COVER FEATURE

A Decade of Hardware/Software Codesign

Hardware/software codesign has been a recognized research field for about a decade. Within that time, it has moved from an emerging discipline to a mainstream technology.

Wayne Wolf
Princeton University

The term hardware/software codesign surfaced in the early 1990s to describe a confluence of problems in integrated circuit (IC) design. Microprocessors had been in use for over a decade at that point, but microprocessor-based systems were almost exclusively board-level systems. A class of designers who were largely separate from IC designers integrated microprocessors with standard hardware components on a board. Much of the code was in assembly language.

By the 1990s, it was clear that microprocessor-based system design would become an important design discipline for IC designers as well. Large 16-bit and 32-bit microprocessors had already been used in board-level designs, and it was apparent that Moore's law would eventually lead to chips that were large enough to include both a CPU and other subsystems. This raised two classes of problems: System design methodologies would have to handle large predesigned CPUs, and software would have to be treated as a first-class component in chip design.

Researchers developed some basic approaches to the design of embedded software running on CPUs, and their work formed the roots of a hardware/software codesign methodology. As the "The Impetus for Codesign" sidebar explains, hardware/software codesign tries to increase the predictability of embedded system design by providing *analysis* methods that tell designers if a system meets its performance, power, and size goals and *synthesis* methods that let researchers and designers rapidly evaluate many potential design methodologies.

After a decade of research, hardware/software codesign has a rich literature that is impossible to survey exhaustively in one article. Thus, this short recap merely introduces some of the decade's major research themes.

FIRST STEPS

One of the earliest codesign efforts was the SOS system from Prakash and Parker of the University of Southern California,[1] which could synthesize an arbitrary multiprocessor topology and schedule and allocate processes onto the multiprocessor. The system formulated the synthesis problem as a mixed integer-linear program, so it was slow and could not handle large problems, but it was important foundational work.

About a year later, the CODES workshop in Colorado and the CASHE workshop in Austria introduced several pieces of significant research that had evolved in parallel. From these, hardware/software partitioning emerged as an important first step in creating models and algorithms. Two early systems, Vulcan from Stanford[2] and Cosyma from the Technical University of Braunschweig,[3] took complementary approaches to this basic problem.

As Figure 1 shows, hardware/software partitioning maps a design onto the target architecture. A system includes a single CPU and one or more application-specific ICs connected by a bus.

In the early designs, the ASIC acted as an accelerator[4] rather than as a coprocessor in that the CPU's execution unit did not dispatch it. Designers gener-

Published by the IEEE Computer Society
0018-9162/03/$17.00 © 2003 IEEE

ally assumed that the CPU and ASIC were to be on separate chips, although this was not an essential assumption. The CPU and ASIC communicated by shared memory or registers. This architecture let the system allocate computationally intensive tasks to the ASIC, while allocating work less suited to direct hardware implementation to the CPU.

The input to both Vulcan and Cosyma was a C-like program. Based on an analysis of the performance and cost of various implementations of the program, some of the program's functions were put in the ASIC while other parts were implemented in software running on the CPU. Vulcan and Cosyma designers took a very different route to the end product, however. Vulcan initially put all the functionality into hardware and moved operations to the CPU to minimize cost; Cosyma started with all operations on the CPU and migrated operations to the ASIC to meet the performance goals.

Designers of both systems had to analyze performance along three dimensions: hardware, software, and system. Of these, hardware performance analysis had the most available infrastructure and so was the easiest. The goal was to determine the hardware unit's maximum clock frequency, but the analysis had to be quick so that the designer could evaluate many designs during synthesis. The solution was to use high-level synthesis techniques to estimate the longest path through the logic.

Software performance analysis presented more of a quandary. The problem to be solved was similar in formulation to a well-known hardware problem—worst-case execution time—but few researchers had studied this aspect of software performance. One of the few solutions proposed by the early 1990s was Chang-Yun Park and Alan Shaw's path-enumeration algorithm.[5] Unfortunately, the codesign community was not aware of this work, and Cosyma ended up estimating software performance by running test cases on the target processor, while Vulcan analyzed the program's control dataflow.

