VC Formal Lab
Formal Testbench Analyzer (FTA) App
Setup and Standard Usage

Learning Objectives

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

- Set up design, properties, and verification environment
- Run interactively with Verdi GUI
- Run checks in FPV App mode
- Switch to FTA App mode and validate injected faults
- Debug and resolve non-detected faults
- Debug and resolve non-activated faults
- Run updated set of properties
- Cluster faults for easier analysis

Lab Duration: 30 minutes

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/FTA/

Directory Structure	
FPV	Lab main directory
README_VCFormal_FTA.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/

Prepare your Environment

1. Set environment variable pointing to your VC Formal installation directory:

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

2. Add path $VC_STATIC_HOME/bin to the PATH environment variable.

3. Change your working directory to FTA/run:

```
%cd FTA/run
```

Now you are ready to begin the lab.

Create a run.tcl Setup File

VC Formal has a Tcl-based command interface. It is common to start with a Tcl file to set up and compile a design. In this step, you will create a VC Formal Tcl file for the DUT, a traffic light controller, used in this lab.

4. Open file run.tcl (any arbitrary name is ok to use) using any text editor:

```
%vi run.tcl
```

5. Add command to enable FPV App mode (default when starting VC Formal):

```
set_fml_appmode FPV
```

   Before using the FTA flow, a set of properties verified with FPV is required. Therefore, we start the verification with the FPV App mode.

6. Specify DUT top level module name as Tcl variable:

```
set design traffic
```

7. Add command to compile DUT, SVA properties, and include "-inject_fault all" to specify RTL and Connectivity faults to be analyzed when switching to the FTA flow:

   The DUT files and filelist are located under directory FTA/design. The assertion and bind files are located under directory FTA/sva.

```
read_file -top $design -format sverilog -sva \
  -vcs {-f ../design/filelist +define+INLINE_SVA \
  ../sva/traffic.sva ../sva/bind_traffic.sva} \
  -inject_fault all
```

Since the DUT includes inline properties, the "+define+INLINE_SVA" string is added to the compilation command.

Note: To use unified usage model to compile design, use these commands instead of `read_file` to compile design and SVA properties:

```
set_fml_var fml_multi_step_fta true
analyze -format sverilog \
  -vcs {-f ../design/filelist +define+INLINE_SVA \
  ../sva/traffic.sva ../sva/bind_traffic.sva}
elaborate $design -sva -inject_fault all
```

8. Add clock and reset setup information to the Tcl file:

```
create_clock clk -period 100
create_reset rst -sense high
```

Add commands to initialize the DUT by holding reset active until sequential elements (latches and flip flops) values are stable.

```
sim_run -stable
sim_save_reset
```

