



Debug for Assertion Based Verification

Supporting OpenVera™ Assertions

Bassam Tabbara

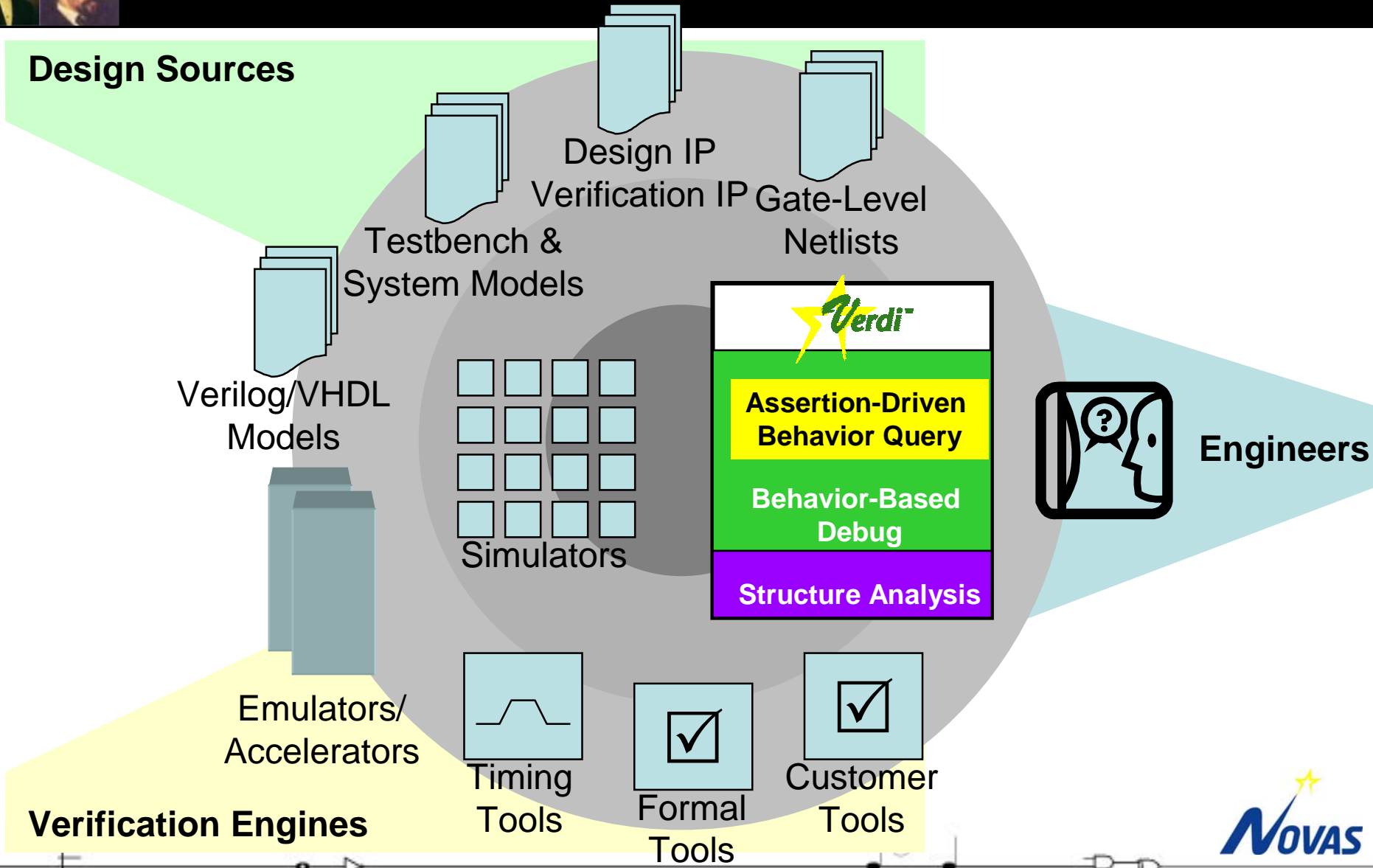
Technical Manager, R&D

Agenda

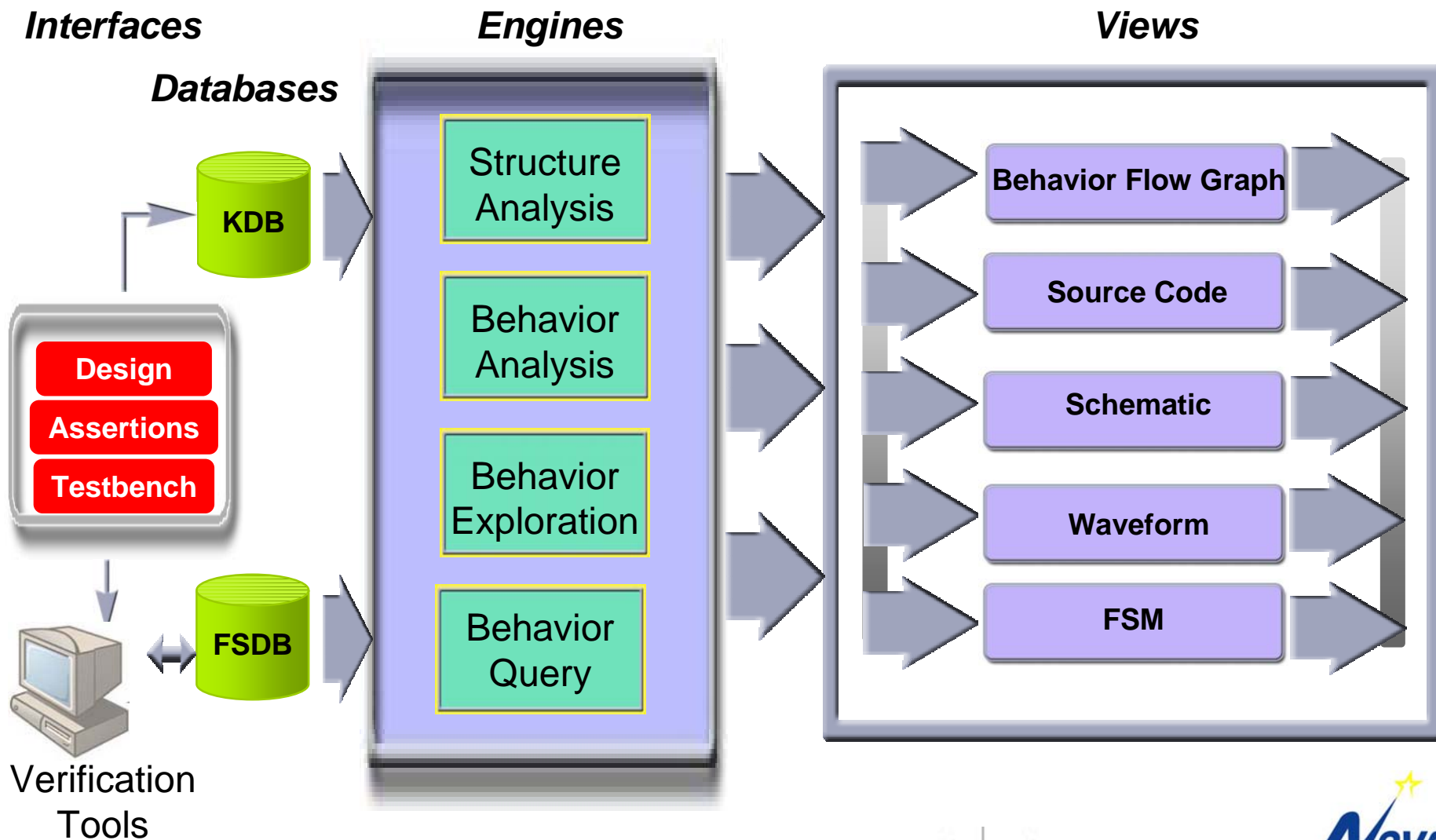


- Our role in Verification
- Our Technology
- Open Interoperability
- Behavior-Based Debug
 - Verdi™
 - Extending with Assertions
- Behavior Query

Verification Universe



