifo_signoff.tcl -verdi &
```

Fix any compilation warnings and errors in your checker and run proof of all assertions in your checker. Debug and fix any CEX/Failures in the design and checkers you have written.

Check Data Integrity of FIFO

3. Now try Part B of the lab. This has questions Q4 & Q5 to complete. Pass the "`+define+LAB_PART_B`" option to the "`read_file`" command and re-run the setup.

```
read_file -top $top -format sverilog -sva \
     -vcs {-f ../scripts/filelist.flist ../sva/fifo_sva.sv\
     +define+LAB_PART_B}
```

Fix any compilation warnings and errors in your checker and run proof of all assertions in your checker. Debug and fix any CEX/Failures in the design and checkers you have written.

Step 2: Signoff

Setup Design for Coverage and FTA

Now that you have completed writing assertions and proved them to ensure there are no failures or unreachable covers, setup the design for VC Formal Signoff. Open the "fifo_signoff.tcl" file and follow the steps below.

4. Set Signoff configuration command as shown below. Add the below command before "`read_file`" command in "fifo_signoff.tcl."

```
signoff_config -type all
```

5. Specify the read_file command as shown below.

```
read_file -top $top -format sverilog -sva \
   -vcs {-f ../scripts/filelist.flist ../sva/fifo_sva.sv \
    +define+LAB_PART_B}
```

Configure Fault Injection for FTA

FTA uses Synopsys Certitude to instrument faults in the design. You can specify modules where faults need to be injected by using the "`fta_init`" command.

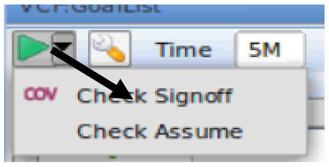
6. Add the below command before "`read_file`" command.

```
fta_init -fast_sanity -scope {fifo}
```

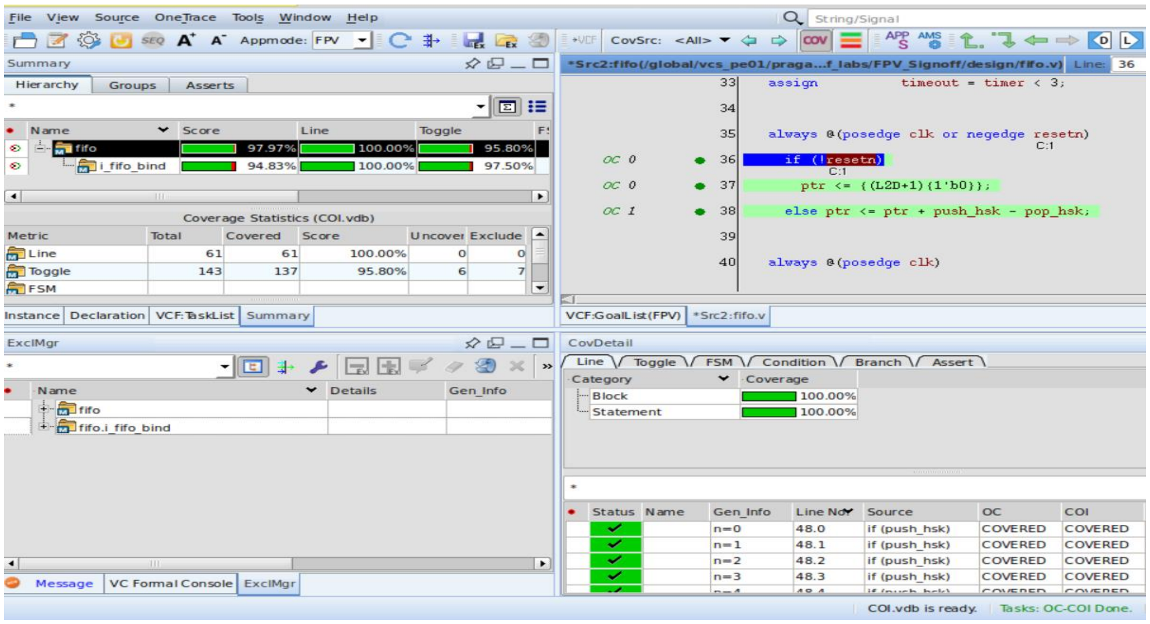
7. After adding commands from previous step, run the setup again.

Formal Signoff – Low Effort – Cone of Influence (COI) Analysis

8. Once the tool has finished running FPV, select "Check Signoff" option present in the drop down menu of the play button.



This will run Cone-of-Influence (COI) + Over Constraint (OC) analysis and exclude any cover items which are Unreachable in the design.


File View Source OneTrace Tools Window Help
Appmode: FPV
CovSrc: <All>
String/Signal
Summary
Hierarchy Groups Asserts
Instance Declaration VCF:TaskList Summary
ExclMgr
VCF:GoalList(FPV) *Src2:fifo.v
CovDetail
Line Toggle FSM Condition Branch Assert
Message VC Formal Console ExclMgr

Name	Score	Line	Toggle
fifo	97.97%	100.00%	95.80%
i_fifo_bind	94.83%	100.00%	97.50%

Coverage Statistics (COI.vdb)

Metric	Total	Covered	Score	Uncover	Exclude
Line	61	61	100.00%	0	0
Toggle	143	137	95.80%	6	7
FSM					

Name	Details	Gen_Info
fifo		
fifo.i_fifo_bind		

*Src2:fifo(/global/vcs_pe01/praga...f_labs/FPV_Signoff/design/fifo.v) Line: 36

