



SpringSoft

TECHNOLOGY NEWSLETTER

December 2009

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.

Happy Holidays from SpringSoft!

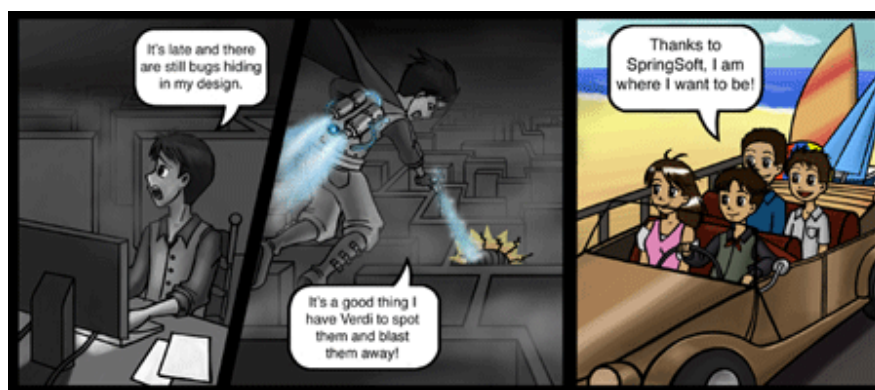


Engineering Superhero Corner:

[SpringSoft Superhero Contest Winner Announced](#)

The SpringSoft community voted Vagner Pires, a design engineer at a large semiconductor company, as having the most creative story about how he overcame design and verification challenges to become an engineering superhero and make a difference in his personal life.

Vagner based the short story on his past experiences having to work nights and weekends to perform tedious debugging tasks, sometimes missing out on quality time with his family. Once he adopted the Verdi System, he was able to quickly find the cause of design errors and correct them, dramatically improving his overall verification cycle and keeping his projects on schedule.



[<Click to read more>](#)

2009 'Best of' Issue!

Verification Tips:

Debug Macros in Verdi – From February

Macros can be defined within or outside of the current scope and can be called multiple times within the design. During debug, engineers may want to see the definition of the macros and signal values within the macro; however, in the conventional approach when engineers encounter a macro, they can only search the macro name in the source code to find the definition and jump to the definition to view the content. The pain of this approach is that it is easy for engineers to forget where the result jumped from when searching the macro content.

<Click to read more>

Visualizing, Analyzing and Debugging SystemVerilog Testbench Environments – From June

Regardless of the underlying library, the most interesting and useful data for the engineer are the transactions between the sequencer and the driver and between the monitor and analyzers. Ideally, this traffic needs to be recorded into a format that is useful for post-simulation analysis and debug. Because a transaction is a much more convenient high-level encapsulation of data for OVM and VMM-based environments, transaction-level debug visualization and analysis is clearly desirable.

<Click to learn more>

Verdi and Siloti - A Unified and Refined Use Model – From July

In addition to the upcoming Verdi 2009.07 and Siloti 2009.07 releases (scheduled to be released on the 20th of July), a brand new release named "Novas 2009.07" is to be released on the 13th of July. In the new Novas 2009.07 release, the Verdi™ Automated Debug System and the Siloti™ Visibility Automation System are being unified from a use-model and packaging point of view.

<Click to learn more>

Power Aware Debug – From October

The power architecture in today's designs is getting more and more complex, and multiple power domains with many power modes require a thorough verification method. Some technologies are provided to define and verify the complex power domains. Among these technologies, the standard power formats, Common Power Format (CPF)/Unified Power Format (UPF), and power-aware simulation are widely used to ensure that low-power designs still produce a high confidence chip.

<Click to read more>

IC Design Tips:

Data Management – Auto Check-in and Check-out – From July

Data Management (DM), revision or version control, is used to manage the data and information related to a project. Generally, the project is ongoing and worked on by multiple members of a design team. Only one member of the design team may have write access to modify data files at any given time.

<Click to read more>

Forward & Backward Compatibility – From November

Software is considered **backward** (or downward) **compatible** if newer versions or releases can read, view, and operate on data that was created with prior versions or releases. This should not be confused with **forward**

compatible which means that older software can read, view, and operate on data that has been created with newer releases. Most design automation software is backward compatible.

<Click to read more>

Technical Articles:

Addressing the Power-Aware Debug Challenge – ECNAAsiamag.com, Nov, 2009

When it comes to design verification, one fact is undeniable: as the biggest contributor to design cycle time, it remains a key pain point in the development of today's sophisticated SoCs. As if that weren't a difficult enough challenge to contend with, today's engineers must also now confront concerns over power and its impact on the overall design, verification and implementation process. One reason for the concern is the growing number of applications with their ever increasing functionality that demands low-power operation to support a longer battery life. Increasing power density, driven by higher clock speeds and shrinking process geometries, is another reason for concern over power. Just as critical, is the fact that today's SoCs are often composed of multiple blocks running multiple applications with varying power requirements.

<Continue Reading>

Support:

All Worldwide Support Consolidated To One Site

No matter where you are worldwide, you can now find SpringSoft's top rated support in one place. Just go to <http://support.springsoft.com>

Click here to go to Support

Product Releases:

A new Novas 2009.10 version, which unifies the Verdi™ Automated Debug System and the Siloti™ Visibility Automation System from a use-model and packaging point of view, was released in October as the default product installation package for Springsoft Novas Debug Solutions, including Verdi and Siloti.

These refinements make it easier and more natural for both Siloti and Verdi users. Users will now be able to access Siloti features through the familiar Verdi interface rather than having to go outside the Verdi flow to prepare and set up for Siloti's unique data expansion capabilities. These improvements will have no impact on licensing.

Novas (Verdi/Siloti): Current release - 2009.10

Laker: Current release - 2009.9 (v5)

Certitude: Current release - 2.7

News:

SpringSoft & Magma Validate Full Interoperability of Custom Chip Design Tools with TSMC 65nm iPDK – Dec. 1, 2009

SAN JOSE, Calif., December 1, 2009 — SpringSoft, Inc. (TAIEX: 2473), a global supplier of specialized IC design software, and Magma® Design Automation Inc. (Nasdaq:LAVA), a provider of chip design software, have completed cross-tool validation using TSMC's 65nm interoperable process design kit (iPDK). The companies demonstrated full interoperability between the SpringSoft Laker™ Custom Layout Automation System and

Magma Titan™ Mixed-signal Design Platform running in the OpenAccess™ environment. This validation saves time and effort of setting up an interoperable flow and ensures consistent results.