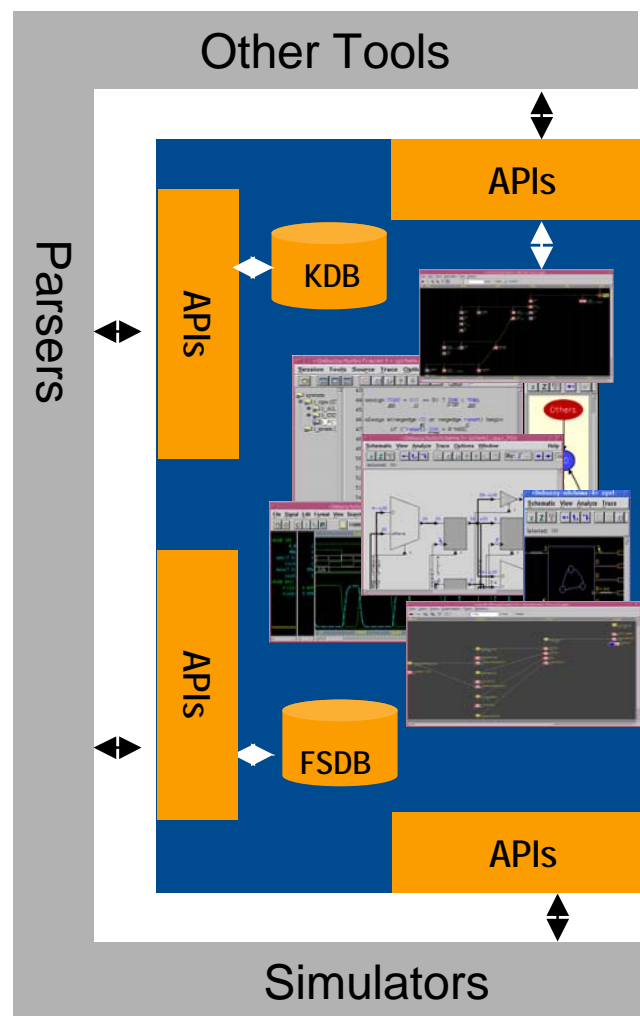
Novas Technology



Open Interoperability



- Waveform database APIs
 - Writer for detection tools
 - Reader for analysis tools
- Design database APIs
 - Writer for adding data
 - Reader for tools
- Visualization tool control API
 - TCL functions for every command, and more
- Simulation control API
 - Debug system as simulator GUI



