

# Assertion Based Verification

James M. Lee

Author: Verilog Quickstart



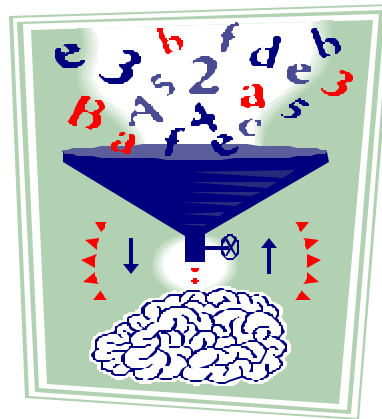
# Introduction

- Complexity
  - Complex designs are hard to verify
- Verification
  - New verification tools and methods are needed to combat complexity
- Assertion-based techniques
  - Enable earlier detection and faster debug of design problems

# Complexity

## Complexity Drivers

- Large chip, large team
- High-performance design tactics
- Outside IP, many languages / tools

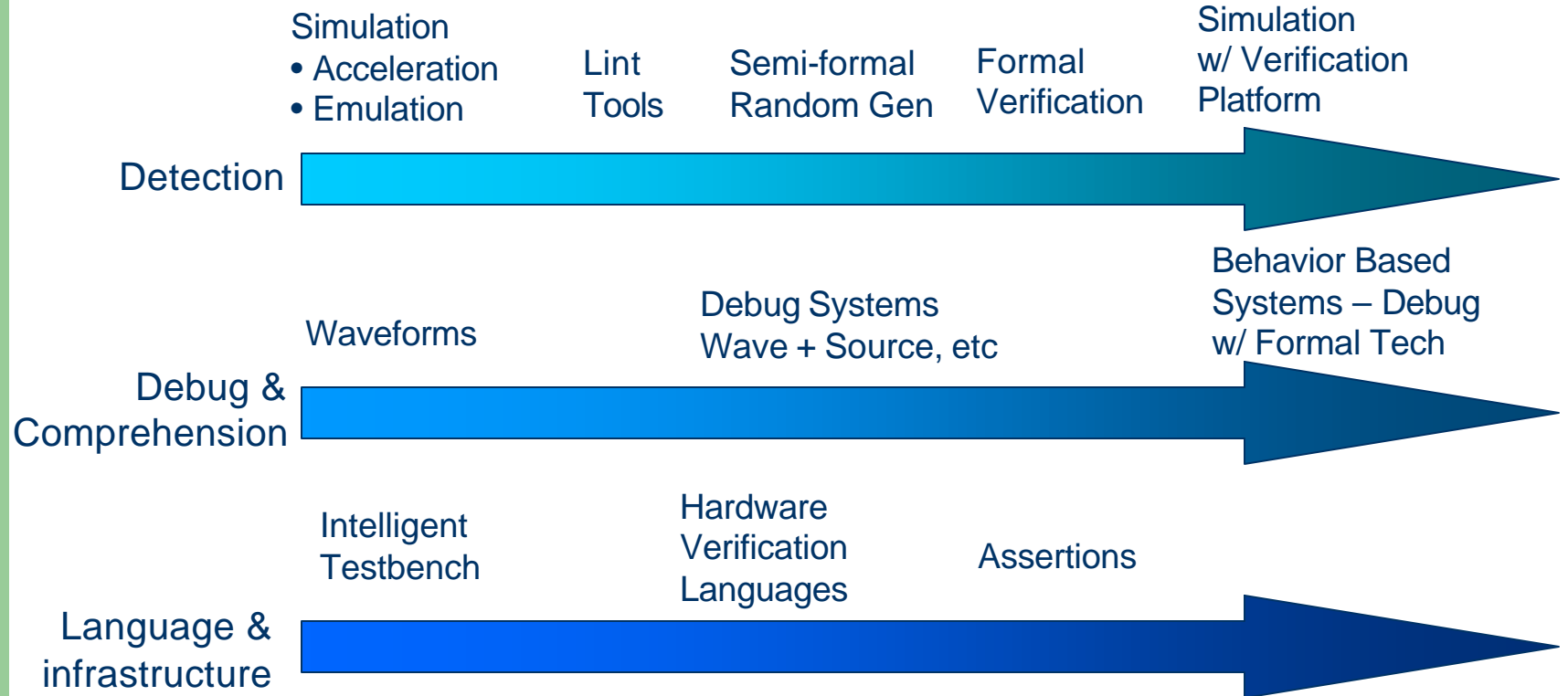


## Complexity Impact

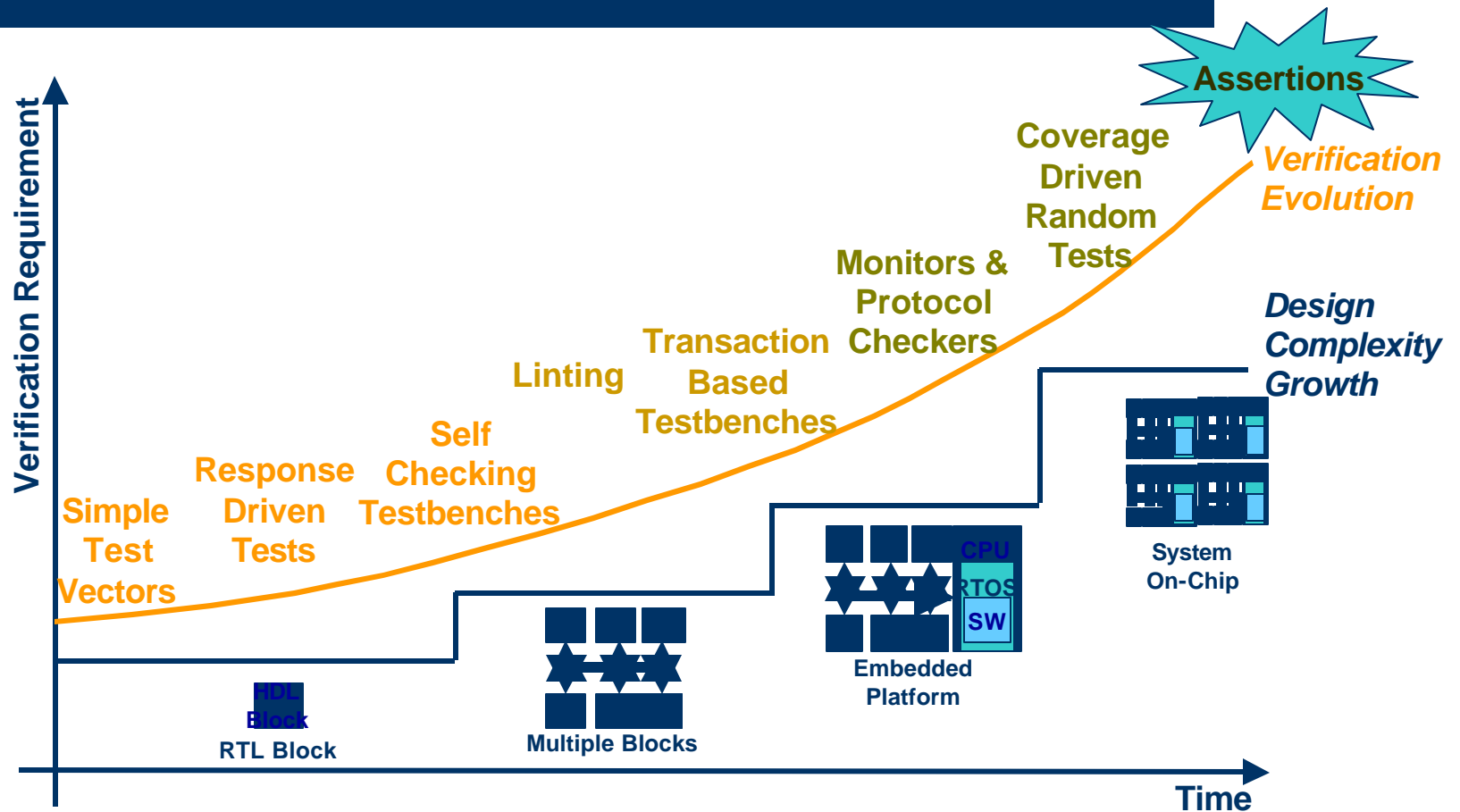
- Lack of design understanding
- Incomplete, delayed verification
- Long debug cycles

