

kfegn gu^(^BVHU0Y\$*&^*%jhg
&b\$oygbhf^%*bvesgrf&g&r\$cv
%vb(bv\$vio_&h\$#@KLW\$NM00(U
gh*k^The*searchjfor0thejk#_l=
jkcj8JIFJ N74 N&ij vji6BtVFA85bp
fdy 74N VAIU&BCJ^&^lijT^*)^(
VT39Nk7vMr8wk0wr
va**perfect**vYv49t78y
sNrUI&k86Alp*Tr5Tl
4f448&^Bcqrzf9623
Bg%67T**language**BN
6b&^oAVQ+RfruAt#4
G THE ELECTRONICS INDUSTRY IS RACING TO FIND A
SOLUTION TO SYSTEM-LEVEL DESIGN THAT INTEGRATES
WITH THE CURRENT DEVELOPMENT METHODOLOGY. iv7rvRtu
E4(76nb^%vKupf-ui

MANUFACTURING CAPABILITIES have been the driving force that fueled the development of new and more capable EDA tools since the industry's inception.

We have now reached a technological inflexion point

that will redefine the industry. The use of the latest available IC-manufacturing technology is no longer the primary goal of most designers. To be competitive, products today must provide complex functions. Parasitic phenomena at 130-nm feature size and smaller directly impact a device's ability to function. The average designer is not trained to find and fix these problems; thus, development cycles become longer and costlier. It is more desirable to find a way to avoid the problems instead of hoping to avoid them or fixing them during layout. Doing so requires careful and precise system-level design to ensure that products will work and be on schedule.

Most IC products developed today require at least one respin, due to design errors. The average direct cost of one respin is more than a million dollars, and the cost of missing the market window is in the tens of millions or more. Therefore engineers, when possible, are conservative in choosing a fabrication process and prefer to let someone else explore the hidden traps of a new process technology. They spend more time at the beginning of the project, planning, defining, and partitioning the product. Unfortunately, the tools available for such tasks are, in general, inexact. They consist of descriptions written in languages used for human communication that have informal semantics, spreadsheets that you cannot translate into executable models, and drawings that are approximate and difficult to update or translate into a form that you can use for product development. Some rigorous tools for system-level work exist, but few are integrated with the rest of the development flow. Therefore, engineers must manually re-enter the information into the existing tools. The process is understandably error-prone due to both the magnitude and the complexity of the data to be manually translated.

THE DRIVE TOWARD NEW MODELING LANGUAGES

In the late 1990s, as Verilog became the prevalent hardware-modeling language, designers began to find that the language had too little power to allow them to model system-level constructs. Some EDA vendors, mostly new ventures, tried to find a solution to this problem. Most of the difficulties at the time consisted of deciding on trade-offs between hardware and software implementations and supporting a design methodology that would improve the quality of communication between hardware engineers and software developers. In September 1999, organizers, with the backing of approximately 50 companies, formed OSCI (Open SystemC Initiative). Its initial goals were to promote the exchange of system-level intellectual-property products and hardware/software co-design using a common C++ modeling platform. The primary justification for choosing C++ as the base language was the fact that C and C++ were the dominant languages that chip architects and software engineers used.

Co-Design Automation, now part of Synopsys, viewed Verilog as a starting point for developing a system-level-modeling language because of its popularity with hardware designers. The company an-

At a glance42

VHDL does not need the "System" moniker44

For more information46

nounced a new language, Superlog, at the 1999 Design Automation Conference. As the name implies, Superlog is a superset of Verilog. In June 2002, Accellera announced the release of Version 3.0 of a new language called SystemVerilog. Co-Design had worked with the Accellera committee to extend the capabilities of Verilog. After its acquisition of Co-Design, Synopsys donated more Superlog features to Accellera. The standards-making consortium has now released Version 3.1 of SystemVerilog.

After the restandardization of Verilog in 2001, its working group decided that the new version did not answer the requirements of system-level modeling and that another version of the language was needed. The group works under the auspices of the IEEE, and its PAR (Project Authorization Request) number is 1364.

AT A GLANCE

- ▶ Modeling languages are too weak for electronic-system-level design.
- ▶ SystemC, SystemVerilog, and Verilog 2005 have many common features.
- ▶ The working groups hope to merge SystemVerilog and Verilog 2005 into one language.
- ▶ A working group is also adding features to VHDL.

