



# **Assertion Based Verification**

## **... in the Context of SystemVerilog**

**Victor Berman, Erich Marschner, Frank Schirrmeister**

# Agenda



- Application and use of assertions
- Assertion Languages
- Tool Support



# Application and use of assertions

# Do Assertions Really Work?



## Assertions in real designs:

<b>Assertion Monitors</b>	<b>34%</b>
Cache Coherency Checkers	9%
Register File Trace Compare	8%
Memory State Compare	7%
End-of-Run State Compare	6%
PC Trace Compare	4%
Self-Checking Test	11%
Simulation Output Inspection	7%
Simulation hang	6%
Other	8%

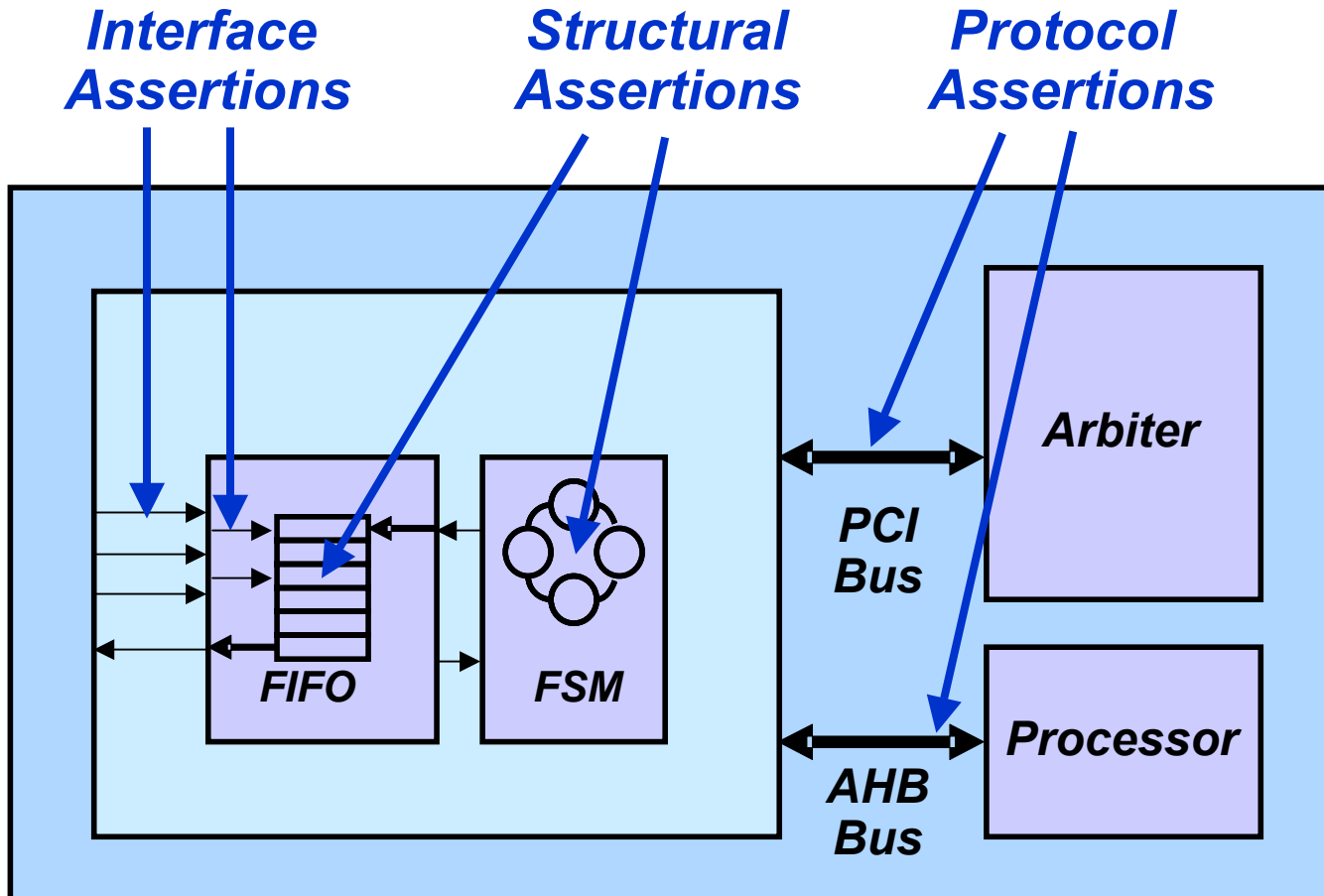
Kantrowitz and Noack [DAC 1996]

