

Getting Hardware and Software to Speak the Same Language

By providing an industry standard language that can be used for system, hardware and software design, SystemC is enabling systems, hardware and software teams to "speak" the same language. This means much better communications between the groups. It also means that, as the software and hardware are developed, they can be simulated against the system specification so any design problems can be resolved much earlier in the design process.

In today's world of multi-national design efforts, one language for system-level design is imperative. If someday some group manages to develop the world's ideal system-level design language, then it can be considered for adoption. Meanwhile, everyone is trying to deal with the complexity of today's designs. SystemC will go a long way towards shortening the time it takes to get good system-level design tools to market. SystemC will also provide a common language for IP model exchange, which is essential as well.

INTRODUCTION

The hardware and software worlds are colliding in ways almost unimaginable ten years ago. In the 1980s, products were highly proprietary. In the 1990s, proprietary products gave way to standards-based architectures with much wider adoption. Now speed to market is critical, as the loss of a few months can mean the loss of a major selling season, such as Christmas in the U.S., or even worse, loss of significant market share.

With the need to quickly get new products to market that incorporate rapidly changing industry standards, what is implemented in software in one version of the product is moved to hardware in the next, then back to software as standards and features change. Consumer and communications products are experiencing this the most, as new high-speed broadband and wireless protocols are fighting for market dominance, new capabilities are introduced, and products have shorter and shorter life cycles.

How are design teams that never worked together before - possibly never even knew each other or worked in different locations - going to handle this

rapid transition? It's particularly hard because hardware engineers "speak" in hardware description languages like Verilog and VHDL while software developers "speak" in software languages such as C or C++.

A SYSTEM IS MOSTLY SOFTWARE

Since a modern system, such as those in today's Systems-on-a-Chip (SoC), is often 80% software and 20% hardware, and most system and software design is done in C/C++, it makes sense for hardware to move up to C/C++ for high-level modeling. However, hardware designers have been hesitant to move up to C/C++ because the C language is missing some very important constructs for hardware design, such as a good way to describe clocking and concurrency.

Since the C language lacks these important constructs, most hardware designers have viewed designing in C as an intellectual exercise to be confined to the domain of the system architect or system integrator. After all, they would just have to re-code everything in VHDL or Verilog after the C design was complete.

HARDWARE DESIGNERS WRESTLE WITH COMPLEXITY

Two big trends came together in the late 1990s to make more hardware engineers take a second look at the C language. First, chips started to get really big. Hardware simulators were choking on the complexity. Only parts of the big chips could be simulated at one time. Hardware designers started to look to alternative verification programs such as formal verification, using equivalency checking on the huge designs. But equivalency checking doesn't tell the designer if the intent of the design is correct. Daring hardware designers started simulating higher level descriptions of major parts of the chip design in C/C++, just to deal with the complexity.

Second, designers wanted to re-use more and more parts of the design, either by creating their own Intellectual Property (IP) blocks, or by using IP available

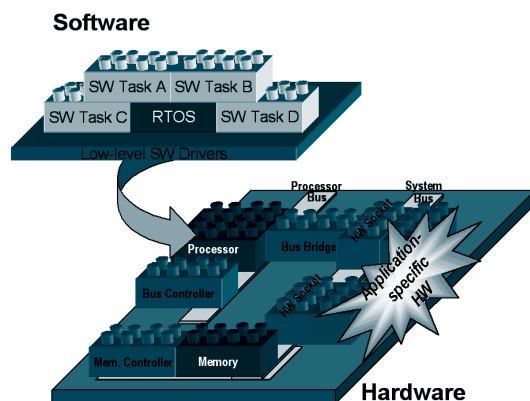


Figure 1. The Challenge, getting the software to work with the hardware.

