

Raising the Level of Abstraction Reduces System-on-Chip Verification

March 2004

Rindert Schutten

Introduction

The perils of verifying today's complex systems on chips (SoCs) have been heavily touted in the technical media. Accounts of companies spending more than 70 percent of their project budgets on making sure that their chips meet specifications and perform as intended are numerous.

There has been an increased focus on verification technology in the last few years to address this "verification crisis." Indeed, progress has been made. For example, random-based stimulus-generation techniques, techniques to gather coverage metrics, and so forth are emerging technologies that have helped turn the "art" of verification into more of an engineering discipline. However, technology nodes of 130nm, 90nm and smaller enable astonishingly complex designs that cannot effectively be verified through current verification techniques. New, more-effective methodologies are required to deal with this complexity. One fundamental methodology that addresses complexity comprises technologies and flows enabling engineers to design and verify on higher levels of abstraction than RTL.

Raising the level of abstraction is a common approach for dealing with complex SoC design problems. Addressing a problem on a "higher level of abstraction" simplifies the problem because not all details need to be taken into account. A designer can focus on the key issues. The key to success is the choice of the level of abstraction. On one hand, the designer has to allow for sufficient detail to be ignored, so that the problem is simplified and the designer is not bogged down by detail. On the other hand, the designer must make sure that the solutions conceived on this "higher level" still solve the problem when all the details of the implementation level are included. The process is always a balance between the two.

This paper discusses a methodology that shows how abstract, transaction-level, modeling techniques tightly linked with advanced RTL-verification techniques offer a way to break through the complexity associated with the verification of today's SoCs. This paper also explains how to best use multi-level interfaces in SystemVerilog that enable refinement from transaction level to RTL while linking in assertions to provide for a powerful methodology of correctness by construction.

SoC Verification Today

The way mainstream design and verification is handled today is shown in Figure 1.

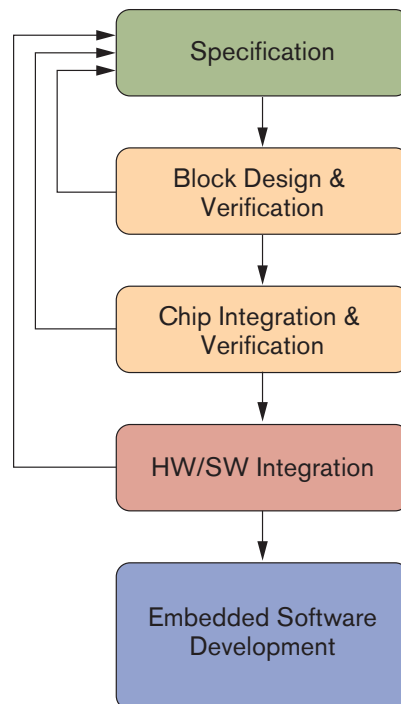


Figure 1: Current verification flows typically perform HW/SW integration near the end of the design project.

A design team starts with the development of a specification, usually a natural language document that describes the functionality to be achieved by the target chip. Typically this specification includes a high-level architectural description, identifying key modules, buses, processors, DSPs and so forth. This specification is then passed on to various sub-teams or individual designers who are assigned various blocks. In most cases designers are also responsible for the verification of their blocks. In practice this means that some testbenches and tests are developed to gain confidence in the correctness of each block before releasing it to the chip integration phase. Then, after the RTL design phase where most of the blocks have been “completed” and are “up and running,” the chip-level verification and debugging process begins.

A second team—the verification team—is responsible for the verification of the chip. This team takes the same specification that was given to the designers, creates the testbenches that provide the stimulus and develops tests to exercise the design. To provide the stimulus, the externally-visible interfaces are connected to a smart testbench and large amounts of data, often pseudo-random in nature, are sent through the chip. Outputs are monitored for expected results and code coverage measured. When bugs are found the design module is passed back to the designer for repair. After the bugs have been fixed the design is again turned over to the verification team for more tests, until another bug is found. For most chips this process takes much iteration and often includes some specification revision and refinement, which may lead to changes to blocks that were believed to be already completed.

