A Survey on System-On-a-Chip Design Languages

Ali Habibi and Sofiène Tahar

Electrical and Computer Engineering Department, Concordia University, Montreal, Quebec, Canada
Email: {habibi, tahar}@ece.concordia.ca

***Abstract*—Advancement in the microelectronics era made it possible the integration of a complete yet complex system on a single chip. Over 10 million gates, integrated together and running a real-time optimized software red crossed classical design techniques. Register level will serve as an assembly language for the new design languages or so called system level languages. The problematic is to define a language that can allow the design of such a complex systems. In this paper, we explore different paradigms of state-of-the-art of the System-on-a-Chip (SoC) modeling and design. In particular, this paper presents the main proposals in defining a system level language and discusses their advantages and drawbacks.**

I. Introduction

A decade ago, the EDA industry went progressively from gate level to register-transfer-level abstraction. This is one of the basic reasons why this process gained a great increase in the productivity. Nowadays, an important effort is being spent in order to develop system level languages and to define new design and verification methodologies.

The reason for all this activity is simple. Register-Transfer Level (RTL) hardware design is too low as an abstraction level to start designing multimillion-gate systems. What is needed is a way to describe an entire system, including embedded software and analog functions, and formalize a set of constraints and requirements—all far beyond the capabilities of existing Hardware Description Languages (HDLs). VHDL and Verilog both will become the assembly languages of hardware design. Designers have no choice but to write RTL code for things that are performance-critical. For everything else, they will stop at a higher level.

System level languages proposals can be classified into three main classes. The first is to reuse classical hardware languages such as extending Verilog [13] or VHDL [19]. Second, is readapting software languages and methodologies (C/C++ [21], Java [3], UML [9], etc.). The third is recreating new languages specified for system level design, e.g. Rosetta [2].

The rest of the paper is organized as follows. In Section 2, we will introduce the concept of a System Level Language (SLL). In Section 3, we consider the main proposals regarding the definition of a hardware SLL. We will focus on three different approaches: re-using existing hardware languages, adapting software languages and recreating new languages. In Section 4, we will conclude the paper.

II. System Level Design

Modern electronic systems grow in complexity encouraging combinations of different types of components: microprocessors, DSPs, memories, etc. On the other side, and in the same time, designers and programmers are asked to meet these demands for shorter development time and lower development cost. Designers agreed that there is no way to meet the difficult targets unless we shift to a higher abstraction levels in the design process, so-called System Level Design. System level design, as a new level of design, requires system level tools that simultaneously handle both hardware and software, for modeling, partitioning, verification and synthesis of the complete system.

A fact is that the level of highest abstraction will be the one that survives, and that is clearly the software domain. Considering the economic reality, this transition will not be abrupt, but it will occur more as an evolution than a revolution. The most likely transition will be along the lines that software followed as it evolved from a strict use of hand-coded assembler in the fifties to extensive use of compilers in the sixties.

The most realistic scenario will start by migrating the non-critical portions of time-to-market-driven designs to higher levels. Then, progressively over time, more sophisticated compiler and synthesis technology augmented by increasing hardware functionality will extend the reach of automatic techniques until only extremely critical portions of highly performance-driven designs will be implemented at the register transfer level.

Eventually, the software and hardware design will end by getting into a single flow. Optimistically, in a few years, the difference, if it will exist, will be a matter of compiler options ("-software" or "-hardware"). However, to get there, we need at first to define a system level language.

The usage of a system level language is a direct result of the way the SoC design flow is processed [14]. Following the software evolution, the most likely methodology would be to push towards successive refinement, an extremely successful methodology in software development.

No doubts, a short-term solution will come from languages that "push up" from or extend current HDL-based

Proceedings of The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications ISBN 0-7695-1929-6/03 $17.00 © 2003 IEEE

methodologies. We cannot omit that languages like SuperLog [24] hold a great promise for the next three to four years basically because SuperLog does not require a radical shift in methodologies and allows groups to retain legacy code. This approach can be seen as a refinement of existent languages.

