

# Eliminating the Design and Verification Bottlenecks



## Tens of thousands of assertions free - Learn in under an hour

**Assertive Design** blends new and existing technologies to enable real assertion-based design and verification for the first time, with

- Automatically created check/error assertions
- Automatically created coverage assertions
- Built-in full functional coverage
- New approach to IP development and reuse.

This software suite provides the highest quality RTL design and system verification using assertions without the need for separate assertion development. This allows for cost reduction and quality benefits without the time and expense of developing the assertions.

**Assertive Design's** tools add value to the user, offering

Design

- Efficient coding
- Early and predictable bug reports
- Higher quality code
- Error reports that point to bug source
- Early completion of tested code.

IP

- Simplified IP interconnect
- Mechanisms to ensure correct usage
- Assertions-to-go (™)
- Full functional coverage
- Bottom-up abstraction.

Verification

- Automated block-level assertions
- Effective constrained random testing
- Built-in full functional coverage
- Focus on architectural verification
- Lower cost, higher quality verification.

Features

- Verilog-like syntax - learn in an hour
- Graphical debugger
- Graphical functional coverage explorer
- Mix and match with legacy design components
- Assertions may be packaged with a component.

**Assertion benefits** the user in two distinct areas – checking and coverage. Check assertions fire on unexpected events and errors, while coverage assertions fire on expected activity.

Check assertions provide a drastic reduction in time required to complete the design and verification tasks. Bugs are found earlier, and the assertion reports point to the source of the failure rather than to the symptom. Software development is developed with less effort since the assertions will identify stimulus and configuration errors early.

Coverage assertions will greatly increase the quality of the design by providing metrics for the verification. Functional coverage with coverage assertions is more accurate and higher level than code coverage. Now functional coverage-driven verification can be run with much more detailed coverage information available.

### The Assertive Design Tool Suite

- DPSL(™): The DesignPSL(™) language
- DGEN: The DPSL(™) source analyzer
- AGEN: The assertion generator
- ASIM: The assertion simulator
- DVIEW: The graphical debugger
- RICGEN: The Reusable Interconnect RIC(™)
- AVIEW: The assertion results processor

For more information, please contact:

Angelo Guadango, President  
(617) 283 2280

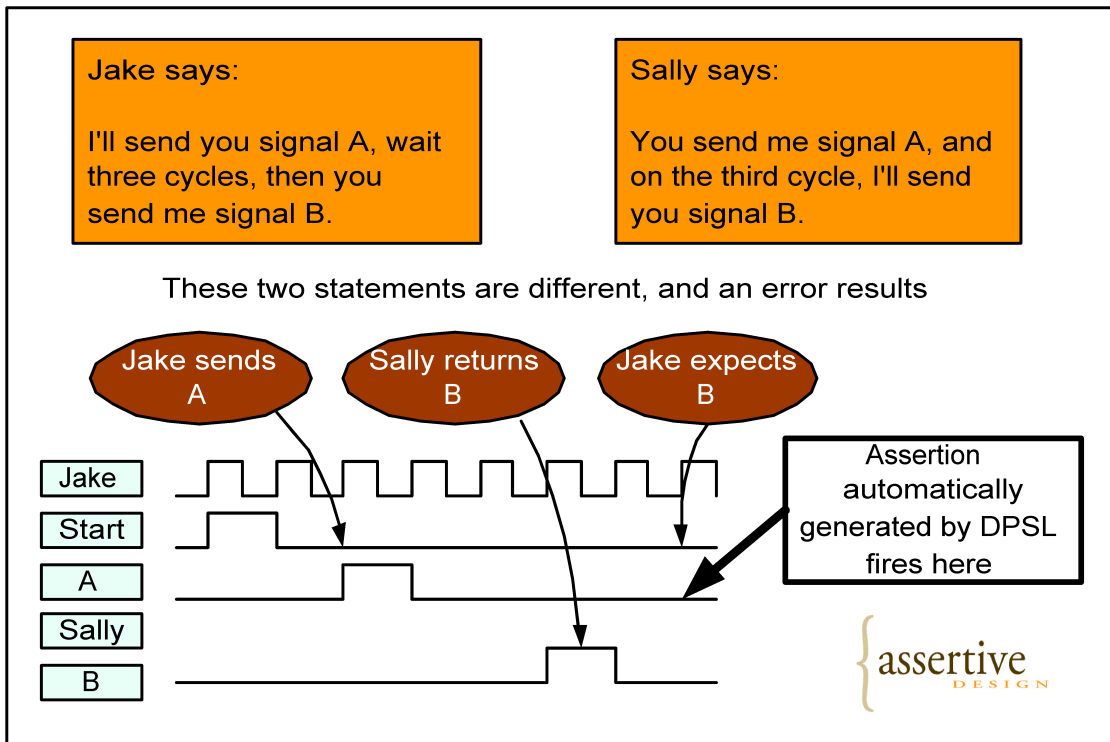
angelo@assertivedesign.com  
<http://www.assertivedesign.com>