When the whole chip works the software component is integrated. Even though co-verification techniques may be used in addition to sign-off simulation models or cycle-accurate models, there is typically not enough performance or capacity available to run any significant amount of software. In many cases, long, realistic data streams need to be processed to fully cover the functionality. Especially in the networking and multimedia domains long data streams are required but are hard to run. Hence, there is a high probability that design bugs will not be found before going to tapeout. Often because of specification changes required in this phase—perhaps because chip performance goals were not being met—an entire re-design of multiple RTL modules is needed. Trying to validate the architecture at RTL can be an expensive and time-consuming proposition.

Raising the Level of Abstraction

To improve productivity, eliminate frequent specification changes and redesign, and enhance the quality of results, we introduce a higher level of abstraction in the form of a transaction-level model of the chip, preceding the hardware development tasks. This model is used to fully analyze the selected architecture, including buses, I/O and processor(s). It helps to ensure up-front that the bus infrastructure meets the target performance and to ensure a correct partitioning of the functionality that is implemented by the hardware and software modules. This high-level model serves two purposes:

- Provides an executable specification of the architecture that forms the starting point for the hardware designers
- Serves as a software execution platform of the target chip to develop the hardware-dependent software

With these deliverables as a goal of this modeling effort, transaction-level models need to exhibit cycle-accurate behavior of the bus infrastructure, while the functional blocks connected to this bus are latency approximate. Our experience shows that models with these properties simplify the problem. However they offer sufficiently accurate analysis to draw solid macro-architectural conclusions concerning bus architectures (including performance analysis) and hardware/software (HW/SW) partitioning that hold when the architecture is refined into an RTL implementation. This improved flow is shown in Figure 2.

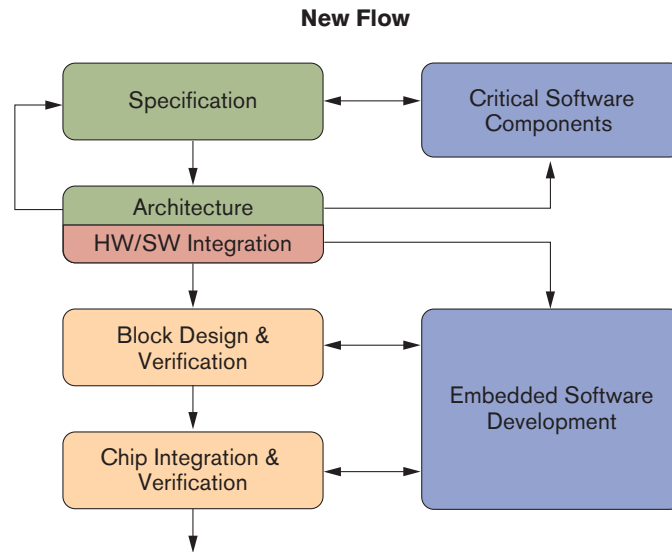


Figure 2: Moving HW/SW integration up-front in the flow makes it an integral part of architecture design and enables software design to start early.

There are several ways that a transaction-level architecture model of the target chip helps to create a more effective overall design and verification flow. Let's explore these in more detail as follows:

- Through abstract, transaction-level models of the target chip designers can explore the architecture, that is, analyze the effects of realistic traffic that the chip needs to handle and ensure that the proposed architecture, including HW/SW partitioning, provides the required performance at minimal cost.
- It moves the HW/SW integration phase up-front in the overall SOC design process. This really has two benefits: critical software development can start earlier, making the software component of the chip less likely to become a delaying factor, and critical interaction between hardware and software is dealt with at the architecture level, when changes are still "cheap" to incorporate.

This solution differs from existing HW/SW co-verification products on the market today. With these products designers typically need to have the RTL for the complete SoC available; only the processor is abstracted to a higher instruction-set simulator (ISS) level. Although ISS processor models speed up the overall simulation, the large amount of RTL present in the simulation does not deliver the required execution performance to actually verify software content in the design. Our proposed solution here elevates the complete SoC to the transaction level of abstraction and therefore yields the required performance.

Another benefit of this approach is reuse of the transaction-level stimulus environment. To perform architectural analysis and run software on realistic data streams, a high-level stimulus environment must be created. This environment often is reused as a stimulus environment during the RTL development phase as well. However, this environment typically has limited value in finding deep or corner-case bugs in the RTL implementation. Why is this?