The medium-range solutions will likely be C++-based languages that have hardware design capabilities, we mention here SystemC [27] and Cynlib [5] as two promising examples (representations). The revolution can also come from the Java Language side.

Long-term language solutions will, however come from new languages developed just specifically to work at the system level such as Rosetta [1], which promises to make true system-level design a reality.

III. SoC System Level Languages

A. Requirements for SoC System Level Languages

Any language proposing to support system-on-chip design must address two important design characteristics. These are the integration of information from multiple heterogeneous sources and the ability to work at high levels of abstraction with incomplete information [14]. This can be defined as:

- Support for integrating domain specific specifications
- Composition of models of computation that describe systems and components
- Support for more abstract modeling
- Composition of systems and components defined using different models of computation
- Representation and integration of constraint information
- Moving up from implementation technology
- Support for domain specific specification
- Support for predictive analysis and verification: within specific domains and across multiple domains
- Adherence to formal semantics.

B. Re-use of Existenting Hardware Languages: SystemVerilog

SystemVerilog [25] is a radically revised Verilog language reaching toward much higher levels of abstraction. SystemVerilog blends Verilog, C/C++ and Co-Design Automation's SuperLog [24] language to bring unprecedented capabilities to chip designers.

SystemVerilog [25] is a set of extensions to the IEEE 1364-2001 Verilog [13] to aid in the creation and verification of abstract architectural level models. The key features include interfaces that allow module connections at a high level of abstraction; C-language constructs such as globals; and an assertion construct that allows property checking. SystemVerilog includes the synthesizable subset of SuperLog, and its assertion capability will likely be derived from the Design Assertion Subset [24] that Co-Design and Real Intent Corp. recently donated to the Accellera standards body.

With all these enhancements, SystemVerilog may remove some of the impetus from C-language design, at least for register-transfer-level chip designers. The basic intent is to give Verilog a new level of modeling abstraction, and to extend its capability to verify large designs.

SystemVerilog borrows the C-language "char" and "int" data types, allowing C/C++ code to be directly used in Verilog models and verification routines. Both are two-state signed variables. Other new constructs include "bit," a two-state unsigned data type, and "logic," a four-state unsigned data type that claims more versatility than the existing "reg" and "net" data types.

The slogan of the people supporting Verilog is: "The right solution is to extend what works". However, this may face a harsh critic from the people supporting C++ based solutions and who advocate that Verilog or SystemVerilog or whatever is the Verilog based name will not offer a shorter simulation time. The debate is quite hard and the last word will not be said soon!

C. C/C++ Based Approach

Some optimistic software designers are supporting the establishment of C/C++ [8] or even Java [12] based languages for future SoC designs. They think that, over time, improvements in automatic methods and increases in hardware functionality will extend the pieces handled automatically to where whole designs can be implemented by "cc -silicon". In the near term, however, tools are not going to be good enough and manual refinement will be required. This means that the version of the language that will be used must allow for the expression of hardware at lower levels, not only at the algorithmic level.

Over the past decade, several different projects have undertaken the task of extending C to support hardware [7], including SpecC [8] at the University of California, Irvine, HardwareC [15] at Stanford University, Handel-C [7] at Oxford University (now moved to Embedded Solutions Ltd.), SystemC++ [18] at C Level Design Inc., SystemC [27] at Synopsys Inc., and Cynlib [5] at CynApps.

This variety of projects falls roughly into two complementary categories. The first, exemplified by SpecC, has focused on adding keywords to the basic C language, supporting hardware description at a high level as a basis for synthesis. The second, exemplified by SystemC, exploits the extensibility of C++ to provide a basic set of hardware primitives that can be easily extended into higher level support [21]. These two complementary approaches span all levels of hardware description (algorithmic, modular, cycle-accurate, and RTL levels).

