Debug Workshop

Dramatically enhance your debugging efficiency 2010

What is Debug Workshop?

- We will point out typical bug scenarios that designers often encounter and then introduce the most powerful debugging features in Verdi/nAnalyzer/Siloti to help designers solving these bugs
- After lecture and demo, we will offer on-line laboratories to let users get familiar with these features

Debug Workshop Agenda (1/5)

- Trace Signal with Wrong Value
  - Automatically find out the root cause if signal's value goes wrong after simulation
- Trace Unknown Value
  - Automatically find out the root cause of a signal with unknown value

Debug Workshop Agenda (2/5)

- Import Design Faster
  - Shorten the time to import huge design
  - Only import some sub-blocks but mapping to the whole-chip simulation results
- Debug Memory
  - Calculate memory content without dumping and trace its written time
- Compare Simulation Waveforms
  - Find the difference between 2 FSDBs
  - Automatically find the root cause of a mismatch

Debug Workshop Agenda (3/5)

- Reduce Simulation time and FSDB Size
  - Use System Tasks
  - Use Virtual FSDB
  - Use Essential Signal Dumping
- Debug Timing Problem after Post-Simulation
  - Debug the path of timing violation in STA report efficiently
  - Automatically find out the root cause of wrong transition caused by timing issue
  - Make the process of detecting, isolating, and fixing problems much more efficient

Debug Workshop Agenda (4/5)

- Debug SystemVerilog Design
  - Unify environment for debug and analysis of different languages (i.e. Verilog, VHDL, SystemVerilog, etc.)
  - Easily browse SystemVerilog code
- Assertion-Based Verification by SVA
  - Start from the assertion failure on waveforms and trace back to assertion codes or HDL sources
  - Check new or modified assertions without re-running simulation
- Analyzer SystemVerilog Testbench (SVTB)
  - Understand SVTB by Testbench Browser
  - Log Messages into FSDB for Debugging SVTB

Debug Workshop Agenda (5/5)

- Debug Design with Power Intent CPF/UPF Specification
  - Understand power specification of CPF/UPF
  - Debug power aware simulation failures

Trace Signal with Wrong Value

- Common Problems
  - Must open multiple source files and correlate signals in waveform to judge real driver
  - Easily get lost in a huge unfamiliar design
- SpringSoft Solution
  - Use Active Trace with Active Annotation in source code and waveform
  - Debug on Verdi's Temporal Flow View by Trace This Value

Trace Signal with Wrong Value
Active Trace(1/2)

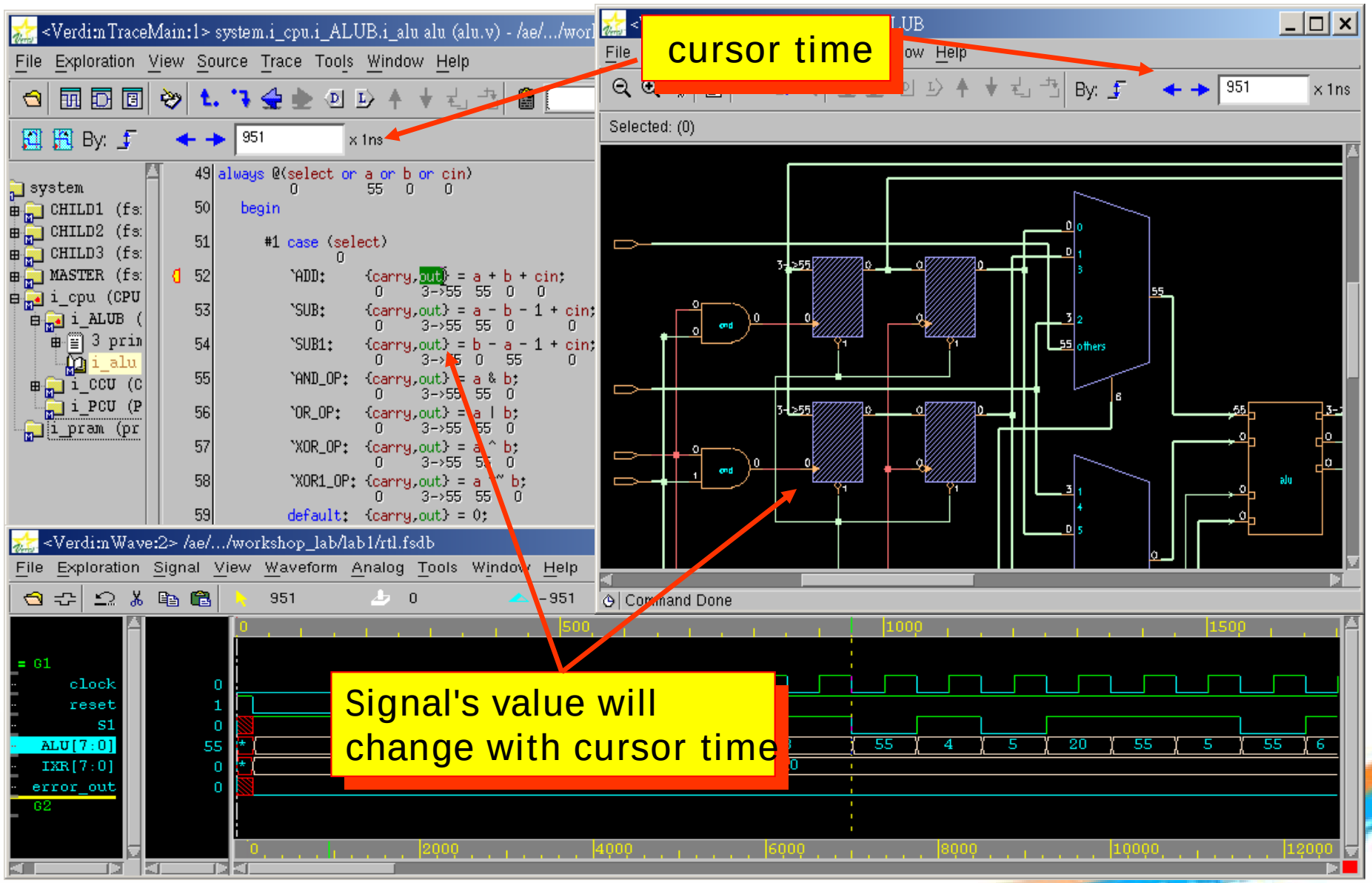
- Purpose
  - Refer to simulation result to catch the real driver for specific signal at a given time
- Usage
  - Import design with FSDB
  - Do Active Trace either by
    - ***Select interested signal and set cursor to the wrong value*** in waveform window
    - ***Double click at this wrong value***
  - Or ***select signal and do Active Trace with RMB*** (right mouse button) in source code window
- Verdi will highlight the active driving statement in source code window

Trace Signal with Wrong Value
Active Trace(2/2)

Trace Signal with Wrong Value
Active Annotation(1/2)

- Purpose
  - Annotate simulation result in source code or schematic window to view signal value with design context
- Usage
  - *nTrace → File → Load simulation result(FSDB)*
  - *nTrace → Source → Active Annotation or nSchema → Schematic → Active Annotation*
- Signal's value will change with cursor time

Trace Signal with Wrong Value
Active Annotation(2/2)



SpringSoft
Accelerating Engineers

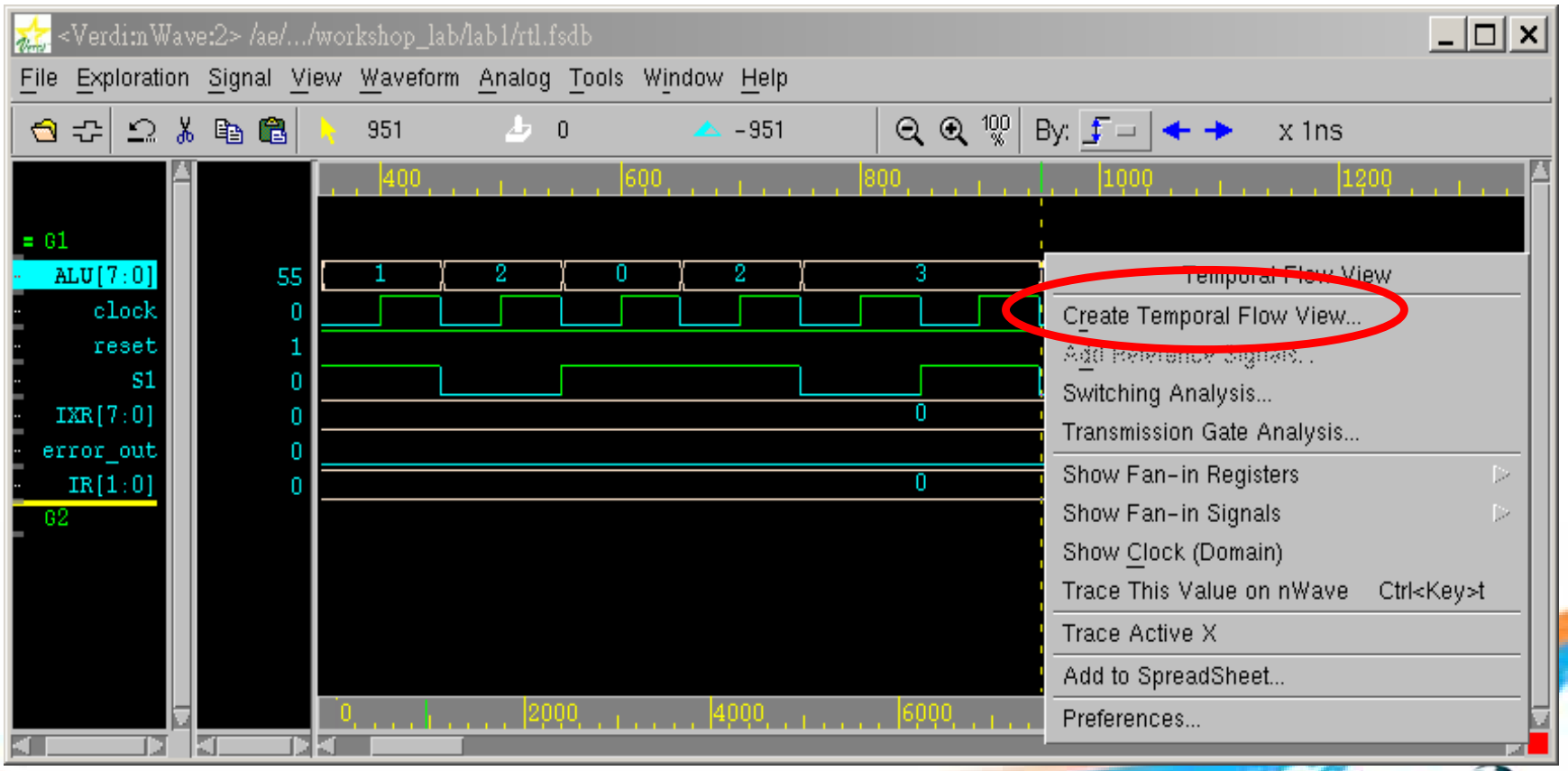
Trace This Value Automatically

- Flow View
  - Add the signal with the bad data value to the waveform.
  - View relevant design behavior over time.
    - Create *Temporal Flow View* for the problem signal
    - Use *Trace This Value* to locate the cause of the specific data value across multiple cycles
  - Visualize structure and source code over time
  - Trace and analyze multiple paths in a single view
- Waveform
  - Add the signal with the bad data value to the waveform.
  - Locate the cause of the bad value and trace the relevant signals in the waveform.
    - Use *Trace This Value* on nWave with RMB
  - Analyze the waveform and source code to find the cause of the bug

Trace This Value Automatically
View Design Behavior Over Time (Verdi)

- Use Create Temporal Flow View on selected signal
  - From nWave, place cursor at the time of interest and invoke RMB menu → *Create Temporal Flow View*

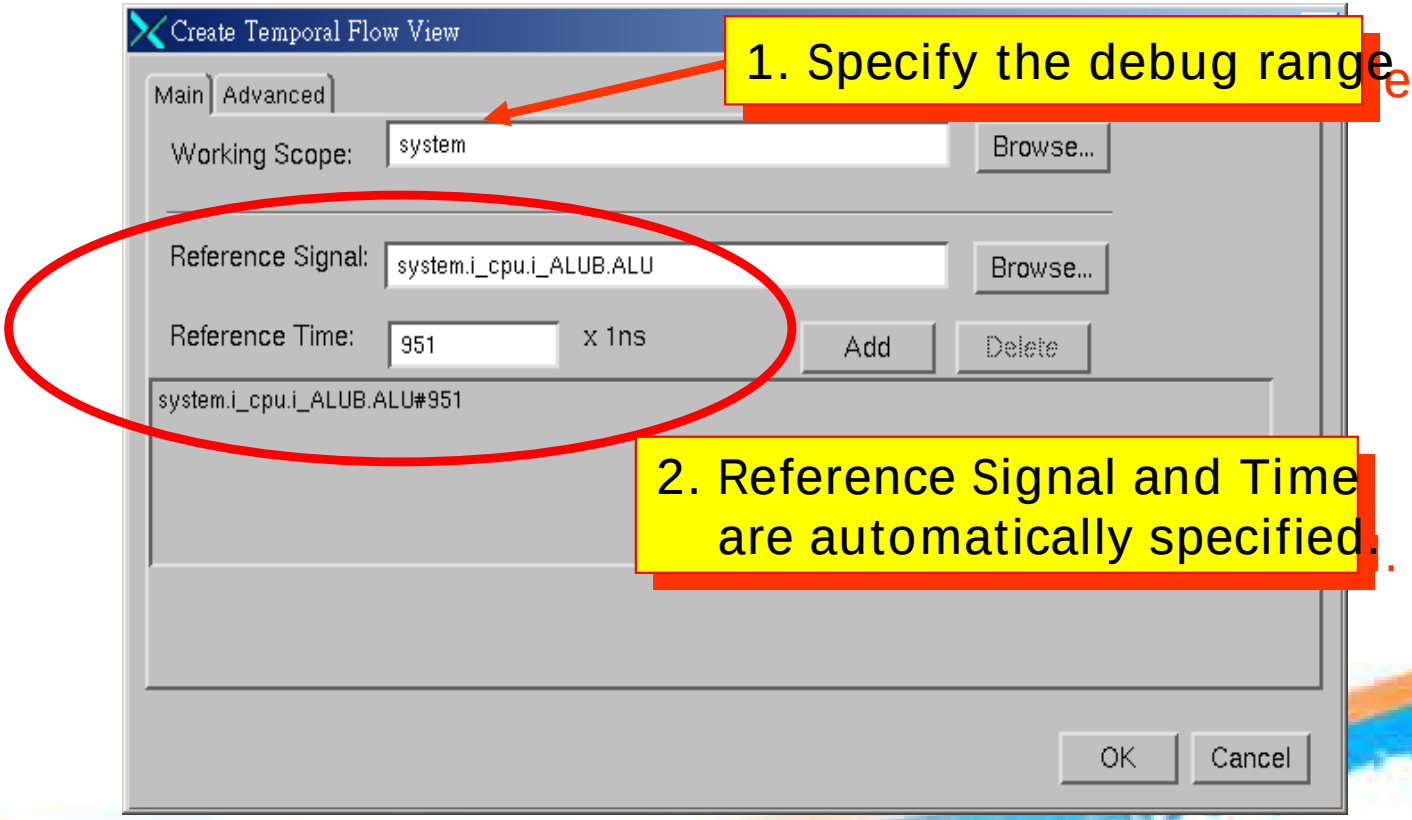
Hint: RMB means Right-Mouse-Button



SpringSoft Accelerating Engineers

Trace This Value Automatically
Create Temporal Flow View Options

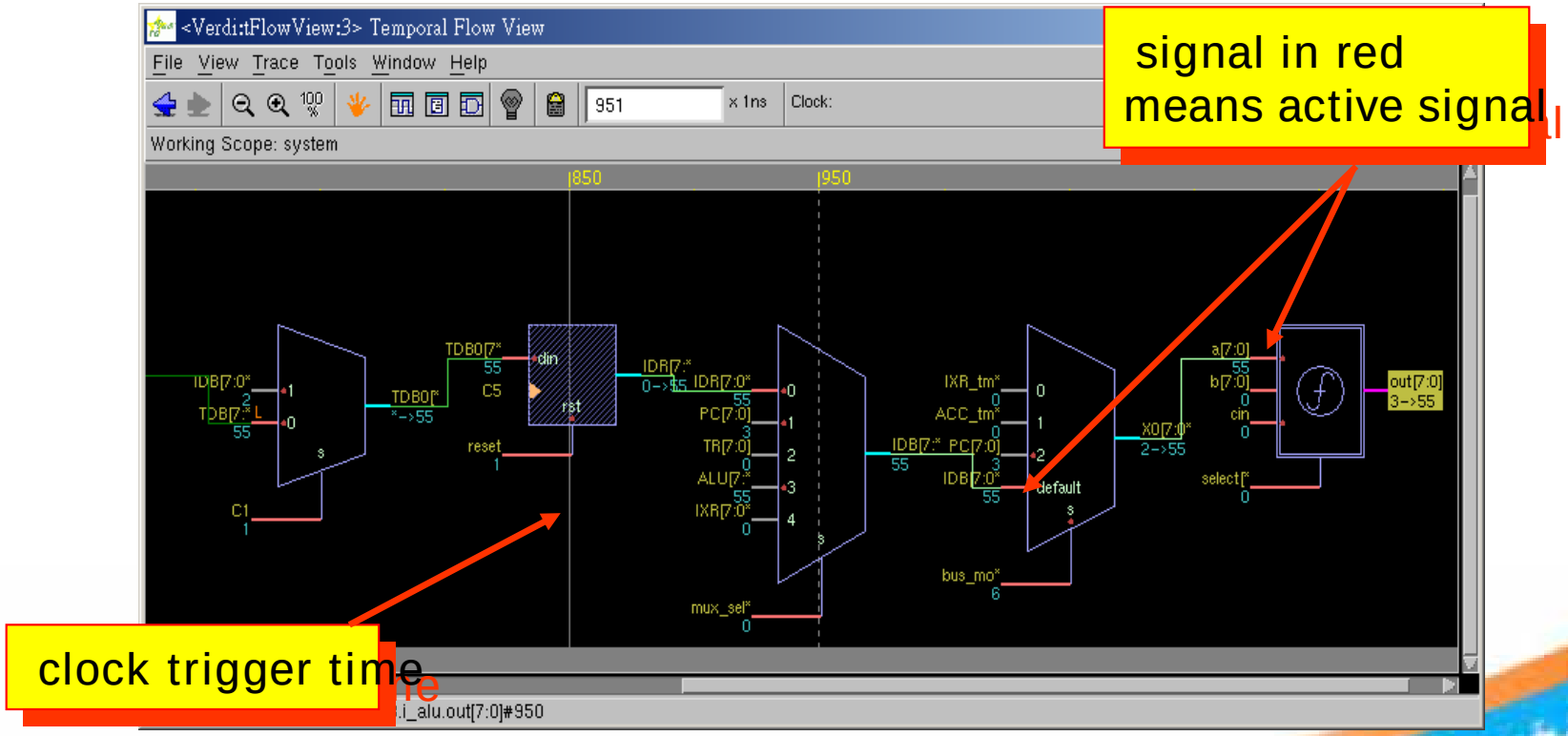
- In the **Create Temporal Flow View** window
  - Confirm the working scope is correct
    - Verdi will perform behavior analysis on this scope and all scopes beneath it
  - Click OK



SpringSoft
Accelerating Engineers

Trace This Value Automatically
Automatically Locate Source of Bad Value

- Use *Trace This Value* to locate the cause of the specific data value across multiple cycles
  - Place the cursor over a signal and invoke *RMB → Trace This Value*
    - This command automatically expands nodes in the Temporal Flow View to trace a value back to its source.



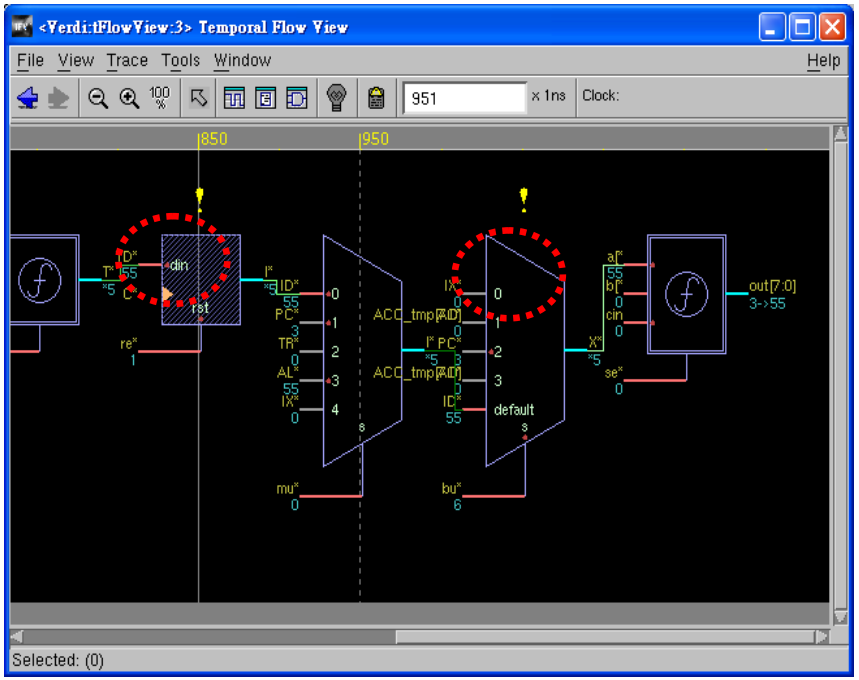
SpringSoft Accelerating Engineers

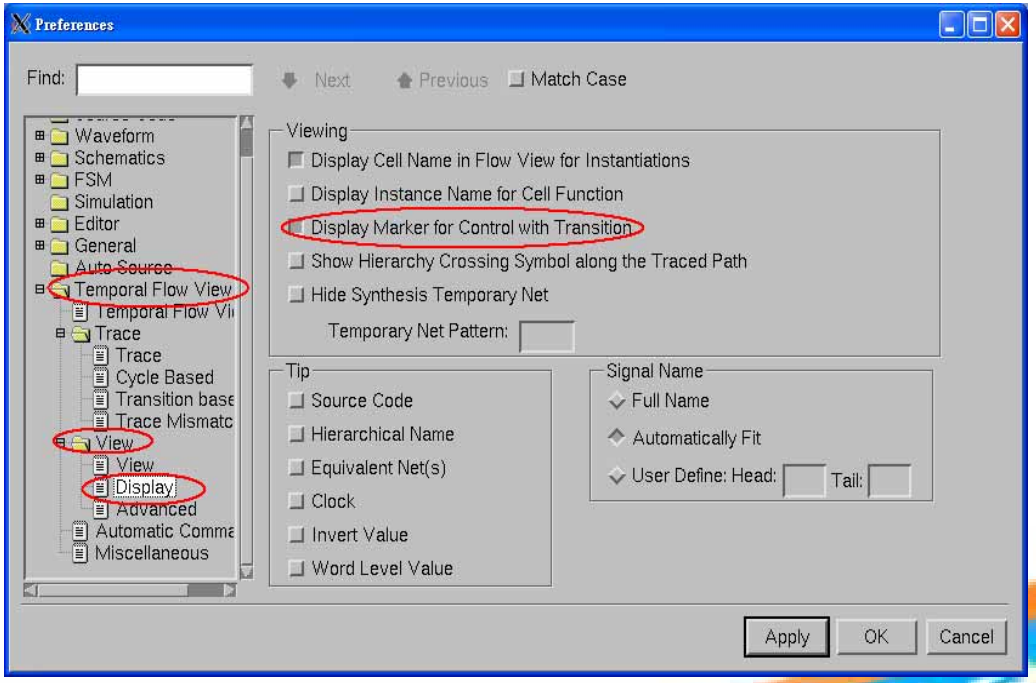
Trace This Value Automatically
Tracing Details

- Trace behavior will base on the options for "trace this value"
- To set the trace option
  - *Tools → Preferences → Temporal Flow View→ Trace→ Cycled Based*
- Trace This Value Options:
  - Trace the same VALUE(s) on data path: Over multiple cycles, trace all active data signals whose value matches the original data value. (Default ON)
    Recommend to turn off for gate-level
  - Continue tracing if only one ACTIVE data path: If there is only one active data signal, continue tracing on the signal even if the value doesn't match the original data value. (Default ON)
  - Continue tracing if only one TRANSITION on active fan-in's: If there are multiple active fan-in signals but only one with a transition, continue tracing on the one with the transition even if it is on a control signal. (Default ON)
  - Stop when CONTROL has transition: Tracing stops when the control signal has a transition. (Default OFF)
    - Note: These options are "OR" together. Whenever one of the conditions is true, tracing continues from the corresponding nodes.
  - Specify Max Level, Ask Every Time or No Limit: Specify the maximum number of levels for tracing. Default is 10 levels.

Trace This Value Automatically
Display Decision Points on the Temporal Flow View

- Place an exclamation mark on the fan-in nodes with transitions on the control signals. Recommend to enable this option!
  - Enable Display Marker for Decision Point under *Tools* → *Preferences* → *Temporal Flow View* → *View* → *Display* → *Display Marker for Control with Transition* (Default is OFF)
  - To clear the marks, disable the option.

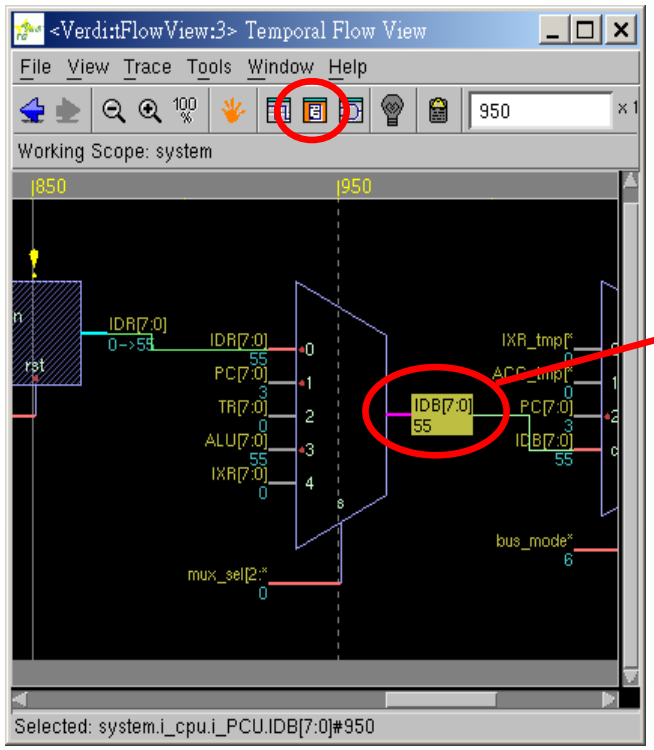


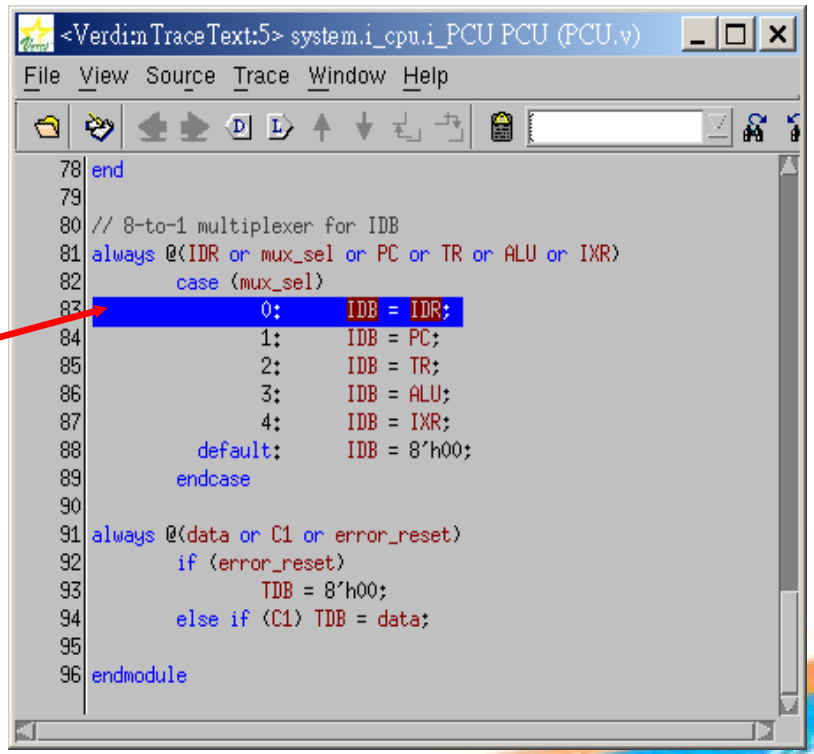


SpringSoft
Accelerating Engineers

Trace This Value Automatically
Visualize Structure and Source Code

- Synchronize the structure and source code
  - Click on the "Enable source code automatically" icon 
  - Once the option is enabled, click each node to show the associated source code.
  - Preference to show on nTrace window or another source window

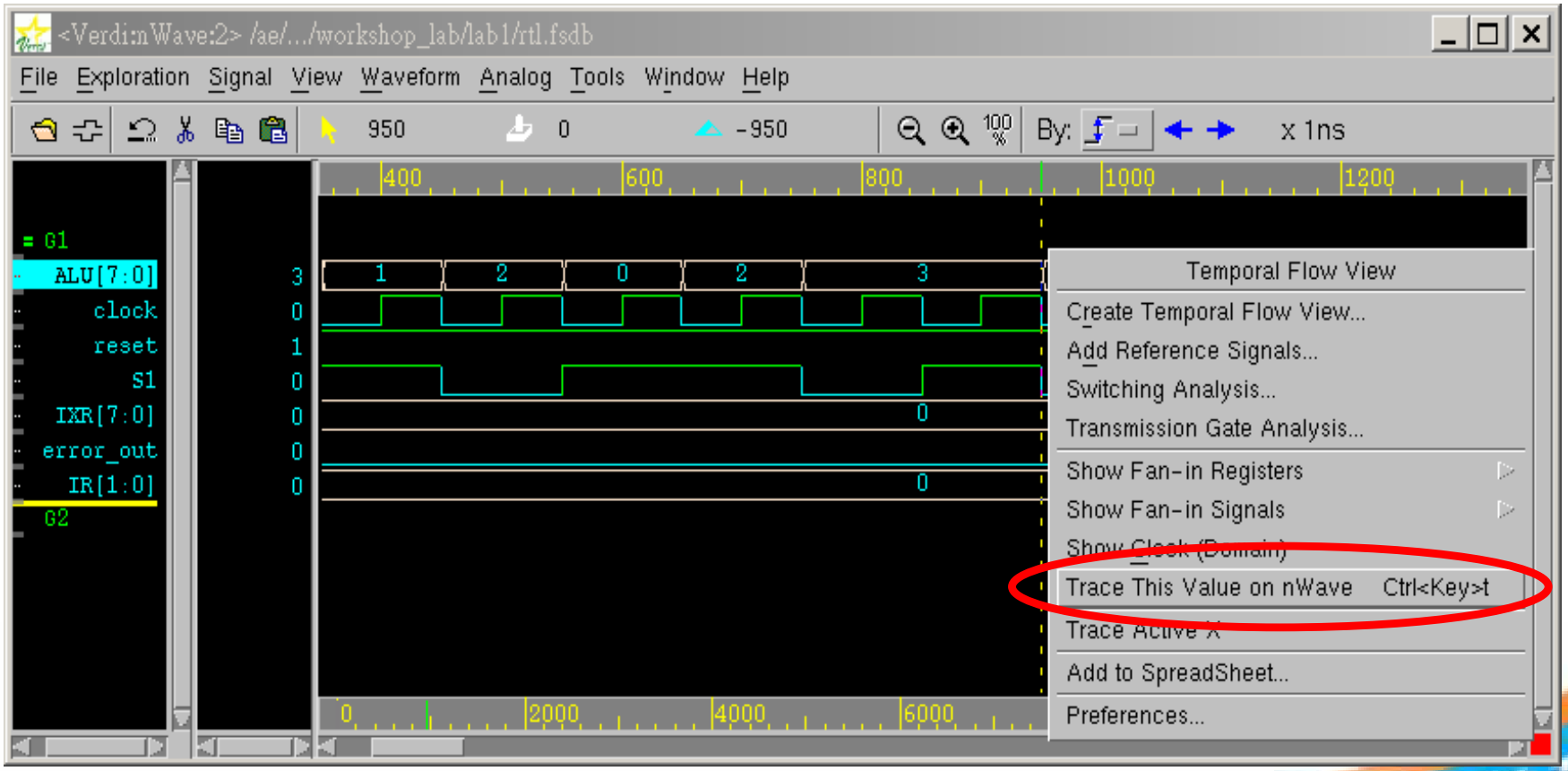




SpringSoft
Accelerating Engineers

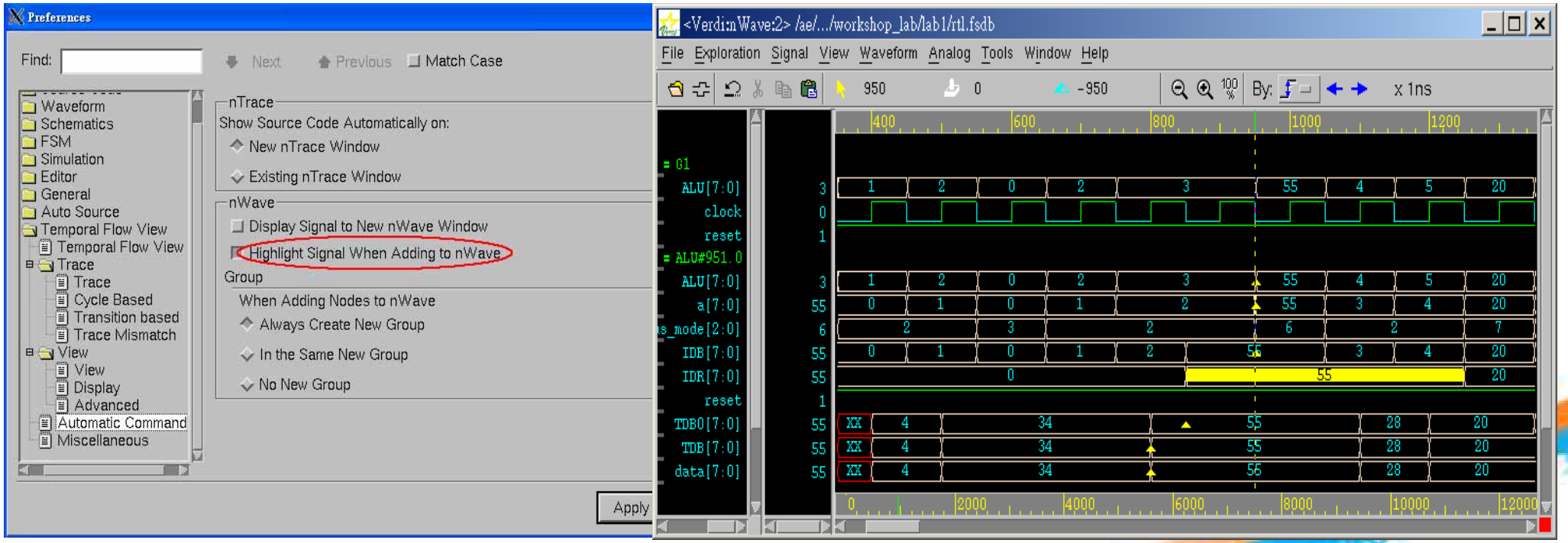
Trace This Value on Waveform(1/2)

- Automatically trace active paths in nWave
  - Place the cursor over a signal and invoke *RMB → Trace This Value on nWave*



Trace This Value on Waveform(2/2)

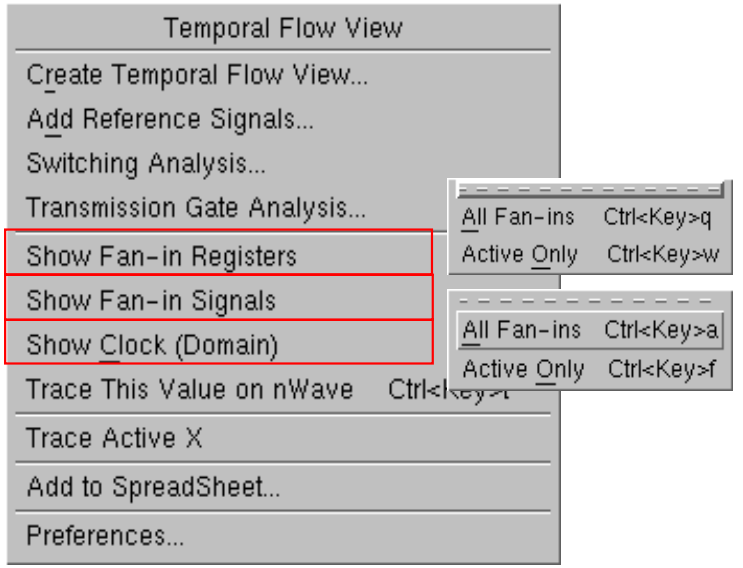
- On nWave, select RMB → Preferences → Temporal Flow View→ Automatic Command and enable ***Highlight Signal When Adding to nWave***
  - Execute ***RMB → Trace This Value*** on nWave after enabled.
  - The "arrow" symbol indicates sample points on internal nets
  - The "highlight" indicates registers



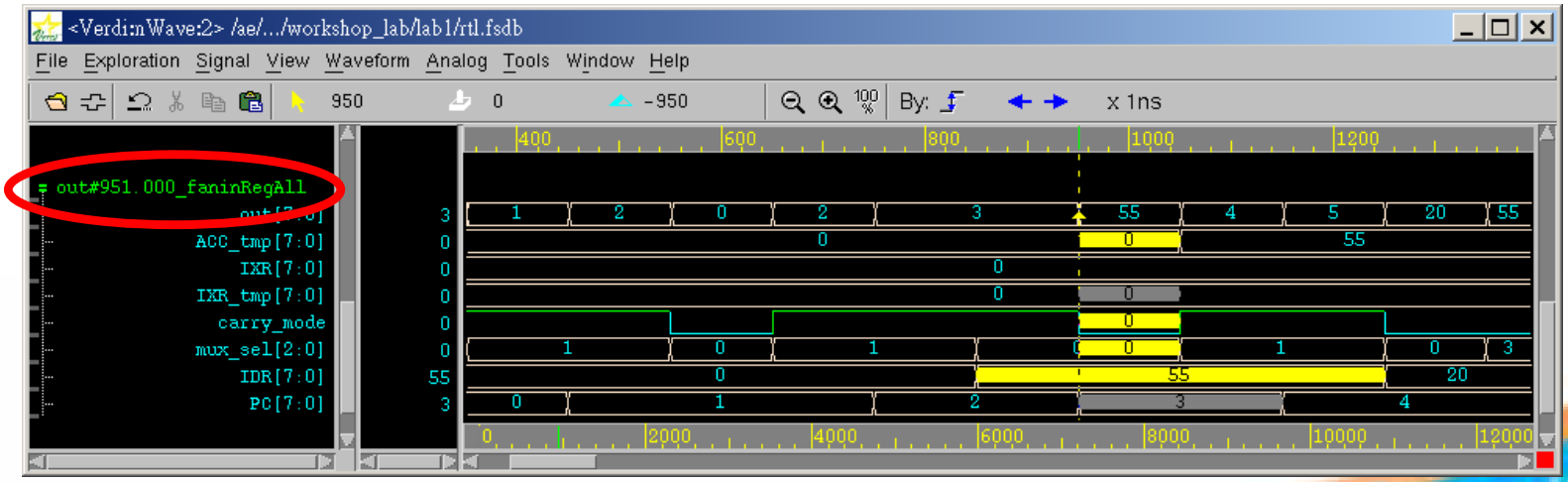
SpringSoft
Accelerating Engineers

Other Trace Commands

- Place the cursor over a signal at the desired time and invoke the RMB menu
  - Show Fan-in Registers
    - Active only or All fan-in registers in one cycle
  - Show Fan-in Signals
    - Active only or All fan-in signals in one connectivity
    - Similar to "trace driver and drag into waveform"
  - Show Clock
    - Show triggering clock



- Each command adds a new group of signals



SpringSoft Accelerating Engineers

Lab1 (1/3)

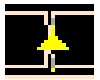
- Use "Active Trace" to trace wrong value
  - Verdi -f run.f -ssf rtl.fsdb &
    - In source window, **Drag&Drop** scope system.i_cpu.i_ALUB to nWave
  - Debug ALU[7:0] 3→55 @951ns
    - **Active Trace** for signal ALU on nWave
    - **Active Annotation** on nTrace
    - **Active Trace** for signal "a" on nTrace until TDB0
    - Trace TDB0's driver on partial schematic (**Tools → New Schematic → Connectivity**)
    - **Active Annotation** on nSchema
    - In schematic, select "TDB0" and move cursor to last transition (34→55) at time 800
    - **Double click** on mux input pin(1) to expand to a latch
    - **Double click** on latch data pin
    - **Double click** on tri-state buffer input to trace to memory
      - Which address does value 55 read out? 2

Lab1 (2/3)

- Use "Trace This Value" in temporal flow view to debug ALU@951
  - **RMB** on ALU[7:0] 3→55 @951ns in nWave
  - **Create Temporal Flow View**
  - Enable "Cycle Based", then click **OK**
  - In temporal flow view ,double click on a[7:0] to trace its active statement
    - What's the driving logic? Mux
    - Which is active data signal? IDB
    - Which is the active control signal? bus_mode
  - In temporal flow view, **RMB** on "IDB" and "**Trace This Value**"
    - How many cycles does Verdi trace back? 2 cycles
    - What's the value does memory read out for address 2? 55

Lab1 (3/3)

- Try to show the latch's source code
- Try to show Marker "!" for control pin with transition on temporal flow view from preference setting
  - How many marker shown? 3

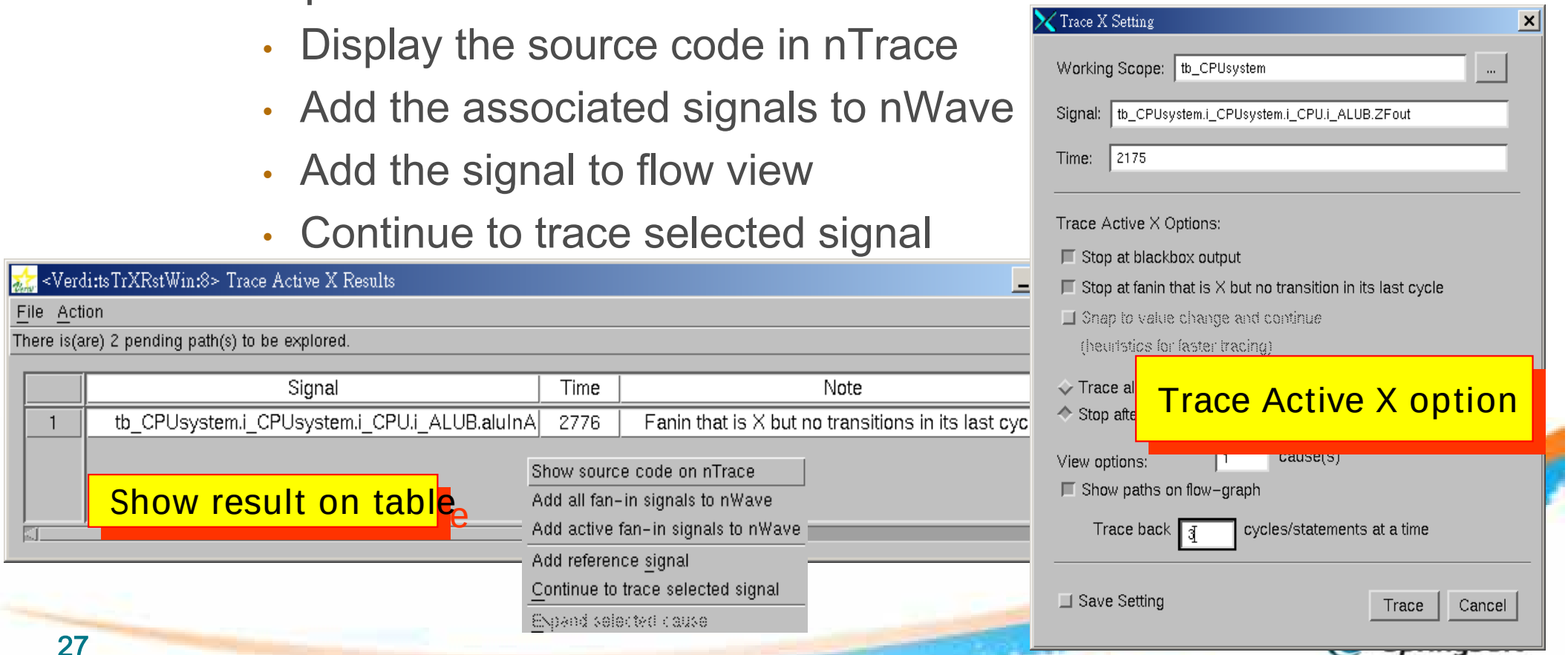
- Use "Trace This Value" on waveform for signal "ALU" @951
  - RMB on waveform and select "preference"
    - Enable "Highlight Signal When Adding to nWave" Option (Temporal Flow View tab→ Automatic Command tab)
  - RMB on signal "ALU" @951 and "Trace This Value on nWave"
    - All the active signals will show in waveform
    - The sampling point will mark with 
    - The register signals will mark with a rectangular yellow bar

Trace X Cause

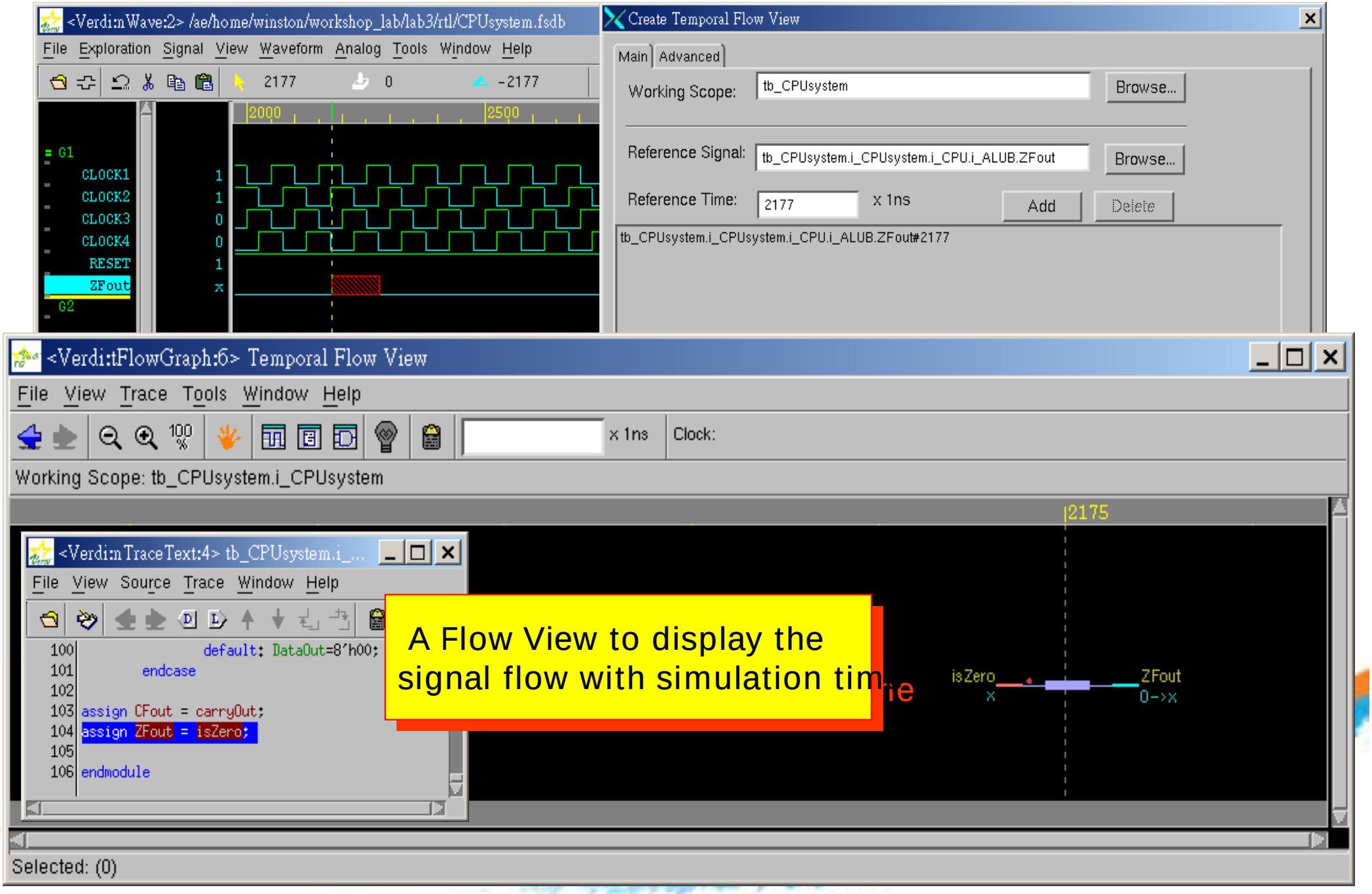
- Common Problem
  - Hard to locate the source of X value since there are multiple causes, such as timing problem, no driver, bus contention, no initial value...etc.
  - The number of logic gates with X value is too large to debug one by one
- SpringSoft Solution
  - Verdi's *Trace Active X* to locate the X cause on Flow View or waveforms

