DIGITAL INDUSTRIES SOFTWARE

Questa Visualizer adds coverage analysis to the platform

Executive summary
Questa Visualizer Debug is Siemens EDA's high performance, scalable, context-aware debugger supporting the complete logic verification flow including simulation, emulation, prototyping, testbench, low-power, and assertion analysis. Visualizer recently introduced new functionality for coverage support giving the user many ways to analyze and improve coverage closure with ease. This paper will highlight some of the benefits that these kinds of features will bring to the process.

Yara Esam
Siemens EDA

SIEMENS

Contents

Introduction	3
Overview of visualizer coverage	3
Where to start?	4
Ease of navigation	5
Explore code coverage	5
Toggle coverage	7
FSM coverage analysis	9
Testplan tracker	10
Debug this!	11
Exclude all!	14
Covergroup analysis	16
Conclusion	17

Introduction

Questa Visualizer Debug is Siemens EDA's high performance, scalable, context-aware debugger supporting the complete logic verification flow including simulation, emulation, prototyping, testbench, low-power, and assertion analysis. Intuitive and easy to use, Visualizer improves debug productivity of today's complex SoCs and FPGAs. Visualizer recently introduced new functionality for coverage support giving the user many ways to analyze and improve coverage closure with ease. Starting with coverage gives you another way to diagnose problems. Visualizer provides great improvement to the coverage data representation, taking advantage of its existing rich visualization capabilities. It is built with improved capacity and performance to handle the largest coverage models. The visualization tools in Visualizer display UCDB results for both code coverage and functional coverage. Now that you have everything in Visualizer, you can go from coverage into the full power of debug in a single environment. Both designers and verification engineers can analyze and debug coverage issues with the help of design and waveform data available in Visualizer, all in one tool. This is only the first step in the overall requirements for coverage closure productivity. Siemens EDA is investing heavily in verification closure by introducing both collaboration and data analytics to the coverage closure process. This paper will highlight some of the benefits that these kinds of features will bring to the process.

Overview of visualizer coverage

The Visualizer Debug Environment provides two invocation modes for analyzing coverage results using the unified coverage database (UCDB) obtained from Questa simulation:

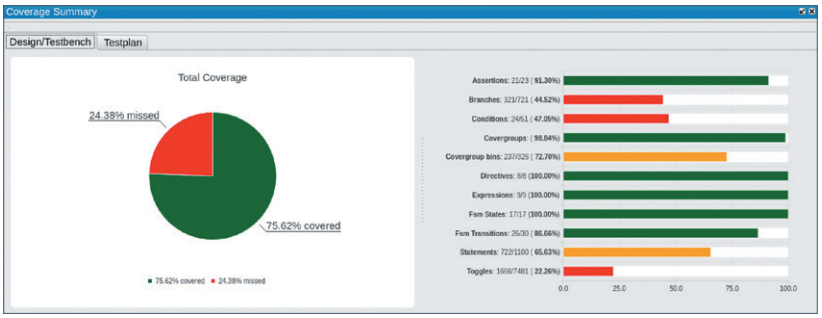


Figure 1. Coverage summary window.

- **Coverage View Mode** allows accessing all coverage analysis windows
- **Coverage Debugging** mode allows accessing all coverage analysis windows in addition to all debugging capabilities within Visualizer

Where to start?

Visualizer coverage windows provide a variety of ways to help you start exploring the coverage statistics of your design. To get started, you can use the coverage summary window to see an overview of coverage types across your design as shown in figure 1. It displays two types of color-coded charts that provide a graphical overview of the entire coverage space. The left side shows a pie chart that represents total coverage percentages for covered (green) and missed (red) bins. The right side of the window shows bar graphs for the hit bin count, total bin count, and total coverage percentage for each cover type. Both code and functional coverage types in the given design are listed.

To analyze coverage per design hierarchy, you can start with the design coverage window shown in figure 2. It displays a textual listing of the hierarchical structure of your design along with columns of statistics for code coverage types (statement, branch, toggle, FSM, condition, and expression) as well as functional coverage types (covergroups, assertions, and directives) provided for the hierarchical instances shown in the left column (design hierarchy).

You can also discover the coverage distribution across your design units with the help of Visualizer's design units coverage window as shown in figure 3. It shows all the design units in a listed format with local coverage calculated and displayed for each design unit.

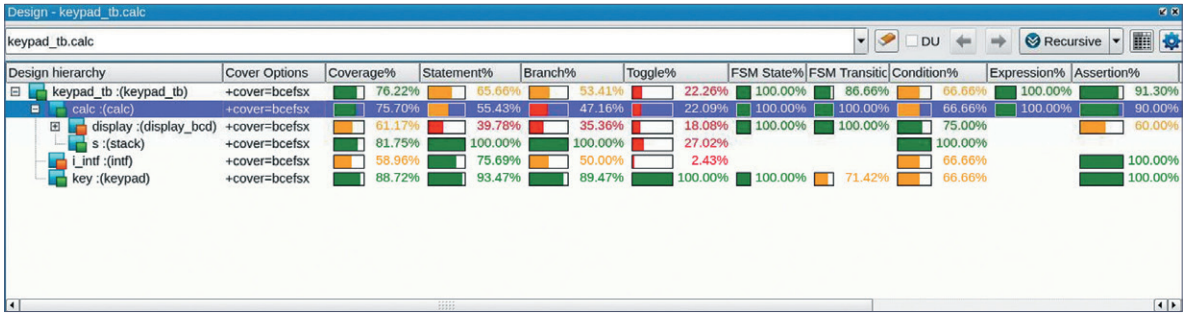


Figure 2. Design window.

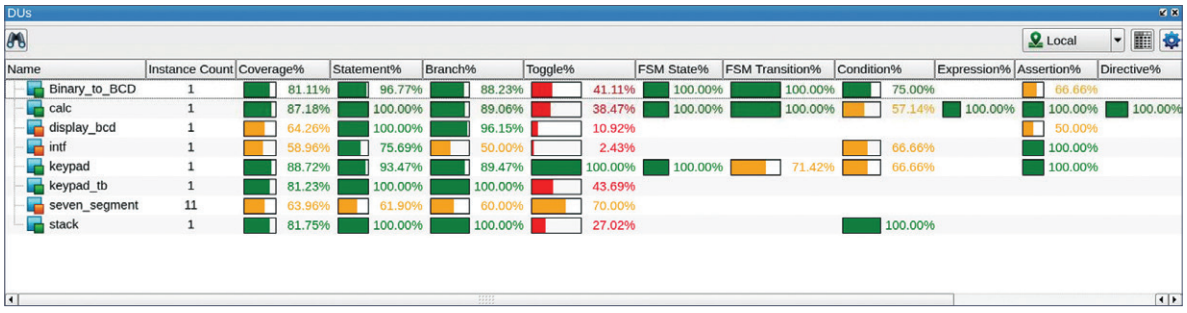


Figure 3. Design units (DUs) window.

Ease of navigation

By your first look, you can pick your critical coverage areas. For the example in Figure 1, they mainly exist in branch, toggle, and condition coverage (code coverage) and if you need a deeper analysis for a particular coverage type, you can simply double-click on the horizontal bar for a cover item to navigate to the specific coverage window related to that cover item. Double-clicking in Visualizer provides an easy way to navigate across the different coverage and source windows. Double-click navigation keeps all of your opened windows in sync as we will see later in this paper.

