

FIRST THE BAD NEWS: IF YOU HAVE ONLY RECENTLY BEGUN TO FULLY EXPLOIT HDL-BASED DESIGN WITH VHDL OR VERILOG, IT WILL SOON BE TIME TO MOVE ON AND MOVE UP TO A HIGHER LEVEL OF ABSTRACTION. BUT THE GOOD NEWS IS THAT THE LANGUAGES YOU'LL BE USING MAY NOT BE ENTIRELY NEW, AND THEY MIGHT EVEN BE VERY FAMILIAR.

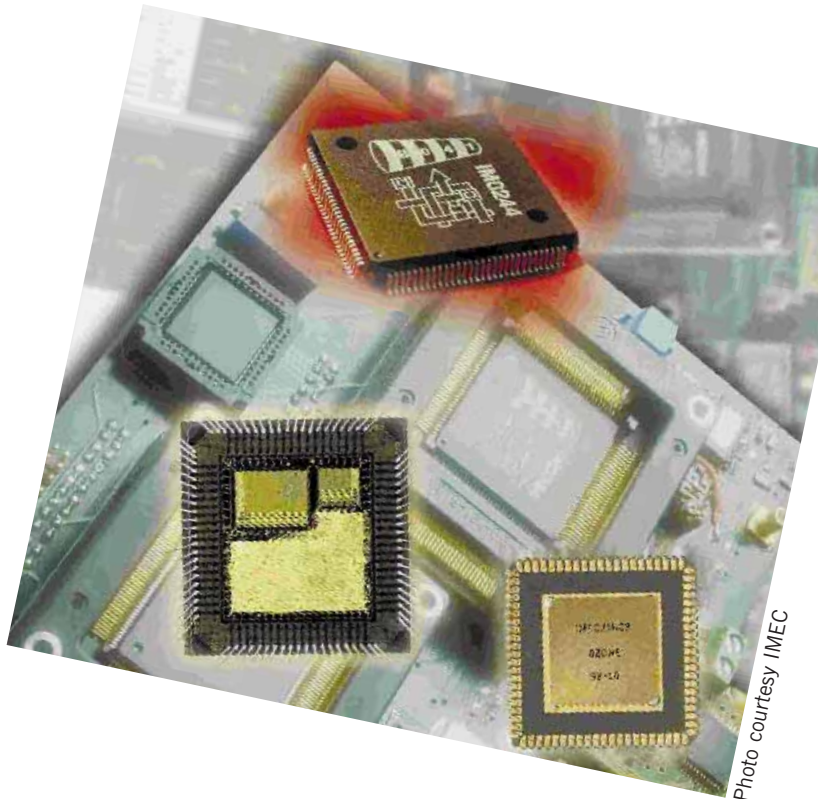


Photo courtesy IMEC

System-level design languages: to C or not to C?

FOR AS LONG AS EDA HAS EXISTED, its innovators have had their sights on a single high-level-design environment. In such an environment, you could model the overall behavior of a system before you detail its design and long before you build a prototype. You could plan and explore which areas of your design you could best realize in hardware and which you could best implement in

software. And you could use the description you created during the detail-design stages of both the system hardware and the software, constantly refining the system and moving toward a finished product.

The EDA industry has been successful in building this type of tool chain, but achieving a smooth transition from a system description at the highest level to implementation has largely remained in the

academic-research domain. Until now, the fact that the chain has been broken, lacking the link between high-level modeling and actual chip design, has not been fatal to the product-design process. Now, however, the ability to bridge the gap between system-level description and design implementation is moving from an aspiration to a necessity. Not surprisingly, the main force driving this movement is complexity: rising gate counts that you

At a glance **136**

Several more layers... **136**

What do you mean by "architecture"? **138**

need to design in less time. As your designs increase in complexity, the number of degrees of freedom for decisions at the architectural level—such as how you should break down into functional blocks the operations that your design will have to carry out and how those blocks should interact—expands dramatically. You must have some sort of executable specification at a high, behavioral level—at which simulation will quickly reveal the effects of architectural changes—to explore the options before you proceed further into the design.

Such specifications have existed for some time now; most often, they are in C, C++, or System Description Language (SDL) or unified modeling language (UML) in the telecommunications sector. Even if your job title doesn't include "system architect," you may write such specifications; they define the overall shape of the product and may form the basis for a testbench, against which you can measure subsequent implementations of the design. Until now, however, the most common practice has been to take the outline that this top-level

AT A GLANCE

▶ Design complexity is continuing to rise, forcing another upward move in the level of abstraction at which the design is started.