<Click to read more>

Upcoming Events:

EDS Fair 2010 – Jan 28-29, 2010

See Springsoft at EDS Fair 2010 in Kanagawa, Japan

Go to: <http://edsfair.com/e> to learn more.

Find SpringSoft on Twitter and on Facebook.

Twitter: www.twitter.com/SpringSoft

Facebook: Search SpringSoft in Facebook and become a fan!

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 2009, SpringSoft, Inc. All rights reserved.

Debug Macros in Verdi

Macros can be defined within or outside of the current scope and can be called multiple times within the design. During debug, engineers may want to see the definition of the macros and signal values within the macro; however, in the conventional approach when engineers encounter a macro, they can only search the macro name in the source code to find the definition and jump to the definition to view the content. The pain of this approach is that it is easy for engineers to forget where the result jumped from when searching the macro content. And unlike normal source code, there is no value annotated in the macro content, so that the cause and effect is hard to understand. Furthermore, what if the macro is instantiated multiple times? It will make macro debugging even harder.

To resolve the above problems, the Verdi™ Automated Debug System provides “Macro Debugging” capability to enable users to quickly expand the macro content and annotate the signal value at the selected time – just like it is the normal source code. Turn on the **Source | Expand Macro** command in the *nTrace* window to enable the capability. Once the option is on, a **+** icon appears to the left side of the macro. Clicking on the **+** icon will expand the macro content and clicking on the **-** icon will collapse it. All tracing commands are available in the expanded macro content. For example, you can double-click on the signal to trace its active driver – just like you do when debugging normal source code. Furthermore, if the FSDB file is loaded and you have turned on the **Source | Active Annotation** option in the *nTrace* window, the simulation dumped values will be annotated under the signal inside macro content. The following figure demonstrates a simple example of how to debug a macro in the Verdi system.

The image shows a screenshot of the Verdi IDE interface. On the left, a list of macros is visible: 78 *MACRO1(CLOCK2), 79 *MACRO2(CLOCK4), and 80 *MACRO3(CLOCK2). The main window displays the expanded code for *MACRO1(CLOCK2). The code includes an always block for a clock edge and a begin block with an if statement for !RESET and an else if statement for !R_Wmode. The else if block contains an assign statement for TDB = #1 ExtData, 55. A yellow callout box points to the +/- icon on the left of the macro name, stating "Click on the +/- icon to expand/collapse the Macro." Another yellow callout box points to the !R_Wmode signal in the code, stating "Double-click to trace active driver." A third yellow callout box points to the TDB = #1 ExtData, 55 assignment, stating "Dumped values will be annotated." On the right, a separate window shows the expanded code for *MACRO1(CLOCK2), with line numbers 135 to 143. The code includes assign statements for TR_load, IDR_load, ResultBusMode, R_Wmode, and VMAMode. A yellow callout box points to the R_Wmode signal in the code, stating "Double-click to trace active driver."

Visualizing, Analyzing and Debugging SystemVerilog Testbench Environments

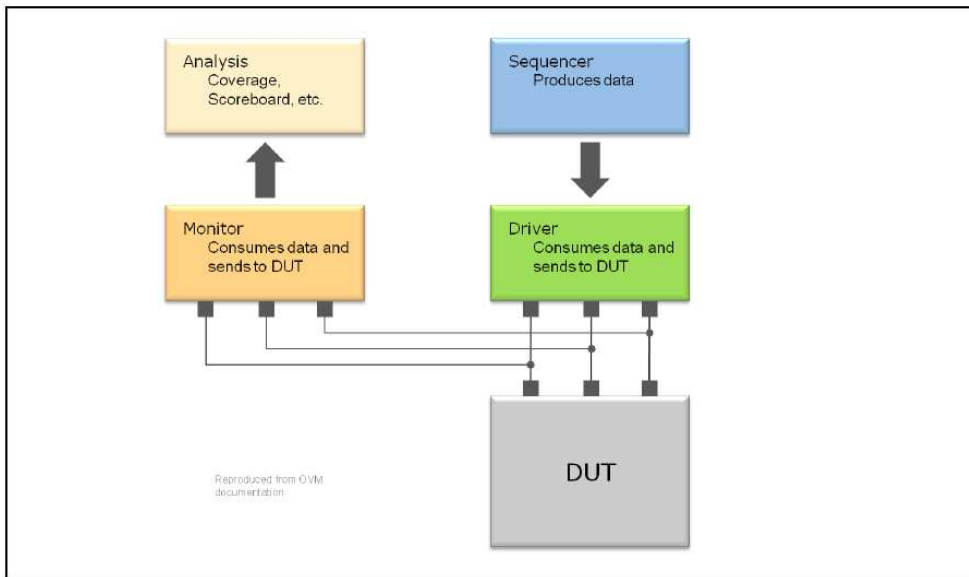


Figure 1: Basic components of an OVM testbench.

[Figure 1](#) depicts the basic components of an OVM-based transaction-level verification environment. VMM-based environments are similar in nature.

Regardless of the underlying library, the most interesting and useful data for the engineer are the transactions between the sequencer and the driver and between the monitor and analyzers. Ideally, this traffic needs to be recorded into a format that is useful for post-simulation analysis and debug. Because a transaction is a much more convenient high-level encapsulation of data for OVM and VMM-based environments, transaction-level debug visualization and analysis is clearly desirable.

SpringSoft's Solutions

SpringSoft's approach to testbench debug builds on existing logging and interactive mechanisms to provide the engineer insight into what is going on in the testbench during simulation (see Figure 2). It makes the logging process much more sophisticated and automated so that most of the debug and analysis of testbench activity can be done at that level. The approach, advanced logging, can be used to directly identify a problem or, in cases where a problem is identified as being on the testbench side with more detail required, it can drive the interactive inspection.