Explore code coverage

One of the great improvements in Visualizer is the ability to see multiple coverage types simultaneously. The code coverage outline window (figure 4) shows all covered (executed), uncovered (missed), excluded statements, branches, conditions, expressions, FSM states and transitions in one window, along with signals that have and have not been toggled for the instance you select from the design window (*Keypad_tb.calc* in figure 2).

One of the most powerful features in Visualizer coverage is the customization options such as filtered searching and editing the column's visibility which is available in all Visualizer coverage windows. The window toolbar buttons provide filtering by covered items (green check) only, missed items (red X) only and disable the excluded items (E), all for a specific coverage type or for all types.

We are going to analyze the *condition* coverage for a specific instance to decide to either fix the code or simply exclude the unwanted items.

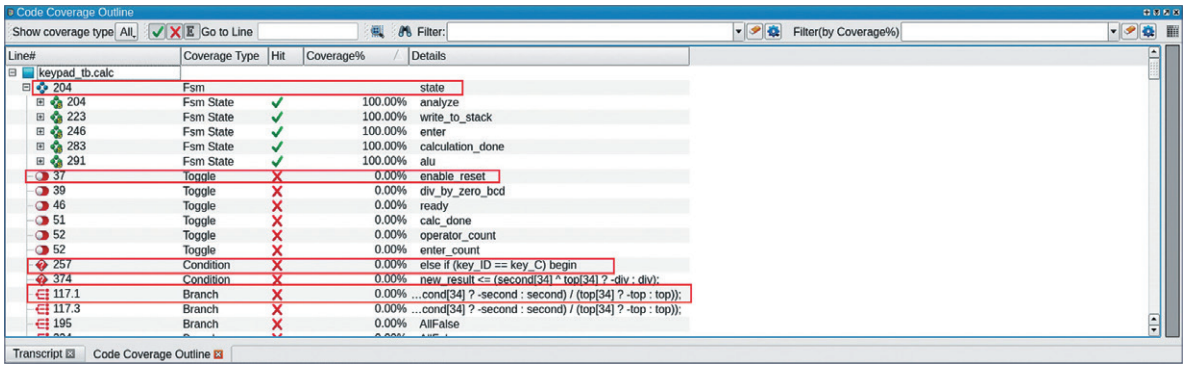


Figure 4. Code coverage outline window.

The following figure shows the missed condition coverage items for the "calc" instance selected from the design window (figure 2). We will filter the view to show condition coverage only. Now double-clicking on the condition under analysis (line 374) causes the source code window to get highlighted and the coverage details window to be opened for more detailed information on the condition item.

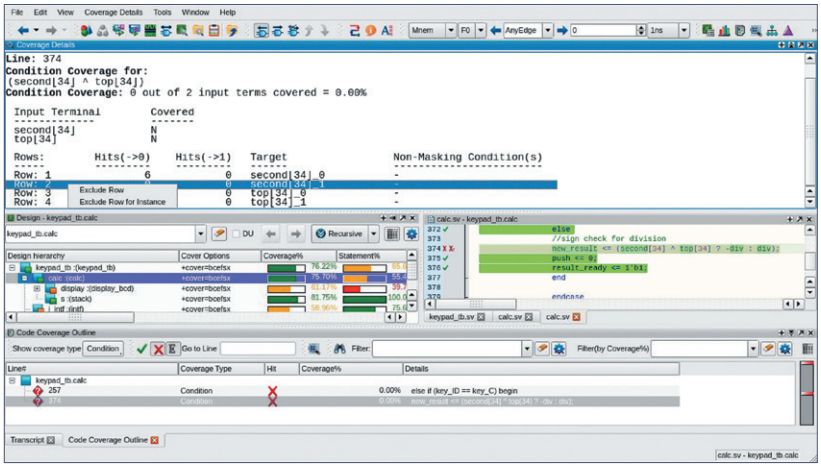


Figure 5. Source window is synced with code coverage window.

For further analysis on the condition at line 374, and from the coverage details window, we can see that the "second" and "top" signals need to be 1 to cover this item, and per the design, the "second" signal will never have a negative value, so we can exclude "row 2" from the coverage details window.

Now we have the excluded item in the pending exclusion window as shown in figure 6. We are going to explore this window in a later section.

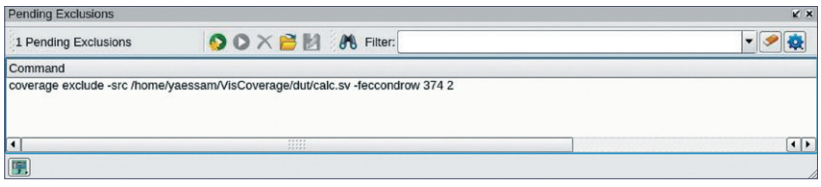


Figure 6. Pending exclusions window.

In figure 7 you can see the "X" coverage indicator at line 374. It shows the multiple uncovered types in this line as below. It indicates that condition and true branch coverage are not covered!

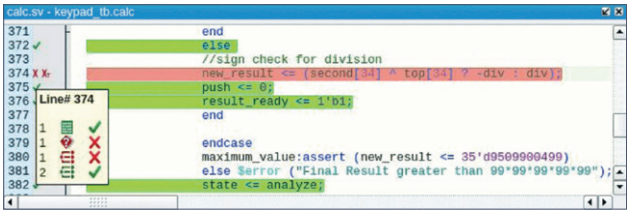


Figure 7. Multiple uncovered typed.

To analyze this line further, you can go to the code coverage window and write the line number "374" in the "go to line" box to show all uncovered coverage types at this line, the first uncovered coverage type at this line will be highlighted as shown in figure 8.

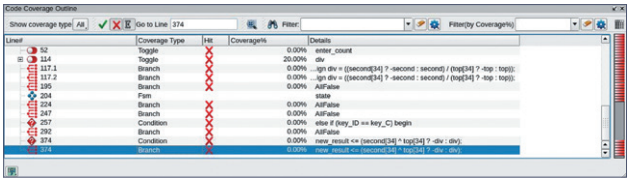


Figure 8. Line highlighting in code coverage window.

Double clicking on the branch coverage type for line 374 in the code coverage window opens the coverage details window for that type for more analysis as shown in figure 9. This shows how many times the line is activated (167) and how many times the branch expression evaluated to true (0).