▶ VHDL and Verilog are too inflexible and too verbose to handle the transition to designing at a higher abstraction level.

▶ None of the language options proposed to handle this abstraction shift is entirely new; they range from extended HDLs and ANSI C to System Description Language.

▶ The migration will fundamentally change the way you and your team undertake system-level design.

spec-writing exercise provides and discard much of the information it contains. When you write the code that defines the hardware in VHDL or Verilog, you re-code it by hand.

This method has several drawbacks. The most obvious one is that it is a du-

plication of effort that lengthens the design cycle, so using this method is a time-consuming bottleneck. More subtly, manually recoding the architecture that you first chart in the high-level model is an interpretation rather than a transformation of the design. Even if the new implementation of the design works—that is, if it carries out the intended functions, which you must painstakingly verify—it may not reproduce the precise design intent expressed with the original.

EDA- industry analysts, who like to see long-term cyclical trends in these matters, rationalize that fundamental changes in logic-design methodology have occurred on roughly a 10-year cycle. Now, about a decade since the widespread adoption of logic synthesis, the time is ripe for a further seismic shift. Even if you don't entirely subscribe to that analysis, the pressures of escalating complexity and the closely related need for designs to painlessly assimilate circuit functions that you import fully formed (namely, intellectual property (IP) blocks) means that the established design path is rapidly losing momentum.

SEVERAL MORE LAYERS

VHDL and Verilog do not handle higher levels of expression well. Even in their own domain they are often criticized for verbosity; that is, there is simply too much code for the amount of hardware they describe. This problem is becoming more complex. The languages are not tuned for expressing the kind of algorithmic functions that you might use to represent the activity within a complex circuit block on the highest black-box level. Recall that neither VHDL nor Verilog was conceived as a hardware *design* medium. Verilog has most of its roots in board-level simulation; VHDL was originally a description language, intended mainly to document and to specify rather than to design. But conventional high-level computing languages were designed to execute algorithms in a computational medium, so why not use them?

If you look at the aggregate offering across a number of suppliers, a design flow is evolving; you use a high-level language early in a process but not necessarily at the earliest stages. This change does not replace but augments algorithmic exploration tools, such as Matlab (www.matlab.com), COSSAP (www.synopsys.com), or SPW (www.cadence.com). However, a language such as C can express a number of levels of circuit function, beginning at the top, black-box block level. From there, you add detail, replacing a few statements of C that simply implement a mathematical algorithm. More detailed coding shapes how your design will compute the functions, and you will start to add timing onto that coding. At some point, you must determine which functions you will do in hardware and which you will do in software. The

hardware portion will be ready for the key step that distinguishes this phase of EDA tools from previous tools: compilation or translation to VHDL or Verilog. In an ideal world, implementation from this point to physical layout would be automatic. In reality, this point is where the established EDA tool chain takes over. Many participants in this new sector envisage shaping the hardware detail before leaving the C (or other language) domain, by writing the code in a style that drives the HDL in a well-defined manner; look for terms such as "RTL-C."

You will still write HDLs similar to the way you now write C code for an embedded-processor application. Just as you now insert a segment of assembler to implement a critical routine in the middle of a high-level-language microcontroller program, you will insert a section of

HDL for a critical part of your hardware. The new class of translation tools will pass this HDL unchanged for synthesis, so from the highest level you will be able to define well-controlled hardware structures for the most demanding parts of the design.

You will likely see analogous approaches evolve among designers, too. Some designers determine that the most efficient style is pushing the entire approach through from the high level and exploiting the automated tools to the maximum. Others continue to demand hands-on control of important details.

The EDA sector is following a traditional route toward this transition: It has spawned several start-up companies, each offering and exploring distinct approaches to the problem.

A number of tools that effectively translate from a higher level into today's synthesizable HDL formats are facilitating the move to a higher level expression of design intent as a routine starting point. And more tools are in development. A key question to ask yourself is: "What is the most appropriate language for that next step?" It's important to remember that when you embark on this transition, you are not just moving to a more compact and powerful way of coding for hardware. (However, doing so may be a worthwhile and useful objective.) Rather, you are moving into the hardware/software co-design regime (see sidebar "Several more layers").

