Verification Continuum^{TM}

VC Formal Signoff Methodology Guide

Version T-2022.06-SP1, September 2022

SYNOPSYS®

Copyright Notice and Proprietary Information

© 2022 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at
http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

1 Contents

2 Introduction

The VC Formal signoff methodology guide is intended to help navigate through the complex world of FPV signoff. The document outlines the best practices one should follow for a successful FPV signoff. It also provides performance considerations and identifies common pitfalls (challenges) one can encounter with corresponding solutions and workarounds.

For more details on the features and command options described in this document, refer to the *VC Formal User Guide* and *VC Static Command Reference Guide* available with VC Formal release or on SolvNet.

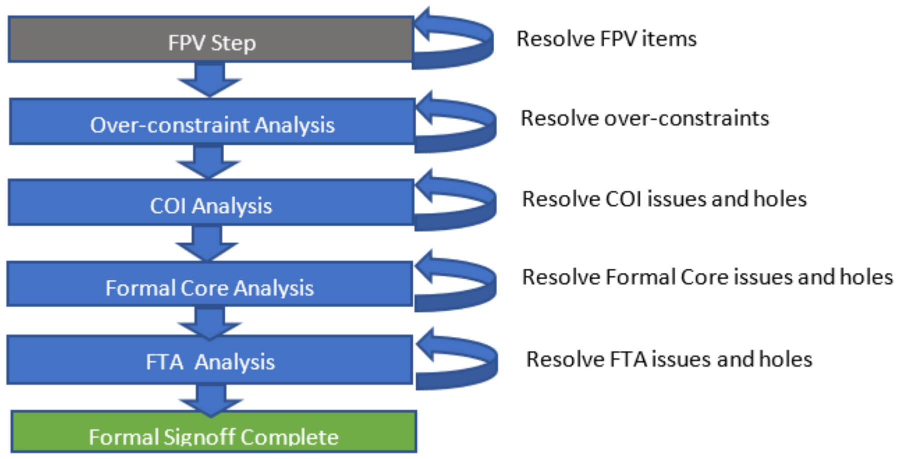
See section Resources and References for more information on the available VC Formal documentation.

2.1 Stages of Formal Signoff

Formal signoff is the process of gathering data about the Formal Property Verification (FPV) to determine whether the verification of a block or project is complete. The signoff process provides a step-by-step approach to try and answer the question "are we done?" with the verification of the block or project. Similar to code coverage or functional coverage metrics which are required for simulation-based verification signoff, formal verification requires a different set of tools and metrics for formal signoff.

Formal signoff is not a single step, but a process that exposes holes (also known as coverage gaps), bugs, and other vulnerabilities in the FPV environment or verification process. To make the process scalable, manageable, and successful, it is broken into a series of steps. The steps gradually go from less stringent to most stringent set of analysis and metrics. Similarly, each step becomes more complex and requires more compute resources and time to complete. It is highly advised that each step is followed strictly and that holes identified in a previous step are addressed prior to moving to the next step.

Figure 2-1 Formal Signoff Stages



The best practices for each step and recommendation for resolving common pitfalls are presented in the subsequent sections of this document.

2.2 Prerequisites for Formal Signoff

The recommended prerequisites for FPV is to increase the success at each formal signoff stage. While one can proceed without meeting the prerequisites, resolving these issues later in the process is inefficient and more time and resource consuming.

1. The standalone FPV environment should be correct, and all development should be complete.
2. There should not be any failing assertions (CEXs), vacuous assertions, and uncoverable cover properties.
3. The assertions should be either proven, or the inconclusive or bounded assertion should have reached a sufficient depth.

2.3 Goals for Formal Signoff

Every project has its own set of goals and metrics to achieve formal signoff based on the resources and time availability, criticality of the blocks/projects in question, and ability to make fixes/ECOs.

The following are the ideal and strictest set of goals for formal signoff:

1. All covers/vacuity-checks are reachable/covered with exceptions justified
2. No CEXs exist. All assertions are proven or inconclusive/bounded
3. 100% COI coverage or relevant logic
4. 100% formal core coverage
5. 0% FTA non-detected

3
Recommended Formal Signoff Setup Options

VC Formal has many options and settings that can be leveraged based on the situation. This section provides the settings and recommendations for the most balanced setup across the complete formal signoff flow.

Using Line and Branch Coverage Metrics

Use the line and branch coverage metrics. The other coverage metrics are compute-intensive, and should be used after careful consideration.

```
read_file … -cov line+branch …
```

Avoiding Constraint Minimization

Use the following formal variable to help with performance:

```
set_fml_var fml_formal_core_reduced_constraints false
```

Typical Formal Variable Settings

Enable the following formal variables:

```
# Include reset branches for coverage
set_fml_var fml_reset_property_check true
set_fml_var fml_track_loc_reset true

# Preserve source location of constant selects
set_fml_var fml_track_loc_const_select true

# Preserve source location of expression values
set_app_var keep_source_loc_expr true

# If branch coverage has been enabled
# Include branch coverage in analysis
set_fml_var fml_cov_enable_branch_cov true

# If toggle coverage is enabled
# Include toggle of primary inputs in coverage analysis
set_app_var fml_cov_tgl_input_port true
```

Configuring Formal Core Settings

Use on-the-fly formal core computation for better performance.

To enable formal core collection during proof, use the following command before `check_fv`:

```
check_config -formal_core both
```

Disabling Assertion Coverage Metrics in Coverage Database

The assertion coverage metric in Verdi coverage is simulation centric which is not required for formal sign off:

```
set_app_var fml_disable_assert_shape_dump true
```

Downgrading CFC_NO_FC_SUB Error to Warning

There can be local assertions which can be proven without looking at the Formal Core. The Formal core computation in such cases amounts to zero which is acceptable.

The CFC_NO_FC_SUB error message should be downgraded in such cases:

```
set_message_severity -names CFC_NO_FC_SUB warning
```

Injecting All Possible Faults

Inject all possible faults using the following commands during fault instrumentation:

```
fta_init -fast_sanity -scope ...
read_file...-inject_fault all -cov line+branch...
```

OR

```
fta_init -fast_sanity -scope ... -top ...
set_app_var fml_multi_step_fta_flow true
analyze ...
elaborate -sva...-inject_fault all -cov line+branch...
```

4 Over-constraint Analysis

Over-constraint analysis is to ensure that there are no constraints in the design preventing legal areas of code from being exercised. The under-constraints may cause false failures which are undesirable, however, over-constraints are more dangerous because they can lead to false proofs. The uncoverable code should be carefully reviewed to ensure that there is no unintentional over-constraining happening.

This analysis can be performed before or after the `check_fv` command. It needs the clock and reset definitions along with the Formal initial state setup to work correctly. It takes the assumes (SVA and fvassume) into consideration for its analysis.

Code coverage targets specified in the setup are analyzed by formal engines to determine the un-reachability of the targets.

4.1 Performing Over-constraint Analysis

Reading Existing Waiver Files

Read in the waiver files to exclude coverage targets from formal analysis:

```
read_waiver_file -elfiles …
```

Running Over-constraint Analysis

Run over-constraint analysis using the following command:

```
compute_over_constraint -par_task FPV -block
```

Saving Over-constraint Analysis Results

Save the over-constraint analysis results using the following command:

```
save_over_constraint_results
```

Note

You can use the following commands to run over-constraint and bounded coverage together.

```
compute_bounded_cov -par_task FPV -max_proof_depth -1 -block
save_bounded_cov_results
```

For more details on these commands, see section *Over-constraint (OC) Analysis* in the *VC Formal User Guide.*

The following results are saved:

- Reachable targets are saved in the VDB database *FPV_OA.vdb*.

- ❖ Unreachable targets are saved in the exclusion file as *FPV_OA_unr.el_w_constraint.el*
- ❖ Unreachable goals without constraints are saved in *FPV_OA_unr.el_wo_constraint.el*
- ❖ You can provide specific names for the VDB database and exclusion files.

Analyzing Results and Applying Waivers

Apply waivers on any expected uncoverable cover items. The uncoverable items without constraints are dead code, they should be explained and waived after review. The uncoverable items with constraint will require further debug to find out which constraint(s) is causing the corresponding code to be uncoverable. Use the following command for the analysis:

```
compute_reduced_constraints -property {<cover_prop_name>} -block
```

Reviewing Results in Verdi Coverage GUI

Use this command to open Verdi Coverage to view results:

```
view_coverage
```

You can also specify the VDB name and the el file(s) along with `view_coverage`, if needed:

```
view_coverage -cov_input FPV_OA.vdb -elfiles { FPV_OA_unr.el_wo_constraint.el }
```

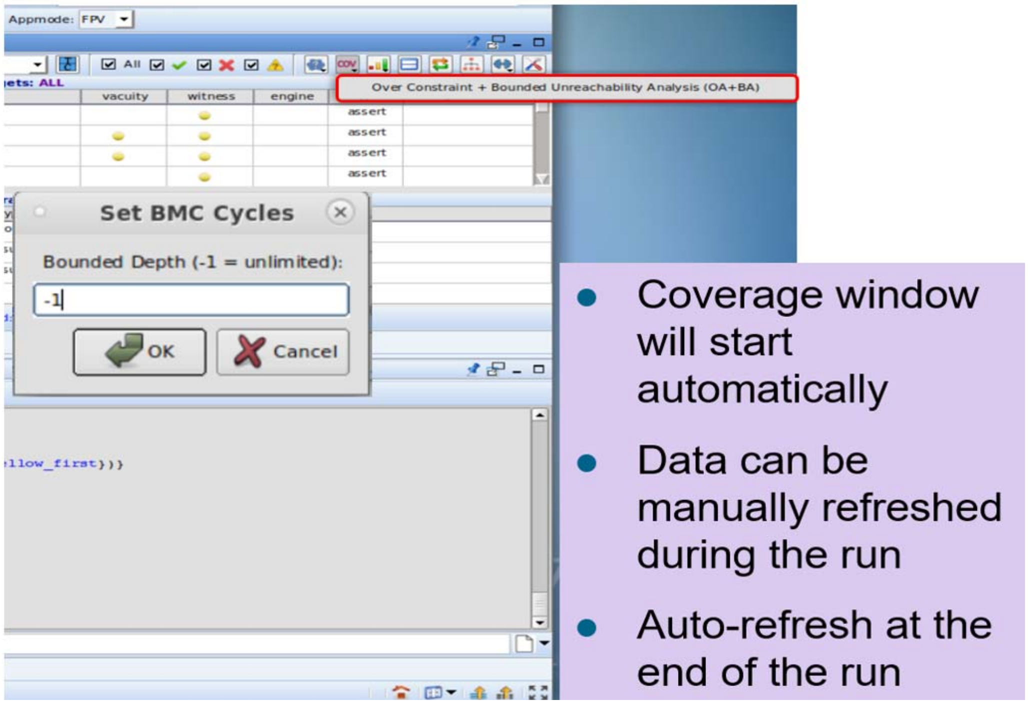
Analyzing Coverage Data and Saving Waivers

Analyze the coverage data from Verdi coverage view. Waive uncoverable items that can be explained. Save all waived items to exclusion file in Verdi coverage as needed.

