

July 2010

Welcome to the SpringSoft Technology Newsletter. This is a monthly e-mail newsletter distributed to our customers, partners, and friends to provide information on our Novas Verification Enhancement and Laker Custom IC Design technologies. We hope the information in this newsletter will help you to use our products more effectively in your design and verification environments.

Please see the links at the end of this newsletter for subscription information or to provide feedback that will help us improve future editions.

Verification Tips:

[Expand SystemVerilog Implicit Ports in Source Code View](#)

SystemVerilog provides the ability for implicit port connections. Instead of specifying the detailed port connection every time, designers can use “*” to specify port connections as long as the port name and size matches the connecting net or bus name and size.

[<Continue Reading>](#)

[Directly Import Liberty File](#)

Previously if you had a Synopsys Liberty File (.lib file) and wanted to import it into the Verdi™ Automated Debug System as the symbol library, you had to use the *syn2SymDB* utility to pre-compile the file into Novas symbol library first and then it could be imported into the Verdi system. Starting from the Novas 2010.07 release version, the ability to directly import liberty files is provided. With this capability you can directly import the Synopsys Liberty File from the GUI or from the command line when invoking the Verdi system.

[<Continue Reading>](#)

[Certitude™ Methodology Primer: Early Application Provides Significant Value](#)

The Certitude™ Functional Qualification System identifies holes and weaknesses in the verification environment that can let RTL bugs slip through the process undetected. Although you can apply Certitude at many different points throughout the verification process, experience has shown that an incremental approach beginning in the early stages is often best – providing immediate value and improving the verification environment over time while balancing resources vs. other verification activities.

[<Continue Reading>](#)

Custom IC Design Tips:

[Using PyCells in the Laker Matching Device Creator and Stick Diagram Window](#)

Beginning in Q3, 2010, the OpenAccess version of Laker Custom Layout System will enable the use of interoperable PyCells™ with the popular Laker transistor-level floor planning tools, Matching Device Creator and Stick Diagram window. Previously this unique technology was only available for use with our patented Laker MCells™. With PyCell “stretch handle” and auto-abut capabilities enabled, PyCell placement results can be

optimized in the same way as when using MCells; you no longer to have to select and place each PyCell manually.

<Continue Reading>

Technical Articles:

Getting Productive! – Components in Electronics; June 2010

Power has emerged as a major concern in IC design. Emerging power format standards allow one power definition to be used starting from the early stage of design and throughout the complete design, verification and implementation cycle. However these standards also increase complexity in verification and debug.

<Continue Reading>

Technical Papers:

PCell Caching in OpenAccess

In computer programs, caching is used to store the output from commonly used functions on the disk so that, when executing a repeated instruction, the results may be obtained more quickly without having to reprocess the request. This same mechanism can be used to speed up the display of parameterized cells (PCells) in custom IC design. Some Electronic Design Automation (EDA) tools cache PCells automatically for performance reasons; some require additional licenses; and others offer no caching at all. In addition to the performance benefits, PCell caching can be used to make tool-specific PCells visible in other tools in the design flow.

<Continue Reading>

Current Product Versions:

Certitude 2010.07

Verdi/Siloti 2010.07

Laker 2010.05 (Open Access version)

ADP 2010.02 (Open Access version)

Laker 2010.03 (Laker db version)

ADP 2010.03 (Laker db version)

Need to update your version? Go to www.springsoft.com/support

Events:

SpringSoft Community Conferences (SCC's)

SCC's worldwide will be beginning this September.
Click here to see this year's schedule and to register to attend.

SpringSoft Community News!

In the past few months we have added some new features to our website, have you seen them yet?

Video Hints & Tips – Watch quick tips on how to use SpringSoft products more effectively
EDA Blog – Read about and comment on SpringSoft Technology

Find SpringSoft on Twitter and on Facebook.

Twitter: www.twitter.com/SpringSoft

Facebook: www.facebook.com/SpringSoft

Newsletter Subscription Information:

If you would like to be removed from the SpringSoft Newsletter distribution list, or if you consider this message as unsolicited commercial e-mail, please email unsubscribe@springsoft.com . Type the word "Remove" in the subject line and hit "Send."