As a first step, determine which languages are in contention. Several distinct categories exist. Perhaps the most obvious approach is to develop today's HDLs (VHDL and Verilog) so that they handle a higher level of abstraction. A variety of sources, including established EDA players, start-up ventures, and IEEE language committees, have proposed extensions of both languages. A second set of candidates for high-level design languages is mainstream high-level computing languages, including C, C++, Java, and extensions of them; yet a third category is purpose-designed specification languages, such as SDL and UML. However, using these languages involves ex-

tending them downward to reach toward the physical world.

C AND C++

Because many first-pass system descriptions are already in C, many EDA suppliers in this field believe that C is the most appropriate vehicle to use as a next-generation language. However, those against this use of C often cite C's lack of support for certain key features that are necessary to describe hardware—most notably, concurrency. C is a sequential computing language that does not describe concurrent processes. Hardware is intrinsically parallel, and specifying it requires a method of prescribing concu-

WHAT DO YOU MEAN BY "ARCHITECTURE"?

An analogy commonly used to describe the history of EDA is the progressive demolition of brick walls. This analogy represents discontinuities in the design process that automated tools have overcome. An example is the step from logic design to physical design. The expression "to throw the design over the wall" represents the shift from one design discipline to another, when logic design is complete and silicon, or board, layout begins. As the penalties associated with the two-step approach have become unacceptable, physical-synthesis tools have emerged to unify the process. For high-level designs, the brick walls are between hardware, software, and architectural designers. Working in languages that don't (at least initially) carry any implications of how you will realize functions and algorithms means that, in many organizations, teams are going to have to become more close-knit.

At CoWare (www.coware.com), European Sales Manager John MacDermott points out that subtle—and in some cases non-technical—issues exist. For example, when you have assembled the entire team dealing with the front-end design, make sure you understand each other. Architec-

ture does not always mean the same thing for everyone involved. If you come from a hardware background, "architecture" implies microprocessor architectures, and you may be allowing previous experience with microprocessor cores to shape your thoughts. In other words, you think in terms of physical architectures. If you are used to working at the top level, you may be thinking purely in terms of function, with little thought as to how the functional blocks that the design's algorithms need will map onto hardware and software blocks. Or, if your background is software, "architecture" may mean the programming model of the processor cores that the design is likely to use. MacDermott notes that one of the first tasks in introducing a co-design environment is establishing a glossary in which everyone involved agrees on what the words mean.

When hardware and software co-design becomes a reality and you can move the functional blocks of the design between the two domains with a consistent view of the complete project, any architectural changes you make instantly reflect on the work of everyone involved. This fact may require a fundamental

change in your working methodology and a need for everyone to understand of the implications of individual design decisions on the work of the entire team.

CoWare's N2C is a system-on-chip (SOC) design environment that claims to provide support from the highest level of specification to transfer into HDL synthesis and importation of intellectual-property (IP) blocks. According to MacDermott, N2C spans from an untimed, completely abstract description to a timed C for translation to HDL. However, you need not take the whole span of that process onboard at once. For RTL designers, moving to RTL-C is a relatively small step.

MacDermott contends that most of the value you add to your project lies in behavioral C, which defines the algorithmic basis for the way you get the job done. N2C focuses on maximizing that value. The mechanics of the task include mostly interface design: interfacing logic blocks in hardware or writing all the service functions in software.

MacDermott says, "These interfaces are either right or wrong. By themselves, they add no value to the project." Automating as much of this part of the design as possible is the

basis for CoWare's Interface Synthesis. From an untimed-C description, the flow is to hardware and software partitioning. From there, the hardware flow adds a shell to the C that superimposes rough timing. At this level, simulation is still fast because you perform it by running compiled C. But underlying the C description is the fact that an instruction-set simulator has already been introduced. In subsequent stages, the designer uses the tool to add more detail to the bus-cycle-accurate model, in turn leading to cycle-accurate RTL-C.

MacDermott also notes the move to CoWare's "platform-based design" in which the platform or processor-plus-logic foundation of the design is a known quantity. In effect, the platform becomes an application-specific standard part that the silicon vendor provides around base IP, which allows you to concentrate on the higher level aspects of the design. In some companies, MacDermott says that people have been unconsciously using this process.