Run Over-constraint Analysis in the GUI

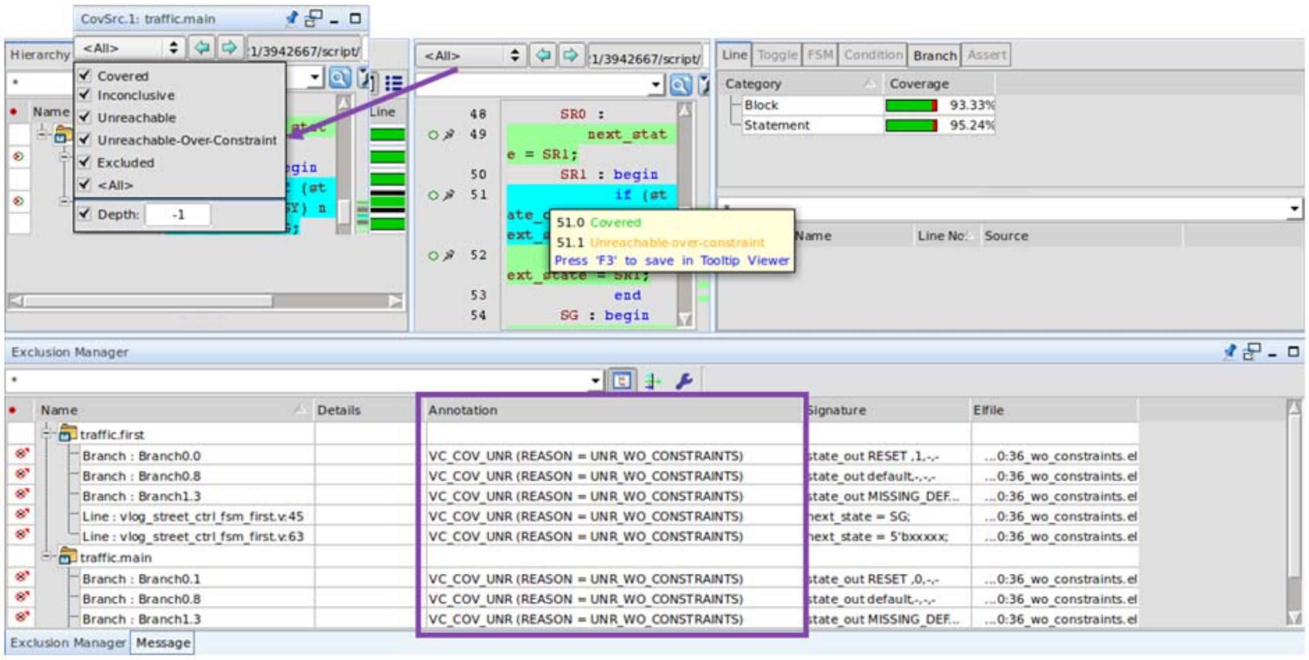
You can use the GUI to perform over-constraint analysis.

1. Invoke the run by clicking on the COV icon.



- Coverage window will start automatically
- Data can be manually refreshed during the run
- Auto-refresh at the end of the run

2. Review coverage items with over-constraint in the annotated exclusion items.



```
48          SR0 :
49              next_stat
e = SR1;
50          SR1 : begin
51              if (st
ate_
ext_
52
ext_state = SR1;
53              end
54          SG : begin
```

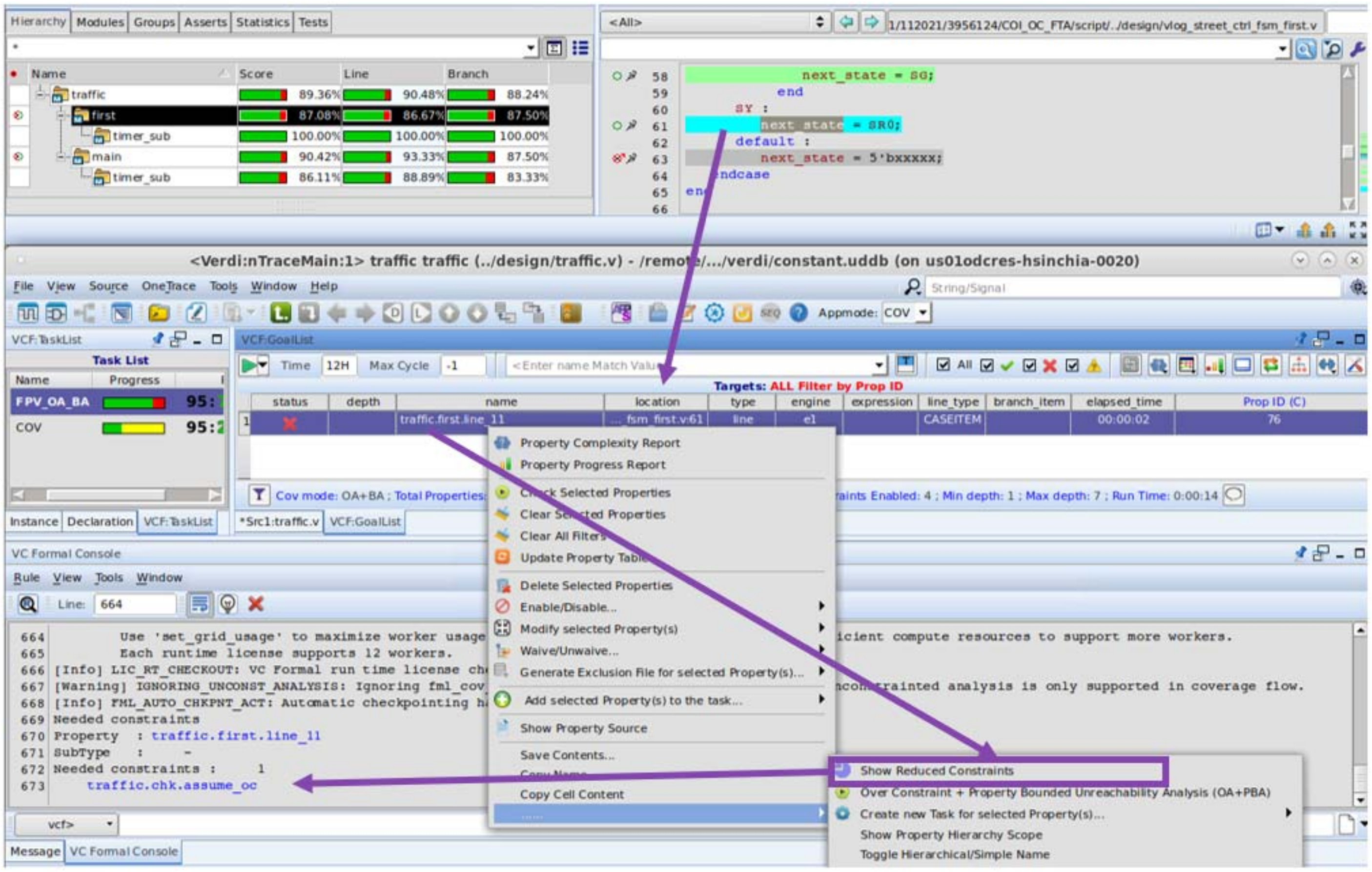
51.0 Covered
51.1 Unreachable-over-constraint
Press 'F3' to save in Tooltip Viewer

Category	Coverage
Block	93.33%
Statement	95.24%

Exclusion Manager

Name	Details	Annotation	Signature	Elfile
traffic.first				
Branch : Branch0.0		VC_COV_UNR (REASON = UNR_WO_CONSTRAINTS)	state_out RESET ,1,-,-	...0:36_wo_constraints.el
Branch : Branch0.8		VC_COV_UNR (REASON = UNR_WO_CONSTRAINTS)	state_out default,-,-,-	...0:36_wo_constraints.el
Branch : Branch1.3		VC_COV_UNR (REASON = UNR_WO_CONSTRAINTS)	state_out MISSING_DEF...	...0:36_wo_constraints.el
Line : vlog_street_ctrl_fsm_first.v:45		VC_COV_UNR (REASON = UNR_WO_CONSTRAINTS)	next_state = SG;	...0:36_wo_constraints.el
Line : vlog_street_ctrl_fsm_first.v:63		VC_COV_UNR (REASON = UNR_WO_CONSTRAINTS)	next_state = 5'bxxxxx;	...0:36_wo_constraints.el
traffic.main				
Branch : Branch0.1		VC_COV_UNR (REASON = UNR_WO_CONSTRAINTS)	state_out RESET ,0,-,-	...0:36_wo_constraints.el
Branch : Branch0.8		VC_COV_UNR (REASON = UNR_WO_CONSTRAINTS)	state_out default,-,-,-	...0:36_wo_constraints.el
Branch : Branch1.3		VC_COV_UNR (REASON = UNR_WO_CONSTRAINTS)	state_out MISSING_DEF...	...0:36_wo_constraints.el

3. Debug by identifying which constraints are involved in over-constraining by running **Show Reduced Constraints** of the uncoverable target.


Hierarchy
Modules
Groups
Asserts
Statistics
Tests
<All>
1/112021/3956124/COI_OC_FTA/script/../design/vlog_street_ctrl_fsm_first.v
<Verdi:nTraceMain:1> traffic traffic (../design/traffic.v) - /remote/.../verdi/constant.uddb (on us01odcres-hsinchia-0020)
File
View
Source
OneTrace
Tools
Window
Help
String/Signal
Appmode: COV
VCF:TaskList
Task List
VCF:GoalList
Time 12H
Max Cycle -1
<Enter name Match Value>
All
Instance
Declaration
*Src1:traffic.v
VC Formal Console
Rule
Line: 664
vcf>
Message
Property Complexity Report
Property Progress Report
Check Selected Properties
Clear Selected Properties
Clear All Filters
Update Property Table
Delete Selected Properties
Enable/Disable...
Modify selected Property(s)
Waive/Unwaive...
Generate Exclusion File for selected Property(s)...
Add selected Property(s) to the task...
Show Property Source
Save Contents...
Copy Name
Copy Cell Content
Show Reduced Constraints
Over Constraint + Property Bounded Unreachability Analysis (OA+PBA)
Create new Task for selected Property(s)...
Show Property Hierarchy Scope
Toggle Hierarchical/Simple Name

Name	Score	Line	Branch
traffic	89.36%	90.48%	88.24%
first	87.08%	86.67%	87.50%
timer_sub	100.00%	100.00%	100.00%
main	90.42%	93.33%	87.50%
timer_sub	86.11%	88.89%	83.33%

```
                        next_state = SG;
                    end
                SY :
                    next_state = SR0;
                default :
                    next_state = 5'bxxxxx;
            endcase
end
```

Name	Progress
FPV_OA_BA	95:
COV	95:2

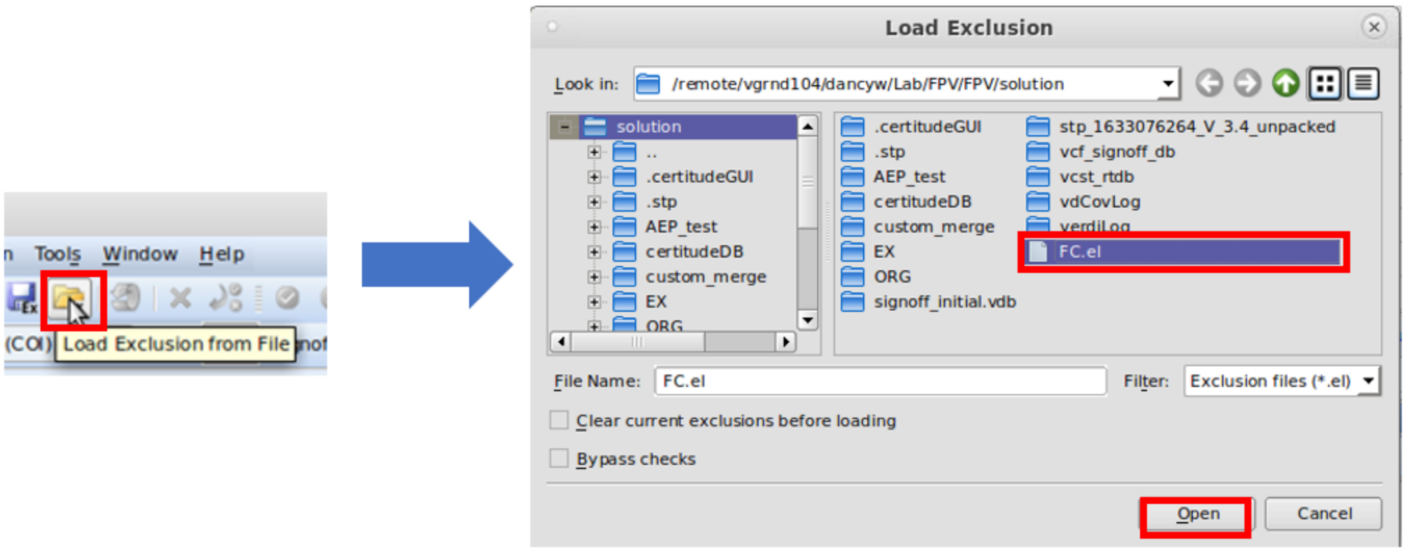
Targets: ALL Filter by Prop ID

status	depth	name	location	type	engine	expression	line_type	branch_item	elapsed_time	Prop ID (C)
✗		traffic.first.line_11	... fsm_first.v:61	line	e1		CASEITEM		00:00:02	76

