Panel

SystemC and SystemVerilog: Where do they fit? Where are they going?

Organiser: Donatella Sciuto
Departimento di Elettronica e Informazione
Politecnico di Milano
Milano, Italy
+39-02-2399-3662
sciuto@elet.polimi.it

Organiser and Moderator: Grant Martin
Cadence Berkeley Labs
1995 University Avenue, Suite 460
Berkeley, CA 94704 U.S.A.
+1-510-647-2804
gmartin@cadence.com

Panelists

Wolfgang Rosenstiel
University of Tübingen
Germany

Stuart Swan
Cadence Design Systems
U.S.A.

Frank Ghenassia
ST Microelectronics
France

Peter Flake
Synopsys
U.S.A.

Johny Srouji
Intel
Israel

Abstract

There is tremendous interest in design languages these days - and more particularly, SystemC and SystemVerilog. Sometimes the truth about design languages can be obscured by marketing and the press. This panel is meant to deepen the technical understanding of the DATE audience on the issue of design languages. It contains five technical experts - an academic expert in design languages and SystemC and SystemVerilog in particular; a language expert for each of SystemC and SystemVerilog; and a user expert for these two languages. The language experts have been heavily involved in the specification and evolution of their respective languages. The user experts have been heavily involved in developing use methodologies for these languages within their own design communities, and in applying them to real design problems. The panelists will consider the questions:

- *what are the key capabilities of these languages and what do they offer to users?*
- *which design problems are they best used for? what is their scope?*
- *how has application of these languages to real design problems improved the productivity of designers and the quality of the design results?*
- *where should the languages develop further capabilities?*

1. Introduction: Grant Martin, Cadence Berkeley Labs

The world of design languages has certainly evolved at a rapid rate in the last year or two. The ominous clouds pointing to a language war have indeed been blown away by harmonious winds of co-operation and peaceful coexistence. Accellera and IEEE 1364 have announced plans for working together in evolving the future of Verilog. Progress is being made in standardising SystemC via the IEEE process.

There is a growing realisation among researchers, design users and EDA developers that design languages may both have their own unique application niches and some overlap in concepts and constructs; and that indeed, such overlap may be beneficial. This is because conceptual and semantic overlap may allow the construction of more integrated design flows, in which models created in one notation and design environment can be re-used in another, either directly in some cases, through encapsulation in others, or via synthesis and translation processes. Semantics which are common, or at least translatable, between languages, may actually contribute to reusable models and better design flows.

What is most important about the new and evolved design languages is to ensure a common basis of understanding of the

1530-1591/04 $20.00 (c) 2004 IEEE

languages, their features and capabilities. The panel position statements offered here all help to contribute to this goal:

- Wolfgang Rosenstiel surveys key capabilities of SystemC and SystemVerilog and answers the key questions from the abstract, also providing a number of useful references.
- Stuart Swan brings a perspective on interoperability between SystemC and SystemVerilog.
- Frank Ghenassia discusses in particular the importance of transaction-level modelling and the key language requirements to support this.
- Peter Flake surveys SystemVerilog in particular in answering the key panel questions.
- Johny Srouji discusses from a user perspective the most important contributions of SystemVerilog.

We trust that the panel presentations and responses to user questions will contribute to the design language education of the user community in general.

2. Wolfgang Rosenstiel, University of Tübingen

There is tremendous interest in design languages these days, especially in SystemC and SystemVerilog. This position statement is my personal opinion based on some experiences of our own research group at the University of Tübingen and the Computer Science Research Centre FZI. I hope this view will deepen the technical understanding of the DATE audience on the issue of design languages in general and SystemC and SystemVerilog in detail. For further readings I recommend the enclosed list of references.

In the following I want to comment on the questions for the panelists with respect to SystemC and SystemVerilog.

2.1 What are the key capabilities of these languages and what do they offer to users?

SystemC is not a new language. It is basically a class library built with standard C++ together with an event-driven simulation kernel. The class library enables the modelling of hardware/software systems by particularly including means for describing concurrent behaviour, a notion of time, and special hardware data types. Above this SystemC core language, further design libraries have been developed for special use by individual users. The event-driven simulator works with events and processes. The SystemC methodology relies on modules and ports for representing structure. Interfaces together with primitive or hierarchical channels are used to describe communication.