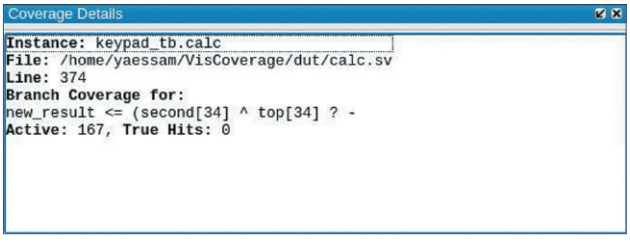
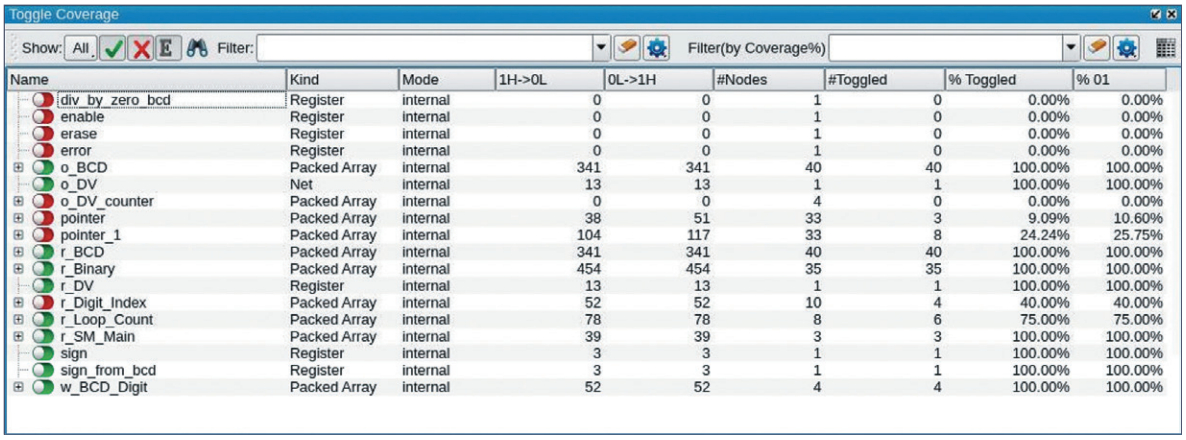


Figure 9. Branch coverage information in the coverage details window.

Toggle coverage

Toggle coverage analysis can be done on a separate window as shown in figure 10.

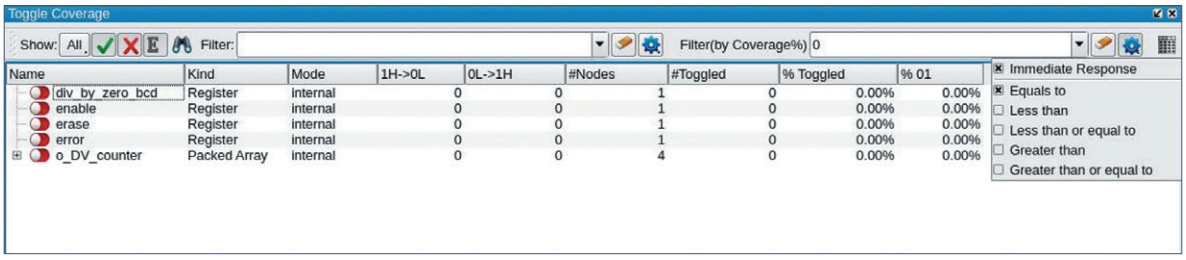
This window displays all variables and their toggle percentage along with transition counts. Many toolbar options help in customizing your display.



Name	Kind	Mode	1H->0L	0L->1H	#Nodes	#Toggled	% Toggled	% 01
div_by_zero_bcd	Register	internal	0	0	1	0	0.00%	0.00%
enable	Register	internal	0	0	1	0	0.00%	0.00%
erase	Register	internal	0	0	1	0	0.00%	0.00%
error	Register	internal	0	0	1	0	0.00%	0.00%
o_BCD	Packed Array	internal	341	341	40	40	100.00%	100.00%
o_DV	Net	internal	13	13	1	1	100.00%	100.00%
o_DV_counter	Packed Array	internal	0	0	4	0	0.00%	0.00%
pointer	Packed Array	internal	38	51	33	3	9.09%	10.60%
pointer_1	Packed Array	internal	104	117	33	8	24.24%	25.75%
r_BCD	Packed Array	internal	341	341	40	40	100.00%	100.00%
r_Binary	Packed Array	internal	454	454	35	35	100.00%	100.00%
r_DV	Register	internal	13	13	1	1	100.00%	100.00%
r_Digit_Index	Packed Array	internal	52	52	10	4	40.00%	40.00%
r_Loop_Count	Packed Array	internal	78	78	8	6	75.00%	75.00%
r_SM_Main	Packed Array	internal	39	39	3	3	100.00%	100.00%
sign	Register	internal	3	3	1	1	100.00%	100.00%
sign_from_bcd	Register	internal	3	3	1	1	100.00%	100.00%
w_BCD_Digit	Packed Array	internal	52	52	4	4	100.00%	100.00%

Figure 10. Toggle coverage window.

You can easily play with the filters provided to show only the nodes you want to analyze (the uncovered or partially covered nodes) by writing “0” in the “filter (by coverage%)” box and modify the filter options (via the “gear” icon) to be "equals to", as shown in figure 11.



Name	Kind	Mode	1H->0L	0L->1H	#Nodes	#Toggled	% Toggled	% 01
div_by_zero_bcd	Register	internal	0	0	1	0	0.00%	0.00%
enable	Register	internal	0	0	1	0	0.00%	0.00%
erase	Register	internal	0	0	1	0	0.00%	0.00%
error	Register	internal	0	0	1	0	0.00%	0.00%
o_DV_counter	Packed Array	internal	0	0	4	0	0.00%	0.00%

Figure 11. Toggle coverage filter options.

Filter by variable types using the "show" drop down menu as shown in figure 12.

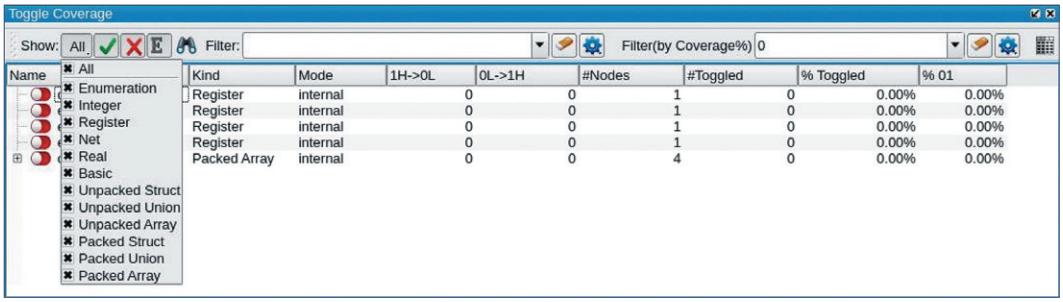


Figure 12. Toggle coverage shows drop down menu.

Customize the column visibility as shown in figure 13.

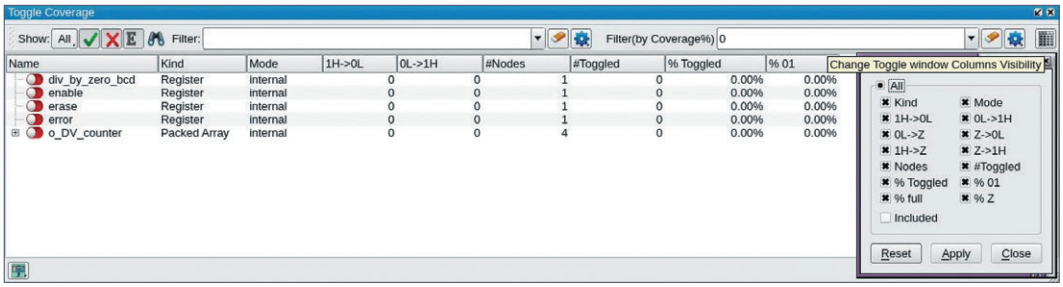


Figure 13. Toggle coverage column visibility.

Moreover, as we examined before, double-clicking on any toggle item highlights it in the source window and opens the coverage details window for more information.