Cov mode: OA+BA ; Total Properties ... aints Enabled: 4 ; Min depth: 1 ; Max depth: 7 ; Run Time: 0:00:14

```
        Use 'set_grid_usage' to maximize worker usage ...icient compute resources to support more workers.
        Each runtime license supports 12 workers.
[Info] LIC_RT_CHECKOUT: VC Formal run time license ch
[Warning] IGNORING_UNCONST_ANALYSIS: Ignoring fml_cov ...nconstrained analysis is only supported in coverage flow.
[Info] FML_AUTO_CHKPNT_ACT: Automatic checkpointing h
Needed constraints
Property   : traffic.first.line_11
SubType    :     -
Needed constraints :     1
    traffic.chk.assume_oc
```

4. Analyze all the uncoverable items and waive them as shown in Analyzing Results and Applying Waivers, and save the waivers as shown in Reviewing Results in Verdi Coverage GUI. Once this is done you can read any additional exclusion files through the GUI as shown in the following figure:



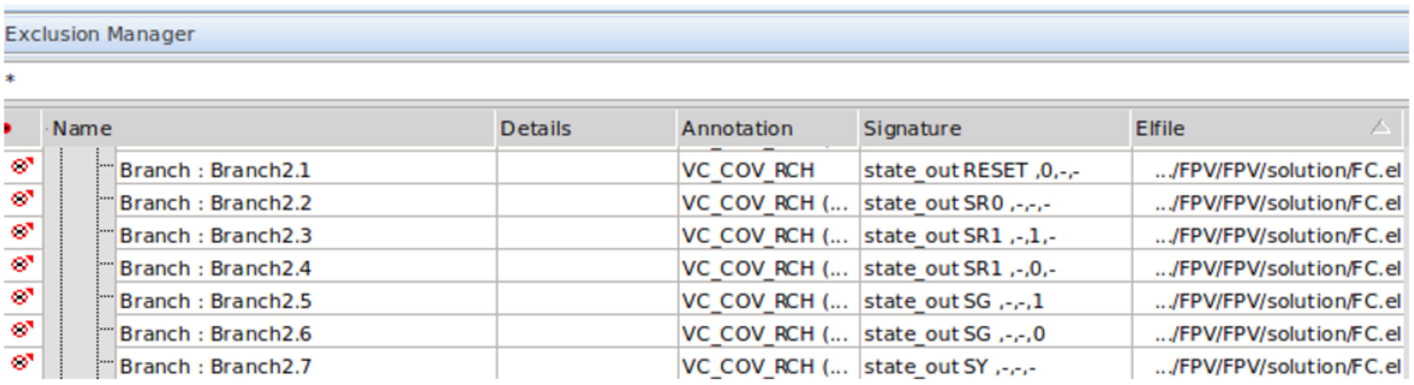
Look in: /remote/vgrnd104/dancyw/Lab/FPV/FPV/solution

File Name: FC.el

Filter: Exclusion files (*.el)

Clear current exclusions before loading

Bypass checks



Exclusion Manager

Name	Details	Annotation	Signature	Elfile
Branch : Branch2.1		VC_COV_RCH	state_out RESET ,0,-,-	../FPV/FPV/solution/FC.el
Branch : Branch2.2		VC_COV_RCH (...	state_out SR0 ,-,-,-	../FPV/FPV/solution/FC.el
Branch : Branch2.3		VC_COV_RCH (...	state_out SR1 ,-,1,-	../FPV/FPV/solution/FC.el
Branch : Branch2.4		VC_COV_RCH (...	state_out SR1 ,-,0,-	../FPV/FPV/solution/FC.el
Branch : Branch2.5		VC_COV_RCH (...	state_out SG ,-,-,1	../FPV/FPV/solution/FC.el
Branch : Branch2.6		VC_COV_RCH (...	state_out SG ,-,-,0	../FPV/FPV/solution/FC.el
Branch : Branch2.7		VC_COV_RCH (...	state_out SY ,-,-,-	../FPV/FPV/solution/FC.el

5 COI (Property Density) Analysis

COI analysis extracts registers, primary inputs, primary outputs, and lines of code in the fanin cone of assertions.

Support in VC Formal

- By default, only user-defined SVA are considered for COI analysis.
- User defined covers, Tcl asserts (fvasserts) and Tcl covers (fvcovers) can be enabled if needed. Asserts (SVA and Tcl) are the verification objects of focus from a signoff perspective, and therefore, it is recommended to focus on these for COI analysis.

5.1 Running COI Analysis

Setting up Coverage Model

Perform the coverage setup instruction described in section Using Line and Branch Coverage Metrics for setting up the coverage model required for COI analysis.

Enabling COI Coverage

The analysis result can be mapped to code coverage by enabling the following formal variable:

```
set_fml_var fml_enable_prop_density_cov_map true
```

Reading Existing Waiver Files

See section Reading Existing Waiver Files to read the existing waiver files:

```
read_waiver_file -elfiles …
```

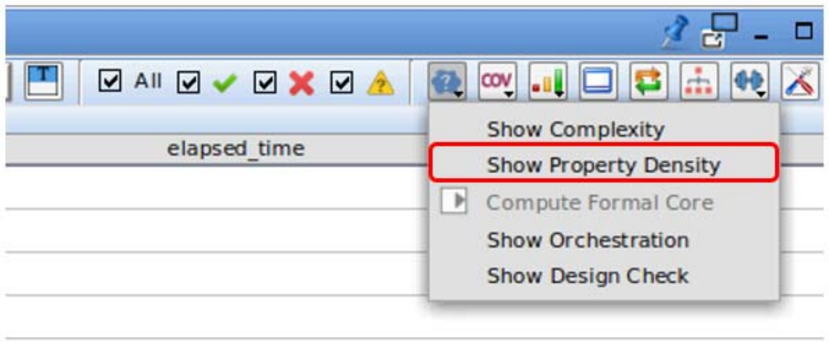
Getting a Report of the COI Analysis

This can be run before or after `check_fv` command. By default, COI analysis is done ignoring snips and black box information. If you want to include these, you can use the `-abstracted` option with the `report_assertion_density` command.

```
report_assertion_density -include scriptProps -property [ get_prop -usage assert ]
```

Running COI in GUI

Alternatively, in VCF GUI drop-down menu can be used to generate the same information.



Saving COI Analysis Results to a VDB

Save the COI analysis results to VDB using the following command:

```
save_property_density_results -db_name <db_name>
```

Viewing the Results in Verdi Coverage

Perform the steps in section Reviewing Results in Verdi Coverage GUI.

Analyzing Coverage and Saving Waivers

Analyze the coverage data from Verdi coverage view. Waive uncoverable items that can be explained. Save all waived items to exclusion file in Verdi Coverage as needed.

6 Formal Core Analysis

Formal core is defined as a sub-set of COI that is used by a Formal engine to verify a given property. It is always less than or equal to the COI of the property. If there are multiple properties being verified, the formal core is the union of all the properties.

In formal core analysis, the engine traverses through the hierarchy and extracts registers, inputs, and constraints responsible for the proof of an assertion. We then map these RTL structures to the source code through their respective code coverage items (line, branch, toggle, and condition). These metrics can then be used to signoff the formal verification effort for your design.

Support in VC Formal

- By default, only user-defined SVA and Tcl asserts (fvasserts) are considered for formal core analysis.
- User-defined and Tcl covers (fvcovers) can be enabled using the `-subtype` option if needed. Asserts are the verification objects of focus from a Signoff perspective and hence, it recommended to focus on these for formal core analysis.

Note
Formal Core analysis is a good way of debugging Unreachable/Uncoverable covers, however it is not relevant for signoff computation

- Formal Core does not currently have support for liveness assertions, functions, toggle coverage for outputs of blackboxes and RMA support.

6.1 Performing Formal Core Analysis

Setting up the Coverage Model

See section "Using Line and Branch Coverage Metrics" for setting up the coverage model required for formal core analysis.

Reading Existing Waivers

See section "Reading Existing Waiver Files" to read the existing waivers

```
read_waiver_file -elfiles …
```

Completing FPV run

Use the `check_fv` command to complete the FPV run.

Computing Formal Core

Once `check_fv` has run, use the following command to compute formal core

```
compute_formal_core -ignoreStatus -property [ get_props -usage assert -status {proven
inconclusive} ] -block
```

Mapping the Formal Core to Code Coverage

Use the following command to map the formal core to the code coverage:

```
compute_formal_core_coverage -property [ get_props -usage assert -status {proven
inconclusive} ] -par FPV -block
```

Launching Coverage Results

Load the Verdi Coverage (screen shot in section Reviewing Coverage Results in Verdi Coverage View) to review coverage results.

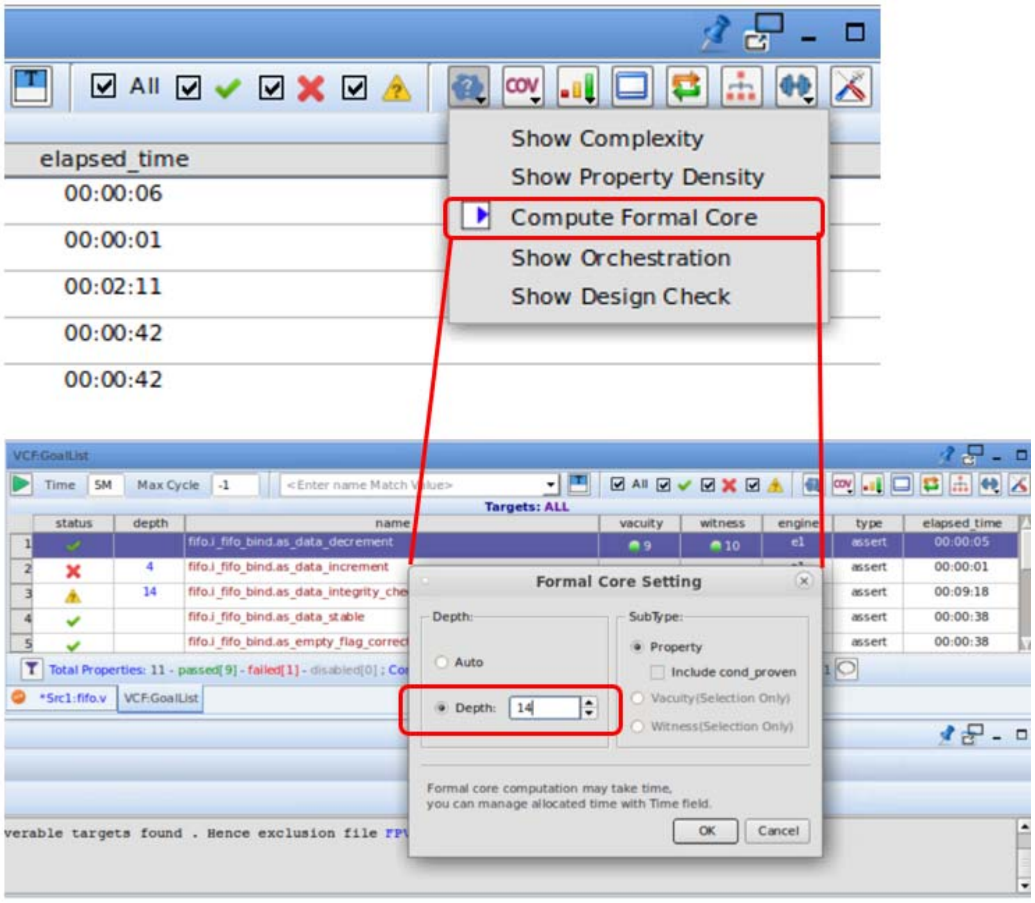
```
view_coverage -auto_save -task FPV_COV_FCORE -mode FC -property [ get_props -usage
assert -status {proven inconclusive} ] -status -reload
```

Analyzing Coverage Data and Saving Waiver

Use the steps in section Analyzing Coverage and Saving Waivers to open Verdi coverage.

