The Next IC Design Methodology Transition Is Long Overdue

Michael Meredith and Steve Svoboda, Open SystemC Initiative

Anyone with a tapeout deadline looming knows how finishing a new chip design keeps getting harder and harder. Fortunately, newly standardized high-level languages like SystemC™, along with modern high-level synthesis (HLS) tools are offering designers a way to escape the RTL "productivity jailhouse."

Since the turn of the century, the semiconductor industry has been suffering slowing engineering productivity growth. While during that period the semiconductor industry has seen major sales growth, as semiconductors continue working their way into almost every facet of modern life, with very few exceptions profitability has remained stagnant or even declined.

Moore's Law is often cited as the key driver and enabler of the semiconductor industry, and indeed the ability of the semiconductor manufacturing technologies to produce smaller transistors has very closely followed the smooth curve that it predicts. What is missing from this analysis is that in order for electronics companies to use these transistors to develop products, the ability of design engineers to create designs needs to increase at the same rate as the ability to manufacture transistors grows.

Two factors have contributed to this growth of design productivity. The first is reuse. Very early on, engineers learned to reuse standardized transistor-layouts ("gates"), which were then combined into "cells" and "macrocells" enabling standard-cell ASIC design. These cells/macrocells were later combined into "cores" and "subsystems" to create modern SoCs.

The second contributor to design productivity is the development of tools that allow the designer to work at a higher level of abstraction. At some point in the early 1990s the industry made a transition from mainly designing semiconductors using schematics at the gate level to mainly using logic synthesis at the register-transfer level (RTL).

From the mid-1970s to the late 1990s, these two factors combined to raise designer productivity from 4-5 gates/day up to 500-1000 gates/day. Figure 1 shows a slow increase in productivity that can be attributed to reuse and incremental tool improvements. This is followed by a jump in productivity at the point where the level of abstraction moved up to RTL. Since about 2005, however, IC designer productivity has leveled off. Current RTL-based design and verification methodologies have reached their limits.

Fortunately, new methodologies have started to emerge and be adopted by the industry. As of now, at least 24 (over 50%) of the world's leading semiconductor companies have begun adoption and deployment of high-level synthesis. HLS is the fastest-growing segment of EDA. Leading companies such as ST Micro, Sony, and NEC are already taping-out designs using HLS. The chairman of NEC said as early as 2002 that their internal HLS tool was used in all digital designs as their mainstream design flow. More recently, it was mentioned at a DAC panel last year that Sony had successfully completed more than 30 designs

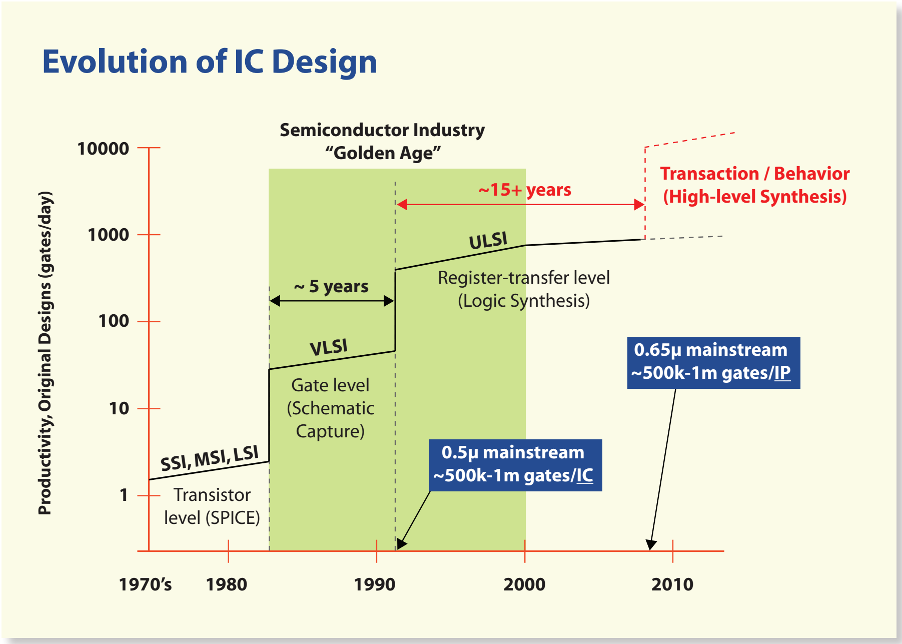


Figure 1. This chart shows the evolution of IC design from the mid-1970s to the present. The mid-1980s to 2000 was the semiconductor industry's period of fastest innovation, growth, and value-creation. This was driven in very large part by an over-100x increase in designer productivity.

involving over 200 engineers using HLS. ST Micro has been quoted by public sources as saying that 90% of all their new digital IP development is done by starting at a behavior level of abstraction and using high-level synthesis to create RTL.