```
         33  assign          timeout = timer < 3;
         34
         35  always @(posedge clk or negedge resetn)
                                             C:1
OC 0     36    if (!resetn)
                    C:1
OC 0     37      ptr <= {(L2D+1){1'b0}};
OC 1     38    else ptr <= ptr + push_hsk - pop_hsk;
         39
         40  always @(posedge clk)
```

Category	Coverage
Block	100.00%
Statement	100.00%

Status	Name	Gen_Info	Line No	Source	OC	COI
✓		n=0	48.0	if (push_hsk)	COVERED	COVERED
✓		n=1	48.1	if (push_hsk)	COVERED	COVERED
✓		n=2	48.2	if (push_hsk)	COVERED	COVERED
✓		n=3	48.3	if (push_hsk)	COVERED	COVERED

COI.vdb is ready. Tasks: OC-COI Done.

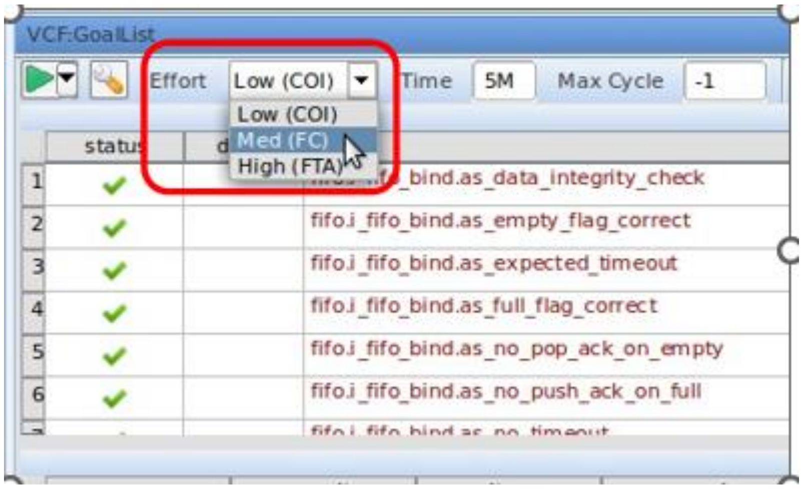
The tool will show the COI + OC coverage highlighted in the above image as 97.97%. The other highlighted red box in CovSrc window indicates the covered or uncovered status from different analysis. By default, OC is done in each effort level.

You can also run low effort or COI analysis from the VC Formal Console using the command below.