Running Formal Core Analysis using GUI

You can also use do the same process using VC Formal GUI as shown in the following figure:



elapsed_time
00:00:06
00:00:01
00:02:11
00:00:42
00:00:42

Time: 5M Max Cycle: -1

	status	depth	name	vacuity	witness	engine	type	elapsed_time
1	✓		fifo.i_fifo_bind.as_data_decrement	● 9	● 10	e1	assert	00:00:05
2	✗	4	fifo.i_fifo_bind.as_data_increment				assert	00:00:01
3	⚠	14	fifo.i_fifo_bind.as_data_integrity_che				assert	00:09:18
4	✓		fifo.i_fifo_bind.as_data_stable				assert	00:00:38
5	✓		fifo.i_fifo_bind.as_empty_flag_correc				assert	00:00:38

Total Properties: 11 - passed[9] - failed[1] - disabled[0] ; Co

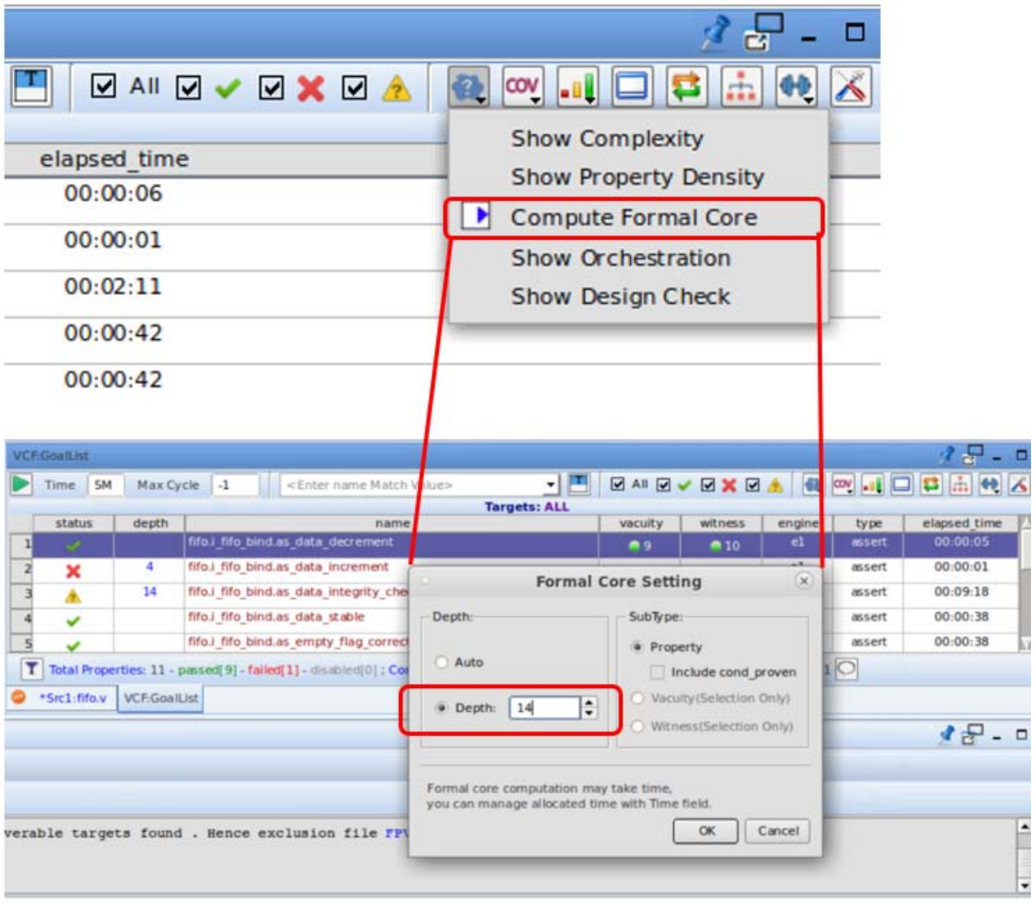
```
verable targets found . Hence exclusion file FP
```

Formal Core Setting

Depth: [ ] Auto [✓ Depth: 14]

SubType: [✓ Property] [ ] Include cond_proven [ ] Vacuity(Selection Only) [ ] Witness(Selection Only)

Formal core computation may take time,
you can manage allocated time with Time field.



elapsed_time
00:00:06
00:00:01
00:00:42
00:00:42

Time 5M Max Cycle -1 <Enter name Match Value>

	status	depth	name	vacuity	witness	engine	type	elapsed_time
1	✓		fifo.i_fifo_bind.as_data_decrement	● 9	● 10	e1	assert	00:00:05
2	✗	4	fifo.i_fifo_bind.as_data_increment				assert	00:00:01
3	⚠	14	fifo.i_fifo_bind.as_data_integrity_che				assert	00:09:18
4	✓		fifo.i_fifo_bind.as_data_stable				assert	00:00:38
5	✓		fifo.i_fifo_bind.as_empty_flag_correc				assert	00:00:38

Total Properties: 11 - passed[9] - failed[1] - disabled[0] ; Co

*Src1:fifo.v VCF:GoalList

```
verable targets found . Hence exclusion file FP
```

Formal Core Setting

Depth: ○ Auto [✓ ● Depth: 14]

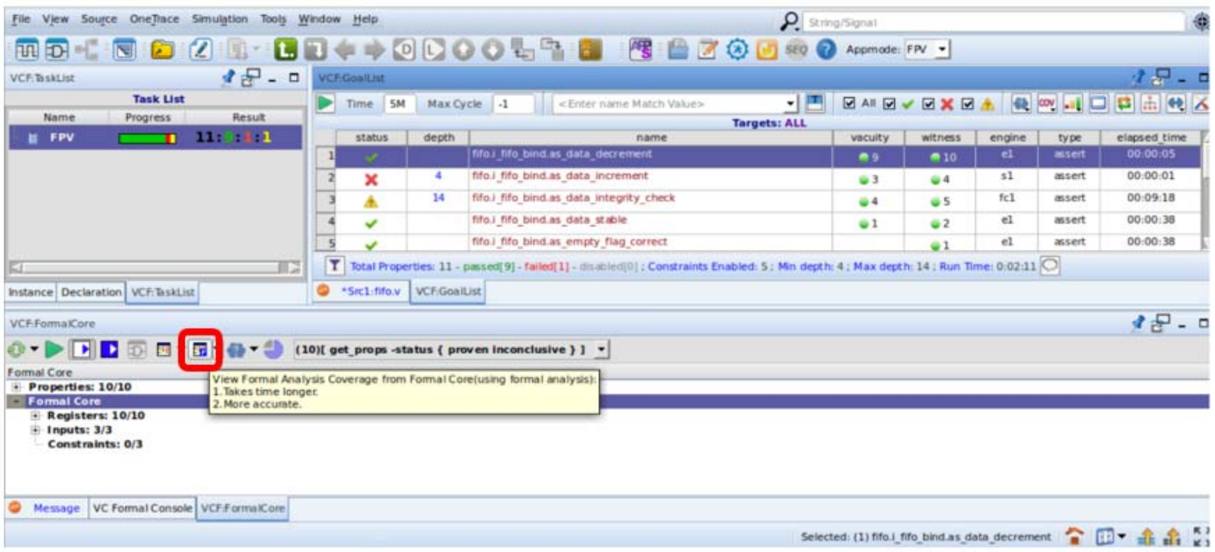
SubType: [✓ ● Property] ☐ Include cond_proven ○ Vacuity(Selection Only) ○ Witness(Selection Only)

Formal core computation may take time,
you can manage allocated time with Time field.

Note
Please make sure you enter the lowest depth amongst all inconclusive assertions in that session.

Reviewing Coverage Results

The Formal Core tab will be launched as shown below which will have all the registers, inputs and constraints affecting the assertions in that session. By clicking on the button highlighted below, you can launch Verdi Coverage view.



VCF:TaskList — Task List

Name	Progress	Result
FPV		11 : 8 : 1 : 1

VCF:GoalList — Time 5M, Max Cycle -1, Appmode: FPV — Targets: ALL

	status	depth	name	vacuity	witness	engine	type	elapsed_time
1	✓		fifo.i_fifo_bind.as_data_decrement	● 9	● 10	e1	assert	00:00:05
2	✗	4	fifo.i_fifo_bind.as_data_increment	● 3	● 4	s1	assert	00:00:01
3	⚠	14	fifo.i_fifo_bind.as_data_integrity_check	● 4	● 5	fc1	assert	00:09:18
4	✓		fifo.i_fifo_bind.as_data_stable	● 1	● 2	e1	assert	00:00:38
5	✓		fifo.i_fifo_bind.as_empty_flag_correct		● 1	e1	assert	00:00:38

Total Properties: 11 - passed[9] - failed[1] - disabled[0] ; Constraints Enabled: 5 ; Min depth: 4 ; Max depth: 14 ; Run Time: 0:02:11

VCF:FormalCore

```
(10)[ get_props -status { proven inconclusive } ]
```

Formal Core
- Properties: 10/10
- Formal Core
  - Registers: 10/10
  - Inputs: 3/3
  - Constraints: 0/3

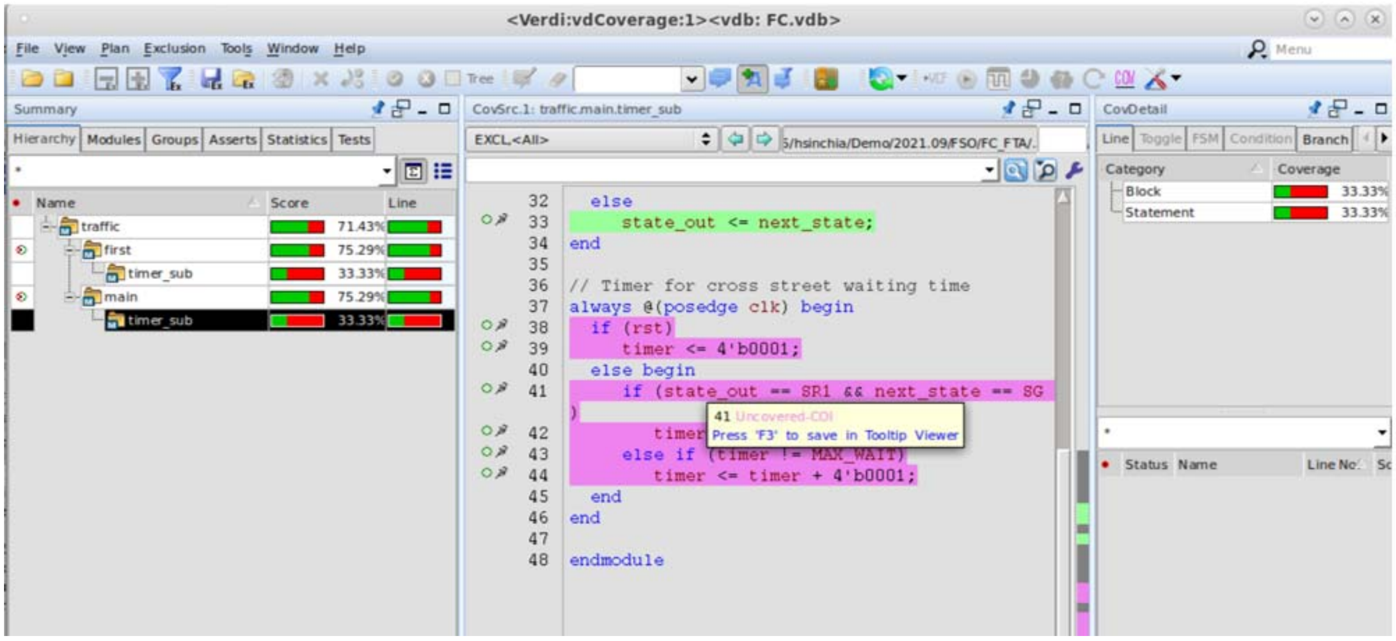
View Formal Analysis Coverage from Formal Core(using formal analysis):
1. Takes time longer.
2. More accurate.

Selected: (1) fifo.i_fifo_bind.as_data_decrement

File View Source OneTrace Simulation Tools Window Help
String/Signal
Instance Declaration VCF:TaskList
*Src1:fifo.v VCF:GoalList
Message VC Formal Console VCF:FormalCore

Reviewing Coverage Results in Verdi Coverage View

In the Verdi coverage view, you can analyze the code that was covered during the assertion(s) proof.



