SystemC Standard

Dr. Guido Arnout

President and CEO
CoWare, Inc.
Santa Clara, CA 95051
Tel : 408-748-2929
Fax : 408-748-2939
e-mail : guido@coware.com

Abstract - The emergence and great popularity of system-on-chip (SoC) designs has brought with it a variety of suggestions for a single language that can describe all of the functional requirements for those highly complex designs. This paper takes a look at the requirements for system-level design languages and evaluates what it will take for any of these languages to be successful.

I. INTRODUCTION

The hardware design world standardized on two languages for hardware design: VHDL and Verilog. Lately, there has been a growing effort to find a language that can describe the hardware at a much higher level. Also, as software has become a much more important part of all electronic systems, sometimes as much as 90% of the design is software, efforts have been made to find a system-level design language that can work for both hardware and software high-level descriptions.

The driving forces behind the urge towards a system-level design language are the simultaneous increase in design complexity, with multi-million gate designs, and increase in pressure to get designs out faster with first-time design success. System-level design is required to get new designs to market fast and to manage design complexity.

Because this is a new market, there are a number of suggestions for system-level design languages. This paper takes a serious look at system-level design languages and what it will take for a language to succeed in this fast-growing market.

II. THE NEED FOR A SYSTEM-LEVEL DESIGN LANGUAGE

There are four major reasons why a system-level design language is now necessary. First, SoC designs combine hardware and software, not just hardware only as in traditional ASIC design. As a matter of fact, there is more software than hardware in most designs (see Fig. 1). Therefore, there is a need for a language that describes the functionality of both the software and hardware. Hardware description languages only describe the hardware. In addition, major design elements can move from hardware to software and back to hardware in successive generations, particularly as standards are established or changed. Therefore, it is essential that the system be defined first, and the exact implementation (hardware or software) be established later in the design process.

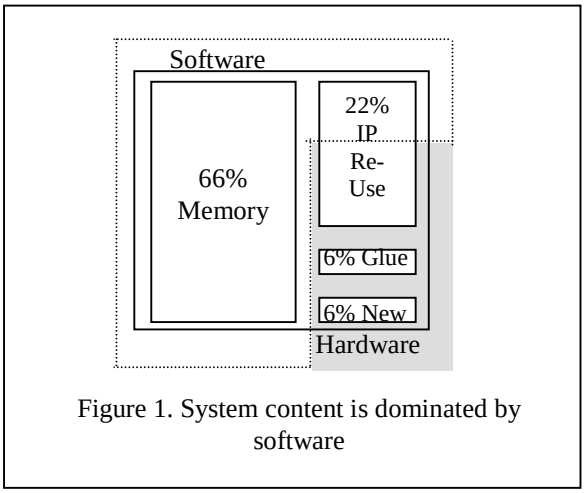

Figure 1. System content is dominated by software

Second, SoC designs are increasingly incorporating hardware and software intellectual property (IP) from various sources. All of these sources need to use a common system-level design language so the entire system can be modeled. Currently, with a variety of non-standard system-level design languages, IP vendors must choose the language(s) they want to support. Pitty the poor designer faced with integrating IP described in different and incompatible modeling languages.

Third, even hardware-only designs are becoming too complex to simulate in RTL. Simulating the entire design at a higher level provides much faster simulation times and lets the designer test the behavior of the entire chip before it is produced.

Fourth but definitely not the least important, system-level

design is no longer a luxury – it's a necessity. There's no way a multimillion-gate design can be successfully implemented the first time without careful system-level planning. System-level design is also required to develop a virtual prototype of the hardware that the software designers can use to begin software development. The old model of waiting until the hardware is finished to begin software development is totally out of date. Time-to-market is too important.

Integrated Circuit Engineering (ICE), a market research company based in Scottsdale, Arizona USA, claims that by 2004, 70% of all ASICs will be a system on a chip, and 85% of all ASICs will contain some form of licensed IP[1]. Therefore, there will be increasing pressure on the semiconductor vendors, ASIC suppliers, IP providers, and EDA tool vendors to standardize on a common language that can be used with tools and IP for system-level design.

III. WHAT IS A SYSTEM-LEVEL DESIGN LANGUAGE?

Before the various proposals for system-level design languages can be evaluated, it is important to clarify what exactly is a system-level design language. Fundamentally, a system-level design language is a notation that embodies semantics for describing a system prior to mapping it onto an architecture[2]. Components must be able to be described without making assumptions about the implementation. Take, for example, a microprocessor-based system. The system-level design language must have a way to describe the behavior of the components in the system irrespective of whether they will be mapped to software running on a microprocessor, or mapped to application-specific hardware. Equally, it should be possible to use the language to construct a performance model. Such a model allows exploration of the architecture of the system without specifying exactly what microprocessor or bus specification will be used in the implementation, or without specifying the functional behavior in full. That decision can be made later after the entire system is described and simulated at a very high level. Then the best microprocessor and bus architecture for that particular application can be selected and implemented. It should further be possible to add either performance detail to the behavioral model, or behavioral detail to the performance model, to achieve a mapping of function onto architecture.