In the architecture-exploration phase the stimulus focuses on the intended functionality. Realistic stimulus patterns (often highly repetitive), for example, sending 1,000 variable-length Ethernet frames through an Ethernet switch, must be created to observe the effects of this traffic on the architecture in terms of resource utilization and performance. The results of this analysis are then used to make HW/SW tradeoffs. The characteristics of the data required to perform this analysis differ significantly from the type of stimulus required to reach all corner cases. Transaction-level models hide detail; that's why they are fast to develop and provide high-execution performance. The stimulus environment operates on this same abstraction level and therefore does not address the pin/signal-level accuracy and control required to verify the much more detailed RTL models.

Verifying RTL implementations is about finding bugs and ensuring that a design is bug free before tape out. To reach all the corner cases, unintended usages, various exceptions, (shared) resource management, bus contentions, and so forth requires a more sophisticated stimulus and analysis environment. This is exactly where advanced RTL verification techniques, built on random-stimulus generation, design-for-verification, and coverage make the difference before tape out. Transaction-level modeling and modern RTL verification methodologies are complementary techniques.

Creating Transaction-Level Models

Abstract, transaction-level models can be written in any language suitable for the task. However, in practice there are only two choices – SystemC™ or SystemVerilog because both languages have specific capabilities to support transaction-level modeling. Key to writing transaction-level models is the ability to abstract over communication and to express complex algorithms efficiently. SystemC's notion of channels and SystemVerilog's notion of interfaces both permit the encapsulation of interface details, while supporting complex data types, including the control structures to implement complex algorithms, and supporting built-in handling of concurrency.

The combination of these properties enables designers to quickly describe in a cycle-accurate manner the behavior (without having to resort to signal-level descriptions) of the complete SoC infrastructure, while connecting behavioral/functional models of the blocks making up the SoC.

In practice, which language to use depends on the experience and preference of the chip architect. A team already familiar with C++/SystemC may prefer to develop the transaction-level models in SystemC, while a team fluent in RTL design wanting to move up in abstraction may prefer SystemVerilog. As shown in Figure 3, the availability of efficient integration between Synopsys' verification platform tools, System Studio and VCS™, makes the import and co-simulation of SystemC models into a SystemVerilog environment straightforward. Therefore the choice of language to implement the transaction-level model becomes mainly a personal or team preference.

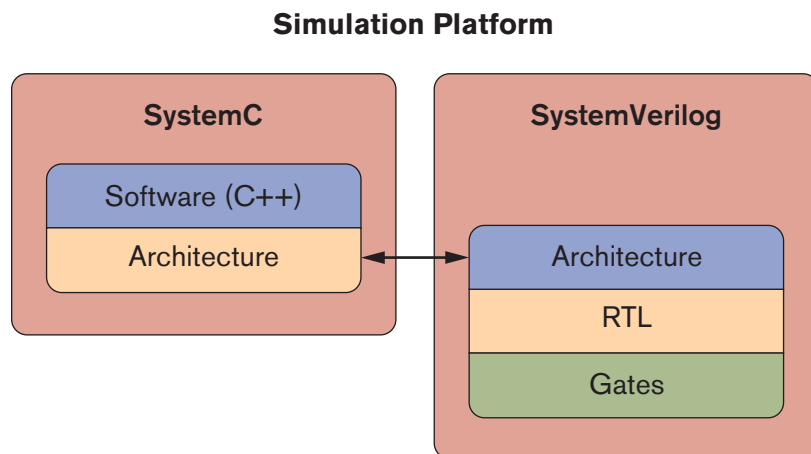


Figure 3: Synopsys' Discovery™ Verification Platform supports both SystemC and SystemVerilog to enable a seamless flow from concept to gates.

Moving Down to RTL

Import capability and corresponding co-simulation is crucial when moving to RTL. The flow from transaction level to RTL requires the step-wise refinement of the blocks and the chip infrastructure. With such a capability in place a designer can effectively replace high-level transaction models by the RTL version and run the same (or shortened) test suite, albeit often at a lower speed because of the greater detail present in the RTL model.

Typically, this process is performed on a single block at a time to maximize performance and to ensure the ability to quickly pinpoint any discrepancies in the new RTL model relative to the transaction-level model. The refinement from transaction-level models into pin-accurate, synthesizable RTL models is performed using SystemVerilog because it covers both abstraction levels, transaction and RTL. During this phase, designers add assertions, create or reuse (pin-level) interface constraints, and complement the transaction-level stimulus with smart stimulus automatically generated from the interface constraints to ensure full coverage of all the corner cases of the block. As shown in Figure 4, this process enables formal analysis as well as simulation for the RTL blocks.