9. Save run.tcl file and exit editor.

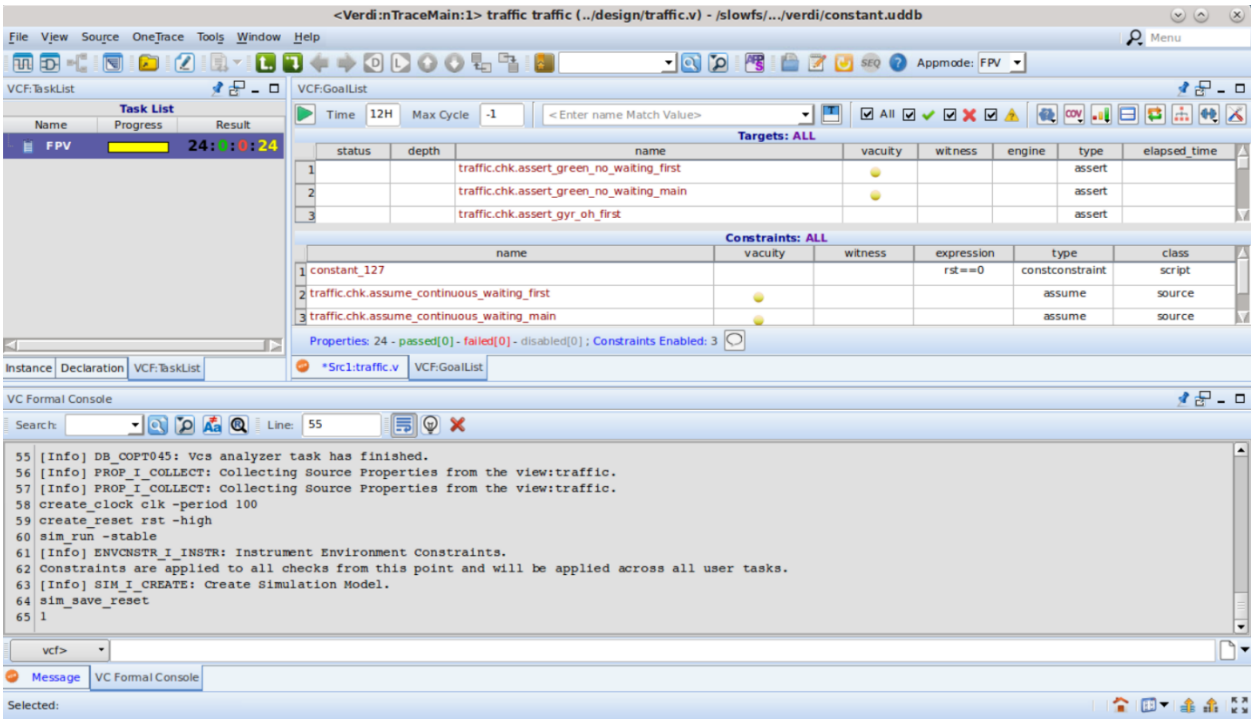
Start VC Formal in Verdi GUI Mode

10. Start the tool in Verdi GUI mode:

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

VC Formal starts in the Verdi GUI mode, with icons, tables, tabs, and windows especially designed for property verification with the FPV App. FPV is the default App mode.

Initial configuration is shown with the “VCF:TaskList” tab on the top left, the “VCF:GoalList” tab on the top right, and the “VC Formal Console” shell at the bottom.



11. Get familiar with the Verdi GUI instance:

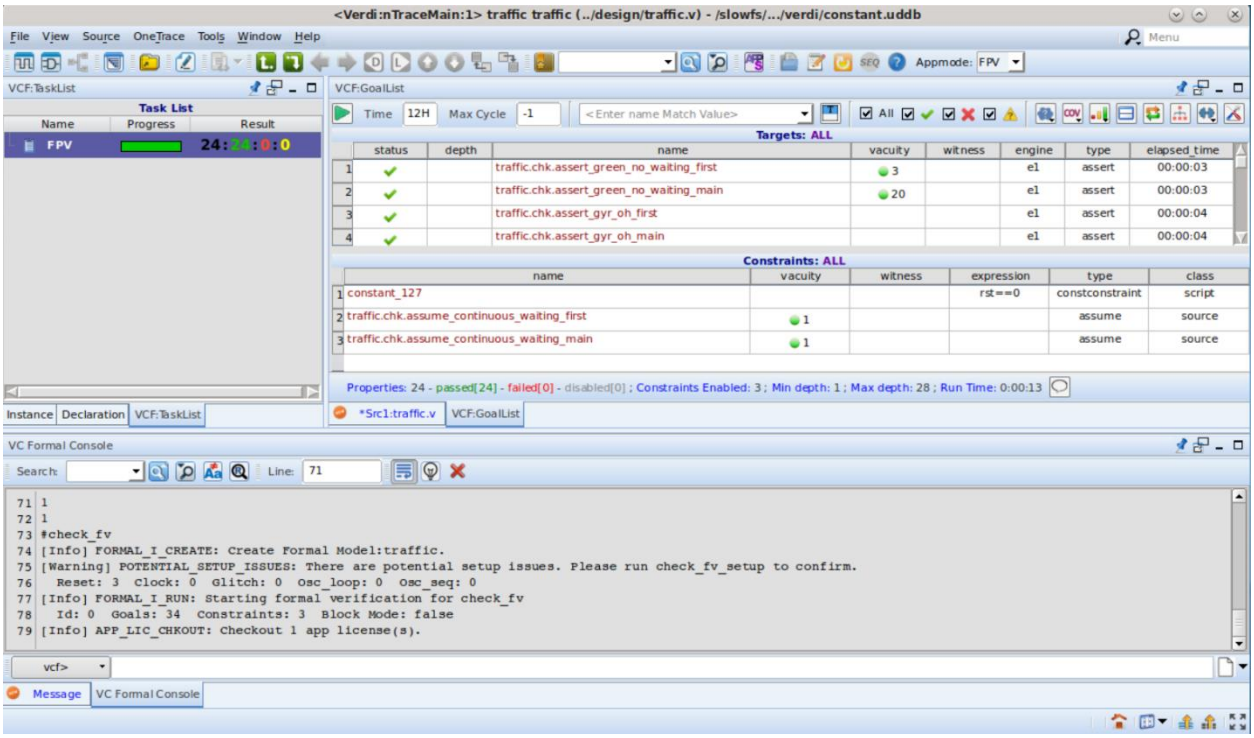
Check the source file tabs, properties to be checked under the “Targets: ALL” table as well as properties specified as constraints in the “Constraints: ALL” table.

If desired, customize the columns shown by clicking on the Customize View Settings icon at the upper right of the property table.

Run Formal Proofs and Review Results

12. Start property verification:

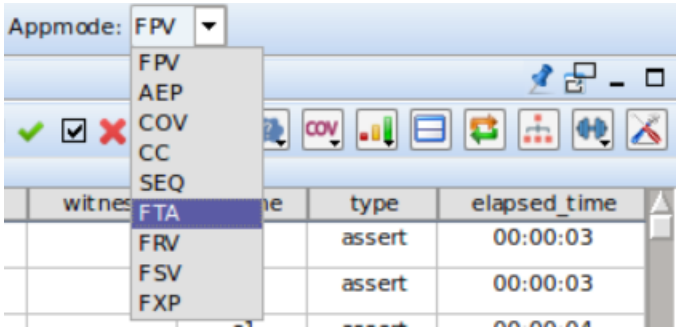
Click on the Start Check icon .