Verdi's Trace Active X(1/3)

- In nWave, use *Trace Active X* to locate the cause of the unknown across multiple cycles.
  - Place the cursor at the transition to X and invoke *RMB* → *Trace Active X*
  - Report the results in a table
    - Display the source code in nTrace
    - Add the associated signals to nWave
    - Add the signal to flow view
    - Continue to trace selected signal

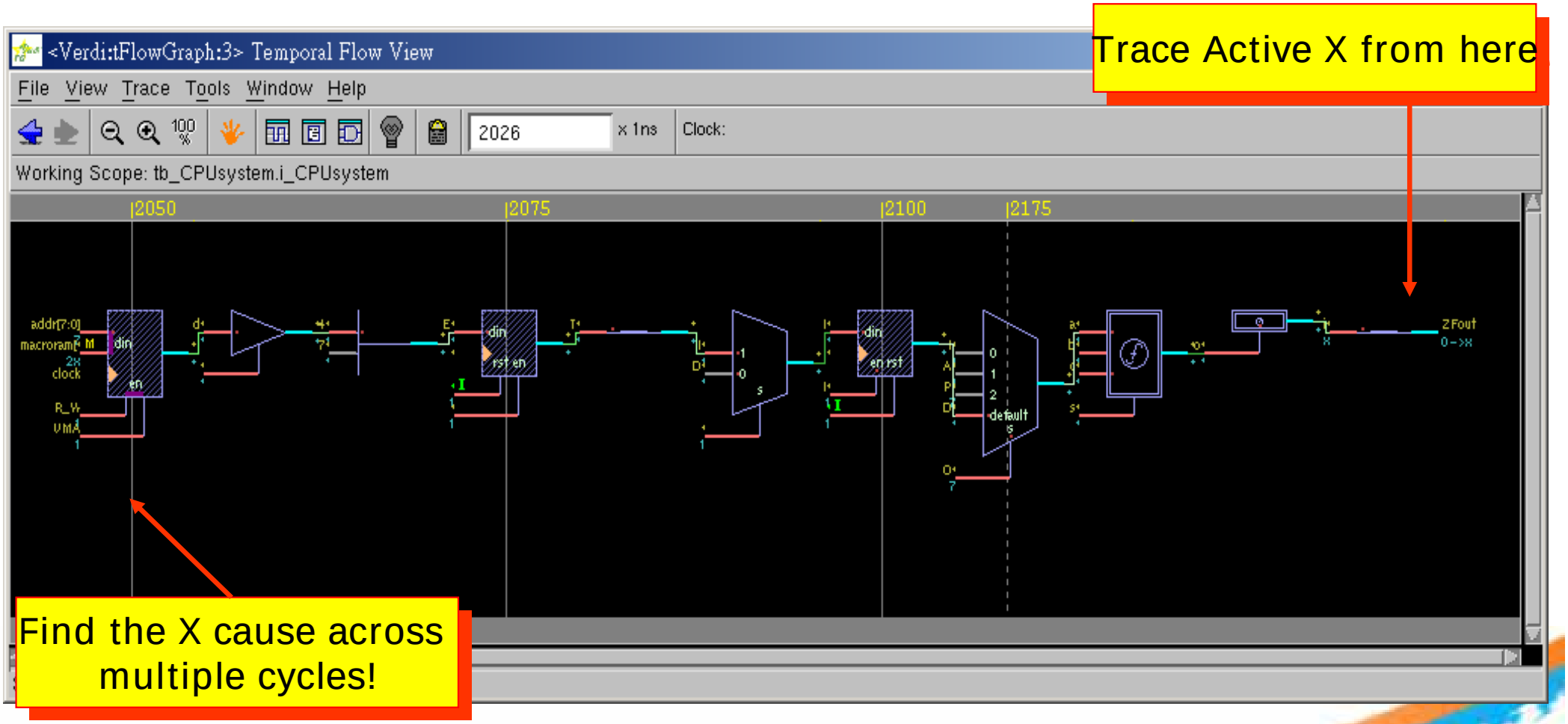


Verdi's Trace Active X(2/3)



Verdi's Trace Active X(3/3)

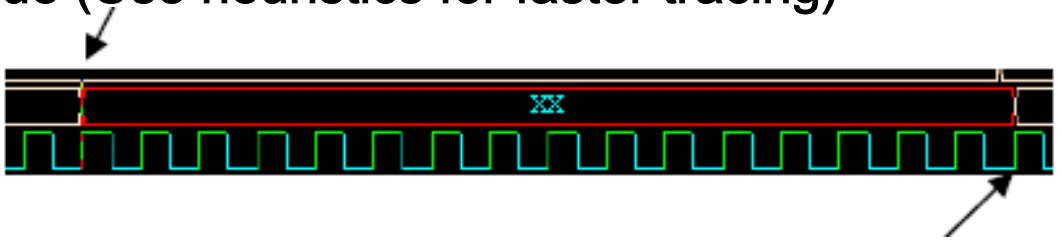
Hint: Click **RMB**



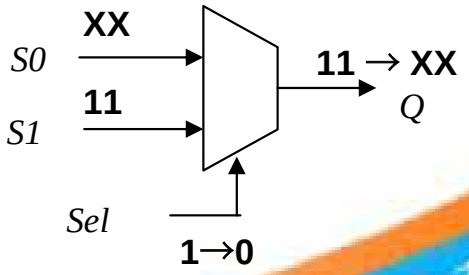
SpringSoft
Accelerating Engineers

Options for Trace Active X

- Stop at black box output signal (Default: ON)
- Stop at fanin that is X but no transition in its last cycle (Default: ON)
  - Verdi traces X transition and stops when none of the fan-in signals with X has transition in the current cycle
  - E.g. – Verdi stops tracing at Q. If this option is off, it continues tracing S0
- Snap to value change and continue (Use heuristics for faster tracing) (Default: OFF)



  - Jump to the first X transition
  - When the second option "Stop at fan-in that is X but no transition in it's last cycle" is on, this option has to be off because it can't snap to value change when the other option is on.
- Trace all causes (Default: OFF)
  - Trace all possible causes at once – breadth tracing
- Stop after finding # causes (Default: ON, with 1 cause)
- View options (Flow View) (Default: OFF)
  - Allows users to visualize the paths in flow view



Recommended Approach

- Use "fast search" to locate obvious errors (uninitialize, setup/hold time violation, no drivers...)
  - Stop at black box output signal (ON)
  - Stop at fanin that is X but no transition in its last cycle (OFF)
  - Snap to value change and continue (ON)
- Locate the active X source precisely
  - Stop at black box output signal (ON)
  - Stop at fanin that is X but no transition in its last cycle (ON)
  - Snap to value change and continue (OFF)
- The number of causes
  - Use the default setting : Stop after finding 1 cause
  - Specify a higher number or use "Trace all causes" if it keeps stopping at similar causes
    - E.g. un-initialization registers

Lab2 (1/2)

- Use "Trace Active X" to trace unknown
  - source "SourceMe" to set symbol library
  - Verdi -f run.f -ssf gate.fsdb &
  - Drag&Drop scope system.i_cpu.i_ALUB to nWave
  - Debug S1 with X at 1256ns
    - RMB on signal "S1"and "Trace Active X"
    - Create Temporal Flow View
    - Enable "Cycle Based", then click "OK"
    - In temporal flow view, RMB on "S1" and "Trace Active X"
    - Disable "Stop at Fan-in that is X but No Transition in Its Last Cycle" and click "Trace"

Lab2 (2/2)

- The debug results:
  - Pop up a Trace Active X Result table to show the unknown cause
  - Show the trace path in TFV
- RMB on signal "system.i_ALUB.n770" in the table and "Show source code on nTrace"
  - Highlight the problem logic U278 in nTrace
- Select "n773" in nTrace and invoke Tools → New Schematic → Connectivity
  - You can see signal n733 is floating (No driver)
- RMB on signal "system.i_ALUB.n770" in the table and "Add all fan-in signals in nWave"
  - You can see signal n733 is floating (high Z)

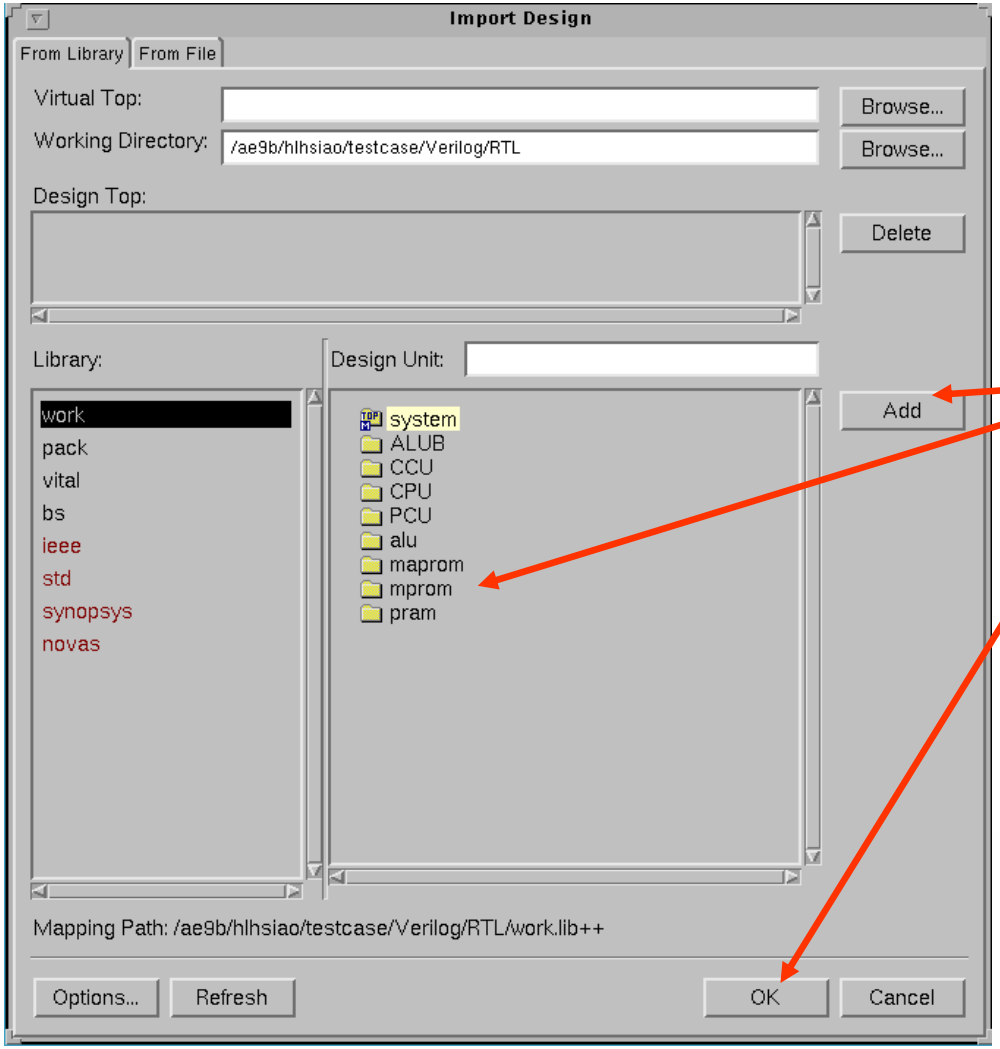
Import Design Faster

- Common Problem
  - It takes a lot of time to compile big design
  - Reloading design will compile all files
  - Loading sub-blocks with full-chip's simulation result will break down most features in Verdi
- SpringSoft Solution
  - Use *batch mode compilation* style
  - Set *Virtual Top*

Import Design From Library(1/2)

- Compile your Verilog design
  - vericom <your Verilog options>
- Compile your vhdl design
  - vhdlcom <your vhdl options>
- Importing design from library
  - Through Command Line
    - Verdi -lib libraryName –top TopModule
  - Through GUI
    - (nTrace) File → Import Design → From Library
- Use  to edit source code
- Reload Design will only compile modified files

Import Design From Library(2/2)



1.Select Design Unit
2.click "Add" button
3.click OK to
import design

SpringSoft
Accelerating Engineers

Import Design Faster
Set Virtual Top(1/6)

- Problem
  - Can't active annotation
  - Can't active trace from waveform
  - Can't drag signals from Source Code window to waveform window
- Cause:
  - hierarchy mismatch between Hierarchy Browser window and FSDB
- Solution:
  - set *virtual top*

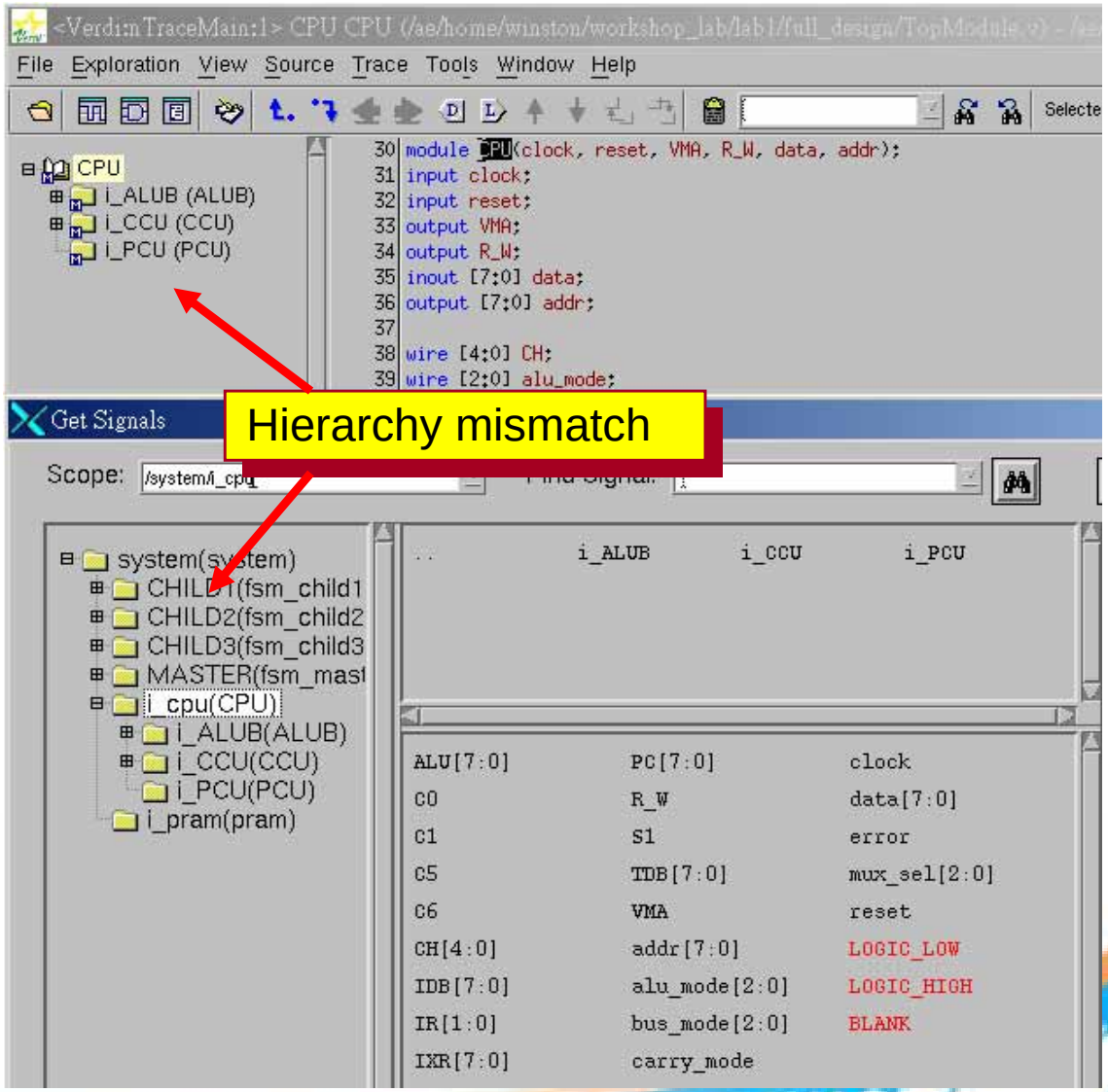
Import Design Faster
Set Virtual Top(2/6)

- Define the virtual hierarchy in a file as following format
  - Module_Name = Scope_Name_in_FSDB
  - For example : cpu=system.i_cpu
- Import design with virtual top file into Verdi
- Verdi will create a virtual top based on what virtual top file defines

Import Design Faster
Set Virtual Top(3/6)

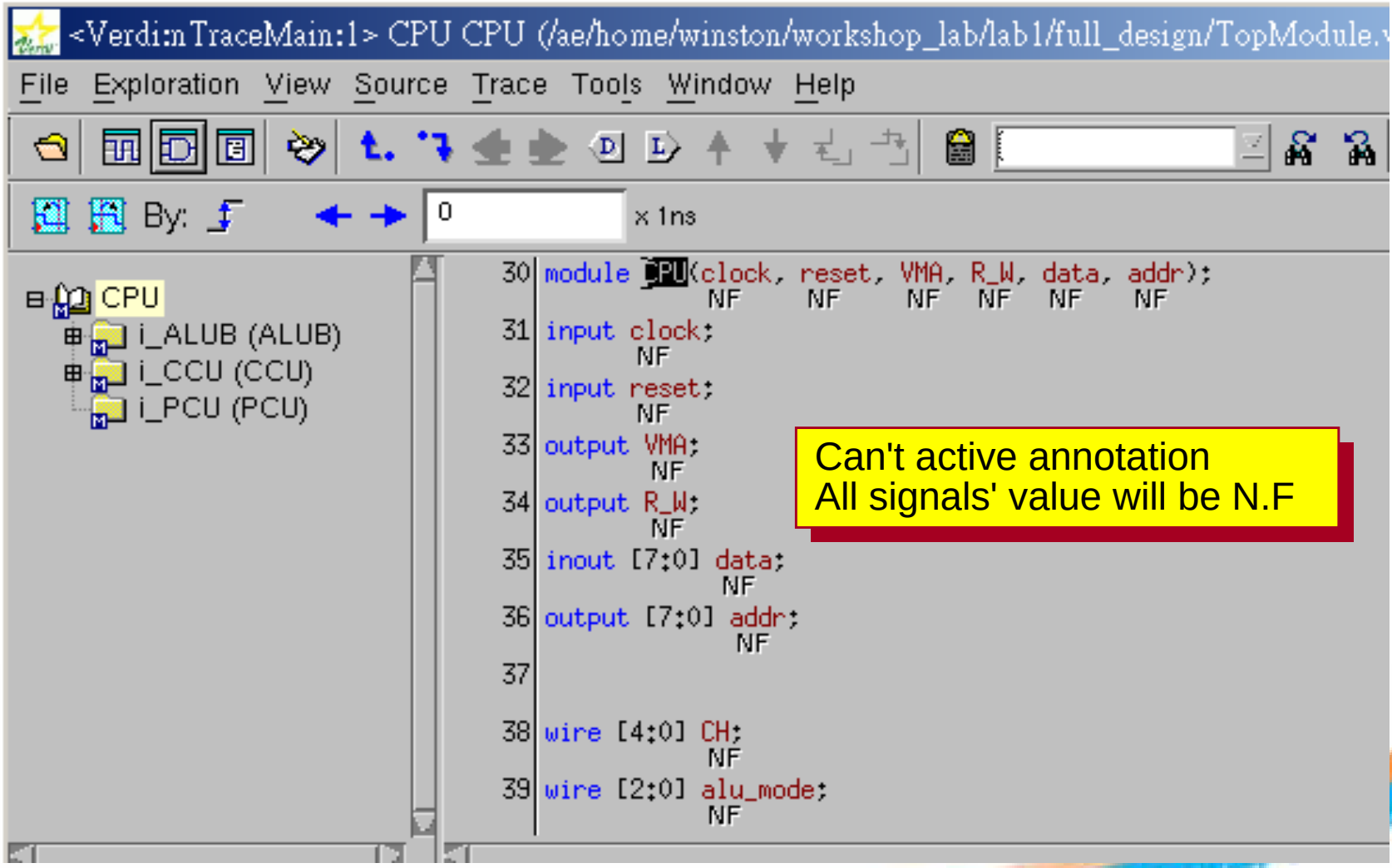
Your Design

Simulation is not only for your design, but also for whole chip

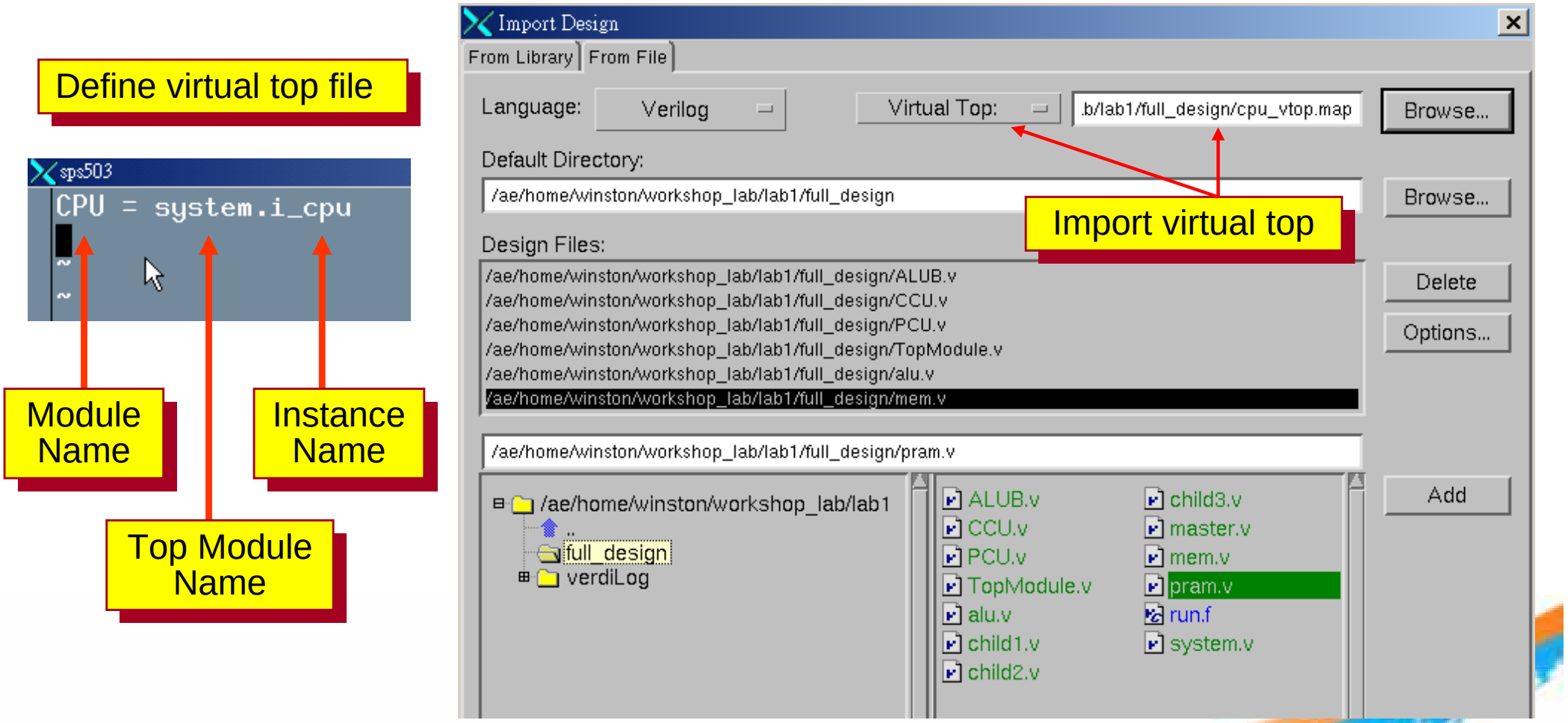


SpringSoft
Accelerating Engineers

Import Design Faster
Set Virtual Top(4/6)



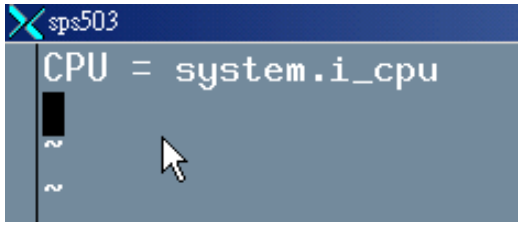
Import Design Faster
Set Virtual Top(5/6)

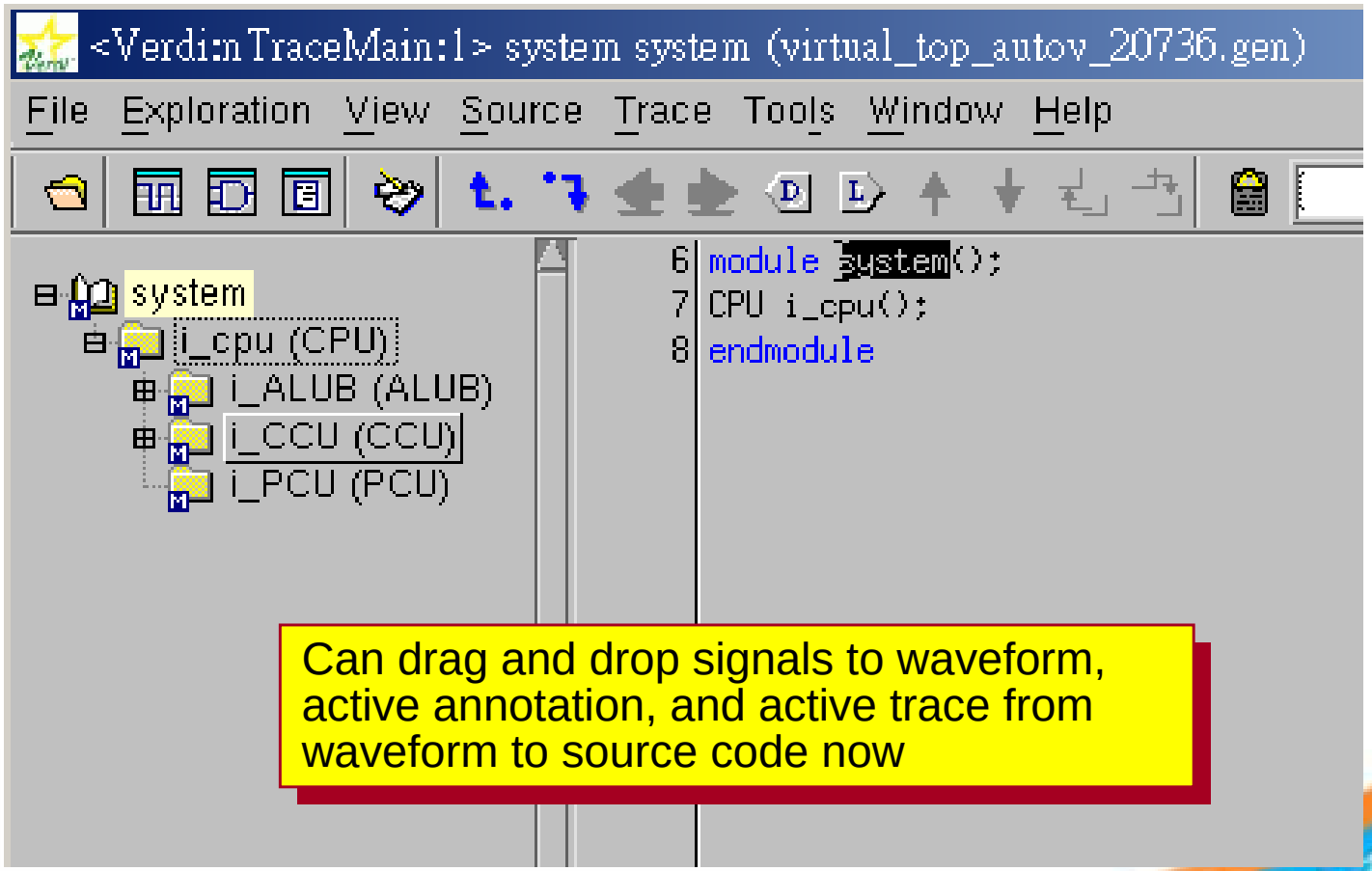


SpringSoft
Accelerating Engineers

Import Design Faster
Set Virtual Top(6/6)

Define virtual top file





Lab3-1

- Use batch mode compilation to shorten the time to import huge design
  - Compile Design
    - vericom -f run.f
  - Import Design from Library
    - Verdi -lib work -top system &
  - Modify system.v and save it
  - Increment Compile
    - vericom -smartinc -f run.f
  - Reload Design to check the new codes
    - File → Reload Design

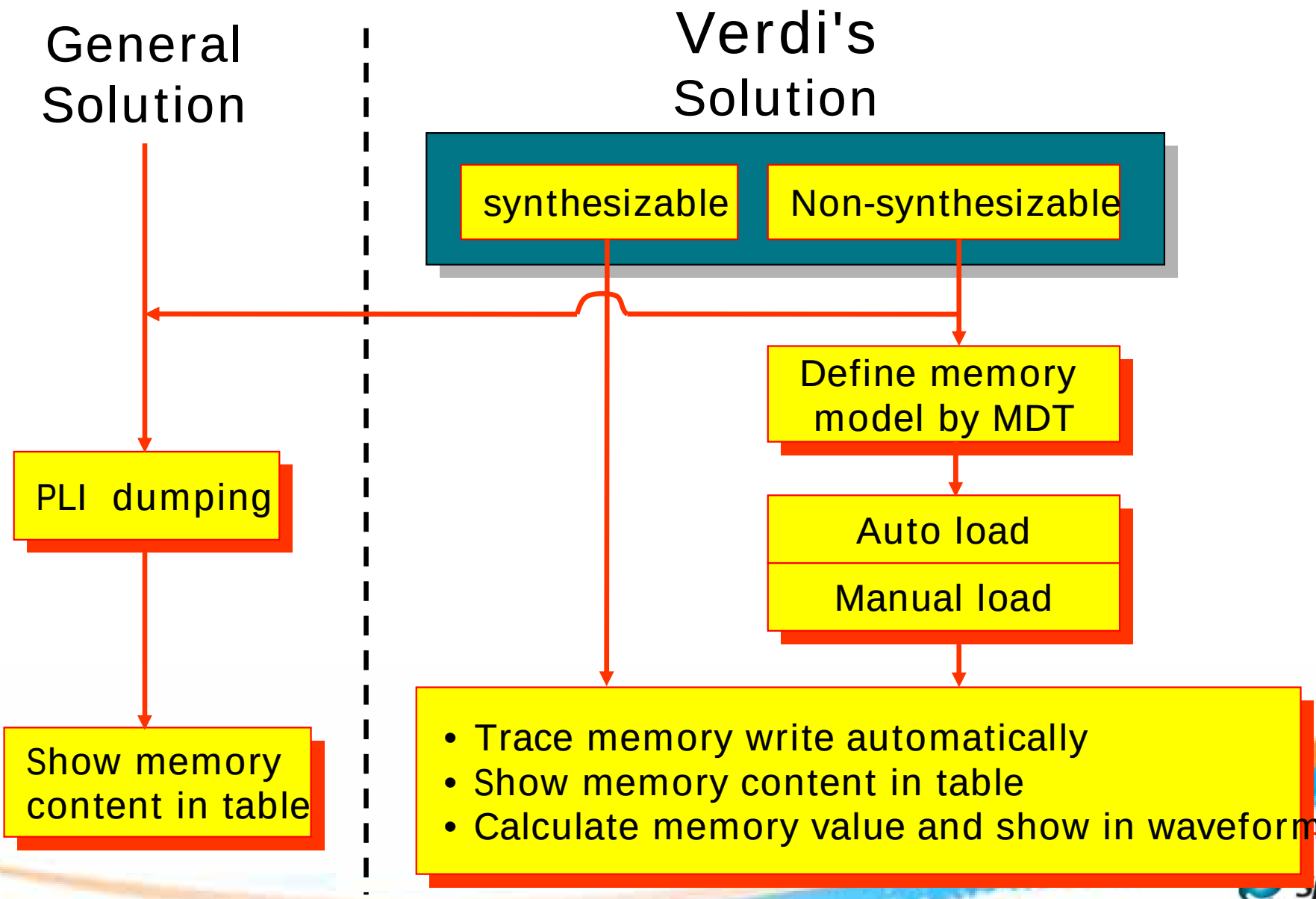
Lab3-2

- Set Virtual Top to solve the hierarchy mismatch between Hierarchy Browser window and FSDB
  - Invoke Verdi
    - Verdi &
  - Import Design → ALUB.f
    - **Tools → New Waveform** to open nWave and **load** rtl.fsdb
    - **D&D** between nWave and nTrace or **Active annotation** on nTrace
      - It don't work
  - Specify virtual top file "vtop.map" when import design
    - Verdi -vtop vtop.map -f ALUB.f -ssf rtl.fsdb &
    - Check the hierarchy
    - **D&D** between nWave and nTrace or **Active annotation** on nTrace
      - It works

Debug Memory

- Common Problem
  - Hard to dump the contents of memory
  - Difficult to view memory contents with traditional views (i.e. waveforms)
  - Many different memory models
- SpringSoft Solution
  - PLI dumping and show memory content
  - Don't need FSDB dumping, trace memory content automatically and locate the last write

Memory Debug Flow

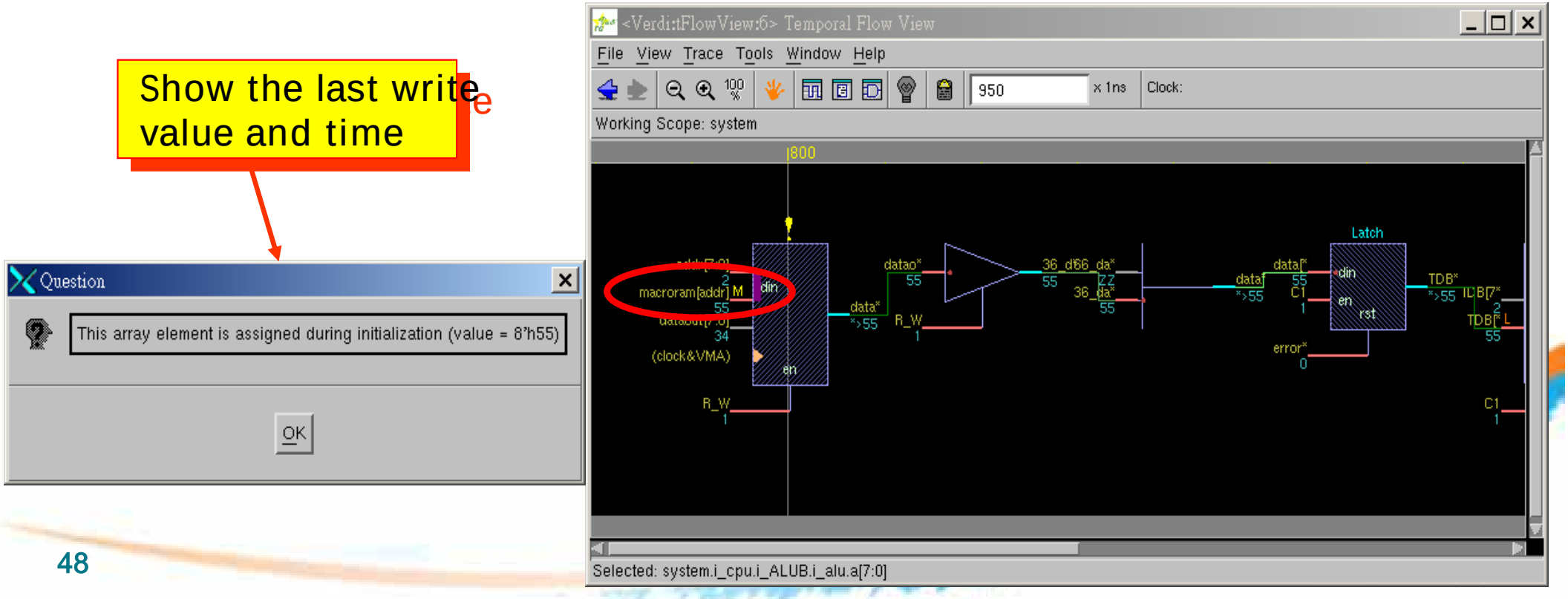


Verdi Debug Solution

- Synthesizable 2-D and Multi-Dimensional Arrays
  - Automatically locate the last write
  - Trace memory content in 2-D table or waveform viewer
- Non-Synthesizable Static RAM
  - Requires user-defined or pre-defined vendor memory models
  - Trace memory content in 2-D table or waveform viewer

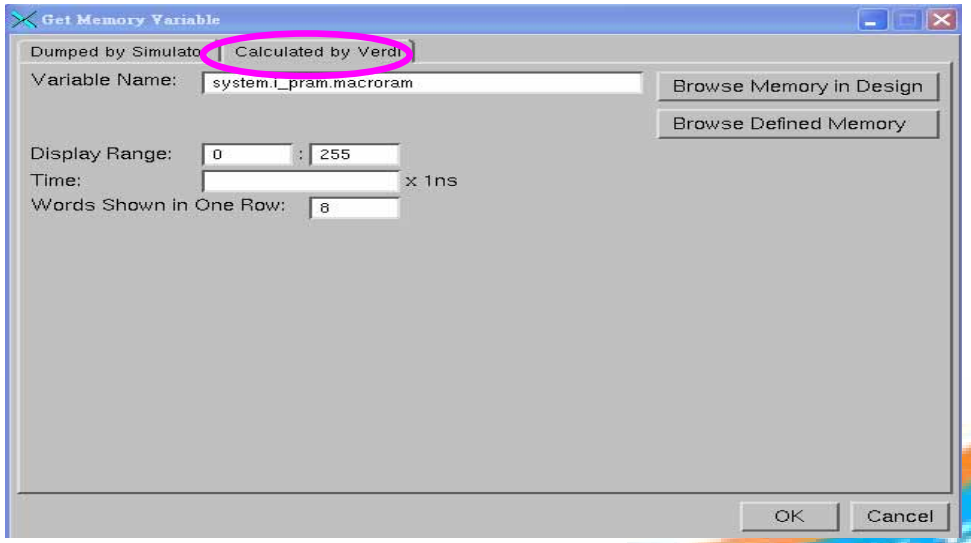
Locate Memory Write

- Verdi will calculate the the memory value for the specific address with its last write time
- Usage
  - In the flow view, double-click the memory signal
  - Or in nTrace invoke, **RMB → Debug Memory → Trace Memory Write**



Show Memory Content in Table(1/2)

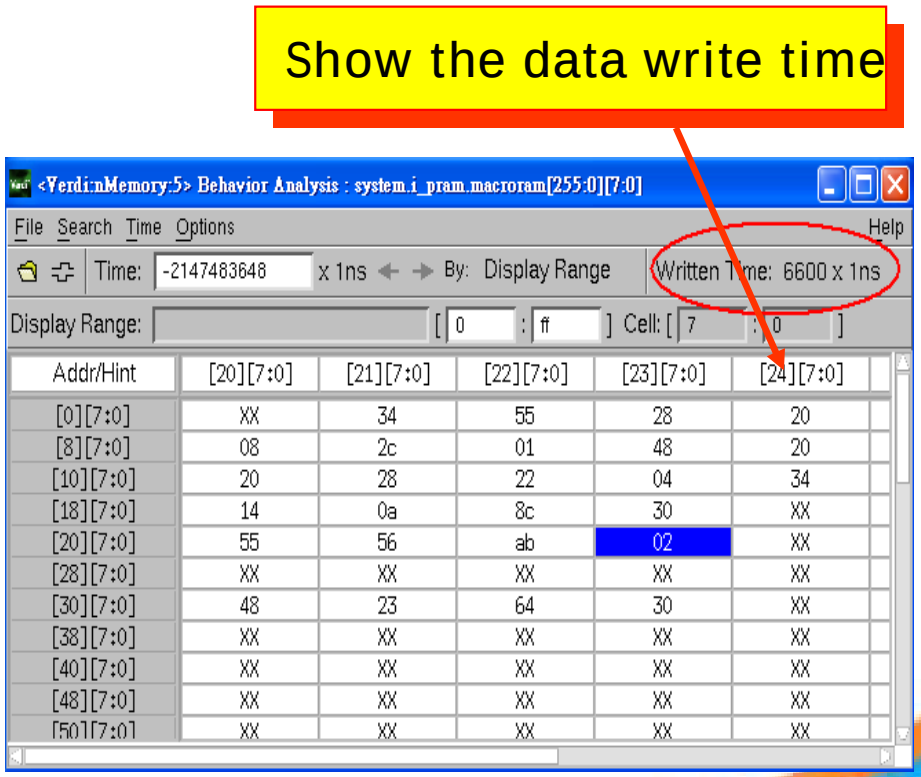
- Show memory contents for multiple locations
  - In the flow view, **Tools → Show Memory Contents**
  - Or nTrace, **RMB → Debug Memory → Show Memory Contents**
- Specify address range
  - Memory range specified in RTL is used by default
- Specify time:Trace latest write from 0 to specified time
  - In nTrace, default time is global cursor.
  - In flow view, default time is last memory access



SpringSoft
Accelerating Engineers

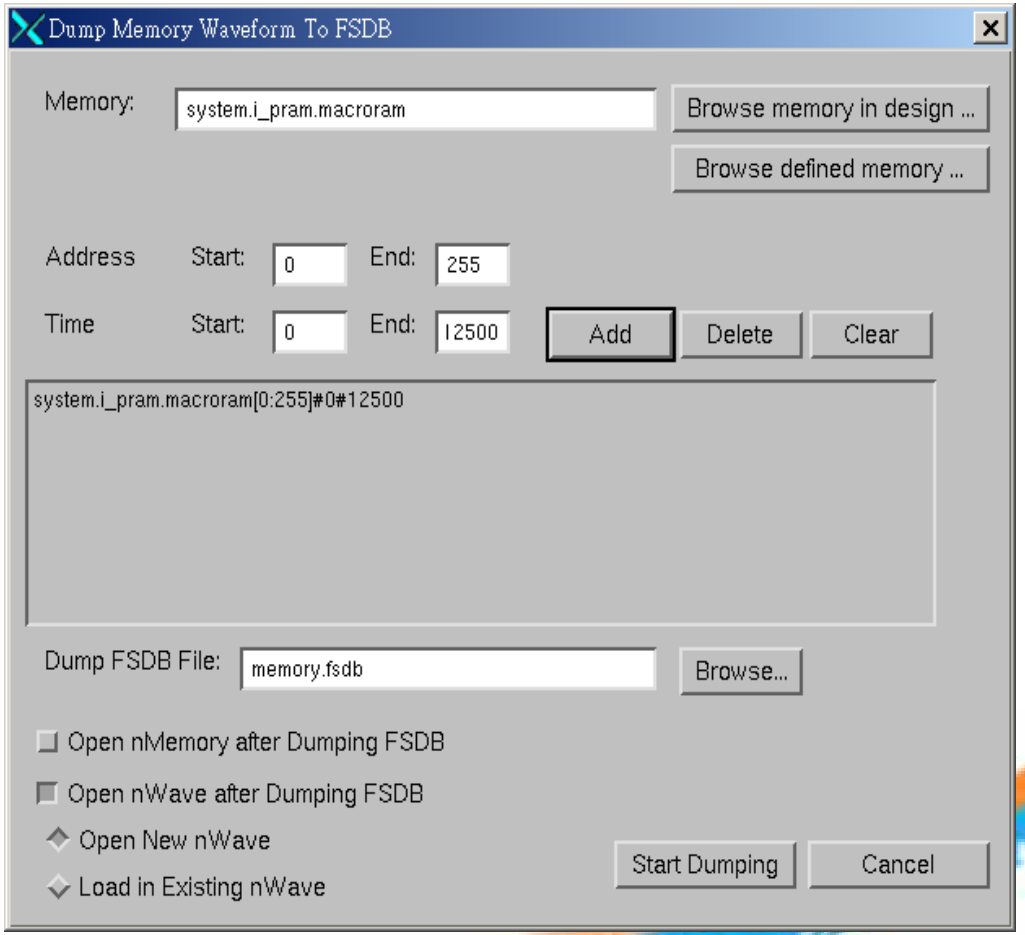
Show Memory Content in Table(2/2)

- Search for specific values
  - Search → Find
- Synchronize memory table with nWave
  - Time → Sync Cursor Time
- Customize the table
  - Options → Display Mode



Show Memory Contents in Waveform(1/2)

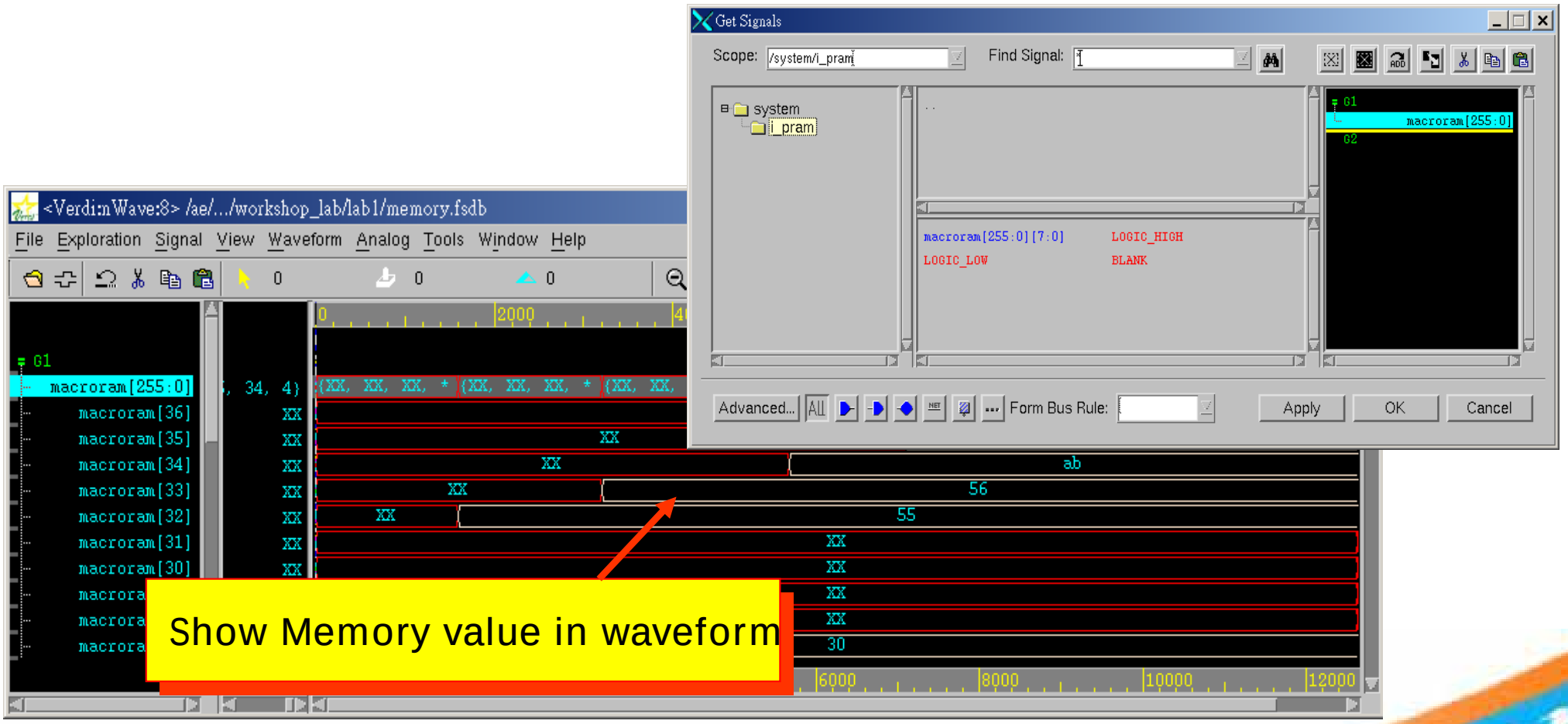
- Trace memory contents in nWave
  - In flow view, **Tools → Dump Memory Waveform to FSDB**
  - Or in nTrace, **RMB → Debug Memory → Dump Memory Waveform to FSDB**
- Specify address and time
- Click Add
- Specify a memory dump FSDB file name
- Check "Open nWave after Dumping FSDB"
- Click "Start Dumping"



SpringSoft Accelerating Engineers

Show Memory Contents in Waveform(2/2)

- Read the new FSDB into the original nWave window or new waveform window



SpringSoft
Accelerating Engineers

Lab4-1 (1/2)

- Debug memory for synthesizable memory
  - **Verdi -f run.f -ssf rtl.fsdb &**
    - Get signal "system.i_cpu.ALU[7:0]" in nWave
    - **RMB** on ALU[7:0] 3→55 @951ns in waveform
    - **Create Temporal Flow View**
    - Set the working scope to "system" and enable "Cycle Based" , then click **OK**
    - In temporal flow view (TFV), **RMB** on signal "a" and "**Trace This Value**"
      - It will trace to the memory @800
    - **Double click** on memory net "macroram" in TFV
      - What's the last write time for value 55 at address 2? Initialization

Lab4-1 (2/2)

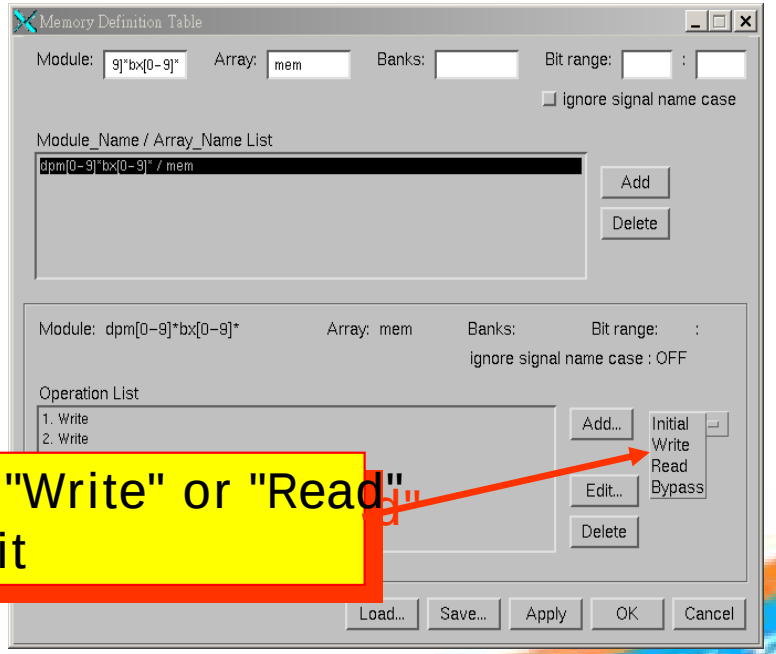
- Select memory net "macroram" in TFV and invoke **Tools → Show Memory Content...**
  - Specify display range 0:31, time 800ns, and click **OK** in the pop-up form
  - What's the value at address 7 ? 20
  - What's the last written time for address 7? 0ns
- Select memory net "macroram" in TFV and invoke **Tools → Dump Memory Waveform to FSDB**
  - Specify End time as 5000ns, click "**Add**" ,click "**Start Dumping**"
  - Show the memory "macroram" in nWave
  - What's the value of macroram[10]? 1

Verdi Debug Solution

- Synthesizable 2-D and Multi-Dimensional Arrays
  - Automatically locate the last write
  - Trace memory content in table or waveform viewer
- Non-Synthesizable Static RAM
  - Requires user-defined or pre-defined vendor memory models
  - Trace memory content in table or waveform viewer

Create Memory Definition File(1/3)

- Purpose: let Verdi to know the complex memory model behavior through simple definition table
- Usage
  - From nTrace, **Exploration → Memory Definition Table**
  - or **RMB** on memory → **Debug Memory → Memory Definition Table**
- Define the clock, control logic, address and data associated with the memory operations
  - Initial (optional)
  - Write (must)
  - Read (optional)
- Save the memory definition to a file(mdt) for
- future usage.