Behavior-Based Debug



- Builds on top of design knowledge architecture
 - Includes all traditional structure-based views and operations
 - Applies synthesis and formal technologies
- Behavior analysis engine is the heart of the new approach
 - Infers formal logic model from HDL and assertion source
 - Identifies control/data path, separates active/inactive logic
- Behavior Exploration
- Assertion-Driven capabilities align with assertion-based verification
 - Trace from assertion trigger/pass/fail data to assertion source to HDL source
 - View assertions in waveforms, source browser, hierarchy browser
 - Check assertions post-simulation using signal dump data

Verdi™ - Behavior-Based Debug System

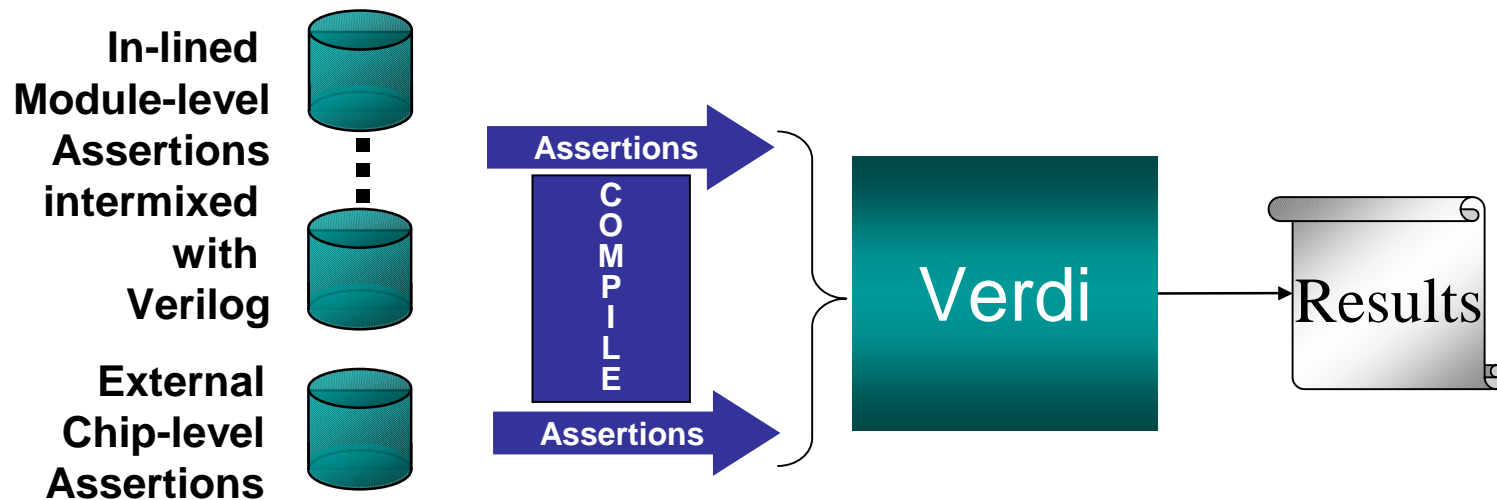


The screenshot displays the Verdi software interface, which is used for behavior-based debugging. It is divided into several main sections:

- Schematic View (Top Left):** Shows a circuit diagram with components like multiplexers and a clock source. A red box highlights a specific part of the circuit. The selected component is `...tem_i_CPUsystem.i_pram.data[7:0]`.
- Trace Window (Top Right):** Shows a tree view of the design hierarchy, including `tb_CPUsystem`, `i_BJkernel`, `i_BJsource`, `i_CPUsystem`, `i_CPU`, `i_ALUB`, `i_CCU`, `i_PCU`, and `i_pram`. Below the tree is a code editor showing the configuration for the `i_pram` component:

```
35 reg BUSY;  
36 reg Reading;  
37 reg [7:0] Writing;  
38  
39 reg [7:0] macronam [255:0];  
40  
41 assign data = R_W ? dataout : 8'hz;  
42  
43 initial
```
- Trace View (Bottom):** Shows a detailed trace of the system's behavior over time. The trace is organized into columns representing different components and their signals. The time axis is marked with values like 650, 675, 700, 775, and 800. Signals shown include `dataout[7:0]`, `data[7:0]`, `data[7:0]`, `ExtData[7:0]`, `TDB[7:0]`, `IntDataIn[7:0]`, `IDR_d[7:0]`, `DataIn[7:0]`, `a[7:0]`, `b[7:0]`, `AluOut[7:0]`, `ACC[7:0]`, `R_W`, `VMA`, `addr[7:0]`, `macronam[addr]`, `RESET`, `R_Wmode`, `DR_load`, `RESET`, `select[2:0]`, and `ADD`.