It took SystemC less than two years to emerge as a widely used language. In my opinion, this is due to the fact that SystemC adopted object-oriented system-level design - the most promising method already applied by the majority of firms during the last couple of years. In addition, SystemC added numerous tools already in use at many EDA firms. Even before the introduction of SystemC, many system designers had attempted to develop executable specifications in C++.

SystemVerilog is a new language. It is an extension of the well known and widespread hardware description language Verilog in order to support higher levels of abstraction for modelling and verification. The current version 3.1 of SystemVerilog is an extension of SystemVerilog 3.0. SystemVerilog 3.0 adds several new constructs to Verilog-2001 to particularly improve productivity and readability as well as modular design. I especially want to stress interfaces to encapsulate communication in more communication-centric modular designs.

SystemVerilog 3.1 added special verification support including test bench capabilities. Also interesting are new object-oriented constructs including classes. Built-in synchronisation primitives such as semaphores and mailboxes and mechanisms for dynamic process creation, process control, and inter-process communication support the construction of higher level models. It also includes dynamic memory management in a re-entrant environment to support automatic garbage collection. Last, but not least, assertion mechanisms for verification and functional coverage have been added.

2.2 Which design problems are they best used for?

Due to its C++ compatibility, SystemC supports software compatibility and is an ideal candidate to improve the design process at the software/hardware interface. In a hierarchical top-down design flow the user can start with executable functional specifications without any timing information, to be further refined to the well-known transaction level, in order to model the communication of system-level processes. Transactions are non-atomic communications, normally with bidirectional data transfer, and consist of a set of messages that are usually modelled as atomic communications. The messages have unidirectional data transfer, but often bidirectional control flow. Finally these transactions are implemented at a cycle-true and bit-accurate signal level.

SystemVerilog has strong roots in designing and describing hardware and provides full compatibility with Verilog. SystemVerilog therefore still contains all the features necessary for a complete path to implementation including synthesis and simulation with back-annotation. The SystemVerilog methodology can be described as a bottom up oriented approach, providing new means of abstracting hardware descriptions up to the transaction level and providing additional test bench and verification capabilities like assertions, constraints, randomization etc.

On the other hand, the new interface constructs and object-oriented features of SystemVerilog such as classes, together with communication and synchronisation primitives like mailboxes and semaphores as well as dynamic processes, several new data types, direct programming interface to C, passing function call arguments by reference etc. also supports a top down design flow from a transaction level oriented and communication-centric description to the register transfer level and further down to a gate level implementation.

2.3 What is their Scope?

Both languages stress the importance of verification support for complex SOCs including improvements for hardware verification as well as for the verification of hardware-dependent software. In today's design flows the software development can often only start after the hardware is available. This causes unacceptable delays for the software development. The idea of transaction level modelling (TLM) is to provide in an early phase of the hardware development transaction level models of the hardware, especially of the microprocessor and DSP cores of the SOCs to be developed. Based on these TLMs a fast enough simulation environment is the basis for the software development. The presumption is to run these transaction level models at several tens or some hundreds of thousand transactions per second, which is fast enough for software test and debugging.

Most of the recent SystemC activities are oriented towards transaction level modelling. The goal of this work can be described as follows. In large systems-on-chips the software is getting more and more complex and is in the overall development process often on the critical path. Therefore many activities try to shorten the software development cycle or at least to start the software development as early as possible. First standardisation proposals for SystemC TLM libraries as well as the first SystemC TLMs of corresponding cores are available. TLM in SystemC is quite mature and widely used. Several examples are described in [1].

In many references including [4], the scope of SystemVerilog is defined as hardware design and verification including simulation. Many SystemVerilog 3.1 extensions deal with test bench support. Several language constructs have been added to describe in SystemVerilog not only the design under test but as well the test bench including the corresponding test bench features like assertion mechanisms and coverage constructs. Also the interface constructs of SystemVerilog 3.0 as well as the object-oriented constructs of SystemVerilog 3.1 support integrated design and verification in one language, i.e. in SystemVerilog. In particular, the interface refinement possibilities allow the re-use of test benches on different levels of detail. In addition, hardware design, simulation, and synthesis are of course also in the scope of SystemVerilog.

In addition SystemVerilog has the very same potential as SystemC to support transaction level design and transaction level abstraction and refinement. TLMs in SystemVerilog - if available - could serve the same purposes with respect of improving the software development, test, and debugging cycle of hardware-dependent software.

