Verdi
Basic Training & Technical Overview

Novas Software, Inc

Based on Verdi 5.4v6

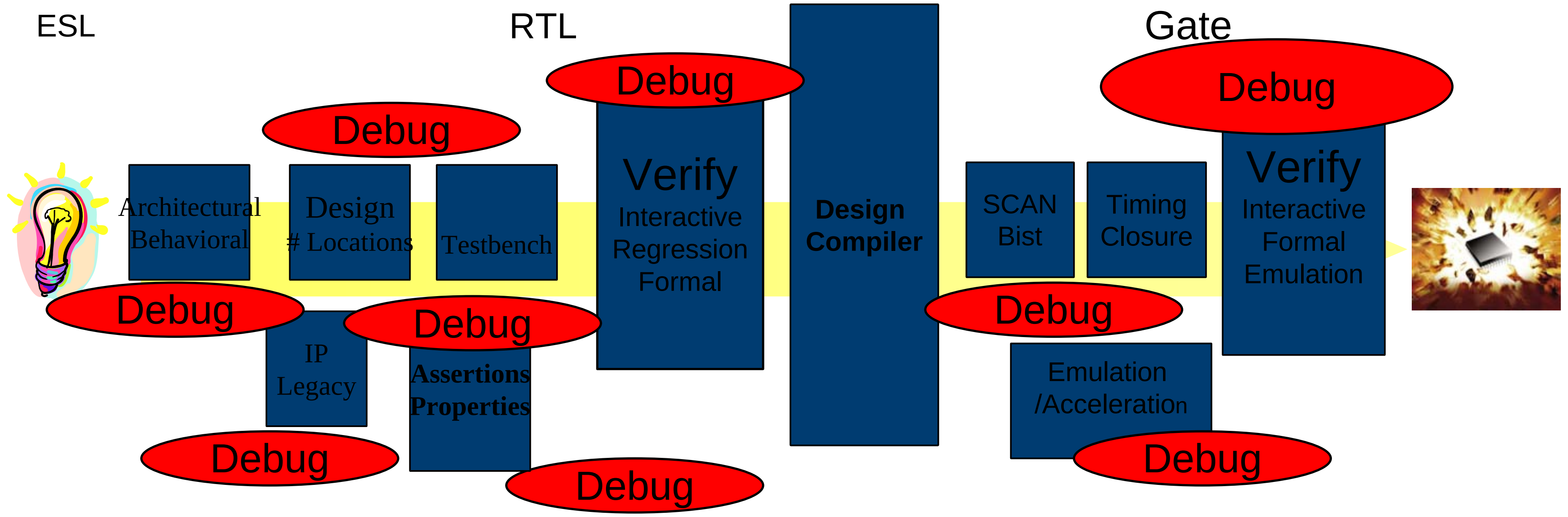
Training Course

- Verdi Background and Overview
- Loading Your Design
- nTrace
- nSchema & nState
- Break
- nWave
- Debugging with Verdi

Verdi Background and Overview

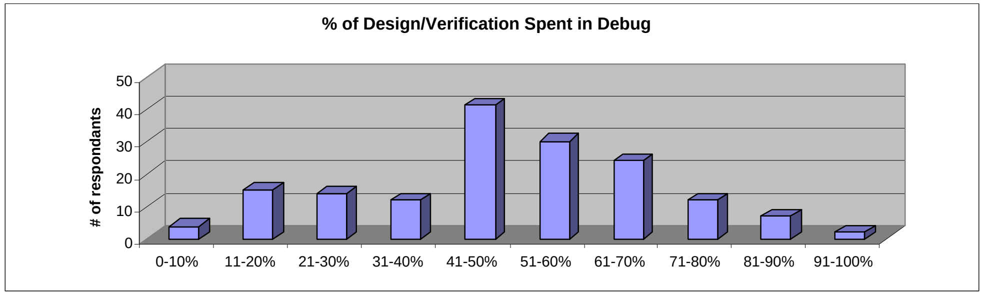
Concepts and Technology

Debug in the Design Flow



Do you know how much time your teams spend debugging?

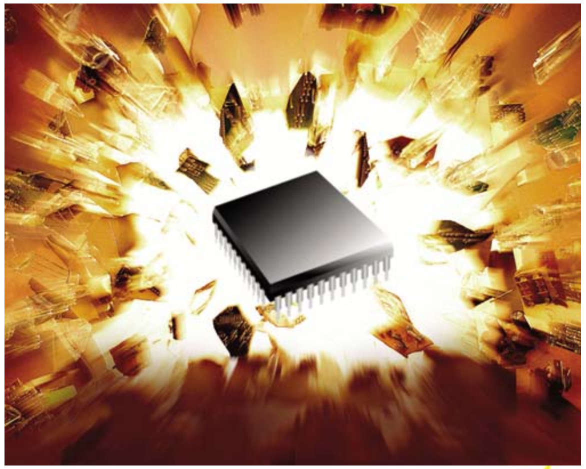
What else could you be doing with that time?



Source: Novas Customer Survey, 161 Respondents

The Barriers To Efficient Debug

- Complex Designs
  - Increasing device size and complexity
  - Barrier to required design understanding
- Complex Behavior
  - Complex design cause and effect scenarios
  - Increases conditions to be verified
- Complex multi-tool, multi-lingual, multi-team environments
  - Compounds methodology complexity



NOVAS

Novas Solves the Tough Debug Issues

Issues

Complex Environments

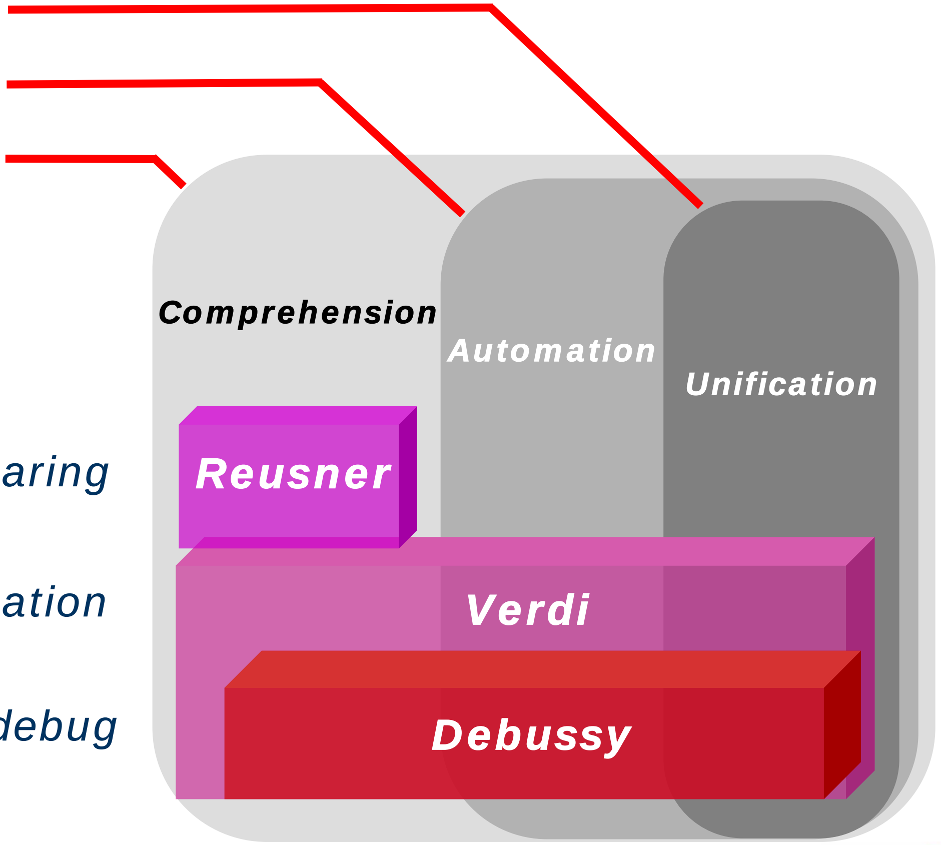
Complex Behavior

Complex Designs

Knowledge sharing

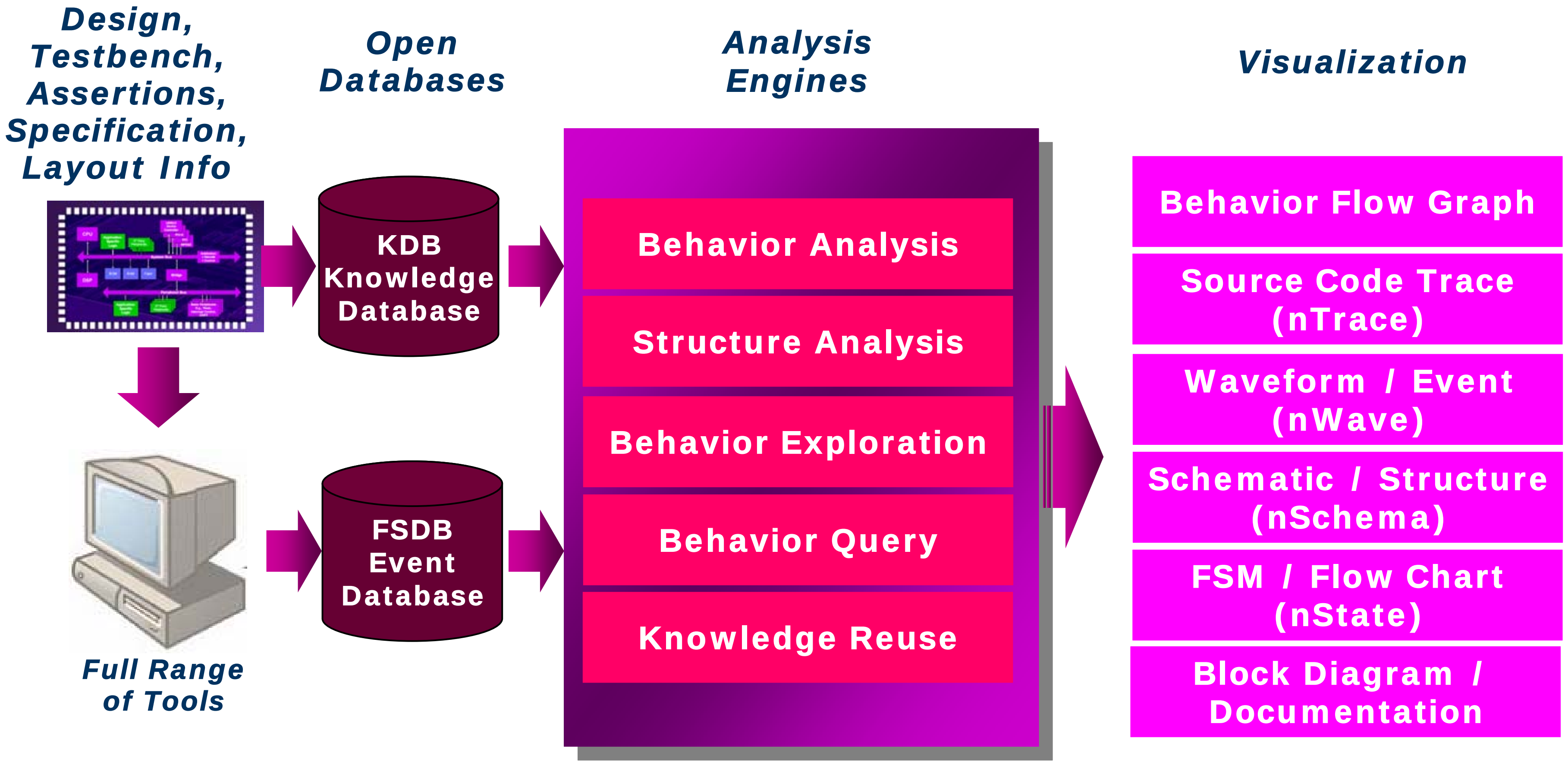
Increasing automation

Powerful comprehension and debug



NOVAS

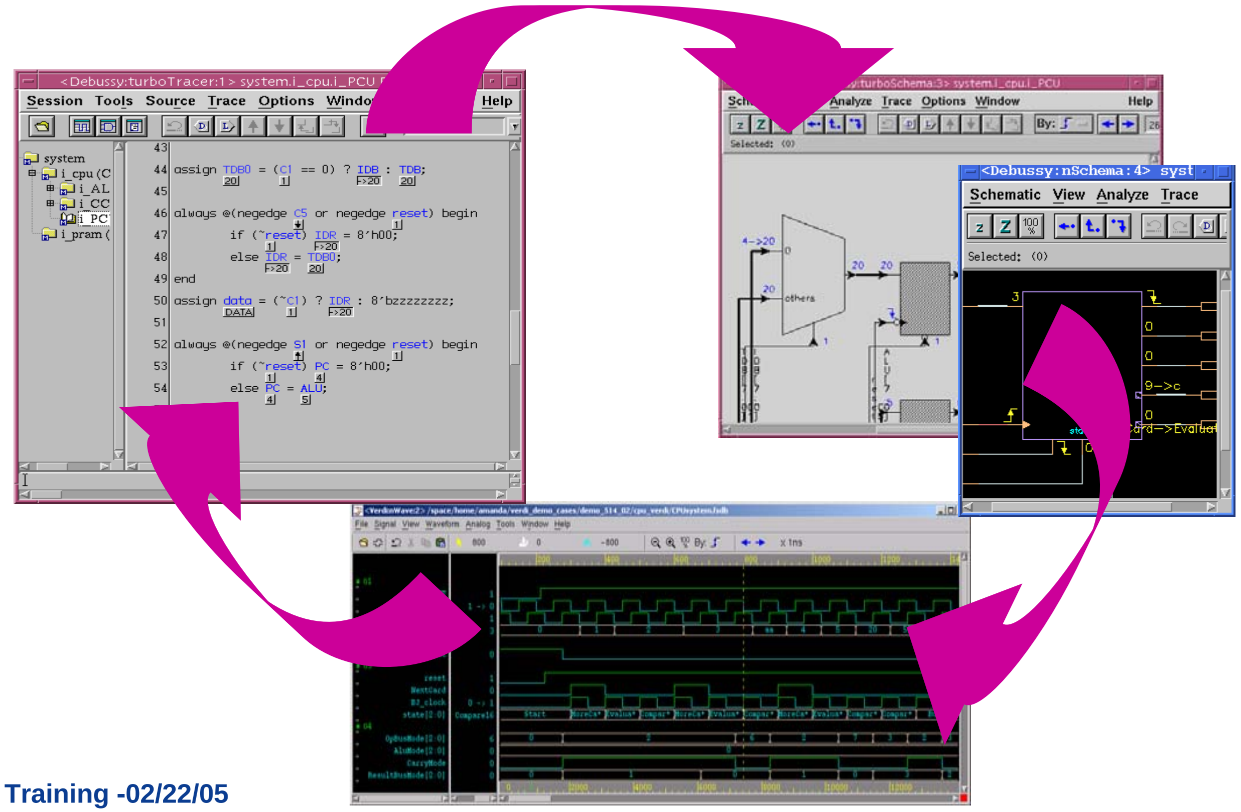
Novas Technology



Comprehensive Debug System - More Than Simple Visualization

Structural / Event Debug Process

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



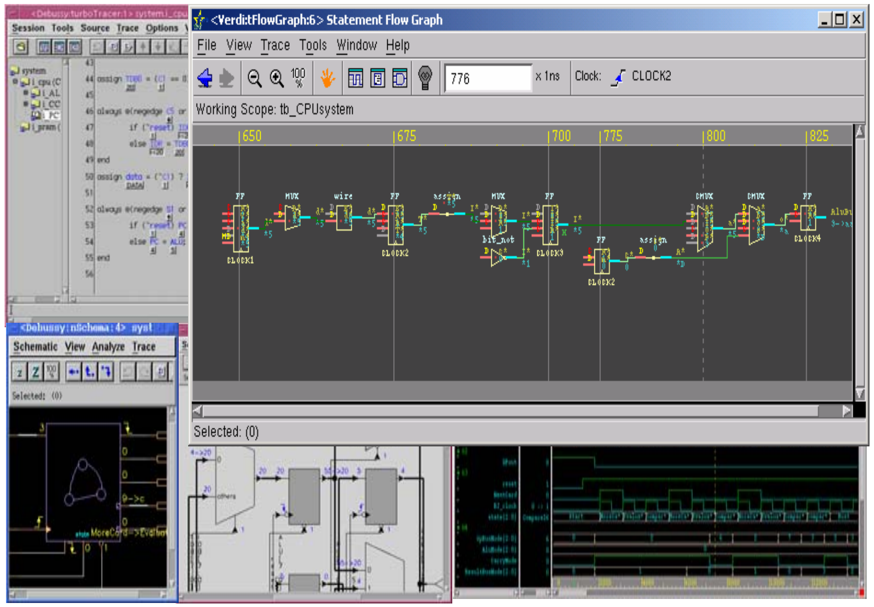
<Debussy:turboTracer:1> system.i_cpu.i_PCU
Session Tools Source Trace Options Window Help
system
i_cpu
i_AL
i_CC
i_PC
i_pram
<Debussy:turboSchema:3> system.i_cpu.i_PCU
Schematic View Analyze Trace Options Window Help
<Debussy:nSchema:4> syst
Schematic View Analyze Trace
Selected: (0)
others

```
assign TDB0 = (C1 == 0) ? IDB : TDB;

always @(negedge C5 or negedge reset) begin
        if (~reset) IDR = 8'h00;
        else IDR = TDB0;
end

assign data = (~C1) ? IDR : 8'bzzzzzzzz;

always @(negedge S1 or negedge reset) begin
        if (~reset) PC = 8'h00;
        else PC = ALU;
```

Automating Debug
Critical Productivity Enhancement

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

- Time and structure in single view
  - Immediate visual of design behavior
- Automated cause / affect tracing
  - Understand problem areas quickly
- Link back to regular views
  - Bind problem to modelling issue



<Verdi:tFlowGraph:6> Statement Flow Graph
File View Trace Tools Window Help
776
x 1ns
Clock: CLOCK2
Working Scope: tb_CPUsystem
650
675
700
775
800
825
Selected: (0)
Session Tools Source Trace Options
<Debussy:nSchema:4> syst
Schematic View Analyze Trace

EDN Magazine Innovation Award 2002 Finalist

NOVAS

Benefits

- Captures design structure and behavior.
- Generates structural diagrams of RTL or gates.
- Verification results put in context of the design.
- Helps you understand unfamiliar or legacy code.
- Automates difficult debug operations.
- Maintains consistent view of the design through design and verification process.
- Dramatically cuts your debug time over traditional methods.

Training Course

- Verdi Background and Overview
- Loading Your Design
- nTrace – View and Debug Source Code
- nSchema – View and Debug Designs Graphically
- nState – View and Analyze State Machines
- nWave – View Simulation Results in Waveforms
- Debugging with Verdi

Before you start…

- Environment Setup
- Verdi Setup File – novas.rc
- Library Mapping
- Symbol Libraries

Environment Setup

- Add binary to the search path.
  - *setenv NOVAS <Novas_install_dir>.*
  - *set path=($NOVAS/bin $path).*
- Specify search path of license file.
  - *setenv NOVAS_LICENSE_FILE <license_file>*

    or
  - *setenv LM_LICENSE_FILE <license_file>:$LM_LICENSE_FILE*

Verdi Setup File – novas.rc

- The setup file for Verdi is called ***novas.rc*** and contains:
  - Mapping information for pre-compiled designs.
  - User preferences (set via **Tools → Preferences**).
  - Reference to another novas.rc file (optional)
- Search sequence for **novas.rc** files:
  1. -rcFile <filename> command line option (read/write pointer)
  2. NOVAS_RC environment variable (read/write pointer)
  3. ./novas.rc
  4. $HOME/novas.rc
  5. <verdi install>/etc/novas.rc
- Specify path for Verdi setup file
  - *setenv NOVASRC <path>/novas.rc*
  - Use *–rcFile* on the verdi command line to overwrite the environment variable.

Logical Library Mapping Details – novas.rc

- Map a library logical name to a physical location in the novas.rc file.
  - Mapping format (in novas.rc):

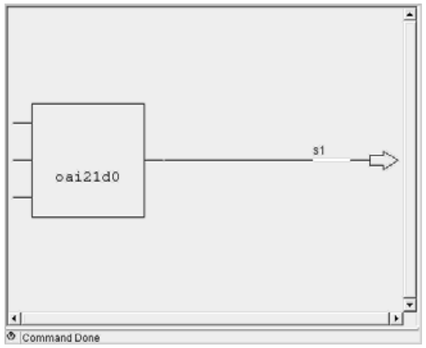
```
[Library]
logical name =  physical location
pack = ../library/pack
            Example:
            [Library]
            pack = ./work
            vital = ./work
```

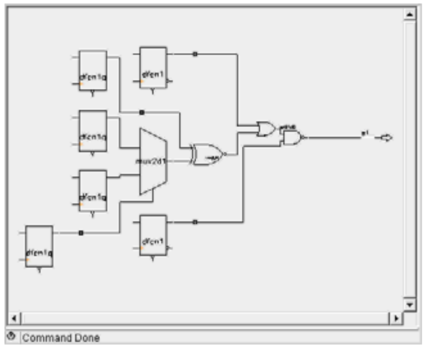
- Necessary for VHDL or mixed language designs.

Symbol Libraries

- Why are symbol libraries important?
  - Without the symbol mappings for gate cells…
    - Active fan-in cone, Fan-in cone and Fan-out cone schematics will **NOT** function properly.
    - Schematics will show **ONLY** squares instead of logic cells.
  - Symbol mappings differentiate logic gates from Sequential.
- Check the Novas web page, http://www.novas.com/ in the downloads section for a list of available libraries, Library Developer's Guide and cell definitions.

Unmapped fanin cone





Mapped fanin cone

Creating Symbol Libraries

- **Automatic** (recommend) - If the **.lib** files of Synopsys are available
  - Use the **syn2SymDB** utility to create symbol library target_lib.lib++

    syn2SymDB –o target_lib synopsys_technology.lib
- **Manual** - If the **.lib** files of Synopsys are not available
  - You have special cells to create (ASIC/FPGA macro cells)
  - Create a .map file using the Library Developer's Guide, then use the **map2SymDB** utility to create the symbol mappings library target_lib.lib++

    map2SymDB –o target_lib manual_technology.map

Specify Symbol Library and Paths

- Use common delimiters between libraries and paths.
- For NOVAS_LIBS only specify the root of the <root>.lib++ directory name.

  setenv NOVAS_LIBS “<LIB ROOT#1> <LIB ROOT#2> ...”

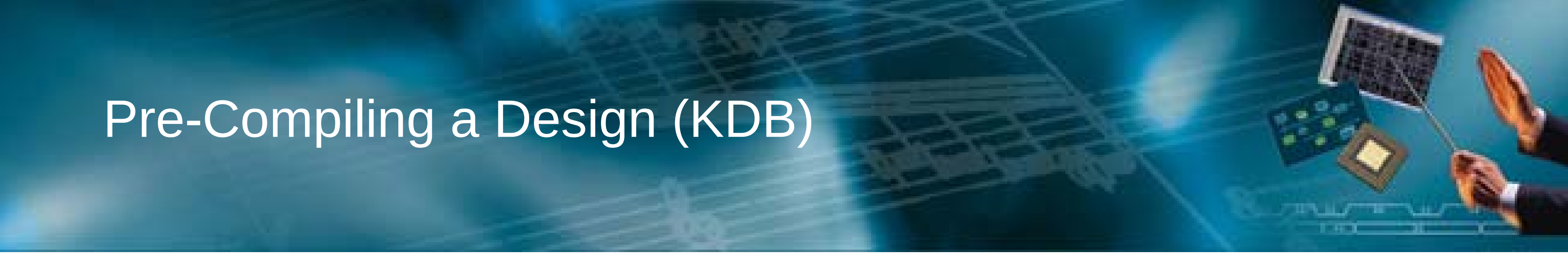
  setenv NOVAS_LIBPATHS “<Directory#1> <Directory#2> …”
- **Ex:**

  setenv NOVAS_LIBS “abc def etc”

  setenv NOVAS_LIBPATHS “/home /home/on/the/range /etc”
- Where abc, def and etc represent libraries abc.lib++, def.lib++ and etc.lib++ and they can be found in the different paths listed.
- For the performance issue, set the same process libraries to NOVAS_LIBS.

Loading Your Design

- Pre-Compiling Design
- Pre-Compiling VHDL - vhdlcom Utility
- Pre-Compiling Verilog - vericom Utility
- Verdi Command Line Options
- Load Pre-Compiled Design from Command Line
- Load Design Files from Command Line
- Load Pre-Compiled Design from the GUI
- Load Design Files from the GUI
- Using Virtual Top



Pre-Compiling a Design (KDB)

- Compile once, load many times.
- Saves load time and overall memory
- Necessary for mixed language or VHDL designs.
  - Pre-compile VHDL source files using **vhdlcom** utility.
- Optional for Verilog language designs.
  - Pre-compile Verilog source files using **vericom** utility.
- By default, a library called *work.lib*++ is created.
- Uses the **novas.rc** file logical mapping mechanism

Pre-Compiling Verilog – vericom Utility

vericom [Verdi options] [<your verilog options>]

- Use **vericom -h** to get all command line options.
- Use vericom to replace verilog simulator commands to create the Verdi mirror database optimized for debug.
  - *MTI: **vlog** top.v **–work** /home/work **–f** other_files.f*
  - *NC: **ncvlog** top.v **–WORK** /home/work **–FILE** other_files.f*
  - *VCS: **vcs** top.v **–f** other_files.f*
  - *Verdi: **vericom** top.v **–work** /home/work **–f** other_files.f.*

Pre-Compiling VHDL – vhdlcom Utility

vhdlcom [verdi options] [<your vhdl options>]

- Use **vhdlcom -h** to get all command line options.
- Verdi provides pre-compiled versions of STD, IEEE, SYNOPSYS, and NOVAS libraries which are located at: *$NOVAS/etc/kbd/vhdl/*
- Use vhdlcom to replace vhdl simulator commands to create the mirror database.
  - *MTI: **vcom –93** top.vhdl **–work** /home/work **–f** other_files.f*
  - *NC: **ncvhdl –V93** top.vhdl **–WORK** /home/work **–FILE** other_files.f*
  - *Verdi: **vhdlcom –93** top.vhdl **–work** /home/work **–f** other_files.f*
- The vhdlcom or vericom library is of the format work.lib++, pack.lib++, etc. Thus it never conflicts with simulators libraries.

Verdi Command Line Options

verdi [general options]...

- Use **verdi -h** to get all command line options.

  -f <fileName> : specify a file that lists all source files

  -vhdl | verilog: specify language type for import design from source

  -vtop : loads Virtual top file

  -play commandHistoryFile : play command history file.

  -ssc licenseFile : specify the license file name.

  -ssr sessionFile : load session file (*.ses).

  -ssf fastFile : load fast file (*.fsdb).

  -sswr restoreFile(s) : load waveform restore file(s) (*.rc).

Load Pre-Compiled Design From Command Line

verdi -lib <pre-compiled source> -top <top level>

- Reference pre-compiled source on the command line.

  verdi –lib abc –top system

- Where abc is the pre-compiled library, abc.lib++ that the design was pre-compiled into with vericom/vhdlcom.
- Where system is the top level module/architecture

Load Design Files From Command Line

verdi [verdi options] [<your verilog options>]

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

  verdi –f run.f

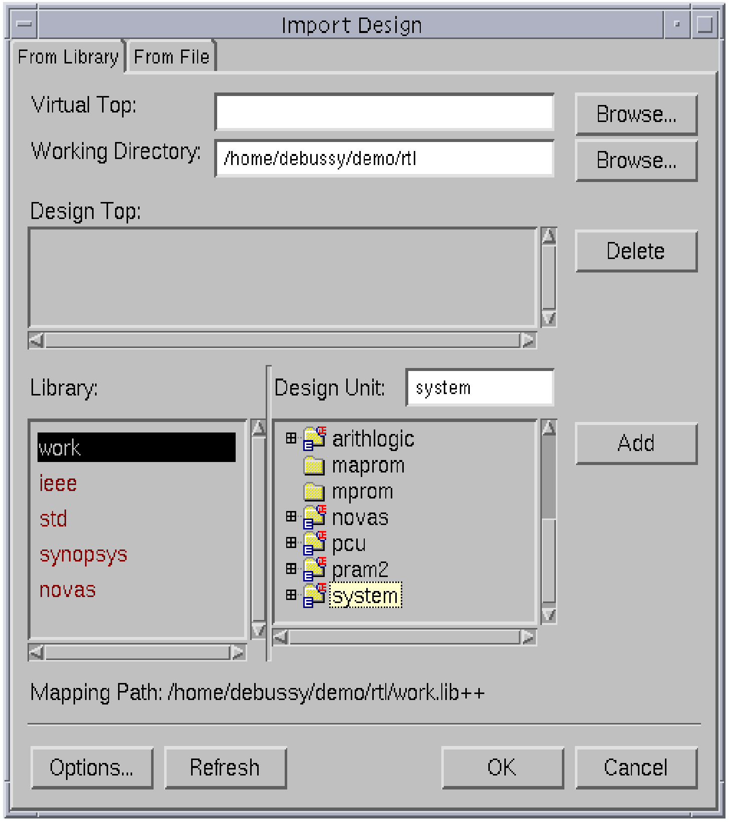
  - Where run.f is a file that contains the verilog source files and any command line switches.

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

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

Load Pre-Compiled Designs from the GUI

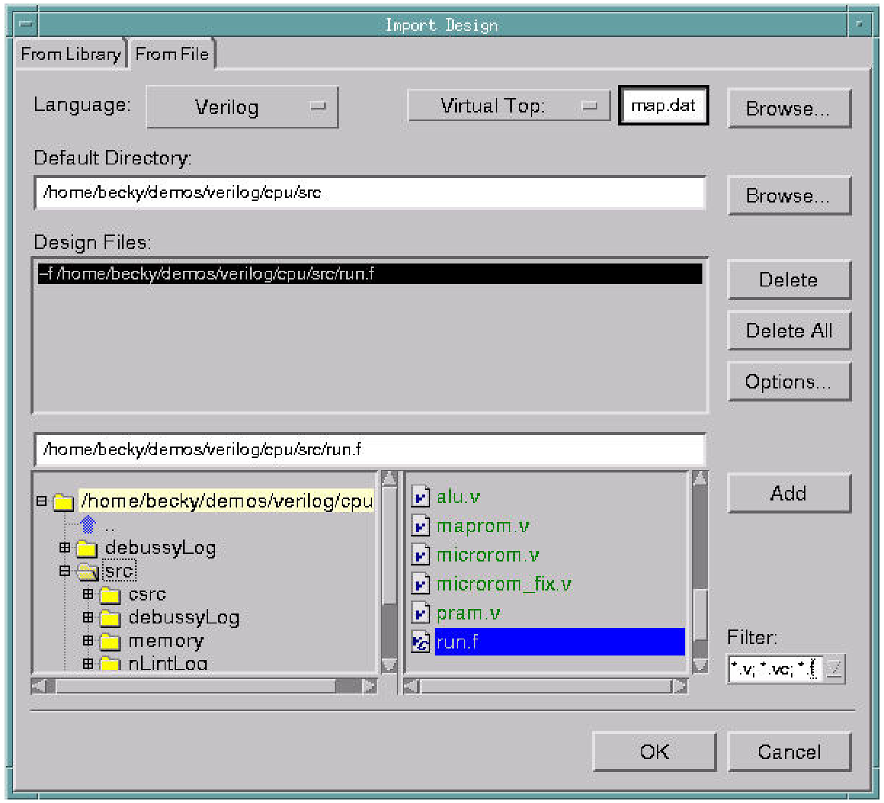
- Specify library mapping before importing design.
- Use **File → Import Design...** or **Import Design** icon to import your design.
  - Select **From Library** tab.
  - Lists all libraries in current directory or specified in map file.
  - Supports mapping file to set a Virtual Top.



Load Design Files from the GUI

- Use **File → Import Design...** or **Import Design** icon to import your design.
  - Select **From File** tab.
  - Specify language and run file or individual source files.
  - Supports mapping file to set a Virtual Top.

Note: The imported design does not get saved as a library.


Import Design

From Library | From File

Language: Verilog Virtual Top: map.dat Browse...

Default Directory: /home/becky/demos/verilog/cpu/src Browse...

Design Files:
-f /home/becky/demos/verilog/cpu/src/run.f

Delete | Delete All | Options...

