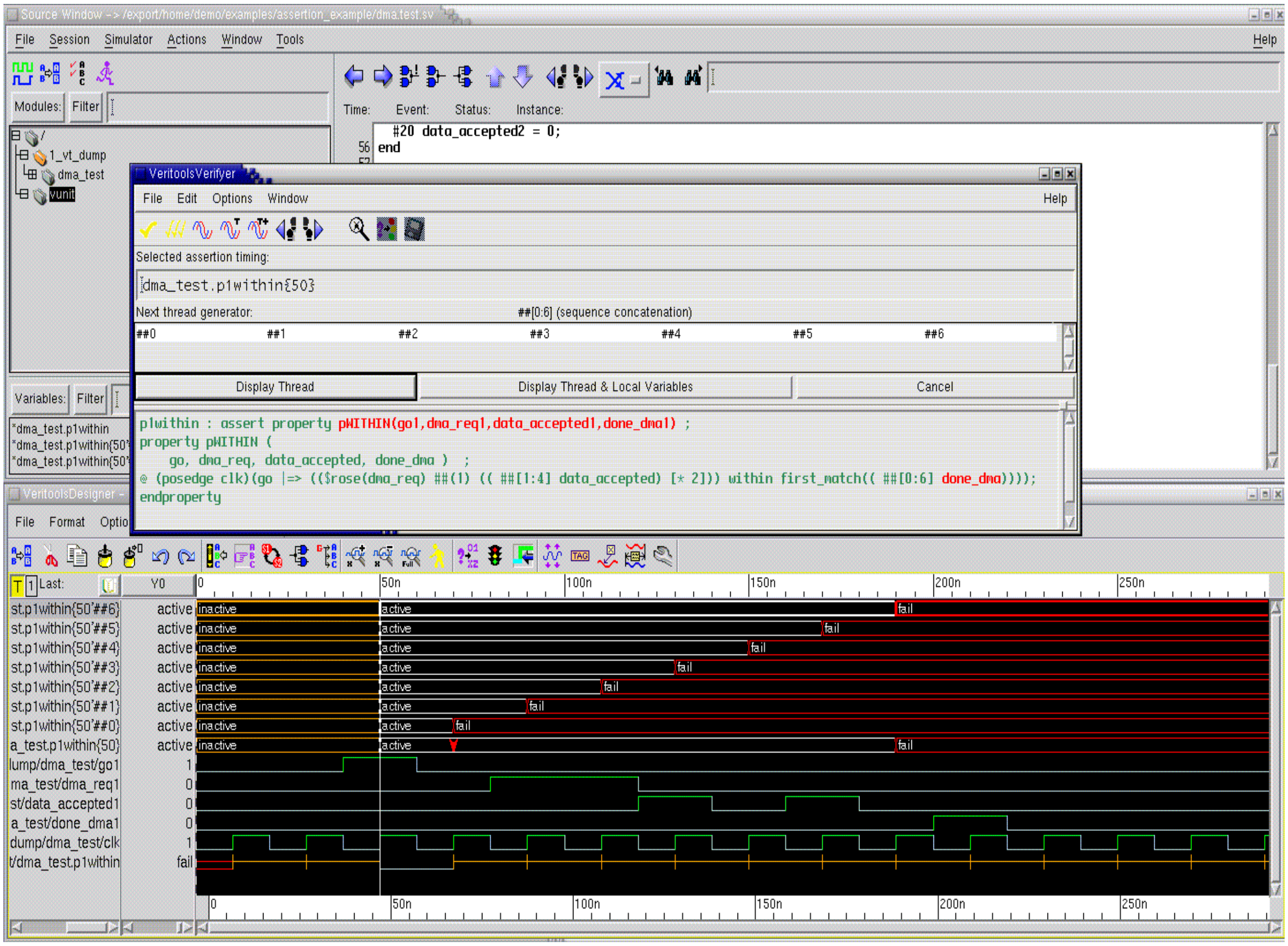
Veritools Data Sheet

VeritoolsVerifyer-Universal

A SystemVerilog Assertion Simulator and Analyzer added to VeritoolsDesigner becomes the ***VeritoolsVerifyer-Universal. Runs native on Windows and on Linux, Centos 5, 6, 7, and Ubuntu***