# Eliminating the Design and Verification Bottlenecks



Tens of thousands of assertions free - Learn in under an hour

Assertive Design's tools catch design errors automatically



Assertive Design's DesignPSL<sup>(tm)</sup> automatically creates, runs, and checks assertions without designer intervention as they go through their normal design process. No additional work is required on the part of the designers

## The Assertive Design Tool Suite

- DPSL<sup>(tm)</sup>: The DesignPSL<sup>(tm)</sup> language
- DGEN: The DPSL<sup>(tm)</sup> source analyzer
- AGEN: The assertion generator
- ASIM: The assertion simulator
- DVIEW: The graphical debugger
- RICGEN: The Reusable Interconnect RIC<sup>(tm)</sup>
- AVIEW: The assertion results processor

For more information, please contact:  
Angelo Guadango, President  
(617) 283 2280  
angelo@assertivedesign.com  
<http://www.assertivedesign.com>

# An Assertive Design Evaluation Comparison



## A USB 2.0 Slave in DesignPSL

As an interesting testcase for Assertive Design's new DesignPSL<sup>(tm)</sup> tool suite a USB 2.0 Slave core has been designed that exactly matches the core available from OpenCores.org except that this is written in DesignPSL<sup>(tm)</sup> and a simple testbench is provided. The testbench may be used to test either the DesignPSL<sup>(tm)</sup> version or the original Verilog version directly from OpenCores.org. Some interesting comparisons are:

Code Comparison	DesignPSL	Original USB	
Lines of source code:	3,444	-	DPSL is smaller, but the generated Verilog is similar in size. DPSL gives a greater than 1:1 ratio of assertions to lines of code
Lines of Verilog RTL:	5,063	5,122	
Number of Check Assertions:	6,094	0	
Number of Cover Assertions:	4,149	0	
<b>Simulation Comparison of a 152,570 cycle test</b>			
Number of Check assertions run:	51,011,626	0	Assertions are pre-compiled and tuned for the design to minimize assertion simulation overhead
Number of Cover assertions fired:	533	0	
Seconds of simulation runtime:	67	34	
<b>Synthesis Results</b>			
Size of Design in Gates:	33,917	35,249	Design speed benefits from consistent Verilog output designed for optimal synthesis results
Cycle Time in ns:	7.9	8.5	

The vast number of check and coverage assertions automatically generated reduces the amount of time it takes to detect and isolate a bug by a factor of 20. The time to fix a bug in a design is reduced by at least a quarter.

The design itself has fewer bugs. Since bugs per line of code is a constant, designing with DesignPSL<sup>(tm)</sup> results in almost 50% fewer bugs than the original. The intent of this example was to exactly reproduce the module hierarchy of the OpenCores.org example. However, the use of DesignPSL<sup>(tm)</sup> allows for easier removal of much of the hierarchy without any loss in the ability to understand the design. This would allow for substantially fewer lines of DesignPSL<sup>(tm)</sup> code than is seen above with even fewer bugs.

DesignPSL<sup>(tm)</sup> is easier to write and maintain than Verilog. For the designer the design process more closely matches what they are actually trying to accomplish with the design while at the same time giving them the full control they expect from an RTL language. The design itself will take about half the time it does today.

Using DesignPSL<sup>(tm)</sup> for the design of something such as this USB 2.0 core will remove more than 85% of the overall project time required to get a working module integrated and working in a larger design. Larger designs see even greater reductions in time since more and more of the time gets devoted to verification.

### The Assertive Design Tool Suite

DPSL<sup>(tm)</sup>: The DesignPSL<sup>(tm)</sup> language  
DGEN: The DPSL<sup>(tm)</sup> source analyzer  
AGEN: The assertion generator  
ASIM: The assertion simulator  
DVIEW: The graphical debugger  
RICGEN: The Reusable Interconnect RIC<sup>(tm)</sup>  
AVIEW: The assertion results processor

For more information, please contact:  
Angelo Guadango, President  
(617) 283 2280  
angelo@assertivedesign.com  
<http://www.assertivedesign.com>

# Redefine the power of RTL design



## Reusable InterConnect (RIC) (tm)

**Assertive Design's** RIC<sub>(tm)</sub> gives IP designers unprecedented control and visibility over RTL design and use. At the same time, IP users can integrate the reusable interconnect (RIC) <sub>(tm)</sub> RTL with complete ease, selecting those access methods and sub-blocks that meet their needs, without paying a size or performance penalty for unused sections of IP. No longer is IP only at the module level!

