

AFTER YEARS OF OVERCLAIMING AND UNDERPERFORMING, ESL DESIGN HAS A ROLE IN

ELECTRONIC-SYSTEM-LEVEL DESIGN:  
IS THERE FIRE  
BENEATH THE  
SMOKE?



It was a shimmering promise on the horizon: As SOCs (systems on chips) became more complex, we would simply move from RTL (register-transfer level) to the next-higher level of abstraction—what some experts called ESL (electronic-system-level) design. We would express the behavior of the system in a high-level language, such as C++. We would model and explore the system at that level, partition it into hardware and software components, and then push a button. Scripts and ESL-synthesis tools would digest our ESL design and give us back a nearly optimal RTL design or even a netlist together with the necessary software. Design productivity would once again be ahead of complexity.

But that was, as they say, then. Today, many failures, false claims, and too-limited tools later, many designers put ESL in the same category as gallium arsenide: a permanent technology of the future. Yet, this dismissal is as inaccurate as some of those early claims of mission accomplished. ESL tools today play key roles in many teams' design flows. Those roles differ in different teams and application areas. But the results are sufficiently important that a design manager today dismisses ESL at his own peril.

### CONFUSION AND VARIETY

"In reality, most of what we call power users have been using ESL tools since about 2004," says Gary Smith, noted industry analyst and founder of Gary Smith EDA. "But many of those tools have been internally created, and there has been a lot of confusion about just what is an ESL tool."

The confusion is understandable. There are several levels of abstraction hiding under the title "system-level." And there are many kinds of tools, with different expectations, at each of those levels.

"The initial idea was that ESL was about architectural design," Smith says. This process is far simpler than pushbutton behavioral synthesis. It simply means a textual, executable representation of a system architecture that would allow experimentation at a very high level—sort

of a language-based version of the time-honored architectural-design tools, the white board and the spreadsheet.

"But, in 2004, the tools were consumer-application-focused and really were for algorithm design, not architectural design," Smith continues. "There wasn't any focus on the level of abstraction that included blocks like processors and memories."

Even within the algorithm area, there were—and remain—strong differences. Some designers see an algebra-based tool such as the MathWorks' Matlab as the natural way to describe and manipulate algorithms. Others believe C or C++ should play that role, even though these languages carry a strong syntactic bias toward a particular legacy implementation—the Digital Equipment PDP-11 minicomputer. Still others believe that algorithm development should occur in a purpose-built language for describing algorithms—neither purely algebraic nor purely procedural and sequential. This last point of view nearly died out when a generation of interesting languages languished in disuse a few years ago, but it is now making a revival. "The appearance of multicore processing in the SOC world has spurred lots of new work on ESL tools," Smith observes.

A different view of ESL comes from project leaders who are simply trying to get an earlier start on software develop-



ment. In these projects, designers use ESL tools to create a software-based virtual prototype. Designers merge the prototype with, for example, driver code they are developing and test the integrated software system much as they will later test the code running on the real hardware.

Others still feel that the proper emphasis of ESL should be on the architectural level of implementation, not on simply expressing the algorithm. This notion has led to languages—primarily C-derived—and tools that attempt to describe an implementation at an abstract level and either attach to it or infer from it as much information as possible about the structure, timing, and—increasingly—energy consumption of the RTL design it implies. There has been recent work here, as well.

With all of these approaches, expectations, and needs, there is certain to be confusion. Perhaps the best way to clarify the situation is to talk with some design teams who are actually using the tools at these various levels (see sidebar “Look into the future”).

## ALGORITHM DEVELOPMENT

Especially in the world of digital-signal processing, just getting the algorithm correct can be a major part of the design process. This challenge has at least two phases: getting the mathemat-

### AT A GLANCE

▣ Pushbutton synthesis of ESL (electronic-system-level) design to a full-chip netlist remains a dream.

▣ Engineers are using ESL tools in limited ways in some applications.

▣ System exploration and modeling can work well.

▣ Synthesis of algorithmic C to RTL (register-transfer level) works for datapaths.

▣ Tool developers are gradually expanding the ability to synthesize more-general RTL from C.

ics right and getting the translation from mathematics to hardware and software right. As Smith observes, this problem allowed perhaps the earliest successful uses of ESL tools, and it continues to be important.