<b>Assertion Monitors</b>	<b>25%</b>
Register Mismatch	22%
Simulation "No Progress"	15%
PC Mismatch	14%
Memory State Mismatch	8%
Manual Inspection	6%
Self-Checking Test	5%
Cache Coherency Check	3%
SAVES Check	2%

Taylor et al. [DAC 1998]

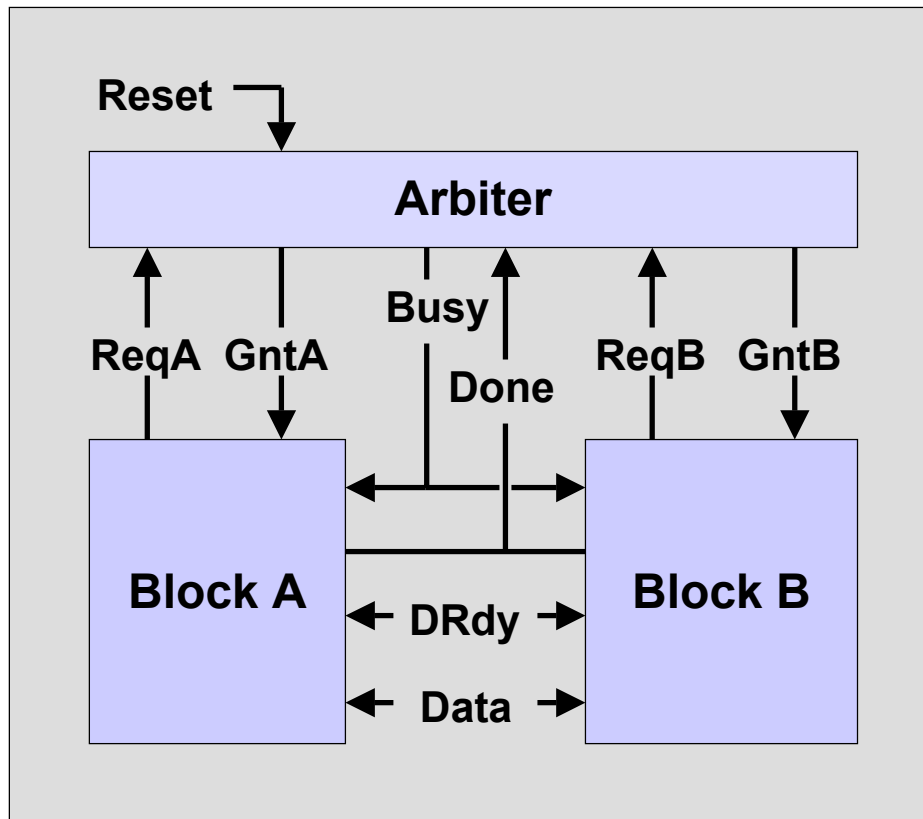
- **34%** of all bugs were found were identified by assertions on **DEC Alpha 21164** project  
*[Kantrowitz and Noack DAC 1996]*
- **17%** of all bugs were found were identified by assertions on **Cyrix M3(p1)** project.  
*[Krolnik '98]*
- **25%** of all bugs were found were identified by assertions on **DEC Alpha 21264** project.  
*[Taylor et al. DAC 1998]*
- **50%** of all bugs found were identified by assertions on **Cyrix M3(p2)** project  
*[Krolnik '98]*
- **85%** of all bugs were found using over **4000** assertions on **HP**  
*[Foster and Coelho HDLCon 2000]*
- **10,000** assertions in Cisco RTL project  
*[Sean Smith 2002]*
- **Thousands** of assertions in Intel Pentium project *[Bentley 2001]*
- *Many other examples. . . . .*