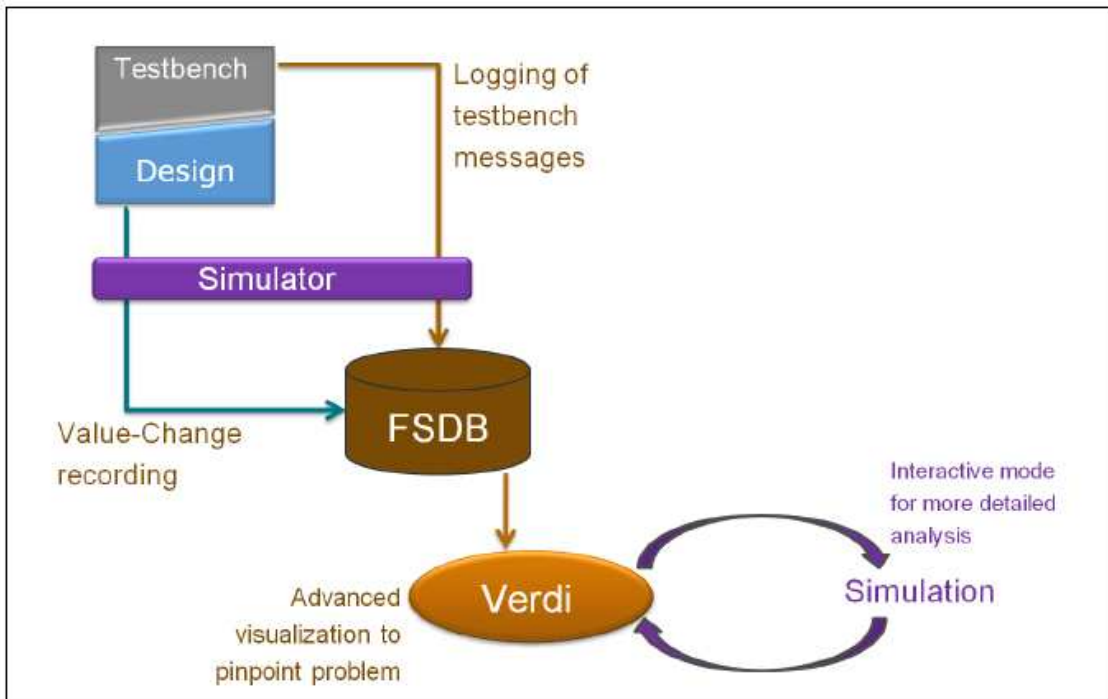


Figure 2: A flow consisting of advanced logging coupled with interactive inspection

As illustrated in Figure 3, the logging output is automatically captured into the same debug database as the design results (SpringSoft's de-facto standard FSDB format). This is fundamental to enabling advanced visualization, debug and analysis functionality.

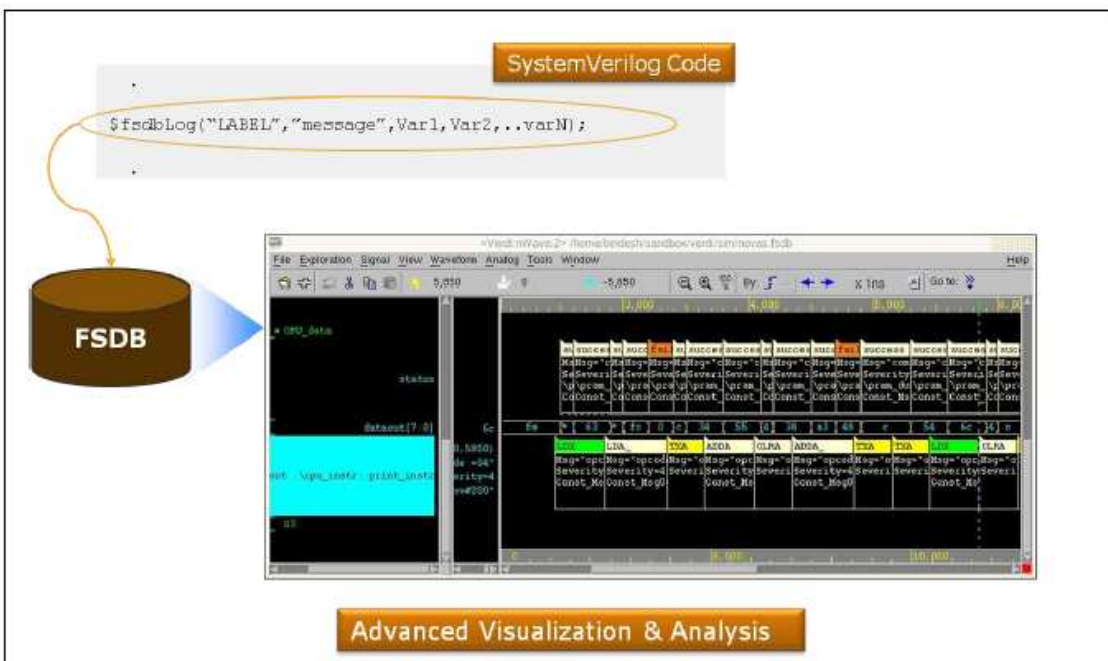


Figure 3: Advanced logging into the FSDB database, allows advanced visualization and analysis in SpringSoft's Verdi Automated Debug System

With advanced logging and Verdi, engineers can observe what is going on in the entire environment. The system captures and allows the engineer to view:

- Transaction-level messages
- Severities, variable states, etc... as properties or attributes of the message

- The call stack, which can later be leveraged in further debug automation

Special-purpose features can be added to these views to help the engineer easily identify messages of interest from the logged data. For example, advanced filtering and highlighting can be used to filter or colorize specific messages based on some condition (e.g., highlight in red any messages that have "WRITE" as their label and "address=5"). Logged message viewing applications also enable engineers to quickly search and locate messages that match user-specified search criteria.

