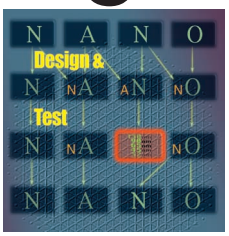
Hardware/Software Interface Codesign for Embedded Systems



Separate hardware- and software-only engineering approaches cannot meet the increasingly complex requirements of embedded systems. HW/SW interface codesign will enable the integration of components in heterogeneous multiprocessors. The authors analyze the evolution of this approach and define a long-term roadmap for future success.

Ahmed A. Jerraya
TIMA Laboratory

Wayne Wolf
Princeton University

An embedded computing system is an application-specific electronic subsystem that is used in a larger system such as a consumer appliance, medical device, or automobile. Embedded systems can embody complete system functionality in several ways—for example, by using software running on CPUs or in specialized hardware accelerators.

Technological evolution—particularly shrinking silicon fabrication geometries—is enabling the integration of complex platforms in a single system on chip (SoC). In addition to specific hardware subsystems, a modern SoC also can include one or several CPU subsystems to execute software and sophisticated interconnects.

Mastering the design of these embedded systems is a challenge for both system and semiconductor houses that used to apply a software- or hardware-only strategy. In addition to classic software and hardware, SoC engineers must design hardware-dependent software and software-dependent hardware. Codesigning these HW/SW interfaces requires a new kind of engineer who understands both hardware and software design.

Ninety percent of new application-specific integrated circuits (ASICs) fabricated using 130-nm technology already include a CPU,[1] and 65-nm SoCs with more than 100 processors could become commonplace by 2007. Multimedia platforms such as Nomadik and Nexperia are examples of multiprocessor SoCs that use digital signal processors, microcontrollers, and other kinds of programmable processors.[2] These systems exploit heterogeneous cores to meet tight performance and cost constraints. As the trend of building heterogeneous multiprocessor SoCs accelerates, they will be composed of multiple, possibly highly parallel processors for use in applications such as mobile terminals, set-top boxes, and game, video, and network processors. To facilitate communication, these chips will also contain sophisticated networks-on-chips (NoCs).

Providing SoCs consisting of an assembly of processors executing tasks concurrently will require design methodologies to focus on selecting and using either programmable or dedicated processors in place of the gates and arithmetic logic units that current methods use. Compared with conventional ASIC design, such a multiprocessor SoC requires a fundamental change in chip design.

MULTIPROCESSOR PLATFORMS

Application requirements force today's system designers to develop specific platforms for different design spaces. Some have speculated that the semi-

0018-9162/05/$20.00 © 2005 IEEE

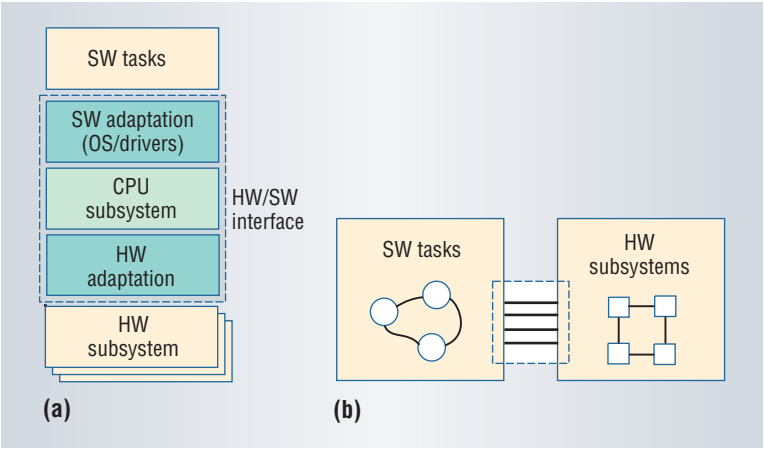


Figure 1. Evolution of interface-based design. (a) Current methodology prevents designing the software until the hardware platform design is complete. (b) HW/SW interface codesign requires abstract models of both types of components.

conductor industry is moving toward a universal chip that will provide all the computation power that all applications require. This solution should be developed when it becomes feasible; using a standard platform is likely to become the preferred option because it would eliminate the cost and effort required to build specific HW/SW platforms.