Source Window -> /export/home/demo/examples/assertion_example/dma.test.sv
File Session Simulator Actions Window Tools Help
Modules: Filter
Time: Event: Status: Instance:
/
1_vt_dump
dma_test
vunit
Variables: Filter
VeritoolsVerifyer
File Edit Options Window Help
VeritoolsDesigner
File Format Optio
Display Thread
Display Thread & Local Variables
Cancel
T 1 Last:
Y0
0
50n
100n
150n
200n
250n
inactive
active
fail

```
   #20 data_accepted2 = 0;
56 end
```

Selected assertion timing: dma_test.p1within{50}

Next thread generator: ##[0:6] (sequence concatenation)

##0 ##1 ##2 ##3 ##4 ##5 ##6

```
p1within : assert property pWITHIN(go1,dma_req1,data_accepted1,done_dma1) ;
property pWITHIN (
    go, dma_req, data_accepted, done_dma )  ;
@ (posedge clk)(go |=> (($rose(dma_req) ##(1) (( ##[1:4] data_accepted) [* 2])) within first_match(( ##[0:6] done_dma))));
endproperty
```

Signal	Y0
st.p1within{50'##6}	active
st.p1within{50'##5}	active
st.p1within{50'##4}	active
st.p1within{50'##3}	active
st.p1within{50'##2}	active
st.p1within{50'##1}	active
st.p1within{50'##0}	active
a_test.p1within{50}	active
lump/dma_test/go1	1
ma_test/dma_req1	0
st/data_accepted1	0
a_test/done_dma1	0
dump/dma_test/clk	1
t/dma_test.p1within	fail

SVA based Functional Verification

- Stand Alone SystemVerilog Assertion simulator and evaluator
- SystemVerilog Assertions and results are displayed in the design hierarchy
- Assertion timing results can be displayed along with signal components for any SVAssertions
- SVA unique execution threads can be displayed along with local variables
- The SVAssertion analyzer can display the part of each assertion that passed and the part that failed evaluation

SVA based Functional Verification Coverage

- A 'What if' window allows users to create, modify and test assertions
- Users can edit and rerun assertion evaluations an unlimited number of times with no re-simulating
- Assertion coverage metrics provide full Functional Verification Coverage
- Functional Verification Coverage provides accurate indication of completeness of verification process
- Test setup and hold times in gate designs using analog simulation data

Veritools

download VeritoolsVerifyer from our web site: www.veritools.com

330 Lunada Dr, Los Altos, CA 94022; phone: (650) 533-5595; email: schop@veritools.com, schop@earthlink.net

Verify your design in a fraction of the time and cost!

In today's ASIC design environment, the percentage of successful ASIC tape outs has been declining over the past several years, due to the fact that ASIC designs have been getting progressively more complex and increasingly more difficult to verify and test.

To increase the success rate of new ASIC designs, companies have started to incorporate SystemVerilog assertions into their design verification process. Many of today's designs have over 30,000 lines of assertion code. The next generation of designs are expected to have over 100,000 lines of assertion code, just to insure the design under test is functionally correct. Design managers and engineers find that it is getting increasingly more difficult to efficiently manage such large amounts of assertion code and to see what the assertion coverage really is. It is even difficult to verify that the assertions are actually testing what design engineers had intended these assertions to test.

The VeritoolsVerifyer, a standa lone SystemVerilog Assertion analyzer allows designers to both manage their assertion code, and to quickly verify that the assertions do what the designer had intended them to do.

Manage SVAssertion code: The user's SVAssertion tests and even the results of evaluations are displayed right in the design hierarchy color coded to show the outcome of the evaluation.

Stand alone SystemVerilog Assertion evaluation: The evaluation of the SVAssertions is done with a result file generated from a Verilog, VHDL or SystemVerilog simulation. The users' SVAssertions can be evaluated either by linking them into the design via the "bind" function or by embedding them into the source code. The user's SVAssertions can be evaluated an unlimited number of times without requiring any re-simulation. SVAssertions evaluation results are displayed in the user's design hierarchy in a color code to indicate the outcome of the evaluation: Yellow indicates the assertion result was only vacuously true, Green, the assertion result was evaluated to a "pass", and Red indicates the assertion evaluation failed at least once.