The language should support notions of structure and hierarchy that allow hugely complex systems to be broken down into manageable blocks. Multiple levels of abstraction are also clearly necessary, to be able to start at a more abstract level and add implementation detail in a process of step-wise refinement. This applies both to the content of each of the blocks and the communication signals, data-types and protocols between them.

The language must include the ability to incorporate the description of constraints, such as event ordering, timing, dependencies and concurrency, with granularity and scheduling mechanisms. This introduces another area of complication. Particular mechanisms are often suited not just to a certain level of abstraction, but also to a certain application domain.

SDL[3], Esterel[4] and Lotos[5] are examples of languages, mainly for describing software, at a high level of abstraction in relatively narrow application domains. UML[6] shows signs of emerging as a more general level above C++ for software.

IV. DEVELOPING AN ENTIRELY NEW LANGUAGE

The fact that there was no one language that already existed that was perfect for system-level design was recognized years ago, but the pain of designing huge SoC designs made this very apparent by the mid-1990s. Over the past five years, there have been a number of attempts to define an entirely new language that would meet all of the requirements for system-level design.

A. Rosetta: The Search for an Ideal Language

In the fall of 1996, a language development effort was kicked off by the EDA Industry Council Project Technical Advisory Board (PTAB). This has evolved into the SLDL Initiative currently sponsored by VHDL International. The work being done on this can be seen from their web site at www.inmet.com/SLDL.

The SLDL Initiative quickly realized that it was not possible to meet all the criteria for a system-level design language, for all levels of abstraction in all application domains, with a single language. In 1999, this group proposed a new framework called Rosetta. The plans for Rosetta include the integration of multiple domain theories into a common semantic framework that supports the ability to budget and decompose system-wide capabilities and enable early estimation, co-synthesis and formal verification by EDA tools[7]. A preliminary and incomplete Rosetta semantic definition is available at that web site. It will take a few more years to flush out the specs for the language because of the complexity of this project.

Once Rosetta is finalized, it still will have to stand the test of being adopted by the hardware, software and architectural design communities. Design tools, including simulators, will need to be developed specifically for this language. Getting engineers to adopt a new language has proven, over the years, to be a very challenging and time-consuming process.

B. SuperLog to the Rescue?

One EDA company, Co-Design Automation, is suggesting that its SuperLog language is the ideal solution for future design. Twelve other EDA companies have announced their intention to bring out new tools to support SuperLog[8]. According to the Co-Design Automation web site at www.co-design.com, SuperLog provides the simplicity of

Verilog with the power of the C programming language. Yet it is clear that it does not meet all the criteria for a system-level design language and is clearly HDL-centric.

Again, there are significant challenges to be overcome to get designers to move to a new language. Time will tell if any system design companies will sign up to adopt this language in their organizations.

C. New Languages – Many Challenges

The fact that efforts exist to develop entirely new languages shows that the system-level design market is emergent in nature and a single solution is not yet apparent. All of the efforts discussed above should be applauded for their initiative and idealism. However, if the past is any indication of the future, it will be difficult to get the market to adopt any one of these languages.

V. EXTENDING VHDL OR VERILOG

If VHDL and Verilog lack the ability to describe hardware at a systems level, some have suggested that the best approach is to add those system capabilities to those languages. There are several efforts underway to do this.

A. OVI's Plans for Verilog

In November 1999, Open Verilog International (OVI) announced its plans for system-level design. A semantic reference manual (SRM) is being drafted, with a goal of standardization on semantics by June 2000. The SRM is also expected to define levels of abstraction, describe how hierarchy is expressed, talk about data objects and data types, scheduling mechanisms and protocol abstractions[9].

It is expected that OVI's effort will be much more focused on hardware design. According to OVI's Chairman Dennis Brophy, the work "is firmly rooted to enhance the productivity of the Verilog HDL user as a first step[10]."

B. Plans for VSPEC

Developed under the direction of Professor Perry Alexander at the University of Cincinnati and funded by an Arpa RASSP contract, VSPEC is a VHDL-based interface language that can represent both hardware and software as well as formal properties and constraints by adding annotation to VHDL entities[11]. According to the VSPEC web site at http://www.ececs.uc.edu/~kbse/projects/vspec/#VSPEC, the latest release of the VSPEC parser, which is still under development, was done in June 1999.

