Veritools would like to thank all of the people who came by our booth and made our DAC experience such a success.

Making SystemVerilog Assertion debugging fast and easy

Thank you for visiting our booth at this year's DAC conference in San Francisco. We enjoyed talking with you and appreciated your interest with regard to our tools and especially our new assertion debugging software.

At DAC we presented several new solutions for both the digital and analog designers this year. For digital we presented "Coverage driven verification" using the new VeritoolsVerifyer, and it's built in SystemVerilog Assertions simulator and SVAssertion debugging software. This tools allows designers to run their SystemVerilog Assertion code off-line from their SystemVerilog design simulation using our built in SystemVerilog Assertion simulator. For analog, we presented Veritools Caliper, a software tool so users can run their HSpice measure scripts any number of times without the requirement to re-simulate their analog design.

Below is 3-4 minute demo of this new SVAssertion stand alone simulator with the automatic assertion debugging software that I showed at DAC to attendees many of whom said they were currently very frustrated over the lack of easy to use SVAssertion debugging tools.

VERITOOLSVERIFYER – AUTOMATIC DEBUGGING OF THE USERS SYSTEMVERILOG ASSERTIONS!

Many DAC attendees that came to our booth this year asked what we had to provide coverage driven verification and what software did we have to improve the job of debugging of SVAssertions.

My first question was to each SystemVerilog Assertion designer:
What did they think about debugging SVAssertions today, did they think that SystemVerilog assertions were hard to debug?

Almost to a person their universal reaction was:

"**No SVAssertions are not hard to debug, with the currents tools, they are literally impossible to debug"**.

To show how easy it was to debug complex assertions, with the Veritools software, I demonstrated our SystemVerilog Assertion software to show just how simple it was to debug SVAssertions graphically. The following are the very easy to follow 3 – 4 minute demonstration, to show exactly how automatic SVAssertion debugging works using 5 very simple steps.

STEP 1. Evaluating your SVAssertions:

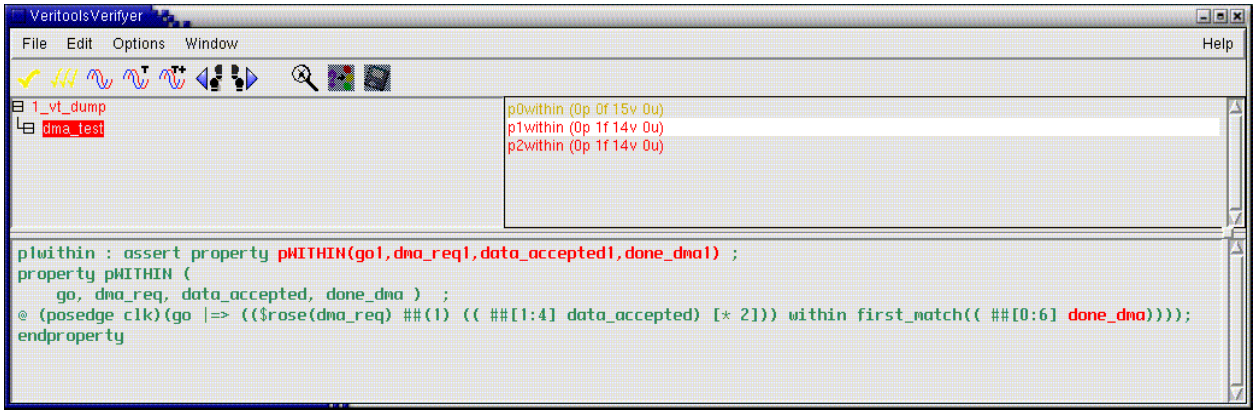
Just load into the VeritoolsVerifyer your Verilog/VHDL/SystemVerilog design, the SVAssertion code and a waveform file, VCD, utF file, etc.

Next, Select any module, in the hierarchy module window, to display all of the assertions in that module, in the assertion selection window just to the immediate right of the hierarchy module window. Modules with any failing assertion are color coded red, if all assertions in a module are passing evaluation, it is color coded green. All modules above any module with a failing assertion are also color coded Red.

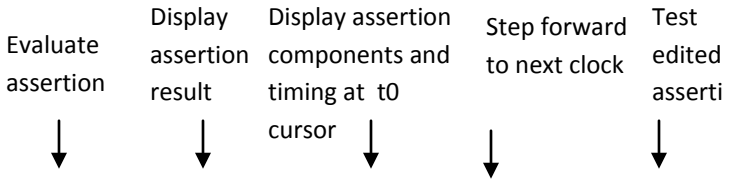
Then, Select any single assertion, in the assertion selection window, to display the SVAssertion code for that assertion, in the window just immediately below the module and assertions selection windows, collectively known as the **hierarchy windows**.