/home/becky/demos/verilog/cpu/src/run.f

```
⊟ /home/becky/demos/verilog/cpu
    ..
  ⊞ debussyLog
  ⊟ src
    ⊞ csrc
    ⊞ debussyLog
    ⊞ memory
    ⊞ nLintLog
```

```
alu.v
maprom.v
microrom.v
microrom_fix.v
pram.v
run.f
```

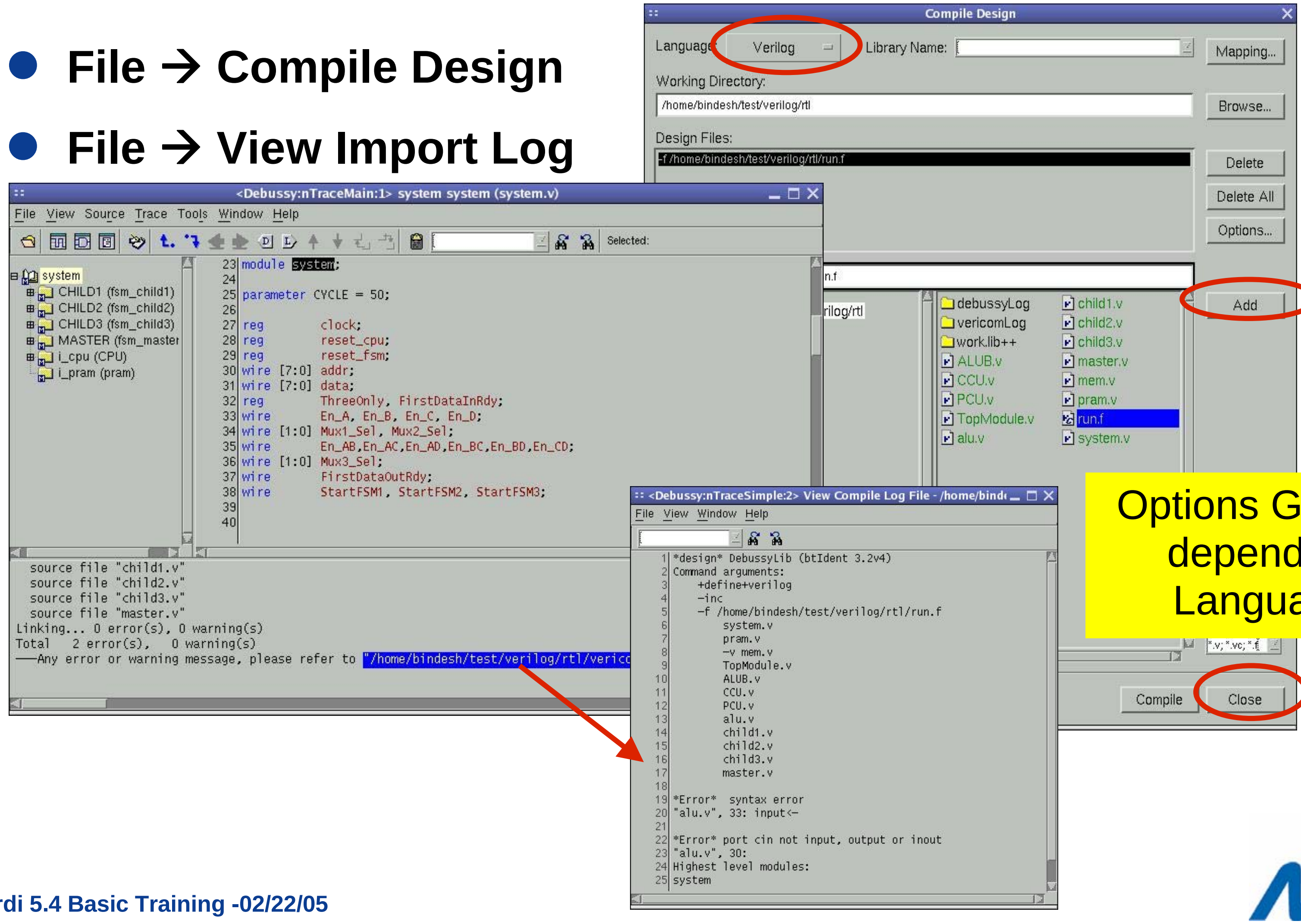
Add

Filter: *.v; *.vc; *.f

OK | Cancel

Compile Design in GUI

- **File → Compile Design**
- **File → View Import Log**



```
module system;

parameter CYCLE = 50;

reg         clock;
reg         reset_cpu;
reg         reset_fsm;
wire [7:0]  addr;
wire [7:0]  data;
reg         ThreeOnly, FirstDataInRdy;
wire        En_A, En_B, En_C, En_D;
wire [1:0]  Mux1_Sel, Mux2_Sel;
wire        En_AB,En_AC,En_AD,En_BC,En_BD,En_CD;
wire [1:0]  Mux3_Sel;
wire        FirstDataOutRdy;
wire        StartFSM1, StartFSM2, StartFSM3;
```

```
   source file "child1.v"
   source file "child2.v"
   source file "child3.v"
   source file "master.v"
Linking... 0 error(s), 0 warning(s)
Total    2 error(s),    0 warning(s)
——Any error or warning message, please refer to "/home/bindesh/test/verilog/rtl/verico
```

```
*design* DebussyLib (btIdent 3.2v4)
Command arguments:
    +define+verilog
    -inc
    -f /home/bindesh/test/verilog/rtl/run.f
        system.v
        pram.v
        -v mem.v
        TopModule.v
        ALUB.v
        CCU.v
        PCU.v
        alu.v
        child1.v
        child2.v
        child3.v
        master.v

*Error*  syntax error
"alu.v", 33: input<-

*Error* port cin not input, output or inout
"alu.v", 30:
Highest level modules:
system
```

Options GUI will depend on Language

Using Virtual Top

- Scenario:
  - Design top doesn't match simulation top (i.e. don't wish to include the test bench, test bench code not available or only looking at a portion of a design)
- Solution:
  - Create a file called map.dat that maps the top design module to the simulation hierarchy.

    Format: <*module name*> = <*full hierarchical path*>

    Example: *CPU = system.i_cpu*
  - Add *–vtop map.dat* to the Verdi command line and a pseudo hierarchy will be created.
- Why?
  - If a virtual top is not used, active annotation will not work and D&D to *nWave* will not work because hierarchical paths will not match.

Loading Your Design – Summary

- At this time you should:
  - Understand the Verdi environment setup
  - Know how to pre-compile a design
  - Know how to invoke Verdi

Training Course

- Verdi Background and Overview
- Loading Your Design
- nTrace – View and Debug Source Code
- nSchema – View and Debug Designs Graphically
- nState – View and Analyze State Machines
- nWave – View Simulation Results in Waveforms
- Debugging with Verdi

nTrace - View and Debug Source Code

- Mouse Actions
- Overview
  - Hierarchy Browser
  - Source Code Browser
  - Message Window
- Searching
  - Find Scope
  - Find Signal/Instance
  - Find String
- Trace Drivers and Loads
- Bookmarks
- Source Code Editing
- Miscellaneous
- Lab

Mouse Actions (all modules)

- **Left Mouse Button(LMB)**
  - Single-click to select design objects like signals, instances, groups, etc.
  - Hold the Shift key and click the LMB to add object to selection list.
  - Double-click(DC) to take action.
  - Drag LMB through an area to select the objects enclosed.
    - Drag LMB in schematics or waveforms will zoom in area.
- **Middle Mouse Button (MMB)**
  - Drag & Drop (D&D) across windows.
- **Right Mouse Button(RMB)**
  - Context sensitive menu.

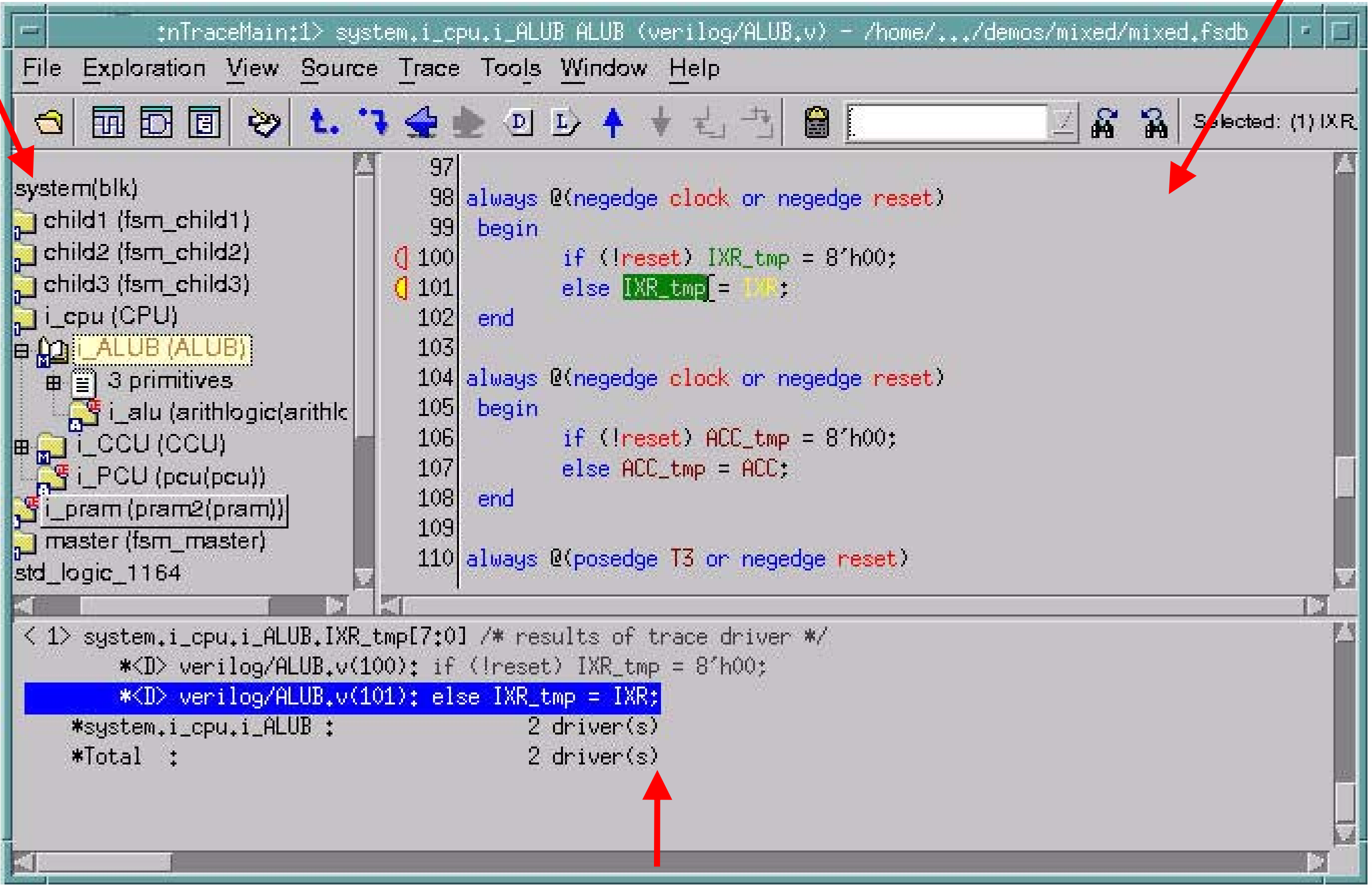
nTrace Overview

- Displays source code and hierarchy
  - Compiles the design
  - Required for *nSchema* and *nState*
  - Context sensitive tracing of drivers and loads

nTrace - Overview

Hierarchy Browser

Source Code Browser



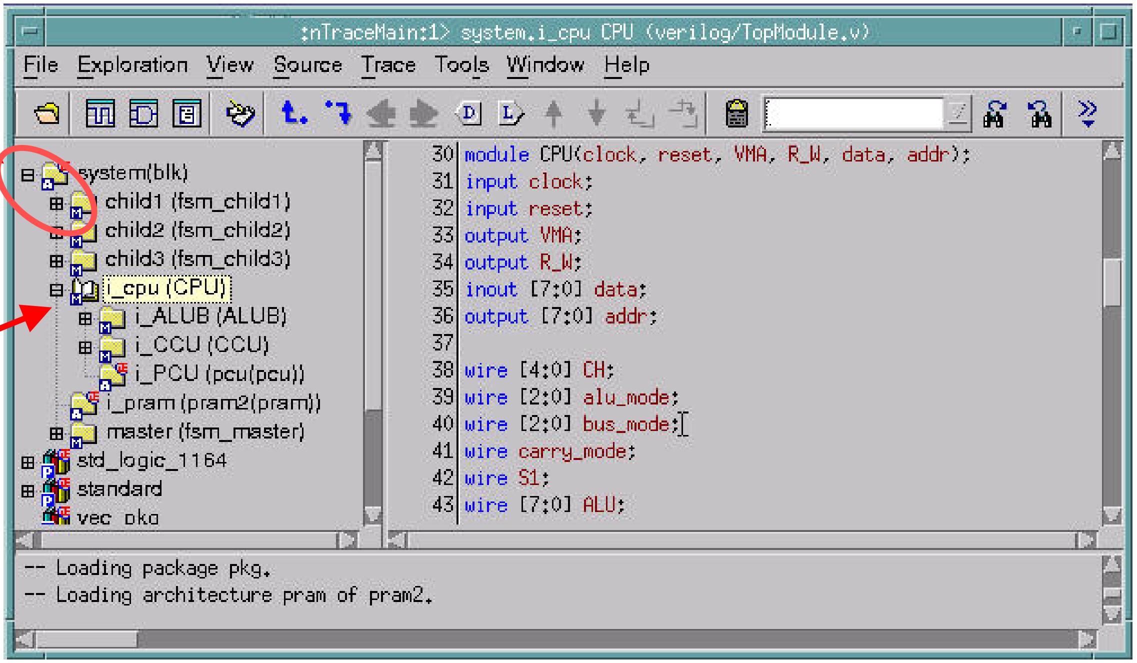
```
always @(negedge clock or negedge reset)
 begin
        if (!reset) IXR_tmp = 8'h00;
        else IXR_tmp = IXR;
 end

always @(negedge clock or negedge reset)
 begin
        if (!reset) ACC_tmp = 8'h00;
        else ACC_tmp = ACC;
 end

always @(posedge T3 or negedge reset)
```

```
< 1> system.i_cpu.i_ALUB.IXR_tmp[7:0] /* results of trace driver */
       *<D> verilog/ALUB.v(100): if (!reset) IXR_tmp = 8'h00;
       *<D> verilog/ALUB.v(101): else IXR_tmp = IXR;
   *system.i_cpu.i_ALUB :                 2 driver(s)
   *Total  :                              2 driver(s)
```

Message Window

nTrace – Hierarchy Browser

- Displays VHDL architectures and packages, Verilog modules, primitive instances, tasks and functions.
- Expand/Collapse design tree.
- Double click folder to change current scope.
- Open Folder icon indicates current scope.



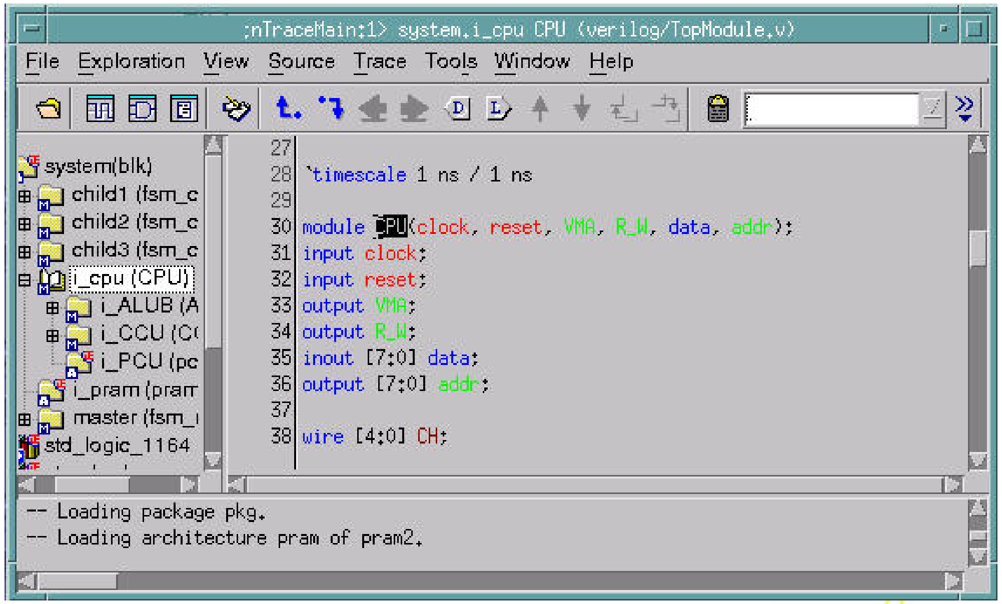
```
module CPU(clock, reset, VMA, R_W, data, addr);
input clock;
input reset;
output VMA;
output R_W;
inout [7:0] data;
output [7:0] addr;

wire [4:0] CH;
wire [2:0] alu_mode;
wire [2:0] bus_mode;
wire carry_mode;
wire S1;
wire [7:0] ALU;
```

```
-- Loading package pkg.
-- Loading architecture pram of pram2.
```

- Use **File → Print** to print out design tree.

nTrace – Source Code Browser

- Color-coded source code display.
- Double click
  - Instance name to go to architecture or module definition.
  - Module name or architecture type to return to calling instance.
  - Architecture name to go to entity definition.
  - Signal name to find the driver.
- vi compatible binding keys.

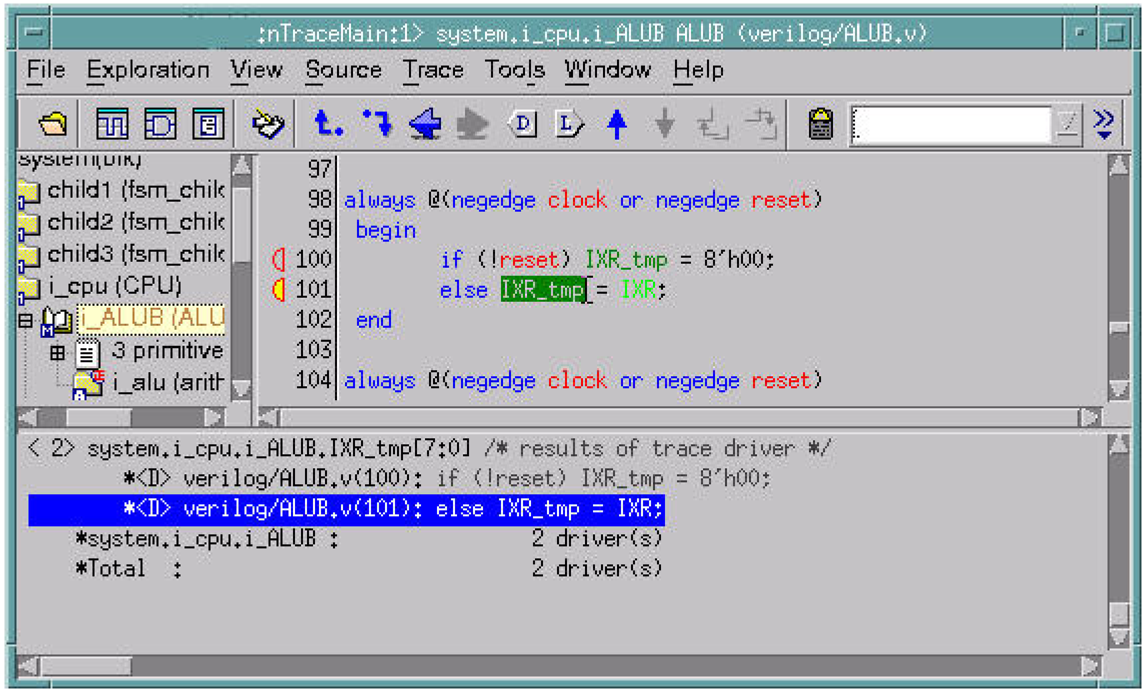


```
`timescale 1 ns / 1 ns

module CPU(clock, reset, VMA, R_W, data, addr);
input clock;
input reset;
output VMA;
output R_W;
inout [7:0] data;
output [7:0] addr;

wire [4:0] CH;
```

```
-- Loading package pkg.
-- Loading architecture pram of pram2.
```

nTrace – Message Window

- Log of debug session.
- Double click on line to go to that location in source code.



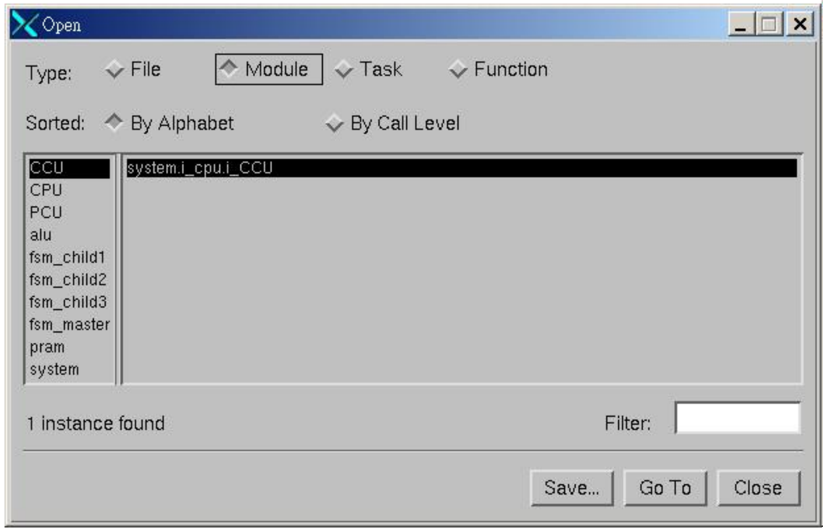
:nTraceMain:1> system.i_cpu.i_ALUB ALUB (verilog/ALUB.v)
File Exploration View Source Trace Tools Window Help
system(blk)
child1 (fsm_chil
child2 (fsm_chil
child3 (fsm_chil
i_cpu (CPU)
i_ALUB (ALU
3 primitive
i_alu (arit

```
97
98  always @(negedge clock or negedge reset)
99   begin
100         if (!reset) IXR_tmp = 8'h00;
101         else IXR_tmp = IXR;
102  end
103
104 always @(negedge clock or negedge reset)
```

```
< 2> system.i_cpu.i_ALUB.IXR_tmp[7:0] /* results of trace driver */
        *<D> verilog/ALUB.v(100): if (!reset) IXR_tmp = 8'h00;
        *<D> verilog/ALUB.v(101): else IXR_tmp = IXR;
    *system.i_cpu.i_ALUB :                   2 driver(s)
    *Total   :                               2 driver(s)
```

nTrace – Find Scope

- **Source → Find Scope...** is a fast way to directly open a Verilog module or VHDL architecture if the design tree is deep.
- Choose by type: File, Module, Task or Function.
- Enter search filter using wildcards.



Open

Type: File [✓ Module] Task Function

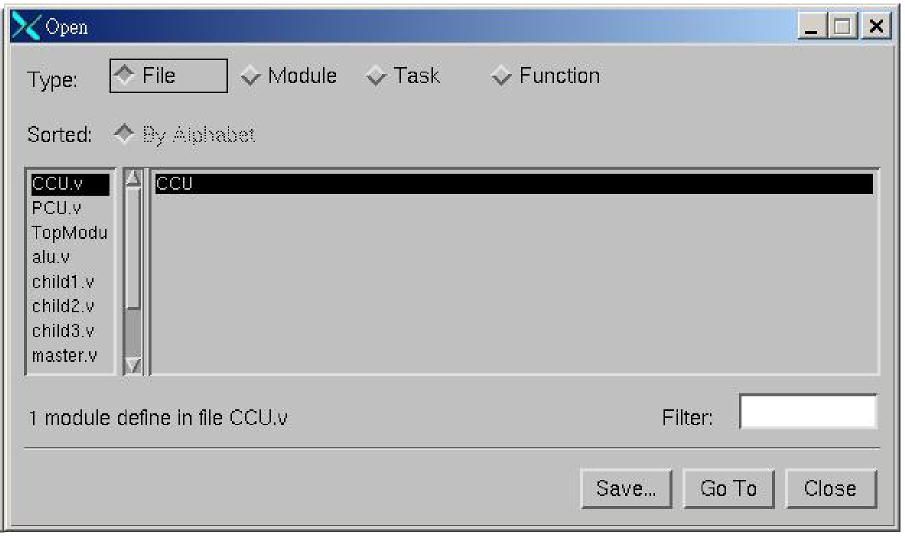
Sorted: [✓ By Alphabet] By Call Level

```
CCU          system.i_cpu.i_CCU
CPU
PCU
alu
fsm_child1
fsm_child2
fsm_child3
fsm_master
pram
system
```

1 instance found

Filter:

Open by Module



Open

Type: [✓ File] Module Task Function

Sorted: By Alphabet

```
CCU.v        CCU
PCU.v
TopModu
alu.v
child1.v
child2.v
child3.v
master.v
```

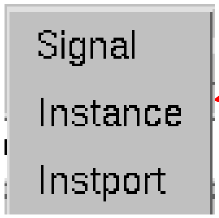
1 module define in file CCU.v

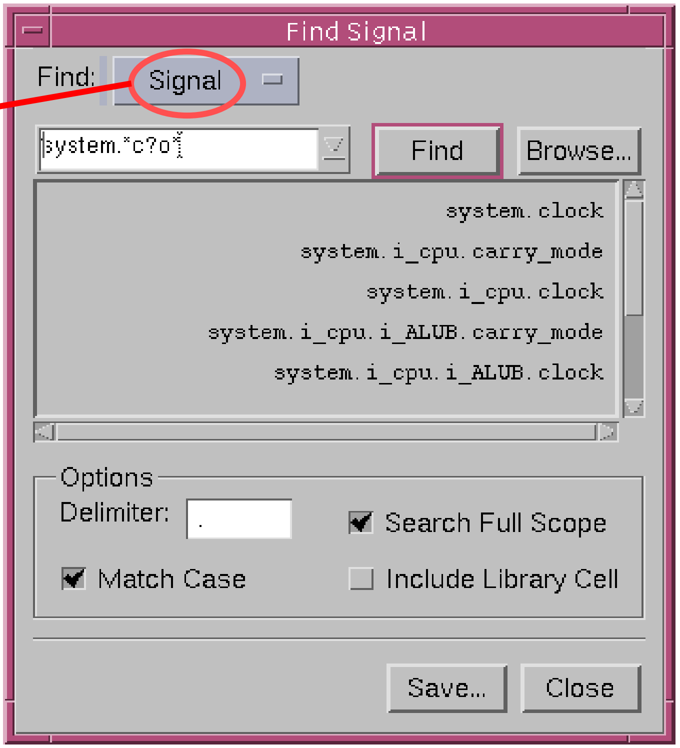
Filter:

Open by File

nTrace – Find Signals and Instances

- **Source → Find Signal...** is a fast way to find signals/instances/Instport in full design or selected scope.
- Supports wildcard characters and regular expressions.





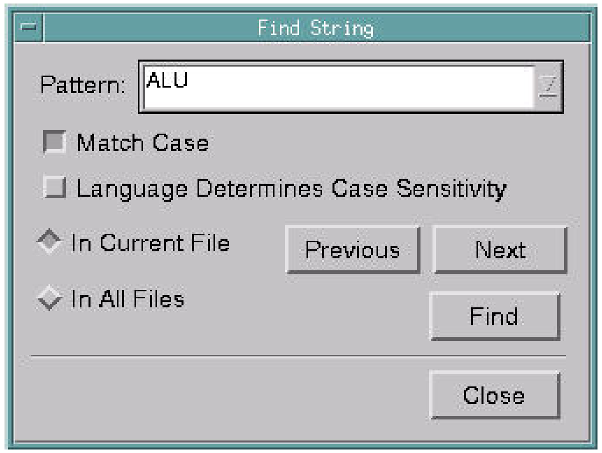
```
system.*c?o*
```

```
system.clock
system.i_cpu.carry_mode
system.i_cpu.clock
system.i_cpu.i_ALUB.carry_mode
system.i_cpu.i_ALUB.clock
```

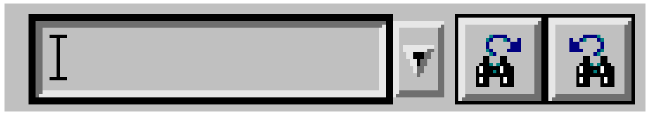
Options
Delimiter: .
[✓] Search Full Scope
[✓] Match Case
[ ] Include Library Cell

nTrace – Find Strings

- **Source → Find String…** quickly locates a string in current / all files.



- **Find String** on toolbar quickly locates a string in current scope.
  - Enable/disable case matching under **Tools → Preferences**, **Source Code** tab, **Miscellaneous** tab.



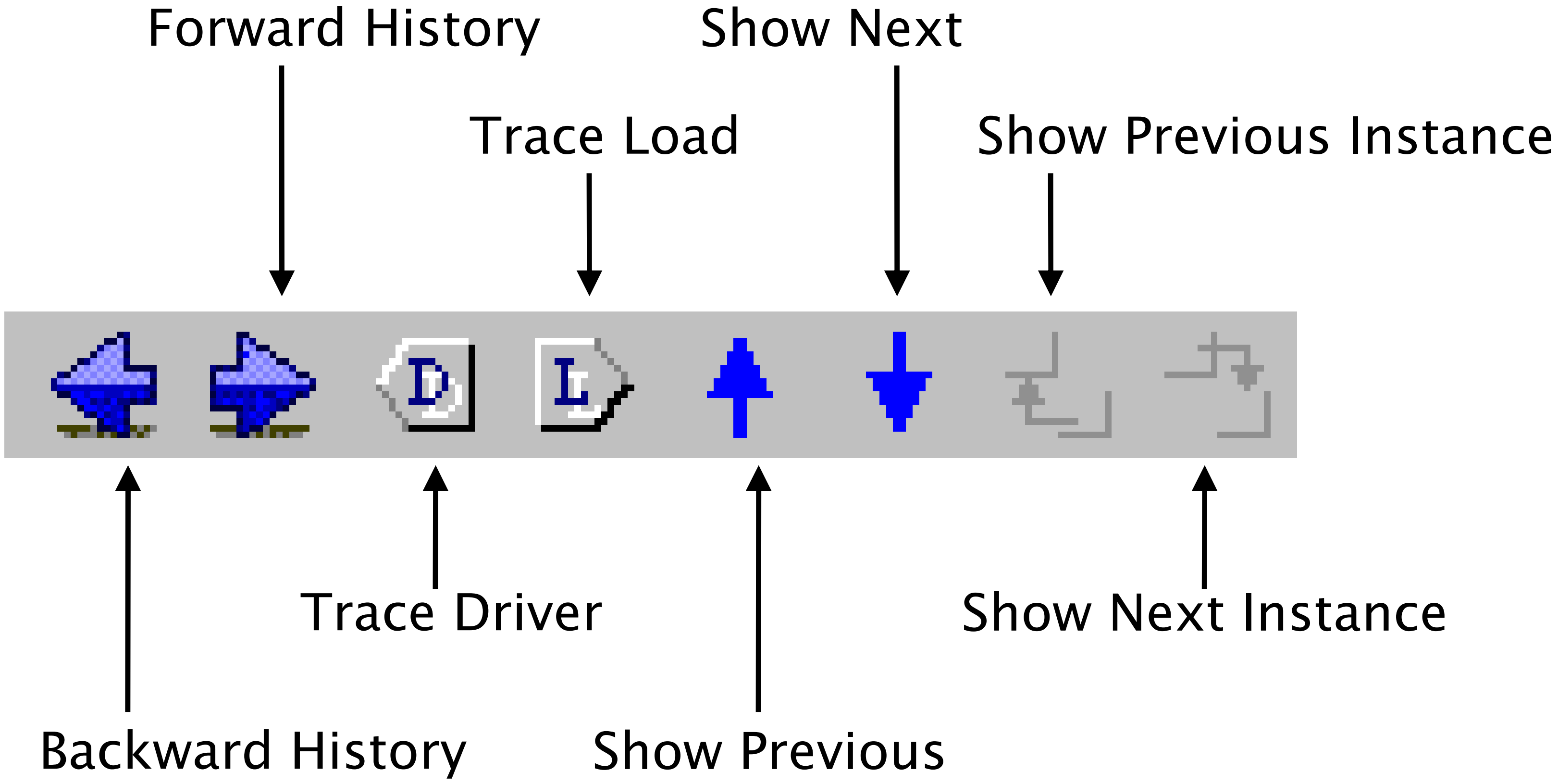
- Use **Source → Go To** command to jump to specified line or First Executable Line.

nTrace – Trace Drivers and Loads

- To quickly find all the ***drivers*** of a signal
  - Double click a signal.
  - Select signal and click on *Trace Driver* icon.
  - **Trace → Driver** or Right Mouse Button menu.
- To quickly find all the ***loads*** of a signal
  - Select signal and click on *Trace Load* icon.
  - **Trace → Load** or Right Mouse Button menu.
- To quickly find all the ***drivers and loads*** of a signal
  - **Trace → Connectivity** or Right Mouse Button menu.
  - Dropping a signal in the source window will trace the connectivity of the signal.
- By default, tracing will cross hierarchical boundaries.
  - Use **Trace → Trace Cross Hierarchy** to keep within a hierarchy.