System performance analysis was also complex. In general, a CPU-ASIC system is both a multiprogramming and a multiprocessing system: It includes multiple processes that the designer can interleave on the CPU; it also includes multiple processing elements so that multiple processes can run simultaneously.

Cosyma and Vulcan both used simplified computational models to make this problem more tractable. Both assumed that the implementation was single-threaded—the CPU sat in an idle loop while the ASIC performed its function. Thus, designers could determine total system execution time from the order in which the system executed processes.

In another significant early paper, Asawaree Kalavade and Edward A. Lee[6] presented a codesign methodology that emphasized the problems in multirate signal-processing systems and the roles of simulation, analysis, and synthesis.

The Impetus for Codesign

Board-level systems had used microprocessors, even complex microprocessors, for at least a decade before hardware/software codesign emerged as a discipline. Around 1980, for example, automotive engine controllers appeared—microprocessors that used fairly sophisticated algorithms to control emissions and fuel economy.

It is not entirely clear why it took so long for embedded system design to emerge as an academic discipline. Certainly, moving the locus of CPU-based designs from boards to chips gave embedded microprocessors added cachet as an intellectual problem. The cost of design mistakes is also much higher in chips than on boards.

Perhaps the main rationale for putting microprocessors into ICs is that it made the design of complex digital systems predictable. Embedded CPUs let designers separate a complex digital design problem into two subproblems: design of the embedded CPU hardware and design of the software running on the CPU. CPU designs can be encapsulated as intellectual property, and CPUs are by far the most successfully reused hardware in the world. Software can also, to some extent, be reused from design to design.

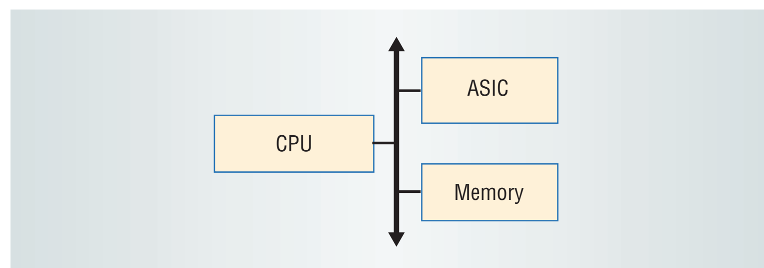

Figure 1. Target architecture for hardware/software partitioning. The system includes a CPU and application-specific IC (ASIC).

MATURATION

Hardware/software codesign rapidly took off after the first CODES and CASHE workshops. In the next few years, researchers tackled several problems, and their work made it possible to apply codesign to a wider range of systems.

Early work also dealt with cosimulation and soon recognized it as an essential component of a codesign methodology. The challenge is to perform cosimulation at mixed abstraction levels to execute enough input vectors to validate the design. With mixed-level simulation, designers can trade off simulation performance for accuracy by choosing the detail level at which to simulate various system components.

In 1992, Becker, Singh, and Tell[7] developed a cosimulator that linked a hardware simulator to executions of application software. The Ptolemy

Making the best use of a platform FPGA requires identifying an application that maps well onto it.

environment[8] was another early system for simulating signal processing and heterogeneous hardware/software systems. This early work in cosimulation stimulated later work to understand the relationships among various computational models and their unifying frameworks.

The worst-case execution time problem for software received a great deal more attention as codesign became more established. Li, Malik, and Wolfe[9] developed an implicit path-analysis algorithm that was more efficient than Park and Shaw's path-enumeration method. Li and coworkers subsequently extended their algorithm to model the effect of instruction caches.

Other researchers revisited the system-performance problem, adopting models developed for real-time systems as a partial solution. Rate-monotonic scheduling[10] received some attention as a way to analyze the performance of a set of processes on a single CPU. Yen and Wolf[11] developed a multiprocessor performance algorithm that analyzed the performance of a set of processes (including data dependencies) mapped onto a network of processors, with each processor running a rate-monotonic scheduler.