*Risk of buggy products and missed schedules*

# Verification Tool Evolution

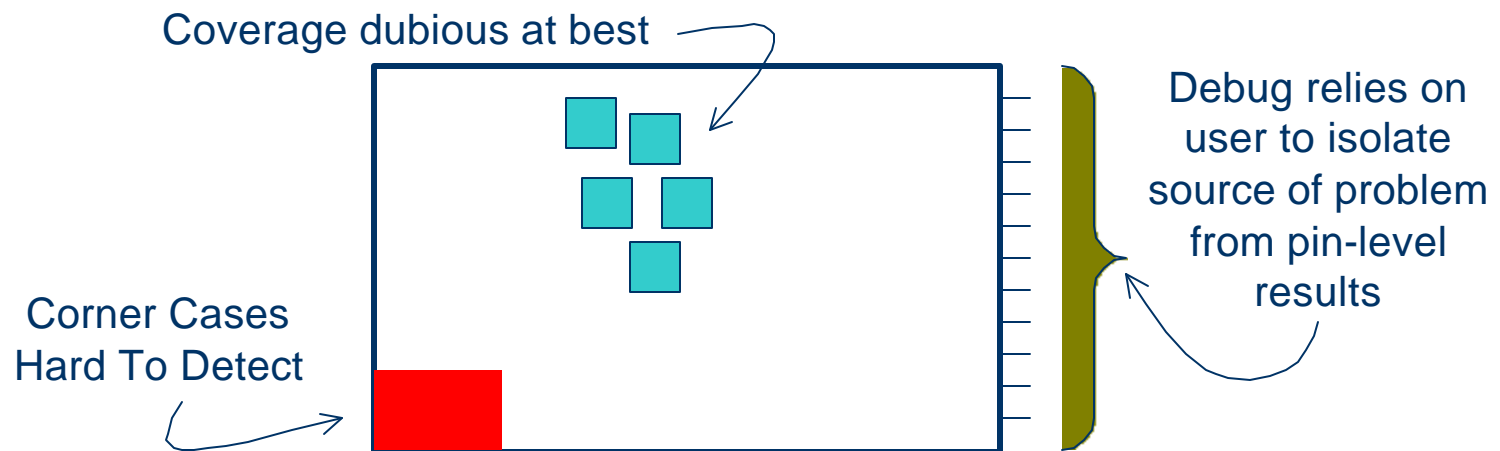


# Verification Methodology Evolution



# Today's Methodology

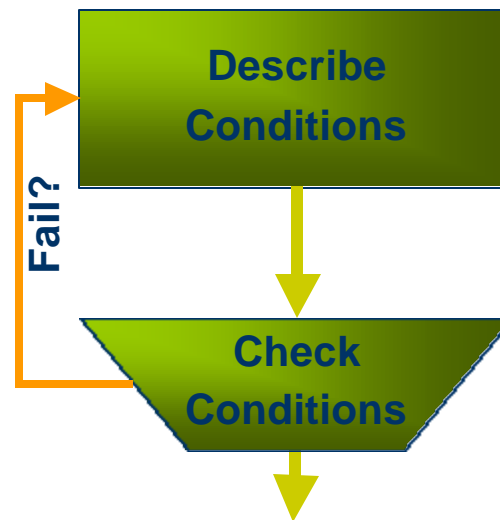
## Pure Simulation Based Verification Breaking



*We Need A Better Way.....*

# Assertion-based Verification

Failures indicate improper operation localized to an operation or set of signals



- Describe expected or unexpected conditions in device under test
  - Specialized languages
  - Language extensions or features
- Check to make sure that conditions are satisfied
  - Dynamically during simulation
  - Statically using formal
  - Hybrid using semi-formal

# Assertion-based Verification

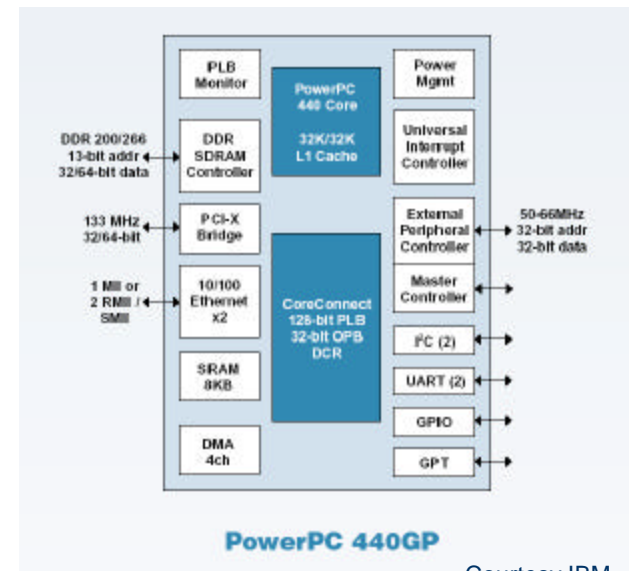
- Potential benefits
  - Capture concise designer knowledge in RTL
  - Detect more bugs earlier
    - Combination of simulation and formal verification
  - Debug root cause of problem more quickly
    - Intelligent debug systems that leverage assertion results and formal techniques
- Current situation
  - In its infancy in terms of use, but beginning to move into mainstream verification environments
  - OVL and things like “assert one hot” are analogous to spice level modeling
  - Complicated, proprietary language formats create usage overhead and unstable methodology

