Panel:

The Next HDL: If C++ is the Answer, What Was the Question?

Chair: Rajesh Gupta, UC Irvine
Organizers: Shishpal Rawat, Intel Corporation and Ingrid Verbauwhede, UCLA

Abstract

The focus of this panel is on issues surrounding the use of C++ in modeling, integration of silicon IP and system-on-chip designs. In the last two years there have been several announcements promoting C++ based solutions and of multiple consortia (SystemC, Cynapps, Accellera, SpecC) that represent increasing commercial interest both from tool vendors as well as perhaps expression of genuine needs from the design houses. There are, however, serious questions about what value proposition does a C++ based design methodology bring to the IC or system designer? What has changed in the modeling technology (and/or available tools) that gives a new capability? Is synthesis the right target ? or Validation? Tester modeling or testbench generation? This panel brings together advocates and opponents from the user community to highlight the achievements and the challenges that remain in use of C++ for use in microelectronic circuits and systems.

Position Statements

Gerard Berry
Esterel Technologies, France

Modern electronic design deals with larger and functionally richer objects than before. They must be described at several interrelated levels: physical netlists, RTL, behavioral, embedded software, etc. Checking the functional correctness of large designs can only be done at higher levels of abstraction, using fancy simulation and verification algorithms. Classical HDLs are targeted to lower levels and are inadequate for general algorithmic programming. Therefore, it is tempting to embed both the designs and the algorithms in a general-purpose language such as C++, relying on mechanisms such as object-orientation to ease integration. Then, designs can be massaged, simulated, and verified in a single framework, whose linguistic basis furthermore belongs to the common background of most users.

However, there are many hidden drawbacks. C++ is a complex language with a complex semantics. It is unclear that the embedded designs will themselves have a well-defined semantics, a preliminary to any verification activity. Object-oriented design is much more complex than beginners believe, and badly designed object structures become a rigid nightmare. Code reuse is non-trivial: template libraries inflate rapidly even for classical data structures. Finally, making users agree on unified, understandable, and flexible C++ descriptions of the various levels will be difficult.

Problems will not be magically solved by Esperanto-C++, as for any other field of computing. The initial enthusiasm should rapidly leave place to a serious study of the concerns above. Finally, there is clearly still room for high value-added cleanly designed languages for specific problems (data paths, pipeline structures, controllers, etc.).

Ramesh Chandra
ST Microelectronics, San Diego, CA

The proliferation of C++ based design techniques and the wide interest generated is an indication of the fact that the designers were increasingly facing questions that were not answered satisfactorily with current HDL-based design techniques. These questions include:

- Shorter TTM and more complex designs are demanding higher designer productivity and design management. This requires moving up a ladder in design abstraction. C++ is an ideal choice for behavioral levels and beyond.

- Verification is increasingly becoming a bottleneck in delivering designs. The designers are faced with multiple vendor specific simulation tools, languages, co-simulation, PLI etc. C++ integrates the design and verification into a common framework, facilitating improved verification productivity.

- SOC designers are increasingly facing the need for a common platform for hardware-software co-design. Until the arrival of recent C++ based techniques they had to manage separately the HDL for hardware and c/c++ for software and struggle at times with a late integration.

- Hardware architecture design and design exploration was mostly limited to careful architecture and RTL design by expert designers and gate-level optimizations by synthesis tools. C++ based design techniques have opened new frontiers for architecture and design space exploration.

- For complex designs, functional or performance models have normally been built in C/C++ and at some point the design with HDL is kicked off. With the advent of C++ oriented techniques, the designer can now make a smooth and transparent migration from modeling to design phase and maintain a complete consistency between the executable models and design implemented through different phases.

Daniel Gajski
University of California, Irvine

"C++ as presently practiced in EDA industry is creating more chaos than progress and if the path is followed it will slow down progress in EDA for 10 years to come." I make this statement based on two observations: firstly, let us not deceive ourselves that adding classes is just another use of C++. In fact, we have to learn the meaning of every class and understand what it is going to do in our design. Therefore we are creating a new language—we call it C++, but C++ with classes for hardware is essentially a new language. Secondly, simulation is a weaker concept than synthesis and verification. As long as it is syntactically correct and produces correct results, anybody can make a model and do simulation. To synthesize and verify, we have to understand what that model means. Therefore, synthesis and verification imposes much stronger requirements on the language, on the tools, and every-

thing else. Therefore, the only way to solve this problem is to have some fixed levels of abstraction and their meaning, which is semantics, so that I know what you mean when you write a + b. Syntax is not enough. Consider VHDL: it is a simulation language, and it really is not synthesizable. However, after ten years of messing around, we have a VHDL subset, in fact, many subsets introduced by different EDA vendors, which are synthesizable. To avoid this mess for the coming next 10 or 15 years, let us now define a synthesizable subset and not expand it. Then we will have one language that's synthesizable, verifiable, and simulatable