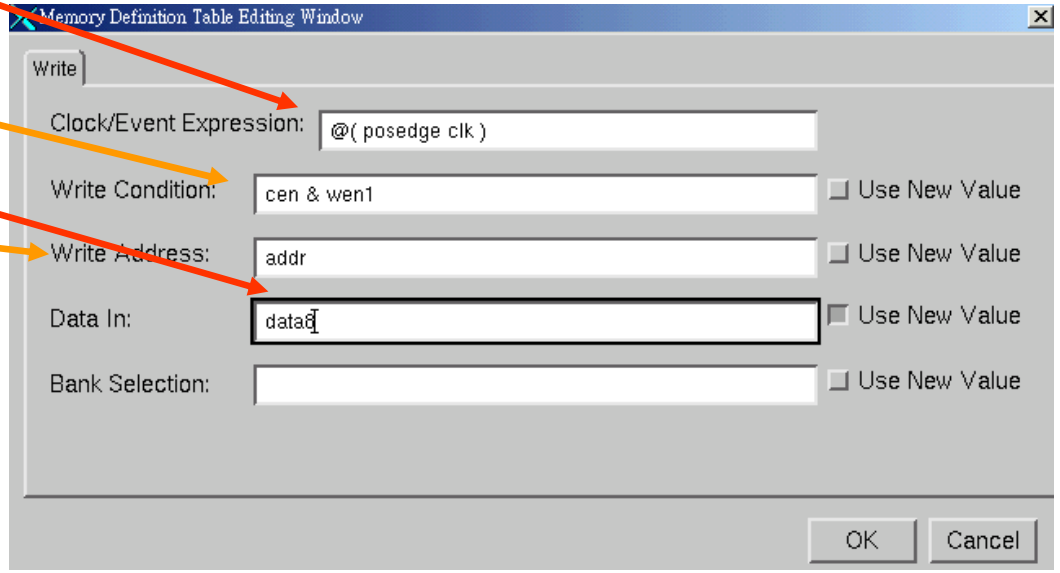
1. Select "Initial" or "Write" or "Read"
2. Click Add.. to edit

SpringSoft
Accelerating Engineers

Create Memory Definition File(2/3)

- Memory Write – Verilog example

```
always @(posedge(clk)
 if (cen & wen1)
    M[addr][0:7] := data8;
```



- MDT Description:

  clock : @(posedge clk)

  condition: cen & wen1

  address : addr

  data-in : data8

Create Memory Definition File(3/3)

- Memory Read – Verilog example

```
always @(posedge clk)
  if (cen & !(wen1 & wen2 & wen3 & wen4))
    data32 <= oe ?M[addr][0:31]:'bz;
```

- MDT Description:

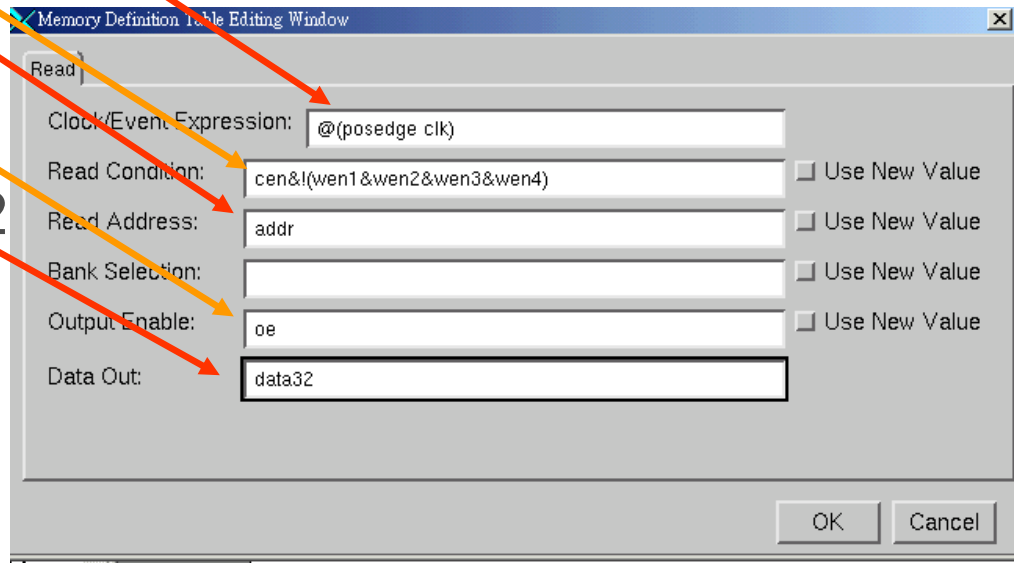
clock : @(posedge(clk)

condition : cen & !(wen1 & wen2 & wen3 & wen4)

address : addr

output_enable : oe

data-out : data32



Pre-Defined Vendor Memory Model

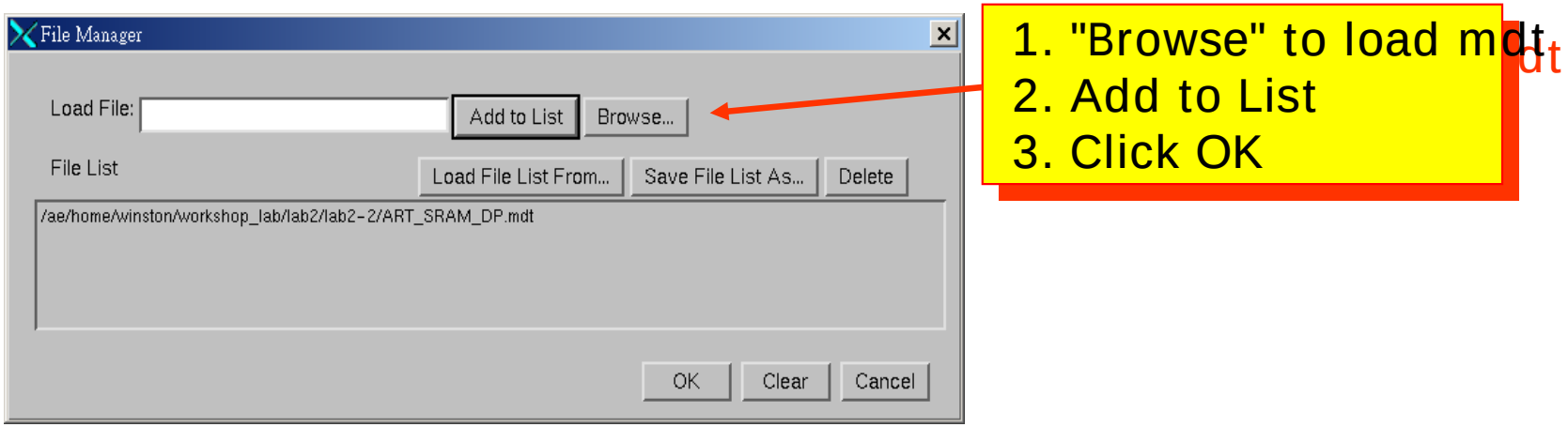
- Provide popular vendor MDT for memory model
- User only needs to set module mapping in novas.rc of MDT section before loading MDT into Verdi
- Located at <Verdi_installation_dir>/share/mdtlib
  - Artisan supported models
    - ART_RF_SP.mdt: Artisan Single-Port Register File
    - ART_RF_2P.mdt: Artisan Two-Port Register File
    - ART_SRAM_SP.mdt: Artisan Single-Port SRAM
    - ART_SRAM_DP.mdt: Artisan Dual-Port SRAM
  - Virage supported models
    - VIR_SRAM_SP.mdt: High Performance 1P(1rw) Sync High Density RAM
    - VIR_SRAM_DP.mdt:: High Performance 2P(2rw) Sync High Density RAM
    - VIR_RF_SP.mdt: High Performance 1P (1rw) Synchronous Register File
    - VIR_RF_DP.mdt: High Performance 2P (1r,1w) Sync Reg File

Memory Model Mapping

- Mapping format for memory models in [MDT] section of novas.rc
  - Verdi_supported_vendor_memory_name = user_memory_module_name
  - Default novas.rc contains:
    [MDT]
    ART_RF_SP = spr[0-9]*bx[0-9]*
    ART_RF_2P = dpr[0-9]*bx[0-9]*
    ART_SRAM_SP = spm[0-9]*bx[0-9]*
    ART_SRAM_DP = dpm[0-9]*bx[0-9]*
    VIR_SRAM_SP = hdsd1_[0-9]*x[0-9]*cm4sw1
    VIR_SRAM_DP = hdsd2_[0-9]*x[0-9]*cm4sw1
    VIR_RF_SP = rfsd1_[0-9]*x[0-9]*cm2sw0
    VIR_RF_DP = rfsd2_[0-9]*x[0-9]*cm2sw1
- Example:
  - Consider Artisan Single Port Register File with design memory module name as sram256bx16
  - novas.rc file would be updated as:
    [MDT]
    ART_RF_SP = sram[0-9]*bx[0-9]*

Load Existing Memory Definition Files

- Support to load existing MDT manually or automatically
  - From the GUI:
    - Exploration → **Load Memory Definition Table** from File



  - From the Verdi command line:
    - Individual file: -mdt <MDT_file>
    - List of memory definition files: -mdt <MDT_list>
  - Automatic loading:
    - setenv MDT_LIBPATHS < MDT_path1 MDT_path2>
    - setenv MDT_LIBS < MDT_file1 MDT_file2>

Debug Non-Synthesizable SRAM

- After the memory model is defined, Verdi can
  - Automatically calculate the last write for specific address
  - automatically calculates and displays the memory contents in table
  - Show the calculated result into fsdb and show on waveform

Debug Memory (General Solution)

- Use PLI/FLI calls to dump contents of memory/MDA models
  - Verilog –
    - $fsdbDumpMem(macroram, 0, 256);
      - Specified at the memory WRITE operation
    - Initial #200 $fsdbDumpMemInScope(1, TB, 500);
    - $fsdbDumpMDAOnChange(1, system.i_pram);
    - $fsdbDumpMDAOnChange(screen, 10, 20, 15);
      - reg [7:0] screen[79:0][0:24][1:0]
  - VHDL –
    - Memories declared as signals dumped on-change using fsdbDumpvars
    - Memory variables dumped on-demand with fsdbDumpMem and related calls

Array Name | Start Addr | Depth

Level | Scope | Cycle Time

$.....(system);

screen[10][15][1:0]~screen[29][15][1:0]

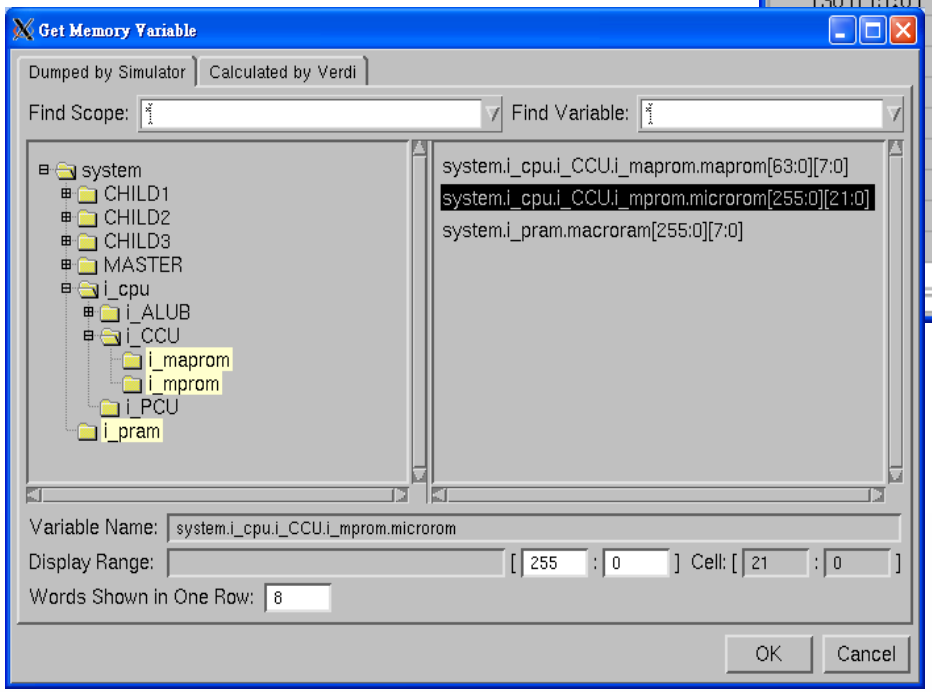
Total 40 cells will be dumped

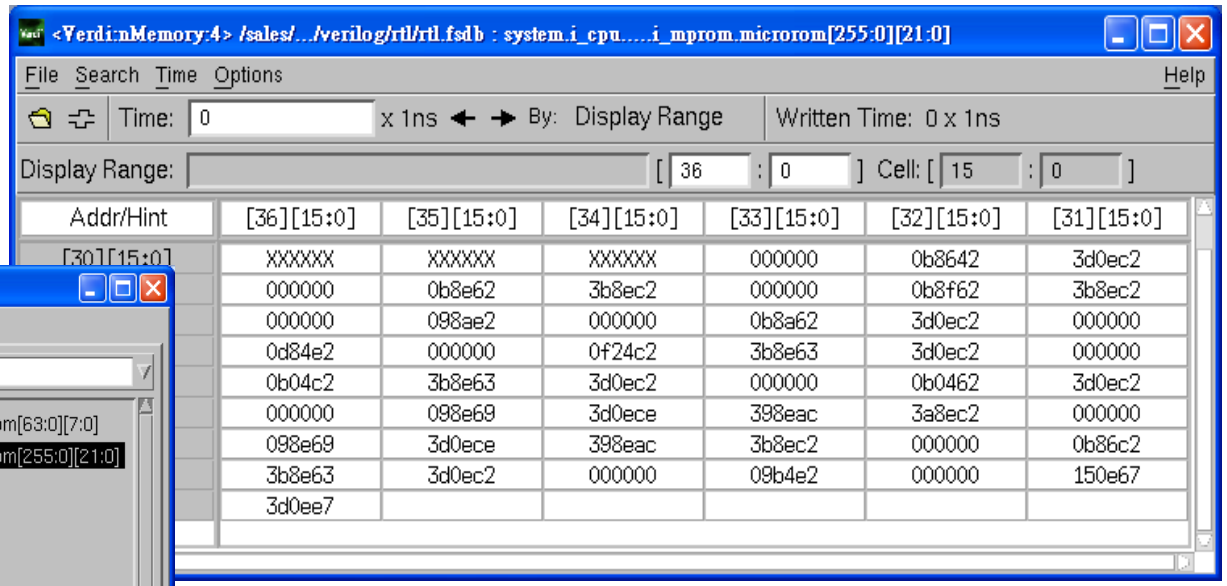
Debug Memory (General Solution)

- Visualize memory/MDA contents in a 2-D table format or a waveform viewer
  - In nTrace or nWave, **Tools** → **Memory/MDA** to display the memory/MDA
    - Synced with other views in time
    - Cells that change are highlighted in red
    - Drag & drop cells to nWave to view as waveforms

Memory/MDA Viewer

- Use File → Get Memory Variable to select a dumped MDA
  - Choose the range and cell dimensions to display





The range and cell dimension can also be modified from the viewer tool bar

SpringSoft Accelerating Engineers

Lab4-2 (1/2)

- Use pre-defined MDT to trace memory
  - **Verdi -f run.f -ssf memsys.fsdb &**
  - Change window time unit to "1ns" in nWave
  - Change scope to "tb_memsys.memsys0" in hierarchy browser and select instance "ram0" in source code line 45
    - Does the memory module name match the mapping for Artisan dual port SRAM in MDT section of novas.rc? Yes, ART_SRAM_DP = dpm[0-9]*bx[0-9]*
  - Load ART_SRAM_DP.mdt into verdi
    - In source window, **RMB** on instance "ram0" and invoke **"Debug Memory"** → **"Memory Definition Table"**
    - Click "**Load**" and then "Browse..." and **double click** in ART_SRAM_DP.mdt (File Manager Window)
    - Click "**Add to list**" and click "**OK**"
    - Select "**1.write**" and click "**Edit..**" to see the write operation content (Memory Definition Table Window)
    - Click "**OK**" to close the Memory Definition Table Editing Window
    - Click "**OK**" to close Memory Definition Table window

Lab4-2 (2/2)

- Show memory content in table
  - In source window,**RMB** on instance "ram0" and invoke **"Debug Memory" → "Show Memory Content..."**
  - Specify display range from 0 to 127, time to 30000 ns and click "**OK**"
  - Do Behavior Analysis and set working scope to tb_memsys
    - What's value at address 48? 3ed0
    - What's the last write time at address 48? 29530 ns
- Show memory content in waveform
  - In source window, **RMB** on instance "ram0" and invoke **"Debug Memory" → "Dump Memory Waveform to FSDB..."**
  - Specify display range from 0 to 127, time from 0 to 30000, then click "Add" then click "Start Dumping"
  - Show the memory "mem" on waveform
    - What's the value of mem[72] at 29530 ns? 3ed0

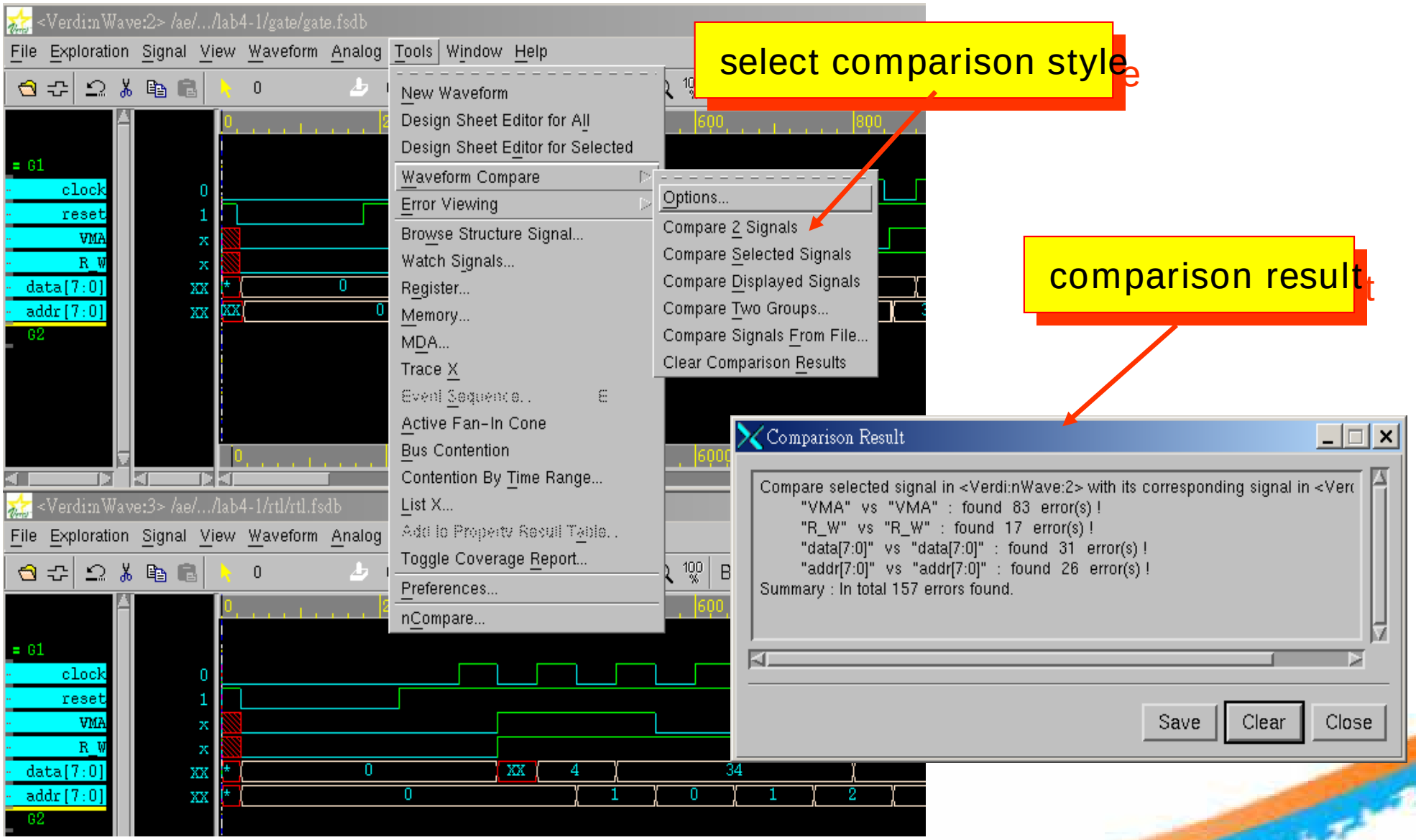
Compare Simulation Waveforms

- Common Problem
  - Trace the signal with good and bad values in two waveform viewers
  - Continue tracing drivers of the signal in two waveforms backward over time and compare their values until finding the cause
- SpringSoft Solution
  - Compare two different designs' FSDBs
    - Waveform comparison and then Trace this value
  - Compare the same design's FSDBs
    - Waveform comparison and then behavior trace for FSDB mismatch

Use Waveform Comparison

- Purpose
  - Fast compare 2 simulation result for interested signals
- Usage
  - Load 2 fsdb into waveform
  - Tile & Sync. 2 waveforms
    - Window→Tile Waveform
    - Window→Sync. Waveform View
  - Select interested signals
  - Compare with desired style
    - Tools→Waveform Compare→...

Use Waveform Comparison
Waveform Compare



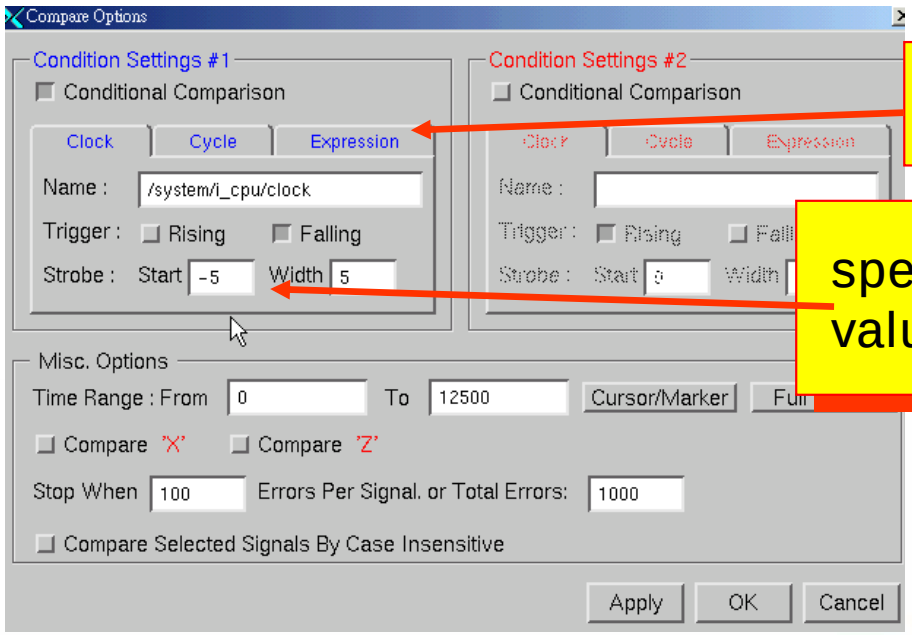
SpringSoft
Accelerating Engineers

Use Waveform Comparison
Comparison Style

- Compare 2 Signals
  - Signal name can be different
- Compare Selected Signals
  - Only matched signal name will compare
- Compare Displayed Signals
  - Only matched signal name will compare
- Compare 2 Groups
  - Compare signals based on displayed order
  - Bit width must match, signal name can be different
- Compare Signal from File
  - Example: /test/top/i_cpu/alu

Use Waveform Comparison
Compare Options(1/2)

- Clock
  - Compare signals based on Rising/Falling edge of a specific signal
  - Support hold time/set up time with strobe
- Cycle
  - Compare signals based on cycle time
- Expression
  - Compare signals when expression is true or false



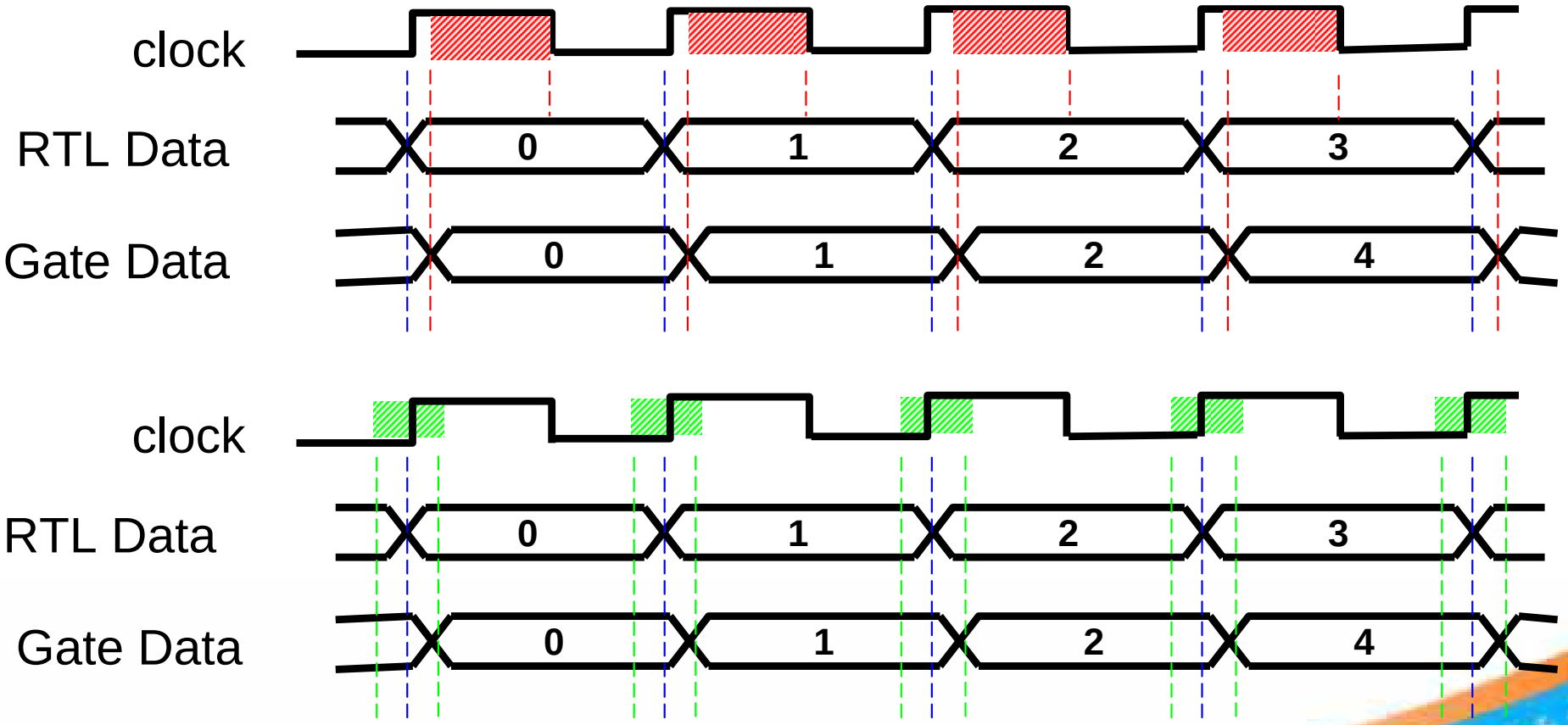
specify compare options

specify positive edge/negative start value to compare hold/set up time

SpringSoft Accelerating Engineers

Use Waveform Comparison
Compare Options(2/2)

- Specify proper value for "strobe"
  - Check register output
  - Check set-up time / hold time



Lab5-1 (1/2)

- Use nWave to compare two waveforms
  - At gate: **Verdi -f run.f -ssf gate.fsdb &**
    - (nWave) **Get** signals → All signals in scope "system.i_cpu.i_ALU"
    - **Tools → New Waveform** to open a new nWave and load "../rtl/rtl.fsdb" for the same signals
    - (nWave @gate) **Window → Tile Waveform**
    - (nWave @gate) **Tools → Waveform Compare → Options**
    - Enable the two Conditional Comparison
      - D&D to specify the 'clock" to Clock Name and set up the Strobe: Start and Width for the two condition settings: 4//36 and 5//45
      - Specify Time Range From: 0 to 10000

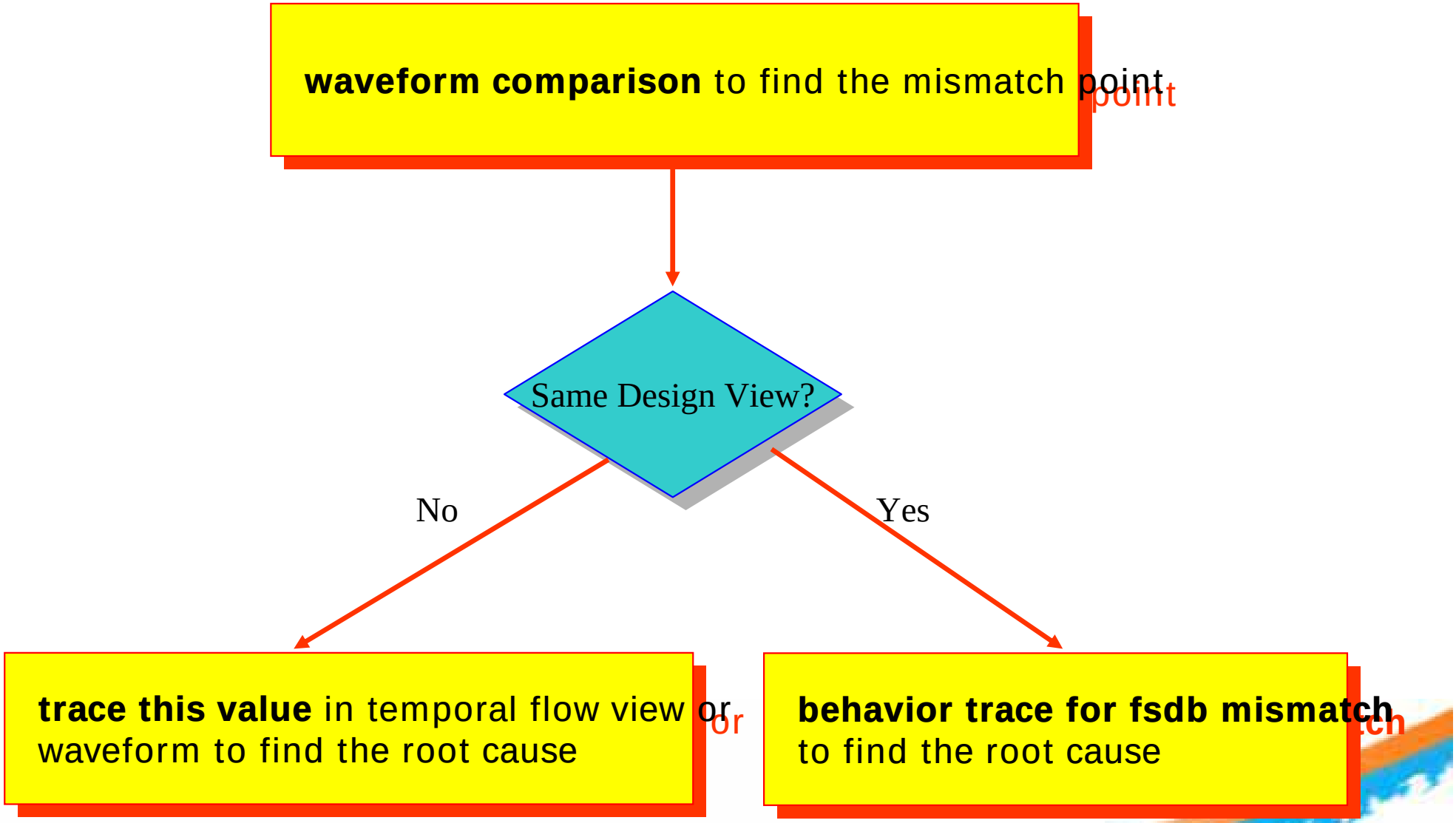
Lab5-1 (2/2)

- (nWave @gate) **Tools** → **Waveform Compare** → **Compare Two Groups** → specify the Group Name "G1" and "G1" → **OK**
  - How many mismatches are found? 6
- Click the blue right arrow on toolbar to view mismatches
- (nWave @gate) **Select** clock signal and **Put** Cursor @240ns
- Select clock and **View** → **Grid Options** → **enable** "Grid on Rising Edge" / "Grid Count with Start Number" 1 / "Lock Grid Count" and **click** OK
  - How many clock cycles are there in the gate waveforms? 154
  - Which clock cycle is the first mismatch found? 17
  - You can try other comparison style by yourself

Faster Debug between 2 FSDBs

- Common Problem
  - Trace the signal with good and bad values in two waveform viewers
  - Continue tracing drivers of the signal in two waveform backward over time and compare their values until locate the cause
- SpringSoft Solution
  - **Waveform comparison → Trace this value** in flow view for different design
  - **Waveform comparison → behavior trace for fsdb mismatch**

Debug Flow between 2 FSDBs

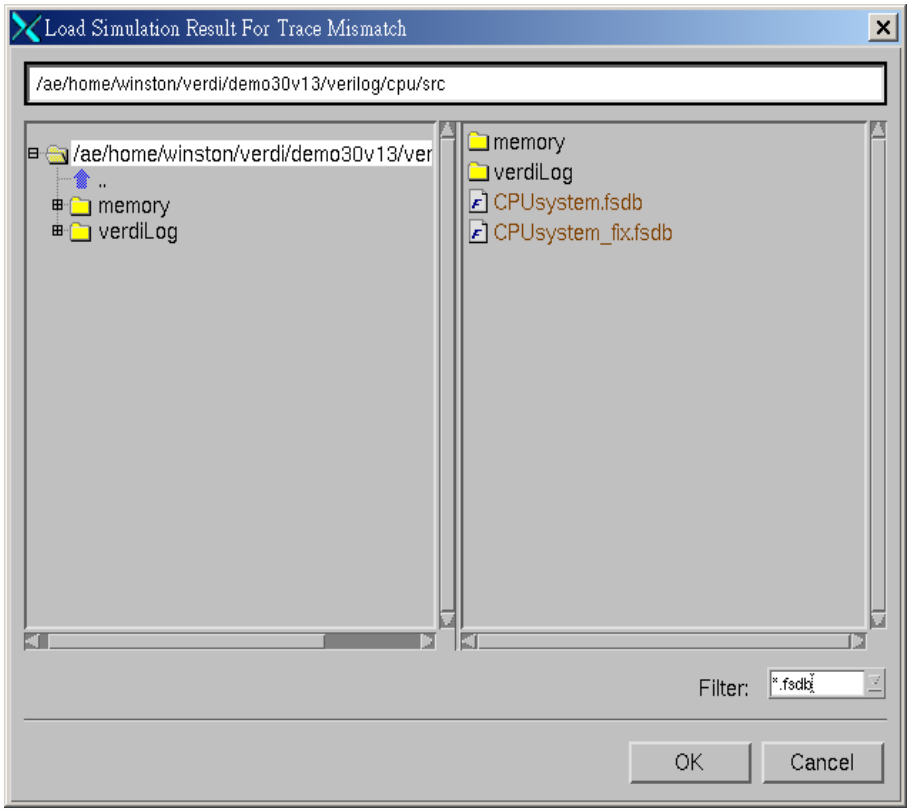


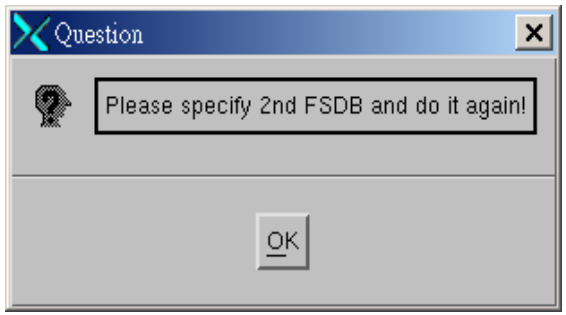
Behavior Trace for FSDB Mismatch
The Same Design

- Purpose:
  - Automatically trace the root causes of mismatches across multiple cycle in the flow view
- Typical causes for mismatches
  - Use different options from the same simulator
  - Two different simulators
  - Race conditions
- Usage:
  - Use waveform comparison find the difference
  - Create new **Temporal Flow View** for mismatched signal
  - Use "**behavior trace for fsdb mismatch**" to find the cause

Behavior Trace for FSDB Mismatch
Trace Mismatch Cause (1/3)

- Create flow view for the mismatched signal
- Load in 2nd FSDB file in flow view
  - Use **File → Load 2nd FSDB for Trace Mismatch**

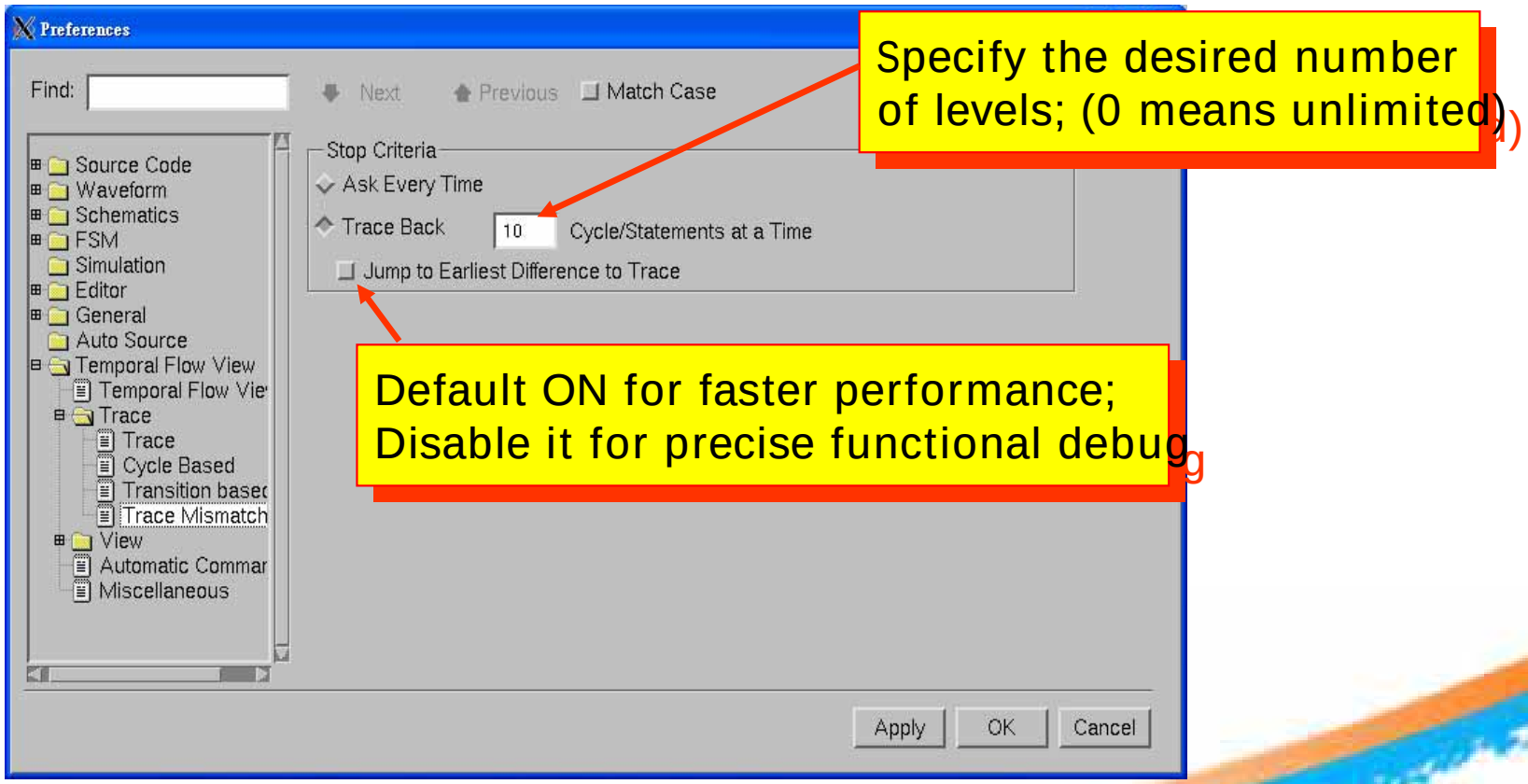




PS: If you directly trace mismatch without loading the *2nd* FSDB, one warning message and the GUI will pop up automatically.

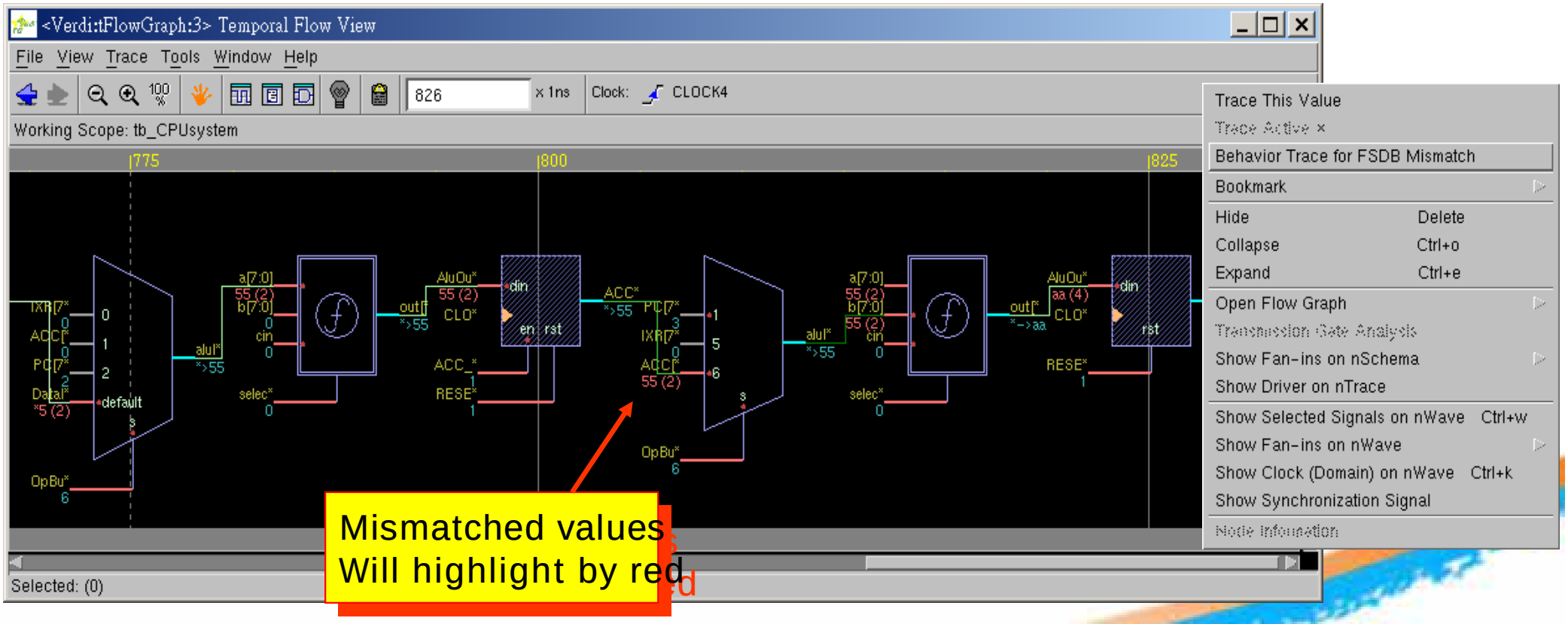
Behavior Trace for FSDB Mismatch
Trace Mismatch Cause (2/3)

- Specify the number of cycles/statement to trace back(default 3)
  - Use **Preferences → Temporal Flow View → Trace → Trace Mismatch**



Behavior Trace for FSDB Mismatch
Trace Mismatch Cause (3/3)

- Locate the register causing the mismatches
  - Select the problem signal in Temporal Flow View and use RMB → Behavior Trace for FSDB Mismatch.