He adds, "They need to start doing it explicitly." System-level design changes the relationships not only within the design team, but also with external suppliers.

rency. Nevertheless, Frontier Design's (www.frontierd.com) general manager, Herman Beke, believes that the sheer momentum of all the work that is currently done in C virtually dictates its success. Beke notes that, among EDA vendors and users, multiple sets of proposals exist for extensions of basic C to add support for concurrency, most often by adding C++-type classes. He says that proponents of C must now agree on a de facto standard for these extensions. Beke expects moves to resolve this situation before the end of 1999. Frontier expects users to migrate over time to designing with full C++, but he believes the move to fully object-oriented programming will be a big step for hardware designers to undertake and one that will come gradually. Frontier sells the Architectural Synthesis Tool Kit, and the A|RT C-to-HDL-conversion tool (Figure 1). The tool kit is a set of libraries and tools that allow you to express a design in behavioral C in an architecture-independent form. Beke emphasizes the importance of doing as much architectural exploration as possible before synthesis, using the tool kit to control resource allocation, assignment, and scheduling to optimize designs (Figure 2).

The A|RT tool set focuses on implementation of DSP algorithms and includes support for fixed-point data and operators via a C++ class library. (You can download the A|RT Library from Frontier Design's Web site.) You use standard commercial C++ compilers, but you work mostly with a basic C subset. This subset includes only a few special operators to develop fixed-point algorithms and accurately handles quantization and overflow effects. Once the behavioral C algorithms are complete, the builder tool automatically translates the design to VHDL or Verilog.

Cadence (www.cadence.com) Labs has been working with the French research organization CMA (Sophia Antipolis, France) and the computing-research organization Inria to develop an alternative approach to extending C. Researchers at CMA have developed Esterel, one of a class of languages known as synchronous-reactive programming languages (see www.inria.fr/meije/esterel). Esterel describes communicating processes by means of complex reactive commands, and Cadence believes that synchronous-

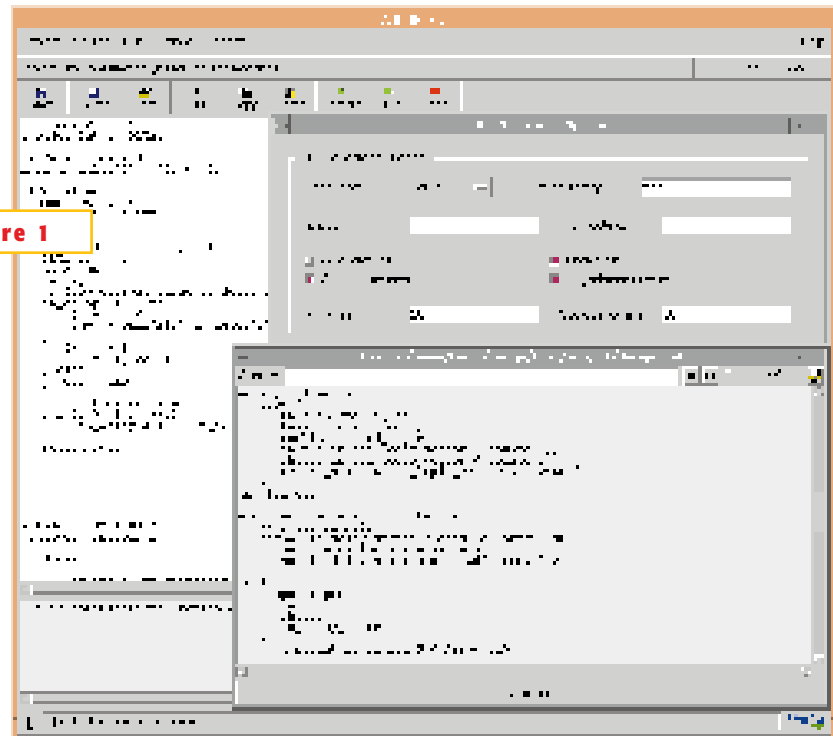


Figure 1

Frontier Design's A|RT Builder tools show extended-C and generated-VHDL code.

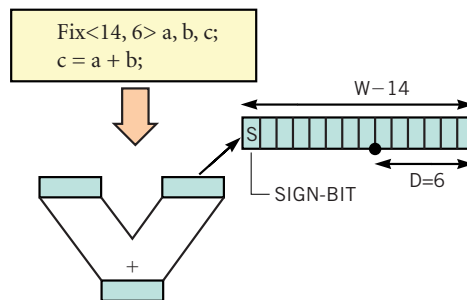
reactive programming is a suitable vehicle for building system-level components for system-on-chip (SOC) design. However, Cadence does not think it needs to introduce a completely new language into the SOC arena. Rather, the company believes that a hybrid language, ECL (Esterel C), is the way forward. ECL will be based on conventional C, augmented with reactive statements to add concurrency and control features. New constructs for waiting, concurrency, and pre-emption have descriptors, such as "await," "emit," "parallel," and "do/abort."