We'd appreciate hearing your suggestions, comments or questions about the SpringSoft Newsletter. Please feel free to contact Karim Azar at +1 (408) 467.7860 or karim_azar@springsoft.com

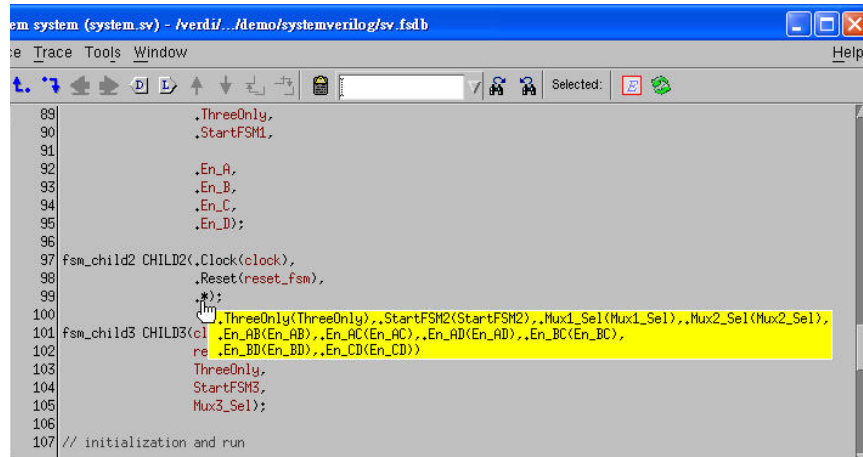
Novas, Laker, Verdi, Siloti and nWave are trademarks and Debussy is a registered trademark of SpringSoft, Inc. All other trademarks are property of their respective owners.

Copyright 2010, SpringSoft, Inc. All rights reserved.

Expand SystemVerilog Implicit Ports in Source Code View

SystemVerilog provides the ability for implicit port connections. Instead of specifying the detailed port connection every time, designers can use “.*” to specify port connections as long as the port name and size matches the connecting net or bus name and size.

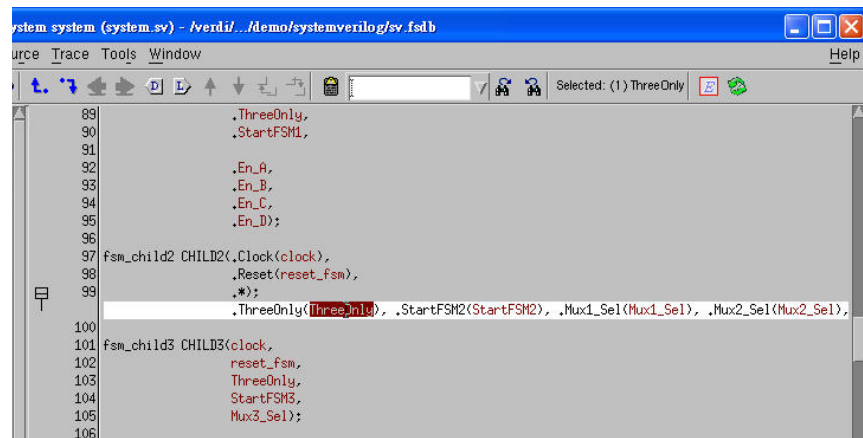
The Verdi™ Automated Debug system also provides string support for debugging the SystemVerilog implicit port. If there are implicit ports defines in the source code, a tip window will appear to show the detailed connections when you move the mouse cursor over the “.*”. The following figure shows this capability:



```
em_system (system.sv) - /verdi/.../demo/systemverilog/sv.fsd.b
File Trace Tools Window Help
Selected:
89         .ThreeOnly,
90         .StartFSM1,
91
92         .En_A,
93         .En_B,
94         .En_C,
95         .En_D);
96
97 fsm_child2 CHILD2(.Clock(clock),
98                 .Reset(reset_fsm),
99                 .*);
100
101 fsm_child3 CHILD3(.cl .ThreeOnly(ThreeOnly), .StartFSM2(StartFSM2), .Mux1_Sel(Mux1_Sel), .Mux2_Sel(Mux2_Sel),
102                 .En_AB(En_AB), .En_AC(En_AC), .En_AD(En_AD), .En_BC(En_BC),
103                 .re .En_BD(En_BD), .En_CD(En_CD))
104                 ThreeOnly,
105                 StartFSM3,
106                 Mux3_Sel);
107 // initialization and run
```