# What's Needed

- Not just higher level assertions, but
  - Methodologies for choosing good assertions
  - Intelligent debug systems to understand and analyze assertions and their results
    - Is the assertion described correctly?
    - How do I quickly get from the assertion failure to the bug in the design description?
  - Standard assertion format and good tool support

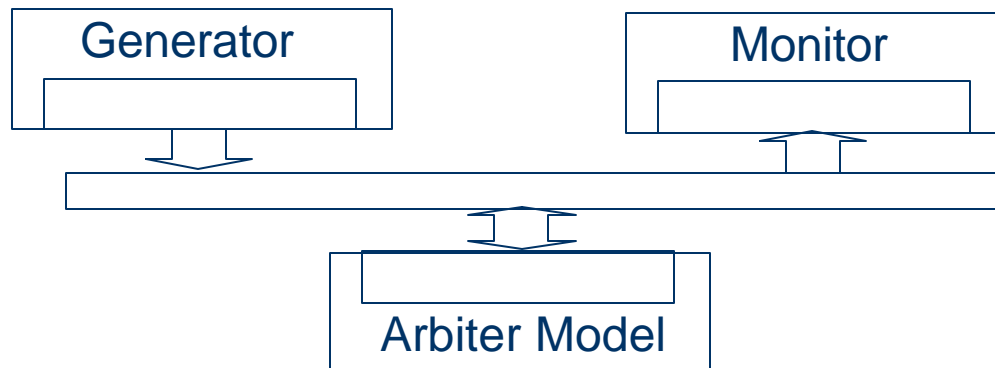
# Case Study IBM PLB BUS

- IBM Power PC is used in many high performance SOC's
- 128 bit wide bus @ >200 MHZ
- Split read and write busses
- Pipelined accesses
- IBM provides PLB toolkit and monitor
- PLB is commercially referred to as CoreConnect



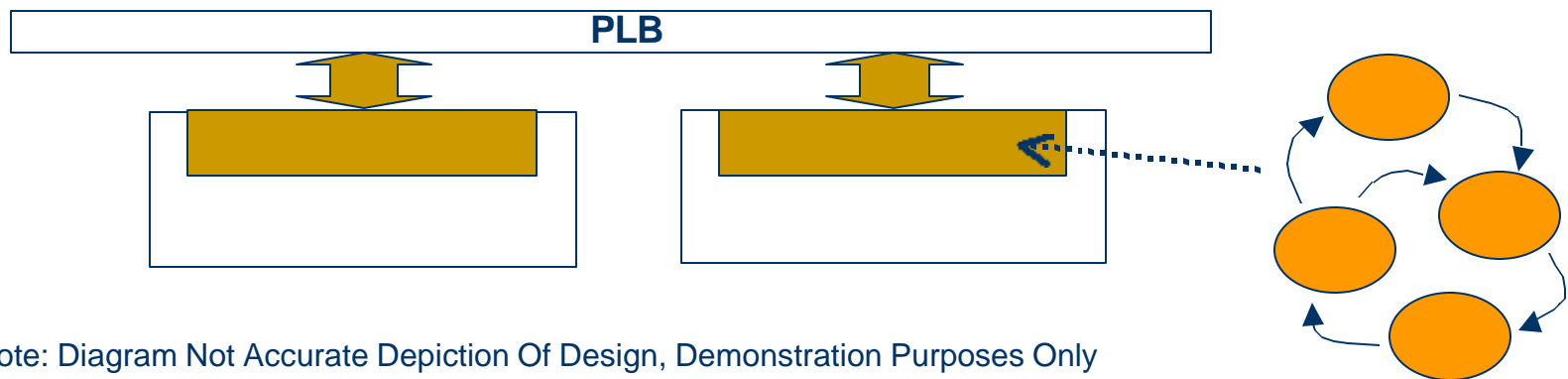
# Designing With PLB Bus

- The bus protocol has several ways of starting and ending transactions.
- The Toolkit provides *behavioral* arbiter, masters and targets and ways to generate traffic
- The Toolkit monitor lets you know if you screw up.