Assertions Usage Model



Assertions can be created/used by design or the verification teams

Assertion Extensions and Technology



- KDB extensions to store definitions and relationship to design
 - Sufficient for transactions
- FSDB extensions to store results during verification – pass/fail/ongoing
 - Sufficient for transactions
- Assertion-specific display mechanisms
 - Waveforms
 - Source with annotation
 - Hierarchy browser
 - Requires further extensions for transaction support

Assertion Import



Import Design Property

Default Directory:
/da3/jenkuo_liao/DEMO/Verilog/RTL

Design Files:
/da3/jenkuo_liao/DEMO/Verilog/RTL/ALUB.ova
/da3/jenkuo_liao/DEMO/Verilog/RTL/BJkernel.ova
/da3/jenkuo_liao/DEMO/Verilog/RTL/BJsource.ova

/da3/jenkuo_liao/DEMO/Verilog/RTL/BJsource.ova

- DEMO
 - Courses
 - VHDL
 - Verilog
 - Behav
 - FSM
 - Gate
 - RTL
 - memory
 - sdf
 - src

- DebussyLog
- ReusnerLog
- vericomLog
- work.lib++
- ALUB.ova
- BJkernel.ova
- BJsource.ova
- CCU.ova
- P5.ova
- PCU.ova
- PV.ova
- TopModule.ova
- alu.ova

Filter: *.ova

OK Cancel

<Debussy:nTraceMain:1> system

File View Source Trace Tools W

- Import Design...
- Reload Design L
- View Import Log
- Compile Design...
- Load Simulation Results...
- Close Simulation Results
- Load Property...
- Load Property Simulation Results...
- Close Property Simulation Results...
- Load SDF Files...

Assertion Management and Control



The screenshot illustrates the process of managing assertions in a Verilog project. It shows three windows:

- Project Tree (Left):** Displays the hierarchy of the design, including modules like `i_BJkernel`, `i_BJsource`, `i_cpu`, and `i_ALUB`. Under `i_ALUB`, there are three assertions: `check1`, `check2`, and `forbid1`. `check1` and `check2` are checked, while `forbid1` is unchecked. A yellow callout box with the text "double click" points to `check1`.
- Code Editor (Middle):** Shows the Verilog code for the `i_ALUB` module. The code includes inputs like `IR`, `IDB`, `PC`, `CH`, `alu_mode`, `bus_mode`, `carry_mode`, `clock`, and `reset`, and outputs like `S1`, `ALU`, `IXR`, and `error_out`. It also defines registers `IXR`, `IXR_tmp`, and `ACC_tmp`.
- Zoomed-in Code Editor (Right):** Shows a close-up of the `assert check1 ; check(te3);` statement. A red arrow points to the `check1` identifier, and a yellow callout box with the text "double click" points to it.

Interfacing to Assertion Tools: Interactive Control APIs



- Client/Server model
 - For each selected assertion
 - VCS™
 - Enable/Disable
 - Register callback
 - Other
 - **Ovasim** shared library
- Direct invoke
 - Novas Checker (runs on design trace)