The tip window is just for you to view the connection. If any debugging action is required, the Verdi system provides another command to “expand” the implicit port – just as if the connection has been specified in detail in the source code, so that users can execute any tracing commands on the connection.

In *nTrace*, turn on the **Source → Expand Implicit Port** command to expand the SystemVerilog implicit port statements, the detailed connection will be annotated under the source code, all tracing commands and active annotation are available in the expanded statement. Referring to the following figure, a plus or minus icon will also be created in the line number field, to let you collapse or expand this expanded statement.

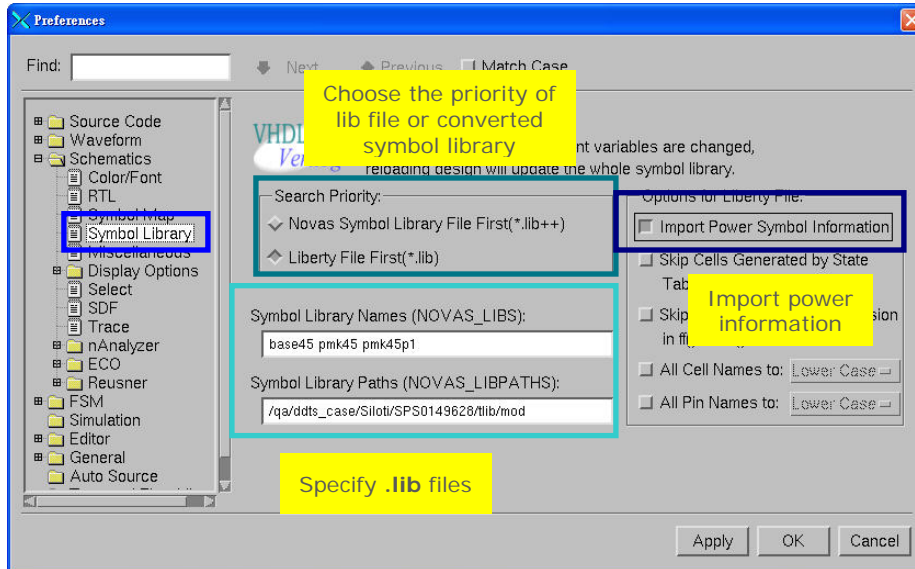


```
ystem_system (system.sv) - /verdi/.../demo/systemverilog/sv.fsd.b
File Trace Tools Window Help
Selected: (1) ThreeOnly
89         .ThreeOnly,
90         .StartFSM1,
91
92         .En_A,
93         .En_B,
94         .En_C,
95         .En_D);
96
97 fsm_child2 CHILD2(.Clock(clock),
98                 .Reset(reset_fsm),
99                 .*);
100
101 fsm_child3 CHILD3(.cl .ThreeOnly(ThreeOnly), .StartFSM2(StartFSM2), .Mux1_Sel(Mux1_Sel), .Mux2_Sel(Mux2_Sel),
102                 .En_AB(En_AB), .En_AC(En_AC), .En_AD(En_AD), .En_BC(En_BC),
103                 .re .En_BD(En_BD), .En_CD(En_CD))
104                 ThreeOnly,
105                 StartFSM3,
106                 Mux3_Sel);
107
```