NOVAS

nTrace – Trace Driver/Load Tool Bar Icons

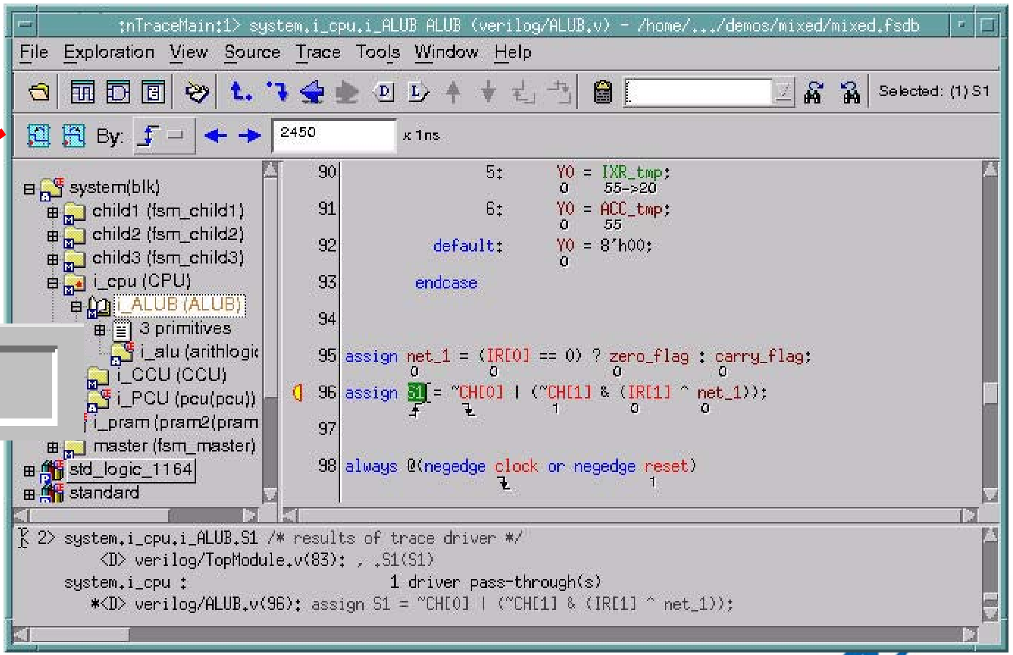


nTrace – Search, View and Debug Source Code

- Searching
  - Find Scope
    - **Source → Find Scope**
  - Find Signal/Instance
    - **Source → Find Signal**
  - Find String
    - **Source → Find String**
- Bookmarks
    - **Source → Bookmark**
- Multiple Source Code Windows
- Editing Source Code
    - **Tools → Preferences...** to select your preferred editor.
    - **Source → Edit Source File**

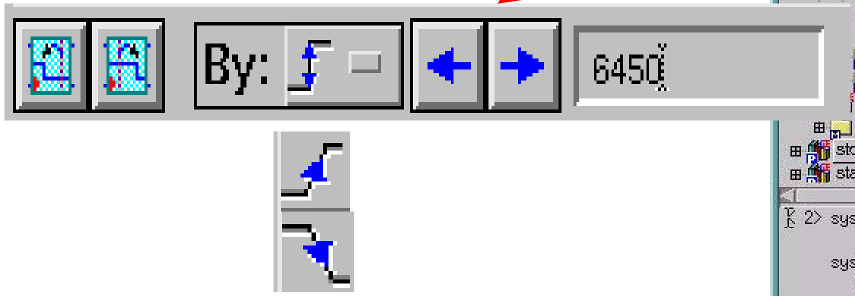
nTrace – Active Annotation

- Display simulation results on source.
  - **Source → Active Annotation** in *nTrace*
- Signal values/transitions are synchronized with cursor time.
- Select signal(s) and go to next or previous transitions.
  - *Any Change*, *Rising* or *Falling*



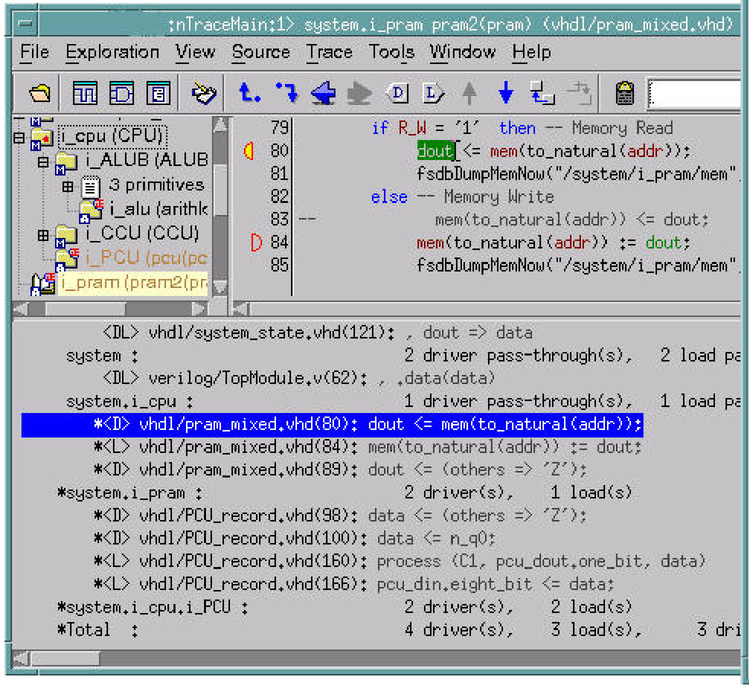
```
            5:       Y0 = IXR_tmp;
                     0    55->20
            6:       Y0 = ACC_tmp;
                     0    55
        default:     Y0 = 8'h00;
                     0
      endcase


assign net_1 = (IR[0] == 0) ? zero_flag : carry_flag;
         0       0               0           0
assign S1 = ~CH[0] | (~CH[1] & (IR[1] ^ net_1));
                            1        0       0

always @(negedge clock or negedge reset)
                                      1
```

```
2> system.i_cpu.i_ALUB.S1 /* results of trace driver */
      <D> verilog/TopModule.v(83): , .S1(S1)
   system.i_cpu :                    1 driver pass-through(s)
     *<D> verilog/ALUB.v(96): assign S1 = ~CH[0] | (~CH[1] & (IR[1] ^ net_1));
```



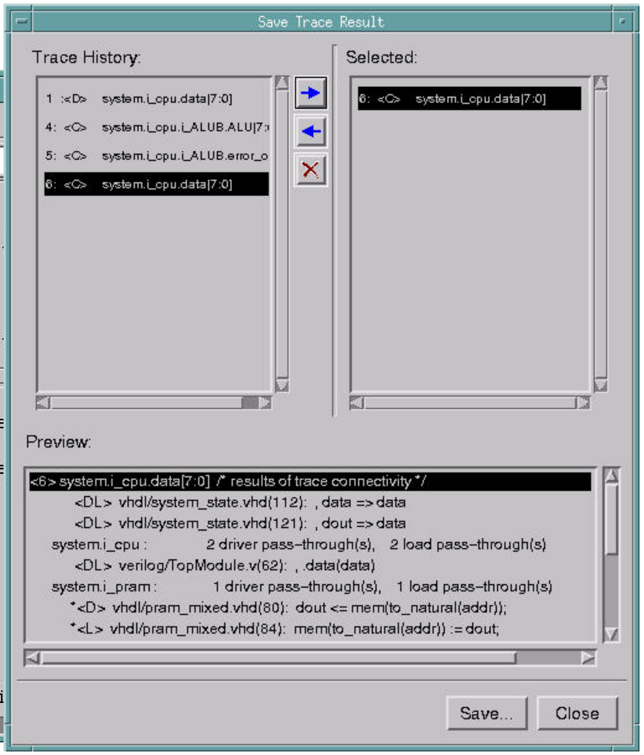
nTrace – Trace Results and Report

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


:nTraceMain:1> system.i_pram pram2(pram) (vhdl/pram_mixed.vhd)
File Exploration View Source Trace Tools Window Help
i_cpu (CPU)
i_ALUB (ALUB
3 primitives
i_alu (arithk
i_CCU (CCU)
i_PCU (pcu(pc
i_pram (pram2(pr

```
79           if R_W = '1'  then -- Memory Read
80              dout <= mem(to_natural(addr));
81              fsdbDumpMemNow("/system/i_pram/mem"
82           else -- Memory Write
83 --             mem(to_natural(addr)) <= dout;
84              mem(to_natural(addr)) := dout;
85              fsdbDumpMemNow("/system/i_pram/mem"
```

```
      <DL> vhdl/system_state.vhd(121): , dout => data
   system :                        2 driver pass-through(s),   2 load pa
      <DL> verilog/TopModule.v(62): , .data(data)
   system.i_cpu :                  1 driver pass-through(s),   1 load pa
      *<D> vhdl/pram_mixed.vhd(80): dout <= mem(to_natural(addr));
      *<L> vhdl/pram_mixed.vhd(84): mem(to_natural(addr)) := dout;
      *<D> vhdl/pram_mixed.vhd(89): dout <= (others => 'Z');
  *system.i_pram :                 2 driver(s),    1 load(s)
      *<D> vhdl/PCU_record.vhd(98): data <= (others => 'Z');
      *<D> vhdl/PCU_record.vhd(100): data <= n_q0;
      *<L> vhdl/PCU_record.vhd(160): process (C1, pcu_dout.one_bit, data)
      *<L> vhdl/PCU_record.vhd(166): pcu_din.eight_bit <= data;
  *system.i_cpu.i_PCU :            2 driver(s),    2 load(s)
  *Total  :                        4 driver(s),    3 load(s),      3 dri
```



Trace History:

```
1 :<D>   system.i_cpu.data[7:0]
4: <C>   system.i_cpu.i_ALUB.ALU[7:
5: <C>   system.i_cpu.i_ALUB.error_o
6: <C>   system.i_cpu.data[7:0]
```

Selected:

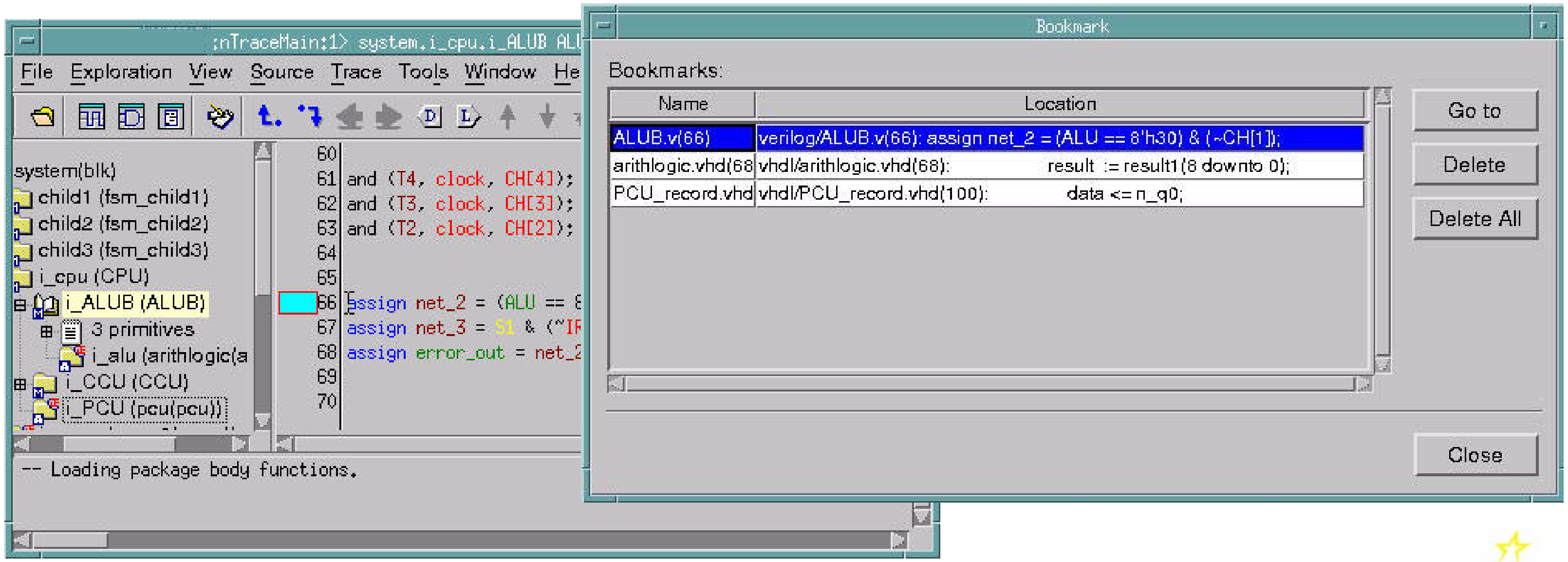
```
6: <C>   system.i_cpu.data[7:0]
```

Preview:

```
<6> system.i_cpu.data[7:0] /* results of trace connectivity */
      <DL> vhdl/system_state.vhd(112): , data => data
      <DL> vhdl/system_state.vhd(121): , dout => data
   system.i_cpu :          2 driver pass-through(s),   2 load pass-through(s)
      <DL> verilog/TopModule.v(62): , .data(data)
   system.i_pram :          1 driver pass-through(s),   1 load pass-through(s)
     *<D> vhdl/pram_mixed.vhd(80): dout <= mem(to_natural(addr));
     *<L> vhdl/pram_mixed.vhd(84): mem(to_natural(addr)) := dout;
```

nTrace – Bookmarks

- Click on **Bookmark** icon on toolbar or use Ctrl-F2 to toggle a bookmark on/off at the current line.
- Use **Source → Bookmark** command to get list of current bookmarks.
- Supports unlimited bookmarks.



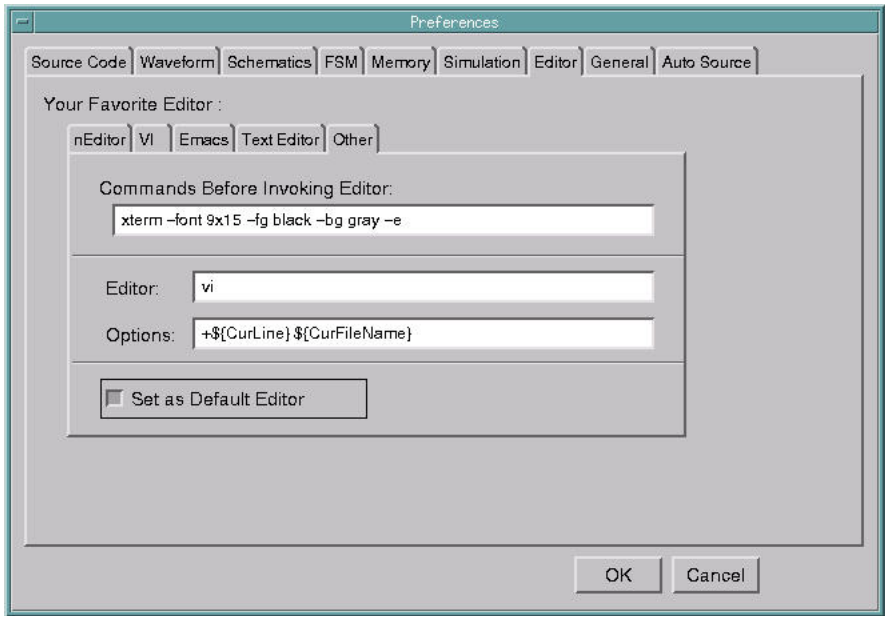
```
60
61 and (T4, clock, CH[4]);
62 and (T3, clock, CH[3]);
63 and (T2, clock, CH[2]);
64
65
66 assign net_2 = (ALU == 8
67 assign net_3 = S1 & (~IR
68 assign error_out = net_2
69
70
```

```
-- Loading package body functions.
```

Name	Location
ALUB.v(66)	verilog/ALUB.v(66): assign net_2 = (ALU == 8'h30) & (~CH[1]);
arithlogic.vhd(68	vhdl/arithlogic.vhd(68): result := result1(8 downto 0);
PCU_record.vhd	vhdl/PCU_record.vhd(100): data <= n_q0;

nTrace – Source Code Editing

- Use **Source → Edit Source File** command or click icon to edit your current module.
- Use **Tools → Preferences...** to select your preferred editor.


Preferences
Source Code | Waveform | Schematics | FSM | Memory | Simulation | Editor | General | Auto Source
nEditor | VI | Emacs | Text Editor | Other
OK
Cancel

Your Favorite Editor :

Commands Before Invoking Editor: xterm –font 9x15 –fg black –bg gray –e

Editor: vi

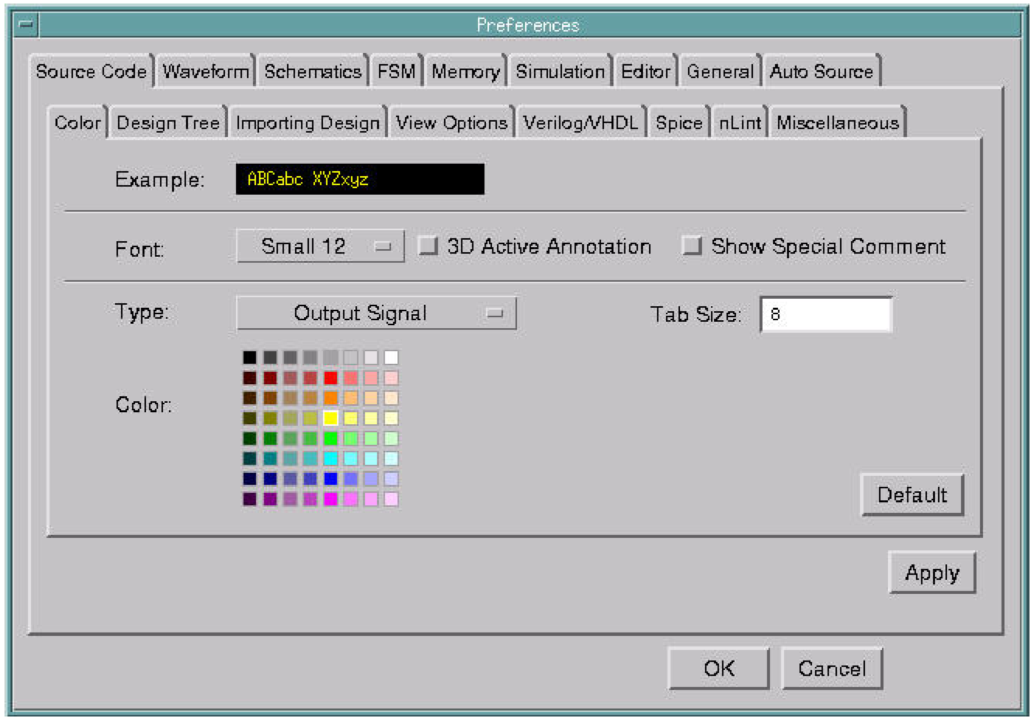
Options: +${CurLine} ${CurFileName}

☐ Set as Default Editor

- Use **File → Reload Design** command to update the changes.

nTrace – Miscellaneous

- Use **Tools → Preferences...** to customize the appearance and fonts of the hierarchy browser and source code browser.



Preferences

Source Code | Waveform | Schematics | FSM | Memory | Simulation | Editor | General | Auto Source

Color | Design Tree | Importing Design | View Options | Verilog/VHDL | Spice | nLint | Miscellaneous

Example: ABCabc XYZxyz

Font: Small 12 | ☐ 3D Active Annotation | ☐ Show Special Comment

Type: Output Signal | Tab Size: 8

Color:

Default

Apply

OK | Cancel

- Use **Help → Command Reference** for additional information.

nTrace – Wrap-up

- Questions?

- Take 20 minutes to complete the *nTrace* LAB

Training Course

- Verdi Background and Overview
- Loading Your Design
- nTrace – View and Debug Source Code
- nSchema – View and Debug Designs Graphically
- nState – View and Analyze State Machines
- nWave – View Simulation Results in Waveforms
- Debugging with Verdi

nSchema – View and Debug Designs Graphically

- Overview
- Invoke *nSchema*
- Resultant Schematics
- Display Source Code
- Traverse Hierarchy
- Searching Commands
- Display Options
- Schematic Window Types
  - Full Hierarchical
  - Browser
  - Flatten
- View Detail RTL
- Clock/Reset Tree
- Miscellaneous
- Lab

nSchema Overview

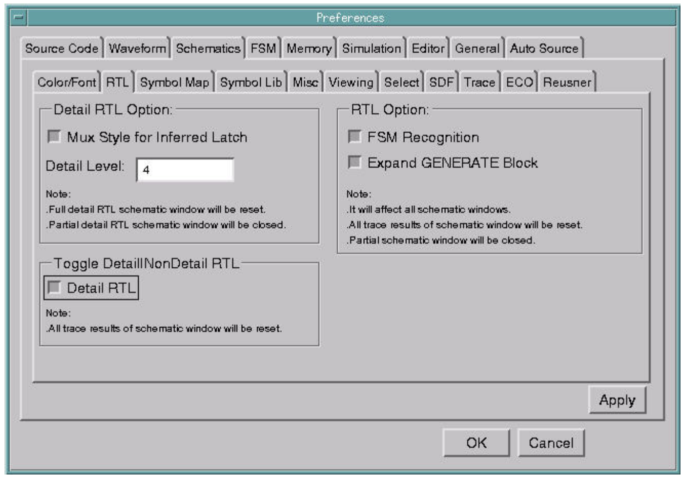
- Automatically generates schematic of design
  - Gate Level, RTL, behavioral Level
  - Hierarchical or flat
  - Fan-in or fan-out cones
  - Highlight logic between registers

Invoke nSchema

- The “**New Schematic**” icon
  - Select an instance in Hierarchy Browser and click on **New Schematic** icon.
  - Drag & Drop an instance from the hierarchy browser or a Verilog module or VHDL architecture from the Source Code Browser to **New Schematic** icon.
- Execute **Tools → New Schematic → Current Scope** in *nTrace*.

nSchema – Resultant Schematics

- Storage elements need to be recognized for complete tracing ability.
- Gate level schematic displays cell level netlist.(Symbol libraries required)
- RTL logic diagram displays synthesized view.
  - Signal type -- clock, reset, set, flip-flop / latch output or tri-state output
  - Block type -- latch, flip-flop or combinational logic
  - Enable/Disable **Detail RTL** option under **Tools → Preferences**, **Schematics** tab, **RTL** tab.
- State machine recognition as symbolic or logical views.
  - Enable/Disable **FSM Recognition** option under **Tools → Preferences**, **Schematics** tab, **RTL** tab.



Preferences
Source Code | Waveform | Schematics | FSM | Memory | Simulation | Editor | General | Auto Source
Color/Font | RTL | Symbol Map | Symbol Lib | Misc | Viewing | Select | SDF | Trace | ECO | Reusner
Apply
OK
Cancel

Detail RTL Option:

Mux Style for Inferred Latch

Detail Level: 4

Note:
.Full detail RTL schematic window will be reset.
.Partial detail RTL schematic window will be closed.

RTL Option:

FSM Recognition

Expand GENERATE Block

Note:
.It will affect all schematic windows.
.All trace results of schematic window will be reset.
.Partial schematic window will be closed.

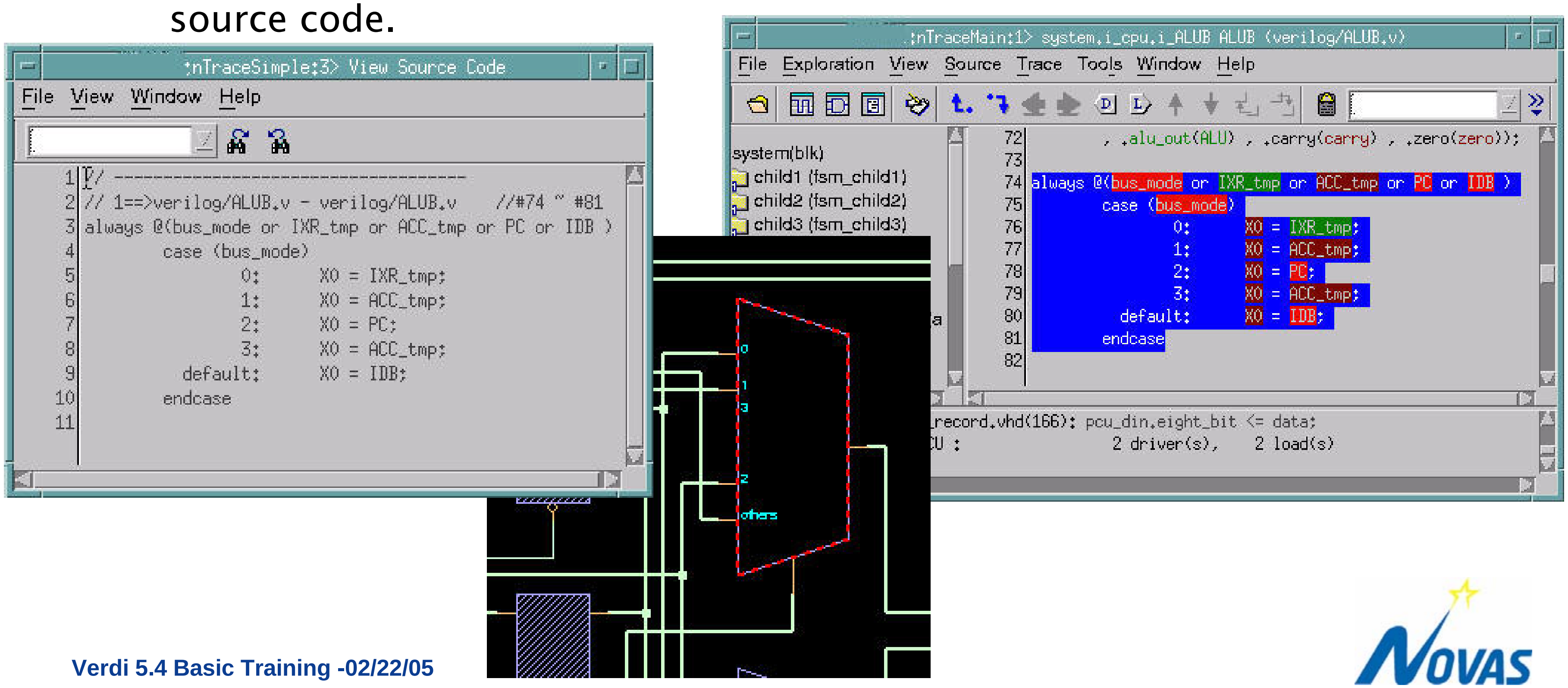
Toggle Detail/NonDetail RTL

Detail RTL

Note:
.All trace results of schematic window will be reset.

nSchema – Display Source Code

- To quickly understand what RTL construct created a logic cell
  - Double-click on the lowest level RTL to opens *View Source Code* box of the corresponding RTL code.
  - Drag & Drop a schematic object to *nTrace* displays the corresponding source code.


:nTraceSimple:3> View Source Code
File View Window Help
:nTraceMain:1> system.i_cpu.i_ALUB ALUB (verilog/ALUB.v)
File Exploration View Source Trace Tools Window Help
system(blk)
child1 (fsm_child1)
child2 (fsm_child2)
child3 (fsm_child3)
0
1
3
2
others

```
// ------------------------------------
// 1==>verilog/ALUB.v - verilog/ALUB.v     //#74 ~ #81
always @(bus_mode or IXR_tmp or ACC_tmp or PC or IDB )
        case (bus_mode)
                0:      X0 = IXR_tmp;
                1:      X0 = ACC_tmp;
                2:      X0 = PC;
                3:      X0 = ACC_tmp;
          default:      X0 = IDB;
        endcase
```

```
       , .alu_out(ALU) , .carry(carry) , .zero(zero));

always @(bus_mode or IXR_tmp or ACC_tmp or PC or IDB )
       case (bus_mode)
                0:      X0 = IXR_tmp;
                1:      X0 = ACC_tmp;
                2:      X0 = PC;
                3:      X0 = ACC_tmp;
         default:       X0 = IDB;
       endcase
```

```
_record.vhd(166): pcu_din.eight_bit <= data;
CU :                    2 driver(s),    2 load(s)
```

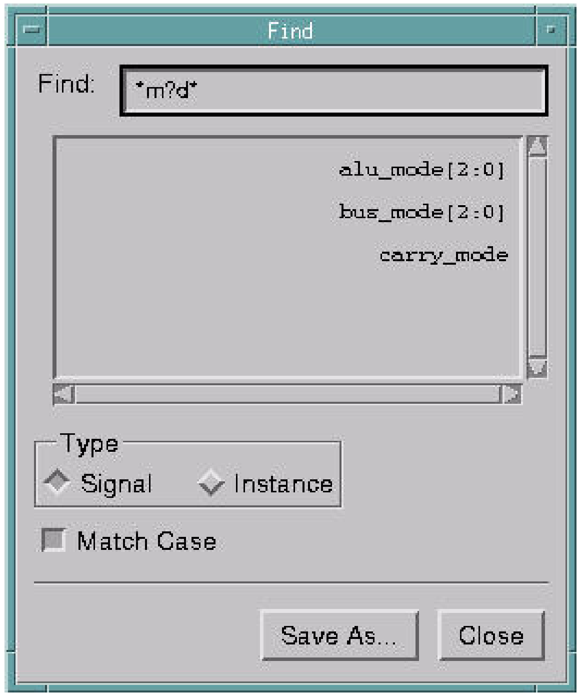
nSchema – Traverse Hierarchy

- Push Into a Module
  - Double click on instance or instance port.
  - Select an instance and click the **Push View In** icon
  - Invoke right mouse button menu and select **Push View In** command.
  - Pull down **View → Push View In**.
- Pop View Up
  - Use the **Pop View Up** icon
  - Double-click on a schematic port.
  - Invoke right mouse button menu and select **Pop View Up** command.
  - Pull down **View → Pop View Up** or **View → Pop View Up from Port**.
- Click the **Last View** icon to go to previous view.

nSchema – Searching Commands

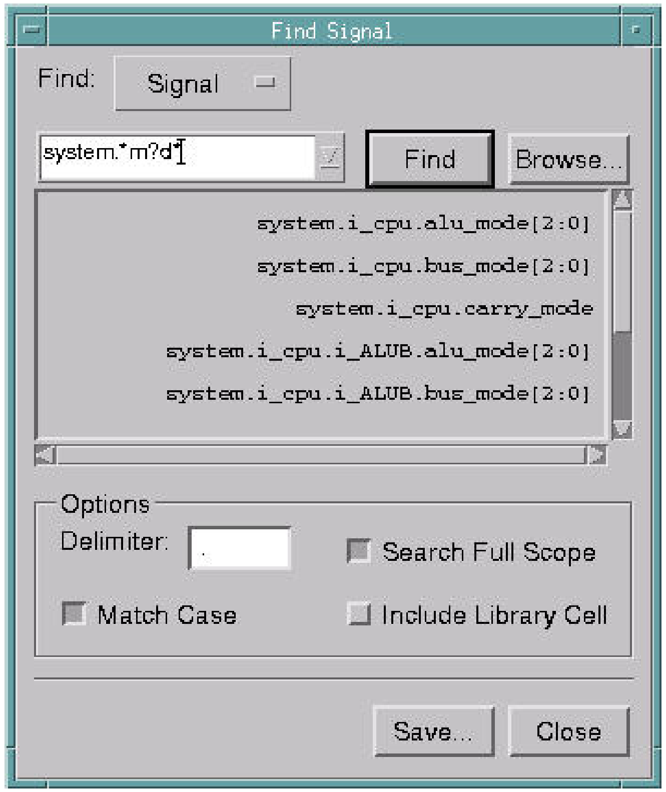
- **Schematic → Find Signal/Instance...**

  finds signals or instances in full design.
- **Schematic → Find In Current Scope...**
  finds signals or instances in current scope.



