How to Avoid the Traps and Pitfalls of SOC Design

Chances are pretty good that your current SOC design approach is making your job much harder than it needs to be. Start out on an easier path with this white paper.

A Short History of IC, ASIC, and SOC Design

The history of IC design is full of massive tectonic shifts. They’ve been occurring every 12 years or so since semiconductors arrived on the scene half a century ago. SOCs are ICs, so they’re not immune to these shifts. It’s been a little over 12 years since we first started to put processors on ASICs—which created the SOC. So we are about due for a tectonic shift.

The shift is caused by the exponential increases in SOC design complexity. In 1995, SOC designs usually involved one microprocessor core and fewer than a million gates. RTL design techniques—which became popular in the late 1980s—worked pretty well at that level of design complexity. Today’s SOCs incorporate hundreds of millions of gates and multiple processor cores. As a result, design techniques that worked in 1995 have become cumbersome and clumsy, SOC design cycles are stretching out, and project schedules are slipping.

Similar situations have occurred before—at least three or four times. Early in the history of the IC, engineers designed chips and photomasks strictly by hand, using Xacto knives to cut polygons from rubylith. Then the industry graduated to early CAD tools that drew the polygons using a photoplotter. This single advance enabled the design of more complex chips. Then engineers started to drop entire transistors into their designs instead of drawing the individual polygons that comprised a transistor. This advance in design technique made even more complex IC designs possible. Soon engineers were designing with gates instead of transistors. By the late 1980s, the industry had left schematic design behind entirely and switched to RTL design and logic synthesis. Finally, IC designers started to adopt large blocks of logic. We call those blocks IP.
Each step up in design-abstraction level permitted the design of more complex ICs. Those advances allowed IC designers to keep pace with the IC-manufacturing advances that put more transistors on each new piece of silicon. In this way, IC design and manufacturing pretty much kept in step with each other for decades. Sometimes one would get a little ahead, usually manufacturing, but no serious problems arose because advances in design and manufacturing appeared in time to solve problems before seemingly insurmountable problems became truly insurmountable.

Endless Rise in Complexity and Performance Requirements

Today, as always, you must design chips almost as complex as theoretically possible to be competitive. If you don’t, someone else will and the extra complexity will provide a competitive edge. That statement has remained true as the definition of “really complex” has evolved from hundreds of thousands of gates in the 1980s to hundreds of millions of gates today. The application drivers have also evolved. In the 1960s, the driver was military hardware. In the 1970s, it was mainframe computers. In the 1980s, it was personal computers. In the 1990s, it was cellular telephone handsets. Today, the applications driving the industry are wireless communications, multimedia, and all manner of consumer electronics.

As a result, there’s an endless thirst for more computing horsepower to run the increasingly complex algorithms needed to handle digital media. And it’s not just processing that you need to worry about. You also need to worry about moving large amounts of data onto and off of the chip and within the chip and everyone needs to worry about power dissipation.

Conventional approaches to system design no longer work in this new, more complex world. Design teams have had to abandon general-purpose processors and DSPs for media processing because they’re just not energy efficient and they don’t deliver the required performance unless clocked at multi-GHz rates. Fixed-function hardware is more efficient but often lacks the flexibility needed to help SOC designs span more than one narrow product niche, which is necessary to get the sales volume required to justify the design effort. Without such flexibility, you can’t recover the development costs of these incredibly complex SOCs.

Old Bus-Based Architectures aren’t Efficient

In addition, our 40-year fixation on bus-based system architectures creates many on-chip communications bottlenecks that need not be there. Using shared on-chip buses creates multiple resource conflicts that can be avoided by using other types of on-chip communications techniques such as networks on chip (NoCs) and point-to-point communications. These days, when you see shared-bus architectures in massively complex SOC designs, you should immediately suspect that the design team may not have spent enough time on the early system architectural design and verification. Heavy use of global, on-chip buses needlessly confines your design’s performance.
Clock Rates Can’t Keep Rising

There’s a growing industry consensus that the era of the ever-higher clock rate has ended. For the first decade of SOC design (1995-2005), system designers relied on clock rate to solve all manner of system-level ills while relying on multitasking and multithreading to save the cost of additional on-chip processors. Need two processors? Just run the one you have at twice the clock rate. These IC designers were saving silicon costs over everything else (design time, verification time, risk, power dissipation, etc.) The cure ultimately turned out to be worse than the disease. Our industry now has a major hangover from using high clock rates and the name of that hangover is excessive power dissipation.