<Verdi:vdCoverage:1><vdb: FC.vdb>

Summary

Name	Score	Line
traffic	71.43%	
first	75.29%	
timer_sub	33.33%	
main	75.29%	
timer_sub	33.33%	

CovSrc.1: traffic.main.timer_sub

EXCL,<All>

5/hsinchia/Demo/2021.09/FSO/FC_FTA/.

```
32      else
33          state_out <= next_state;
34    end
35
36    // Timer for cross street waiting time
37    always @(posedge clk) begin
38      if (rst)
39          timer <= 4'b0001;
40      else begin
41          if (state_out == SR1 && next_state == SG
        )
42              timer
43          else if (timer != MAX_WAIT)
44              timer <= timer + 4'b0001;
45      end
46    end
47
48    endmodule
```

41 Uncovered-COI
Press 'F3' to save in Tooltip Viewer

CovDetail

Category	Coverage
Block	33.33%
Statement	33.33%

File View Plan Exclusion Tools Window Help
Menu
Hierarchy Modules Groups Asserts Statistics Tests
Line Toggle FSM Condition Branch
Status Name Line No. Sc

Applying and Saving Waivers

Analyze cover items outside the Formal Core and uncoverable items need to be explained and waived. Save all waived items to exclusion file in Verdi Coverage as needed.

Saving Formal Core Coverage Database

Once all the results have been analyzed, save Formal Core results into a VDB along with the exclusions using the below command:

```
save_formal_core_results -db_name <db_name> -elfile_name <exclusion_file.el>
```

Best Practices

- ❖ Regardless of what orchestration was run for FPV, default is the best orchestration for Formal Core analysis. So, do not forget to switch the `fml_effort` back to default.
- ❖ Some engines do not generate Formal Core on-the-fly, so it needs to be calculated explicitly for those engines. We recommend the following `compute_formal_core` commands after `check_fv` to take care of this.

  ```
  compute_formal_core -ignoreStatus -property [ get_props -usage assert -status {proven
  inconclusive} ] -block
  compute_formal_core_coverage -property [ get_props -usage assert -status {proven
  inconclusive} ] -par FPV -block
  ```

Performance Considerations

1. Increase number of cores and memory. Use at least the same number of cores and memory as FPV
2. Limit the depth.

   If formal core analysis is taking significant time, you can limit the depth of computation using the `-depth` option. This will be an approximation, but it will get us some metrics for signoff

   ```
   compute_formal_core -ignoreStatus -property [ get_props -usage assert -status { proven
   inconclusive } ] -depth N -block
   ```

Note
Start with a value of N = least inconclusive depth amongst assertions. If this is taking time, decrement the value of N by 1 (-depth N-1) each time to see if you are making progress. Finally settle on a value of N where you can get a good ROI for your efforts.

3. Divide and conquer

   This is a good way of computing formal core for individual groups of assertions where the design is big. Merge individual VDBs using the following `urg` command to hopefully improve formal core computation performance.

   ```
   urg -dir vcf_fcore_1.vdb -dir vcf_fcore_2.vdb -dbname vcf_fcore.vdb
   ```

4. When to stop

   If you have tried all the above recommendations, monitor throughput of code cover items being processed. If the progress has stagnated, stop the formal core computation. This would be a logical closure point for your analysis.

7 FTA (Formal Testbench Analyzer)

The quality of the Formal Testbench (set of properties for Formal Property Verification) is often difficult to measure. FTA inserts faults (or mutations) in the RTL and checks if the existing set of assertions can catch them. A fault that none of the assertions can catch indicates a potential verification hole.

One solution is to add more assertions or improve the quality of the existing Formal Testbench. Correlating FTA results with standard line and branch coverage metrics further refines the Formal Signoff coverage.

FTA is a computationally intensive step where conceptually FPV must be performed on each fault inserted (although there are many optimizations in practice). Therefore, it is important that efficient execution be considered through the process. Notable considerations are to debug/analyze results immediately as they become available. There is no need to wait for the entire FTA process to be complete before addressing the gaps and holes identified. Another consideration which is expanded on below, is different approaches to reduce the complexity by either reducing the faults inserted or applying divide-and-conquer approaches. Finally, since signoff is an important and final step of FPV, providing additional compute resources and increasing timeouts also improves the quality of results.

Support in VC Formal

- ❖ By default, only user-defined SVA and Tcl asserts (fvasserts) are considered for FTA analysis.
- ❖ Liveness properties in the Formal Testbench are not currently supported for FTA fault analysis and are ignored. Only safety properties are considered.

7.1 Performing FTA Analysis

Setting up FTA

See section Injecting All Possible Faults for FTA setup.

Running FPV Analysis

If you have not already run it as shown in Completing FPV run.

Once the FPV run has finished, configure the target FPV properties and fault classes for analysis.

```
configure_fta_props -fault_class {TopOutputsConnectivity InternalConnectivity} -status
{proven inconclusive} -par_task FPV
```

Optimizing faults before running FTA

The following command executes Certitude fault collapsing step to minimize the number of faults to analyze:

```
fta_optimize_faults
```

Setting effort level

Set the `fml_effort` effort level to default for FTA run, as this effort level gives best overall results during fault computation:

```
set_fml_var fml_effort default
```

Reading existing waiver files

Use the following to open Verdi coverage:

```
read_fta_waiver -elfiles …
```

Running FTA Fault Analysis

Run the FTA analysis using the following command:

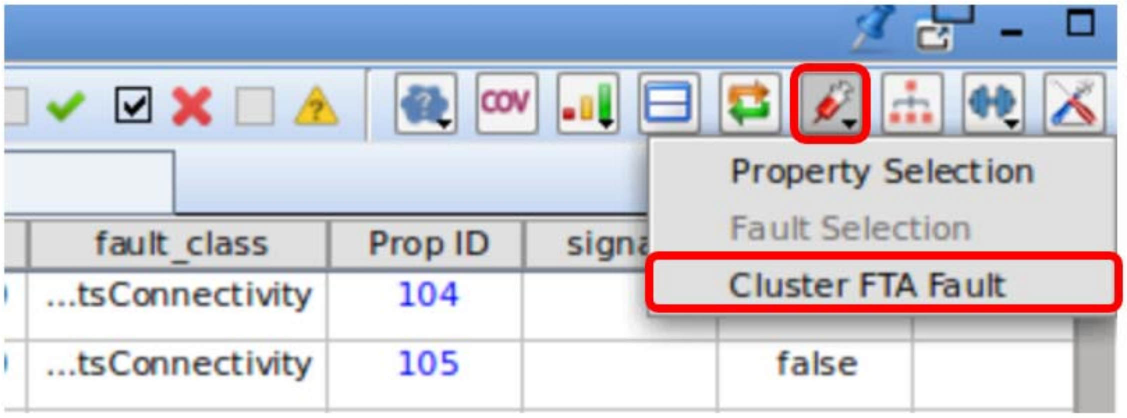
```
set_fml_appmode FTA
compute_fta -par_task FPV -block
```

Clustering Faults

The following steps helps to identify groups of key faults to debug and target together. Add assertions based on the fault cluster that you are trying to debug.

```
    cluster_fta_faults
    report_fta_fault_clusters
```

You can also do it in the GUI as shown below



fault_class	Prop ID	signa…		
...tsConnectivity	104			
...tsConnectivity	105		false	

Debugging Non-Detected Faults

Use the following to leverage the SEQ application capabilities:

```
set_fml_var fml_fta_seq_debug true -global
debug_fta
export_fault -par_task SEQ -property …
```

Saving Coverage Results

Save the results into a VDB and exclusions into a file if coverage (line and branch) is enabled.

```
save_fta_cov_results -name <db_name> -elfiles <exclusion_file.el>
```

Opening Verdi Coverage GUI

```
view_coverage -cov_input <db_name>
```

Note
If you have specified a DB name in 7.2.10, pass it here. By default, the database is stored as *FPV_FTA.vdb*. If you also have exclusion files, specify them using additional `-elfiles` option.

Analyzing Coverage Results

Check cover items associated with the faults which are outside the COI/Formal Core and apply waivers as needed. All the uncoverable items are to be explained and waived.

Saving Waiver Files

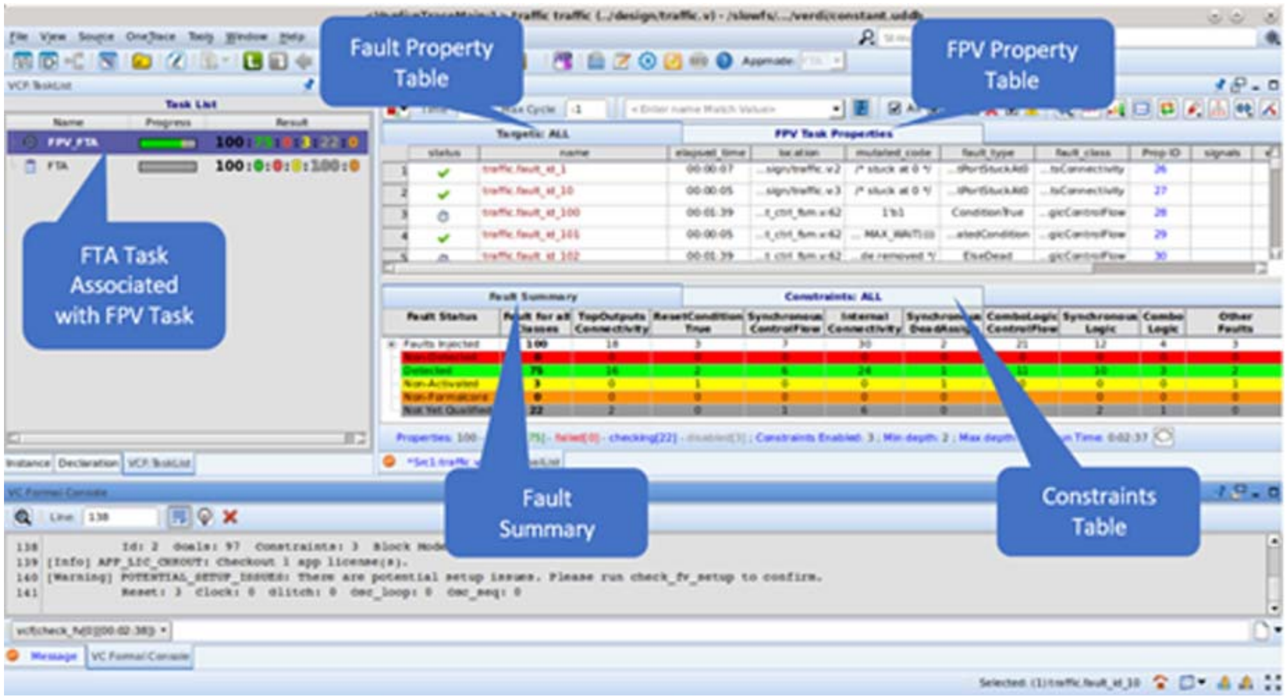
Save waiver items using the following command for reuse them with the `read_fta_waiver` command.

```
save_waiver_file -fta -file …
```

Merging Coverage Results

When coverage is enabled, multiple coverage databases can be merged with URG as described in section Saving Formal Core Coverage Database.

FTA Fault Table and Viewing



```
            Id: 2  Goals: 97  Constraints: 3  Block Mode
[Info] APP_LIC_CHKOUT: Checkout 1 app license(s).
[Warning] POTENTIAL_SETUP_ISSUES: There are potential setup issues. Please run check_fv_setup to confirm.
            Reset: 3  Clock: 0  Glitch: 0  Comb_loop: 0  Comb_seq: 0
```

Best Practices

- ❖ Start reviewing FTA results as soon as non-detected faults are being reported. You do not have to wait for the run to complete before you begin analyzing the results.
- ❖ If there are inconclusive faults in the FTA run, identify the design block that includes many of them and add local assertions thus improving the Formal Testbench for that block. Run FTA once COI coverage is 100% and Formal Core coverage is greater than 90%. These Formal Signoff methods take less effort than FTA and will help in the overall quality and performance of the FTA run.
- ❖ By default, FTA works on the COI of assertions. It can be restricted to Formal Core which should help boost performance. When fault qualification on Formal Core is enabled, faults outside the Formal Core are not considered and are not marked as Formal Core.

```
set_fml_var fml_qual_fault_in_fcore true
```

Note
Make sure you have fully computed Formal Core for target properties before running command `compute_fta`. If you are not planning to run Formal Core, ignore the setting and VC Formal considers the COI for its FTA computations.