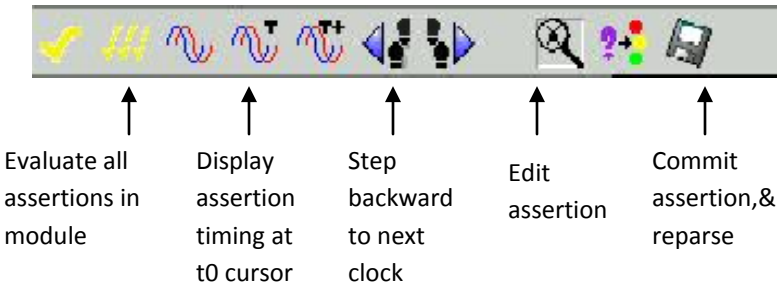
Press "**Evaluate all Assertions**" All assertions in the selected module and below are then rapidly evaluated, with the evaluation results shown in a color coded hierarchy window, the window just below the icons on the left.

That's it! It is just that easy!



SVAssertion toolbar icons.





Notice: that these steps have just automatically debugged your assertions and have even color coded the exact part of any selected assertion that is causing the failure of this assertion. In the example above, "dma_done", color coded Red, indicates that this is the term that is causing this assertion to not pass evaluation.

[NOTE: While many users expressed their frustration that assertions were impossible to debug, this very easy to use software will actually "automatically" debug the assertions for the user.]

The very next step is to find the reason why any assertion is not passing evaluation.

STEP 2. Finding the actual assertion error:

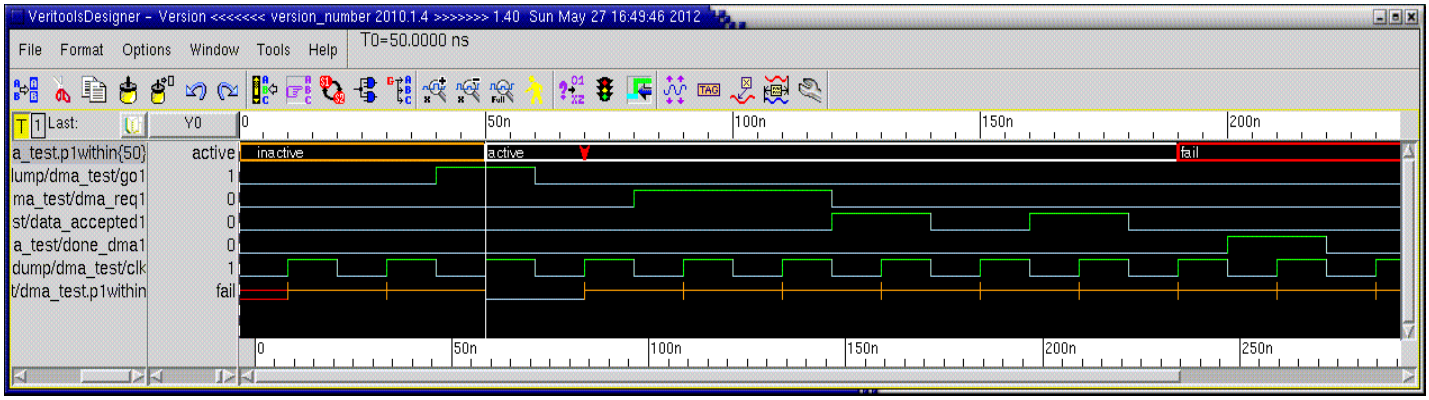
To find out what exactly what issue is actually causing any assertion to fail evaluation do the following steps.

Press Display Assertion result: The **result** waveform for this assertion will be shown on the waveform window, this is the signal located at the very bottom of the waveform display and shown below, labeled ...."dma_testp1within". A low going pulse shows that this assertion is failing evaluation. A high going pulse would show that this assertion was passing evaluation. If the signal is nether low nor high this is considered a vacuously true condition, and is colored Orange, in effect this assertion was not evaluated at this time point because the condition for it to go active was not present at this time point.

Press "**Display assertion components and timing at t0**", to show all of the signals used to evaluate this assertion, along with the timing waveform for this assertion at the t0 cursor. NOTE: The **Assertion timing** signal will show when this assertion was inactive, then when it went active and then the exact time point where this assertion went from active to either a pass or a fail evaluation.