What Does Work?

Most agree that multiple on-chip processors running at lower clock rates is now the correct approach to SOC design and customized, application-specific processor cores are always more energy-efficient than general-purpose processors or DSPs. Some SOC designers also agree that global on-chip buses are just as bad as high-clock rate processors. Long on-chip buses exhibit high capacitance and excessive power. In addition, the more traffic you place on a bus, the faster it has to run. Heavily loaded buses also exhibit more overhead as multiple masters vie for control of the bus. Networks on chip and point-to-point interconnect are now more appropriate and make better use of the massive interconnect made possible with nanometer silicon.

Let’s look a little more at customized, application-specific processors. As already mentioned, they’re always more energy-efficient than general-purpose processor cores or DSP cores. That’s because they contain the task-specific datapath logic you’d design into RTL, but that specialized datapath logic is controlled by firmware instead of a hard-wired state machine. Firmware control gives you flexibility, which you can use to:

- Add features after the chip is manufactured
- Adapt to changes to standard (again after the chip is manufactured)
- Fix bugs (Yes! Even after the chip is manufactured)

Intelligent Interconnect Design

System-appropriate interconnect is a topic that doesn’t get as much discussion as it should. Simply put, designing appropriate interconnect prevents unnecessary on-chip communications bottlenecks and simplifies the overall system design. It prevents the interconnect from becoming a scarce on-chip resource, keeping on-chip data rates low. Lower speed on-chip communications make the chip more energy efficient because various on-chip blocks are not trying to drive long, highly capacitive on-chip buses for transactions between adjacent blocks, for example. By using intelligent interconnect design, you increase the overall on-chip bandwidth while cutting interconnect capacitance and clock rate. Both cut energy consumption.
On-Chip Memory Continues to Grow

Now consider what’s happening to on-chip memory. Figure 1 shows that the average SOC devotes more than 80% of the silicon to memory already and this percentage will continue to grow.¹ There’s an increasing amount of data to process and that takes more on-chip memory. You don’t want to put all of that memory in one block. Then the memory interface and the memory needlessly become critical bottlenecks, just like the shared on-chip buses.

![Figure 1: On-chip SOC memory usage: 1999-2014](Image)

Shared memories must support the data-bandwidth needs of all the connected memory users (usually processors or other RTL blocks) and that means the memories must run fast, at the required aggregate rate. Faster on-chip memories are harder to build and require “hotter” semiconductor fabrication processes that increase manufacturing costs. Also, faster memories dissipate more power and consume more energy. Finally, one big on-chip memory is likely to be inconveniently located for most of the memory-using blocks on the chip, which will further complicate system design. For all of these reasons, on-chip memory configuration deserves more thought.

Save Valuable Design Time

At this point, if you’re an experienced SOC designer steeped in the tradition of saving gates above all else—like clock frequency, design time, verification time, and schedule—you’re asking yourself, “What’s all of this going to cost?” If you take nothing else away from reading this white paper, understand this: silicon is not the most expensive item you need to conserve any longer. Moore’s Law has ensured this to be true and you’ll find the proof below. Today, time is your most precious resource and you need to save it like a miser. That’s time as in time to market (miss a market window and lose everything) and engineering time, which costs literally an arm and a leg no matter where in the world the design work is done.

So if you’re going to adopt a better SOC design methodology, what should it be like? It should allow you to design more efficient systems on silicon quickly, reliably, as close to the inherent limits of the silicon as you want, while accommodating the design and development costs and preventing schedule slips and missed market windows.

That’s a tall order. But it’s achievable! Today. Here’s what the International Technology Roadmap for Semiconductors had to say about the topic way back in 2001, when we already knew about this problem:

\[
\text{The cost of design is the greatest threat to continuation of the semiconductor roadmap.}
\]

Let’s briefly discuss three of today’s largest, most important application domains and the trends in these application domains: networking, multimedia, and wireless. Each of these domains shares some attributes with the others and each has its own special challenges.