- ❖ Recommend using connectivity classes of faults for initial FTA runs as shown in Running FPV Analysis. Using all fault classes can create many faults and have an adverse impact on performance.
- ❖ Configure FTA to instrument faults in individual instances if needed using the following command

```
set_fml_var fml_fta_inst_qual true
```

- ❖ Configure FTA to do fault analysis on both proven and inconclusive assertions as shown in Running FPV Analysis. By default, only proven properties are considered. For inconclusive assertions, FTA currently analyses faults up to the lowest inconclusive depth of assertions in that session.
- ❖ Detected faults have priority over non-detected faults when it comes to coverage representation in the source browser. Please review non-detected faults in FTA source view window.
- ❖ Check assertion quality and add more assertions where faults are reported as inconclusive to increase detection rate.

Performance Considerations

1. Restrict scope of fault insertion using a divide-and-conquer approach by specifying the portion of the design for fault analysis.

```
fta_init -scope …
```

2. FTA run times are generally longer than its corresponding FPV run. Increase number of cores and memory for individual jobs. Provide run time at least 2X of FPV time

```
set_grid_usage -type …
set_fml_var fml_max_time …
set_fml_var fml_max_mem …
```

Note
By default, FTA uses FPV parent task grid setup and settings.

3. FTA has its own high effort level `fta_high_effort` and time limit per fault `fml_max_time_per_fault`. Use these options for better convergence if you have more time and resources.

```
set_fml_var fml_fta_high_effort true
set_fml_var fml_max_time_per_fault 5M
compute_fta -par_task FPV -incr -block
```

Note

Do not forget to pass the `-incr` option to subsequent `compute_fta` runs to continue FTA from the previous computation.

4. If you have tried all the above recommendations, monitor convergence of fault properties being processed. If progress is tapering out, stop the proof.

8 Formal Signoff Dashboard

Formal Signoff Dashboard is the next generation signoff solution from VC Formal where all the Signoff stages are integrated in an interactive GUI environment. The signoff dashboard focuses on coverage metrics, and helps interpret formal verification results in code coverage semantics making signoff on your formal verification effort easier.

Support in VC Formal

The support available in the signoff flow described in the previous chapters applies to the signoff dashboard. See section *Supported in VC Formal* in previous chapters for details.

8.1 Using Formal Signoff Dashboard

Enabling Signoff Dashboard

Set the following environment variables to enable formal signoff dashboard and the next generation signoff flow:

```
setenv SNPS_VCF_SIGNOFF_DASHBOARD_BETA 1
setenv VERDI_ENABLE_SYNC_SOURCE_WINDOW 1
```

Recommended Settings

Use the following settings to setup the flow for signoff.

- Using Line and Branch Coverage Metrics
- Avoiding Constraint Minimization
- Configuring Formal Core Settings
- Downgrading CFC_NO_FC_SUB Error to Warning

Specifying the Scope for FTA Fault Instrumentation

Use the following command to specify the scope for FTA fault instrumentation:

```
fta_init -fast_sanity -scope ...
```

Specifying the Coverage and Fault Models

Use the following command to specify both coverage and fault you want to run:

```
signoff_config -type {line branch fault_conn fault_rtl}
```

Setting Up FPV for FTA

Complete the `read_file`, `create_clock` and `create_reset` commands, initialization steps and run FPV as mentioned in Completing FPV run.

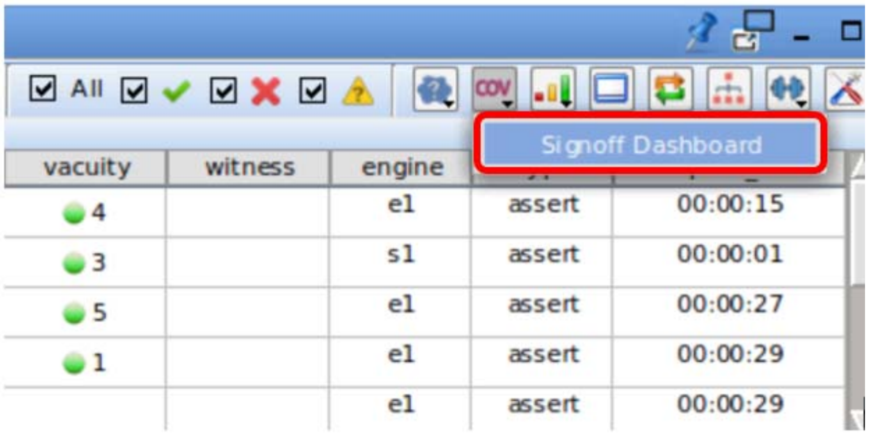
Configuring FTA

Once FPV has been run, perform the following steps to configure tool for FTA analysis:

- Running FPV Analysis
- Optimizing faults before running FTA
- Setting effort level

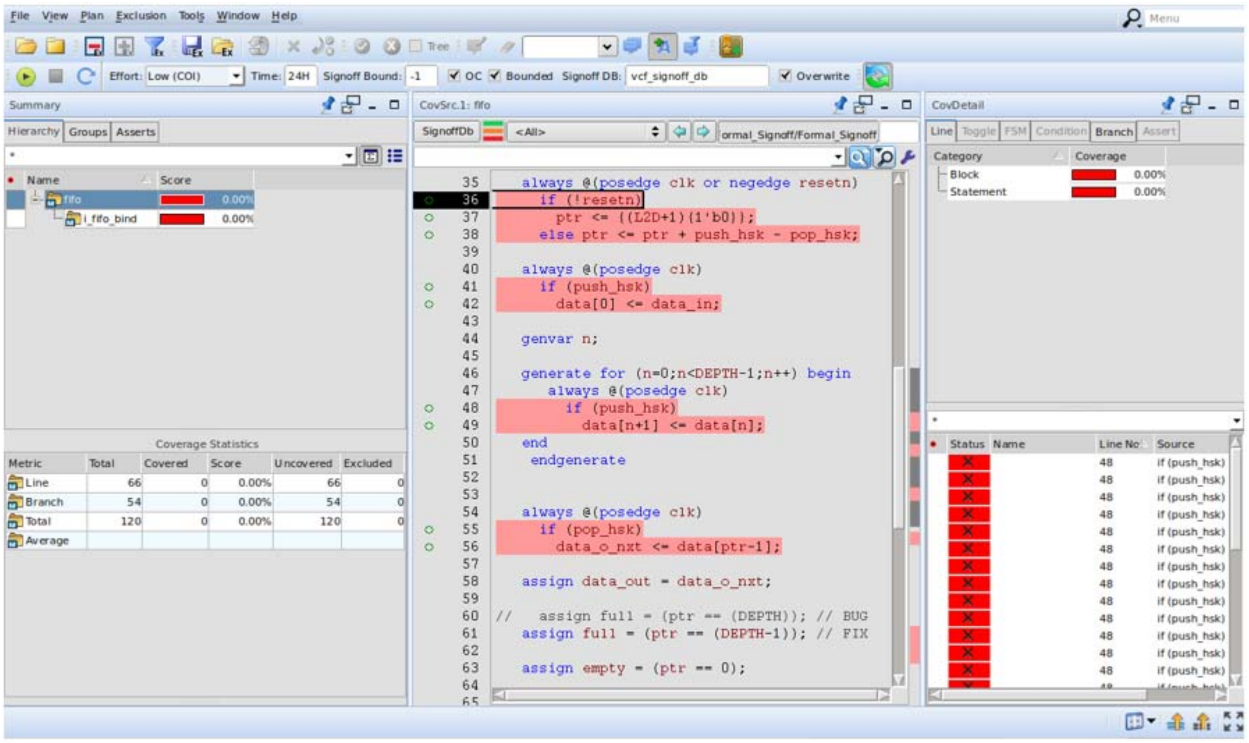
Opening Formal Signoff Dashboard from FPV GUI

1. Click **Signoff Dashboard** from the tasklist as shown in the following figure:



vacuity	witness	engine		
● 4		e1	assert	00:00:15
● 3		s1	assert	00:00:01
● 5		e1	assert	00:00:27
● 1		e1	assert	00:00:29
		e1	assert	00:00:29

The signoff dashboard appears as shown in the following figure:


File View Plan Exclusion Tools Window Help
Menu
Tree
Summary
Hierarchy Groups Asserts
CovSrc.1: fifo
SignoffDb
<All>
ormal_Signoff/Formal_Signoff
CovDetail
Line Toggle FSM Condition Branch Assert

Effort: Low (COI) Time: 24H Signoff Bound: -1 ✓ OC ✓ Bounded Signoff DB: vcf_signoff_db ✓ Overwrite

Name	Score
fifo	🟥 0.00%
i_fifo_bind	🟥 0.00%

Coverage Statistics

Metric	Total	Covered	Score	Uncovered	Excluded
Line	66	0	0.00%	66	0
Branch	54	0	0.00%	54	0
Total	120	0	0.00%	120	0
Average					

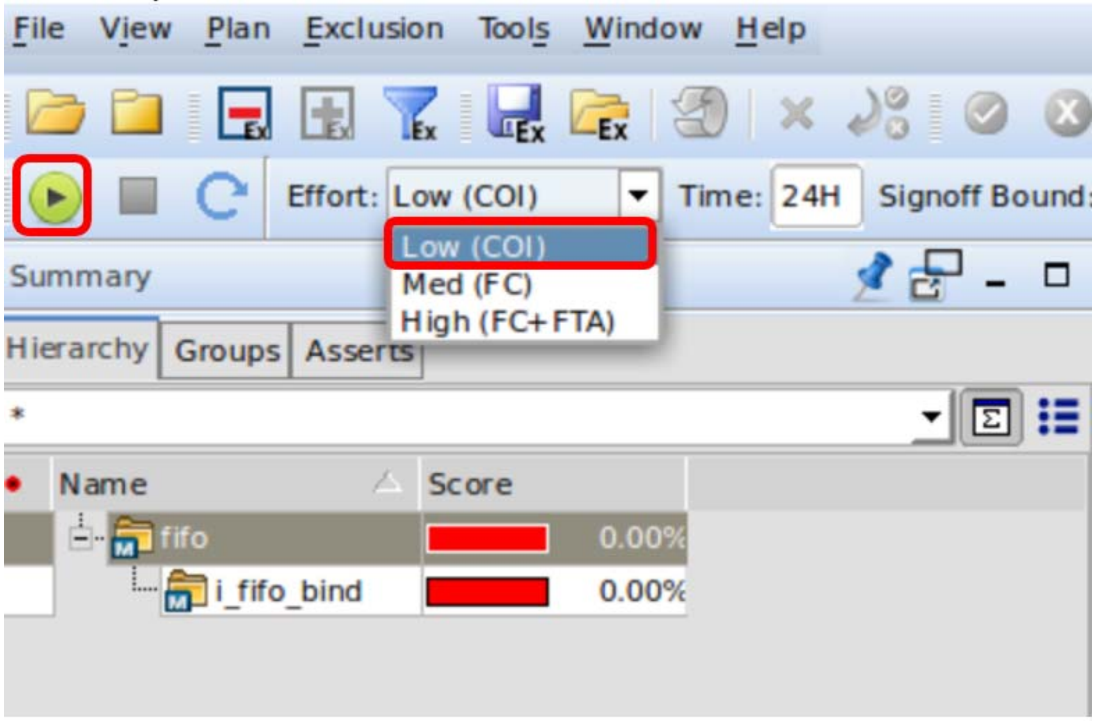
```
    always @(posedge clk or negedge resetn)
      if (!resetn)
        ptr <= {(L2D+1){1'b0}};
      else ptr <= ptr + push_hsk - pop_hsk;

    always @(posedge clk)
      if (push_hsk)
        data[0] <= data_in;

    genvar n;

    generate for (n=0;n<DEPTH-1;n++) begin
       always @(posedge clk)
         if (push_hsk)
           data[n+1] <= data[n];
    end
     endgenerate


    always @(posedge clk)
      if (pop_hsk)
        data_o_nxt <= data[ptr-1];

    assign data_out = data_o_nxt;

//    assign full = (ptr == (DEPTH)); // BUG
    assign full = (ptr == (DEPTH-1)); // FIX

    assign empty = (ptr == 0);
```