SpringSoft
Accelerating Engineers

Lab5-2 (1/2)

- Use "Behavior Trace for FSDB Mismatch" to locate the root cause
  - Verdi -f run.f -ssf bug.fsdb&
    - **Get** signal "tb_CPUsystem.i_CPUsystem.i_CPU.Aluout" in nWave
    - Use waveform compare to compare above signal for bug.fsdb and good.fsdb
      - When does the first mismatch occurs? 777ns
    - **RMB** on signal Aluout @777 (bug.fsdb) and "**Create Temporal Flow View**"
    - **Select** Cycle-Based and click **OK**
    - In temporal flow view (TFV), RMB on signal "a" and invoke "**Behavior Trace for FSDB Mismatch**"
      - Update.. the "good.fsdb" for 2nd FSDB
    - Verdi will show the mismatch path and show different value first_fsdb(second_fsdb) in TFV

Lab5-2 (2/2)

- In TFV, RMB on mux input "IntDataIn" and "**Behavior Trace for FSDB Mismatch**"
  - change the "Trace Back Cycles/Statement at a Time" to "10"
  - It will trace to mprom_out @575
- "**Behavior Trace for FSDB Mismatch**" for mprom_out
  - It will stop on Memory
- In TFV, **double click** on signal "microrom"
  - The memory initial file will show on nTrace
- Open file "./memory/microrom.dat" and "./memory/microrom_fix.dat" and compare the initial value at address 1
  - The initial value difference cause the mismatch on signal Aluout @777 and other time point

Reduce Simulation Time and FSDB Size

- Common Problem
  - Designs are getting larger such that simulation lasts too long and results into a huge-size FSDB
- SpringSoft Solution
  - Use **System Tasks** to dump partial FSDB based on user-specified condition
  - Use **Virtual File** to watch several split FSDB more conveniently
  - Use **Siloti** to dump the minimum set of signals but still retain full visibility

System Tasks

- $fsdbDumpvars
  - Dump full or partial FSDB based on user-specified condition
- $fsdbAutoSwichDumpfile
  - Split FSDB size automatically when simulation
- $fsdbDumpOff/$fsdbDumpOn
  - Dump concerned time span
- For more system tasks, refer to
  - The section of FSDB dumping commands in linking_dumping.pdf after Verdi2008.01
  - Appendix B in reference.pdf before Verdi2007.10

$fsdbDumpvars

- $fsdbDumpvars([level] [, module | var]*);
- Example:
  - $fsdbDumpvars;
  - $fsdbDumpvars(0, system);
  - $fsdbDumpvars(1, system.i_cpu, system.i_cpu.i_PCU.net1);

$fsdbAutoSwitchDumpfile

- $fsdbAutoSwitchDumpfile(File_Size, "FSDB_name", Number_of_Files [, "log_filename"]);
  - Filesize unit: Mbytes
  - The minimum file size of the FSDB file is 2M
  - Number_of_Files: 0 means no limit
- Example:

```
initial
  begin
    $fsdbAutoSwitchDumpfile(200,"my.fsdb",10);
    $fsdbDumpvars;
  end
```

$fsdbDumpon and $fsdbDumpoff

- Used to dump on and dump off
- Example

```
initial
     begin
      $fsdbDumpvars;
      #1000
      $fsdbDumpoff;
      #1000
      $fsdbDumpon;
      #1000
      $finish;
    end
```

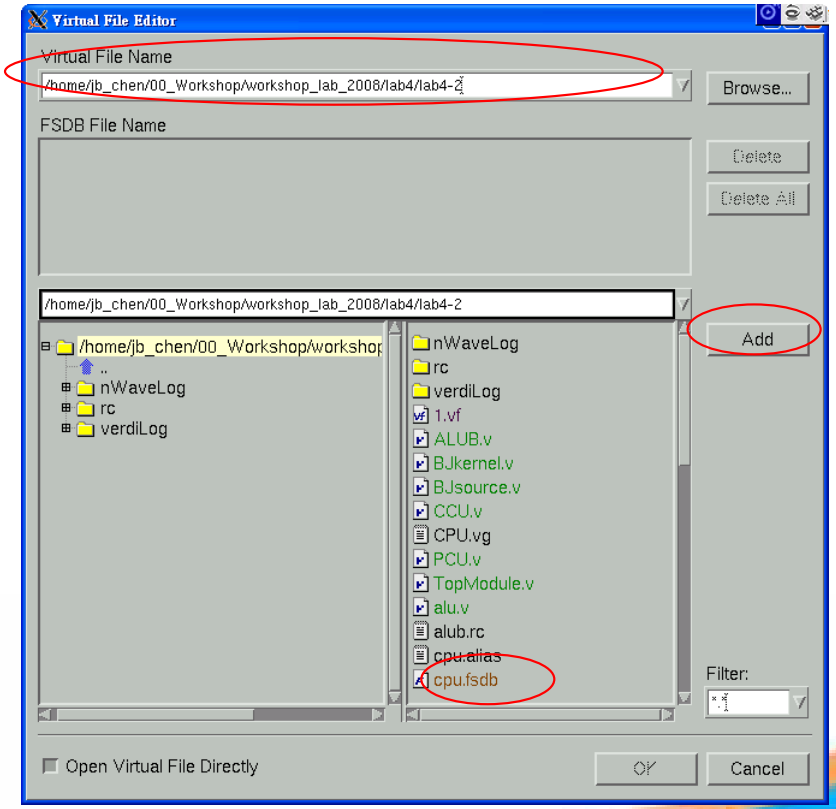
- Dump time span 0~1000,2000~3000

Lab6-1

- Use System Tasks to Reduce Simulation Time and FSDB Size
  - Edit the test bench file "system.v"
  - How to only dump signal below scope system.i_cpu from time 0ns~3000ns and 5000ns ~10000ns by using the following system task
    - $fsdbDumpvars;
    - $fsdbDumpoff
    - $fsdbDumpon
  - Dumping result and example source code
    - verilog.fsdb / system.v.ok
  - Use nWave to open verilog.fsdb to see the dumping result
    - Only contains signals below scope system.i_cpu

Virtual File Editor

- Merge multiple FSDB files to one virtual file.
  - FSDBs generated by system tasks, **fsdbDumpvars, fsdbAutoSwitchDumpfile, fsdbDumpon/fsdbDumpoff**
- In **nWave**, invoke **File →Edit Virtual File** to Edit Virtual File form.
- The resulting virtual file can be used in the Open File form.

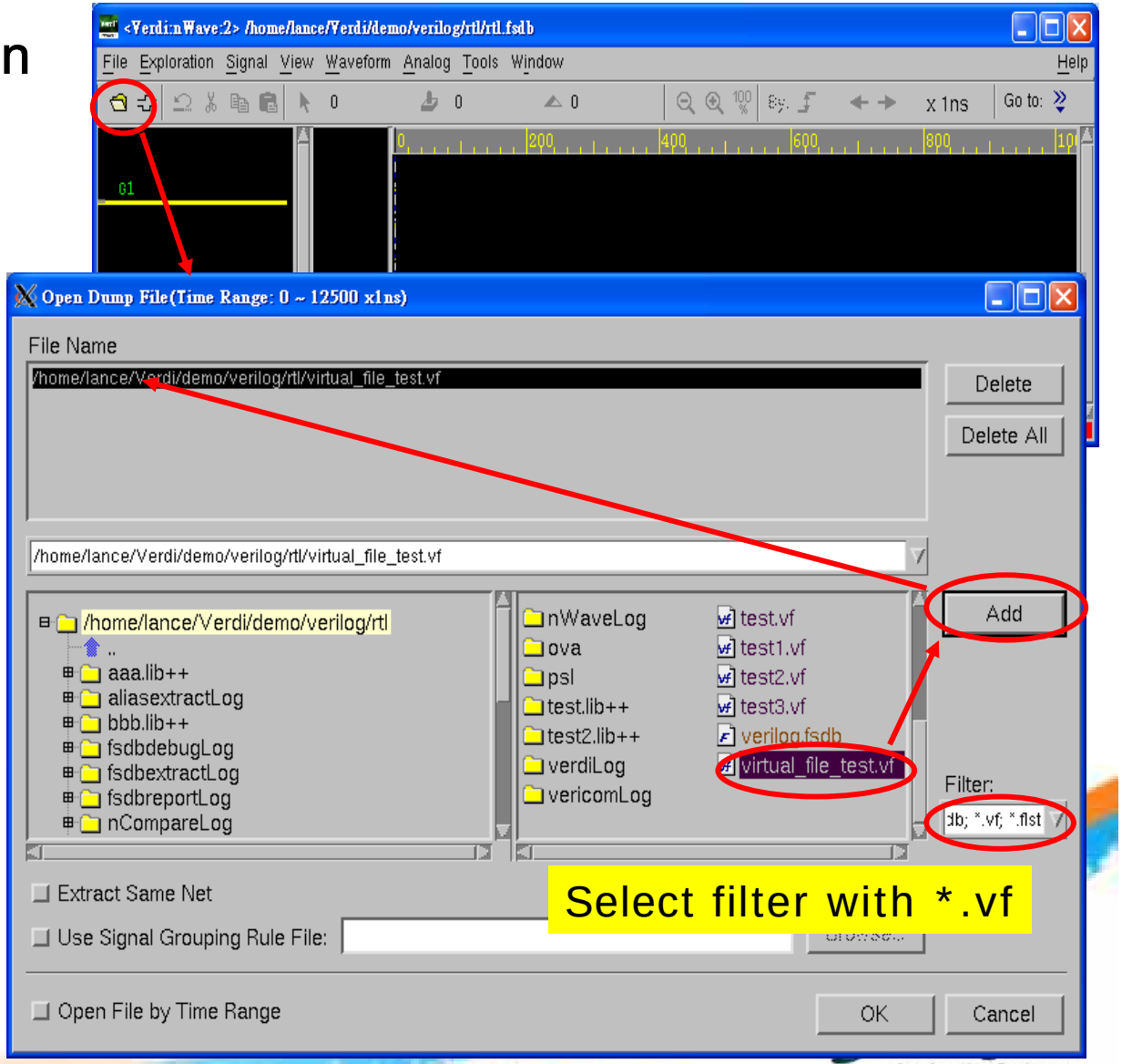


Load FSDB by Virtual File

- Command line:

  % verdi –ssf <virtual_file>.vf

- In nWave, use **File →Open**
  - Filter includes *.vf
  - Select the virtual file.
  - Click OK.



Lab6-2 (1/2)

- Use Virtual File Editor to manage different FSDB files by time range
  - Invoke nWave to open different FSDBs
    - **File** → **Open** time1.fsdb (0ns ~5000ns)
      - **Get** signal "system.clock", "system.addr" and "system.data"
      - Contain signals from 0ns ~5000ns
    - **File** → **Open** time2.fsdb (5000ns ~17500ns)
      - Get signal "system.clock", "system.addr" and "system.data" again
      - Contain signals from 5000ns ~17500ns
    - **File** → **Close** → **All**
  - Use Virtual File Editor
    - **File** → **Edit Virtual File**
      - **Select** time1.fsdb and time2.fsdb → **click** "Add"
      - Specify Virtual File Name in Save As: **bytime.vf**
      - **Enable** "Open Virtual File Directly" → **click** "OK"
      - **Get** signal "system.clock", "system.addr" and "system.data" again
      - You can see the full range of simulation time (0ns ~ 17500ns)
    - **File** → **Close** → **All**

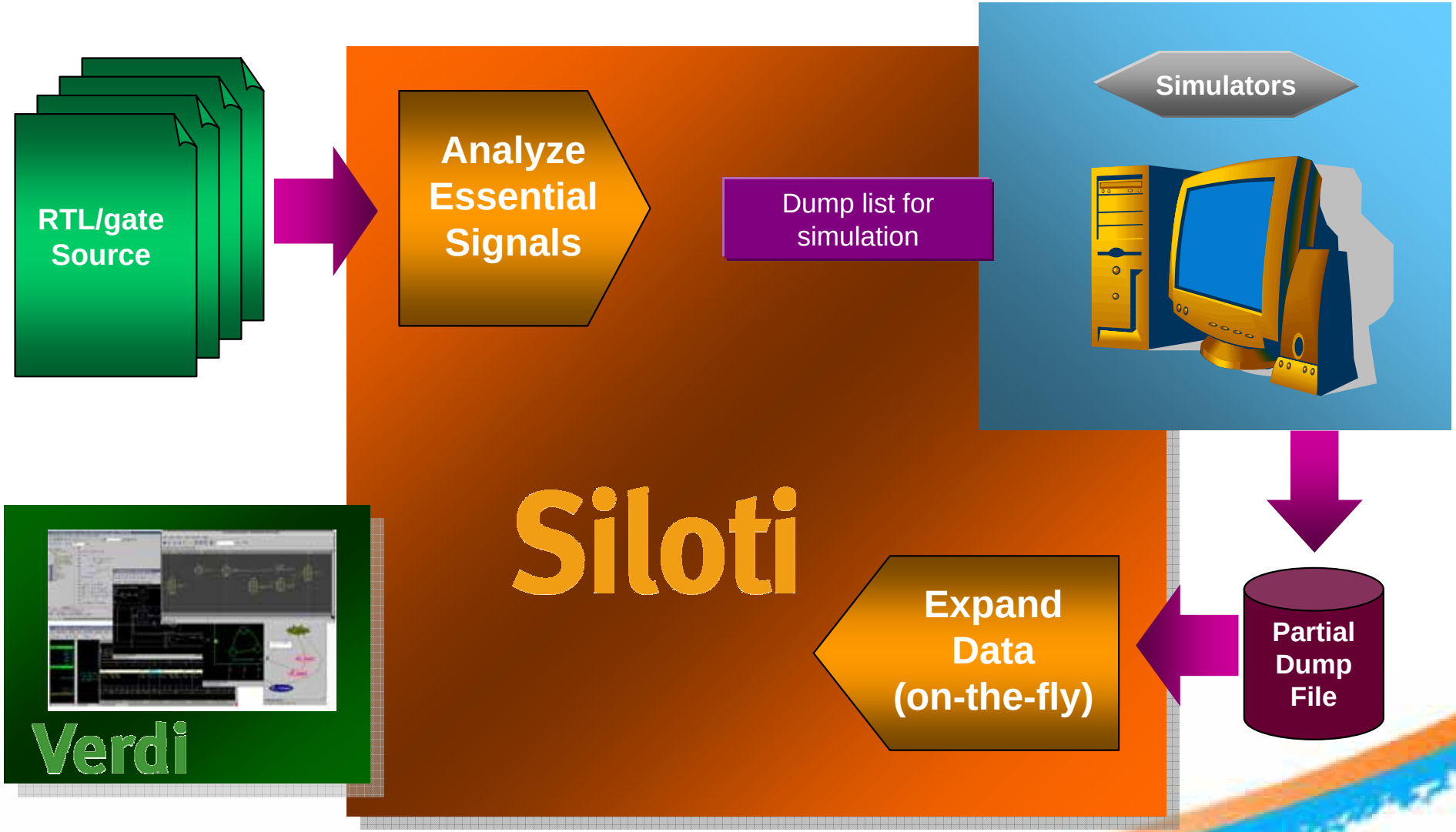
Lab6-2 (2/2)

- Use Virtual File Editor to manage different FSDB files by design hierarchy
  - Invoke nWave to open different FSDBs
    - **File → Open** pram.fsdb
      - Get signal "system.i_pram.clock" (only contains system.i_pram)
    - **File → Open** cpu.fsdb
      - Get signal "system.i_cpu.error" (contains the other scopes and no i_pram)
    - **File → Close → All**
  - Use Virtual File Editor
    - **File → Edit Virtual File**
      - **Select** pram.fsdb and cpu.fsdb → **click** Add
      - Specify Virtual File Name in Save As: **byscope.vf**
      - **Enable** "Open Virtual File Directly" → **click** OK
      - Get signal "system.i_pram.clock" and "system.i_cpu.error"
      - You can get signals of all scopes from different FSDBs
    - **File → Close → All**

Siloti – SimVE

- Common Problem
  - Designs are getting larger which causes increased simulation times
  - Data storage is at a premium
  - Can lose 1-2 days waiting for the next simulation
- SpringSoft Solution
  - Essential Signal Analysis engine determines minimum signals to dump for full visibility.
  - Data Expansion engine processes limited data and calculates values for combinational signals.
  - Significantly reduces dumping requirements while retaining full visibility.

Siloti – SimVE
How it Works

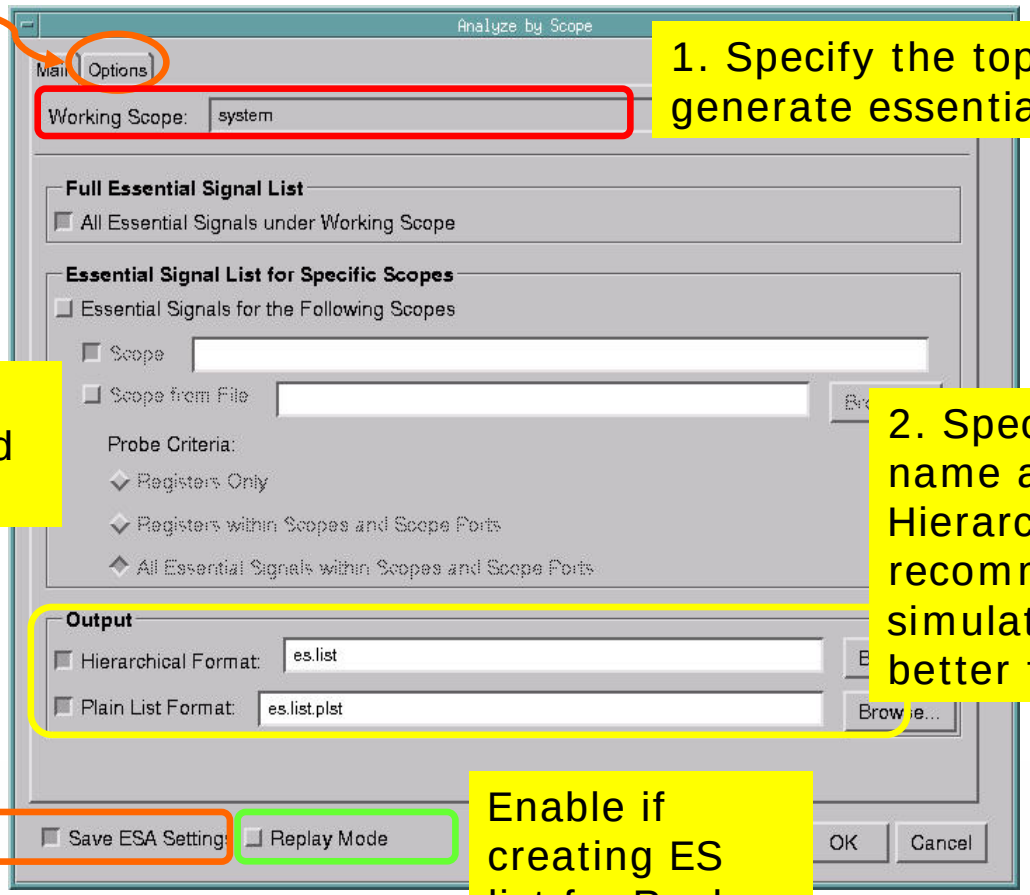


Patent Pending Technology

SpringSoft
Accelerating Engineers

Essential Signal Analysis – GUI

- In Siloti nTrace, invoke **Visibility → Essential Signal Analysis → Analyze by Scope**.



Click **Options** tab to see additional options.

1. Specify the top level scope to generate essential signals for.

Optional - specify multiple scopes and the probe criteria.

2. Specify the output file name and file format. Hierarchical format is recommended for simulation; plain list is better for readability.

Optional - enable to save settings to novas.rc.

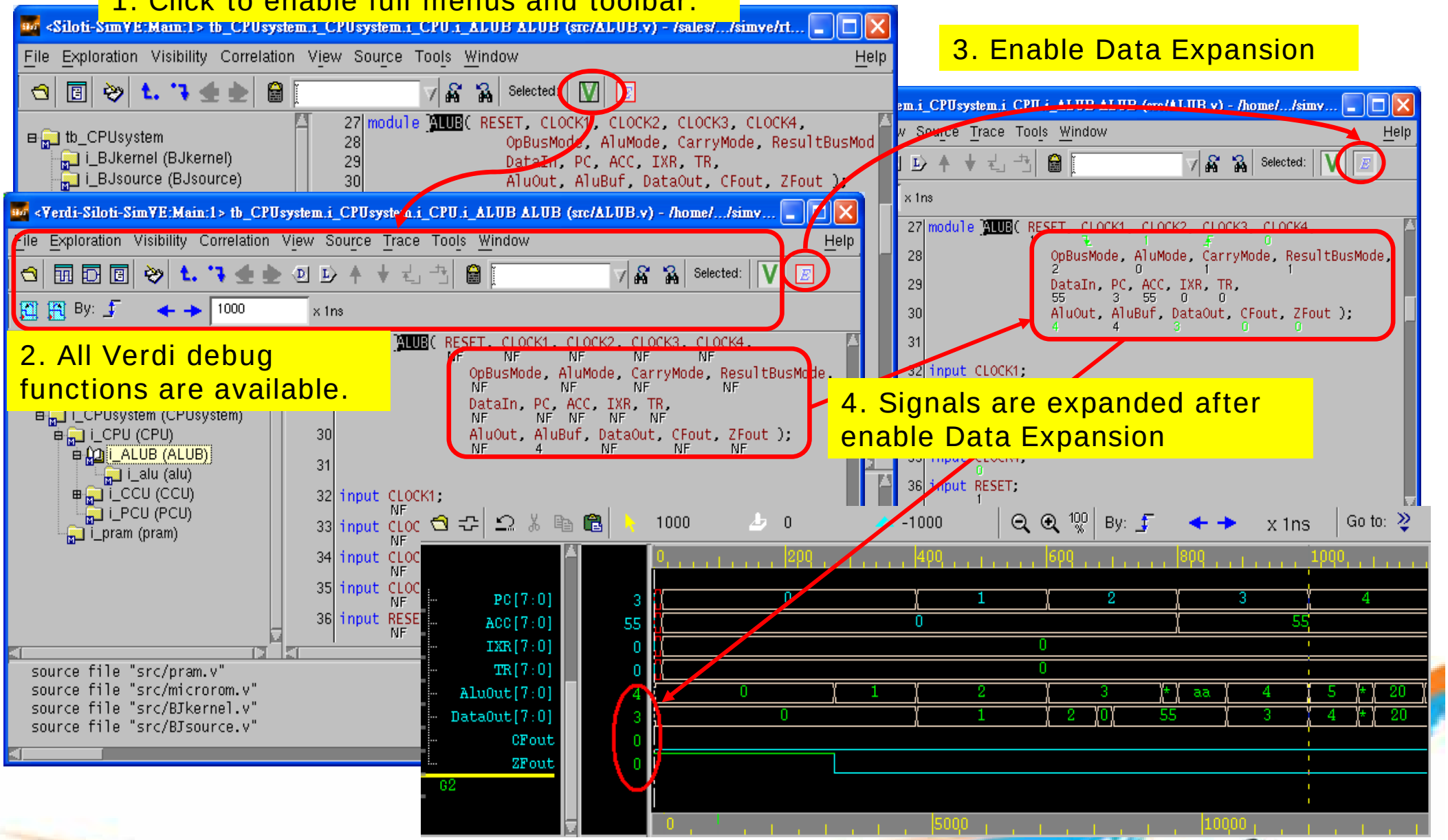
Enable if creating ES list for Replay.

Generate Essential Signal FSDB

- Replace $fsdbDumpvars with $fsdbDumpvarsES, e.g.
  - $fsdbDumpvarsES("es.list", "myfsdb.fsdb"); (Verilog)
  - fsdbDumpvarsES("es.list", "myfsdb.fsdb"); (VHDL)
- Run simulation to generate essential signal FSDB.

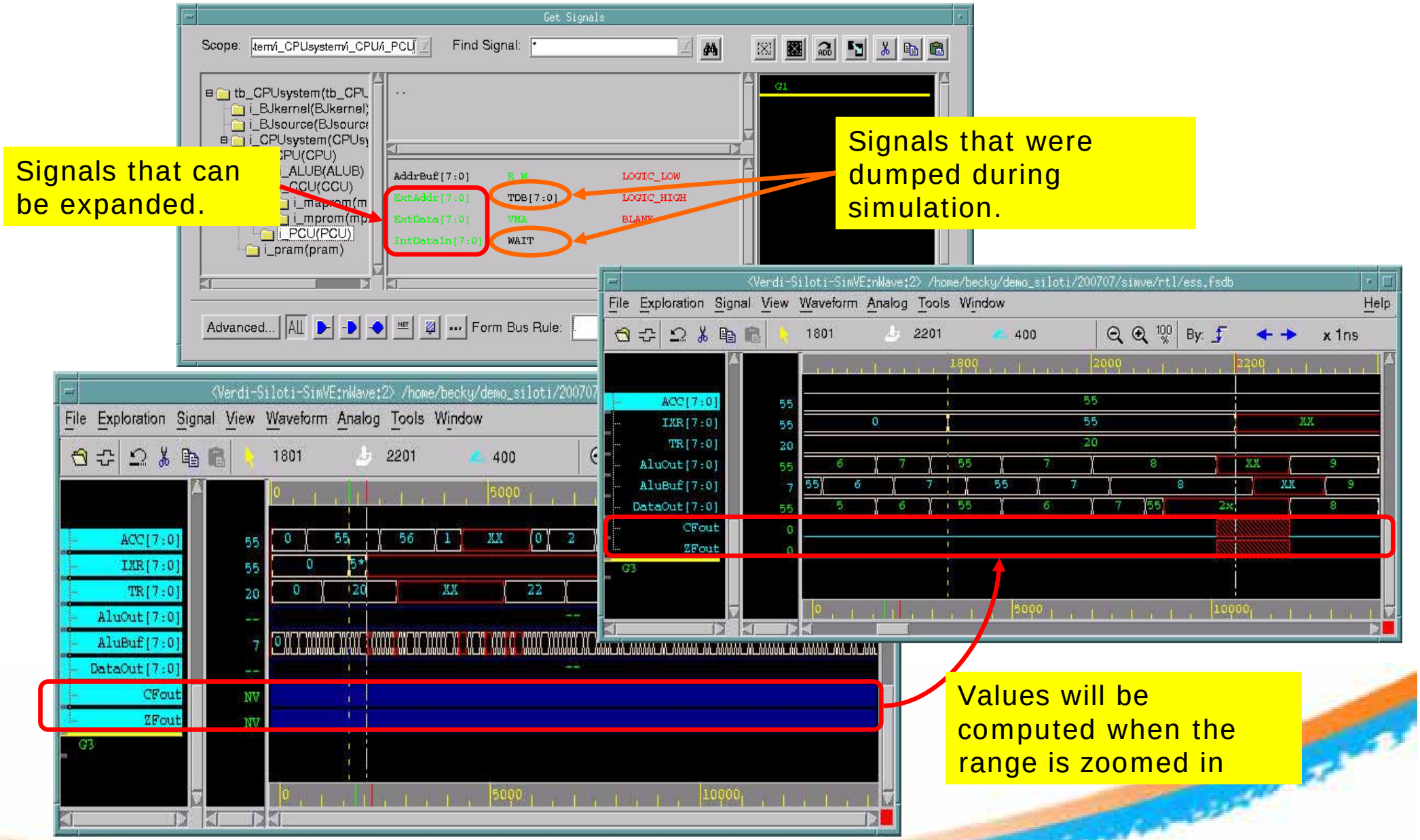
Data Expansion to Debug

1. Click to enable full menus and toolbar.



SpringSoft Accelerating Engineers

Data Expansion about Waveform



SpringSoft
Accelerating Engineers

Lab6-3 (1/2)

- Use Essential Signal Dumping to reduce simulation time and FSDB size
  - Generate essential signal list
    - esa -bas tb_CPUsystem -f run.f -all_eslist fullchip.list
      - The essential signals will be listed in "fullchip.list"
      - See the log file "./esaLog/siloti.log"
      - What is the percentage of essential signal (bits)? 6.09%(286/4696))
  - Use system task to dump essential signal FSDB
    - $fsdbDumpfile("ess.fsdb");
    - $fsdbDumpvarsToFile("fullchip.list");
  - Example source code
    - src/tb_CPUsystem.v
  - Run simulation to dump "ess.fsdb"

Lab6-3 (2/2)

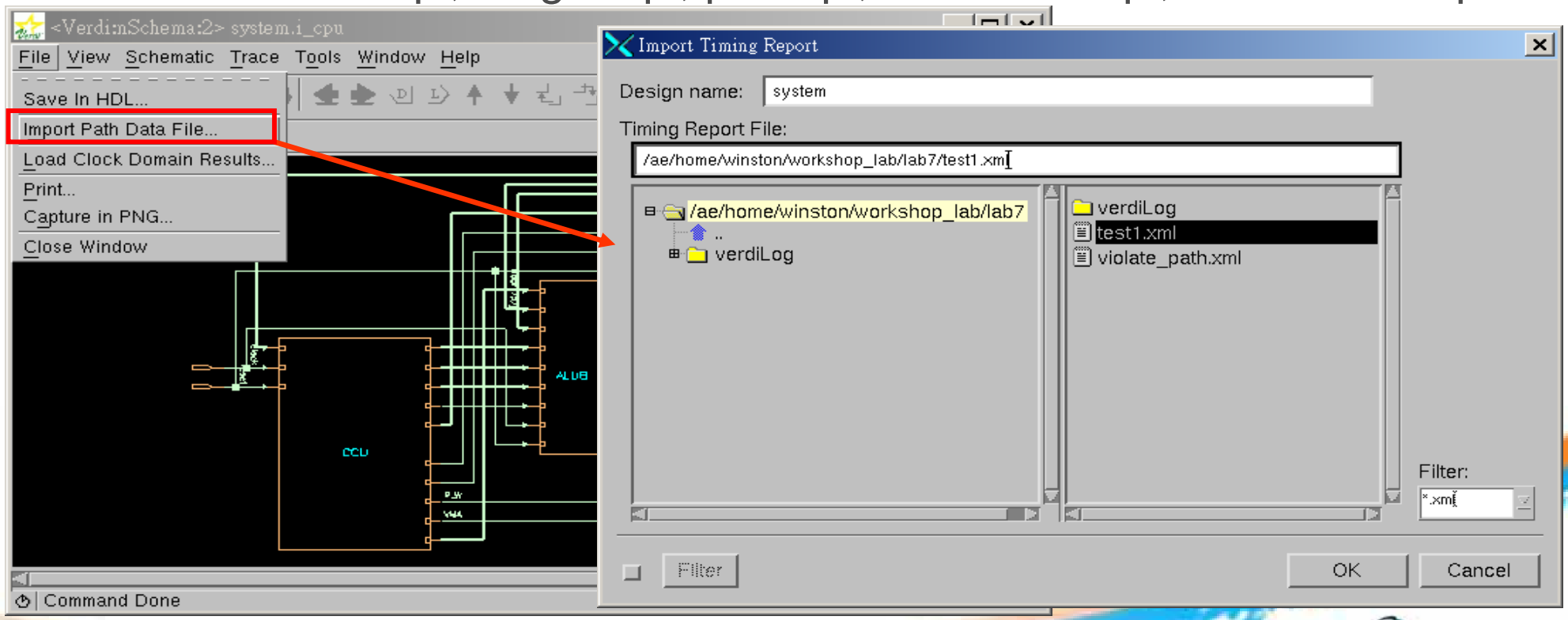
- Use Data Expansion to get the full visibility of design
  - Invoke Siloti
    - **siloti -f run.f -verdi -ssf ess.fsdb &**
  - Turn on Active Annotation in Siloti
    - Change scope to "tb_CPUsystem.i_CPUsystem.i_CPU"
    - **Source → Active Annotation**
      - There are many signals show "NF" (Not Found in FSDB)
    - **D&D** "tb_CPUsystem.i_CPUsystem.i_CPU" scope to nWave
    - A window pops and shows "List of Unrecognized Signal"
  - Turn on Data Expansion in Siloti
    - **Tools → Visibility → Data Expansion → Enable Data Expansion**
    - Do behavior analysis and set the working scope to "tb_CPUsystem" and **click** "OK"
      - The annotated signals will be calculated on demand
    - **D&D** tb_CPUsystem.i_CPUsystem.i_CPU scope to nWave
      - The calculated signal will be displayed in nWave.

Debug Timing Problem after Post-Sim

- Common Problem
  - Difficult to trace the transition resulted from delay of combinational logics
  - Can't combine timing information and function into an isolated debug platform
- SpringSoft Solution
  - Import timing reports by multiple STA tools into Verdi to debug critical paths
  - Use Temporal Flow View to automatically find out the root cause of a wrong transition

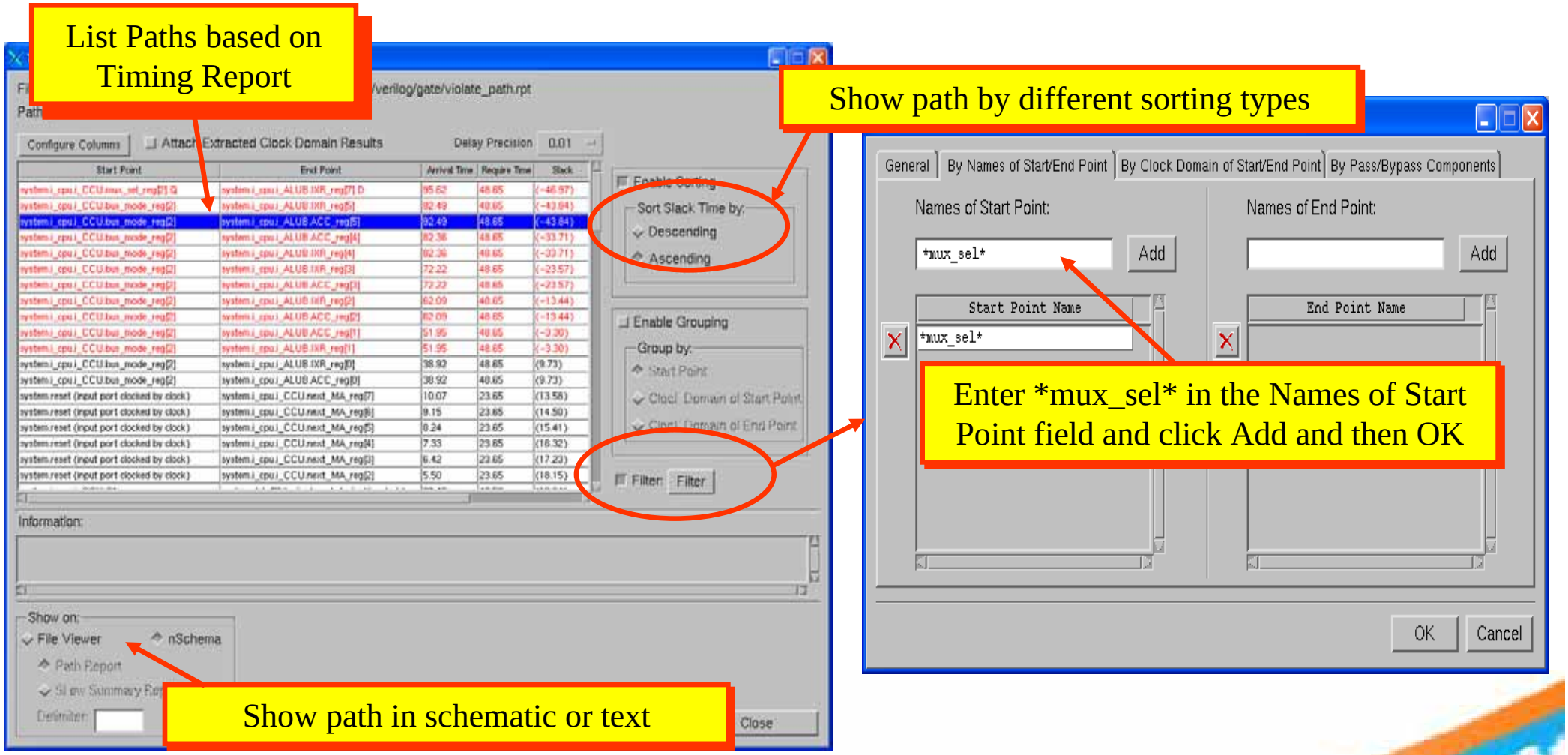
Import Timing Report

- Use utility to translate report file into Verdi readable xml format.
  - Ex: astrol2Xml.pl prime_time.rpt –o <output_file_name>
  - File → Import Path Data File... → Select XML file
- Support scripts:
  - pt2Xml.pl, AMBIT.pl, astrol2Xml.pl, DC.pl, EinsTimer.pl, HAL2Xml.pl, magma.pl, pearl.pl, RTLC2Xml.pl, and SE2Xml.pl



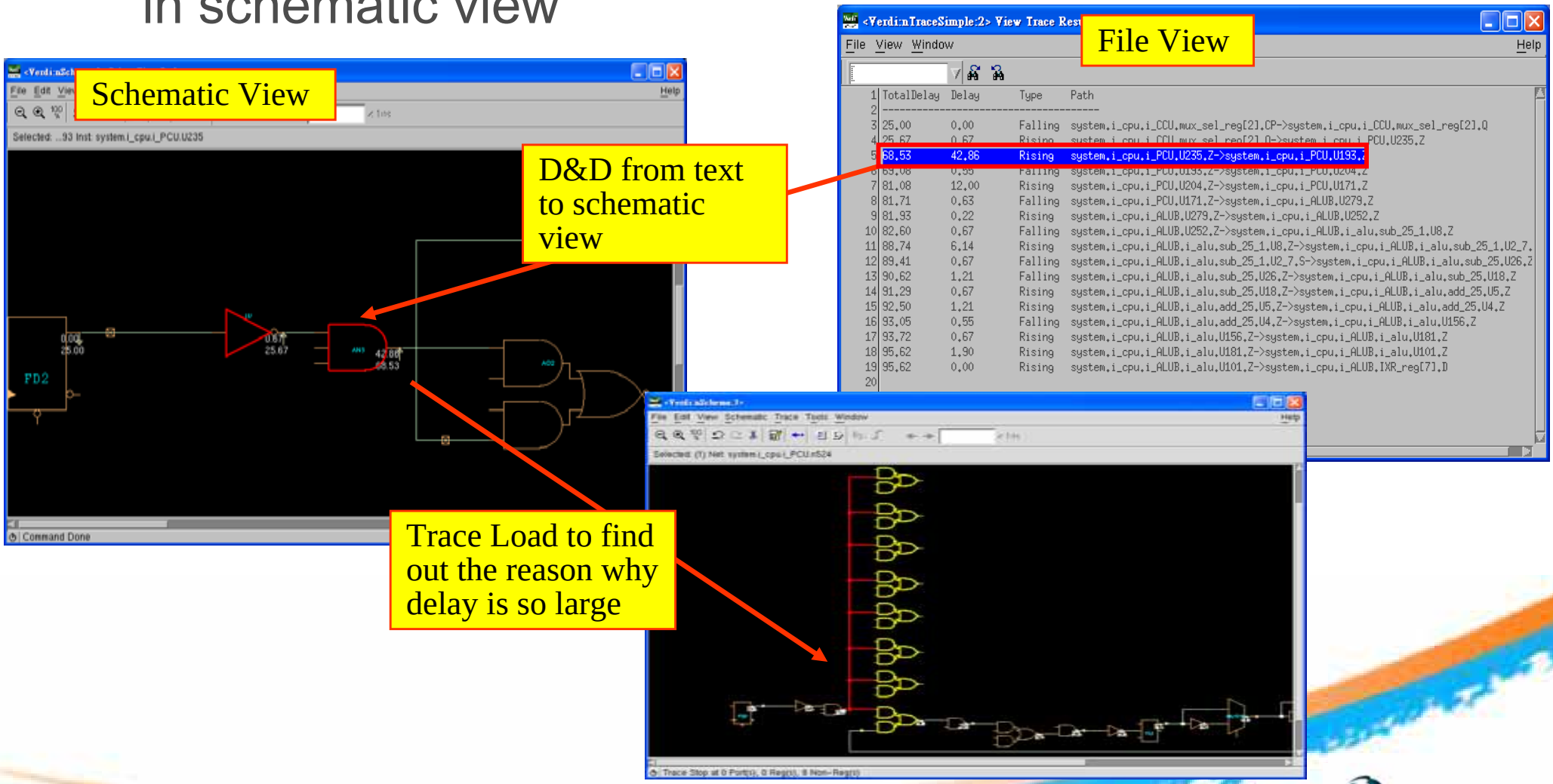
SpringSoft
Accelerating Engineers

Show Timing Report Paths(1/2)



Show Timing Report Paths(2/2)

- Show detail path information in schematic or text
- Support Drag & Drop interested delay from text to highlight in schematic view

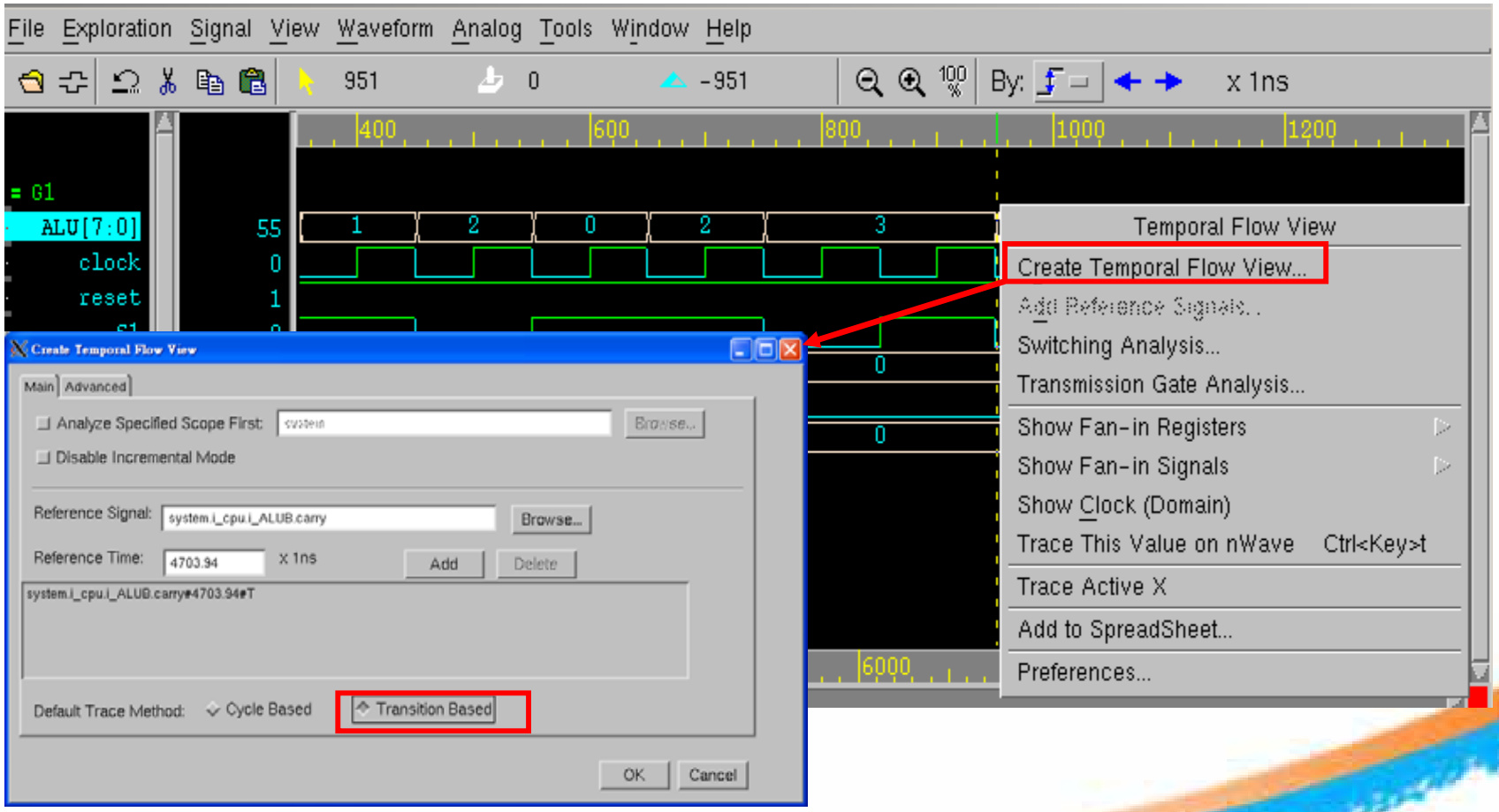


Lab7-1

- Convert timing report and import it to debug
  - Invoke Verdi
    - ./RUN
  - (nTrace or nSchema) **File → Import Path Data File → Select the violate_path.rpt** file
  - Enable the option: "**Enable Sort**" in Timing Report window
  - Click "**Ascending**" on "Sort Slack Time By:"
  - Investigate into the most serious timing-violated path
    - Show this path on nSchema
    - Show this path on File Viewer
    - **D&D** the 42.860 delay from File Viewer into nSchema
    - Debug the reason of this huge delay on schematic window
      - **Tools → New Schematic → Connectivity**

Trace Triggering Path (1/5)

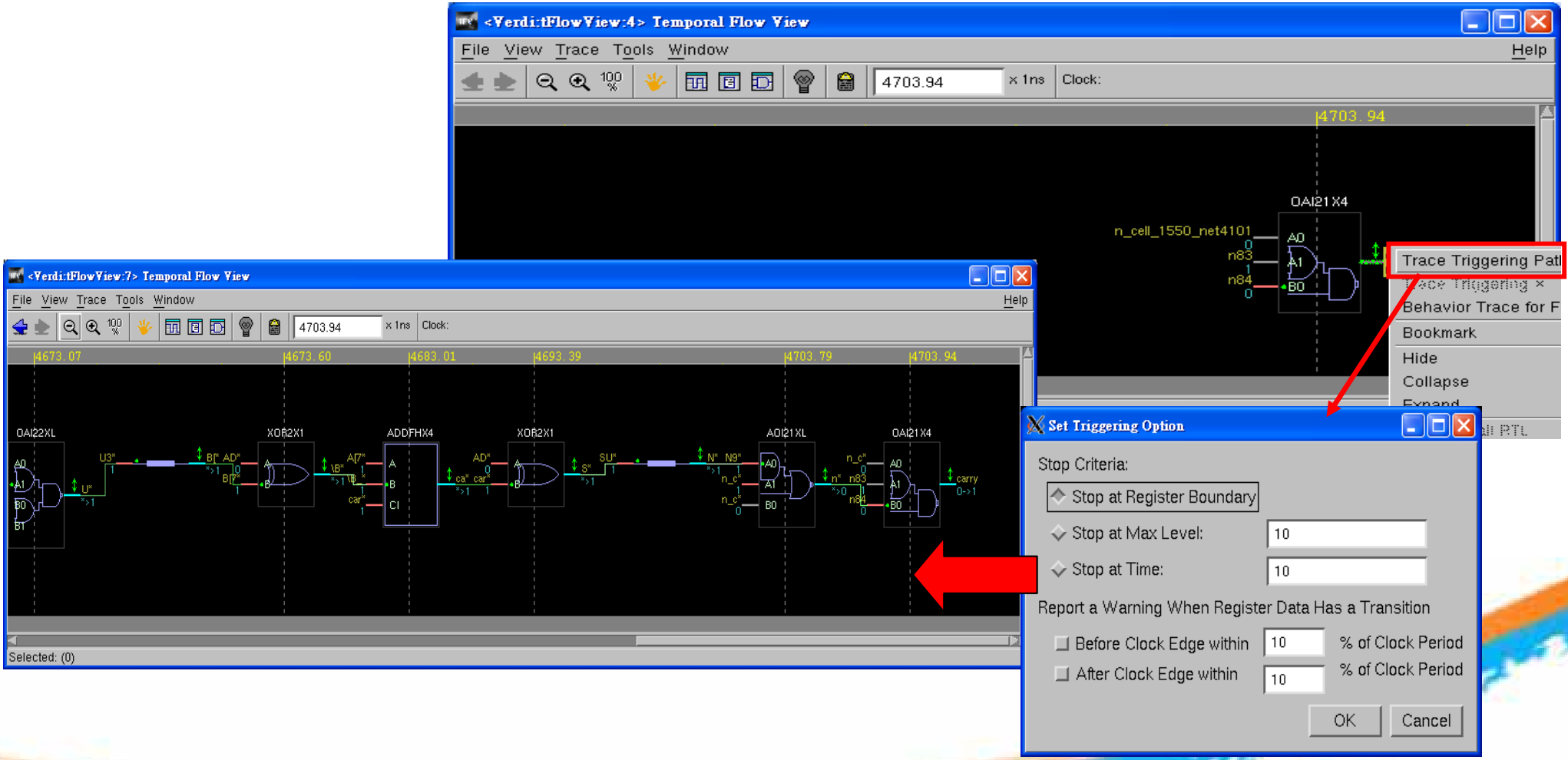
- Purpose:
  - Help users to debug timing or glitch problems on gate-level designs



SpringSoft
Accelerating Engineers

Trace Triggering Path (2/5)

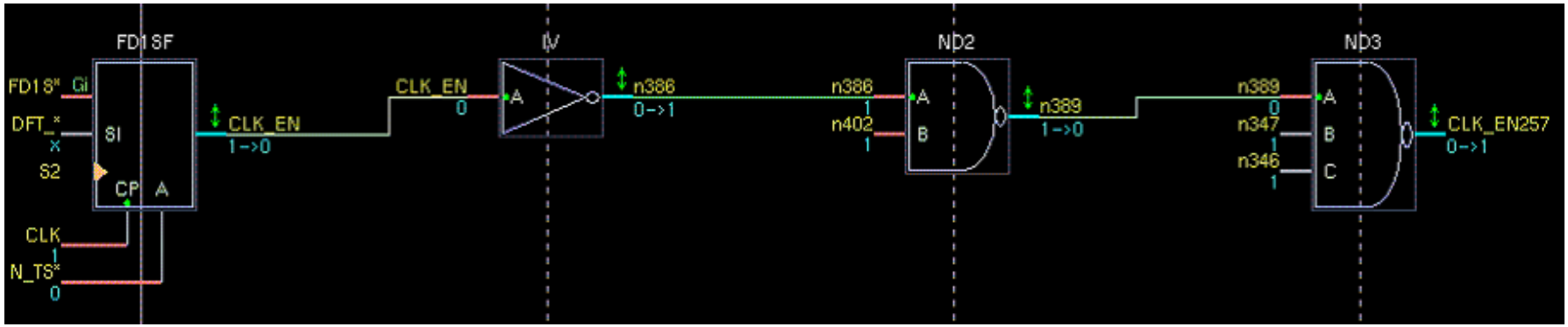
- In the Temporal Flow View, invoke the right mouse button menu on the signal wanted to trace, and select Trace Triggering Path from the menu



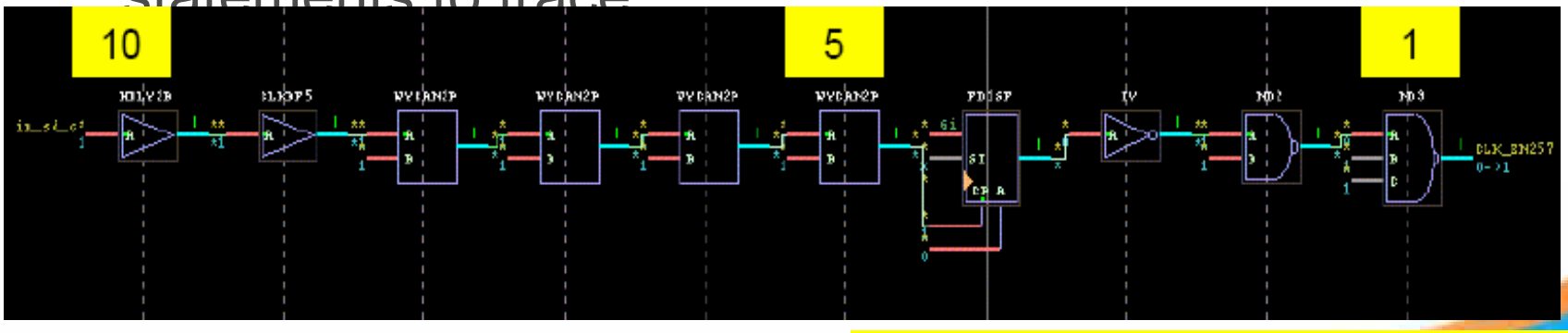
SpringSoft
Accelerating Engineers

Trace Triggering Path (3/5)