Networking

The networking domain is heavily dominated by standards. Complicated standards. In addition, different networking applications have differing quality-of-service (QoS) requirements. Telephony requires constant bit rate service. Multimedia streaming needs higher bandwidth but variable bit-rate service. Data applications such as email and file transfers can get by with available-bit-rate service. The SOC you design for networking may need to support one or more of these requirements and the only way you’ll know if your design can successfully support these diverse requirements, short of building the chip, is to simulate candidate system architectures until you develop one that works under all foreseeable conditions.

In addition, the SOC requirements change radically depending on whether you’re designing a core network device or a terminal device. Core network devices focus on physical and link-layer protocols. The performance requirements are really demanding and there’s not too much flexibility because of the standards involved. Terminal network devices implement the higher, more complex protocol layers.
They operate at lower data rates. They’re more control oriented and therefore they require more implementation flexibility.

**Multimedia**

The multimedia application domain is something most of us are familiar with. Here you’re working with images, audio, and video. There are many standardized codecs for compressing and decompressing this digital media and there are new standards appearing all the time.

**Wireless**

The rapid rise in digital communications, cellular telephony, and wireless networking has revolutionized the wireless domain, which is actually more than 100 years old. Here, you see aggressive use of DSP. There are many wireless coding standards. New ones appear all the time. New device types such as wireless PDAs and mobile Internet devices (MIDs) appear all the time. This is a rapidly changing market.

In fact, all three of these domains change rapidly and therefore require rapid response from SOC design teams and a lot of flexibility in the resulting SOC designs.

There are common trends here. Performance requirements are increasing exponentially. Each new device generation must perform more processing than the last. New standards appear at an increasing rate. Energy efficiency has become one of the most important product features, if not the most important.

**SOC Design Trends – Divide and Conquer**

These application-domain trends in turn drive SOC design trends. SOCs get more complex with each new generation because of the twin drivers of market demand and Moore’s Law enablement. There’s only one way to deal with this rising design complexity. It’s a way developed by the Roman engineers, possibly the Egyptian engineers, thousands of years ago. It’s called divide and conquer. You divide the problem into functional blocks and then connect the blocks. You implement some blocks with a processor. Other will be implemented with RTL hardware. Blocks implemented with processors will be programmable, therefore flexible. Fixed-function RTL blocks are not nearly as programmable or flexible.

This design approach is the only way you can successfully design complex SOCs. And whatever design methodology you use will need to support this approach.
Using Multiple Processors in SOCs

Let’s briefly discuss multiple processor SOCs or MPSOCs. They’re a reality. Tensilica has clients that routinely put three to six processors on a chip. One of our clients, Cisco, has created two router chips that they use in a full range of network routers for TCP/IP packet processing. One of Cisco’s chips has 40 on-chip processors. The other has 192 processors. Clearly, MPSOCs are real and are here now.

However, there’s no widespread design methodology that rigorously takes you from a design spec to an optimized MPSOC architecture. Our industry still needs such a design methodology. Shortly, this white paper will discuss the state of the art in system-level design tools. But before we get to that, let’s set the stage by briefly discussing how the industry got to where it is and then how we understand the situation.

First, let’s look at the evolution of the SOC. See Figure 2 for the progression from ASICs to SOCs. ASICs appeared in the 1980s in the form of gate arrays, which were custom chips that vacuumed up all of the glue logic on a board-level design. Back then, there weren’t enough gates on an ASIC to implement a processor, so there were no SOCs. SOCs—by definition—are ASICs that incorporate one or more processors.

---

**Figure 2: Board-level system with ASIC to SOC evolution over three decades**
By the mid 1990s, IC manufacturing technology had advanced enough to allow designers to put a processor on a standard-cell ASIC along with memory and glue logic. This was the start of the SOC era. Back then, system-level design needed no significant advances because SOC-based systems essentially looked just like earlier, board-level, single-processor systems. Architecturally, there was little or no difference so system design didn’t really change as its focus moved from board to chip.

Thanks to Moore’s Law we’re now in the MPSOC era. SOC designers can easily put several processors on an SOC, creating an MPSOC. In fact, you can now put enough processors on a chip to require some sort of differentiation among the processor types. That means that one or two on-chip processors take the traditional CPU role on these chips while other processors take on specialized roles.