1) C-Based Solutions

As a C-based solution, in this paper, we will consider the case of SpecC and HardwareC.

a) SpecC

The SpecC language [8]is defined as extension of the ANSI-C programming language. This language is a formal notation intended for the specification and design of digital embedded systems including hardware and software. Built on top of ANSI-C, the SpecC supports concepts essential for

embedded systems design, including behavioral and structural hierarchy, concurrency, communication, synchronization, state transitions, exception handling and timing [8].

To defend SpecC against C++ proposals and mainly SystemC, when this latter was first introduced in 1999, SpecC supporters advanced that SystemC is primarily aimed at simulation, however, SpecC was developed with synthesis and verification in mind. They also considered that SystemC targets RTL design, but SpecC is a system-level design language intended for specification and architectural modeling. Nevertheless, after the release of the version 2.0 of SystemC, all these argument were broken. In fact, SystemC is nowadays supporting most system level design requirements.

b) HardwareC

Under the same class of C-based languages we find also a language called HardwareC [16]. This is a language that uniformly incorporates both functionality and design constraints. A HardwareC description is synthesized and optimized by the Hercules and Hebe system [6], where tradeoffs are made in producing an implementation satisfying the timing and resource constraints that the user has imposed on the design. The resulting implementation is in terms of an interconnection of logic and registers described in a format called the Structural Logic Intermediate Format [15].

HardwareC attempts to satisfy the requirements stated above. As its name suggests, it is based on the syntax of the C programming language. The language has its own hardware semantics, and differs in many respects from C. In particular, numerous enhancements are made to increase the expressive power of the language, as well as to facilitate hardware description [16].

2) C++-Based Solutions

As any real system a SoC is composed by a number of entities and objects interacting together. This is the reason for the limitation of C-based proposals. By reference to software design, languages that can be used at the system level have to be preferably object-oriented. That is why nowadays more mature proposals are based on C++. In this paper, we will discuss mainly two C++ based proposals: Cynlib and SystemC.

a) Cynlib

Cynlib [5] provides the vocabulary for hardware modeling in C++. It is a set of C++ classes which implement many of the features found in the Verilog and VHDL hardware description languages. It is considered as a "Verilog-dialect" of C++, but it is more correct to say that it is a class library that implements many of the Verilog semantic's features. The purpose of this library is to create a C++ environment in which both hardware and testing environment can be modeled and simulated.

Cynlib supports the development of hardware in a C/C++ environment. To do this, Cynlib extends the capabilities of C/C++ by supplying many features such as: concurrent execution model, cycle-based, modules, ports, threads, etc.

b) SystemC

SystemC [27] comes to fill a gap between traditional HDLs and software-development methods based on C/C++. Developed and managed by leading EDA and electronics companies, SystemC comprises C++ class libraries and a simulation kernel used for creating behavioral- and register-transfer-level designs. Combined with commercial synthesis tools, SystemC can provide the common development environment needed to support software engineers working with C/C++ and hardware engineers working in HDLs such as Verilog or VHDL.

New releases of Cadence Design Systems Inc.'s Signal Processing Worksystem (SPW) [4] software and Axys Design Automation Inc.'s MaxSim Developer Suite increase support for SystemC. Mentor Graphics Corp. has also added a C language interface to its Seamless hardware/software co-verification environment [11] that lets designers use mixed C/C++ and HDL descriptions for hardware. The interface, called C-Bridge, will be included with Seamless version 4.3, which is to be released in late March 2003 [11].

D. Java-Based Proposals

While there has been some discussion about the potential of Java as a system-level language or high-level hardware description language, LavaLogic may be the first commercial EDA provider to bring that option into contemporary design systems with a language called JHDL [3]. LavaLogic's Java-to-RTL compiler is an "architectural synthesis" tool that turns Java into synthesizable HDL code. LavaLogic is offering a tool that will take high-level Java descriptions down to gate-level netlists, starting with FPGAs [12].

According to Java advocates, Java appears to be the purest language to solve the productivity problems currently at hand. They also claim the language can express high-level concepts with far less code than today's HDLs, yet offer more support for concurrency than C or C++.