Find: *m?d*

```
alu_mode[2:0]
bus_mode[2:0]
carry_mode
```



Find: Signal

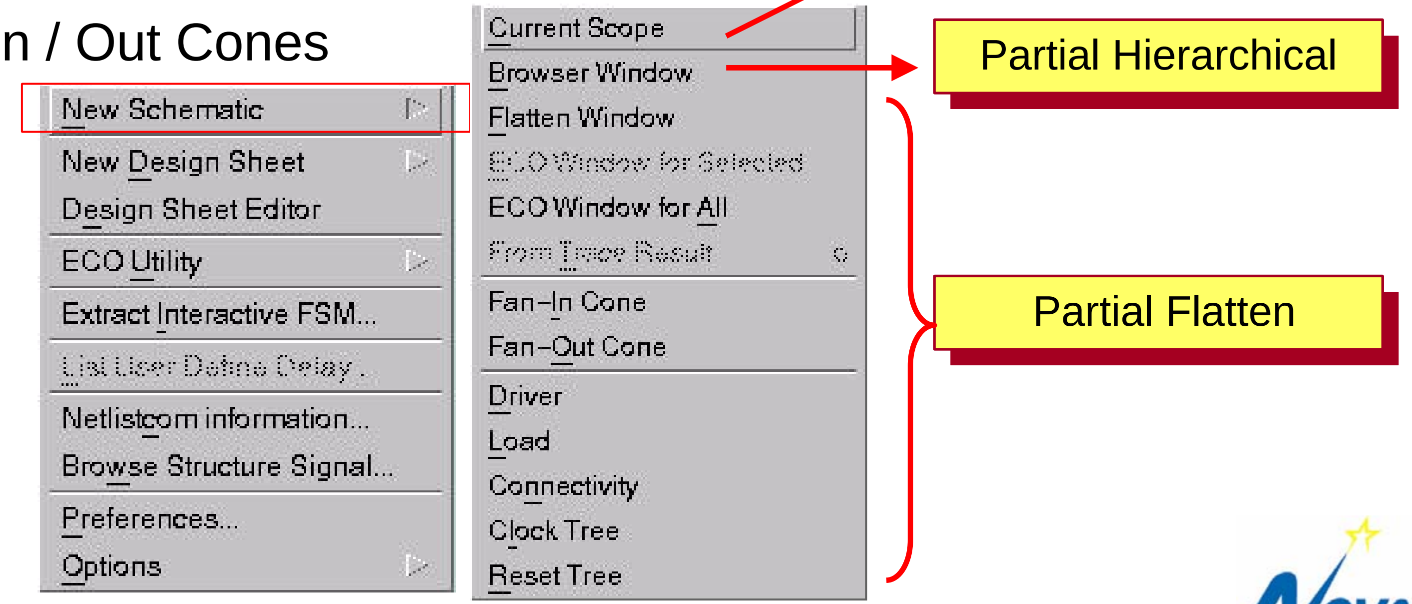
system.*m?d*

Delimiter: .

```
system.i_cpu.alu_mode[2:0]
system.i_cpu.bus_mode[2:0]
system.i_cpu.carry_mode
system.i_cpu.i_ALUB.alu_mode[2:0]
system.i_cpu.i_ALUB.bus_mode[2:0]
```

nSchema – Schematic Window Types

- Full Hierarchical Window – default type
- Partial Hierarchical Window
  - Browser Window
- Partial Flatten Window
  - Flatten Window
  - Fan-In / Out Cones

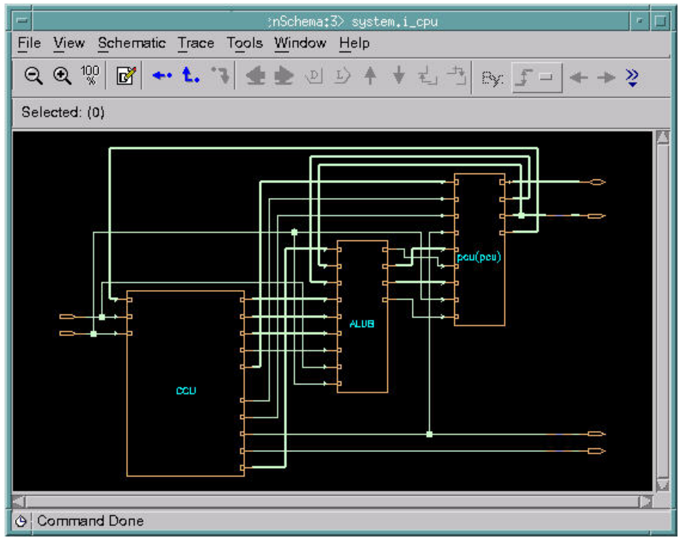


nSchema – Full Hierarchical Window

- View, traverse and understand design hierarchy graphically.
- Double click a symbol port to push into the lower level and highlight the associated net.
- Double click on a net to highlight

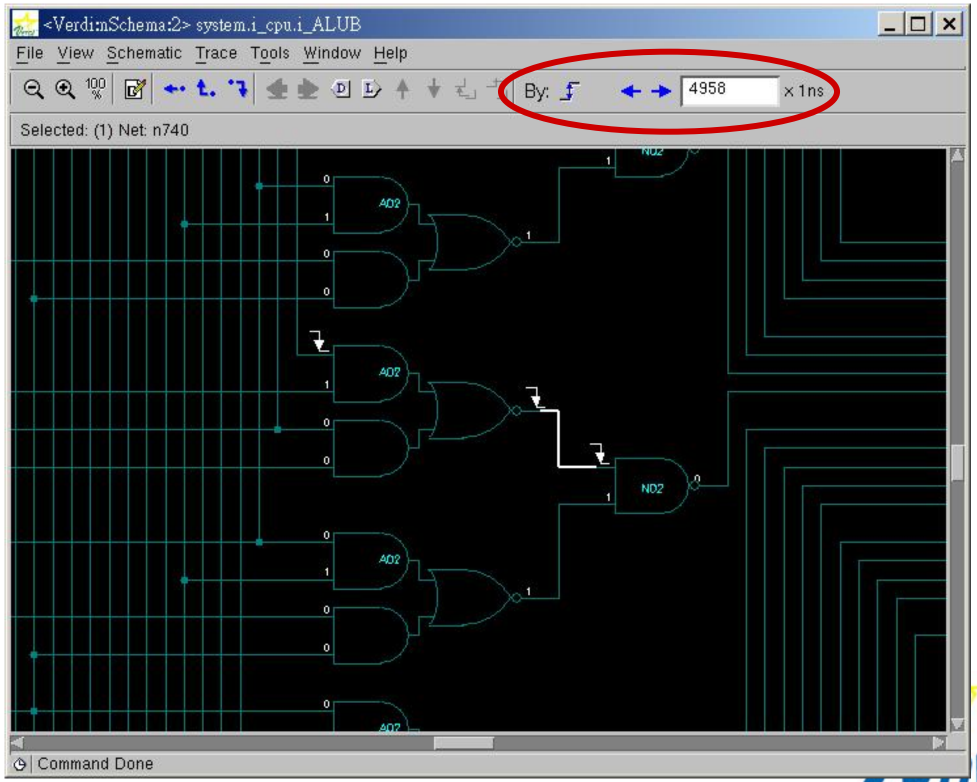
  all instances which connect to this net.
- Trace menu will highlight a variety

  of different objects in current window.
- Search commands
  - **Schematic → Signal/Instance/Instport…**
  - **Schematic → In Current Scope…**



nSchema – Active Annotation

- Display simulation results on source.
  - **Schematic → Active Annotation** in *nSchema*
- Signal values/transitions are synchronized with cursor time.
- Select signal(s) and go to next or previous transitions.
- *Any Change*, *Rising* or *Falling*



<Verdi:nSchema:2> system.i_cpu.i_ALUB
File View Schematic Trace Tools Window Help
By:
4958
x 1ns
AO2
ND2

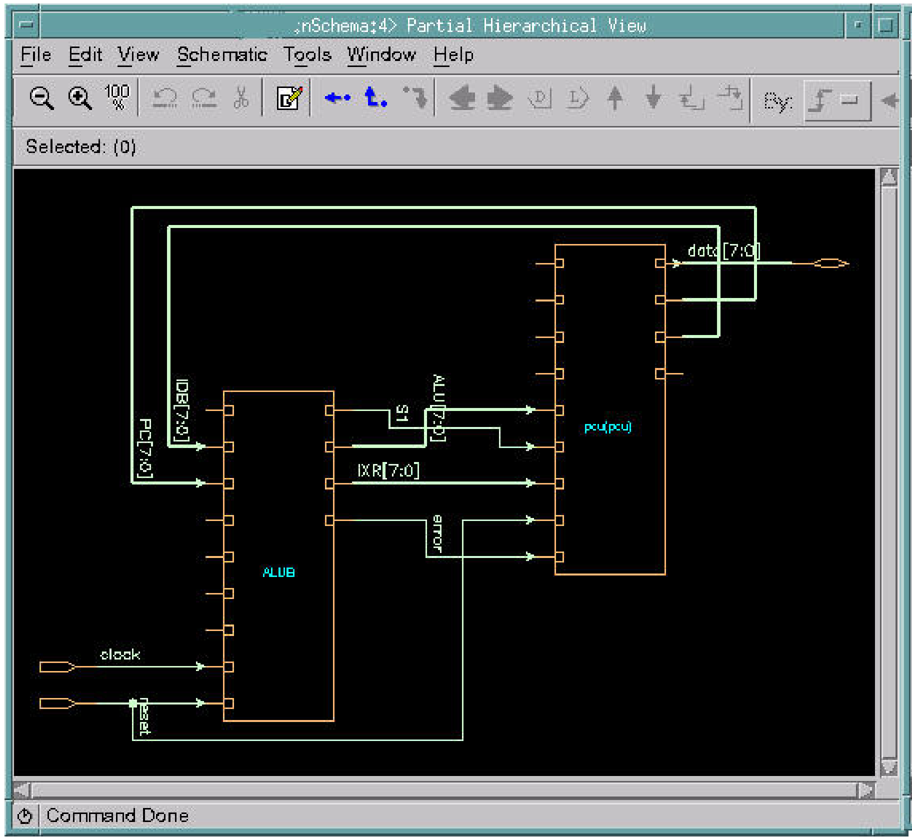
Selected: (1) Net: n740

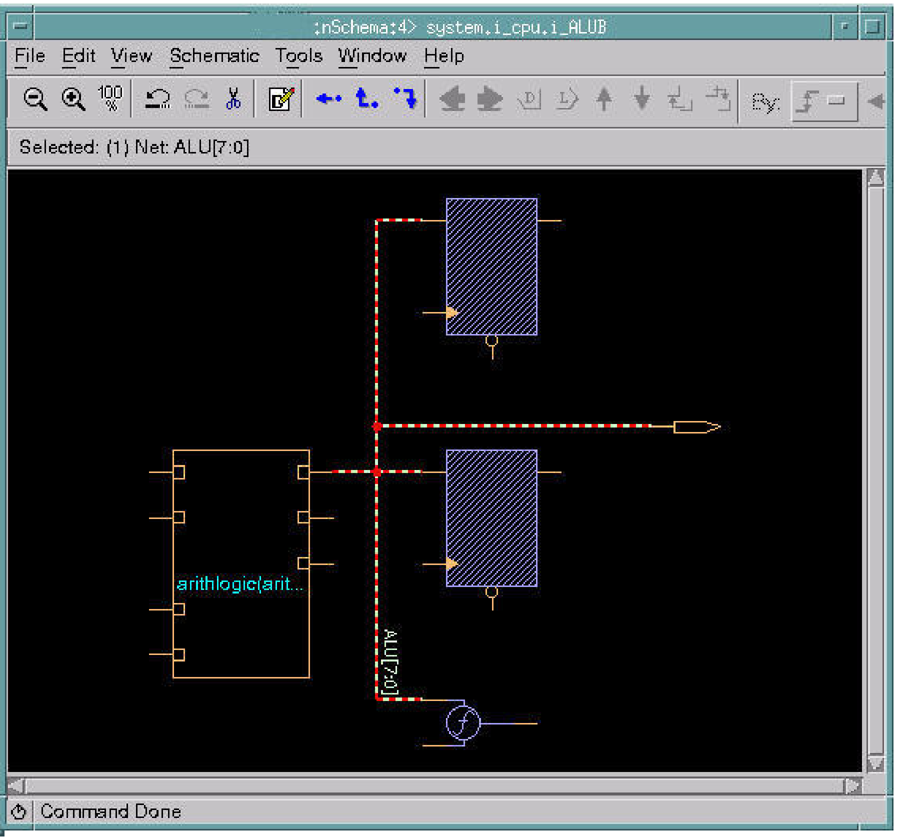
Command Done

nSchema – Create Partial Hierarchical Schematics

- Focus on a specified instance or net for debugging.
  - Select net(s) or instance(s) while holding Shift key.
  - Use **Tools → New Schematic → Browser Window** to generate partial schematic.
    - Hierarchical schematic with only those instances connected to selected nets.
    - Hierarchical schematic with only those nets connected to selected instances.
- Double click on symbol port to push into the lower level and highlight the associated net.
- Double click on I/O port to pop up one level.
- Add / delete objects.
- Undo / redo edits.

nSchema – Browser Window Example



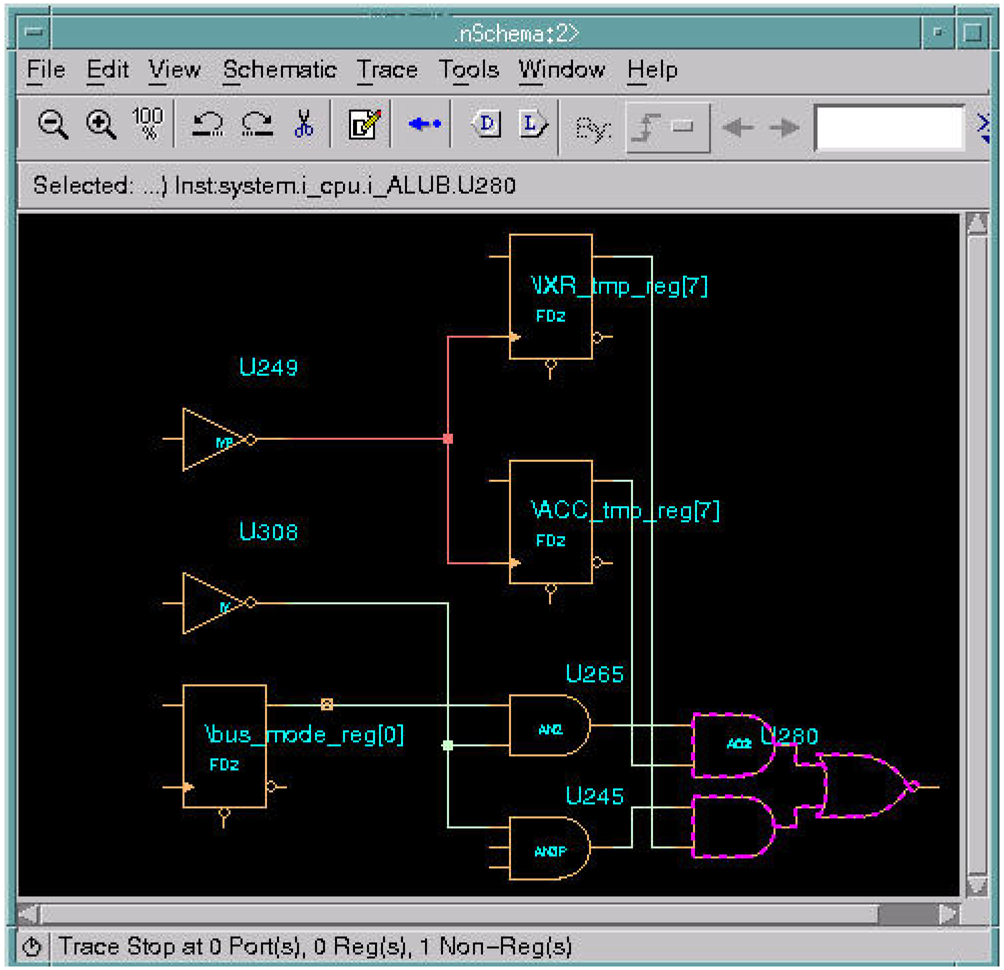

:nSchema:4> system.i_cpu.i_ALUB
File Edit View Schematic Tools Window Help
Selected: (1) Net: ALU[7:0]
arithlogic(arit...
ALU[7:0]
Command Done

NOVAS

nSchema – Create Flattened Schematics

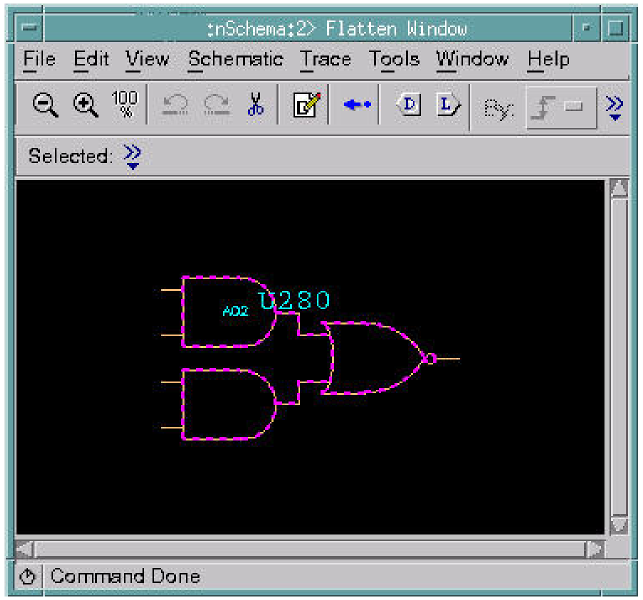
- Focus on a user specified area of the design.
  - Select primitive.
  - Use **Tools → New Schematic → Flatten Window** to generate flat schematic.
  - Add logic by double clicking an instance pin or drag and drop from other windows.
- Select object and use <Delete> key or ✂ icon to remove object from schematic.

nSchema – Flatten Window Example



Selected: ...) Inst:system.i_cpu.i_ALUB.U280

Trace Stop at 0 Port(s), 0 Reg(s), 1 Non-Reg(s)

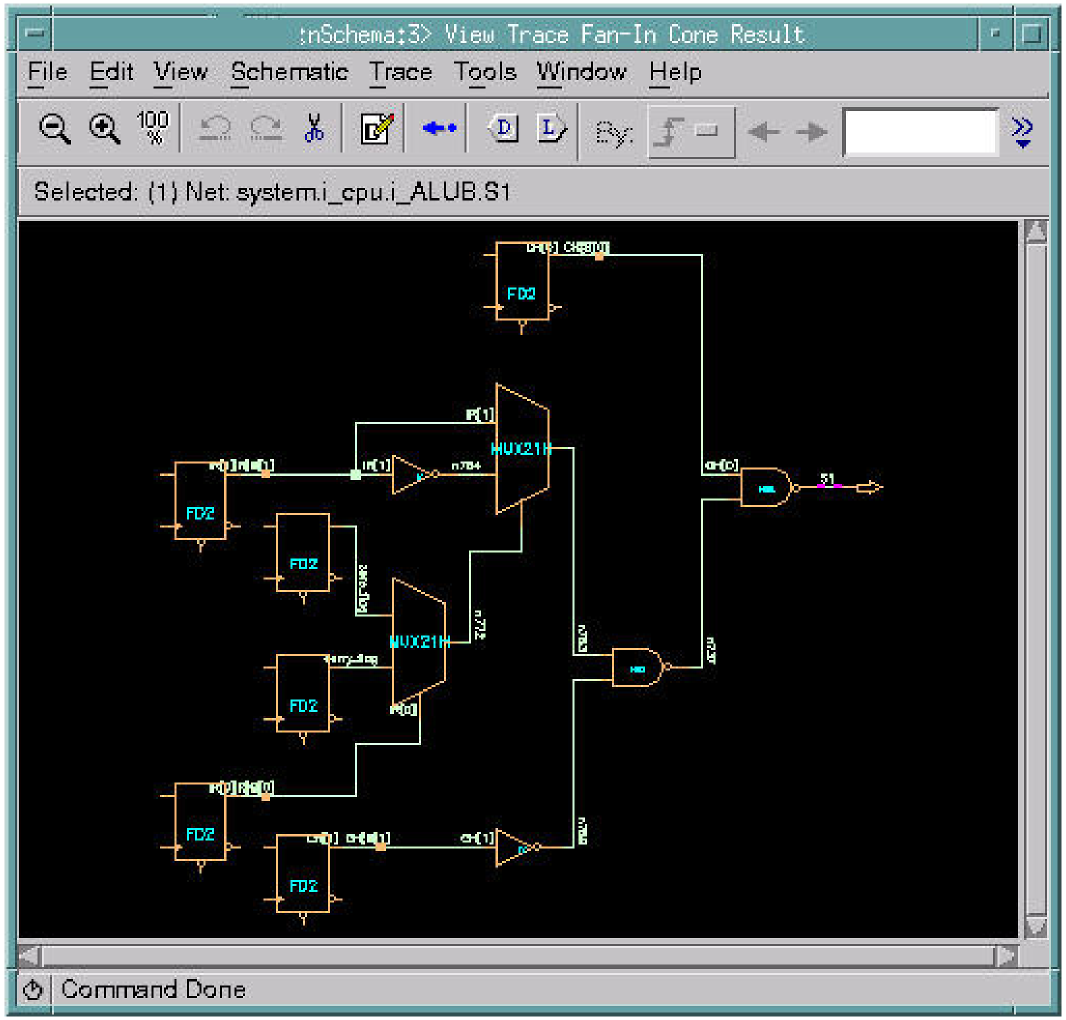

:nSchema:2> Flatten Window
File Edit View Schematic Trace Tools Window Help
Selected:
AO2
U280

Command Done

nSchema – Create Flattened Schematics – Fan-in / Fan-out Cones

- Focus on all drivers or loads of selected object.
  - Select object (net or primitive instance).
  - Use **Tools → New Schematic → Fan-In Cone / Fan-Out Cone** to generate flat schematic.
  - Add logic by double clicking an instance pin or drag and drop from other windows.
- Select object and use <Delete> key or ✂ icon to remove object from schematic.

nSchema – Fan-in Cone Example

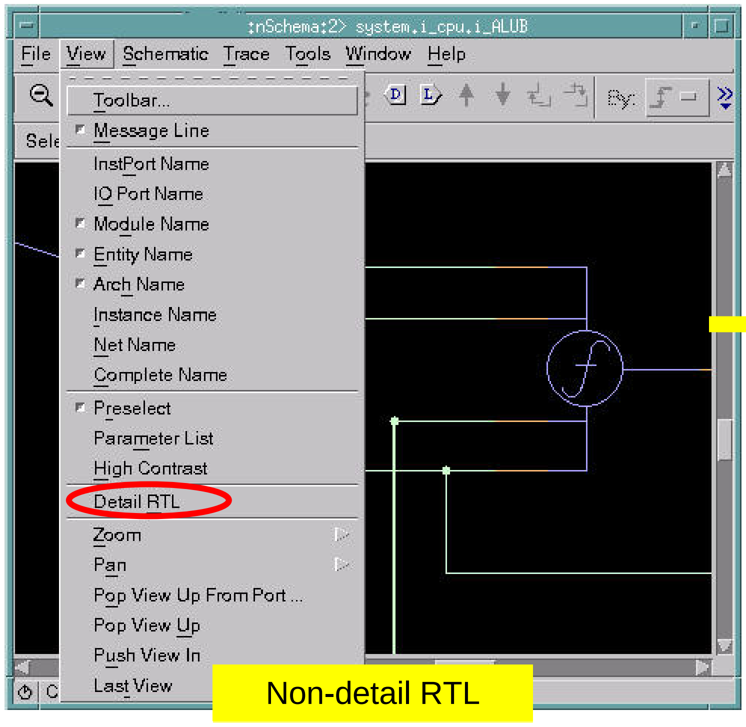

:nSchema:3> View Trace Fan-In Cone Result
File Edit View Schematic Trace Tools Window Help
100 %
FD2
MUX21H
S1

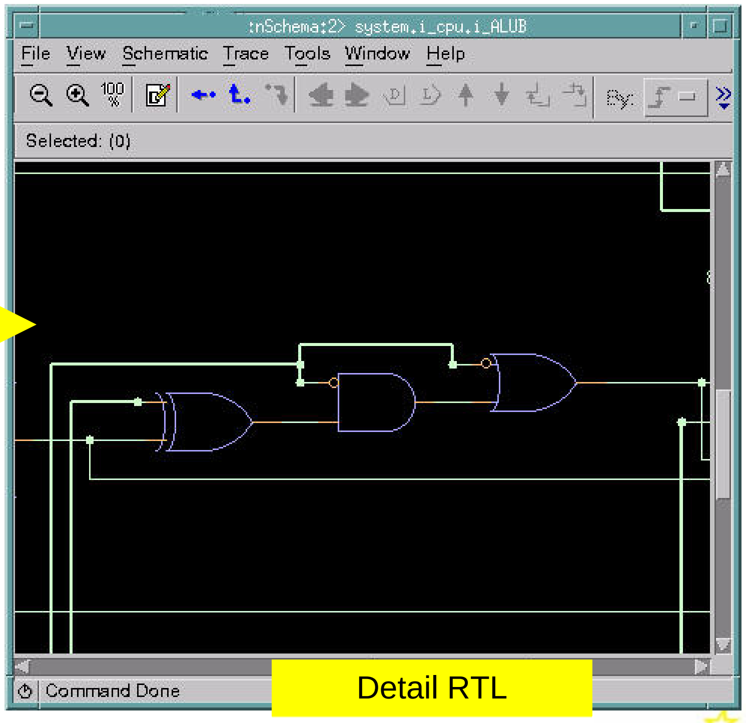
Selected: (1) Net: system.i_cpu.i_ALUB.S1

Command Done

nSchema – View Detail RTL on a Window Basis

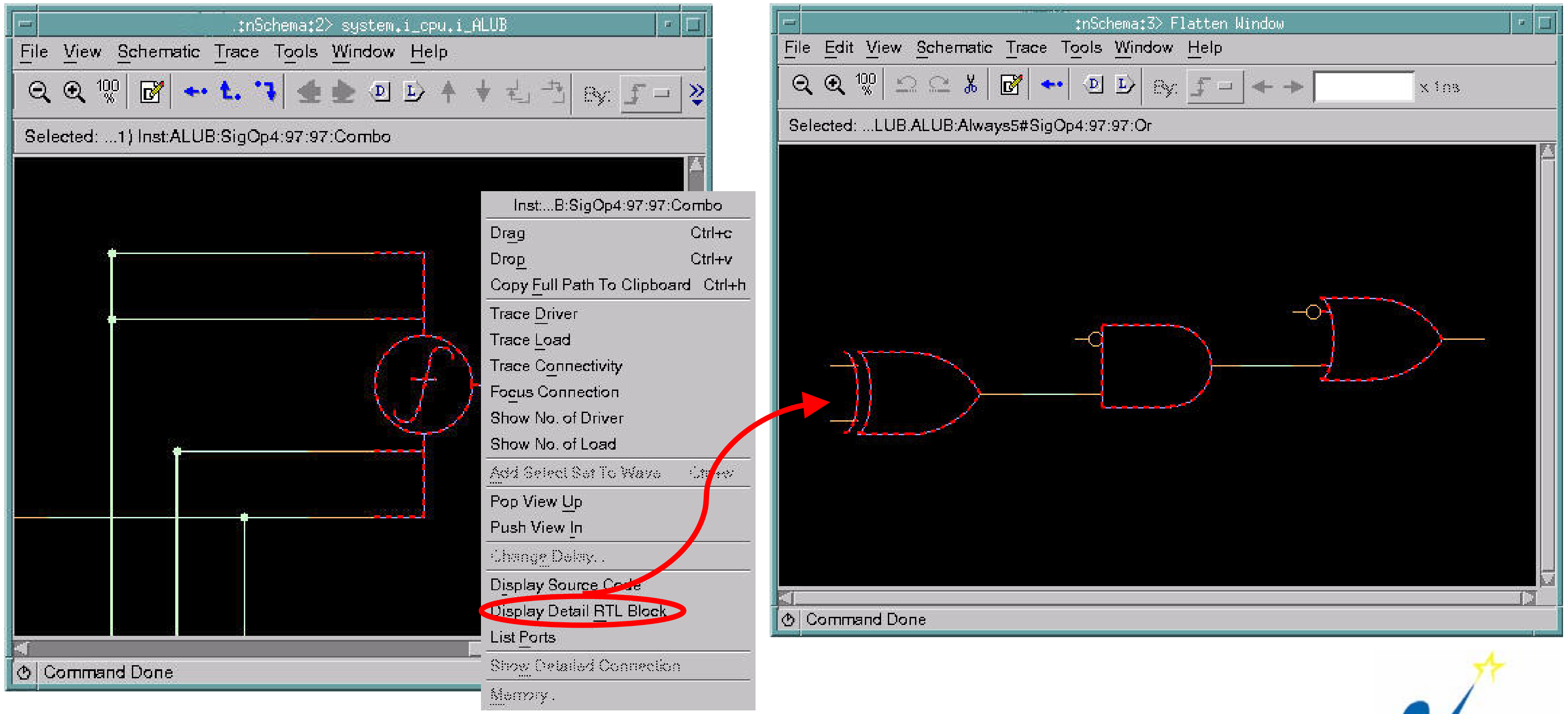
- New option: **View → Detail RTL**
  - Effective in that window only




:nSchema:2> system.i_cpu.i_ALUB
File View Schematic Trace Tools Window Help
Selected: (0)
Command Done
Detail RTL

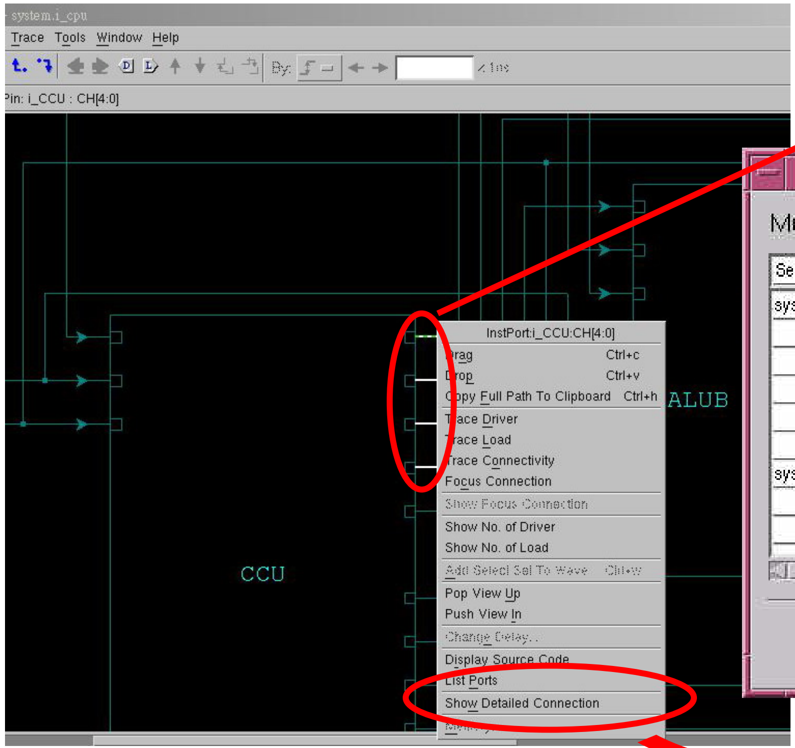
nSchema – View Detail RTL on Instance Basis

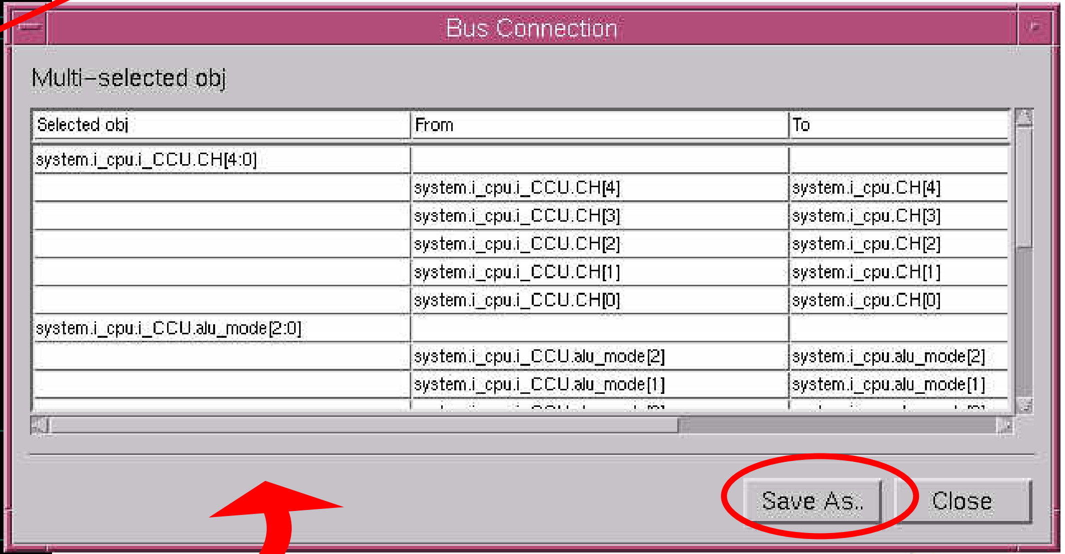
- On RTL Block, RMB → **Display Detail RTL Block**
  - New window with details will pop up


:nSchema:2> system.i_cpu.i_ALUB
File View Schematic Trace Tools Window Help
Selected: ...1) Inst:ALUB:SigOp4:97:97:Combo
Inst:...B:SigOp4:97:97:Combo
Drag Ctrl+c
Drop Ctrl+v
Copy Full Path To Clipboard Ctrl+h
Trace Driver
Trace Load
Trace Connectivity
Focus Connection
Show No. of Driver
Show No. of Load
Add Select Set To Wave Ctrl+w
Pop View Up
Push View In
Change Delay...
Display Source Code
Display Detail RTL Block
List Ports
Show Detailed Connection
Memory...
Command Done
:nSchema:3> Flatten Window
File Edit View Schematic Trace Tools Window Help
Selected: ...LUB.ALUB:Always5#SigOp4:97:97:Or
Command Done

nSchema — Show Detailed Connections for IO Pins

- Select one or more pins then invoke **RMB → Show Detail Connection**
- All detailed connection information will be shown and can be saved as text file.



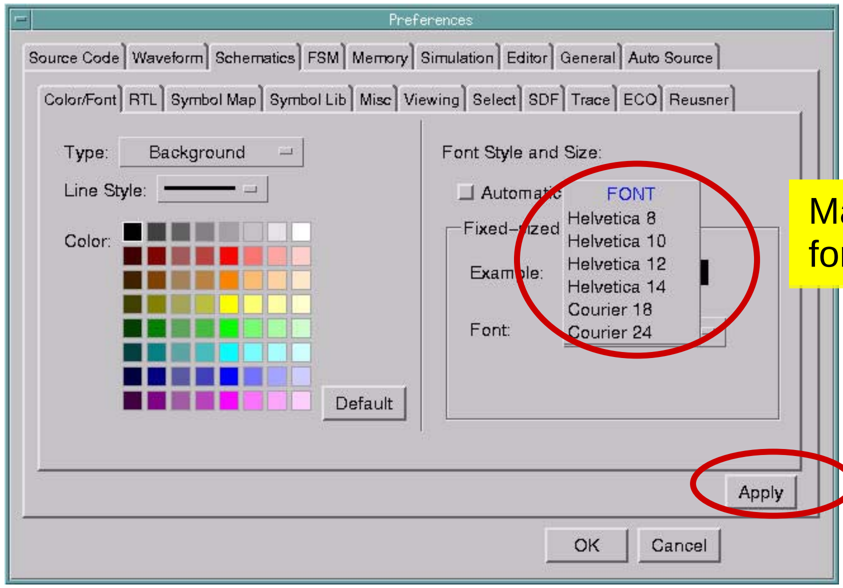


Multi-selected obj

Selected obj	From	To
system.i_cpu.i_CCU.CH[4:0]		
	system.i_cpu.i_CCU.CH[4]	system.i_cpu.CH[4]
	system.i_cpu.i_CCU.CH[3]	system.i_cpu.CH[3]
	system.i_cpu.i_CCU.CH[2]	system.i_cpu.CH[2]
	system.i_cpu.i_CCU.CH[1]	system.i_cpu.CH[1]
	system.i_cpu.i_CCU.CH[0]	system.i_cpu.CH[0]
system.i_cpu.i_CCU.alu_mode[2:0]		
	system.i_cpu.i_CCU.alu_mode[2]	system.i_cpu.alu_mode[2]
	system.i_cpu.i_CCU.alu_mode[1]	system.i_cpu.alu_mode[1]

nSchema – Miscellaneous

- Use **Tools → Preferences...** to customize the appearance globally.