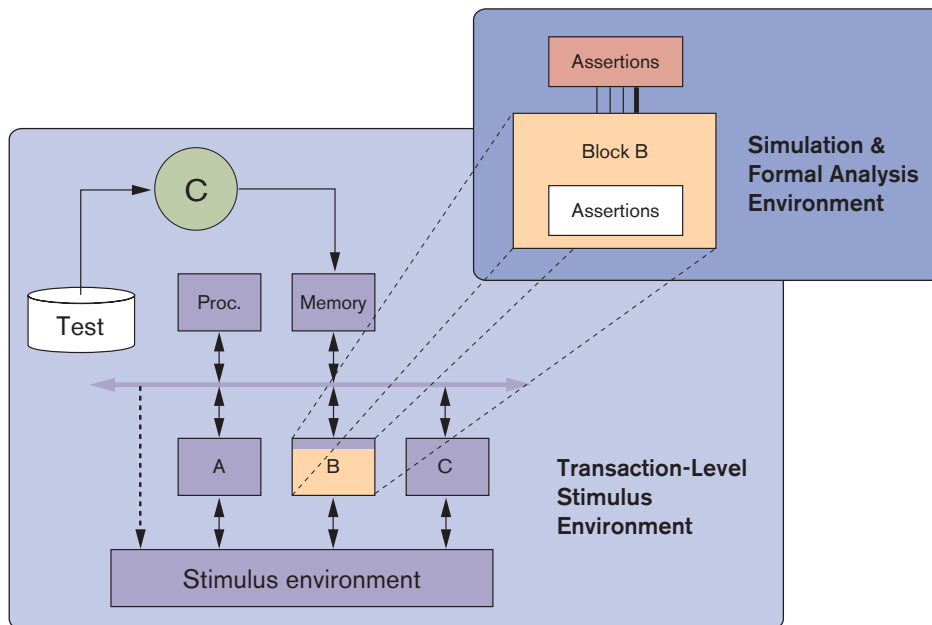


Figure 4: When refining the architecture into RTL, individual blocks are verified and then included in an overall chip verification where the transaction-level model optionally can be used as a reference model.

SOHO Router Example

Let's look at a real design example to ensure we keep our feet on the ground. A four-port DSL router is a popular device that can be found in many homes to connect multiple PC's to a single DSL connection. In Figure 5 we show a simplified diagram of its functionality. On the DSL side it supports the IP over ATM protocol on the DSL line, while on the Ethernet side the IP packets are encapsulated in an Ethernet frame. The router performs the assembly/disassembly of the IP packets and other IP and ATM protocol functions, and moves packets coming on the DSL line to one of the Ethernet ports, and vice versa.

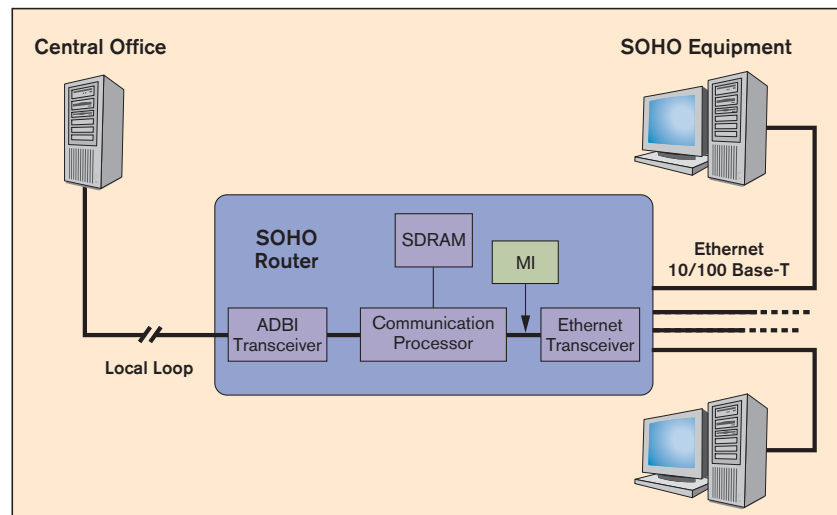


Figure 5: The typical functionality of a SOHO router.