Hardware cost estimation also received a great deal more attention. Frank Vahid and Daniel Gajski[12] used incremental hardware cost estimation to reduce the computational cost of analyzing hardware performance.

As hardware/software codesign matured, researchers began to explore various computational models for embedded systems.[13,14] C had the advantage of being widely used but was not ideal for specifying concurrent systems. On the control side, M. Chioda and colleagues built on the Esterel model to develop the codesign finite state machine (CFSM) model,[15] which describes concurrent, communicating processes. On the data side, the synchronous dataflow model evolved to describe multirate data-oriented computations, such as those in signal processing.

Other researchers developed methods to target more general architectures. In 1997, Kalavade and Lee extended their early work to handle general architectures.[16] Also at that time, Wolf developed a synthesis method that could handle arbitrary interconnection topologies and arbitrary combinations of CPUs and ASICs.[17]

Low-power design became a dominant theme during the 1990s, which prompted research into techniques for low-power cosynthesis. Fornaciari and colleagues[18] developed a modeling system to estimate the power consumption of an embedded system during cosynthesis.

Once the architectural methods had matured, research turned to system implementation issues, one of which was interface generation. Daveau and colleagues[19] developed models and algorithms for implementing interface protocols.

A challenge in interface generation was how to synthesize software to run on the embedded system because software structure could greatly influence the system's performance and power consumption. Much effort went into synthesizing CFSM models, for example. On the data side, Bhattacharyya and colleagues[20] described methods for generating efficient code from dataflow models.

Work also continued on cosimulation. For example, Zivojnovic and Meyr[21] developed methods for compiled cosimulation.

MOVING INTO THE MAINSTREAM

Hardware/software partitioning is now a practical design task, thanks to reconfigurable computing. Several manufacturers have announced platform field-programmable gate arrays (FPGAs) that combine a programmable logic fabric with one or more CPUs. Designers implement the CPUs on these chips separately from the programmable logic.

The platform FPGA seems to be the chip for which cosynthesis was created: The chip's internal architecture is exactly what hardware/software partitioning targets. Researchers must solve several problems, however, before cosynthesis becomes a commonplace tool for platform FPGA design.

Making the best use of a platform FPGA requires identifying an application that maps well onto it. A key problem in CPU/ASIC architectures is the communication between the CPU and the ASIC. Several sources of delay can nullify any performance gains achieved with the ASIC: physical communication delays, synchronization delays, and so on. A good application for this style of architecture has operations that can be moved to the ASIC with relatively small communication cost and that designers can easily overlap with useful work on the CPU.

Research must also look at creating interfaces for both the FPGA fabric and CPU sides of the system. On the CPU side, drivers are required to turn software operations into peeks and pokes on the hardware. On the FPGA fabric side, interfaces to the system bus must be built. The FPGA fabric and CPU can communicate directly by shared memory.

There is also still some debate as to what language is best for describing the input to hard-

ware/software partitioning algorithms. Software languages like C bias the implementation in favor of software, while hardware languages like Verilog bias results toward hardware. An alternative might be to describe the system in two languages—describe some obvious hardware functions in a hardware description language and describe the rest of the functions in a software language. Consequently, when operations are moved across the partition, only a relatively small part of the total specification must be translated.

The system on chip is another venue in which codesign is increasingly important. Because SoCs do not have a fixed architecture, a variety of algorithms for analyzing and synthesizing general architectures are important to SoC cosynthesis. SoC design is IP-oriented, so designers can use CPUs, predesigned special-purpose logic, and even FPGA fabrics as components. Cosynthesis for SoCs involves determining how best to use large IP blocks without requiring designers to write their descriptions directly in terms of these blocks. The design space is also very large and irregular, making design-space exploration more challenging.

One approach to SoC design that has become popular in the past few years is *platform-based design*. A platform is a predesigned architecture that designers can use to build systems for a given range of applications. A platform FPGA is an example of a platform, but a platform also can be any architecture built from CPUs, custom logic, and interconnection hardware.