- There are three options to select to determine when the tracing command should stop
  - Stop at register boundary: Trace will stop at the registers



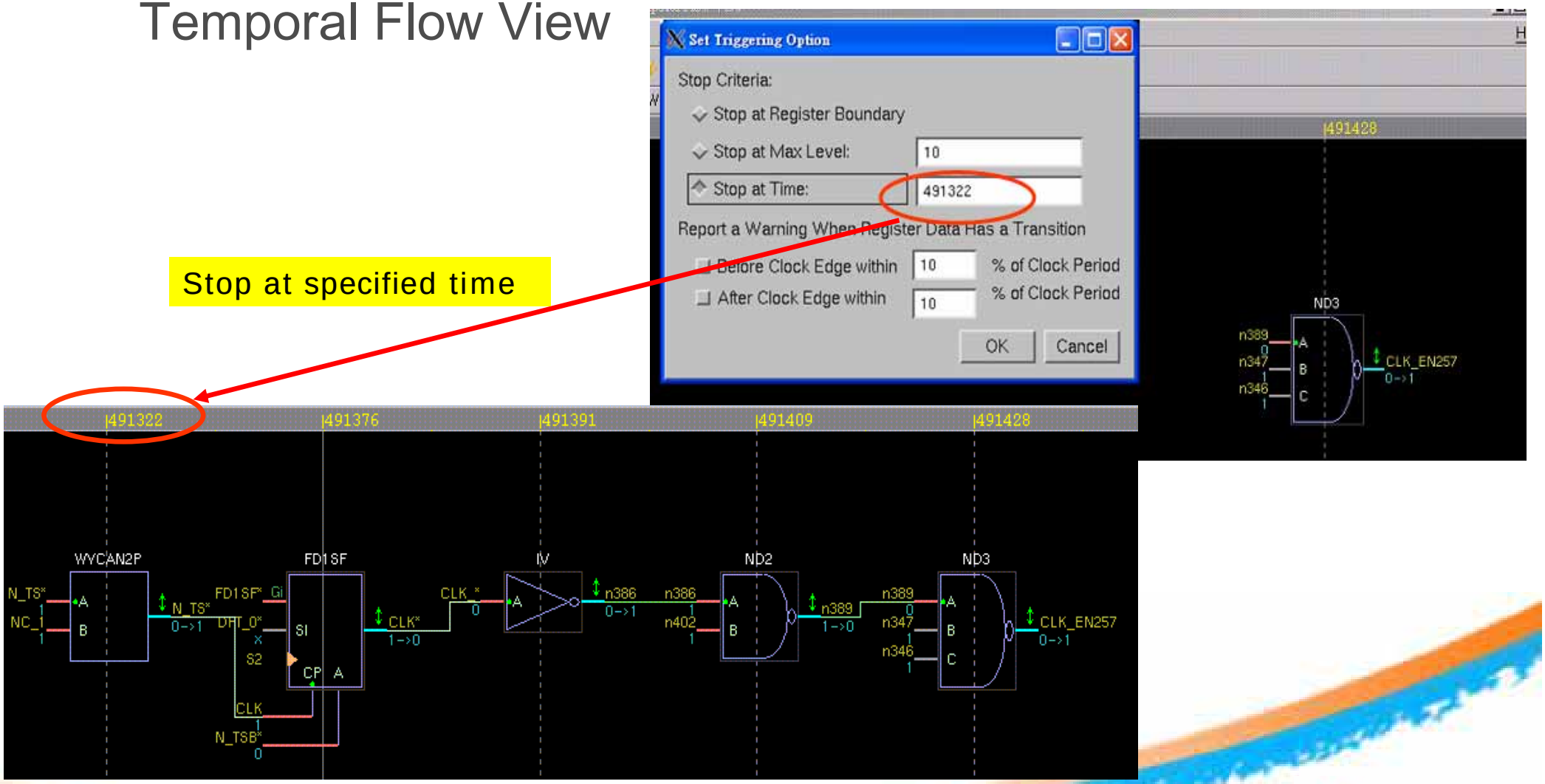
  - Stop at max level: Specify the maximum levels of statements to trace



Results of Stop at Max Level=10 Option

Trace Triggering Path (4/5)

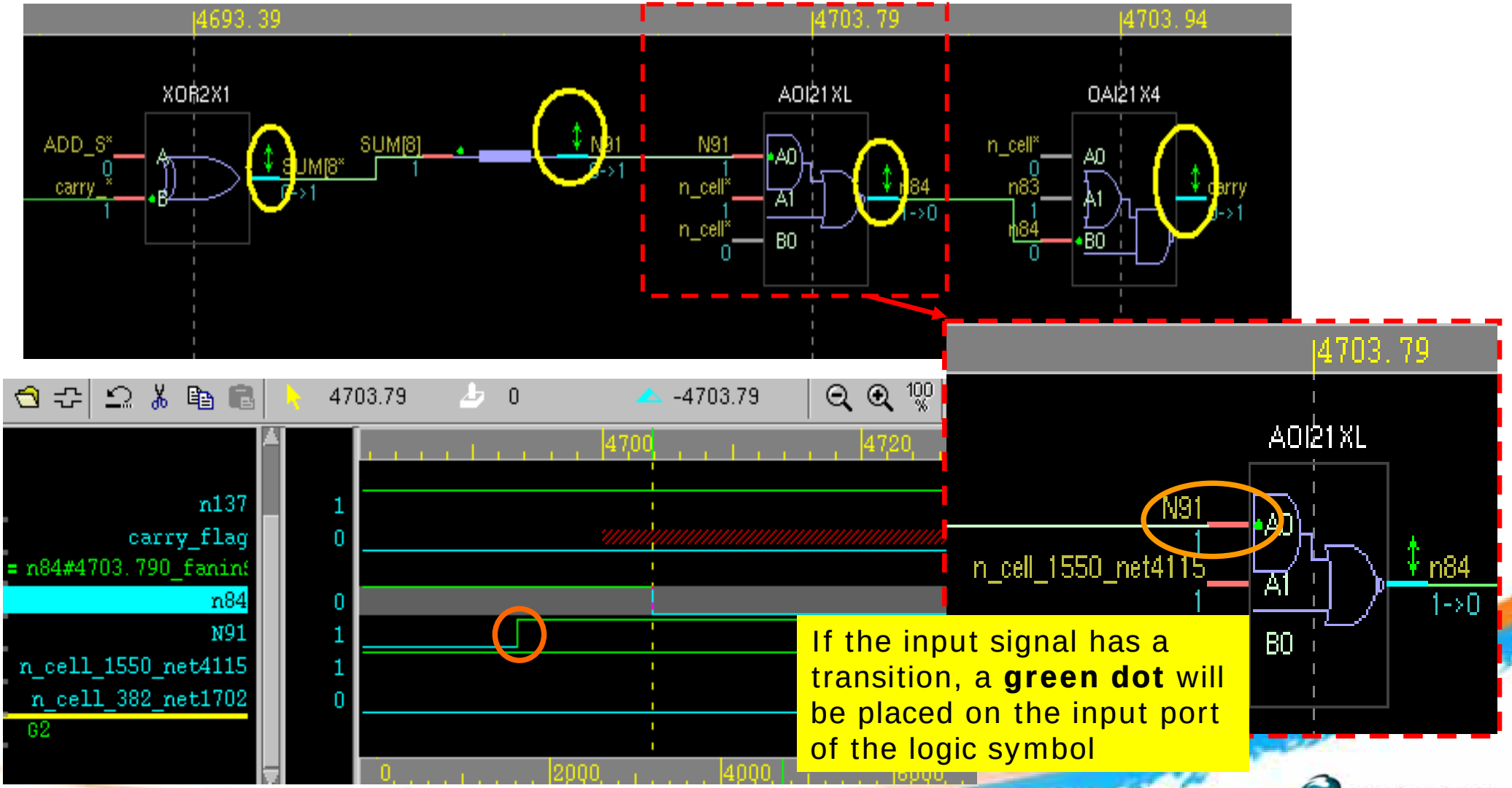
- Stop at time: Specify the time where trace command should stop. The time unit is the same as the x-axis of the Temporal Flow View



SpringSoft
Accelerating Engineers

Trace Triggering Path (5/5)

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



Lab7-2 (1/2)

- Use Trace Triggering Path to Automatically find out the root cause of wrong transition caused by timing issue
  - Invoke Verdi at gate directory

    ./RUN
  - Select "carry_flag" in both waveform windows
  - In the gate nWave (with gate.fsdb), use **Tools → Waveform Compare → Compare 2 Signals** to compare "carry_flag" from two FSDB files (gate v.s. rtl)
  - Use the search mechanism to find the mismatch
    - The first mismatch starts at time 4700 ns
  - Select the "carry" and change cursor to transition at time 4703.94
  - Click **RMB → Create Temporal Flow View**
  - Enable "Transition Based", then click **OK**
  - In the Temporal Flow View, select the "carry" output port and invoke **RMB → Trace Triggering Path**

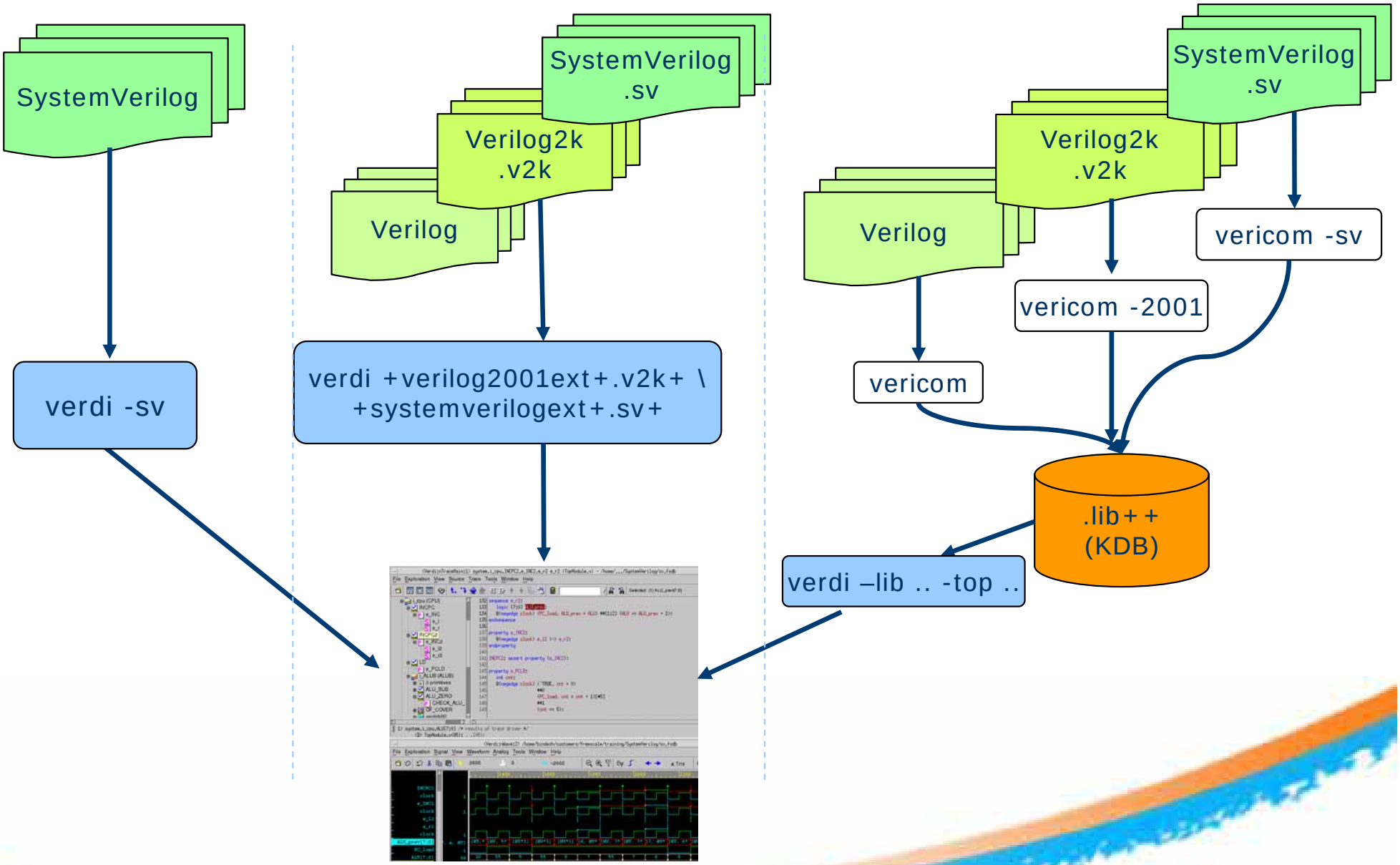
Lab7-2 (2/2)

- In the "Set Triggering Option" form, select "**Stop at Register Boundary**" as the stop criteria and click **OK**
- Scroll to show the tracing path and then invoke **View** → **Fit Time** to display the entire path
  - You can see the time from the first FF trigger to the carry trigger is 53.94ns
- Zoom in around the FF at time 4650.39, **select** net "n10" and **press** Ctrl-k
  - This adds the clock signal to the waveform
  - You can compare this FF's clock "clock" to the carry_flag FF's clock "T2" to see there's 50.82ns to meet time
  - The carry FF's clock exceeds by ~3ns

Debug SystemVerilog Design

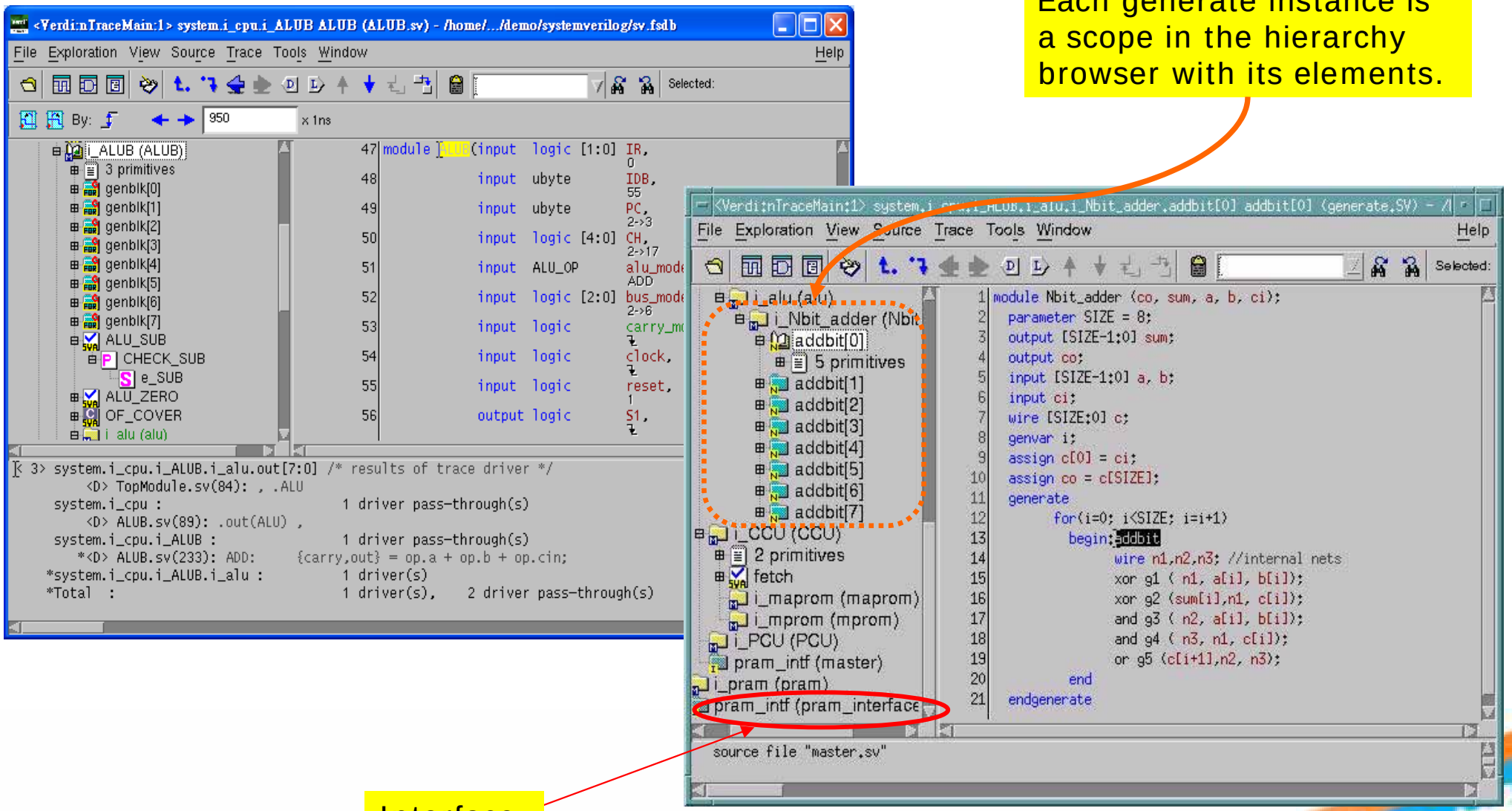
- Common Problem
  - Need an efficient method to study and/or debug SystemVerilog source codes with simulation results
  - Require an environment for analysis between different languages designs
- SpringSoft Solution
  - Able to compile and import the SystemVerilog design into Novas debug platform
  - Support generate construct
  - View SystemVerilog simulation results in nWave, annotate on the source codes, or automatically trace paths in Temporal Flow View

Import SV Design into Verdi



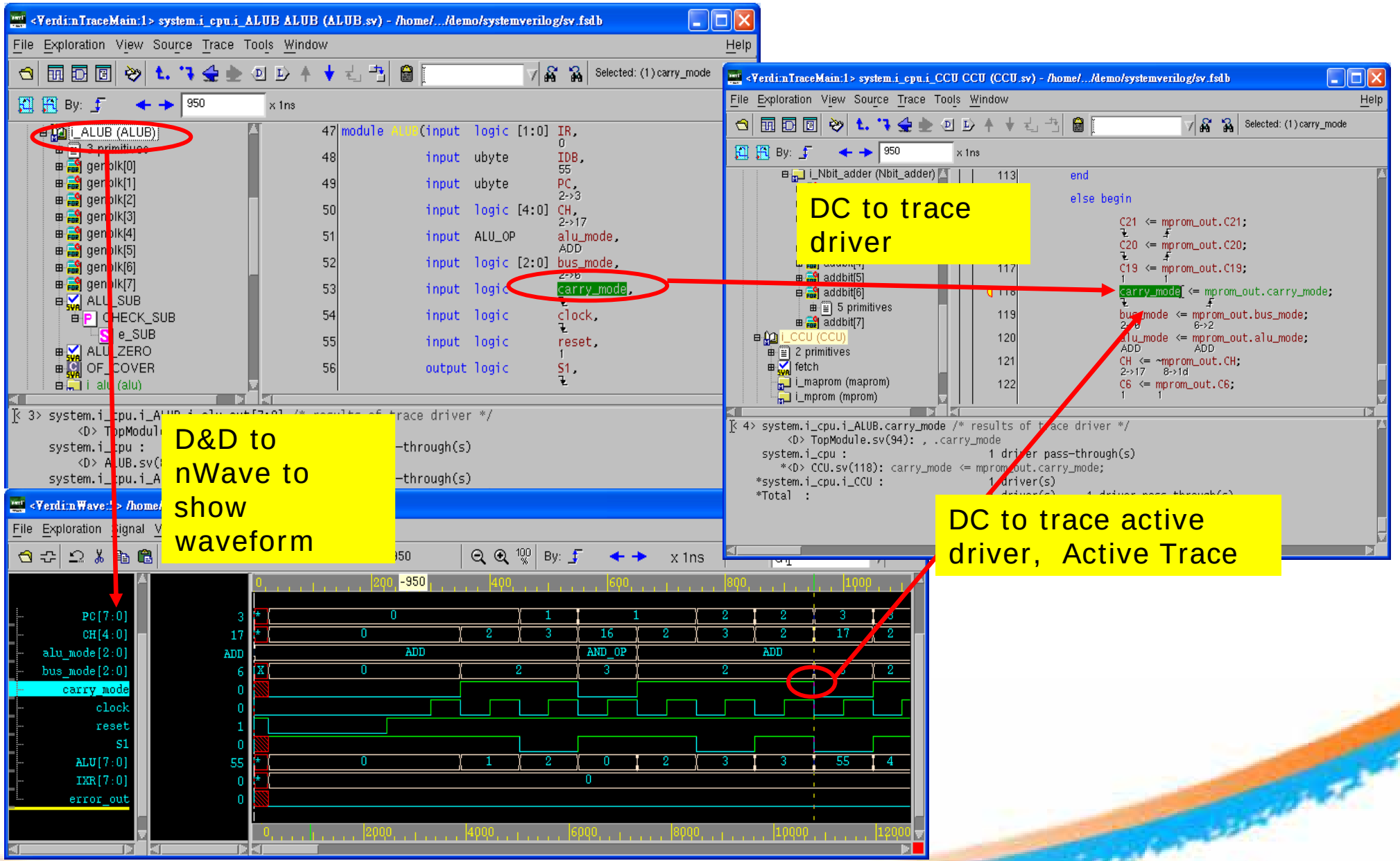
View SystemVerilog in nTrace

Each generate instance is a scope in the hierarchy browser with its elements.



SpringSoft Accelerating Engineers

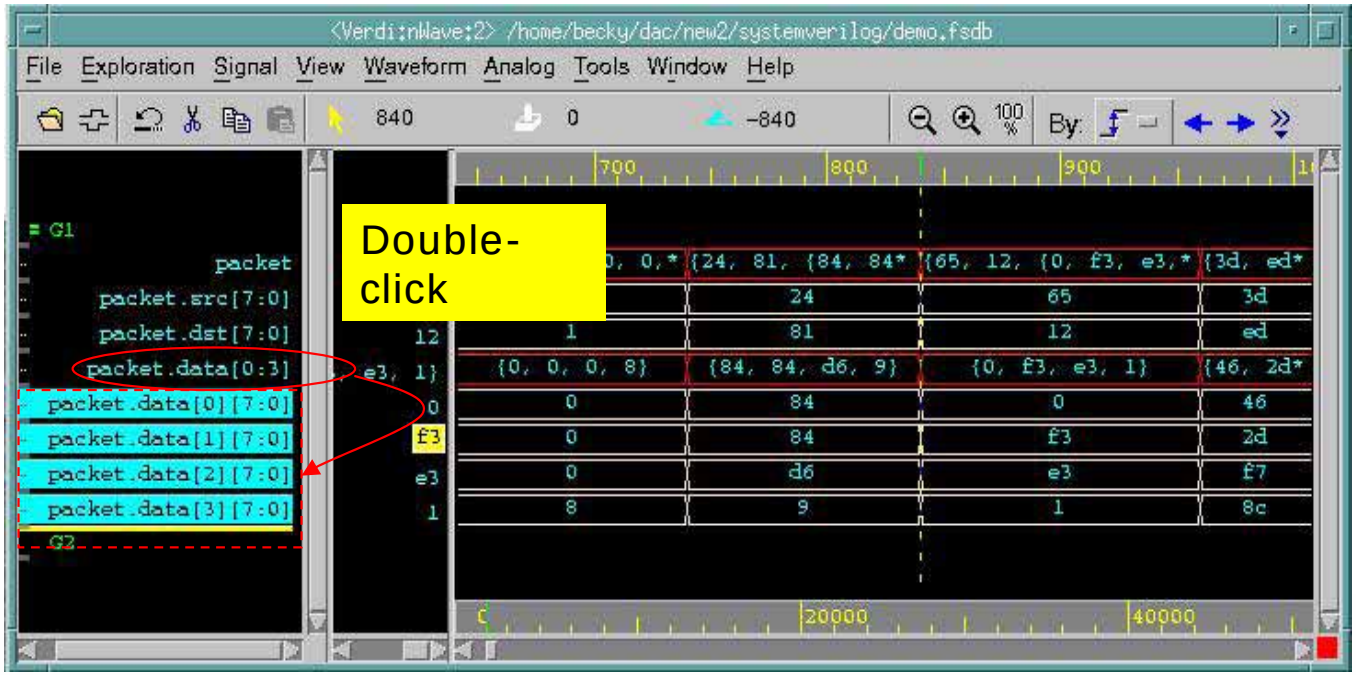
Trace Driver and Active Trace



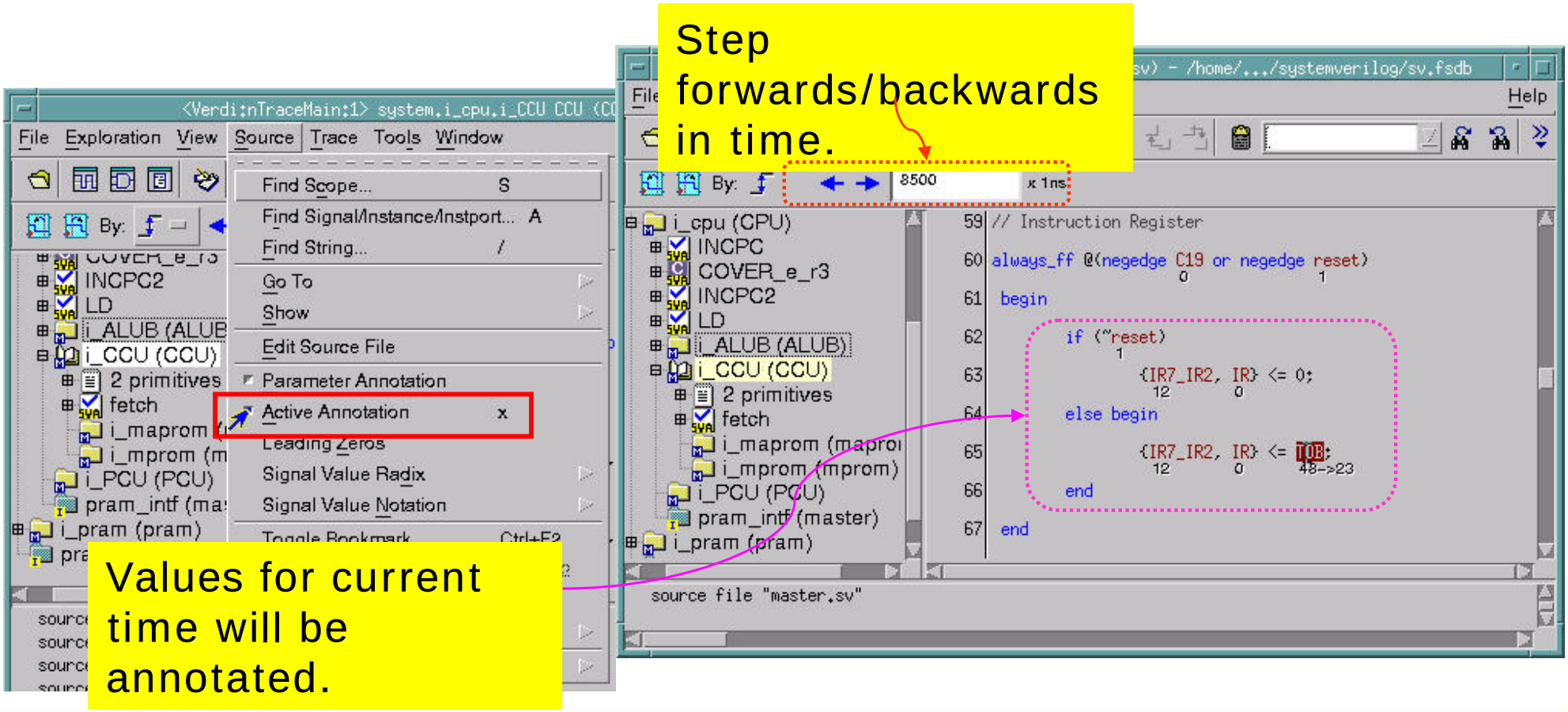
SpringSoft
Accelerating Engineers

View Simulation Results – Waveform

- Elements of complex signals can be expanded by double-clicking the name in nWave and shown as individual signals.

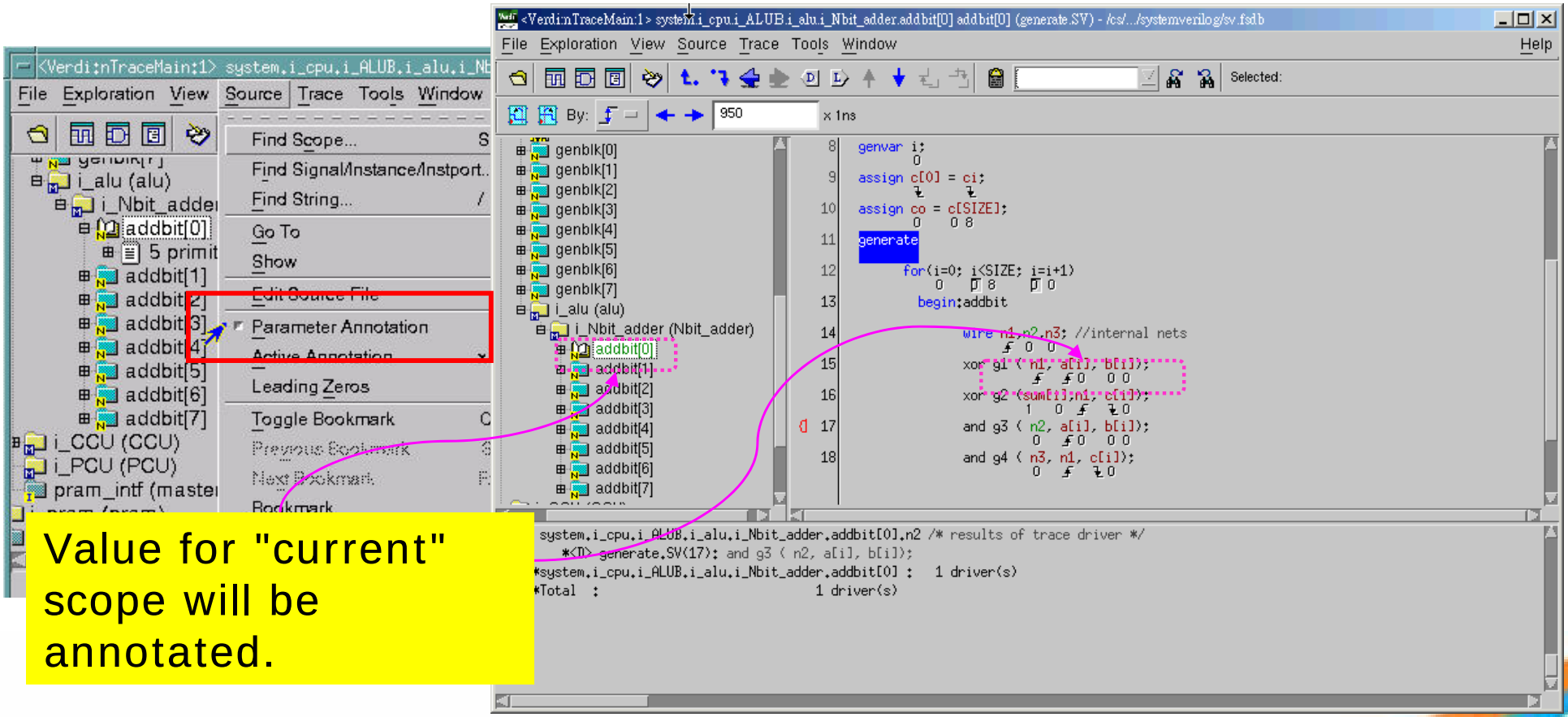


View Simulation Results – Active Annotation



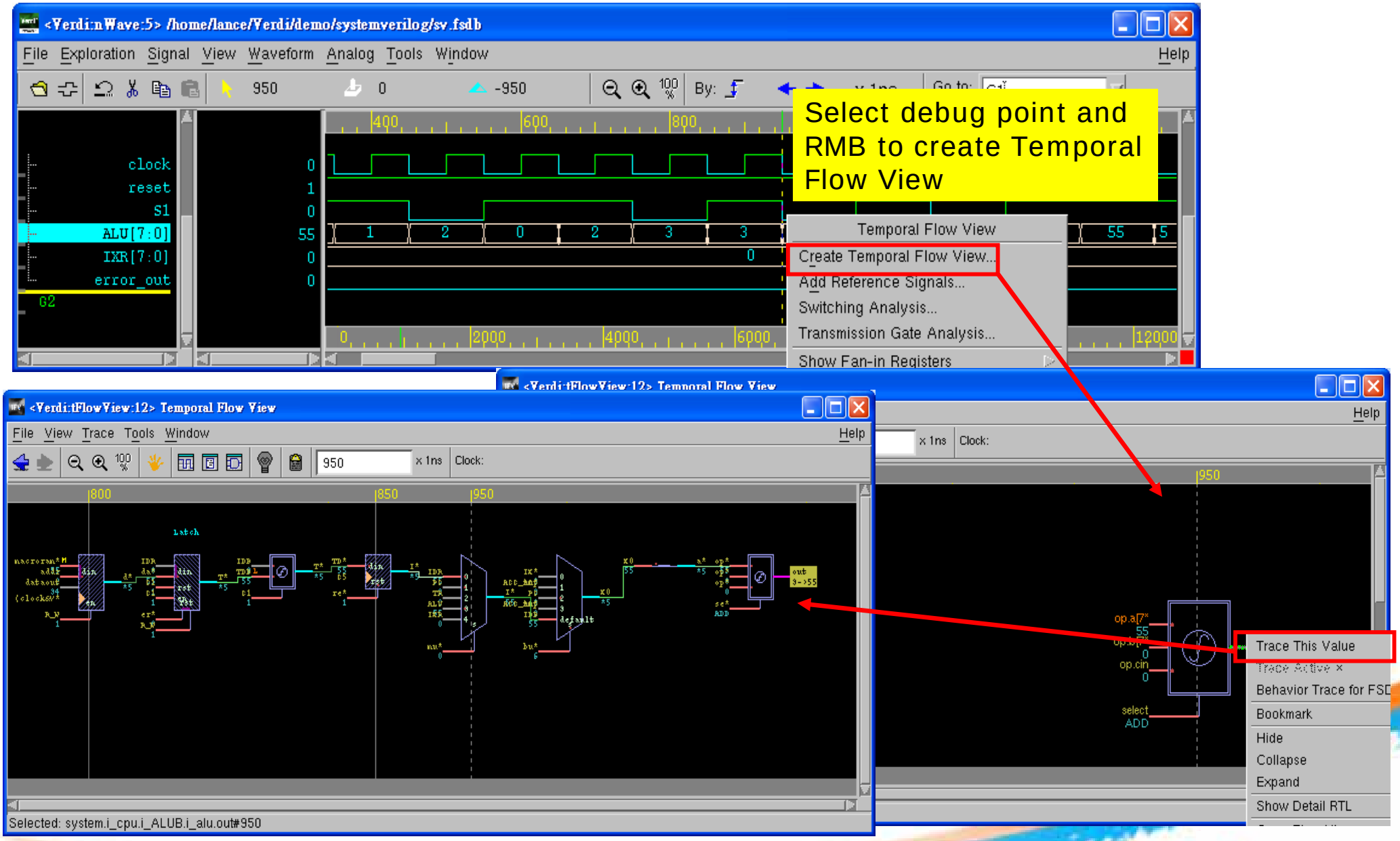
View Simulation Results –
Parameter Annotation for 'generate'

- To 'compute' and annotate variables inside a generate block (genvars)...



SpringSoft
Accelerating Engineers

Debug SV by Temporal Flow View



SpringSoft Accelerating Engineers

Lab8 (1/2)

- Debug with SystemVerilog Design
  - Import system verilog design (2 way)
    - Verdi -f run.f -sv -ssf sv.fsdb &
    - Verdi -f run.f +systemverilogext+.sv+.SV+ -ssf sv.fsdb &
  - In the nTrace hierarchy browser, expand "**i_cpu**" and "**i_ALUB**" to show the SV code.
  - Double-click on "**carry_mode**" to trace driver.
  - Click the New Schematic icon to show the "**i_CCU**" SV code in the schematic view.
  - Drag and Drop the "**i_alu**" instance from nTrace hierarchy browser to nWave to show the related waveforms.
  - In nWave, double-click the transition to value 55 at time 950 on signal out to locate the active driver.
    - (Note: some of the signals are complex structures.)
  - In nTrace, enable **Source → Active Annotation**.

Lab8 (2/2)

- In nTrace, scroll the "**i_alu**" source code to locate the "**Nbit_adder**" instantiation and double-click the instance name to locate the code.
  - The module includes a generate statement and there are multiple 'instances' shown in the hierarchy browser.
- In the hierarchy browser, double-click on "**addbit[2]**" to show the code.
- In nTrace, enable **Source → Parameter Annotation** to display the parameter definition.
- In the hierarchy browser, double-click on "**addbit[6]**" to show the code.
  - The code is the same as before; however, the local variable values are different.

Assertion-Based Verification by SVA

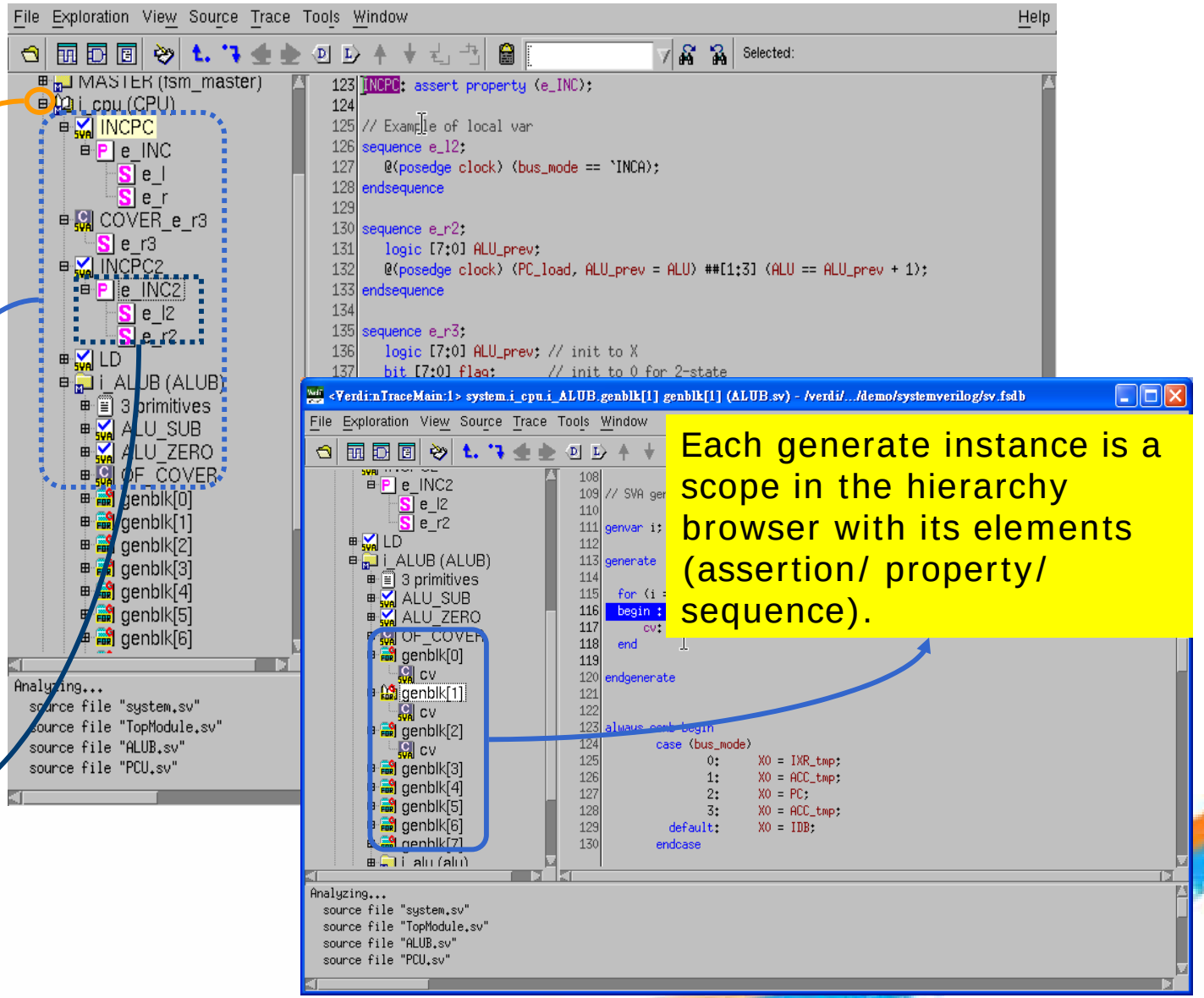
- Common Problem
  - Hard to display assertion results as waveforms along with design signals
  - Need to manually analyze the assertion construct and calculate values
  - Checking new or modified assertions needs re-running simulation
- SpringSoft Solution
  - Use Property Tools, which include Assertion Analyzer and Assertion Evaluator, to do efficient debug

View SVA –
Hierarchy Browser and 'generate' in nTrace

Hierarchy can be expanded/ collapsed by clicking on +/- sign or double-clicking. Double-clicking also changes scope.

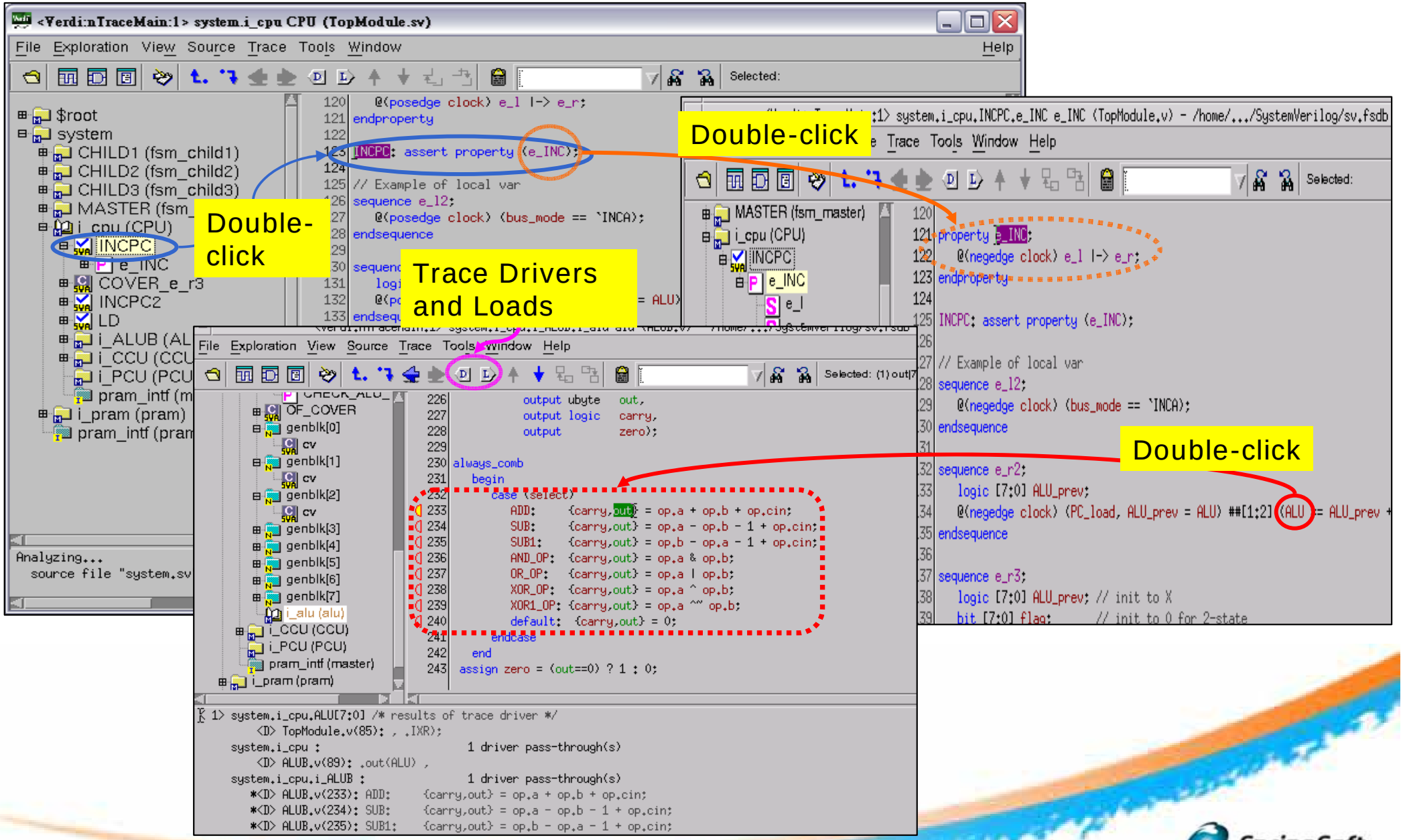
Special icons for SVA asserts, properties, sequences, covers.

Assertion structure is represented hierarchically.



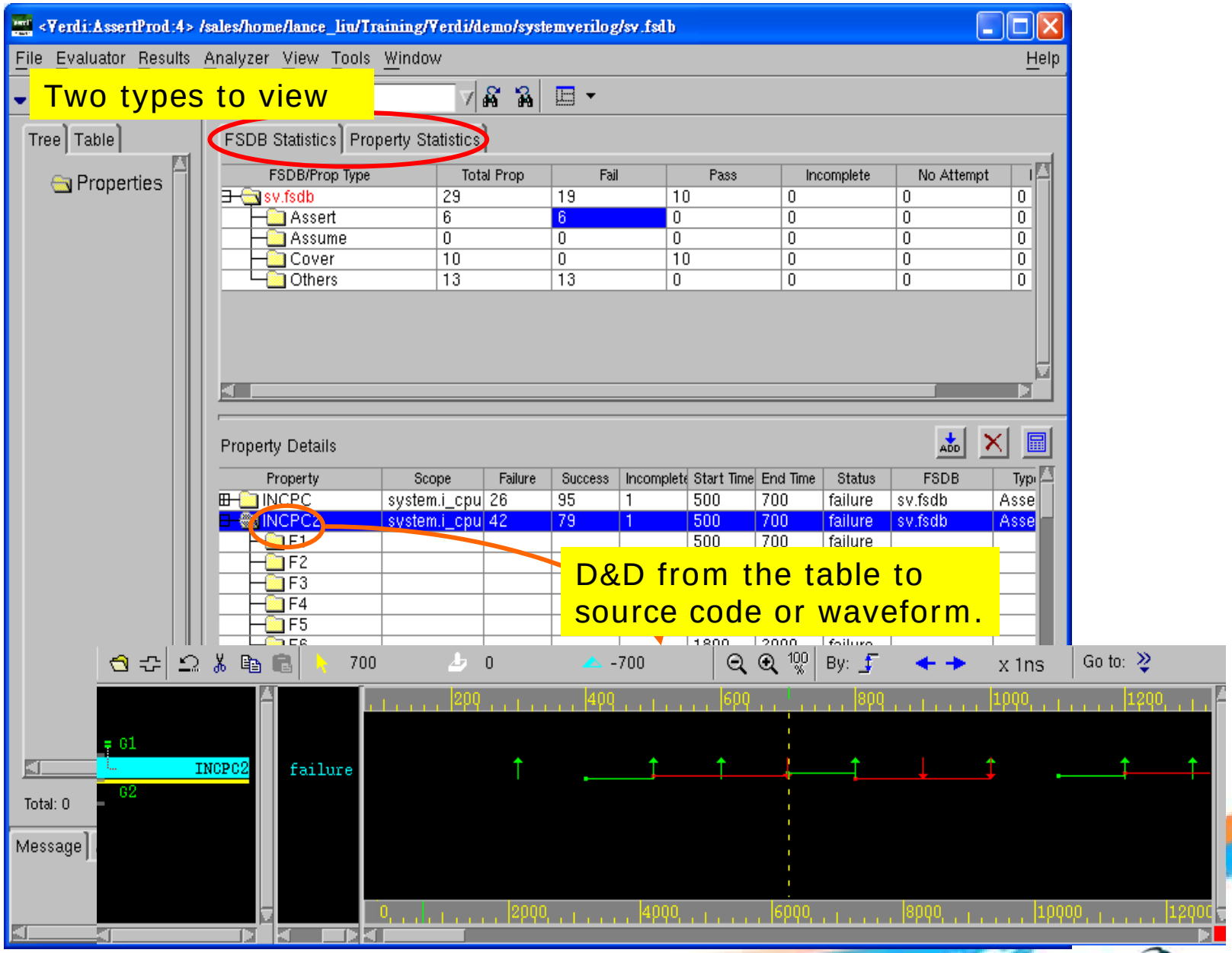
Each generate instance is a scope in the hierarchy browser with its elements (assertion/ property/ sequence).