Proven properties from the FPV run will be used in the FTA flow for fault detection.

Switch to FTA App Mode and Validate Injected Faults

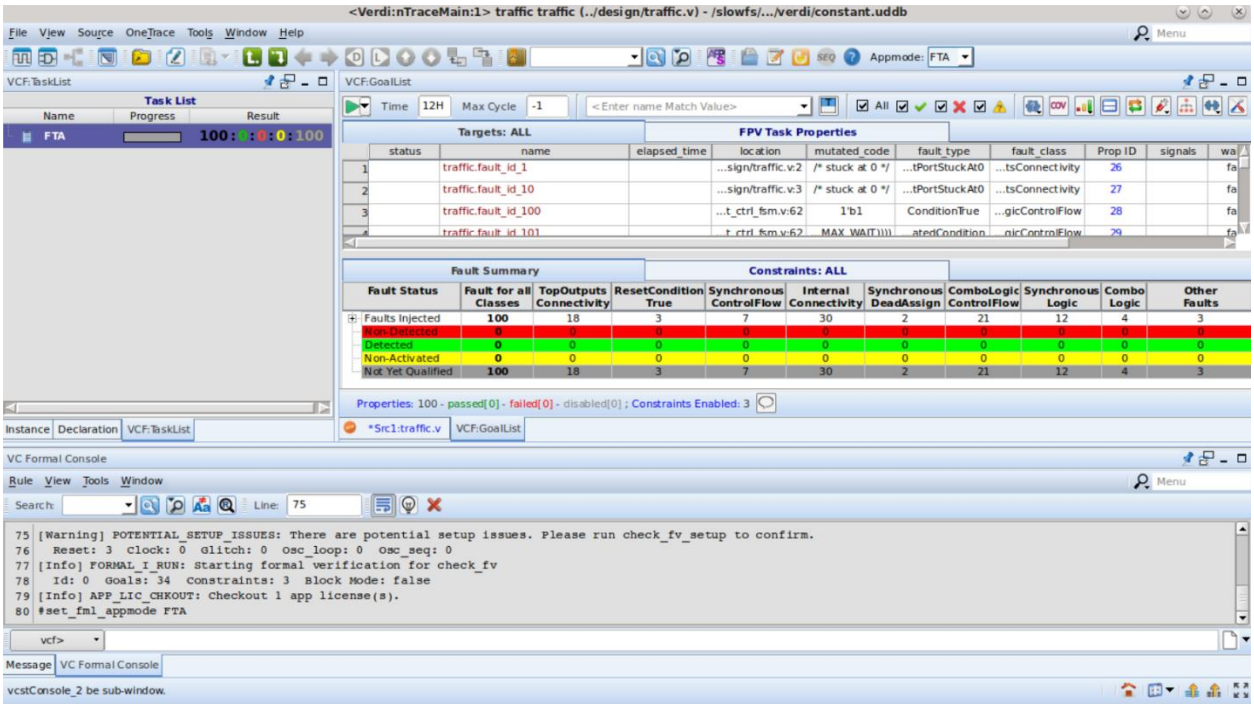
13. Switch to FTA App mode:

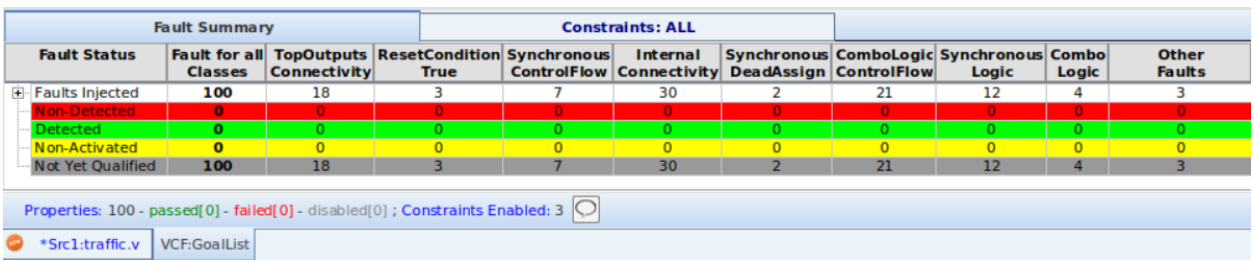


Observe that the “VCF:TaskList” and the “VCF:GoalList” tab are now showing the injected faults as targets for formal proof.

Equivalent command to switch to the FTA App mode:

```
set_fml_appmode FTA
```





14. Start property verification:

    Click on the Start Check icon .