**Processors in the Dataplane**

Networking engineers coined the term “control plane” to describe the part of the system were CPUs do their thing. The other part of the chip, the part handling high-speed data, is called the dataplane. For networking engineers, the dataplane handles packet traffic. In other types of chips, the data might take the form of audio or video or some other form of high-speed, streaming data.

If you drill down into a dataplane, you’ll actually see a variety of processing categories, all taking place simultaneously as suggested by Figure 3. Some of these categories include audio and video encoding/decoding and pre- or post-processing, baseband DSP, and security protocol processing. There are of course many other processing types as well. You might find an RTL block or an actual processor performing the processing in an SOC dataplane. Again, RTL and customized processor cores are far more efficient at these types of processing than are general-purpose processor cores and DSPs.
When you’re designing multiprocessor systems, the first step is generally pretty easy. You can easily chunk tasks into control plane and dataplane categories. For example, a system that performs packet processing with Quality-of-Service requirements must perform several control-plane tasks such as applying policies, managing the network, and so on. Dataplane tasks include queuing, scheduling, routing, packet classification, and security. Similarly, an advanced set-top box’s control plane handles configuration management and the user interface. Its dataplane handles audio and video decoding and graphics.

Finally, a mobile internet device or MID has all of the control-plane functions of a desktop or notebook computer including the operating system’s user interface, database and file management, traditional applications, and games. However, these devices also have dataplane functions such as audio and video encoding/decoding, and wireless communications. These functions could be handled by a CPU running at multiple GHz but will be far more energy efficient if these tasks are handled by optimized dataplane processors.

As Table 1 illustrates, we know how to chunk applications into control-plane and dataplane categories. But that doesn’t mean you can easily convert these categories to efficient silicon implementations, especially with the rapid increase in available transistors made possible by Moore’s Law. The industry is now at a point where system designers have more Moore’s Law than they can handle.
Table 1: Control plane and dataplane tasks in SOCs

<table>
<thead>
<tr>
<th>Application</th>
<th>Control Plane Processing Tasks</th>
<th>Dataplane Processing Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Forwarding with QoS</td>
<td>Policy Application</td>
<td>Queuing</td>
</tr>
<tr>
<td></td>
<td>Network Management</td>
<td>Scheduling</td>
</tr>
<tr>
<td></td>
<td>Signaling</td>
<td>Routing</td>
</tr>
<tr>
<td></td>
<td>Topology Management</td>
<td>Classification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Encryption/Decryption</td>
</tr>
<tr>
<td>Advanced Set-Top Box (ASTB)</td>
<td>Configuration Management</td>
<td>Audio Decoding</td>
</tr>
<tr>
<td></td>
<td>User Interface</td>
<td>Video Decoding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2D and 3D Graphics</td>
</tr>
<tr>
<td>MID</td>
<td>User Interface</td>
<td>Audio Codecs</td>
</tr>
<tr>
<td></td>
<td>Database Management</td>
<td>Video Codecs</td>
</tr>
<tr>
<td></td>
<td>Office Applications</td>
<td>Wireless PHY and Link</td>
</tr>
<tr>
<td></td>
<td>Games</td>
<td>Layers</td>
</tr>
</tbody>
</table>

Dealing with Rising Complexity

Figure 4 shows the rise of ASIC complexity, measured in gate count, over the past two decades.\(^2\,\(^3\)\) Each lithographic advance gives us more gates to work with. In 1990, ASIC designers had about 200,000 gates per chip to work with. Today, the number of available gates easily surpasses three or four hundred million.

Figure 4: Design productivity gap: a growing gap between available and used gates on an SOC

---


Divergence of the two curves in Figure 4 clearly shows that our ability to create designs with all of these gates now severely lags the number of gates available. From before 1990 through 2002, the number of gates used was essentially the same as the number of gates available. However, since 2002, design complexities have lagged what could be achieved. Figure 4 demonstrates the failure of our SOC design methodologies to keep pace with Moore’s Law. Why is this bad? Because Moore’s Law is an efficiency curve. The closer you get to the Moore’s Law curve, the more competitive you are.

Figure 5 shows you when that break occurred. The wheels actually came off at the 90nm technology node. At that point, available gate capacity raced ahead of our ability to use those gates. At this point, the RTL design abstraction level proved inadequate to the magnitude of the system-design task. At this point, EDA thought leaders started talking about higher abstraction levels. They also started to discuss the idea of platform-based design.