In designing this device, we follow the steps for an AMBA-based solution as follows:

1. Create the transaction-level model of the target chip in SystemC. We use the Synopsys DesignWare® SystemC AMBA bus transaction-level models connected to the behavioral-level models of the four main blocks—the ATM control block, a memory model, the assembly/disassembly unit, and (replicated four times) the Ethernet MAC block.
2. Connect the traffic sources, IP generators/analyzers stacked on top of the ATM generators/analyzers from the Synopsys DesignWare Telecom Workbench library, to generate realistic traffic (we have chosen to abstract over the ADSL physical line and “bypass” the functionality of encoding/decoding ATM over ADSL). Telecom Workbench library components are written in C++.
3. After completion of the platform model in Synopsys' System Studio, perform the analysis of bus utilization and buffer-management issues by running realistic data through the model using the analysis features of the tool. As a result, we identify our optimal partition for which functionality to implement in hardware and which to implement in software.
4. We replace the SystemC model that handles the assembly/disassembly functionality by an actual ARM 946 ISS processor model wrapped in SystemC. This results in a complete virtual system prototype of the target chip to be passed on to the software-development team, enabling them to develop or port the rest of the software for this SOHO router. This model executes at speeds of 100 Kcps, fast enough to run significant software development and analysis.
5. Refine the high-level blocks. Using DesignWare AMBA Bus Functional Models (BFMs), create automatic stimulus for each of the blocks, add additional assertions while developing the RTL, and verify all the corner cases by simulating the RTL. Then, replace the original transaction-level model of the AMBA bus by the full RTL model, and run some tests to ensure correct integration.
6. Finally, replace the AMBA SystemC transaction-level model of the bus by its RTL equivalent, hook up all the RTL blocks including the processor (using ARM's DSM model) and run a limited amount of software to come out of reset and process some realistic data to ensure overall chip integrity.

This approach has clear advantages. Although the creation of the higher-level model early in the project appears to take extra time, this is earned back many times over by the confidence that it provides in knowing that the right architecture is implemented, and eliminates iteration because of specification changes. Moreover, hardware and software development can proceed concurrently because both rely on a single, unambiguous golden-architecture model at the transaction level as the starting point. All hardware and software integration issues are taken care of early in the project. This integration is completed before the RTL becomes available, using the transaction-level model, making the final integration a straightforward process.

The benefits provided by this flow enable a divide-and-conquer approach that really works. High-level architectural issues are quickly modeled and analyzed at the transaction level with the abstracted model of the chip. Bus utilization under various traffic conditions is easily explored. Buffer management policies can be devised, packet/cell-loss impact can be analyzed, and overall throughput performance can be analyzed. With this model cycle accurate, designers obtain the accuracy required for making the correct HW/SW tradeoffs early and are confident that these decisions will hold true when the fully-detailed RTL design is available. For example, latency assumptions captured through assertions ensure that refinement can proceed in a straightforward manner knowing that the ensuing simulations will flag any violations of these assumptions.

Conclusion

A two-phase design (and verification) flow starting with the creation of a cycle-accurate, transaction-level model of the target chip ensures creation of the right architecture. By using a step-wise refinement of this model into synthesizable RTL overall design time is shortened, while achieving better results. Specifically, because this flow enables moving the HW/SW integration to the front of the process, concurrent hardware and software development can become a reality.

Language capabilities in SystemC and SystemVerilog—and their simulators—support such a flow today. The matching functionality of SystemC channels and SystemVerilog interfaces enables a flow where architects and designers both can work in their language of choice. Advanced simulation tools such as VCS and System Studio as integral components of the Synopsys Discovery™ Verification Platform offer a seamless methodology to connect transaction level and RTL level. This frees the architects to use C++/SystemC for architectural modeling while easing the RTL designer's efforts for implementing the refinements of these models in a preferred HDL world. Additionally, SystemC-based IP is increasingly available for designers to import into the SystemVerilog-based transaction-level models of their complex SoCs.

An evolution of Verilog and fully backward compatible with Verilog, SystemVerilog allows HDL-RTL designers to move up to the architectural domain in their design process. Through SystemVerilog they gain a richer modeling functionality enabling them to leverage their knowledge of HDL design and move up in abstraction. Moreover, SystemVerilog as the new unifying design and verification language enables new levels of efficiency to speed up the steps to implementation.