In many ways, platform-based design is the antithesis of codesign, since designers use the platform largely as is. However, the platform must at some point be able to accommodate the desired range of applications. Codesign is an ideal way to explore the design space and to create a suitable platform architecture.

OPEN PROBLEMS

Current embedded computing systems are far more sophisticated than a decade ago. Many systems such as automotive engine controllers or personal digital assistants use a large 32-bit CPU. Others use several large CPUs. Laser printers, for example, typically use multiple processors; high-end cameras use a 32-bit CPU and other processors; a digital set-top box uses multiple processors. The Viper[22]—just one of many modern complex embedded systems—includes two processors, a MIPS 32-bit CPU, and a TriMedia very large instruction word (VLIW) processor, plus three buses and a variety of units with special-purpose functions.

Several long-standing problems remain. Researchers are still working to define and redefine computational models for jointly describing hardware and software systems. System-level performance analysis is a complex problem that analysts must study under a variety of operating conditions suitable for various application types. And research continues to evaluate algorithms for design-space exploration—work that includes applying genetic algorithms and other advanced methods to codesign.

Codesign is an ideal way to explore the design space and to create a suitable platform architecture.

Memory systems continue to be the subject of research,[23] since their design profoundly influences the system's performance as well as its energy consumption. Cache models are one aspect of particular importance in understanding memory systems. The better the cache model, the easier it is to predict how changes to hardware or software will influence system performance and power.

Software optimizations let designers implement programs the best way possible on the available cache. With cache synthesis, they can choose a cache configuration for a particular application. Many researchers are also starting to think about alternatives to traditional caches. The scratch-pad memory, for example, is a software-managed small memory that provides fast access to data without the burden of working around a fixed cache-management policy.

Emerging problems are also likely to keep researchers busy for quite some time. Besides being a theoretical topic, the search for computational models has recently extended to new modeling languages. SystemC (www.systemc.org) and SpecC (www.specc.gr.jp/eng/index.html) have emerged as system-level design languages. The design of languages like these must consider computational models, system design methodologies, simulation, language acceptance, and many other factors.

Researchers are developing methods to analyze new classes of architectures that are starting to become common in embedded systems. VLIW processors, for example, have become popular for signal processing and networking applications. Efforts are under way to develop new methods for performance analysis and code generation with the aim of making VLIW-based architectures more useful. FPGAs are another example of an architectural element that needs more study as a medium for implementing embedded systems.

System-level power management[24] is well suited to codesign because designers can use the application's characteristics to optimize the management

strategy and its implementation in hardware and software. Given the prime importance of power management in digital systems, we can expect to see even more work in this area in the future.

Another emerging problem is how to evaluate the effect of networks on chips [25] on codesign. On the one hand, NoCs provide a more structured system that should be easier to analyze. On the other, NoCs are themselves complex systems that are not trivial to analyze for performance or power, so adding them to an architecture makes it that much harder to analyze.

An increasing number of embedded systems connect to the Internet, which imposes new workloads and new mixtures of hard and soft deadlines. Synthesis techniques targeted to Internet-enabled machines may be necessary to get sufficiently good results. Internet-enabled applications will also increase the demand for self-organizing systems that can adapt to changes in the environment, device failures, the addition of new devices, and other changes.

Finally, as VLSI systems become more complex, the role of codesign will expand to include systems built of many SoCs. Many applications for embedded computers require physically distributed computation to deal with fast response rates. The automobile is a prime example of a physically distributed computing platform. Designers must create all the chips in these systems together to ensure that they jointly satisfy the application's performance requirements.

