Verdi Fundamental Training

Based on Verdi 2010.01

SpringSoft Notifications

Copyright

Trademarks

The product names used in this document are the trademarks or registered trademarks of their respective owners.

Confidentiality

The information in this document is confidential and is covered by a license agreement between SpringSoft and your organization. Distribution and disclosure are restricted.

Goals

- To help you understand complex designs more easily.
- To help you trace back to the root cause quickly when finding bugs.
- To do debugging and verification in one unified and friendly environment.

Target Audience

- Architecture engineer
- Development engineer
- Verification engineer

Prerequisites

- Basic HDL/HVL coding skills: Verilog, VHDL, SystemVerilog, SVA.
- Familiarity with standard simulators.

Overview

- Technology Background
- Set Up the Environment
- Understand FSDB Dumping Tasks and Utilities
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM view
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

Glossary

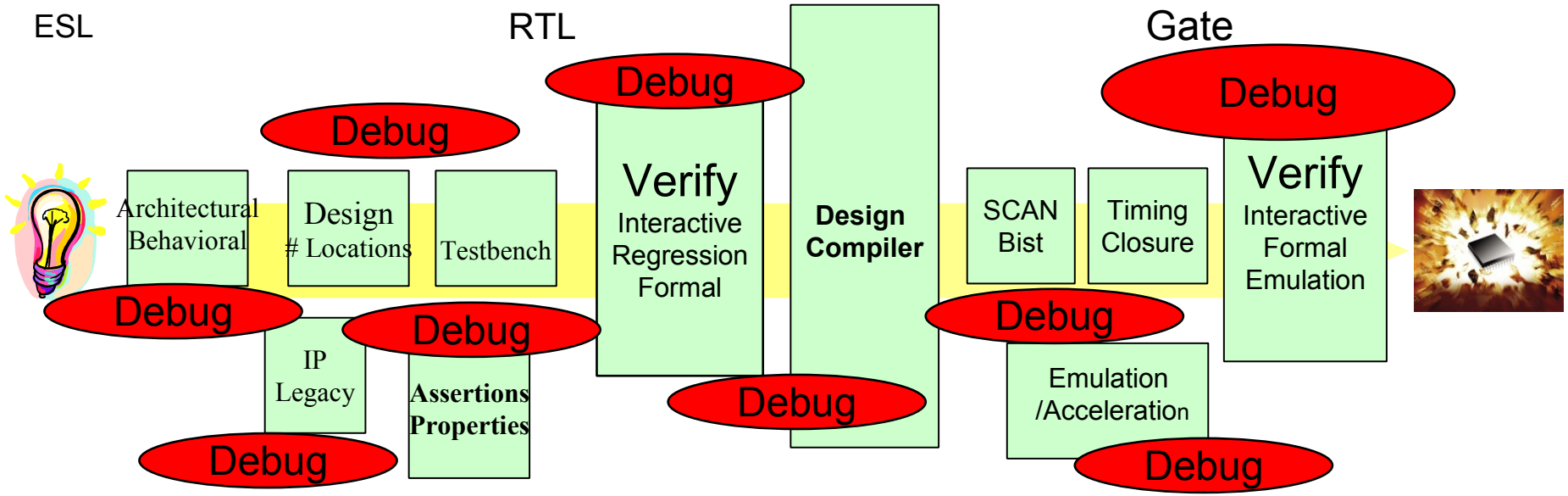
- RMB = Right Mouse Button
- MMB = Middle Mouse Button
- LMB = Left Mouse Button
- DC = Double-click
- D&D = Drag and Drop
- KDB = Knowledge Database
- FSDB = Fast Signal Database
- TFV = Temporal Flow View
- BA = Behavior Analysis

Topics

- Technology Background
- Set Up the Environment
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM View
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

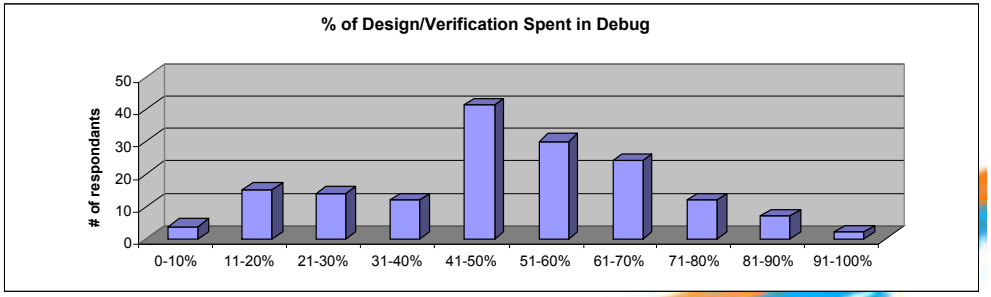
Technology Background

Debug in the Design Flow



Do you know how much time your teams spend debugging?

What else could you be doing with that time?



Source: Novas Customer Survey, 161 Respondents

Barriers to Efficient Debug

- Complex Designs
  - Understanding is more difficult
  - Complex behavior
    - Complex design cause and effect scenarios
    - Increases conditions to be verified
  - Complex multi-tool, multi-team environments
    - Compounds methodology complexity

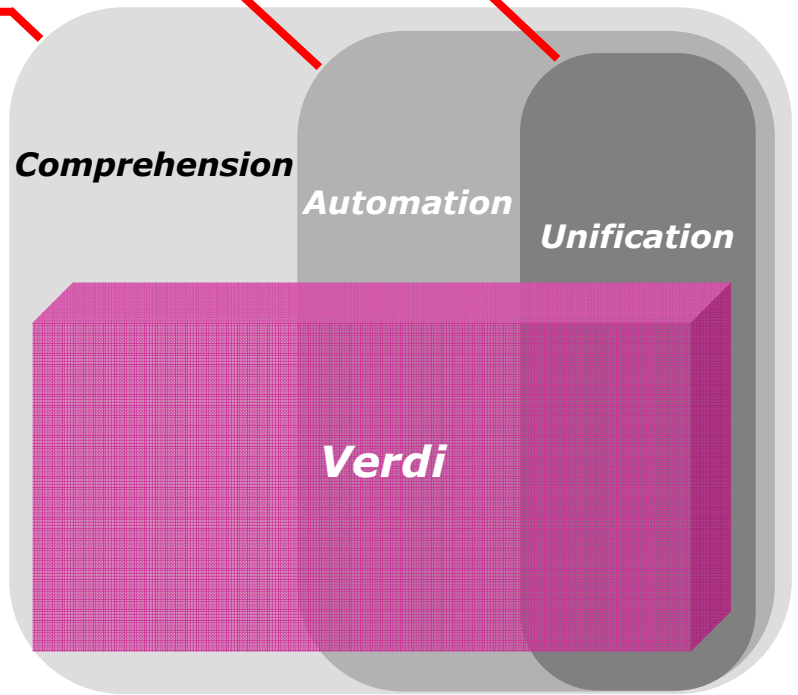
Novas Solves the Tough Debug Issues

Issues

Complex Environments

Complex Behavior

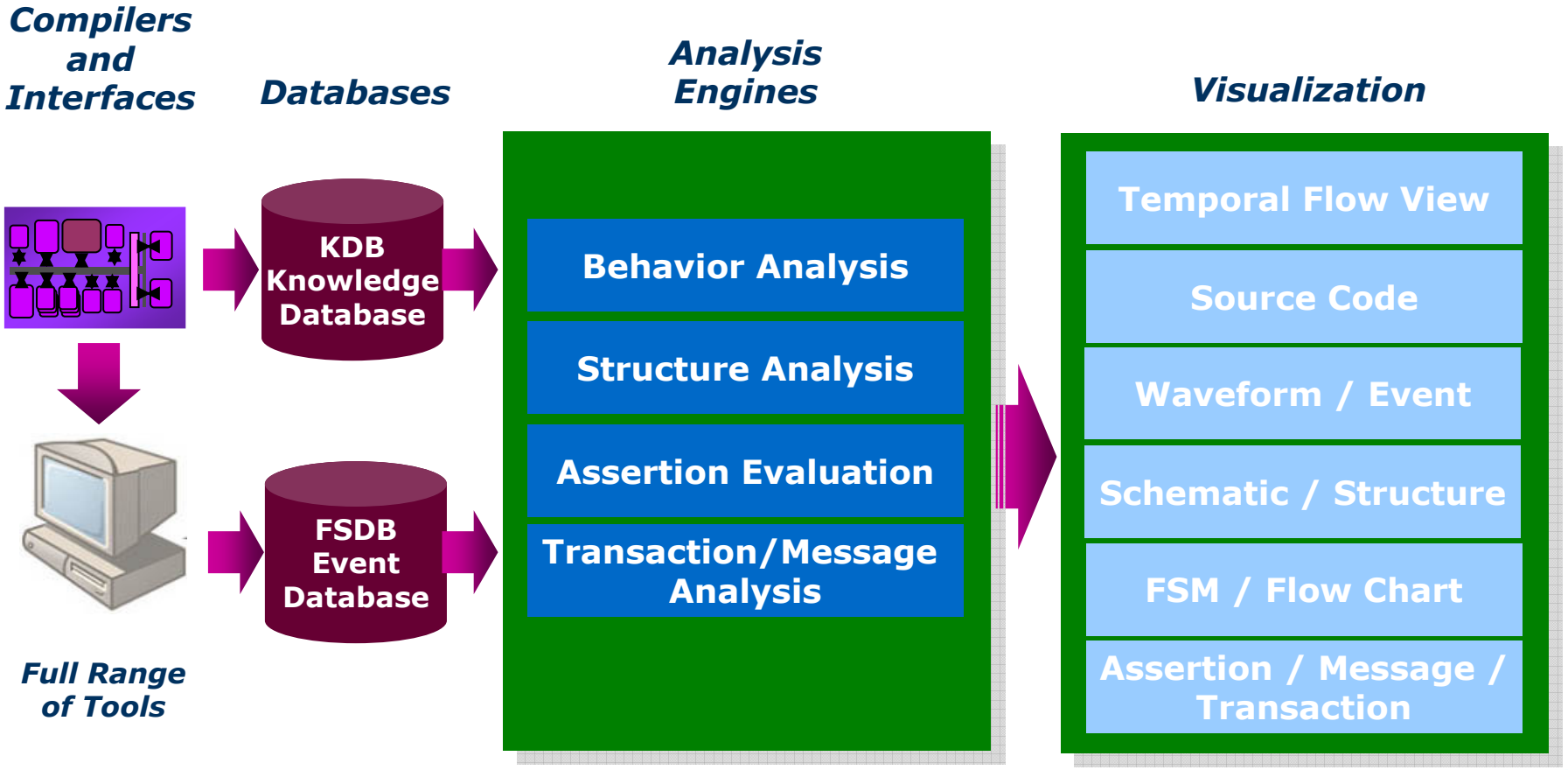
Complex Designs



Increasing automation

Powerful comprehension and debug

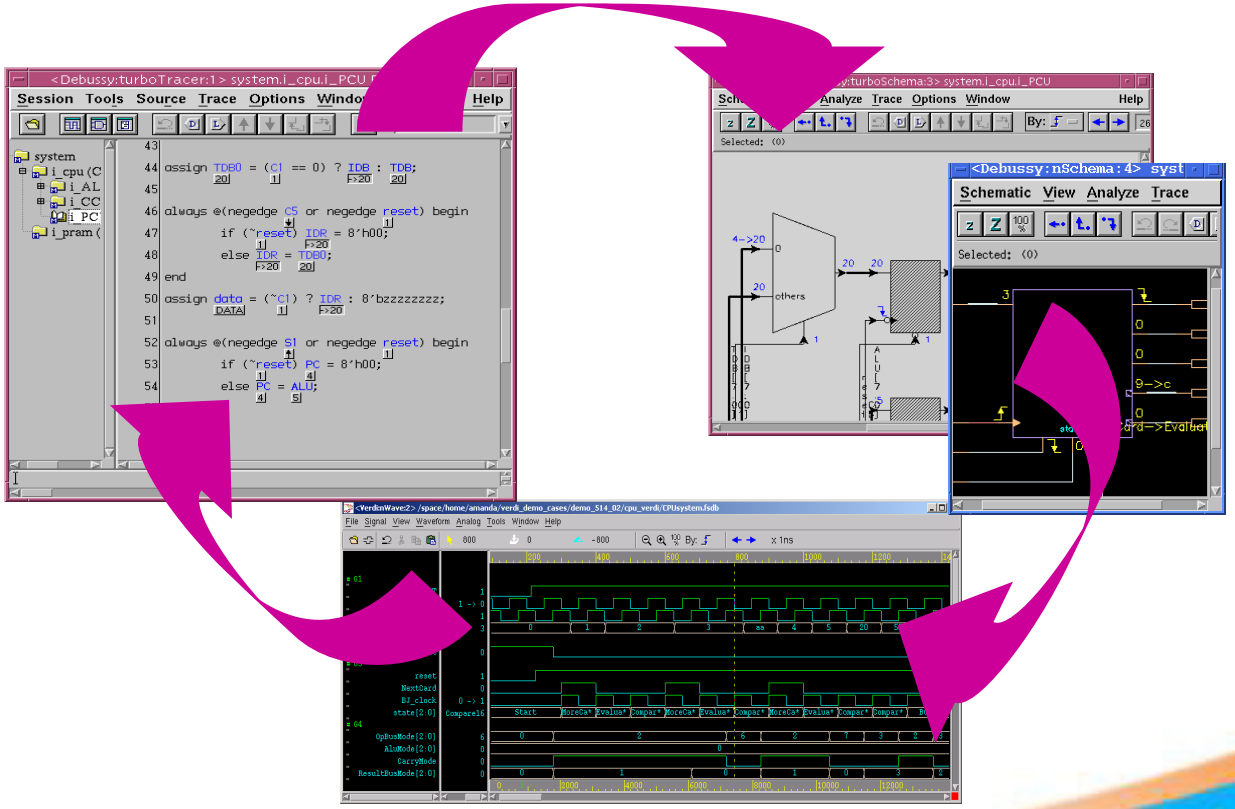
Novas Technology



Comprehensive Debug System - More Than Simple Visualization

Structural / Event Debug Process

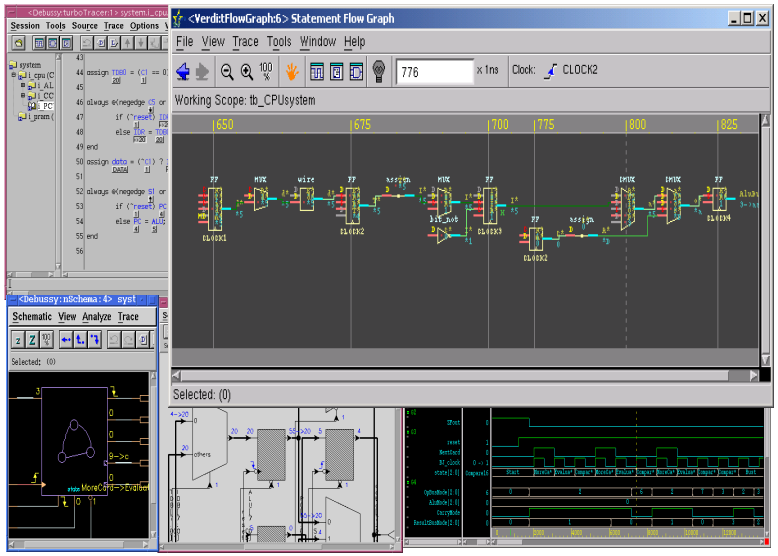
Current process involves comparing events, code and structure across multiple windows to track possible issues
- Time consuming, error prone, complicated



Automating Debug Critical Productivity Enhancement

Utilizing behavior based debug to automate time consuming aspects of debug process

- Time and structure in single view
  - Immediate visual of design behavior
- Automated cause / affect tracing
  - Understand problem areas quickly
  - Cuts your debug time over traditional methods



EDN Magazine Innovation Award 2002 Finalist

Topics

- Technology Background
- Set Up the Environment
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM View
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

Objectives

Set Up the Environment

- After completing this section, you should be able to…
  - Set up the Environment Before Running
    - Specify the Path and Environment Variables
      - Binary, License, Symbol Libraries
    - Use the Setup File – novas.rc
  - Specify Preferences
  - Understand FSDB Dumping Tasks and Utilities
    - Fast Signal Data Base (FSDB) Introduction
    - Link Novas Object Files for FSDB Dumping with Simulators
    - FSDB Utilities

Specify the Path and Environment Variables

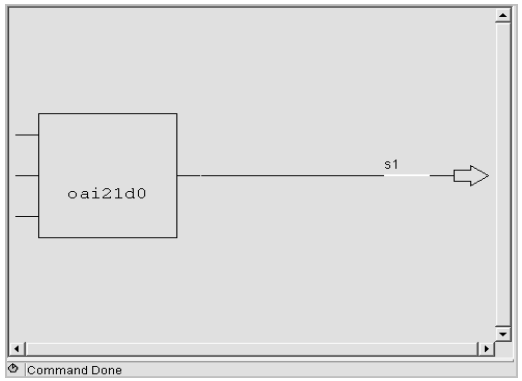
- Binary File
  - *setenv NOVAS_INST_DIR <Path to Novas installation>*
  - *set path = ($NOVAS_INST_DIR/bin $path)*

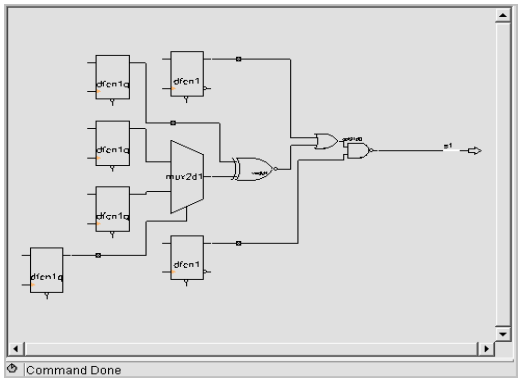
- License File
  - *setenv NOVAS_LICENSE_FILE <license_file>:$NOVAS_LICENSE_FILE*
  - Or alternatively use *LM_LICENSE_FILE*
    - The search priority will be lower than *NOVAS_LICENSE_FILE*

Symbol Libraries Background

- Why use symbol libraries?
  - Provides mapping between source code and logic cell.
- Without the symbol mappings
  - Active fan-in cone, fan-in cone and fan-out cone schematics will **NOT** function properly.
  - Schematics will show **ONLY** squares instead of logic cells.

Unmapped fan-in cone





Mapped fan-in cone

Using Symbol Libraries

- Create a symbol library using one of the following methods:
  - **syn2SymDB**: For Synopsys logical library file in ACSII format (.lib).
  - **map2SymDB**: For the prepared map file.
  - Check the document **Library Developer's Guide** from <verdi_install>/doc/Symbol_Library.pdf for detailed format.

- Set the symbol library environment variables

  setenv NOVAS_LIBS "<LIB ROOT#1> <LIB ROOT#2> ..."

  setenv NOVAS_LIBPATHS "<Directory#1> <Directory#2> …"

- Set the symbol library through the GUI.
  - Invoke **Tools → Preferences** command to open *Preferences* form.
  - Select the folder **Schematics → Symbol Library.**
  - Input the symbol library name in **Symbol Library Names** field.
  - Input the path in **Symbol Library Paths** field.
  - The setting will be saved into the *novas.rc* resource file.

Use Setup File – novas.rc

- Contains:
  - Mapping information for pre-compiled designs.
    - Map a library logical name to a physical location
    - Mapping format

      [Library]
      logical name = physical location (ex, pack = ./work)
      pack = ../library/pack (ex, vital = ./work)
  - User preferences (set via **Tools → Preferences**).

- Specify path…
  - Using NOVAS_RC environment variable.

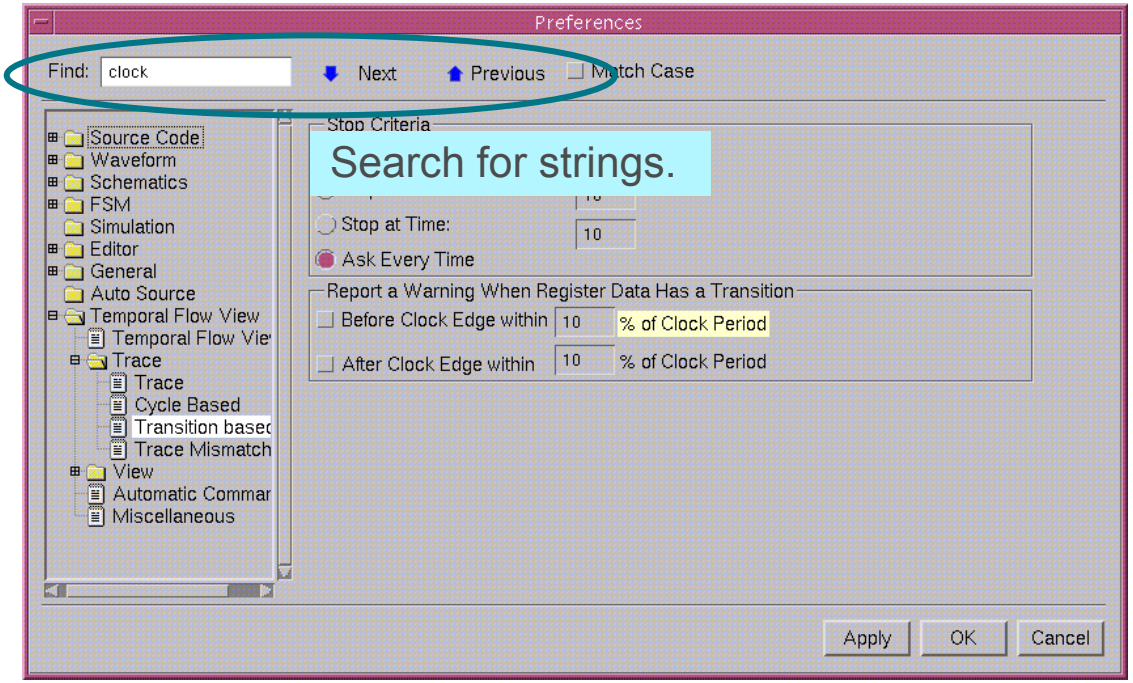
    `setenv NOVAS_RC <path>/novas.rc`
  - Directly by -rcFile <filename> on the command line.

- Search Sequence
  1. -rcFile <filename> command line option (read/write pointer)
  2. NOVAS_RC environment variable (read/write pointer)
  3. ./novas.rc
  4. $HOME/novas.rc
  5. <Verdi install>/etc/novas.rc

Specify Preferences

Overview

- Open *Preferences* form by invoking **Tools → Preferences** from any window.
  - Changes to the form are saved in the *novas.rc* resource file.
  - Search for keywords in the **Find** field.



- Refer to the Appendix for Frequently Used Preferences.

Fast Signal Data Base (FSDB)

Introduction

- Compact binary file format which contains the simulation signal data.
- Open file format so other vendor tools can dump data, e.g. Verisity, Vera, Ikos, etc.
- Use system tasks for both VHDL and Verilog to dump values during simulation.

Fast Signal Data Base (FSDB)

Frequently Used System Tasks

- **fsdbDumpfile** - Specify FSDB file name, limit the FSDB file size
  - Uses the sliding window scheme to keep the last signal values in the FSDB file, and drops the old values if file size exceeds the limitation.
- **fsdbDumpvars** - Dump signal value changes of specified instances and depth.
  - Can also specify the FSDB file name, different FSDB file names can be specified in each fsdbDumpvars command.
- **fsdbDumpon/ fsdbDumpoff** - Turn on/off FSDB dumping.
  - Can specify FSDB file name to turn on/off specific FSDB file.
- **fsdbSwitchDumpFile** - Switch dumping to another FSDB file.
- **fsdbAutoSwitchDumpfile** - Limit FSDB file size and switch dumping to new FSDB file automatically.

NOTE 1: The above tasks are for VHDL simulators, add “$” in the prefix for Verilog simulators.

NOTE 2: For more system tasks, refer to *Linking Novas Files with Simulators and Enabling FSDB Dumping* (<NOVAS_INST_DIR>/doc/linking_dumping.pdf).

Link Novas Object File for FSDB Dumping

- Setup tool environment variables:
  - setenv LM_LICENSE_FILE <simulator_license_file>
  - <Simulator>_INST_DIR
  - NOVAS_INST_DIR
  - Use LD_LIBRARY_PATH to specify the appropriate shared object files, and common library located in **${NOVAS_INST_DIR}/share/PLI/lib**

    ```
    setenv LD_LIBRARY_PATH
    ${NOVAS_INST_DIR}/share/PLI/<simulator>/<platform>:
    ${NOVAS_INST_DIR}/share/PLI/lib/<platform>
    ```

    - Supported simulators: **VCS** / **IUS** / **MODELSIM**
- Create working library and compile designs for simulation.
- Run and call FSDB dump functions on the simulator command prompt.

NOTE: For more simulator linking details, refer to the <NOVAS_INST_DIR>/doc/linking_dumping.pdf file.

FSDB Utilities

Process FSDB Files in Batch

- **vfast**
  - Convert VCD files to FSDB files.
- **fsdb2vcd**
  - Convert FSDB files to VCD files.
- **fsdbextract**
  - Extract signals, scopes, time periods from existing FSDB files without re-simulating.
- **fsdbmerge**
  - Merge several FSDB files into one.
- **fsdbreport**
  - Generates a report of value changes for specified signals.

NOTE: Use <utility> -h for a list of all options or see the *Verdi Command Reference* for more information.

NOTE: Refer to the *Utilities* chapter of the *Verdi Command Reference* for the complete utility list.

Summary

- In this section, you have learned…
  - How to set up the environment before running Verdi.
  - How to specify preferences.
  - How to understand FSDB dumping tasks and utilities.

Topics

- Technology Background
- Set Up the Environment
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM View
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

Objectives

Import Design

- After completing this section, you should be able to…
  - Import Design on Command Line
    - From File
    - From Library
    - By Replaying a TCL Command File
  - Import Design from GUI
    - From File
    - From Library

Import Design on Command Line

From File

verdi [Verdi options] [<your Verilog options>]

- Reference source files on the command line (Verilog only).
- Verdi takes all the Verilog command line options.

  % verdi –f <file_list> +define+GETTHEMOSTOUTOFYOURTIME

  % verdi <source_file1> <source_file2>…

  - Where file_list is a file that contains the Verilog source files and any command line switches.

  % verdi –ssv –v lib.v +libext+.v –ssy –y /src/abc –y /src/def top.v

  - Where –ssv and –ssy are needed only once for design modules specified with the –v and –y and top.v is the design file.

Import Design on Command Line

From Library

- Compile once, load many times.
- Saves load time and overall memory.
- Necessary for mixed language or VHDL designs; optional for Verilog only language designs.

- Pre-compile design into library: *vericom* for Verilog code, *vhdlcom* for VHDL code.

```
% vericom -lib <libName> block1.v block2.v block3.v
% vhdlcom -lib <libName> block1.vhd block2.vhd block3.vhd
```

- Then import the compiled libraries and specify the top module.

```
% verdi -lib <libName> -top TopBlock
```

For example:

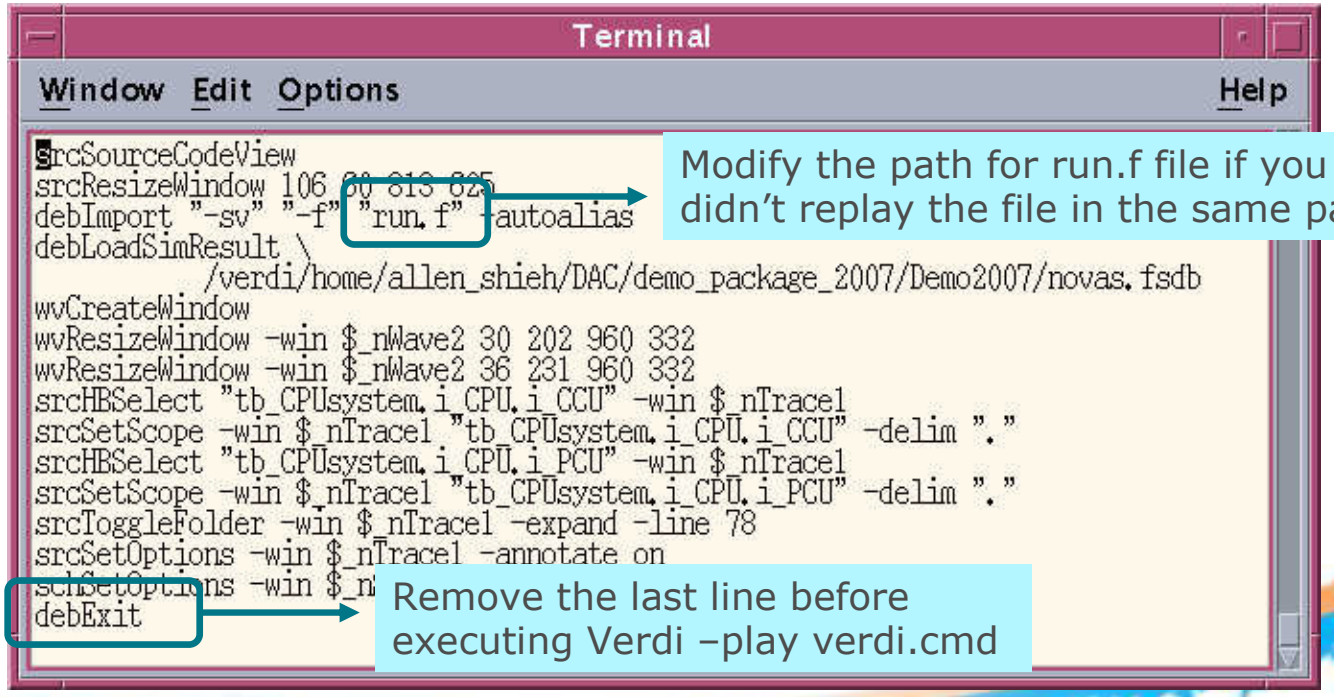
```
% verdi -lib work -top system
```

Import Design on Command Line

By Replaying a TCL Command File

- Every TCL command executed in Verdi has an equivalent TCL command saved in the ./verdiLog/verdi.cmd file.
  - This file may be copied and modified to automate frequent commands.
  - The new file may then be replayed on the Verdi command line using:

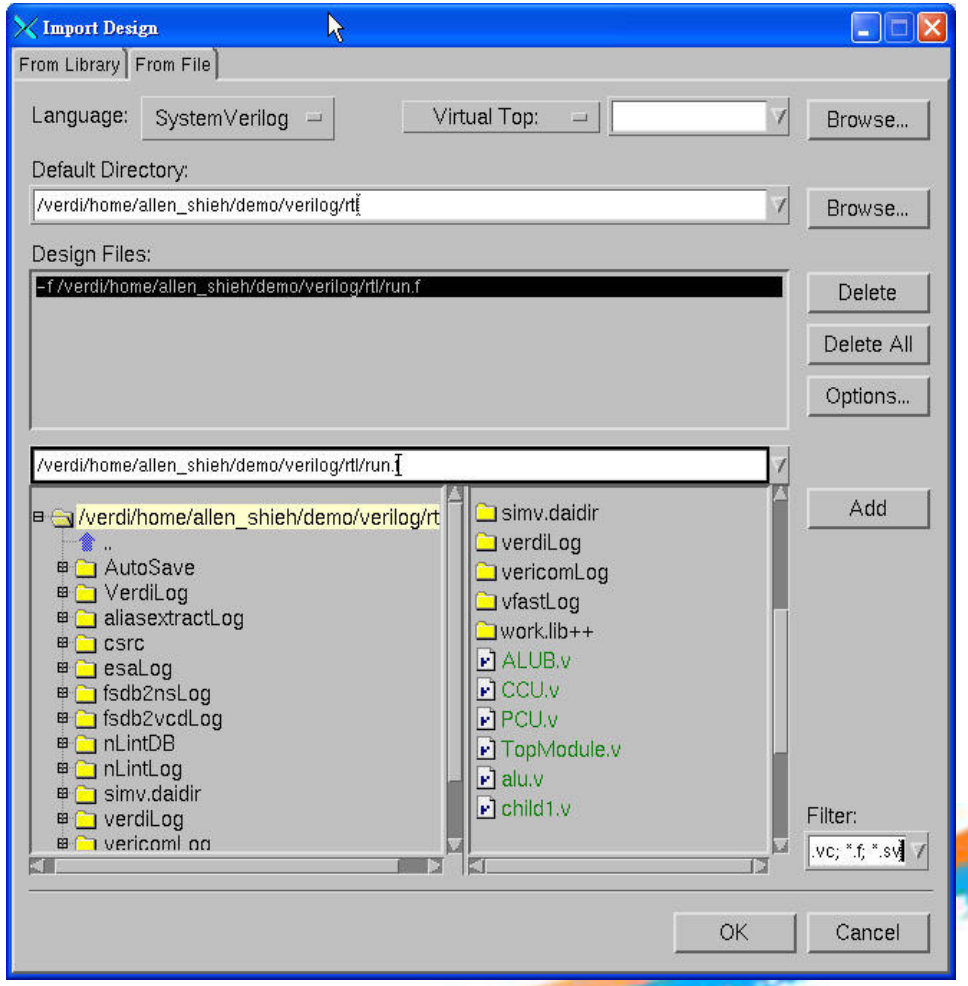
    % verdi –play <file>.cmd
- Example for how to modify a verdi.cmd file:



Import Design from GUI

From File

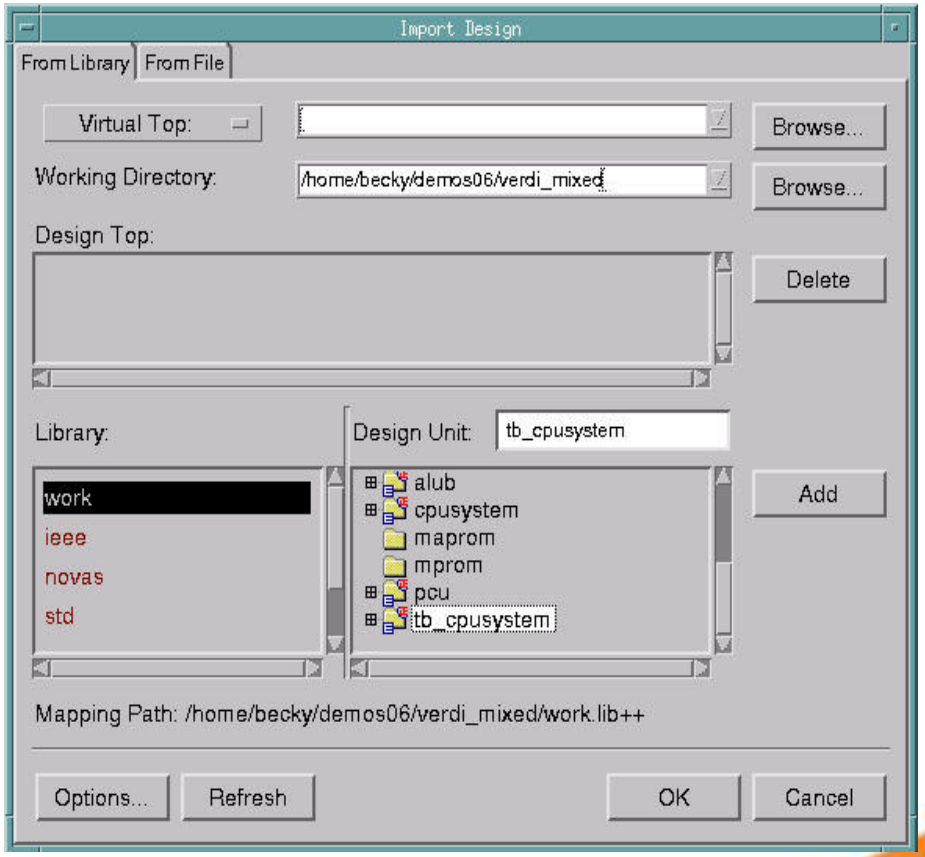
- Enter **verdi** on command line to launch Verdi.
- Use **File → Import Design** or **Import Design** icon  to import your design.
  - Select **From File** tab.
  - Specify language.
  - Specify run file or individual source files.
  - Supports mapping file to set a Virtual Top.
    - Refer to *Verdi Application Training* for the details of Virtual Top.
- Not recommended for VHDL.



Import Design from GUI

From Library

- Enter **verdi** on command line to launch Verdi.
- Use **File → Import Design** or **Import Design** icon to import your design.
  - Select **From Library** tab.
  - Specify the library and the top module.
  - Supports mapping file to set a Virtual Top.
    - Refer to *Verdi Application Training* for the details of Virtual Top.



Summary

- In this section, you have learned…
  - How to import designs on the command line
    - From file
    - From library
    - By replaying a TCL command file
  - How to import designs from the GUI
    - From file
    - From library

Topics

- Technology Background
- Set Up the Environment
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM View
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

Objectives

Debug in Source Code View

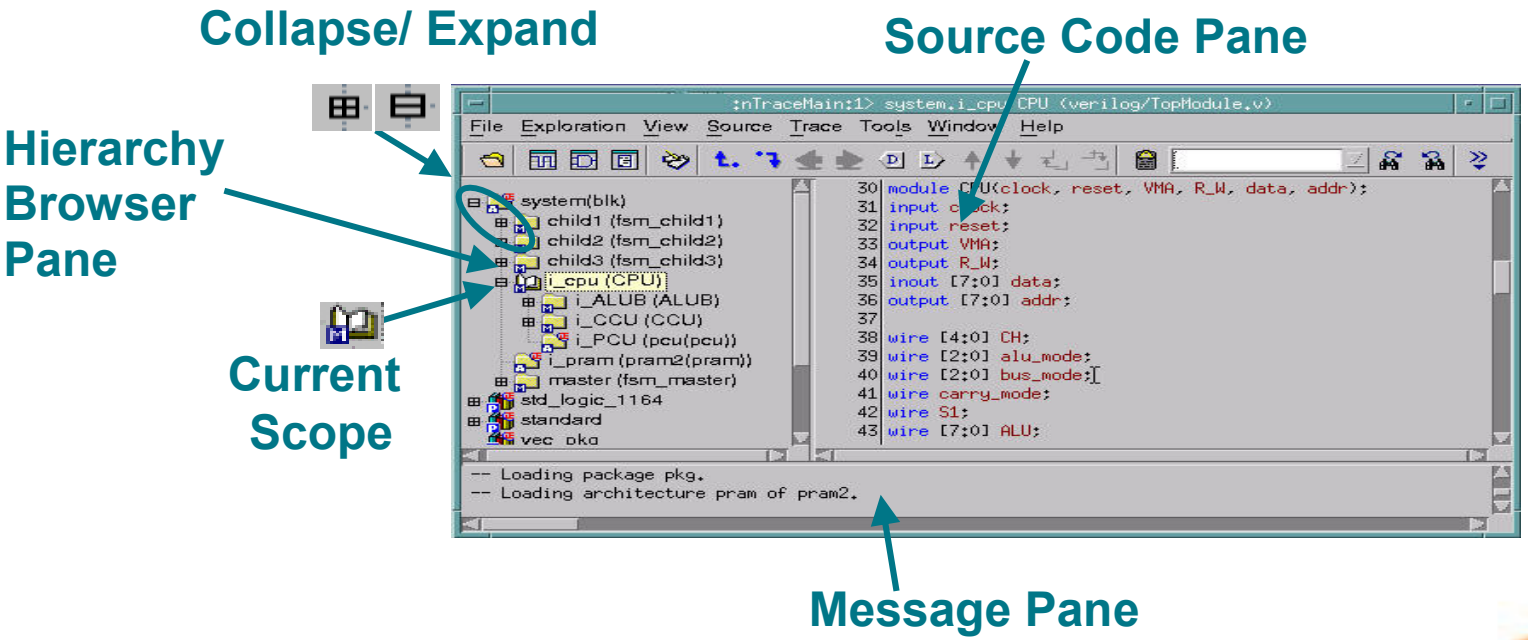
- After completing this section, you should be able to…
  - Open *nTrace* Window
  - Search and Collapse Hierarchy Browser
  - Traverse the Source Code
    - Find Operations
    - Use Bookmarks
    - Collapse Source Code
    - Tip to Show Implicit Port
    - Double-Click Module to Show All Instances
  - Operate on Signals
    - Trace Signals – Driver, Load, Connectivity, Fan-in, Fan-out
    - Enable Active Annotation
    - Perform Active Trace
  - Correlate Other Views

Open *nTrace* Window

- After importing the design, Verdi invokes the *nTrace* Window automatically.