# Kinds of Assertions



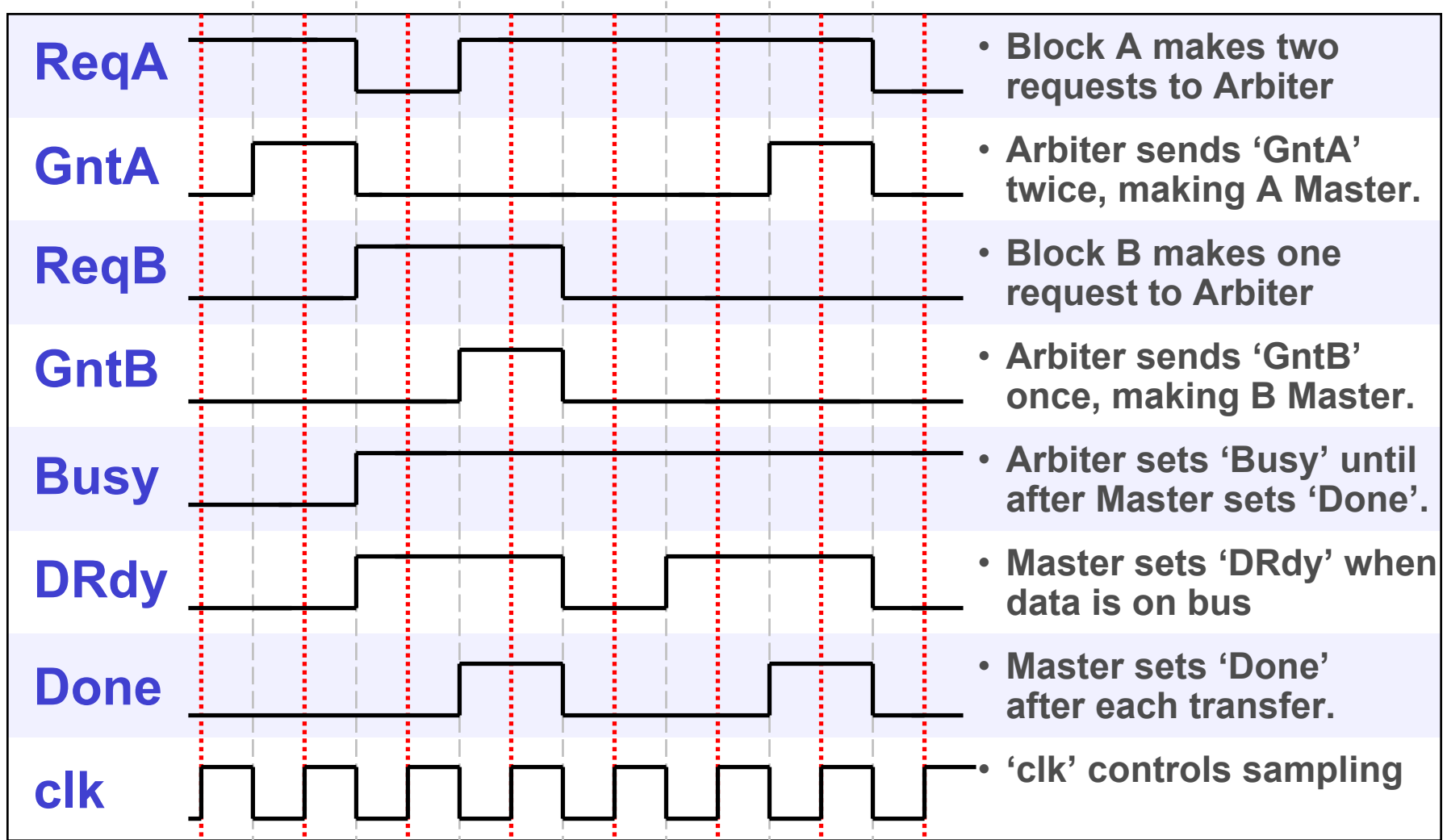
# A Simple Example

- Two blocks A,B exchange data via a common bus.



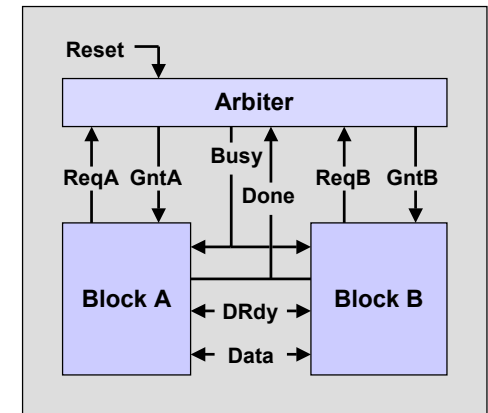
- A and/or B sends 'Req' to the Arbiter.
- Arbiter does round-robin scheduling between A,B.
- Arbiter sends 'Gnt' back to A or B, making it Master.
- Arbiter sets 'Busy' while A or B is Master.
- Master sets 'DRdy' when Data is on the bus.
- Master sets 'Done' in the last cycle of a grant.
- 'Reset' resets the bus.