View SVA –
Source Code Tracing



SpringSoft
Accelerating Engineers

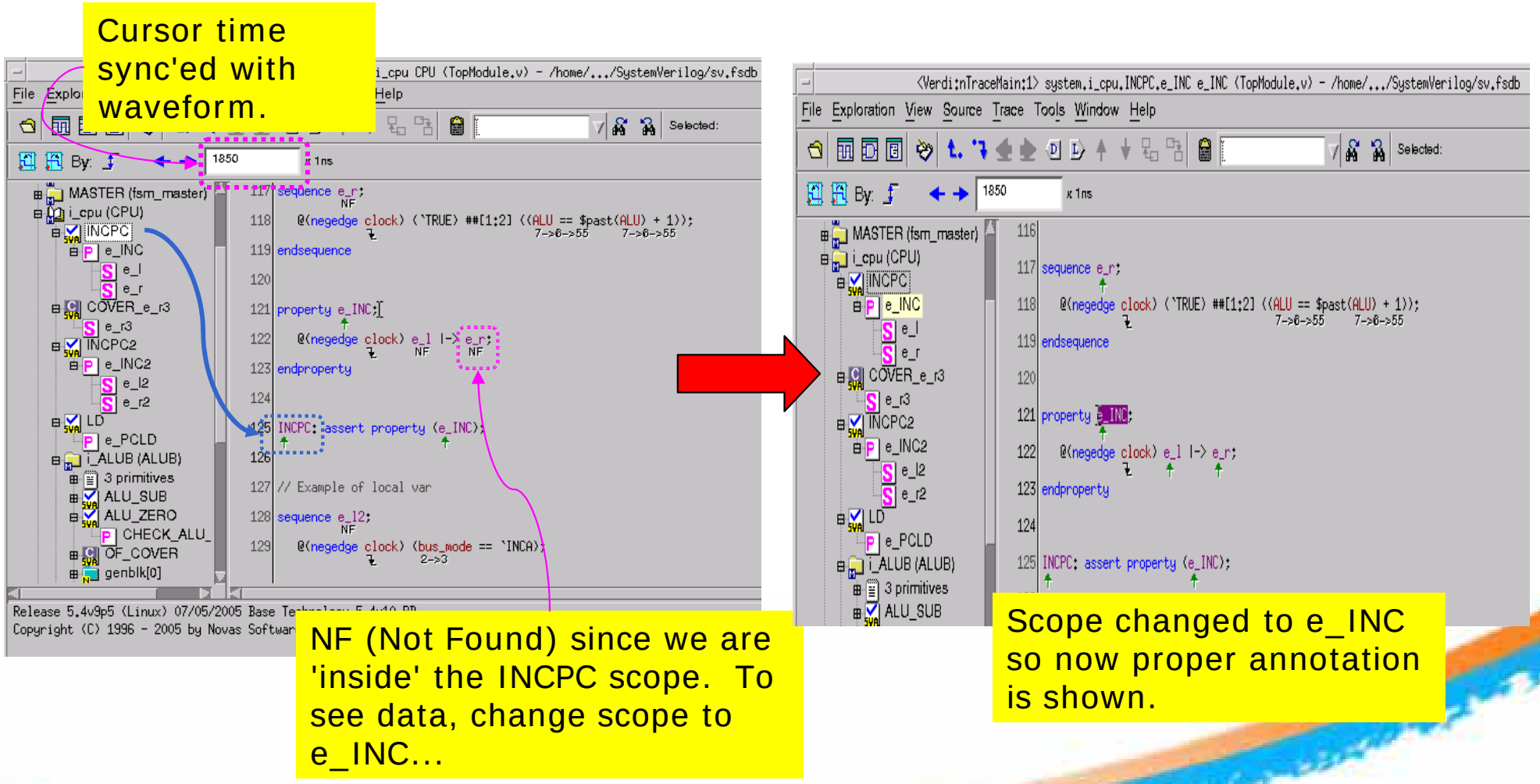
View SVA –
Statistics and Details



SpringSoft
Accelerating Engineers

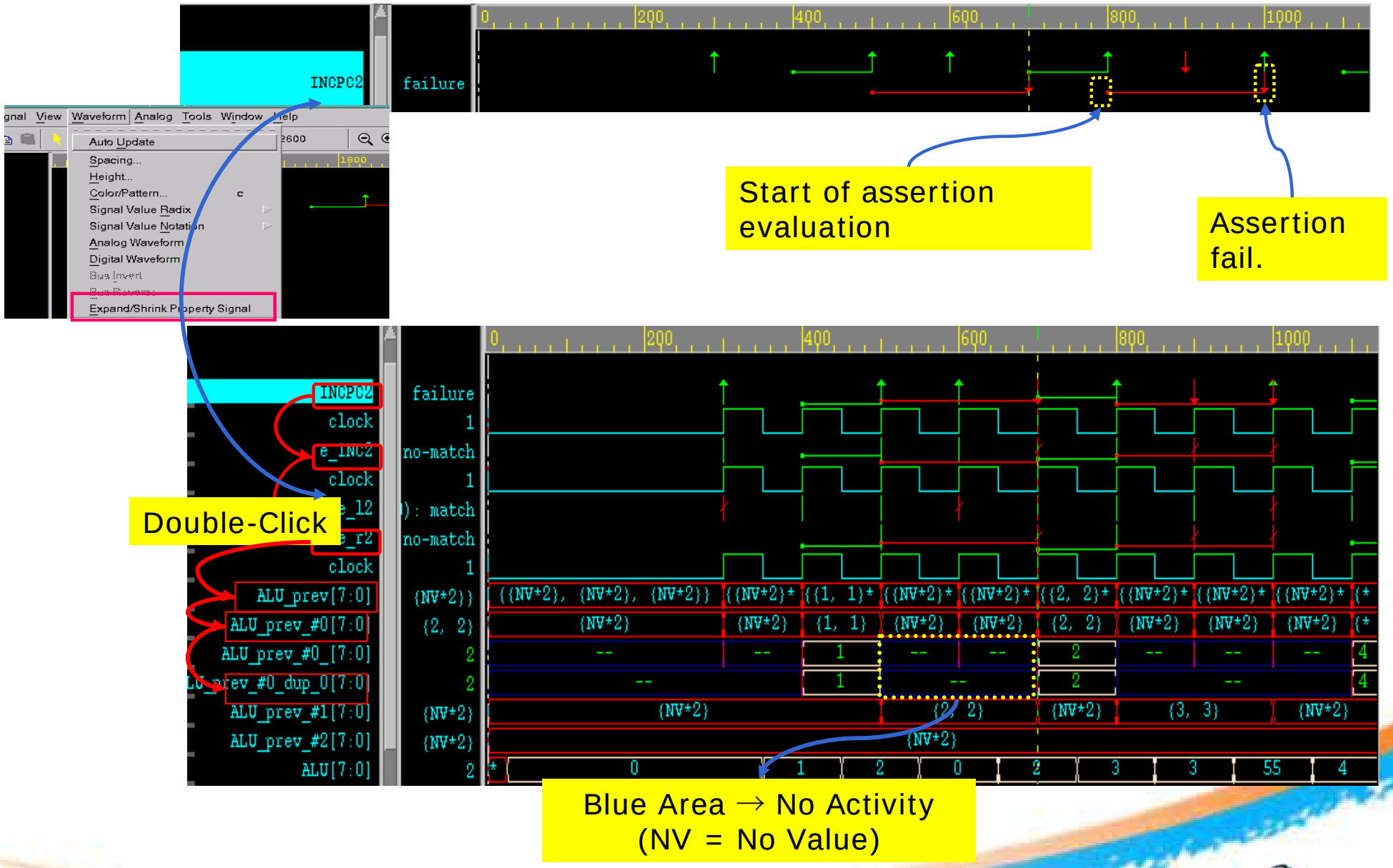
Source Code Annotation

- Visualize data next to the variable in the source code by enabling **Source → Active Annotation**

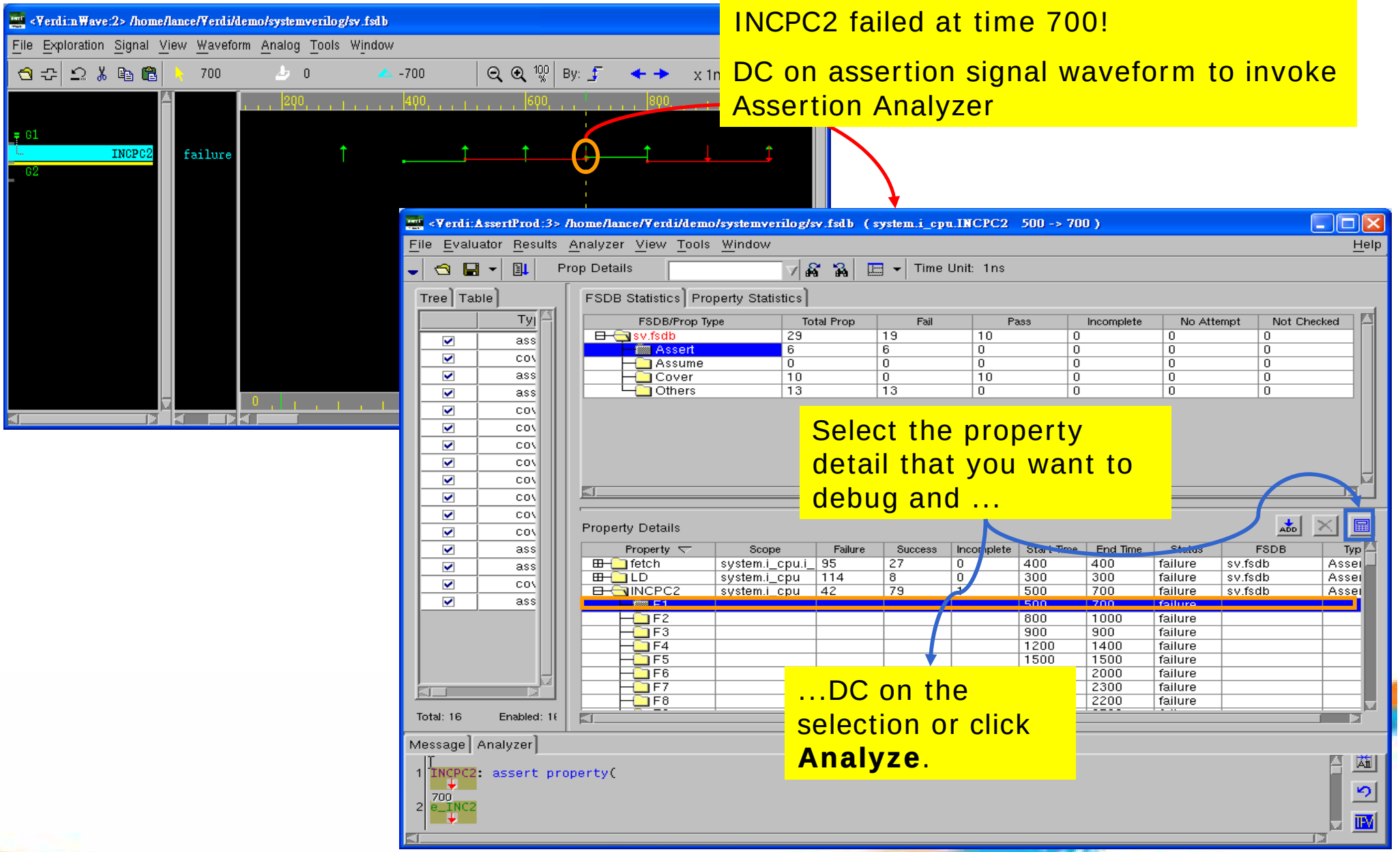


SpringSoft Accelerating Engineers

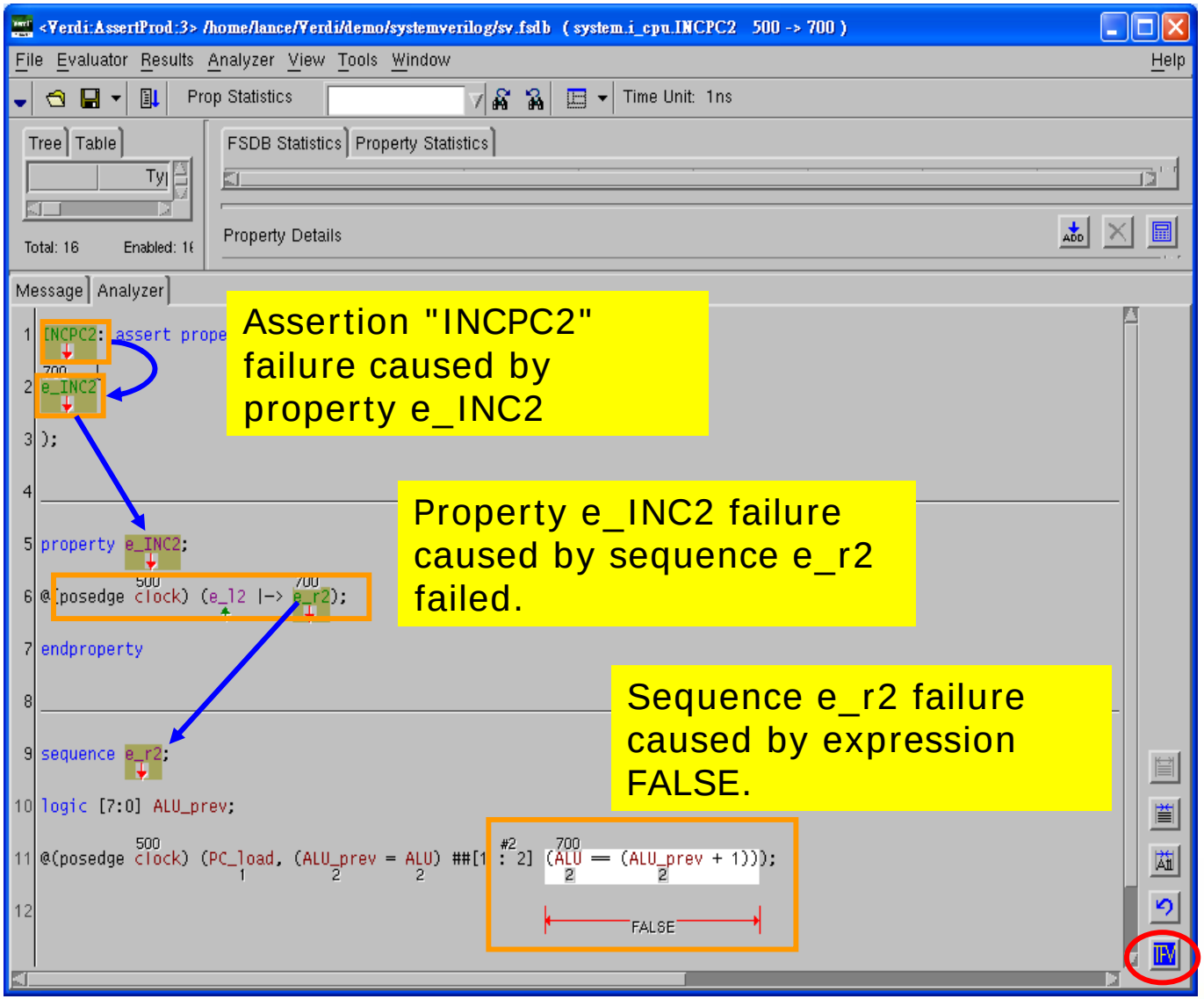
View Assertion Simulation Results



Assertion Analyzer (1/2)



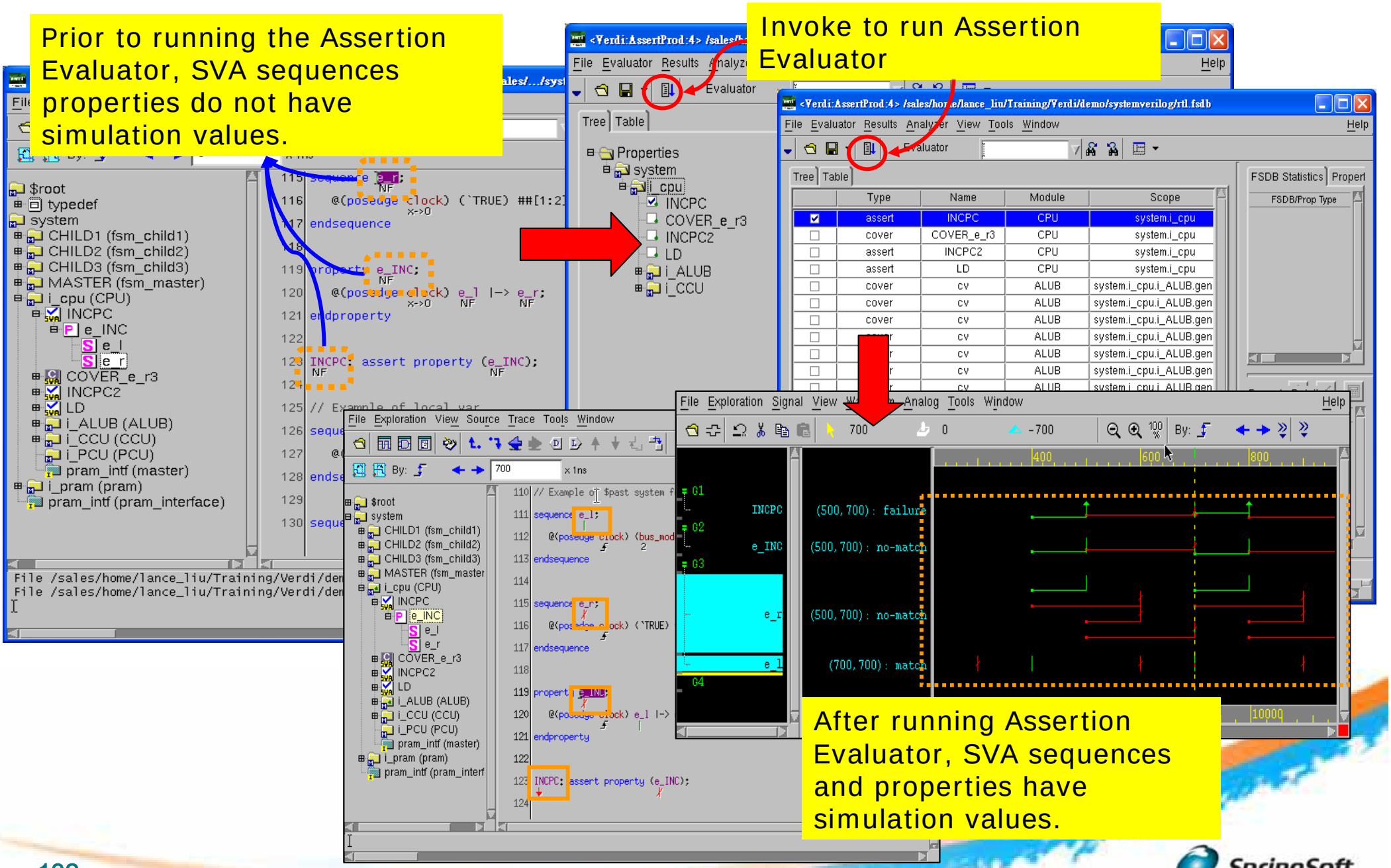
Assertion Analyzer (2/2)



Invoke Temporal Flow View to debug

SpringSoft Accelerating Engineers

Assertion Evaluator



SpringSoft
Accelerating Engineers

Lab9 (1/4)

- Trace assertion codes and view simulation result on waveform and property tool and then debug with assertion analyzer
  - Invoke Verdi
    - verdi -f run.f -ssf sv.fsdb -sv &
  - In the nTrace hierarchy browser, expand "i_cpu" and "i_ALUB" to show the SVA assertion codes
  - Double-click on "ALU_SUB" to show the assert code and related property
  - Double-click on "CHECK_SUB" to show the property code and related sequence.
    - See the symbols in the hierarchy browser
  - Double-click on "e_SUB" to show the sequence code
  - In nTrace, invoke Tools → Property Tools
    - You can see all the properties in the design on FSDB Statistics tab or Property Statistics tab. This will display the results as a summary of success/fail per assertion in the table

Lab9 (2/4)

- On "**FSDB Statistics**" tab, double-click "**Assert**" type and it will show to "**Property Detail**" pane.
- D&D the "**INCPC2**" assertion from the Property Tools window to nWave
  - The Success/Failure will shown in nWave
- In nTrace, enable **Source** → **Active Annotation**.
- Double-click on "**INCPC**" to show the source code and use the search forward/backward icons  to step through changes on the assertion
- In nWave, double-click on "**INCPC2**" to expand the property and signals.
- Double-click on "**e_INC2**" to expand the sequence and signals.
- Double-click on "**e_r2**" to expand the signals and local variable.
- Double-click on "**ALU_prev**" to expand and show the different attempts
- Double-click failure arrow at 700ns of "**INCPC2**" in nWave to open Assertion Analyzer window
  - On the Assertion Analyzer window, you can check the assertion code failure causes and invoke Temporal Flow View to debug
- In nTrace, **File** → **Exit**

Lab9 (3/4)

- Check new or modified assertions without re-running simulation
  - Invoke Verdi
    - verdi -f run.f -ssf rtl.fsdb -sv &
  - In the nTrace hierarchy browser, double-click on "i_cpu" to expand and then expand "INCPC", "e_INC" and "e_r".
  - In nTrace, enable **Source → Active Annotation**.
    - You can see 

and even use the search forward/backward  icons , the value is still NF, Not Found.
  - In nTrace, invoke **Tools → Property Tools**
  - In Property Tools window, invoke **Evaluator → Disable All Props** to disable all property in Tree or Table tab.

Lab9 (4/4)

- **Right-Mouse-Button** → **Get Progs** to open the Get Properity Form on Tree or Table tab and select **"INCPC"** on scope **system.i_cpu**.
- Invoke **Evaluator** → **Evaluate** to setup the evaluation
- **Enable** the "INCPC" on Tree or Table tab and Change the Store Tab in right window from "Asserts Only" to "All" and then **Click** Run
- Click "**Yes**" when question window is shown
- In nTrace, use the search forward/backward icons  then you can see the values of assertion codes are annotated
- **D&D** the "INCPC" assertion from the nTrace window to nWave and then see the Success/Failure on nWave

Analyzer SystemVerilog Testbench (SVTB)

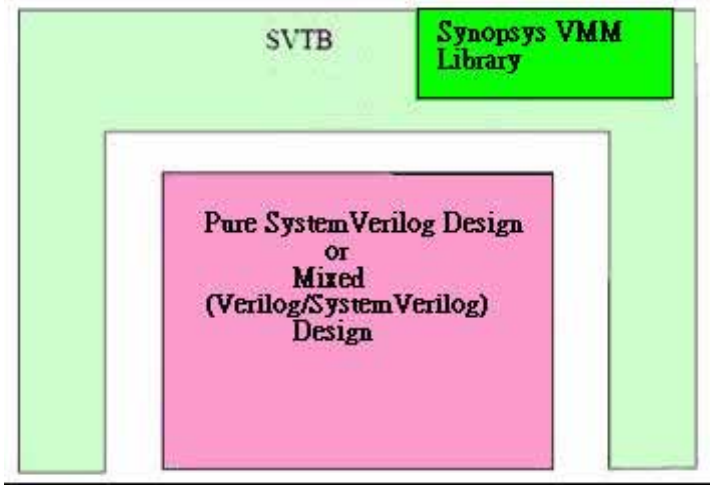
- Common Problem
  - Hard to understand the inheritance relationship between classes
  - Hard to crossing probing the error message from text log to the design
- SpringSoft Solution
  - Use Testbench Browser to understand the class and constraint information
  - Use fsdbLog integrated with nWave to get flexible visualization

Import SVTB into Verdi

Import SVTB with VMM Library

- Import SystemVerilog designs linked with the Synopsys VMM library.
  - The VMM library source code path should be specified in the **+incdir+<directoryname>** argument.
    - The argument is used for specifying the search path for files used by the 'include statement.

```
% verdi –sv –f run.f +incdir+your_vcs_vmm_library_path
```



- Add option **–ntb_opts vmm** in command line if you don't have the VMM library source.
  - The option loads a dummy VMM library for preventing compile errors.
    - The dummy libraries are under <Verdi_install>/etc/kdb/verilog/vmm directory.

```
% verdi –sv –f run.f -ntb_opts vmm
```

Import SVTB into Verdi

Import SVTB with OVM Library

- Import SystemVerilog designs linked with the Cadence and Mentor's OVM library.
  - The OVM library source code path should be specified in the **+incdir+<directoryname>** argument.
  - Use one of following two methods to access OVM library.
    - Use package technique:
      - Import **ovm_pkg** into Verdi (add file **ovm_pkg.sv** in run.f).
      - Add **import ovm_pkg::*;** in source code.
    - Use include technique:
      - Add **`include "ovm.svh"** in source code.
  - If the OVM library is from Cadence IUS, two extra options **+define+INCA** and **-ignorekwd_config** need to be added.

```
% verdi -sv test.sv +incdir+.
+incdir+${IUS81_HOME}/tools/ovm/src +define+INCA
-ignorekwd_config
```

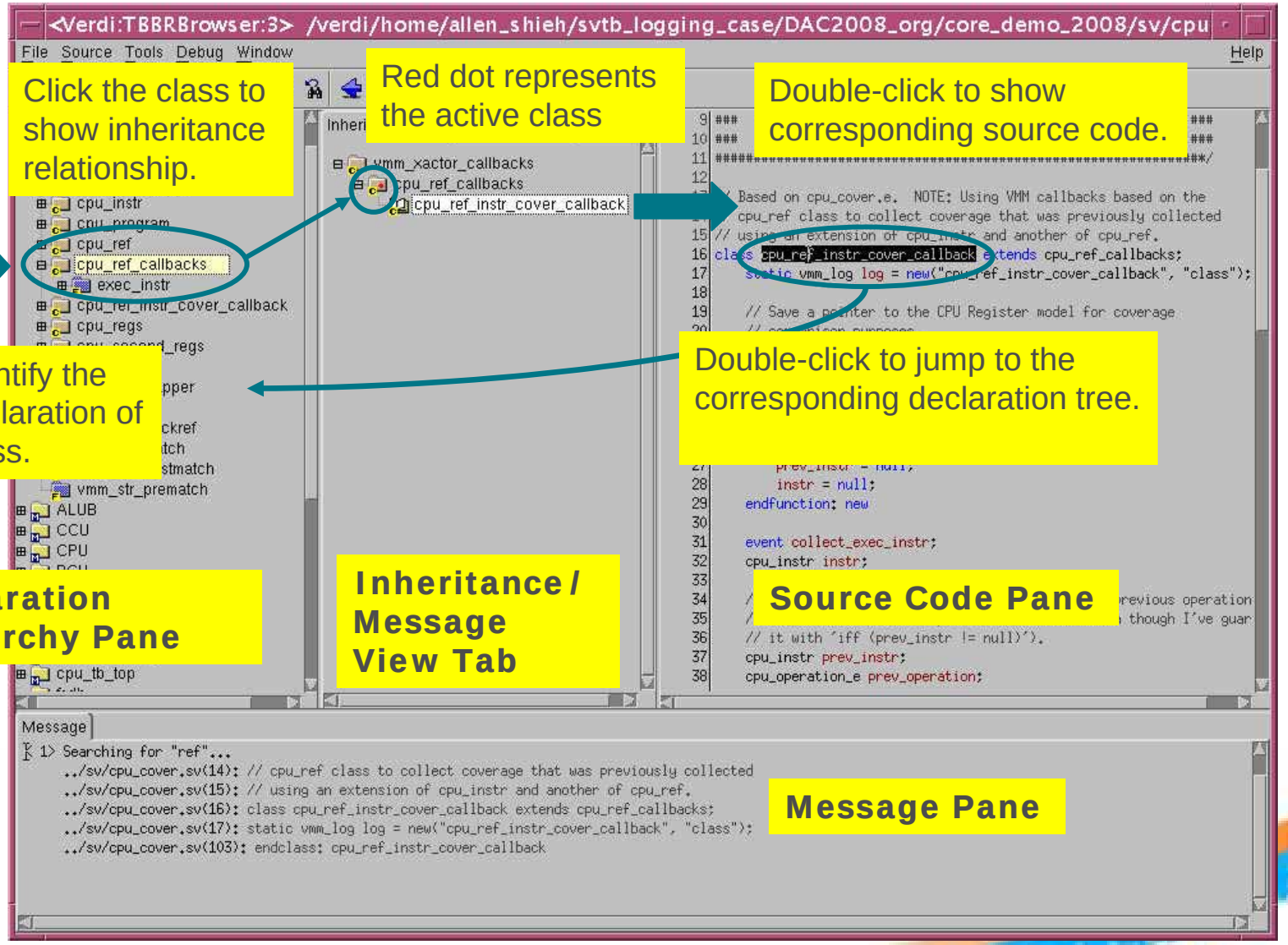
Browse SVTB Source Code

Testbench Browser

- Use the following environment variable to enable the feature:
  - setenv VERDI_SVTB_BETA 1
- In *nTrace*, invoke **Tools** → **Testbench Browser** to open the *Testbench Browser* window.
  - Use **–tbbr** command line option when invoking Verdi to bring up the *Testbench Browser* automatically.
  - Five areas to browse SVTB:
    - **Declaration Hierarchy Pane**
      - Browse the declaration of classes in your design.
    - **Inheritance View Tab**
      - Display the inheritance tree for the selected class and function.
    - **Message View Tab**
      - Display logging message information.
    - **Source Code Pane**
      - Display the SVTB source code for the active class.
    - **Message Pane**
      - Log results of **Find String** & **Find Scope.**
  - Use D&D to correlate with other Verdi views.

Browse SVTB Source Code

Testbench Browser – A Debugging Example



fsdbLog Syntax

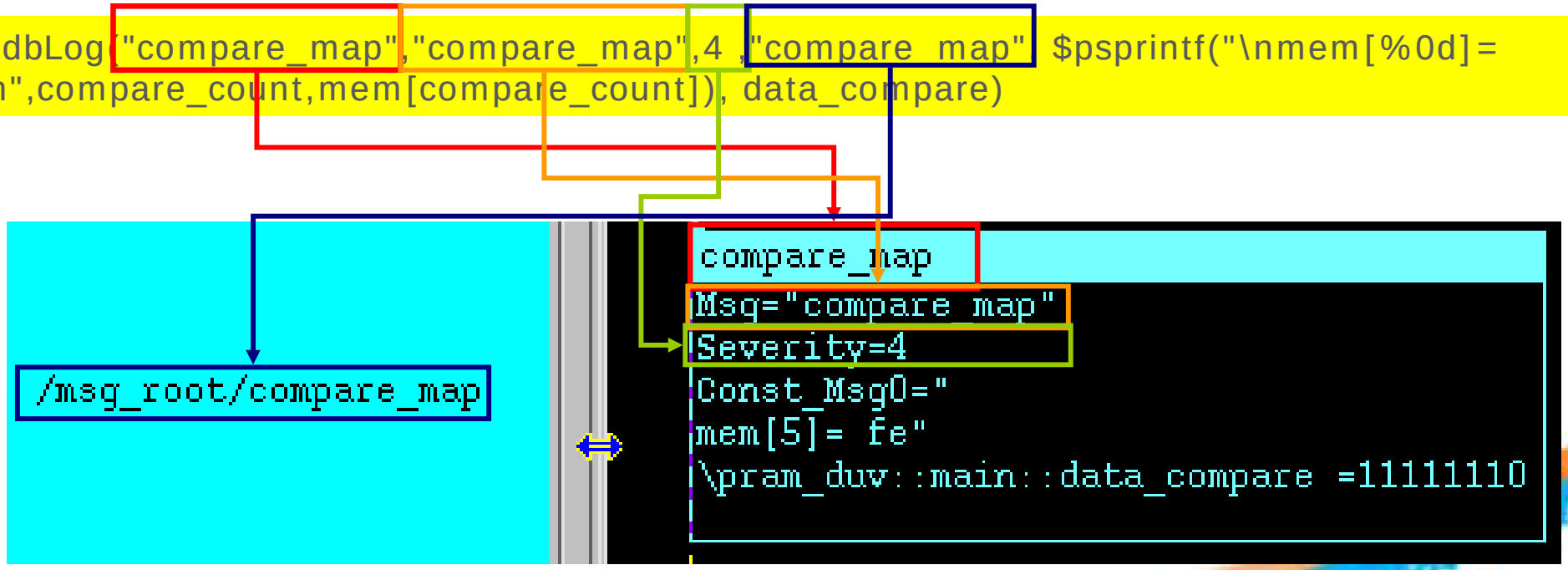
```
$fsdbLogInit();
```

Initial fsdbLog dumping

```
$fsdbLog("label", "message", severity, "stream_name" [, variables |"string"]* );
```

Print log into fsdb

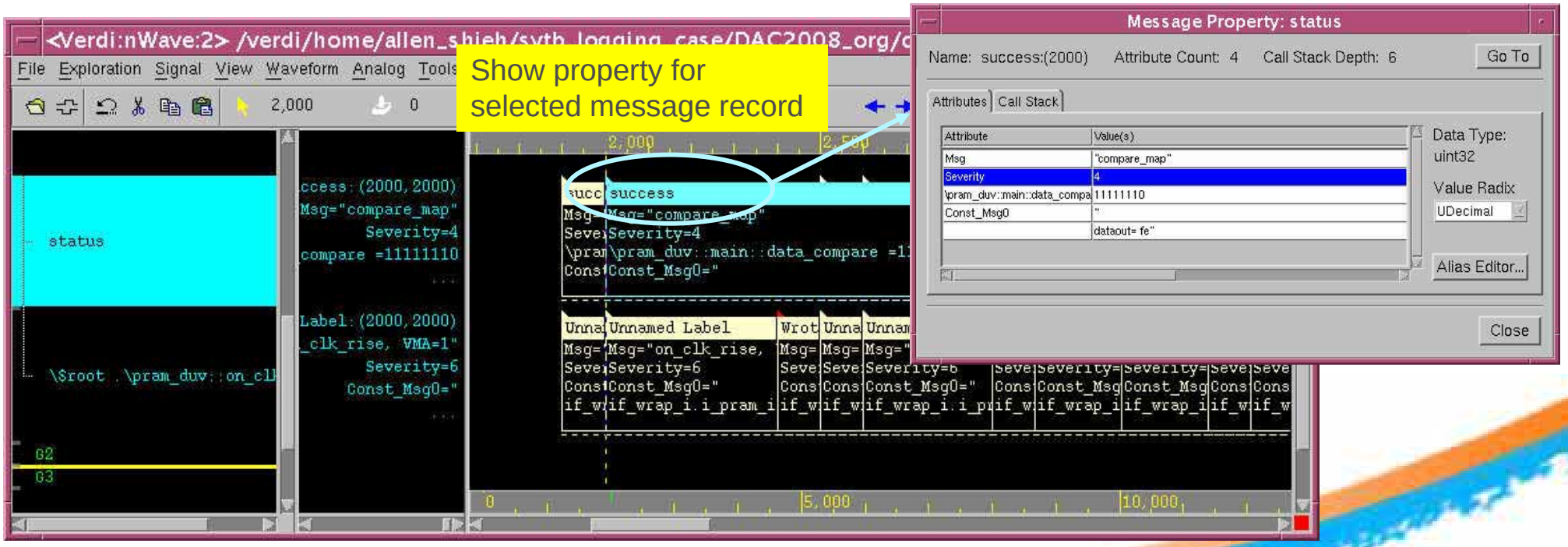
```
$fsdbLog("compare_map","compare_map",4 ,"compare_map" $psprintf("\nmem[%0d]=
%h",compare_count,mem[compare_count]), data_compare)
```



Debug Logged Messages

Display Messages in nWave

- Invoke **Signal** → **Get Signals** command in *nWave* to add messages to the waveform.
  - Logged messages will be saved under **msg_root** scope.
- Select a message stream in waveform pane, use RMB → **Properties** to bring up the *Message Property* form.
  - Displays the detailed properties for the selected message record.

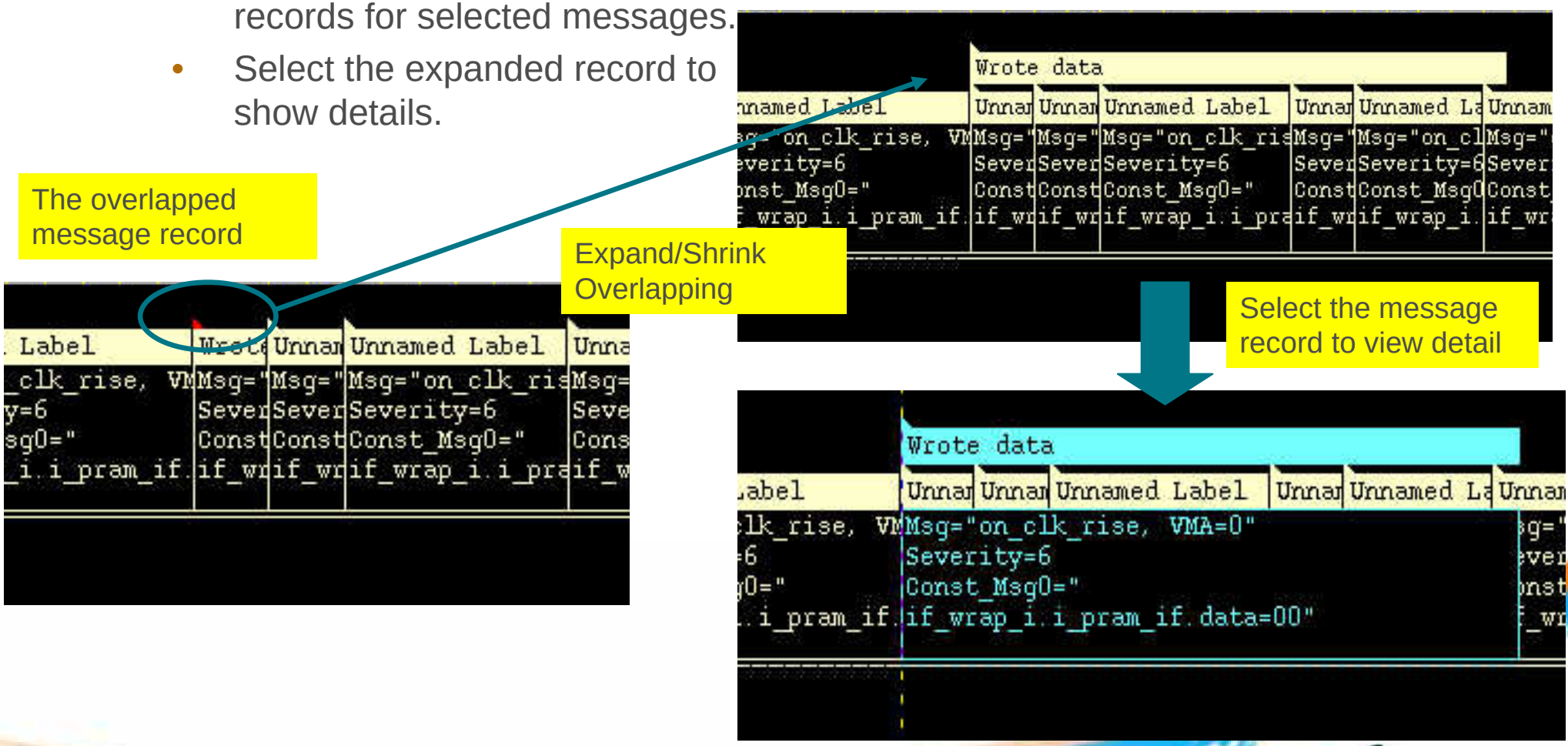


SpringSoft
Accelerating Engineers

Debug Logged Messages

Expand Overlapped Message Records

- The red triangle flag indicates an overlapped message record at the same begin time.
  - Use **Waveform** → **Message** → **Expand/Shrink Overlapping** to expand records for selected messages.
  - Select the expanded record to show details.

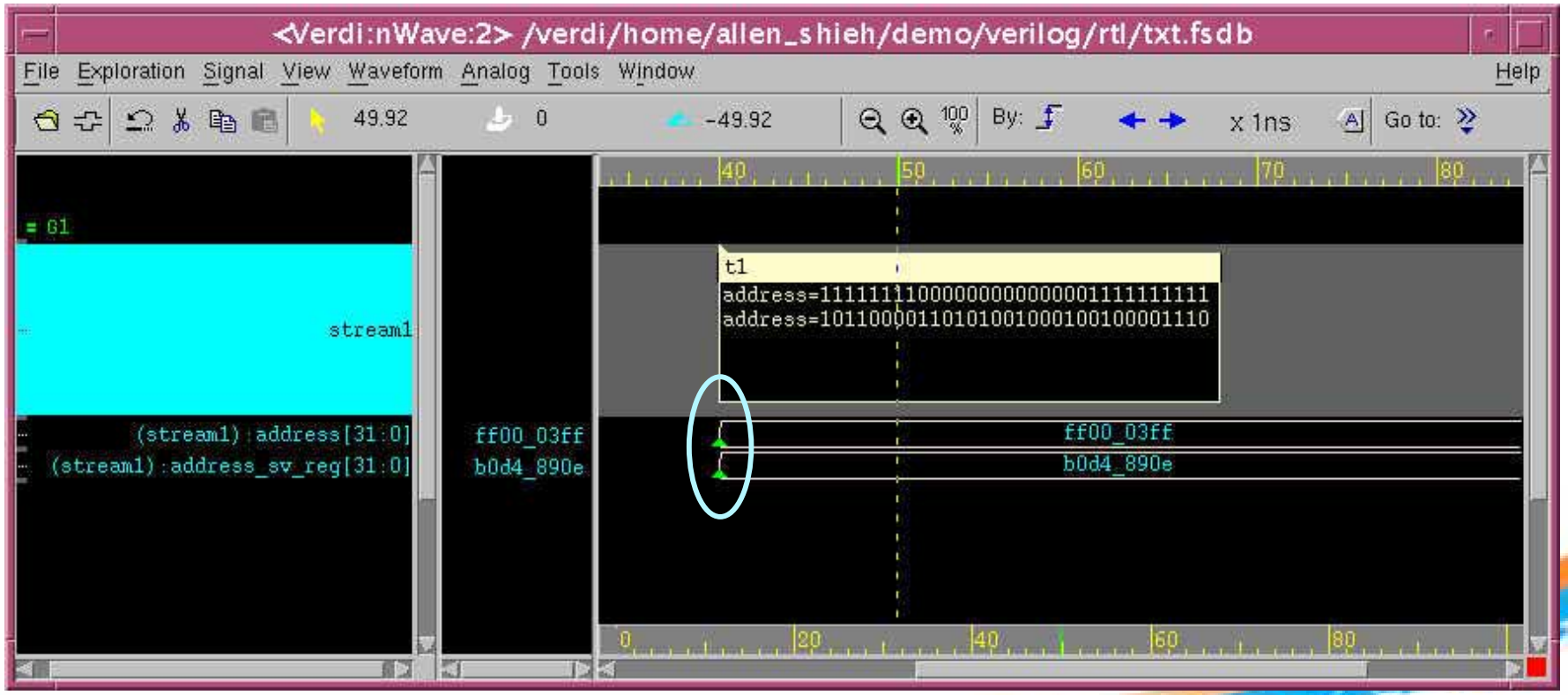


SpringSoft
Accelerating Engineers

Debug Logged Messages

Expand Attributes for Messages

- Select a message stream, invoke **Waveform** → **Message** → **Expand Attributes** command.
  - Expanded attribute nodes are added under the message.
    NOTE: the value changes from expand attribute are not actual value changes.
  - The green marks show the occurring time of the message records.

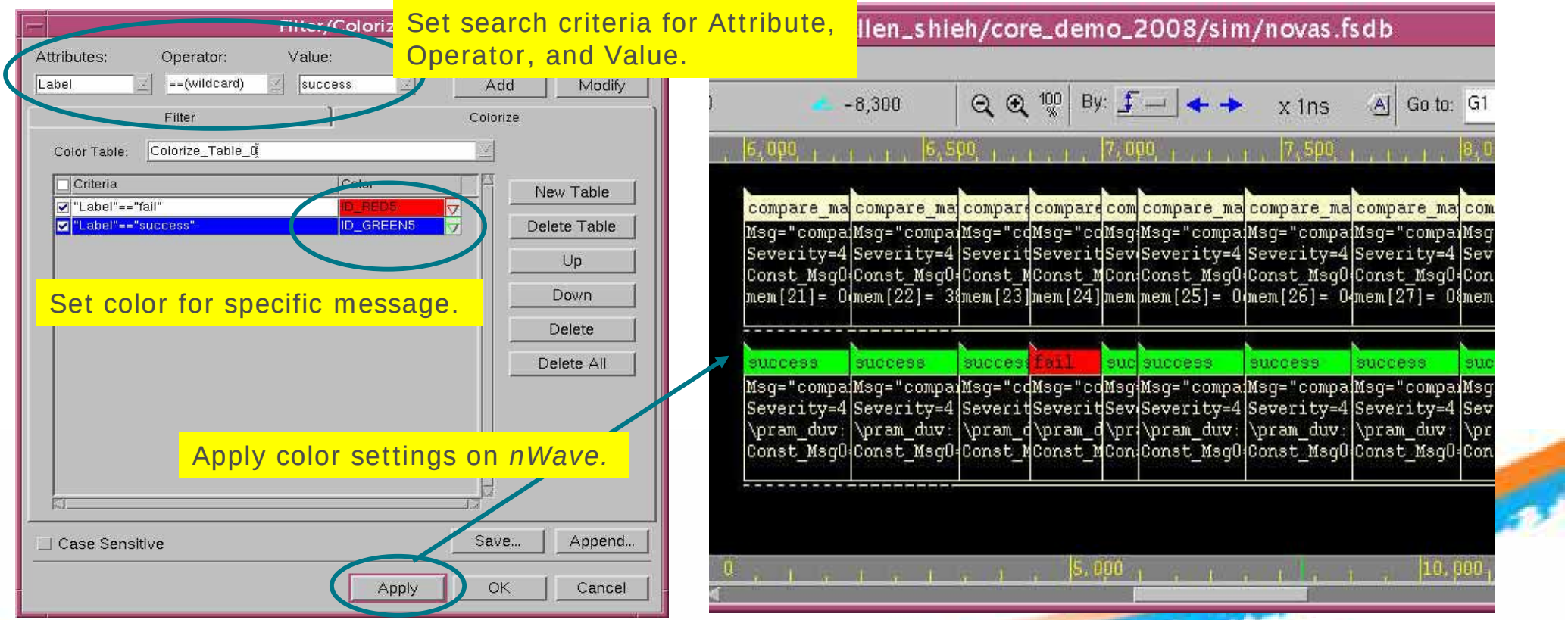


SpringSoft
Accelerating Engineers

Debug Logged Messages

Colorize Messages in nWave

- Invoke **Waveform** → **Message** → **Filter/Colorize** command to open *Filter/Colorize* form.
  - Click **Color** tab to color messages based on a specified condition.
  - Use **Waveform** → **Message** → **Clear Colorization Result** to clear specified colors.

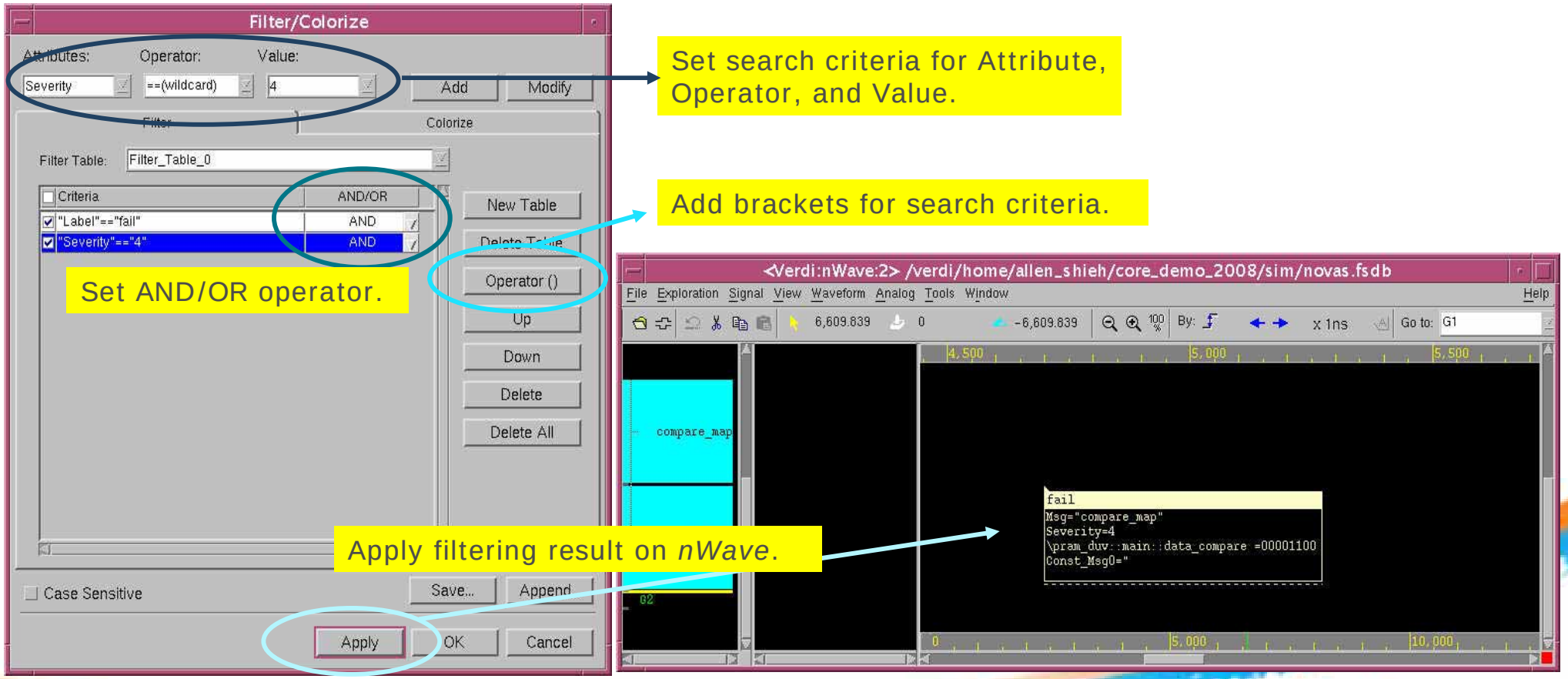


SpringSoft
Accelerating Engineers

Debug Logged Messages

Filter Messages in nWave

- Invoke **Waveform** → **Message** → **Filter/Colorize** command to open *Filter/Colorize* form.
  - Click **Filter** tab to filter messages based on specified conditions.
  - Use **Waveform** → **Message** → **Clear Filtering Result** to clear filters.