```
signoff_check -effort low -time 5M -signoff_bound -1 \
     -signoff_dashboard
```

Formal Signoff – Med Effort – Formal Core (FC) Analysis

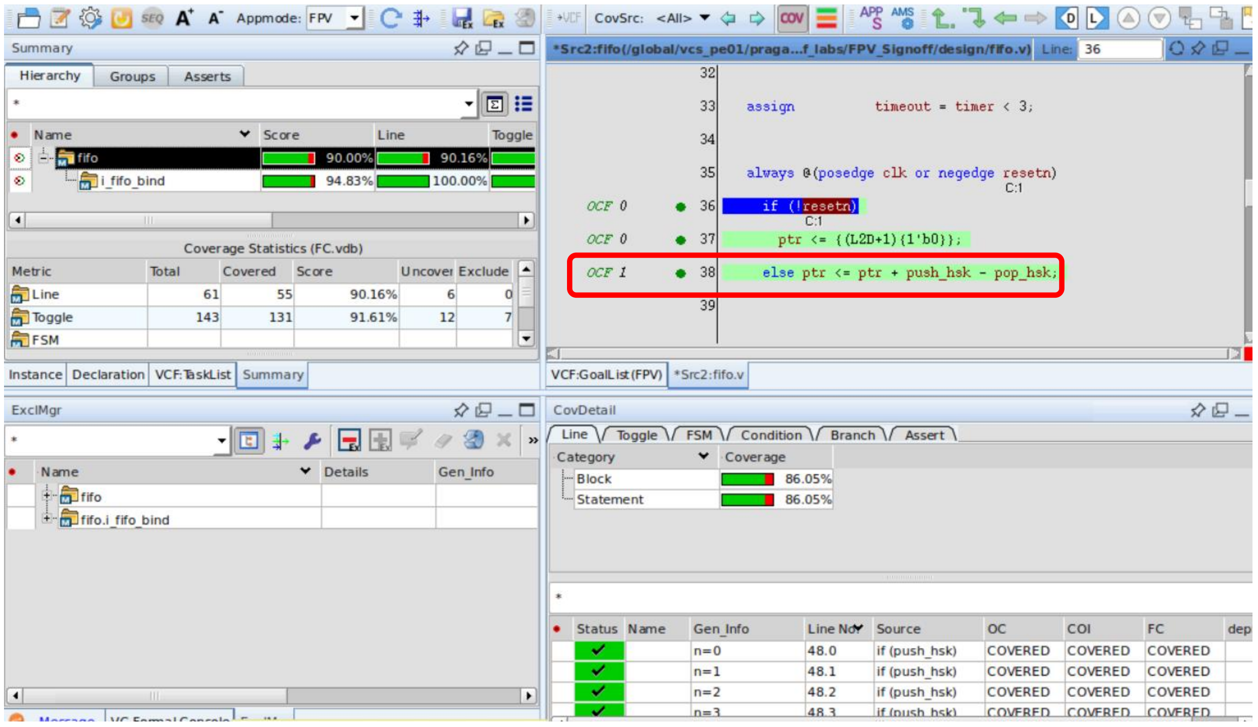
9. Once effort Low has been run, you can run Formal Core analysis with effort Med to increase the coverage granularity of the verification performed. Select "Med (FC)" effort as shown below and run "Check Signoff"



Effort: Low (COI) → Med (FC) | Time: 5M | Max Cycle: -1

	status	name
1	✓	…bind.as_data_integrity_check
2	✓	fifo.i_fifo_bind.as_empty_flag_correct
3	✓	fifo.i_fifo_bind.as_expected_timeout
4	✓	fifo.i_fifo_bind.as_full_flag_correct
5	✓	fifo.i_fifo_bind.as_no_pop_ack_on_empty
6	✓	fifo.i_fifo_bind.as_no_push_ack_on_full
		fifo.i_fifo_bind.as_no_timeout

10. Once Formal Core is computed, it will give you a pop-up asking if you should load the FC coverage database. Click "Ok" to load the Formal Core coverage %. In this case, the FC coverage is computed as 90%.



Name	Score	Line	Toggle
fifo	90.00%	90.16%	
i_fifo_bind	94.83%	100.00%	

Coverage Statistics (FC.vdb)

Metric	Total	Covered	Score	Uncover	Exclude
Line	61	55	90.16%	6	0
Toggle	143	131	91.61%	12	7
FSM					

*Src2:fifo(/global/vcs_pe01/praga...f_labs/FPV_Signoff/design/fifo.v) Line: 36