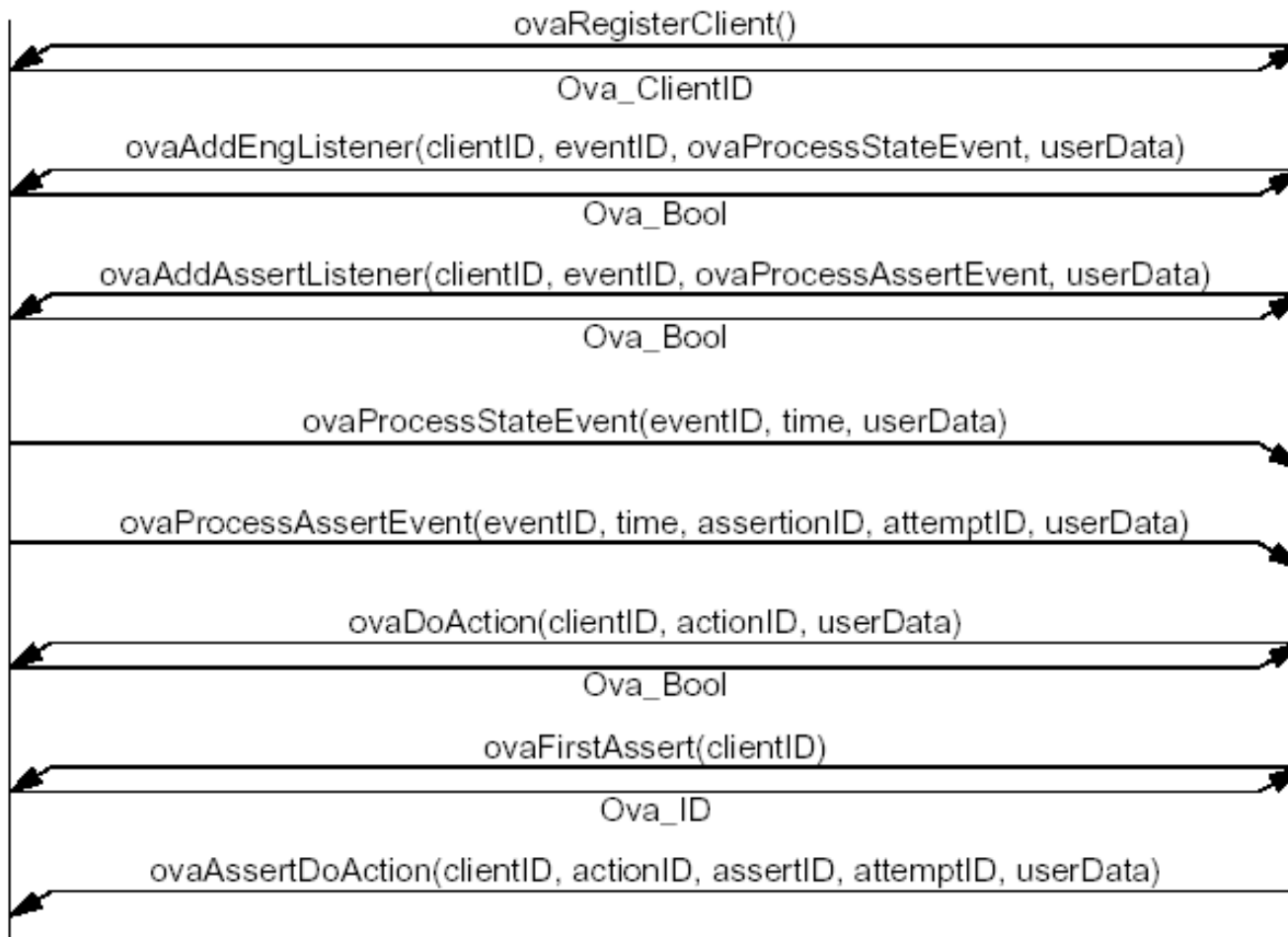
Interfacing to Assertion Tools: Interactive Control APIs (VCS)

The OVA Engine

The Client

OVA

Verdi



Capturing and Storing Assertion Results



- Extend the var type in FSDB with assertion:
 - Event (a.k.a. temporal expression)
 - Check
 - Forbid
- Var has signal and clock *hooks*
 - event e1 = a #[3] b;
 - a
 - b
 - clock
- Var value is composed of
 - A time period [begin_time end_time]
 - A result [success/failure/ongoing/explosion/unknown]
- VCS FSDB dumper extended to support assertion results

Assertion Source Annotation



```
<Debu... Trace Text:3> test.oVA - /da3/jenkuo_liao/OVA ...
File View Source Window Help
clock
By: 4050 x 1ns
1
2 scope system,i_cpu,i_ALUM {
3   clock posedge clk {
4     event te1 = a && b;
5     event te2 = posedge s;
6     event te3 = if (matched te1) then te2;
7     assert check1 : check(te3);
8   }
9 }
10
11 scope system,i_cpu,i_CCU {
12   clock posedge clk {
13     event te4 = posedge c;
14     event te5 = d;
15     event te6 = if (matched te4) then te5;
16     assert check4 : check(te6);
17   }
18 }
```