Category	Coverage
Block	🟥 0.00%
Statement	🟥 0.00%

Status	Name	Line No	Source
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)
✗		48	if (push_hsk)

Signoff Step in "Low" effort

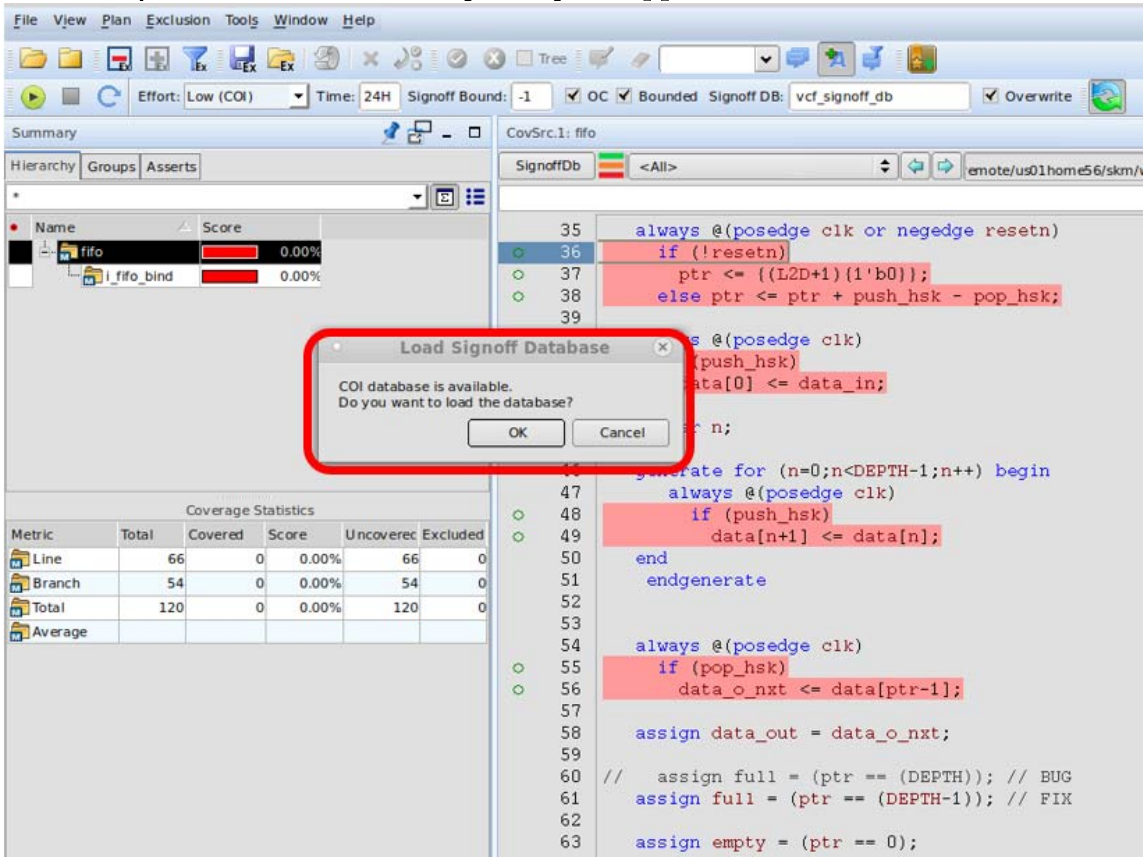
Select effort **Low (COI)** as shown in the following figure, and click the play button to run COI analysis.



Effort: Low (COI) Time: 24H Signoff Bound:

Name	Score
fifo	🟥 0.00%
i_fifo_bind	🟥 0.00%

Once the COI analysis is done, the following dialog box appears.



Effort: Low (COI) Time: 24H Signoff Bound: -1 ✓ OC ✓ Bounded Signoff DB: vcf_signoff_db ✓ Overwrite

Name	Score
fifo	🟥 0.00%
i_fifo_bind	🟥 0.00%

Load Signoff Database

COI database is available.
Do you want to load the database?

OK Cancel

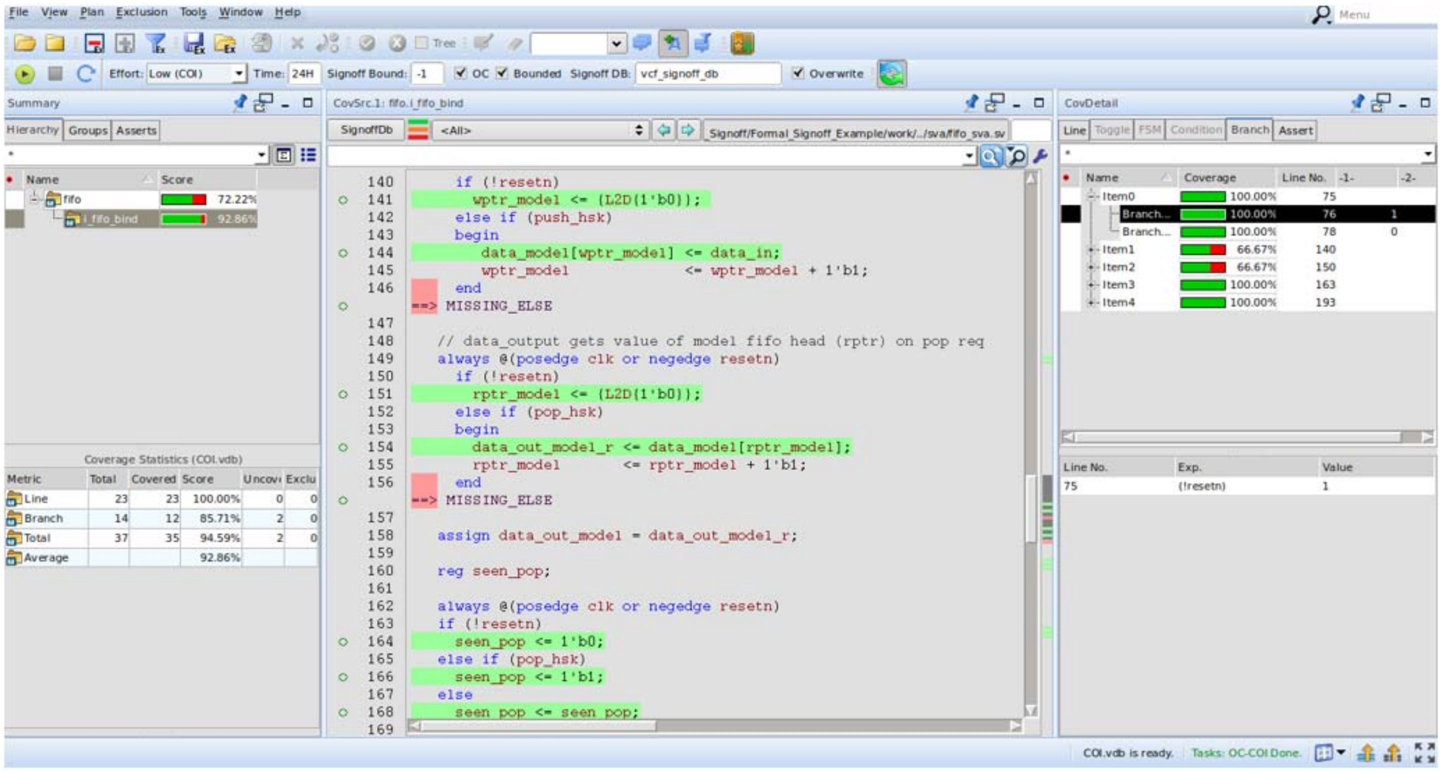
Coverage Statistics

Metric	Total	Covered	Score	Uncoverec	Excluded
Line	66	0	0.00%	66	0
Branch	54	0	0.00%	54	0
Total	120	0	0.00%	120	0
Average					

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

s @(posedge clk)
(push_hsk)
ata[0] <= data_in;

r n;

rate for (n=0;n<DEPTH-1;n++) begin
   always @(posedge clk)
     if (push_hsk)
       data[n+1] <= data[n];
end
 endgenerate


always @(posedge clk)
  if (pop_hsk)
    data_o_nxt <= data[ptr-1];

assign data_out = data_o_nxt;

//   assign full = (ptr == (DEPTH)); // BUG
assign full = (ptr == (DEPTH-1)); // FIX

assign empty = (ptr == 0);
```

Reviewing COI Coverage

Click **OK**, and COI analysis results appears as shown in the following figure:


File View Plan Exclusion Tools Window Help
Menu
Tree
Summary
Hierarchy Groups Asserts
CovSrc.1: fifo.i_fifo_bind
SignoffDb
<All>
CovDetail
Line Toggle FSM Condition Branch Assert
COI.vdb is ready.
Tasks: OC-COI Done.

Effort: Low (COI) Time: 24H Signoff Bound: -1 ✓ OC ✓ Bounded Signoff DB: vcf_signoff_db ✓ Overwrite

Signoff/Formal_Signoff_Example/work/../sva/fifo_sva.sv

Name	Score
fifo	72.22%
i_fifo_bind	92.86%

Coverage Statistics (COI.vdb)

Metric	Total	Covered	Score	Uncov	Exclu
Line	23	23	100.00%	0	0
Branch	14	12	85.71%	2	0
Total	37	35	94.59%	2	0
Average			92.86%		

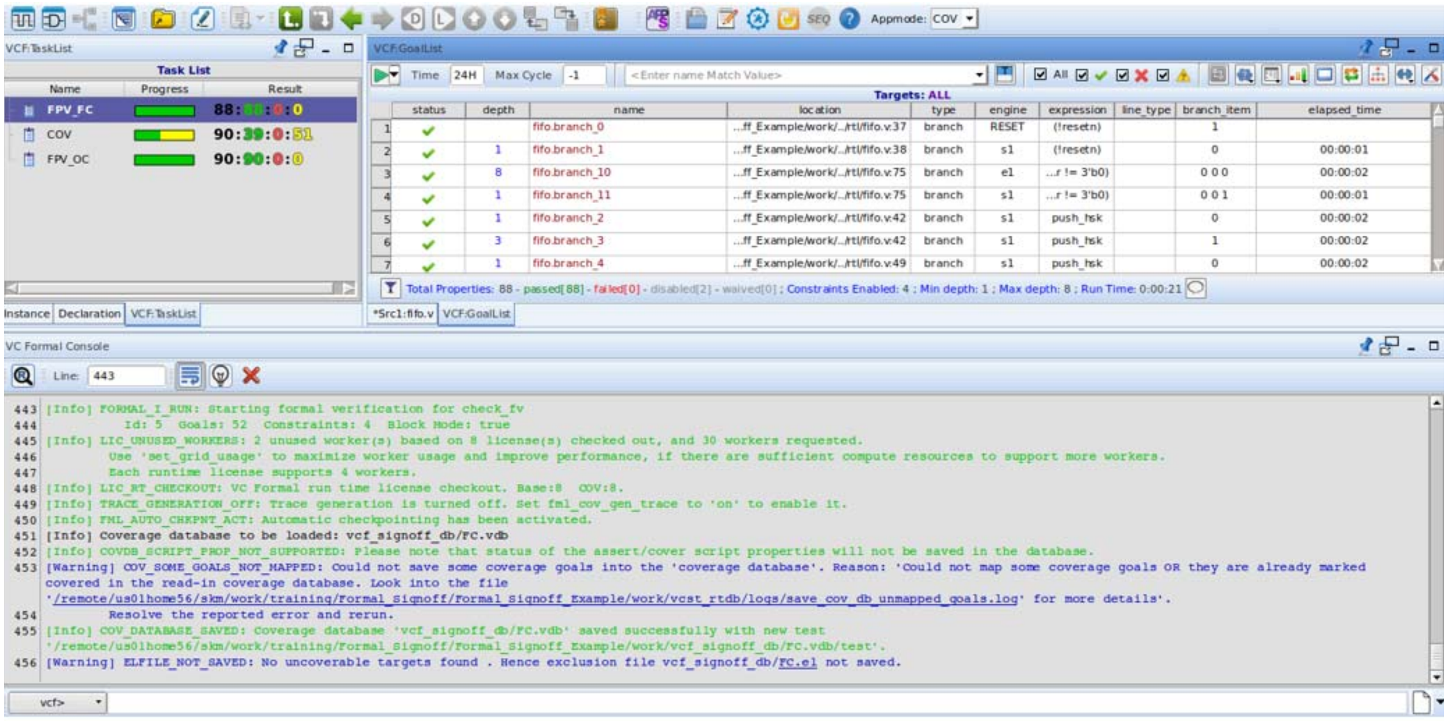
```
    if (!resetn)
      wptr_model <= {L2D{1'b0}};
    else if (push_hsk)
    begin
       data_model[wptr_model] <= data_in;
       wptr_model             <= wptr_model + 1'b1;
    end
==> MISSING_ELSE