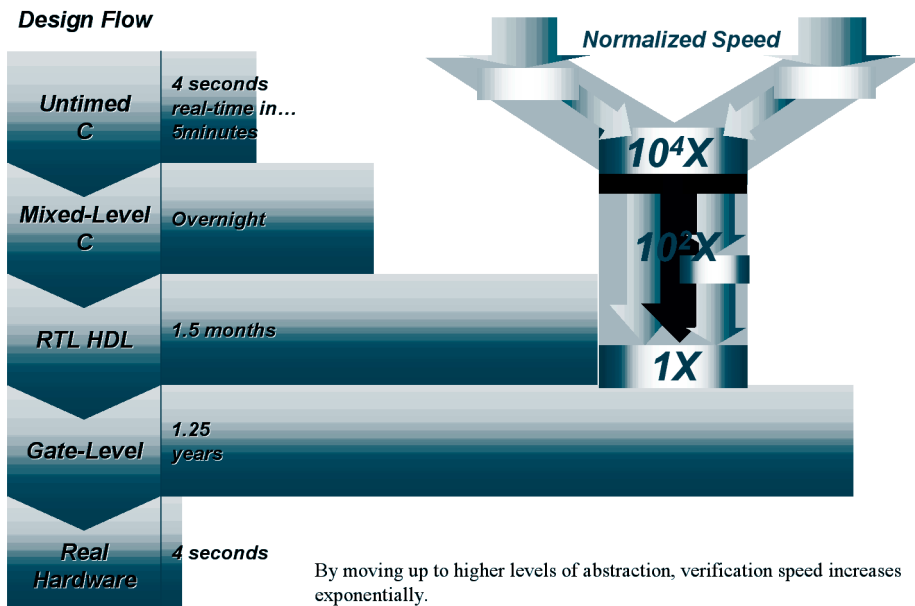


Figure 2. Closing the verification gap with C.

from third parties. They wanted to get a microprocessor core from one vendor, add a memory module from another vendor and maybe even a DSP from yet another vendor, and integrate that all into one chip. Many of these IP modules were "hard" cores - black boxes for design purposes that could not be touched. Designers needed to describe these IP blocks at a higher level of abstraction than RTL and started to use C/C++ for these descriptions and high-level modeling.

TOOLS EMERGE TO DEAL WITH COMPLEXITY

By the mid 1990s, the first tools started to come to market to help hardware designers deal with the complexity of these huge designs. At first co-verification was used between the hardware and software, but co-verification suffers from two major drawbacks. First, there is the question of what exactly the hardware and software is being verified against. Often there wasn't a good system specification for that verification task. Second, if verification was a bottleneck just for the hardware, add the software in and the problem only becomes worse.

In the late 1990s, tools emerged to handle hardware/software co-design at a systems level. A variety of proposals were put forth on different languages for system-level design. Because there was no existing "perfect" system-level design language, some people suggested developing a totally new language, which would take years. Other more pragmatic people suggested modifying C++ to include the constructs for hardware design.

By 1999, the market was becoming fragmented with a variety of proprietary C extensions for hardware design, one of the most successful variants of which was offered by CoWare. Because no standard existed, IP providers had to write several models for each piece of IP depending on the proprietary extensions used with

different tools.

JUMP STARTING A STANDARD

In early 1999, CoWare realized that it could take a long time for the market to establish a defacto standard, and the exact implementation didn't matter nearly as much as the speed of establishing one version as a standard. Rallying support from IP providers such as ARM, systems houses such as Sony, and semiconductor companies such as STMicroelectronics, CoWare joined forces with Synopsys to propose an industry consortium to establish one version of C++ extensions for hardware design.

The Open SystemC Initiative (OSCI) was announced in September 1999 with over 50 charter members - a strong showing from systems companies, semiconductor companies, IP providers, software companies and EDA companies. And now, a year and a half later, it looks like the various industries involved have done an excellent job of coming together to jump start a standard for C++ extensions for hardware and system design.

What is the SystemC language? SystemC is a modeling platform consisting of C++ class libraries and a simulation kernel for design at the system, behavioral and register-transfer levels. Designers create models using SystemC and standards ANSI C++. By using SystemC, EDA vendors can create tools that are automatically interoperable. System C is available for free via an open source license available at "<http://www.SystemC.org>".

Participation from the wide-ranging user community is encouraged. SystemC is fully compatible with a broad range of leading compilers for Solaris, Linux, and Windows NT and 2000 platforms, including the g++, egcs, Sun's SparcWorks C++ compilers and Visual C++ compilers. This lets designers use SystemC on a broad range of host computers.