# Simple Example Bus Protocol



# Some Assertions to Check

- A Grant never occurs without a Request.
  - `assert never GntA && !ReqA`
- If A (B) receives a Grant, then B (A) does not.
  - `assert always GntA -> !GntB`
- A (B) never receives a Grant in two successive cycles.
  - `assert never GntA && next GntA`
- A Grant is always followed by Busy.
  - `assert always GntA || GntB -> next Busy`
- A Request is eventually followed by a Grant.
  - `assert always ReqA -> eventually GntA`

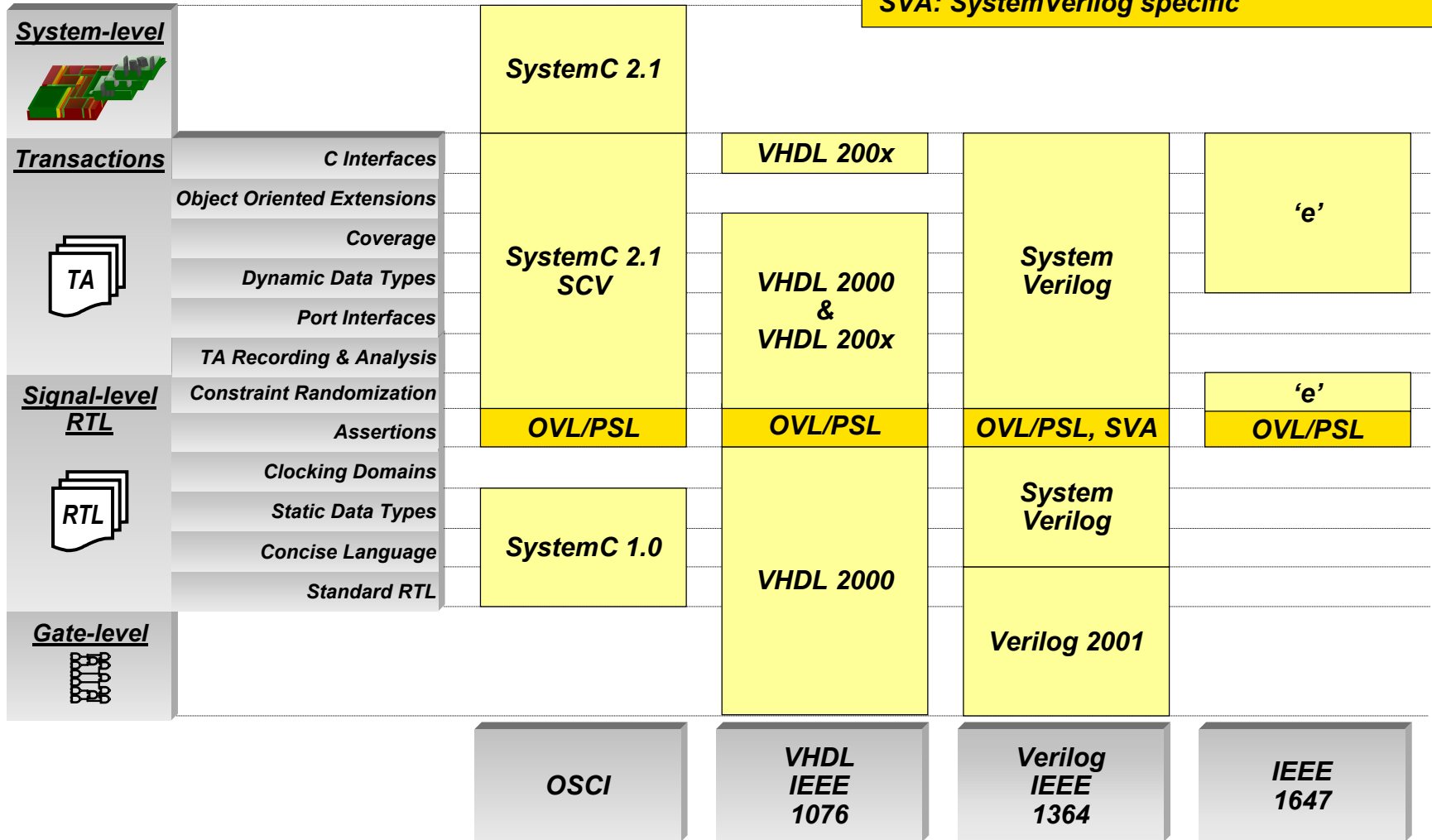


# Assertion Languages

# SystemVerilog in a multi-lingual world



*PSL and SVA complementing each other*  
*PSL: Language neutral*  
*SVA: SystemVerilog specific*