Bob Davenport, DSP and system-design consultant with MC2 Technology Group, uses ESL tools in this way. He is a fan of Agilent’s SystemVue, a block-diagram-driven exploration environment that finds use in signal processing in communications and media-processing applications. Davenport says that he uses the tool for developing algorithms—often assembling them from functional libraries that are part of the environment—exploring them, and then generating C code, which he then

passes to Texas Instruments Code Composer Studio for optimization.

“For example, we may start with equations for a phase-locked loop, model it, and analyze its performance,” Davenport says. “Then, we can generate data, graph it, and use the graphs in reports.”

An important part of the capability for Davenport is that he can explore an algorithm in floating-point arithmetic with no regard to the number of significant digits and then insert tokens to set integer precision for specific signals. “So, if you are working on something like a frequency-domain GPS [global-positioning-system] correlator, for instance, you can get the core algorithm working first and then explore how to reduce the data width for implementation,” he says. He can achieve this goal simply by creating a fixed-point implementation, feeding it and the original floating-point model the same inputs, and comparing the outputs. If there are differences, he probes back through the algorithm to see where they are appearing.

Davenport uses his flow primarily for generating code for DSP cores. But the package can also output hardware descriptions for FPGAs, a capability that Davenport says is less well-developed but that he intends to explore further.

## VIRTUAL PLATFORMS

For some design teams, ESL is about algorithm development. For others, however, it is about concurrent hardware/software development. The IP (intellectual-property)-development group within Synopsys is a case in point. Interface IP for such interfaces as USB or PCIe (peripheral-component-interconnect express) requires driver software—not only so the licensee can use the interface, but also so the silicon team can do verification. But there is no time for the traditional approach of getting the hardware working in accordance with the specs, using the hardware to write the driver, and then using the driver and hardware to verify each other.

So, Synopsys has been using a virtual-prototyping facility from its own product line to pull driver development back in parallel with hardware development. Joachim Kunkel, vice president and general manager of the solutions group, and Joel Gotesman, R&D manager for the



Figure 1 Synopsys designers created a SystemC transaction-level model of this development board to enable hardware/driver co-design.



team, describe the process for a recent modification to the DesignWare USB OTG (On-The-Go) IP core.

According to Gotesman, the team started out with a hardware-evaluation board that the Synopsys DesignWare

team uses widely (Figure 1). “The board is intended for IP-design evaluation,” Gotesman explains. “It has a Samsung

## LOOK INTO THE FUTURE

As the use of ESL (electronic-system-level) synthesis spreads, the next steps are coming into focus. One lingering question is how synthesis applies to nondatapath structures and to control logic. For designers whose requirements don’t easily map into a traditional pipelined datapath, this issue is serious (Figure A). It appears that researchers are making progress, however.

One hopeful sign was the announcement last month of the long-rumored ESL-synthesis tool from Cadence. Ca-

dence based the tool on work that has been going on for years at Cadence Berkeley Laboratories. The company calls the tool, somewhat inaccurately, a C-to-silicon compiler, even though it generates RTL (register-transfer-level) code. You won’t know how well these claims pan out until reports start coming in from users. Companies have made such promises before. But the simple fact that Cadence is willing to boast about control-logic compilation in its introduction materials suggests