  % verdi –f run.f

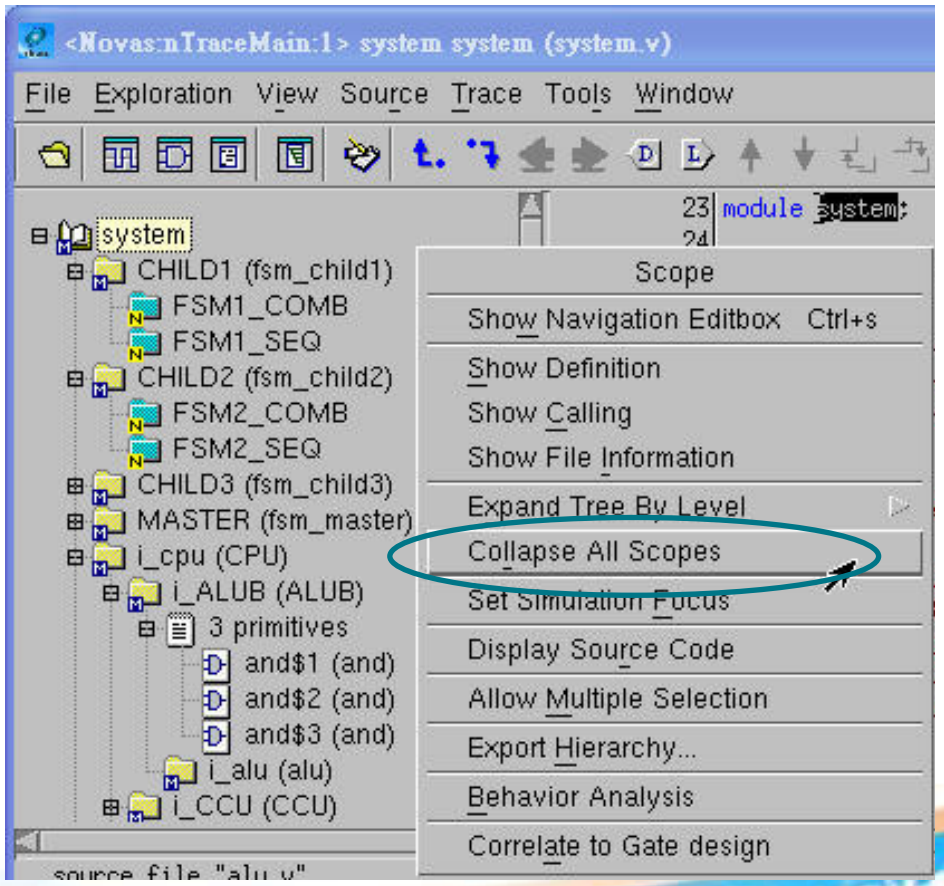
- The *nTrace* window contains three re-sizable sub-windows:



Search and Collapse Hierarchy Browser

Collapse All Scopes

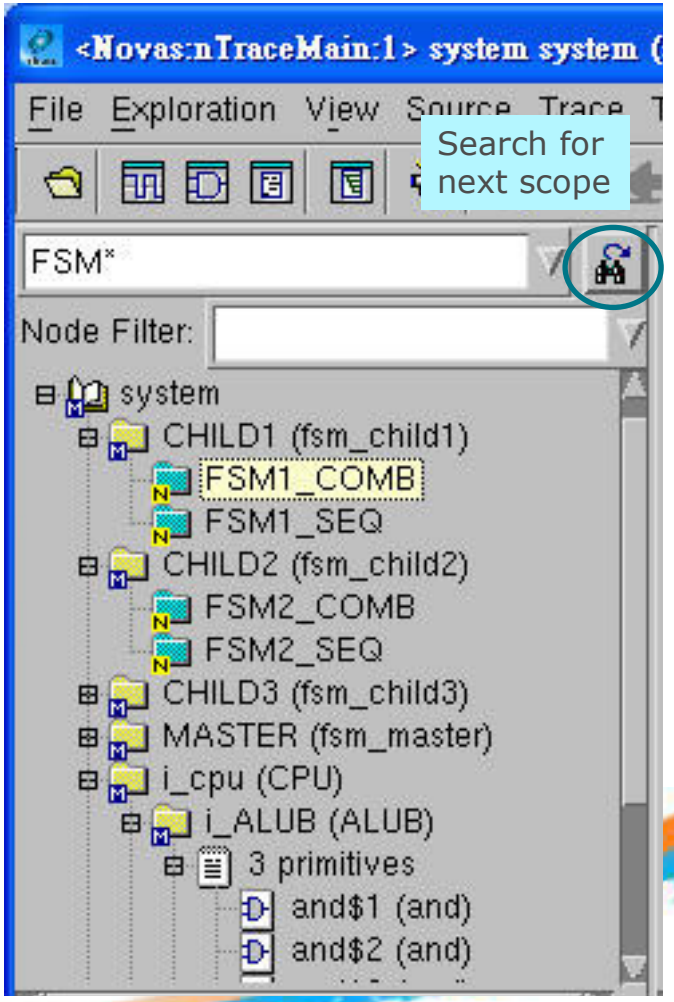
- In the *hierarchy browser pane*, click the Right Mouse Button (RMB) to select the **Collapse All Scopes** command.
  - Collapses all open scopes.



Search and Collapse Hierarchy Browser

Search Scopes

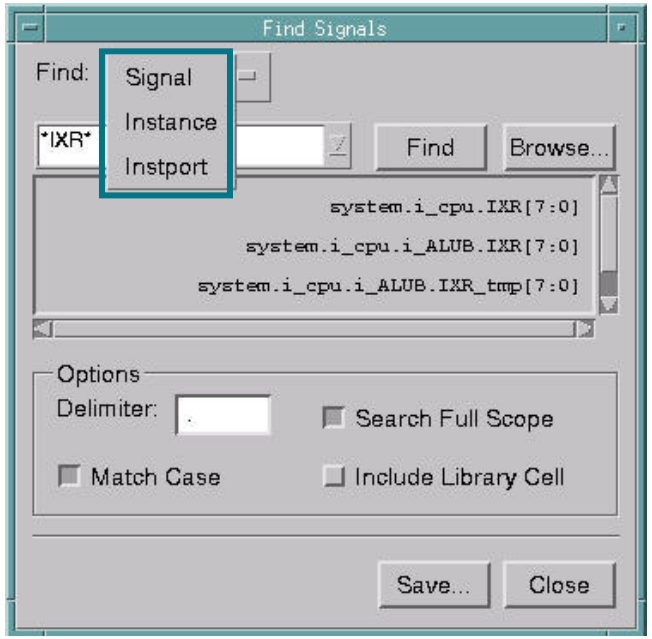
- In the hierarchy browser pane, invoke **RMB → Show Navigation Editbox** to turn on the *Navigation Box.*
  - Search Scopes:
    - Type a wildcard string in the *Navigation Box*, and press **Enter**.
    - The hierarchy browser jumps to the first matched scope, click on **Next Node** icon to jump to the next match.
    - For example, type **CH*** in the *Navigation Box*, **CHILD1**, **CHILD2**, and **CHILD3** will be matched.
  - Filter Scopes:
    - Type a wildcard string in the Node Filter field, only matched scopes will be shown



Traverse Source Code

Find Operations (1/2)

- **Source → Find Signal/Instance/Instport** is a fast way to find signals, instances, or instance ports in full design or selected scope.

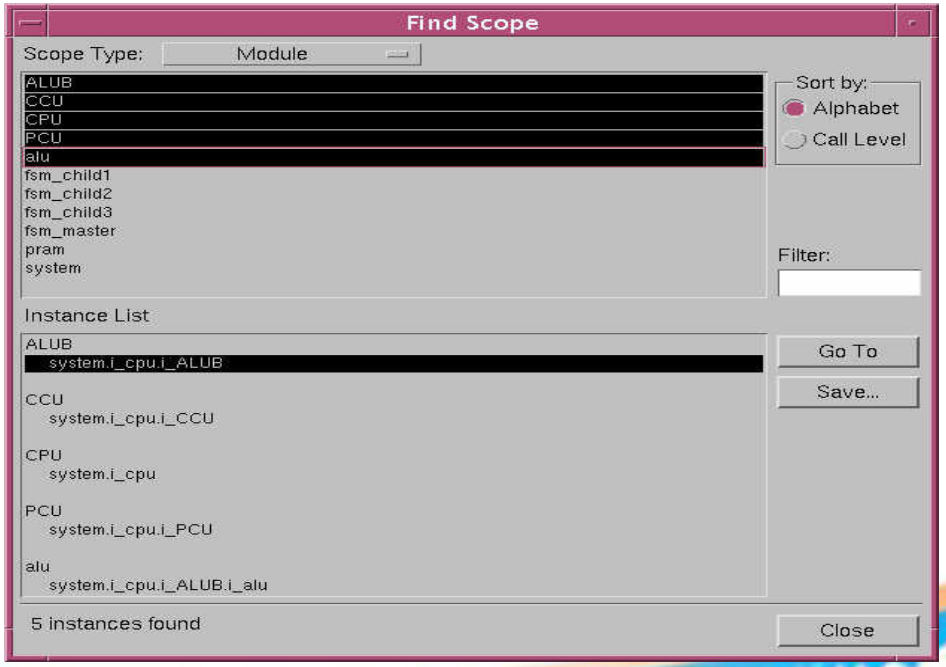


- **Find String** on toolbar quickly locates a string in current scope.

Traverse Source Code

Find Operations (2/2)

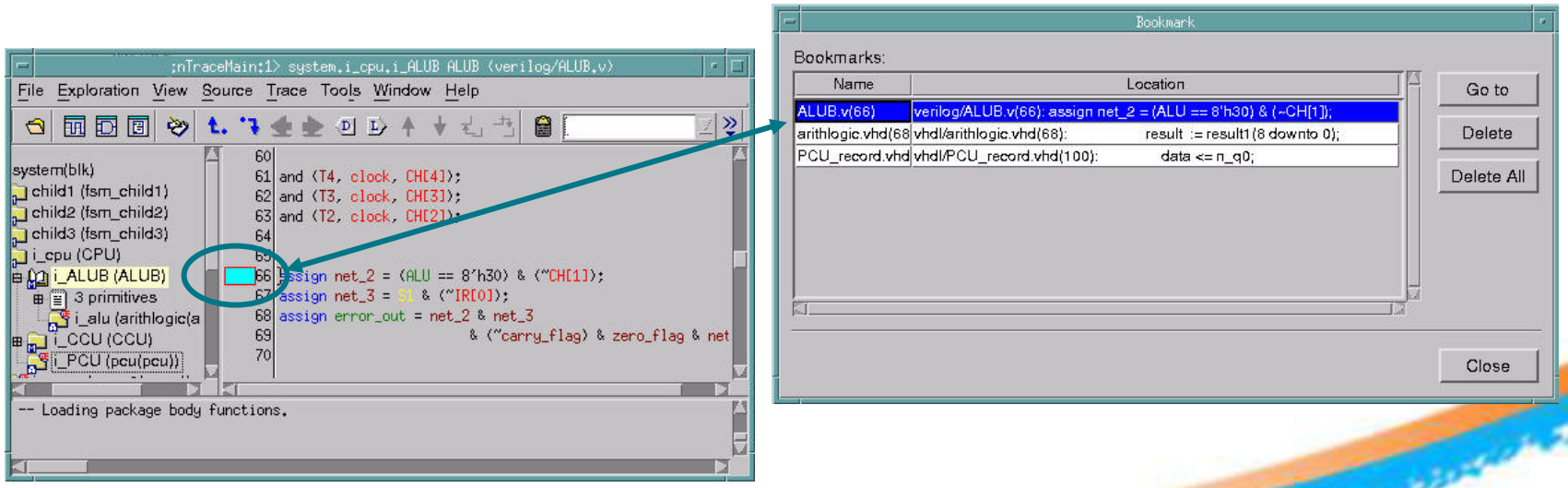
- **Source → Find Scope** is a fast way to directly open a Verilog module or VHDL architecture if the design tree is deep.
  - Select scopes to list related instances.
  - Multiple selection is allowed.
    - Applies to Module/File/Function/Task types in **Scope Type** field.
    - The complete **Instance List** can be saved by clicking the **Save** button.
  - Only single selection is allowed in **Instance List** since it affects the **Go To** button.



Traverse Source Code

Use Bookmarks

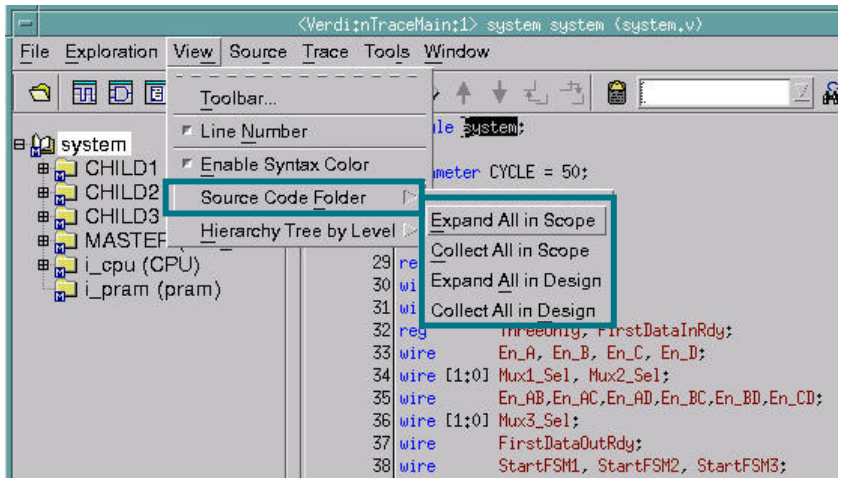
- Add bookmarks:
  - Click on **Bookmark** icon on toolbar 
  - **Ctrl-F2** to toggle a bookmark on/off at the current line.
  - Supports unlimited bookmarks.
- Use **Source → Manage Bookmarks → Edit** command to get list of current bookmarks.
  - Enter strings in the **Name** field to name your bookmarks

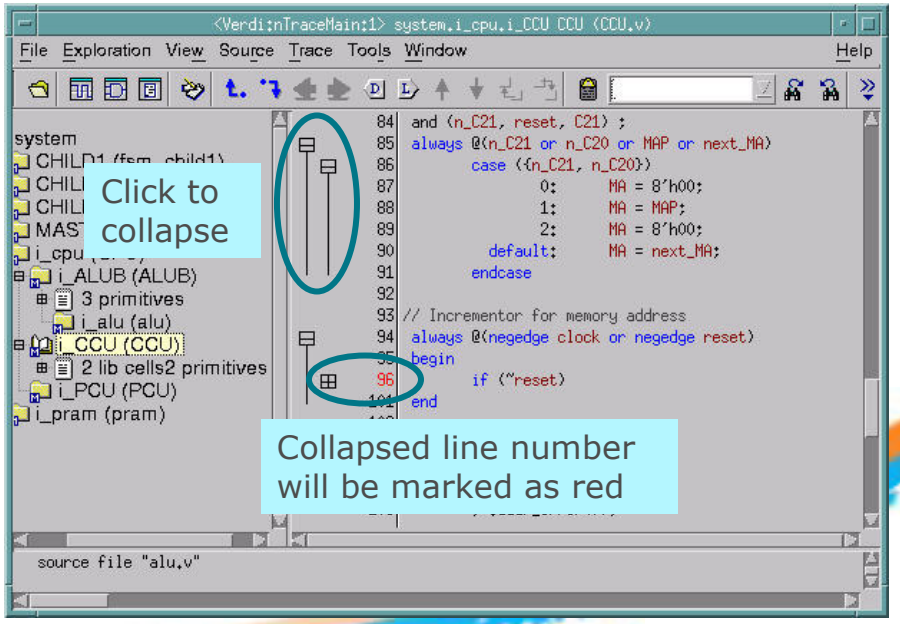


Traverse Source Code

Collapse Source Code

- Collapse source code at user-specified lines.
- Enable **Automatic Source Code Folding** option on the **Source Code → Miscellaneous** page of the *Preferences* form.
  - Specify the number of levels to collapse.
- Use **View → Source Code Folder** or click the mark "+" or "-" to expand or collect the folded source code.

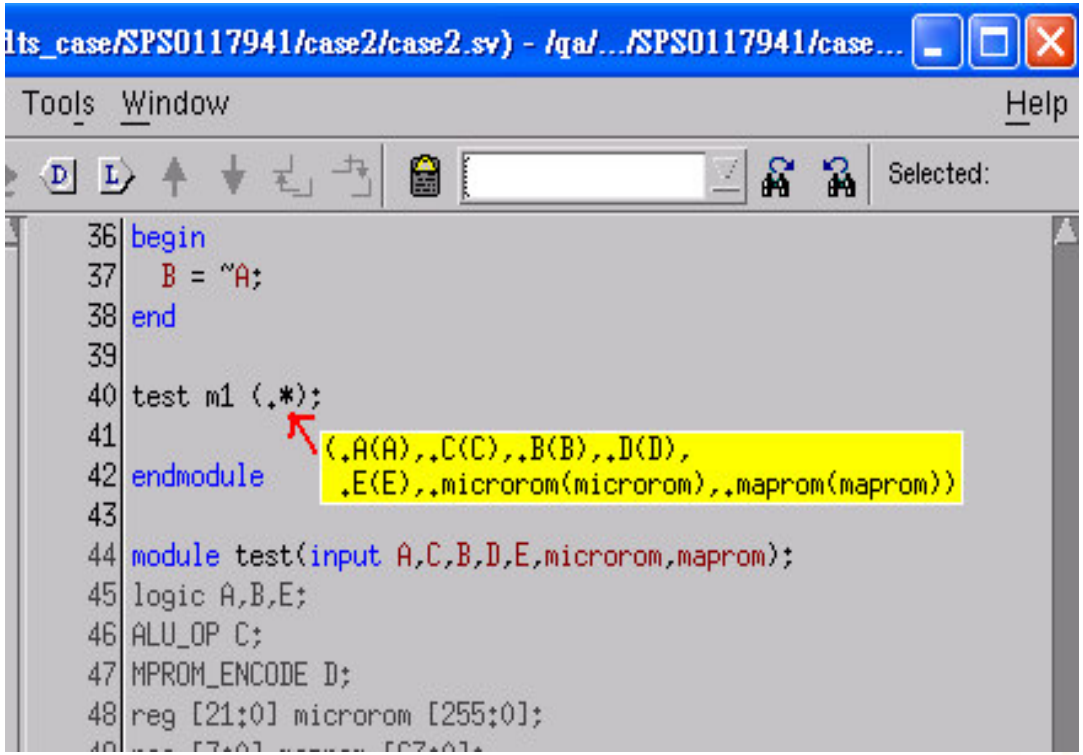




Traverse Source Code

Expand Implicit Port Instantiations in Tip

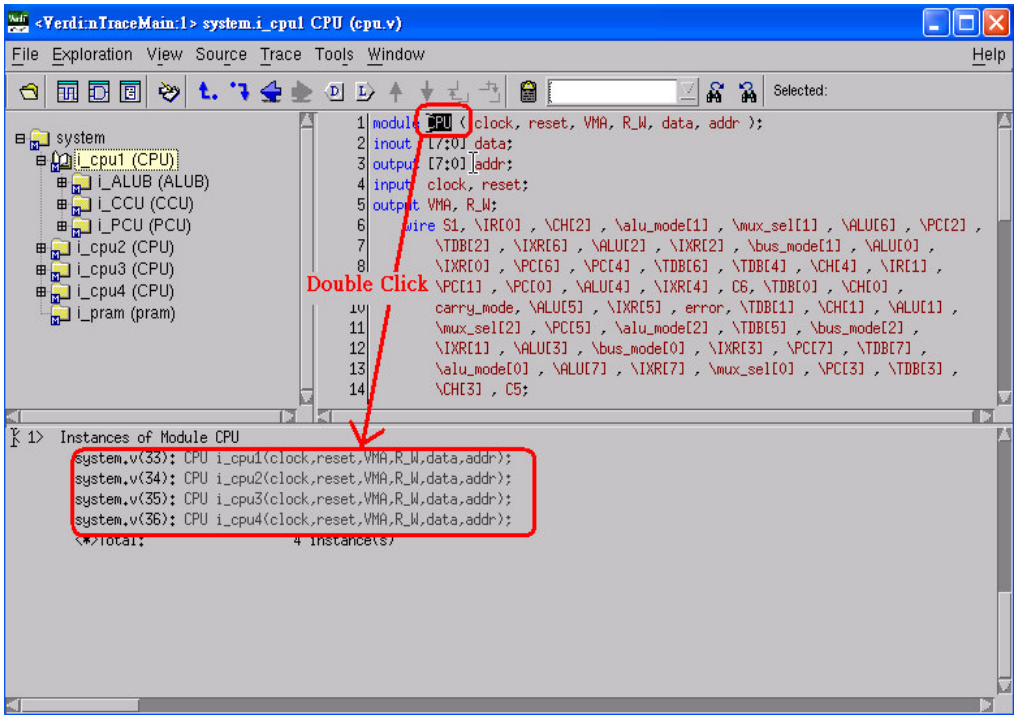
- In the source code pane, put cursor on the implicit port (.*).
  - A tip window opens to show the detailed port instantiations.
  - Supported in SystemVerilog designs.



Traverse Source Code

Double-click a Module Name to List all Instances

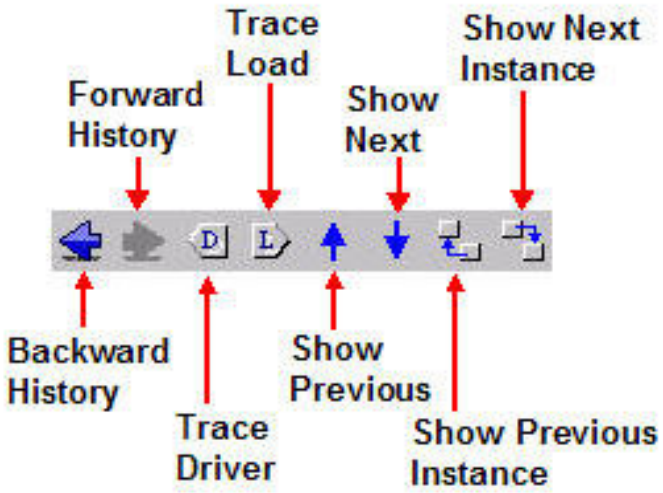
- Double-click on a module name in the source code pane.
  - All instances will be listed in the message pane.
  - Double-click a line in the message pane to jump to the expected instance.



Operate on Signals

Trace Driver and Loads

- To quickly find all the ***drivers*** of a signal:
  - DC a signal.
  - Select signal and click on *Trace Driver* icon.
  - **Trace → Driver** or Right Mouse Button menu.



- To quickly find all the ***loads*** of a signal:
  - Select signal and click on *Trace Load* icon.
  - **Trace → Load** or Right Mouse Button menu.

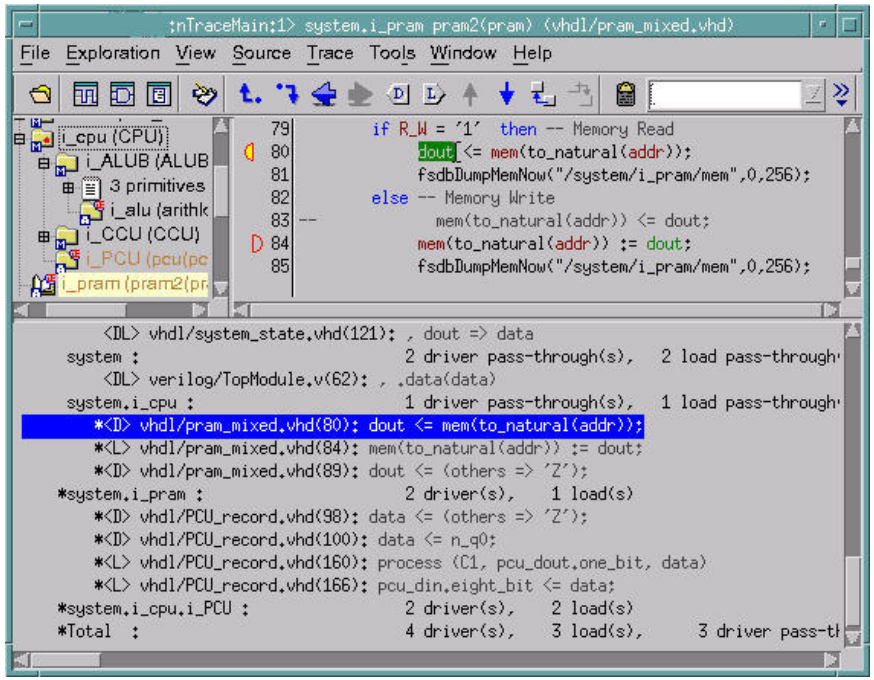
- To quickly find all the ***drivers and loads*** of a signal:
  - **Trace → Connectivity** or Right Mouse Button menu.
  - Dropping a signal in the source window will trace the connectivity of the signal.

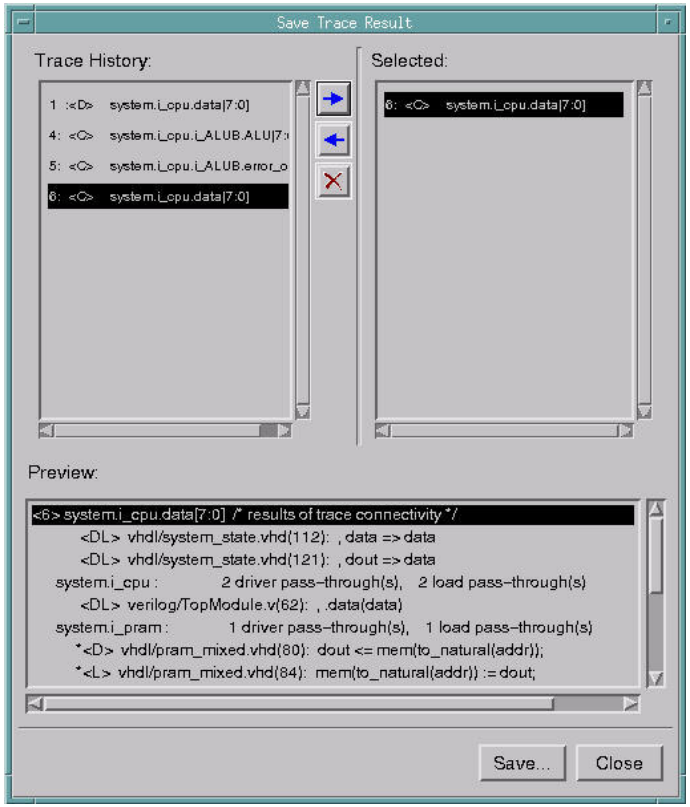
- By default, tracing will cross hierarchical boundaries.
  - Disable **Trace → Trace across Hierarchy** to keep within a hierarchy.

Operate on Signals

Trace Results Report

- **Trace → Save Trace Results** saves Driver/Load/Connectivity in a file for future review.
- Record maximum 32 level trace results.

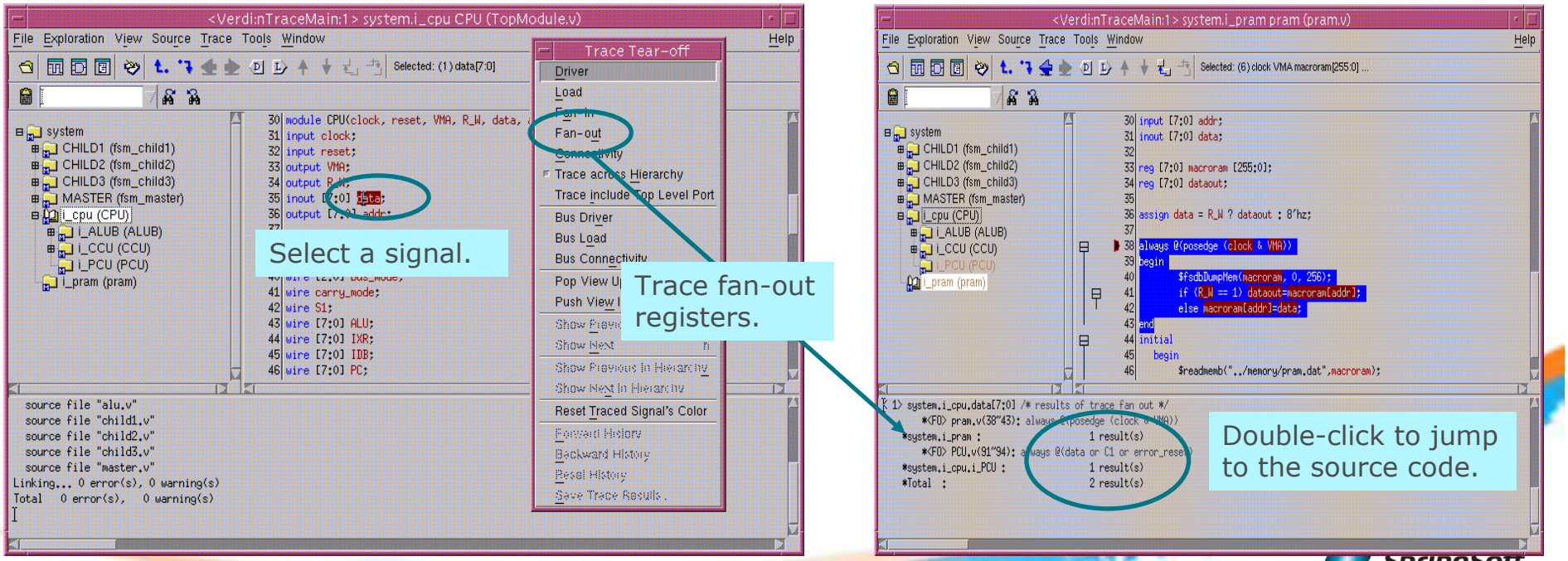




Operate on Signals

Trace Fan-in and Fan-out

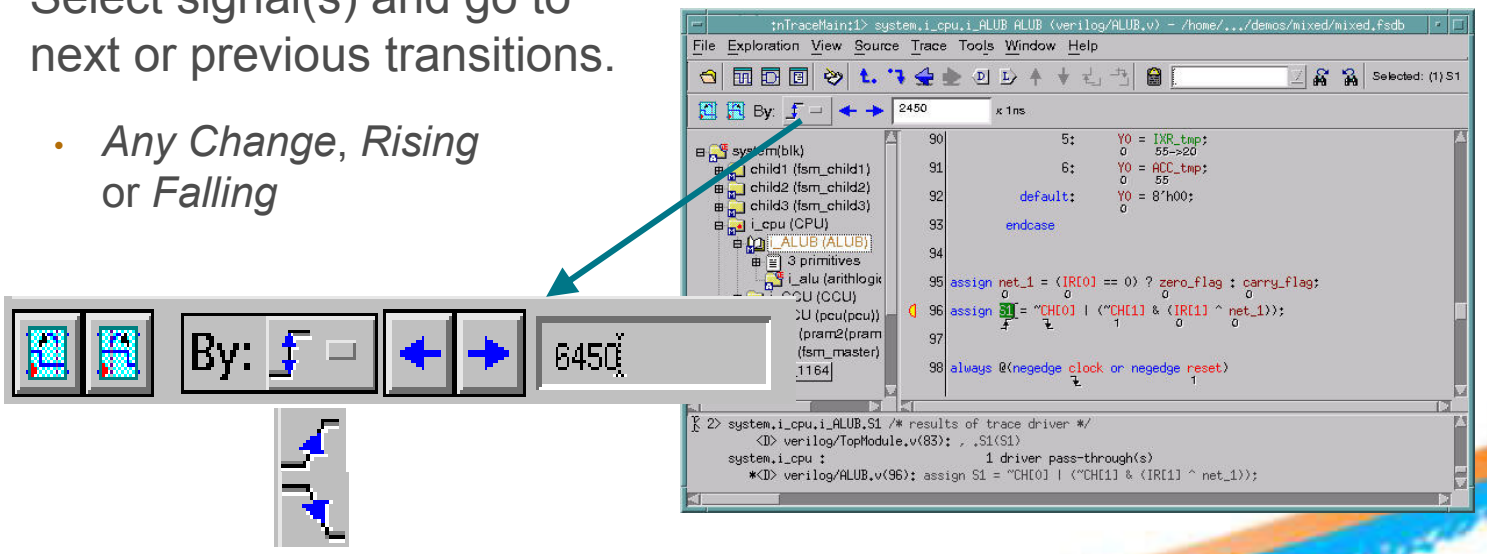
- Select a signal in the source code pane and invoke the command **Trace → Fan-in** or **Trace → Fan-out** command.
  - Traces fan-in or fan-out registers.
  - Jumps to the first trace result in the source code pane.
  - Lists all traced results in the message pane
    - Jump to a different trace result in the source code pane by double-clicking in the message pane.



Operate on Signals

Enable Active Annotation

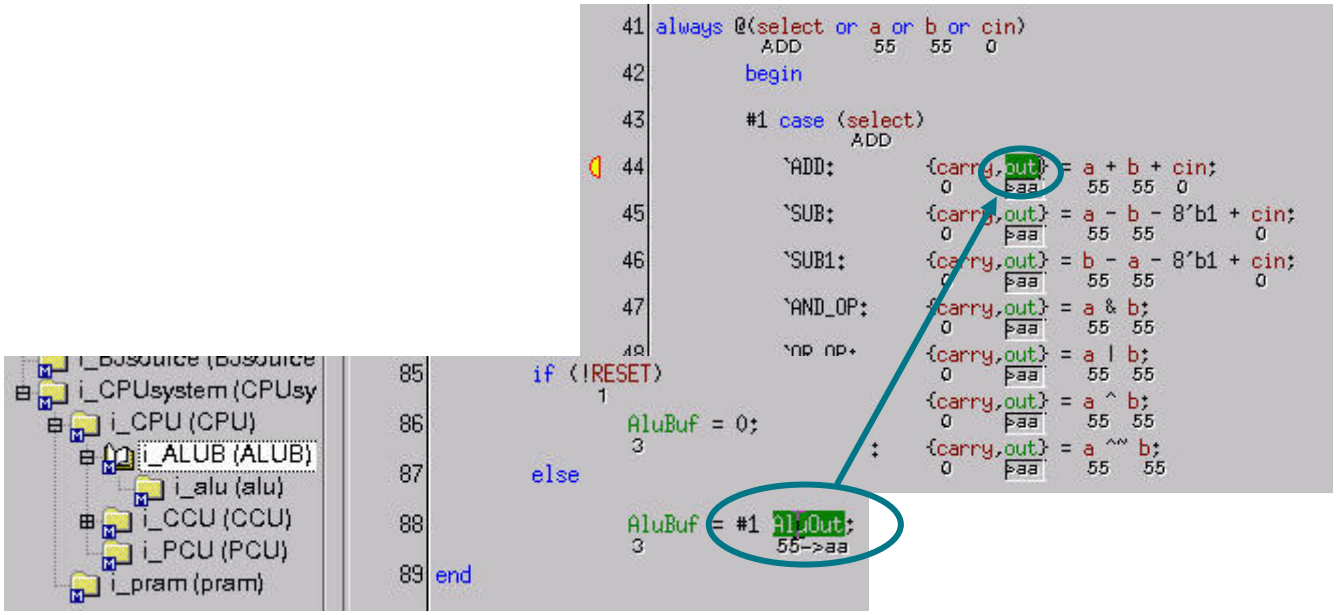
- Use the **File → Load Simulation Results** command to load FSDB file.
- Enable **Source → Active Annotation** (or hot key x) in *nTrace*.
  - Display simulation results on source.
  - Signal values/transitions are synchronized with cursor time.
  - Select signal(s) and go to next or previous transitions.
    - *Any Change*, *Rising* or *Falling*

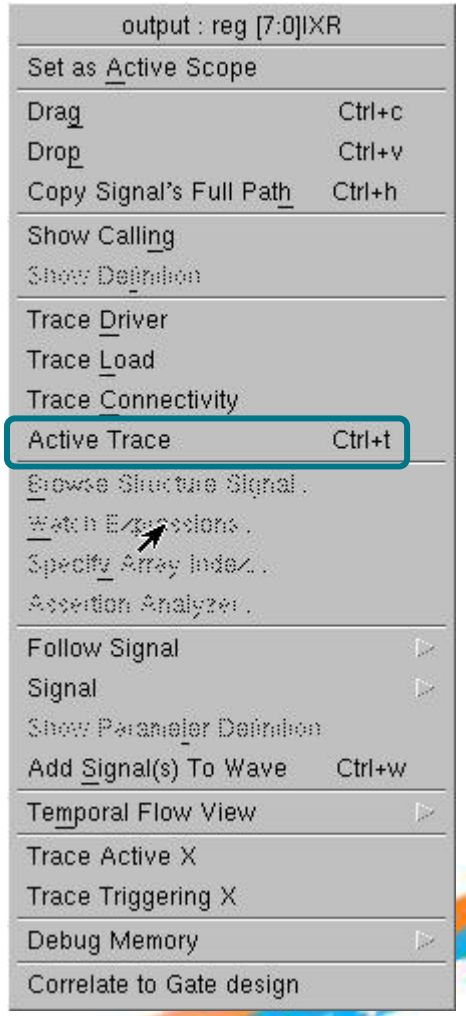


Operate on Signals

Perform Active Trace

- Locate the active driver for a specific signal.
- Select one signal and invoke **RMB → Active Trace** or Ctrl-t.
  - The driving statement at the cursor time will be displayed.