Platform-Based Design

High-level or system-level abstraction combined with programmability creates a platform. You want to create platforms because it’s expensive to create a new chip. Rising complexity levels, as discussed above, means that each SOC needs to address multiple related applications for economies of scale. That’s the only way to effectively amortize the non-recurring engineering costs (NRE). Programmability buys you that needed flexibility to add features, to retarget the chip for other applications, to extend the use of the chip for one or two additional product generations, and to fix bugs.
Be realistic. These chips are complex. There will be bugs. How will you fix them? If your SOC has a high degree of programmability, chances are very good that you can fix a bug in firmware and save yourself and your company or client a chip respin.

The Four Phases of Platform Design

Figure 6 divides the design process into four phases. The first phase is the functional or architectural level. This level is the highest level of abstraction. Next, there’s the MPSOC platform-design phase. Essentially, this is where you implement the abstract architecture using a mix of different computational blocks (with processors or RTL), and then interconnect these blocks. A level down, you create the high-level IP needed for the platform design. At the lowest abstraction level, you develop the semiconductor implementation technology and basic IP such as library cells.

![Figure 6: The four phases of platform design](image)

At the functional or architectural level, you design a system divorced from implementation using the best algorithms you can find or develop. Here is where you develop the control-plane software, probably in C or C++. It’s also where you develop the dataplane algorithms. Also at this level, you tie all of the software together and see if your system design works.

The platform-design phase is implementation dependent. At this point, you’ve picked an implementation technology. For example, you pick the 90 or 65nm technology node or you pick a structured ASIC depending on project constraints and goals. You then map your abstract system onto the implementation technology and create any required hardware-dependent software.

Once you’ve defined the platform at the block level, it’s time to develop the blocks. These blocks include processing elements, on-chip interconnect, any domain-specific I/O (it seems like a new one of these domain-specific I/O standards pops up each month or so), and any stand-alone RTL blocks.

The lowest design level is the development or selection of the process node or IC vendor, and the basic blocks such as standard library cells, I/O cells, and memories.

With that background, let’s again consider the functional-design phase. This phase is really where you get the most design leverage in terms of performance and energy efficiency. This is currently the high-interest design zone—that higher level of abstraction.
ESL – Electronic System Level Design

As mentioned above, early SOC development was an RTL-based design and integration process. The design was not modeled until all of it was available at least at RTL level. Of course, you might also be integrating pre-designed IP blocks only available at gate level. Once the design was complete at the RTL level, you could verify it by simulation but system verification using RTL simulation was extremely slow and could only be performed extremely late in the design process. The speed of RTL verification was often only a few cycles or perhaps tens or hundreds of cycles per second—if you were lucky—so it was completely inadequate for the verification of any software in the system. Running the system software on an RTL model of the processor, let alone the entire system, was far too slow to be useful. In addition, simulation speed got slower and slower as ASIC size grew from hundreds of thousands to hundreds of millions of gates.

This RTL-centric design methodology allowed for very little design-space exploration. The effort of designing or integrating particular blocks into the design meant that you had no time to try different system architectures for achieving product goals. Because alternative hardware function blocks or alternative processors had different interfaces, the job of ripping up and re-stitching early block choices took more time and effort than was available. As a result, this RTL verification could complement gate-level simulation, but would only be useful for systems with relatively low complexity levels.

Figure 7 shows a simple SOC with a processor, memory, RTL blocks, I/O blocks, and a bus for on-chip communications. The illustrated system employs classic microprocessor-based architecture, which engineers have used for the last 40 years. But how do we know that an SOC based on this architecture will work in its intended application? System-level modeling, not ad-hoc build and test, is the answer. In the days of board-level design, the build-and-test approach was inexpensive and the system-level modeling tools didn’t exist. For chip-level design and its associated development costs, it is essential to verify the system before all the detailed implementation.

Figure 7: A simple processor-based SOC design
In addition, SOCs are getting more and more complex so the problem’s only getting worse. Multiple processors, many dedicated accelerators, and an increasing number of peripheral interfaces mean that the design and verification problem is growing exponentially. Fully implementing and then verifying an SOC is like buying a car without first taking it for a test drive. Picking one system architecture and set of functional blocks without exploring alternatives is like shopping for the car by deciding to buy the first one that drives past you in the street. Better design approaches are available.