On the other hand, recently the SystemC verification (SCV) library has been added to SystemC in order to add test bench functionality to SystemC.

2.4 How has application of these languages to real design problems improved the productivity of designers and the quality of design results?

As far as SystemC is concerned there are plenty of success stories. More examples can be found in [1] and [5]. In addition, the European SystemC Users Group home page [7] lists special reports including several real life industrial experiences which have been presented during the eight previous meetings. All these pages contain also further links to various SystemC products, tools, courses etc.

Success stories cover the full range of applications like modelling on different levels, verification, and design. Most of the users work with the powerful, robust, and quite efficient reference implementation.

SystemVerilog is a new language without a reference implementation. Currently there is a very detailed, consistent, and complete language reference manual available [4]. It contains many examples as well as a full BNF definition. The SystemVerilog Home Page [6] lists in addition several references to further documentation as well as products, tools, tutorials etc. Based on these descriptions we have had no problems in writing SystemVerilog design and test bench examples on many different abstraction levels from timed and untimed transaction level down to the RT and gate level.

2.5 Where should the languages develop further capabilities?

From my point of view, SystemC 2.0 as well as SystemVerilog 3.1 are mature enough and ready to use. Further developments should therefore especially concern two things, i.e.

- Corresponding products and tools including the necessary vendor support for production use, and
- Corresponding IP-libraries and test bench libraries on different levels of abstraction for the broad range of applications to support cost effective, efficient, and productive design and verification.

2.6 References

[1] W. Müller, W. Rosenstiel, J. Ruf (Eds.), *SystemC - Methodologies and Applications*, Kluwer Academic Publishers, 2003.

[2] T. Grötker, S. Liao, G. Martin, S. Swan, *System Design with SystemC*, Kluwer Academic Publishers, 2002.

[3] SystemC 2.0.1 Language Reference Manual, Revision 1.0, http://www.systemc.org/projects/systemc/document/SystemC_v201_LRM/

[4] SystemVerilog 3.1 - Accellera's Extensions to Verilog®, http://www.eda.org/sv/SystemVerilog_3.1_final.pdf

[5] OSCI SystemC Home Page, http://www.systemc.org

[6] SystemVerilog Home Page, http://www.systemverilog.org

[7] European SystemC Users Group Home Page, http://www-ti.informatik.uni-tuebingen.de/systemc.

3. Stuart Swan, Cadence Design Systems

Is there a language war brewing between SystemC and SystemVerilog? Probably not. While there is certainly some overlap between the two languages, the similarities of the two languages are outweighed by their significant differences.

SystemC excels at system design and verification. It cleanly supports transaction level modelling and verification, HW/SW co-design, and SOC architectural analysis and optimisation. Because it is entirely based on C/C++, SystemC provides an ideal environment for integrating verification and design components that are written in C/C++, including embedded software components. Its C/C++ basis also allows SystemC to leverage the vast amount of tools, books, and expertise that exist for C/C++.

SystemVerilog excels at hardware design and verification from the register transfer level to the gate level. SystemVerilog retains Verilog's conciseness and ease of use for these modelling levels, and it adds a wide array of features to support verification. Because it is based on Verilog, SystemVerilog leverages all of the tools, knowledge, and IP that already exist for Verilog.

There is certainly some overlap between the two languages, but this overlap is desirable since it enables unified design flows between SystemC and SystemVerilog to be constructed. In the cases where designers could use either SystemC or SystemVerilog for a particular design task, the choice will probably not be determined solely by the technical features of the languages. Other factors that must be considered include tool and IP availability and cost, designers' existing knowledge of the languages and willingness to learn new languages, organisational considerations, and the amount of legacy design and IP in a particular language.

It is likely that in the future both SystemC and SystemVerilog will be widely adopted and that they will both evolve further. It is also likely that unified design flows and tools that support both languages will be widely used.

4. Frank Ghenassia, STMicroelectronics

A Transaction-Level Model (TLM) denotes an IP and SoC VLSI hardware abstraction level. It is defined as follows:

An SoC is composed of a set of communicating hardware components.

A component (IP) may be either programmable (i.e. a processor) or hardwired (i.e. fixed behaviour). Each component is composed of a finite set of possible states and a set of concurrent threads of execution. Each thread is communicating with other threads of its component and also with other (threads of) components. Communication can be:

- Exchange of data
- Synchronisation to inform or be informed of some change of the system state. A main usage of synchronisation is to ensure data consistency, preventing threads from reading data content with unknown state (valid or invalid) or writing data to (possibly temporarily) inaccessible memory areas.