# Design Project

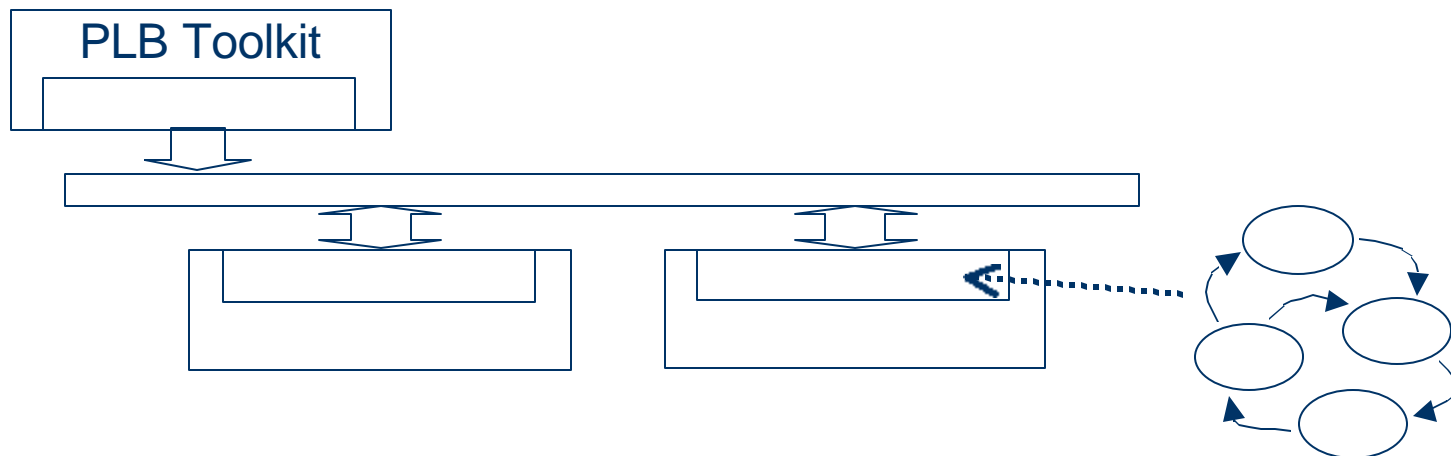
- Design re-usable Master and Slave interfaces.
- Use the interfaces on several blocks that integrate into a PPC based SOC
- Test in context of PLB toolkit.



Note: Diagram Not Accurate Depiction Of Design, Demonstration Purposes Only  
Actual Circuit Diagram Proprietary Information

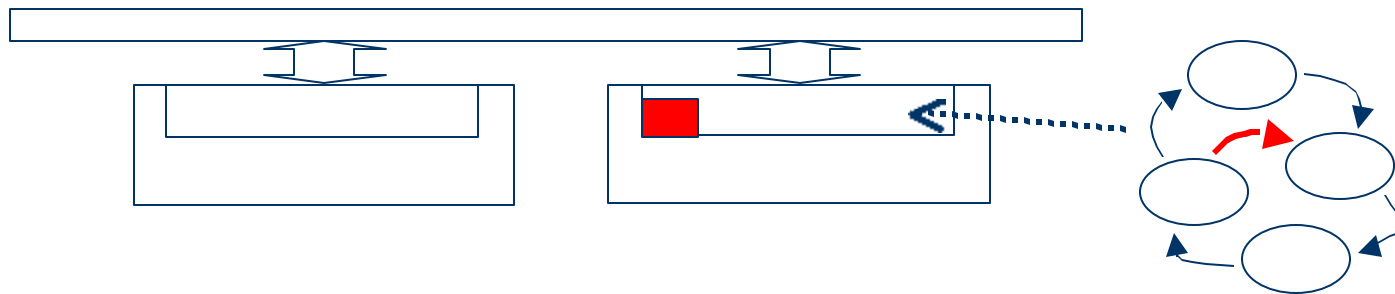
# Verification Plan

- Test all normal and error functionality of the design blocks in PLB context.
- Use PLB toolkit and additional bus master to generate live bus traffic and cross traffic.