Early ESL Approaches

From the mid 1990s onwards, EDA vendors began to create many proprietary, ad-hoc methods to allow higher levels of architectural design and verification. For example, Mentor Seamless provided a library of processor instruction-set-simulator (ISS) models written in C or C++, together with transactors that allowed these models to talk to simple memory models and to HDL hardware models using transaction-level modeling. In the late 1990s, early Electronic System Level (ESL) design tools such as CoWare’s N2C, Cadence’s Alta VCC, and Synopsys’ Cocentric System Studio offered proprietary ways to create and connect models in C or C++ into a system model, using transaction-level concepts. However, these tools lacked interoperability due to their proprietary modeling abstractions.

Sometimes IP providers provided modeling environments for their own IP that could also be used to model other third party IP blocks. For example, Tensilica’s XTMP is C-based modeling environment that allows cycle-accurate models of preconfigured and customized Xtensa processors to be linked to models of other system-level components using a transaction-level modeling API. Again, creating a model of a complete system using such an approach usually requires a large amount of ad-hoc modeling. Nevertheless, such approaches can deliver performance on the order of a few tens or hundreds of thousands of simulated system cycles per second, much faster than RTL modeling. This simulation speed often justifies the modeling effort when compared with RTL simulation speeds of a few to a few tens or hundreds of cycles per second.

Many proprietary approaches to system-level modeling using a variety of languages such as C, C++, Java were tried and discarded during the mid to late 1990s. At the same time, there was an active debate about what kinds of system-level modeling and languages were important. Fans of discrete event, dataflow, and synchronous languages such as Esterel were all involved in the debate.

Proprietary ESL Approaches Didn’t Work

The main issues with all of these proprietary or research approaches to ESL design were that they were too proprietary, too specialized, or both. No commercial EDA tool vendor wanted to invest the money needed to build all of the required models if they only worked in one vendor’s tool environment. Experience with many HDLs and the eventual standardization of VHDL and Verilog taught EDA tool users the
perils of proprietary design languages and the virtues of interoperable tools based on standard languages. Specialized languages such as dataflow or synchronous ones only solved part of the design and verification problem. Designers desired an ESL design approach that allowed multiple modeling styles to be integrated.

The Open SystemC Initiative, OSCI, was born out of this desire but the organization made several initial missteps. Its original legal infrastructure required two levels of revision to get arch rivals Cadence and Mentor Graphics to join. Its original system-level modeling approach, based on CoWare’s Master-Slave library using Remote Procedure Calls, lacked the needed flexibility to integrate different modeling styles. OSCI’s original focus on HDL replacement confused a lot of people and diverted attention from system-level design needs. Time resolved most of these issues. In particular, Professor Dan Gajski’s team at UC Irvine worked on SpecC, which added channel and interface modeling techniques that improved SystemC’s system-level modeling abilities. Finally, the industry was well on the way to standardization.

**System Modeling Using SystemC**

Standardized SystemC permits true system-level modeling. Additional modeling styles are supported either by conventions or by modifying the SystemC kernel, which is available in source form. This openness allowed for a lot of early experimentation and system-level, design-space exploration. Various groups started to experiment with different modeling approaches. The availability of the SystemC modeling libraries, and simulation kernel, in source form, made experimentation easy. There were many thousands of downloads, a lot of university and research interest, and quite a bit of commercial experimentation. Over time, academic and commercial users moved from experiment to early adoption.

Some new companies interested in high level synthesis recognized early on that SystemC offered a superior input mechanism for synthesis than the failed “behavioural HDLs” offered previously. Early ESL tool vendors with proprietary languages later adopted SystemC input subsets. Other companies such as Mentor for its Catapult C tool used their own versions of C for input but generated SystemC output models for building system simulations.

Once SystemC began to be more widely accepted, a number of proprietary tools began to incorporate it or be rebuilt on top of it. Out of its N2C tool, CoWare created Platform Creator (now called Platform Architect). Axys rebuilt its MaxSim simulation kernel on top of SystemC, creating the SoC Designer tool. Axys was later bought by ARM, and the tool became ARM Realview ESL SoC Designer. In the summer of 2008, ARM moved the tool and team to Carbon Design Systems, which now supports SoC Designer.