Modeling at higher levels of abstraction (such as TLM) has been used for years by companies to accelerate simulation and verification of complex systems. Just as the development of RTL logic synthesis in the early 1990s led to a transition from using HDLs for verification only to broad adoption of HDLs in the RTL design flow, the development of high-level synthesis is moving companies toward TLM-driven design at a high level of abstraction.

Moving to a higher abstraction level offers opportunities for new kinds of reuse not possible at the register-transfer level. Microarchitecture decisions that bind an IP block to a particular application are handled automatically by high-level synthesis. Instead of having to specify what each part of the hardware is doing in every clock-cycle, the designer instead focuses on defining the functionality that meets the requirements and the range of latency and throughput constraints on that functionality. Instead of having to specify communication between blocks in terms of cycle-by-cycle signal behavior, designers can apply reuse to these interfaces, encapsulating the detailed protocol in terms of pre-defined "transactions." With a modern high-level synthesis tool, the designer can then automatically explore the area, speed, and power of a broad range of microarchitectures that meet those constraints, selecting the one that fits the intended application based on performance, power, and chip area considerations. The automatically-created RTL is used for logic synthesis and for the rest of the standard implementation flow. In this way, design teams can adopt and deploy high-level synthesis without disturbing their proven RTL-based procedures and tool flows.

Which Input Language Is Most Suitable for HLS?

Quite a few different languages have been proposed for use by design teams using HLS. The first HLS tools used RTL languages such as Verilog and VHDL. These languages were unsatisfactory for a number of reasons including their inadequacy as true high-level languages and the need for language translation from the C algorithm specification and the implementation language. Most recent approaches have tried to stay closer to the C algorithm by using a variation of C or a C++ library to add hardware capabilities to the C language. Synthesis variations on the C language include HardwareC, Handel-C, Cyber-C, Bach C, and Mentor's Algorithmic C. Each of these uses C extensions or C++ libraries to address hardware requirements.

The fundamental differences between C++ and SystemC-based high-level synthesis flows arise out of the nature of the two languages themselves:

- C++ is a sequential programming language with a single flow of control.

  SystemC is a hardware design language with explicit support for modeling multiple independent entities that coordinate their activities by communicating through channels.

- With C++ you execute a program.

  With SystemC you simulate hardware behavior.

- C++ supports the specification and validation of a single finite state machine.

  SystemC supports the specification and validation of multiple interacting FSMs.

In simple dataflow cases, such as where each part of an algorithm consumes a fixed number of values and produces a fixed number of values, it is feasible to write the overall system as a set of C++ subroutines and write a main program that calls the subroutines in a fixed order. The outputs of one subroutine are used as inputs to later-executing subroutines. This is often used as a kind of "simulation" of the interoperation of the subroutines. Other cases—such as those where the number of values produced and consumed by each entity is dependent on the data being processed, or where there are feedback loops in the dataflow—are more difficult to model and verify using only a sequential language like C++.

There are number of flaws in this approach. The greatest of these is that it is entirely non-standard and means that the IP product of the engineer's effort can only be used with a single vendor's toolset. Another great flaw is that the "simulation" that is performed by executing the C++ program does not in fact verify the correct concurrent execution of the implied modules. It also fails to verify the signal-level behavior that will be required in hardware to implement the data communication and synchronization protocols.

SystemC is the accepted industry standard for modeling concurrent hardware modules in C++. It provides a set of concurrent simulation semantics in a way that is supported by multiple vendors. Because the simulation semantics are standard and conform to the commonly accepted hardware design paradigm, it is possible to co-simulate these SystemC modules along with modules written in Verilog, VHDL, or SystemVerilog. Indeed, all of the major HDL simulator vendors provide a co-simulation capability for SystemC modules.