SystemC, Verilog, and SystemVerilog have received much press coverage in the past 12 months, making the language characteristics and development status deserving of a closer look. In addition,

the IEEE is also working on a new version of VHDL in a project known as VHDL-200X, under PAR number 1076 (see sidebar “VHDL does not need the ‘System’ moniker”).

Unfortunately, the VHDL-200x Working Group seems bent on developing an alternative to SystemVerilog for hardware modeling. The problem with this approach is that VHDL will never be a successful alternative to Verilog or SystemVerilog. VHDL has much richer syntax and semantics and, as such, is better suited to modeling heterogeneous systems, not just electronic hardware. In fact, systems on chips and multichip or multiboard systems will require the concurrent modeling of hardware, software, and mechanical systems. This task best suits VHDL, and the working group should follow this direction. **Table 1**

TABLE 1—LANGUAGE FEATURES

	VHDL 1993	Verilog 1995	SystemC	Verilog 2001	System Verilog 3.1	Verilog 2005	VHDL 200X
Switch-level modeling	X	X		X	X	X	X
ASIC timing	X	X		X	X	X	X
Concurrency	X	X	X	X	X	X	X
Design modularization	X	X	X	X	X	X	X
Gate-level modeling	X	X	X	X	X	X	X
Gate-level timing	X	X	X	X	X	X	X
Four-state logic	X	X	X	X	X	X	X
Event handling	X	X	X	X	X	X	X
Basic data types	X	X	X	X	X	X	X
Basic behavioral constructs	X	X	X	X	X	X	X
Dynamic generation of hardware	X				X	X	X
Configurations	X				X		X
Simple assertions	X				X	X	X
Assertions (formal methods)					X	X	X
Dynamic-memory allocation	X		X		X	X	X
Pointers	X		X		X		X
Multidimensional arrays	X		X	X	X	X	X
Records	X		X		X	X	X
Enumeration	X		X		X		X
Automatic variables	X		X	X	X	X	X
Signed numbers	X		X	X	X	X	X
User-defined logic types	X						X
User-defined resolution functions	X						X
Void type			X		X		
Union	X		X		X		X
Behavioral constructs	X		X		X	X	X
Classes with methods and inheritance			X		X		X
Sequential regular expressions	X	X	X	X	X	X	X
Temporal-property definitions					X	X	X
Scheduling for testbench and assertions					X	X	X
Semaphores	X				X		X
Stimulus generation					X	X	X
Constrained random data generator					X	X	
Coverage monitoring					X	X	X
Strings and strings operations	X				X		X
Standard C interface					X	X	X
Transaction modeling			X			X	X
Module encryption						X	

summarizes the most salient features of the various modeling languages, both existing and proposed.

SYSTEMC

OSCI has identified five requirements that guide the development of SystemC. According to these requirements, Sys-

temC must support a high-level description of system functions to be implemented in hardware and software, permit system architects to defer hardware/software partitioning decisions as late as possible, provide a path from system representation to software implementation, provide a path from system

representation to hardware implementation, and provide mechanisms for managing the complexity of leading-edge systems.

SystemC is C++ with new classes. Its major advantages are its maturity and its public-domain status. Because designers have used C++ in a range of applica-

VHDL DOES NOT NEED THE "SYSTEM" MONIKER

By Stephen Bailey, Mentor Graphics Corp

The EDA industry has witnessed an eruption of new or revised languages to address system-level design. These languages are often identified with the prefix "System." Curiously, the VHDL community has not advertised the existence of work on "SystemVHDL." However, this lack of publicity does not indicate that VHDL is stagnant with no plans for future enhancement. Rather, a VHDL language-revision effort promises to bring advanced, higher level design and verification capabilities to VHDL. Because VHDL has been the historic leader in advanced HDL capabilities, it has found extensive use in design contexts more abstract than RTL (register-transfer level). Therefore, calling a revision of VHDL "SystemVHDL" would imply that VHDL shared the same significant shortcomings of other HDLs. Because this work is a logical continuation of VHDL, the IEEE is identifying the revision effort by the fact it is the first significant revision of the new millennium, calling it VHDL 200X.