```
           assign          timeout = timer < 3;

           always @(posedge clk or negedge resetn)
                                               C:1
OCF 0   ●    if (!resetn)
                 C:1
OCF 0   ●      ptr <= {(L2D+1){1'b0}};
OCF 1   ●    else ptr <= ptr + push_hsk - pop_hsk;
```

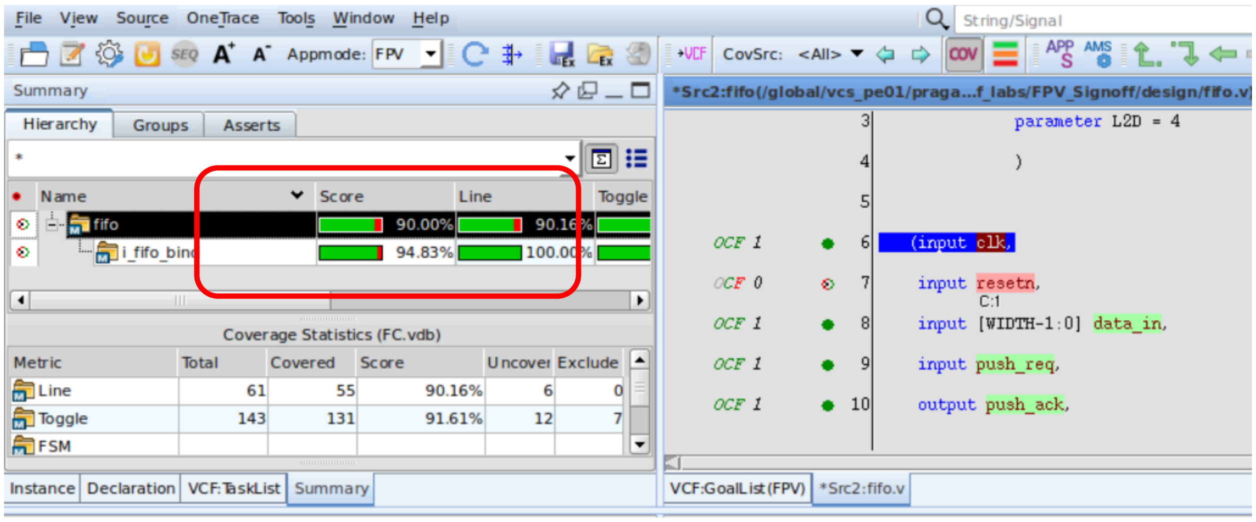
Name	Details	Gen_Info
fifo		
fifo.i_fifo_bind		

Category	Coverage
Block	86.05%
Statement	86.05%

Status	Name	Gen_Info	Line No	Source	OC	COI	FC	dep
✓		n=0	48.0	if (push_hsk)	COVERED	COVERED	COVERED	
✓		n=1	48.1	if (push_hsk)	COVERED	COVERED	COVERED	
✓		n=2	48.2	if (push_hsk)	COVERED	COVERED	COVERED	
✓		n=3	48.3	if (push_hsk)	COVERED	COVERED	COVERED	

Here, in CovSrc window, the highlighted red box shows that the line number 38 is covered in both OC, COI, and FC.

11. In the Source Browser of the Signoff Dashboard window, you can see code coverage in the Formal Core of the assertions that have been proven.
    a. Green → In the Formal Core
    b. Red → Outside the Formal Core


File View Source OneTrace Tools Window Help
String/Signal
Appmode: FPV
CovSrc: <All>
Summary
Hierarchy Groups Asserts
Instance Declaration VCF:TaskList Summary
VCF:GoalList(FPV) *Src2:fifo.v

*Src2:fifo(/global/vcs_pe01/praga...f_labs/FPV_Signoff/design/fifo.v)

Name	Score	Line	Toggle
fifo	90.00%	90.16%	
i_fifo_bin	94.83%	100.00%	

Coverage Statistics (FC.vdb)

Metric	Total	Covered	Score	Uncover	Exclude
Line	61	55	90.16%	6	0
Toggle	143	131	91.61%	12	7
FSM					

```
                parameter L2D = 4
                )