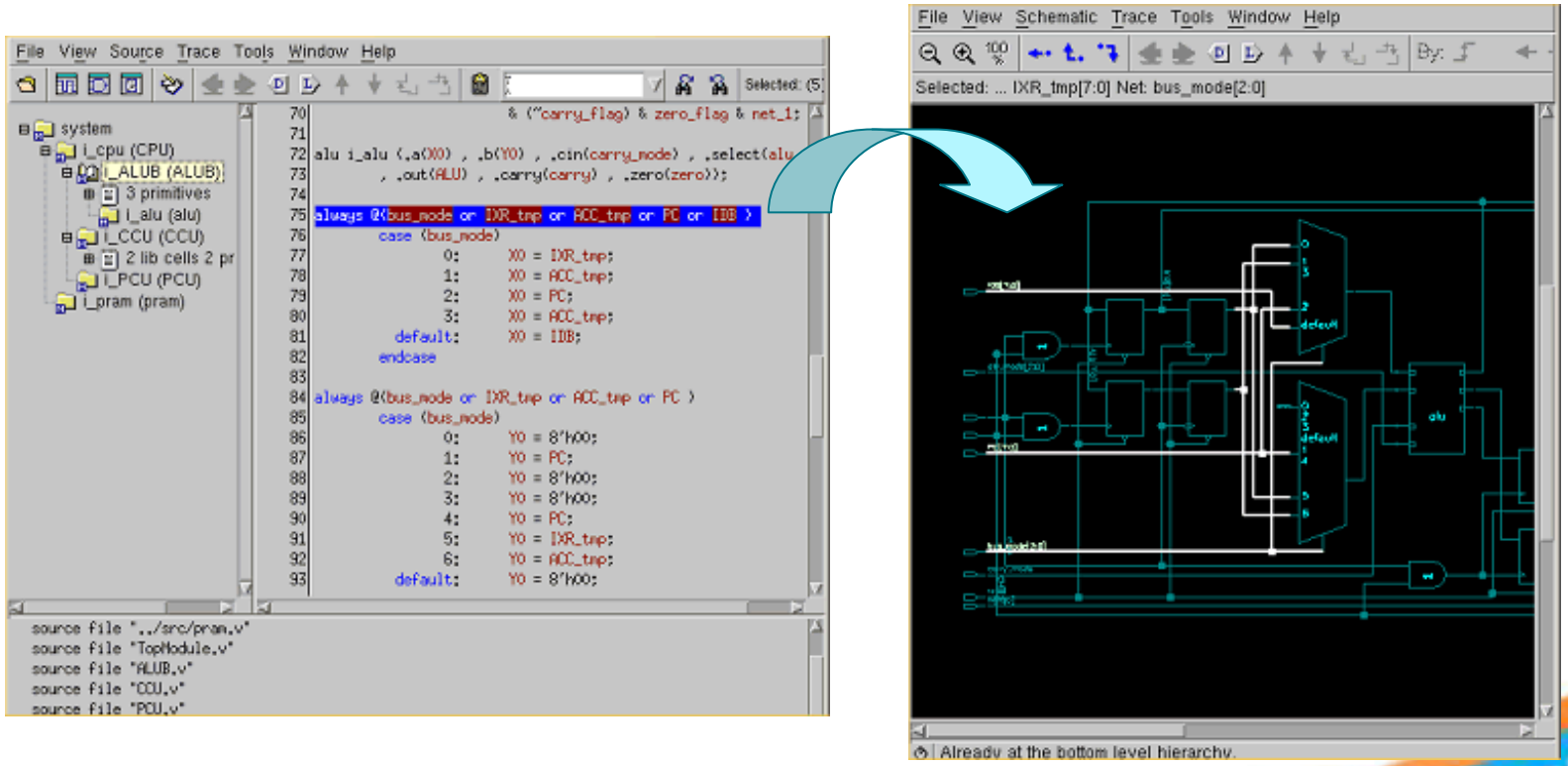




Correlate Other Views

nTrace to nSchema

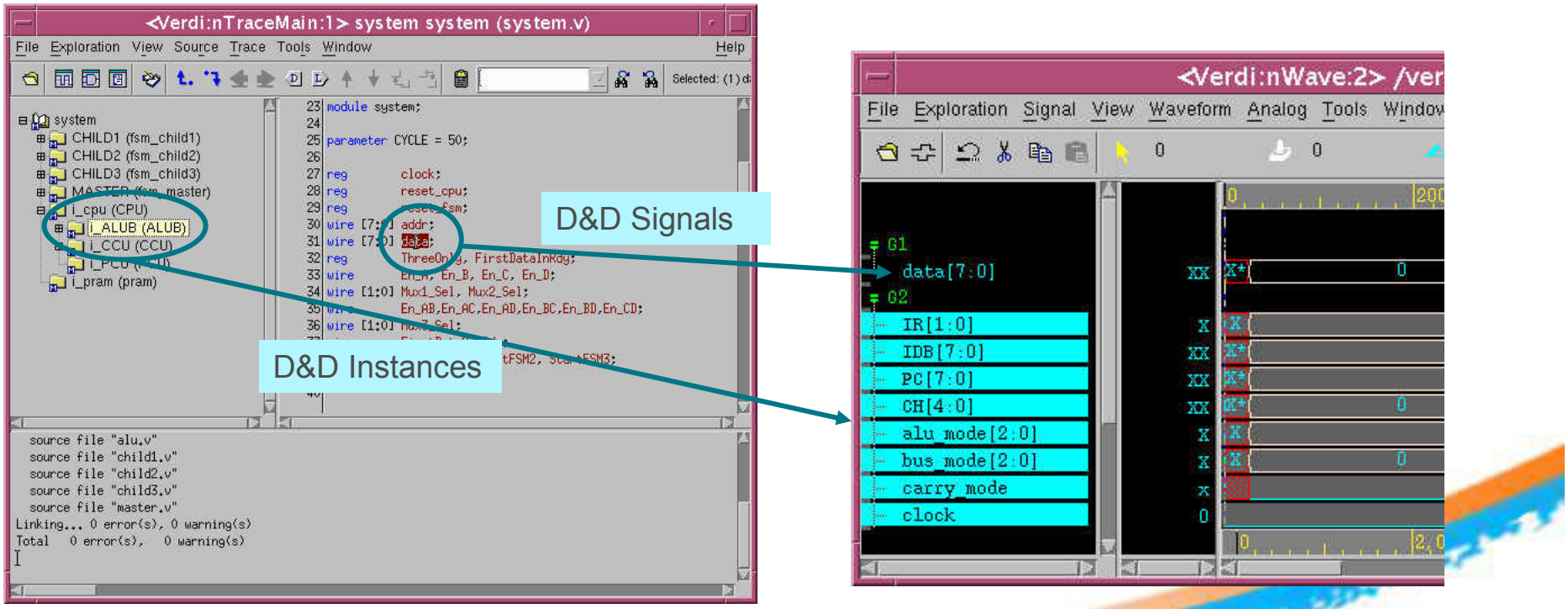
- Select object(s) (e.g. instance, signal, …) in *nTrace.*
- D&D selected object(s) into *nSchema* to highlight it.



Correlate Other Views

nTrace to nWave

- Select object in *nTrace* Source Code Pane or Hierarchy Browser Pane, D&D to *nWave*
  - D&D signals will add signals into *nWave.*
  - D&D instances will add all I/O signals of that instance into *nWave.*



Summary

- In this section, you have learned…
  - How to create *nTrace* window
  - How to search and collapse hierarchy browser
  - How to traverse the source code
  - How to operate on signals
  - How to correlate other views

Lab

- Refer to the *Verdi User's Guide and Tutorial* document in <Verdi_install>/doc/tutorial.pdf.
- Follow the steps in *nTrace Tutorial* chapter to practice the lab.

Topics

- Technology Background
- Set Up the Environment
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM View
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

Objectives

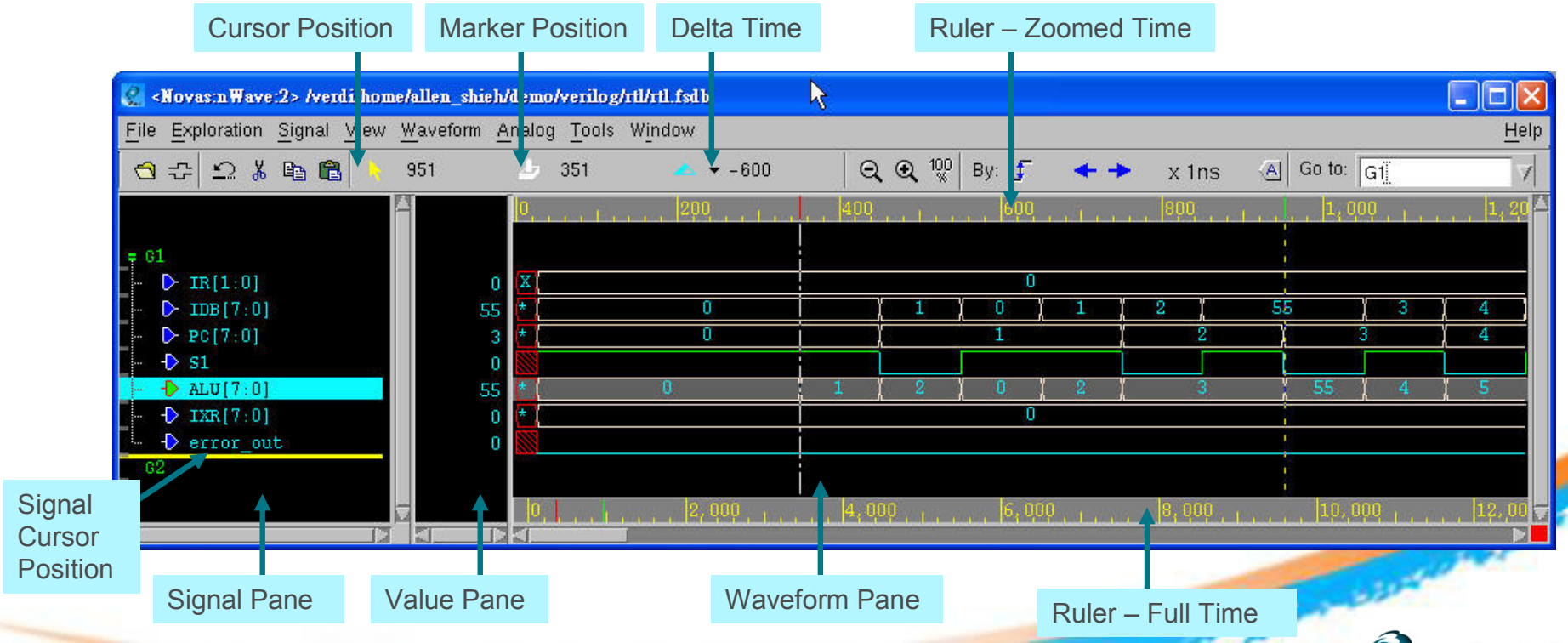
Debug in Waveform View

- After completing this section, you should be able to…
  - Open *nWave* Window
  - Load Simulation Results
  - Add Signals
  - Organize and Operate on Signals
    - Group Operations
    - Bus Operation
    - Add Comments
    - Search for Values
    - Show Driver and Load Signals
    - Save and Restore Signals
  - Manipulate Waveform Display
    - Set Markers
    - Waveform Alias
    - Display Glitches
    - Event Sequences
    - Count Events
    - Display Grid
    - Select Stuck Signals
    - Sort Signals by Transition

Open *nWave* Window

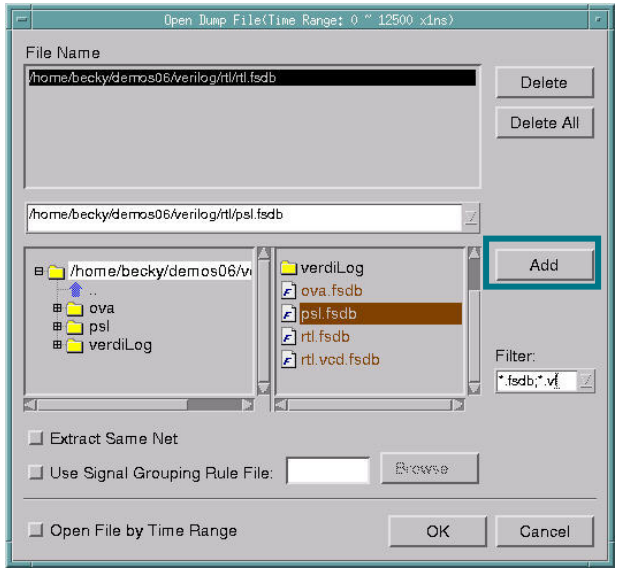
- In *nTrace*, invoke **Tools → New Waveform** or click on the **New Waveform** icon 
- On the Verdi command line, use the option **-ssf <filename>.fsdb**

  % verdi –f run.f –ssf gate.fsdb



Load Simulation Results

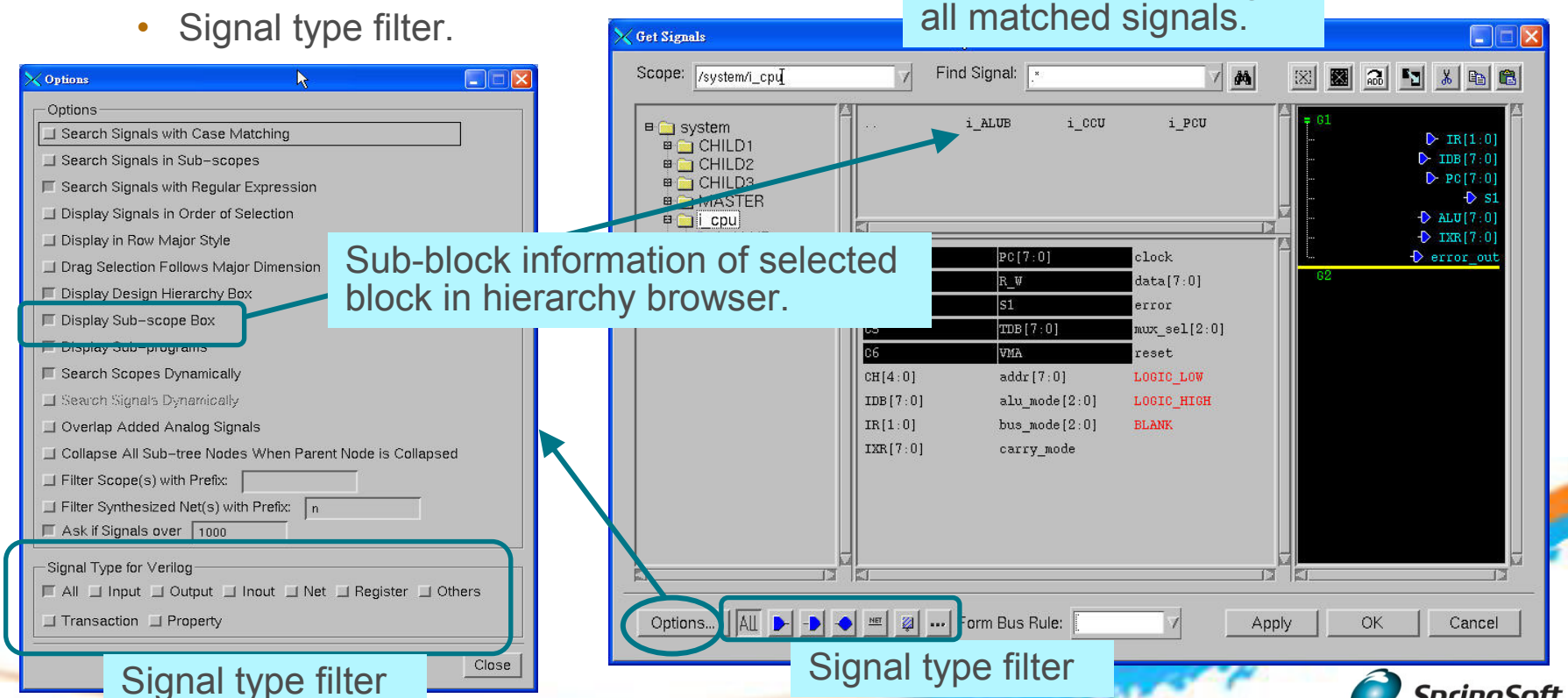
- In *nWave*, use **File → Open** or toolbar icon  to load results.
  - Load VCD files generated by *$dumpvars*.
    - VCD files are automatically converted to FSDB at load time.
    - gzipped VCD files are supported.
      - Supports file naming rules as “xxx.*vcd.gz*” or “xxx.*dump.gz*” only.
  - Load FSDB Files generated by *$fsdbdumpvars*.
  - Select one or more files to **Add** and open in the same waveform window.
    - Use **File → Set Active** to specify which file is active.
  - Open several files in different *nWave* windows.
    - Use **Window → Change to Primary** to specify which window is active.



Add Signals

Get Signals from nWave

- Use **Signal → Get Signals** or toolbar icon 
  - Directly form bus based on user-defined rules.
  - Option to find signals in full scope.
  - Option to display signals in order of selection not alphabetical.
  - Signal type filter.



Add Signals

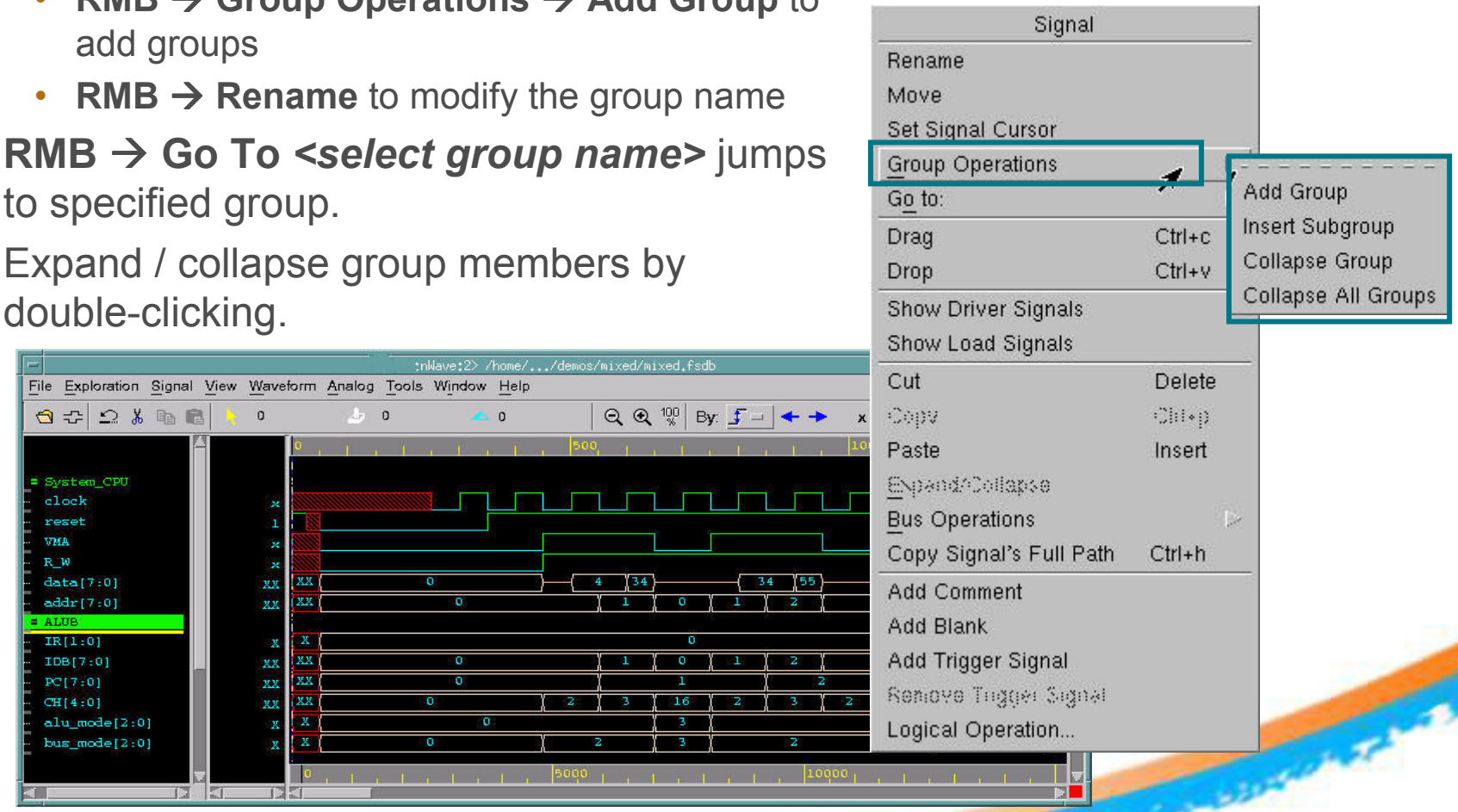
Add Signals from Other Windows

- D&D signals/instances from other windows to *nWave*.
  - All instance I/O.
  - Selected signal(s).
  - Signals in text region.
  - Schematic hierarchical blocks or logic gates
- Select signals and use **RMB → Add Signal(s) To Wave** in *nTrace*.
- Select objects and use **RMB → Add Select Set To Wave** in *nSchema*.
- After executing a trace command in *nSchema*, use **Trace → Add Result to Wave**.

Organize Signals

Create Group for Signals

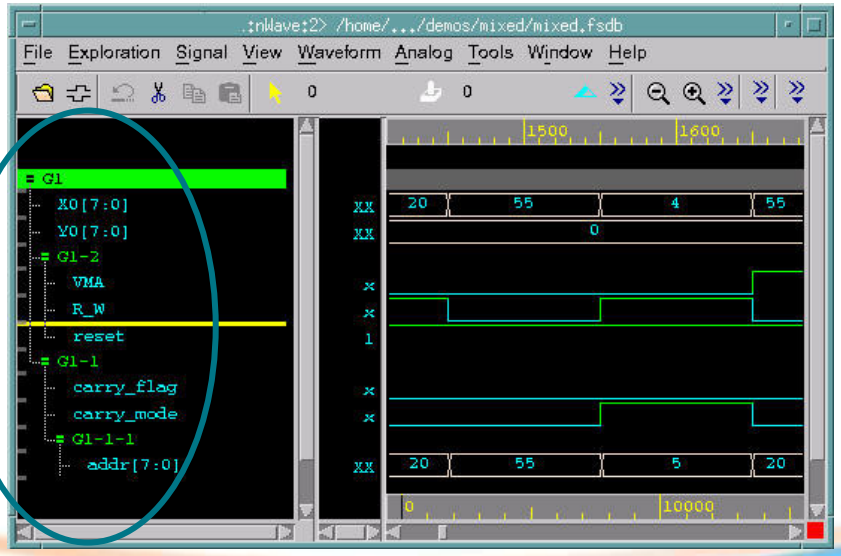
- Select a group in signal window :
  - **RMB → Group Operations → Add Group** to add groups
  - **RMB → Rename** to modify the group name
- **RMB → Go To <*select group name*>** jumps to specified group.
- Expand / collapse group members by double-clicking.

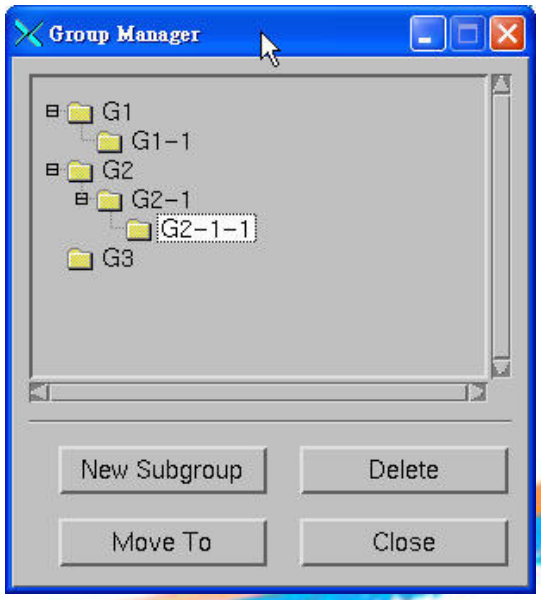


Organize Signals

Group Operations

- Invoke **RMB → Group Operations → Insert Subgroup** to create sub-groups of selected groups
- Each group/sub-group can be renamed with **RMB → Rename**
- Invoke **View → Group Manager** to open the *Group Manager* form
  - Select a group in *Group Manager* form and click **New Subgroup** to add a new group under the selected group.
  - Click the **Move To** button to move the selected group/signals in *nWave* into the selected group in the *Group Manager*
  - Delete the selected group and subgroups

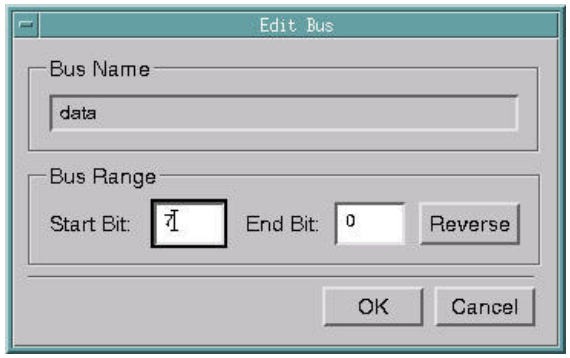


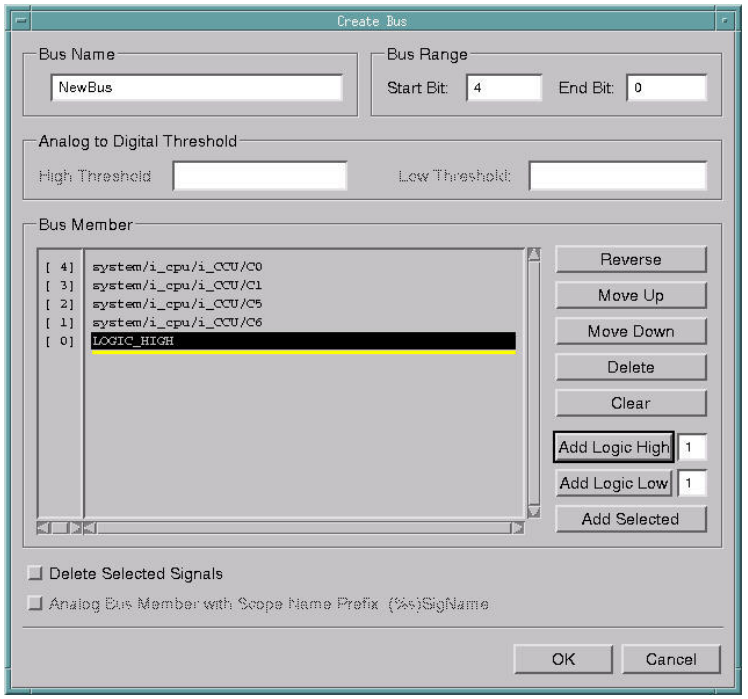


Operate on Signals

Create and Edit Buses

- Invoke **Signal → Edit Bus** or **RMB → Bus Operations → Edit Bus** in signal pane to edit selected bus.



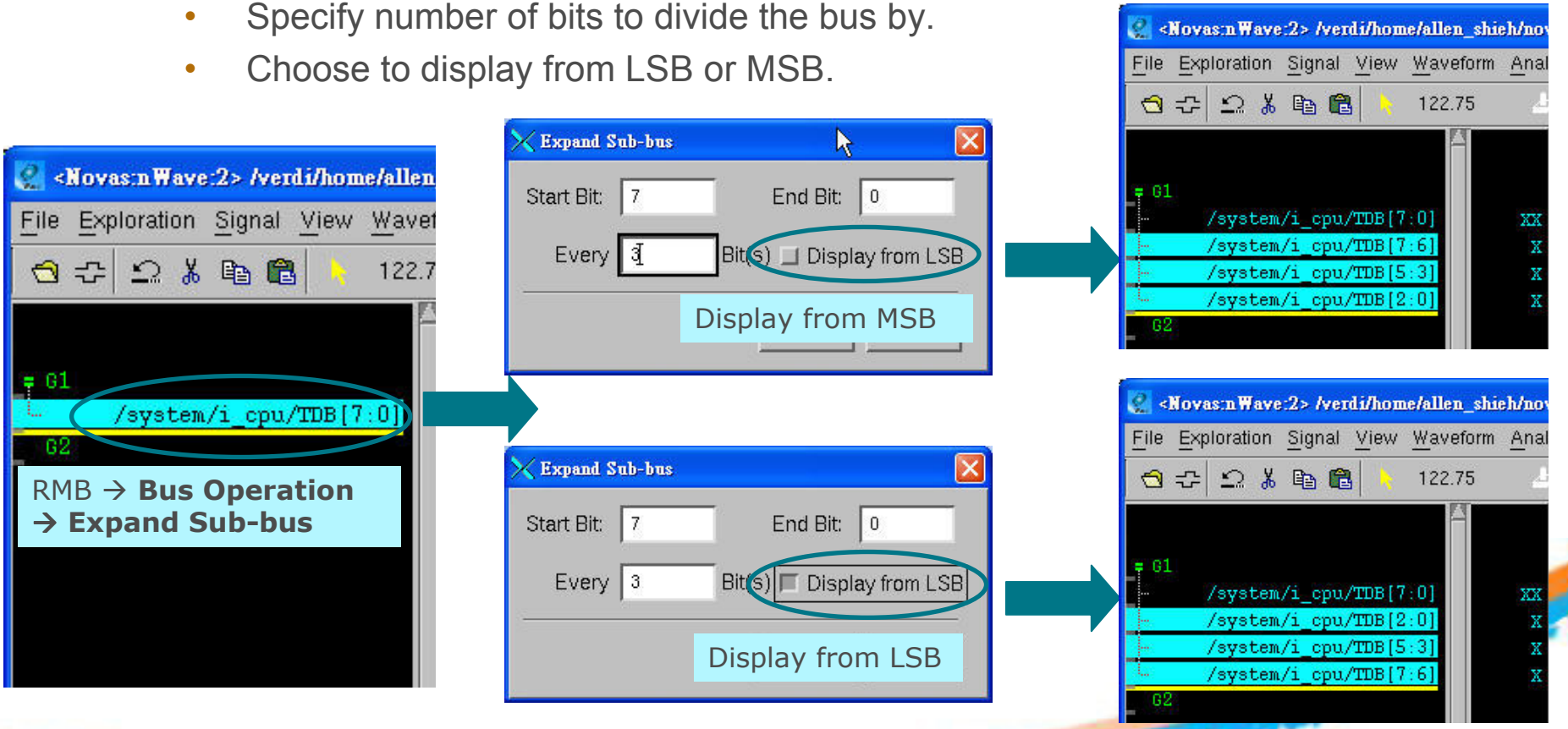


- Create bus from selected signals.
  - Invoke **Signal → Create Bus** or **RMB → Bus Operations → Create Bus** in signal pane.
  - Add Logic 0 or Logic 1 as place holders.

Operate on Signals

Expand Buses

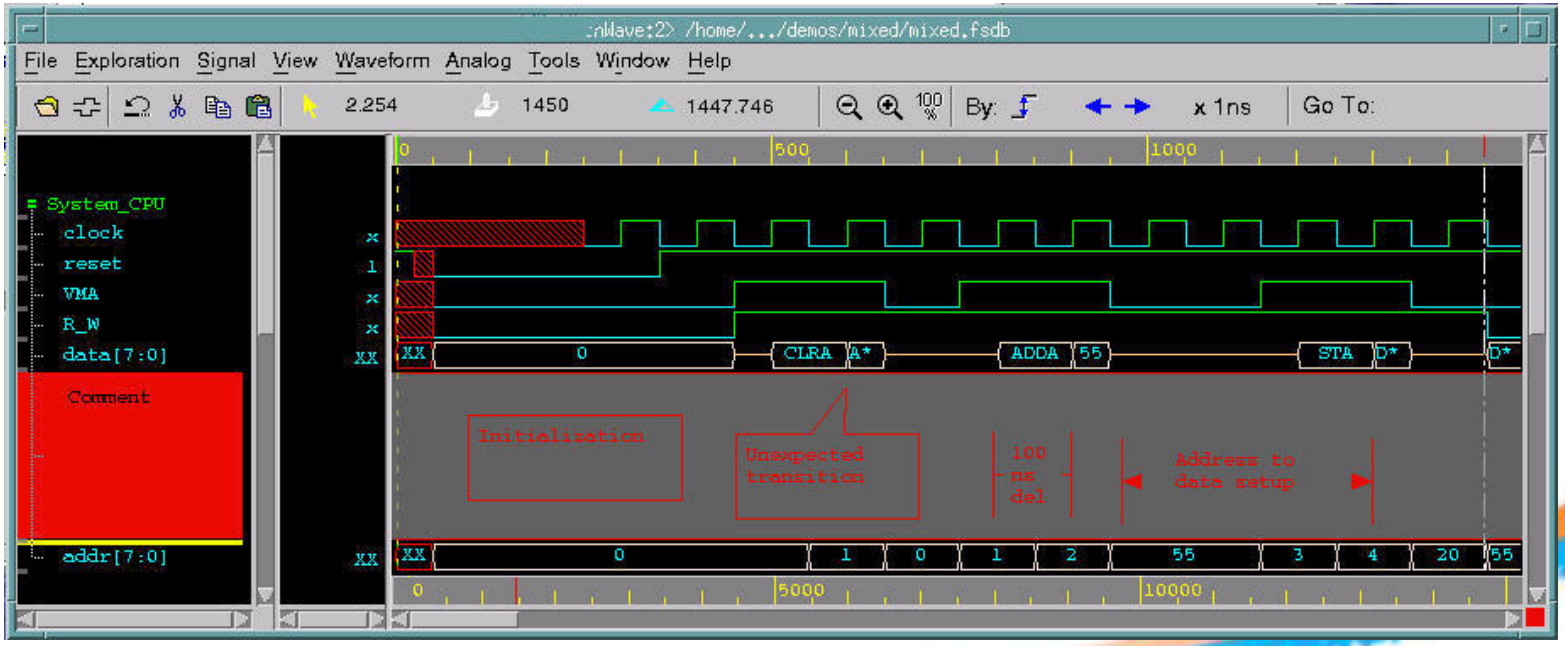
- Expand / Collapse bus by double-clicking.
- Select a bus in the *nWave* signal pane and invoke RMB → **Bus Operation → Expand Sub-bus** to open *Expand Bus* form.
  - Specify number of bits to divide the bus by.
  - Choose to display from LSB or MSB.



Operate on Signals

Add Comments to Signals

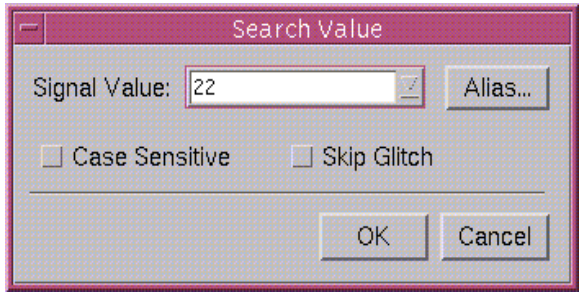
- Use **Signal → Comment → Insert** to insert a comment field at cursor position.
- Use **Signal → Comment → Add Square Box** or other sub-commands to add different types of comment boxes
  - Four comment box types are provided: **Square Box**, **Attached Square Box**, **Period Box** and **Arrow Period Box**.

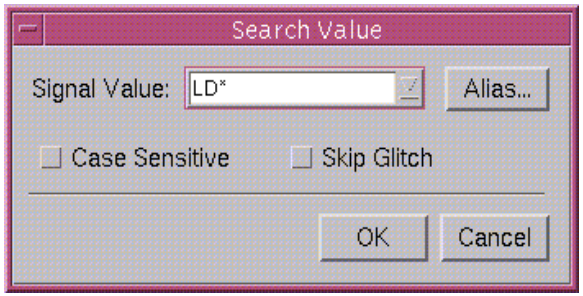


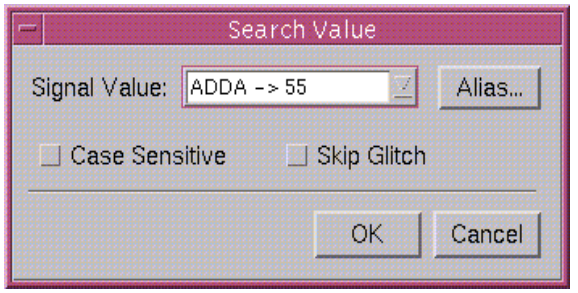
Operate on Signals

Search for Values

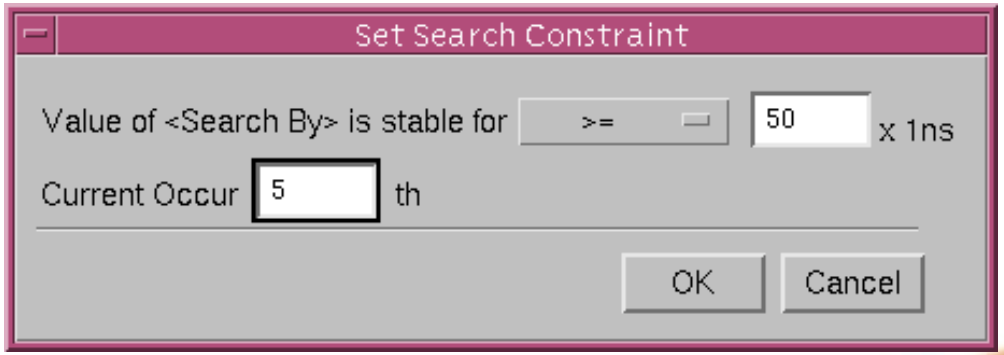
- Use **Waveform → Set Search Value** to search signal values.
  - Search value accepts mnemonics, wild cards and transitions







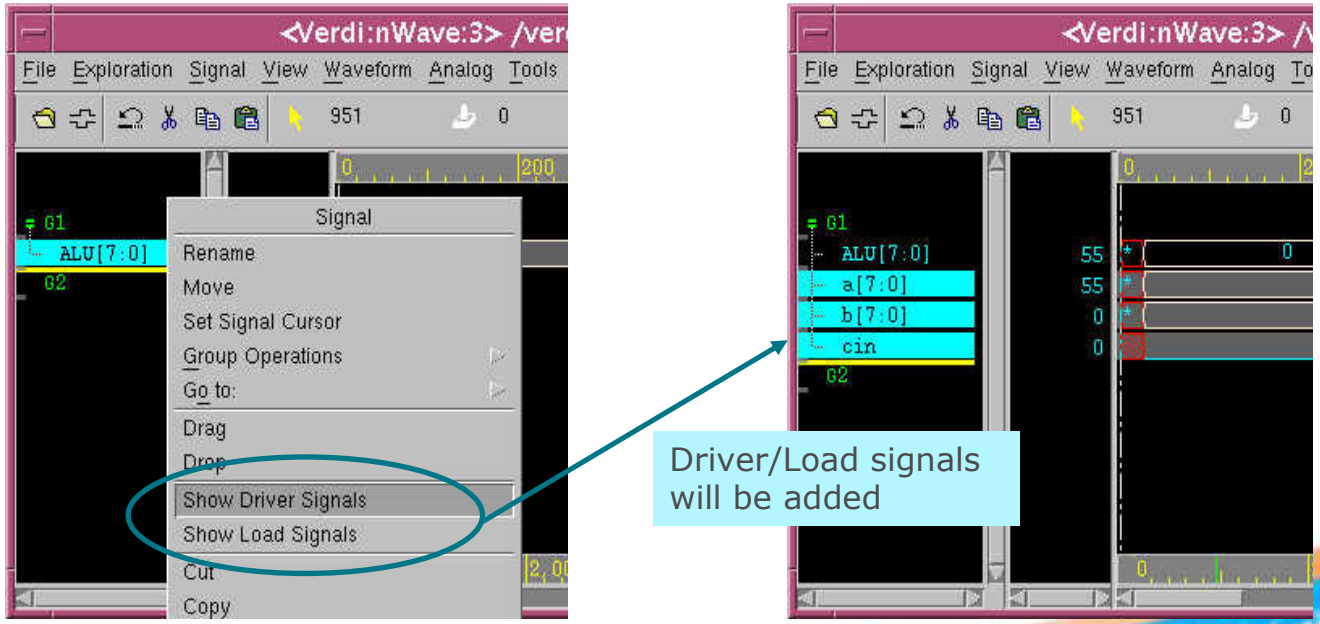
- Use **Waveform → Set Search Constraint** to constrain search to a duration of time (set to <= 0 for glitches) or nth occurrence of a condition.