    Note that in FTA App mode, the Start Check icon matches the following command that creates a new FPV_FTA task and starts the verification:

```
compute_fta -par_task FPV
```

15. When check completes, report results:

```
vcf> report_fv -list
```

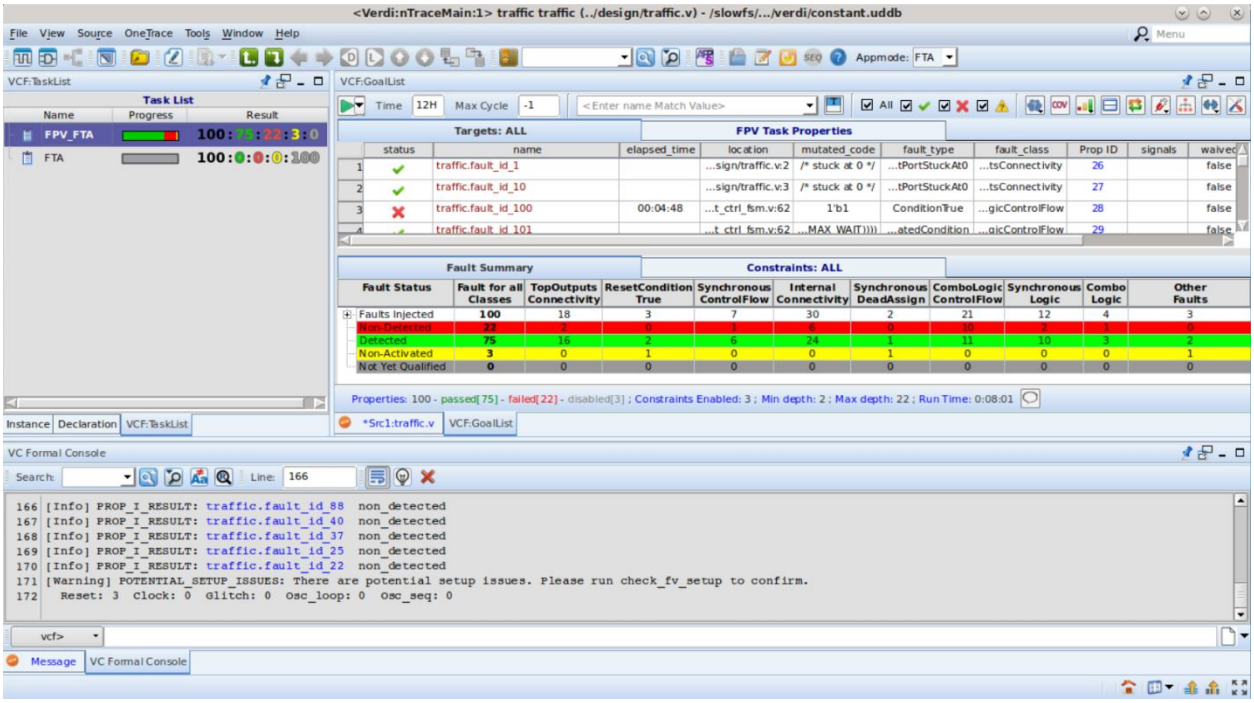
    As another option, enter run check command with callback task:

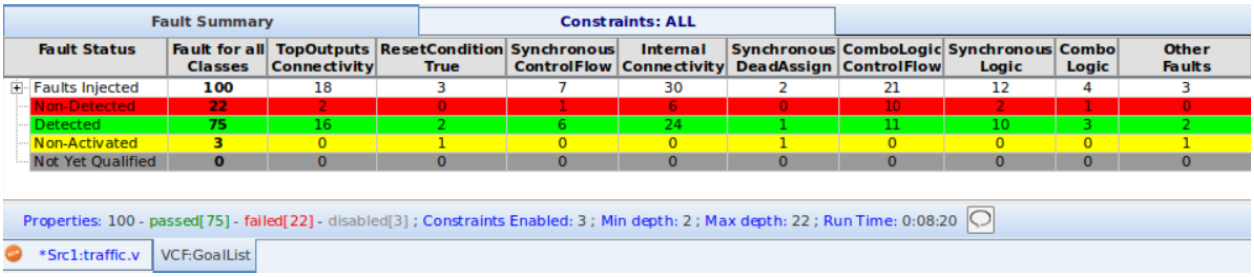
```
vcf> compute_fta -par_task FPV -run_finish {report_fv
-list > results.txt}
```

16. Filter "Targets" table to keep failed targets:

Select to view only failed targets

.





Debug Non-Detected Faults

17. Debug non-detected faults:

    Double click on the TopOutputsConnectivity x Non-Detected cell in the “Fault Summary” table showing two non-detected faults to filter the “Targets: Failure” table and show the corresponding goals.