Ongoing Innovation

Advanced logging, coupled with interactive inspection, offers a clear advantage over other currently used approaches for comprehending, analyzing and debugging SystemVerilog testbench environments. Yet it still requires the engineer to instrument the code to log information of interest. OVM and VMM libraries can provide some level of automation for this task, relieving a significant burden from the engineer.

SpringSoft continues to research new techniques for further automation. One idea is to leverage the structure and transaction-based nature of the OVM and VMM libraries, as well as the object-oriented nature of SystemVerilog. Ongoing research is now focusing on pre-embedding relevant logging instrumentation into the standard functions and macros which are used to pass data between different layers of a testbench. This would enable the automatic recording of all transaction-level data passed between the different testbench layers which can then be analyzed in traditional views like waveforms. Moreover, the completeness and easy availability of this data could also potentially drive new views based on UML-like sequence diagrams.

The content is excerpted from the article [Visualizing, Analyzing and Debugging SystemVerilog Testbench Environments](#).

Verdi and Siloti - A Unified and Refined Use Model

In addition to the upcoming Verdi 2009.07 and Siloti 2009.07 releases (scheduled to be released on the 20th of July), a brand new release named "Novas 2009.07" is to be released on the 13th of July. In the new Novas 2009.07 release, the Verdi™ Automated Debug System and the Siloti™ Visibility Automation System are being unified from a use-model and packaging point of view.

These refinements make it easier and more natural for both Siloti and Verdi users. Users will now be able to access Siloti features through the familiar Verdi interface rather than going outside the Verdi flow to prepare and set up for Siloti's unique data expansion capabilities. These improvements will have no impact on licensing.

The new automatic handling of *Essential Signal (ES) FSDB* dump files enables seamless use of the Verdi and Siloti systems in one package and tool. When a non-ES FSDB dump file is loaded into the integrated Novas system, it behaves like Verdi. On the other hand, when an ES FSDB dump file is loaded, the unified tool will automatically detect the essential dump file as such and prompt the user to turn on Data Expansion (DE) at FSDB load-time. At this point, the users can choose not to turn on Data Expansion and debug with limited visibility (Data Expansion can be manually turned on later if needed).

From a packaging point of view, the benefit of the unification of the Verdi and Siloti package is a reduction in the effort required for download and installation – one package instead of two. Additionally, it is easier to integrate the single tool into scripts and existing environments.

Currently, the Novas 2009.07 release is considered **beta**. The unified tool will be available after the 13th of July and posted under the *Novas Verification Enhancement system (beta)* section of the SpringSoft support website (<http://support.springsoft.com>). After the beta version is installed, you can use the "verdi" command to invoke the unified system.

Refer to the `verdi_siloti_consolidation.pdf` file provided in the `<NOVAS_INST_DIR>/doc/notes` directory for additional details.

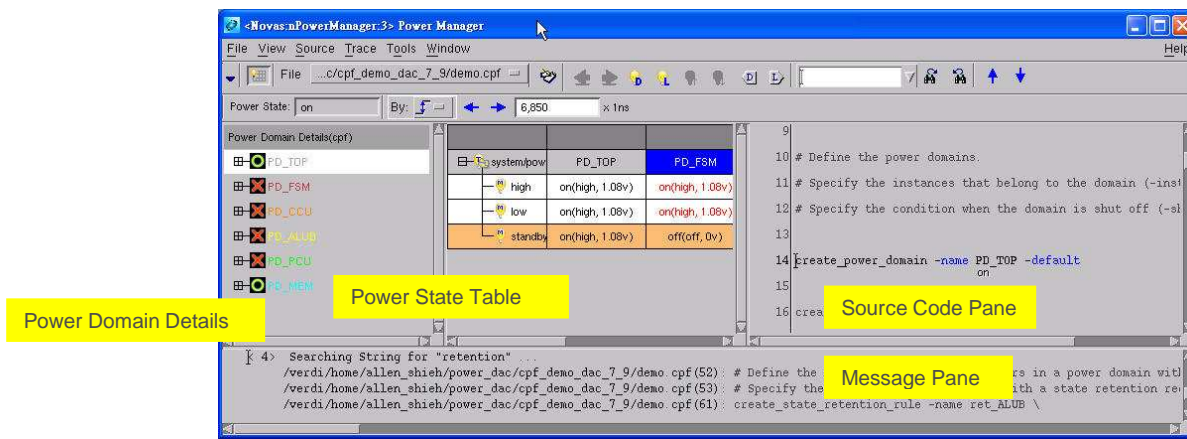
Power Aware Debug

The power architecture in today's designs is getting more and more complex, and multiple power domains with many power modes require a thorough verification method. Some technologies are provided to define and verify the complex power domains. Among these technologies, the standard power formats, Common Power Format (CPF)/Unified Power Format (UPF), and power-aware simulation are widely used to ensure that low-power designs still produce a high confidence chip.

In order to support the standard power formats and also visualize the power intent and simulation results, the Verdi™ Automated Debug system started supporting power-aware debug in 2009.10. The standard Verdi debug capability can be used to debug power issues which are found by these visualizations.

The Power Manager window

The following figure shows the GUI of the *Power Manager* window which will be brought up automatically after users imported the UPF or CPF file. There are four panes in the Power Manager window to visualize the imported power intent: *Power Domain Details*, *Power State Table*, *Source Code Pane*, and *Message Pane*.



The Power Domain Details pane shows all power domains defined in this design and is called the power domain details pane. Every domain is marked with a different color, and these colors can be re-defined by the **Tools → Highlight Power Domain** command in *nTrace*. All power definitions for the selected power domain are shown in a structural way so the related definitions can be easily found. A viewing filter is also provided for the power domain details pane. Use the option commands under **View** to optimize the information displayed in this pane. If the FSDB file has been loaded and the **Source → Active Annotation** toggle command has been turned on, the domain status will be displayed dynamically by “on” or “off” icon according to the time in the **Cursor Time** field on the toolbar.

The Power State Table pane of the browser shows the power state table of the imported design. The table will be enabled by default if there is power mode/power state information in the imported CPF/UPF file. If the **Source → Active Annotation** toggle command has been turned on, the active state will be highlighted. The row for active power mode will be highlighted with an orange background color, and the un-matched state in the non-active rows will be marked with red text.