OCF 1   ●      (input clk,
OCF 0          input resetn,
                     C:1
OCF 1   ●      input [WIDTH-1:0] data_in,
OCF 1   ●      input push_req,
OCF 1   ●      output push_ack,
```

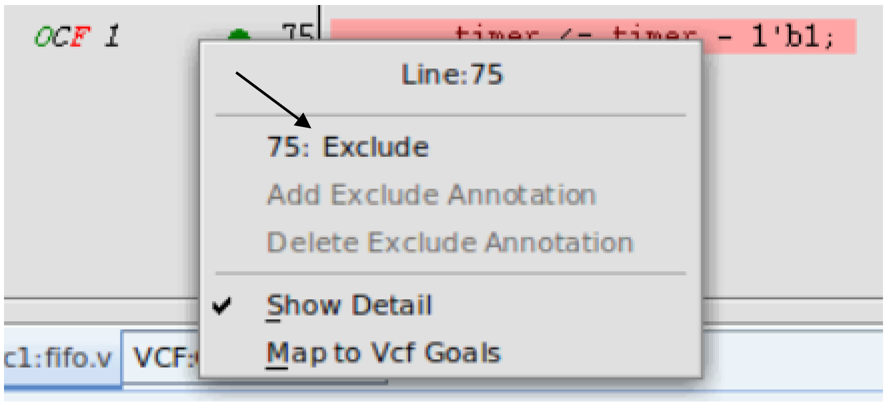
Code coverage outside the Formal Core means a portion of the code is not being tested. Make sure this is justified. If not, please add more assertions to achieve 100% Formal Core coverage.

You can also run med effort or FC analysis from the VC Formal Console using the command below.

```
signoff_check  -effort med -time 5M -signoff_bound -1 \
 -signoff_dashboard
```

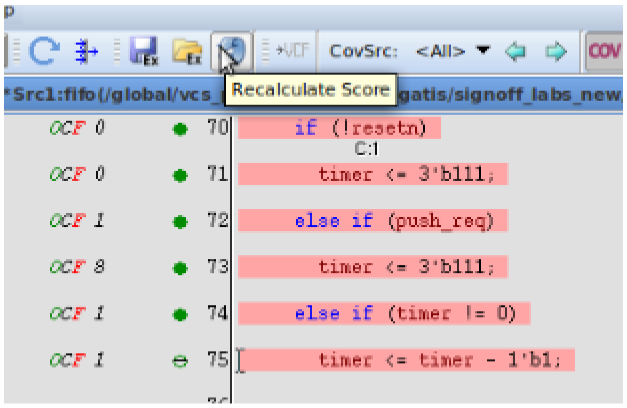
Exclude and save manual exclusion

12. To mark a goal as excluded, Right Click on the respective line. and select the “Exclude” option.



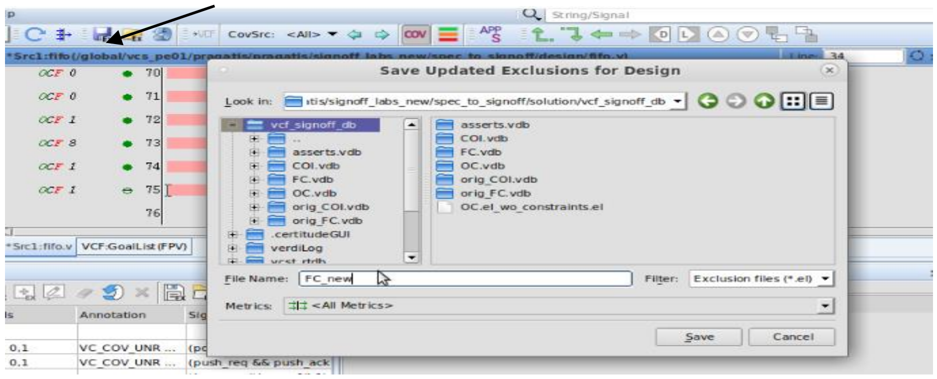
```
OCF 1    75    timer <= timer - 1'b1;
```

13. Once the goal is excluded, recalculate the score to mark the exclusion.