Based on the above definition, untimed TLM models can be developed to enable:

- Early (functional) embedded software development
- Development of functional verifications tests (and associated output data) for the RTL
- Support the definition of the SoC (functional) architecture

Untimed TLM models can be annotated with timing delays to enable:

- Early embedded software optimisation for real-time constraints
- Development of performance verification tests for the RTL
- Support the definition of the SoC (timing) micro-architecture

The timing annotations are kept separate from the TLM untimed model. This modelling approach (separation of the untimed TLM model from the timing module) has the following benefits:

- Functional correctness of the TLM specification cannot rely on some timing behaviour of the implementation
- Complete separation of behaviour and timing issues
- Unique description of functional behaviour (as opposed for example to a solution where an untimed TLM is refined to add timing)
- Ability to dynamically enable and disable timing modules during the simulation

At STMicroelectronics, the TLM modelling approach was initiated in early 2000. At that time, we needed an open standard language with appropriate modelling support. SystemC2.0 was the only solution. In 2004, it seems that a new language, SystemVerilog3.1 is emerging as a language with potential appropriate support for TLM modelling. More important than the choice of the language, the usage of the appropriate TLM abstraction level is *the* concept we need to promote. It is the key enabler for early embedded software development, usage of a golden reference model for functional verification and also (micro-) architecture specification.

5. Peter Flake, Synopsys

5.1 Introduction

SystemVerilog is the evolution of Verilog, the widely used hardware description language. Verilog began by providing high performance, high accuracy gate level models coupled to an intuitive behavioural modelling language. It has now become a popular design language for RTL synthesis.

5.2 Key Capabilities

SystemVerilog offers both hardware design and verification capabilities in a single language. It includes many of the advanced design features proven in VHDL, but also provides further extensions in terms of data types, encapsulation mechanisms and assertions. In particular, the interface construct encapsulates communication both at the netlist level, as a wire bundle, and at the transaction level, as method calls. An assertion can apply not only to a Boolean expression, as in VHDL, but also to a sequential property. The property can be a regular expression, which represents a checker automaton very concisely.

SystemVerilog also has the features of hardware verification, or testbench, languages: constrained random pattern generation, classes, and dynamic processes. The syntax for specifying constraints allows a solver to relieve the user of the burden of writing a constrained random generation algorithm.

Scheduling extensions provide a clear mapping between event-driven and cycle-based semantics, ensuring consistent results across simulation, synthesis and formal verification tools. Formal property checkers can therefore easily be incorporated into a simulation-based verification methodology.

SystemVerilog has a high performance Direct Programming Interface to C, as well as the Verilog Programming Interface for traversing the design and interacting with simulation. These support the integration of third party tools and custom packages into the design and verification flow.

5.3 Scope

SystemVerilog provides the features required for the design of complete complex chips, such as leading-edge microprocessors, from the transaction-level model to the gate level netlist. Having a single language allows the testbench to be re-used at various modelling levels and the simulation results compared.

It also supports system-on-chip design using IP from multiple vendors, by allowing verification features, such as assertions to monitor bus protocols, to be included with the design. These can be conveniently encapsulated in interfaces to simplify system integration.

The C interface simplifies hardware/software co-verification for platform-based designs, whether via an instruction set simulator or via a directly driven bus functional model.

5.4 Productivity

The application of SystemVerilog to real design problems has demonstrated a substantial productivity gain due to the reduced amount of code needed to get the same quality of results from synthesis. Less code means not only less time for entry and updating, but also fewer bugs. In addition to a reduced amount of design code, less verification code is needed for the same functionality, which means that more thorough verification is possible at an early stage of the design. Furthermore, the ability to use the same language for design and testbench eases use and debug, and hence improves productivity.

5.5 Future Capabilities

At present, SystemVerilog is targeted at the digital domain. The analogue/mixed signal area is growing in importance, and there is an existing Accellera Verilog-AMS standard. This is therefore a possible future direction for SystemVerilog evolution.