NOTE: When **Display Assertion result** is pressed, the t0 cursor will also automatically be placed on the failing edge of the very first assertion evaluation result waveform pulse that indicates an assertion failure. The assertion timing and the color coding of the assertion code in the assertion code text window is then defined by the simulation time at this t0 cursor time point when the user presses "**Display assertion components and timing at t0"**. Since any assertion can be

evaluated any numbers of times during any simulation, to a passing or not passing evaluation, if at any point the assertion does not pass evaluation, the assertion is considered to have not passed evaluation, even if at all other times it passed.



In this example the term that is failing this evaluation is dma_done, see the code displayed in Step 1. The assertion code requires that; "within first_match ((##[0-6] dma_done))" dma_done must go high. But if you count the clocks, from the start of this assertion, skipping one clock because of the term ##1, dma_done does not go high until clock 7, and the exact time for the failure point is also noted to be clock 6 in the assertion timing waveform. This is when there is no longer any possibility that this assertion can pass evaluation. So to fix this assertion all a user has to do is make the number 6, the number 7 or any number greater than 6.

STEP 3. Edit the parameter that caused this assertion to fail, and rerun assertion simulation

When an assertion is not passing evaluation, and the user has found the cause, the user can then edit their assertion code and instantly re-run assertion simulation for this one single assertion to see if they have now fixed this assertion.

Press Edit Assertion. This allows the user to now modify the assertion in the assertion code text window which has now become a "What if window". The source code text window is the window below the two hierarchy windows, see the figure below.

To fix the assertion in this example make the number of clocks in the within first_match expression, just before the signal dma_done, 7 or any number greater than 7:

ie. "within first_match ((##[0-6] dma_done))" => "within first_match ((##[0-7] dma_done))"

Then re-test the now modified assertion code.

Press Test edited assertion. This re-simulates the modified assertion code. If this assertion now passes evaluation, the **Assertion Result signal pulse** at t0 will now become a high going pulse instead of a low going pulse. The assertion in the assertion selection window with go from being colored Red to being colored Green, and the assertion timing signal will now go from Fail condition to a Pass condition:

This will show that the now modified code passes the assertion evaluation.

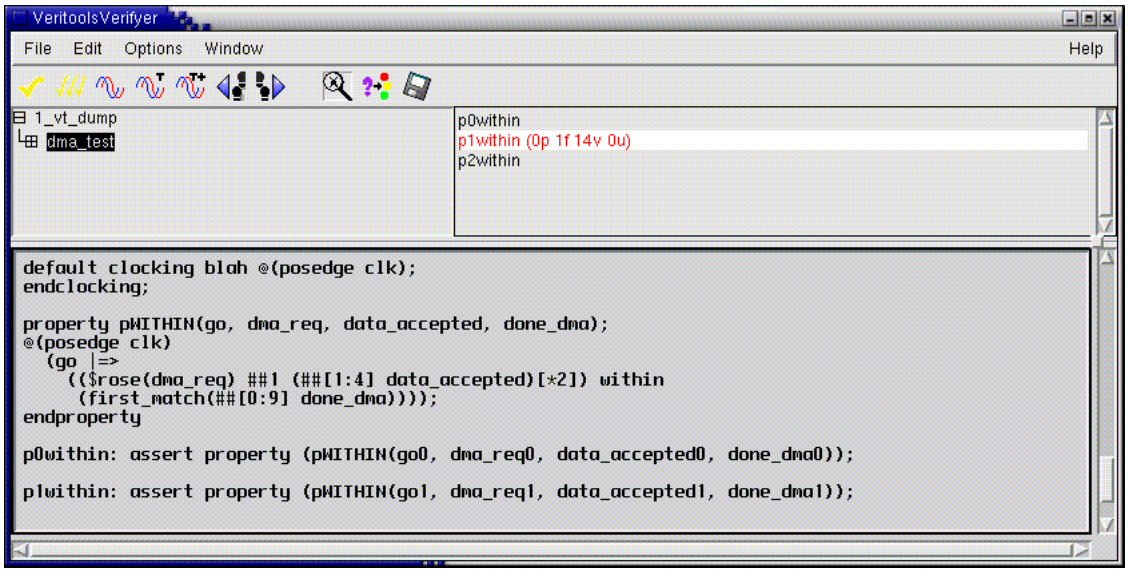
Users can re-edit this assertion and press **Test edited assertion,** an unlimited number of times in order to find the correct parameters in front of dma_done or any parameter to make this or any assertion pass evaluation. Fixing and then re-testing the users failing assertions, is an operation that could have taken many hours or even days using just a simulator approach, to evaluate and fix any failing assertion, can now be done in minutes. The process that was described by verification engineers before as impossible is now quick and easy.