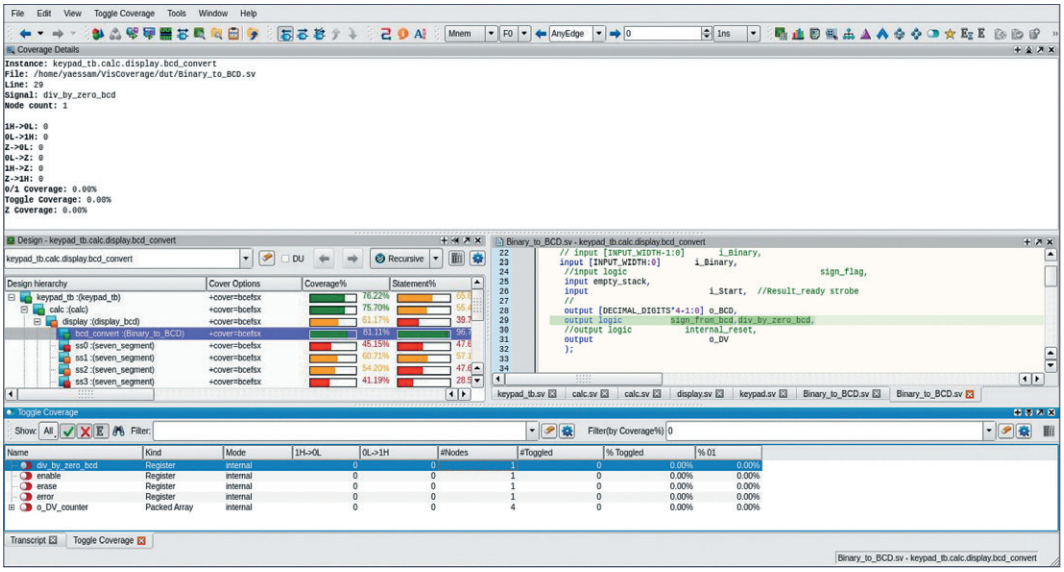


Figure 14. Coverage windows syncing.

You can exclude any toggle node either by the RMB menu in the toggle window or the RMB menu in the source window as shown in figure 15.

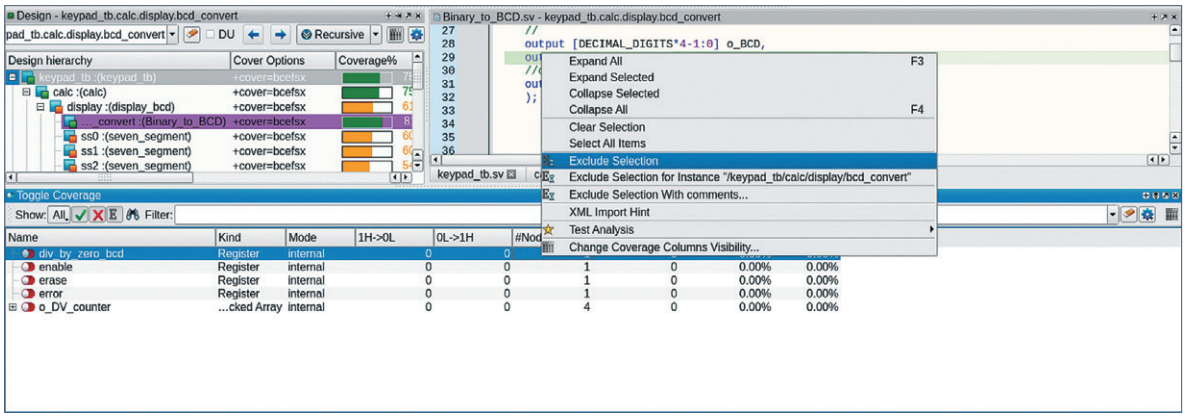


Figure 15. Exclude toggle node by RMB in source window.

FSM coverage analysis

Using the summary coverage window shown in figure 1, you can see that the 86.66% coverage number means that there are some missing FSM transitions. Double-clicking on the FSM transitions horizontal bar takes you to the FSM List window. It provides a list of all the finite state machines that are present in the design at one place, and by double-clicking on the FSM with missing transitions, the FSM coverage window and coverage details window are opened as shown in figure 16.

The FSM coverage window and the source window show all covered states and transitions in green while all the missing ones are in red. Double clicking on any FSM state/transition highlights it in the source window. You can easily exclude the undesired transitions using the RMB menu.

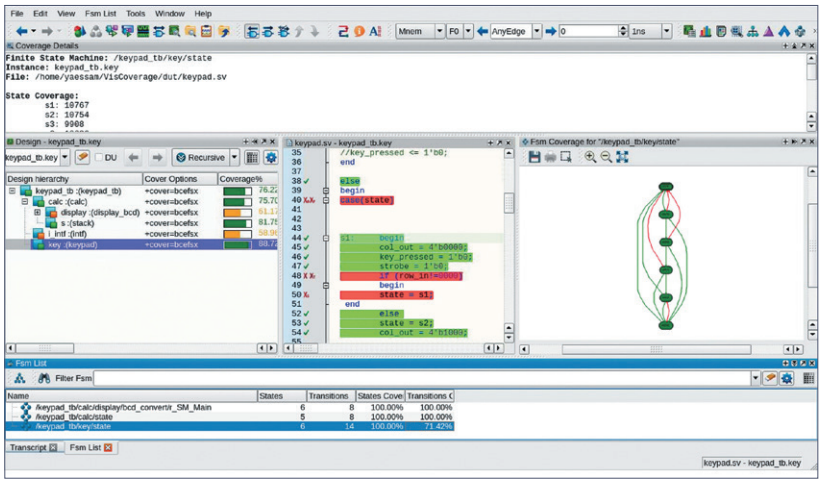


Figure 16. FSM list and coverage windows.

The coverage details window shows all the states and transitions as shown in figure 17. All uncovered transitions are shown with "0" coverage.

