Hardware Synthesis from C/C++ Models

Giovanni De Micheli
CSL - Stanford University
Stanford, CA 94305

Abstract

Software programming languages, such as C/C++, have been used as means for specifying hardware for quite a while. Different design methodologies have exploited the advantages of flexibility and fast simulation of models described with programming languages. At the same time, the mismatch (of software languages) in expressing power (for hardware systems) has caused several difficulties. In the recent past, novel approaches have helped in reducing the semantic gap, and in easing the creation of design flows that support system-level specifications in C/C++.

Hardware Synthesis from C/C++ Models

The current and future design of electronic circuits and systems is characterized by several features and constraints. First, the system complexity is increasing, and at the same time the design time has to shrink. Thus, design methodologies and tool flows must support synthesis from high-level specifications and fast means of verification. Second, most electronic design will target embedded systems, with an increasingly larger component of software. Efficient design and optimal implementation require exploiting hardware/software co-design strategies. Balancing the hardware and software components in the search of on optimal implementation may require the *migration* of software blocks to hardware or vice versa. Third, design of complex circuits and systems will leverage more and more the *re-use* of existing hardware and software components. Efficient component re-use requires specification at a high-level of abstraction as well as the ability of mapping the specifications to different targets. Programming language models in C/C++ can be compiled into object code for several architectures, and it is highly desirable to be able to compile them into hardware as well.

Within this context, it is clearly obvious why designers write functional models of hardware/software systems, as well as of hardware circuits, using familiar programming languages. Functional models can be evaluated quickly by simulation, and can be directly compiled when a software solution is sought. At the same time, *legacy* models of software functions written in programming languages can be used as part of a new system design.

The drawbacks of using programming languages for hardware design are a few. I shall outline the most important ones. First, hardware circuits can execute operations with a wide degree of concurrency. Conversely, software programming languages like C/C++, were conceived for uni-processor sequential execution. Second, the specifications of hardware circuits entail some structural information. For example, the interface of a circuit with the environment (and/or internal blocks) may require the definition of the input/output ports and the data formats across these ports. In addition, a designer may want to express hints (or constraints) on some partitions of the circuit. Such structural information and constraints are missing from programming languages. Third, detailed timing of the operations is very important in hardware, because of performance and interface requirements of the circuit being described. On the other hand, most programming languages do not support timing constructs.

As a result, it is common practice that designers model circuits and systems using C/C++, perform functional simulation, and then translate the portion of the model to be implemented in hardware in a suitable subset of a *hardware description language* (HDL), such as Verilog HDL or VHDL, that can be synthesized into logic gates. Needless to say, this manual translation is a time-consuming, error-prone and tedious task.

Over the last decade, a few research groups have tried to ease the mapping of hardware models in programming languages into corresponding HDL models. Most approaches both extended and restricted programming language constructs. Extensions are needed to express concurrency, structural information (partitions, I/Os, data formats) and various types of constraints. Restrictions were motivated by avoiding constructs with no hardware meaning (e.g., print statements), as well as avoiding constructs whose translation into hardware is difficult (e.g., pointers).

Whereas extending a software model with annotations to support hardware synthesis is a task usually acceptable by a designer, because he just adds information to steer a particular implementation, the restriction on the usable constructs is problematic. For example, a designer who has a legacy model in the C language that uses pointers, and who has a design system that does not support pointers, has to re-write the model. Sometimes, model re-writing is more time consuming than generating a new model from scratch. As a result, the research trend in this area has been to extend the subset of C/C++ that can be synthesized into hardware as much as possible. At the same time, an important research objective is to insure that the mapping of C/C++ models to hardware is efficient, i.e., not wasteful of silicon area, memory space and performance.

I shall briefly summarize the research trajectory in this area over the last decade. For the sake of conciseness, I shall report on the major contribution that differentiate modeling and synthesis approaches. In addition, I shall concentrate on the use of C/C++ for hardware modeling and synthesis. A very large body of work exists in modeling hardware with programming languages with only simulation support; this will not be reported here.

Stroud et al. [8] developed the design system CONES at AT&T in the late eighties. As the name suggests, a cone is a block of combinational logic, whose output (cone vertex) is either a primary output or an input to a register. Cones of combinational logic were modeled in the C programming language, using assignment, branching and iterative constraints. Such cone models were expanded into *sum of products*, and then minimized.