Operate on Signals

Show Driver and Load Signals

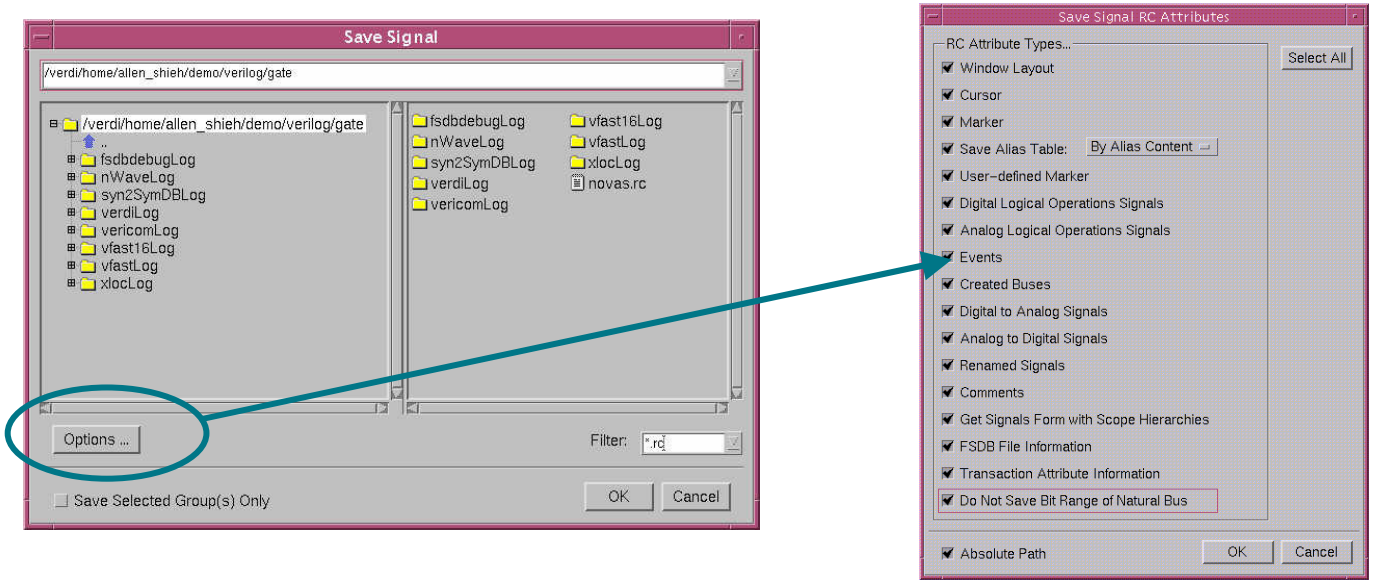
- Select a signal in *nWave* signal pane, invoke **RMB → Show Driver Signals** or **Show Load Signals** commands to trace driver or load.
  - Driver/load signals will be added in signal pane.
  - All pass through signals will be ignored.
  - The command is only valid when the design is also imported – not valid in standalone *nWave.*



Operate on Signals

Save and Restore Signals

- Use **File → Save Signal** to save all signals to a loadable file
  - Click **Option** button to specify signals attributes
  - Note: turn on **Do Not Save Bit Range of Natural Bus** option when debugging parameterized designs (with the same signal name but different bus range)



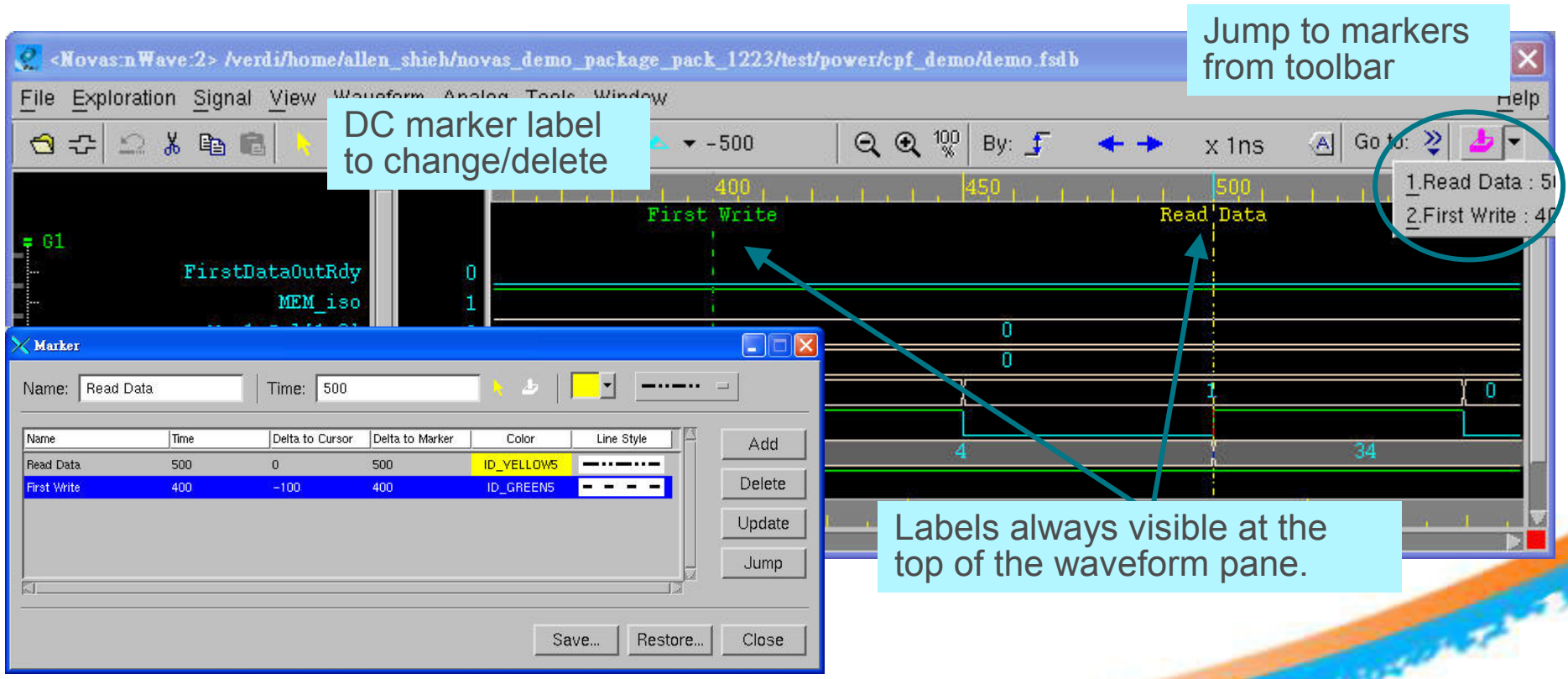
- In *nWave*, use **File → Restore Signal** to load the saved file
- On the command line, use **–sswr <file>.rc** to restore.

NOTE: If there is a FSDB file open in *nWave,* the **Restore Signal** command will not overwrite the loaded simulation results.

Manipulate Waveform Display

Set Markers

- Use **Waveform → Marker** to name and place markers.
- Multiple markers supported, markers can be saved and restored.
- Specify name, color, and line style for labels to differentiate markers.



Manipulate Waveform Display

Waveform Alias (1/3)

- Use **–autoalias** option on Verdi command line for automatic mnemonic recognition for ‘defines and parameters.

  % verdi -f run.f –autoalias

- Use **aliasextract** to extract an alias file from a compiled library. Default is extracted.alias.

  % vericom –f run.f –lib work

  % aliasextract -lib work

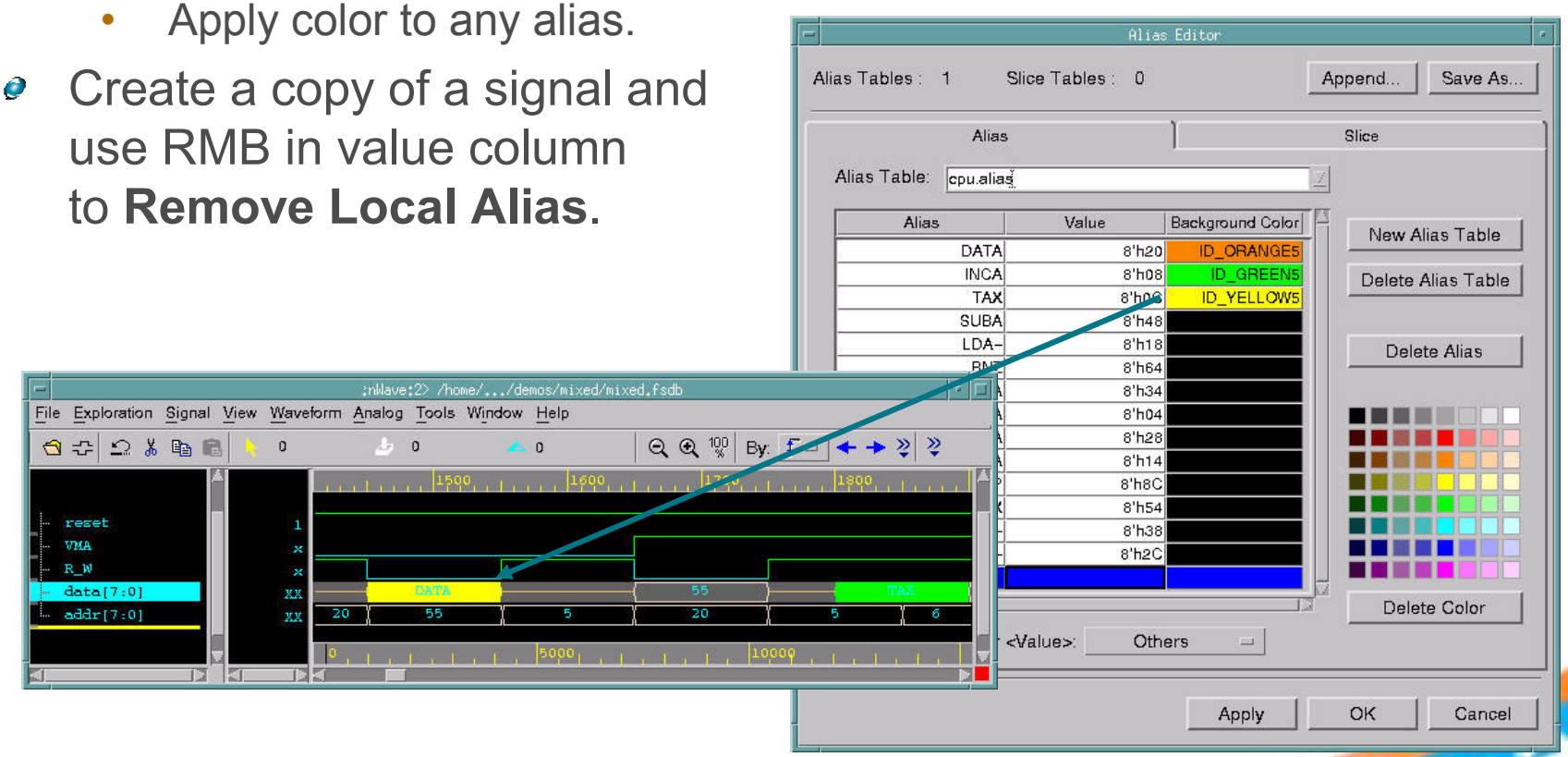
- Use **–aliasFile** option on Verdi command line to load an alias file.

  % verdi -top system -aliasFile extracted.alias

Manipulate Waveform Display

Waveform Alias (2/3)

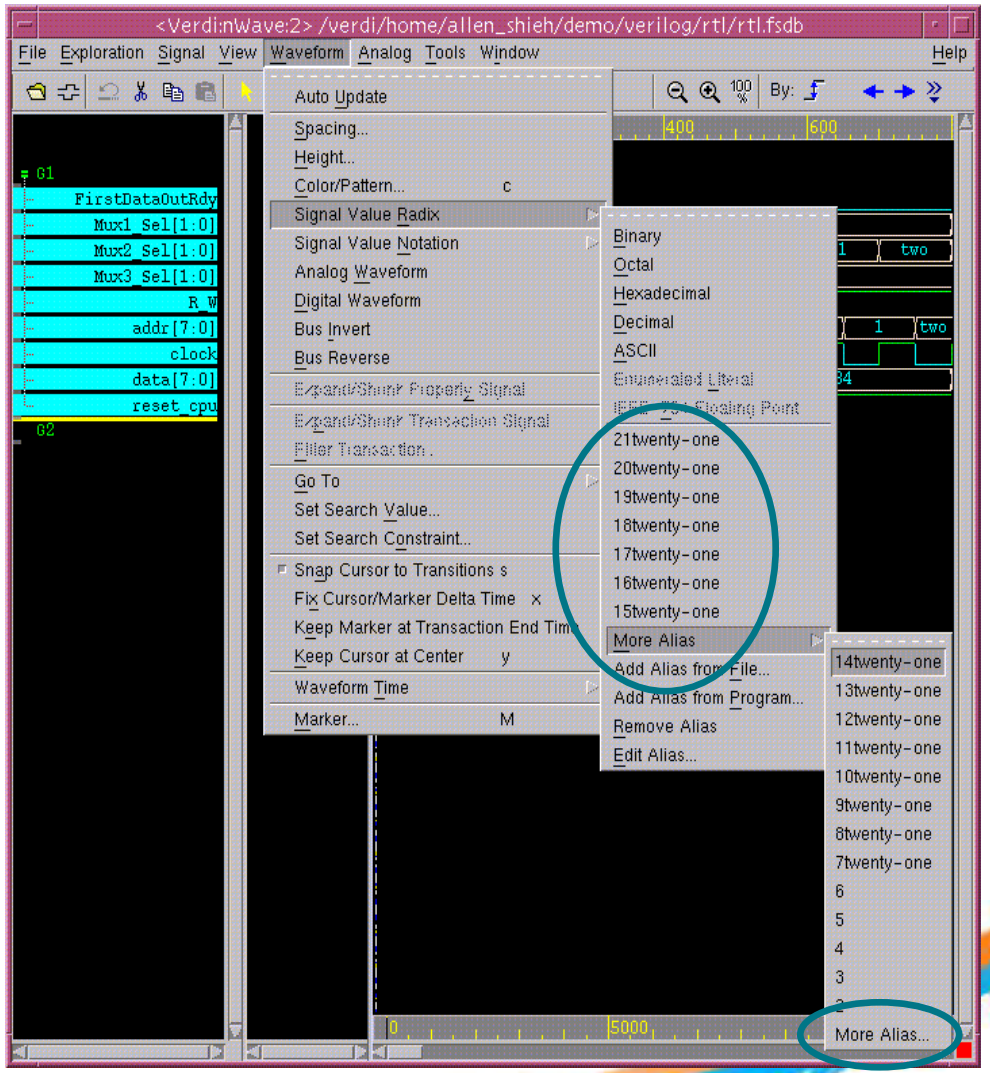
- Use **Waveform → Signal Value Radix → Edit Alias** to create/update an alias file and apply to a signal.
  - Apply color to any alias.
- Create a copy of a signal and use RMB in value column to **Remove Local Alias**.



Manipulate Waveform Display

Waveform Alias (3/3)

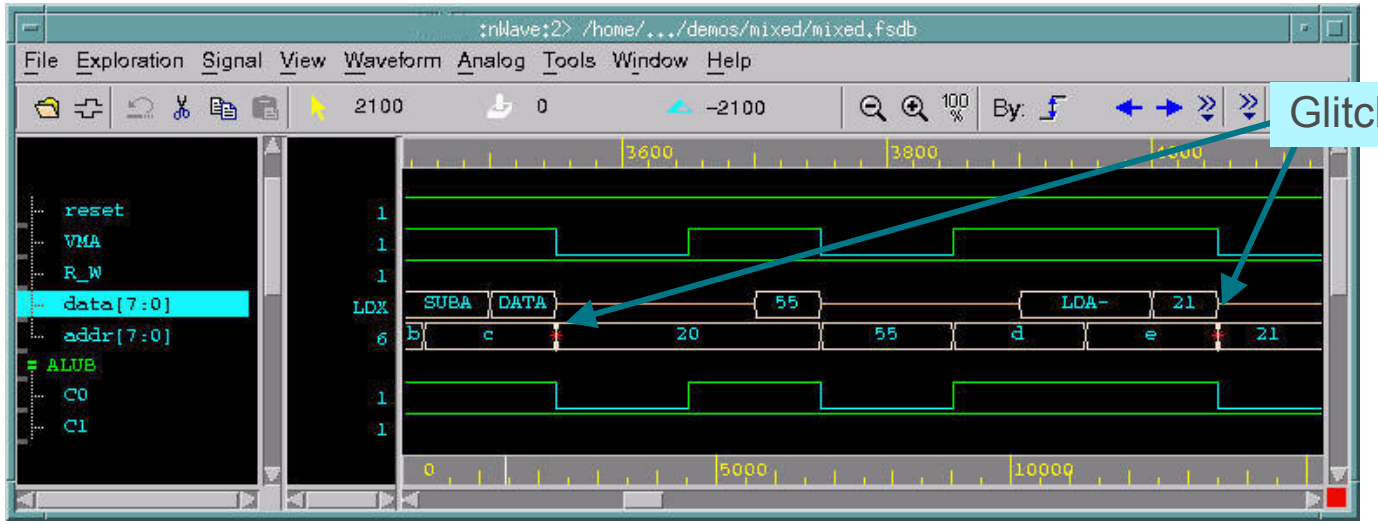
- Supports up to 20 aliases in a list.
  - Select **More Alias** in the first level to see the second level aliases
  - Select **More Alias** in the second level will bring up ***Alias Editor.***
    - No number limitation in this form.



Manipulate Waveform Display

Display Glitches

- Use **View → Display Glitch** to identify glitch location.



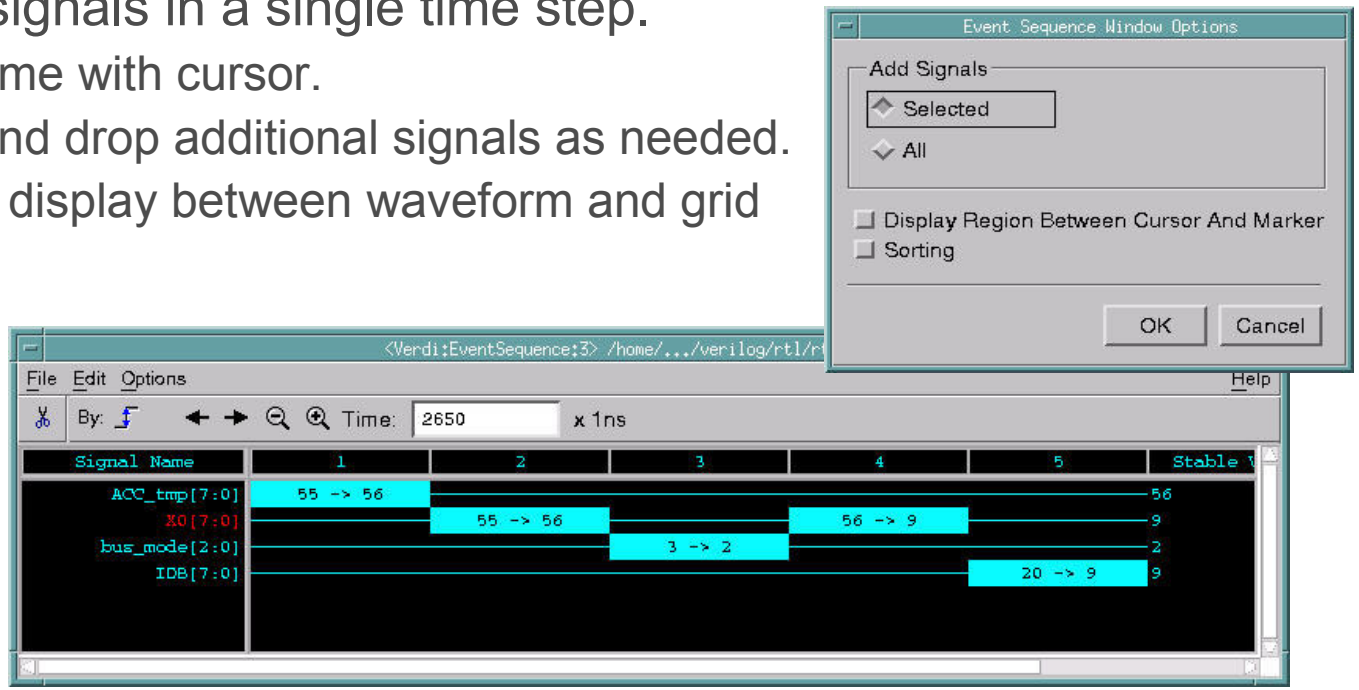
- Use **Waveform → Set Search Constraint** to search for glitch events.
  - Set **Value of <Search By> is stable for** to **<=** and enter **0** in the time field.

NOTE: Simulation results file must be generated with the following environment variable set::
setenv NOVAS_FSDB_ENV_MAX_GLITCH_NUM 0

Manipulate Waveform Display

Event Sequence

- Use **Tools → Event Sequence** to display order of events on selected signals in a single time step.
  - Sync time with cursor.
  - Drag and drop additional signals as needed.
  - Switch display between waveform and grid mode.

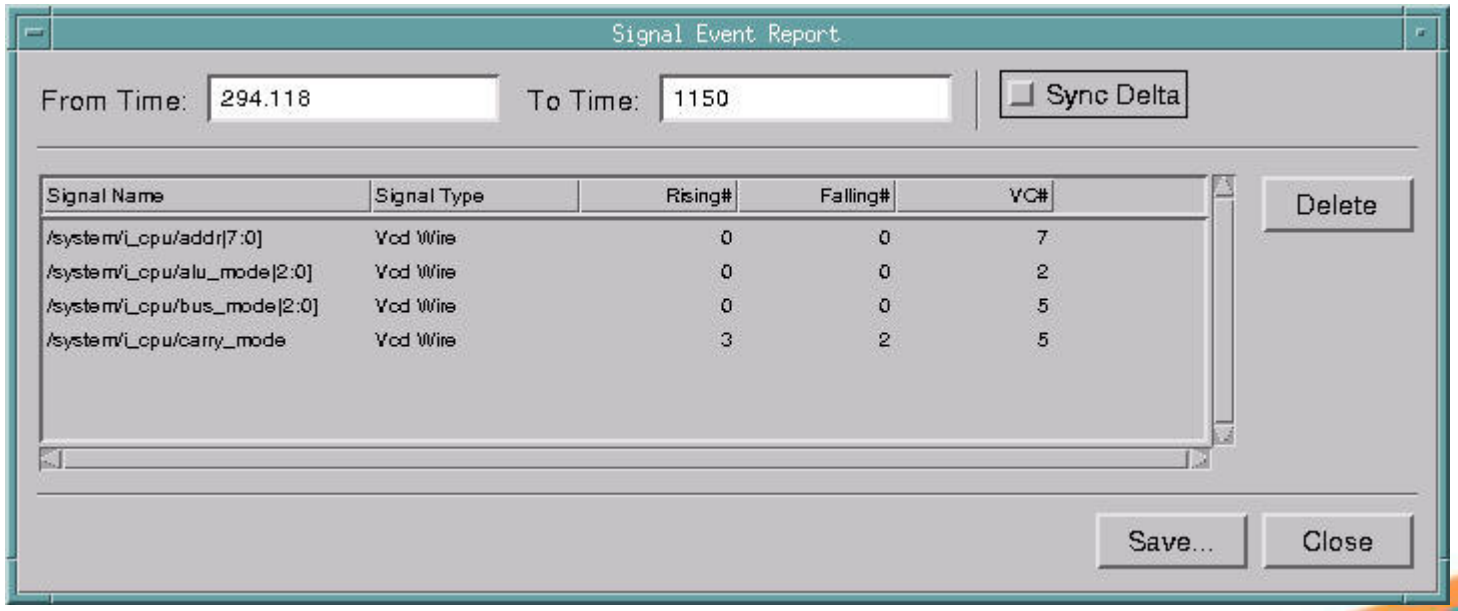


NOTE: Simulation results file must be generated with the following environment variable set:
setenv NOVAS_FSDB_ENV_DUMP_SEQ_NUM on

Manipulate Waveform Display

Count Events

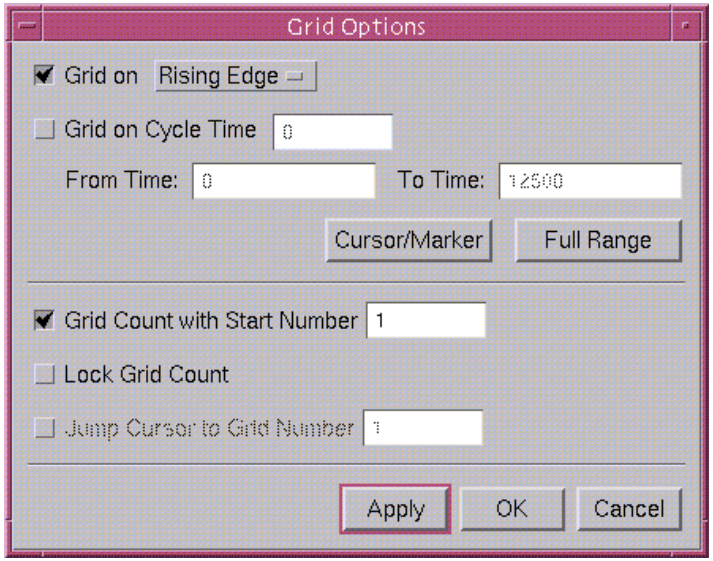
- To get the statistics of event status during one time period.
  - Total number of Rising / Falling / Value changes
- In *nWave*, select signals and invoke **View → Signal Event Report**.
  - Specify the time range or enable the **Sync Delta** option.

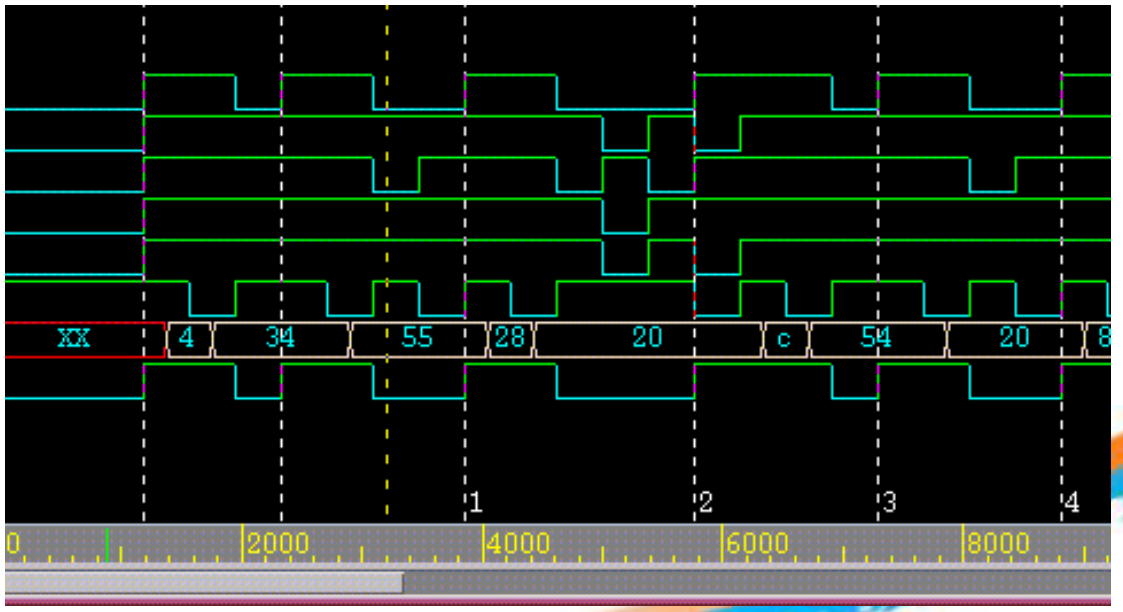


Manipulate Waveform Display

Display Grid

- In *nWave*, use **View → Grid Options** to open the *Grid Options* form.
  - Add grids based on the rising/falling edges of the selected signal.
  - Add grids based on user defined cycle in a specific time range.
  - Count number for grids.
    - Can specify starting number.
    - Counting will start on the first grid after the current cursor time.

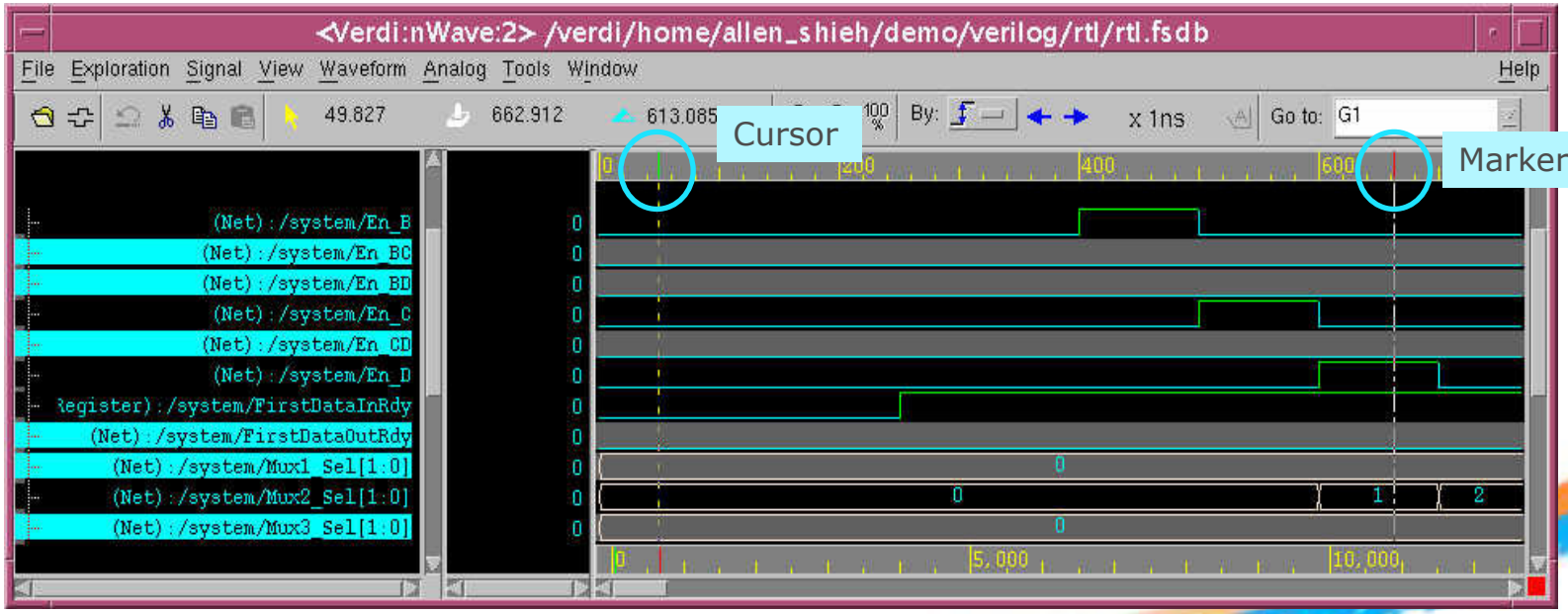




Manipulate Waveform Display

Select Stuck Signals

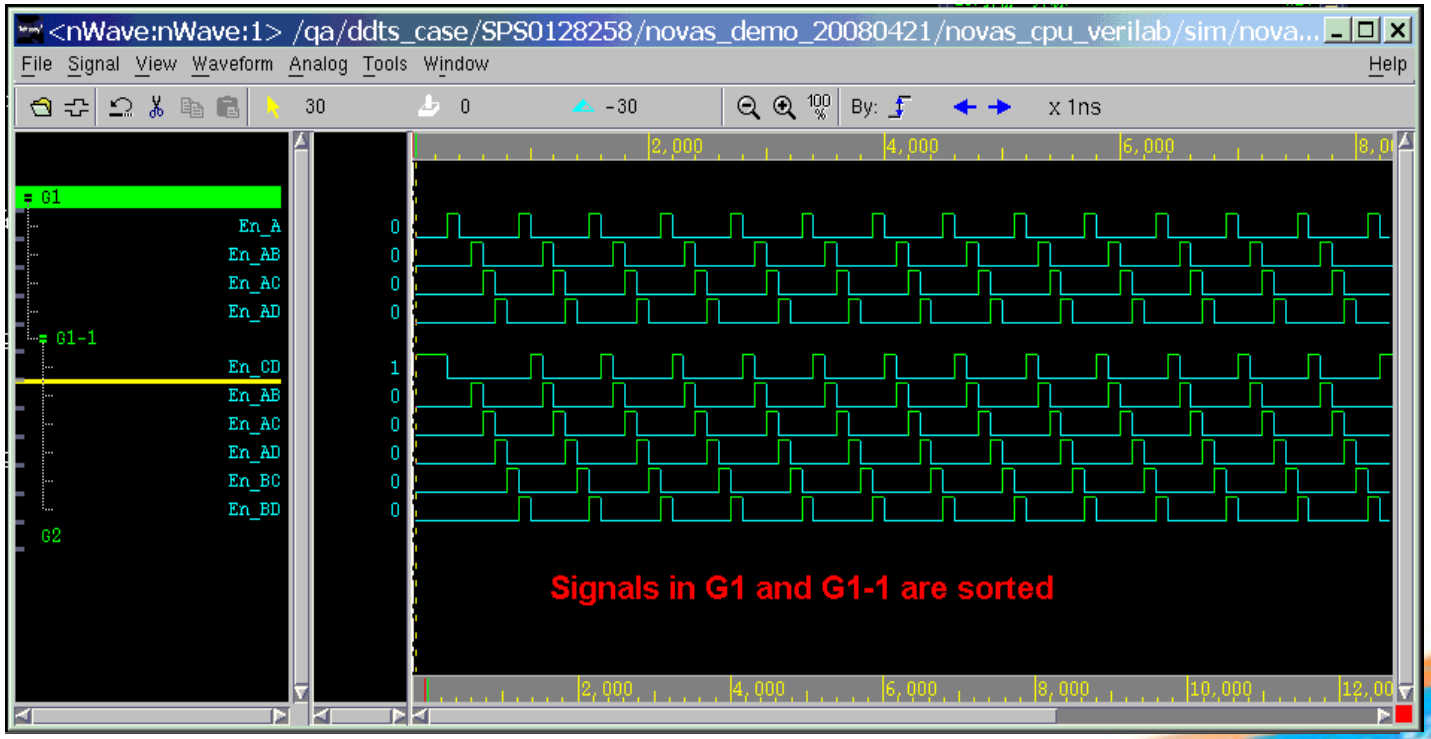
- Use Left Mouse Button/Middle Mouse Button to set **Cursor/Marker** time respectively.
- Invoke the **Signal → Select → Stuck Signals with Cursor/Marker** command.
  - Signals with no transition within Cursor/Marker time will be selected.
  - User can press Delete key to remove these selected signals.



Manipulate Waveform Display

Sort Signals by Transition

- Select a Signal Group in *nWave*, invoke **View → Sort Signals by First Transition** command to sort signals based on the first transition.
  - The initial unknown value will be ignored.



Summary

- In this section, you have learned…
  - How to open *nWave*
  - How to load simulation results
  - How to organize and operate on signals
  - How to manipulate the waveform display

Lab

- Refer to the *Verdi User's Guide and Tutorial* document in <Verdi_install>/doc/tutorial.pdf.
- Follow the steps in the *nWave Tutorial* chapter to practice the lab.

Topics

- Technology Background
- Set Up the Environment
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM View
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

Objectives

Debug in Schematic View

- After completing this section, you should be able to…
  - Open *nSchema*
    - Create Different Schematic Window Types
  - Correlate Other Views
  - Manipulate Viewing
    - Enable Detail RTL – Window or Instance
    - Hide Extraneous Buses
    - Colorize Hierarchies
  - Trace Signals
    - Fan-in/Fan-out/Two Points/Two Signals

Open *nSchema*

- Select an instance in the *nTrace* hierarchy browser and click on **New Schematic** icon. 

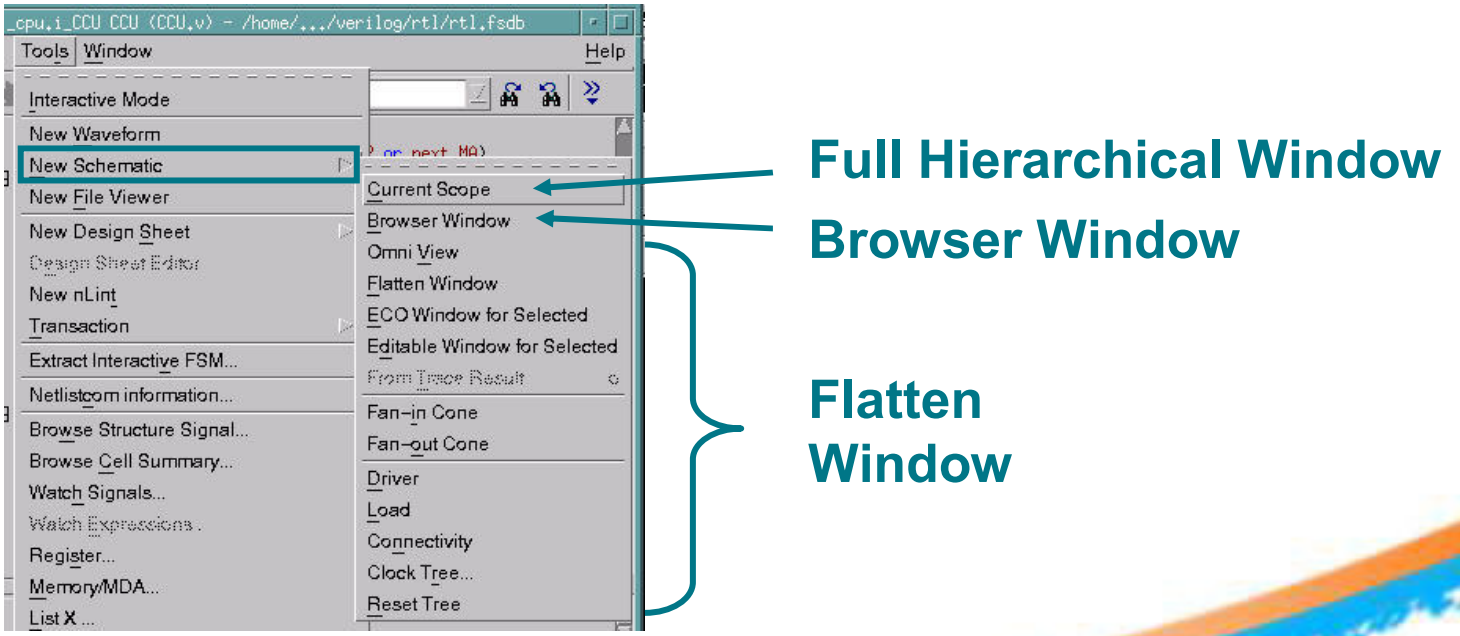
- **D&D** an instance from the hierarchy browser or a Verilog module or VHDL architecture from the *nTrace* source code pane to **New Schematic** icon. 

- Execute **Tools → New Schematic → Current Scope** in *nTrace*.

Open *nSchema*

Different Schematic Window Types

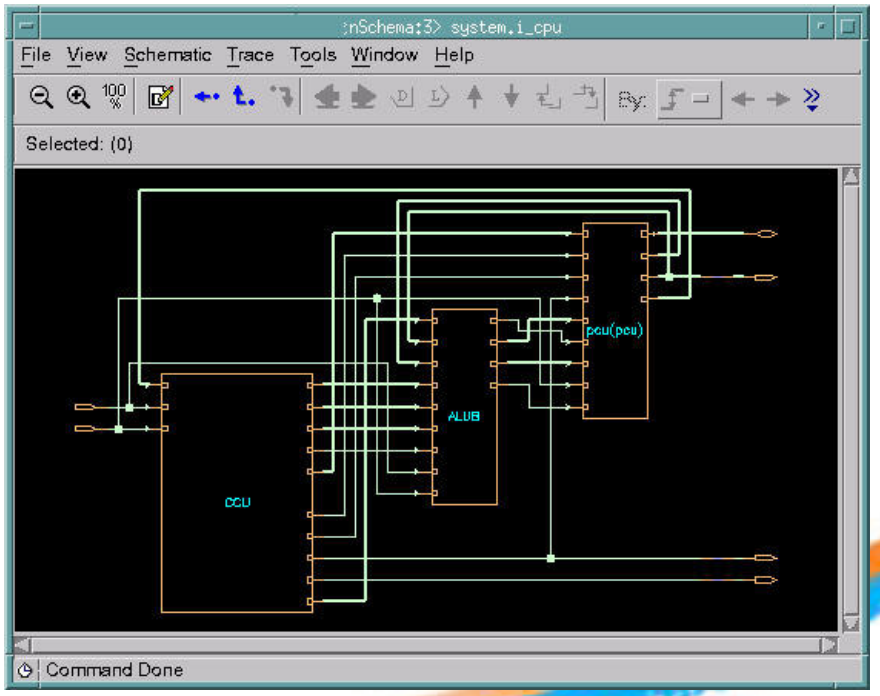
- **Full Hierarchical Window** – Shows complete objects in specific scope. Default schematic view.
- **Browser Window** – Shows partial schematic in specific scope.
- **Flatten Window** – Shows partial schematic in flatten view, i.e. schematics cross all scopes.