However, due to many factors, it is likely that SoC designs will use multiple platforms in the foreseeable future. It is possible to build one or more platforms that effectively provide the basis for many products within a particular application space—such as Texas Instruments' TI-OMAP and STElectronics' Nomadik for mobile terminals, and Philips' Nexperia for digital TV.[2] However, enhancing such hardware platforms for use across multiple applications is not viable because they must meet several stringent design constraints simultaneously: hard real-time performance, low power consumption, and low cost. Under these circumstances, the platform must be specialized to exploit a given application's characteristics.

Further, applications that have different combinations of requirements demand multiple architectures. For example, although AVC/H.264 is a common standard for video compression, different types of video compression systems require different platforms. The computation complexity required for video compression in digital cinema and high-definition video (HDV) cameras is more than 32 tera instructions per second. A cell phone with a video camera uses much smaller frames and lower frame rates, which requires less computation but imposes more stringent power consumption requirements. Because cell phones must also be more physically compact than high-end video cameras, they require more highly integrated architectures. Thus, even this one application can require different platforms.

HDV recording illustrates the need for heterogeneous platforms. Assume that the design uses a pure software approach with a SoC platform consisting of programmable processors. To meet the computation requirements, the SoC platform requires 32,000 RISC processors running at 1 GHz. In the foreseeable future, such a SoC platform may not be realizable in terms of either chip area or power consumption. Such a platform has a significant limitation in terms of power consumption because it would require numerous transistors, and the leakage current—which is proportional to the number of devices—would dominate power consumption. However, when implemented as a mixed HW/SW design, the same MPEG-2 encoder would require only a four-processor solution for a digital cinema application.[3]

INTERFACE-BASED DESIGN

For quite some time, Moore's law has driven advances in chip density that far outpace advances in designer productivity. To get back on track, designers must work at higher levels of abstraction. The productivity of a designer who can generate only 100 lines of Hardware Description Language (HDL) code per day is higher if those lines represent large blocks rather than logic gates.

SoC design generally requires developing complex software, entailing hundreds of thousands of lines of code, to run on the SoC platform. The designer must accomplish this work while balancing the competing constraints of a short time-to-market window and ever-increasingly complex functionality. Scaling current ASIC design approaches to such highly parallel multiprocessor SoCs is difficult, and using classic methods to design these new systems would result in unacceptable realization costs and delays.

These constraints are pushing SoC design toward an interface-based methodology that takes advantage of intellectual property.

Current design methodology

Traditional ASIC designers have a hardware-centric view of the system design problem. Similarly, software designers have a software-centric view. SoC designs require creating and using radical new design methodologies because some of the key problems in SoC design lie at the boundary between hardware and software.

As Figure 1a shows, a SoC can include specific hardware subsystems and one or several CPU subsystems to execute software. The design includes a hardware adapter—a bridge or communication coprocessor—to connect the CPU subsystems to the other subsystems. Each CPU subsystem includes a register transfer level (RTL) or gate model of the CPU and a set of peripherals connected using the CPU bus.

In the final design, the system compiles and represents software as binary code that it can load in

the CPU subsystem's memory. The current SoC design process uses separate teams working serially to create the hardware and software designs.

The first step consists of designing the hardware and validating it through RTL simulation using classic HDL simulators, which are much too slow to handle the embedded CPUs. Using a CPU instruction-set simulator can accelerate this simulation. Instruction-set simulators can use a cosimulation backplane to connect to the HDL simulator.[4]

The next step involves testing an operating system or middleware on the hardware platform and then porting the software to the OS or middleware. Thus, the software design team can begin only after the hardware platform design is complete. This often leads to poor hardware designs because problems caught during software development cannot be fixed in the platform. It also means that the design process takes far too long.[5]

A new approach

Concurrent HW/SW design requires abstract models of both types of components, as Figure 1b shows. Ideally, the design process would start with a set of software tasks communicating with a set of hardware subsystems. Because software components run on processors, the abstraction needed to describe the interconnection between the software and hardware components is totally different from the existing abstraction of wires between hardware components as well as the function call abstraction that describes the software.

The HW/SW interface abstraction must hide the CPU, a hardware module that executes a software program. On the software side, the abstraction hides the CPU under a low-level software layer ranging from basic drivers and I/O functionality to sophisticated operating systems and middleware. On the hardware side, the interface abstraction hides CPU bus details through a hardware adaptation layer generally called the CPU interface. This can range from simple registers to sophisticated I/O peripherals including direct memory access queues and complex data conversion and buffering systems.