C++ Constructs	
C++ data types - Integer C++ data types - Operators on C++ integer data types	**User-defined data types** - structures - classes - inheritance - operator overloading
Control structures - if/else constructs - for() and while() loops - switch() constructs - references - statically-determined pointers	**Templates** - template classes - template functions

Figure 2. This table lists C++ constructs included in the draft synthesizable subset.

SystemC Constructs		
Modules & Threads	**Ports**	**Data Types**
SC_MODULE	sc_in<>	bool
SC_CTHREAD	sc_out<>	sc_int<> sc_uint<>
SC_METHOD	sc_export<>	sc_bigint<> sc_biguint<>

Figure 3. This table lists SystemC constructs included in the draft synthesizable subset.

Since 2005, the IEEE 1666™ standard for SystemC has provided standard syntax and semantics for representing hardware using a good high-level language that is appropriate for synthesis. Beyond this, industry agreement on a standard synthesizable subset is needed to facilitate portability of synthesizable hardware designs between SystemC synthesis tools. This work is currently underway in the Open SystemC Initiative's (OSCI) synthesis working group.

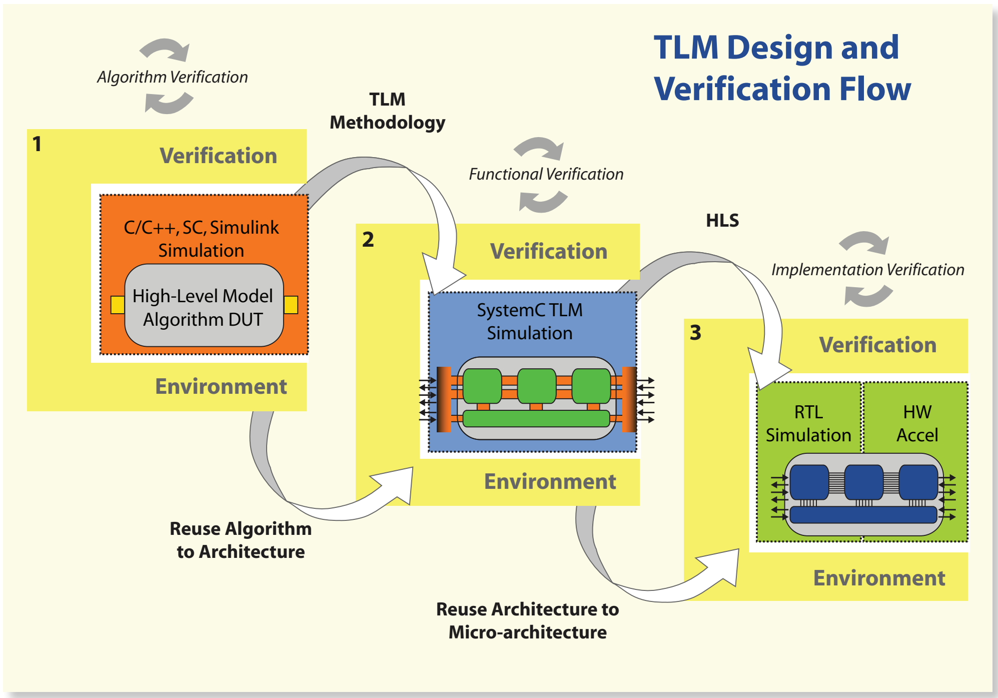
The OSCI SystemC Synthesis Draft Standard

The OSCI synthesis working group (SWG) is focused on defining a meaningful minimum synthesizable subset of SystemC to establish a baseline for transportability of SystemC design code between HLS tools while allowing for continued innovation by SystemC-based HLS tool developers. With the participation of more than one dozen EDA, semiconductor, and systems companies, this working group has released a draft document for public review that describes this subset.

The subset includes a broad range of C++ and SystemC constructs consistent with the HLS requirement that it must be possible to determine the structure of the hardware statically at the time the HLS tool processes the design source code. Consequently, a C++ construct (such as a template) is included in the subset, while dynamic resolution of virtual functions is not. Figures 2 and 3 summarize the included constructs.

Expanding from HLS Toward an Integrated Design and Verification Flow

Working at a higher level vs. lower level of abstraction can be compared to cutting wood with a chain-saw vs. with an axe. It can be a lot more productive, but can be also a lot more dangerous if you don't verify what you're doing at every step. New, advanced verification methodologies will also need to be developed to enable broad adoption of HLS. Leading EDA companies today are investing significantly to develop advanced tools and methodologies enabling comprehensive verification at abstraction levels above RTL. One key is to develop approaches for verifying high-level/TLM IP that optimally map different verification tasks to each level of abstraction in order to minimize the overall time and effort spent. The basic elements of this new approach are illustrated in Figure 4.