# PSL and SVA



- PSL acts as specification
  - abstract
  - language-independent
  - declarative
- SVA acts as implementation
  - SystemVerilog-specific
  - encodes abstract assertion
  - procedural

```
...  
always @(posedge clk) disable iff !resetn  
  assert property ( req |=> ##[0:$] ack );  
  
always @(posedge clk) disable iff !resetn  
  assert property  
    ( req |=> !req throughout ack [*->] );  
...
```

# PSL and SVA



## PSL

```
assert always  
( req -> next eventually! ack );
```

```
assert always  
( {req} | => {[*]; ack} abort !resetn )  
@(posedge clk);
```

```
always @(posedge clk) disable iff !resetn  
assert property ( req | => ##[0:$] ack );
```

## SVA

```
assert always  
( req -> next never req until ack );
```

```
assert always  
( {req} | => {req[=0] && ack [->]} abort !resetn )  
@(posedge clk);
```

```
always @(posedge clk) disable iff !resetn  
assert property  
( req | => req[*=0] intersect ack [*->] );
```

```
always @(posedge clk) disable iff !resetn  
assert property  
( req | => !req throughout ack [*->] );
```

# Correlation of SVA and PSL Features



<u>Operators</u>	<u>PSL</u>	<u>SVA</u>	<u>Notes</u>
sequence delimiters	{...}	(...)	
zero or more	[*]	[*]	
one or more	[+]	[*1:\$]	
consecutive repetition	[*count] [*range]	[*count] [*range]	
arbitrary repetition	[=count] [=range]	[*=count] [*=range]	SVA borrowed from PSL, but added *
"goto" repetition	[->count] [->range]	[*->count] [*->range]	SVA borrowed from PSL, but added *
sequence disjunction		or	
sequence conjunction	&	and	
length-matching sequence conjunction	&&	intersect	
a occurs in some cycle(s) during b	<i>within</i>	<i>within</i>	PSL v1.1 'within' = {[*];a;[*]} && {...}
a occurs in every cycle during b	{a[*]} && {b}	<i>throughout</i>	PSL v1.1 might add 'throughout' also
sequence concatenation (non-overlapping)	;	##1	
sequence concatenation (overlapping)	:	##0	
sequence same-cycle implication	->	->	
sequence next-cycle implication	=>	=>	
boolean implication	->		
boolean bi-directional implication	<->		
always	<i>always</i>	<i>always</i>	
never	<i>never</i>	<i>always not</i>	
eventually	<i>eventually!</i>		
next (family of operators)	<i>next*</i>		
until (family of operators)	<i>until*</i>		
before (family of operators)	<i>before*</i>		
interrupt (with success)	<i>abort</i>	<i>disable iff</i>	

OVA features
PSL features
Features resulting from PSL/OVA merging in DWG

# SVA / PSL semantic alignment



- SVA and PSL have identical semantics where they overlap
  - Sequences
  - Sequence implication
  - Disable/Abort
  - Clocking
- A proof of semantic equivalence for non-derived operators has been completed.
  - Draft document now being reviewed for publication



# Tool Support

## Cadence Incisive Verification Platform

- Native compilation and execution of PSL/Sugar assertions
- Optimizes PSL/Sugar assertions into Incisive single kernel engine
- PSL/Sugar assertions are treated as verification objects
- Interactive simulation & debugging: breakpoints, interrogation, probing

## Mentor Scalable Verification Platform

- [...] Mentor Graphics Scalable Verification platform [...] to include standards support for all major existing and emerging design languages -- Verilog 2001, VHDL, SystemVerilog (first phase of v3.1), SystemC 2.0.1 (including SystemC Verification Library 1.0) and the ***Property Specification Language 1.0 (PSL)***
- Monday October 27<sup>th</sup> 2003