The main point all Java advocates stressed in comparing their approach to the C/C++ based ones is the concurrency. In fact, a classic problem with C and C++ is their inherent inability to express concurrency. In Java, in contrast, concurrency can be explicitly invoked with threads. Nevertheless, this unique criterion for comparison is not enough to balance the choice C/C++ to Java. Java can be classified as the "next best" choice after C++, since it does not support templates or operator overloading, resulting in a need for numerous procedure calls.

E. Developing a New Language: SuperLog

SuperLog [24] is a Verilog superset that includes constructs from the C programming language. Because of its Verilog compatibility, it has earned good reviews from chip designers express considerable skepticism about C language hardware design. SuperLog holds great promise for the next three to four years. It is the case mainly because SuperLog does not require a radical shift in methodologies and allows groups to retain legacy code.

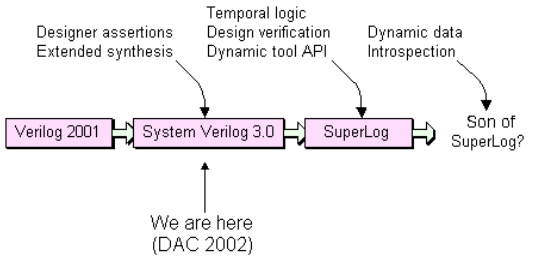

Fig. 1. From Verilog to SupeLog [17].

SuperLog (sometimes said to be "Verilog done well!" [23]) combines the simplicity of Verilog and the power of C, and augments the mix with a wealth of verification and system features [10]. For now, we have Verilog 2001 at the lower end of the sophistication spectrum ("jolly nice but lacking a lot of features" [17]) and SuperLog at the other end ("incredibly powerful, but as yet falling well short of industry-wide support" [17]). This is where Accellera reenters the picture to come up with a common meeting ground, which is SystemVerilog 3.0, Figure 1 [17].

In a certain way, as defined, SuperLog is a smart idea to deal with system level design. In fact, SuperLog utilizes the power of C with the simplicity of Verilog to provide the right balance for productive design.

IV. Conclusion

System-on-a-Chip design, as an open project, initiated a debate between hardware and software communities. The real issues for embedded chip design are increasingly embedded software issues. Three main trends are proposed to design SoC. The software based methodologies consider that basing everything from the HDL approach is starting from the wrong point. On the other side, hardware designers are looking into upgrading Verilog to support system level design requirements. They think that the best is to go for what it works. The third group is supporting the idea of defining new languages designed from scratch for system level design. Nevertheless, this last approach is facing very hard critics because most designers think that the idea of a grand unified language is fundamentally flawed. The best choice will be to have multiple languages targeting specific areas. These languages will be easier to learn than a grand language and they will also be easier for tools to parse.

Whatever the syntax looks like C, Verilog, VHDL or UML, most options share a common goal: the transition of functional design from the minutiae of Boolean logic, wiring and assembly code up to the level of the designer's issues. Thus, functionality becomes modeled as an architectural interaction of behaviors, protocols and channels. This is great, except that most languages handle only digital functionality and cannot comprehend trade-offs of power, timing, packaging or cost.

There are also shortcomings with today's methodology in the face of the growing use of SoCs. Individual designs that were once an entire system are now blocks in an SoC. The associated verification strategy with many of these blocks was not designed to scale up to a higher integration level. If the methodology does not permit easy integration of block-level verification code into a system-level SoC environment, then the verification task will become a major bottleneck to the entire system design flow.

References

[1] P. Alexander and D. Barton, "*A Tutorial Introduction to Rosetta*," Hardware Description Languages Conference, San Jose, CA., March 2001.

[2] P. Alexander, R. Kamath and D. Barton, "*System Specification in Rosetta*", presented at the IEEE Engineering of Computer Based Systems Symposium, Nashville, Edinburgh, UK. April 2000.