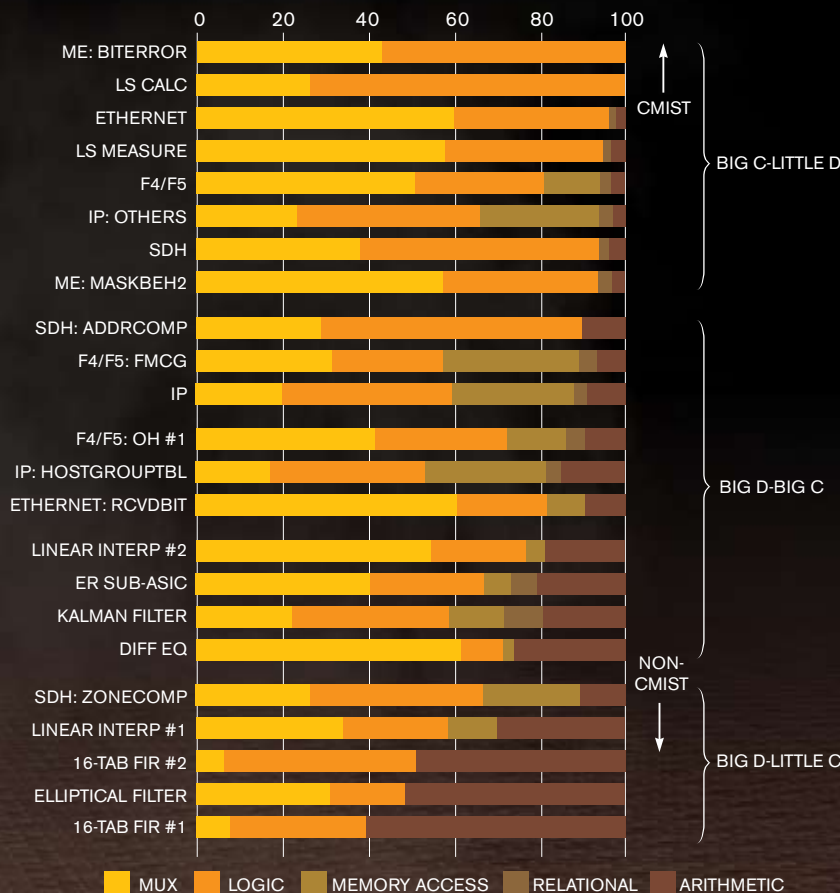
that it has achieved at least an incremental improvement.

Grant Martin, chief scientist at configurable-processor vendor Tensilica, has for years been working on the control-logic-synthesis problem. “For the current tools, the sweet spot for ESL synthesis is still datapath logic,” he says. “If there is a clear exception, it might be in the networking space, where some design teams have been using [ESL-synthesis-tool set] BlueSpec in areas such as packet-classification engines.”

Martin points to several issues with C-to-RTL control-logic synthesis. One is that the strategies that have so far worked best have been less dependent on C sources and not as algorithmic in their approach as the tools for datapaths. It’s not that C is a poor language for expressing control logic, Martin says. Some people are comfortable using C dialects in this way. But, in C, he points out, execution time for a control algorithm is generally not an issue. In control-logic hardware, every cycle matters.

Additionally, designers who specialize in control logic have developed their own tools and flows. Switching to a new approach may not excite them unless that approach can demonstrate a substantial improvement in their productivity—without being a threat to their careers.

Still, there is the experience of networking designers who have made just this transition. Also, Martin points out, in some specialized applications, control-logic synthesis is well-established. Tensilica, which uses a proprietary synthesis tool to generate the control logic for a CPU core with extended instructions, is a case in point. “I think it’s theoretically possible to do a more general tool,” Martin says. “But specialized tools, such as ours, may be more common.”



NOTE: CMIST=CONTROL- AND MEMORY-INTENSIVE SYSTEMS.

Figure A Tasks vary widely in their requirements for control, storage, and datapath functions. So, in some cases an ESL-synthesis tool’s handling of control logic can be critically important, while in other cases it is not an issue at all.

ARM-based chip, a display, connectors to the outside world, and an FPGA daughterboard.”

The DesignWare development team some time ago re-created this board as a transaction-level model in SystemC. At this level, the model builders may represent the board as a bus with a set of functional blocks. Each block has a defined set of legal transactions it can perform with the bus and a set of functions it can perform in response to a transaction. This model becomes a software virtual-prototyping environment. Timing information is usually not in the model at this level.

In addition to the transaction-level model of the board, development teams working on new IP blocks start out by creating a transaction-level model, again in SystemC, of the IP they are developing. Combining the model of the board with the model of the IP as they would implement it in the daughterboard FPGA, the team gets a virtual prototype.

In the case at hand, the design team’s job was to extend the USB OTG IP to support descriptor-based DMA (direct-memory access). Its flow allowed the team to start with the virtual prototype of the development board and the USB OTG IP and to simply modify the IP to support the new feature. This approach yielded a software virtual prototype the team could hand to the driver developers, so driver development could start almost as soon as the team nailed down the functions.