Graphical Waveform Result Display: By selecting any assertion, users can display the "assertion results" in the waveform window and see exactly where each assertion reached a pass or fail condition. Assertion results can then be selected for any assertion evaluation to display the SVAssertion timing for this SVAssertion evaluation. Users can display on the waveform window, the signals that were used in the evaluation for this assertion and can even locate and display every independent execution thread for any assertion, along with the local variables that apply to this thread to find out why any particular evaluation failed. Normally evaluating SVAssertions along with the design results in a significant slowdown in simulator performance because SVA assertions have one important dimension not in Verilog or VHDL. Each and every SVAssertion can and sometimes will execute each and every single clock cycle, even if it is already being executed. Because this tool uses the same simulation result files that are generated during normal design simulations, there is never any negative impact on simulation run times when adding SVAssertions to the verification process. And because the assertions evaluations have been removed from the simulation process, simulation speeds are dramatically improved, in many cases by a 5-10 X improvement in speed.

Assertion analyzer; This tool set also includes an assertion analyzer, software that will tell the users exactly what part of their assertion is failing and what part has been evaluated to a pass condition. The operator or signal that caused any assertion to fail will be color-coded red.

"What If'' Capability: VeritoolsVerifyer includes a "What if" capability to allow the user to fix the conditions that may have prevented an assertion from going to an assertion pass condition. If any assertion fails, the user can bring up the assertion execution that failed, modify the assertion and re-evaluate this assertion, instantly. This process can be repeated an unlimited number of times to quickly find the correct Assertion code that will cause it to pass. The assertion expression evaluator in the "What If" window can generally re-evaluate any modified assertion in seconds, even when using very large simulation result files.

In a simulator only based approach users have to:

Run simulation and find the assertions that are failing
Load the source code for the failing assertion into an editor
Edit the failing assertion code
Commit the new code and re-compile the source code
Rerun the design and assertion simulation
Reload the simulation results into an assertion analysis tool
Re-verify that the new assertion is passing or failing
Repeat this process if the assertion is still failing

Using the "What if" capability users can:

Run assertion evaluation, find the failing assertions
Press Edit, edit the Assertion code for any assertion that failed
Press "Test Edited Assertion"
New result showing whether assertion is passing and new assertion timing displays on waveform window, in most cases in seconds.
Repeat from "Edit the Assertion", as many times as necessary.

Users can modify, re-evaluate and re-verify that the new SVAssertion is passing or failing in most cases in seconds. Using a simulator-only based approach to verify SystemVerilog Assertion code, each iteration could take as long as several hours. When using the VeritoolsVerifyer, a new simulation run is only required if a change is required to the user's design or test vectors.

Assertions and Gate Design: The exact same Assertions that are run against the RTL design can be also be used to verify the gate design that was synthesized from this RTL code. Veritools provides an entire set of tools to allow users to use analog simulation results to achieve the highest possible accuracy in the final Assertion analysis of FF set-up and hold timing.

SVAssertion Coverage with complete assertion metrics: The VeritoolsVerifyer includes SVAssertion coverage so users and design managers can see the overall coverage of the SVAssertions in their functional verification process. Results are accumulated and tabulated so users can see a percentage of functional coverage for any part of their design and can go to any part of their design where additional assertion tests may be needed. Users can in fact quickly go from their coverage results to the exact waveforms for the failing assertion in order to quickly debug the failing assertion code.

Other tools with comparable features, cost many times the price.

These tools support **Verilog, Verilog 2005, VHDL, SystemC** and **SystemVerilog**, and, in addition, supports standalone SystemVerilog assertion evaluations including a complete suite of tools to do assertion metrics for coverage analysis. Veritools products are available on Linux systems, 32- and 64-bit systems, and on native Windows. All Rights Reserved, Copywrite 1992-2019. Veritools, Inc. Other Trademarks are owned by their respective corporations.