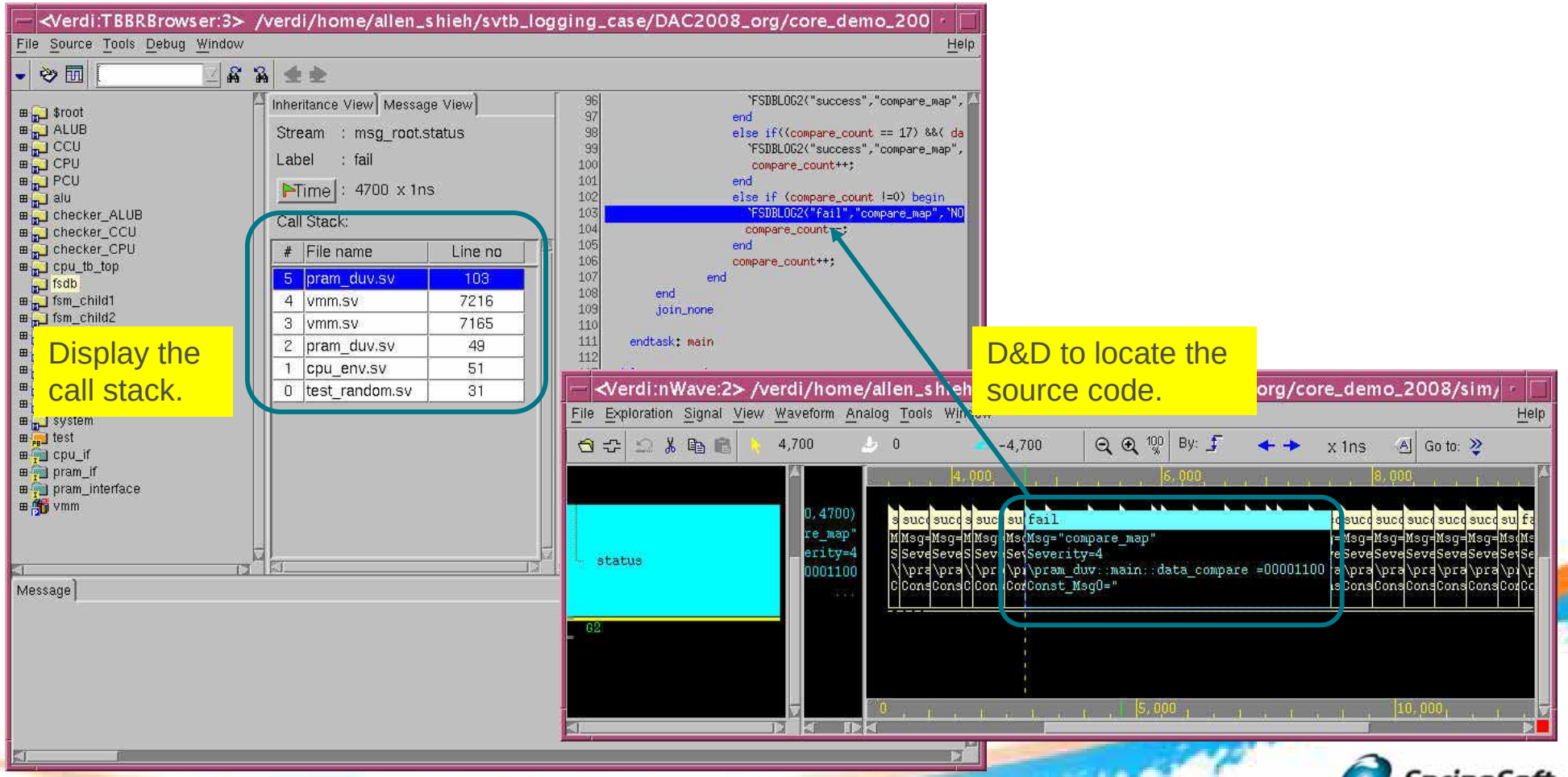


SpringSoft
Accelerating Engineers

Debug Logged Messages

Locate Messages in Testbench Browser Window

- Drag & drop the record from *nWave* to the source code pane of *Testbench Browser* to locate the source code.
  - Call stacks will be shown in *Message View* pane.



SpringSoft Accelerating Engineers

Lab10 (1/5)

- Understand testbench created by SystemVerilog Testbnech constructs with VMM library.
  - Invoke Verdi
    - cd sim;
    - verdi -macrodebug -nologo -sv display.svh -f run_sv.f -ssf novas.fsdb +define+SVA_ENABLE +incdir+../sv -ntb_opts vmm ../sv/test_random.sv &
  - Show a SystemVerilog testbench module – note the different icons for “**module**” vs. “**program**” blocks in the nTrace hierarchy browser.
  - Double-click on "**test**" to show the class with C folders in hierarchy browser.
  - Bring up Testbench Brower (TBBR) from nTrace, **Tools** → **Testbench Browser**.
  - Because testbench is software oriented, in this window will provide declaration based hierarchy tree to help user understand the structure of their testbench.

Lab10 (2/5)

- Expand **$root** node
- There're several classes declared under global area of the testbench.
- Double-click on class "**cpu_program**", you can see there're functions and tasks.
- The inherited function/task will be marked with pink arrow to indicate it's declared from other class
- On Inheritance View, you can realize the inheritance relationship from the selected class. The **cpu_program** is extended from **vmm_xtractor** class, so this program will inherit the function from it.
- Click on the function "**new**", you can see the inheritance relation of functions
- Click on "**cpu_address_restrictions**", this is a constraint in SVTB program. Switch to Constraint tab, it will show you the variables been constrained by the selected constraint.
- Select **start_write_addr** and switch back to **Constraint** tab, you can see this variable is actually constrained by 2 constraints.

Lab10 (3/5)

- SVTB simulation result visualization for logging
  - Verdi provides a mechanism that allows users to dump out logging information directly into Verdi's FSDB database
  - Bring up nWave, **File** → **Restore Signal** → **svtb.rc** (The logged message can be found on **msg_root** folder in get signal form)
  - You can see the log messages are saved into Verdi's waveform format. It becomes easier to see the time information for the messages.
  - Select one of the messages from the **compare_map** stream and do **RMB** → **Properties**….
  - The detailed information including all the fields with their value and call stack can be seen in this window. You can also change radix for the value here. Select the **\pram_duv::main::data_compare** attribute and change Radix to **Hexadecimal**.

Lab10 (4/5)

- Some fields in message boxes may be important for us and we need to monitor their value change along the simulation time.
- Select the **compare_map** stream and do **Waveform → Message → Create Attribute Signals**. In the resulting pop-up form, enable (toggle) the box next to \pram_duv::main::data_compare and press OK.
- A new signal row will be created to represent the waveform for **\pram_duv::main::data_compare**.
- The green triangle indicates whenever a change is detected compared to the previously logged value rather than a typical value-change type transition vertical line.
- Let's see how logging message can help you to debug testbench. Expand "**CPU_data**" group signals.
- Select "**status**" and do **Waveform → Message → Filter/Colorize ...**
- Column change to "Label" and Criteria "=" "**fail**", then press "**Apply**".

Lab10 (5/5)

- The information in the table now only contains “**fail**” messages and the messages on the waveform also include only the “**fail**” parts.
- Click the **fail** in 6800 ns. The signal “**dataout**” is actually output from testbench. We found there’s an error transition **97 → c**.
- The message could contain more information than is practical to view in a waveform. Verdi’s Message Analysis Window can help you to manage messages efficiently in a textual spreadsheet-like window. In nWave, do **Tools → Message → Message Analysis Window**. Click on **Get Stream button** and select “**status**”, press **OK**.

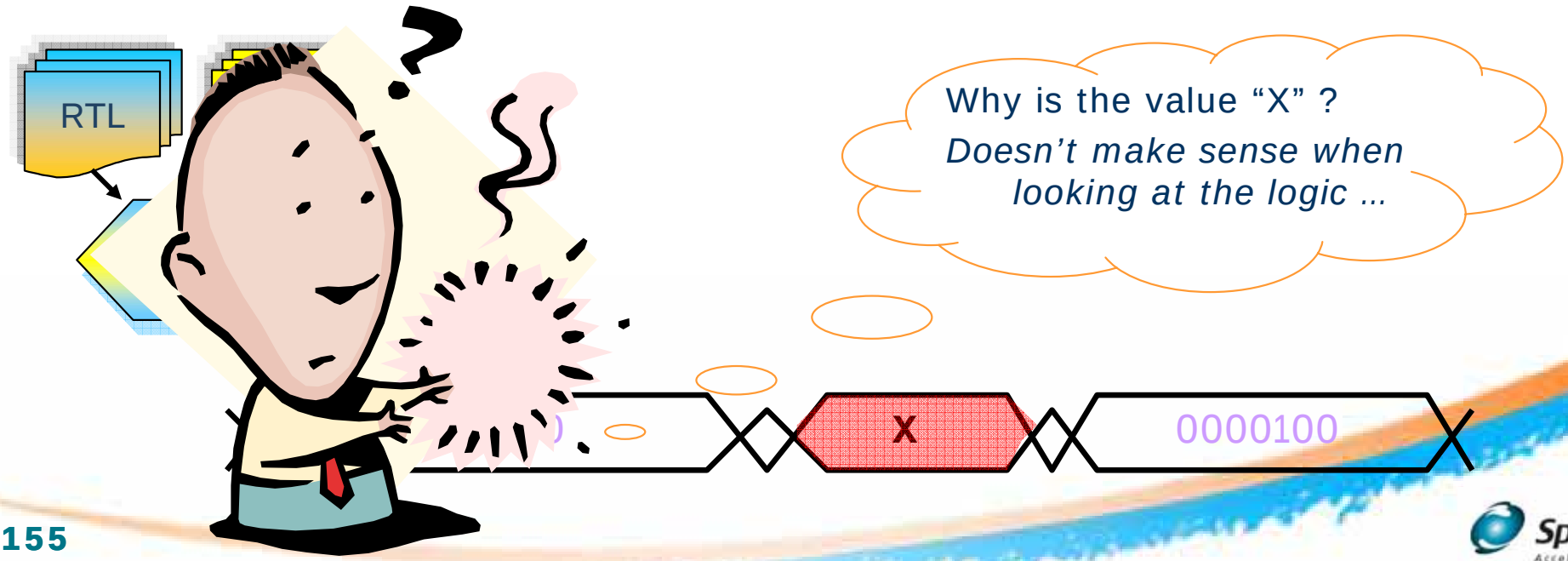
Debug Design with Power Intent CPF/UPF

- Common Problem
  - Hard to understand how the power intent changes RTL designs
  - Hard to identify simulation failures come from power off or functional bugs?
- SpringSoft Solution
  - Use Power Manager to understand power specification
  - Use verdi trace result with power intent annotation to debug simulation failures
  - Power Impacted Signals Auto-Extraction
  - Power-Aware X Detection

Power-Aware Debug Introduction

Power Intent Impacts Debug

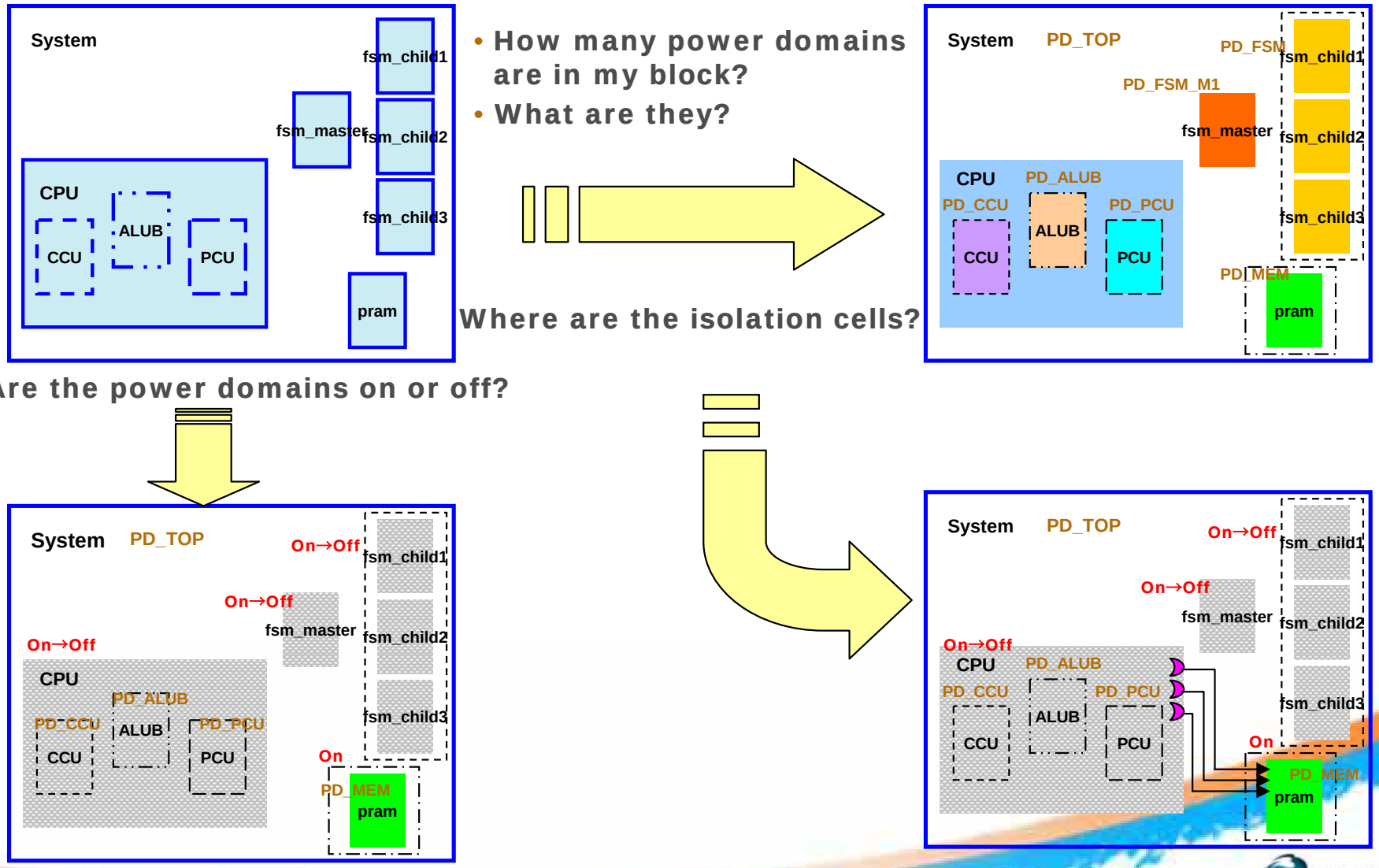
- UPF & CPF are used to specify power intent
  - How does the power intent impact my RTL?
  - How many power domains are in the design?
  - Which power domain does each instance belong to?
  - Where are the retention and isolation rules?
  - What are the power states?
- Debug of power-aware simulation failures
  - What is the cause of this X? Is it Power off? Functional bugs?
  - Debug tools must understand the power specification



Power-Aware Debug Introduction

Power-Aware Debug Challenges

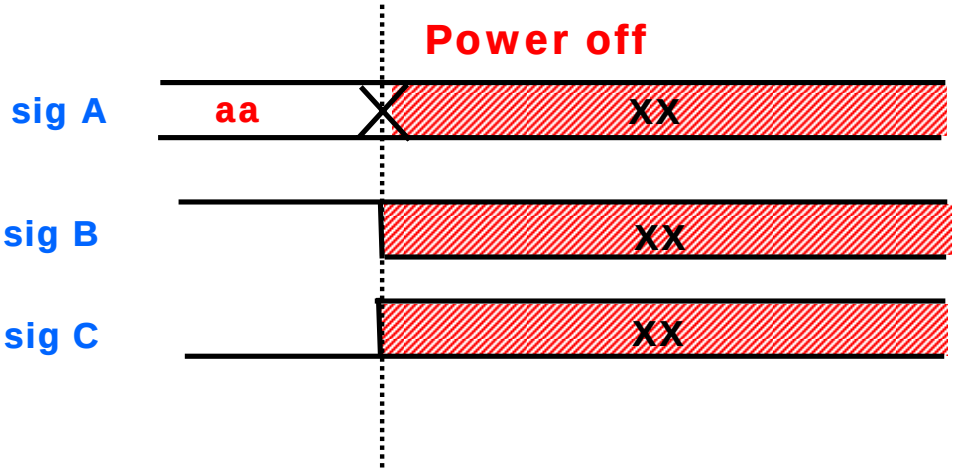
- Comprehension – Understand how the power intent changes RTL designs



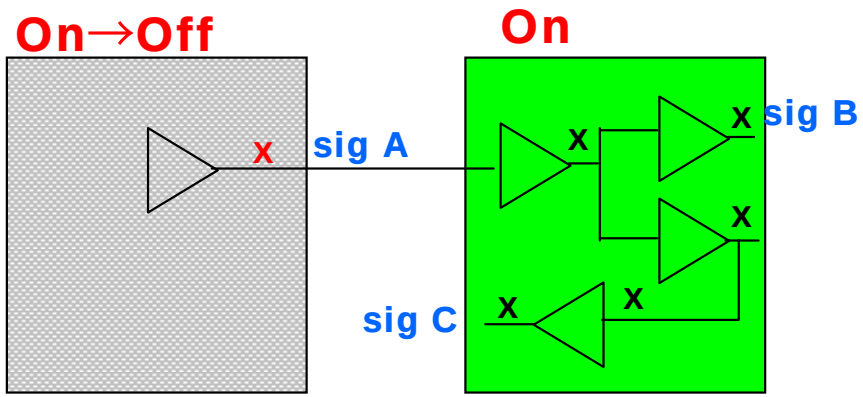
Power-Aware Debug Introduction

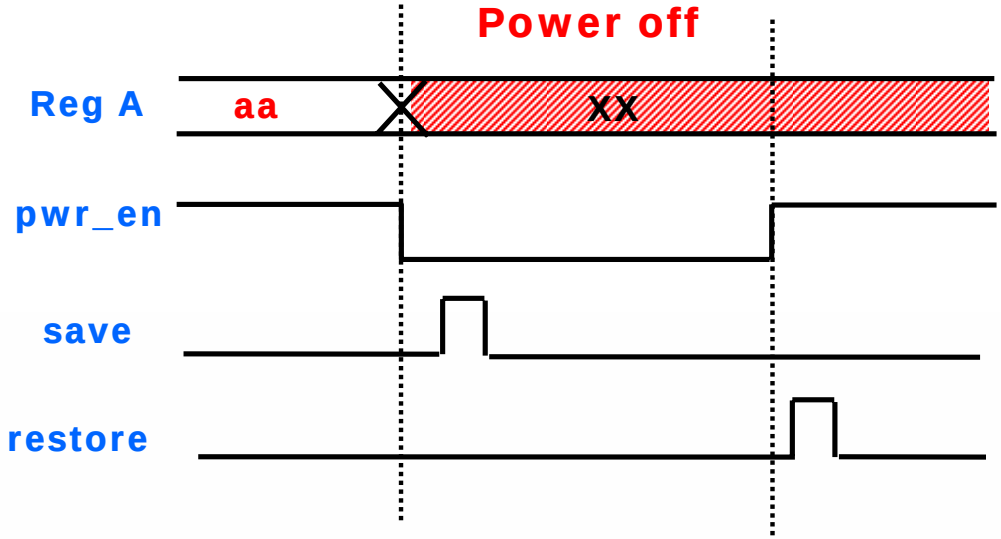
Power-Aware Debug Challenges

- Debug what causes “x”

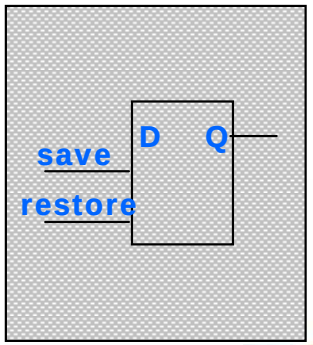


Missing Isolation Cells



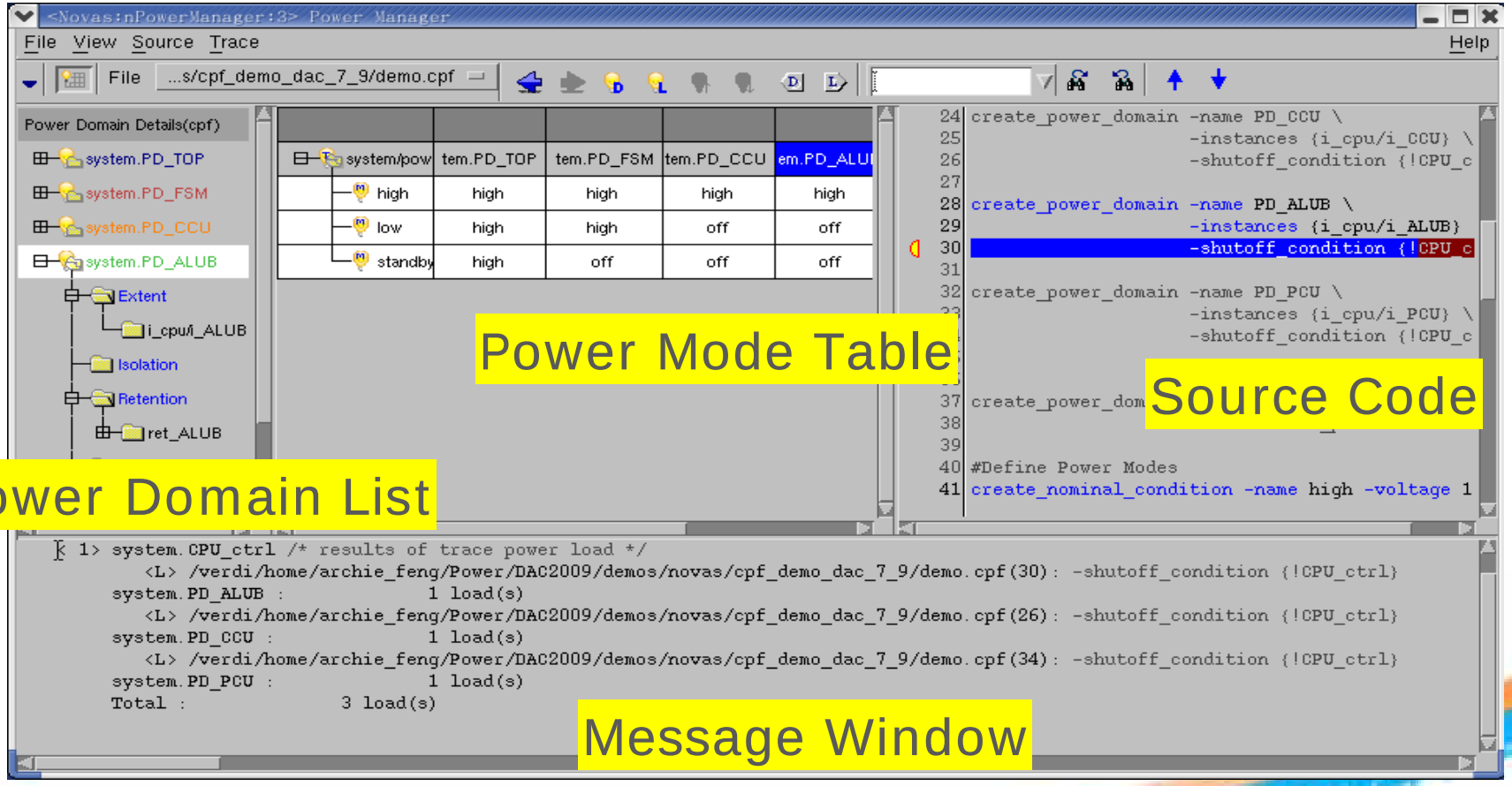


Incorrect save & restore sequence



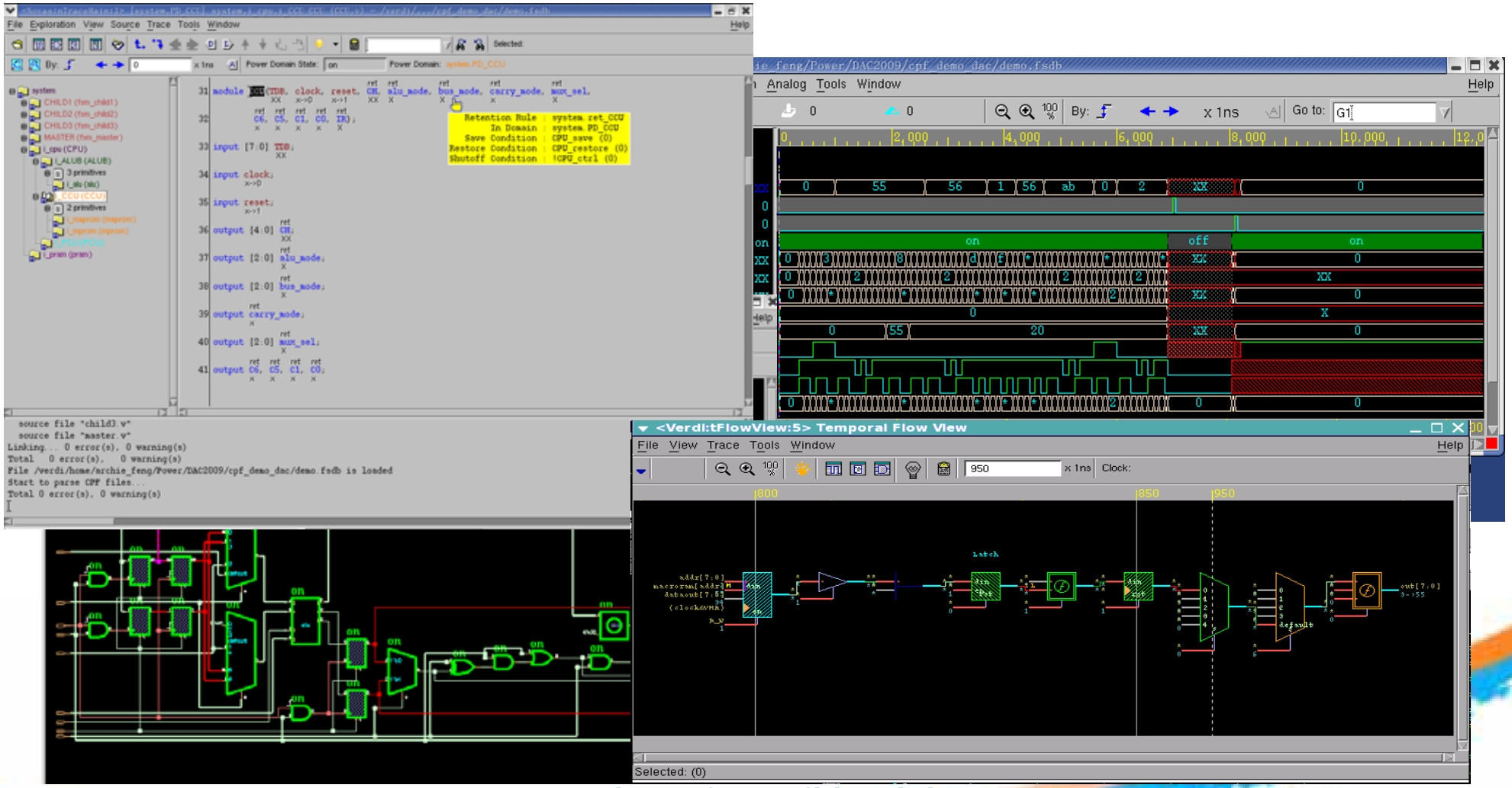
Comprehension – Power Manager

- Provide information for each power domain
- Easily locate the power intent rules
- Monitor the power mode transition

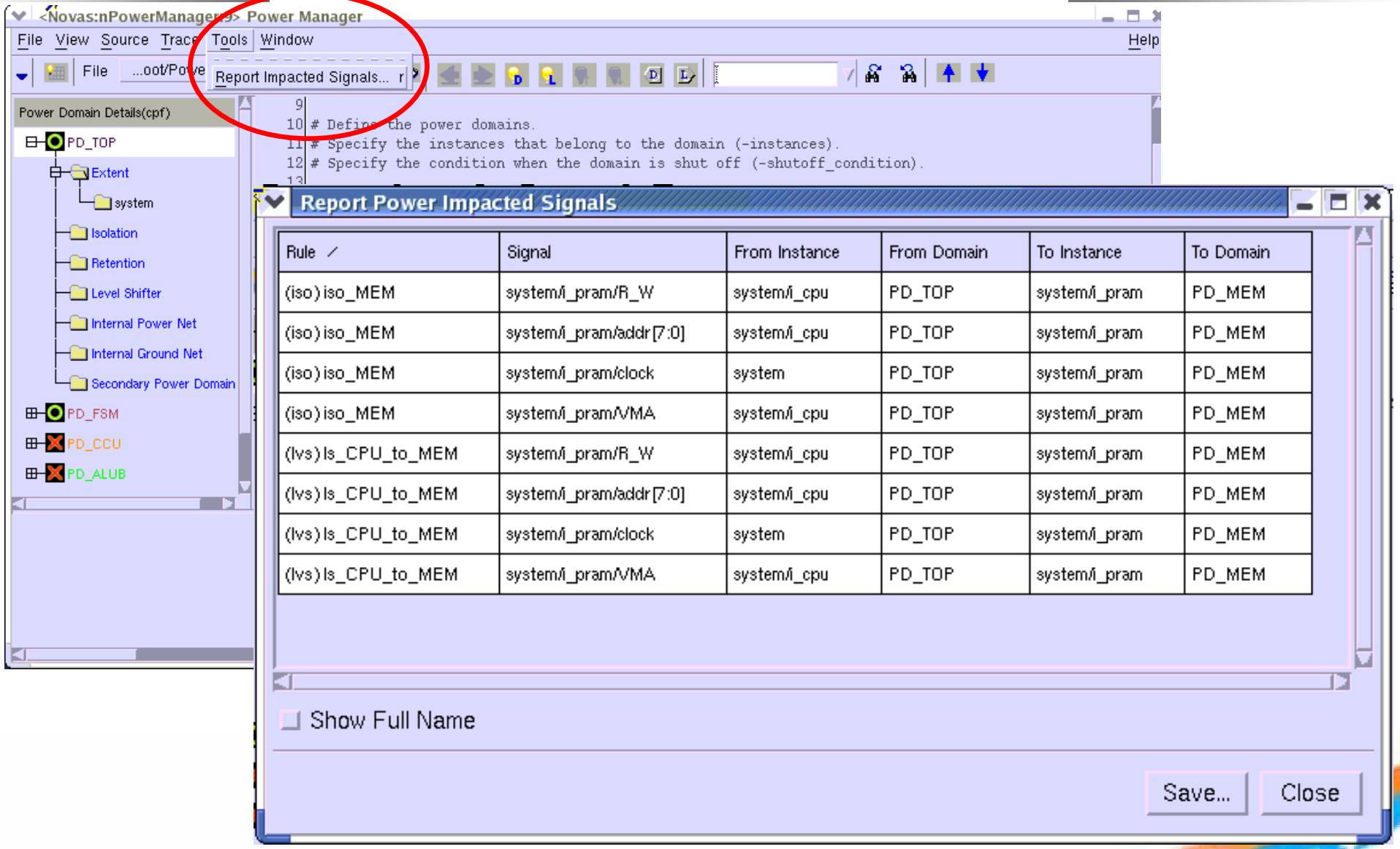


Comprehension – Power Aware Views

- Provide specialized visualization for power intent
- Tip windows for detailed power design information



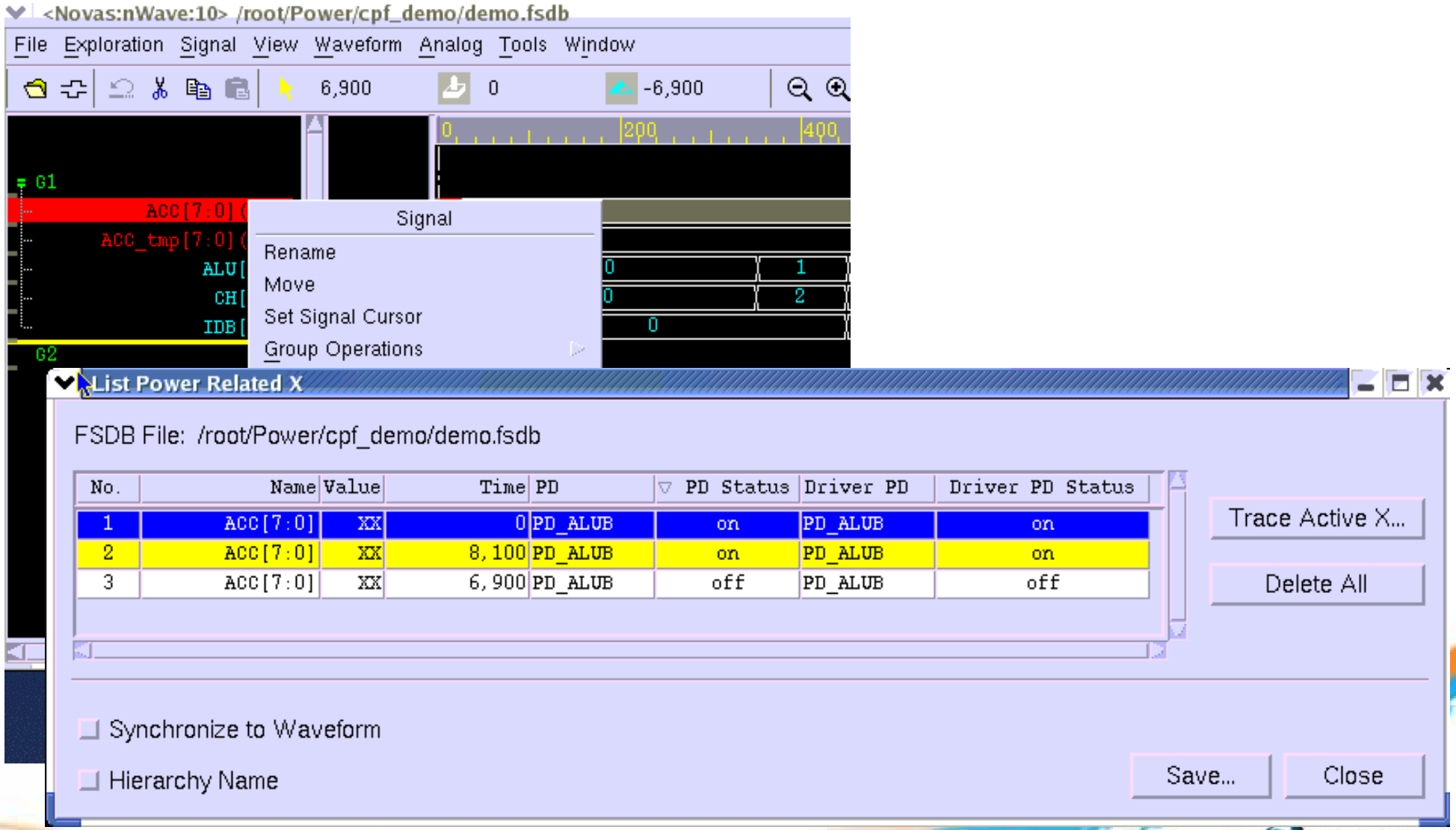
Power Impacted Signals Auto-Extraction



Rule	Signal	From Instance	From Domain	To Instance	To Domain
(iso) iso_MEM	system/i_pram/R_W	system/i_cpu	PD_TOP	system/i_pram	PD_MEM
(iso) iso_MEM	system/i_pram/addr[7:0]	system/i_cpu	PD_TOP	system/i_pram	PD_MEM
(iso) iso_MEM	system/i_pram/clock	system	PD_TOP	system/i_pram	PD_MEM
(iso) iso_MEM	system/i_pram/VMA	system/i_cpu	PD_TOP	system/i_pram	PD_MEM
(lvs) ls_CPU_to_MEM	system/i_pram/R_W	system/i_cpu	PD_TOP	system/i_pram	PD_MEM
(lvs) ls_CPU_to_MEM	system/i_pram/addr[7:0]	system/i_cpu	PD_TOP	system/i_pram	PD_MEM
(lvs) ls_CPU_to_MEM	system/i_pram/clock	system	PD_TOP	system/i_pram	PD_MEM
(lvs) ls_CPU_to_MEM	system/i_pram/VMA	system/i_cpu	PD_TOP	system/i_pram	PD_MEM

Power-Aware X Detection

- Select signals with "X", then Tools → List Power Related X



Lab11 (1/7)

- Import CPF file and understand CPF by Power Manager
  - Import Design
    - verdi -f run.f &
  - Load CPF/UPF power file into Verdi
    - **File → Import CPF/UPF Files**
    - Select **CPF1.1** in the Language field, and double click on the file '**rtl.cpf**' then click the OK button.
    - After loading is complete, the Power Manager window is opened.
  - Understand power information in Power Manager
    - Expand the "**PD_CCU**" node in the left pane to show the shut-off condition and Retention rule for PD_CCU power domain
    - Double click on the node in the browser
  - List the signals which impacted by isolation/level shifter rule
    - In Power Manager, **Tools → Report Impacted Signals**

Lab11 (2/7)

- Power Intent Annotation in nTrace
  - Show different power domain with different color in hierarchy browser.
    - Put cursor on "i_ALUB" (under system.i_cpu) , a tip window will be pop up for showing the power information.
    - Double-click "i_ALUB" to show the source code which have ret annotation on some of the signals. (the annotation is above the signal IXR)
    - Double-click "i_pram" (under system) to show level shifter and isolation annotations.
    - Put mouse cursor on one of the "i/l" signals to show the tip window with Isolation and Level Shifted rules
  - Cross probing between nTrace and Power Manager
    - Select "i_pram" in nTrace hierarchy browser and drag and drop to Power Manager window.
    - Go to the Power Manager window – it locates the source code of i_pram power intent.

Lab11 (3/7)

- Power Intent Annotation in nSchema
  - Show different power domain with different color in schematic.
    - Double-click "i_cpu", click nSchema icon. Point out different color annotation in schematic. This is a quick way to see number of power domain at hierarchy system.i_cpu.
    - Select "ALUB", and put mouse cursor on the "Power Domain: system.PD_ALUB" bar. It shows the shut-off condition.
    - Double-click on "ALUB", and show all instance with the same color. (They have the same power domain).
    - Select one of the red signal (retention signal) and show "Power Signal Type: ret" on the top of nSchema.
    - Put mouse cursor on the bar of " Power Signal Type: ret", a tip window shows save/restore condition.

Lab11 (4/7)

- Debug Design with Power Intent Annotation
  - Learn how power information is visualized in the trace results
  - In nTrace, **File → Load Simulation Results → load rtl.fsdb**
  - Open nWave. Drag-and-drop the "**i_ALUB**" scope from the nTrace window to the nWave window.
  - Select the signal "**ALU[7:0]**" in the nWave window. Click on transition **3 → 55** at time 951.
  - Double-click on the transition to show the active driver in the nTrace.
  - Turn on active annotation (**Source → Active Annotation**) in the nTrace.
  - Select signal "**a**". Do active trace by **Ctrl+t**
  - The power domain and on/off state are shown on the bar and message window.
  - Select signal "**IDB**". Do active trace by **Ctrl+t**
  - The trace result is on another power domain. With power intent annotation, you can understand and debug how CPF/UFP change your data path.

Lab11 (5/7)

- Debug Missing Isolation Rule
  - Go to the nWave window and zoom to 100%
  - Turn on the option **View → Power → Mask Power Off**, Verdi will automatically identify the X caused by power off and marked them in a shade mask.
  - Drag & Drop the scope **i_pram** from nTrace to nWave. See the waveform around time 3900ns, the signal **data[7:0]** has X, but doesn't have mask which means it's not caused by the power off.
  - Select **data[7:0]** in nWave Signal Pane, click on right mouse button and invoke the command **Power → Add Power Domain**. You can see the power is on.
  - Double-click on the transition to X at time 3900 of **data[7:0].**
  - Drag & Drop **C1** and **IDR** from nTrace to nWave. These signals have the shade mask on it which means it is caused by power off.
  - Select **C1** and click right mouse button, invoke the command **Power → Add Power Domain** to show the power is off.
  - The X was caused by missing isolation rules. We should have isolation cells on C1 and IDR so it doesn't corrupt the signals in power on module.

Lab11 (6/7)

- Debug Power Sequences
  - Lets' look at other Unknowns. Zoom out 100%. Let's start with **S1** and see what causes it to be X.
  - Select **S1** and invoke **Right-Mouse-Button → Power → Add Power Domain**. Why S1 still X after power is turn on?
  - Double-click on the transition to X at time 6850 of S1.
  - Select **CH[0]**. Do active trace by **Ctrl+t**
  - Show CH has ret annotation and it is in a different power domain. (check the power domain on toolbar). Put mouse cursor on CH to show save/restore condition. Drag & Drop signal **CH** to nWave.
  - Select CH[7:0], invoke **Right-Mouse-Button → Power → Add Power Domain.**
  - Trace the control signals for retention rule. Select CH[7:0], invoke **Right-Mouse-Button → Power → Retention rules → Add Ret controls.**
  - The X cause by the retention of CH is not correct. The save sequence is too late (after power off).

Lab11 (7/7)

- Select **CH[7:0]**, invoke **Right-Mouse-Button → Power → Retention rules → Locate Ret. Command.** This will jump to the Power Manager and automatically locate the rule.
- In Power Manager, turn on **Source → Active Annotation**. The power annotations will be shown in the right pane.
- The power state will also be shown in the middle pane (the Power Mode Table). Verdi automatically identify the power states at any given simulation time.
- In nWave, move the mouse cursor to time 3900 to show the "**low**" state is highlighted in the Power State Table.
- In nWave, select **CH[7:0]** and invoke **Right-Mouse-Button → Power → Power → Add Power Mode**, to trace power states in nWave.
- In Power Manager, you can also trace the active driver in HDL side.
- Select **CPU_save** in the Source Code Pane of Power Manager, invoke **Right-Mouse-Button → HDL Active Trace**. It locates the active driver in nTrace Source Code Pane.