The result, according to Kunkel, is that the software team had the enhanced device driver debugged and ready to go four weeks before the hardware team had the RTL ready to program into an FPGA. At that point, Gotesman explains, the team could do a mixed-mode simulation including the transaction-level virtual prototype of the board, the RTL model of the new IP block, and the actual ARM code for the device driver. “This [approach] did require some adjustments to the SystemC code to include more timing information,” Gotesman explains. From there, the team could move directly to the IP in the physical FPGA and then on to test silicon. So the ESL description of the development board in effect becomes a testbench for both the ESL

description of and the RTL view of the new IP.

## **HARDWARE SYNTHESIS**

Synopsys’ use of virtual prototypes begs for one more step. Once you debug the virtual prototype, it would be great to push that magic button and synthesize the SystemC model into RTL or even into a netlist. To many designers, that scenario is the real—and unmet—promise of ESL design. But some teams are taking that approach, at least for certain kinds of structures.

One case in point is a recent project at Toshiba in Japan. Akiyoshi Oguro, LSI-division chief technology officer and group leader, and Takashi Okawa, assistant manager, describe a project to produce a hardware eigenvalue-decomposition engine from a C-language source. (Eigenvalues are sets of scalars associated with a linear system of equations.) Oguro’s team is working on processing automotive-collision-avoidance radar signals using eigenvalue decomposition as a powerful but computationally intensive numerical algorithm for image recognition.

Oguro’s team started with a description of the algorithm in Catapult C, evaluating three numerical techniques before settling on one for implementation. Okawa says that the presence of good math libraries—with features such as saturating integer arithmetic—is important in choosing a language for modeling at this level. Otherwise, much of the team’s time will go into coding numerical methods rather than exploring the algorithm.

Using a Mentor Graphics synthesis flow, the team added architectural constraints to the Catapult C code and synthesized RTL from the result. “It’s important to be able to separate the architectural constraints from the algorithm,” says Okawa. “Otherwise, the two get mixed up, and you have to change your algorithmic source code to make adjustments to the implementation.”

The team spent a short while examining the RTL code but moved quickly to implement it in an FPGA. They fed the FPGA real images they recorded from an automotive radar to verify the operation of the algorithm.

The ability to synthesize datapaths



from C has been around for a long time now, so it might come as no surprise that the Toshiba team succeeded. What might be less obvious is that synthesis now handles more subtle issues than just the arithmetic logic and registers in the datapath itself. One of the continuing problems with ESL synthesis—as with conventional RTL design—has been initialization sequences. It's not that hard to accidentally create a valid datapath that you can start only in simulation. But according to Oguro, the flow Toshiba used includes an auto-reset function that generated a correct initialization sequence in the RTL.

Was it worthwhile to use ESL synthesis in this case? Oguro estimates that the project would have taken about six months using conventional techniques. Using ESL synthesis, the designers had the information they needed from the FPGA in one month.

### FROM DATAPATHS TO SOCs

Synthesis of a functional subblock is an important step, but today's design teams need to understand entire SOCs in which the blocks have complex interactions. They also must move quickly from understanding to implementing initial RTL designs. Those problems face design teams at STMicroelectronics, long a leader in applying advanced methods, as the company develops SOCs for media-rich applications.

"Our goal is to automate the entire path between algorithm and implementation," says Pascal Urard, director of systems design at STMicro. "To [achieve this goal] we have put in place tools that take us from C++ to RTL, and we then connect to a conventional Synopsys synthesis flow."

The process begins with exploring the functions of a new design in C++. At this early stage, the team tries to estimate not only the operation of the chip design but also the throughput and latency of blocks and—critically, these days—the energy consumption. The team also begins the partitioning of functions into hardware and software.

Many architects argue that, in theory, this partitioning should happen as late as possible. But Urard warns that, with currently available tools, this approach is impractical. "Even at the system level, you don't use the same algorithms if you are targeting hardware that you do if you are targeting software," he says. Thus, the choice of target must happen early.