**Assertive Design** blends new and existing technologies to enable real assertion-based design and verification for the first time. RIC<sub>(tm)</sub> provides a new level of power and ease of use for both the IP designer and user:

- IP and reusable RTL can be developed and modified without affecting users
- User integration of RIC<sub>(tm)</sub> RTL is far easier than with current IP blocks
- Built in check and functional assertions generated automatically, can be shipped with resulting Verilog

**Assertive Design's** RIC<sub>(tm)</sub> benefits to the system designer

Real encapsulation of IP

- Bus definitions specified within RIC
- Interface definitions specified within RIC<sub>(tm)</sub>
- User selectable interfaces
- User protected from interface and IP details

Debug

- Errors detected and displayed automatically
- Integrated error, coverage, and design display
- High-level debugging
- Contextual information provided

Simple user access to IP

- Easy integration of IP
- IP assertions available to the user
- IP sub-components only instantiated when selected
- User not affected by IP modifications

IP & Reuse

- Complete control over an IP block
- Designer specifies and controls user access
- Built-in functional and error assertion generation
- Modular design integrates user-selected components.

**Assertive Design's** RIC<sub>(tm)</sub> interface allows busses, ports, and control logic to be defined as a part of the IP. A designer is able to provide multiple interfaces and support logic; only those that are chosen by the user will have assertions and gates generated for them. As always with the Assertive Design tool suite, check and cover assertions are generated automatically for the IP block. With RIC<sub>(tm)</sub> they are generated based upon the specific usage. Assertions-to-Go (tm) allow the assertions to be packaged with Verilog IP.

### The Assertive Design Tool Suite

- DPSL<sub>(tm)</sub>: The DesignPSL<sub>(tm)</sub> language
- DGEN: The DPSL<sub>(tm)</sub> source analyzer
- AGEN: The assertion generator
- ASIM: The assertion simulator
- DVIEW: The graphical debugger
- RICGEN: The Reusable Interconnect RIC<sub>(tm)</sub>
- AVIEW: The assertion results processor

For more information, please contact:

Angelo Guadango, President  
(617) 283 2280

angelo@assertivedesign.com  
<http://www.assertivedesign.com>

# How Does DesignPSL Cut 75% of Your Front-End Time?



**Imagine that the first time you compile your RTL you now have:**

- Man-months or years of functional coverage instrumentation, tailored specifically for your design.
- Man-months or years of block-level functional checks to verify your design.
- You start at the first compile, with 3/4 of the verification task already done for you.
- Best of all, when you change your design, this part of the verification suite changes with it automatically

**DesignPSL attacks every aspect of the RTL design and verification problem:**

- More efficient design entry - more functionality per line without behavioral abstraction.
- Immediate feedback of block-level functional bugs - often starting with your first compile.
- Continuous checking of every aspect of the design, finding bugs automatically as you run even simple tests
- Bugs are found and reported at the source - file, line, column, of **your** source
  - No working backwards from an error report in a verification test or manually generated assertions
  - Verifier now spends little or no time determining if a failure is caused by the test or the design
  - Designer does not have to reverse-engineer the test. The assertion tells them what is wrong.
- Find bugs your tests would have missed
  - Targeted tests for functionality often miss violations of designer assumptions.
  - Tests are looking for a specific response to a specific stimulus, rather than all possible problems.
  - DesignPSL is testing all designer assumptions all the time.
- Run fewer tests with full functional coverage feedback, to tell you if your test is working right away.
- Make your random testing far more efficient, with dynamic coverage information driving random constraints.
- Reduce or remove the need to write brute-force iterative tests of block-level functionality.
- Make synthesis easier with consistent, designer for synthesis, RTL output of DesignPSL
- Writes 3/4 of your verification suite for you - the block level tests and functional coverage instrumentation.

## **The Assertive Design Tool Suite**

DPSL<sup>(tm)</sup>: The DesignPSL<sup>(tm)</sup> language  
DGEN: The DPSL<sup>(tm)</sup> source analyzer  
AGEN: The assertion generator  
ASIM: The assertion simulator  
DVIEW: The graphical debugger  
RICGEN: The Reusable Interconnect RIC<sup>(tm)</sup>  
AVIEW: The assertion results processor

For more information, please contact:

Angelo Guadango, President  
(617) 283 2280

angelo@assertivedesign.com  
<http://www.assertivedesign.com>