Targets: Failure Filter by fault_class, enabled... | FPV Task Properties

	status (V)	name	elapsed_time	location	mutated_code	fault_type	fault_class (V)	Prop ID	signals	waive
1	✗	traffic.fault_id_13	00:05:56	...sign/traffic.v:3	/* stuck at 0 */	...tPortStuckAt0	...tsConnectivity	43		false
2	✗	traffic.fault_id_4	00:06:27	...sign/traffic.v:2	/* stuck at 0 */	...tPortStuckAt0	...tsConnectivity	72		false

Fault Summary | Constraints: ALL

Fault Status	Fault for all Classes	TopOutputs Connectivity	ResetCondition True	Synchronous ControlFlow	Internal Connectivity	Synchronous DeadAssign	ComboLogic ControlFlow	Synchronous Logic	Combo Logic	Other Faults
Faults Injected	100	18	3	7	30	2	21	12	4	3
Non-Detected	22	2	0	1	6	0	10	2	1	0
Detected	75	16	2	6	24	1	11	10	3	2
Non-Activated	3	0	1	0	0	1	0	0	0	1
Not Yet Qualified	0	0	0	0	0	0	0	0	0	0

Properties: 100 - passed[75] - failed[22] - disabled[3] ; Constraints Enabled: 3 ; Min depth: 2 ; Max depth: 22 ; Run Time: 0:08:20

*Src1:traffic.v | VCF:GoalList

Double click on the name for property “traffic.fault_id_13” to open the highlighted source code window with the color red identifying the location of Non-Detected faults.

Go To:

```
1  module traffic(clk, rst,
2                 green_main, yellow_main, red_main,
3                 green_first, yellow_first, red_first,
4                 waiting_main, waiting_first);
5
6  input clk;
7  input rst;
8  input waiting_main;
9  input waiting_first;
10 output green_main, yellow_main, red_main,
11        green_first, yellow_first, red_first;
12
13 //parameter MAX_WAIT = 4;
14
```

13 fault_check /* stuck at 0 */
14 fault_check /* stuck at 1 */
15 fault_check /* port value negated */

*Src1:traffic.v | VCF:GoalList | FaultsInSrc:traffic.v ×

The fault where signal “yellow_first” is stuck at 1’b0 is not detected by the current set of properties. The same stuck-at-0 fault for signal “yellow_main” also remains not detected.

Checking signals “{green_first, red_first, yellow_first}” in the DUT, we can see that at least one of them is asserted. Therefore, we can make the $onehot0 property stronger by using $onehot instead in file traffic.sva.

18. Modify properties:

Click on “Edit Source File” icon and modify the following assertions.

Original assertions:

```
// 6)   Red, green and yellow mutex
assert_gyr_oh_first: assert property ( @(posedge clk)
     $onehot0({green_first,red_first,yellow_first}));
assert_gyr_oh_main: assert property ( @(posedge clk)
     $onehot0({green_main,red_main,yellow_main}));
```

Modified assertions:

```
// 6)   Red, green and yellow mutex
assert_gyr_oh_first: assert property ( @(posedge clk)
     $onehot({green_first,red_first,yellow_first}));
assert_gyr_oh_main: assert property ( @(posedge clk)
     $onehot({green_main,red_main,yellow_main}));
```

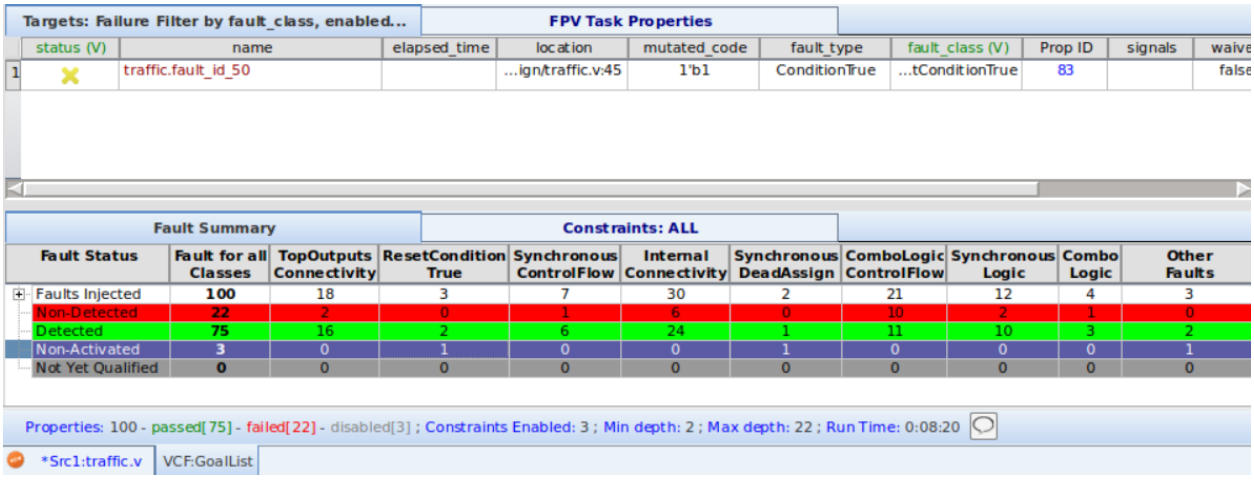
19. Save file:

    Click on the Save icon .