Open *nSchema*

Full Hierarchical Window

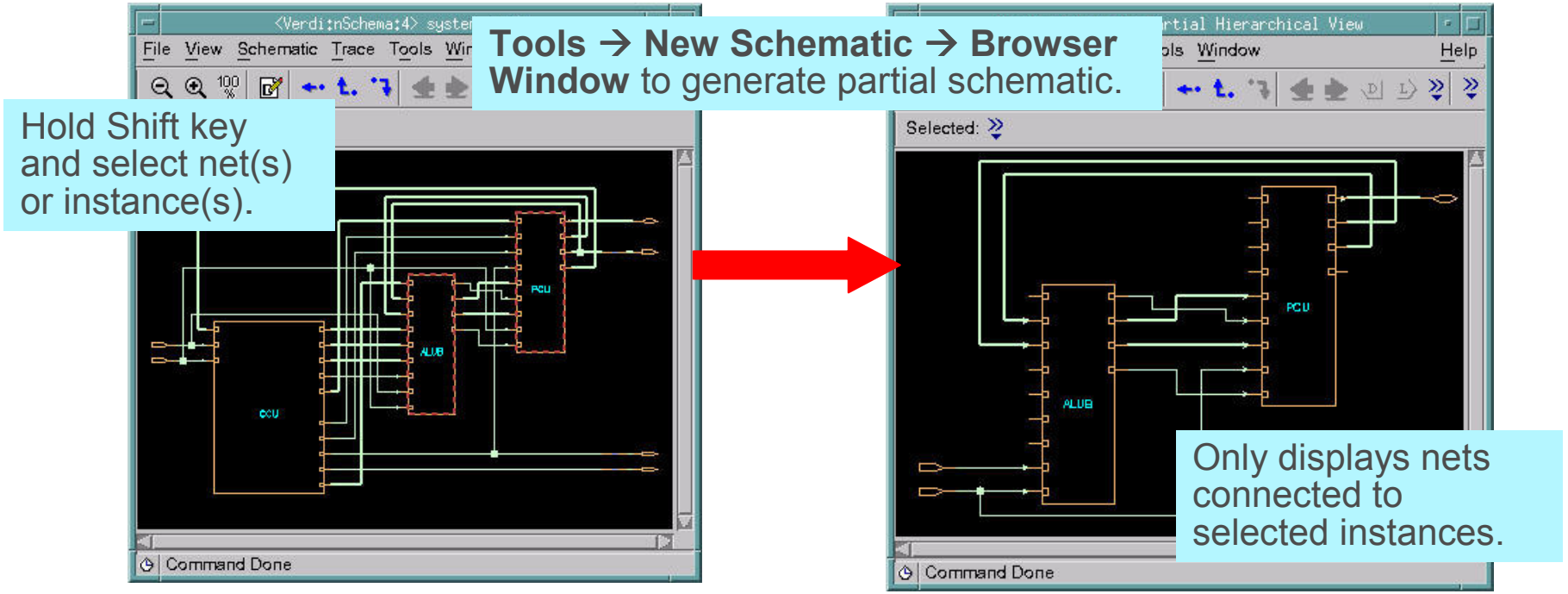
- View, traverse and understand design hierarchy graphically.
- **DC** a symbol port to push into the lower level and highlight the associated net.
- **DC** on a net to highlight all instances which connect to it.
- Trace commands will highlight a variety of different objects in current window.
- Search commands:
  - **Schematic →**
    Find Signal/Instance/Instport.
  - **Schematic →**
    Find In Current Scope.



Open *nSchema*

Browser Window

- Focus on specified instances or nets for debugging.

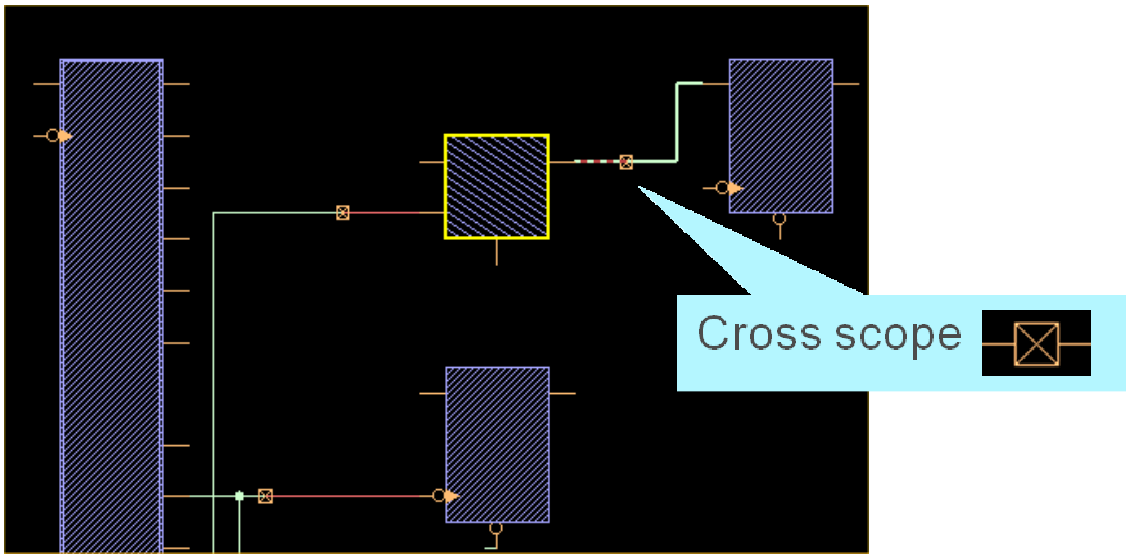


- DC on symbol port to push in, pop up or expand logic.
- Use Delete key or  to remove object(s).
- Undo / redo edits.
- No trace commands in this view.

Open *nSchema*

Flatten Window

- Focus on a specified primitive or net for debugging.

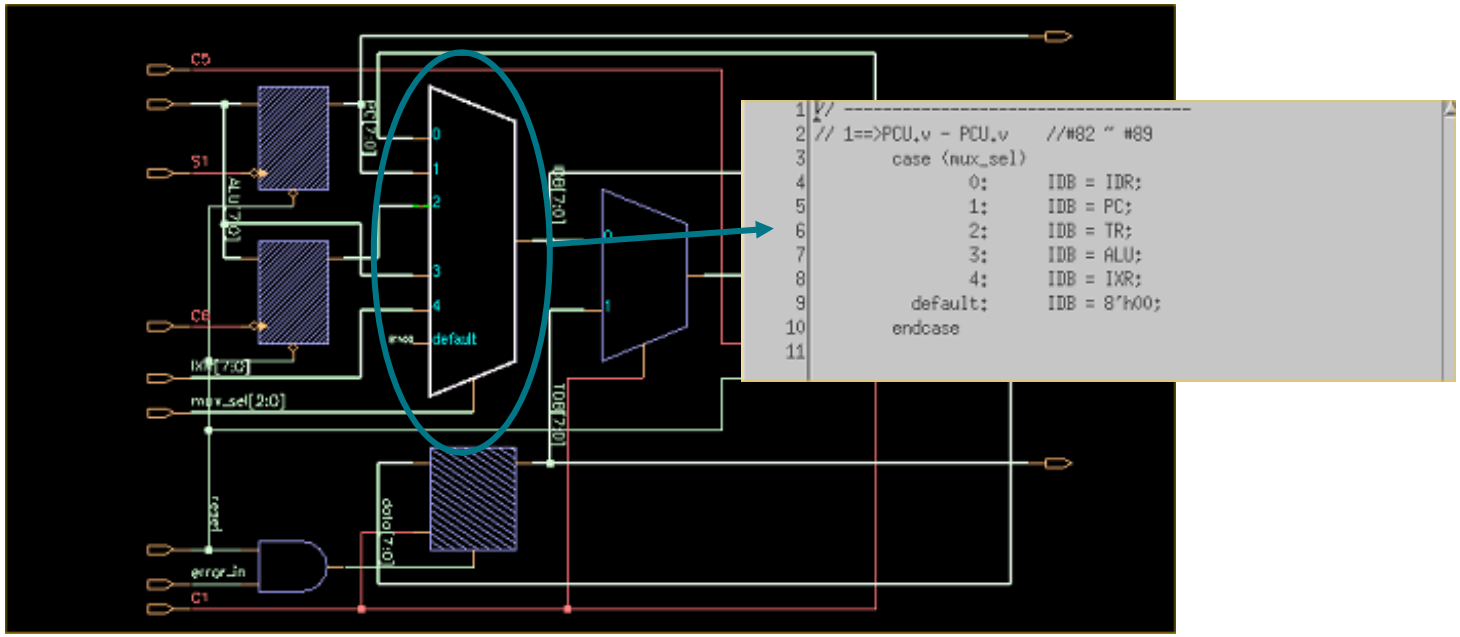


- DC on instance pin or D&D from other windows to add objects.
- Use <Delete> or  to remove selected object(s).

Open *nSchema*

View Detail Source Code for RTL Blocks

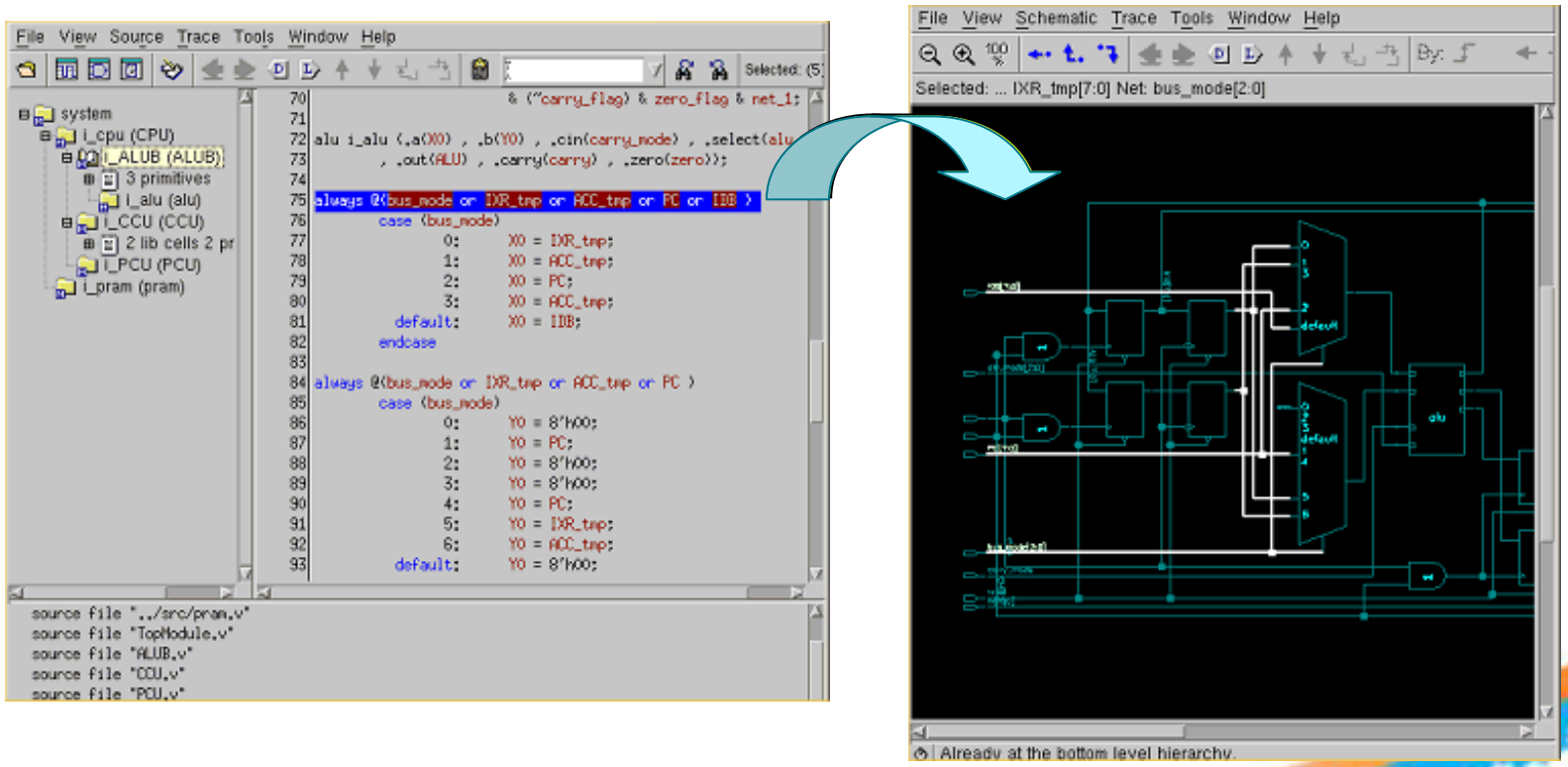
- DC on a bottom level RTL Block to show its corresponding source code.



Correlate Other Views

Drag & Drop between nSchema and Other Windows

- Select objects (instance, signal, …) in *nTrace.*
- D&D selected objects into *nSchema* to highlight them.



Manipulate Viewing

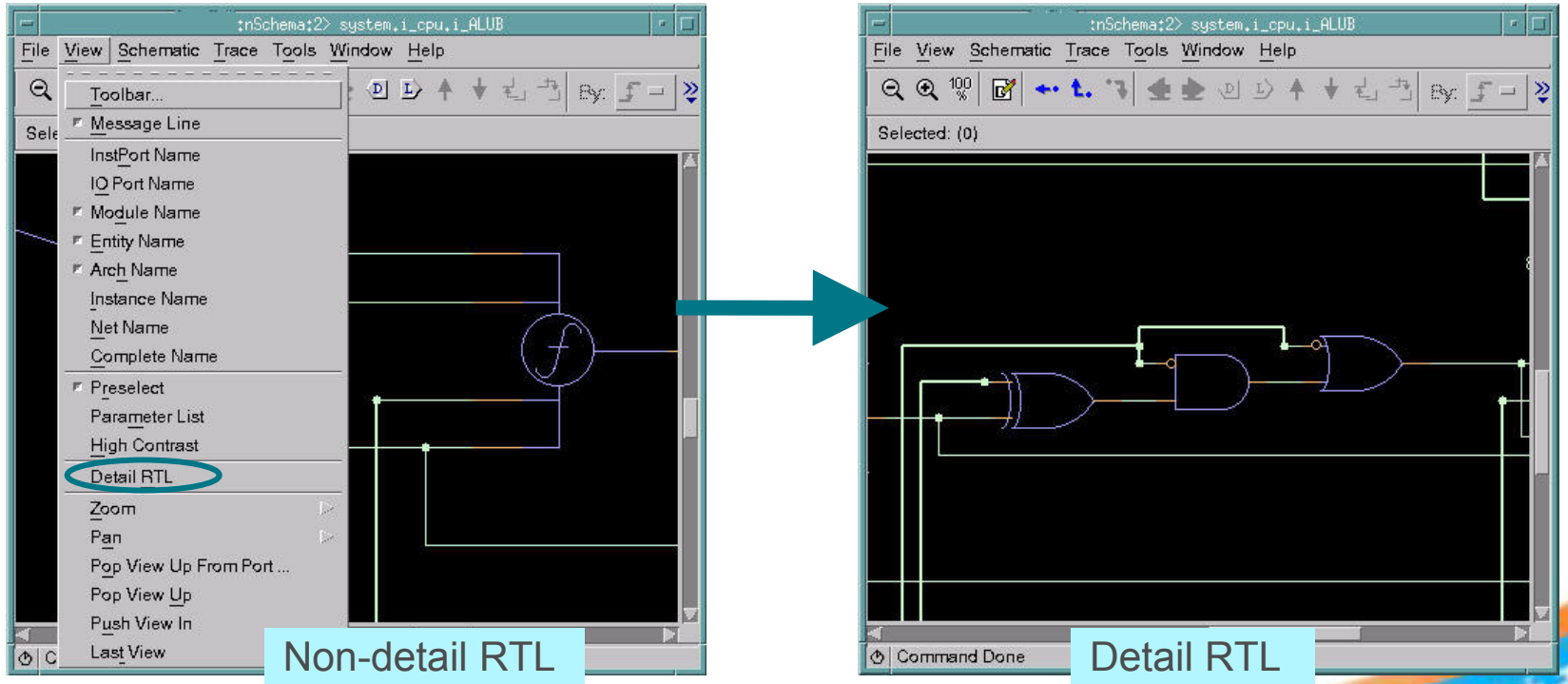
Miscellaneous Viewing Commands

- **View → Preselect**
  - Highlight signals/blocks when the cursor moves over them.
- **View → InstPort Name**
  - Turn on/off port names of instances.
- **View → IO Port Name, Instance Name, Net Name**
  - Turn on/off pin, instance, local net names.
- **View → High Contrast**
  - Display all schematic in dark blue except the selected objects which are highlighted in white.
- **Tools → Preferences → Schematics → Display Options → Viewing**
  - Change default viewing attributes when opening new *nSchema* windows.

Manipulate Viewing

View Detail RTL on a Window Basis

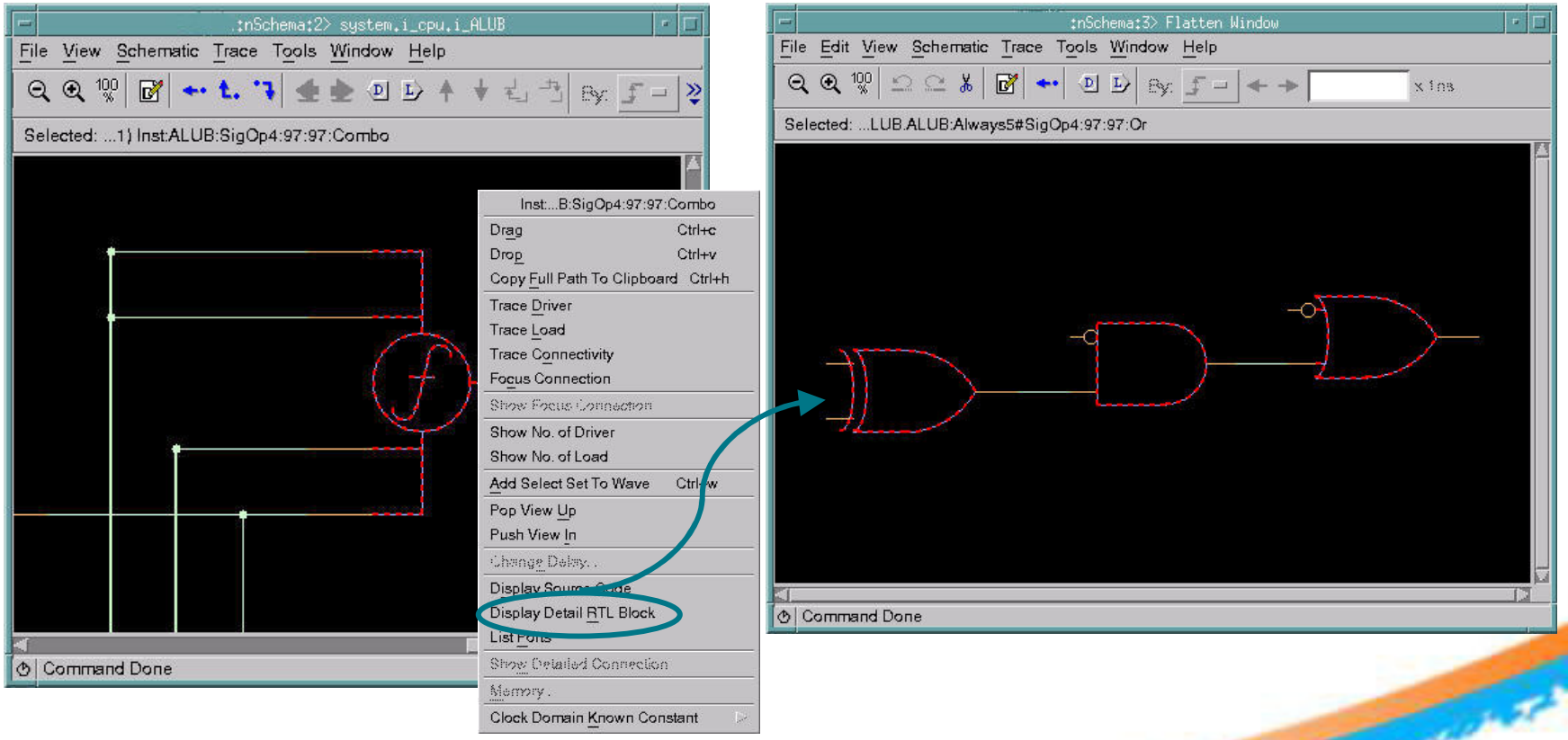
- Invoke **View → Detail RTL** to show Boolean equivalent of function symbols.
  - Effective in the current window only.



Manipulate Viewing

View Detail RTL on Instance Basis

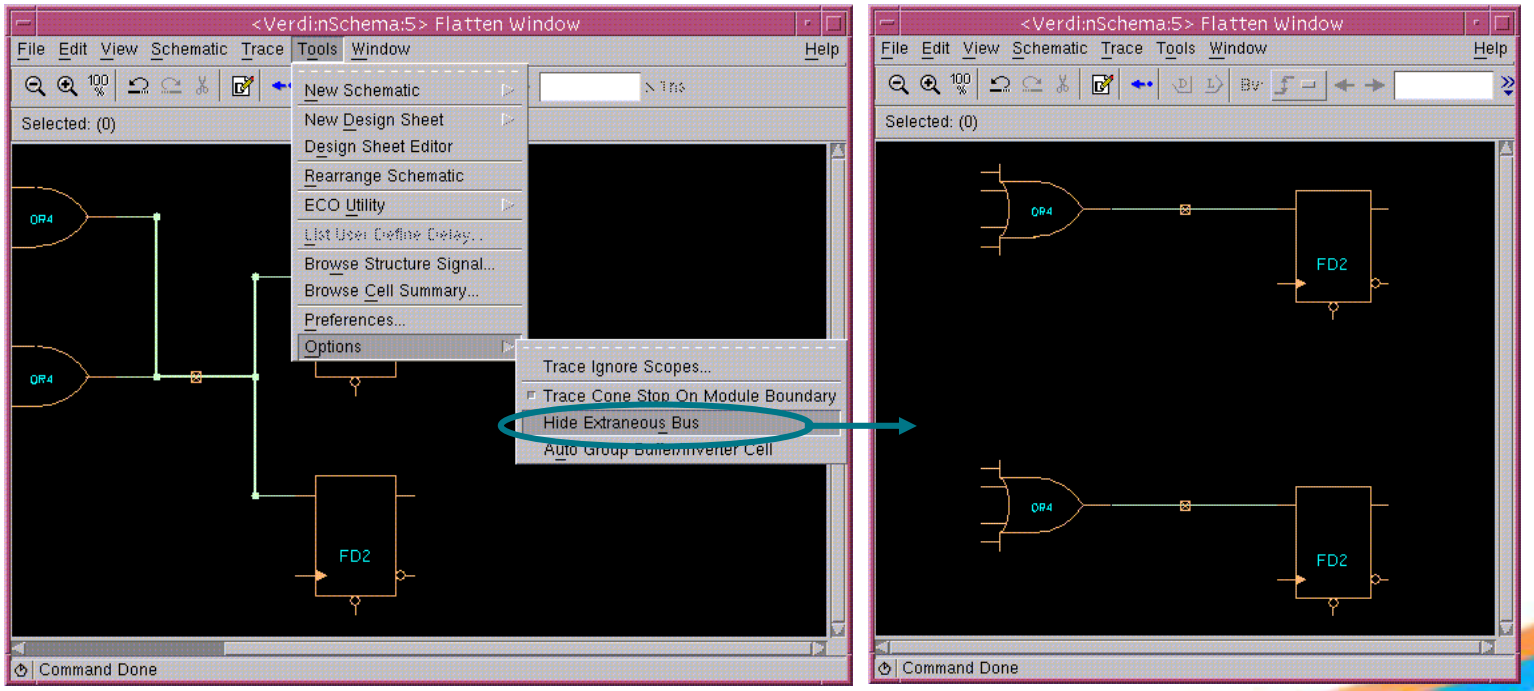
- On RTL block, invoke RMB → **Display Detail RTL Block**.
  - New window with details will open.



Manipulate Viewing

Hide Extraneous Bus in Flatten Window

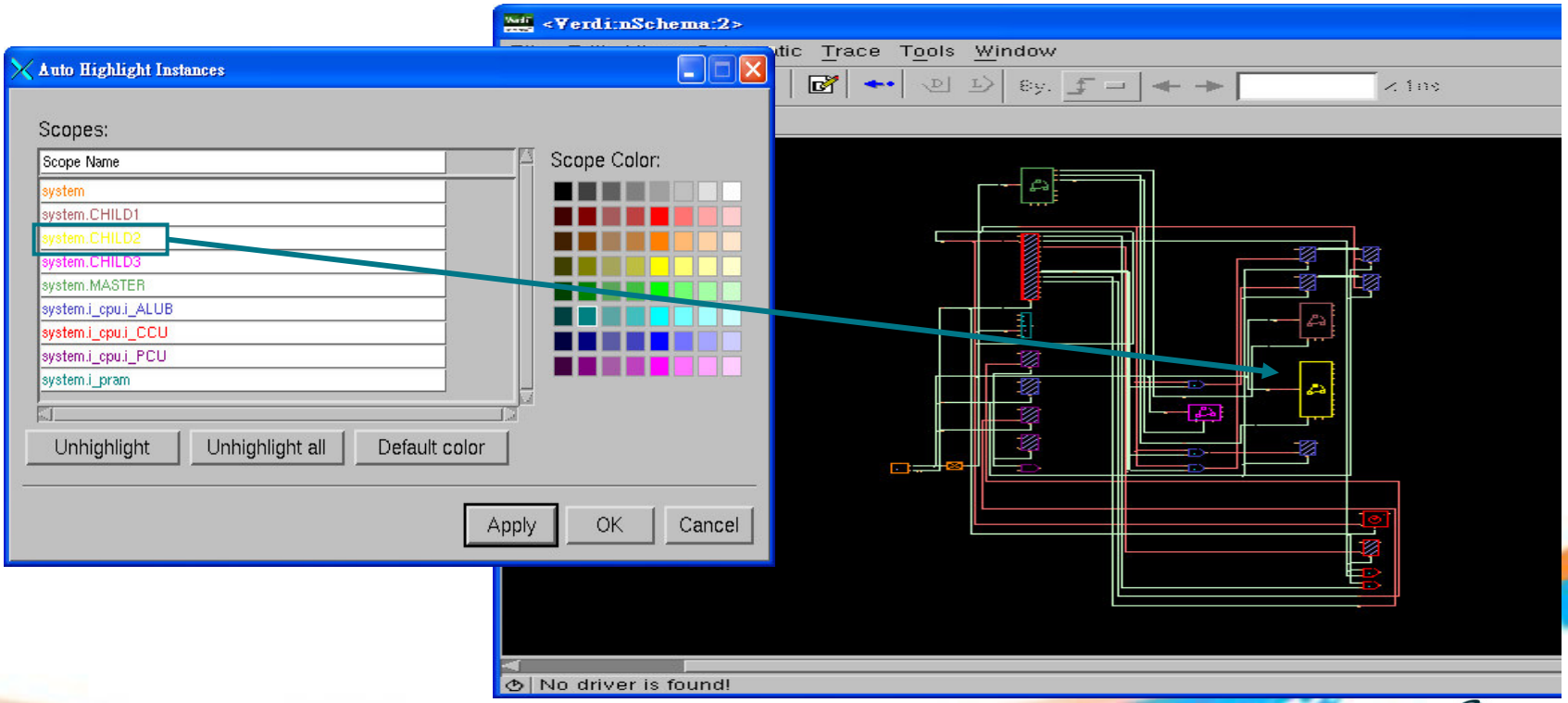
- Remove extraneous bus connections to display a cleaner schematic for bus signals.
  - Enable **Tools → Options → Hide Extraneous Bus**.
    - This option is only available in flattened schematic windows.



Manipulate Viewing

Colorize Different Hierarchies in Flatten Window

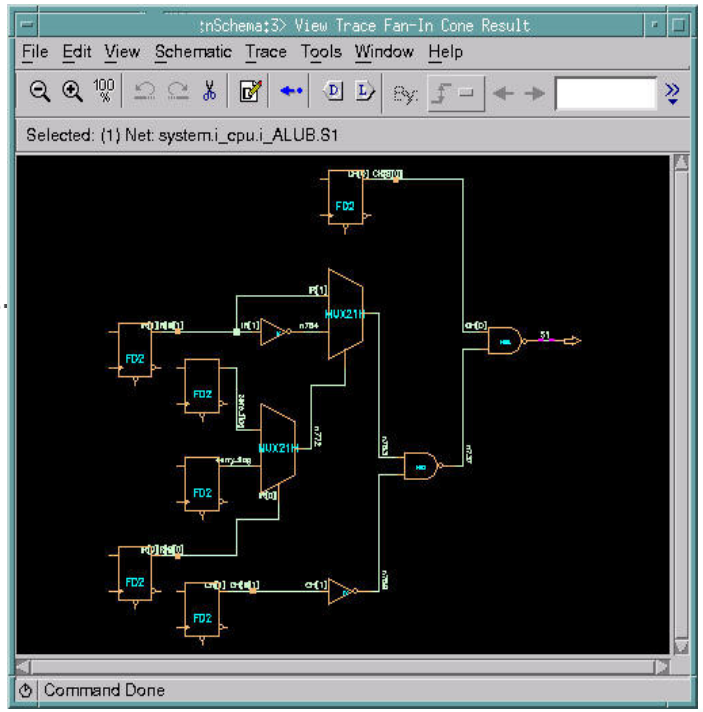
- Enable **Automatically Color Instances in Flatten Window** option on the **Schematics → Display Options → Schematic** folder of the *Preferences* form (invoked with **Tools → Preferences**).
- In flatten schematic window, invoke **Schematic → Color Instances by Scope** to change default scope colors.



Trace Signals

Trace Fan-in Cone / Fan-out Cone

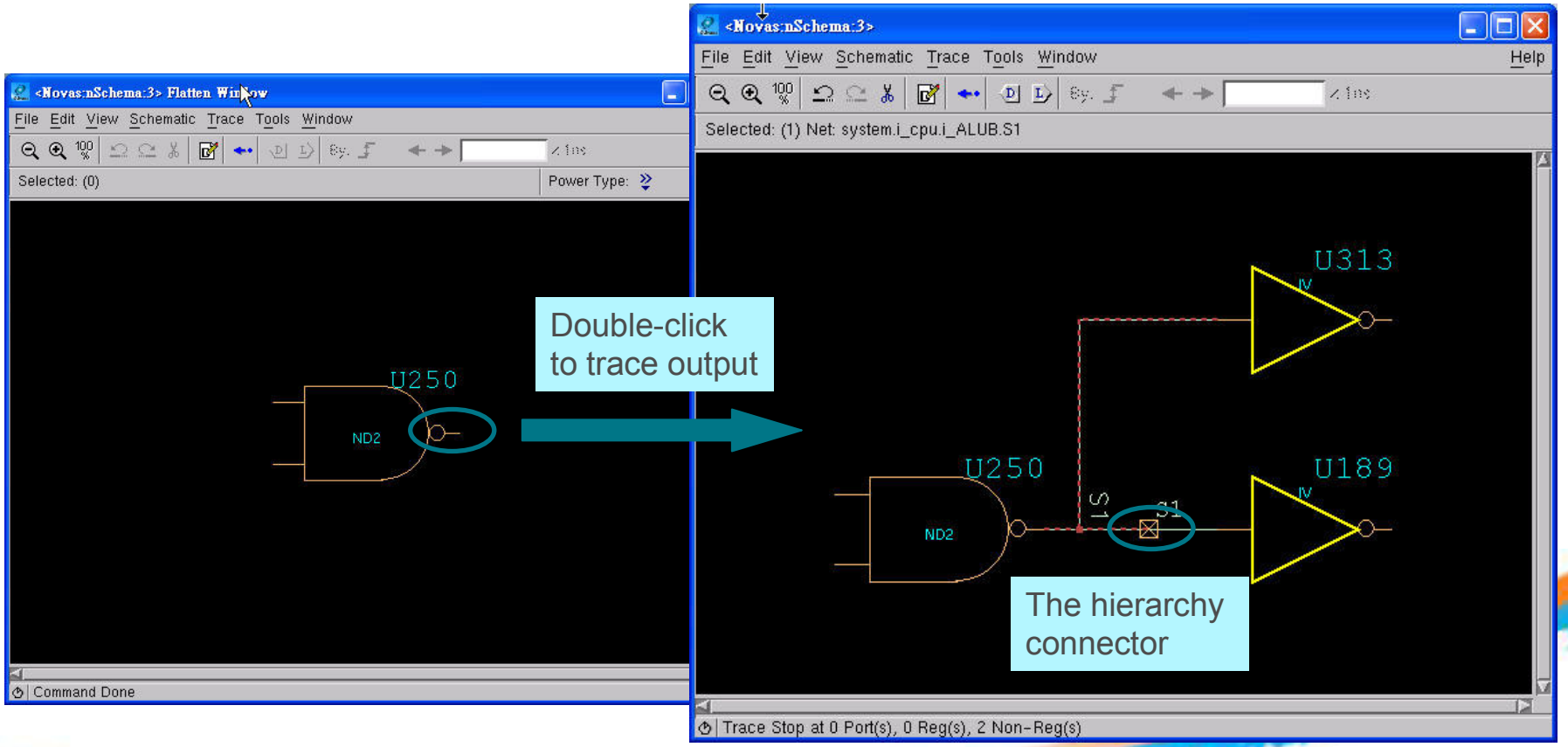
- Invoke **Tools → New Schematic → Fan-in Cone / Fan-out Cone**
  - Generate new schematic window to show all the fan-in or fan-out logic.
  - Trace net to register / tri-state output or design boundary.
- **Fan-In Cone**
  - Shows all the logic affecting the selected net.
  - Useful to trace drivers in depth to find the cause of error.
- **Fan-Out Cone**
  - Shows all the loading logic.
  - Useful to see global signal distribution (e.g. clock, reset, etc.).
- Trace fan-in/out more then one level.
  - Specify the level number in the **Fan-In/Out Level** field on the **Trace** page under the **Schematics** folder of the *Preferences* form (invoked with **Tools → Preferences**).



Trace Signals

Expand the schematic

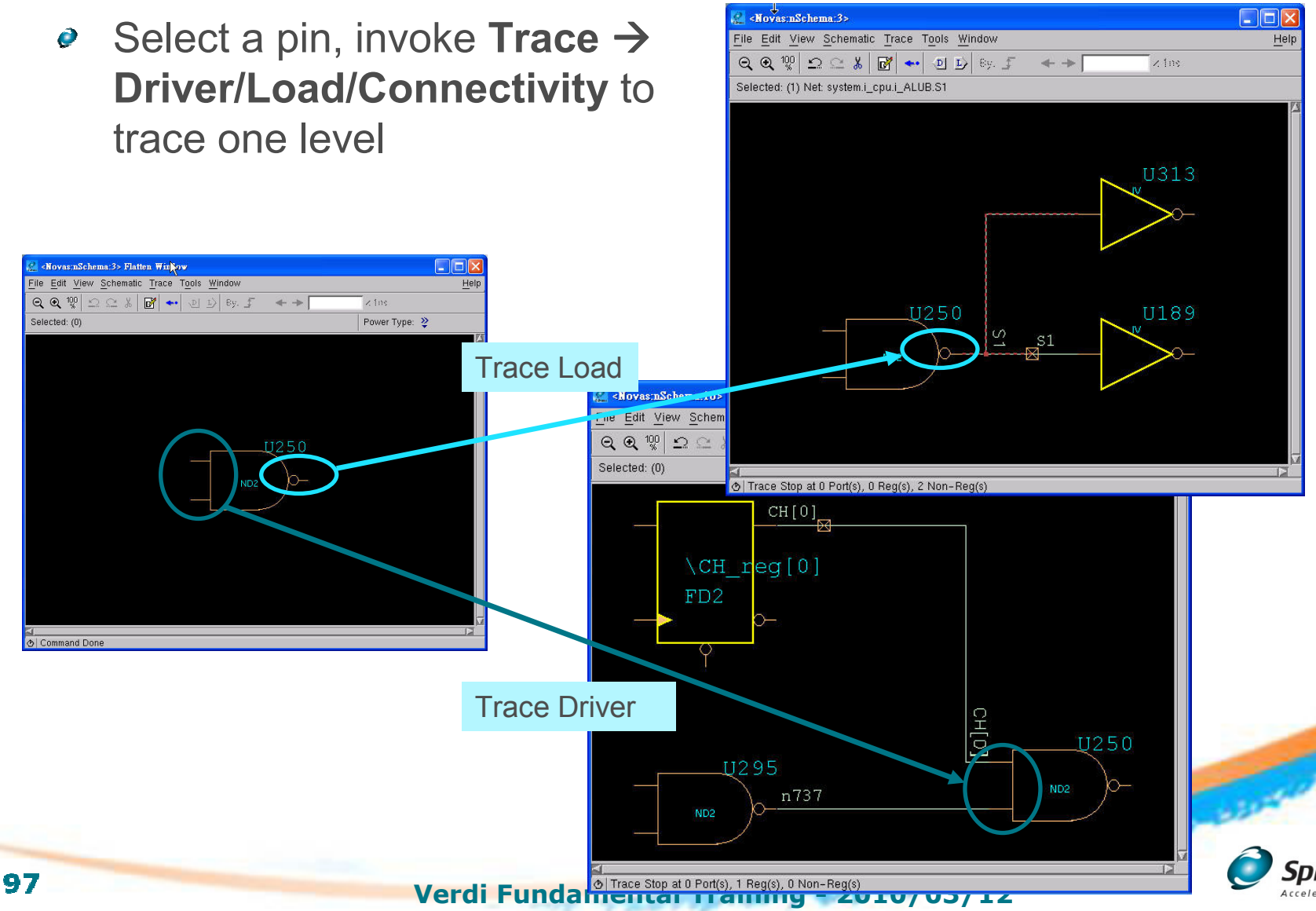
- Double-click on the instance pin to expand one level
  - The hierarchical connector will be added if tracing through hierarchies



Trace Signals

Trace Load/Driver/Connectivity

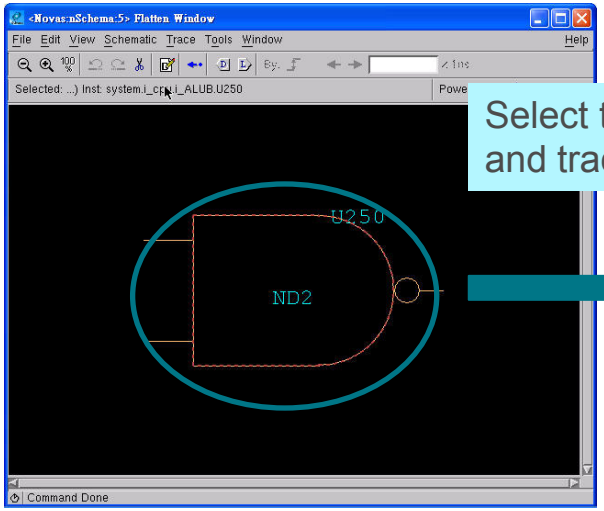
- Select a pin, invoke **Trace → Driver/Load/Connectivity** to trace one level



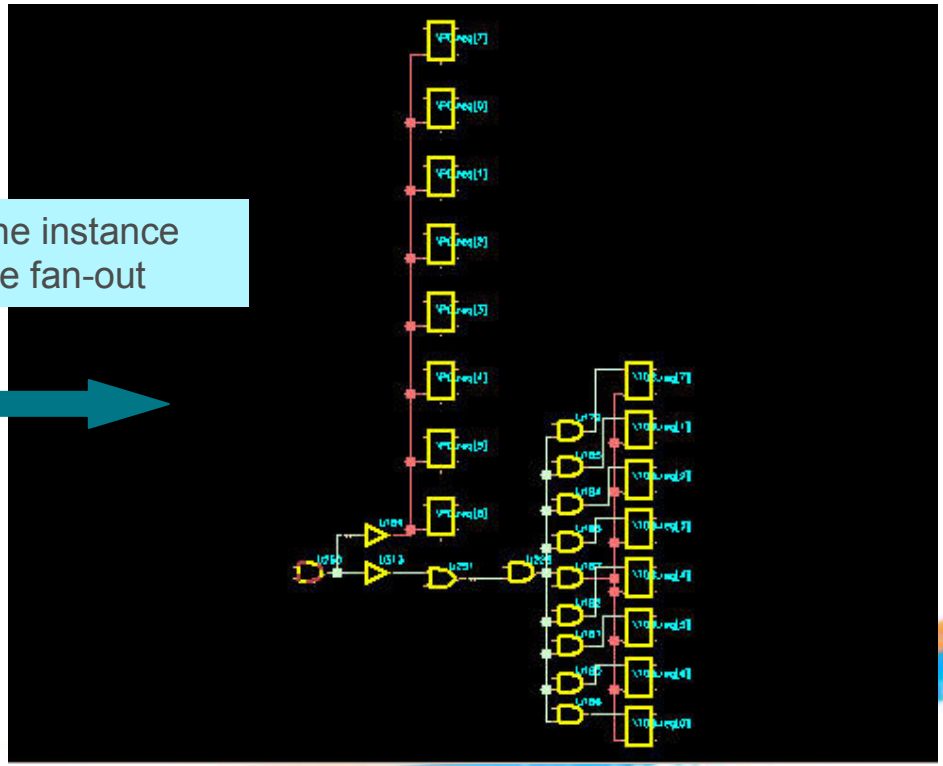
Trace Signals

Trace for Fan-out

- Select an instance or pin, invoke **Trace → Fan-out Cone**
  - Trace to storage elements
  - Useful to see global signal distribution (e.g. clock, reset, etc.).
  - Specify the level number in the **Fan-In/Out Level** field on **Schematics → Trace** folder of the *Preferences* form **(**invoked with **Tools → Preferences)**.



