The NC-SC Simulator runs architecturally accurate transaction-level models 100 times faster than RTL. It is ideal for architecture analysis, system-level verification, and embedded software development, and it supports transaction recording and analysis for the SystemC® Verification Library.
**NC-SC Simulator**

The NC-SC Simulator is fully compatible with the Incisive platform and with the Open SystemC Initiative® (OSCI) reference implementation. It is also the first commercial implementation of the SystemC Verification Library. Transaction recording, viewing, and analysis capabilities enable teams to maximize functional coverage across the entire design with 100 times the simulation performance of RTL. Most significantly, and unlike the OSCI reference implementation, the NC-SC Simulator enables you to prepare SystemC® models and testbenches for multilanguage simulations using the Incisive Unified Simulator. This capability lets you re-use the system-level testbench and transaction-level models for functional verification of your RTL implementation.

**Benefits**

- Offers 100 times the performance of equivalent RTL
- Speeds system-level verification
  - Allows architectural performance analysis
  - Enables earlier and faster embedded software development
  - Advances evaluation and design-in
- Reduces testbench development time with transaction-level support
- Decreases debug time through transaction viewing and debug environment
- Enables functional coverage with transaction analysis

**Features**

**Fully compatible with Incisive Unified Simulator**

The NC-SC Simulator provides two integration paths to the Incisive Unified Simulator. It allows SystemC modules to be instantiated within HDL modules and vice-versa. This is a key feature that prepares the design for multi-language (SystemC, Verilog®, and VHDL) simulation in the Incisive Unified Simulator. It also allows SystemC models to be connected to HDL through out-of-module-references. Testbench components, like transactors that interpret transactions to signals and vice-versa, use this feature to observe and stimulate internal HDL design elements from SystemC models. Development of these transactors can be accomplished without having an HDL simulator in place.

**Simulation and debug environment**

The simulation and debug environment makes it easy to manage multiple simulation runs when analyzing the design and testbench. Its transaction/waveform viewer helps to quickly trace design behavior back to the source. Its source viewer lets designers examine their design and access results in both interactive and post-processing debug modes.

**Functional coverage analysis**

Powerful transaction analysis capabilities can be used to measure and improve functional coverage. A form-based interface on top of a Tcl/Tk interpreter lets users quickly write queries to analyze and post-process large simulation runs. The simulation and debug environment displays pie charts, Gantt charts, histograms, and other forms of statistical visualization of the data.

Transaction analysis features make it possible to characterize a directed random or fully random test suite, verifying that all of the specification’s functional criteria have been covered. For example, in a communication device it is possible to determine if all router paths were exercised, or to ensure that all router inputs were simultaneously sending packets to the same output. It is also possible for a processor to verify that specific types of interrupts occurred during specific operations.

**Architectural and performance analysis**

NC-SC transaction analysis capabilities can also be used for architectural and performance analysis. The Tcl interface and transaction exploration features are useful for validating the performance of components against system specs, such as validating an arbiter against latency and quality of service specs. Results can be displayed in a variety of report formats that include graphics and tables.

**Ideal for hardware/software co-design**

Architecturally accurate transaction-level models describe each memory location and communication protocol in the system. Using the NC-SC Simulator to run verified transaction-level models 100 times faster than RTL is a perfect fit for embedded software development concurrent with RTL development. Embedded software can provide valuable testbench components for RTL verification in the fully compatible Incisive Unified Simulator, and performing hardware/software co-design in the same simulation environment enables the design team to identify performance bottlenecks early enough to make efficient hardware changes instead of settling for software patches.

**Interactive command support**

Support for key Tcl commands, like describe, value, scope, and probe, enables users to access simulation information interactively. They can interact with their designs through a single interface and a single set of commands. Test and analysis scripts can be reused in the Incisive Unified Simulator, whether the design is SystemC, Verilog, or VHDL.

**Compatible with OSCI SystemC Initiative**

To advance SoC design and verification, Cadence has been a long time and leading member of the Open SystemC Initiative (OSCI). This initiative has standardized on one SystemC language that is capable of describing hardware systems at the behavioral level, transaction level, and register-transfer level. Transaction-level models are especially powerful because they allow for a complete specification of the architecture and communication mechanisms, but without the RTL detail of each wire and hardware signal. Abstracting away the detail enables the NC-SC simulator to simulate transaction-level models 100 times faster than equivalent RTL.
First commercial implementation of the SystemC Verification Library

The OSCI extended SystemC language with powerful verification technologies, including new data structures, enhanced concurrency capabilities, and constrained randomization. The OSCI accepted the reference implementation referred to as the SystemC Verification Library (donated by Cadence). This enables a single language to be used for the creation of a system-level design and a comprehensive testbench consisting of verification components like stimulus generators, monitors, checkers, and transactors. The NC-SC Simulator supports all of the facilities necessary to enable system- and block-level verification at both the transaction- and register-transfer levels, so that the testbench can be reused for RTL verification and the transaction-level models can be reused as reference models.

Extensions to the SystemC Verification Library

The NC-SC Simulator supports extensions to the SystemC Verification Library, including dynamic thread creation, concurrent thread synchronization (using barriers, mutexes, semaphores, and signals), arbitrary expressions of these synchronization primitives, and additional verification data structures (associative array, smart queues, and more).