# Incisive Assertion Browser



SimVision: Assertion Browser 1

File Edit View Windows Help

Total: 34 Inactive: 34 (100%) Begun: 0 (0%) Finished: 0 (0%) Failed: 0 (0%)

Assertion Name	Module	Instance	Current State	Begun Count	Finished Count
AddrHeldWhenMasterBusy	ahbCompliance	main.ahbMonitor_ABV.	inactive	0	0
AddressIncBySizeDuringAll	ahbCompliance	main.ahbMonitor_ABV.	inactive	0	0
Address...	ahbCompliance	main.ahbMonitor_ABV.	inactive	0	0
Always...	ahbCompliance	main.ahbMonitor_ABV.	inactive	0	0
Bursts...	ahbCompliance	main.ahbMonitor_ABV.	inactive	0	0
Bursts...	ahbCompliance	main.ahbMonitor_ABV.	inactive	0	0
Busy...	ahbCompliance	main.ahbMonitor_ABV.	inactive	0	0
ControlMustBeConstantDuri	ahbCompliance	main.ahbMonitor_ABV.	inactive	0	0
ControlMustBeConstantWhe	ahbCompliance	main.ahbMonitor_ABV.	inactive	0	0

Displayed: 34 Inactive: 34 (100%) Begun: 0 (0%) Finished: 0 (0%) Failed: 0 (0%)

Filters

Name Filter: \* Module Filter: \* Instance Filter: \*

Display assertions with value:  Inactive  Begun  Finished  Failed

Assertion Browser displays assertion states and counts

Assertions are fully integrated as first class objects

SimVision: Set Breakpoint

Breakpoint Name: (optional)

Break based on...

Time | Line | Signal | Condition | Assertion

+ Add Browser

Assertion: main.ahbMonitor\_ABV.ahb\_compl.AddrHeldWhenMasterBusy

Break on: State change

State change  Undefined  Inactive  Begun  Finished  Failed

Stop only if this condition also is true:

Execute the following commands at each break:

Always stop at breakpoint

Keep breakpoint

OK Cancel Apply Help

# Incisve Debug Environment



**Double-clicking on Assertion name brings up the assertion source**

Assertion Name	Component	State
NeverMoreThan16WaitStates	ahbCompliance	failed
AddressNotAlignedToTrans	ahbCompliance	begun
BurstIsNotTooLong	ahbCompliance	finished
BusyAndSeqNeverFollowNo	ahbCompliance	finished
ReadyMustBeZeroDuringFir	ahbCompliance	begun

```
390 // ((ahbSize_i == bits16) && (ahbAddr_i[0] == 1'b0)) ||
391 // ((ahbSize_i == bits32) && (ahbAddr_i[1:0] == 2'b00)) ||
392 // ((ahbSize_i == bits64) && (ahbAddr_i[2:0] == 3'b000)) ||
393 // ((ahbSize_i == bits128) && (ahbAddr_i[3:0] == 4'b0000)) ||
394 // ((ahbSize_i == bits256) && (ahbAddr_i[4:0] == 5'b00000)) ||
395 // ((ahbSize_i == bits512) && (ahbAddr_i[5:0] == 6'b000000)) ||
396 // ((ahbSize_i == bits1024) && (ahbAddr_i[6:0] == 7'b0000000))
397
398
399 /* Behavior: always the maximum number of wait states is 16 */
400 // sugar property NeverMoreThan16WaitStates = always
401 //   {(ahbReady_i == 1); (ahbReady_i == 0)} | =>
402 //   {{(ahbReady_i == 0) [*0..15]}; (ahbReady_i == 1)}
403 //   abort (ahbResp_i != OKAY);
404
```

**Assertion failures shows up as events on a probe.**

**Assertion activity is recorded as a transaction**

**Goto-cause on failure event brings up the assertion source**

# PSL/Sugar Consortium

... formed at DAC'03



- PSL/Sugar endorsement and adoption is growing rapidly...



<http://www.pslsugar.org/>



“A Vendor-Neutral standard that’s open to anyone to use and/or build tools around so customers are not locked to a single vendor”

# Summary



- PSL is production proven and available today
- PSL and SVA are complementing each other
- SVA properties can be translated to PSL easily
  - for export to other languages
  - for clarity
- Customer have a choice
  - use PSL today
  - learn about SVA when required

**cādence**<sup>®</sup>

**cādence**<sup>®</sup>