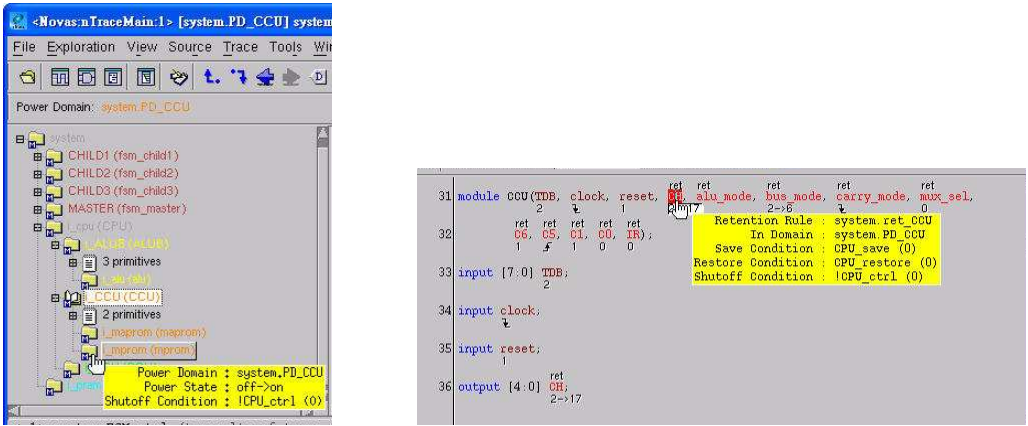
The Source Code Pane of the *Power Manager* window shows the source code of the imported CPF/UPF power file. The content will be changed accordingly when double-clicking different items in the Power Domain Details pane. Just like the source code pane in the *nTrace* window, enabling the **Source → Active Annotation** command will annotate the FSDB value for signals under the source code if the FSDB file is loaded into the Verdi system. Tracing capabilities including tracing HDL driver/load and tracing power driver/load are also available in the Source Code Pane, where tracing power driver/load can help to locate the associated power commands and tracing HDL driver/load provides the same signal tracing capabilities as *nTrace*.

The bottom pane of the *Power Manager* window is called the Message Pane. When you invoke the **Source → Find** command to find any item in the power file, all searched results will be listed in the message pane. The **Find Type** could be **String**, **Signal**, **Scope**, **Domain**, and **Power Net**. Double-clicking on the search result message will jump to the corresponding line in the source code pane.

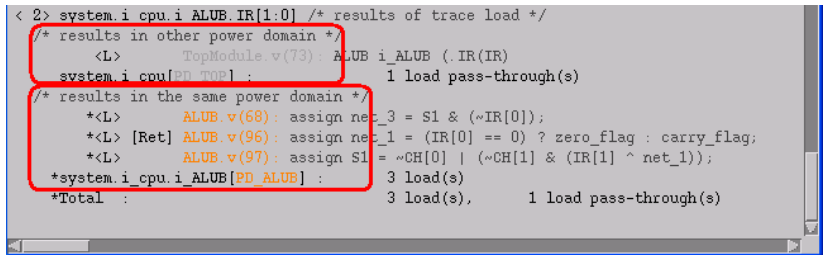
Besides the dedicated *Power Manager* window, the Verdi system also extended the power aware debugging capability to the current debug windows, such as *nTrace*, *nWave*, *nSchema*, and *Temporal Flow View*.

Power Debug in nTrace

After importing the UPF/CPF file, the color for power domains will be highlighted in the *nTrace* Hierarchy Browser pane. Every instance will be highlighted with a different color according to its power domain, and the color is synchronized with the setting in the *Power Manager* window. Besides colorizing instances, all Power Aware Objects (which includes Isolation, Retention, Level Shifted signals...etc) will also be highlighted in the *nTrace* Source Code Pane. Also, a tip with power information is supported in the *nTrace* window. When the mouse cursor is placed over a node in the Hierarchy Browser pane, the node's domain, state, and conditions will be displayed in the yellow tip. When the mouse cursor is placed over a retention/isolation/level-shifted signal in the source code pane, the associated rules and conditions will be shown in the yellow tip. The following figure shows an example for the above capabilities.

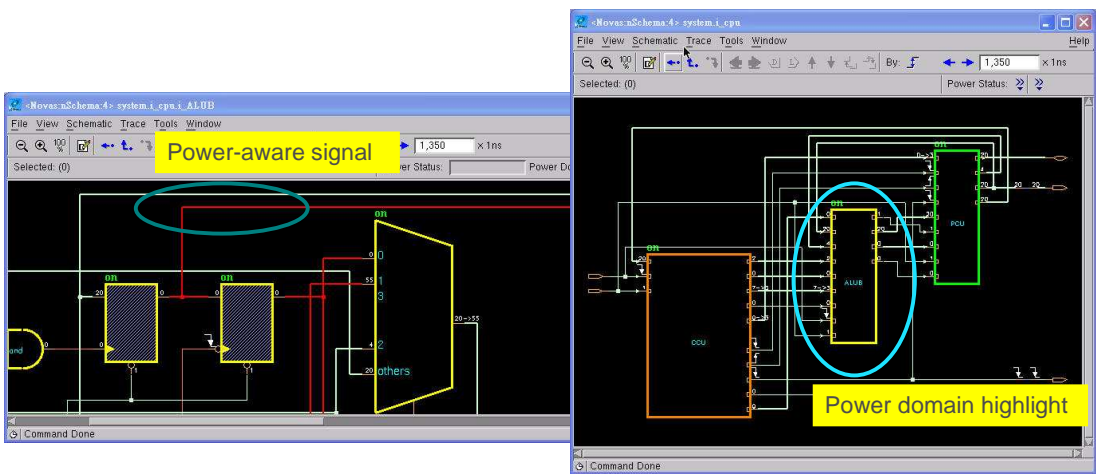


Furthermore, when tracing drivers or loads in *nTrace*, some additional messages will be added in the Message Pane to indicate whether the result belongs to the same power domain or a different one and the result will be colored to match its power domain settings.