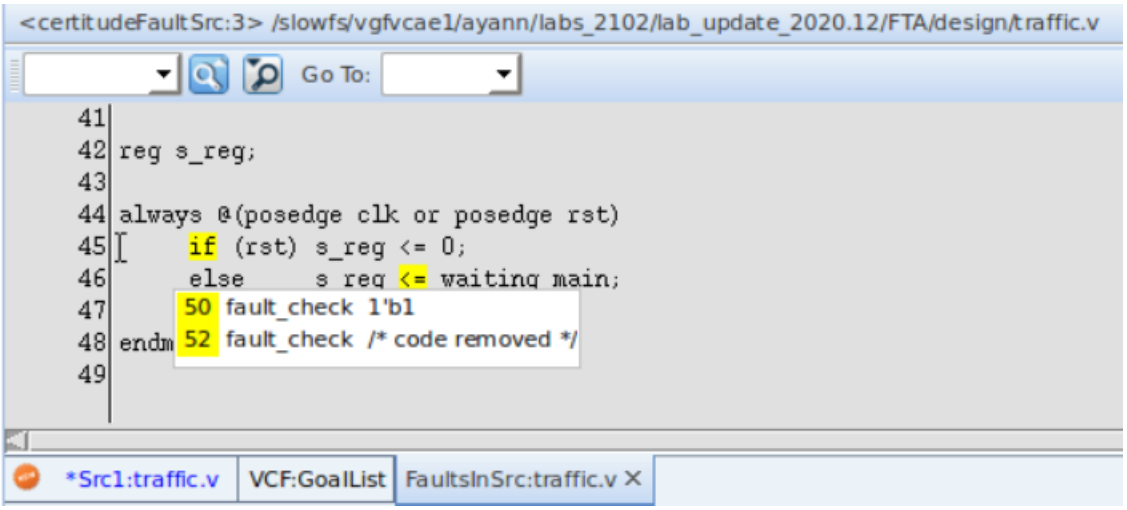
Debug Non-Activated Faults

20. Debug non-activated faults:

    Double click on the ResetConditionTrue x Non-Activated cell in the “Fault Summary” table showing one non-activated fault to filter the “Targets: Failure” table and show the corresponding goal.



    Double click on the name for property “traffic.fault_id_50” to open the highlighted source code window with the color yellow identifying the location of Non-Activated faults.



Non-activated faults indicate source code that is outside the cone of influence (COI) of the current set of properties used for the FTA flow. Adding properties that cover those signals can help eliminate non-activated faults.

21. Add properties in Tcl script:

    Click on the Edit Tcl Project File icon on the upper left and add the following commands to the Tcl file before command `sim_run`.

```
fvassert s_rst -expr {($past(rst) |-> s_reg==1'b0)}
fvassert s_wma -expr {(##1 s_reg==$past(waiting_main))}
```

    While this is a trivial example, it illustrates the FTA flow for non-activated faults. Ideally, the properties are included in the Formal Testbench and have high verification value.

22. Save edited run.tcl Tcl file:

    Click on the Save icon .

Restart the Run and Verify Fix

23. Add commands to Tcl script to enable FTA flow:

```
check_fv -block

set_fml_appmode FTA
compute_fta -par_task FPV -run_finish {report_fv
-list > results.txt}
```

24. Save edited run.tcl Tcl file:

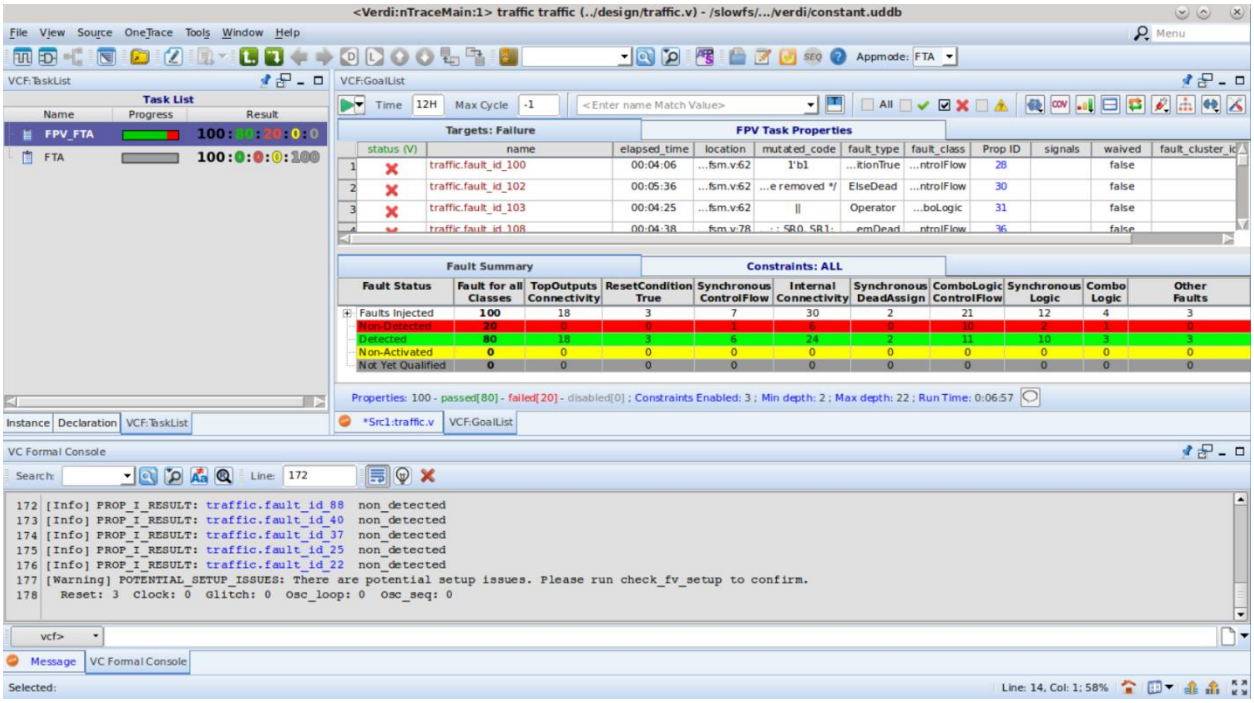
Click on the Save icon .