ECL is not yet a product, but Cadence will use some of these techniques in a technology called Virtual Component Co-design (VCC), which it will formally introduce this year. A paper at the 1999 Design Automation Conference details ECL's workings (Reference 1). For now, Cadence is keeping its options open.

Mike Gianfagna, Cadence's vice president of product marketing for design and verification tools says, "Since our customers must have the ability to integrate IP from multiple sources, it is our primary strategy to support IP developed in multiple languages within the context of our system-design tools." He notes that many companies are chasing a standard, and Cadence anticipates that one of its competitors will soon develop a strictly C++ approach. For now, however, Cadence is, "choosing to play the field."

All of the major EDA vendors are taking a position on the coming shift in design languages, but they are avoiding taking a position that will subsequently restrict them. Mentor Graphics (www.mentor.com) intends to be "language-

Figure 2



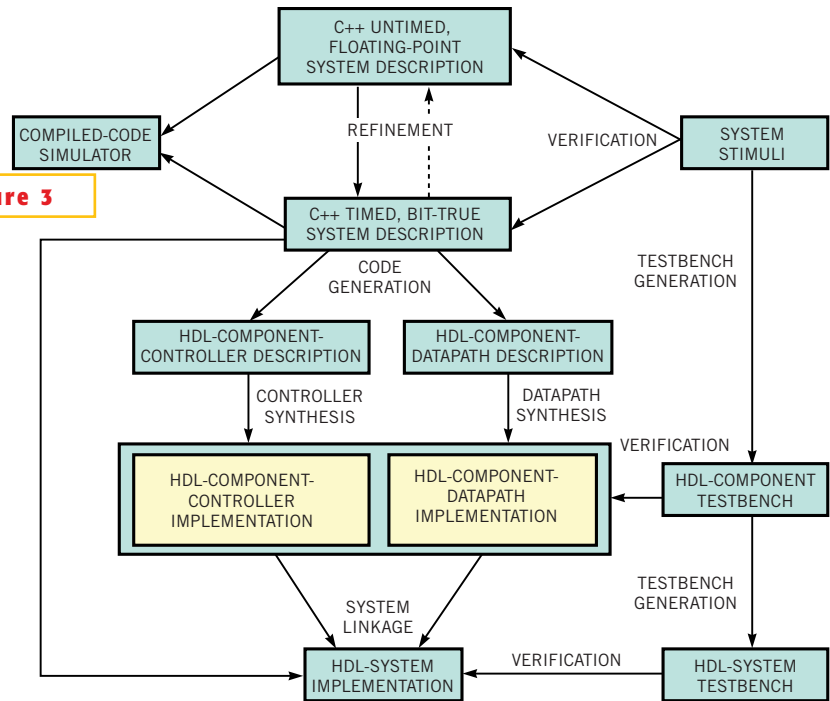
Frontier Design's A|RT Library allows you to control hardware dimensions; the Fix <14,6> statement specifies a variable with a 14-bit-long word and 6-bit precision.

neutral” and sees some of its customers using ICL’s (www.icl.com) Supervise (via its Model Technology subsidiary) in UML, SDL, and C++ and C with extensions. Like other companies, Mentor contends that mixing high- and low-level representations will be necessary to bring RT-level IP into designs that are conceived at the system level.

Synopsys (www.synopsys.com) is planning an announcement with CoWare. Synopsys’ strategic marketing director, Brian Barrera, had previously argued that the last thing the EDA industry needs is a completely new proprietary design language, because it would prevent companies from producing the broad range of tools to support it. Barrera maintains that the answer is to use a language that is already in widespread use. He notes that standard HDLs are bad candidates because they were written to describe hardware, not software or system-level algorithms. Barrera advocates a C-based modelling infrastructure and gives weight to Synopsys’ argument for suitable C++ class libraries that engineers with expertise in base (non-object-oriented) C can use.

The IMEC research organization in Belgium has followed this path and developed a methodology based on C++.