A BRIEF HISTORY OF SYSTEMC

A preliminary version 0.9 of SystemC was released to the user community for free download from "http://www.SystemC.org" in late 1999. Version 0.9 was based on work by Synopsys, augmented by CoWare's experience with its 'RTC' language. SystemC version 1.0, released in April 2000, was the result of review and feedback from over 2200 users in over 500 different institutions, as well as close collaboration with industry bodies such as the Virtual Socket Interface Association (VSIA). SystemC version 1.0 includes the basic extensions for hardware design, including supports for modules, ports, new data types, more extensive runtime error reporting and support for fixed-point data types, thanks to contributions from Synopsys and Frontier Design.

SystemC Version 1.1 extended further into the system level by adding two key components that raise the abstraction level of SystemC well above existing hardware description languages. These two components, contributed by CoWare, consist of a method to describe abstract communications protocols and a computational model to simulate high-level behavior and communications known as Remote Procedure Call (RPC). Fully compatible with version 1.0, these components, which have become known as the Master-Slave Communication Library, enable highly efficient modeling and fast simulation of the communication between SystemC models. The method is especially efficient where the communication is to be implemented as a bus-based interface on an SoC.

In late January 2001, version 1.2 beta and a roadmap to SystemC 2.0 were announced at the SystemC user's forum in Japan. SystemC v1.2 builds on wide user experience (over 4500 individuals downloaded v1.1) to solidify the Master-Slave Communications Library. Users can now define their own protocols in addition to the standard bus protocols provided; in addition, there are many bug fixes. Version 1.2 also adds some new capabilities consistent with the SystemC

roadmap, adding a new model of time, extensions to event notification and dynamic sensitivity. We believe that SystemC 1.2 contains all the necessary extensions for modeling SoC-based IP.

Plans for SystemC v2.0 include extending the system-level foundation up to a yet more abstract level for modeling much larger systems. SystemC v2.0's flexible semantic foundation additions will support more models of computation within one environment, supporting virtually all system modeling needs, extending beyond SoC to general electronic systems. Version 2.0, the specification for which is available for review now, will be fully backward compatible with v1.2, when the reference implementation is delivered later this year.

USING SYSTEMC FOR SYSTEM-LEVEL SOC DESIGN

One of the biggest problems in system integration and verification is the lack, in most systems, of a good, executable system specification. The specification process for most systems is fraught with error and, therefore, problems are almost guaranteed down the road. Most systems are specified using an English-language description that may look good when filed into a thick notebook, but invariably does not adequately describe every element of the system, leaving a lot to the designer's imagination.

Because the specification is often misunderstood or too vague, most design projects include a lengthy hardware-software integration step at the end of the design process, which is mostly used to rewrite the software to work the way the hardware really was designed, once it is produced. This usually is the result of the inability to capture and validate specifications at the very beginning of the system concept and definition stage. Multiple restarts may even occur.

SystemC provides a common language to describe the system at the highest functional level, before decisions are made about what should be implemented in hardware and what should go in software. Then, as