Press Commit assertion & Reparse

When the user is satisfied that their assertion source code now passes evaluation, they can commit the newly modified assertion code, reparse and re-simulate this assertion with the newly committed assertion code.

This will make the term in the assertion code window that had shown as an error and had been colored Red, “dma_done”, turn to Green to indicate that this assertions committed code now passes evaluation and that the parameter for testing dma_done no longer will cause this assertion to fail.

Note: The users source code is never changed in any way until the user finally presses “**Commit assertion & reparse”. Only then is** the old assertion code saved in a file with a time stamp and the new code replaces the assertion code that did not work.

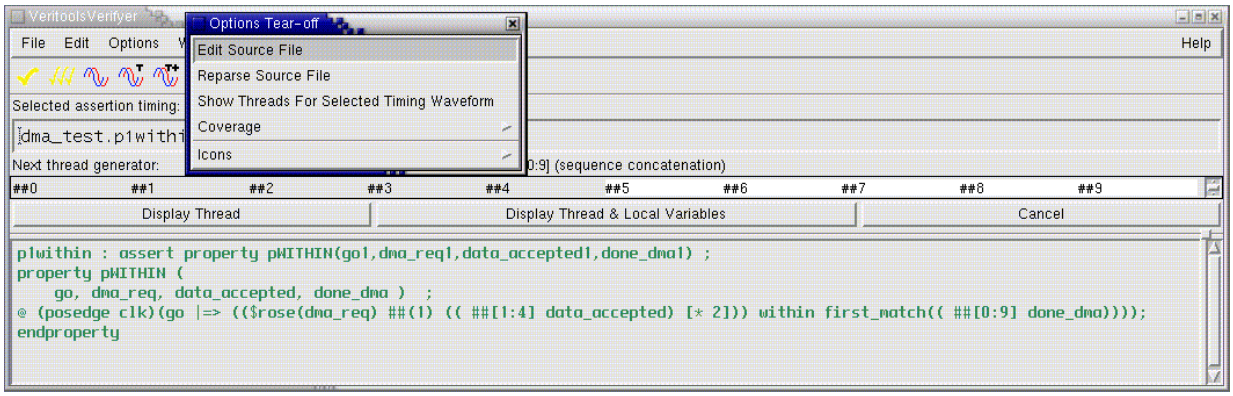


STEP 4. Advanced features - displaying the assertions unique independent execution threads along with the local variables for each unique execution thread.

Many SystemVerilog Assertions not only contain many unique independent execution threads but each thread then may also have and use many local variables that are operated on and/or modified by this tread. In the past, using a design simulator, there has been no way to visually see either these threads or the value of the local variables that were used/or modified by these thread. Users were forced to add print statements and then re-simulate their design, a process that not only took at least several hours but made debugging SVAssertions impossibly slow. The VeritoolsVerifyer gives users not only the ability to visually see all of their unique independent execution threads but also to see the values of all of the local variables that may have been used with each thread and/or modified during assertion simulation. Operations that could have taken a hours or even days previously using a design simulator, can now be done in just minutes.