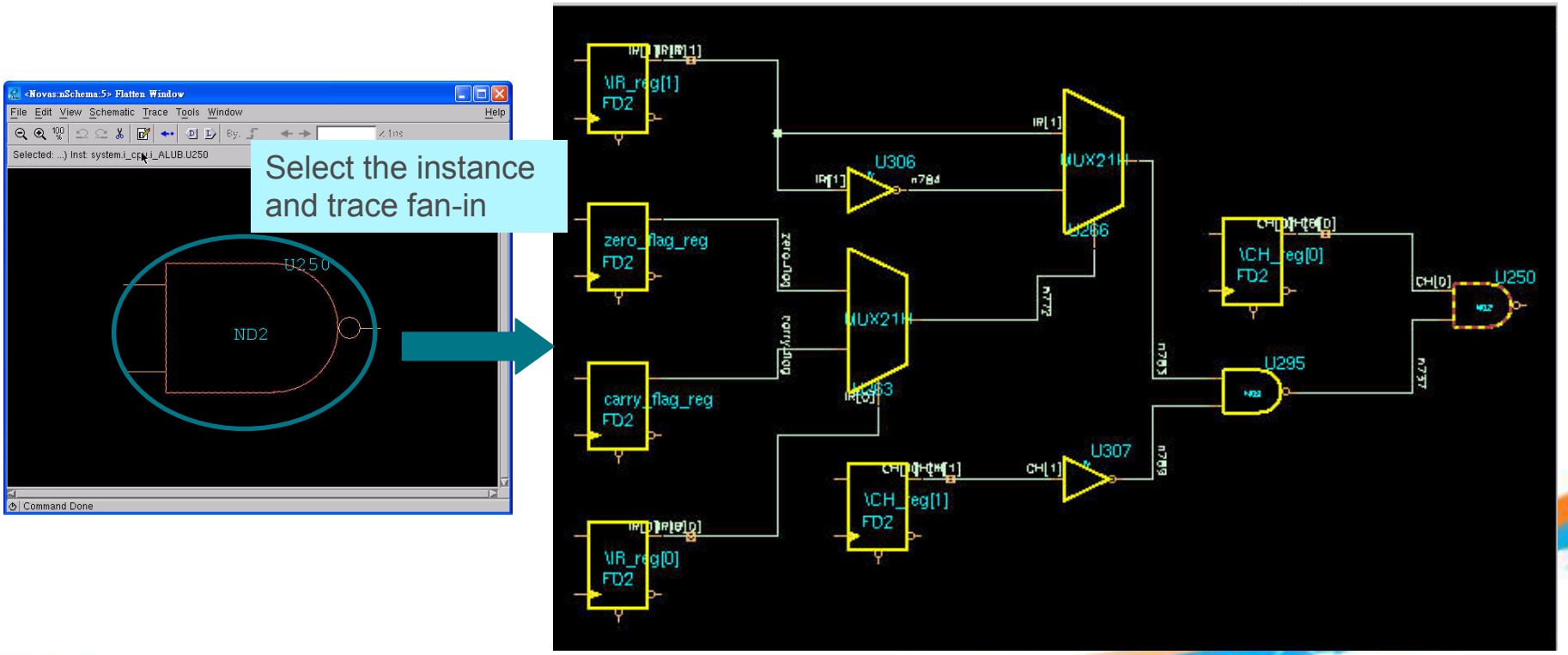
Select the instance and trace fan-out



Trace Signals

Trace for Fan-in

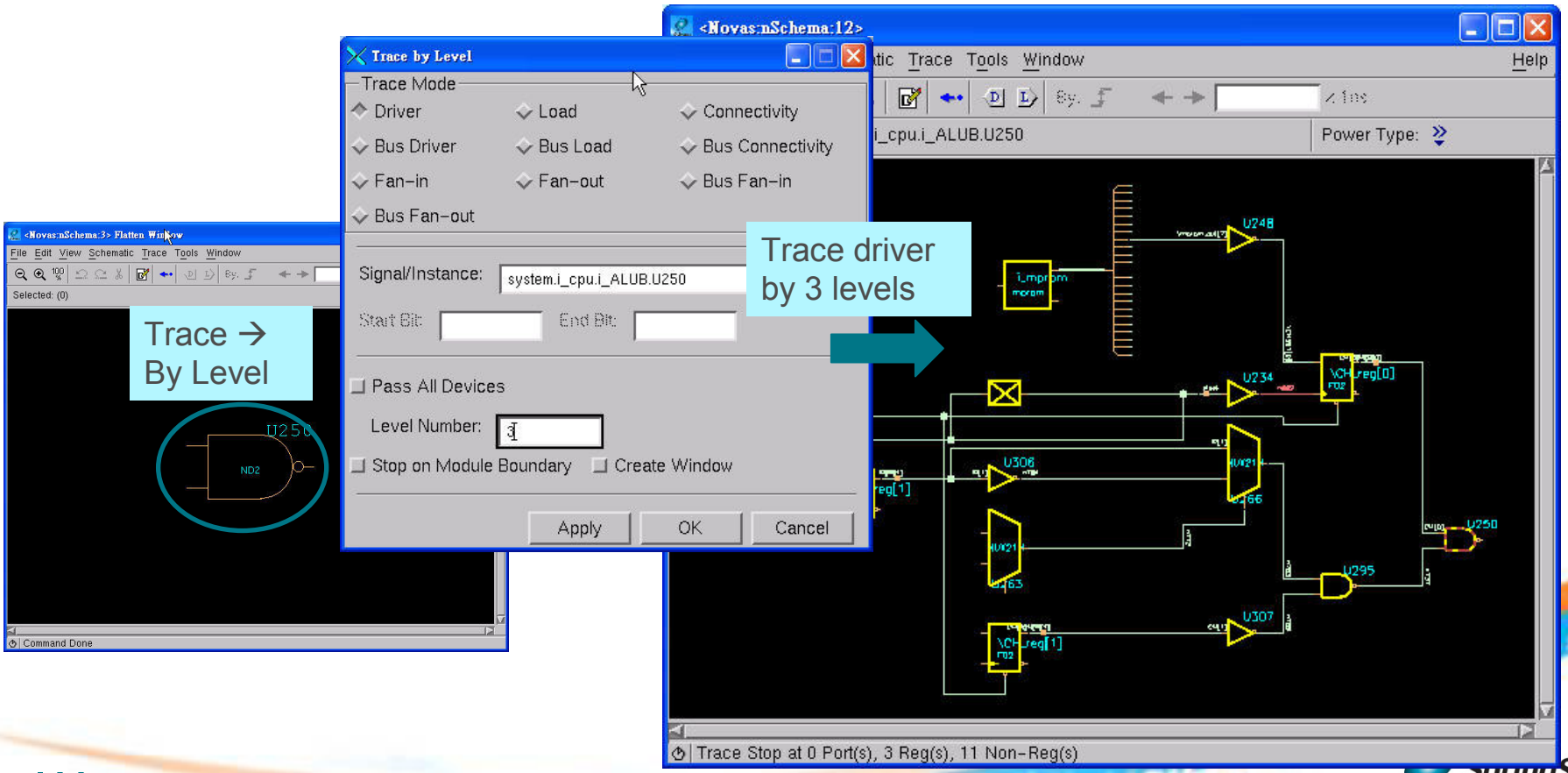
- Select an instance or pin, invoke **Trace → Fan-in Cone**
  - Trace to storage elements
  - Useful to trace drivers in depth to find the cause of error.
  - Specify the trace level in the *Preferences* form



Trace Signals

Trace by Level

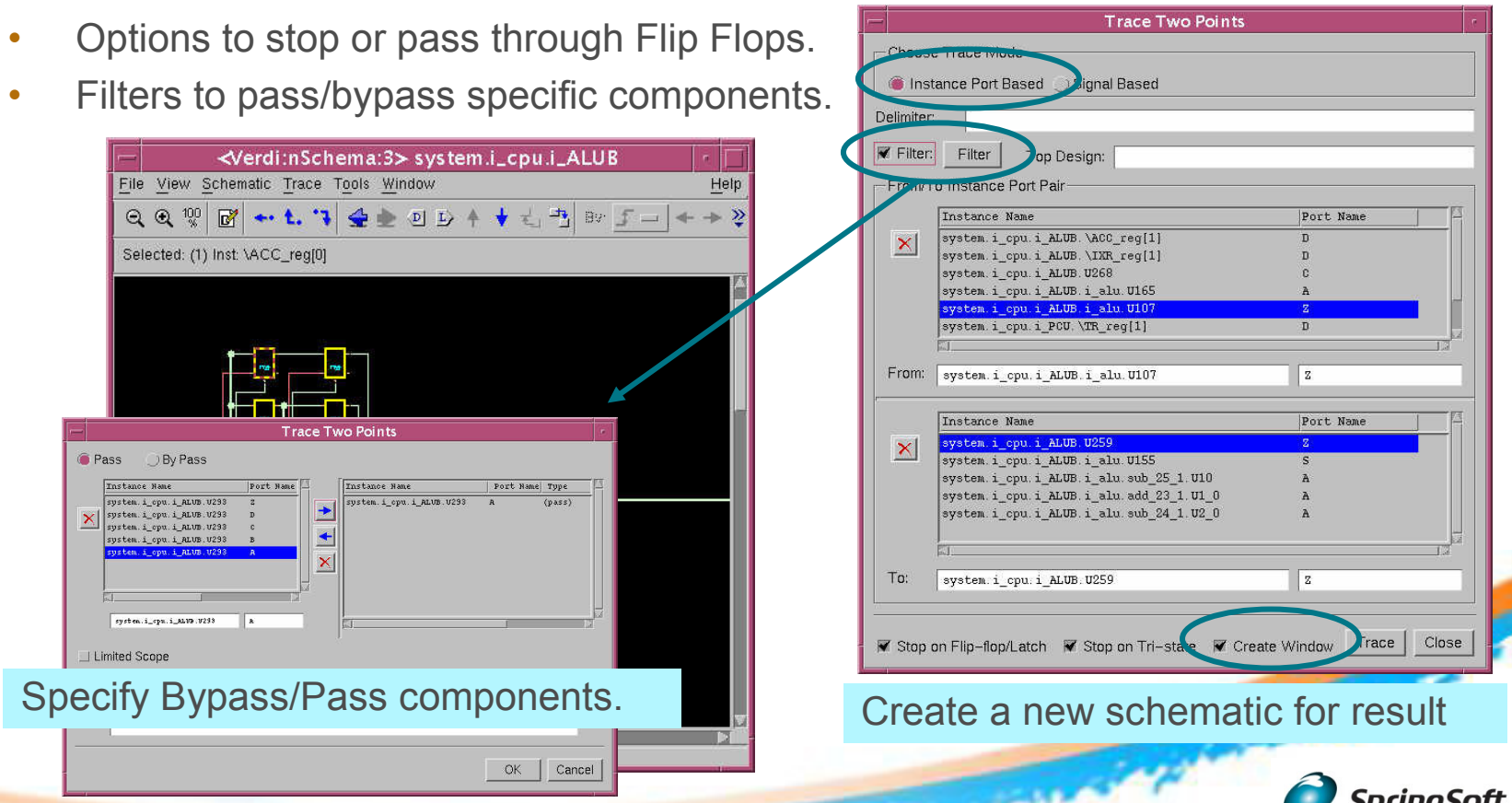
- Select an instance or pin, invoke **Trace → By Level**
  - Specify the Trace Mode (e.g. Driver, Load, Fan-in…etc)
  - Specify Level and other constrains



Trace Signals

Trace All Paths Between Two Points (1/2)

- Invoke **Trace → Two Points** and select **Instance Port Based** option to trace between ports.
  - Specify start/end points by entering the port names or D&D from *nSchema* or *nTrace*.
  - Options to stop or pass through Flip Flops.
  - Filters to pass/bypass specific components.

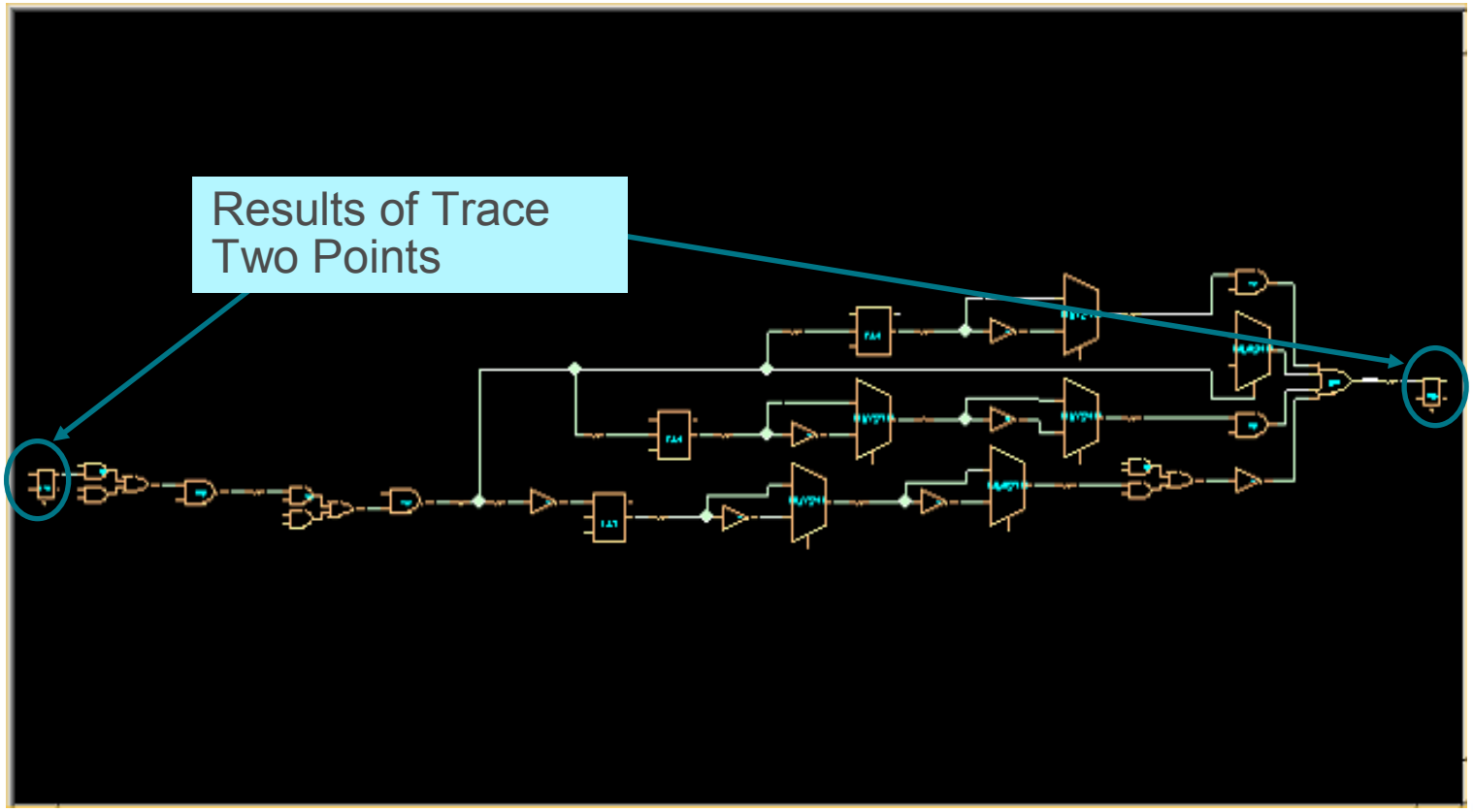


Specify Bypass/Pass components.

Create a new schematic for result

Trace Signals

Trace All Paths Between Two Points (2/2)

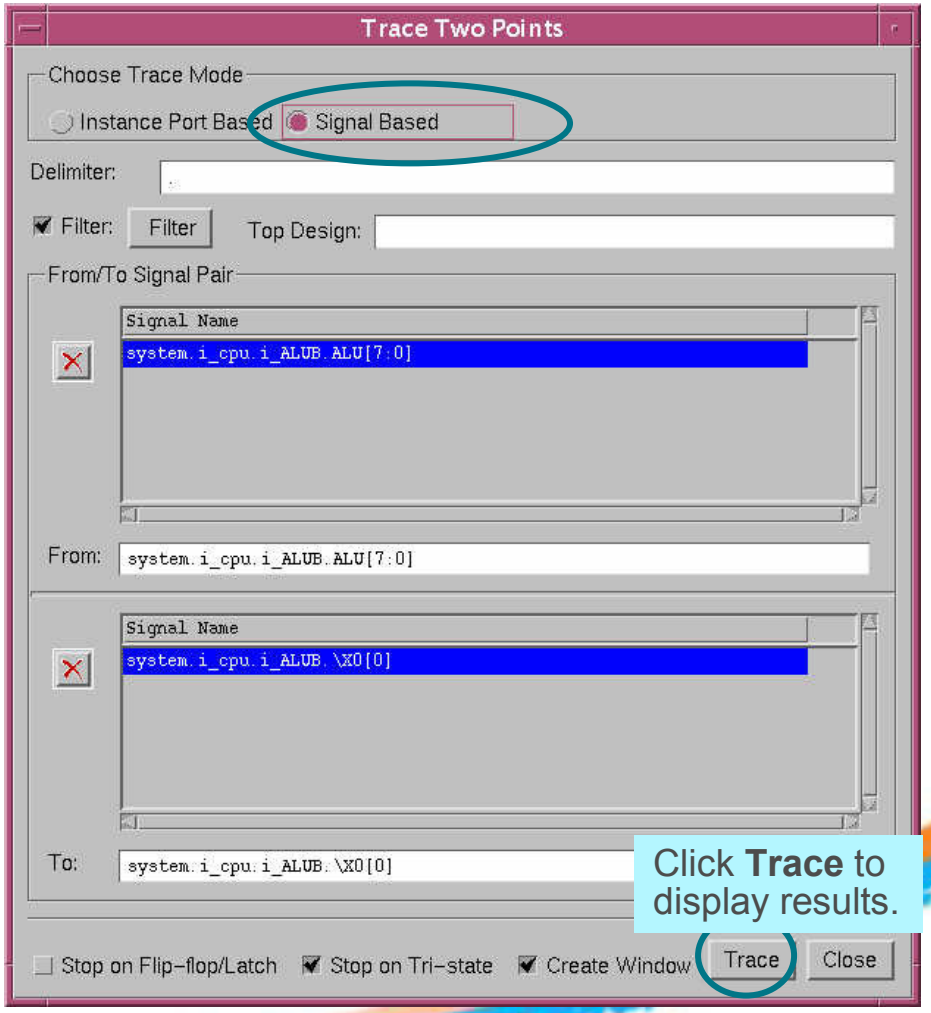


Trace Signals

Trace All Paths Between Two Signals

- In *Trace Two Points* form, select **Signal Based** option in **Choose Trace Mode** section.
  - Drag and drop signals from *nSchema* (or other views) to **From** and **To** fields.
  - Enable **Filter** option and then click **Filter** button to set pass/bypass signals.

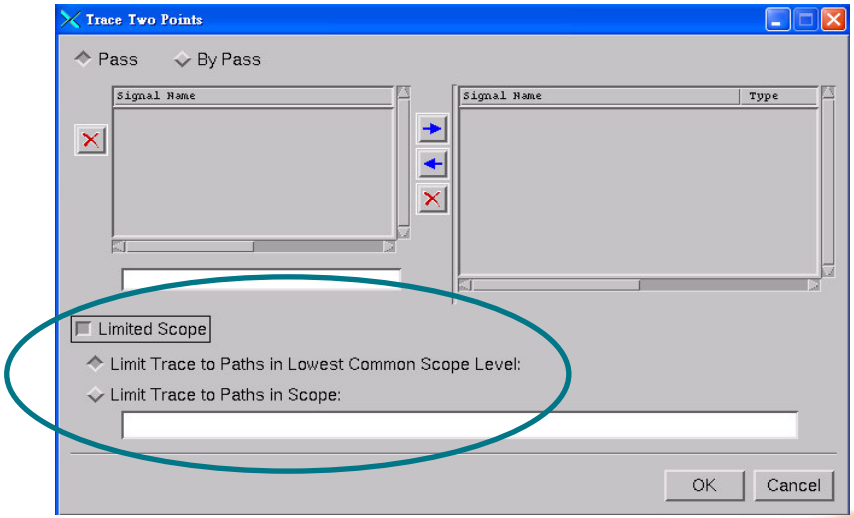
- Trace paths between signals will also show the instances which are connected to the signal in *nSchema*.



Trace Signals

Limit Tracing Scope for Performance

- In *Trace Two Points* form, enable **Filter** option and click **Filter** button.
- In *Trace Two Points Filter* form, select **Limited Scope** option and then select from the following options to limit tracing in specific scopes.
  - **Limit Trace to Paths in Lowest Scope Level**: Automatically set the lowest scope based on your two points/signals.
    - For example, when tracing from signal **A.B.C.D.in** to **A.B.E.F.out**, Verdi will only trace paths under **A.B** and its sub-scopes.
  - **Limit Trace to Paths in Scope**: Specify a scope to limit the trace to.
    - For when you want to focus on a specific scope.



Summary

- In this section, you have learned…
  - How to create different schematic windows.
  - How to manipulate viewing.
  - How to trace signals.
  - How to correlate other views.

Lab

- Refer to the *Verdi User's Guide and Tutorial* document in <Verdi_install>/doc/tutorial.pdf.
- Follow the steps in the *nSchema Tutorial* chapter to practice the lab.

Topics

- Technology Background
- Set Up the Environment
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM View
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

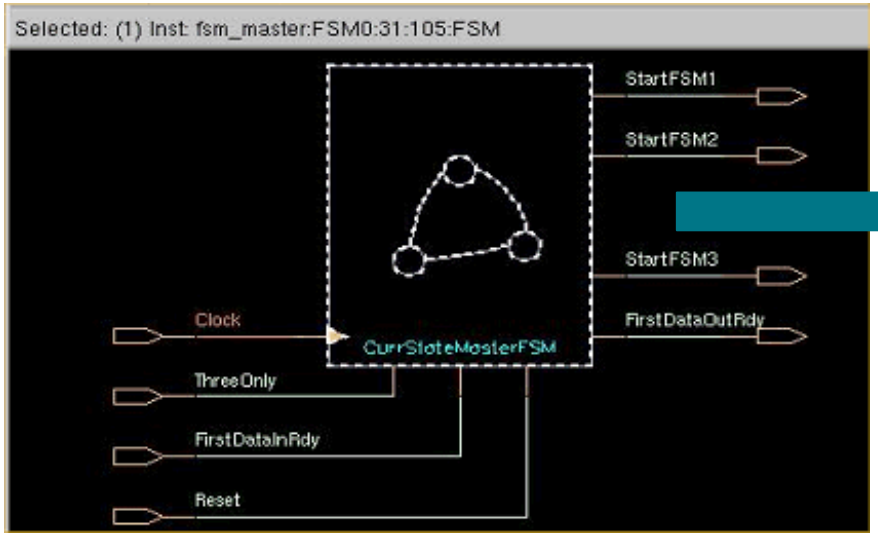
Objectives

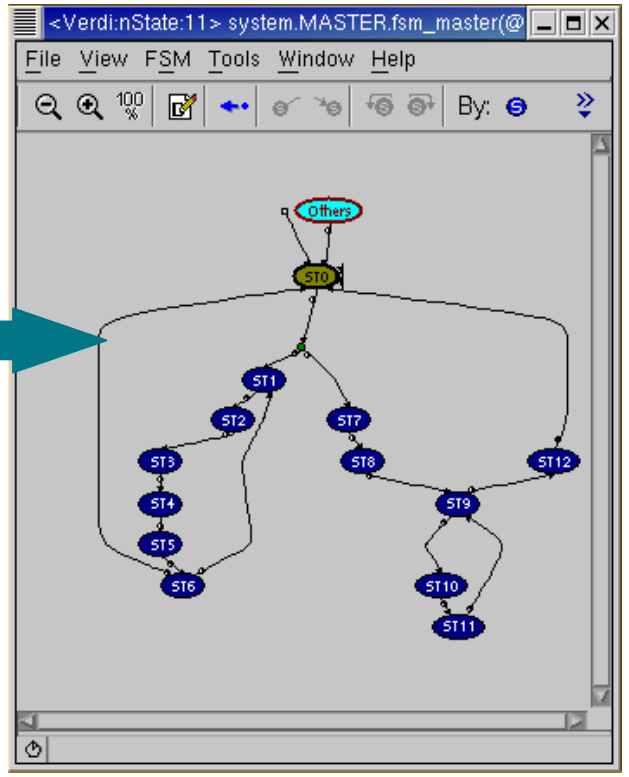
Debug in FSM View

- After completing this section, you should be able to…
  - Open the *nState* Window
  - Manipulate FSM View
    - View Actions & Find States
    - Display FSM Properties
    - Create Partial FSM
    - Use State Animation
  - Analyze State Machines
  - Correlate Other Views

Open *nState*

- DC on the state symbol to invoke *nState.*





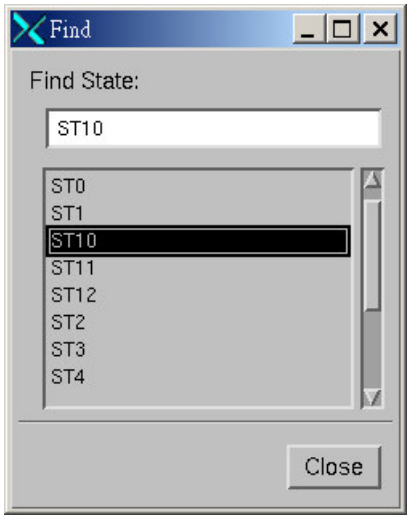
- Verdi automatically identifies FSM code for 1-process, 2-process, or 3-process with conventional encoding of the state variables.

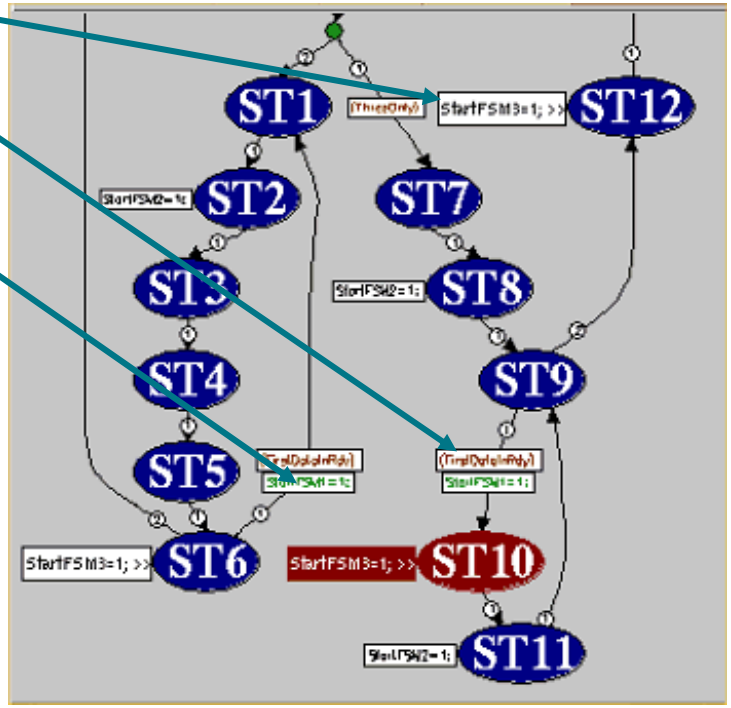
  NOTE: Remember to enable **FSM Recognition** option on the **RTL** page under the **Schematics** folder of the *Preferences* form (invoked with **Tools → Preferences**) otherwise not all FSM will be extracted.

Manipulate FSM View

View Actions & Find States

- **View → State Action**
- **View → Transition Condition**
- **View → Transition Action**
- **FSM → Find State**

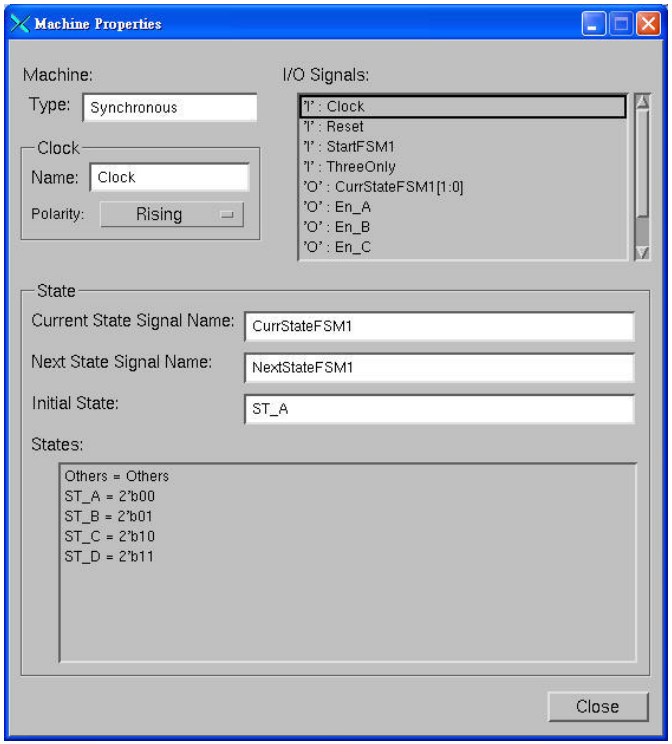


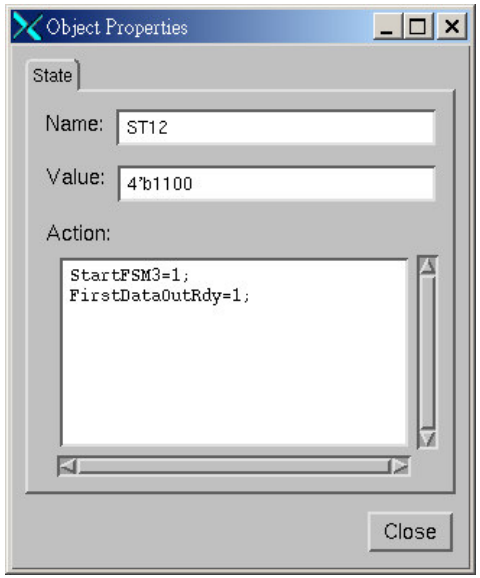


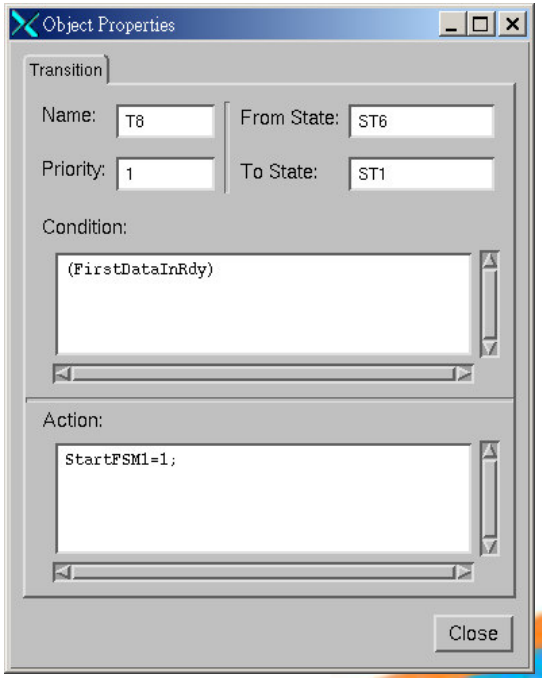
Manipulate FSM View

Display FSM Properties

- Invoke **FSM → Machine Properties** to access the FSM properties.
- Invoke **FSM → Object Properties** to access state and transition properties.



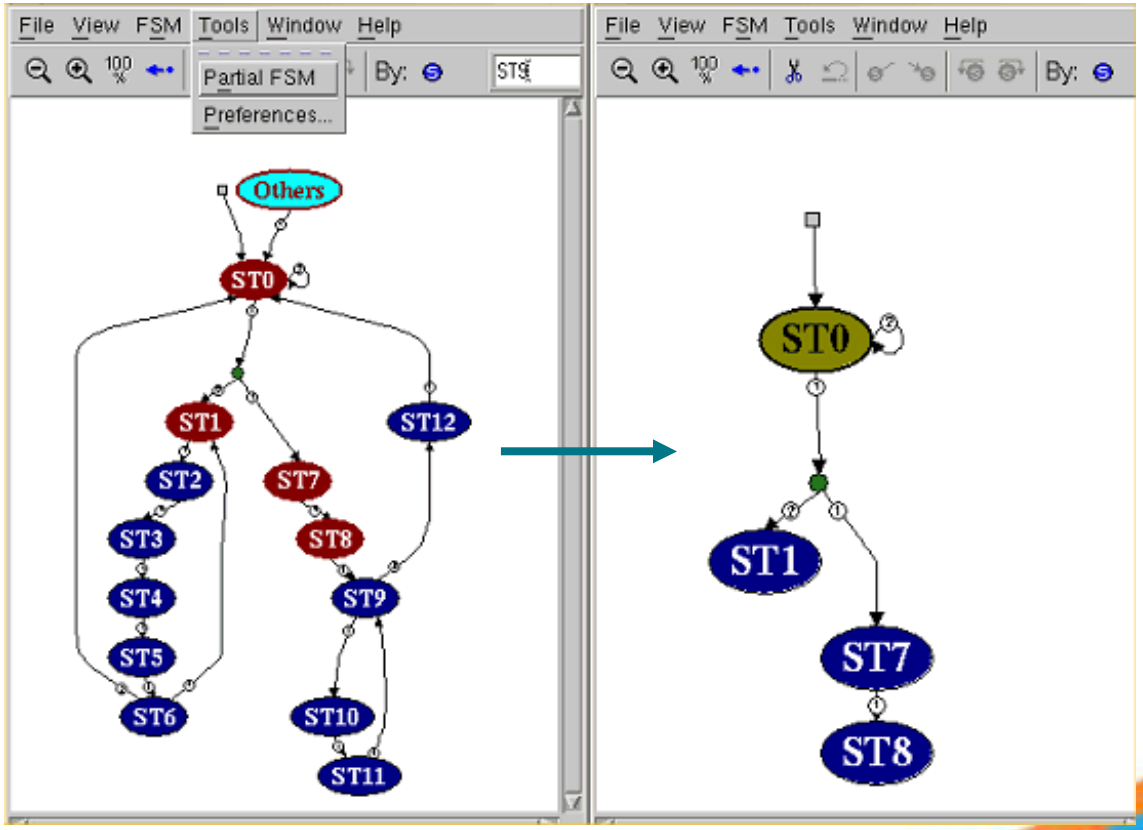




Manipulate FSM View

Create Partial FSM

- Select states in *nState* and invoke window from **Tools → Partial FSM**.
- Expand adjacent states.
  - DC on states.
  - Select a state and invoke **RMB → Expand**.
- D&D to the partial FSM window to add states.



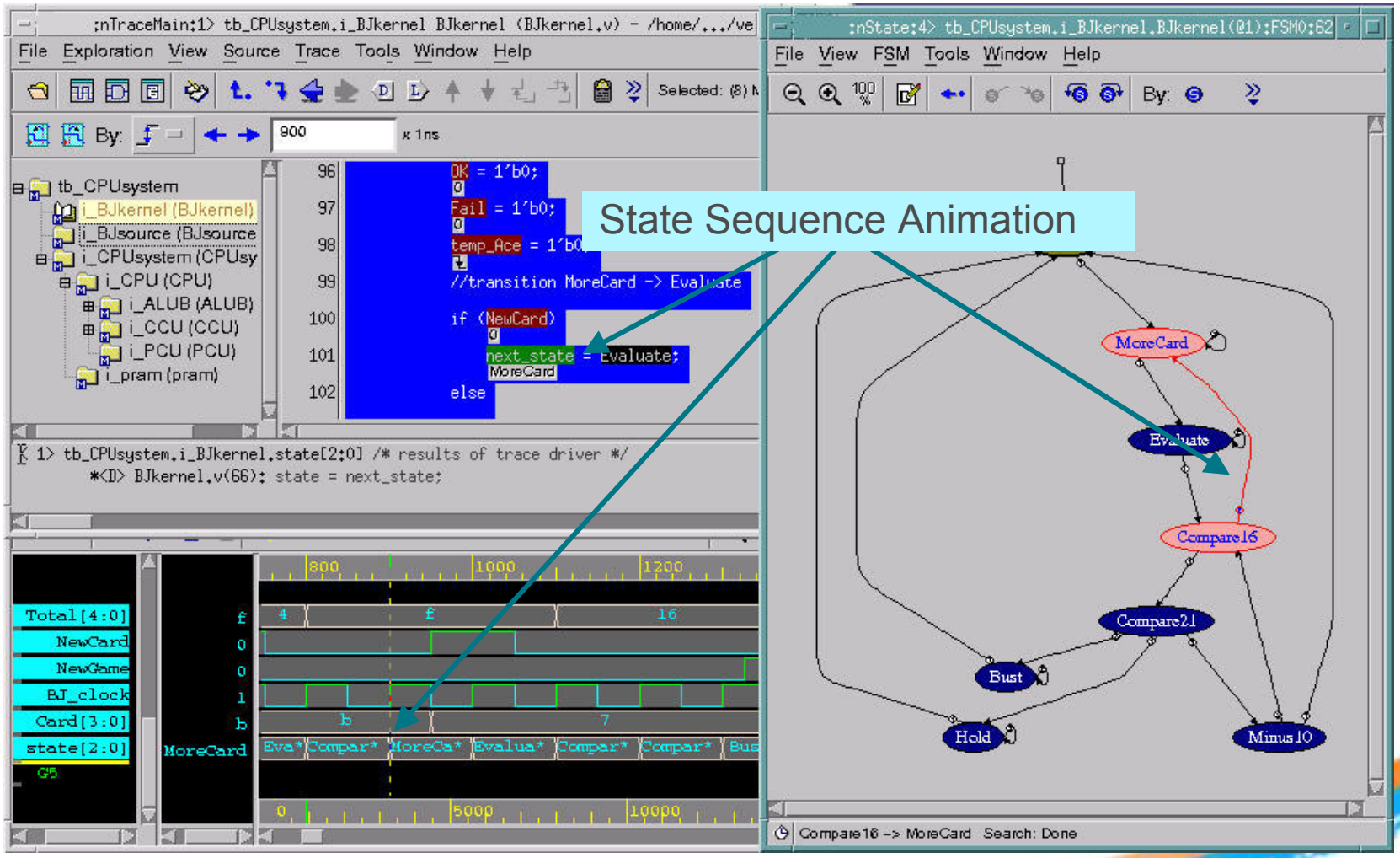
Manipulate FSM View

State Animation (1/2)

- Load simulation results.
- Enable **FSM → State Animation**.
- Use **Previous State / Next State** toolbar icons to step through the states.
- Use **FSM → Edit Search Sequence** to create a sequence of states to search on.