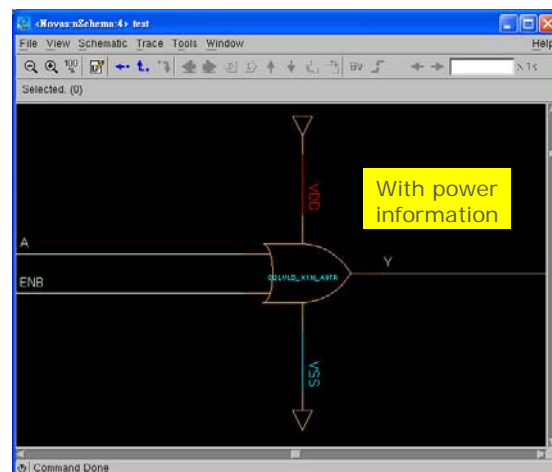
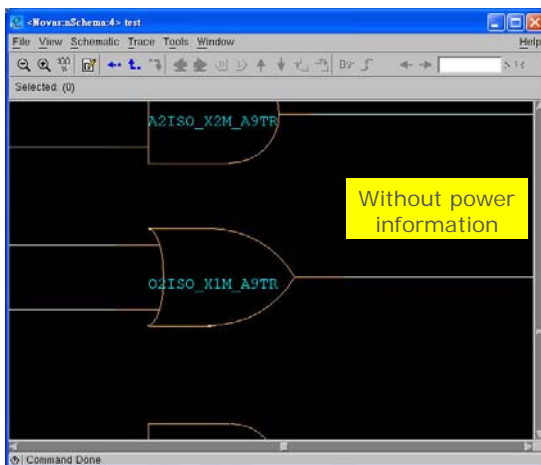
Directly Import Liberty File

Previously if you had a Synopsys Liberty File (.lib file) and wanted to import it into the Verdi™ Automated Debug System as the symbol library, you had to use the *syn2SymDB* utility to pre-compile the file into Novas symbol library first and then it could be imported into the Verdi system. Starting from the Novas 2010.07 release version, the ability to directly import liberty files is provided. With this capability you can directly import the Synopsys Liberty File from the GUI or from the command line when invoking the Verdi system.

There are two methods for importing the liberty file directly: from the GUI mode or from the command line. To import the file in GUI mode, invoke the **Tools → Preferences** command to open the *Preferences* form. Go to the **Schematics → Symbol Library** page, and specify the library file name and path in the **Symbol Library Name** and **Symbol Library Paths** fields, multiple names and paths are allowed in these fields. Power related information can also be imported by turning on the **Import Power Symbol Information** option in this form. See the figure below for an example.



To import the liberty file from the command line, set the name and path of the liberty file through the *NOVAS_LIBS* and *NOVAS_LIBPATHS* environment variables. An additional option, **-powerSymbol**, will be required to import the power information when invoking the Verdi system. The following figures show the difference between importing with and without power information.



Certitude™ Methodology Primer: Early Application Provides Significant Value

The Certitude™ Functional Qualification System identifies holes and weaknesses in the verification environment that can let RTL bugs slip through the process undetected. Although you can apply Certitude at many different points throughout the verification process, experience has shown that an incremental approach beginning in the early stages is often best – providing immediate value and improving the verification environment over time while balancing resources vs. other verification activities.

When Should I Start Running Functional Qualification?

Functional Qualification requires a reasonably complete and functional RTL description and a set of simulation tests that exercise the RTL to verify that it conforms to the functional specification of the design. However, this should not be interpreted as a requirement that “all” simulation tests must be available before functional qualification can begin. In fact, users can gain valuable insight into the verification environment when only a small number of tests are available. Consider, for example, the set of output stuck-at faults injected by Certitude in the early stages of qualification using its default class-based fault prioritization algorithm. A verification environment with a small set of functional tests and the most basic checkers and assertions in place should probably detect these types of gross errors, even if tests that exercise the more complex and subtle aspects of the design are yet to be written. An environment that doesn’t detect these faults is certainly more likely to let serious RTL bugs slip through the process unnoticed.

So, I’ve Qualified the Basic Aspects of My Environment – What’s Next?