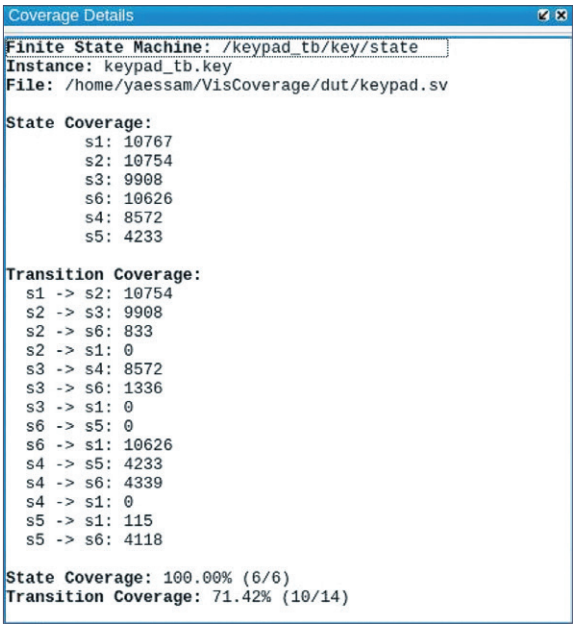
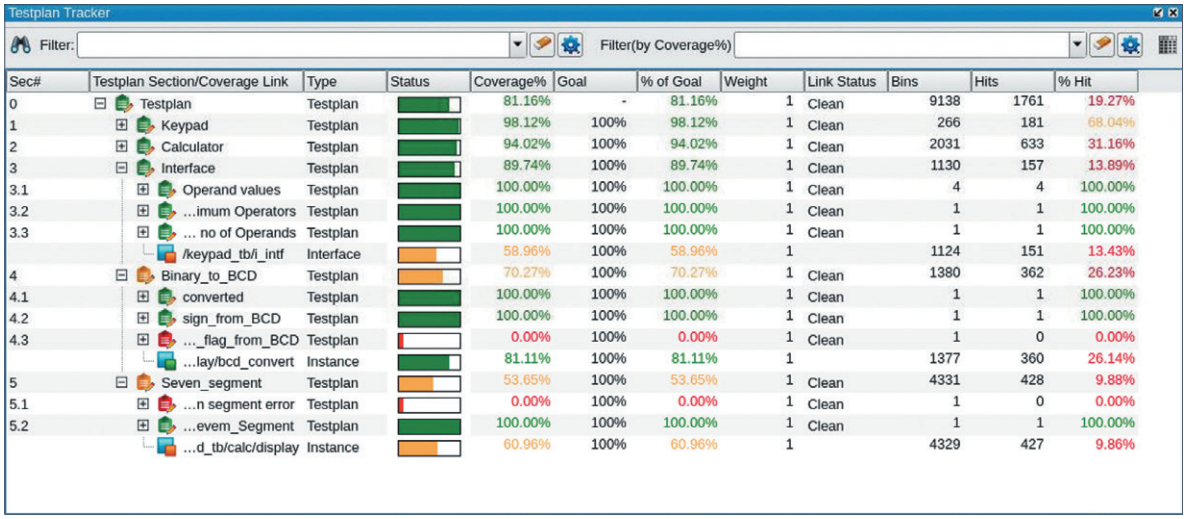


Figure 17. Coverage details window for FSM analysis.

Testplan tracker

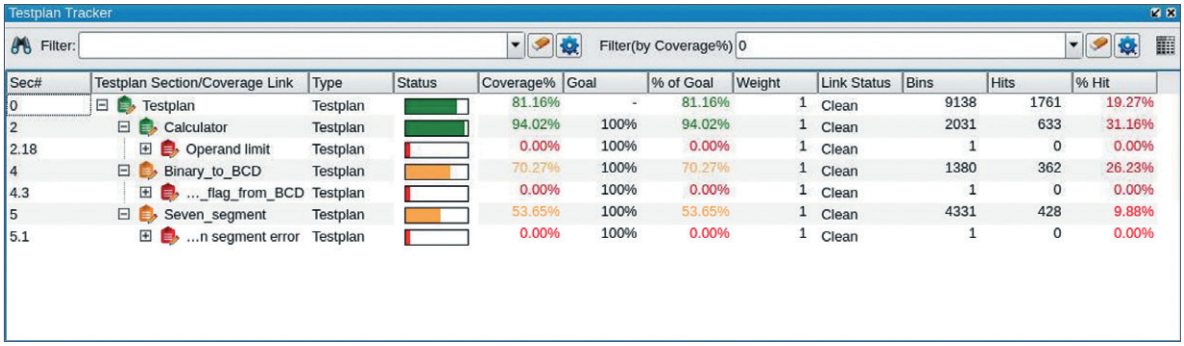
If you are using a testplan-driven coverage flow, you will most likely be interested in tracking the problematic sections in your testplan. You can start your analysis from the testplan tracker window, shown in figure 18.



Sec#	Testplan Section/Coverage Link	Type	Status	Coverage%	Goal	% of Goal	Weight	Link Status	Bins	Hits	% Hit
0	Testplan	Testplan		81.16%	-	81.16%	1	Clean	9138	1761	19.27%
1	Keypad	Testplan		98.12%	100%	98.12%	1	Clean	266	181	68.04%
2	Calculator	Testplan		94.02%	100%	94.02%	1	Clean	2031	633	31.16%
3	Interface	Testplan		89.74%	100%	89.74%	1	Clean	1130	157	13.89%
3.1	Operand values	Testplan		100.00%	100%	100.00%	1	Clean	4	4	100.00%
3.2	...imum Operators	Testplan		100.00%	100%	100.00%	1	Clean	1	1	100.00%
3.3	... no of Operands	Testplan		100.00%	100%	100.00%	1	Clean	1	1	100.00%
	/keypad_tb/i_intf	Interface		58.96%	100%	58.96%	1		1124	151	13.43%
4	Binary_to_BCD	Testplan		70.27%	100%	70.27%	1	Clean	1380	362	26.23%
4.1	converted	Testplan		100.00%	100%	100.00%	1	Clean	1	1	100.00%
4.2	sign_from_BCD	Testplan		100.00%	100%	100.00%	1	Clean	1	1	100.00%
4.3	..._flag_from_BCD	Testplan		0.00%	100%	0.00%	1	Clean	1	0	0.00%
	...lay/bcd_convert	Instance		81.11%	100%	81.11%	1		1377	360	26.14%
5	Seven_segment	Testplan		53.65%	100%	53.65%	1	Clean	4331	428	9.88%
5.1	...n segment error	Testplan		0.00%	100%	0.00%	1	Clean	1	0	0.00%
5.2	...evem_Segment	Testplan		100.00%	100%	100.00%	1	Clean	1	1	100.00%
	...d_tb/calc/display	Instance		60.96%	100%	60.96%	1		4329	427	9.86%

Figure 18. Testplan tracker window.

It shows testplan sections and sub-sections, providing different coverage metrics analysis (coverage percentage, bins count, etc.). Tracking uncovered sections with the least or zero coverage guides you directly to the most problematic areas in your design as shown in figure 19 (sections with zero coverage).



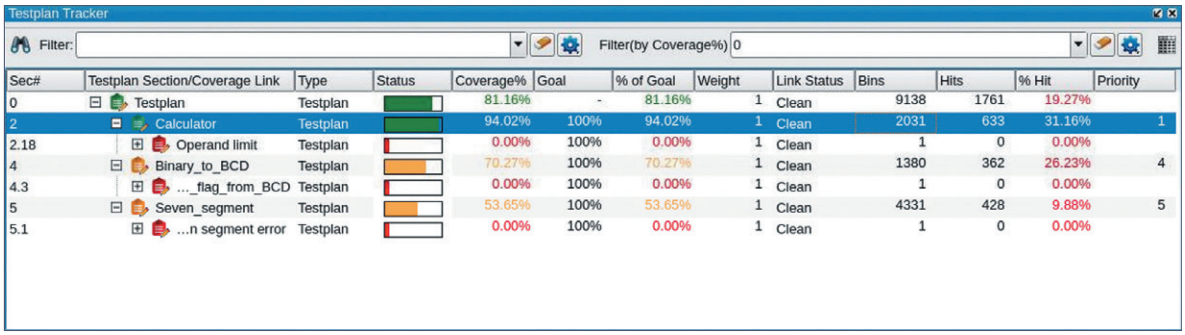
Sec#	Testplan Section/Coverage Link	Type	Status	Coverage%	Goal	% of Goal	Weight	Link Status	Bins	Hits	% Hit
0	Testplan	Testplan		81.16%	-	81.16%	1	Clean	9138	1761	19.27%
2	Calculator	Testplan		94.02%	100%	94.02%	1	Clean	2031	633	31.16%
2.18	Operand limit	Testplan		0.00%	100%	0.00%	1	Clean	1	0	0.00%
4	Binary_to_BCD	Testplan		70.27%	100%	70.27%	1	Clean	1380	362	26.23%
4.3	..._flag_from_BCD	Testplan		0.00%	100%	0.00%	1	Clean	1	0	0.00%
5	Seven_segment	Testplan		53.65%	100%	53.65%	1	Clean	4331	428	9.88%
5.1	...n segment error	Testplan		0.00%	100%	0.00%	1	Clean	1	0	0.00%

Figure 19. Testplan tracker window, zero coverage sections.