Figure 3

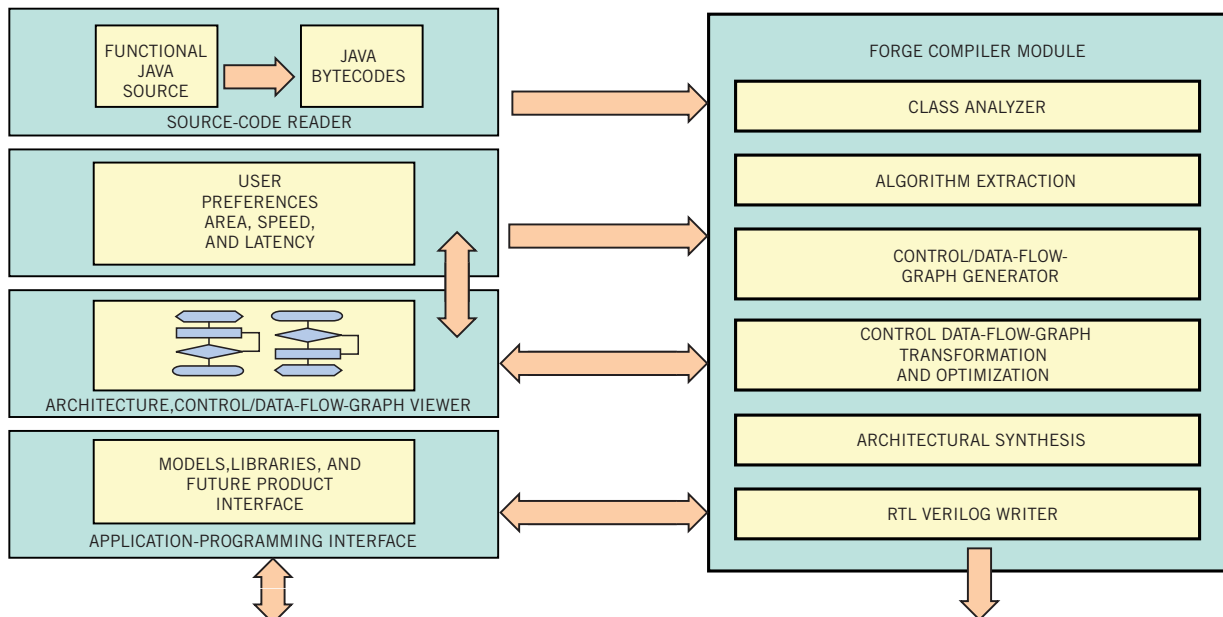


Based on an object-oriented description, this chart shows the design flow through IMEC’s OCAP C++ based environment.

IMEC has developed a cosimulation and co-design environment based on the Tippy multithreading library. You use Tippy to schedule the threads that represent the hardware and software blocks. The li-

brary also gives timing attributes to the threads. Researchers at IMEC have used their C++-based methodology to develop the digital part of BO4, an asymmetric-digital-subscriber-line (ADSL)

FORGE USER INTERFACE



DEVELOPMENT ENVIRONMENT

Figure 4

LavaLogic’s forthcoming Forge-J design environment, which infers structure from a Java description and then outputs HDL code, involves several processes.

modem and to develop an engine for MPEG-4 algorithms. Both of these projects have successfully reached silicon. If you implement it as a single chip, the logic of the ADSL design contains two datapaths: control hardware and a large amount of embedded software. An ARM core controls the ADSL start-up sequence.

The IMEC research team is basing this design environment on C++ and calling it OCAPI (www.imec.be/ocapi). IMEC's head of digital broadband terminals, Serge Vernalde, explains that, in C++, you can change semantics while remaining within the same syntax, using the power of operator overloading. This ability allows you to apply different implementation meanings to objects. For example, if you have an addition statement, you don't directly execute it. Rather, you create an addition object in memory, with which you can then do different things. You can cause the object to be executed by running a simulation, or you can translate it to VHDL. The system generates both synthesizable VHDL and a testbench (Figure 3). In this way, you don't rely on the sequential semantics of C++, but you have a means of handling concurrency. Vernalde adds that IMEC researchers opted for C++ because you can use the open, standard language; adding special constructs to C would make it non-standard.

Vernalde says that the object-oriented learning curve is not that bad: "You learn the objects in the dedicated library and work with them." Of course, if you further develop the libraries yourself, you will need a thorough knowledge of C++ and object-oriented techniques. Vernalde admits that engineers with the necessary crossover skills (of object-oriented programming and hardware design) are now rare, but he believes that skill set will develop.

OCAPI is not a commercial product, but you can use it to gain experience with the C++ approach to system-level design. You can download an evaluation version of the environment and libraries, which are limited only because they handle a maximum of 3000 signals, from IMEC's Web site.