Displaying the Unique independent execution threads

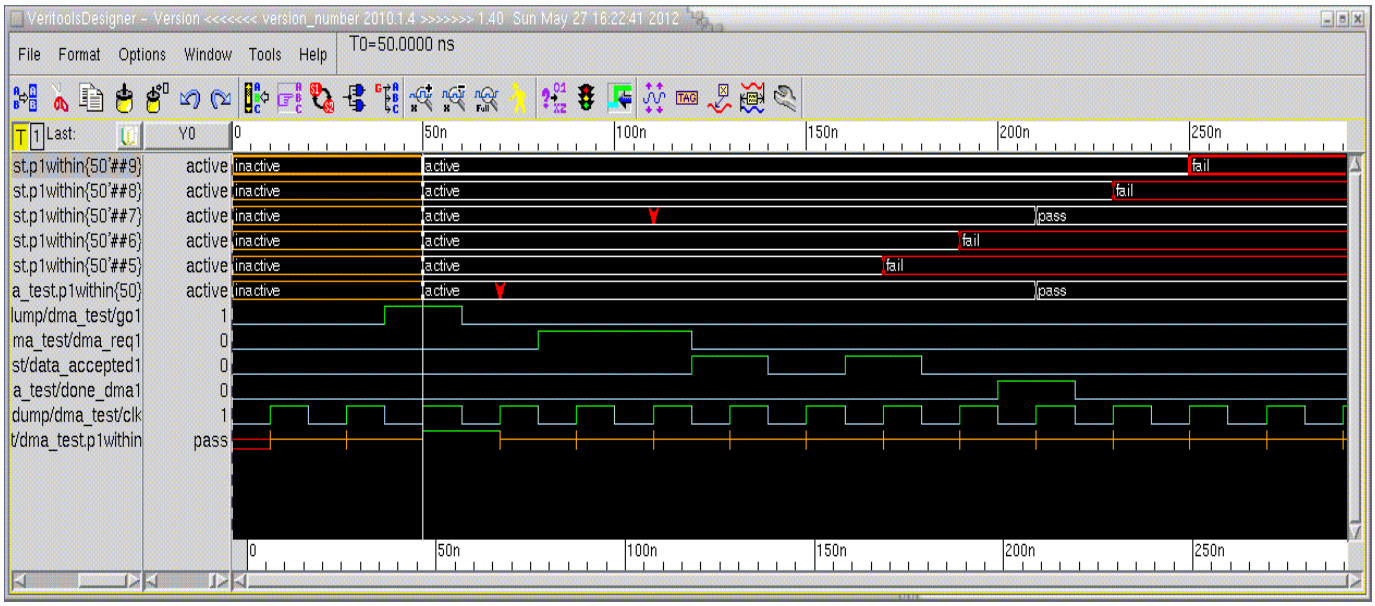
Open the Options menu and press "Show Threads for Selected Timing Waveform"



Note that the users must first select an assertion timing waveform, and then press **"Show Threads for Selected Timing Waveform".** As can be seen in the above figure, the unique independent execution threads are listed in a window just above the source code text window. The user can select any or all of these threads to display.

In this thread display window, the user can then:

Press either **Display Thread, or Display Thread & Local Variables** to display in the waveform window the unique independent execution threads for this assertion defined at the t0 cursor time point along with all of the local variables that were used in the execution of this thread.



As shown in the above waveform window, the unique independent execution threads in thread "within(50)" assertion timing signal, are displayed in the above waveform window and are labeled "within(50##5-8)". The assertion timing signals "within(50 ##, 5, 6, 8 and 9)" all go to a fail condition, the assertion timing signal for "within(50##7)" goes to a pass condition. This indicates that correct number of clocks prior to "dma_done" to allow this assertion pass evaluation is 7 clocks.

Also note that a RED downward pointing arrow in any of the assertion timing waveforms indicates that there are even more unique independent threads starting at this exact time point that were part of this assertion timing signal and these can also be displayed for this assertion timing signal. Threads can be nested down to any number of levels, in typical assertion code.

STEP 5. Display Design Coverage of assertion code

To get coverage driven verification do the following simple operation:

Press **Options => Coverage,** to show coverage for this assertion evaluation.

Each and every assertion evaluated will be listed along with the number of times it passed evaluation, the number of times it failed evaluation and whether it was still pending at the end of the simulation.

Please let Veritools know about your interest regarding an evaluation license of these tools, at no cost to you. Veritools would also be more than happy to arrange a brief 30 minute to 1 hour demo, depending on your questions, at your convenience with an engineer who could also answer any questions you may have.

We look forward to hearing from you.