TLM Design and Verification Flow
Algorithm Verification
1
Verification
C/C++, SC, Simulink Simulation
High-Level Model Algorithm DUT
Environment
TLM Methodology
Functional Verification
2
Verification
SystemC TLM Simulation
Environment
HLS
Implementation Verification
3
Verification
RTL Simulation
HW Accel
Environment
Reuse Algorithm to Architecture
Reuse Architecture to Micro-architecture

Figure 4. TLM design and verification flow.

Design teams that have adopted these new methodologies see improvements in the quality of the circuits they can produce, the speed of their simulation and debug cycles, and in overall productivity, as shown in Figure 5.

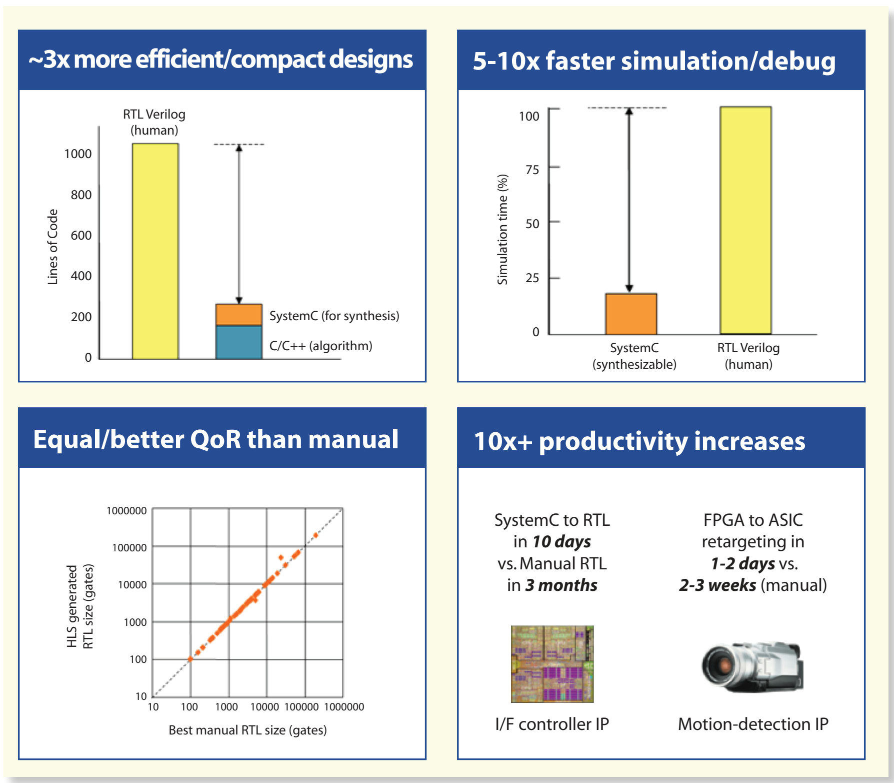


Figure 5. Benefits of new TLM design and verification methodology vs. traditional RTL (summary of actual data obtained in 2007-2008 by early adopters).

Conclusion

Just as design teams in the early 1990s adopted logic synthesis and RTL design to productively take advantage of the larger circuits made possible by new semiconductor manufacturing techniques, today's design teams are adopting high-level synthesis using the SystemC standard to take advantage of the number of transistors available at 90nm and below. Given that the RTL design abstraction has been in use for more than 15 years, it is no longer possible to consider it the leading-edge design approach that is required to bring us new, exciting consumer and industrial electronic products. Fortunately the move to the next level of abstraction using high-level synthesis in SystemC is well underway, and is demonstrating that it can deliver the required productivity.

Michael Meredith

Steve Svoboda

Michael Meredith is Vice President of Technical Marketing for Forte Design Systems and is President of OSCI. Steve Svoboda is Technical Marketing Director of System Design & Verification for Cadence Design Systems and serves as the Cadence representative to the OSCI Promotions Group.

SYSTEM C™

Open SystemC Initiative
Defining and Advancing SystemC Standards

www.SystemC.org