Preferences
Source Code | Waveform | Schematics | FSM | Memory | Simulation | Editor | General | Auto Source
Color/Font | RTL | Symbol Map | Symbol Lib | Misc | Viewing | Select | SDF | Trace | ECO | Reusner
Default
Apply
OK
Cancel

Type: Background

Line Style:

Color:

Font Style and Size:

Automatic

Fixed-sized

Example:

Font:

FONT
Helvetica 8
Helvetica 10
Helvetica 12
Helvetica 14
Courier 18
Courier 24

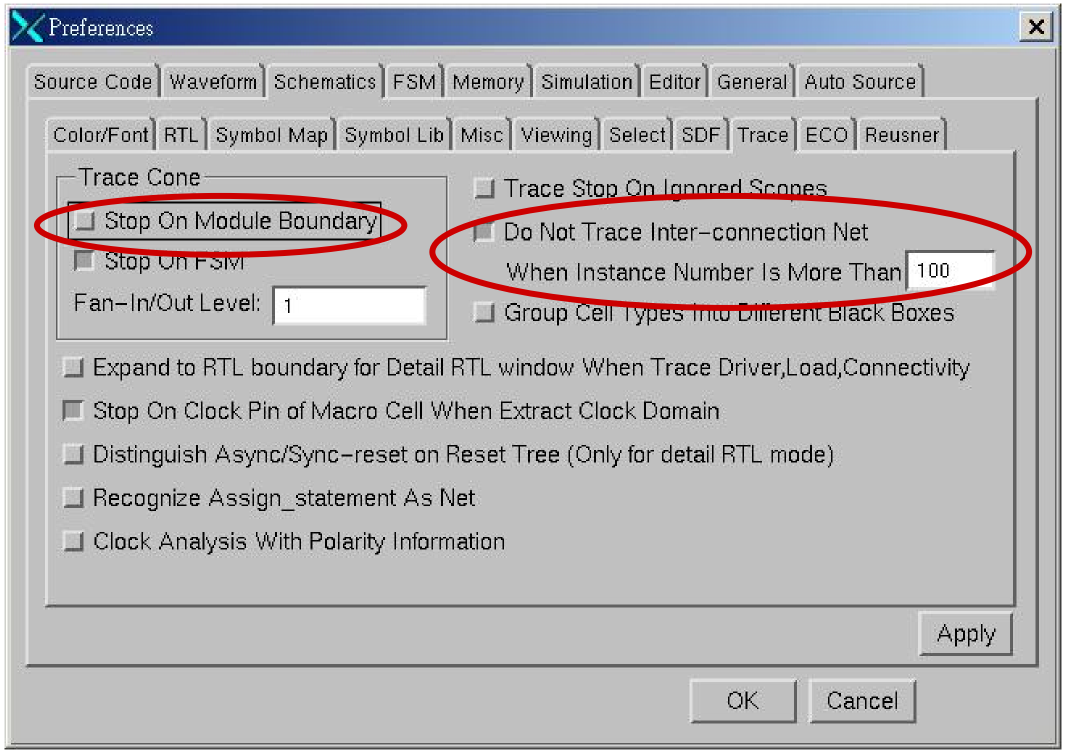
Manually control the font used for names on schematics

Apply settings without leaving the Preferences window

- Use **File → Print** to print multi-page schematics.
- Use **Help → Command Reference** for additional information.

nSchema – Miscellaneous (Count.)

- Use **Tools → Options → Trace Cone Stop On Module Boundary** or **Tools → Preferences... → Trace** tab to force **Trace → Fan-In Cone / Fan-Out Cone** to stop on module boundary.
- Set the constrain for tracing the inter-connection net.



Trace Cone
- [ ] Stop On Module Boundary
- [ ] Stop On FSM

Fan-In/Out Level: 1

- [ ] Trace Stop On Ignored Scopes
- [ ] Do Not Trace Inter-connection Net

When Instance Number Is More Than 100

- [ ] Group Cell Types Into Different Black Boxes
- [ ] Expand to RTL boundary for Detail RTL window When Trace Driver,Load,Connectivity
- [ ] Stop On Clock Pin of Macro Cell When Extract Clock Domain
- [ ] Distinguish Async/Sync-reset on Reset Tree (Only for detail RTL mode)
- [ ] Recognize Assign_statement As Net
- [ ] Clock Analysis With Polarity Information

nSchema – Summary

- At this time you should
  - Understand the *nSchema* functionality.
  - Be able to open a schematic and relate the symbols back to the source code.
  - Know how to locate signal(s) or instance(s) and generate different partial schematics.
  - Be able to access and change the preferences.

Training Course

- Verdi Background and Overview
- Loading Your Design
- nTrace – View and Debug Source Code
- nSchema – View and Debug Designs Graphically
- nState – View and Analyze State Machines
- nWave – View Simulation Results in Waveforms
- Debugging with Verdi

nState – View and Analyze State Machines

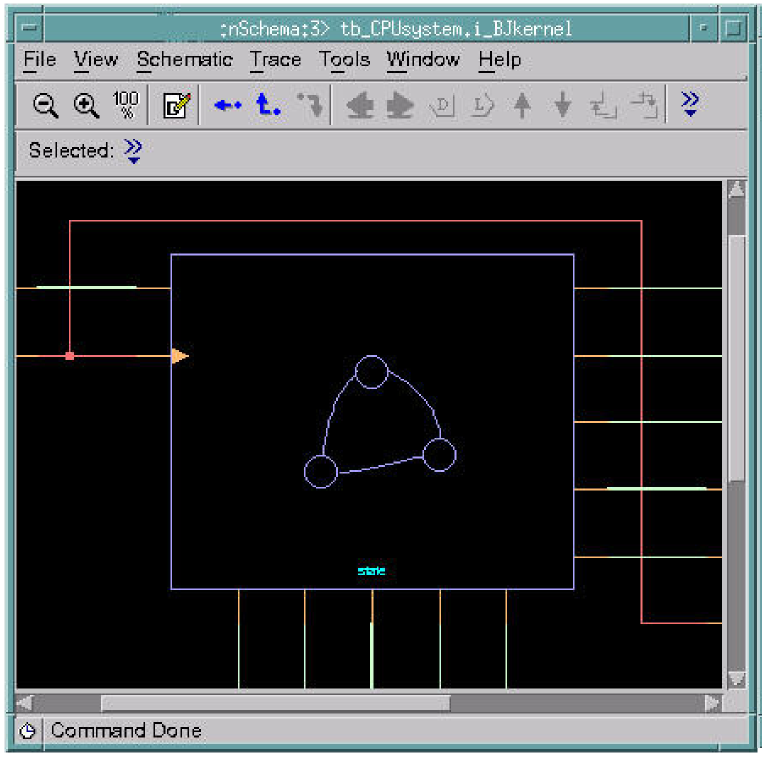
- Overview
- Invoke *nState*
- Display Options
- Partial FSM
- State Animation
- Analysis Report
- Interacting FSM Display
- Miscellaneous
- Lab

nState Overview

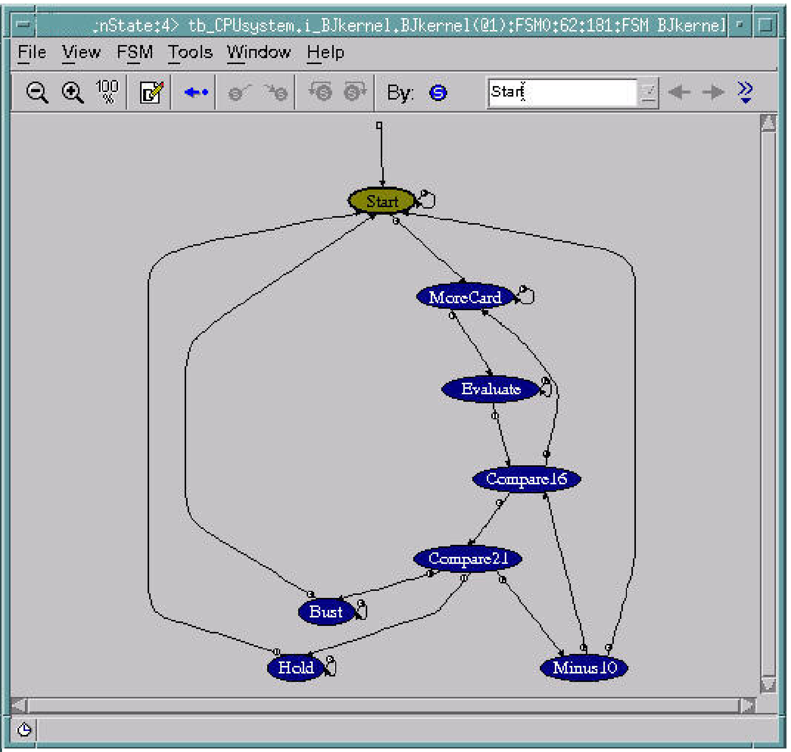
- Display and analyze finite state machines.
  - Show or hide conditions / actions on a state or a transition.
  - Search by State Sequence.
  - State Animation using current simulation results.
  - Generate analysis reports using current simulation results.

Invoke nState

- Double click on the state machine symbol in *nSchema*.



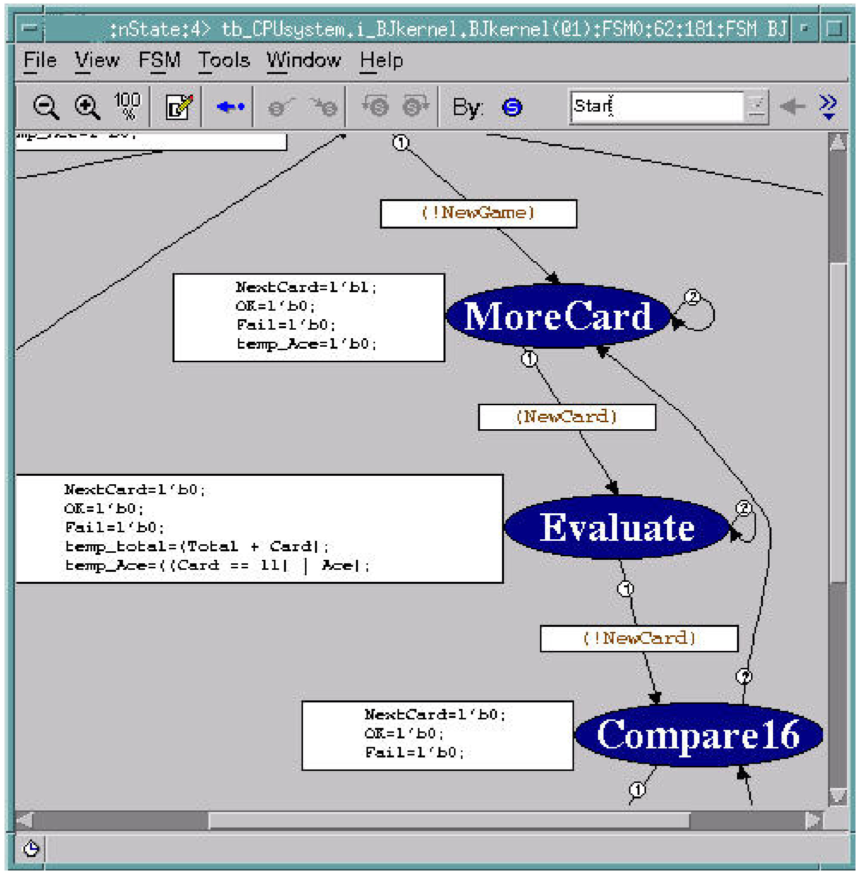
:nSchema:3> tb_CPUsystem.i_BJkernel
File View Schematic Trace Tools Window Help
Selected:
state
Command Done



:nState:4> tb_CPUsystem.i_BJkernel.BJkernel(@1):FSM0:62:181:FSM BJkernel
File View FSM Tools Window Help
By:
Start
Start
MoreCard
Evaluate
Compare16
Compare21
Bust
Hold
Minus10

nState – Display Options

- Use **View → State Action** to display signal assignments for each state.
- Use **View → Transition Condition** to display transition condition on the state diagram.
- Use **View → Transition Action** to display signal assignments for each transition.


:nState:4> tb_CPUsystem.i_BJkernel.BJkernel(@1):FSM0:62:181:FSM BJ
File View FSM Tools Window Help
By:
Start
(!NewGame)
MoreCard
(NewCard)
Evaluate
(!NewCard)
Compare16

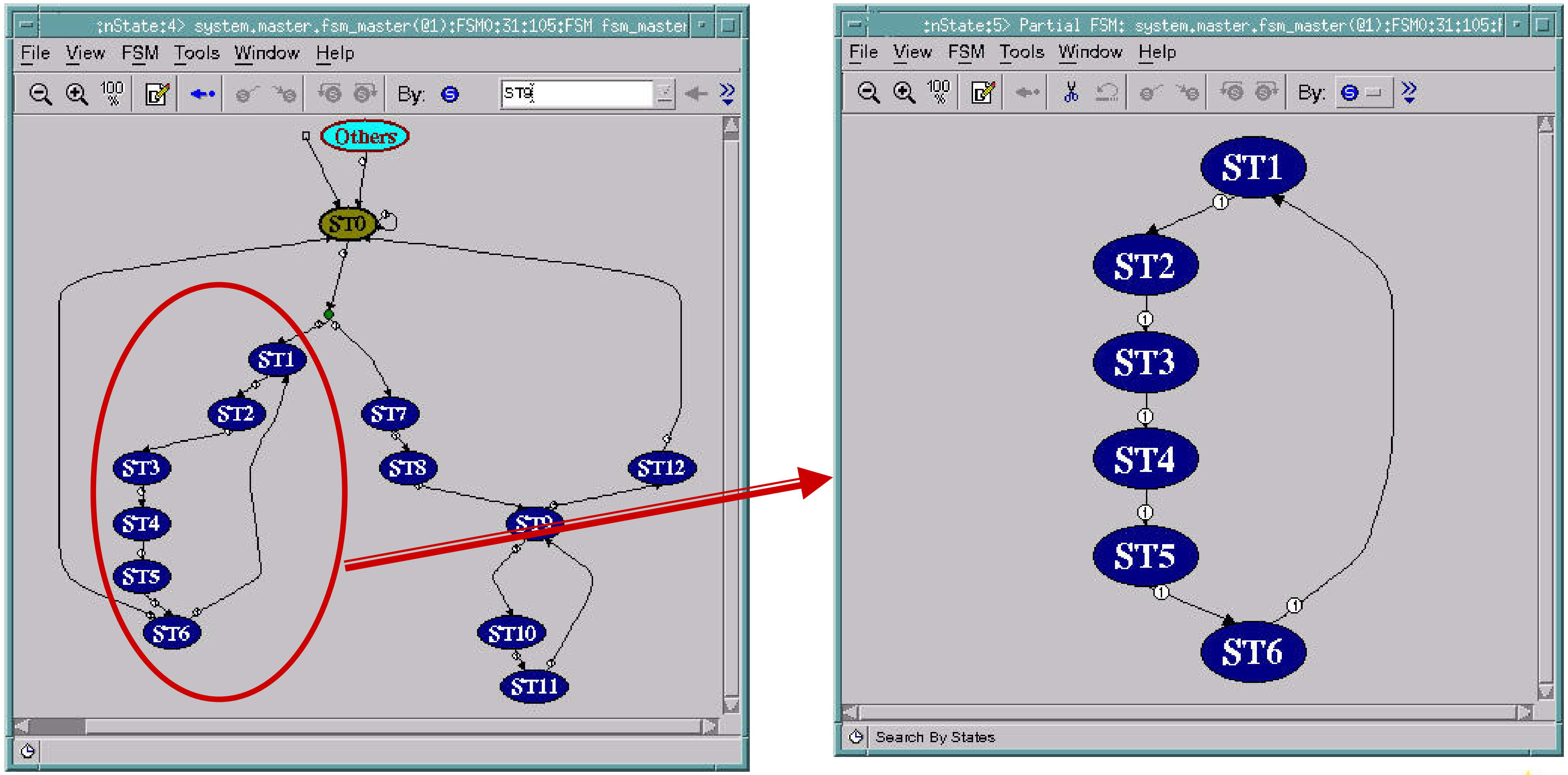
```
NextCard=1'b1;
OK=1'b0;
Fail=1'b0;
temp_Ace=1'b0;
```

```
NextCard=1'b0;
OK=1'b0;
Fail=1'b0;
temp_total=(Total + Card);
temp_Ace=((Card == 11) | Ace);
```

```
NextCard=1'b0;
OK=1'b0;
Fail=1'b0;
```

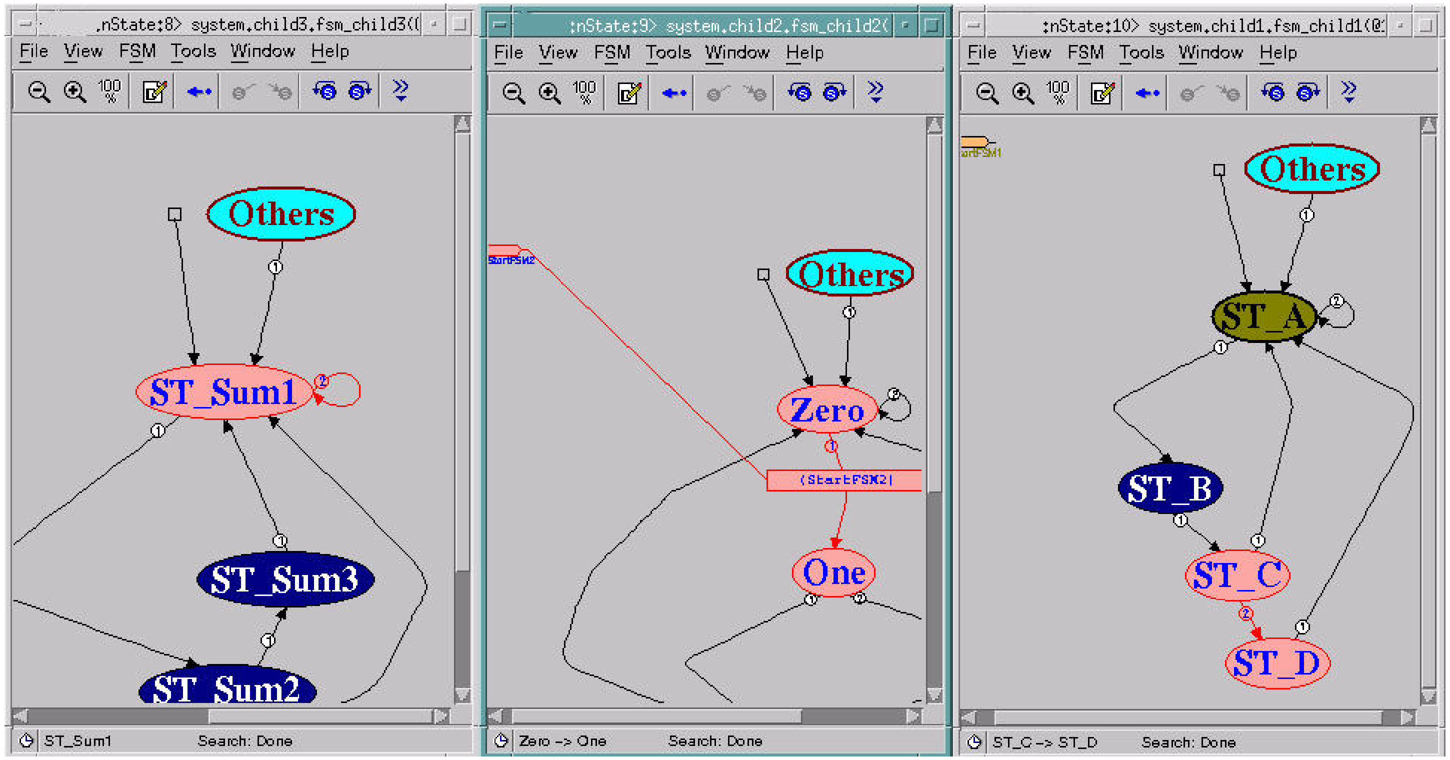
nState – Partial FSM

- Create partial FSM views to aid understanding


:nState:4> system.master.fsm_master(@1):FSM0:31:105:FSM fsm_master
File View FSM Tools Window Help
By:
ST9
Others
ST0
ST1
ST2
ST3
ST4
ST5
ST6
ST7
ST8
ST9
ST10
ST11
ST12
:nState:5> Partial FSM: system.master.fsm_master(@1):FSM0:31:105:f
File View FSM Tools Window Help
By:
ST1
ST2
ST3
ST4
ST5
ST6
Search By States

nState – Interacting FSM Display

- Use **Tools → Extract Interactive FSM…** in *nTrace* or *nSchema*.
  - Shows the interaction between multiple state machines.

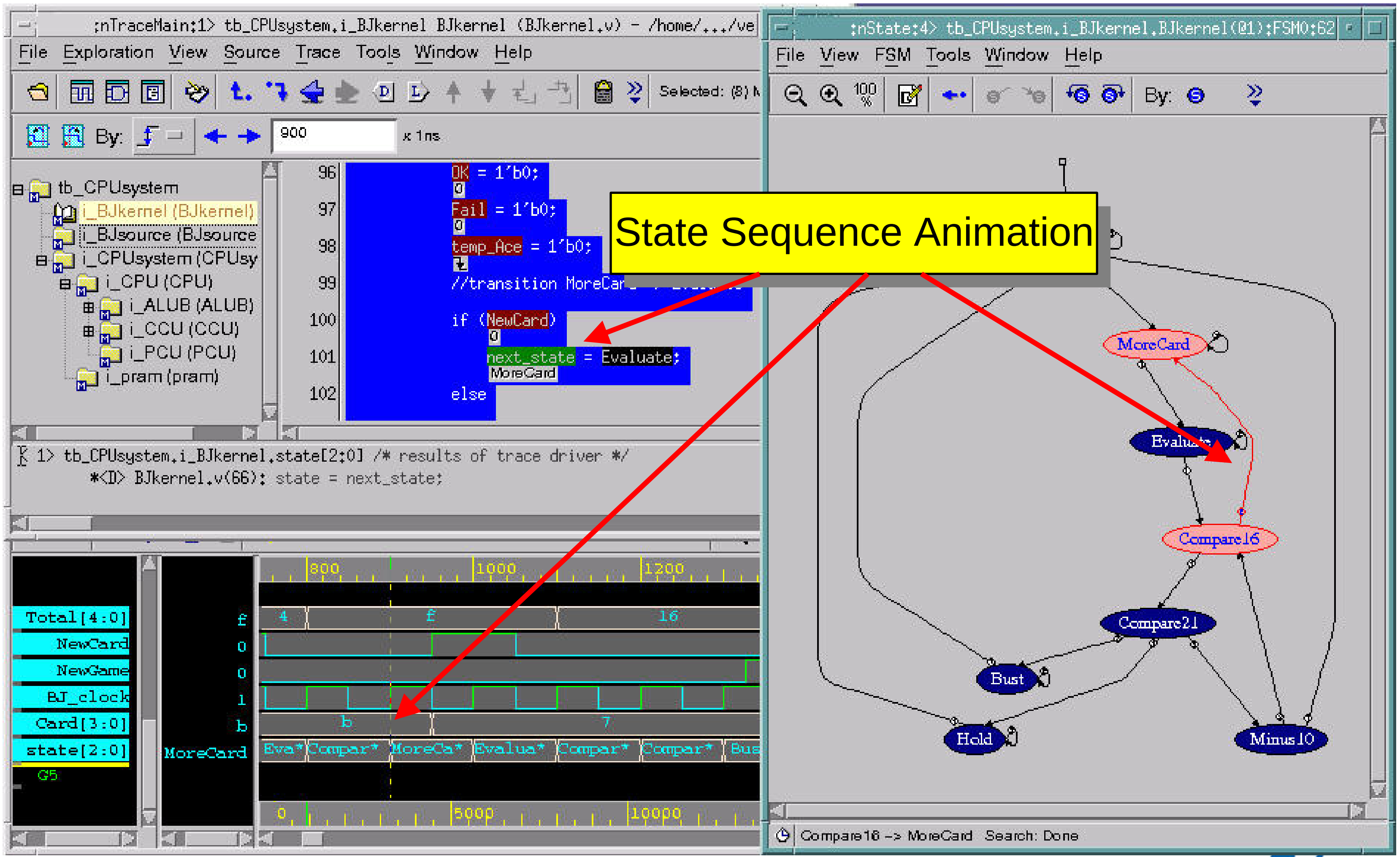


:nState:8> system.child3.fsm_child3(
File View FSM Tools Window Help
Others
ST_Sum1
ST_Sum3
ST_Sum2
ST_Sum1 Search: Done
:nState:9> system.child2.fsm_child2(
File View FSM Tools Window Help
StartFSM2
Others
Zero
(StartFSM2)
One
Zero -> One Search: Done
:nState:10> system.child1.fsm_child1(@
File View FSM Tools Window Help
StartFSM1
Others
ST_A
ST_B
ST_C
ST_D
ST_C -> ST_D Search: Done

nState – State Animation 1

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

nState – State Animation 2



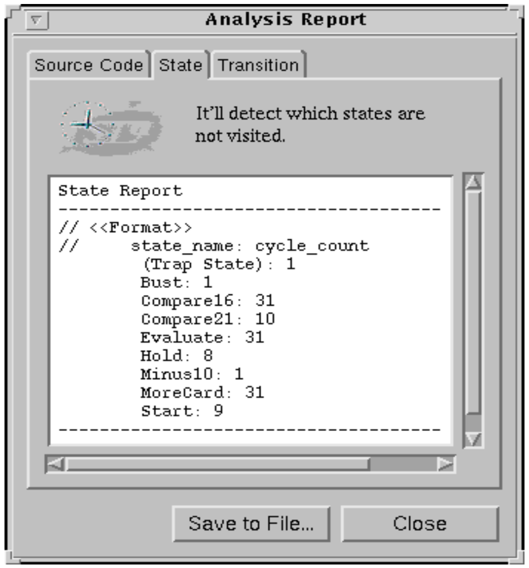
```
OK = 1'b0;
Fail = 1'b0;
temp_Ace = 1'b0;
//transition MoreCard
if (NewCard)
    next_state = Evaluate;
else
```

```
1> tb_CPUsystem.i_BJkernel.state[2:0] /* results of trace driver */
    *<D> BJkernel.v(66): state = next_state;
```

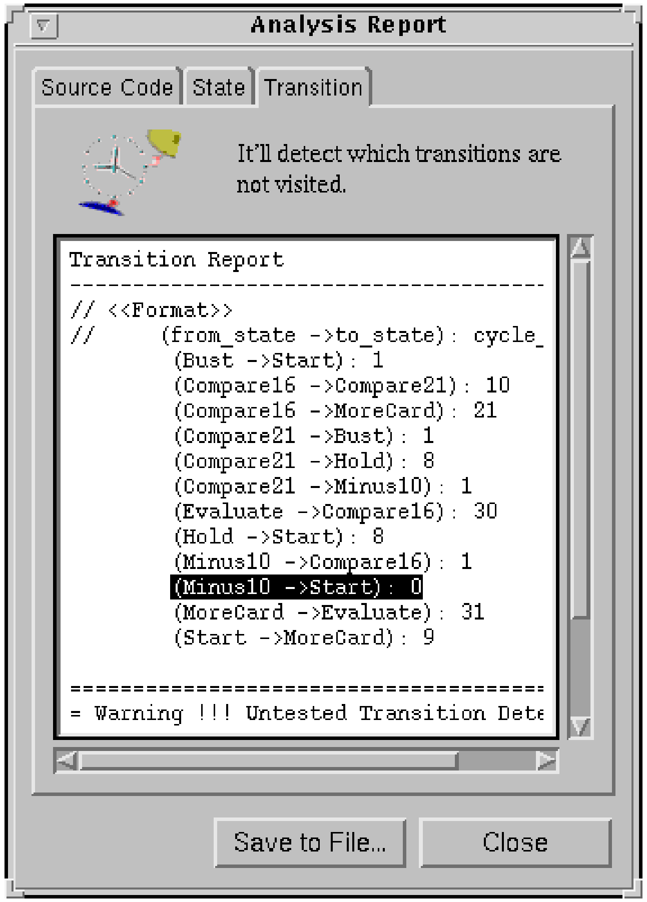
Compare16 -> MoreCard   Search: Done

nState – Analysis Report

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



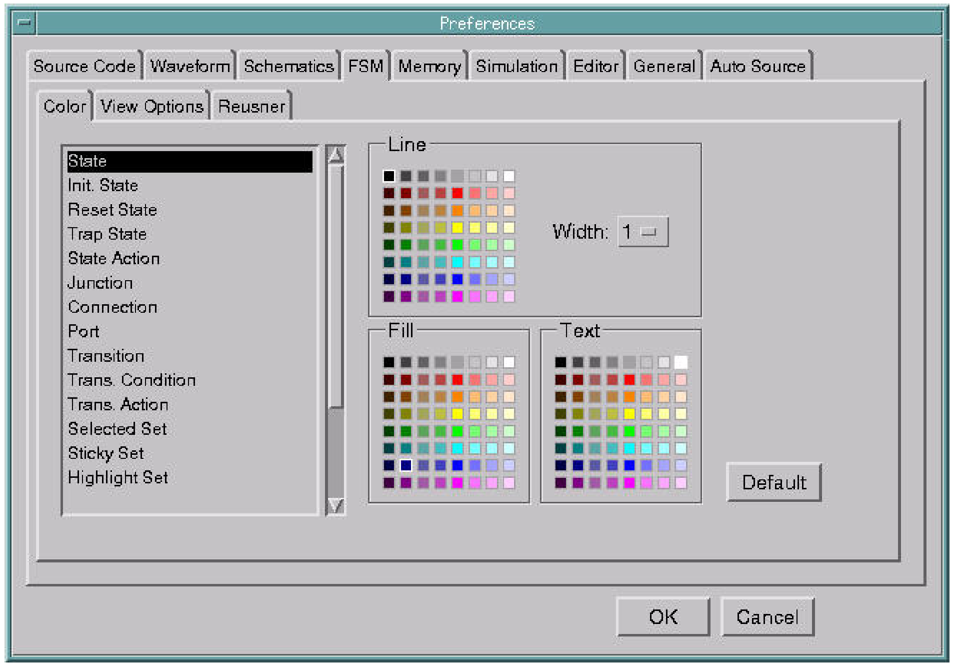
```
State Report
-----------------------------------------
// <<Format>>
//       state_name: cycle_count
          (Trap State): 1
          Bust: 1
          Compare16: 31
          Compare21: 10
          Evaluate: 31
          Hold: 8
          Minus10: 1
          MoreCard: 31
          Start: 9
-----------------------------------------
```



```
Transition Report
---------------------------------------
// <<Format>>
//      (from_state ->to_state): cycle_
         (Bust ->Start): 1
         (Compare16 ->Compare21): 10
         (Compare16 ->MoreCard): 21
         (Compare21 ->Bust): 1
         (Compare21 ->Hold): 8
         (Compare21 ->Minus10): 1
         (Evaluate ->Compare16): 30
         (Hold ->Start): 8
         (Minus10 ->Compare16): 1
         (Minus10 ->Start): 0
         (MoreCard ->Evaluate): 31
         (Start ->MoreCard): 9

=======================================
= Warning !!! Untested Transition Dete
```

nState – Miscellaneous

- Use **Tools → Preferences...** to customize the appearance globally.