6. Johny Srouji, Intel

6.1 Introduction

With the increasing complexity of semiconductor design and rising validation cost reaching levels of 60% of total efforts, the trend in the design community is to implement designs at a higher level of abstraction. The promise of a higher level design is increasing design productivity as well as reducing validation efforts. This is achieved through mechanisms of design capture, exploration, and verification at a level of abstraction that is close to human reasoning. One can assume that number of bugs found in the RTL correlate to the number of lines of code. Ironically, higher level design abstraction challenges the design process itself when the current RTL design languages and EDA tools and methodologies are used.

Languages such as SystemC and SystemVerilog are rich in behavioural and structural constructs which enable modelling designs at different levels of abstraction while not imposing a top-down, bottom-up or even middle-out design flow. In fact, most design flows are expected to be iterative, and it is rare that all modules within a system are modelled at the same level of abstraction. Moreover, many of the new design projects adopting higher level design are proliferation projects, where new logic at various levels of abstraction are added to existing hardware implementations which are at a lower level.

Consequently, with the introduction and acceptance of higher level design entry points, we witness several scenarios in the design flow where different modelling levels need to co-exist. For example, with a detailed implementation-level model as a starting point, a designer might create a more abstract model in order to increase simulation speed. Furthermore, bottom-up abstractions can be useful for the formal verification of a detailed low level Design-Under-Test (DUT). In order to specify interesting formal properties that the DUT needs to hold, the verification engineer must have a good understanding of the functionality of the DUT.

In the next two sections, I will describe the main capabilities of SystemVerilog as a design entry modelling language for HLD, as well as for verification and lower levels of abstractions.

6.2 Key Capabilities

SystemVerilog introduces a unified Hardware Design and Verification language with different levels of structural and behavioural constructs. The language was designed to provide good solutions starting from the simplest Verilog inconveniences to enabling very sophisticated design and validation methodologies.

One of the simple issues in Verilog which is addressed by SystemVerilog is variables and data types. Not only does it adopt a similar variable and data type system as in C, but it also defines additional types that are useful for system design and validation. For example, SystemVerilog introduced *char* and *int* as well as *typedef* constructs that are similar to C. Furthermore, it introduces new data types such as *bit*, *byte*, *logic*, and more. Logic data types can be used almost exclusively throughout the design and remove much of the wire/reg usage issues. In SystemVerilog variables can be used where only wires were previously allowed: to connect ports of module instances and on the left-hand side of continuous assignments. Another issue that SystemVerilog addressed is array or variable copying. It is now possible to copy all the elements of one array to another with a simple assignment instead of looping through the dimensions and copying each word. Other sensible enhancements include the addition of always_comb, always_latch, always_ff and single-driver semantics to more explicitly capture intent and ensure correctness.

Furthermore System Verilog added several useful constructs into the language which are similar to C. Examples are enumerated data types, structures, unions, control statements such as break, jump and continue and enhancements to assignment operators.

System Verilog introduced several major constructs in the language. One good example is the interface construct, which is viewed as one of the major advantages of the language. An interface construct encapsulates the communication between two blocks, allowing a smooth migration from abstract high level design down to lower RTL and structural views of the design. Interfaces encapsulate communication similar to the way a struct construct encapsulates data, and they facilitate design re-use, through encapsulation. Additional power of the interface comes from its ability to encapsulate functionality as well as connectivity, making an interface at its highest level, as a class template.

Other than major constructs to the design language, SystemVerilog introduces verification constructs which are extremely efficient for test bench and formal properties development. SystemVerilog adds assertions, process synchronisation mechanisms, dynamic process control, object oriented classes, associative arrays, direct programming interface, and more.

6.3 Productivity

Along with addressing many simple but productive RTL design features, SystemVerilog provides a platform to model designs from sophisticated, high-level systems all the way through to implementation. With the addition of advanced data types such as structures, unions and enumerations and advanced communication encapsulation mechanisms via interfaces, SystemVerilog enables proper capture of data flow as well as enabling improved design and code organization and readability. Behaviour can be expressed very abstractly at the beginning of a project and refined to the proper implementation abstraction level, RTL to gates. Assertions improve dynamic and formal checking as well as speeding debug and enabling more complete capture of the design constraints and assumptions. The addition of classes and random constraints and programs create a very powerful system for design within one simulation framework.

In summary, SystemVerilog introduced many design and validation constructs creating a unified language that can be used by both design and validation, as well as being able to represent a design at different levels of abstraction. It can be used to code a behavioural model of the design, implementing a test bench and formal properties as well as capturing structural models. These capabilities combined, create a very powerful and useful language framework.