The plans for VHDL 200X include enhancements that facilitate a higher level of modeling abstraction and also include facilities in designer productivity, capabilities for higher levels of abstraction, reduced verbosity, and improved capture of design intent. The plans also include features that enable modeling that is currently difficult or impossible, verification productivity, assertion-based verification, stimulus generation, and high-level modeling (reference

models and transaction-level modeling).

To facilitate design productivity, VHDL provides a robust type system featuring user-defined, enumerated, signed, unsigned, bit, bit_vector, integer and constrained-integer, real, access (pointer), and record types. Full object-oriented-language facilities are missing, although VHDL defines protected types that encapsulate data values (types) with user-defined operations on those data values. The 1076 Committee is working to identify how it can add inheritance and enhanced generics to allow type and subprogram parameterization of models. You can expect that the object-oriented enhancements will also facilitate interface modeling at multiple levels of abstraction. Adding these object-oriented features will facilitate higher levels of design reuse as well as higher levels of abstraction in modeling.

The working group is also looking at ways to reduce the amount of typing required without weakening VHDL's static-verification capabilities. VHDL's strong typing and deterministic-simulation semantics prevent or quickly identify problems such as race conditions, invalid array indexes, and illegal-value ranges. These characteristics save verification time, and, because verification can take up 70% of design time, they help to significantly diminish the verification and debugging effort.

A number of enhancements both reduce the typing burden for the designer and better cap-

ture the designer's intent.

Accurately capturing a process sensitivity based on the combinatorial or sequential nature of the process is a prime example. Eliminating the need for a separate architecture is another. Most of these changes have relatively small impact on the language but a significant impact on improving the designer's experience with VHDL.

In a few rare cases, VHDL struggles to provide the capabilities needed to model certain hardware functions. The prototypical example in this area is the ability to model a jumper or a bidirectional pass-through switch. The working group is addressing these issues.

To enhance the verification productivity of the language, the working group plans to adopt Accellera's PSL (Property Specification Language) as VHDL's property-specification capability. You can use PSL for a multitude of purposes, including:

- defining properties that must hold during verification (assertions);
- defining functions that need to be tested during verification (functional coverage); and
- specifying input constraints, legal sequences, and combinations of legal sequences (stimulus generation).

Object-oriented enhancements have double benefits, because they improve both verification and design productivity. Object-oriented methods simplify the tasks of:

- creating testbenches (verification infrastructure) that you

can reuse at different levels of abstraction and from one design to another;

- creating high-level models used as reference models during verification; and

- modeling common verification data structures, such as transactions and scoreboards.

Planned enhancements to facilitate verification productivity include:

- dynamic process creation/destruction;
- an ability to use properties to specify association of weights to guide stimulus generation;
- access to functional coverage data in the testbench to allow the creation of reactive testbenches;
- standardized signal-access capabilities that allow a testbench to probe and force signals anywhere in the design;
- libraries of predefined, commonly used checkers, and data types, such as FIFOs and associative arrays; and
- more abstract interprocess-communication mechanisms.

Although the next revision of the language is not called SystemVHDL, it will deliver higher level modeling and abstraction capabilities as well as property-specification and verification-automation features that will improve design quality and verification productivity.

AUTHOR'S BIOGRAPHY

Stephen Bailey, a technical marketing engineer at Mentor Graphics, is chairman of the IEEE 1076 Working Group.

tions, including real-time-operating-systems development, it is a robust language. Its public-domain status makes it conveniently inexpensive, but EDA vendors can still generate profits by offering proprietary tools to a large installed base of potential customers. Because C++ is a software-development tool, its semantics do not support modeling-hardware behavior. OSCI solved this problem by using a facility in C++ that allows a developer to define extensions to the language by building a class. In defining a class, an engineer can specify both its syntax and its runtime semantics.

You define new semantics by overloading existing semantics definitions. Overloading means to redefine the rules within a local region of the language—in this case, the SystemC class library. When engineers use the variables and primitives that the library defines, the new rules take effect; when you use variables and primitives defined elsewhere, the original C++ rules take effect. When you define new runtime semantics, you cause the compiler to generate new code that a program that understands the binary strings produced by the C++ compiler can execute. You can then develop a SystemC simulator by extending the runtime behavior of C++ with the additional information provided by the class definition. OSCI needed to add three fundamental concepts to C++: the notion of time, the notion of concurrency, and hardware data types.