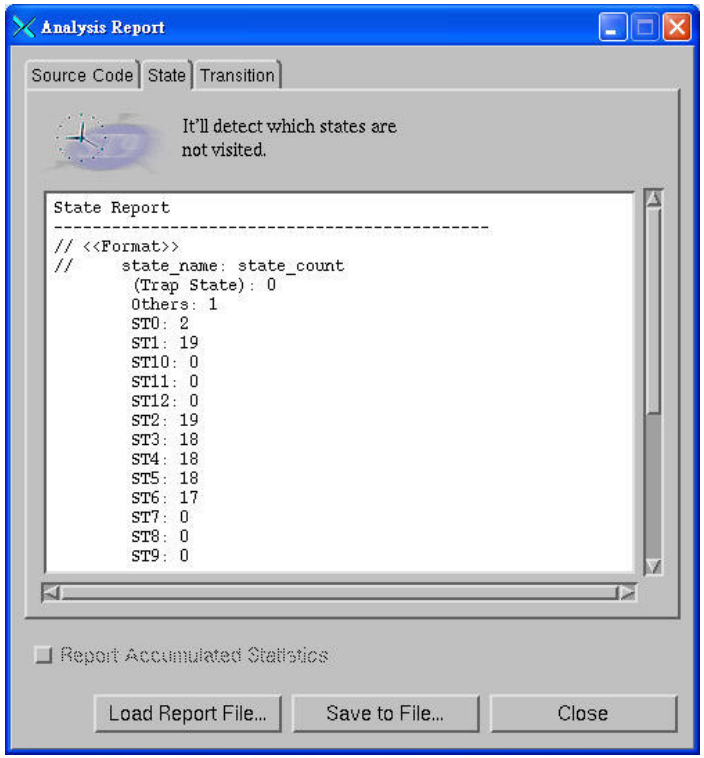
Manipulate FSM View

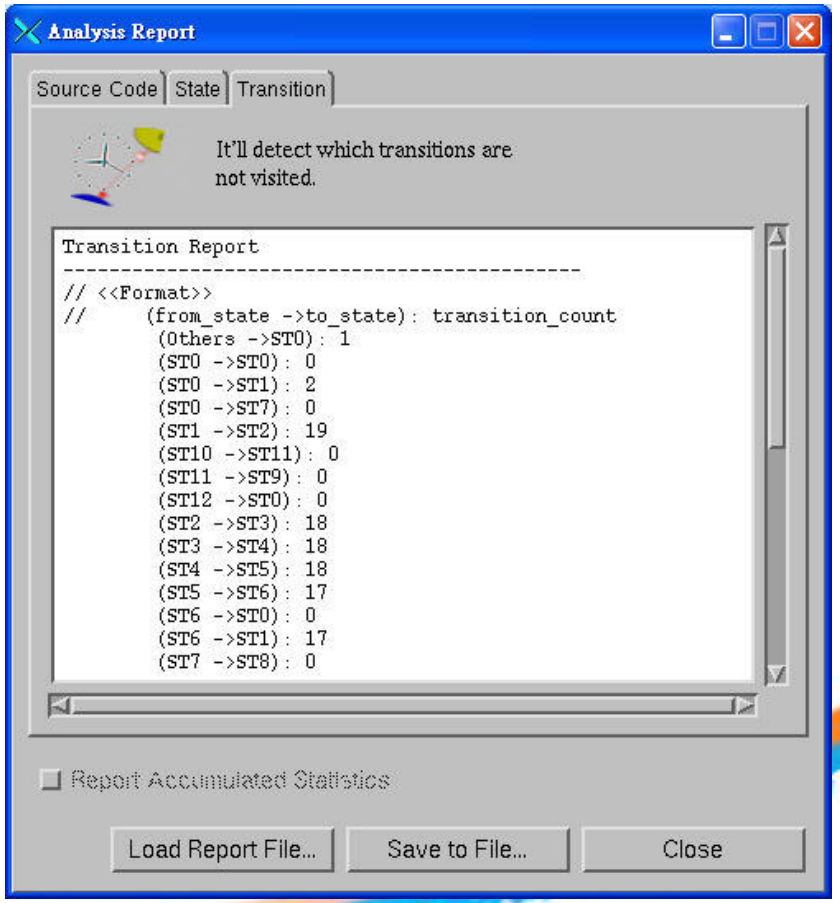
State Animation (2/2)



Analyze State Machines

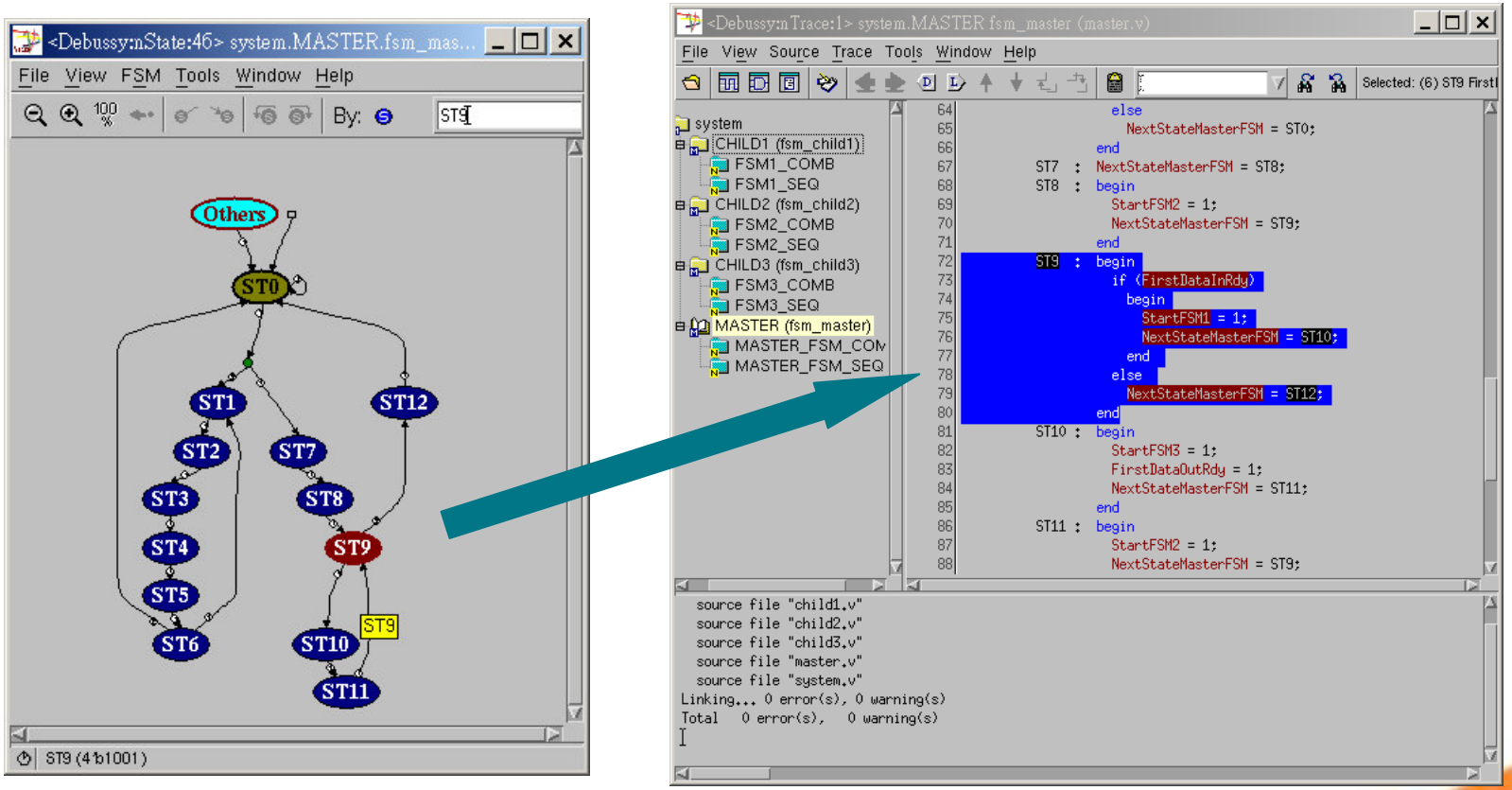
- Use **FSM → Analysis Report** to display analysis report.
  - Details of FSM interpretation.
  - State coverage report.
  - Transition coverage report.
- **Save to File** to save the report.





Correlate Other Views

- D&D the state or junction into *nTrace.*



Summary

- In this section, you have learned…
  - How to open the state machine window.
  - How to manipulate the view and find states.
  - How to display FSM properties and analyze the FSM.
  - How to create a partial FSM.
  - How to use state animation.
  - How to correlate other views.

Lab

- Refer to the *Verdi User's Guide and Tutorial* document in <Verdi_install>/doc/tutorial.pdf.
- Follow the steps in the *nState Tutorial* chapter to practice the lab.

Topics

- Technology Background
- Set Up the Environment
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM View
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

Objectives

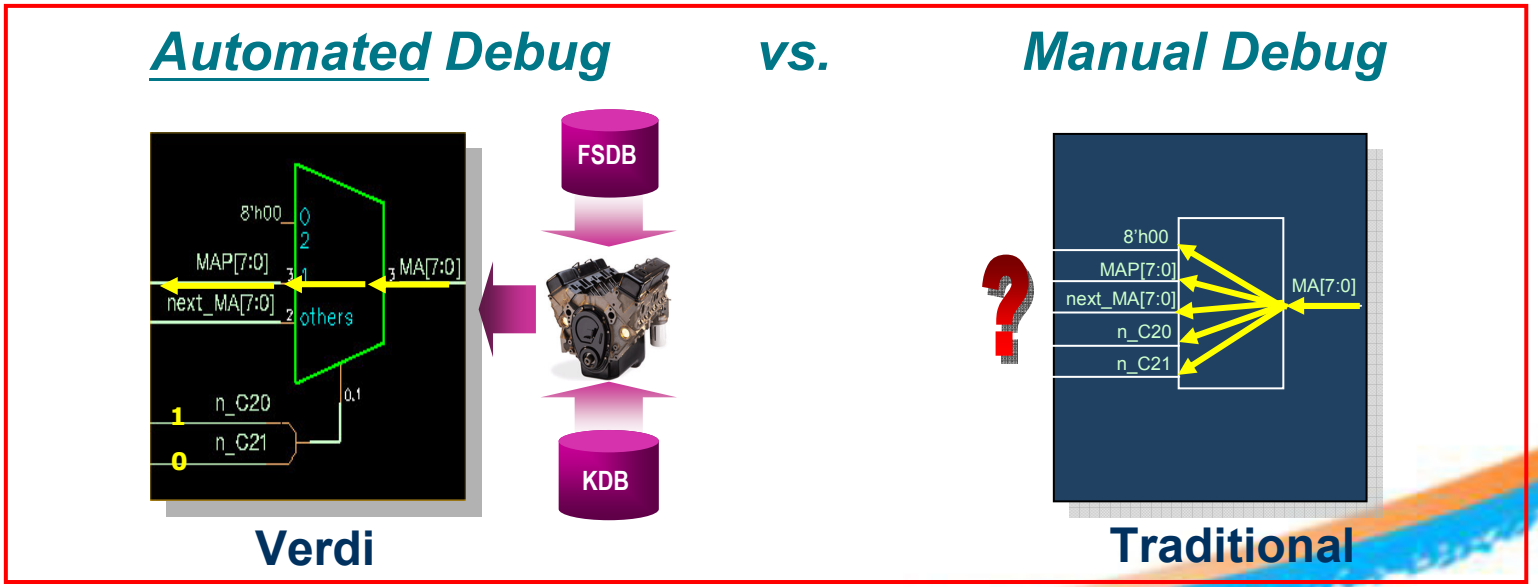
Debug in Temporal Flow View

- After completing this section, you should be able to…
  - Understand what is Temporal Flow View
  - Open a Temporal Flow View (TFV) Window
    - Cycle Based
    - Transition Based
  - Operate in the TFV
  - Correlate Other Views

What is Temporal Flow View?

Behavior Analysis Engine

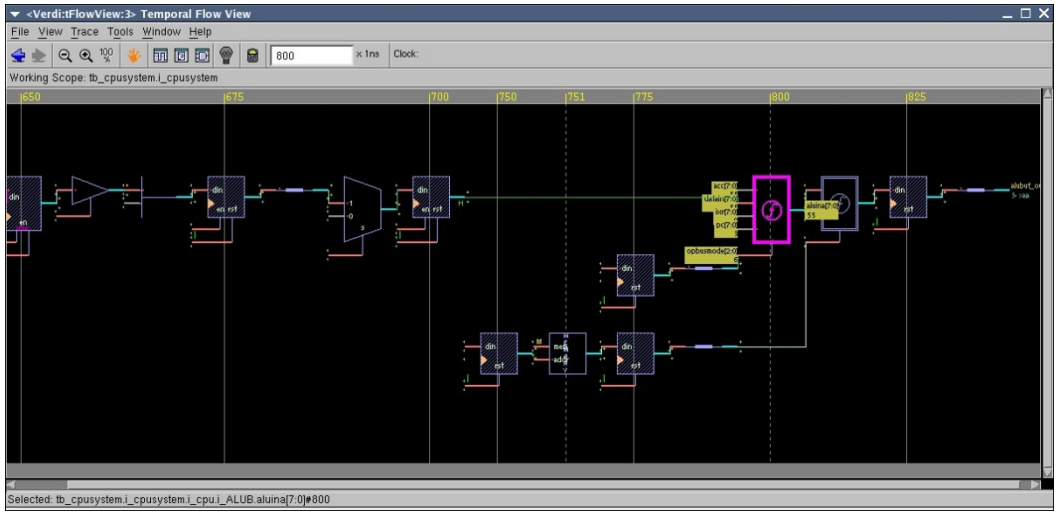
- From the KDB & FSDB, Verdi builds an internal model of actual design behavior using synthesis & formal technology
  - Unroll function over time
  - Differentiate data from control signals
- Uses simulation results
  - Determine clocking
  - Prune inactive elements

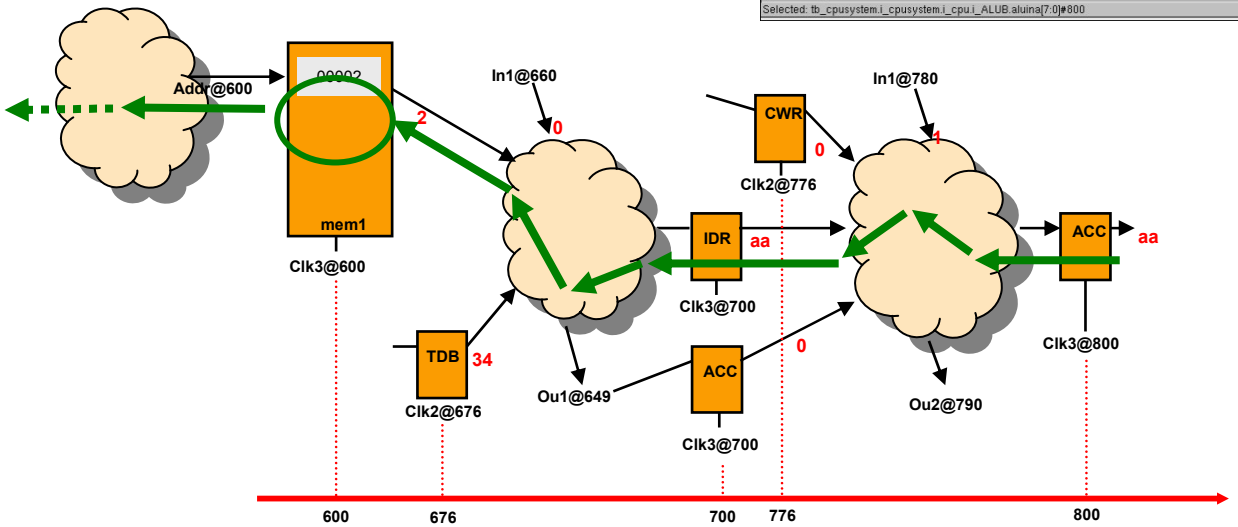


What is Temporal Flow View?

Automatic Cause & Effect Tracing

- Complete, easy to use environment
- Easy visualization of behavior with time and structure

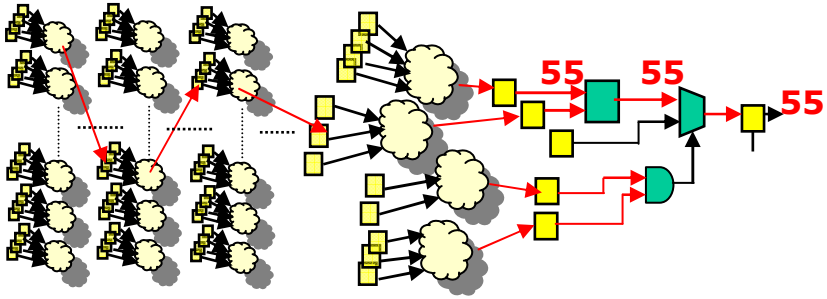




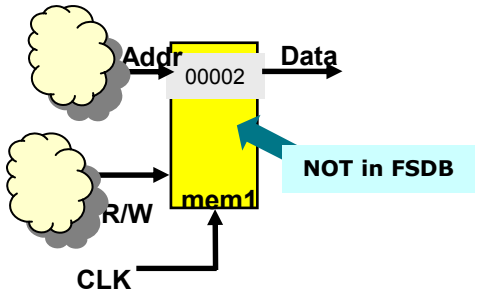
What is Temporal Flow View?

Key Applications of Temporal Flow View Debugging

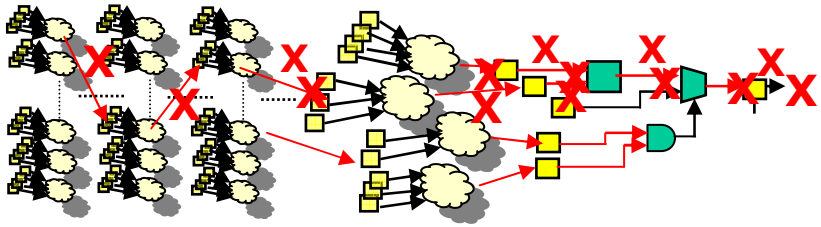
- Understand design behavior and locate the **root cause of a wrong value**



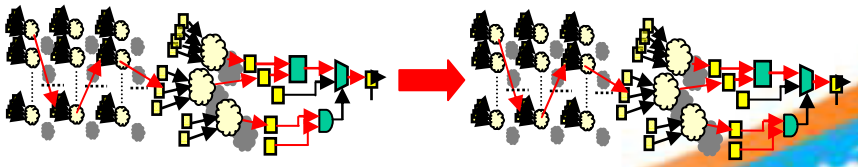
- **Trace memory** content NOT in FSDB and locate memory write



- Locate the root cause of **unknown (X) values**

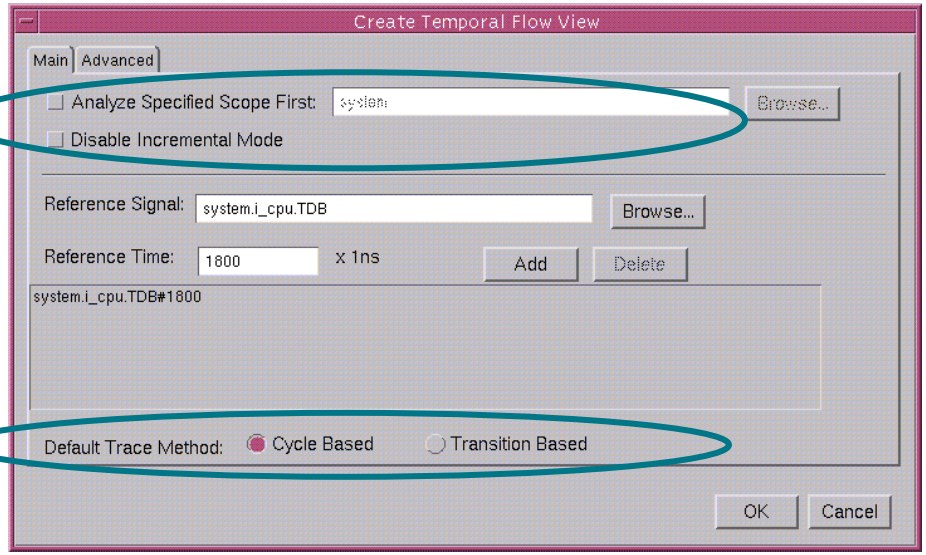


- Locate the **cause of different results** between two simulation runs



Open a Temporal Flow View Window

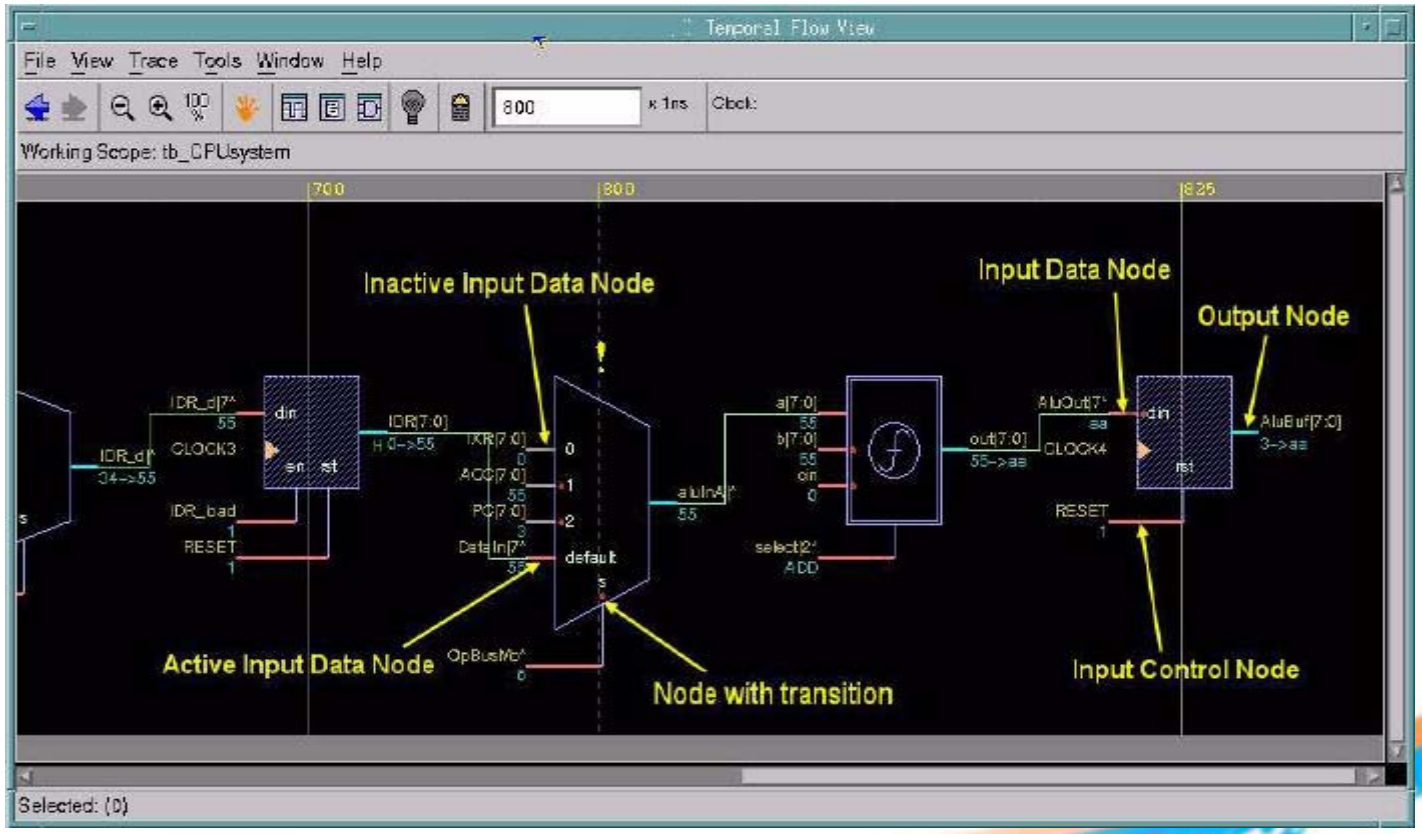
- In *nWave* or *nTrace*, select a signal and invoke RMB → **Create Temporal Flow View**
  - Specify tracing by cycle or transition.
  - Trace incrementally – *Default* and *Recommended.*
    - If **Disable Incremental Mode** and **Analyze Specified Scope First** are turned on, specify the working scope, the time and the signal to trace. Analysis will be completed for everything under working scope.



Open a Temporal Flow View Window

Cycle Based

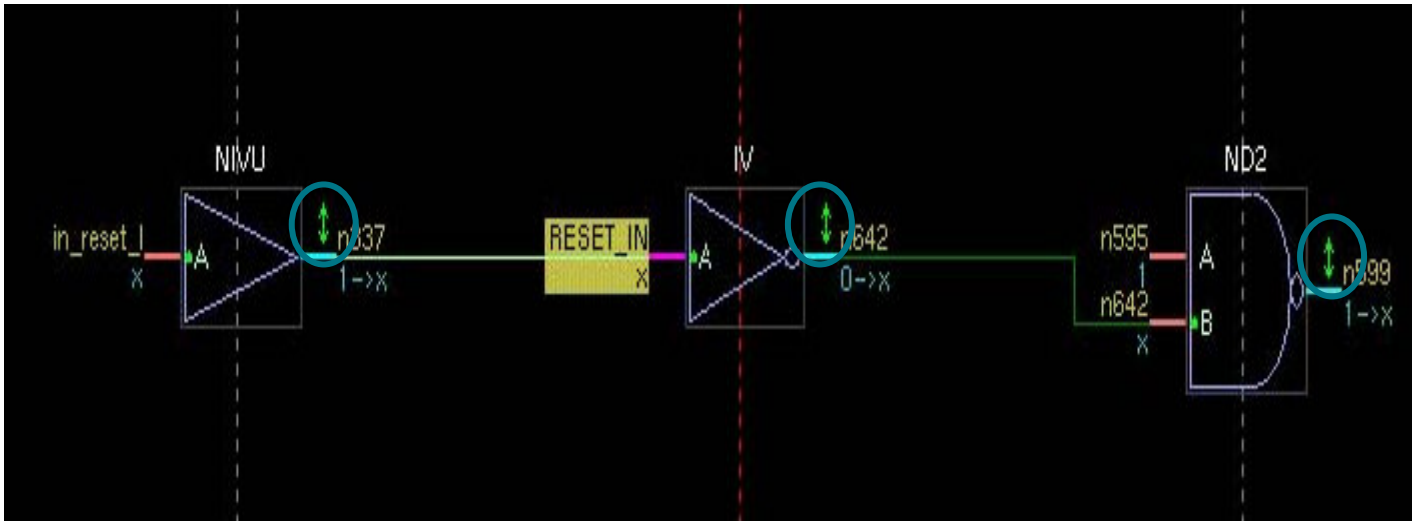
- Structure and time in single view.
- Dynamic analysis finds the active path when tracing.
- Each structure corresponds to one statement.



Open a Temporal Flow View Window

Transition Based

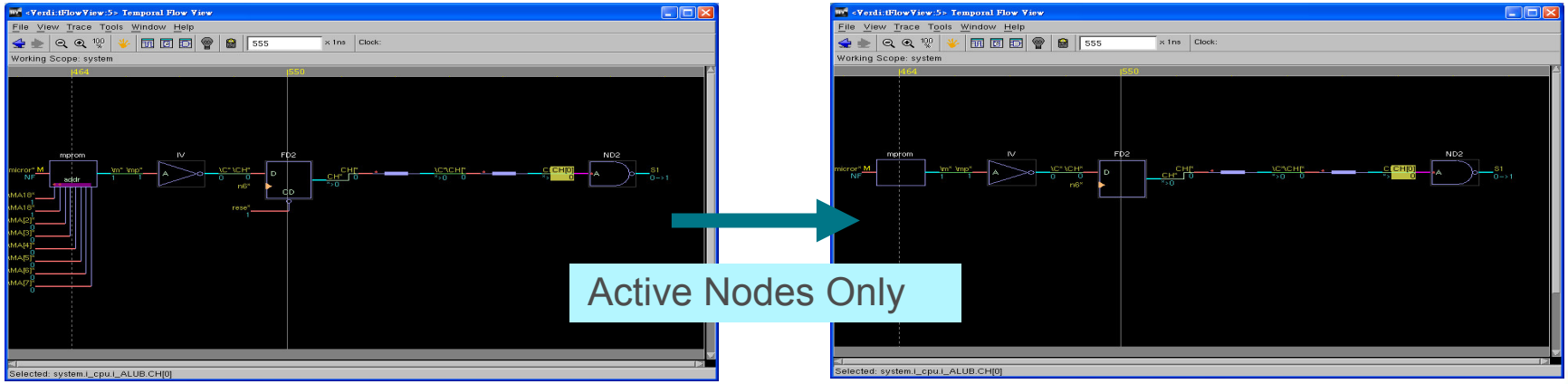
-  distinguishes transition based tracing from cycle-based tracing in the Temporal Flow View.



Operate in the TFV

Trace This Value

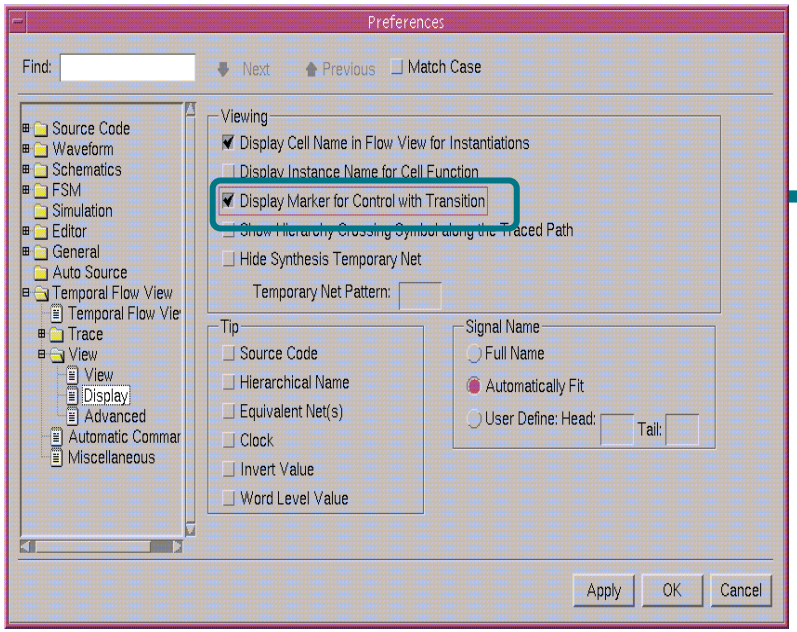
- Use **Trace → Trace This Value** to automatically trace back the data path.
- Use **View → Active Data Nodes Only** or **View → Active Nodes Only** to simplify the view and display active nodes only.
- **DC** to perform Active Trace.

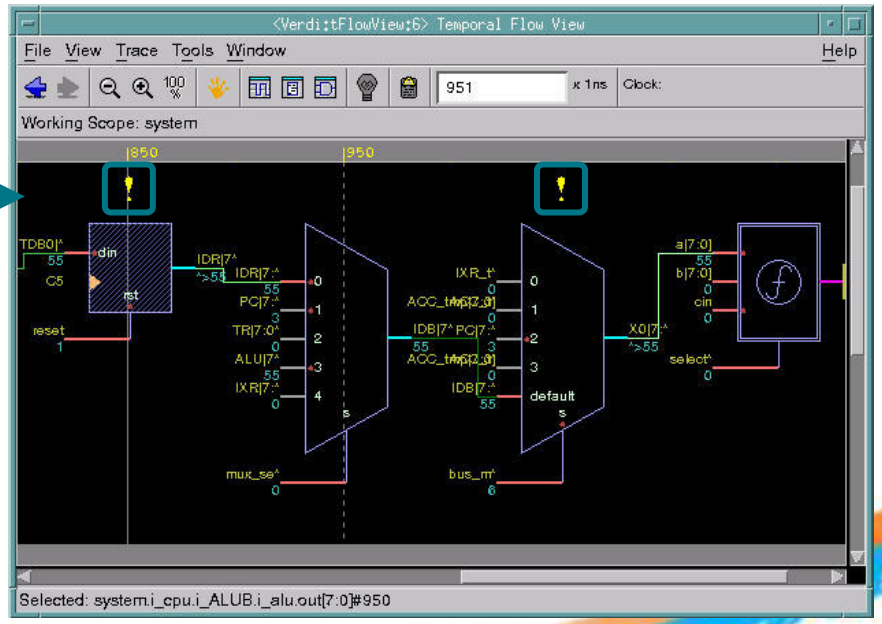


Operate in the TFV

Show Marker for Control with Transition

- Show the output nodes with transition in the tracing path.
  - Enable the **Display Marker for Control with Transition** option on **Temporal Flow View → View → Display** folder of the *Preferences* form (invoked with **Tools → Preferences**).
  - A yellow exclamation point is placed over output nodes.

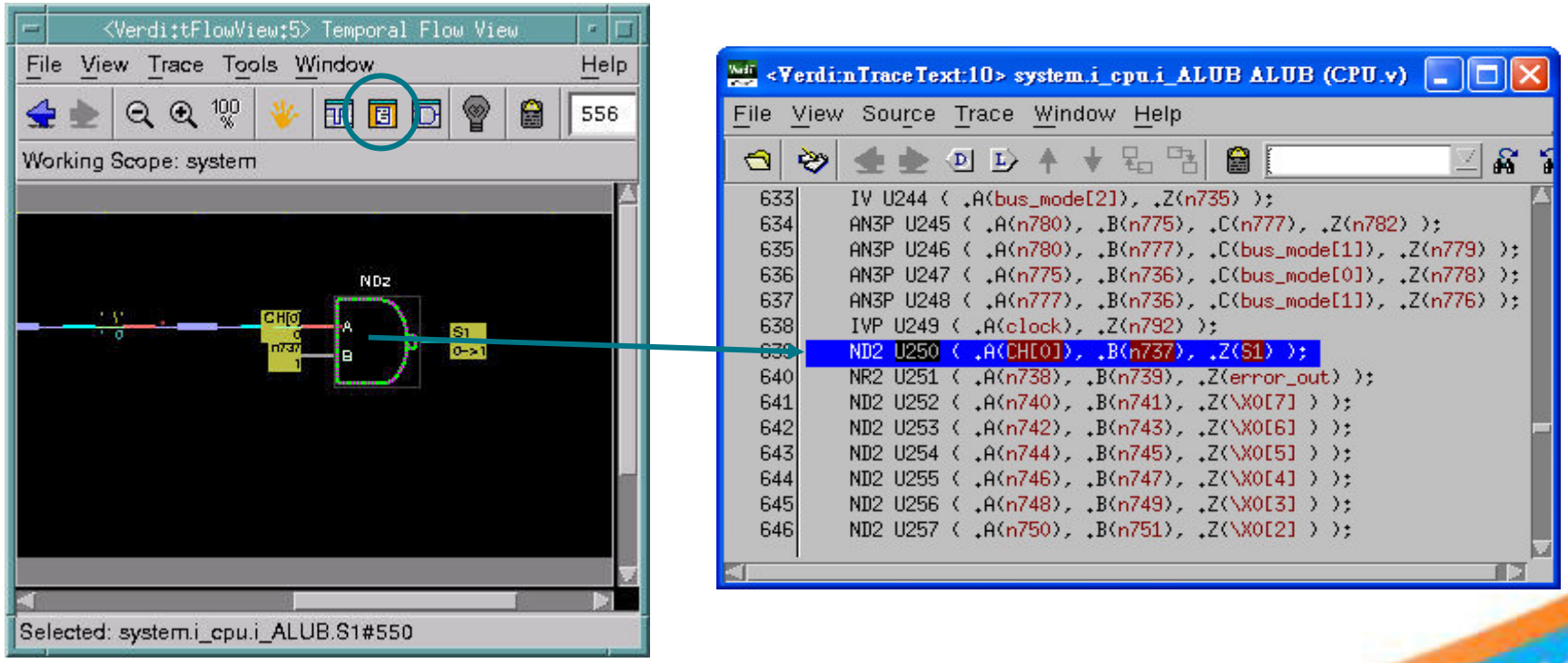




Correlate Other Views

Automatically Display Source Code

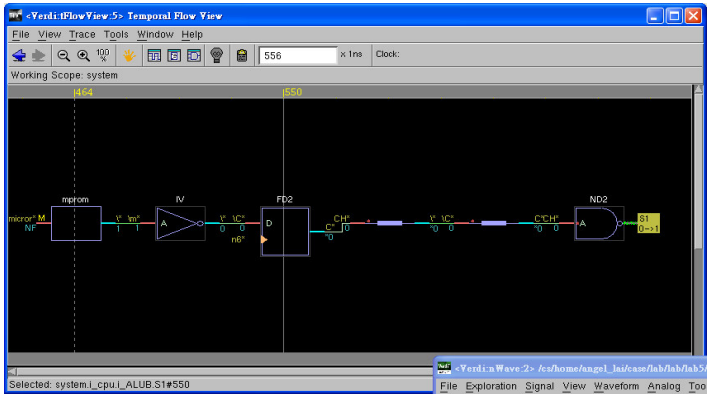
- Enable **Show Source Code Automatically** icon  or use **View** → **Show Source Code Automatically** to view corresponding source code for a selected node.