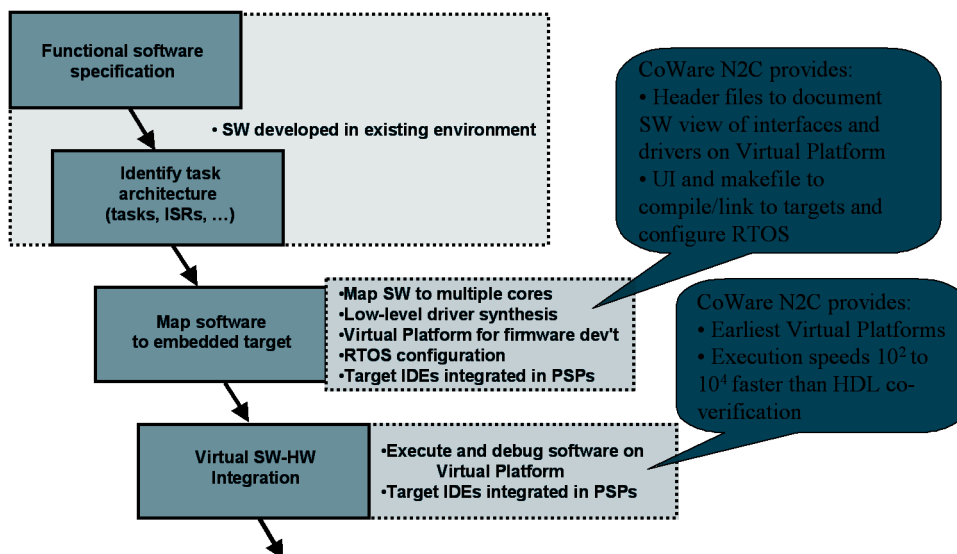


Figure 3. Software development for SoC.

architectural decisions are made, SystemC provides an ideal modeling language to check out the full functionality of the system specification and even develop a testbench.

SYSTEMC FOR HARDWARE DESIGNERS

What does SystemC mean for hardware designers? EDA companies currently are developing tools to speed system-level design in SystemC. IP providers are developing models using SystemC. The market is ready to take off, thanks to the hard work of OSCI.

Companies facing the most pain, such as Sony in consumer electronics and STMicroelectronics in fast-changing communications markets, are already moving up to C/C++ for system-level design, using the CoWare N2C hardware/software co-design tool to take the hardware all the way through to implementation in silicon. These early market adopters are leading the way, proving the tools and methodology that should become mainstream in the years ahead.

SYSTEMC FOR EMBEDDED SOFTWARE DEVELOPMENT

What does SystemC mean for embedded software developers? SystemC enables very early access to a functional virtual prototype of the hardware by beginning hardware design at a higher level of abstraction, in C++. Embedded software developers can take advantage of this to integrate software before a detailed Verilog or VHDL model is available for co-verification or an actual physical prototype is available.

Additionally, tools are coming to market that automate the time-consuming interface creation between the software and the hardware, particularly the processor, on the chip. CoWare N2C pioneered this capability, providing the ability to automatically synthesize the software drivers and hardware glue logic that are so time-consuming to design.

WORKING TOGETHER - HARDWARE AND SOFTWARE

By providing an industry standard language that can be used for system, hardware and software design, SystemC is enabling systems, hardware and software teams to "speak" the same language. This means much better communications between the groups. It also means that, as the software and hardware are developed, they can be simulated against the system specification so any design problems can be resolved much earlier in the design process.

In today's world of multi-national design efforts, one language for system-level design is imperative. If someday some group manages to develop the world's ideal system-level design language, then it can be considered for adoption. Meanwhile, everyone is trying to deal with the complexity of today's designs. SystemC will go a long way towards shortening the time it takes to get good system-level design tools to market. SystemC will also provide a common language for IP model exchange, which is essential as well.

One final note: while agreeing to one language is a major step, it is not enough. Expect to see a lot of tools announced and enhanced to provide a variety of system-level and hardware design capabilities using the SystemC language. Existing system design tools, such as CoWare N2C, are already evolving to include the capabilities in SystemC as those capabilities are released. As more such tools appear, we will start to reap further benefits of a common language as systems developers build design flows in which tools from multiple vendors fully inter-operate ■

Pete Hardee, director of product marketing, joined CoWare, Inc. in September 1998. Previously, he was a product line manager for system-level design tools at Synopsys. In his five years with Synopsys, he held various positions first in applications in the UK then in marketing in the US. Before that, Pete gained ten years' design and management experience in the UK electronics industry, first with Ferranti then Dunlop Aviation Division.

Pete holds a Bachelors degree in Electrical Engineering from Imperial College in London and an MBA from Warwick University.



Dedicated Systems Magazine

Look forward to Issue 1Q 2000
on Development
methodologies & tools



Dedicated Systems VirtualPressRoom™

We welcome news from your company
through press-releases that can be submitted
on our website at

<http://www.dedicated-systems.com/VPR>

Contact Nico Van Wijmeersch for more
information:

Phone : 32-2-520.55.77

Fax : 32-2-520.83.09

Email : info@dedicated-systems.com