A software-programming language needs not understand time, because the hardware that executes the program provides execution timing. But, if you need to model hardware, you need to be able to model time delays and sequence of execution. You need to be able to tell the simulator which hardware model needs to be executed next and how much time it takes for a signal to propagate from one hardware primitive to another. Although software-programming languages support the notion of parallel execution, they do not understand concurrency. Parallel execution means that loosely coupled hardware systems independently execute two or more streams of instructions. The use of semaphore primitives generally synchronize those executions. The primitives operate as global state machines that oversee the proper execution of the intended algorithms.

Hardware concurrency means that, once you turn on the power, all of the hardware works at the same time. Therefore, a simulator must be able to mimic the concurrency while executing on a computer that processes a sequence of instructions that is naturally not concurrent. The C++ native data types are inadequate for modeling hardware. For example, no data type exists that you could use to describe a tristate logic value. SystemC adds a four-value logic system to C++ that allows the modeling of digital logic. It does not yet support interaction between digital and analog logic and

architectural inspection of analog modules. The most current version of the language is SystemC Version 2.0.1.

SYSTEMVERILOG

SystemVerilog is the result of work to extend the modeling capabilities of Verilog 1995 to match and, in a few cases, surpass what VHDL-1993 offers. The fundamental problem Accellera, the industry consortium that is sponsoring the development, faces is to avoid the major drawbacks of VHDL: verbosity and complexity. It is practically impossible to add features to a language without making its syntax—and, more important, its semantics—more complex. The advantage that this project has over VHDL is market demand. VHDL was ahead of its time, and electronics designers used few of the features that it offered. Therefore, the language seemed complicated, and the simulator executed more slowly than Verilog, because it had much more to do. Unfortunately for EDA vendors, the insistence by the US Air Force that their products support all of the language to qualify for Department of Defense contracts forced vendors' hands. It remains to be seen whether designers will be willing to cope with a complex language and slower execution to reap the benefits of greater behavioral modeling and verification capabilities.

The goal of SystemVerilog is to improve productivity in the design of large-gate-count, VC (virtual-cores)-based, bus-intensive chips. It primarily targets the chip implementation and verification flow. Version 3.1 implements four major extensions of Verilog 2001. To support transaction modeling, the language provides a DPI (direct-programming interface) that enables engineers to include C/C++/SystemC functions in the design model. Such a facility allows cosimulation between SystemVerilog and SystemC blocks.

A group of extensions aims to improve the ease of modeling. The ability to model transactions at the interface level eases the development of bus-intensive designs; allowing any data type on each side of a port eases the modeling task; extended data types support the use of C-like languages; and nesting of modules locally to their parent module enhances VC protection. Of course, relaxing the rules for port connections opens the possibility of design errors. Trade-offs between ease of use and correct by con-

FOR MORE INFORMATION...

For more information on products such as those discussed in this article, contact any of the following manufacturers directly, and please let them know you read about their products in *EDN*.

Accellera

www.accellera.org

Aldec

1-800-487-8743

www.aldec.com

@HDL

1-408-441-1317

www.athdl.com

Axis Systems

1-408-588-2000

www.axiscorp.com

Bluespec

1-781-250-2200

www.bluespec.com

Cadence Design Systems

1-800-746-6223

www.cadence.com

CoWare

1-888-269-2738

www.coware.com

EVE

1-888-738-3872

www.eve-team.com

Fintronic

1-650-349-0108

www.fintronic.com

Forte Design Systems

1-800-585-4120

www.forteds.com

Jeda Technologies

1-650-964-5332

www.jedatechnologies.com

Mentor Graphics

1-800-547-3000

www.mentor.com

Novas

1-408-467-7888

www.novas.com

OSCI

www.systemc.org

Real Intent

1-408-982-5444

www.realintent.com

SynaptiCAD

1-540-953-3390

www.syncad.com

Synopsys

1-800-541-7737

www.synopsys.com

TNI-Valiosys

1-408-203-2500

www.tni-valiosys.com

Verisity

1-650-934-6800

www.verisity.com

0-In Design

Automation

1-408-487-3640

www.0-in.com



struction should probably never favor ease of use. The designer has to model correctly only once, but the verification engineer might have to spend many hours, or even days, identifying the problem when two connecting ports of different types do not work.