Preferences

Source Code | Waveform | Schematics | FSM | Memory | Simulation | Editor | General | Auto Source

Color | View Options | Reusner

State
Init. State
Reset State
Trap State
State Action
Junction
Connection
Port
Transition
Trans. Condition
Trans. Action
Selected Set
Sticky Set
Highlight Set

Line

Width: 1

Fill

Text

Default

OK | Cancel

- Use **File → Print** to print multi-page state diagrams.
- Use **Help → Command Reference** for additional information.

nState – Summary

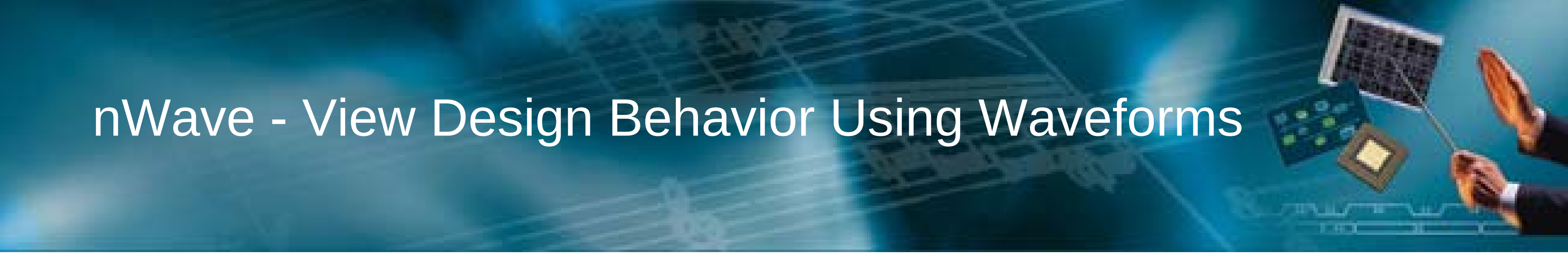
- At this time you should
  - Understand the *nState* functionality.
  - Be able to open a FSM and enable different display options
  - Know how to enable state animation and view analysis reports.
  - Be able to access and change the preferences.

nSchema – Wrap-up

- Questions?
- Take 20 minutes to complete the *nSchema and nState* LAB

Training Course

- Verdi Background and Overview
- Loading Your Design
- nTrace – View and Debug Source Code
- nSchema – View and Debug Designs Graphically
- nState – View and Analyze State Machines
- nWave – View Simulation Results in Waveforms
- Debugging with Verdi



nWave - View Design Behavior Using Waveforms

- Overview
- FSDB File
- Linking Overview
- System Tasks
- FSDB Utilities
- Invoke *nWave*
- Open Simulation Results
- Display and Organize Signals
  - Adding Signals / Get Signals Form
  - Group/Bus Operations
  - Display Options and Preference
- Focus Debug Effort
  - Search Value and Search Constraint
  - Markers
  - Zero Time Glitch and Event Sequence
  - Logical Operations
  - Complex Event
  - Waveform Comparisons
  - Analog v.s. Digital
  - Aliasing
  - Comments
- Save / Restore Signals
- Lab

nWave Overview

- A complete waveform analysis environment.
  - Supports FSDB format (through PLI) or VCD file format.
  - Flexible signal group management.
  - Fully integrated and synchronized with other verdi views.
  - Interactive visual comparison of waveforms.
  - Built-in logic analyzer and event searching.
  - Capable of displaying analog waveforms.

nWave – Fast Signal Database (FSDB) File

- Compact binary file format which contains the simulation signal data.
- Captured during simulation using PLI system tasks for both VHDL and VERILOG.
- Advantages:
  - Smaller in size than VCD.
  - Loads faster than VCD.
  - Open file format so other vendor tools can dump data, Verisity, Vera, Ikos, etc.

nWave – Brief Overview of Object File Linking

- In general, use LD_LIBRARY_PATH to point to pre-compiled PLI shared object files, but simulator and platform dependent.
  - All pre-compiled PLI shared object files are found in the software release tree
  - $verdi_HOME/share/PLI/<simulator>/<platform>
- Nearly all simulators supported, MTI, NC, VCS, etc.
- Standard PLI source files also available for further customization.

nWave – System Tasks to Dump VHDL to FSDB File

- **fsdbDumpfile** - Specify FSDB file name.
- **fsdbDumpvars** - Dump the specified instances and nets.
- **fsdbDumpvarsToFile** - Dump scope/depth from a designated file.
- **fsdbDumpon** - Turn on FSDB dumping.
- **fsdbDumpoff** - Turn off FSDB dumping.
- **fsdbSwitchDumpFile** - Switch dumping to another FSDB file.
- **fsdbAutoSwitchDumpfile** - Limit FSDB file size and switch dumping to new FSDB file automatically.
- **fsdbDumpMem** - Dump the contents of specified memories.

Note: You must link your VHDL simulator before you can use the above system tasks. See Command Reference.

nWave – System Tasks to Dump Verilog to FSDB File

- **$fsdbDumpvars** - Dump the specified instances and nets.
- **$fsdbDumpfile** - Specify FSDB file name.
- **$fsdbDumpon** - Turn on FSDB dumping.
- **$fsdbDumpoff** - Turn off FSDB dumping.
- **$fsdbSwitchDumpFile** - Switch dumping to another FSDB file.
- **$fsdbAutoSwitchDumpfile** - Limit FSDB file size and switch dumping to new FSDB file automatically.
- **$fsdbDumpflush** - Force to dump result to FSDB file.
- **$fsdbDumpMem** - Dump the contents of specified memories.
- **$fsdbDumpStrength** - Dump the strength of signals.
- **$fsdbDumpvarsToFile** - Dump scope/depth from a designated file.

Note: You must link your Verilog simulator before you can use the above system tasks. See Command Reference.

nWave – FSDB Utilities

- vfast
  - Command line conversion of VCD files.
- fsdbextract
  - Extract signals, scopes, time periods from existing fsdb files without re-simulating.
- fsdbmerge
  - Merge several fsdb files into one.
- fsdbreport
  - Generates a report of value changes for specified signals.

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

nWave – Invoke nWave and Load FSDB File

- To open *nWave* from *nTrace*, click on the **New Waveform** icon or **Tools → New Waveform**.
- Use **File → Open** or toolbar icon to open simulation results file.
- On the verdi command line use the option

  -ssf <filename>.fsdb
- Load VCD files generated by $dumpvars.
  - VCD files are automatically converted to FSDB at load time.
- Open several files in the same window.
  - Use **File → Set Active...** to specify which file is active.
- Open several files in different *nWave* windows.
  - Use **Window → Change to Primary** to specify which window is active.

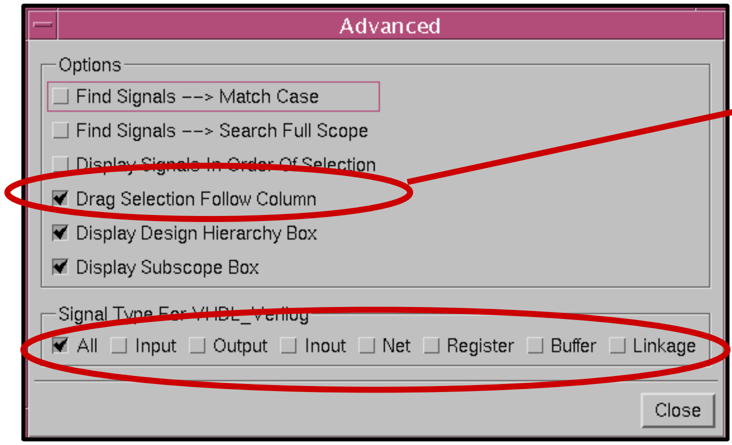
nWave – Adding Signals

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

nWave – Get Signals Form

- Directly Form Bus based on user-defined rules.
- Option to Find Signals in full scope.
- Option to display signals in order of selection not alphabetical.
- Signal type filter.

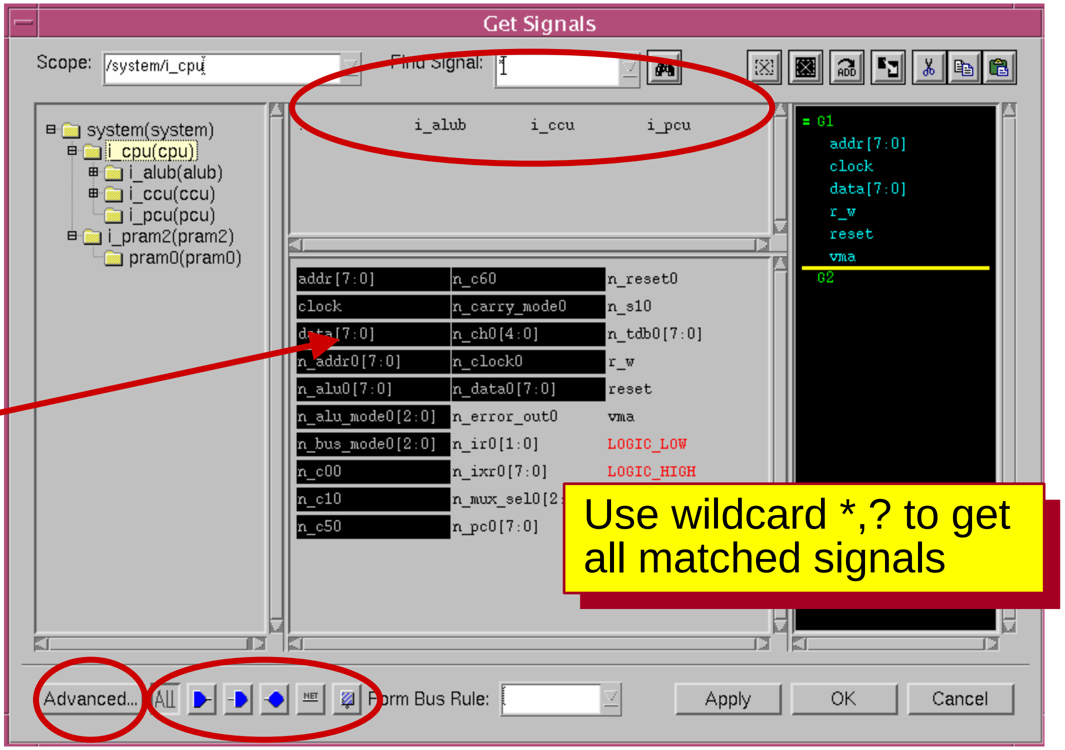
Sub-block information of selected block in hierarchy browser



Advanced

Options
- [ ] Find Signals --> Match Case
- [ ] Find Signals --> Search Full Scope
- [ ] Display Signals In Order Of Selection
- [x] Drag Selection Follow Column
- [x] Display Design Hierarchy Box
- [x] Display Subscope Box

Signal Type For VHDL_Verilog
- [x] All
- [ ] Input
- [ ] Output
- [ ] Inout
- [ ] Net
- [ ] Register
- [ ] Buffer
- [ ] Linkage



Get Signals

Scope: /system/i_cpu

Find Signal:

```
⊟ system(system)
  ⊟ i_cpu(cpu)
    ⊞ i_alub(alub)
    ⊞ i_ccu(ccu)
      i_pcu(pcu)
  ⊟ i_pram2(pram2)
      pram0(pram0)
```

```
i_alub      i_ccu      i_pcu
```

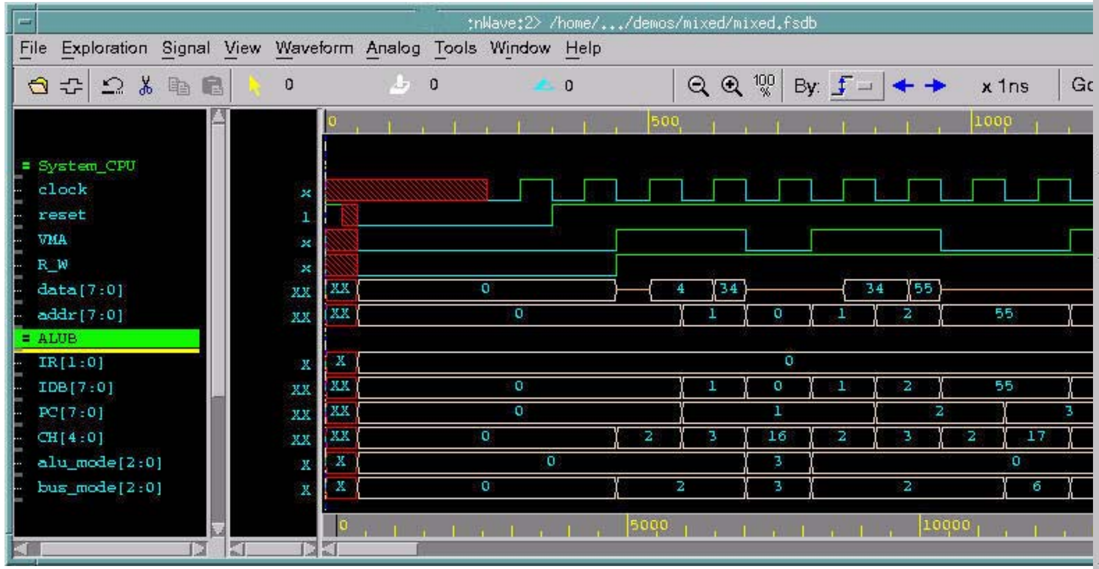
```
addr[7:0]          n_c60            n_reset0
clock              n_carry_mode0    n_s10
data[7:0]          n_ch0[4:0]       n_tdb0[7:0]
n_addr0[7:0]       n_clock0         r_w
n_alu0[7:0]        n_data0[7:0]     reset
n_alu_mode0[2:0]   n_error_out0     vma
n_bus_mode0[2:0]   n_ir0[1:0]       LOGIC_LOW
n_c00              n_ixr0[7:0]      LOGIC_HIGH
n_c10              n_mux_sel0[2:
n_c50              n_pc0[7:0]
```

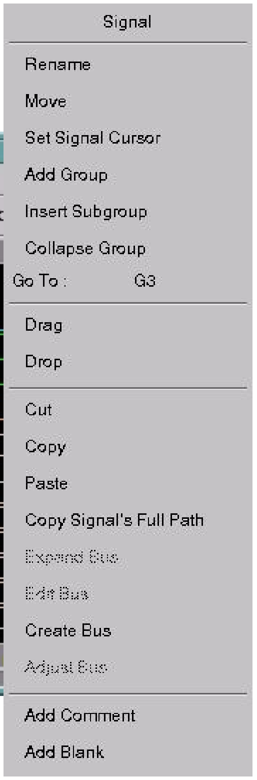
```
= G1
   addr[7:0]
   clock
   data[7:0]
   r_w
   reset
   vma
  G2
```

Use wildcard *,? to get all matched signals

nWave – Group Operations

- Use RMB menu in signal window to:
  - Add Group
  - Rename Group name
- **View → Go To Group…** or RMB **Go to Group** <Select Group name> jumps to specified group.
- Expand / collapse group member by double clicking.

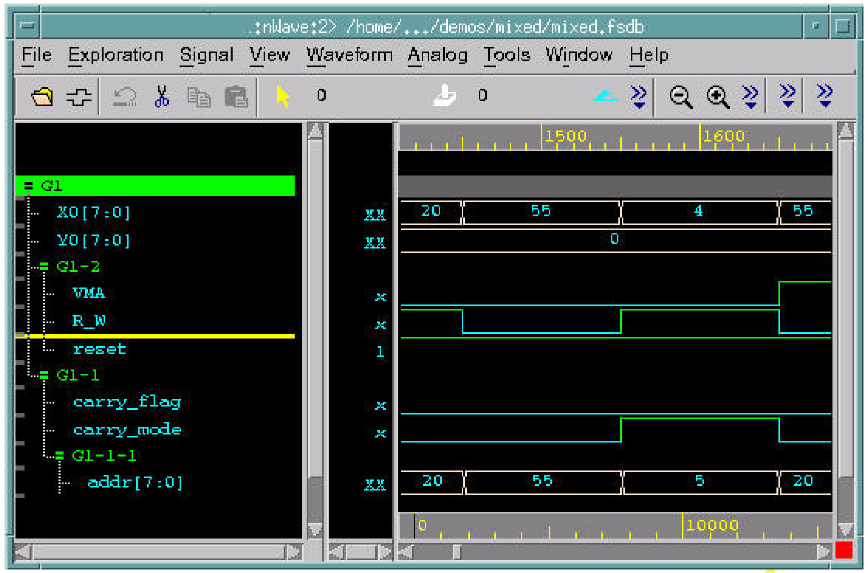




NOVAS

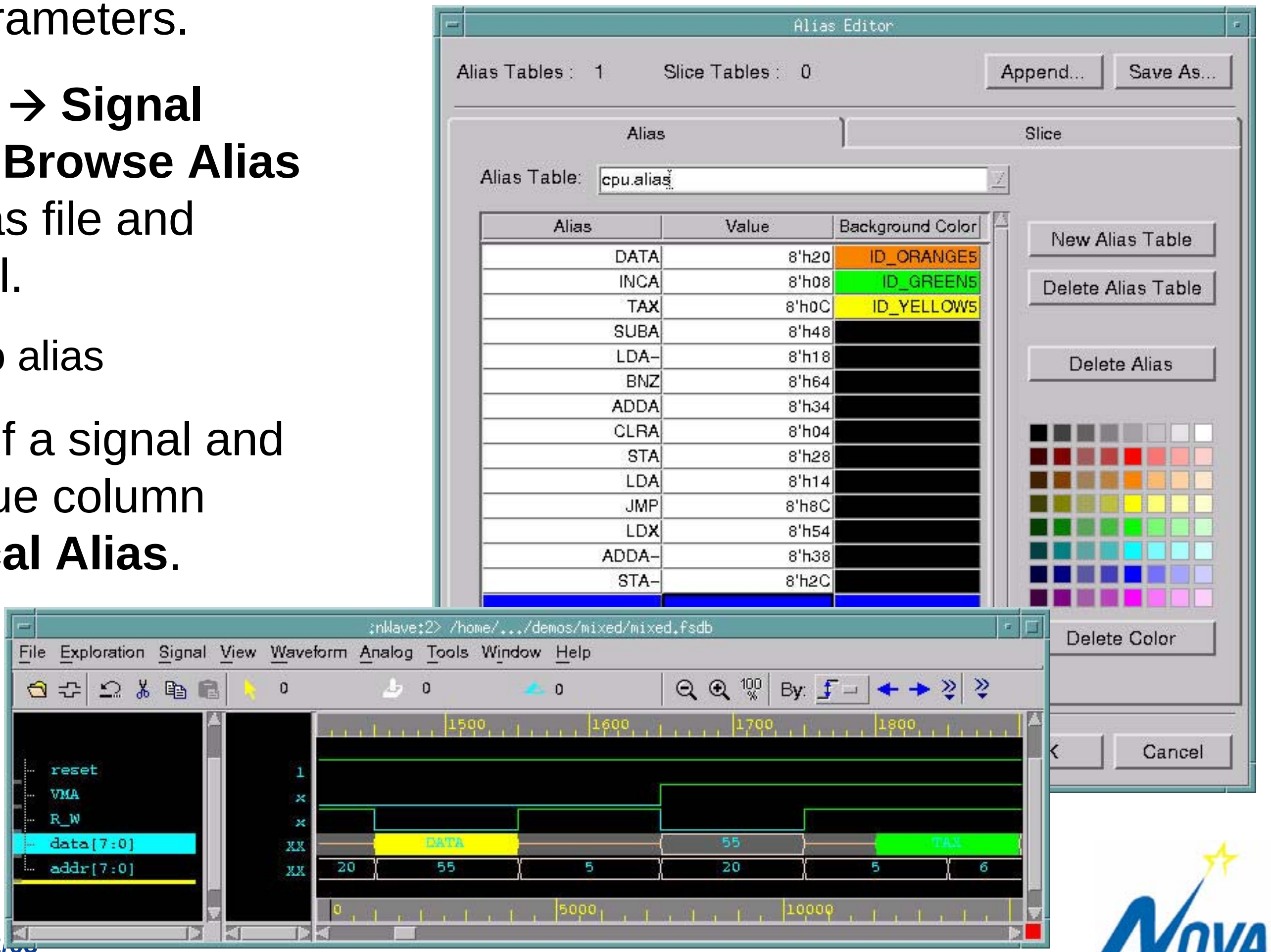
nWave – Hierarchical Groups

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


:nWave:2> /home/.../demos/mixed/mixed.fsdb
File Exploration Signal View Waveform Analog Tools Window Help
G1
X0[7:0]
Y0[7:0]
G1-2
VMA
R_W
reset
G1-1
carry_flag
carry_mode
G1-1-1
addr[7:0]
1500
1600
10000

nWave – Aliasing

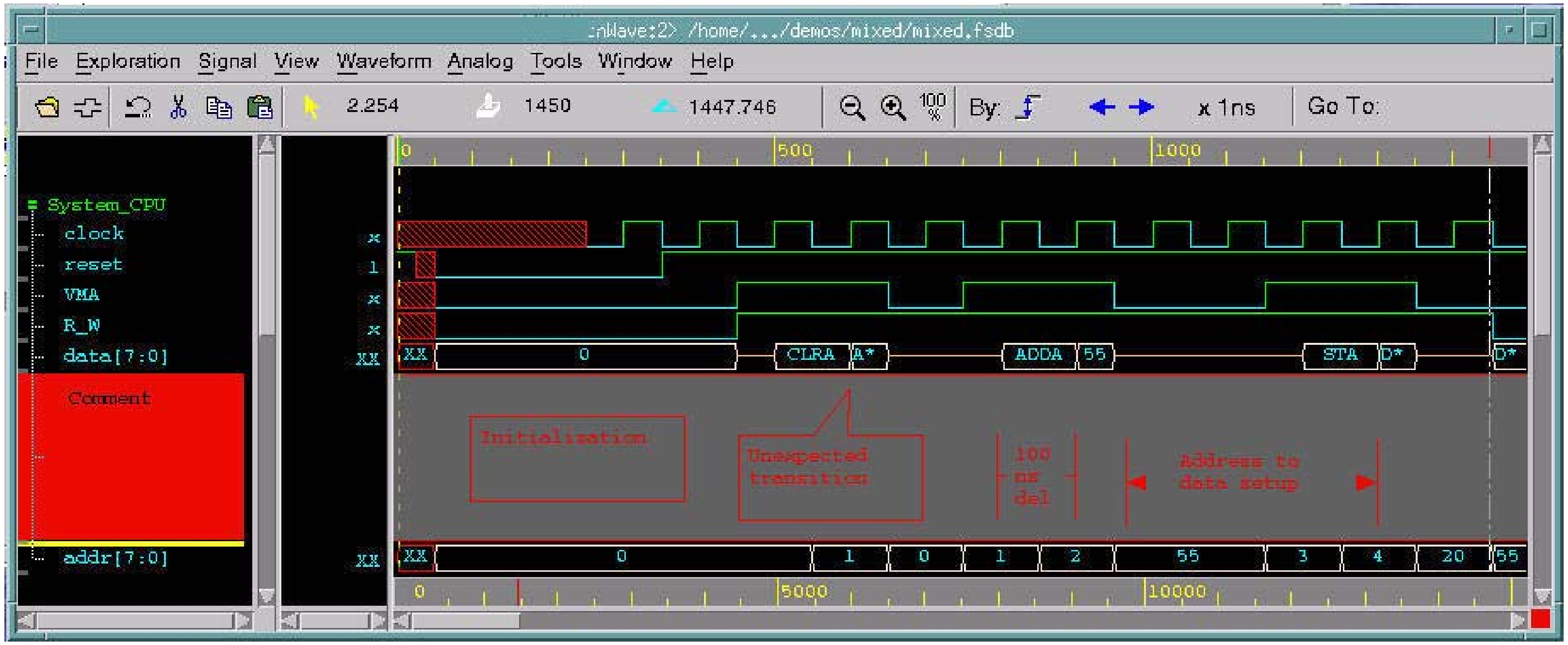
- Use –*autoalias* command line option for automatic mnemonic recognition for ‘defines and parameters.
- Use **Waveform → Signal Value Radix → Browse Alias** to create an alias file and apply to a signal.
  - Apply color to alias
- Create a copy of a signal and use RMB in value column to **Remove Local Alias.**



Alias	Value	Background Color
DATA	8'h20	ID_ORANGE5
INCA	8'h08	ID_GREEN5
TAX	8'h0C	ID_YELLOW5
SUBA	8'h48	■
LDA-	8'h18	■
BNZ	8'h64	■
ADDA	8'h34	■
CLRA	8'h04	■
STA	8'h28	■
LDA	8'h14	■
JMP	8'h8C	■
LDX	8'h54	■
ADDA-	8'h38	■
STA-	8'h2C	■

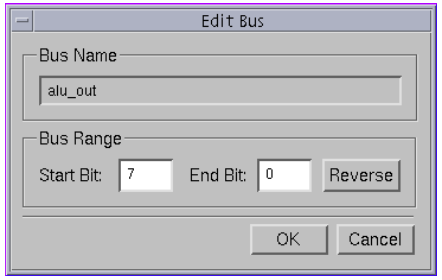
nWave – Add Comment

- Use **Signal → Comment** or RMB menu in signal window to insert a comment field at cursor position.
- Four comment box types provided: box, attached box, period and arrow period.



nWave – Bus Operations

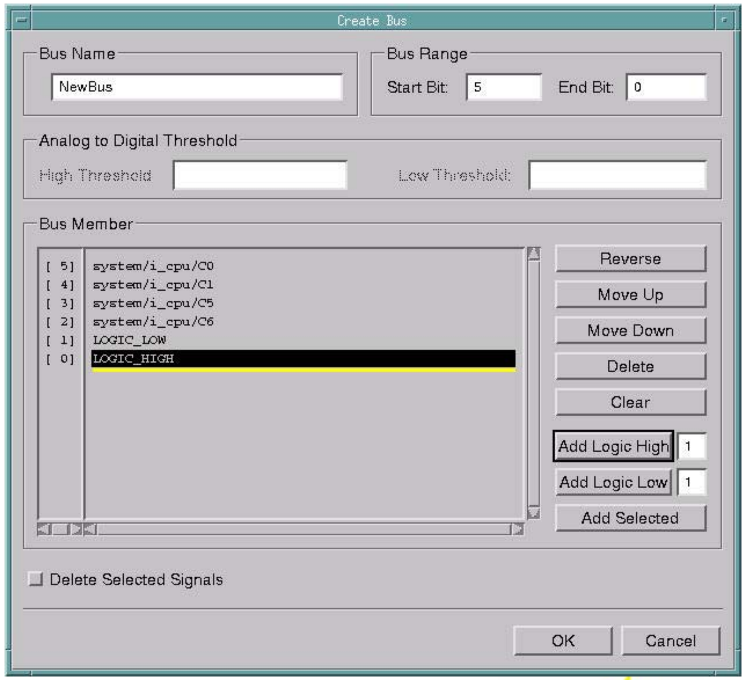
- Invoke **Signal → Edit Bus** or RMB menu in signal window to edit selected bus.
- Create bus from selected signals.
  - Invoke **Signal → Create Bus** or RMB menu in signal window.
    - Add Logic 0 or Logic 1 as place holders.
- Set radix and signal value notation.
  - **Waveform → Signal Value Radix → Binary, Octal, Hexadecimal, Decimal, ASCII, Alias**
  - **Waveform → Signal Value Notation → Unsigned, Signed 2's Complement, Signed 1's Complement**
- Expand / Collapse bus by double clicking.



Edit Bus

Bus Name: alu_out

Bus Range: Start Bit: 7 End Bit: 0



Create Bus

Bus Name: NewBus

Bus Range: Start Bit: 5 End Bit: 0

Analog to Digital Threshold: High Threshold: Low Threshold:

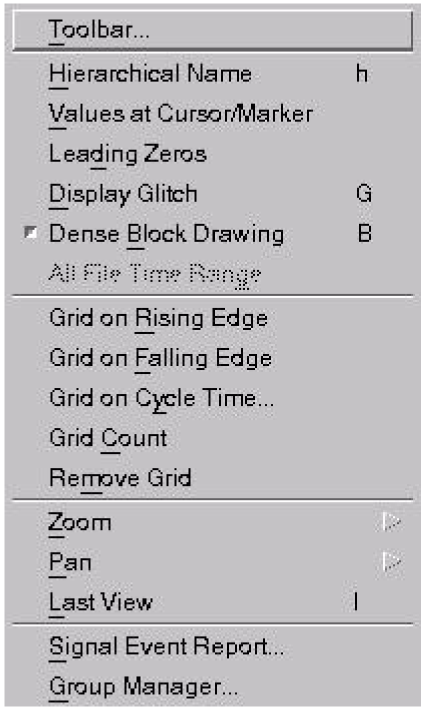
Bus Member

```
[ 5]  system/i_cpu/C0
[ 4]  system/i_cpu/C1
[ 3]  system/i_cpu/C5
[ 2]  system/i_cpu/C6
[ 1]  LOGIC_LOW
[ 0]  LOGIC_HIGH
```

Add Logic High: 1
Add Logic Low: 1
☐ Delete Selected Signals

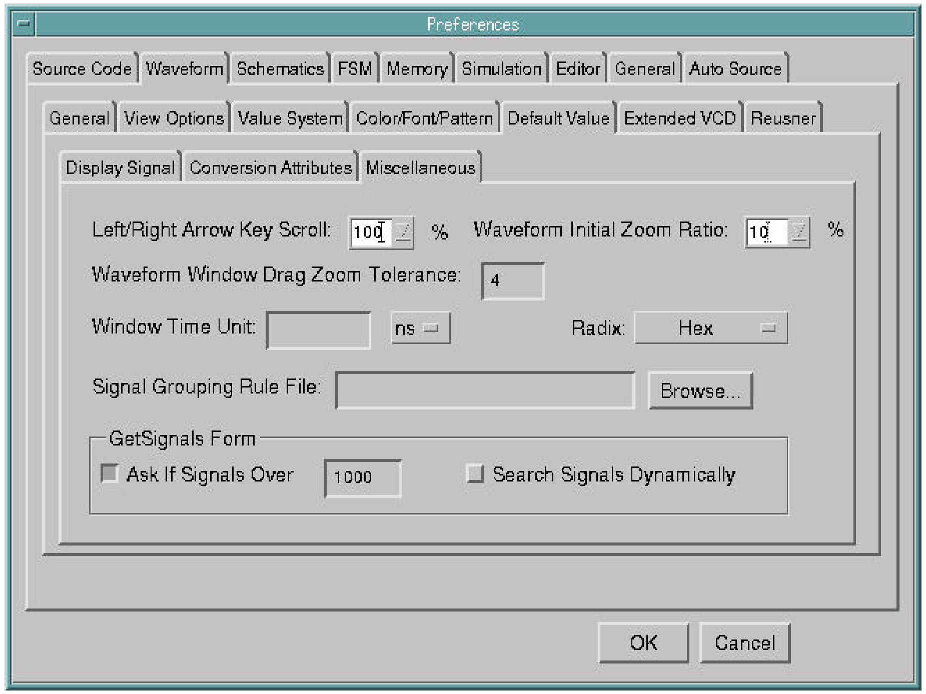
nWave – Display Options

- **View → Hierarchical Name** to display hierarchy before signals
- **View → Values at Cursor/Marker** to display cursor and marker values simultaneously.
- **View → Leading Zeros** to display leading zeros
- **View → Grid …** to display grid lines on rising/falling edges and a grid count.



nWave – Preferences

- Use **Tools → Preferences…** to customize waveform display. (Settings saved in novas.rc)
  - **General**: Display input/output signals in different colors, synchronize file browser, etc.
  - **View Options**: Enable highlight signal, radix display, value and signal alignment.
  - **Value System**: Change appearance based on values and strengths.
  - **Color / Font / Pattern**: Change appearance of windows and signal types.
  - **Default Value**: Change signal height and spacing. Change default time unit.
  - **Extended VCD**: Change appearance based on values and strengths.



Preferences

Source Code | Waveform | Schematics | FSM | Memory | Simulation | Editor | General | Auto Source

General | View Options | Value System | Color/Font/Pattern | Default Value | Extended VCD | Reusner

Display Signal | Conversion Attributes | Miscellaneous

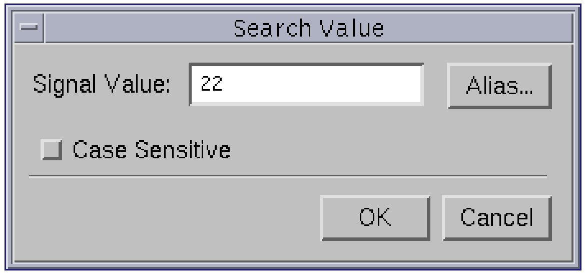
Left/Right Arrow Key Scroll: 100 %
Waveform Initial Zoom Ratio: 10 %
Waveform Window Drag Zoom Tolerance: 4
Window Time Unit: ns
Radix: Hex
Signal Grouping Rule File: Browse...

GetSignals Form
☐ Ask If Signals Over 1000
☐ Search Signals Dynamically

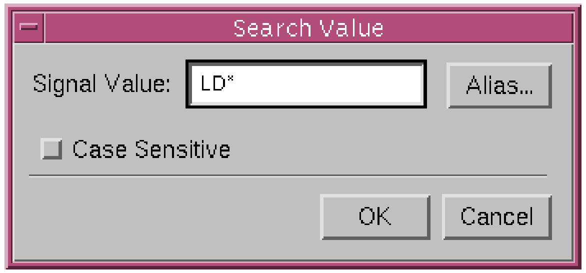
OK | Cancel

nWave – Search Value

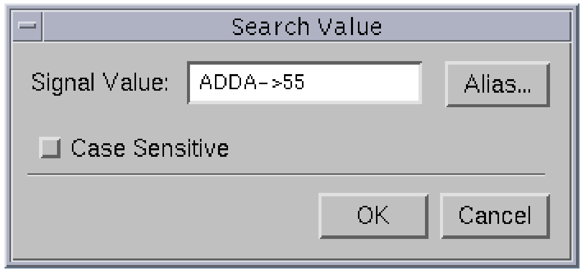
- Use **Waveform → Set Search Value…** to search signal values.
  - Search value accepts mnemonics, wild cards and transitions


Search Value

Signal Value: 22

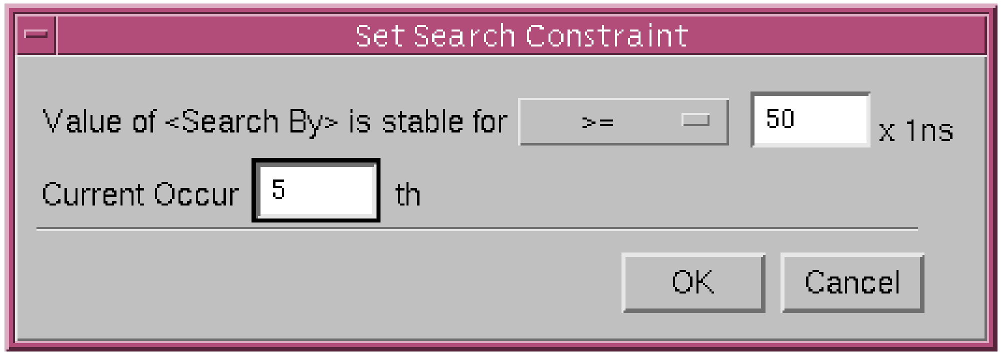

Search Value

Signal Value: LD*


Search Value

Signal Value: ADDA->55

- Use **Waveform → Set Search Constraint…** to constrain search to a duration of time or nth occurrence of a condition.