Kris Konigsfeld
Intel Corporation

C++ is a bad answer to a poorly phrased question. The quest for abstraction, HW/SW co-simulation, simulation performance, faster SOC development, and inexpensive development environments are all reasonable requests of hardware developers and architects. In many domains, hardware developers are coding analysis and automation solutions in languages much better than their HDLs allow them to build hardware. When they code in their HDL, they feel overly restricted and without power to express large portions of the design. Architects, executing in a less constrained environment, have employed C/C++ solutions to achieve fast and abstract descriptions of large systems. So why isn't C/C++ the answer to the designer's dilemma? It is the same reason that Verilog and VHDL were invented instead of using C. The constraints of language analysis, code predictability, hierarchy analysis, netlist generation, synthesis, correct by construction development, and controlled parallel execution all point to an HDL language solution.

The problem is that the EDA industry has been unable or unwilling to re-define, extend, or re-invent a language that comprehends the power of a HDL but achieves the requirements motivated by larger designs.

C++ and other object-oriented languages have improved coding productivity, enabling larger software development projects with faster production and less defects. I believe the EDA industry should use this power to produce a better HDL, not eliminate it.

So, the question is: "What's going to be the definition of the HDL that enables 21st century hardware development?" I don't believe C++ is a good answer, it is just the easiest.

Patrick Schaumont
IMEC, Leuven, Belgium

This position statement builds on the opinion of designers with hands-on experience in creating silicon proof with C++ based design tools. In the past years, several designs were completed at IMEC using C++ design technology. Feedback was collected from their designers by interview and their messages are summarized as follows.

For design at system level, C++ is king. At block level, a symbiosis of HDL and C++ likely is needed. In any case there are clear benefits that come with the use of C++ for electronic system design. First, some specialized system design tasks like fixed point refinement, operation profiling and the construction of hybrid simulations are easy to do with a programming language like C++. In addition, advanced design styles based on object oriented techniques are possible. Next, the openness of C++ provides an ideal container for design management like script creation, gluing of heterogeneous formats and custom code generation.

So where does the confusion, suggested in the panel question, come from? To begin with, advocating C++ as a replacement HDL is a bad idea. It creates false expectations, and it offers a solution on top of an existing one.

Next, the introduction of new design languages like C++ requires education. Universities help in this process. But also EDA companies should not leave everything to their marketing departments. Also the system houses should take some time to really try things out rather then hoping to hear canned answers on trade shows.

And finally, let's keep an eye on the core of the problem. That is, the art of design itself is changing and new design methodologies are needed.

Ingrid Verbauwhede
University of California, Los Angeles

Looking at the latest ISSCC proceedings for true systems on a chip: a fully integrated Bluetooth contains a RF front-end, a baseband processor, an embedded micro-controller for the complete protocol stack, audio codecs, etc. A universal cable set-top box contains cable modems, 2D/3D graphics processor, several ADC's and DAC's, an embedded micro-controller, and so on.

What components of these SOCs can be efficiently modeled, verified, designed with C++? The answer is not the RF front-end, not the ADCs and DACs, not the FFT architecture, not the parallel processing of the video processor. Most likely the protocol stack, some parts of the baseband processors and cable modems can be modeled using C++. It is still possible to model the other components using C++ for verification purposes, but a designer will not want to use it for design purposes because the level of abstraction is not right: for instance, how do I describe efficiently a FFT butterfly in C++?\

I believe C++ will work fine to capture sequential processing. To quote an experience, a design experiment with graduate students showed that the design time for a speech processing algorithm (sequential in nature) in a C++ design environment, took the same amount of design time as implementing the same algorithm on a programmable DSP. However, for C++ to be accepted by the broader design community, the technology proponents have to realize that their technology has a useful but limited application field. Even more important, education of designers is a must.