Debug this!

In this section we will go through a full debugging path for an uncovered item using Visualizer's debugging capability and see how we can fix the code at the same place.

Starting from the testplan tracker window in figure 19, you may need to add more criteria to determine which section to start with. Adding a priority column, as shown in figure 20, makes it more obvious now that the operand limit sub-section has "zero" coverage percentage with very high priority "1". Let's track the assertion type coverage linked to that section.



Sec#	Testplan Section/Coverage Link	Type	Status	Coverage%	Goal	% of Goal	Weight	Link Status	Bins	Hits	% Hit	Priority
0	Testplan	Testplan		81.16%	-	81.16%	1	Clean	9138	1761	19.27%	
2	Calculator	Testplan		94.02%	100%	94.02%	1	Clean	2031	633	31.16%	1
2.18	Operand limit	Testplan		0.00%	100%	0.00%	1	Clean	1	0	0.00%	
4	Binary_to_BCD	Testplan		70.27%	100%	70.27%	1	Clean	1380	362	26.23%	4
4.3	..._flag_from_BCD	Testplan		0.00%	100%	0.00%	1	Clean	1	0	0.00%	
5	Seven_segment	Testplan		53.65%	100%	53.65%	1	Clean	4331	428	9.88%	5
5.1	...n segment error	Testplan		0.00%	100%	0.00%	1	Clean	1	0	0.00%	

Figure 20. Adding priority column in the testplan tracker window.

Double clicking on the assertion type "greater_ than_99_a" opens the assertion coverage window for more analysis as well as highlighting the assertion in the source window.

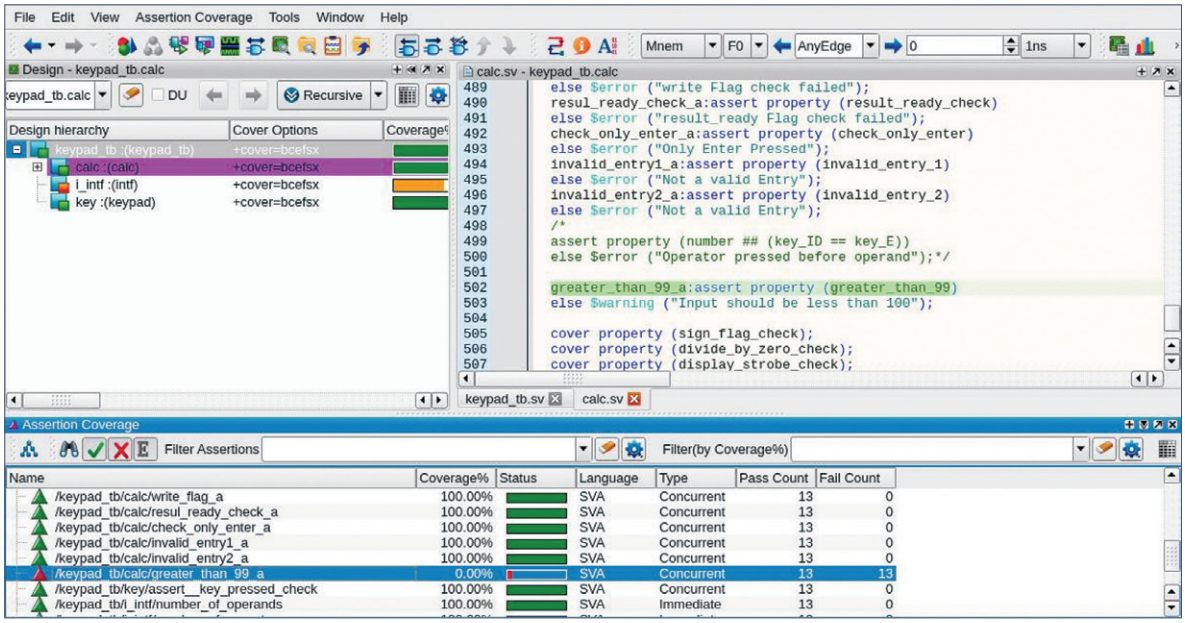


Figure 21. Assertion coverage window.

The assertion coverage window provides more details about the design assertions (status, assertion types, pass/fail count, etc.). It shows all embedded and external assertions that were successfully compiled and simulated during the current session. The “greater_than_99_a” assertion is marked as 0% covered because the assertions failed. For more debugging, you need to examine this property and try to add it to the wave. All of that can be done in the same tool by invoking Visualizer in debugging mode.

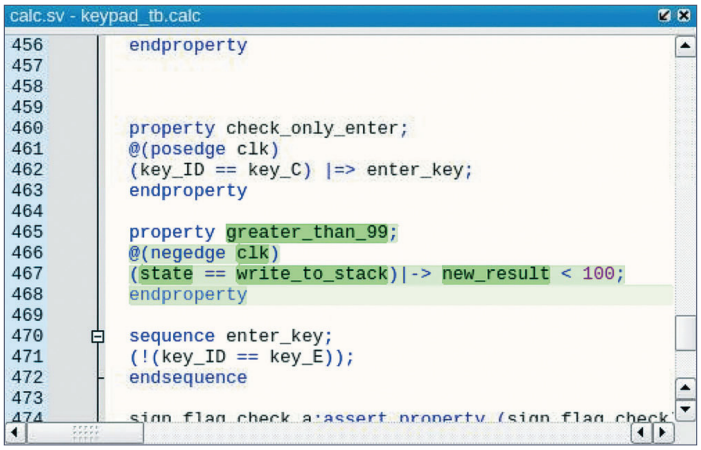


Figure 22. Double clicking source highlighting.

Now by double clicking in Visualizer, you can navigate to the property definition in the source code window as shown in figure 22.

The condition says that at the negative edge of clock, if the state is “write_to_ stack”, the “new_ result” signal should be less than 100. You can examine the values by either enabling value annotation or adding the property to the waveform window as seen in figure 23.

Now you can see that at the negative edge of clock with the state equal to “write_to_stack”, the "new_ result" signal is undefined!