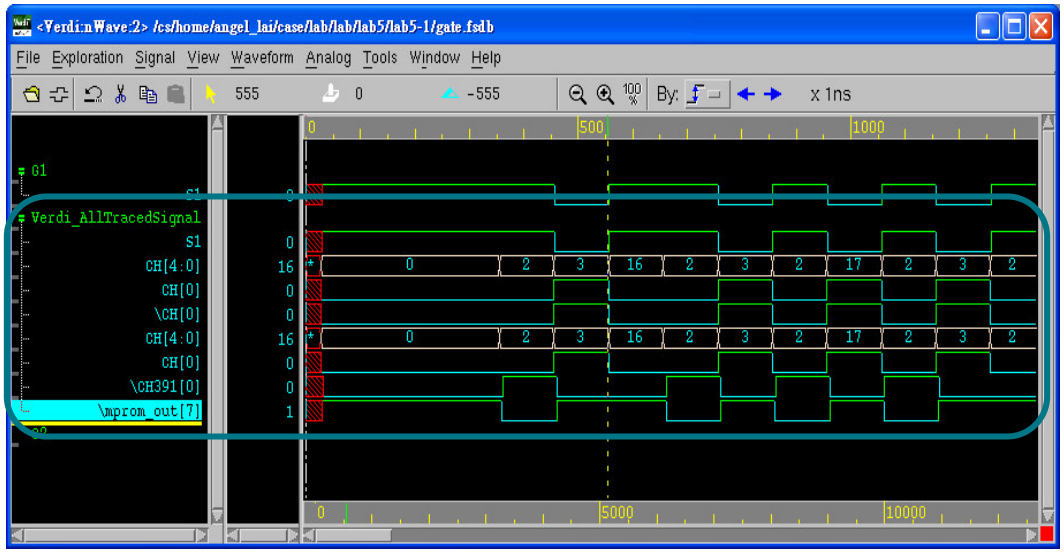


Correlate Other Views

Show Traced Signals in nWave

- To show traced signals in waveform view:
  - Select a signal.
  - Use **Tools → Show All Traced Signals on nWave**.

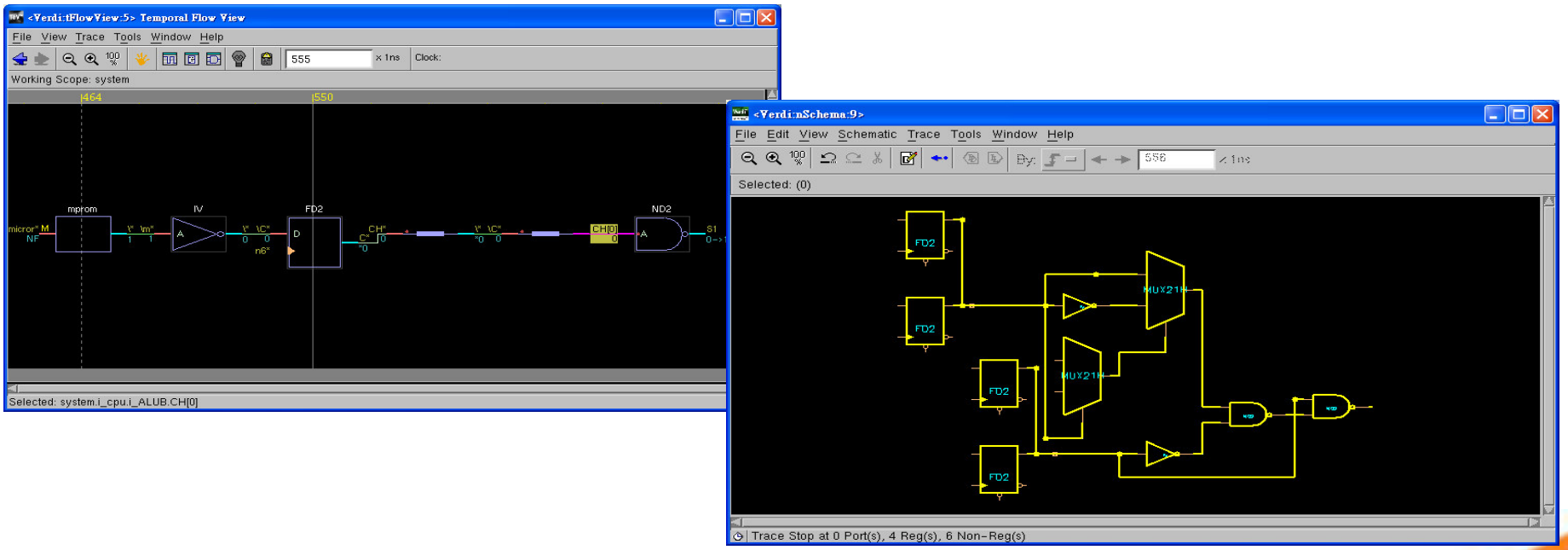




Correlate Other Views

Show Fan-in Signals in nSchema

- To view a signal's fan-in in *nSchema*:
  - Select the signal.
  - Use **Tools → Show Fan-ins on nSchema → Active Only**.



Summary

- In this section, you have learned…
  - How to create the Temporal Flow View (TFV) window.
  - How to operate in the TFV.
  - How to correlate other views.

Lab

- Refer to the *Verdi User's Guide and Tutorial* document in <Verdi_install>/doc/tutorial.pdf.
- Follow the steps in the *Temporal Flow View Tutorial* chapter to practice the lab.

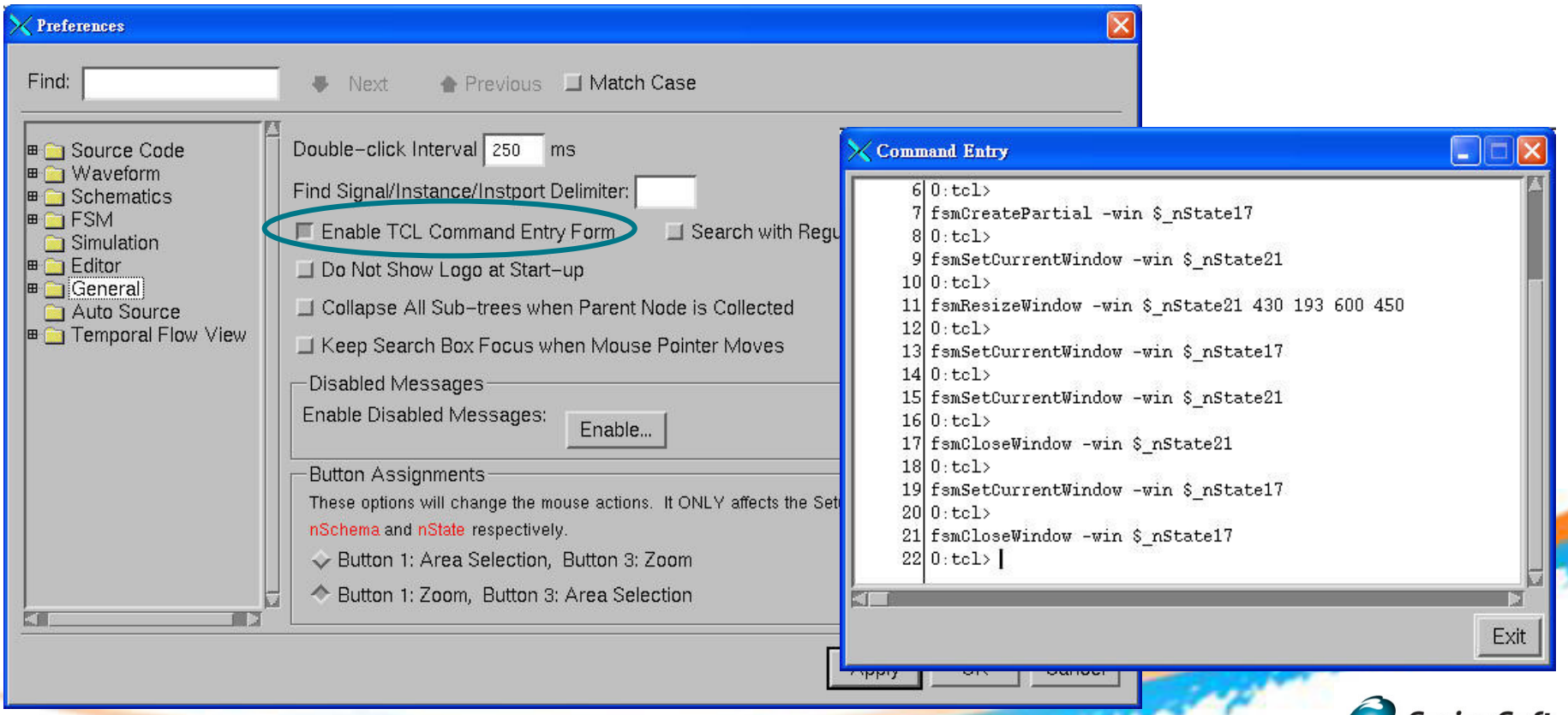
Topics

- Technology Background
- Set Up the Environment
- Import Design
- Debug in Source Code View
- Debug in Waveform View
- Debug in Schematic View
- Debug in FSM View
- Debug in Temporal Flow View
- Appendix: Frequently Used Preferences

Frequently Used Preferences

Enable the TCL Command Entry Form

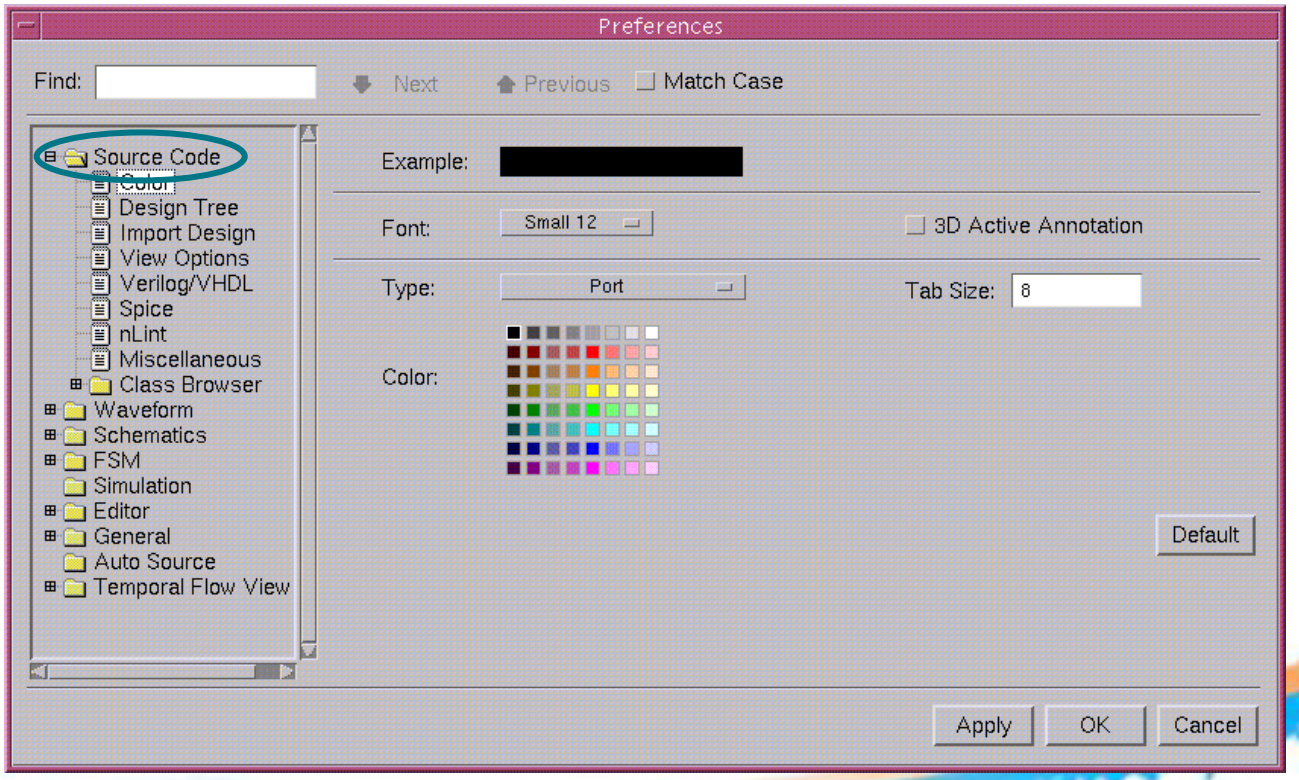
- In the *Preferences* form, select the **General** folder in the browser.
  - Toggle on **Enable TCL Command Entry Form** option.
    - Input TCL commands in the Command Entry form.
    - When you do any action in Verdi, all TCL commands will be logged here.



Frequently Used Preferences

Source Code Related

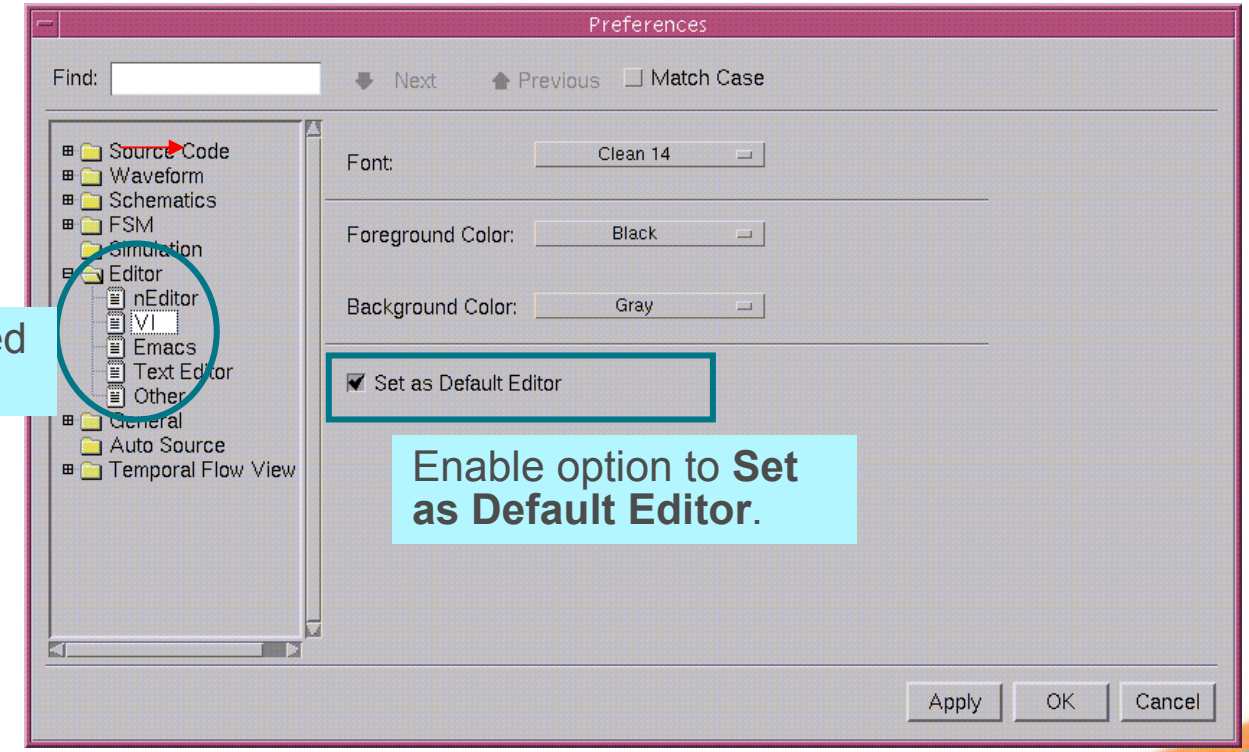
- In the *Preferences* form, select the **Source Code** folder in the browser
  - Select pages under the folder, for example **Color**, **Design Tree,** etc., to customize the appearance and fonts for the hierarchy browser and source code panes.



Frequently Used Preferences

Source Code Related – Specify Default Editor

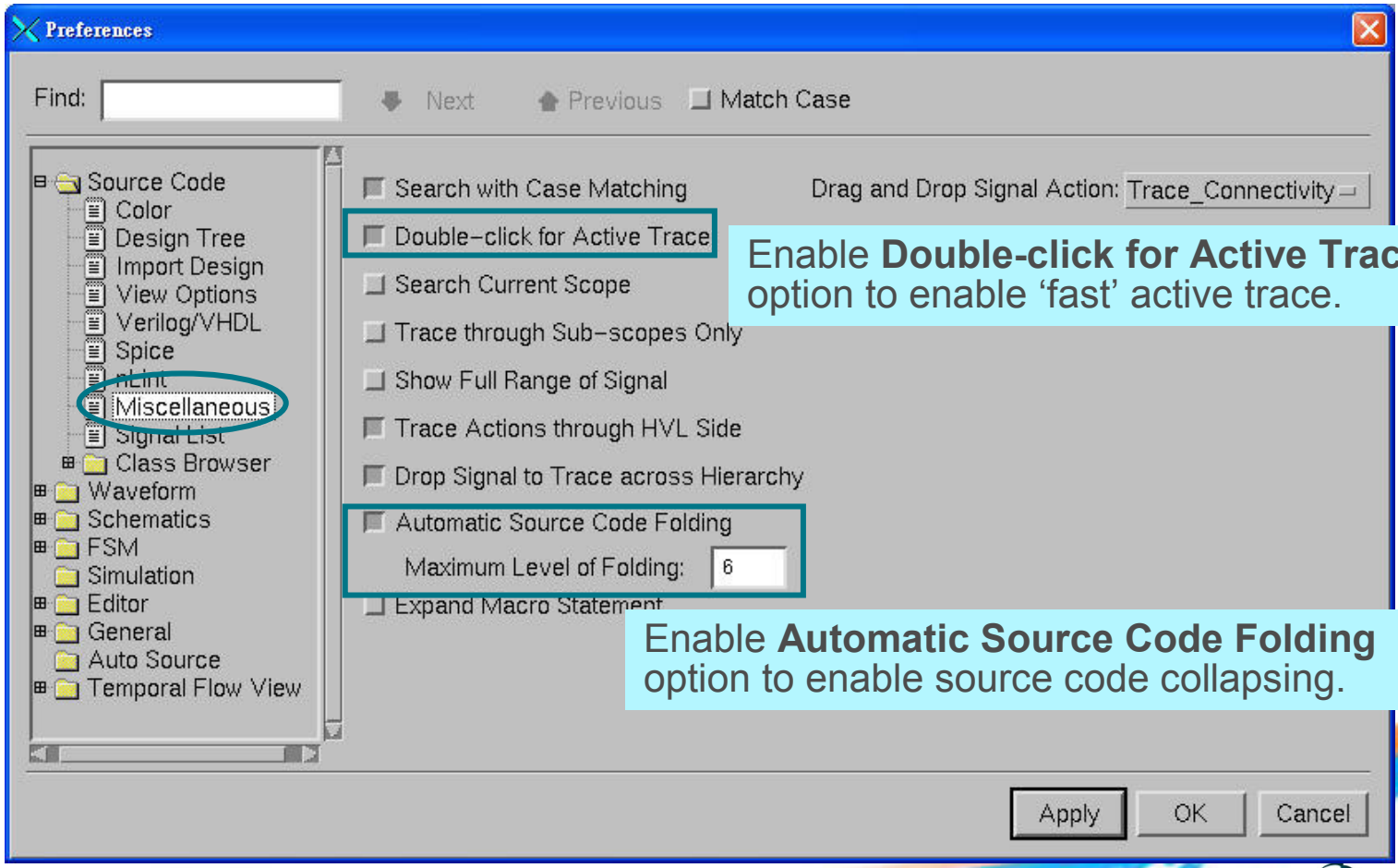
- In the *Preferences* form, select the **Editor** folder to select your preferred editor which is invoked with the toolbar icon 



Frequently Used Preferences

Source Code Related – Enable Active Trace/Folding Source Code

- In the *Preferences* form, select the **Miscellaneous** page under the **Source Code** folder.



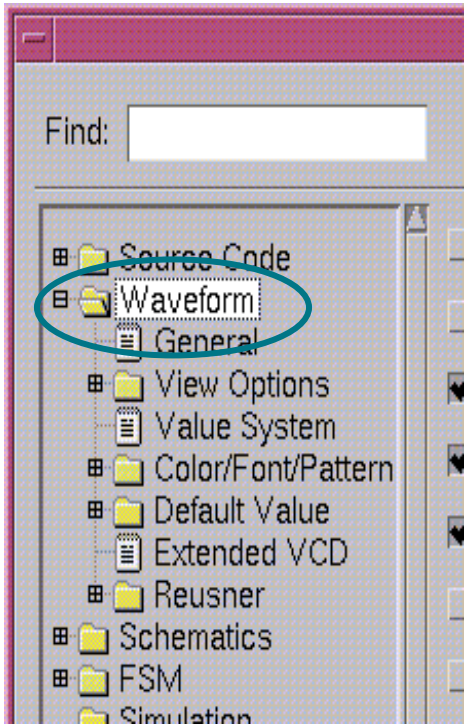
Enable **Double-click for Active Trace** option to enable 'fast' active trace.

Enable **Automatic Source Code Folding** option to enable source code collapsing.

Frequently Used Preferences

Waveform Related

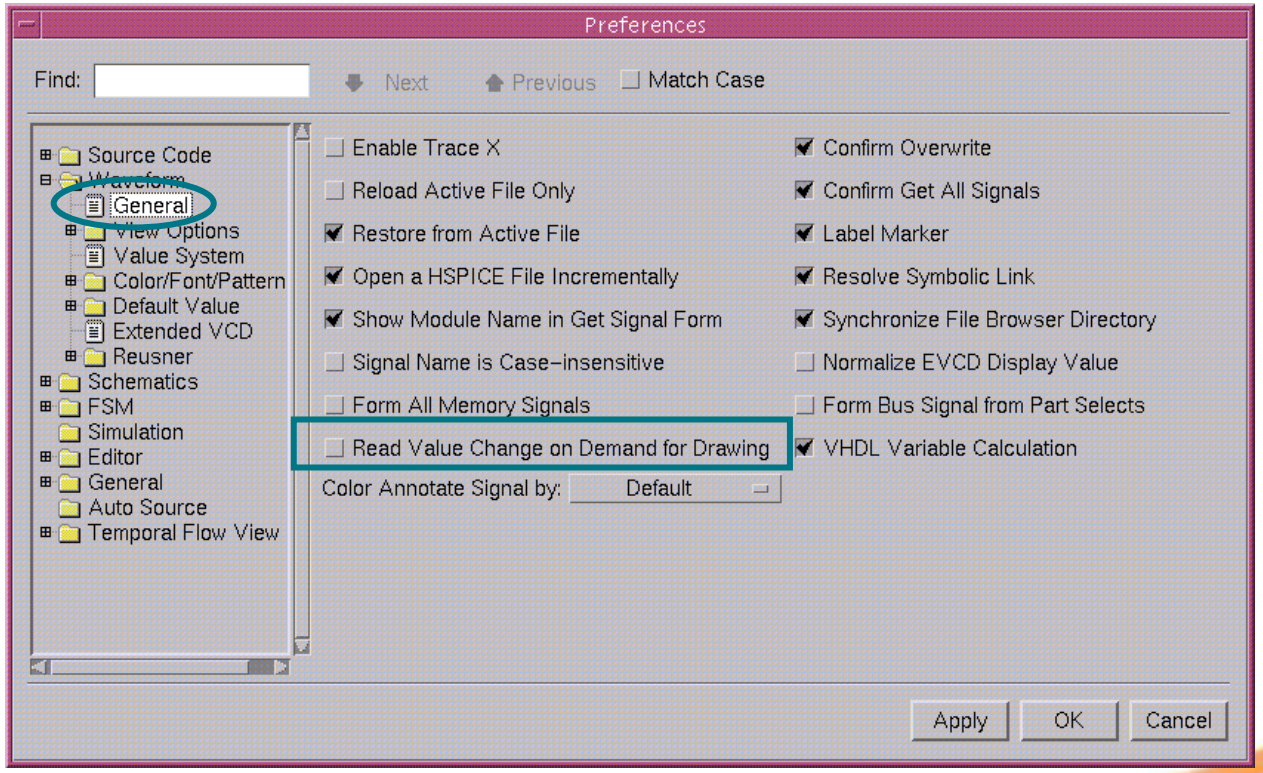
- In the *Preferences* form, select the **Waveform** folder to customize waveform display.
  - **General** page: Display input/output signals in different colors, synchronize file browser, etc.
  - **View Options** folder: Enable highlight signal, radix display, value and signal alignment.
  - **Value System** page: Change appearance based on values and strengths.
  - **Color / Font / Pattern** folder: Change appearance of windows and signal types.
  - **Default Value** folder: Change signal height and file filter.
  - **Extended VCD** page: Change appearance based on values and strengths.



Frequently Used Preferences

Waveform Related – Read Value Change on Demand

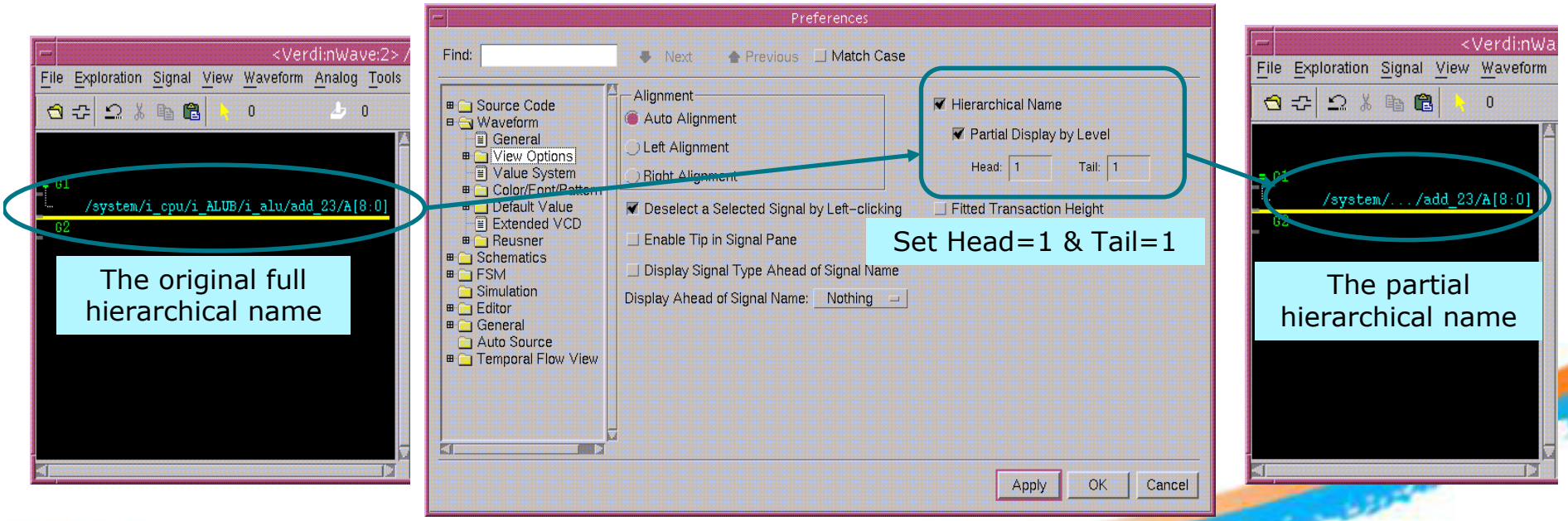
- In the *Preferences* form, select the **Read Value Change On Demand For Drawing** option on the **General** page under the **Waveform** folder to reduce memory usage by reading only when signal changes value.



Frequently Used Preferences

Waveform Related - Partial Display Hierarchical Signal Name

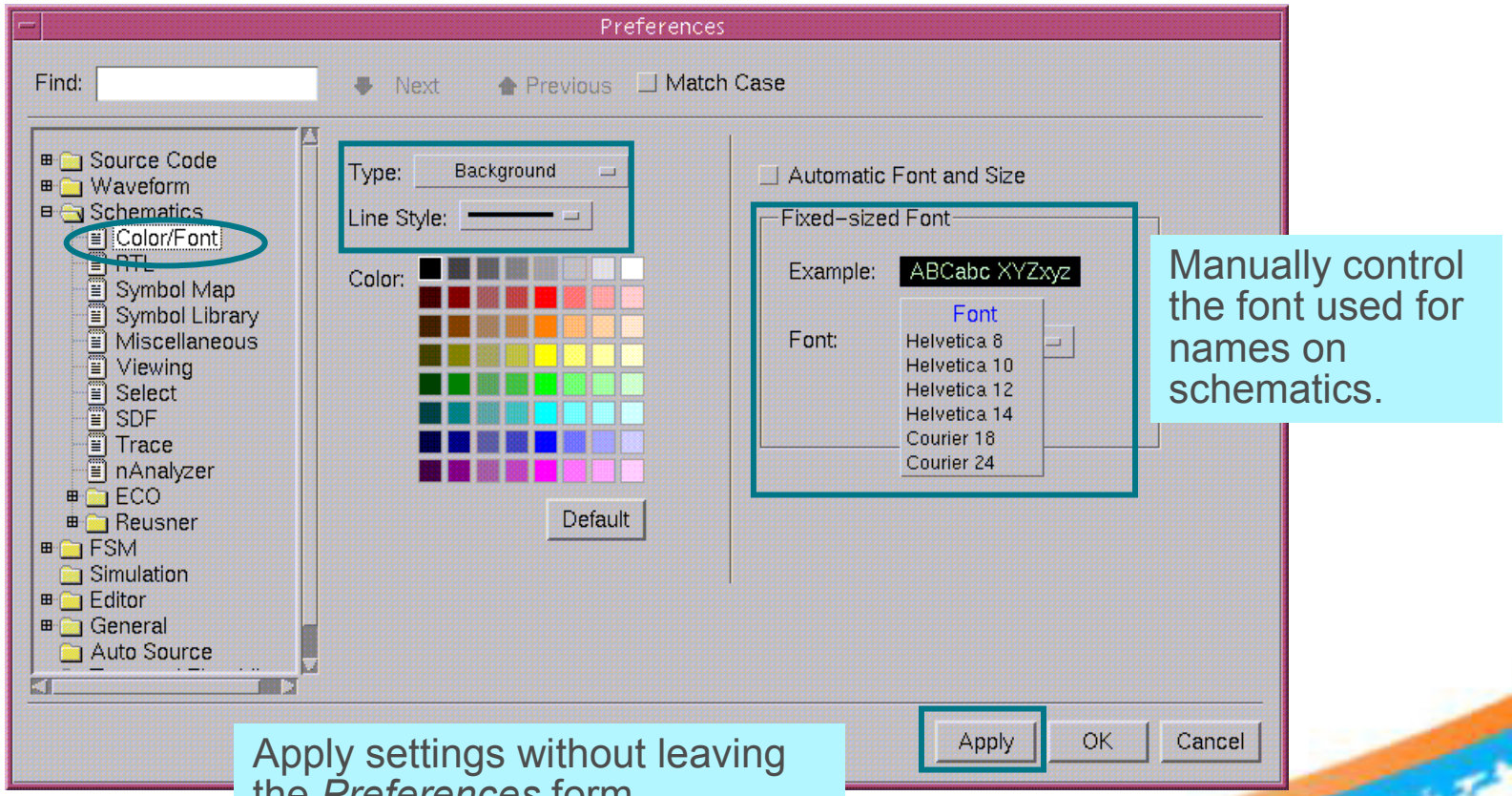
- Turn on the **Hierarchical Name** option and **Partial Display by Level** sub-option on **Waveform → View Options** page of *Preferences* form.
  - Specify the hierarchy level for **Head** and **Tail**.
    - Example: set **Head=1** and **Tail=1**, the signal becomes: **/system/…/add_23/A[8:0]**
  - Flexible for user to focus level and avoid long hierarchy names.



Frequently Used Preferences

Schematic Related – Customize Color and Font

- In the *Preferences* form, select the **Color/Font** page under the **Schematics** folder to customize the appearance globally.



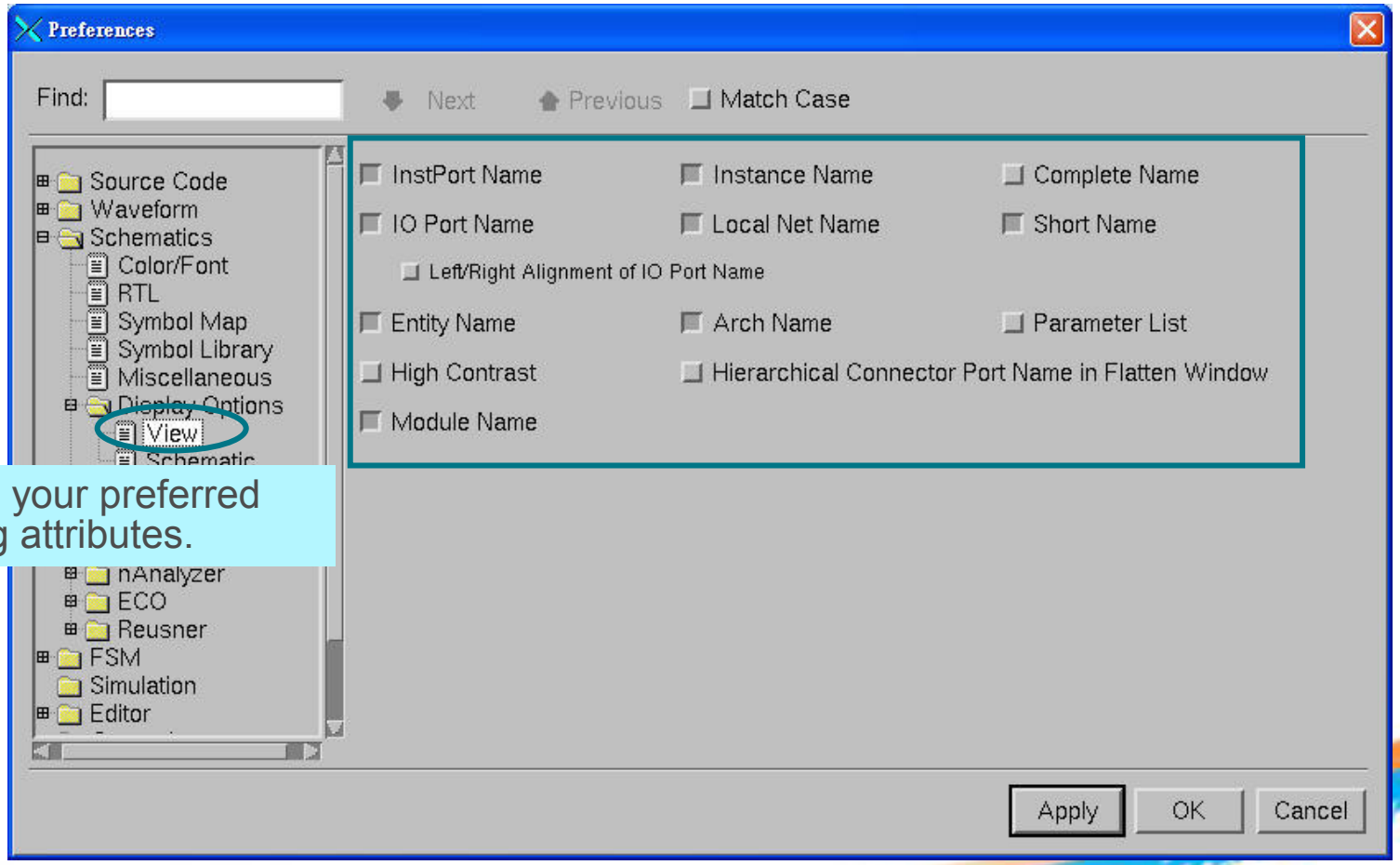
Manually control the font used for names on schematics.

Apply settings without leaving the *Preferences* form.

Frequently Used Preferences

Schematic Related – Customize Viewing Attributes

- In the *Preferences* form, select the **Viewing** page under the **Schematics** folder to change viewing attributes in all windows.

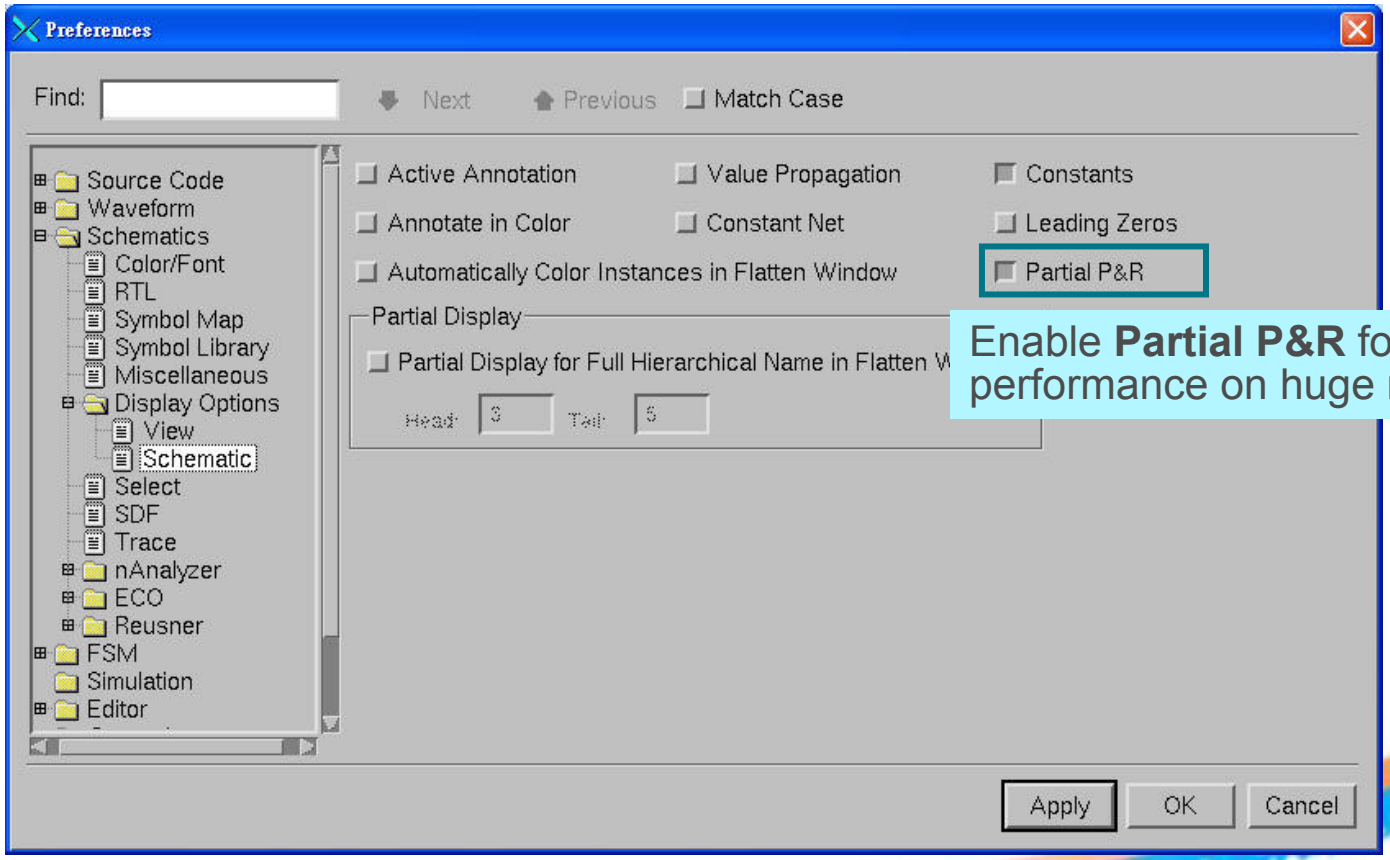


Enable your preferred viewing attributes.

Frequently Used Preferences

Schematic Related – Partial P&R

- In the *Preferences* form, select the **Schematic** page under the **Schematics** folder, turn on **Partial P&R** for improving the drawing performance
  - Incrementally performing Place & Route. Helpful for huge gate-level netlist



Enable **Partial P&R** for improved performance on huge netlists.

Frequently Used Preferences

FSM Related

- In the *Preferences* form, select the **Color** page under the **FSM** folder to customize the appearance globally.