As more tests are written, qualification can proceed incrementally. For example, the completion and use of a “smoke suite” of tests – a set that exercises the major functional aspects of the design and is used as a first-level sign-off before significant RTL changes are checked into the design database – argues for expanded use of functional qualification to provide a more comprehensive measure of verification environment quality and identify holes and weaknesses that themselves invalidate the purpose of the “smoke suite”. Again, Certitude’s automated fault prioritization process can be applied to guide the qualification at this stage with an expanded set of faults that stress-test the maturing verification environment.

Good News: Incremental Qualification Accumulates Results Over Time

Functional qualification with Certitude is a cumulative process. As you improve your verification environment, Certitude ensures that previously undetected faults are detected and then delves deeper into its prioritized fault list to qualify the more subtle aspects of your environment. As a user, you get the benefits of early feedback on your environment – allowing you to find and fix problems to ensure that your environment is robust – and accumulation of qualification results over time to provide a complete assessment of the quality of your verification environment.

Using PyCells in the Laker Matching Device Creator and Stick Diagram Window

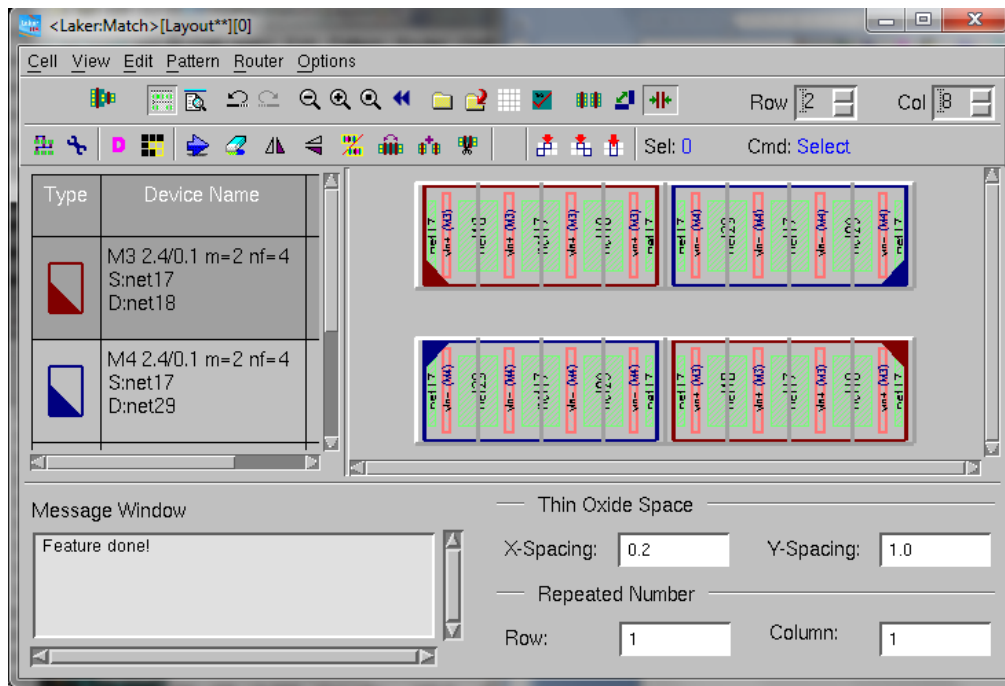
Beginning in Q3, 2010, the OpenAccess version of Laker Custom Layout System will enable the use of interoperable PyCells™ with the popular Laker transistor-level floor planning tools, Matching Device Creator and Stick Diagram window. Previously this unique technology was only available for use with our patented Laker MCells™. With PyCell “stretch handle” and auto-abut capabilities enabled, PyCell placement results can be optimized in the same way as when using MCells; you no longer to have to select and place each PyCell manually.