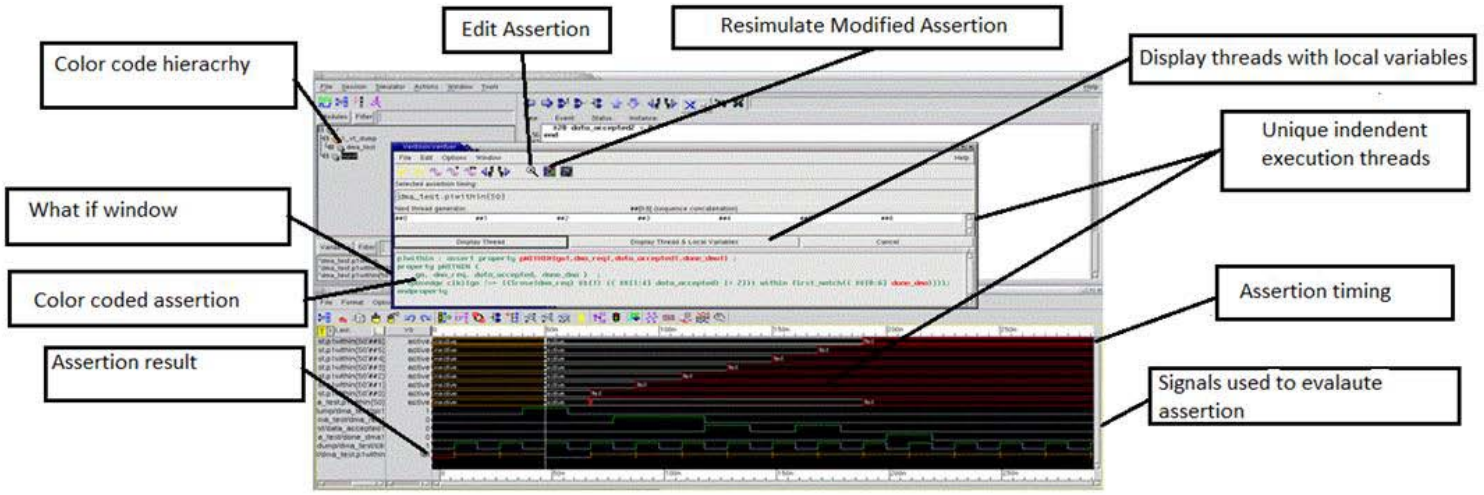
Anne Scott & Bob Schopmeyer
Veritools, Inc.
650 867-7008 & 650 533-5595

Several of the other Veritools software tools demoed at DAC and included in the VeritoolsVerifyer binary are:

The vWave waveform display tool, that is one of the most widely used waveform display tools in the EDA industry

The VeritoolsDesigner, a Verilog/VHDL and SystemVerilog RTL source code debugging suite, has waveforms, source code, schematics (both gate and RTL) and state diagrams synchronized and integrated into one tool.

Costs:
While RTL source code debugging tools today have identical or almost features, the VeritoolsVerifyer, not only includes powerful RTL source code debugging software but includes a standalone SystemVerilog Assertion simulator and SVA Assertion debugging suite, at no additional cost.



The graphical display of assertions makes debugging of even complex SV Assertions fast and easy

The VeritoolsVerifyer software offers the users of SystemVerilog Assertions not only the ability to verify their assertions and but to do real coverage driven design verification?

- **Coverage driven Verification**

For users that want to get first silicon to work, SystemVerilog Assertions and this type of software is critical. Many designs today have over 100,000 assertions and many design managers could not get their ASICs to work without this massive of an assertion effort. But debugging this large a number of assertions using "print" statements, a technique many companies still use, requires an inordinate amount of time and people. Hence many companies have gone to using only extremely simple assertions to eliminate any real debugging effort. But this defeats the very strength of using SystemVerilog Assertions, the ability to put numerous compact and complex assertion together rapidly to thoroughly test their complex designs. VeritoolsVerifyer allows design and verification engineers to make use of large numbers of compact, complex assertions for the first time and allows these assertions to be debugged rapidly with easy to use graphical software.

For analog oriented designers, we presented at DAC the Veritools' Caliper Tool, for doing measures with HSpice measure scripts in batch mode without being forced to re-simulate the analog design.

We have many customers who have HSpice or FineSim simulations that take days to complete. These users were required to rerun their HSpice or FineSim simulations even when they only wanted to see a small number of additional measures. But this is no longer the case with Veritools' Caliper Tool. Users can now just add or modify their measures and then rerun the new measure either interactively or in a batch process without having to re-simulate their design. The Caliper scripts can even be combined with our built-in Perl scripting software for additional more powerful analog measure tools.

Currently the Veritools' vWave software handles the following analog formats:
Synopsys HSpice, and FineSim tr0, ASCII and binary, ac sweep, dc sweep, ft0
HSim .out
Cadence PSF ASCII and binary. WSF ASCII, binary, WSF tr0
Mentor ELDO DOU and COU formats
Texas Instruments analog format
CSDF ac sweep, dc sweep, tr0
Veritools utF (fast) format, VCD
For additional information on this software and for obtaining a no cost license to evaluate this software, please visit our web site at www.veritools.com or send email to Veritools at inquiry@veritools.com, or schop@earthlink.net. The Veritools data sheets are also available: datasheets