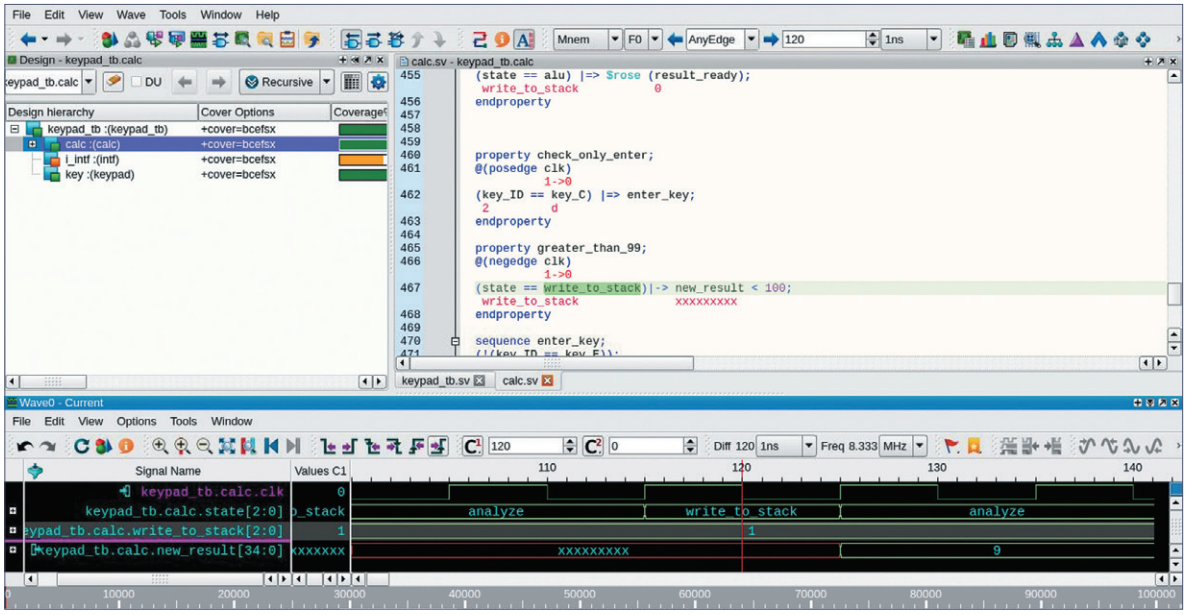


Figure 23. Waveform debugging.

You can fix this in the code by defining an initial value for the “new_result” signal as shown in figure 24.

Rerunning the simulation, going back to the assertion window, and searching for "greater*" in the “filter assertions” box results in getting full coverage for the assertion. By checking the testplan tracker window, we can see the “operand limit” now has full coverage too.

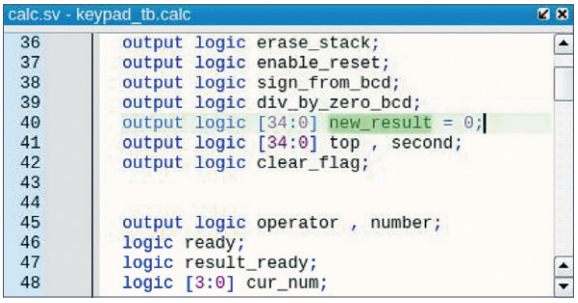


Figure 24. Source code window.

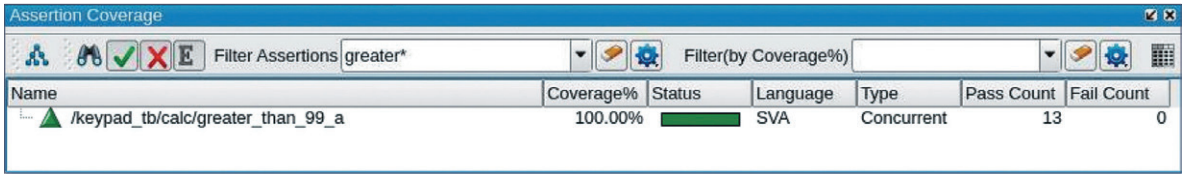


Figure 25. Assertion coverage window with fixed assertion item.

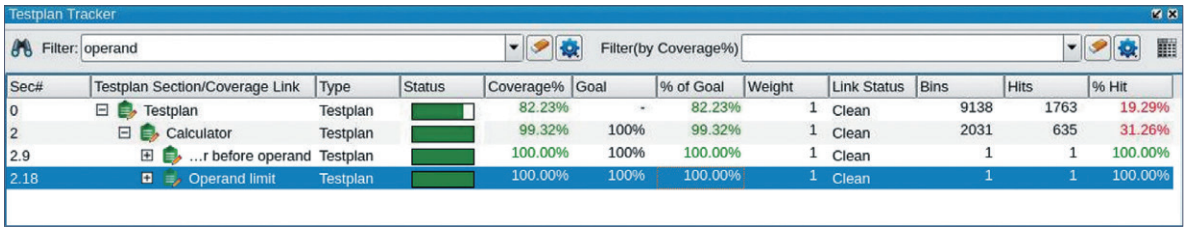


Figure 26. Testplan tracker window after fixing the assertion.

Exclude all!

When you apply an exclusion on a cover item from a coverage window, the exclusion is added to the pending exclusions window. The exclusion is displayed as a textual representation of the coverage exclude command (including arguments) that Visualizer will run to apply the exclusion.

Figure 27. Pending exclusion window.

You can execute the selected commands by clicking the run button in the toolbar, which constitutes a two- step flow for coverage exclusion. You can also copy the exclusion commands written in this window and use them in a do file. All options are listed below in the RMB menu.

Now you can run all the exclusions at once, saving the coverage recalculations that happen after every single execution for the exclusion, and hence improve the GUI performance.

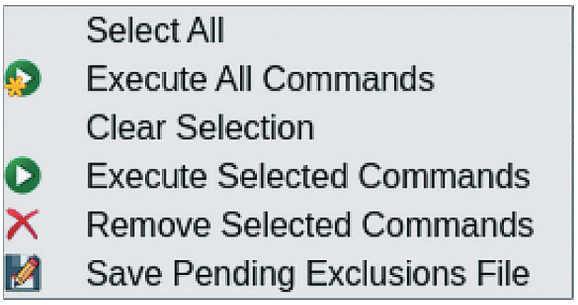


Figure 28. Pending exclusion window RMB menu.

All the executed exclusions are listed in the exclusions list window, which displays the history of items that you have already excluded in this session, and helps you keep a track of any coverage exclusions applied in your design. All excluded items are marked as "E" in the different coverage analysis windows as shown in figure 29.

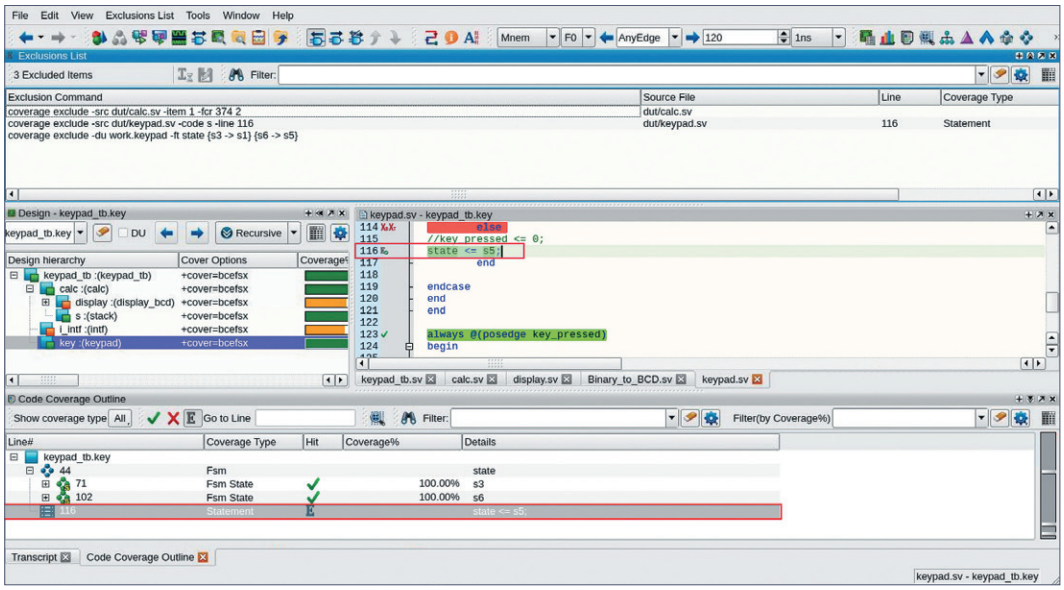


Figure 29. Exclusions list window.