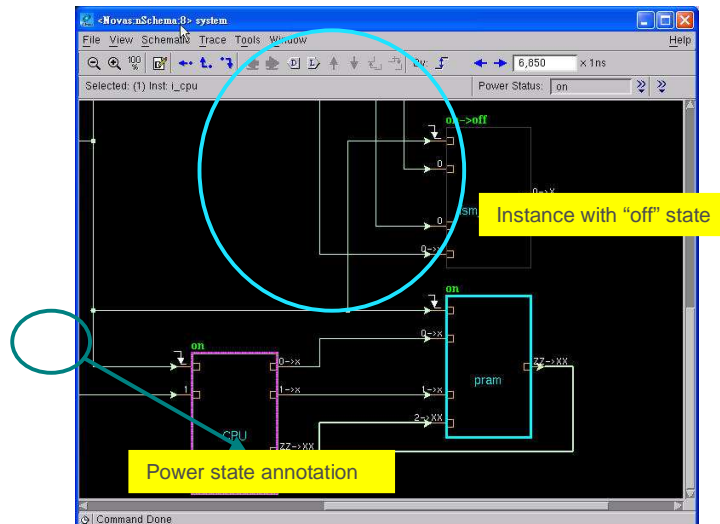


Power Debug in nSchema

Power domains and power aware signals will also be highlighted in the schematic view. The highlight can be turned on/off with the **View → Power Object Color → Power Domain Color** and **View → Power Object Color → Power Signal Color** commands.

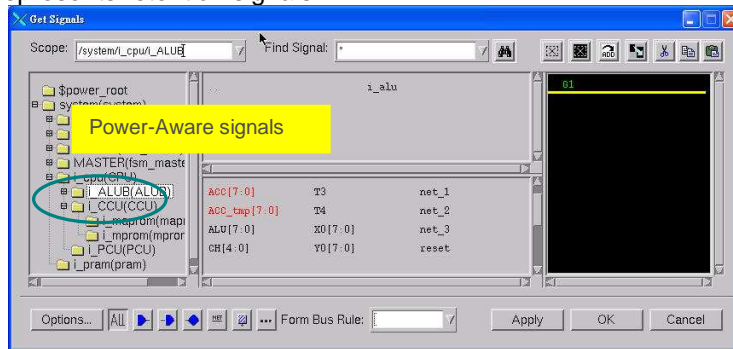


The power state will be annotated after turning on the **Schematic → Active Annotation** command, and the instance with "off" state will be grayed out. The following figure shows an example for this.

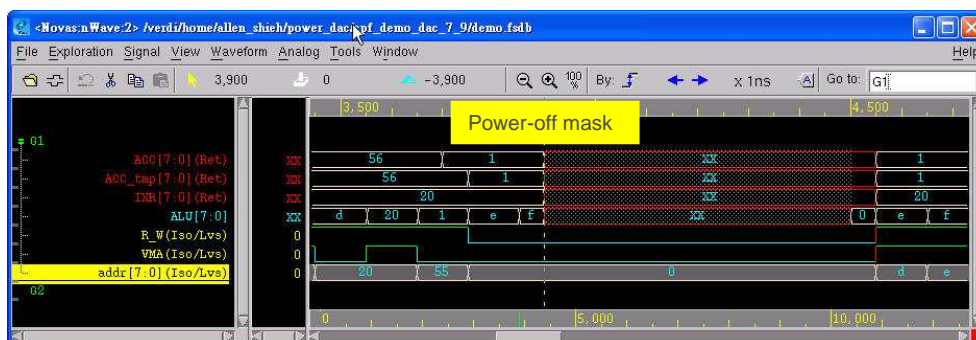


Power Debug in nWave

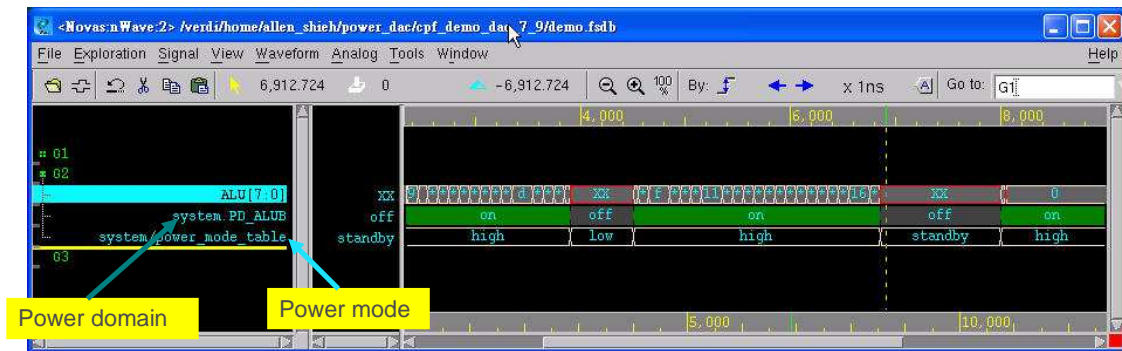
The power aware colors also affect the *nWave* display. When the **Get Signals** command is used in *nWave*, the signal color will follow the setting in this option. The following figure shows an example of color highlighting for power in the *Get Signals* form. The red color represents retention signals.



After adding these signals into *nWave*, the signal retains the color setting in the waveform view. The *nWave* window also adds a suffix to the signal in the signal pane. Turning on the **View → Power Off Mask** toggle command will mask out power-off range in the waveform view. The following figure shows an example for the highlight and power-off mask capabilities.