```
OCF 0    70    if (!resetn)
               C:1
OCF 0    71    timer <= 3'b111;
OCF 1    72    else if (push_req)
OCF 8    73    timer <= 3'b111;
OCF 1    74    else if (timer != 0)
OCF 1    75    timer <= timer - 1'b1;
```

14. Once the coverage score is recalculated, save the exclusion file as directed.


String/Signal
CovSrc: <All>
COV
*Src1:fifo(/global/vcs_pe01/pragatis/pragatis/signoff_labs_new/spec_to_signoff/design/fifo.v)
Line: 34
*Src1:fifo.v
VCF:GoalList(FPV)
Annotation
Save Updated Exclusions for Design
Save
Cancel

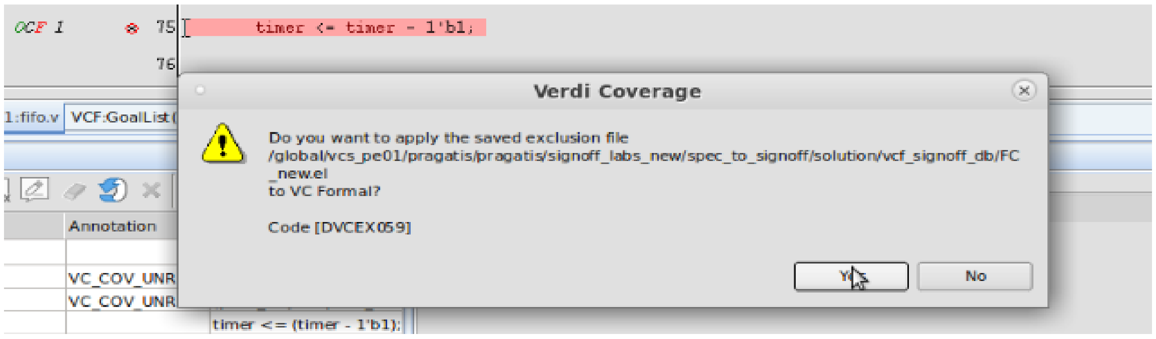
Look in: ›tis/signoff_labs_new/spec_to_signoff/solution/vcf_signoff_db

```
vcf_signoff_db          asserts.vdb
  ..                    COI.vdb
  asserts.vdb           FC.vdb
  COI.vdb               OC.vdb
  FC.vdb                orig_COI.vdb
  OC.vdb                orig_FC.vdb
  orig_COI.vdb          OC.el_wo_constraints.el
  orig_FC.vdb
.certitudeGUI
verdiLog
vcst_rtdb
```

File Name: FC_new
Filter: Exclusion files (*.el)
Metrics: <All Metrics>

```
0,1   VC_COV_UNR ...   (pc
0,1   VC_COV_UNR ...   (push_req && push_ack
```

15. Once the exclusion is saved, tool will pop-up with the message if you need to apply the saved exclusion or not. Click “Yes” to apply it.