  // data_output gets value of model fifo head (rptr) on pop req
  always @(posedge clk or negedge resetn)
    if (!resetn)
      rptr_model <= {L2D{1'b0}};
    else if (pop_hsk)
    begin
      data_out_model_r <= data_model[rptr_model];
      rptr_model       <= rptr_model + 1'b1;
    end
==> MISSING_ELSE

  assign data_out_model = data_out_model_r;

  reg seen_pop;

  always @(posedge clk or negedge resetn)
  if (!resetn)
    seen_pop <= 1'b0;
  else if (pop_hsk)
    seen_pop <= 1'b1;
  else
    seen_pop <= seen_pop;
```

Name	Coverage	Line No.	-1-	-2-
Item0	100.00%	75		
Branch...	100.00%	76	1	
Branch...	100.00%	78	0	
Item1	66.67%	140		
Item2	66.67%	150		
Item3	100.00%	163		
Item4	100.00%	193		

Line No.	Exp.	Value
75	(!resetn)	1

GUI Interactive Control in Signoff Dashboard Flow

All command, messages and detailed progress information is available on the original VCF GUI window and VCF Console.



Task List

Name	Progress	Result
FPV_FC	🟩	88:88:0:0
COV	🟩🟨	90:39:0:51
FPV_OC	🟩	90:90:0:0

Time 24H Max Cycle -1 <Enter name Match Value>

Targets: ALL

	status	depth	name	location	type	engine	expression	line_type	branch_item	elapsed_time
1	✓		fifo.branch_0	...ff_Example/work/.../rtl/fifo.v:37	branch	RESET	(!resetn)		1	
2	✓	1	fifo.branch_1	...ff_Example/work/.../rtl/fifo.v:38	branch	s1	(!resetn)		0	00:00:01
3	✓	8	fifo.branch_10	...ff_Example/work/.../rtl/fifo.v:75	branch	e1	...r != 3'b0)		0 0 0	00:00:02
4	✓	1	fifo.branch_11	...ff_Example/work/.../rtl/fifo.v:75	branch	s1	...r != 3'b0)		0 0 1	00:00:01
5	✓	1	fifo.branch_2	...ff_Example/work/.../rtl/fifo.v:42	branch	s1	push_hsk		0	00:00:02
6	✓	3	fifo.branch_3	...ff_Example/work/.../rtl/fifo.v:42	branch	s1	push_hsk		1	00:00:02
7	✓	1	fifo.branch_4	...ff_Example/work/.../rtl/fifo.v:49	branch	s1	push_hsk		0	00:00:02

Total Properties: 88 - passed[88] - failed[0] - disabled[2] - waived[0] ; Constraints Enabled: 4 ; Min depth: 1 ; Max depth: 8 ; Run Time: 0:00:21

Line: 443

```
443 [Info] FORMAL_I_RUN: Starting formal verification for check_fv
444          Id: 5  Goals: 52  Constraints: 4  Block Mode: true
445 [Info] LIC_UNUSED_WORKERS: 2 unused worker(s) based on 8 license(s) checked out, and 30 workers requested.
446        Use 'set_grid_usage' to maximize worker usage and improve performance, if there are sufficient compute resources to support more workers.
447        Each runtime license supports 4 workers.
448 [Info] LIC_RT_CHECKOUT: VC Formal run time license checkout. Base:8  COV:8.
449 [Info] TRACE_GENERATION_OFF: Trace generation is turned off. Set fml_cov_gen_trace to 'on' to enable it.
450 [Info] FML_AUTO_CHKPNT_ACT: Automatic checkpointing has been activated.
451 [Info] Coverage database to be loaded: vcf_signoff_db/FC.vdb
452 [Info] COVDB_SCRIPT_PROP_NOT_SUPPORTED: Please note that status of the assert/cover script properties will not be saved in the database.
453 [Warning] COV_SOME_GOALS_NOT_MAPPED: Could not save some coverage goals into the 'coverage database'. Reason: 'Could not map some coverage goals OR they are already marked
    covered in the read-in coverage database. Look into the file
    '/remote/us01home56/skm/work/training/Formal_Signoff/Formal_Signoff_Example/work/vcst_rtdb/logs/save_cov_db_unmapped_goals.log' for more details'.
454        Resolve the reported error and rerun.
455 [Info] COV_DATABASE_SAVED: Coverage database 'vcf_signoff_db/FC.vdb' saved successfully with new test
    '/remote/us01home56/skm/work/training/Formal_Signoff/Formal_Signoff_Example/work/vcf_signoff_db/FC.vdb/test'.
456 [Warning] ELFILE_NOT_SAVED: No uncoverable targets found . Hence exclusion file vcf_signoff_db/FC.el not saved.
```

Signoff step in "Med" effort

After COI Analysis is run, run the effort level **Med (FC)**, followed by **High (FC+FTA)** as seen in the drop-down menu in Signoff Step in "Low" effort. You can calculate the various Signoff metrics, and refine them as needed.

Handling Waivers

Analyze all the cover items in red, apply and save waivers. All the uncoverable items need to be explained and waived.

Reviewing Signoff

Review signoff metrics vis-à-vis goals mentioned in Goals for Formal Signoff, and signoff your design.

Best Practices

- The effort levels in signoff dashboard have been ordered based on their place in the signoff flow and the time taken by them to be processed.
- Review best practices for Formal Core analysis in section Best Practices, and FTA analysis in section Best Practices. They are relevant to the Med and High effort settings respectively in Formal Signoff Dashboard.
- In the case you have reached 100% Formal Core and want to skip the **Med (FC)** step, use the following variable before the **High (FC+FTA)** step.

  ```
  set_fml_var fml_signoff_high_effort_fta_only true
  ```

Performance Considerations

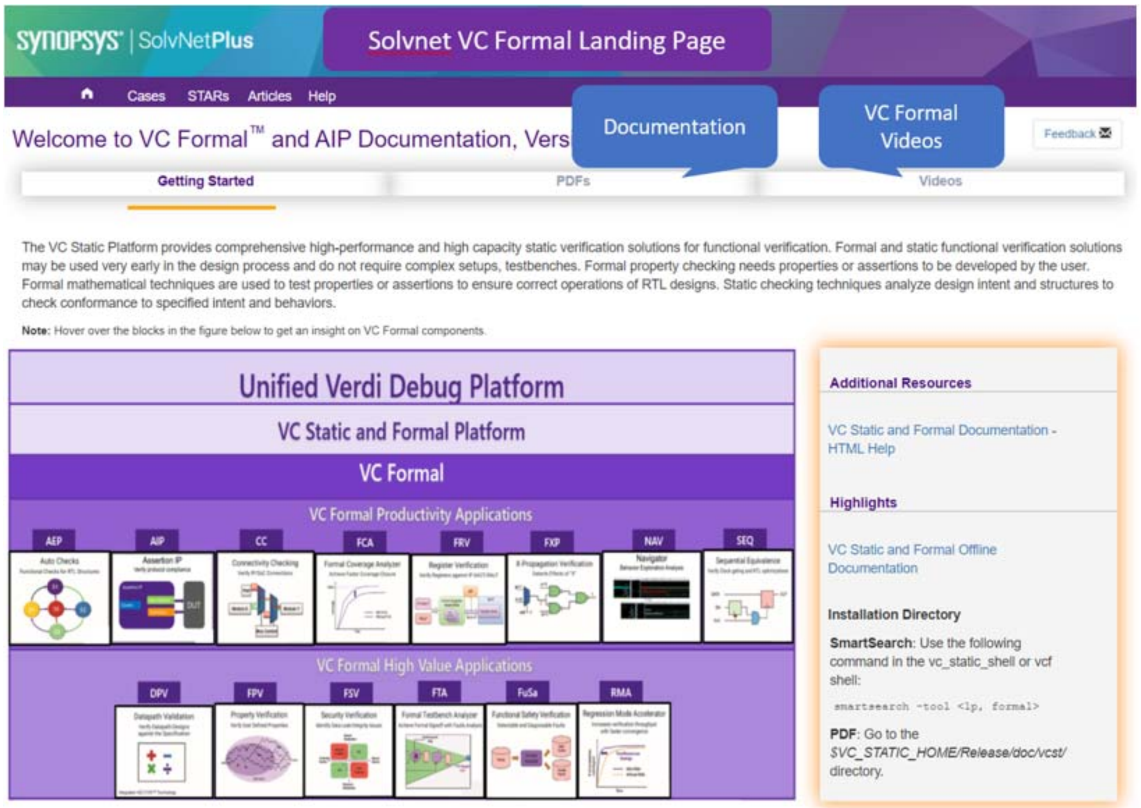
Review performance considerations for Formal Core analysis in Performance Considerations, and FTA analysis in Performance Considerations. They are relevant to the Med and High effort settings respectively in the Formal Signoff Dashboard.

9 Resources and References

VC Formal Online Resources and Documentation

The following resources are available to help you become proficient with VC Formal.

- ❖ VC Formal product page on Synopsys website.
- ❖ VC Formal product page on SolvNetPlus, which includes product documentation and video tutorials.



Welcome to VC Formal™ and AIP Documentation, Vers

The VC Static Platform provides comprehensive high-performance and high capacity static verification solutions for functional verification. Formal and static functional verification solutions may be used very early in the design process and do not require complex setups, testbenches. Formal property checking needs properties or assertions to be developed by the user. Formal mathematical techniques are used to test properties or assertions to ensure correct operations of RTL designs. Static checking techniques analyze design intent and structures to check conformance to specified intent and behaviors.

Note: Hover over the blocks in the figure below to get an insight on VC Formal components.

Additional Resources

VC Static and Formal Documentation - HTML Help

Highlights

VC Static and Formal Offline Documentation

Installation Directory

SmartSearch: Use the following command in the vc_static_shell or vcf shell:

```
smartsearch -tool <lp, formal>
```

PDF: Go to the *$VC_STATIC_HOME/Release/doc/vcst/* directory.

- ❖ VC Formal documentation is available along with the software installation: *$VC_STATIC_HOME/doc/vcst/VC_Formal_Docs*
- ❖ VC Formal past webinars include a rich set of resources to learn about various VC Formal applications.

❖ VC Formal Blog (Informal Chat to keep up with the latest in VC Formal

Hands-on With VC Formal Through Examples

In addition to the resources listed above, VC Formal provides examples along with the software installation, *$VC_STATIC_HOME/doc/vcst/examples*.

For each VC Formal application, there is a separate directory with the example design files, the read me file, and the solutions. It takes 30 minutes to go through each application example to get hands-on experience with VC Formal.

```
$VC_STATIC_HOME/doc/vcst/examples/AEP
$VC_STATIC_HOME/doc/vcst/examples/AIP
$VC_STATIC_HOME/doc/vcst/examples/CC
$VC_STATIC_HOME/doc/vcst/examples/DPV
$VC_STATIC_HOME/doc/vcst/examples/FCA
$VC_STATIC_HOME/doc/vcst/examples/FPV
$VC_STATIC_HOME/doc/vcst/examples/FRV
$VC_STATIC_HOME/doc/vcst/examples/FSV
$VC_STATIC_HOME/doc/vcst/examples/FTA
$VC_STATIC_HOME/doc/vcst/examples/FuSa
$VC_STATIC_HOME/doc/vcst/examples/FXP
$VC_STATIC_HOME/doc/vcst/examples/SEQ
```

Online and Offline VC Formal Trainings

Synopsys offers many online training options to help you become more proficient with VC Formal. Check the Synopsys Online Training & Education site regularly, or talk to your Synopsys representative if you are interested in particular trainings.

Additional Support

If you run into some tool issues or have additional questions, file a case on SolvNetPlus.

You can also contact your Synopsys representation and application engineers for additional help.