

Guest Editors' Introduction: The True State of the Art of ESL Design

Sandeep K. Shukla

Virginia Polytechnic and State University

Gary Smith

Gartner Dataquest

Carl Pixley

Synopsys

■ **ESL, OR ELECTRONIC SYSTEM LEVEL**, is a new buzzword in the EDA industry. It has found its way into the mainstream EDA vocabulary in the past few years because of increased interest in finding new ways to raise the abstraction level for the design entry point during the electronic-systems design process.

In hardware design over the past three decades, the design entry point has moved upward in the abstraction hierarchy—from hand-drawn schematics to gate-level design, to RTL descriptions. As hardware design complexity has become increasingly unmanageable, finding ways to design hardware ICs at higher abstraction levels and developing tools to automatically create the circuits' actual layouts has gained more importance in industry and academia. This upward trend in abstraction has enabled engineers to exploit the scientific and engineering advances that have tracked Moore's law quite closely.

Since the late 1980s, the design entry point's abstraction level had remained almost stagnant at the structural RTL. Behavioral synthesis had remained mostly elusive, with some domain-specific success areas, such as DSP chips. But by the late 1990s, recognition of the so-called "productivity gap problem" led to various attempts at abstraction enhancement. These attempts gave rise to various languages for system-level design, such as SpecC, SystemC, and variants of these. Tools and methodologies for using these languages for design entry have emerged in the market and in academic circles.

In the meantime, integrated systems such as cell phones, network routers, consumer electronics, and personal electronic devices like PDAs started to dominate

the electronics landscape. In contrast to earlier computing devices such as microcontrollers and general-purpose microprocessors (GPPs), these systems had one thing in common: you could distribute their functionality into hardware or software with sufficient fluidity based on various trade-offs in performance, power, cost, and so on. This development broke the hardware abstraction for software development hitherto used in traditional computing platforms, such as GPPs, as illustrated by the Windows and Intel platforms in desktop computing.

Another phenomenon that had occurred for decades in avionics, automotive, and industrial-control systems also gained increased attention among EDA researchers. The design of such systems' embedded software was typically at a much higher abstraction level, using synchronous languages, Argos-like visual environments, and so on to describe required control trajectories. Control-systems engineers had also used Matlab and similar mathematical and visual enhancements of such tools for decades to design, validate, and even synthesize their software. In the meantime, increasingly more computing devices were mixtures of software and hardware, and there was increased flexibility for deciding the hardware-software partitioning. Consequently, architectural exploration at the functional and architectural level became increasingly critical for finding the right trade-off points in the design. It's best to perform such explorations before the designers commit to the RTL hardware logic design, or before the embedded-software writers commit to the embedded-software code.

These evolutionary trajectories of electronic-system design led to the introduction of ESL design. According

to the popular “ESL Now!” Web site (<http://www.esl-now.com>), ESL design concerns the following:

- “the development of product architectures and specifications, including the incorporation and configuration of IP,”
- “the mapping of applications to a product specification, including hardware/software partitioning and processor optimization,”
- “the creation of pre-silicon, virtual hardware platforms for software development,”
- “the determination/automation of a hardware implementation for that architecture,” and
- “the development of reference models for verifying the hardware.”

In this special issue, we explore recent developments in ESL languages, tools, techniques, and methodologies related to improving productivity or enhancing design quality. We wanted this issue to answer the following key questions regarding ESL design:

- What is ESL design, and what are the current languages that support ESL features?
- What tool chains and design flows are appropriate for ESL-based design and validation?
- What new validation techniques and methodologies are available if ESL abstractions are used in a design flow? Are there any test technology benefits?
- Are there major industrial projects today that have been successful due to ESL usage?
- What are the market indicators and forces that might make or break ESL design?

Although the articles in this special issue don’t necessarily answer all these questions, they address some key issues and are quite thought-provoking. In the first article, Patrick Schaumont and Ingrid Verbauwhede focus on two properties they see as key to ESL design: abstraction and reuse. They present an ESL design flow using the Gezel language, and they show with several very different design examples how Gezel supports their case for reuse and abstraction.

The second article, by Ivan Radojevic, Zoran Salcic, and Partha Roop, considers the need for directly expressing heterogeneous, hierarchical behaviors for modeling specific embedded systems. The authors examined two existing ESL languages: SystemC and Esterel. Their analysis led them to create a new computation model as well as a graphical language to gain the direct expressivity

they need for their model. Although there have been various attempts at changing SystemC and Esterel to fit modeling requirements, these authors mainly consider standard SystemC and Esterel here.