```
OCF 1      75  timer <= timer - 1'b1;
           76
```

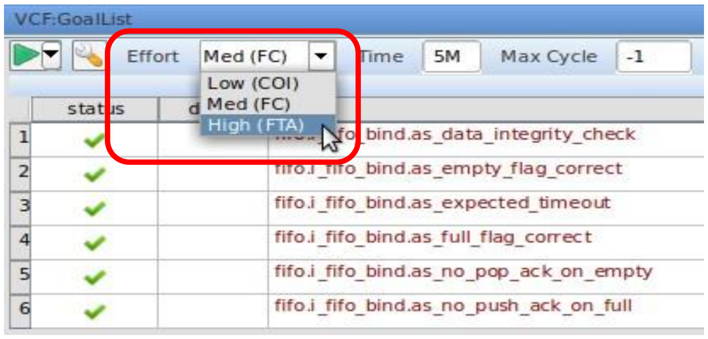
Verdi Coverage

Do you want to apply the saved exclusion file /global/vcs_pe01/pragatis/pragatis/signoff_labs_new/spec_to_signoff/solution/vcf_signoff_db/FC_new.el to VC Formal?

Code [DVCEX059]

Formal Signoff – High Effort – Formal Testbench Analyzer (FTA)

16. To further increase granularity and confidence in Signoff, you can invoke FTA by selecting effort High in VC Formal GUI.

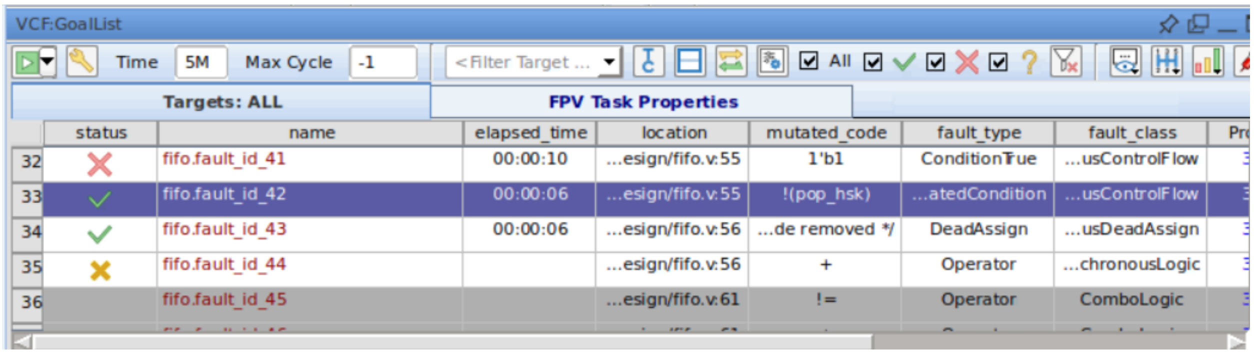


VCF:GoalList

Effort: Med (FC) — Low (COI) / Med (FC) / High (FTA); Time: 5M; Max Cycle: -1

	status	name
1	✓	fifo.i_fifo_bind.as_data_integrity_check
2	✓	fifo.i_fifo_bind.as_empty_flag_correct
3	✓	fifo.i_fifo_bind.as_expected_timeout
4	✓	fifo.i_fifo_bind.as_full_flag_correct
5	✓	fifo.i_fifo_bind.as_no_pop_ack_on_empty
6	✓	fifo.i_fifo_bind.as_no_push_ack_on_full

17. This will launch the FTA GUI as shown below. You could get 50-70 faults depending on how you have implemented the SVA and the auxiliary RTL. So please do not worry about the exact number shown here in the screenshot.



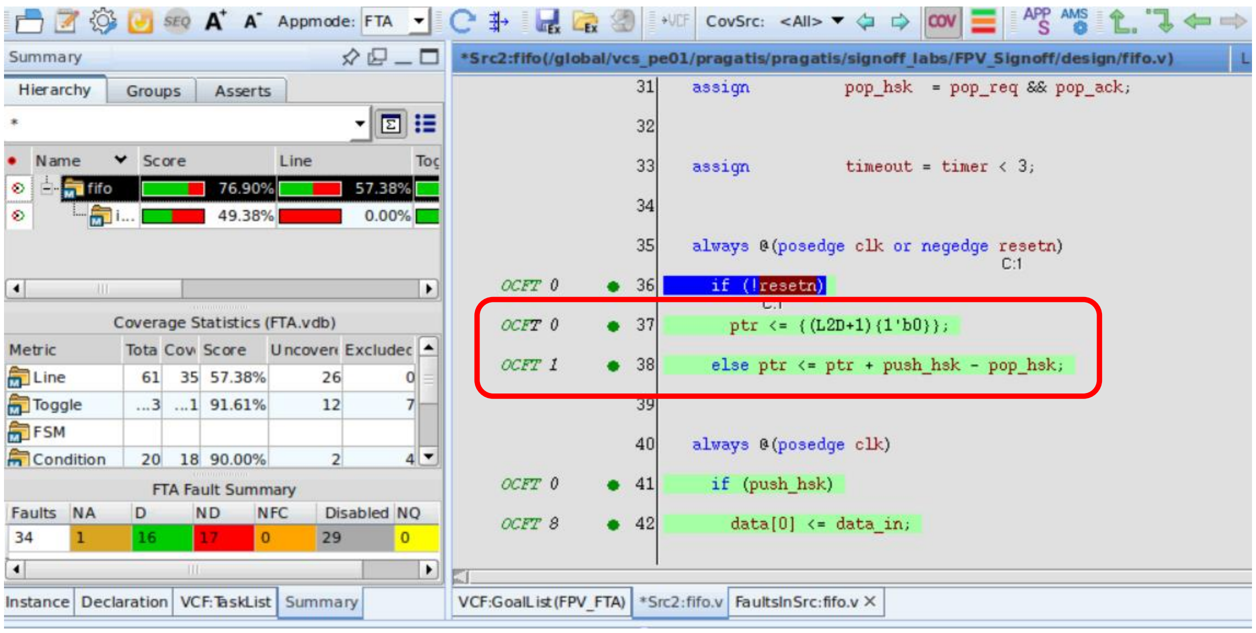
VCF:GoalList

Time: 5M; Max Cycle: -1

Targets: ALL | FPV Task Properties

	status	name	elapsed_time	location	mutated_code	fault_type	fault_class	Pr
32	✗	fifo.fault_id_41	00:00:10	...esign/fifo.v:55	1'b1	ConditionTrue	...usControlFlow	
33	✓	fifo.fault_id_42	00:00:06	...esign/fifo.v:55	!(pop_hsk)	...atedCondition	...usControlFlow	
34	✓	fifo.fault_id_43	00:00:06	...esign/fifo.v:56	...de removed */	DeadAssign	...usDeadAssign	
35	✗	fifo.fault_id_44		...esign/fifo.v:56	+	Operator	...chronousLogic	
36		fifo.fault_id_45		...esign/fifo.v:61	!=	Operator	ComboLogic	

18. Wait for the FTA run to finish. Once the run is finished the tool automatically loads the FTA coverage in the GUI as shown below. The tool also opens a Fault Summary View as shown below to list the outcome of the FTA run.


Appmode: FTA

CovSrc: <All>

Summary

Hierarchy | Groups | Asserts

Name	Score	Line
fifo	76.90%	57.38%
i...	49.38%	0.00%

Coverage Statistics (FTA.vdb)

Metric	Tota	Cov	Score	Uncover	Excluded
Line	61	35	57.38%	26	0
Toggle	...3	...1	91.61%	12	7
FSM					
Condition	20	18	90.00%	2	4

FTA Fault Summary

Faults	NA	D	ND	NFC	Disabled	NQ
34	1	16	17	0	29	0

Instance | Declaration | VCF:TaskList | Summary

*Src2:fifo(/global/vcs_pe01/pragatis/pragatis/signoff_labs/FPV_Signoff/design/fifo.v)