Verification scripts and templates

The NC-SC Simulator includes verification scripts that manage simulation runs and templates for generating testbench components. For example, driver scripts are provided to manage the build and simulation processes, and template generation (wizard) scripts help users jumpstart the development of transactors and other verification components.

Specifications

OSCI-compliant SystemC Version 2.1

- Supports design at the transaction level, behavior level, and register-transfer level
- Communicates through transactions or through signals

OSCI-compliant SystemC Verification Library

- Basic randomization features
- Independent randomization objects (objects may be changed without affecting other random components)
- Simple constraint sets (keep only and keep out)
- User-defined distributions and distribution ranges
- Ability to randomize built-in types as well as user-defined composite types (structs) and user-defined enumerated types
- Complex constrained randomization
  - Constraint specification using Boolean and arithmetic operations
  - Multivariable constraint solving using any number of independent variables
  - Supports hard and soft constraints
  - Supports hierarchical constraints
  - Ability to selectively disable/enable randomization on specific fields of a constraint
  - Ability to overload the value generate for complete pre-generate and post-generate control
- Transaction recording
  - Records transaction type information, along with the beginning and ending time of the transaction
  - Transaction begin/end times can be in the past (when a transaction has been recognized) or the future (when the transaction is scheduled to end)
  - Associates attributes with a transaction, the values of which can be recorded at any time during the transaction

Figure 2: NC-SC simulation and debug environment
− Records relationships between transactions (the predecessor-successor relationship of a split transaction, or the parent-child relationship of a communication protocol)
− Records zero-time error transactions

• Data structures for modeling memories, queues, FIFOs, and for writing a transaction-based testbench
• Sparse arrays for modeling large (gigabyte) memories that require only a small number of addresses for a given simulation run
• List structures (queues, stacks, FIFOs) for managing verification objects that flow through a system
• Introspection facility allows CAD groups to create special verification tools unique for their particular problem (for instance, a score boarding tool can be created using introspection along with knowledge of the type of score boarding that is important for a particular application domain)

Extensions to the SystemC Verification Library

• Additional concurrency synchronization mechanisms
  − Barriers to bring threads together at a specific simulation point
  − Synchronization lists provide both stack and queue synchronization
  − Complex event expression syntax to manage waits on multiple resources, with the ability to figure out which wait causes a thread to reactivate
  − Dynamic thread creation
  − Ability to suspend/resume and kill dynamic threads
  − Ability to wait on thread termination
• Support utilities
  − Wizard for generating transactor descriptions from Verilog or VHDL input

− Wizard for generating specialization of user defined enumerations and structures
− Driver utility to manage the compile and link steps of test generation, and the compile, elaborate, and simulate steps of final simulation
− Visualization and analysis of transaction databases can be done using the transaction analysis features in both the NC-SC Simulator and the Incisive Unified Simulator

Results analysis

• Debug and GUI
  − Integrated source code debugging
  − gdb debugger console
  − Thread debugging
  − Integrated support for IBM Rational PurifyPlus for memory corruption and leak detection, performance profiling, and code coverage analysis
  − Hierarchy navigation and browsing
  − Probing and viewing of values during simulation
  − Waveform window
  − Log signal and transaction data to one database
  − Unified transaction/signal display

• Tcl/Tk scripting for functional coverage analysis
  − Generate raw counts of types of transactions in a simulation run

− Do cross-product analysis between transaction types or between attributes of transactions
− Confirm concurrent behavior of the design (simultaneous burst writes of a dual-port memory)
− Performance analysis of a real-time system

Cadence Services and Support

• Customer-focused solutions that increase ROI, reduce risk, and achieve your design goals faster
  − Collaborative approach and design infrastructure—virtual teaming
  − Proven methodology and flow tuned to your design environment
  − Design and EDA implementation expertise

• Product and flow training to fit your needs and preferred learning style
  − More than 80 instructor-led courses—certified instructors, real-world experience
  − More than 25 Internet Learning Series (iLS) online courses

• Cadence customer support that keeps your design team productive
  − Cadence applications engineers provide technical assistance
  − SourceLink® online support gives you access to software updates, technical documentation, and more—24 hours a day, 7 days a week

Supported Platform Matrix

<table>
<thead>
<tr>
<th>Operating Systems</th>
<th>Solaris 8, 9 (32-, 64-bit)</th>
<th>HP-UX 11.0, 11i (32-, 64-bit)</th>
<th>RedHat Enterprise Linux 2.1, 3.0 (32-bit)</th>
<th>IBM AIX 5.1 (32-bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilers</td>
<td>gcc 3.2.3, Sun Studio 8</td>
<td>gcc 3.2.3, aCC</td>
<td>gcc 3.2.3</td>
<td>VA 6.0</td>
</tr>
</tbody>
</table>

Cadence is transforming the global electronics industry through a vision called EDA360. With an application-driven approach to design, our software, hardware, IP, and services help customers realize silicon, SoCs, and complete systems efficiently and profitably: www.cadence.com

©2012 Cadence Design Systems, Inc. All rights reserved. Cadence, the Cadence logo, Connections, SourceLink, and Verilog are registered trademarks, and Incisive is a trademark of Cadence Design Systems, Inc. Open SystemC Initiative and SystemC are registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission. All others are properties of their respective holders. 20669/0045 1/12 XX/MVC/CS/PDF