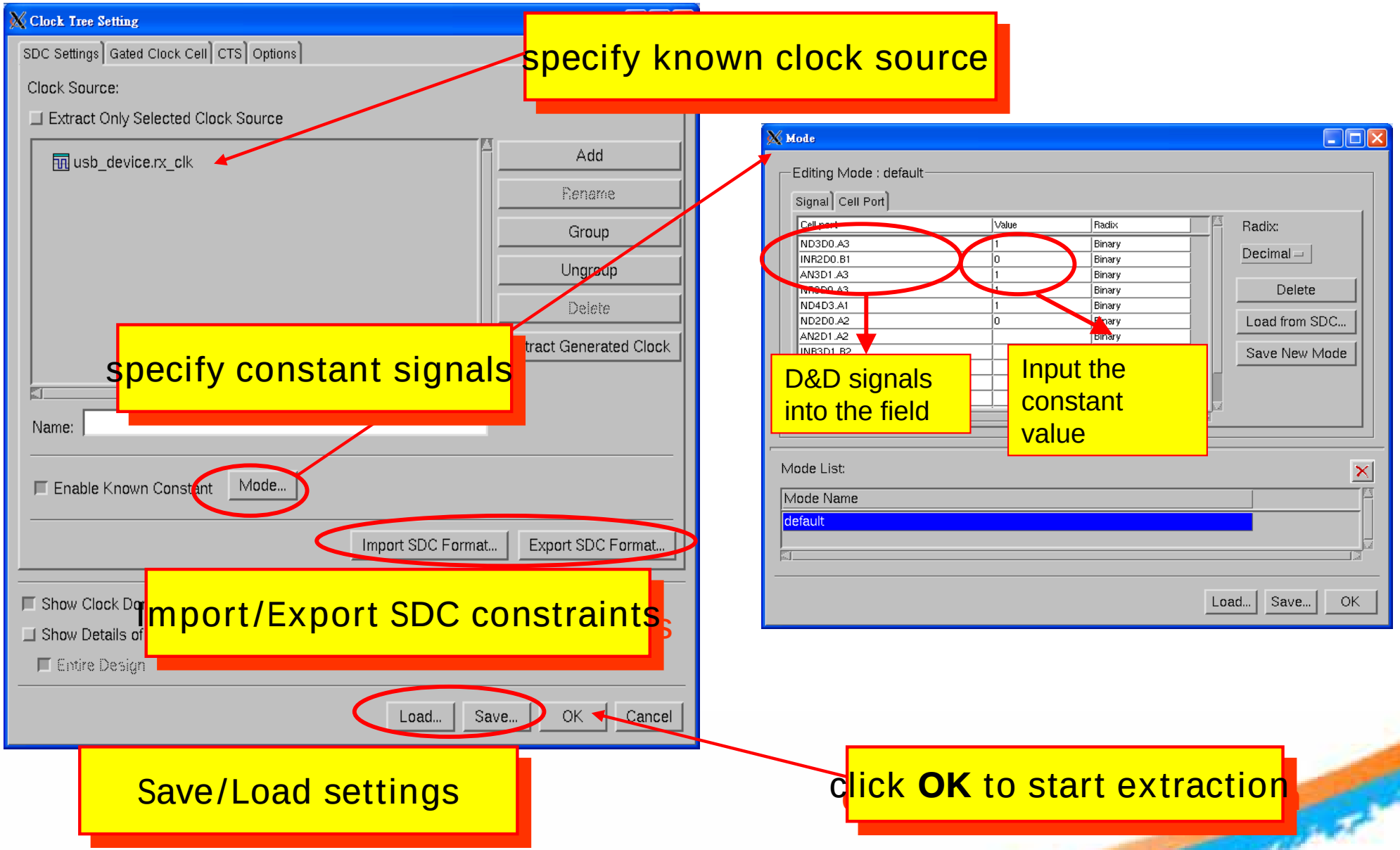
Analyze Multi-Clock-Domain

- Common Problem
  - There are too many clock domains in a design
  - Hard to check data path crossing different clock domain to satisfy synchronization
- Novas Solution
  - *Extract Clock Domain* to analyze how many clock domains in a design
  - *Find Crossing Paths* to find out data path crossing different domain and check out synchronizer simultaneously

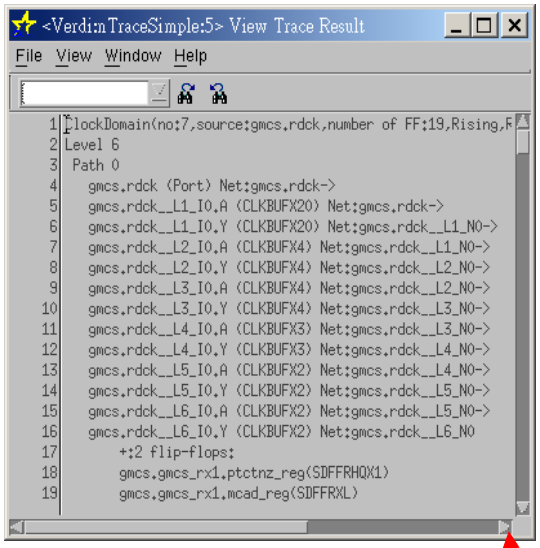
Analyze Multi-Clock-Domain
Extract Clock Domain(1/3)

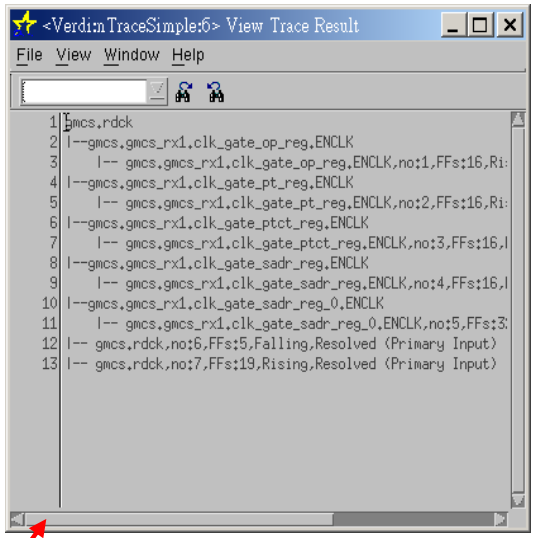
- Purpose
  - extract all clock domains in design
- Usage
  - *nTrace→ Tools→ Clock Analyzer→ Extract Clock Information*
  - *nSchema→ Trace→ Clock Analyzer→ Extract Clock Information*
  - *nTrace→ File→ Load Clock Domain Result* To load the previous result
- Support to set known clock source & constant signals
  - Support to read SDC to map these settings

Analyze Multi-Clock-Domain
Extract Clock Domain(2/3)



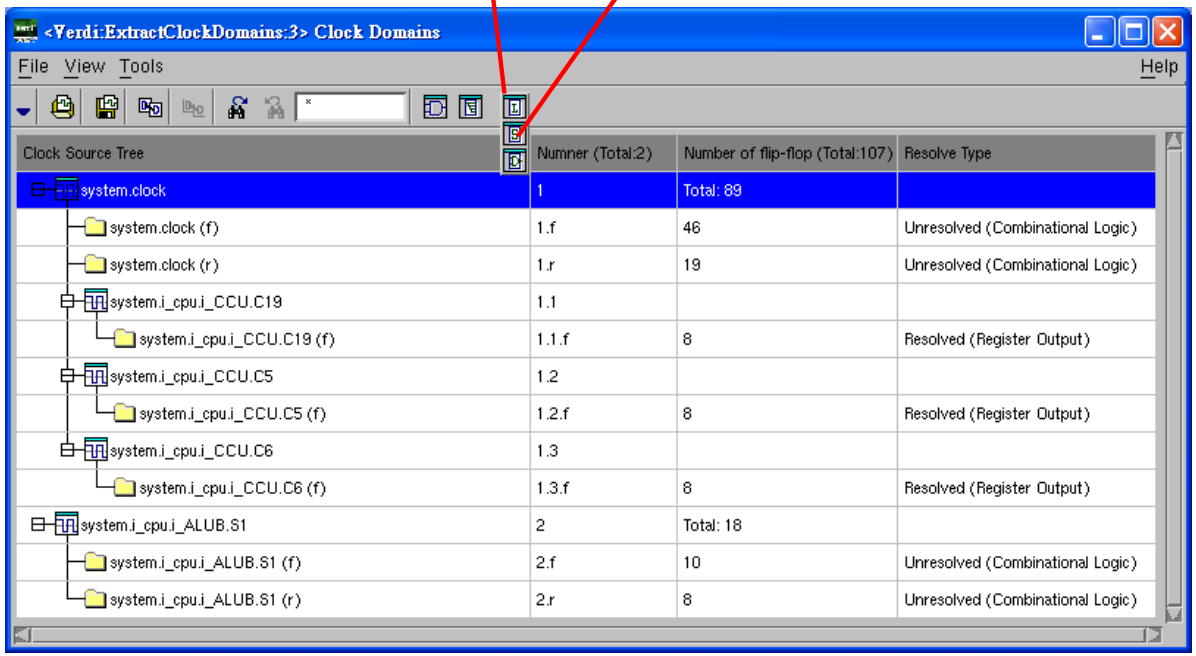
Analyze Multi-Clock-Domain
Extract Clock Domain(3/3)



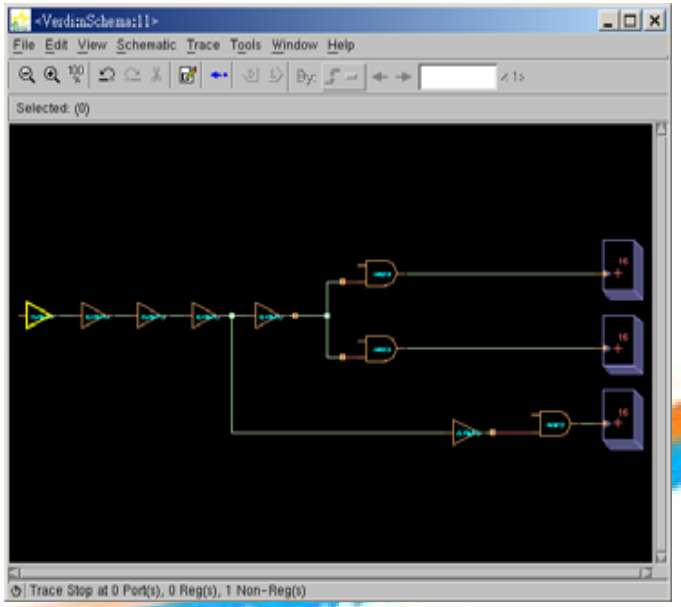


1. A clock source tree is established to show root/generated clocks with the storage elements' numbers

2. Select specific domain to show by schematics or text



Clock Source Tree	Numner (Total:2)	Number of flip-flop (Total:107)	Resolve Type
system.clock	1	Total: 89	
system.clock (f)	1.f	46	Unresolved (Combinational Logic)
system.clock (r)	1.r	19	Unresolved (Combinational Logic)
system.i_cpu.i_CCU.C19	1.1		
system.i_cpu.i_CCU.C19 (f)	1.1.f	8	Resolved (Register Output)
system.i_cpu.i_CCU.C5	1.2		
system.i_cpu.i_CCU.C5 (f)	1.2.f	8	Resolved (Register Output)
system.i_cpu.i_CCU.C6	1.3		
system.i_cpu.i_CCU.C6 (f)	1.3.f	8	Resolved (Register Output)
system.i_cpu.i_ALUB.S1	2	Total: 18	
system.i_cpu.i_ALUB.S1 (f)	2.f	10	Unresolved (Combinational Logic)
system.i_cpu.i_ALUB.S1 (r)	2.r	8	Unresolved (Combinational Logic)

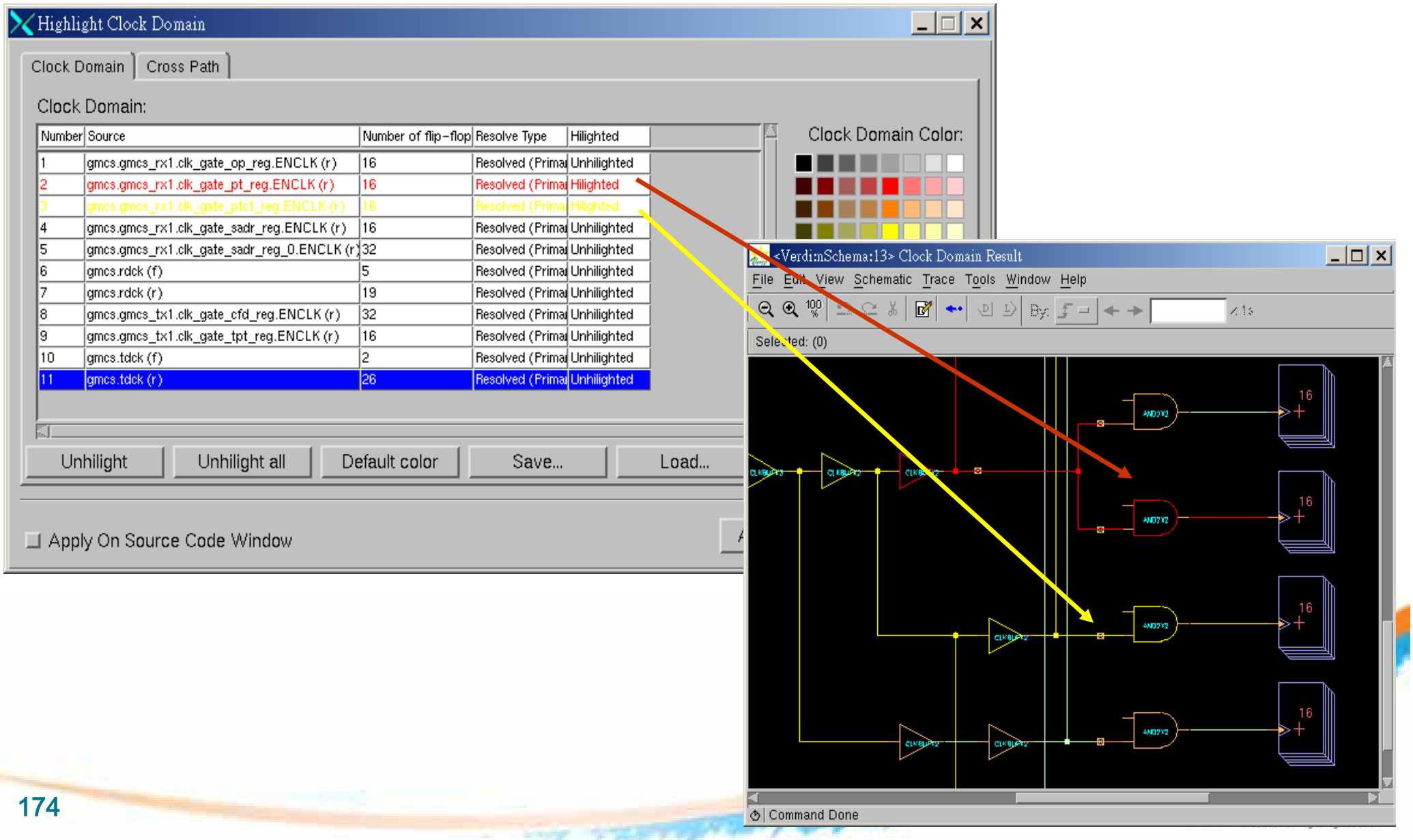


SpringSoft Accelerating Engineers

Analyze Multi-Clock-Domain
Highlight Clock Domain(1/2)

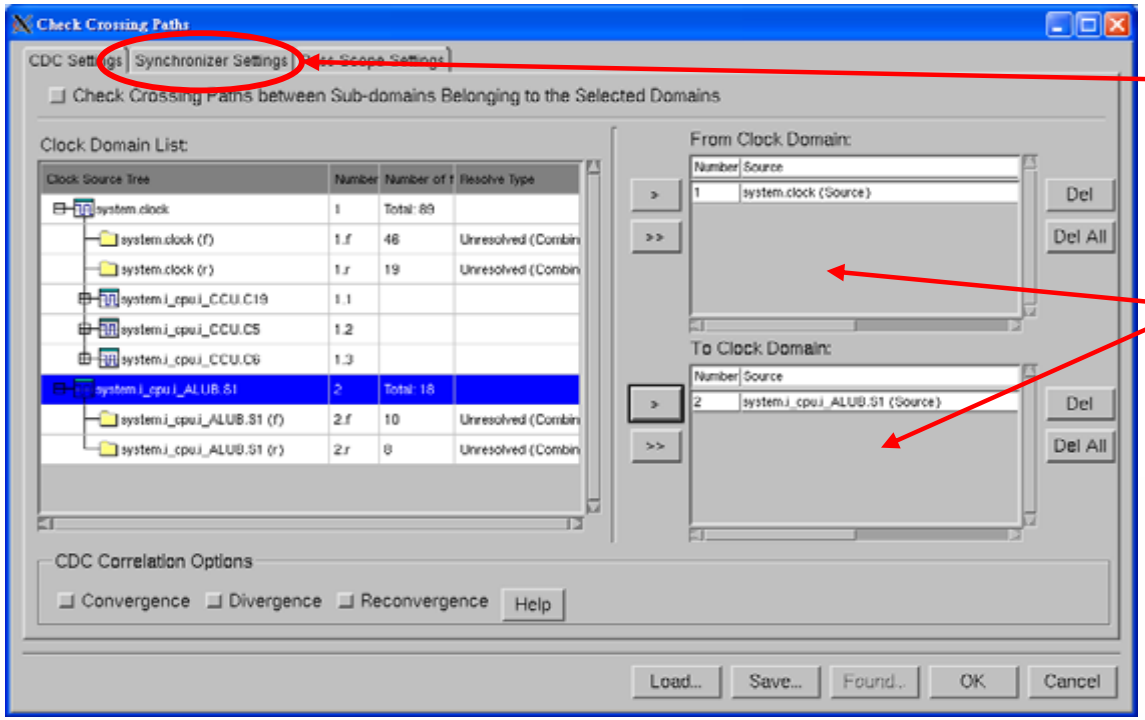
- Purpose
  - Useful to identify different clock domain when debugging
- Usage
  - *Schematic→ Trace→ Clock Analyzer→ Highlight Clock Domains*
- The color applies to all schematic windows

Analyze Multi-Clock-Domain
Highlight Clock Domain(2/2)



Analyze Multi-Clock-Domain
Check Crossing Paths(1/3)

- Purpose
  - Find crossing paths
  - Check synchronizer
- Usage
  - *Extract or load clock domain at first*
  - *Schematic→ Trace→Clock Domain→ Check Crossing Paths*
- Support multiple-to-multiple crossing paths check

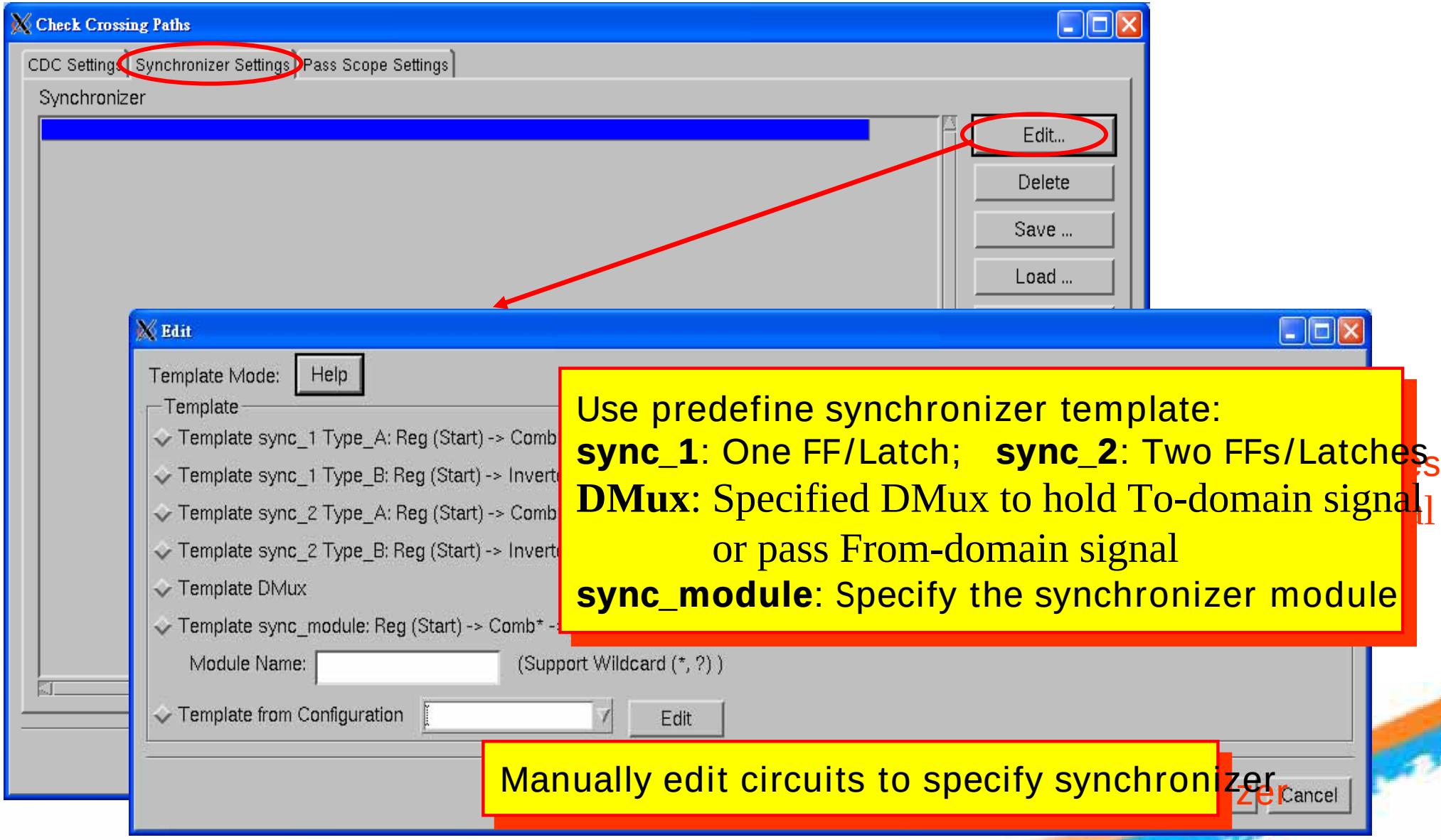


Specify synchronizer

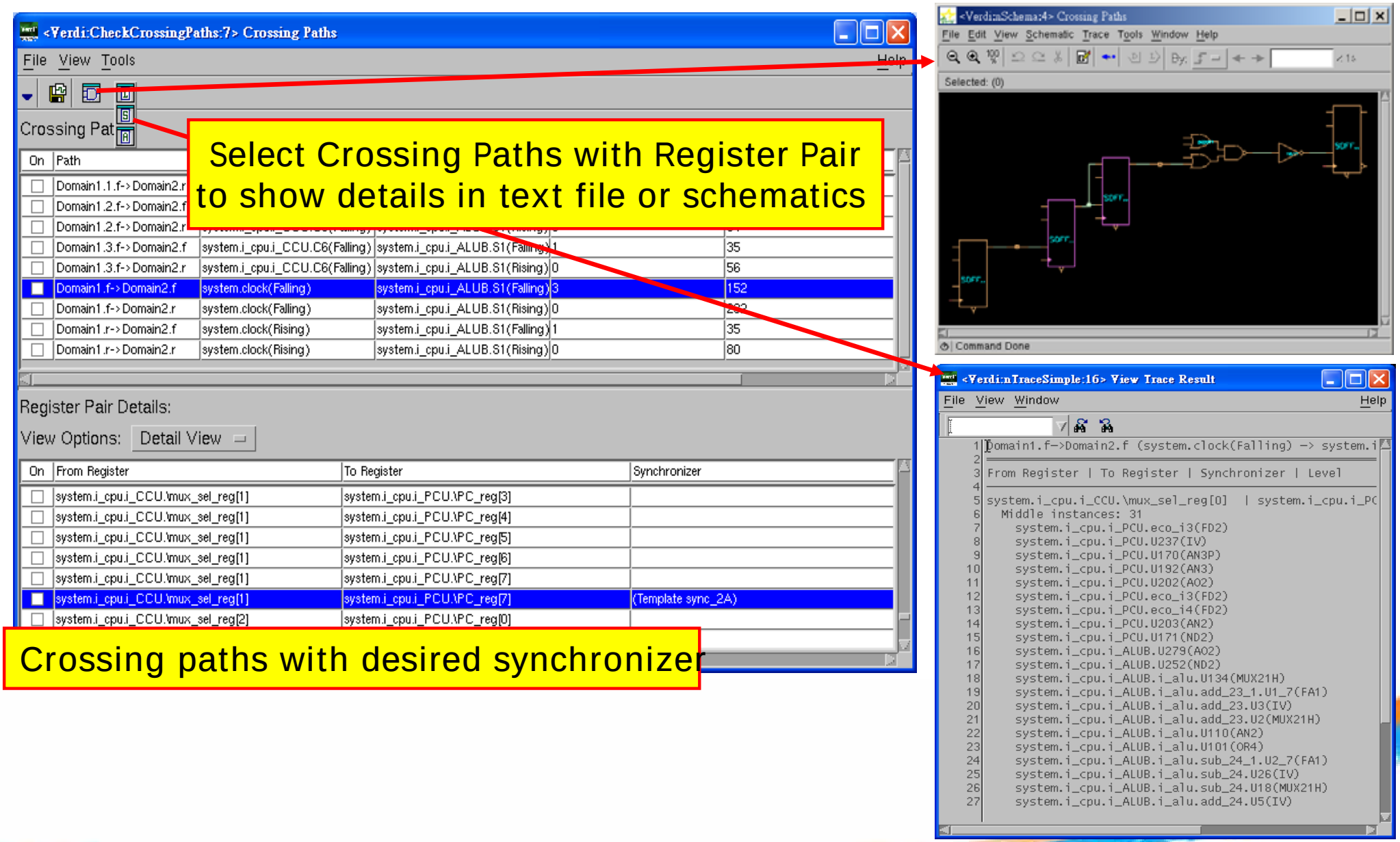
Select multiple From/To Domains to check

SpringSoft
Accelerating Engineers

Analyze Multi-Clock-Domain
Check Crossing Paths(2/3)



Analyze Multi-Clock-Domain
Check Crossing Paths(3/3)



SpringSoft Accelerating Engineers

Lab12 (1/5)

Extract Clock Domain to analyze clock domains in a design

- Invoke Verdi and import design
  - ./RUN
- Open Clock Tree Setting form by **Tools → Clock Analyzer → Extract Clock Information**
- **Click** "Import SDC Format" to **load** "sdc_system.sdc"
- **Click** OK to open the Clock Domains window
  - There is an unresolved clock domain (combinational)
- In the Clock Domains window, **select** the system.i_cpu.i_ALUB.S1 (f) clock domain and click the icon on the toolbar to open nSchema

Lab12 (2/5)

- In the schematic window, enable **View → Instance Name**
- Expand the two inputs of ND2 U250 by **double-clicking** the pins
  - The net associated with port A is the output of a flip-flop
  - If the net associated with port B is assigned as a known constant signal, nAnalyzer would be able to resolve this clock domain as system.i_cpu.ICCU.CH[0].

Modify some constant setting and resolve the domain

- In the schematic window, **Trace → Clock Analyzer → Extract Clock Information**
- In the Clock Tree Setting form, **click** the Mode buton to open the Mode form and **select** Signal tab
- Drag and drop the net "n737" associated with input B of U250 in the schematic for S1 to the Mode window

Lab12 (3/5)

- Click the value column and key in "1"
  - The value setting with 1 will allow the output to be traced through to the A input of U250
- Toggle the "Radix" option to "Binary"
- In the Mode form, click "OK" button
- Click "OK" in the Clock Tree Setting window
- Click "No" button on the Question dialog window
  - A Clock Domain window displays the results. Compare with the domain before modifying the constant setting
    - There are 7 sub-clock domains and all of them are resolved
- The results can be displayed again by invoking **Tools → Clock Analyzer → List Clock Domains** in nTrace or **Trace → Clock Analyzer → List Clock Domains** in nSchema

Lab12 (4/5)

Find Crossing Paths between different domains and check out synchronizer simultaneously

- Open Check Crossing Paths form by invoke **Tools → Check Crossing Paths** in the clock domains form
- Expand all clocks under clock by clicking the "+" symbol and expand system.clock, too.
- **Click** in "From Clock Domain:" and "To Clock Domain:" to add all clock domains
- **Enab** >> "Check Crossing Paths between Sub-domains Belonging to the Selected Domains" option
- **Click** the tab "Synchronizer Settings" in the Check Crossing Paths form
  - **Click** "Edit" button and choose Template sync_2 Type_A and then **click** "OK" button

Lab12 (5/5)

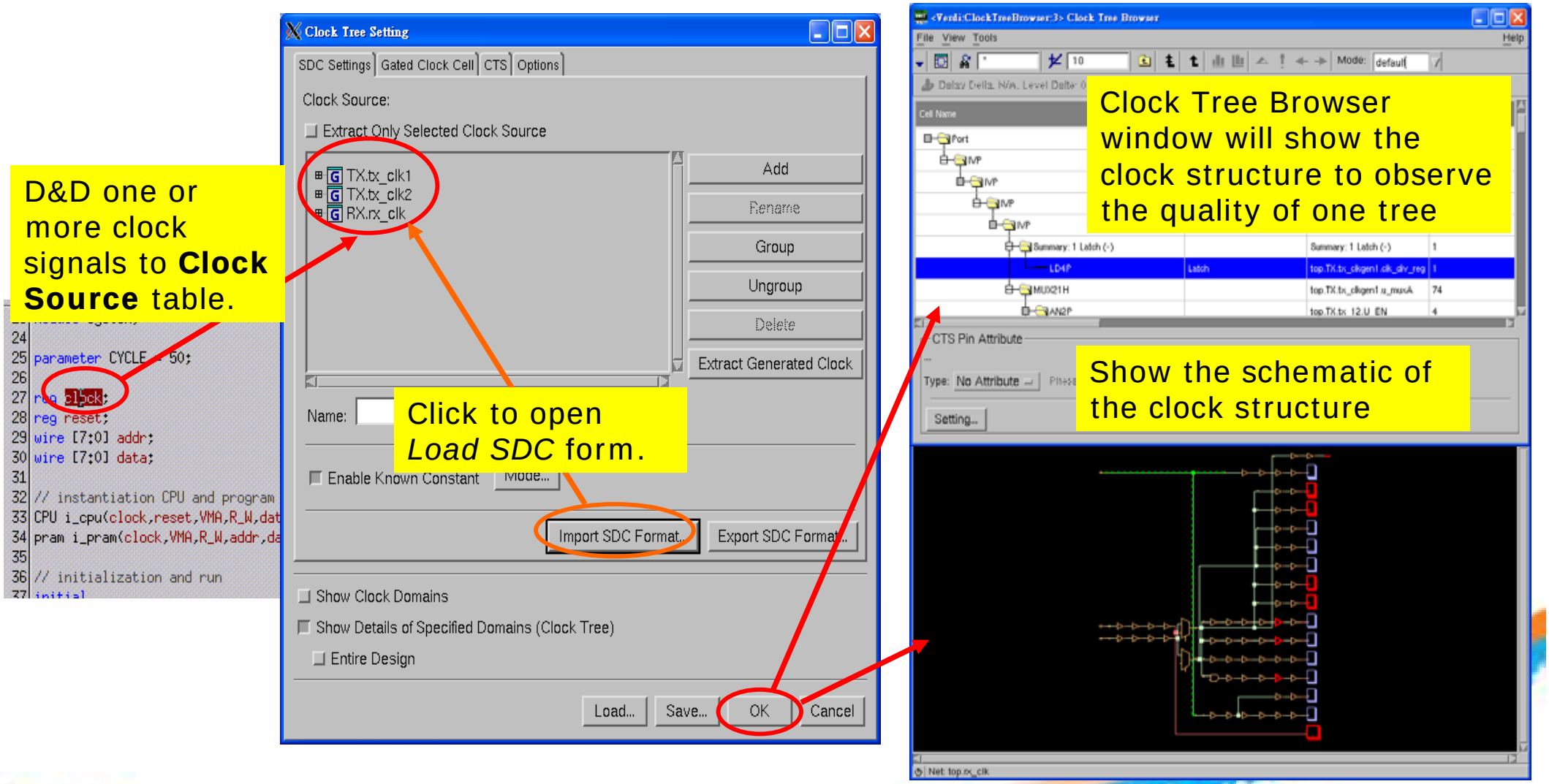
- Click the tab "CDC Setting"
  - Click Convergence, Divergence, and Reconvergence in the CDC Correlation Option field, then click OK
- In the crossing paths browser, you can easily check the synchronizer information or even CDC correlations information.
- Click the path, "Domain 1.f Domain1.5.r"
  - There are 3 paths with the synchronizer of sync_2 Type_A
- Select the first line on the "CDC Correlation Details" field and click New Schematic  icon
- In the schematic, invoke View → High Contrast to display the highlighted color mo clearly.

Analyze Clock Distribution

- Common Problem
  - Easy to lose completeness while preparing CTS constraints for APR tool
  - Clock tree is too complex to isolate and browse
  - Qualifying and then re-balancing clock tree takes too much time and iteration
- Novas Solution
  - *CTS Constraint Checker* to import SDC or existing CTS constraints and export new settings for the usage by popular APR tool
  - Tabular *Clock Tree Browser* to examine tree composition and level information
  - Locate worst clock skew and show histogram of delay distribution

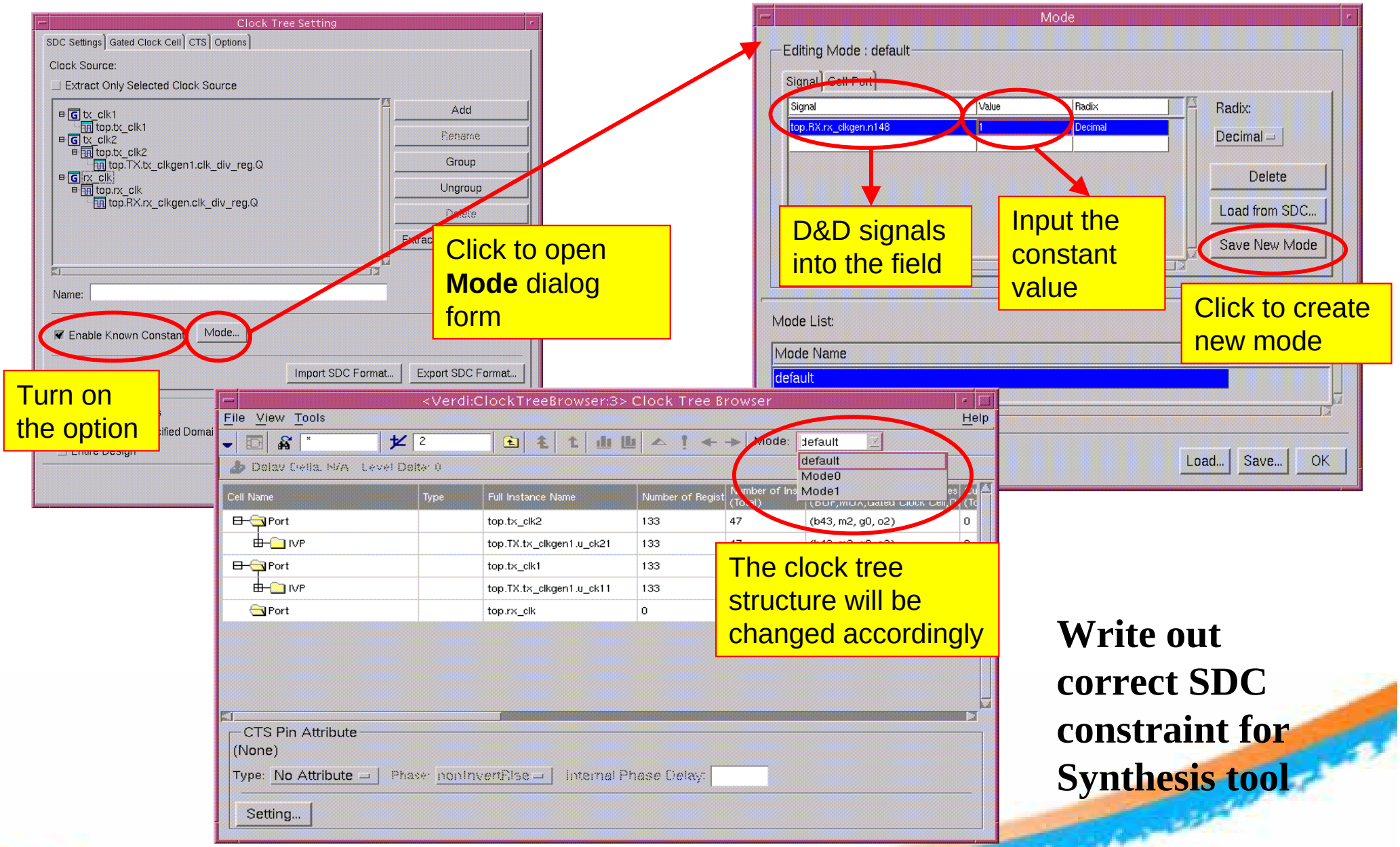
Analyze Clock Signal

- In *nTrace*, invoke **Tools → New Schematic → Clock Tree** to open the *Clock Tree Setting* form.



SpringSoft
Accelerating Engineers

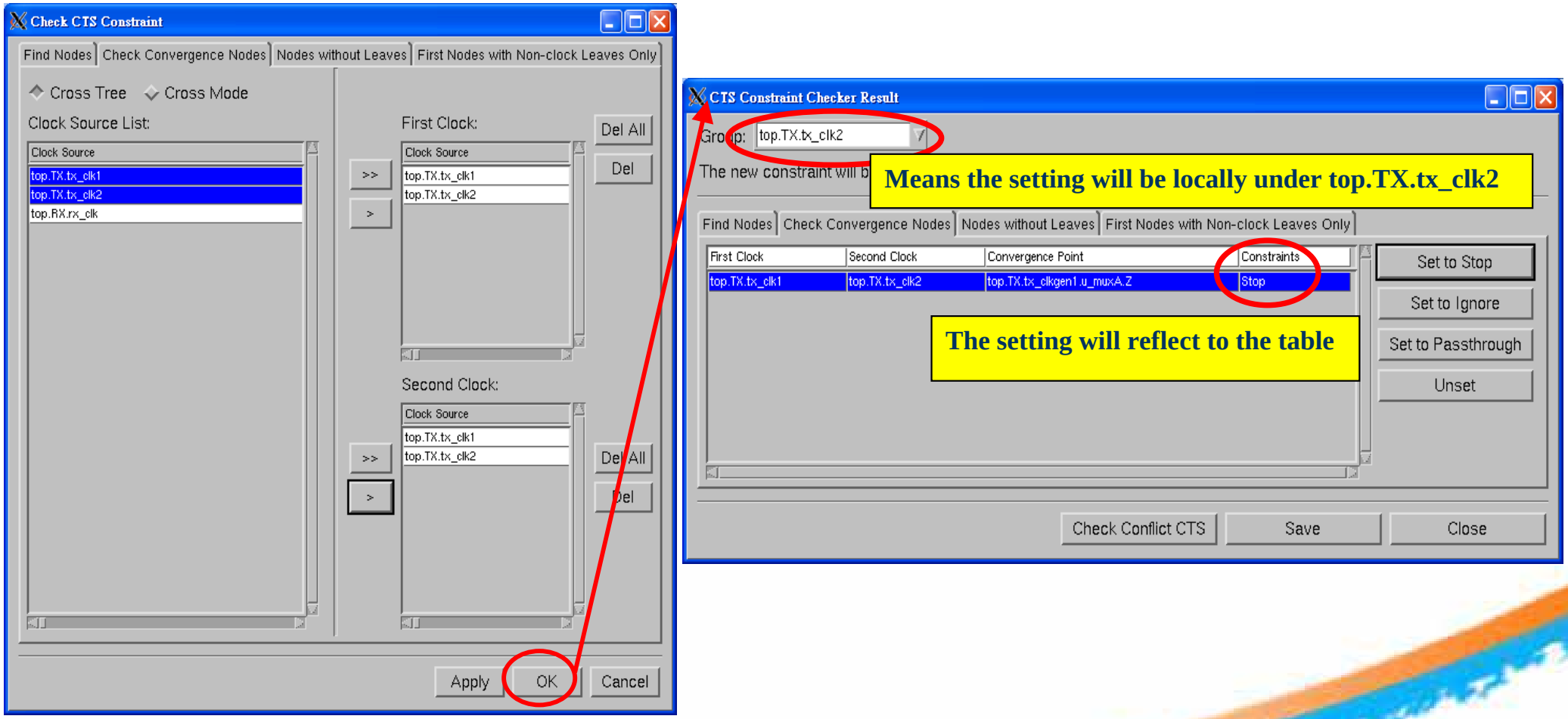
Apply Constant Settings



Write out correct SDC constraint for Synthesis tool

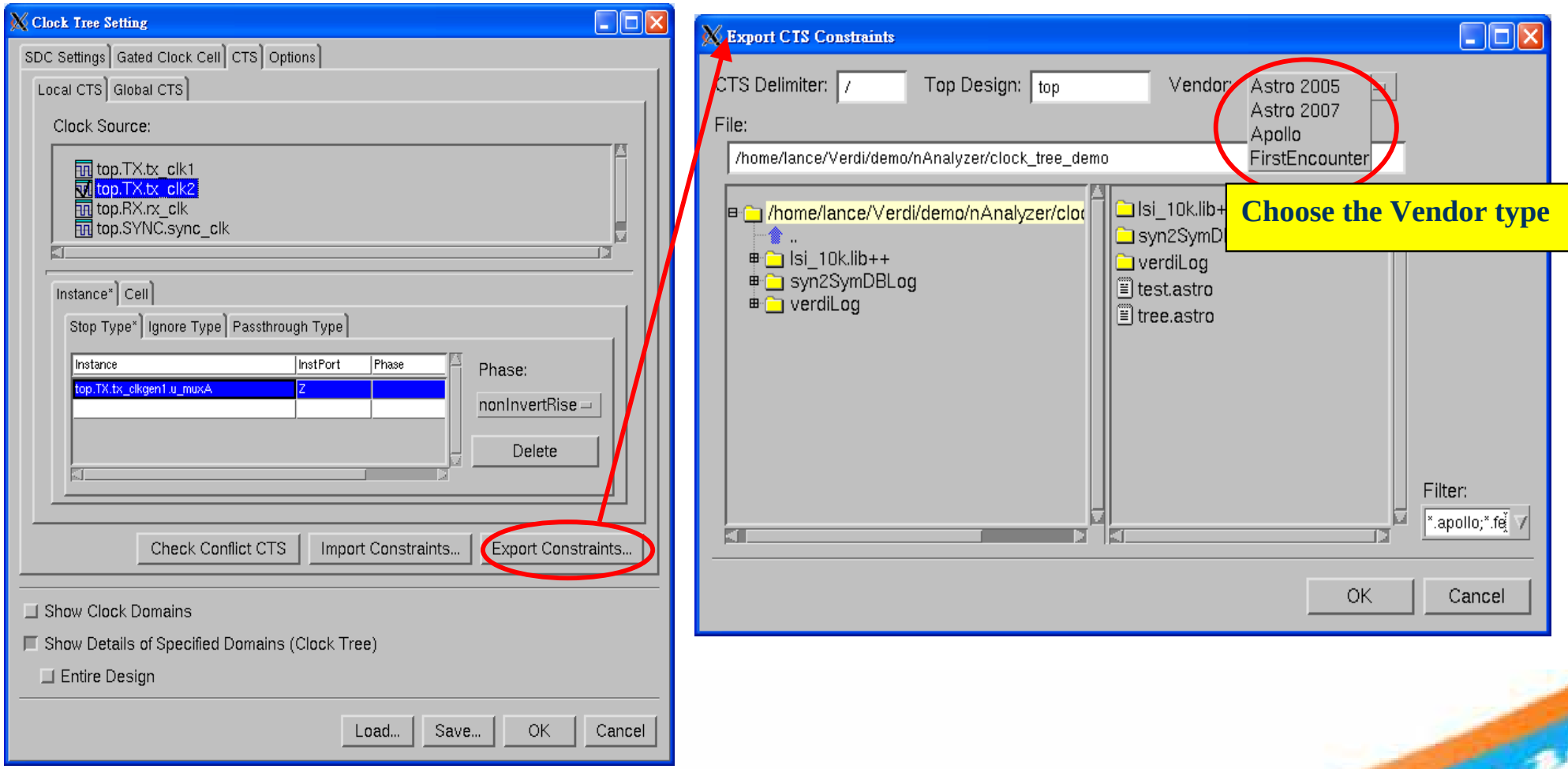
Qualify CTS Script (1/2)

- In the Clock Tree Browser window, invoke Tools → CTS Constraint Checker



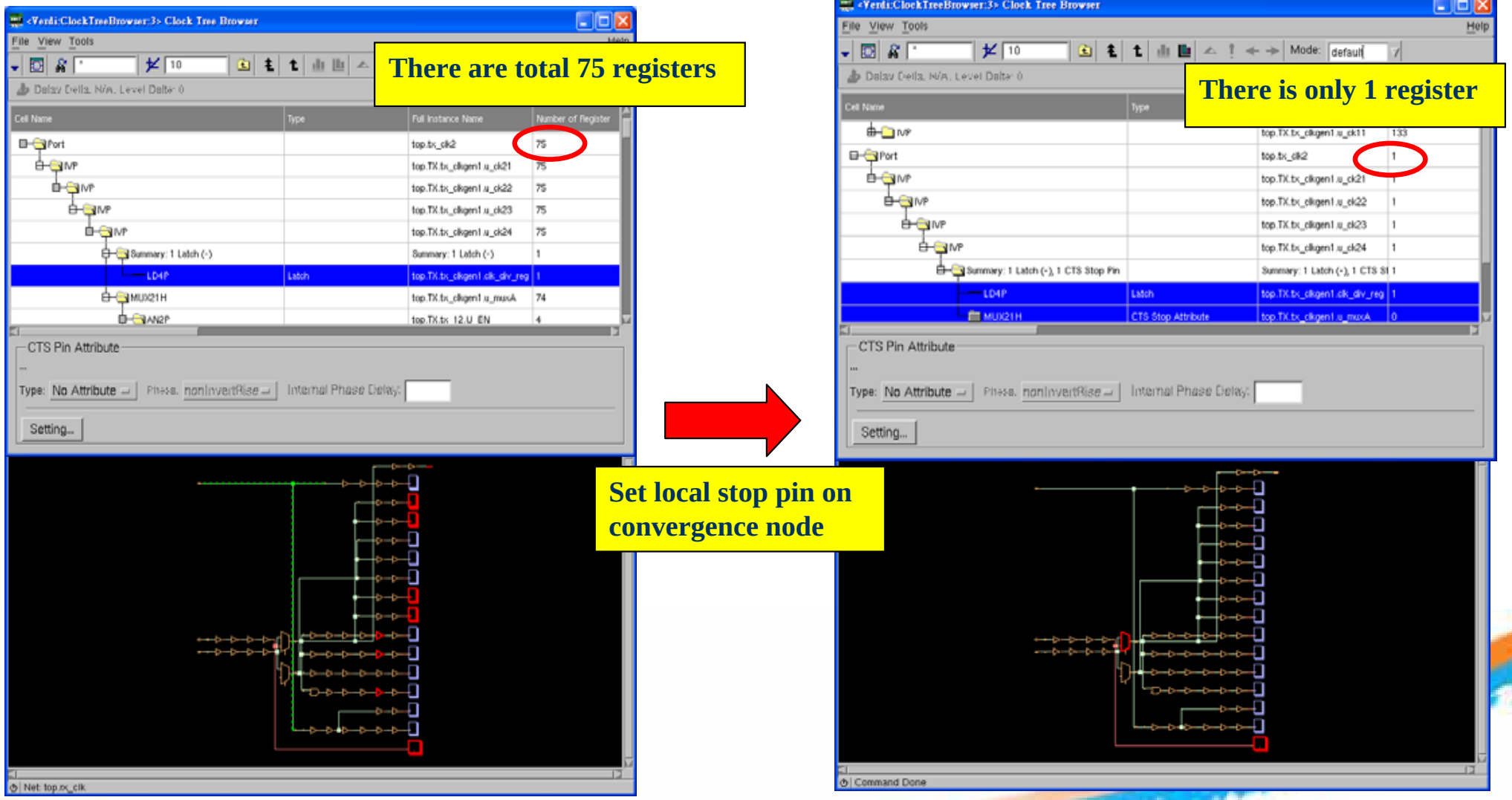
Qualify CTS Script (2/2)

- Re-extract the Clock Tree and Output the CTS Constraints File



Verify Clock Tree Quality (1/2)

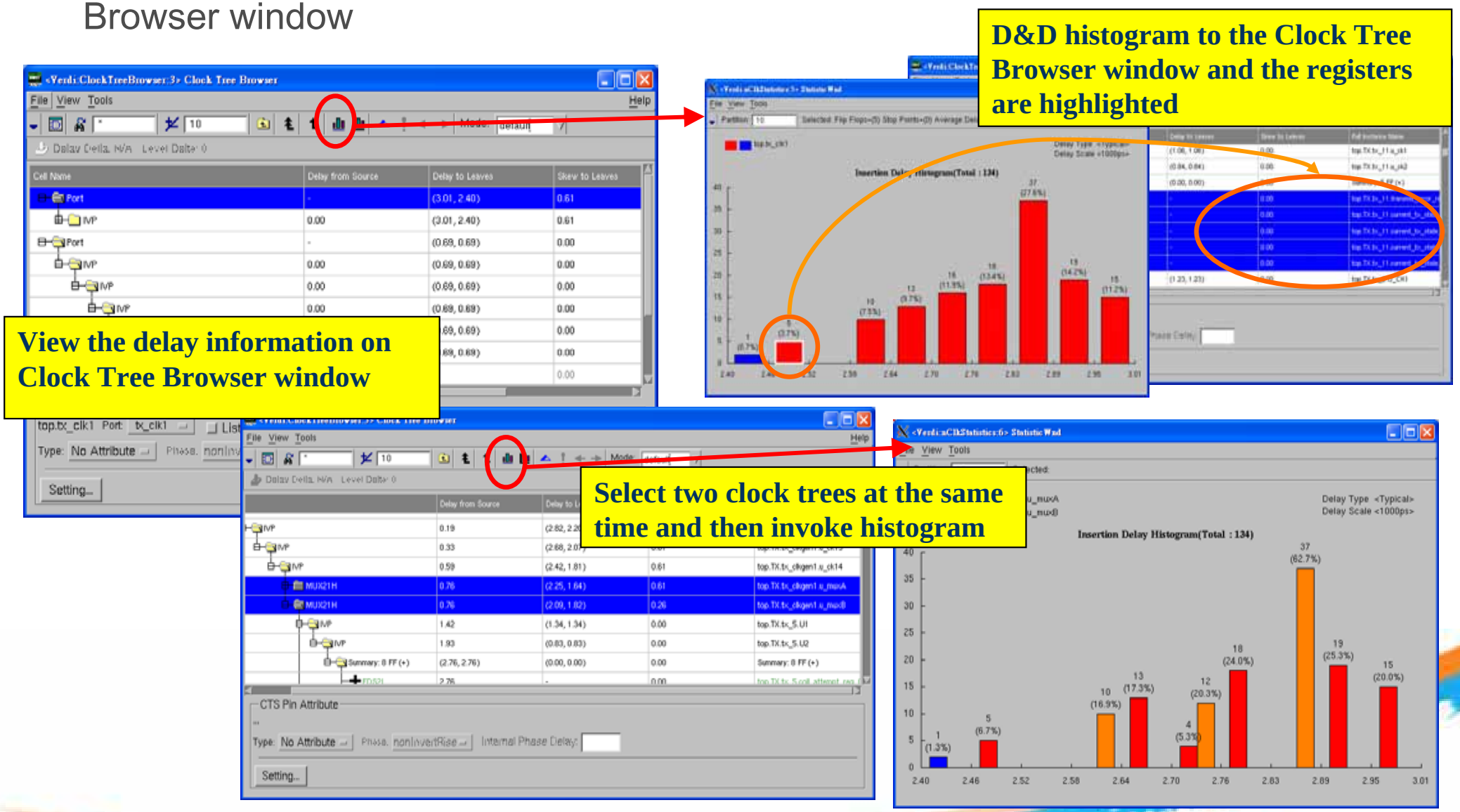
- Analyze CTS Result on Clock Tree Browser after re-extracting the Clock Tree with modified CTS



SpringSoft
Accelerating Engineers

Verify Clock Tree Quality (2/2)

- Import the SDF file by invoking File → SDF → Load SDF Files in the Clock Tree Browser window



SpringSoft Accelerating Engineers

Lab13 (1/5)

Use CTS Constraint Checker to import SDC or existing CTS constraints and export new CTS settings for APR tool

- Invoke Verdi and import design
  - ./RUN
- Open Clock Tree Setting form by **Tools → New Schematics → Clock Tree**
- **Click** "Import SDC Format" button to **load** TOP1.sdc, and then **click** OK button
- **Click** tab "Options" of Clock Tree Setting form and **enable** the option "Stop on Floating Gate", then **click** OK button
  - In the clock tree browser, you can easily recognize the clock tree structure
- In the clock tree browser, invoke **Tools → CTS Constraint Checker**

Lab13 (2/5)

- On the "Find Nodes" tab, **enable** the options "Primary Output Port" and "Floating Gates".
- In the "Check Convergence Nodes" tab, **select** "top.TX.tx_clk1" and "top.TX.tx_clk2"
- **Click** the upper ">" button and the lower ">" button to add both clocks into the "First Clock" and "Second Clock" section.
- **Click** OK button to open CTS Constraint Checker Result window.
- **Select** row 1 and then **click** the "Set to Stop" button.
  - The top.cko instance is the primary output
- **Select** row 2 and then **click** the "Set to Ignore" button.
  - The top.u_ck100 instance is the floating gate
- **Select** the "Check Convergence Nodes" tab.
  - There are two convergence nodes.
- **Switch** "Group" to "top.TX.tx_clk2", **click** "Set to Stop" button to set the two convergent nodes as stop points and as the local CTS constraints.
- **Click** the "Close" button.

Lab13 (3/5)

Re-extract the Clock Tree and Output the CTS Constraints File

- Open Clock Tree Setting form by **Tools → New Schematics → Clock Tree**
- **Click** the "CTS" tab and **select** "Local CTS" tab and clock source: "top.TX.tx_clk2"
  - You can see the local constraints for top.TX.tx_clk2
- **Click** the "Global CTS" tab
  - You can see the different constraints such as Stop Type and Ignore Type.
- **Click** the "Export Constraints" button
  - **Choose** the Vendor type as "FirstEncounter" and **key in** the file name. **Click** OK button
  - Review the constraint file that you exported.
- **Click** OK button in Clock Tree Setting form to Re-extract the clock tree.

Lab13 (4/5)

Use Clock Tree Browser to examine tree composition and level information

- How many registers under the clock top.tx_clk2? 0
- **Expand** clock tree of "top.tx_clk2" to show the leaf nodes
  - The multiplexer MUX21H is denoted as a "CTS Stop Pin"
- **Select** the first row for clock source "top.tx_clk1", and then **scroll right** to see the column Min and Max level to leaves
- What are the max and min number of level to leaves? Min: 7 Max: 11

Locate worst clock skew and show histogram of delay distribution

- Invoking **File → SDF → Load SDF Files** in the Clock Tree Browser and **select** file "top_g1.sdf"

Lab13 (5/5)

- Invoke **View → Configure Level Columns** in clock tree browser
  - On the Configure Columns form, **click** "<<" button to remove all the columns
  - Only add the "Full instance Name", "Delay from Source", "Delay to Leaves" and "Skew to Leaves" to the right panel by **selecting** each target item and then **clicking** ">" button one by one.
  - Click **OK**.
  - The clock skew and delay information are shown in the browser
- **Click** the minus sign "-" in front of all MUX21H to collapse the results belong clock source top.tx_clk1
- **Select** the two "MUX21H" rows (by CTRL key) and then click the histogram  toolbar icon.
  - In the histogram, the X-axis shows the delay values while the Y-axis shows the number of leaf nodes. The distributions of each clock tree are show by color.
  - Take the tallest bar as an example; there are 37 registers whose insertion delays are between 2.83 and 2.89.
- In nSchema, enable **Schematic → Auto Fit Found Object(s)** and then **D&D** the most left blue bar to the nSchema window.
  - It is the output port that you set Stop.

On-line Support: 思源科技中文論壇

- 關鍵字搜尋: “思源科技論壇 ”

SpringSoft
Accelerating Engineers