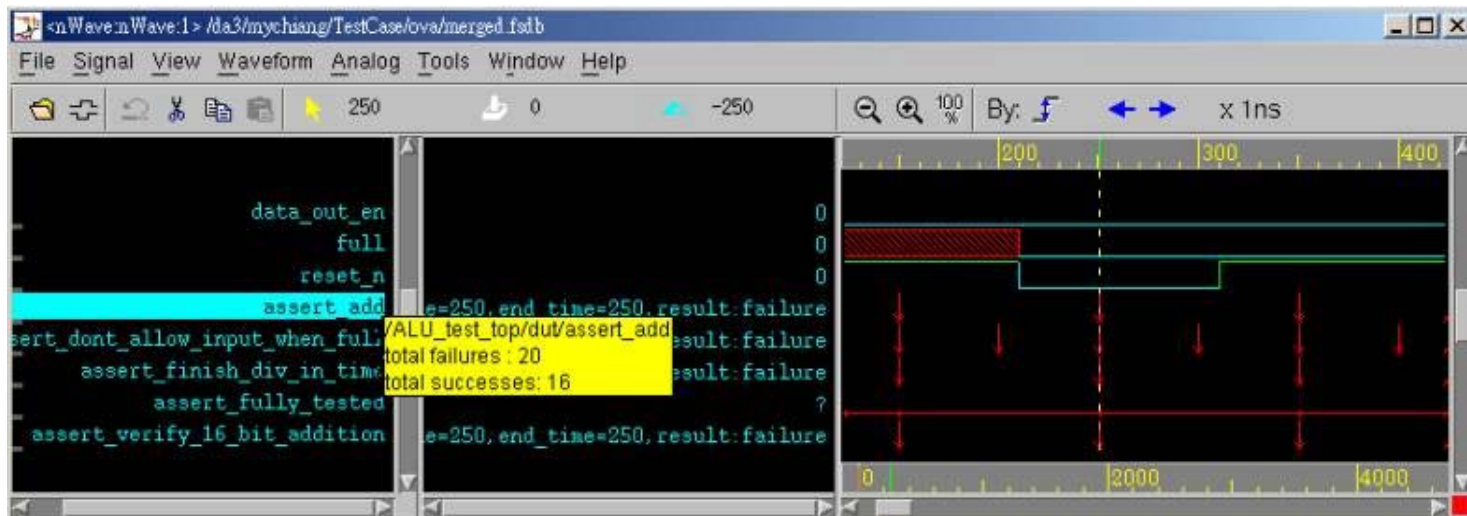
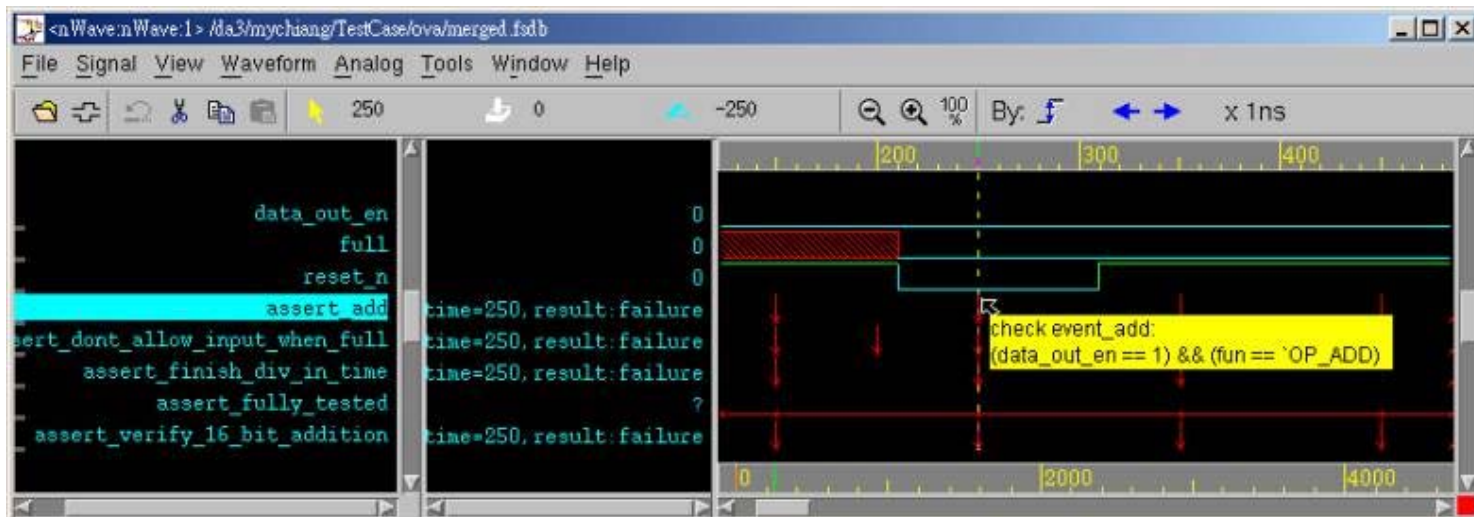
For temporal expression:

- : TRUE (fire event)
- : FALSE (no event)
- : Not Found
- : No Value
- : Under Evaluation