This heterogeneity complicates designing the interface and makes it time-consuming because it requires knowledge of both hardware and software and their interaction. Consequently, HW/SW interface codesign remains a largely unexplored no-man's-land.

General-purpose computer system designers must also consider both hardware and software, but the two are more loosely coupled than in SoC design. Consequently, general-purpose systems typically model HW/SW interfaces twice: once to test the hardware design and the second time to validate software functionality. Using two separate models induces a discontinuity that wastes design time and results in less efficient, lower-quality hardware and software.

This overhead in cost and reduced efficiency are unacceptable for SoC design. Efficiently combining hardware and software to share a single interface requires a new type of HW/SW designer.[5]

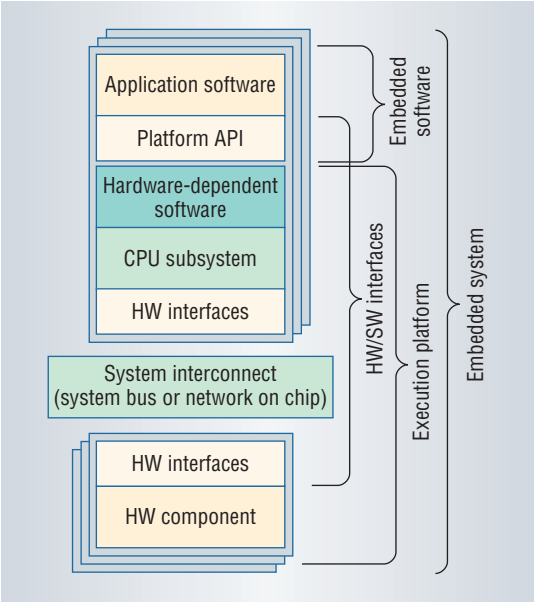


Figure 2. Embedded system architecture. In addition to hardware, a SoC includes classic application software and hardware-dependent software that must be codesigned with hardware interfaces.

LINKING INTERFACES TO EMBEDDED SOFTWARE

Some designers use the term "embedded software" to designate any software in an embedded system, while others use it to mean only that part of the software that is intimately related to hardware—for example, the hardware team generally designs low-level software functions such as drivers and interrupt management.

The generic architecture shown in Figure 2 helps to clarify the relationship between hardware and software. An embedded system is an application-specific HW/SW architecture. Ideally, the application is a body of software to be executed on a hardware platform. The SoC platform itself also includes, in addition to hardware, a software layer called *hardware-dependent software* that must be codesigned with hardware interfaces.

From the software application point of view, the reaction time is measured in milliseconds; at that execution rate, the platform can be abstracted as an application programming interface or programming model. The API hides hardware details such as interrupt controllers or memory and I/O systems. Software designers develop the application software and use real-time techniques to validate the software application properties.[6]

There is a temptation to continue using the traditional software- or hardware-only approach to implement large applications.

The hardware-dependent software supports the API, adapting it to the CPU subsystem. The CPU subsystem can hide the complex architecture it generally requires to meet performance demands. In addition to the classical CPU, the subsystem can include sophisticated I/O and memory subsystems.

The SoC can include several heterogeneous subsystems, including specific hardware components and sophisticated interconnects. When the subsystems and interconnect designs are decoupled, hardware interfaces are required to adapt them in the final SoC. Both application software and hardware-dependent software may be distributed over different subsystems.

To design the application software, classical real-time software designers can use specific analysis tools that support complexity. The hardware-dependent software adapts the application software to a CPU subsystem. In general-purpose computer systems, this layer can use standard components, such as an operating system or middleware, that can be ported to different hardware platforms. When applied to a SoC, however, this solution induces significant overhead in code size, runtime, energy consumption, and other system costs.

Two factors cause this overhead. The operating system and middleware must be ported from a uniprocessor platform to heterogeneous multiprocessor platforms. The systems also must implement full-featured functionality to support various types of embedded software.

A similar distinction exists on the hardware side, where part of the design depends on software and must be isolated.

HARDWARE/SOFTWARE INTERFACE CODESIGN

High-performance embedded systems consist of multiple HW/SW subsystems, with application software tasks distributed over heterogeneous processor subsystems using sophisticated interconnects. The HW/SW interface and the CPU subsystems must handle the interaction between software tasks and the interconnect structure. The interface provides the application software layer with an abstraction of the SoC architecture, called a *parallel programming model*. It also includes a network interface for both multiprocessor booting and interprocessor communication that connects the subsystem to the network.