A new mechanism supports assertion-based verification, enabling a design-for-verification methodology. Synopsys has made “design for verification” a marketing banner in its campaign to capture SystemVerilog seats. Embedded assertions, which you can use to check for problems caused by relaxing the port semantics, capture design intent in terms of functions and constraints; you verify them during simulation. IBM’s (www.ibm.com) formal assertion language, PSL (Property Specification Language), provides the assertion technology that SystemVerilog uses. Support for object-oriented techniques improve the development and reuse of testbenches and hardware models. Accellera has agreed in principle to make the next version of SystemVerilog available to the Verilog 2005 Working Group. The new version is scheduled to become available by June 2004. Accellera made the agreement in the hope of avoiding the existence of two supersets of Verilog 2001, which would create confusion and incompatibility in the market. However, because the technology donation comes later than the Verilog working group had expected, it is bound to create more work for the developers of Verilog 2005.

VERILOG 2005

The IEEE 1364 Working Group, which developed both Verilog 1995 and Verilog 2001, is responsible for developing the IEEE Verilog standard. In March 2003, the group issued a call for donations to update and expand the standard to meet new electronic-system-design challenges. As of September, it had received nine donations from four EDA vendors covering nine subject areas. The goal of the committee is to produce a new IEEE standard in 2005. Unfortunately for the working group, no one has yet donated any of the SystemVerilog work, although Accellera has promised to do so by June 2004.

Cadence Design Systems has donated technology in six areas, and Fintronic, Jeda Technologies, and Verisity have each donated technology that address one area. Cadence’s donations target fundamental extensions to the syntax and se-

mantics of the language. The company has donated new data-type definitions useful for extending Verilog to the system level and rules for guiding how to consistently define both data types and their functions in the language. To this end, it has proposed a set of basic and composite data types, a general methodology for user defined types, and dynamic memory allocation for any type. A donation that proposes a methodology for encrypting portions of the source code of Verilog models aims to offer some proprietary protection for third-party providers of soft cores. A donation that extends the VPI (Verilog Programming Interface) supports extended data types and assertions using PSL.

Two donations target improving the ease of debugging models written in Verilog: One provides the ability to do constrained randomization of test patterns; the other represents a VPI for transaction recording that enables engineers to view and analyze transactions at a higher level of abstraction. Cadence has also donated a set of guiding principles to help produce a coherent language in the short term and ease the path toward future extensions. It aims to provide a structure to the working group so that it can operate more systematically than in the past.

Fintronic has donated technology that supports separate compilation of Verilog models. The feature provides the ability to separately compile parts of a Verilog design. You can then mix the precompiled parts with other Verilog source and binary files to build and simulate a design. This feature can be an alternative but not a direct replacement for the encryption donation from Cadence. The donation from Jeda Technologies proposes to extend the capabilities of testbench development and design debugging by associating another language, Jeda-X, with the Verilog language. The proposed language would provide object-oriented-programming support for writing modular and reusable testbenches. It would also support aspect-oriented programming, concurrent programming, and cycle-based testbenches. Finally, it would provide enhanced list and array data types for behavioral modeling and synchronization primitives for multithreaded execution.

Verisity has donated its own technology to support testbenches and debugging as direct alternatives to the Jeda-X proposal. The company has proposed the

addition of two keywords in Verilog, `keep` and `gen`, which would allow designers to specify both hard and soft rules for keeping all values within specification and call for the generation of new stimuli precisely when required. Another donation from Verisity extends the language to enable designers to collaborate with verification engineers to jointly describe functional coverage points and identify critical combinations of inputs and internal events that must be tested. It has also donated syntax for the test writer, enabling the introduction of `keep` statements that impose restrictions on generation of data values within a test module or modules. The `extend` keyword enables the test writer to describe the basic rules for legal sequences of input in one place and later extend these rules for a test to focus on a single target.