C. Other Plans

Over the past five years, there have been several other suggestions for extensions to VHDL and Verilog. Many of these suggestions have been noted in the press, but little follow-up action has been recorded. Some of these efforts have been abandoned. None have been finalized or accepted by customers.

D. The Challenge of Extending HDLs

It is now well accepted in the industry that existing HDLs cannot effectively be expanded to cover the full range of required system-level design capabilities for a variety of reasons. Probably the biggest reason, and the reason that much of this work has been abandoned, is that systems are mostly software. Why use a hardware description language to describe a system that is mostly software?

Another major contributing reason is that today's multi-million gate designs cannot effectively be simulated in HDLs because HDLs provide a much too low-level view of the hardware. By the time a designer is using an HDL, the system specification is already determined and the architecture is established. If HDL simulators are choking on big designs today, they will choke to death on the big system designs of tomorrow!

Additionally, HDLs don't provide for effective design re-use. Once something is designed in Verilog or VHDL, it becomes too implementation-specific to work in another design.

There seems to be little current interest in extending HDLs to the system level. Even VHDL International, the group chartered with the life of the VHDL language, is looking into new system-level design languages and abandoning efforts to extend VHDL.

VI. PROPOSALS BASED ON C/C++

While some have tried to develop totally new languages, others have taken a more pragmatic approach. Since between 60 and 90 percent of a system is software (this will continue to increase), and most software designers already use the C language, why not just do whatever is necessary to make the C language work for hardware designers?

The biggest benefit to using C/C++ is that many hardware designers already know how to use the language. It is commonly taught in colleges and universities. "The good news from a retraining standpoint is that the C++ design environment can be made quite familiar to a Verilog or VHDL designer[12]."

Before these C-based approaches are discussed, it is important to understand why the standard C/C++ language doesn't work for hardware design. The biggest problem with using C or C++ to describe hardware is that neither have a natural way to represent constrained data types, concurrency, and clocks.

All of the proposals based on C/C++ have some way of adding in the representations for hardware at a systems level. The disagreement in the industry is over how those representations should be made.

A. C-based Solutions

Some companies are suggesting using C. Most notable among these is the suggestion from C-Level Design with their C2Verilog product that takes the C-language models often used for system design and compiles them into synthesizable, register-transfer-level Verilog code[13]. The challenge with using C rather than C++ is that C requires proprietary extensions, such as compiler pragmas or keywords, to represent concurrency, clocks and other hardware design concepts. These proprietary extensions require the use of proprietary C compilers – ANSI standard compilers won't recognize these extensions.

B. CynApps's Effort

Also under development is a solution called CynLib from CynApps (www.cynapps.com). CynLib uses the class capability in C++ to add in hardware capabilities. A number of EDA companies have announced plans to develop tools based on CynLib's extensions to C in the future.

CynApps has announced the CynApps Suite of tools, including the Cynthesizer, which translates C++ into synthesizable Verilog, Cynchronizer, a Verilog-to-C++ translator, Cyn++, a Verilog-like macro language that works with CynLib, and Cyntax, a C++ lint tool that detects a variety of syntax errors[14].

The major limitation to CynLib is that the CynApps suite is specifically aimed at hardware design. System-level design plans are a "likely future direction[15]."

Time will tell if CynLib, like other proposals, will be adopted by a significant number of customers.

C. The SystemC Standardization Effort

In September of 1999, over 55 system, semiconductor, IP, embedded software and EDA companies came together to endorse the Open SystemC Initiative to enable, promote and accelerate system-level IP model exchange and co-design using a common C++ modeling platform. Available through an Open Community Licensing model, designers can create, validate and share models with other companies using SystemC and a standard ANSI C++ compiler. More information is available at www.systemc.org.

The goal of SystemC is to define a modeling platform using C++ class libraries and a simulation kernel, which provides greater interoperability, portability and readability. Before SystemC was proposed, there was no commonly accepted C++ style available in the industry, forcing companies to maintain multiple C++ models in order to exchange and re-use system-level models.

The benefit of using a C++ class library is that it lets designers express such hardware-oriented concepts as concurrency, parallelism, ports, wires and reactivity. "Agreeing on a standard library will allow IP providers and system designers to use one dialect of C++, thus allowing interoperability, and will provide a common platform on which to build synthesis, hardware/software co-design and verification tools[16]."

One area in which SystemC distinguishes itself from other C/C++ variants, particularly those that aim to simply replace Verilog, is in its communications abstractions. These constructs, proven over recent years in real SoC designs in the CoWare N2C design system[17], form a major part of CoWare's technical contribution to the Open SystemC Initiative. The communication abstractions enable designers to model communication between different system hardware components without having to design in specific low-level details such as bus protocols.