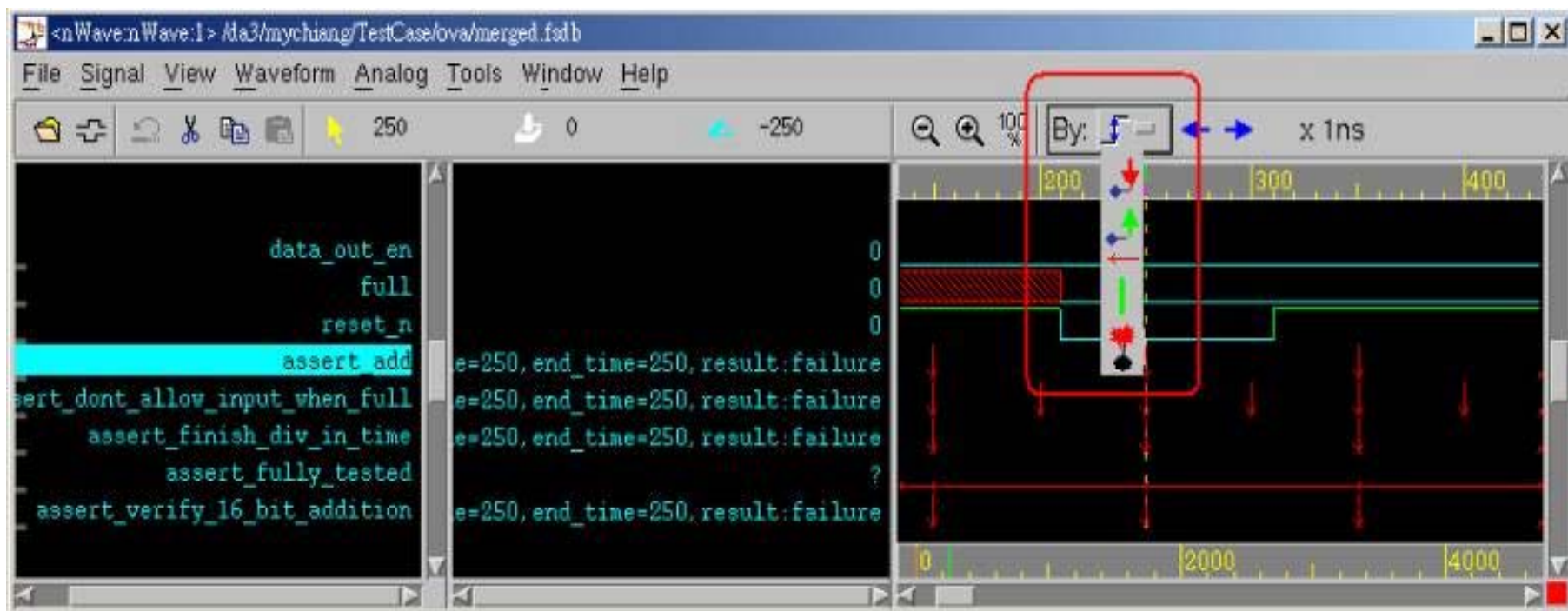
For assertion:

- : SUCCESS
- : FAILURE
- : Not Found
- : No Value
- : Under Evaluation

Visualization and Statistics



Data Search and Traversal



Double-Click, Drag & Drop



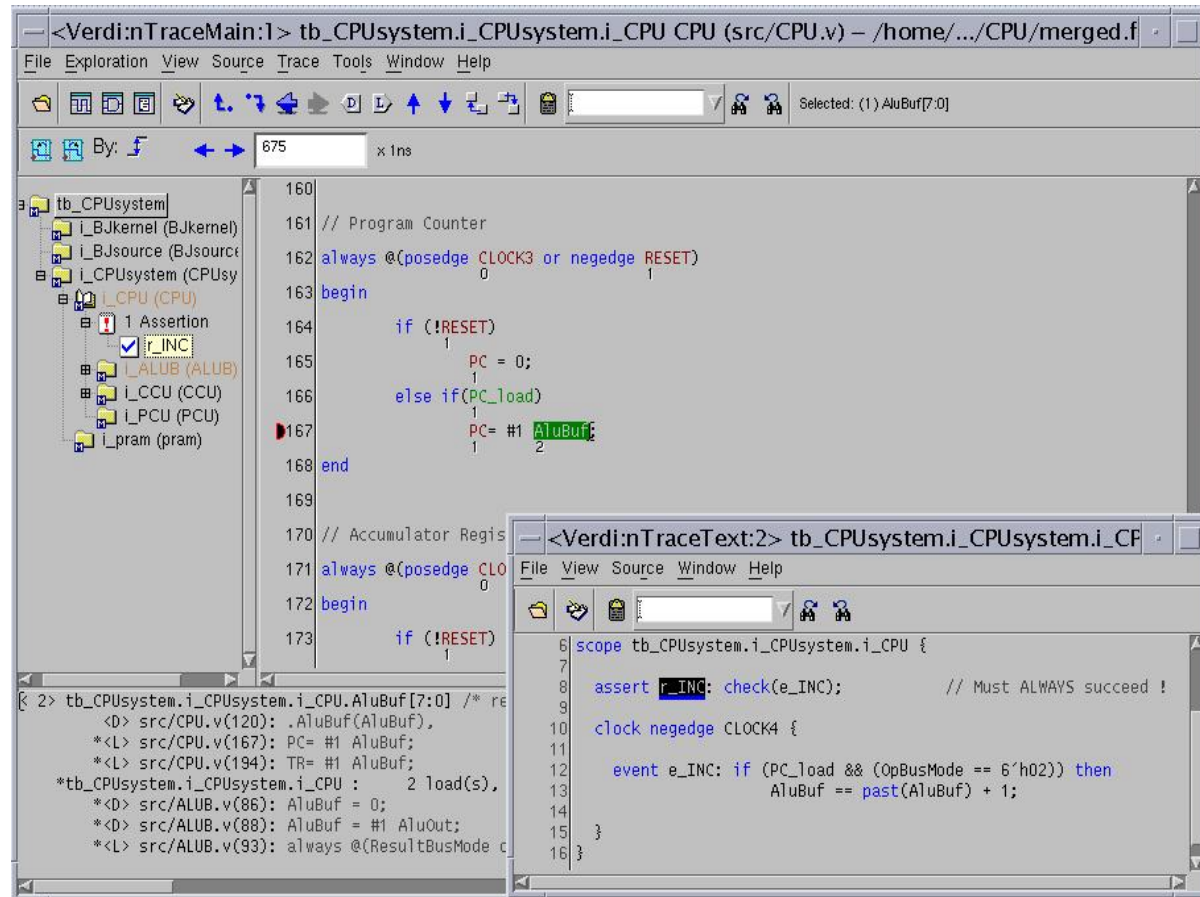
The image illustrates a workflow for navigating between different views in a Verilog IDE:

- 1**: Selecting a component (ALUB) in the project tree.
- 2**: Double-clicking on the selected component to open its source code.
- 3**: Selecting a specific assertion (check1) in the source code.
- 4**: Dragging the selected assertion to the waveform viewer.
- 5**: Dropping the assertion into the waveform viewer to monitor its behavior.
- 6**: Viewing the waveform data (data[7:0]) in the waveform viewer.

Debugging With Assertions



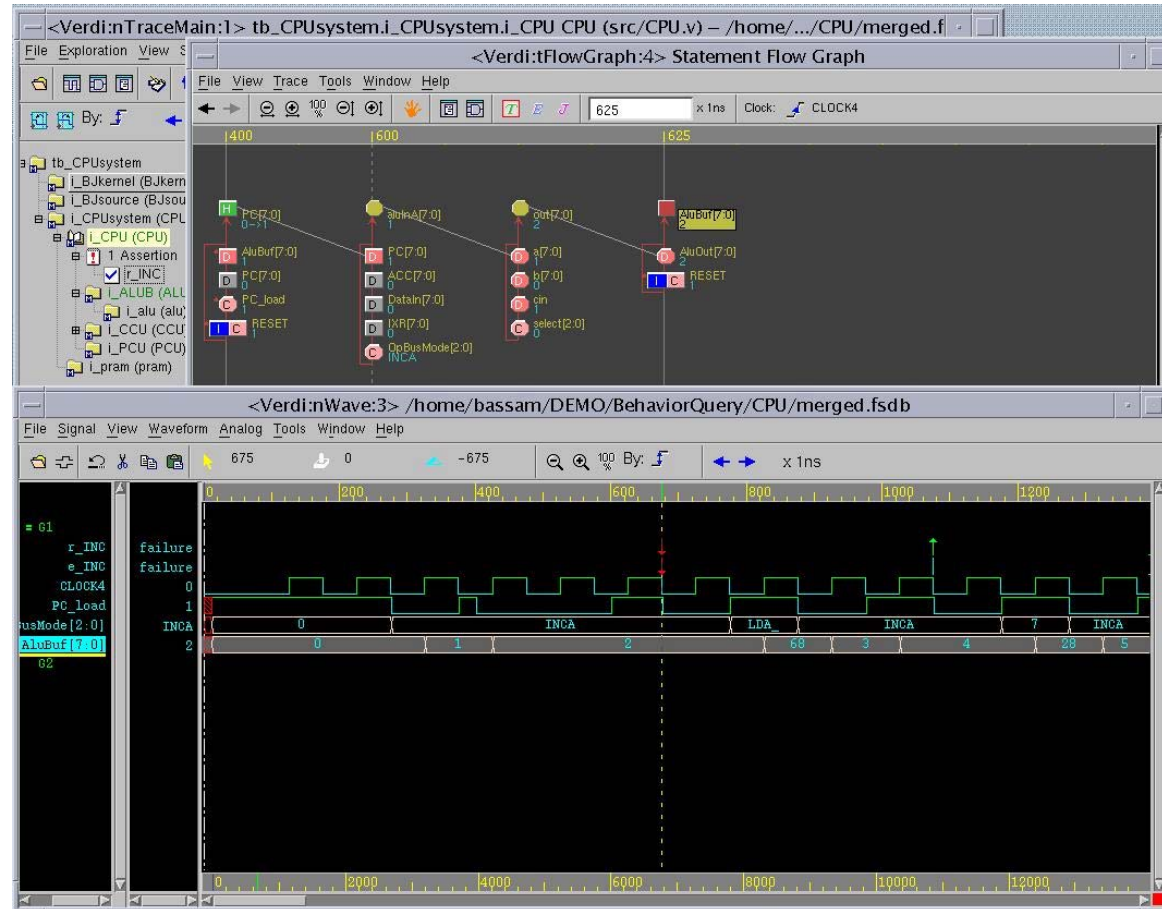
- Load and view assertion source
 - View assertion/design association
 - Trace assertion “drivers” and “loads” across design
 - Cause-and-effect details
- Assertion check
 - Check assertions without re-running simulation



Behavior Query



- View assertion result
 - View assertion/design
- Behavior Query
 - Where do I start tracing?
 - What range of time is relevant?
 - Which signals are relevant ?



Summary