Appendix: SOHO Router Details

The example SOHO router is a device typically used in a home and small office environment to share a single DSL line with multiple computers as shown in Figure. 5. It connects through the local loop to the central office where it connects with the Internet. In terms of protocols it runs the IP protocol on top of ATM over ADSL. On the other side it has typically four Ethernet ports to which individual PCs can be connected. Most commercial devices also include a wireless 802.11 a/b interface or a USB host interface although that is not modeled in this example.

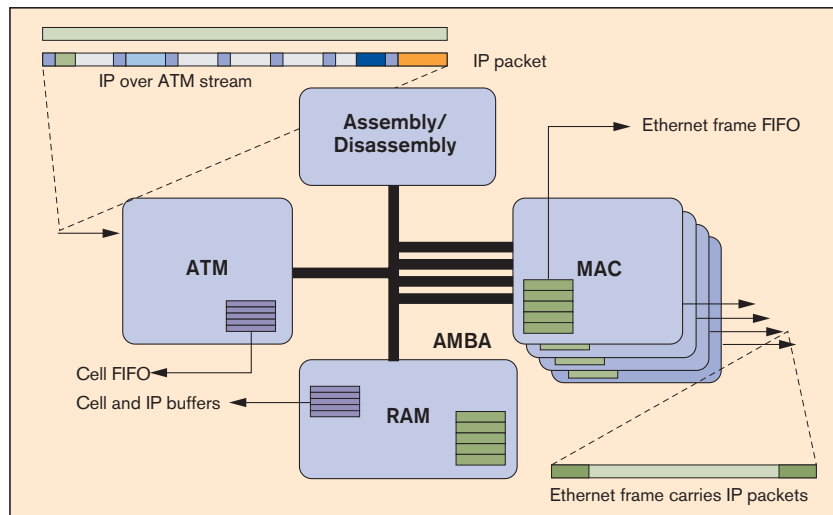


Figure 6: The transaction level model of the communication processor in the SOHO router design.

Functionally, the SOHO router is implemented by a communication processor with the four key blocks depicted in Figure 6 (where the arrows indicate downstream traffic only):

- ATM block reading in the ATM cells and performing the ATM protocol
- Memory block used to store the ATM cell and the IP frames when they are assembled
- Assembly/disassembly module responsible for collecting ATM cells into IP blocks and vice versa
- Ethernet MACs – in this example there are four modules performing the IP over Ethernet implementing a local area network

The first step is to create the individual transaction-level models for the blocks. Each of these blocks is represented by a separate SystemC process. Using the Synopsys DesignWare SystemC AMBA transaction-level models for the AMBA bus, all data transfers between these blocks are modeled in a cycle-accurate fashion using the available API. In the downstream direction, inputs to the ATM block are a stream of ATM cells, making up the IP packets. Moreover, the outputs on the Ethernet blocks are the Ethernet frames encapsulating the IP packets. The AMBA communications infrastructure takes place over 32-bit words. Data transfers from, for example, the ATM control block and the memory, or the assembly/disassembly unit, take place as 32-bit words. Managing these data transfers is handled by the individual blocks (via dedicated DMA channels).

To analyze the architecture we need realistic traffic to stimulate this model. This traffic is most easily provided by commercially-available traffic generators/analyzers. The Synopsys DesignWare Telecom library contains IP and ATM generators and analyzers with the right functionality. On a very high level, we can define various traffic characteristics to simulate real-world data streams with the intent to analyze their effect on the architecture.

With the complete simulation environment in place, analysis can be performed. Specifically, bus utilization and buffer management issues can be fully explored by running thousands of packets through the system and monitoring the utilization of these resources. As a result, we can identify the constraints placed upon the assembly/disassembly process and conclude that a software implementation is feasible given these constraints.

When the processor is identified, an actual ARM ISS processor model (wrapped in SystemC) replaces the SystemC model to perform the assembly/disassembly functionality. The resulting model for the complete communication processor is a cycle-accurate, high-performance model of the chip that can be passed on to the software team to develop or port the rest of the software for this SOHO router.

SYNOPSYS®

700 East Middlefield Road, Mountain View, CA 94043 T 650 584 5000 www.synopsys.com

Synopsys, the Synopsys logo and DesignWare are registered trademarks and systemc and VCS are trademarks of Synopsys, Inc. All other brands, products or service names mentioned herein are trademarks of their respective holders and should be treated as such. All other products or service names mentioned herein are trademarks of their respective holders and should be treated as such. All rights reserved.

Printed in the U.S.A.