Power estimation is another serious issue. Urard says that, surprisingly, early power estimates are almost always within 20% of the final silicon results. "And that [situation occurs] in designs where there are lots of wires running at high frequencies," he says. "Normally, the estimates are much better than that. And, more important, the estimates of power variations are accurate."

Urard sees further progress coming in design estimation at ESL, but it will not be easy. With the growing use of aggressive and dynamic power-management techniques, estimation tools will have to pull information from low-level block and cell libraries to get enough data to form an accurate picture for a multivoltage, multimode analysis.

From estimation and system-level optimization, STMicroelectronics moves on to synthesis using Mentor tools. Urard

## FOR MORE INFORMATION

**Agilent Technologies**  
www.agilent.com

**ARM**  
www.arm.com

**Bluespec**  
www.bluespec.com

**Cadence Berkeley Laboratories**  
www.cadence.com

**Cadence Design Systems**  
www.cadence.com

**Gary Smith EDA**  
www.garysmitheda.com

**MC2 Technology Group**  
rgdavenport@rochester.rr.com

**Mentor Graphics**  
www.mentor.com

**Samsung**  
www.samsung.com

**STMicroelectronics**  
www.st.com

**Synopsys**  
www.synopsys.com

**Tensilica**  
www.tensilica.com

**Texas Instruments**  
www.ti.com

**The MathWorks**  
www.mathworks.com

**Toshiba**  
www.toshiba.com

admits that ESL synthesis today works better for datapaths than for control structures. But he does not see that issue as huge. “Datapaths are the main problems for us in SOCs,” he says. “Much of our control logic is actually embedded in the algorithms, so it does get synthesized. And the tools we have can generate control logic for standard interfaces, as well. So, the only real issue is control structures [neither] embedded in the algorithmic portion of the logic nor defined by an interface protocol.”

There are some other issues with the synthesis process, though. One involves the sensitivity of the synthesis process to coding style. “Small changes in C++ can result in huge changes in the RTL,” Urard warns. “So, you have to learn to code for synthesis. You end up writing code that is more like Matlab than it is like software. On the other hand, however, you don’t want to end up writing RTL in C, either. Fortunately, we have several years’ experience at this [type of work]. We have found that it really takes a new engineer about one project to learn to code effectively for RTL synthesis.”

“The situation is very like the early days of RTL,” Urard continues. “Back then, you didn’t dare put a multiplication symbol in your code. If you wanted a reasonable netlist, you wrote out the adds and shifts in RTL in just the right way. It’s the same now with ESL. You have to take into account the way the synthesis tool works.”

Another issue is more human than algorithmic. The next step after synthesis

is verification. In STMicroelectronics’ flow, that step means as much as possible formal verification of the equivalence of the C++ and RTL designs. And that verification requires that the designers, as they code the C++ for synthesis, also create constraints files for the formal verification tools.

Verification itself runs up against a couple of additional issues, Urard says. First is that perennial problem of formal tools: capacity. “We must verify very-large IP blocks,” Urard observes. “We are always looking for tools with larger capacity.” The other limitation is more difficult: Formal verification can prove the functional equivalence of a C++ and an RTL block. But it can’t verify the scheduling or the timing. The design still has to go through conventional verification with timed models.

Is the flow delivering? Urard says that STMicro will move forward, not retreating from its use of ESL synthesis. “We are seeing four to five times the productivity using this flow, compared with designing in RTL,” he reports.

Based on the experiences of these design teams, it seems clear that ESL design exploration, synthesis of datapath structures, and, to some extent—with senior and experienced people—synthesis of entire IP blocks have all established themselves in real design environments. ESL tools are no longer the pushbutton magic of marketing dreams. They are real and making an important contribution in the shops that have taken the time to learn and adjust to them. For many teams, ESL is no longer tomorrow’s answer; it is today’s. **EDN**

+ Go to [www.edn.com/080821cs](http://www.edn.com/080821cs) and click on Feedback Loop to post a comment on this article.

+ For more feature articles, go to [www.edn.com/features](http://www.edn.com/features).

You can reach Executive Editor **Ron Wilson** at 1-408-345-4427 and [ronald.wilson@reedbusiness.com](mailto:ronald.wilson@reedbusiness.com).