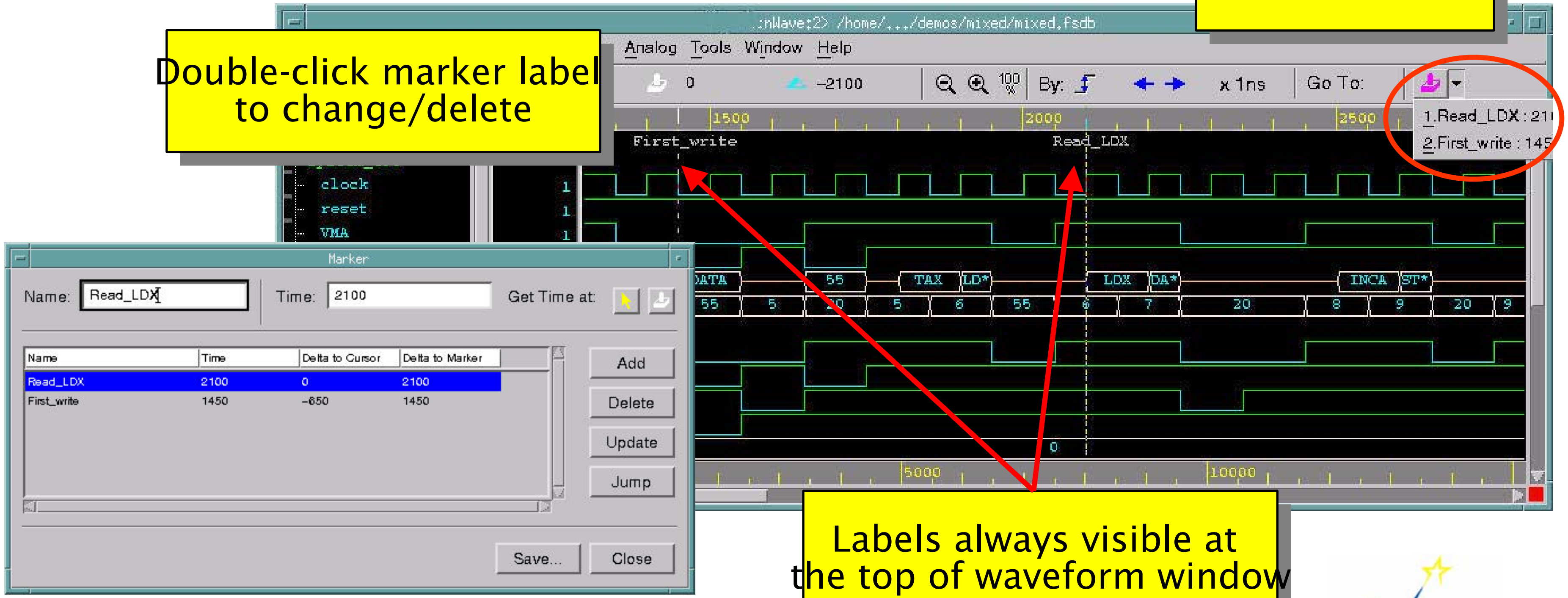

Set Search Constraint

Value of <Search By> is stable for >= 50 x 1ns

Current Occur 5 th

nWave – Markers

- Use **Waveform → Marker** to name and place markers.
- Multiple markers supported.
- Use distinct labels to differentiate markers



Jump to markers from tool bar

Double-click marker label to change/delete

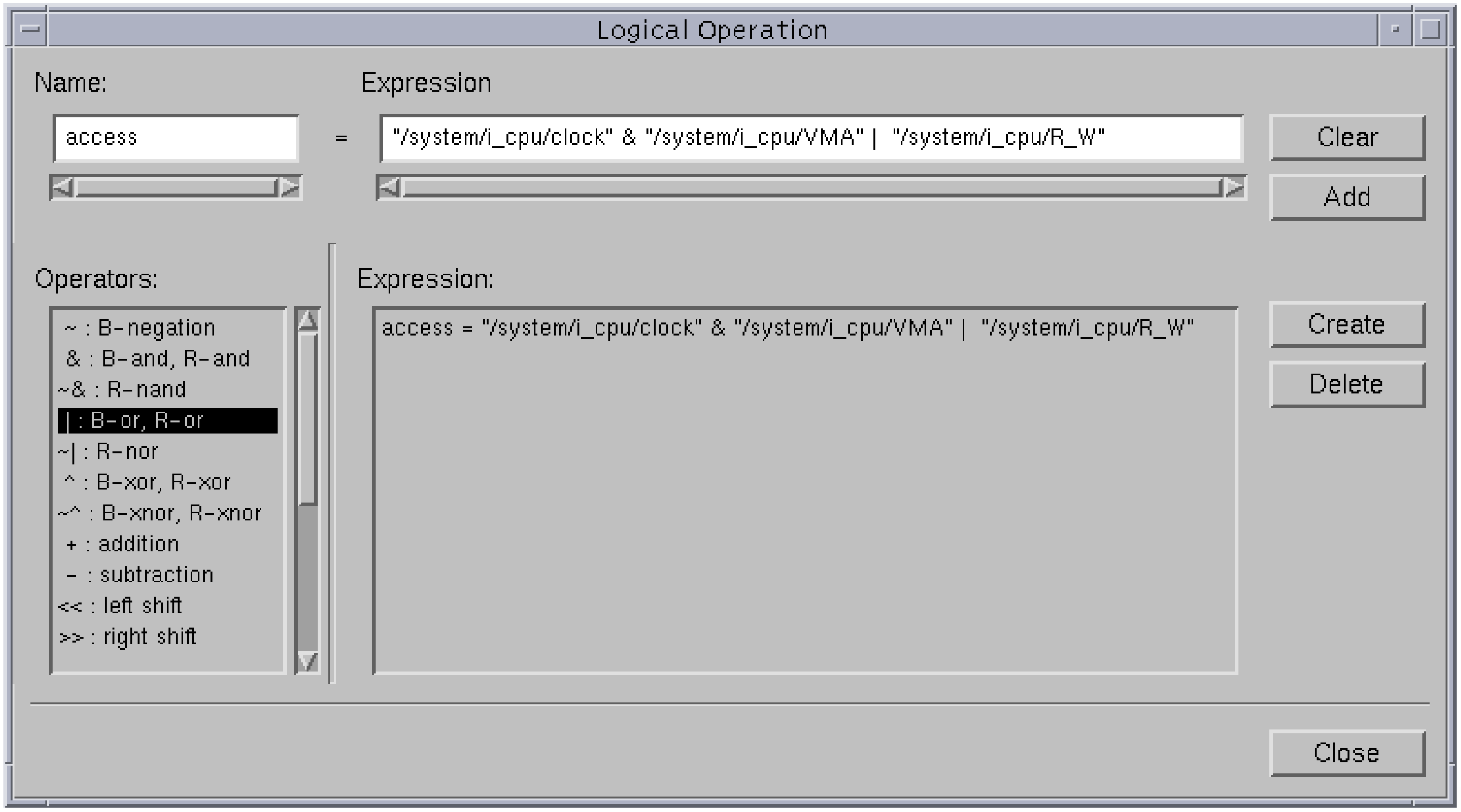
Labels always visible at the top of waveform window

Name: Read_LDX   Time: 2100

Name	Time	Delta to Cursor	Delta to Marker
Read_LDX	2100	0	2100
First_write	1450	-650	1450

nWave – Logical Operation

- Use **Signal → Logical Operation...** to create a new signal from other signals.



Name: access = Expression: "/system/i_cpu/clock" & "/system/i_cpu/VMA" | "/system/i_cpu/R_W"

Operators:

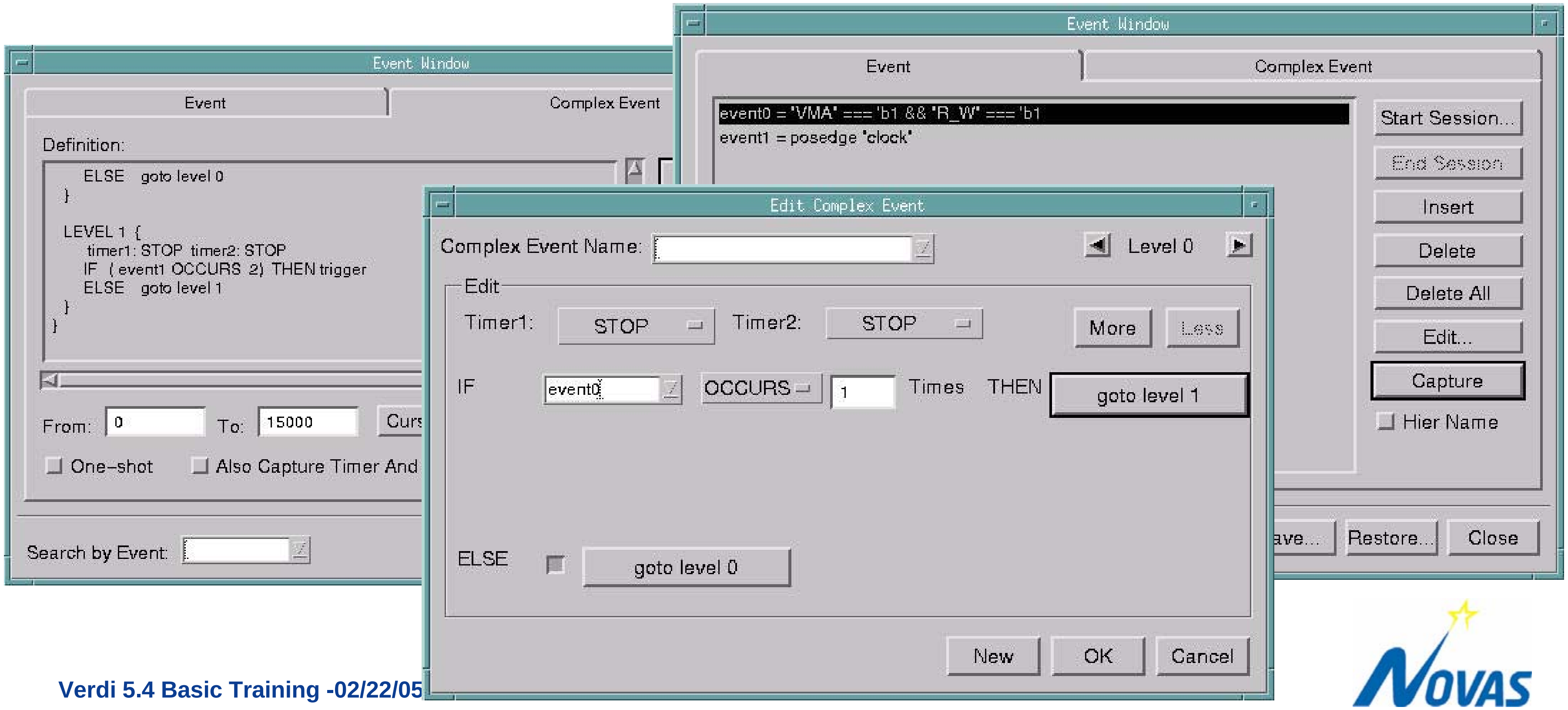
- ~ : B-negation
- & : B-and, R-and
- ~& : R-nand
- | : B-or, R-or
- ~| : R-nor
- ^ : B-xor, R-xor
- ~^ : B-xnor, R-xnor
- + : addition
- - : subtraction
- << : left shift
- >> : right shift

Expression:

access = "/system/i_cpu/clock" & "/system/i_cpu/VMA" | "/system/i_cpu/R_W"

nWave – Complex Event

- Use **Signal → Event** to define and name events.
  - Events are signal value combinations or complex sequences of single events.
- Double clicking on Event Name or Expression will invoke edit mode.



```
event0 = 'VMA' === 'b1 && 'R_W' === 'b1
event1 = posedge 'clock'
```

Definition:

```
    ELSE   goto level 0
  }

  LEVEL 1 {
    timer1: STOP  timer2: STOP
    IF  ( event1 OCCURS  2)  THEN trigger
    ELSE   goto level 1
  }
}
```

From: 0 To: 15000

Complex Event Name: Level 0

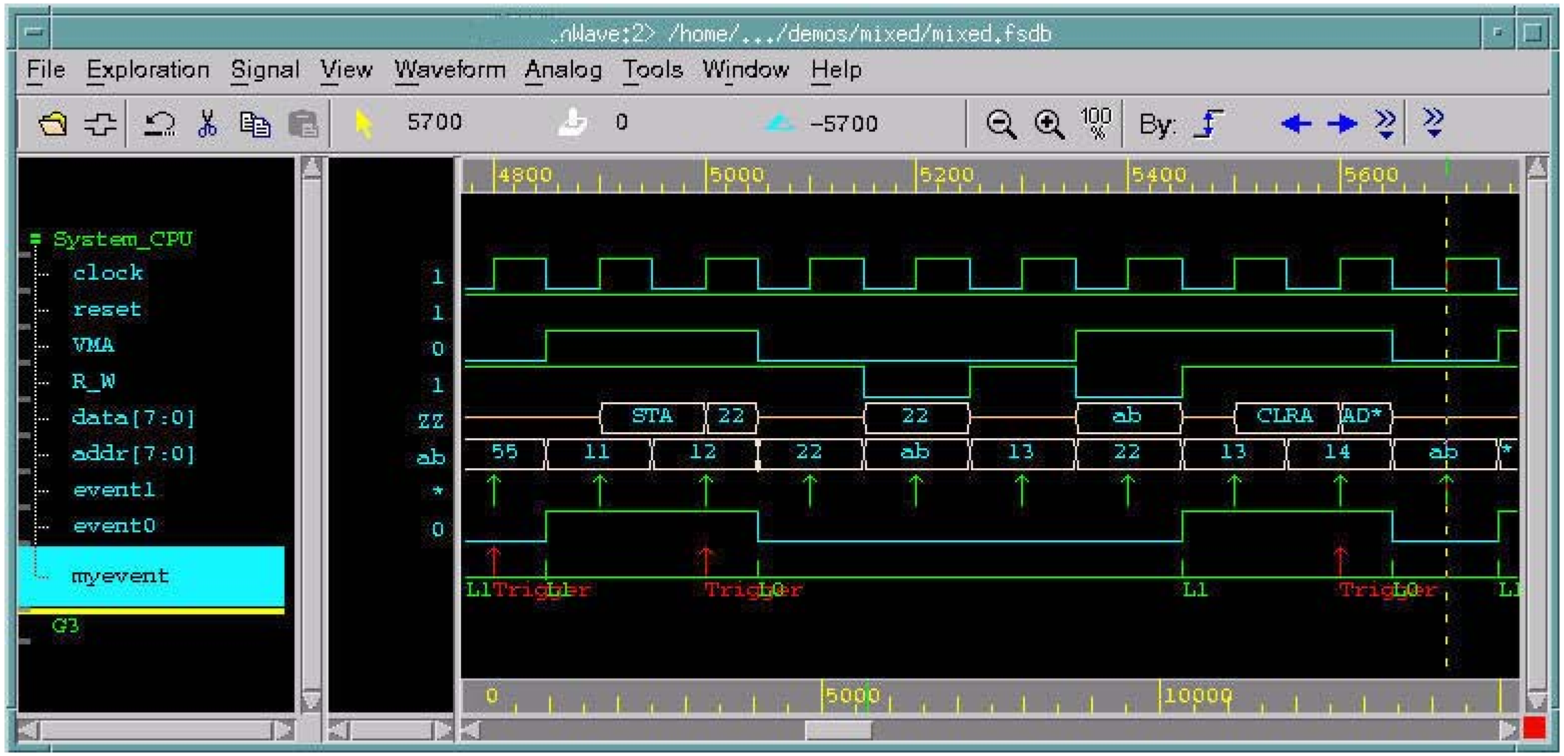
Timer1: STOP Timer2: STOP

IF event0 OCCURS 1 Times THEN goto level 1

ELSE goto level 0

nWave – Complex Event Waveform

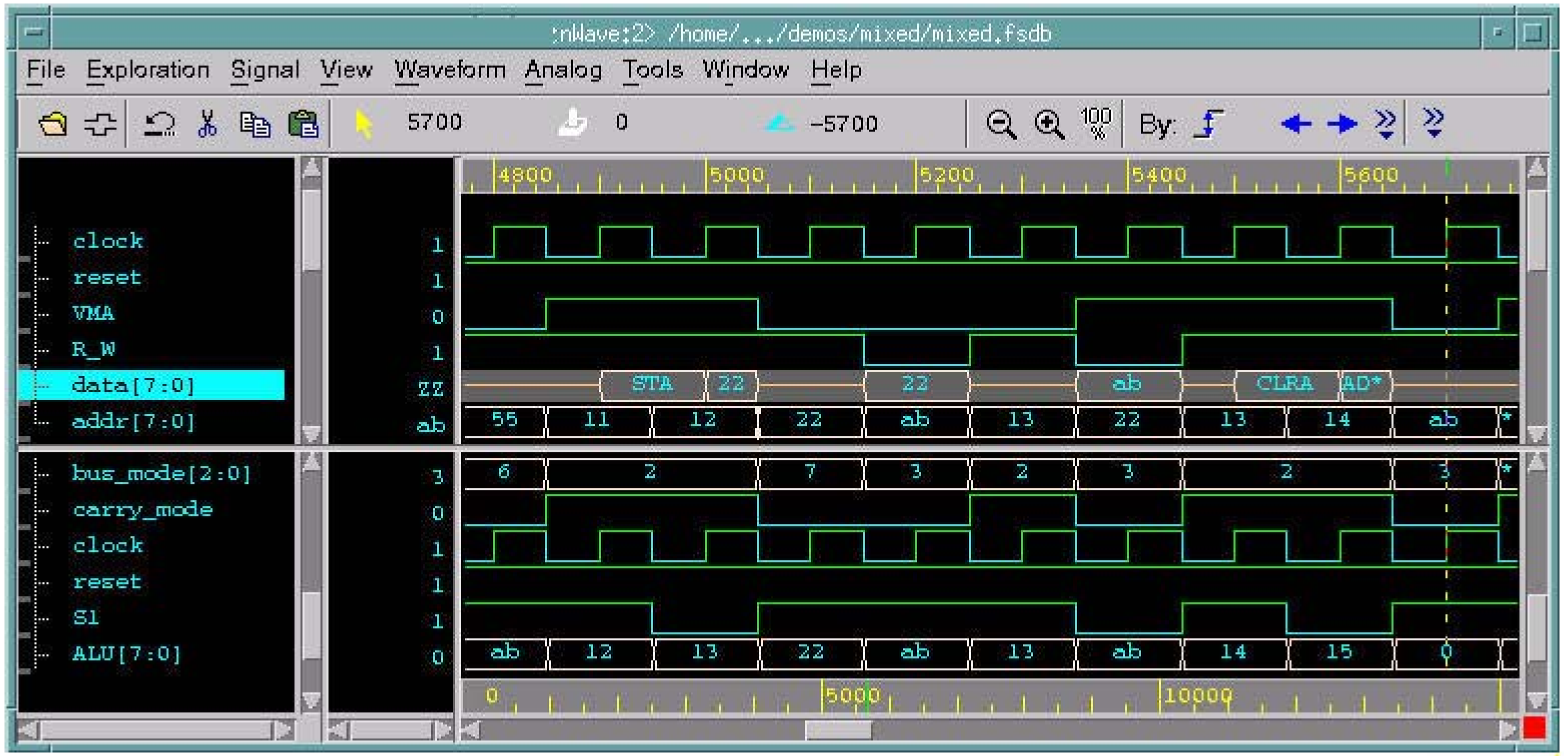
- Display captured event on waveform.



nWave:2> /home/.../demos/mixed/mixed.fsdb
File Exploration Signal View Waveform Analog Tools Window Help
5700
0
-5700
By:
4800
5000
5200
5400
5600
System_CPU
clock 1
reset 1
VMA 0
R_W 1
data[7:0] zz
addr[7:0] ab
event1 *
event0 0
myevent
G3
STA 22 22 ab CLRA AD*
55 11 12 22 ab 13 22 13 14 ab
L1 Trigger L0 Trigger L0 L1 Trigger L0 L1
0
5000
10000

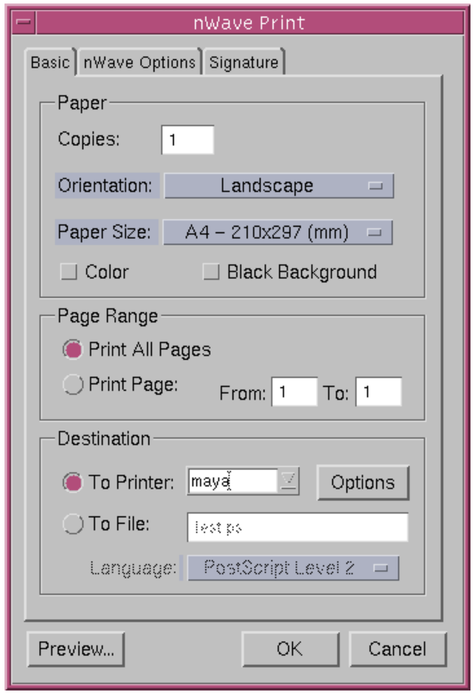
nWave – Split Window

- Use **Window → Horizontal Split** to split window.
- Use **Window → Stop Split** to go back.



nWave – Print

- **File → Print...** command to print out what you see in the window with a PostScript printer.
- Select to print to file or to printer.
- Support print selected signals or displayed signals.



nWave Print

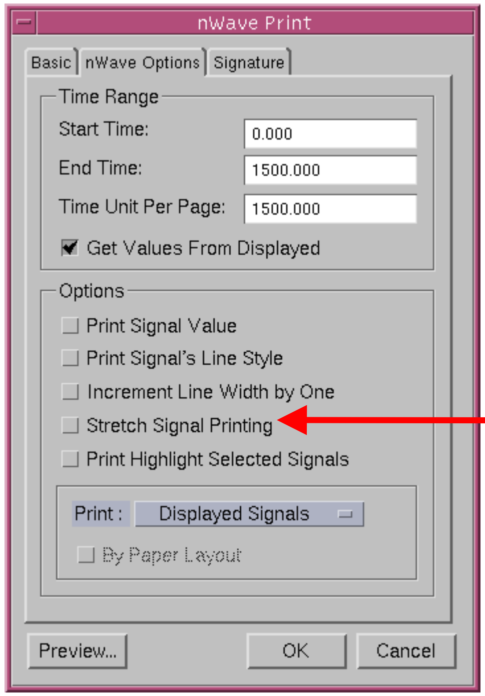
Basic | nWave Options | Signature

Paper
Copies: 1
Orientation: Landscape
Paper Size: A4 – 210x297 (mm)
☐ Color ☐ Black Background

Page Range
[✓ Print All Pages]
Print Page: From: 1 To: 1

Destination
[✓ To Printer:] maya
Options
To File: test.ps
Language: PostScript Level 2

Preview... OK Cancel



nWave Print

Basic | nWave Options | Signature

Time Range
Start Time: 0.000
End Time: 1500.000
Time Unit Per Page: 1500.000
☑ Get Values From Displayed

Options
☐ Print Signal Value
☐ Print Signal's Line Style
☐ Increment Line Width by One
☐ Stretch Signal Printing
☐ Print Highlight Selected Signals

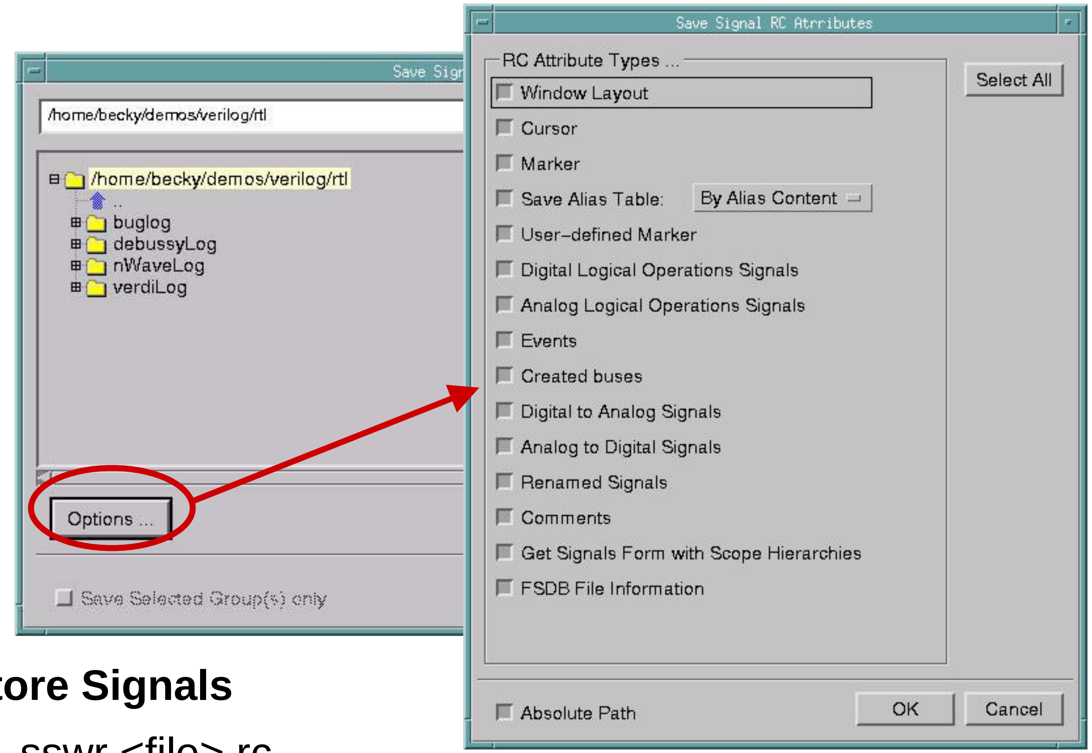
Print : Displayed Signals
☐ By Paper Layout

Preview... OK Cancel

Stretch signals to fill whole page.

nWave – Save and Restore Signals

- Use **File → Save Signals** to save all waveform objects including:
  - Signals
  - Comments
  - Color settings
  - Markers and labels
  - Logical operations
  - Events
  - Created Buses


Save Sig
/home/becky/demos/verilog/rtl
/home/becky/demos/verilog/rtl
..
buglog
debussyLog
nWaveLog
verdiLog
Options ...
Save Selected Group(s) only
Save Signal RC Atrributes
RC Attribute Types ...
Select All
Window Layout
Cursor
Marker
Save Alias Table:
By Alias Content
User-defined Marker
Digital Logical Operations Signals
Analog Logical Operations Signals
Events
Created buses
Digital to Analog Signals
Analog to Digital Signals
Renamed Signals
Comments
Get Signals Form with Scope Hierarchies
FSDB File Information
Absolute Path
OK
Cancel

- Restore the signals saved in a previous session
  - In *nWave*, use **File → Restore Signals**
  - On the command line, use –sswr <file>.rc

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

nWave – Additional Commands

- Logical Operations
  - **Signal → Logical Operation...**
- Logic analyzer functions (Complex Events)
  - **Signal → Event**
- Shift the time of selected signals.
  - **Waveform → Waveform Time → Shift Individual Signal Time…**
- Create independently scrollable signal frames.
  - **Window → Horizontal Split**
- Aliasing
  - **Waveform → Signal Value Radix → Browse Alias**

NOVAS

nWave – Summary

- At this time you should
  - Understand the *nWave* functionality.
  - Be able to open an FSDB file and add signals to the waveform display.
  - Know how to create buses and search for a particular value.
  - Be able to add markers, comments, aliases and logical operations.
  - Know how to compare simulation results.
  - Know how to view the analog waveform
  - Be able to access and change the preferences.

nWave – Wrap-up

- Questions?

- Take 20 minutes to complete the *nWave* LAB

Training Course

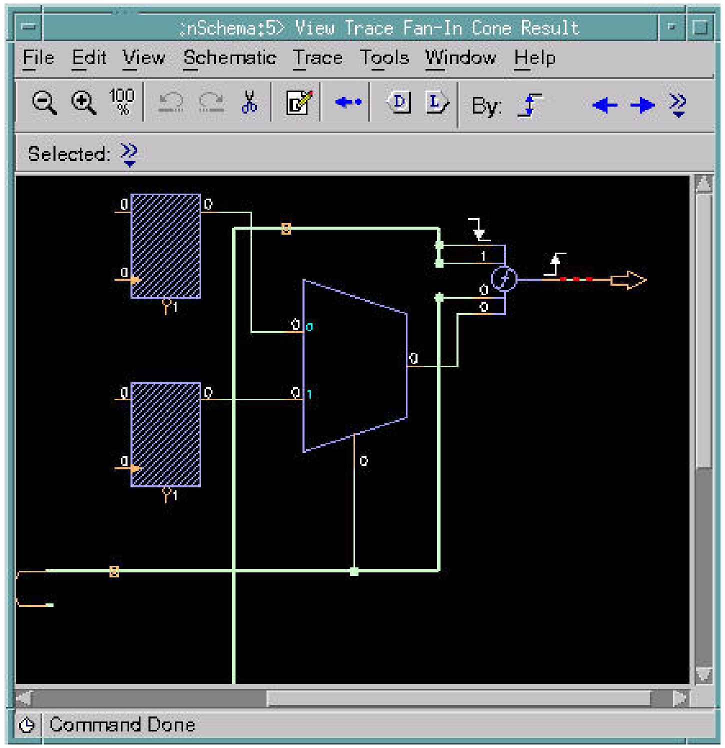
- Verdi Background and Overview
- Loading Your Design
- nTrace – View and Debug Source Code
- nSchema – View and Debug Designs Graphically
- nState – View and Analyze State Machines
- nWave – View Simulation Results in Waveforms
- Debugging with Verdi

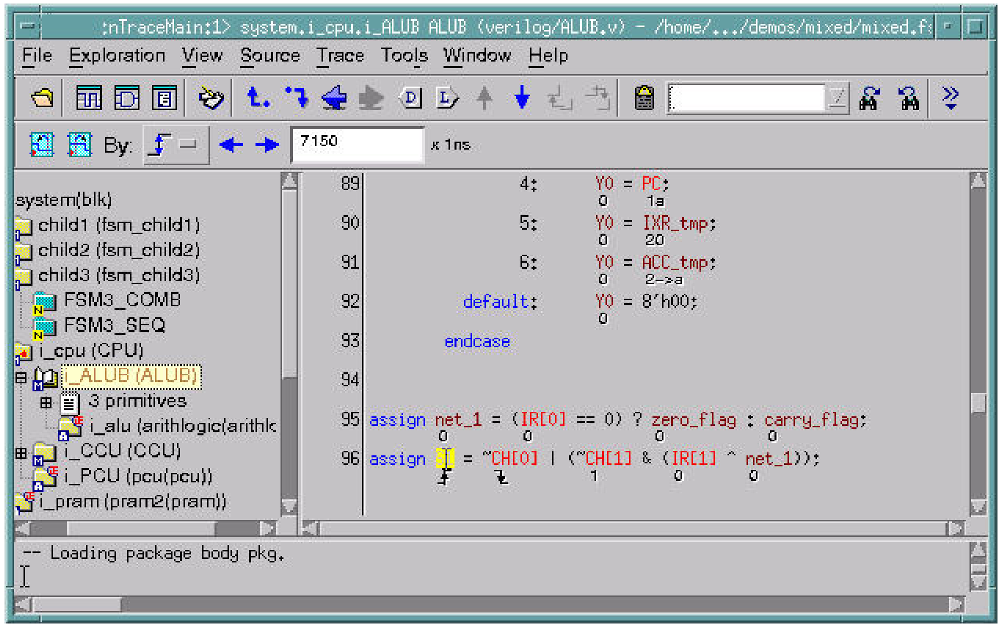
Debugging With Verdi

- Alternate Ways to View Simulation Data
  - Active Annotation
  - State Annotation
  - Complex Signal Browser
  - Watch, Memory or MDA Windows
- Isolate Logic Relevant to a Specific Simulation Time
  - Active Trace
  - Trace X
- Post Process Simulation Data
  - List X
- Save Debug Environment
- Lab

Active Annotation

- Display simulation results on source or schematic.
  - **Source → Active Annotation** in *nTrace*
  - **Schematic → Active Annotation** in *nSchema*.
- Signal values / transitions are synchronized with cursor time.


:nSchema:5> View Trace Fan-In Cone Result
File Edit View Schematic Trace Tools Window Help
By:
Selected:
Command Done



```
                  4:        Y0 = PC;
                            0    1a
                  5:        Y0 = IXR_tmp;
                            0    20
                  6:        Y0 = ACC_tmp;
                            0    2->a
             default:       Y0 = 8'h00;
                            0
          endcase

assign net_1 = (IR[0] == 0) ? zero_flag : carry_flag;
         0         0              0           0
assign S1 = ~CH[0] | (~CH[1] & (IR[1] ^ net_1));
                          1         0        0
```

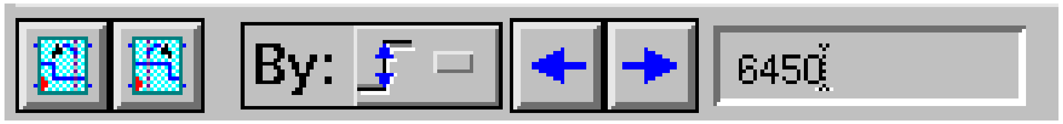
```
-- Loading package body pkg.
```

Active Annotation – Stepping through Time

- Select signal(s) in the source or schematic window and go to next or previous transitions
  - The transition is selectable between: ***Any Change*** ***Rising*** or ***Falling***

- In *nTrace*, search for the previous/next transition for all signals in the selected statement.

- The above commands will move the cursor time in all windows.