When the SoC includes more than one CPU, HW/SW interface design becomes more complicated. Parallel programming models are more complex than uniprocessor programming models; similarly, network interfaces are more complex than a unified memory. Thus, as a recent multiprocessor SoC case study confirms,[3] the HW/SW interface could become a key challenge in heterogeneous SoC design.

Bridging the gap

Because design teams traditionally have applied a software- or hardware-only strategy, there is a temptation to continue using this approach to implement large applications. Software teams claim that their approach results in a shorter design cycle. For example, a pure software approach may reduce the design cycle for derivative design because software is flexible enough to add new functionality. On the other hand, hardware teams argue that their approach is more efficient. While an embedded software approach could result in a larger chip or even a chipset, the ASIC approach will yield a smaller chip.

Even for a single product, achieving the best volume in a given market window considering chip-size and yield in chip production may require combining hardware and software solutions. In terms of yield in chip production, both ASIC and embedded software approaches have pros and cons. The ASIC approach can suffer from low yield in the first few months of chip production until the learning curve improves. However, the reduced chip size may improve total chip production. An embedded software approach can give a good initial yield since it reuses an already proven SoC platform. However, a larger chip size may reduce the effects of yield improvement.

Ultimately, achieving optimal SoC production will require some combination of hardware and software solutions.

Figure 3 shows a simplified flow of concurrent HW/SW design. This codesign scheme opens the design process to several optimizations that are not possible using the classic approach in which hardware and software are designed separately.

The most obvious improvement is better adaptation of the CPU to both hardware and software interfaces. For example, designers can use new flexible processor technologies such as Tensilica[7] to optimize performance at the HW/SW interface by introducing application-specific I/O operation. In addition, using reconfigurable hardware, such as the Xilinx Virtex II Pro, can optimize hardware interfaces to an embedded CPU.

Interface codesign roadmap

The complexity of the HW/SW codesign process will depend on the abstraction level at which the