Other tools kept proprietary approaches but added interfaces to SystemC to allow interoperability. SystemC has now become a well-supported standard. The three major EDA companies, Cadence (with NCSim/Incisive), Mentor Graphics (with ModelSim/Questa) and Synopsys (with VCS) all support mixed SystemC—HDL verification.
Finally, IP providers, such as Tensilica, began creating SystemC-based modeling environments providing transaction-level access to instruction-set simulators such as the Tensilica XTSC modeling tools that came out towards the end of 2006. There are two major abstraction levels used in such tools. The first is cycle-accurate, transaction-level modeling. This form of modeling resembles many of the earlier ad-hoc methods and can be used for system models that run at several hundred thousand cycles per second. The second abstraction level is instruction-accurate, fast-functional modeling. System models using this approach can run 10-1000X faster than simulations run at the cycle-accurate level; that is, many millions to hundreds of millions of system cycles per second.

Creating Virtual Platforms for Entire SOCs

When you combine fast-functional models with fast simulation methods such as just-in-time compiled code simulation, you can create virtual platform or virtual prototype models of entire systems or SOCs. These system-level models and simulations are especially useful for software development and verification, where the level of software testing does not depend on exact hardware timing. These SystemC simulation abilities promote a methodology where cycle-accurate system models permit rapid, early design-space exploration for functional blocks and the development of hardware-dependent software layers. Later, most of the system-level software can be designed and developed without the need for cycle counting by using virtual platforms.

Figure 8 is a screen shot of the CoWare Platform Architect tool. It shows a system design with a Tensilica processor, buses and peripherals. The design was captured using the tool’s graphical user interface. The screen shot also shows several analysis displays. The associated simulation uses SystemC, as optimized by CoWare. CoWare’s Platform Architect allows integration of processor ISS models and transaction-level models of buses, memory components and other peripherals and accelerators. The resulting system model can be run in cycle-accurate mode for detailed performance analysis and design space exploration. In addition, it can be run as a fast-functional virtual platform at a significantly greater speed, to verify software.
Detailed performance analysis uses the cycle-accurate model. The analysis screen in the lower right of Figure 8 shows displays of memory access patterns, data throughput, correlation of function call traces against time, transaction counts, and finally transaction request-response patterns. This feedback can be used to optimize memory sizes and partitioning. It can also be used to decide on levels of bus hierarchy and the placement of accelerators and peripherals, to minimize bus contention. Correlating bus activity with software functions can be used to optimize software, to minimize contention, and to improve system throughput.

Figure 9 shows the Axys/ARM/Carbon Design SoC designer tool. The Figure shows a system with a Tensilica processor integrated with an ARM AMBA bus and other components. Also shown are output views of registers and other system level probes, which can be used to track the system state while debugging.
The CoWare and ARM/Carbon tools shown in Figures 8 and 9 require ad-hoc tool-specific integration of the processor model. The information being passed to and from the processor is usually very similar, but the exact structures and APIs differ depending on the tool. This is today’s state of the art for system modeling tools. The need to create different integrations for each system-modeling tool wastes a lot of engineering time and effort for the IP creators, whether in large design companies or third party IP providers.

When SystemC was first standardized, it lacked interoperable transaction-level modeling. A primitive first effort called TLM 1.0 appeared, but this specification was not adequate for modeling real systems. In addition, other important system modeling needs caused additional interoperability problems to surface involving connections to debuggers, logging and tracing, analyzing transactions, and the creation of virtual platform models. Finally, many people’s notion of transactions has been limited to memory-mapped buses and real systems may have much richer interconnect options that transaction-level models must support.

As illustrated with the CoWare and ARM/Carbon tools, IP developers have had to repeat their integration work with each supported tool—a waste of engineering effort. In addition, the extra integration work required is a barrier to rapid development in the tool and model markets. Designers who want to work at this level can be frustrated by a lack of models or a delay in model availability. A useful standard for TLM models was clearly needed and this need was substantially answered with TLM 2.0.
Enter TLM 2.0 for Easier Modeling

OSCI developed TLM 2.0 over a three-year period, with the final release in June of 2008. TLM 2.0 makes progress on some of the standardization issues listed above including support for virtual platforms using temporal decoupling, which permits models to be simulated out of order. Direct-memory interface methods in TLM 2.0 allow virtual platform models to bypass simulations of complex interconnect networks. The generic transaction definition and extension mechanism, if used well, can enhance model interoperability.