Now you can find that all related coverage analysis windows are updated with the new coverage percentage after running the exclusions as seen in the testplan tracker window and FSM List window shown in figure 30.

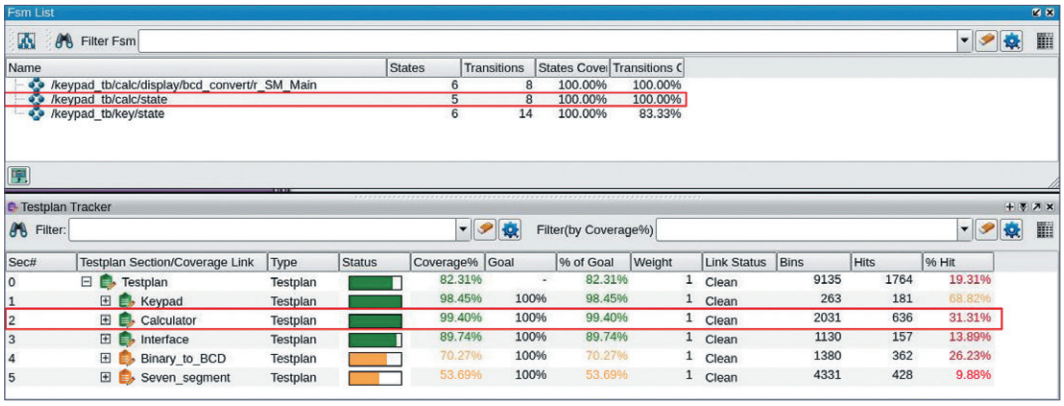


Figure 30. Coverage recalculations after running the exclusions.

Covergroup analysis

Last, but not least, we need to explore the covergroups window, which shows coverage results for SystemVerilog covergroups, coverpoints, crosses and bins in the design. The coverage information is displayed in two panes. The top or left pane lists the covergroups present in the current design, and bottom or right pane (covergroup details) lists bins and associated coverpoints for a selected covergroup with corresponding bins.

As shown in figure 31, double-clicking on "opcode_combinations" covergroup in the top pane of the covergroups window populates the associated coverpoints with the corresponding bins in the bottom pane and highlights the source code window. Then by filtering the coverpoints with "zero" coverage, you can go directly to the missing bins in your selected covergroup.

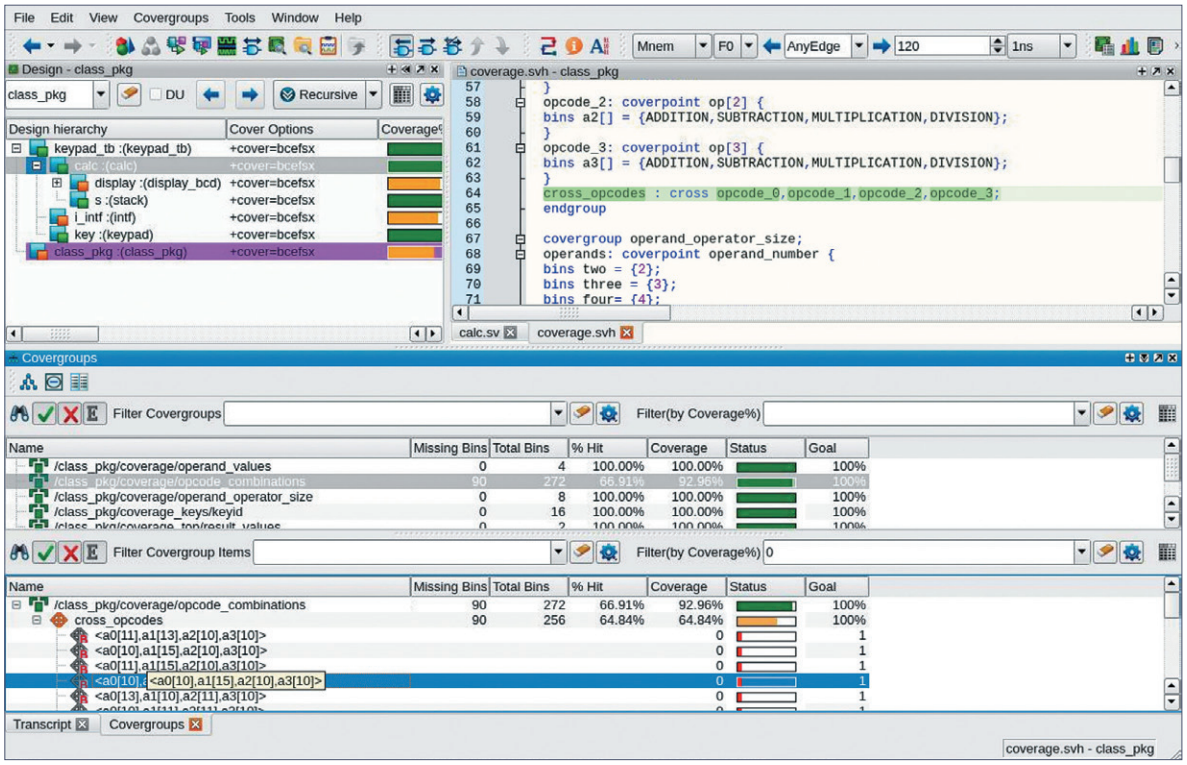


Figure 31. Covergroup window.

Conclusion

The Visualizer Debug Environment helps you fix and debug all the uncovered items in your design at one place. Visualizer provides visualization tools that display coverage results for both code and functional coverage. The graphical user interface shows coverage statistics in instance and design unit views, helps you to see a quick overview of your design coverage in terms of coverage types, testplan sections and design units, and has detailed information on every coverage item via the coverage analysis window to facilitate the deep analysis for that item.

Siemens Digital Industries Software

Americas: 1 800 498 5351

EMEA: 00 800 70002222

Asia-Pacific: 001 800 03061910

For additional numbers, click here.

About Siemens Digital Industries Software

Siemens Digital Industries Software is driving transformation to enable a digital enterprise where engineering, manufacturing and electronics design meet tomorrow. Xcelerator, the comprehensive and integrated portfolio of software and services from Siemens Digital Industries Software, helps companies of all sizes create and leverage a comprehensive digital twin that provides organizations with new insights, opportunities and levels of automation to drive innovation. For more information on Siemens Digital Industries Software products and services, visit siemens.com/software or follow us on LinkedIn, Twitter, Facebook and Instagram. Siemens Digital Industries Software – Where today meets tomorrow.

siemens.com/software

84616-D2 5/22 H