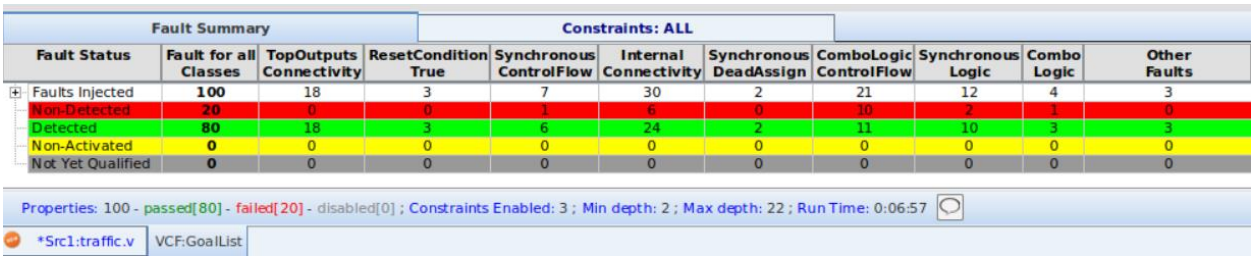
25. Restart VC Formal with the modified assertions and updated Tcl script:

    Click on the Restart VCST icon .

26. Check updated results:

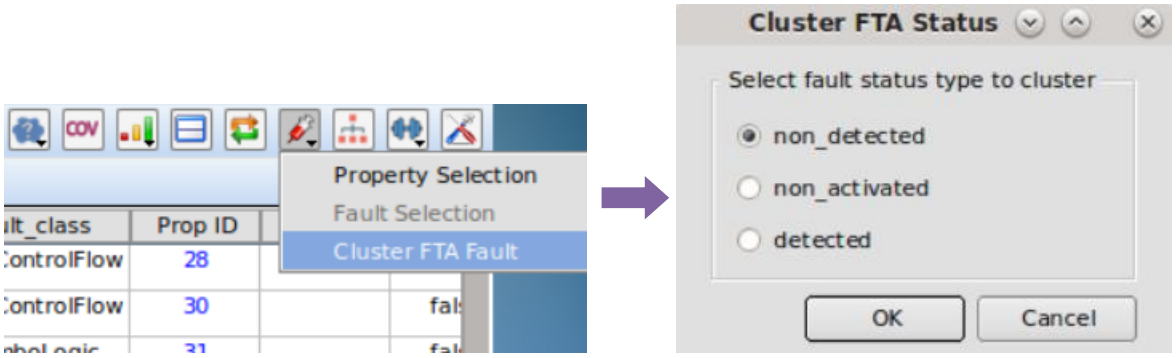
    Observe that the non-detected TopOutputsConnectivity faults and non-activated faults are now detected.