At the same time, Ku et al. [4] developed a language called HARDWAREC, with a much larger expressing power than CONES and that could be fully synthesized. Nevertheless, HardwareC is not just annotated C: it differs in semantics and in the available constructs (e.g., it does not support pointers, but supports send/receive). HARDWAREC has a C-like syntax, and a cycle-based hardware semantics. It supports concurrency, structural and timing constraints, and has an unambiguous hardware semantics, thus making it a suitable front-end for synthesis. The major drawback of HARDWAREC is just being yet another hardware design language, because translation from C to HARDWAREC is not trivial, even though possible with limited effort.

A strong interest of system and vendor industries in C-based hardware design surfaced in the last few years. Most efforts were devoted to embedding the support of C/C++ models into existing design flows, that are compatible with other language models (e.g., Verilog HDL, VHDL). Thus most commercial product focus on design language interoperability and support for interface design. Notable examples are the SCENIC design environment from Synopsys [5], the CYBER design system from NEC, the COWAREN2C toolset from Coware [11], ARTBUILDER from Frontier design [9] and C2VERILOG from C Level Design (formerly Compilogic) [10]. The first three environments and toolsets are described in detail in subsequent papers in this volume.

At the same time, some research groups have explored alternative design paths using C/C++ based model. For example, Ernst et al. [3] developed a design system, called COSYMA that extracts a computational kernel from a C model, and implements it as a hardware co-processor for accelerating software execution. Some groups looked at mapping (part of) a C-model to field-programmable gate arrays, again with the purpose of achieving execution speedups, and thus achieving *re-configurable* co-processors. Researchers at IMEC have developed the OCAPI design environment, that supports C++ based design [6].

Despite all these efforts, providing a synthesis path from an ANSI C model to hardware is still a formidable task. Dynamic memory allocation and recursion require the use of dynamically-changing storage structures, which cannot be cast into hard-wired circuits, as typically done in hardware synthesis. Solutions may come from incorporating memory synthesis techniques [2] into hardware synthesis. The next generation hardware circuits, and corresponding design tools, will probably benefit from having large, local embedded DRAM arrays.

Pointer resolution for hardware synthesis has been though of as a hard problem for a long time. The SPC toolset has demonstrated that most pointers can be resolved at synthesis time [7]. Moreover, pointer resolution and encoding may be done in a way to generate efficient hardware circuits.

In summary, we are now at a stage in which C/C++ models can be synthesized in hardware, thus making it easier to migrate software models into hardware and to avoid HDL-level hardware specification. There are still a few unresolved technical problems to be solved, and present techniques need to be perfected. But examples of C/C++ models that have already been mapped (with synthesis tools) into product-level industrial designs prove that the concept of C/C++ based design is viable and convenient.

References

[1] A. Cataldo, “NEC spins C variant to ease Logic synthesis,” *EE Times*, July 10, 1998.

[2] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele and A. Vandecappelle, “Custom Memory Management Methodology Exploration of Memory Organisation for Embedded Multimedia System Design,” *Kluwer*, 1998.

[3] R.Ernst, J.Henkel and T. Benner, “Hardware-Software Co-synthesis for micro-controllers,” *IEEE Design & Test*, pp. 64-75, December 1993.

[4] D. Ku and G. De Micheli, *High-Level Synthesis of ASICs Under Timing and Synchronization Constraints*, Kluwer Academic Publishers, 1992.

[5] S. Liao, S. Tjiang and R. Gupta, “An Efficient Implementation of Reactivity for Modeling Hardware in the Scenic Design Environment,” *Design Automation Conference*, pp. 70-75, 1997.

[6] P. Schaumont, S. Vernalde, L. Rijnders, M. Engels, and I. Bolsens,. “A Programming Environment for the Design of Complex High Speed ASICs,” *Design Automation Conference*, pp. 315-320, 1998.

[7] L. Sèmèria and G. De Micheli, “SpC: Synthesis of Pointers in C, Application of Pointer Analysis to the Behavioral Synthesis from C,” *ICCAD, Proceedings of the International Conference on Computer Aided Design,* San Jose, CA, November 1998, pp. 340-346.

[8] C. Stroud, R. Munoz and D. Pierce, “Behavioral Model Synthesis with Cones,” *IEEE Design & Test of Computers*, June 1988, pp. 22-30.

[9] www.frontierd.com/artbuilder.htm

[10] www.compilogic.com/c2v.htm

[11] www.coware.com/n2c.html