[3] P. Bellows and B. Hutchings, "*JHDL - An HDL for Reconfigurable Systems"*, IEEE Symposium on FPGAs for Custom Computing Machines",IEEE Computer Society Press",Los Alamitos, CA, Kenneth L. Pocek and Jeffrey Arnold, pp. 175-184, 1998.

[4] Cadence Design Systems, 919 E. Hillsdale Blvd., Foster City, CA 94404, USA, SPW User's Manual.

[5] CynApps, Inc. "*Cynlib: A C++ Library for Hardware Description Reference Manual*". Santa Clara, CA, USA, 1999.

[6] G. De Micheli and D. C. Ku, *HERCULES - A System for High-Level Synthesis*, Proceedings of the 25th ACM/IEEE conference on Design automation, Atlantic City, New Jersey, United States, pp. 483-488, 1988.

[7] G. De Micheli, "*Hardware Synthesis from C/C++ models",* Proceedings of the conference on Design, automation and test in Europe, Munich, Germany, pp. 80, 1999.

[8] R. Domer, D. D. Gajski, A. Gerstlauer, S. Zhao and J. Zhu, "*SpecC: Specification Language and Methodology*", Kluwer Academic Publishers, 2000, pp 313.

[9] B. P. Douglass, "*Real-Time UML*", Addison-Wesley, 2000.

[10] P. L. Flake and Simon J. Davidmann, "*SuperLog, a unified design language for system-on-chip*", Proceedings of the Asia and South Pacific Design Automation Conference, pp. 583 -586, 2000.

[11] R. Goering, "Mentor *adds C interface to verification environment*", EE Times, Feb. 2002.

[12] B. Hutchings and B. Nelson, "*Using General-Purpose Programming Languages for FPGA Design*", Proceedings of Design Automation Conference, pp. 561-566, June 2000.

[13] IEEE Std. 1364-2001, IEEE Standard for Verilog Hardware Description Language, 2001.

[14] A. A. Jerraya et *al.,* "*Multilanguage Specification for System Design and Codesign", In System Level Synthesis* , NATO ASI, 1998.

[15] D. C. Ku, G. De Micheli, "*High-level Synthesis and Optimization Strategies in Hercules/Hebe"*, Proceedings of the European Event in ASIC Design (EuroASIC) Conference, Paris, France, May 1990.

[16] D.C. Ku and G. DeMicheli. "*Hardware C - a language for hardware design", Technical* Report CSL-TR-88-362, Computer Systems Lab, Stanford University, August 1988.

[17] C. Maxfield, "*A plethora of languages*", EE Design, August 2001.

[18] M. Meixner, J. Becker, Th. Hollstein, M. Glesner. "*Object Oriented Specification Approach for Synthesis of Hardware-/Software Systems.*" Proc. GI/ITG/GMM Workshop Methoden und Beschreibungsprachen zur Modellierung und Verifikation von Schaltungen und Systemen, 1999.

[19] ModelSim, "*Foreign Language Interface*", User Manual, Version 5.5e, Aug 2001.

[20] S. Ohr, "*DSP design tools shift to SystemC*", EE Times, Mar 2002.

[21] R. Roth and D. Ramanathan. "*A High-Level Hardware Design Methodology Using C++*" 4th High Level Design Validation and Test Workshop, San Diego, pp 73-80, 1999.

[22] J. Rumbaugh, I. Jacobson and G. Booch, "*The Unified Modeling Language Reference Manual*", Addison-Wesley, 1999.

[23] M. Santarini, "*System-level languages won't appear overnight, experts say*", EE Times, March 2001.

[24] SuperLog. http://www.SuperLog.org, 2003.

[25] S. Sutherland, The Verilog PLI Handbook, Kluwer Academic Publishers, 1999.

[26] S. Swan, "*An Introduction to System Level Modeling in SystemC 2.0*", Open SystemC Initiative Cadence Design Systems, Inc. May 2001.

[27] SystemC. http://www.SystemC.org, 2003.