In the next article, Douglas Densmore, Roberto Passerone, and Alberto Sangiovanni-Vincentelli attempt to stem the seemingly ever-increasing tide of confusion that permeates the ESL world. Not only are software developers and hardware designers having difficulty finding a common language—verbally, as well as design-wise—but communication failures are common within those communities as well. Traditionally, there are three rules of design: First, there is a methodology, then there is a design flow, and last there are the tools necessary to fill that flow. But, as this article points out, we seem to have approached ESL backward. We have built tools, but we have no flow. And, it goes without saying, we have no methodology. No wonder then that the predictions of ESL taking off in the next four years seem to be overly optimistic. Still, the customer demand is there. But these customers have had to fill the need with internally developed ESL tools. The University of California, Berkeley, has long been the champion of platform-based design, and these authors base their taxonomy on a combination of UC Berkeley’s platform work and Dan Gajski’s Y-chart work (at UC Irvine). Hopefully, this taxonomy will help stem the tide of confusion and enable the design community to turn around its ESL efforts.

Finally, the article by Stephen Edwards presents one side of an ongoing debate on the appropriateness of C-like languages as hardware description languages. In the ESL landscape, it is often assumed that a high-level programming language can substitute for a higher-abstraction-level hardware description language. This article attempts to deconstruct such a myth about the C programming language by extensively documenting the shortcomings of such an approach and by identifying the features that an ESL language should have. A brief alternative opinion by John Sanguinetti immediately follows this article.

ESL DESIGN, METHODOLOGIES, LANGUAGES, AND TOOLS are still not clearly identified and taxonomized, and the articles in this special issue attempt to reduce some of the confusion regarding the term *ESL*. However, we believe that we are still in the early stages of ESL-based design. Many more discussions, expository articles, and debates must take place before it can find its permanent design entry point in industry.

The articles in this special issue could not cover everything. Although many of the synthesis technologies mentioned address algorithm design—for instance, for DSP—technologies to synthesize high-level control logic are necessary for ESL design to address the breadth of circuits designed by hardware engineers. In the recent past, researchers insufficiently addressed behavioral synthesis, but this segment is now showing increased activity. Bluespec (<http://www.bluespec.com>), for example, offers new technology to raise the abstraction level for complex control logic and to synthesize RTL design from these descriptions. Other behavioral-synthesis solutions are coming to the market as transaction-level models.

We hope you find the articles in this special issue interesting. We encourage you to send us critiques, comments, or questions about this special issue. Letters to the editor for publication in future issues are also encouraged. Finally, we thank the authors, the reviewers, and the editorial staff at *IEEE Design & Test* for their help in making this issue possible. ■



Sandeep K. Shukla is an assistant professor of computer engineering at Virginia Tech. He is also founder and deputy director of the Center for Embedded Systems for Critical Applications (CESCA), and he directs the Fermat (Formal Engineering Research with Models, Abstractions, and Transformations) research lab. His research interests include design automation for embedded-systems design, especially system-level design languages, formal methods, formal specification languages, probabilistic modeling and model checking, dynamic power management, application of stochastic models and model analysis tools for defect-tolerant system design, and reliability measurement of defect-tolerant systems. Shukla has a PhD in computer science from the State University of New York (SUNY) at Albany. He has been elected as a College of Engineering Faculty Fellow at Virginia Tech, and he is on the editorial board of *IEEE Design & Test*.



Carl Pixley is group director at Synopsys. His pioneering achievements include model checking based on binary decision diagrams (BDDs), Boolean equivalence, alignability equivalence, constraint-based verification, and C-to-RTL verification. Pixley has a PhD in mathematics from SUNY at Binghamton. He is a member of the IEEE and

the Mathematics Association of America, and is verification editor for *IEEE Design & Test*.



Gary Smith is a chief analyst at Gartner Dataquest, where he is part of the Design & Engineering Group and serves in the Electronic Design Automation Worldwide program. His research interests include design methodologies, ASICs, and IC design. Smith has a BS in engineering from the United States Naval Academy in Annapolis, Maryland. He is a member of the Design Technology Working Group for the *International Technology Roadmap for Semiconductors (ITRS)*.

■ Direct questions or comments about this special issue to Sandeep K. Shukla, Department of Electrical and Computer Engineering, Virginia Polytechnic and State University, Blacksburg, VA 24061, shukla@vt.edu; Carl Pixley, Synopsys, 2025 NW Cornelius Pass Rd., Hillsboro, OR 97124, cpixley@synopsys.com; or Gary Smith, Gartner Dataquest, 281 River Oaks Pkwy, San Jose, CA 95134, gary.smith@gartner.com.

For further information on this or any other computing topic, visit our Digital Library at <http://www.computer.org/publications/dlib>.

**The IEEE
Computer
Society**

**publishes over
150 conference
publications a year.**

**For a preview
of the latest papers
in your field, visit**

www.computer.org/publications/