"OCAPI is not a closed toolbox but a philosophy," Vernalde adds. "You can learn design in object-oriented style us-

LISTING 1—SUPERLOG FROM CO-DESIGN

```
typedef struct {string s; ref node left, right;} node;

ref node n, root; // global data - pointers to nodes
int visited = 0; // global data - number of nodes visited

function ref node treeFind(string str, ref node parent);
    if (parent == null) return null;
    visited++;
    if (str == parent->s) return parent; // string compare
    if (str < parent->s) return treeFind(str, parent->left);
    else return treeFind(str, parent->right); // recursion
endfunction

state {S0, S1, S2} cstate; // state variable with enumeration

always @(posedge reset)
    transition (cstate) default->> S0; endtransition

always @(posedge clk iff !reset)
    transition (cstate)
        S0:if (inp == 0)->>S2; // change state
        S2:if (inp == 1)->>S1; else->> S0;
        S1->>S0 n=treeFind("shergar",root);
endtransition
```

ing OCAPI as a starting point." IMEC is supporting the learning process with training courses and has doubled the number of courses it originally planned for this year.

C Level Design's (www.clevedesign.com) name leaves little doubt about where this company enters the arena. Unlike other proponents of C, however, this start-up proposes that you can use the standard ANSI C language as it is without restrictions or additional libraries. You can resolve the concurrency issue, C Level says, because the parallelism that hardware requires is inherent in algorithmic coding and can be inferred and extracted. You can work with tools in either a behavioral- or an RTL-style C; you derive the RTL version from the coding styles that HDLs use. You can translate either style of C to HDL for synthesis; C Level claims that simulation performance increases by orders of magnitude at the high level. As with other C-based approaches, simply compiling and running the C code amounts to a simulation; the System Compiler product family allows you to compare that simulation result with others that you run on the HDL output by the language-synthesis process. (C Level officials say that the company's tools synthesize HDL from the C input, which is distinct from subsequent synthesis of hardware structures from the HDL.) RTL-C is a coding style

that identifies algorithms to handle logic constructs and specifies how to use them to provide a bridge in design style for designers currently working in HDLs.

CynApps (www.cynapps.com) recently made its Cynlib library freely available via an open-source licensing program. Cynlib is a C++ class library that implements HDL-like features in the higher level language. CynApps officials say that the product's modelling and simulation spans abstraction levels from very high level to RTL using only the library and widely available C++ compilers. Cynlib contains a cycle-based simulator that provides cycle-accurate results with minimum intervention from the user to create simulation code. According to the company, the combination is "effectively an HDL dialect of C." CynApps also produces a synthesizer that turns out synthesizable Verilog at the completion of the process of successive refinement from algorithmic level.

VHDL EXTENSIONS

Some EDA suppliers believe that the best way to get to where you want to be is to start from where you are. In other words, if you have working HDL synthesis, you should not seek to replace that synthesis with something different. You should extend its function to the level of abstraction that you require.

ICL Design Automation (www.icl.com/da) has produced VHDL+, a version of VHDL that has definitions at higher levels of abstraction for data, time, and resources than the standard language. VHDL+ is a cornerstone of ICL's Supervise methodology for SOC design. Supervise includes a compiler that converts the VHDL+ into conventional VHDL. (VHDL+ typically uses one-tenth the number of lines of code that conventional code uses to describe equivalent functions.) You can simulate the output of the compiler using any standard VHDL simulator. ICL integrates with Model Technology's (www.model.com) ModelSim. The most recent addition to the suite is SuperviseMX, a multilevel and multilanguage simulator that now supports VHDL+, VHDL, Verilog, C, and C++ and will eventually support SDL, UML, and Java modelling. Event-

driven simulation combines with techniques that ICL calls interface and message-based simulation.

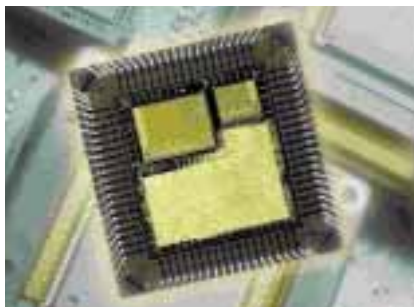
Interface-based design is a key element of the design style that ICL proposes. It describes the interaction between components (functional blocks) in a design by capturing both the behavior of the blocks' ports and the communications protocol you observe when passing information across the ports. With this level of information, interface-based design supports multilevel modelling of the interface so that true multilevel simulation can take place; if the models of the functional units in the design span different levels, the interface model understands how to present the data needs to each level for a simulation to run. ICL claims that this approach has benefits, including faster and more meaningful full-system simulation and the wide applicability of testbenches; you can apply the same testbench across multiple levels of the design hierarchy.