Following is an example of using PyCells in the Matching Device Creator with a user-defined matching pattern:

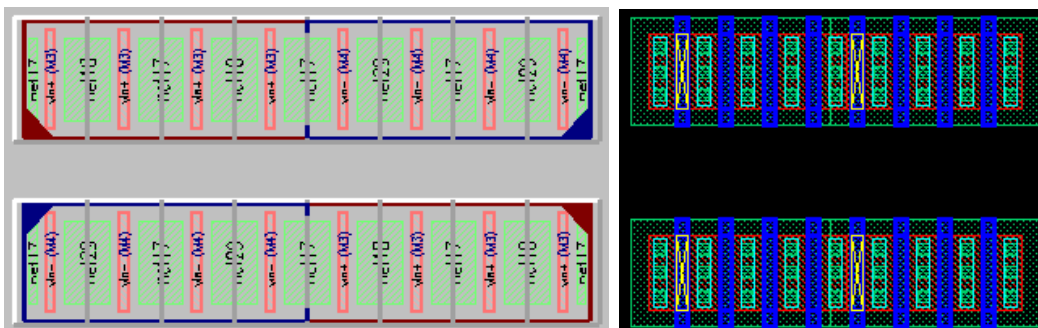
1. In the *Design Browser* window of Laker, select the matching transistors and click the **Match Device Creator** icon.



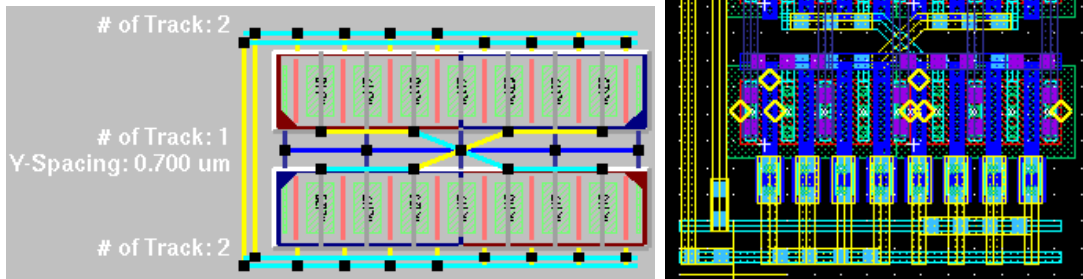
A window like the one below will appear.



2. Modify the floorplan as desired; you can enable OD sharing if your PyCells support auto-abutment.

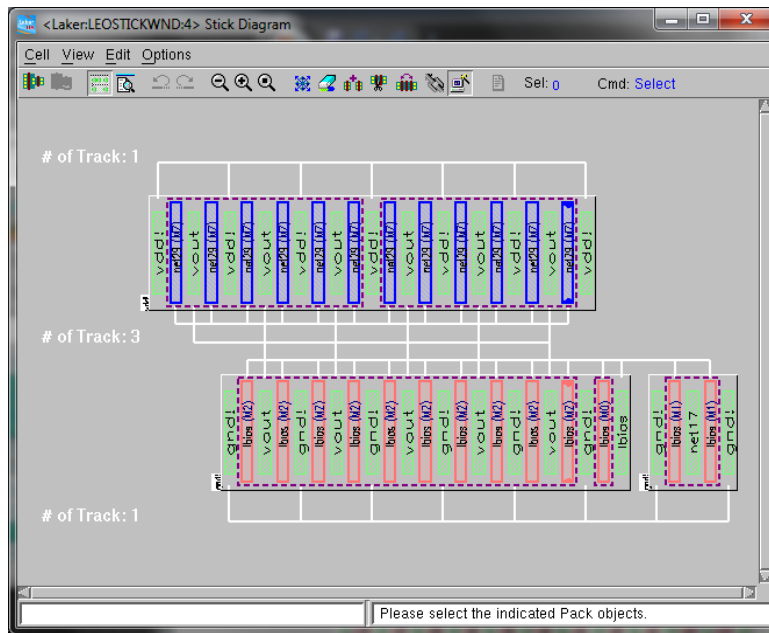


3. The Matching Device Router also works with PyCell devices in the Matching Device Creator



To use PyCells in the Stick Diagram window, proceed as you would if you were using MCells. The following example uses an existing cell template:

1. In the *Design Browser* window, select transistors and click the **Create** icon to bring up the *Stick Diagram* window.



The features: *Merge* (for abutable PyCells), *swap*, *split* and *align* work well when using either MCell or PyCell devices.

2. With PyCell stretch handles enabled, designers can align the transistor gates to minimize wiring jogs.