The working group will face a difficult integration problem when it receives the SystemVerilog donation, because the group has already designed many SystemVerilog features into Verilog 2005. In addition, the two languages are similar but not identical. It will take a great effort, both in engineering and management, to reconcile the differences so that the result will be one language and not two dialects with identical linguistic parentage.

COMMERCIAL SUPPORT

Standard development of the four languages is following different paths. Developers of both SystemC and SystemVerilog designed them as commercial efforts, and the move toward standardization is both an engineering effort and a marketing effort toward language acceptance. Verilog 2005 and VHDL 200x are following the traditional strategy of the Computer Society, one of the professional societies that make up the IEEE, of first developing a standard through an engineering effort and then marketing the standard. As a result, tools to support either of the languages the IEEE developed do not yet exist. A few EDA vendors are making an exception to this approach by providing support for PSL in their VHDL implementations.

Historically, this strategy has proved commercially less successful than the alternative. In the case of VHDL and Verilog,

for example, VHDL suffered commercially by the fact that no EDA tools that used even a portion of the language were available before standardization, whereas Verilog enjoyed active support even during its development stage. Yet EDA vendors are willing to invest in the IEEE development process because it provides valuable engineering insights into how

IT WILL TAKE A GREAT EFFORT, BOTH IN ENGINEERING AND MANAGEMENT, SO THAT THE RESULT WILL BE ONE LANGUAGE AND NOT TWO DIALECTS WITH IDENTICAL LINGUISTIC PARENTAGE.

you should construct a language, and it brings to light customer preferences that would otherwise be difficult to uncover.

Synopsys was the original driver behind the development of SystemC. The initial language development was proprietary, but Synopsys quickly followed the strategy that Cadence developed with respect to Verilog. Synopsys formed an industry consortium, Open SystemC, that was similar to OVI (Open Verilog International), the consortium responsible for the success of Verilog. Open SystemC modified itself into OSCI when it became clear that a consortium under the auspices of one EDA vendor was not conducive to rapid language acceptance. In the process, Synopsys gave up much of its control over the policy and directions of the organization, in favor of wider participation by its two major competitors, Cadence and Mentor. SystemC has not yet enjoyed the popularity of Verilog, in part because the market is smaller and in part because the tools available have not successfully bridged the gap between system design and hardware design.

A comprehensive list of SystemC tools is available on the OSCI Web site, but a few companies deserve special attention, because they have developed products that aim to integrate the use of SystemC with the RTL (register-transfer-level) methodology. Cadence,

EVE (Emulation and Verification Engineering), Mentor, Synopsys, and TNivaliosys offer mixed-language simulators that allow engineers to use SystemC models with VHDL and Verilog models in the same design. CoWare entered the SystemC market early. A number of system companies adopted its ConvergenSC product because it uses a common infrastructure for both design and verification. The product was the first available alternative to the canonical simulator, which OSCI provided free. The reference simulator, as you would expect, offered too few capabilities to be a useful tool for commercial applications. Forte Design Systems provides a free SystemC tutorial on the Web that serves as a useful introduction to the language. The company's Cynthesizer synthesis tool allows designers to bypass the translation from SystemC to either Verilog or VHDL to generate hardware from a SystemC model (**Reference 1**).

A number of EDA vendors are developing support tools for SystemVerilog. Most of the announced work is likely to result in commercial tools in time for the Design Automation Conference in June. Real Intent, Synopsys, and 0-In have announced plans for formal verification products; Novas, SynapticAD, and Synopsys will support testbenches and debugging tools; Aldec, @HDL, and Synopsys will offer simulators; and Axis Systems will support the language with emulation products. Bluespec, Cadence, and Mentor have not specified the nature of their product offerings, although you can expect that both Cadence and Mentor will provide at least simulation support for the language.

Although no commercial support is available for Verilog 2005, Fintronic, Jeda Technologies, and Verisity all offer products that implement the technologies they have donated to the working group. Fintronic uses its separate compilation for Verilog in its FinSim product, Jeda Technologies markets Jeda-X as a testbench-development tool, and Verisity developed the technology for its verification products based on the `e` language. Cadence has used some of the technology it has donated as well, throughout its verification-product line. □

REFERENCE

1. Bhasker, J, *A SystemC Primer*, www.stargalaxy.com.