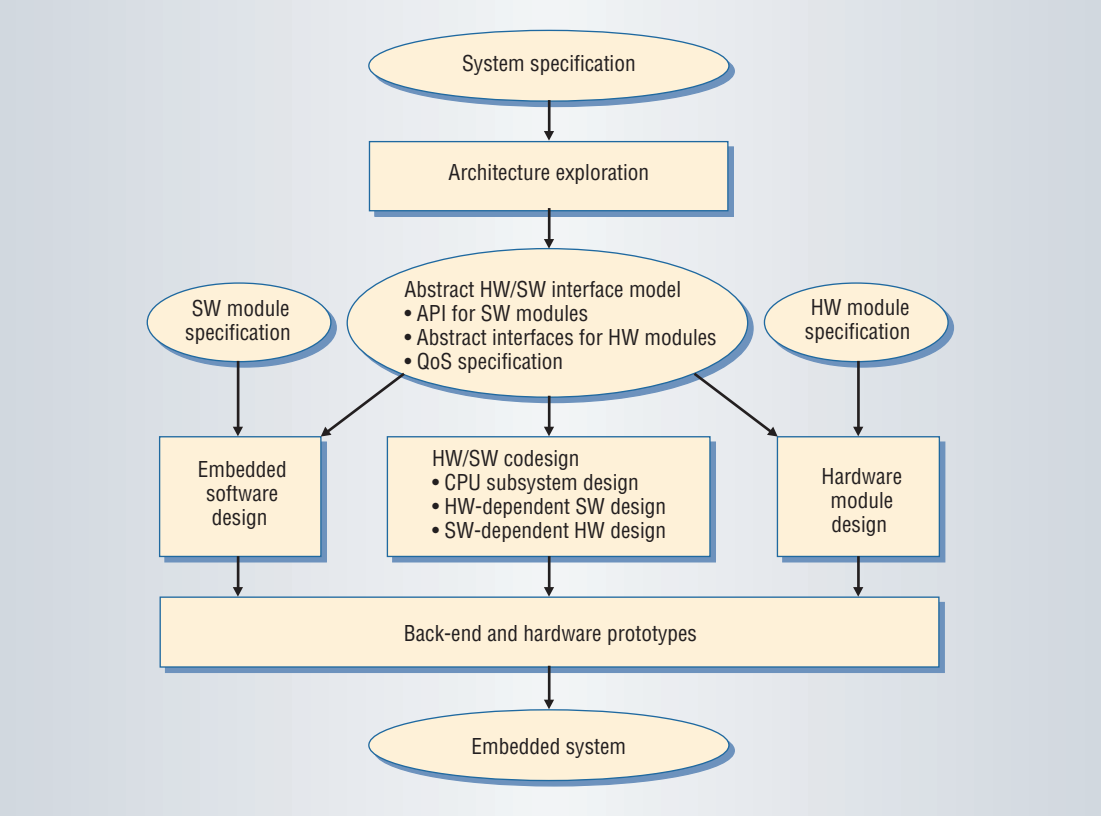


Figure 3. HW/SW interface codesign flow. This scheme enables optimizations that cannot be achieved using separate hardware- and software-only approaches.

process starts. Researchers[2,8,9] have clearly identified five abstraction levels that will constitute key milestones for future HW/SW codesign automation.

Explicit interfaces. The currently used model for SoC design describes hardware as RTL modules. The CPU acts as the HW/SW interface, and designers use explicit memory and I/O architectures to detail the software down to assembly code or low-level C programs.

Data transfer. At this level, the CPU is abstract. Hardware and software modules interact by exchanging transactions through an explicit interconnect structure, a model generally referred to as *transaction-level modeling*. Among the various TLM languages, most were developed using SystemC.[4] In addition to designing interfaces for different hardware modules, refining a TLM model requires designing a CPU subsystem for each software subsystem.

Synchronization. At this level, the interconnect and synchronization are abstractions. The hardware and software modules interact by exchanging data following well-defined communication protocols. The Message Passing Interface (MPI)[8] is an example of this approach. Refining an abstract HW/SW interface model requires first designing the interconnect—a system bus or NoC—and then correcting the synchronization schemes. Data transfer must also be refined down to the RTL.

Communication. At this level, the communication protocol is abstract. The hardware and software modules interact by exchanging abstract data without regard to the protocol used or the synchronization and interconnect the design will implement. The design typically uses the Specification and Description Language[8] to abstract communication. Refining an SDL model requires first selecting a communication protocol—for example, message passing or shared memory—and then following the refinement steps used in lower abstraction levels.

Partitioning. The ultimate abstraction level is the functional model in which hardware and software are not partitioned. Designers can use a variety of models to abstract HW/SW partitioning, including sequential programming languages such as C/C++, concurrent languages, and higher-level models such as algebraic notation—for example, the B language. Refining such a model requires first separating the software and hardware functions and then performing the refinements used in higher abstraction levels.

Toward full codesign

The ultimate goal is to design both hardware and software at all abstraction levels. Figure 4 details one such full codesign scheme. Traditional codesign research has concentrated on HW/SW partitioning, but without solving the problem of abstracting the hardware platform. Rather than using ad hoc hardware models, SoC designs demand a well-thought-out approach to the HW/SW interface. The next steps in automation would be data transfer synthesis, synchronization and interconnect, communication, and HW/SW partitioning.