# Bug missed in RTL/Unit simulation

- Bus loaded with test bench and DUT traffic as well as “random traffic”.
- One corner case of arbitration behavior, listed only as a note in the PLB bus spec, not detected
- Bus interface state machine missed a transaction
- Only found in system integration simulation with the other design blocks and *actual* PLB arbiter.

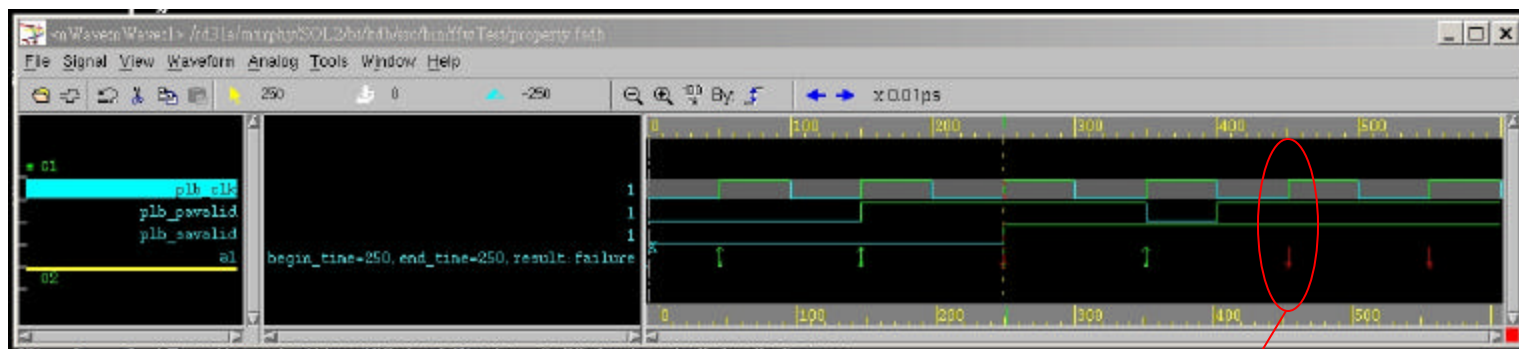


# Bug could have been found sooner with assertions

- This SystemVerilog assertion, based on the PLB spec, detects the bug

```
always @(posedge plb_clk) begin
    a1: assert(!(plb_pavalid && plb_savalid));
        else $error("prim and sec valid at the same time");
    if (plb_savalid && sl_addrack && plb_rnw)
        a2: assert(!plb_rdprim*[0:17];plb_rdprim) @@ (posedge plb_clk);
    if (sl_addrack && state == SA_ACK)
        a3: assert((state != LAST)*[0:18];(state == LAST))
            @@(posedge plb_clk);
end
```

# Isolating the Cause of the Bug



- Expand assertion to see related signals
- Jump from assertion failure to related source code
- Use behavior-based debug capabilities to quickly trace to source of problem

The screenshot shows a source code editor with a tree view on the left and code on the right. The tree view shows a project structure with 'system' and 'cpu' folders. The code on the right is a Verilog module for 'system1\_cpu1\_PLB\_PLB'. A red arrow points from the failure location in the waveform to line 130 of the code, which is the assertion: `assert (plb_pavalid == 1'b0);`. The code also shows the logic for 'plb\_pavalid' and 'plb\_savalid' signals.

## Easy to fix bug

- Although on the actual project this bug was found late in the system integration phase, it was easy to patch the state machine once the case was discovered.
- Hard to find, easy to fix.
- Finding it earlier with an assertion would have been better.
- Embedded assertions in toolkit could have saved on risk and schedule
- Lower level assertions are necessary until higher level protocol assertions and tests are built.

# Conclusions

- Assertion Based Verification can find bugs earlier
- Good Assertions are the key
- Assertions are gaining engineering acceptance and are being standardized – Accellera SystemVerilog
- Need mature methodology to proliferate the use
- The combination of assertion-based simulation and formal verification is key
- Intelligent debug systems are needed to help users effectively understand their assertions and results

# The Design Flow