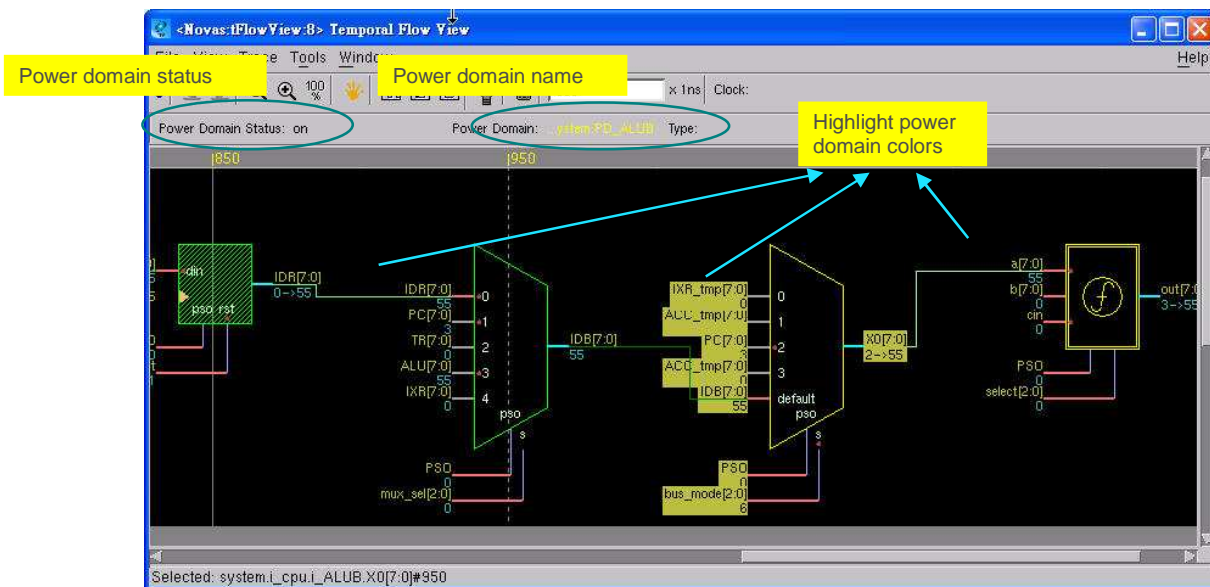
Tracing capabilities for power domains are provided in the waveform view for power-aware signals. For example users can select a signal in the Signal Pane, and invoke the **RMB → Power → Add Power Domain** command to add the power domain which the selected signal belongs to. Users can also select a signal (or domain), and invoke the **RMB → Power → Add Power Mode** command to add the power mode under the selected signal. The following figure shows an example for these two tracing capabilities.



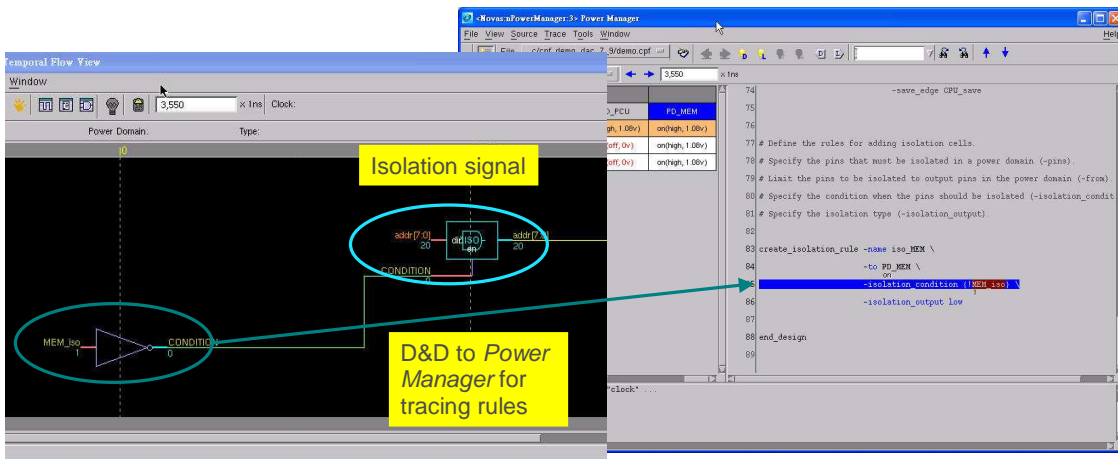
Tracing capabilities are also provided for tracing power-aware signals like Retention, Isolation, and Level-Shifted signals. These tracing commands will trace to the power aware rule and its affected controls and signals, and shows the traced result on *nWave* and the *Power Manager* window. Users can click the right mouse button on these power aware signals to bring up tracing commands.

Power Debug in Temporal Flow View

The *Temporal Flow View* (TFV) window also supports power domain highlight. The path shown in the TFV will have instances highlighted with the color of the respective power domains and the RET/ISO annotation for retention/isolation latches/FFs if there are any in the path. The Power domain name and status will be shown on toolbar. The following figure shows an example for the highlight.



A dedicated symbol is also provided to represent isolation signals. Users can double-click on the CONDITION pin to trace the condition for isolation rule, and Drag & Drop the CONDITION symbol to the *Power Manager* window for tracing power rules. See the following figure for an example:



To read more about the evolution of power-aware debug and the advancements that the Verdi system provides, please see the whitepaper titled *Challenges and Requirements for Power-Aware Debug*.

For more detailed commands and usage, please refer to the *Novas Command Reference* manual.

Data Management: Auto Check-out and Check-in

Introduction

Data Management (DM), revision or version control, is used to manage the data and information related to a project. Generally, the project is ongoing and worked on by multiple members of a design team. Only one member of the design team may have write access to modify data files at any given time. To modify data, a member will need to check-out the data before making modifications. After modifications are complete, the member will then check-in the data making it available to all of the other design team members.

The check-in and check-out of data is not a time consuming or tedious task, but does add a couple of extra steps in the editing process that most designers would like to be able to avoid. Automating the check-out and check-in of data has helped to streamline the process making editing of data simpler for design teams.

Triggering Auto Check-out and Auto Check-in

There are several settings that need to be in place to enable auto check-out and auto check-in of data.

Auto Check-out and Check-in Settings

The generic settings for DM are DMToolType, DMServerName, and DMServerPort. They are specified in the DataManage section of the Laker resource file which is called laker.rc. When using CVS, 3ds Enovia DesignSync®, or ClioSoft SOS, set DMToolType to CVS, DesignSync, or SOS, respectively. The 2 variables, DMServerName and DMServerPort, are not required when using DesignSync or SOS.