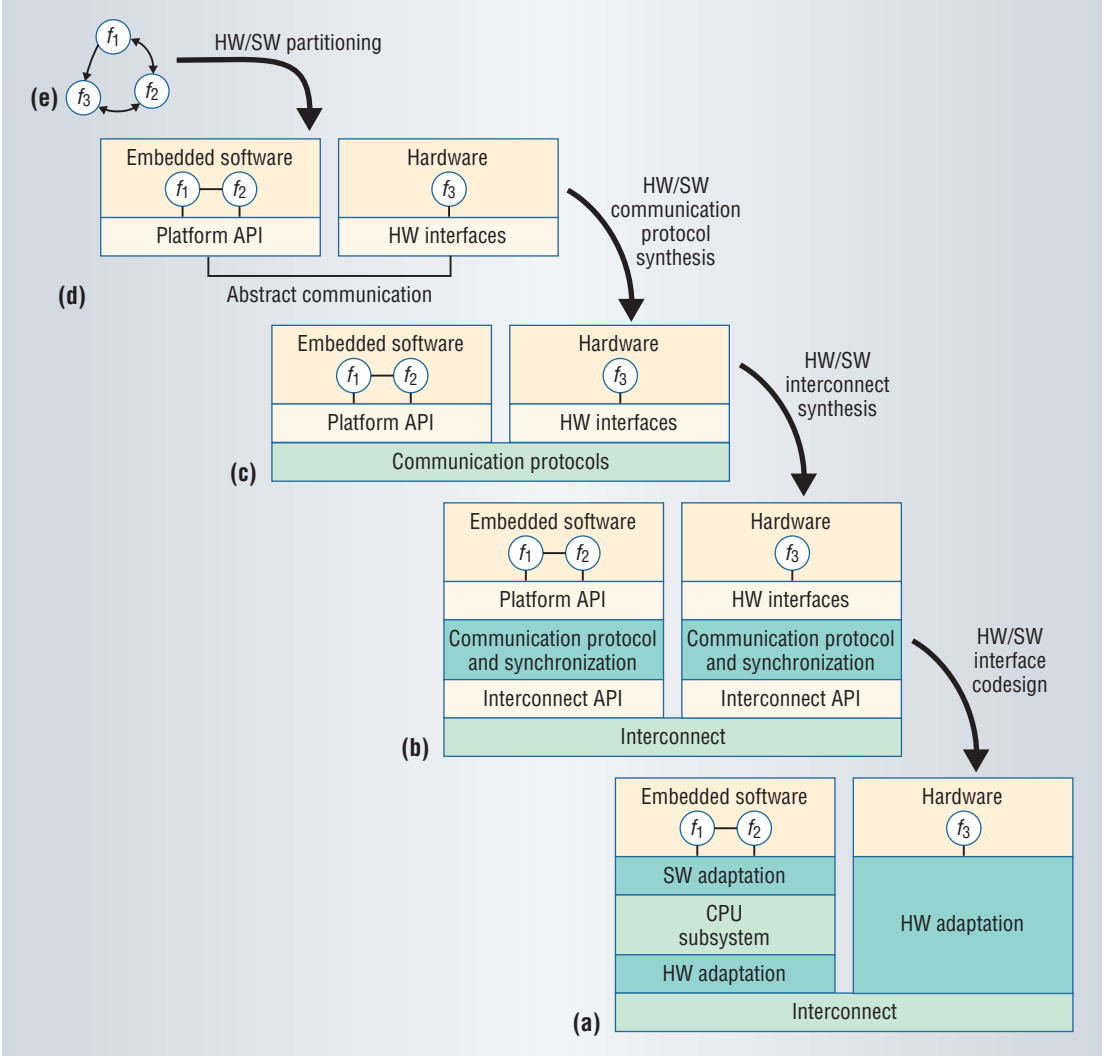


Figure 4. Full HW/SW interface codesign scheme: (a) explicit interfaces, (b) data transfer, (c) synchronization, (d) communication, and (e) partitioning.

OPPORTUNITIES AND CHALLENGES

Successful HW/SW interface codesign will fundamentally improve the SoC design process by increasing both hardware and software quality and reliability and by enabling early verification, reusability, and interoperability. It will also provide opportunities for tackling a number of technical challenges confronting embedded-system designers.

Improved software quality

Embedded software relies on the hardware platform to support complex quality-of-service (QoS) requirements and ensure reliability. Current practice is to use an existing OS or middleware to validate the nonfunctional properties of application software. Because these generally support real-time and delay requirements but not nonfunctional properties such as intersubsystem communication, bandwidth, jitter, and reliable communication, they cannot systematically monitor and guarantee QoS. In this scenario, it is even difficult to guarantee the reliability of the HW/SW interface design itself. Overcoming this challenge requires a QoS-aware HW/SW interface abstraction.

Early verification

Verifying the interface independent of its context is not sufficient—the interface must be verified relative to a given hardware platform. It is not possible to delay performing this verification until the hardware prototype is available. Abstracting the HW/SW interface model will make it possible to verify the interface abstract design itself without using the physical prototype.

Reusability

Mastering embedded system design requires using an efficient method to configure and optimize the HW/SW interface. Using a general HW/SW interface model makes it possible to reuse application software, hardware components, and platform and middleware modules across different products, product families, and even application domains. However, a drawback of generality is inefficiency. For applications that require only a small subset of the complete HW/SW interface functionality, a generic model imposes tremendous overhead that cost-sensitive applications cannot tolerate. A highly configurable and parameterized abstract interface

architecture enables designers to optimize and streamline an instance of the interface to a given application's particular needs.