TLM 2.0 defines standard simulation phases to support two modeling styles. Although this mechanism could have been used to define a standardized cycle-accurate style for interoperability, TLM 2.0 did not take this opportunity, which limits its usefulness. In addition, although the TLM 2.0 working group worked on debug APIs and analysis ports, there is still much more to be done to support good logging, tracing, analysis, and debugger connections. Further, TLM 2.0 only provides memory-mapped transaction types. OSCI’s TLM working group continues and there may be a TLM 3.0 in the future.

Even with TLM 2.0’s limitations, it is quite possible to quickly build effective and fast system-level models of processor centric systems. These system-level models use available IP models and commercial tools, employing SystemC as the infrastructure for connecting the models and for modeling hardware blocks. We now have very credible system-level simulation methods that allow early design-space exploration and system and software verification. This approach is far faster than RTL or gate-level simulation.

Three approaches make it easier to obtain interoperable system models of new or legacy hardware blocks:

1. ESL synthesis tools can use SystemC as input, output or both.
2. Tools such as Carbon Design’s Model Studio can generate interoperable models of RTL hardware blocks.
3. Finally, co-simulation methods with SystemC in HDL simulation tools means that even legacy RTL code can be simulated together with SystemC test benches, ISSs, and models.

If the world unfolds the way it should, then further TLM standardization will lead to a virtuous circle: a self-reinforcing, positive-feedback loop. Models will be easier to integrate into tools, which will lead to the creation of more IP models, and these models will be usable more quickly. Thus there will be an increasing supply of tools and models to satisfy a growing designer demand. System-level modeling of complex SOCs will become the norm, not the exception. Even if the world unfolds a little more slowly, existing capabilities are quite able to support system-level modeling, albeit with more effort than one might wish. So in either case, the gradual adoption of these techniques should continue.
An Open Invitation

To start your design team on that different path to successful tape-out, contact Tensilica for assistance. For more information on the unique abilities and features of the customizable family of Xtensa processor cores and associated development tools, see www.tensilica.com, send an email to sales@tensilica.com, or contact Tensilica directly at:

US Sales Offices:

Santa Clara, CA office:
3255-6 Scott Blvd.
Santa Clara, CA 95054
Tel: 408-986-8000
Fax: 408-986-8919

San Diego, CA office:
1902 Wright Place, Suite 200
Carlsbad, CA 92008
Tel: 760-918-5654
Fax: 760-918-5505

Boston, MA office:
25 Mall Road, Suite 300
Burlington, MA 01803
Tel: 781-238-6702 x8352
Fax: 781-820-7128

International Sales Offices:

Yokohama office (Japan):
Xte Shin-Yokohama Building 2F
3-12-4, Shin-Yokohama, Kohoku-ku,
Yokohama
222-0033, Japan
Tel: 045-477-3373 (+81-45-477-3373)
Fax: 045-477-3375 (+81-45-477-3375)

UK office (Europe HQ):
Asmec Centre
Eagle House
The Ring
Bracknell
Berkshire
RG12 1HB
Tel: +44 1344 38 20 41
Fax: +44 1344 30 31 92

Israel:
Amos Technologies
Moshe Stein
moshe@amost.co.il

Beijing office (China HQ):
Room 1109, B Building, Bo Tai Guo Ji,
122th Building of Nan Hu Dong Yuan, Wang Jing,
Chao Yang District, Beijing, PRC
Postcode: 100102
Tel: (86)-10-84714323
Fax: (86)-10-84724103

Tainan office:
7F-6, No. 16, JiHe Road, ShihLin Dist,
Taipei 111, Taiwan ROC
Tel: 886-2-2772-2269
Fax: 886-2-66104328

Seoul, Korea office:
27th FL., Korea World Trade Center,
159-1, Samsung-dong, Kangnam-gu,
Seoul 135-729, Korea
Tel: 82-2-6007-2745
Fax: 82-2-6007-2746