Example:

```
[DataManage]
```

```
DMToolType = SOS
```

```
DMServerName = localhost
```

```
DMServerPort = 2647
```

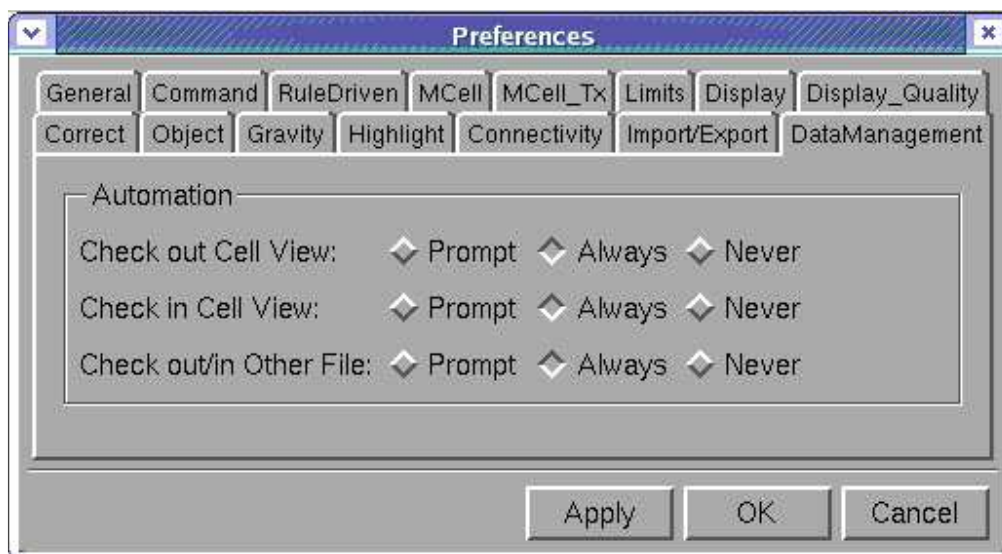


Figure 4: Options → Preferences → DataManagement

There are three options for auto check-out and auto check-in available under **Automation** in the **DataManagement** tab of the **Options→Preferences** form.

- **Check out Cell View:** Specifies the way to handle the check-out of a cellview.
 - **Prompt:** Invoke a prompt form asking the user to check out the cellview.
 - **Always:** Always check out the cellview.

- **Never:** Never check out the cellview.
- **Check in Cell View:** Specifies the way to handle the check-in of a cellview.
 - **Prompt:** Invoke a prompt form asking the user to check in the cellview. The prompt form will only appear when saving a cellview or exiting Laker.
 - **Always:** Always check in the cellview.
 - **Never:** Never check in the cellview.
- **Check out/in Other File:** Specifies the way to handle the check-out and check-in of non-Laker data files such as libfile, techfile, etc.
 - **Prompt:** A prompt form is invoked asking the user to check-out or check-in a file.
 - **Always:** Always check-out or check-in the file.
 - **Never:** Never check-out or check-in the file.

To have data always checked-out or checked-in make sure all three options are set to **Always**.

Laker Auto Check-out and Check-in Scheme

Auto check-out and auto check-in are triggered with all features that open a cellview via dbOpen. Examples are **File → Open** on the Laker Main Window or the Laker Layout Window. When the opened cellview is closed, auto check-in will be triggered. The Laker auto check-in scheme handles only the cellviews that were opened by the auto check-out scheme. Once a cellview is created via Laker features, such as new cell, copy cell, rename cell, make cell, etc..., the auto check-in scheme will be triggered when the cellview is closed. If the check-out and check-in scheme settings are set to **Always**, check-out and check-in will be automatic and transparent to the user. An initial checkin will need to be performed on a library before auto check-out and check-in will take effect.

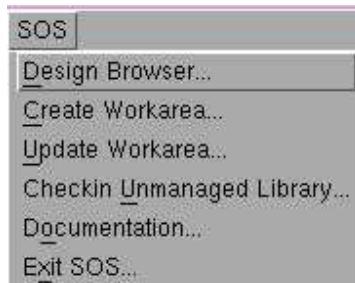


Figure 5: SOS --> Checkin Unmanaged Library

The SOS Design Browser does not support auto check-out and auto check-in. You will receive a prompt even if your settings are set to **Always**.

Laker and SOS compatibility

For auto check-out and auto check-in to work properly the following version combinations of Laker and SOS should be used:

- Laker 3.2v3p8 and SOS 6.20.p3
- Laker 3.2v3p9 and SOS 6.20.p4
- Laker 3.2v3p10 and SOS 6.20.p4
- Laker 3.2v4p2 and SOS 6.20.p4

Laker Custom IC Design: Forward And Backward Compatibility

Software is considered **backward** (or downward) **compatible** if newer versions or releases can read, view, and operate on data that was created with prior versions or releases. This should not be confused with **forward compatible** which means that older software can read, view, and operate on data that has been created with newer releases. Most design automation software is backward compatible. When new versions are released, users may install the newer release, and continue to operate on their existing data. If, however, they utilize features in the newer release that were not present in the prior release, they may encounter forward compatibility issues if they revert back to the prior release. Not all new features will cause problems, but some may. For example if data created in the new release is stored differently in the database, the older release may not be able to recognize it. Most design automation software is not forward compatible. Laker 2009.09 is not forward compatible but it is backward compatible. There are new objects in Laker 2009.09 that when used will render cells non-readable in prior versions like Laker 3.2v4. For example, there are new parameters for contact width and spacing available for MCells. If Laker 2009.09 is used to create them in a design, that design will no longer be viewable in Laker 3.2v4 since Laker 3.2v4 does not support these parameters. Another example, User-Defined-Devices (UDDs) may be encrypted with Laker 2009.09 so that other users may not view or modify the device. This was not available in Laker 3.2v4, so UDDs encrypted with Laker 2009.09 will not be accessible, readable, or compatible in Laker 3.2v4.

There are situations where a user created a design in a newer release and was then able to use a prior release to view or modify the design. In these particular situations, new or modified features had not been used. Once a design is moved to a newer software release, it is recommended that it stay with the newer release and not be attempted to be opened or modified with a prior release, as newer or enhanced features may have been utilized and these features may not be available in prior releases.