Cluster Faults for Easier Analysis

27. Cluster remaining non-detected faults to create groups of faults for easier analysis:

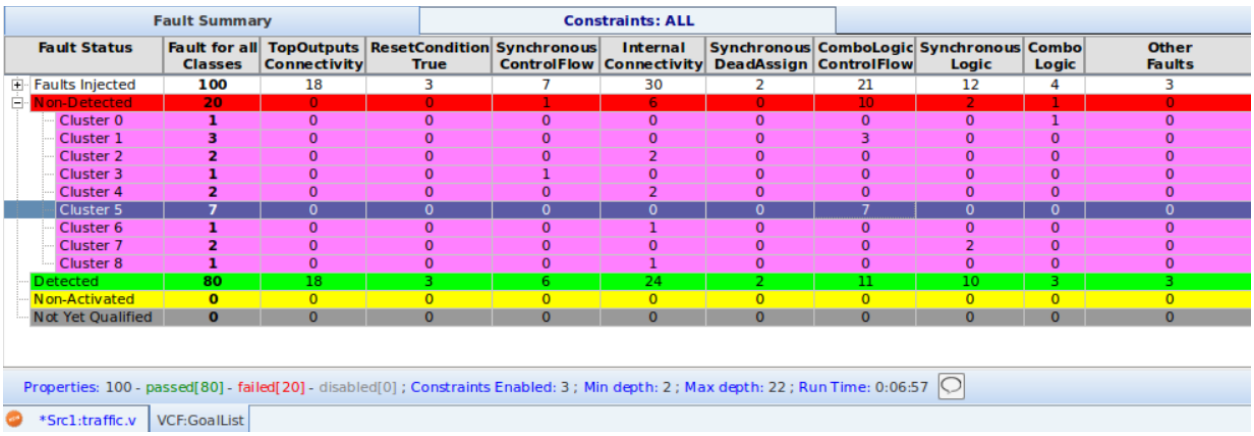


Equivalent command to cluster non-detected faults:

```
cluster_fta_faults -status non_detected
```

28. Check clusters and grouped faults:

After clustering completes, expand the list of clusters for non-detected faults.



Fault Summary | Constraints: ALL

Fault Status	Fault for all Classes	TopOutputs Connectivity	ResetCondition True	Synchronous ControlFlow	Internal Connectivity	Synchronous DeadAssign	ComboLogic ControlFlow	Synchronous Logic	Combo Logic	Other Faults
Faults Injected	100	18	3	7	30	2	21	12	4	3
Non-Detected	20	0	0	1	6	0	10	2	1	0
Cluster 0	1	0	0	0	0	0	0	0	1	0
Cluster 1	3	0	0	0	0	0	3	0	0	0
Cluster 2	2	0	0	0	2	0	0	0	0	0
Cluster 3	1	0	0	1	0	0	0	0	0	0
Cluster 4	2	0	0	0	2	0	0	0	0	0
Cluster 5	7	0	0	0	0	0	7	0	0	0
Cluster 6	1	0	0	0	1	0	0	0	0	0
Cluster 7	2	0	0	0	0	0	0	2	0	0
Cluster 8	1	0	0	0	1	0	0	0	0	0
Detected	80	18	3	6	24	2	11	10	3	3
Non-Activated	0	0	0	0	0	0	0	0	0	0
Not Yet Qualified	0	0	0	0	0	0	0	0	0	0

Clusters are faults grouped together based on class and location in the DUT. Analyzing the faults in one cluster can help derive meaningful properties that detect all faults in that cluster.

Command to list clusters and associated faults:

```
report_fta_fault_clusters
```

Double-click cluster “Cluster 5” which contains the highest number of faults of the same class.

Targets: Failure Filter by fault_class, enabled... | FPV Task Properties

	status (V)	name	elapsed_time	location	mutated_code	fault_type	fault_class (V)	Prop ID	signals	waived	fa
1	✗	traffic.fault_id_100	00:04:06	...t_ctrl_fsm.v:62	1'b1	ConditionTrue	...gicControlFlow	28		false	
2	✗	traffic.fault_id_102	00:05:36	...t_ctrl_fsm.v:62	...de removed */	ElseDead	...gicControlFlow	30		false	
3	✗	traffic.fault_id_88		...t_ctrl_fsm.v:50	...moved code*/;	CaseItemDead	...gicControlFlow	113		false	
4	✗	traffic.fault_id_89	00:04:40	...t_ctrl_fsm.v:51	...de removed*/;	CaseItemDead	...gicControlFlow	114		false	
5	✗	traffic.fault_id_90	00:06:13	...t_ctrl_fsm.v:52	!(PRIORITY)	...tedCondition	...gicControlFlow	116		false	
6	✗	traffic.fault_id_91	00:05:07	...t_ctrl_fsm.v:52	...de removed */	ElseDead	...gicControlFlow	117		false	
7	✗	traffic.fault_id_97	00:06:36	...t_ctrl_fsm.v:58	...de removed */	ElseDead	...gicControlFlow	123		false	

Double click on the name for property “traffic.fault_id_100” to open the highlighted source code window with the color red identifying the location of Non-Detected faults.

<certitudeFaultSrc:3> /slowfs/vgfvcae1/ayann/labs_2102/lab_update_2020.12/FTA/design/vlog_street_ctrl_fsm.v

Go To:

```
48 always @(*)
49 begin
50    case(state_out)
51       RESET : begin
52                if (PRIORITY) next_state = SG;
53                else next_state = SR0;
54             end
55       SR0 :
56          next_state = SR1;
57       SR1 : begin
58             if (state_cross == SY) next_state = SG;
59             else next_state = SR1;
60             end
61       SG : begin
62             if (waiting_cross && (~waiting || timer == MAX_WAIT))
63                next_state = SY;
64             99  fault_check 1'b0
65             100 fault_check 1'b1
66             101 fault_check !((waiting_cross && ((~waiting) || (timer == MAX_WAIT))))
67       SY :  102 fault_check /* code removed */
68          next_state = SR0;
69       default :
70          next_state = 5'bxxxxx;
71    endcase
72 end
```

*Src1:vlog_street_ctrl_fsm.v | VCF:GoalList | FaultsInSrc:vlog_street_ctrl_fsm.v ×