Complex Signal Browser

- Dedicated browser for visualizing complex signals
- What are complex signals?
  - SystemVerilog struct
  - MDAs
  - VHDL record type signals
- Invoke the browser from *nTrace*, *nSchema*, or *nWave*
  - **Tools → Browse Structure Signal…**
  - **RMB → Browse Structure Signal…** (*nTrace* only)

Complex Signal Browser

Hierarchy Signal Tree Window

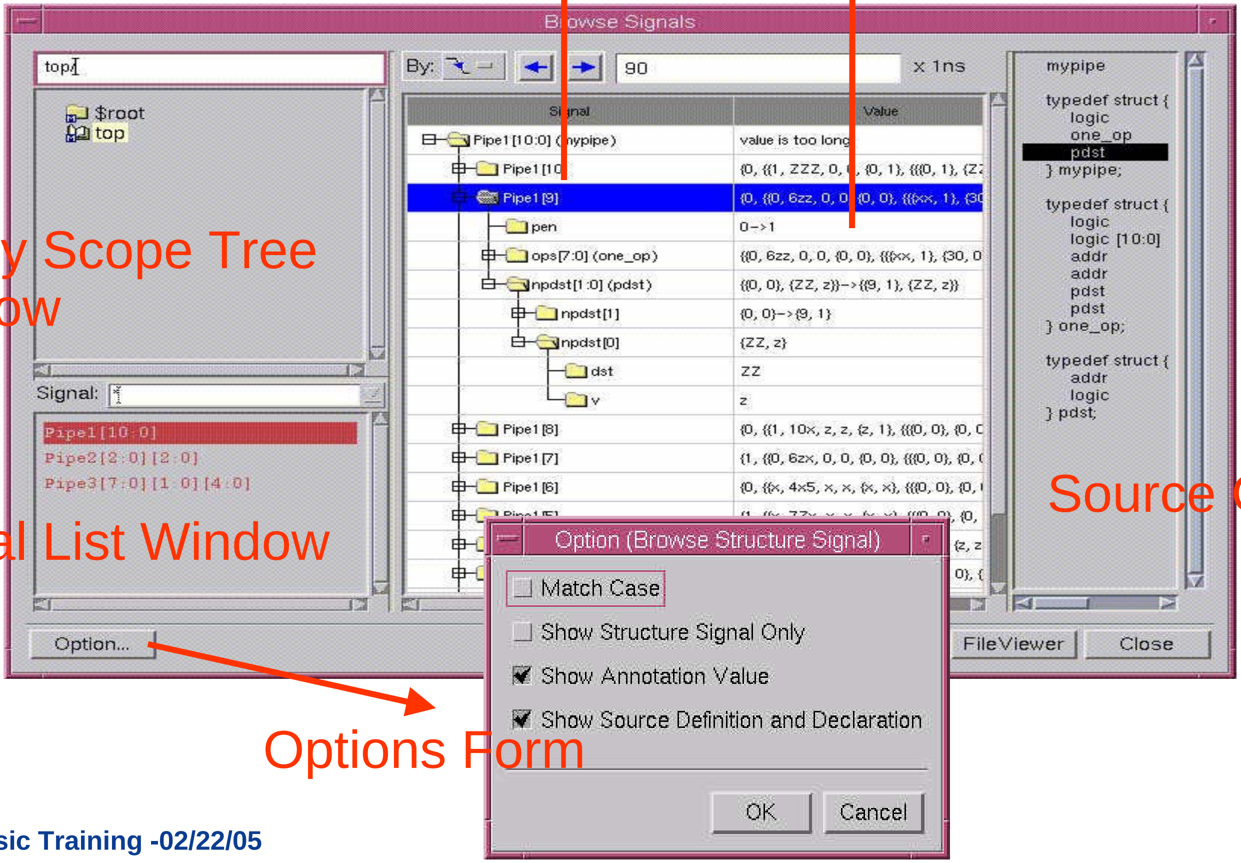
Value Annotation

Hierarchy Scope Tree Window

Signal List Window

Source Code Window

Options Form



Pipe1[10:0]
Pipe2[2:0][2:0]
Pipe3[7:0][1:0][4:0]

Signal	Value
Pipe1[10:0] (mypipe)	value is too long
Pipe1[10	{0, {{1, ZZZ, 0, , {0, 1}, {{{0, 1}, {Z
Pipe1[9]	{0, {{0, 6zz, 0, 0 {0, 0}, {{{xx, 1}, {30
pen	0->1
ops[7:0] (one_op)	{{0, 6zz, 0, 0, {0, 0}, {{{xx, 1}, {30, 0
npdst[1:0] (pdst)	{{0, 0}, {ZZ, z}}->{{9, 1}, {ZZ, z}}
npdst[1]	{0, 0}->{9, 1}
npdst[0]	{ZZ, z}
dst	ZZ
v	z
Pipe1[8]	{0, {{1, 10x, z, z, {z, 1}, {{{0, 0}, {0, C
Pipe1[7]	{1, {{0, 6zx, 0, 0, {0, 0}, {{{0, 0}, {0, (
Pipe1[6]	{0, {{x, 4x5, x, x, {x, x}, {{{0, 0}, {0, (

```
mypipe

typedef struct {
    logic
    one_op
    pdst
} mypipe;

typedef struct {
    logic
    logic [10:0]
    addr
    addr
    pdst
    pdst
} one_op;

typedef struct {
    addr
    logic
} pdst;
```

☐ Match Case
☐ Show Structure Signal Only
☑ Show Annotation Value
☑ Show Source Definition and Declaration

Complex Signal Browser

- General
  - Multiple *Complex Signal Browser* windows can be invoked.
  - D&D is supported between *Complex Signal Browser* and other windows.
- Hierarchical Scope Window
  - Same as *nTrace* but without package nodes
- Signal List Window
  - Supports regular expression in search window
  - Complex signals will be shown with ***Red*** color
  - Signals are sorted alphabetically
  - Two options **Match Case** and **Show Structure Signal Only** in *Option* form will be applied to this window.

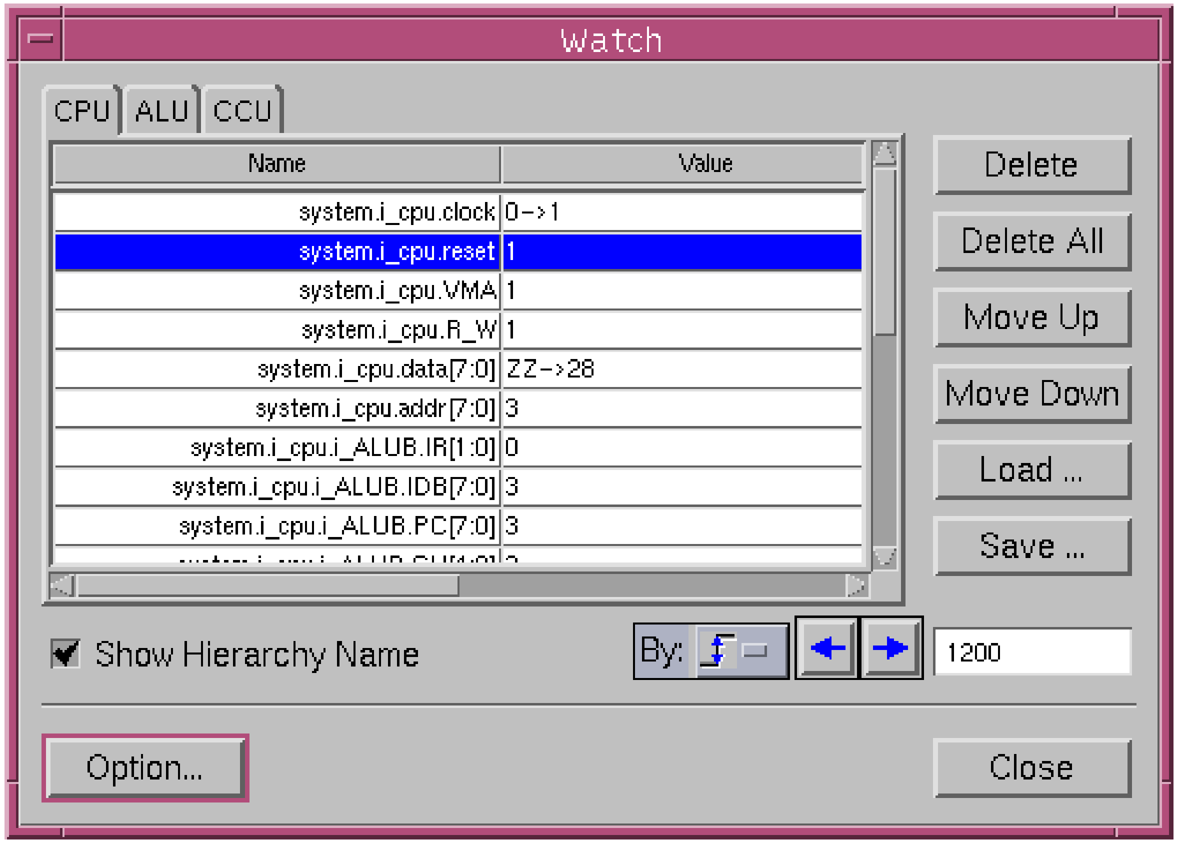
NOVAS

Complex Signal Browser

- Hierarchical Signal Tree Window
  - The signal tree will be built when selecting a signal in *Signal List Window*
  - Settings for **Source → Signal Value Radix** and **Source → Signal Value Notation** in *nTrace* will be applied to this window.
  - If an FSDB is loaded, the annotation value will be shown for all leaf nodes.
  - The *Value Annotation Window* will be closed if **Show Annotation Value** option is turned off in the *Option* form.
  - The annotated value will change according to current time point.
- Source Code Window
  - Enabled/disabled with the **Show Source Definition and Declaration** option in the *Option* form.

Watch Signal

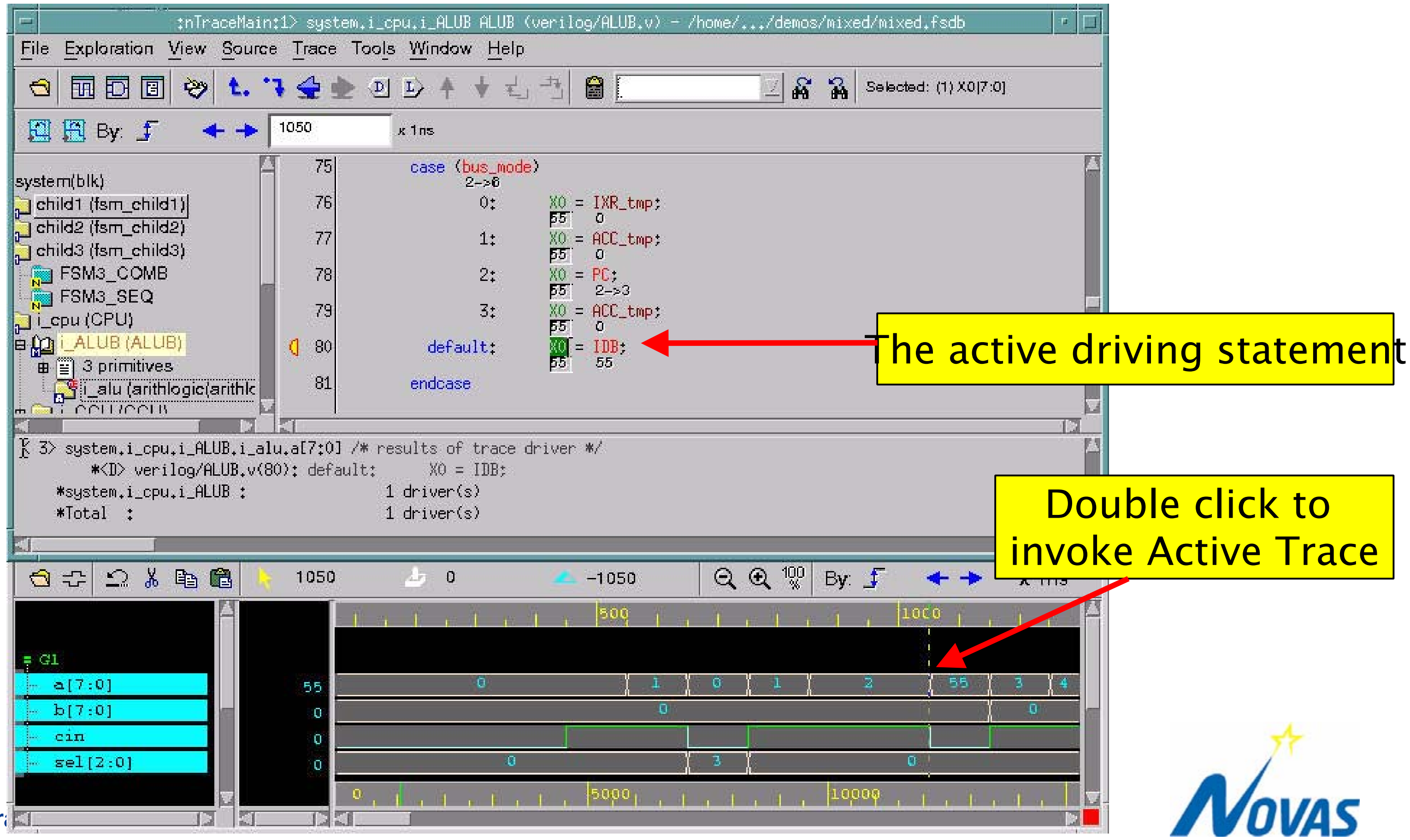
- From *nTrace*, use **Tools → Watch Signal** to view the value of selected signals in a table.



Name	Value
system.i_cpu.clock	0->1
system.i_cpu.reset	1
system.i_cpu.VMA	1
system.i_cpu.R_W	1
system.i_cpu.data[7:0]	ZZ->28
system.i_cpu.addr[7:0]	3
system.i_cpu.i_ALUB.IR[1:0]	0
system.i_cpu.i_ALUB.IDB[7:0]	3
system.i_cpu.i_ALUB.PC[7:0]	3

Active Trace from Waveform

- Double clicking on a signal transition in the waveform window will identify ONLY the active driving statements(s) in *nTrace* causing that transition.



```
75          case (bus_mode)
              2->6
76             0:       X0 = IXR_tmp;
                        55    0
77             1:       X0 = ACC_tmp;
                        55    0
78             2:       X0 = PC;
                        55    2->3
79             3:       X0 = ACC_tmp;
                        55    0
80          default:    X0 = IDB;
                        55    55
81          endcase
```

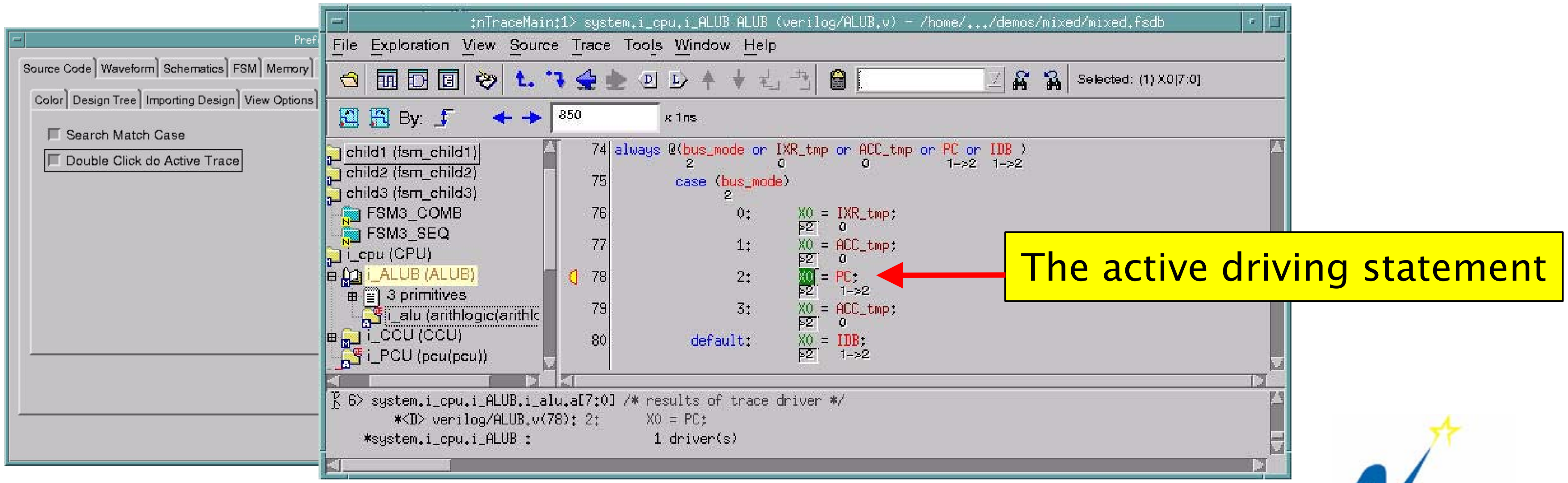
```
3> system.i_cpu.i_ALUB.i_alu.a[7:0] /* results of trace driver */
      *<D> verilog/ALUB.v(80): default:      X0 = IDB;
   *system.i_cpu.i_ALUB :                  1 driver(s)
   *Total   :                              1 driver(s)
```

The active driving statement

Double click to invoke Active Trace

Active Trace in Source

- Used to identify the active driver for the selected signal.
  - Requires design and simulation results.
  - Select signal in *nTrace*.
  - Invoke RMB menu and select **Active Trace**.
    - Ctrl-t also performs an Active Trace.
    - Use **Tools → Preferences**, **Source Code** tab, **Miscellaneous** tab, to enable **Double Click do Active Trace**.



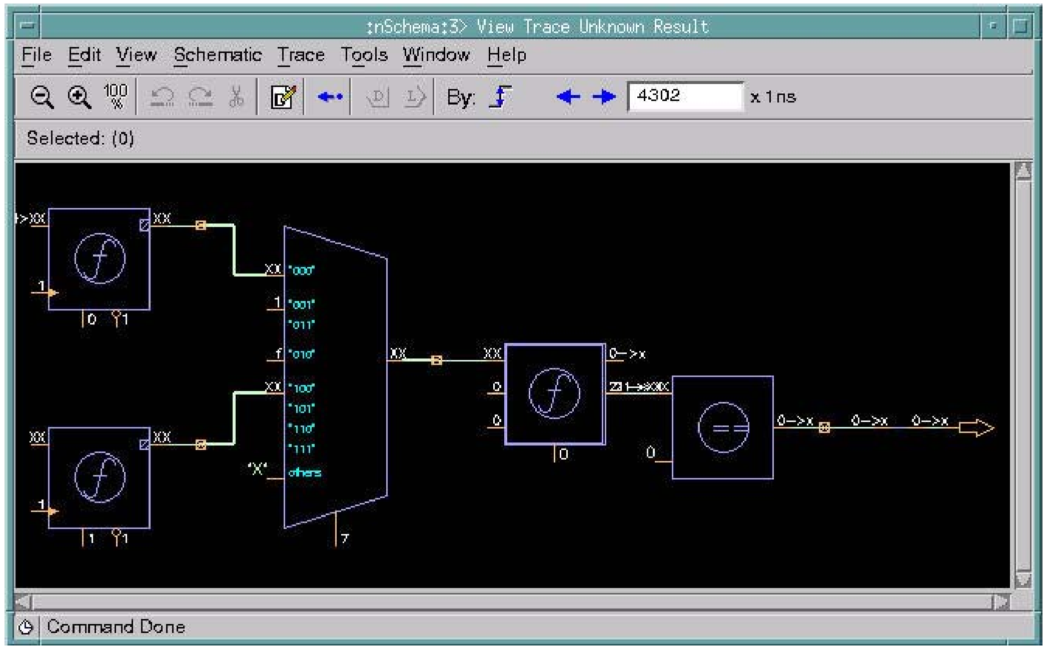
Pref
Source Code | Waveform | Schematics | FSM | Memory
Color | Design Tree | Importing Design | View Options
Search Match Case
Double Click do Active Trace
:nTraceMain:1> system.i_cpu.i_ALUB ALUB (verilog/ALUB.v) - /home/.../demos/mixed/mixed.fsdb
File Exploration View Source Trace Tools Window Help
Selected: (1) X0[7:0]
By:
850
x 1ns
child1 (fsm_child1)
child2 (fsm_child2)
child3 (fsm_child3)
FSM3_COMB
FSM3_SEQ
i_cpu (CPU)
i_ALUB (ALUB)
3 primitives
i_alu (arithlogic(arithlc
i_CCU (CCU)
i_PCU (pcu(pcu))
The active driving statement

```
74 always @(bus_mode or IXR_tmp or ACC_tmp or PC or IDB )
              2           0          0        1->2  1->2
75      case (bus_mode)
              2
76           0:       X0 = IXR_tmp;
                      P2    0
77           1:       X0 = ACC_tmp;
                      P2    0
78           2:       X0 = PC;
                      P2    1->2
79           3:       X0 = ACC_tmp;
                      P2    0
80     default:       X0 = IDB;
                      P2    1->2
```

```
6> system.i_cpu.i_ALUB.i_alu.a[7:0] /* results of trace driver */
       *<D> verilog/ALUB.v(78): 2:       X0 = PC;
   *system.i_cpu.i_ALUB :                 1 driver(s)
```

Trace X

- Display logic cone relevant to the unknown.
  - In *nWave*, select a signal.
  - Place a cursor at the transition to X.
  - Invoke **Tools → Trace X**.
    - Generates a flattened schematic that shows the X propagation path back to first storage element or primary input.
  - In resulting *nSchema*, **Trace → Trace X**.
  - Repeat as needed.


:nSchema:3> View Trace Unknown Result
File Edit View Schematic Trace Tools Window Help
By:
4302
x 1ns
Selected: (0)
Command Done

List X

- Two common causes of X's:
  - Timing violations
  - Un-initialized storage elements

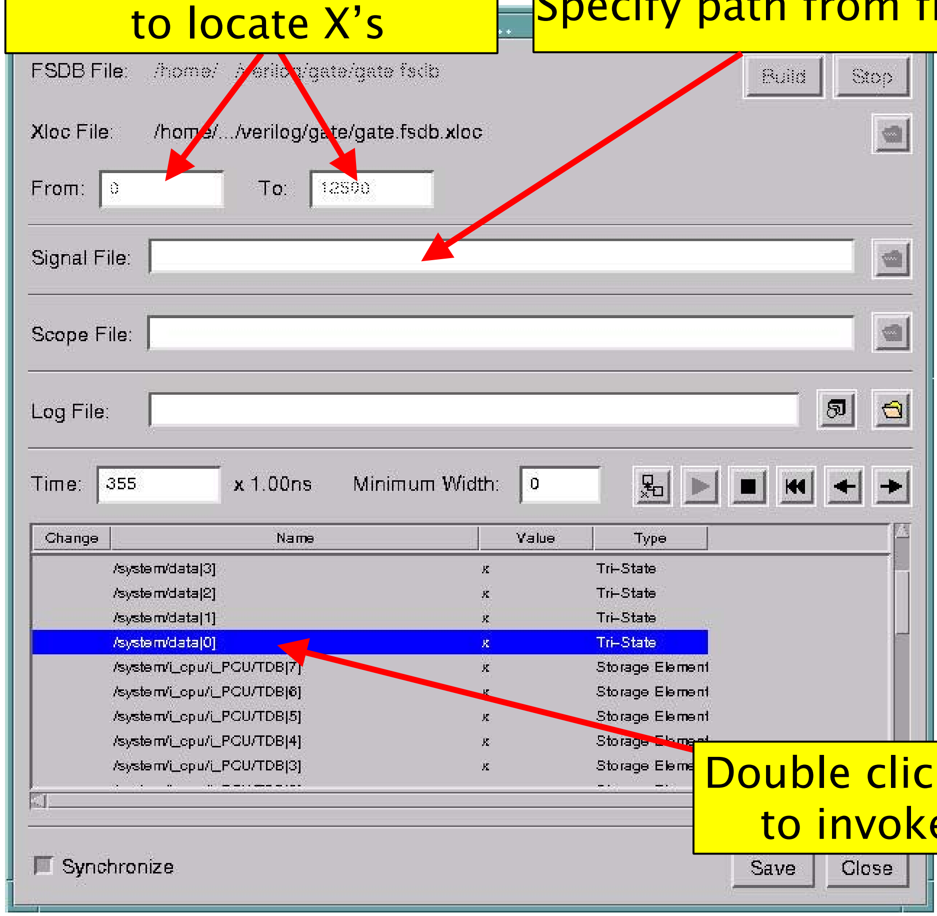
    verdi can help to isolate storage elements and tri-state devices with unknowns.
- In *nWave* or *nTrace*, use **Tools → List X**.
  - Build: parses fsdb file to find storage elements and tri-state devices which have value X.
  - A batch build executable called ***xloc*** is supported.
  - Play: to list storage elements and tri-state devices which have value X at selected time.
- Double click on a signal in the list or press Trace X icon to generate a flattened schematic (Trace X functionality).

NOVAS

List X / Trace X Example

Specify time range to locate X's

Specify path from file



FSDB File: /home/.../verilog/gate/gate.fsdb

Xloc File: /home/.../verilog/gate/gate.fsdb.xloc

From: 0 To: 12500

Signal File:

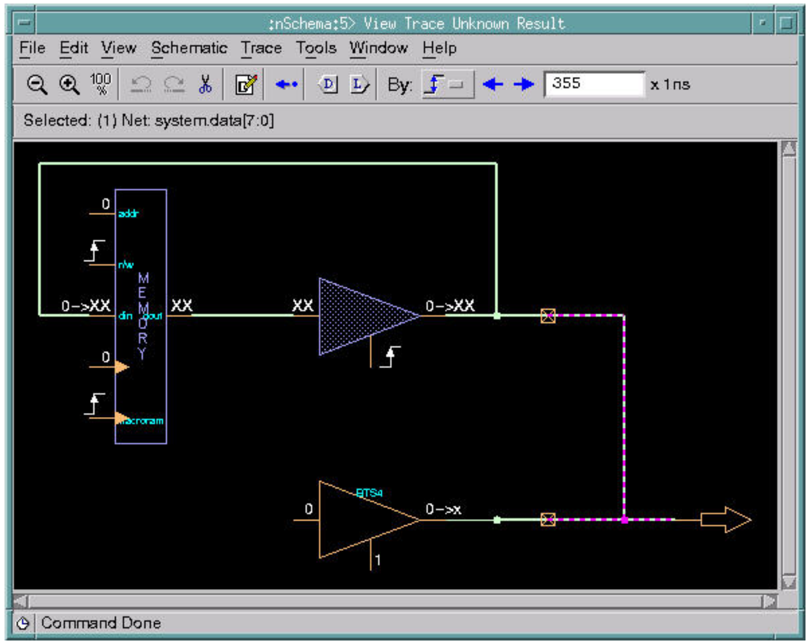
Scope File:

Log File:

Time: 355 x 1.00ns Minimum Width: 0

Change	Name	Value	Type
	/system/data[3]	x	Tri-State
	/system/data[2]	x	Tri-State
	/system/data[1]	x	Tri-State
	/system/data[0]	x	Tri-State
	/system/i_cpu/i_PCU/TDB[7]	x	Storage Element
	/system/i_cpu/i_PCU/TDB[6]	x	Storage Element
	/system/i_cpu/i_PCU/TDB[5]	x	Storage Element
	/system/i_cpu/i_PCU/TDB[4]	x	Storage Element
	/system/i_cpu/i_PCU/TDB[3]	x	Storage Element

Synchronize



:nSchema:5> View Trace Unknown Result

Selected: (1) Net: system.data[7:0]

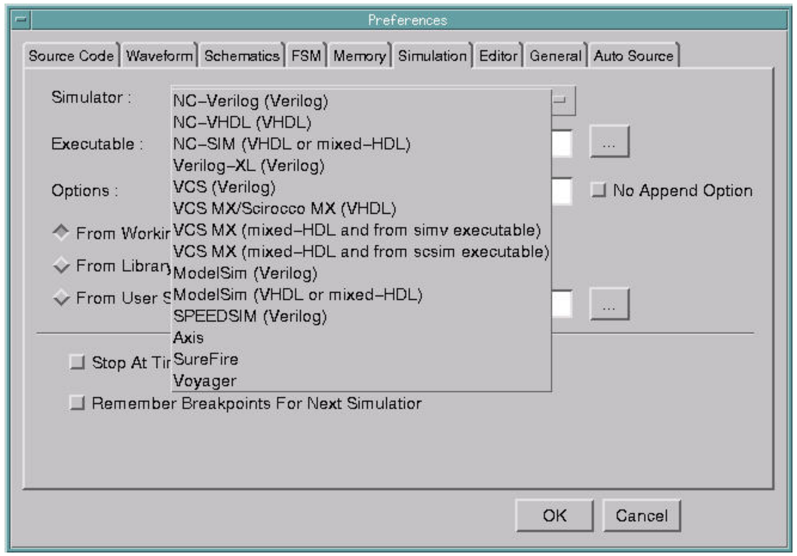
Command Done

Double click on signal to invoke Trace X

NOVAS

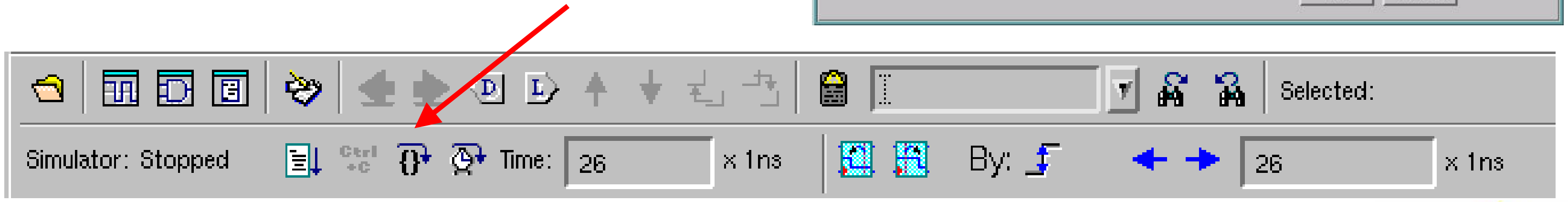
Start Interactive Mode

- Confirm the PLI/FLI shared object files provided by verdi are linked with your simulator.
- Choose the simulator under **Tools → Preferences → Simulation** tab.
  - *Be sure to select stop at time 0.*
- To enable interactive mode select **Tools → Interactive Mode**.
  - This adds a new toolbar and ***Simulation*** and ***Debug*** menu options to *nTrace*.



NC–Verilog (Verilog)
NC–VHDL (VHDL)
NC–SIM (VHDL or mixed–HDL)
Verilog–XL (Verilog)
VCS (Verilog)
VCS MX/Scirocco MX (VHDL)
VCS MX (mixed–HDL and from simv executable)
VCS MX (mixed–HDL and from scsim executable)
ModelSim (Verilog)
ModelSim (VHDL or mixed–HDL)
SPEEDSIM (Verilog)
Axis
SureFire
Voyager

Interactive Toolbar

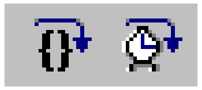


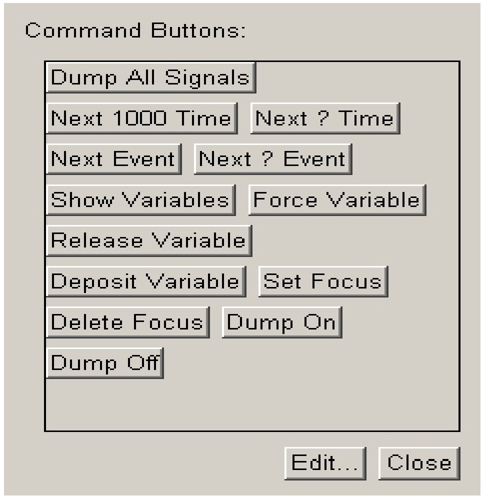
Interactive Simulator Control

- Click the **Run/Continue** icon to start simulator.
- Click the **Stop** icon to stop simulator.
- Run the simulator to specified time by entering a value in the **Time** text field.
- Use **Simulation → Reset** to reset simulator.
- Use **Simulation → Finish** to finish simulation.
- Use **Simulation → Kill Simulator Process** to kill the simulator process.

Other Interactive Control Options

- Execute Verilog or VHDL step by step.
  - Enable Active Annotation for real time updates.
- Use **Debug → Breakpoints** to set and control breakpoints.
  - Double click on the line number in source window to set/clear a breakpoint.
- Use **Debug → Set Focus** to focus on a scope(s).
  - If a focus is set, the source window will only step through the source code that is in the focus modules.
- Use **Debug → User-Defined Commands** to bring up the user-defined commands window for frequently used commands.




Command Buttons:

Save/Restore Debug Environment

- Save the debugging status to a session file
  - In *nTrace,* **File → Save Session...**
  - Information saved:
    - Displayed waveforms and settings.
    - All the windows, locations and content.

- Recover the previous debugging status from a session file.
  - In *nTrace,* **File → Restore Session...**
  - On the command line, use –ssr <file>.ses

Summary

- At this time you should
  - Be able to enable active annotation and debug a simulation problem with active trace.
  - Apply the simulation result to nState
  - Know how to load an SDF file, annotate values and calculate longest/shortest paths.
  - Be able to use different techniques to isolate logic for active fan-ins, bus contention and X detection.
  - Know how to save a debug session for future use.

Wrap-up

- Questions?

- Take 20 minutes to complete the Debug with Verdi LAB

If you have problems…

- Contact support with the following information
  - Complete description of the problem and what you were doing
  - verdi or nLint version
  - Platform and OS version
  - Simulator and version (if simulation problem)
  - Core stack (if applicable)
  - verdiLog directory (at a minimum the verdi.cmd)
  - Support contact:
    - <name>
    - <email>
    - <phone>