

www.scdsource.com  
Wednesday, May, 5th 2010

## In My Opinion

# Is HLS finally converging on a standard language?

By Brett Cline

02/16/10

The arrival of a new decade has brought with it a change in the electronic system level (ESL) landscape. We've seen some new developments and interesting changes. Hardware developers who have decided to create their next design with a high-level synthesis (HLS) tool have yet another critical decision to make: which language to use. There are HLS tools based on ANSI-C, C++, SystemC and even M, the MATLAB math language from The MathWorks. Each language has its advantages and disadvantages – and there are even tools that claim to support more than one of these languages.



Each HLS tool claims similar productivity increases, quality-of-results improvements, and even significant reuse benefits, often leaving hardware developers scratching their heads in confusion. With all of these options out there, what are they to do? Choosing the wrong language or trying to mix languages could lead to some real headaches down the road. Some industry watchers have gone so far as to say there's a new language war brewing, like the one we saw years ago with Verilog and VHDL. But, is it really a language war? Or, is ESL finally moving toward a standard language?

There are many languages that adequately represent functionality, but which do not consider hardware implementation. Still others have been written specifically for hardware, but are non-standard and have no support beyond a single tool. And, there are others that enjoy widespread adoption in some system design spaces and are now trying to double up as hardware design languages. Choosing which language to use and when to use it comes down to answering a few questions.

1. Is the language a good starting point for your design?
2. Will the language allow you to completely describe and verify your hardware?
3. Is the language a standard supported by multiple vendors and users in the community or will it lock the design team into one vendor-specific flow?

And, the correct answers are? It depends on your situation.

Today's most popular ESL languages for HLS are ANSI-C and SystemC. What are their pros and cons? Well, one pro of ANSI-C is that it is indeed a standard and is widely used for algorithm and software development. One of the most attractive benefits of using it for HLS is that, often, the code already exists. Moreover, much existing code – especially that for industry-standard algorithms – has already been thoroughly verified and debugged through repeated use. The theory is that this software – generally written for a programmable microprocessor – can quickly and easily be turned into fixed-function hardware. So, it is the same starting point.

A primary limitation of ANSI-C is that it isn't a hardware design language, at all. It can't handle basic hardware concepts such as bit accuracy, concurrency, hierarchy and clock accuracy. This means that while it can represent a serial functional algorithm (datapath) it cannot be used to represent complex control. To overcome these issues, ANSI-C supporters often add extensions. The con is that these extensions are proprietary. Why? Because even where such extensions can "solve" the problem, ANSI-C cannot be extended in a standardized way without new compilers. The user is locked in to a single flow.

The other language, SystemC, isn't really a language, as such. It's a C++ class library, in effect, a standard extension of C++. C++ is inherently extensible, but non-standard extensions may – again – lock the HLS user into a single flow. That's why adding C++ to an ANSI-C flow doesn't solve the single-flow problem. But the standard SystemC extension *does* solve the single-flow problem. The SystemC classes give C++ the ability to represent in a standard manner what is needed for hardware design – bit accuracy, clock accuracy, hierarchy and concurrency, and so on. Consequently, designers can mix any amount of C++ with the SystemC extensions to accurately model hardware control and datapath.

It was the unsuitability or inadequacy of ANSI C and C++ as hardware design languages that moved the industry to devise and adopt SystemC. SystemC is both C++ and an IEEE 1666 standard, supported by dozens of vendors. SystemC is also the modeling language of choice in the development of virtual platforms for SoC design. As a result, SystemC-driven HLS enables an implementation path from the virtual platform to the RTL.

Designers can model virtually anything in SystemC. Whether it is the complex timed control or the algorithmic datapath that originally was in ANSI-C or a mixture of both, the design can be fully modeled and accurately simulated. At the end of the day, most of the code is essentially the same as the original C/C++ algorithm, but with the key attributes of a genuine hardware design language. SystemC classes are added to represent the hardware details in the algorithm – such as bit accuracy – as well as portions of the design requiring concurrency or enhanced accuracy. And while these additions may not be essential for the accurate synthesis of the model, they are critical for the accurate verification of both the model and the synthesized RTL.

SystemC is the one "language" that fits the bill for abstract hardware design, verification and, ultimately, high level synthesis. This IEEE standard allows the full specification of functional intent at a high level while simultaneously modeling the necessary hardware details – all in the same language. Because it is a standard, it fits into many different flows from a multitude of vendors. And recently, more vendors are attempting to make their ESL tools SystemC-compliant. Why? Because they have finally realized the limitations of their original language of choice – a choice often made years ago, before the introduction of SystemC.

What about the tools that attempt to handle multiple languages at the same time? Well, it sounds like design teams will need a robust, well-tested methodology that guides them in how to figure out which language to use in their specific design, and where. Otherwise, there may be serious confusion and design delay.

Ultimately, the goal is improved productivity and quality of results. This goal will drive the widespread adoption of HLS tools. Is there a language war brewing? Hardly. From where I sit, SystemC is emerging as the language of choice for high-level hardware and system design. So, is HLS finally converging on a standard language? Yes!

*Brett Cline is vice president of marketing and sales at Forte Design Systems. He has held positions at Summit Design, Cadence, and General Electric. Cline holds a B.S.E.E. from Northeastern University, Boston.*

## Further Reading

[Forte boosts Cynthesizer's automation capabilities](#)

All materials on this site Copyright © 2007-2009 Tech Source Media, Inc. All Rights Reserved | [Privacy Statement](#)