SystemC is the result of technical collaboration between Synopsys, CoWare and Frontier Design. Synopsys and CoWare had been developing similar C-based modeling solutions over recent years. Frontier Design and Synopsys collaborated on the fixed-point data types necessary for applications in digital communications, digital audio and digital video. Additionally, the pioneering research in C-based design at IMEC, MIT, Stanford and UCIrvine is the foundation of SystemC.

The goal of the steering committee for the Open SystemC Initiative is to finalize the specification (now available at a 0.9-level for industry review) in 2000. The committee feels it is very important that the industry review and comment on this specification to make sure everyone's feedback is considered.

VII. JUMPING TO JAVA

One last proposal that deserves a brief discussion is to use Java instead of C for system-level design. The problem is that Java just can't compete with C/C++ on simulation speed – Java currently is just too slow. C/C++ compiler technology is more mature, resulting in faster executables and therefore simulation time. Additionally, because Java prevents direct access to hardware, it is not suitable for writing initialization code and device drivers. Also, C/C++ enables easier re-use of legacy software developed for implementing and simulating current embedded systems.

VIII. CONCLUSION

This is an interesting time for system-level design. Certainly, system-level design is a critical issue as time-to-market can drive the success or failure of a product. In the consumer electronics product markets that are driving system-on-a-chip, a system-level design methodology is no longer something that can wait. HDL-based design flows have already run out of steam. Additionally, system-level design is the only way to go to develop products such as cellular phones that have constantly evolving standards. Designing at the system level makes it much easier to incorporate new features and standards and rush derivative products to market. CoWare's customers have proven, in numerous instances, that by using C++-based system design, overall design time can be cut in half.

In 1999 the industry made major steps in attempts at standardization. This year, 2000, will be critical as these

standards are finalized and brought to market. C/C++ is the only practical solution now. Designing in C/C++ represents a major raise in abstraction level for the hardware side. It is recognized that often C/C++ is the implementation level on the software side, since still higher levels of abstraction exist for design entry. However, these higher solutions are fragmented and domain-specific. Furthermore, the portion of the system designed with those solutions is invariably integrated with the rest of the system in C/C++.

The reality is that C/C++ is the only viable candidate for system-level design for the time being. How long is "the time being?" Acceptance of a new language will take six years according to Dataquest's Gary Smith[18]. Meanwhile, a C++ solution such as SystemC has the most promise. It is backed by the largest number of companies in a variety of industries. And the steering committee is really trying to gather input from all of these companies to make sure the libraries will work well for the industry.

Time will tell if any of the totally new system-level design languages will make it to market, let alone be adopted widely. Meanwhile, companies have huge systems to design. Certainly one solution will become a de facto standard.

REFERENCES

[1] Integrated Circuit Engineering, *"ASIC – System on a Chip,"* 1999.
[2] Steven E. Schultz, "The New System-Level Design Language," *Integrated System Design*, July 1998.
[3] www.itu.int
[4] www-sop.inria.fr/meije/esterel/esterel-eng.html
[5] ISO/IEC 8807
[6] www.omg.org/uml
[7] Steven E. Schultz, "Rosetta Could Set Direction for HDL," *Electronic Engineering Times*, November 1, 1999.
[8] Peter Clarke, "Superlog Language Gains a Dozen Backers," *Electronic Engineering Times*, November 22, 1999.
[9] Richard Goering, "Foundation Laid for Next-Generation Languages," *Electronic Engineering Times*, December 3, 1999.
[10] Richard Goering, "Foundation Laid for Next-Generation Languages," *Electronic Engineering Times*, December 3, 1999.
[11] Richard Goering, "Systems-on-Silicon Designs Talk in New Languages," *Electronic Engineering Times*, June 9, 1997.
[12] John Sanguinetti, "higher-Level Design Sees Plusses of C++," *Electronic Engineering Times*, November 1, 1999.
[13] Richard Goering, "C Level Design tackles C-to-HDL," *Electronic Engineering Times*, December 14, 1998.
[14] Richard Goering, "CynApps Leads Charge into C++ Hardware Design," *Electronic Engineering Times*, September 20, 1999.
[15] Richard Goering, "CynApps Leads Charge into C++ Hardware Design," *Electronic Engineering Times*, September 20, 1999.
[16] Richard Goering, "Synopsys Leads C++ Initiative to Transform Hardware Design," *Electronic Engineering Times*, September 27, 1999.
[17] Bolsens, De Man, Lin, Van Rompaey, Vercauteren and Verkest, "Hardware/Software Co-Design of Digital Telecommunication Systems, IEEE Proceedings, Vol. 85, No. 3, March 1997
[18] Gary Smith, Dataquest Software QuickTakes, Issue 34, October 11, 1999.