With Summit Design (www.summit-design.com), ICL is developing a version of VisualHDL for VHDL+ that will apply Summit's graphical-design-entry tools to the higher level language.

The System Level Design Language (SLDL) committee of VHDL International is further developing VHDL with Rosetta. Rosetta is not strictly a language. Rather, it is a framework through which you can impose design constraints on other higher level languages. Eventually, this process will produce workable VHDL. Use of Rosetta is some way off, however; its formal publication is not due until later this year, and tools will follow.

As you might expect from a company that has provided a graphical-entry system for the HDL level, Summit is planning to do the same at the system level. Its Visual SLD design environment will support both textual- and graphical-design capture. The higher level version will support design entry in VHDL, Verilog, C, C++, and ICL's VHDL+. It will also integrate a route to C-Level's C-Sim.

Just as before, entry styles will include block diagrams, flow charts, and other graphical methods, but, in this generation, the underlying expression of the content will be in C; the connections that the system will depict



SOME EDA SUPPLIERS BELIEVE THAT THE BEST WAY TO GET TO WHERE YOU WANT TO BE IS TO START FROM WHERE YOU ARE.

will not be signals but more abstract communication channels. The system will offer a C-to-VHDL translation and will support mixed-level simulation (for example, when some functions are expressed in C and some in HDL). Some of the blocks in the design will become hardware, and some will remain as software. Summit's chief technical officer, Guy Moshe, notes that the environment incorporates VHDL+, but he anticipates difficulties including higher level languages, such as UML.

You can think of Superlog as a new language or as an extension of Verilog. Superlog is a language proposal that start-up Co-Design (www.co-design.com) created. Co-Design's intends to place the language in the public domain; when it does, Co-Design will introduce a simulator and other tools to work with it. Superlog has similarities to Verilog (and, in some respects, to VHDL) but adds C-like high-level-language constructs (**Listing 1**). Co-design believes that only a new language can fulfill all the speed and power requirements to deal with the complexity problem and take a project from conception through implementation and allow design methodologies to evolve into more powerful techniques.

Java is also a proposed vehicle for SOC designs. At Frontier Design, Beke notes that Java has

advantages, but he believes that C and C++ will dominate. Not so at start-up LavaLogic (www.lavalogic.com), which is planning a Java-to-HDL compiler for late 1999. LavaLogic's director of engineering, Don Davis, points out that Java is generally simple to learn, plus it is object-oriented, robust, portable, and multi-threaded. To handle the concurrency issue, Java has mechanisms to extract the parallelism inherent in a design description. These mechanisms, Davis says, include Java's abstract memory handling, stack orientation, and multithreading. He even asserts that, applying LavaLogic's forthcoming tools to Java, you can identify implicit parallelism that the designer may not have intended (**Figure 4**). LavaLogic sees Java's freedom from the use of pointers as an advantage, and the multithreading feature allows you to code explicit parallelism. LavaLogic will call its Java compiler Forge-J, and it will both infer optimized architectures from Java code and output synthesizable Verilog.

SPECIFICATION-LEVEL EXPLORATION

ArchitMate is a new tool from Arexsys (www.arexsys.com) based on the company's "architectural-exploration methodology." Arexsys built this technology on many years' work from the TIMA laboratory (Grenoble, France). The technology initially works with a system-level specification written in SDL. Arexsys then derives an executable specification. The architectural exploration includes synthesis of process communications; allocation of processors to functions; hardware/software partitioning; and automatic generation of software code, interfaces, and hardware-RTL code. ArchitMate reflects Arexsys' strong background in computer-aided software engineering; the company claims that the product most effectively controls switching functions between hardware and software domains. CosiMate is a cosimulation backplane that gives Arexsys' tool suite a multilanguage simulation capability by interconnecting commercial simulators. □

REFERENCE

1. Lavango, Luciano, and Ellen Sentovich, "ECL; A Specification Environment for System-Level Design," paper 29.1, Proceedings of DAC, 1999.

You can reach
Editor Graham
Prophet at +44 118
935 1650, fax +44
118 935 1670,
e-mail graham.prophet@cahnerseurope.com.