Interoperability

Creating abstract HW/SW interfaces facilitates dialogue between design teams that can belong to different companies or even market sectors.[10] For example, an automaker could use a standard interface HW/SW API to develop the car's software while reserving the right to select the hardware platform as late as possible.

The key issue when integrating the parts of an embedded system is the creation of a continuum between the hardware and software, which requires new technologies to effectively integrate components. For example, most conventional parallel programming models—including MPI, the open specifications for multiprocessing (OpenMP), the bulk synchronous parallel (BSP) model, and LogP—are designed for general-purpose computing. SoC APIs must specify application-specific design constraints—for example, in terms of energy consumption, runtime, cost, and reliability.

In addition, abstract HW/SW interface models are widely available for single-processor subsystems and homogeneous multiprocessors, but SoCs involve complex interactions between heterogeneous subsystems. Abstracting multiprocessor platforms require a scalable, configurable interface architecture.

Embedded computing applications often combine several different kinds of algorithms. Specializing cores by operation type would provide substantial savings in cost and power consumption. Embedded applications also show wide variations in data loads during execution. Flexible networks would allow using interconnect resources more efficiently. Finally, using reconfigurable fabrics as embedded system components would make it possible to target a platform to far more products. ■

Acknowledgments

The authors thank all the members of TIMA's System-Level Synthesis Group, especially Sungjoo Yoo, Wander Cesário, and Xi Chen, for their help in preparing this article.

References

1. H. Jones, "Analysis of the Relationship between EDA Expenditures and Competitive Positioning of IC Vendors for 2003," Int'l Business Strategies, 2002; www.edac.org/downloads/04_05_28_IBS_Report.pdf.
2. A.A. Jerraya and W. Wolf, *Multiprocessor Systems-on-Chips,* Morgan Kaufmann, 2004.
3. H. Iwasaki et al., "Single-Chip MPEG-2 422P@HL CODEC LSI with Multi-Chip Configuration for Large-Scale Processing beyond HDTV Level," *Proc. Design, Automation and Test in Europe Conf. and Exhibition* (DATE 03 Designers' Forum), IEEE CS Press, 2003, p. 20,002.
4. SystemC Transaction Level Modeling Working Group, www.systemc.org/projects/tlm.
5. M-W. Youssef et al., "Debugging HW/SW Interface for MPSoC: Video Encoder System Design Case Study," *Proc. 41st Design Automation Conf.* (DAC 04), IEEE CS Press, 2004, pp. 909-913.
6. J.W.S. Liu, *Real-Time Systems,* Prentice Hall, 2000.
7. C. Rowen, *Engineering the Complex SoC: Fast, Flexible Design with Configurable Processors,* Prentice Hall, 2004.
8. D.B. Skillicorn and D. Talia, "Models and Languages for Parallel Computation," *ACM Computing Surveys,* vol. 30, no. 2, 1998, pp. 123-169.
9. W. Wolf, *Computers as Components: Principles of Embedded Computing System Design,* Morgan Kaufmann, 2001.
10. S. Yoo and A.A. Jerraya, "Introduction to Hardware Abstraction Layers for SoC," *Proc. Design, Automation and Test in Europe Conf. and Exhibition* (DATE 03), IEEE CS Press, 2003, pp. 10,336-10,337; http://sigda.org/Archives/ProceedingArchives/Date/papers/2003/date03/pdffiles/04e_1.pdf.

Ahmed A. Jerraya *leads the System-Level Synthesis Group at TIMA (Techniques of Informatics and Microelectronics for Computer Architecture) Laboratory in Grenoble, France. His research interests include embedded computing, flexible multiprocessor SoCs, hardware-dependent software, and computer-aided design. Jerraya received a PhD in computer sciences from the University of Grenoble. He is a member of the IEEE and the ACM and is a board member of the European Design Automation Association. Contact him at ahmed.jerraya@imag.fr.*

Wayne Wolf *is a professor in the Department of Electrical Engineering at Princeton University. His research interests include embedded computing, multimedia systems, VLSI, and computer-aided design. Wolf received a PhD in electrical engineering from Stanford University. He is a Fellow of the IEEE and the ACM. Contact him at wolf@princeton.edu.*