```
          31  assign          pop_hsk  = pop_req && pop_ack;
          32
          33  assign          timeout = timer < 3;
          34
          35  always @(posedge clk or negedge resetn)
                                                   C:1
OCFT 0  ● 36    if (!resetn)
OCFT 0  ● 37      ptr <= {(L2D+1){1'b0}};
OCFT 1  ● 38    else ptr <= ptr + push_hsk - pop_hsk;
          39
          40  always @(posedge clk)
OCFT 0  ● 41    if (push_hsk)
OCFT 8  ● 42      data[0] <= data_in;
```

VCF:GoalList(FPV_FTA) | *Src2:fifo.v | FaultsInSrc:fifo.v ✕

Here, in CovSrc window, the highlighted red box shows that the line 38 is covered in OC, COI, FC, and FTA. While line 37 doesn't have a fault instrumented.

You can also run high effort or FTA analysis from the VC Formal Console using the command below:

```
signoff_check -effort high -time 5M -signoff_bound -1 \
   -signoff_dashboard
```

Non-Activated means faults which are outside the COI
Non-Formalcore means faults which are outside formal core but inside COI.
Non-Detected faults means these faults were not detected by the assertions written in the code.

Overall, it shows that there are portions of the code which are not being tested and point to the inefficacy of your assertions and Formal testbench. All these faults should be justified. If not, please add/edit assertions to achieve 0% Non-Activated, Non-Formalcore and Non-Detected faults.

From a Code Coverage perspective, the goal is 100%. Please review the Non-Activated (yellow), Non-Formalcore (orange) and Non-Detected (red) faults and ensure they have been accounted for in the Source Browser.