Hardware/software codesign is informed by multiple disciplines. Computer architecture tells us about the performance and energy consumption of single CPUs and of multiprocessors. Real-time system theory helps us to analyze the deadline-driven performance of embedded systems; computer-aided design helps in evaluating the hardware cost as well as methods for exploring the design space. Knowledge about all these disciplines has helped transform hardware/software codesign from an art to a science. As ICs become more complex, the technical challenges in codesign will also increase, making this a vibrant field for a long time to come. ■

References

1. S. Prakash and A.C. Parker, "SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems," *J. Parallel and Distributed Computing*, vol. 16, 1992, pp. 338-351.
2. R.K. Gupta and G. De Micheli, "Hardware/Software Cosynthesis for Digital Systems," *IEEE Design & Test of Computers*, Sept. 1993, pp. 29-41.
3. R. Ernst, J. Henkel, and T. Benner, "Hardware/Software Cosynthesis for Microcontrollers," *IEEE Design & Test of Computers*, Dec. 1993, pp. 64-75.
4. W. Wolf, *Computers as Components*, Morgan Kaufmann, 2000.
5. C-Y. Park and A.C. Shaw, "Experiments with a Program Timing Tool Based on a Source-Level Timing Scheme," *Computer*, May 1991, pp. 48-57.
6. A. Kalavade and E.A. Lee, "A Hardware/Software Codesign Methodology for DSP Applications," *IEEE Design & Test of Computers*, Sept. 1993, pp. 16-28.
7. D. Becker, R.K. Singh, and S.G. Tell, "An Engineering Environment for Hardware/Software Cosimulation," *Proc. 29th Design Automation Conf.*, IEEE CS Press, 1992, pp. 129-134.
8. J. Buck et al., "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," *Int'l J. Computer Simulation*, Apr. 1994, pp. 155-182.
9. Y-T. Li, S. Malik, and A. Wolfe, "Performance Estimation of Embedded Software with Instruction Cache Modeling," *Proc. Int'l Conf. Computer-Aided Design*, IEEE CS Press, 1995, pp. 380-387.
10. C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, Jan. 1973, pp. 46-61.
11. T-Y. Yen and W. Wolf, "Performance Estimation for Real-Time Distributed Embedded Systems," *IEEE Trans. Parallel and Distributed Systems*, Nov. 1998, pp. 1125-1136.
12. F. Vahid and D.D. Gajski, "Incremental Hardware Estimation During Hardware/Software Functional Partitioning," *IEEE Trans. VLSI Systems*, Sept. 1995, pp. 459-464.
13. S. Edwards et al., "Design of Embedded Systems: Formal Models, Validation, and Synthesis," *Proc. IEEE*, May 1995, pp. 773-799.
14. A. Benveniste and G. Berry, "The Synchronous Approach to Reactive Real-Time Systems," *Proc. IEEE*, Sept. 1991, pp. 1270-1282.
15. M. Chiodo et al., "Hardware/Software Codesign of Embedded Systems," *IEEE Micro*, Aug. 1994, pp. 26-36.
16. A. Kalavade and E.A. Lee, "The Extended Partitioning Problem: Hardware/Software Mapping, Scheduling, and Implementation-Bin Selection," *Design Automation for Embedded Systems*, Mar. 1997, pp. 125-163.
17. W. Wolf, "An Architectural Cosynthesis Algorithm for Distributed, Embedded Computing Systems," *IEEE Trans. VLSI Systems*, June 1997, pp. 218-229.
18. W. Fornaciari et al., "Power Estimation of Embed-

ded Systems: A Hardware/Software Codesign Approach," *IEEE Trans. VLSI Systems*, June 1998, pp. 266-275.
19. J.M. Daveau et al., "Protocol Selection and Interface Generation for HW-SW Codesign," *IEEE Trans. VLSI Systems*, Mar. 1997, pp. 136-144.
20. S.S. Bhattacharyya et al., "Generating Compact Code from the Dataflow Specification of Multirate Signal Processing Algorithms," *IEEE Trans. Circuits and Systems*, Mar. 1995, pp. 138-150.
21. V. Zivojnovic and H. Meyr, "Compiled HW-SW Cosimulation," *Proc. 33rd Design Automation Conf.*, ACM Press, 1996, pp. 690-695.
22. S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A Multiprocessor SoC for Advanced Set-Top Box and Digital TV Systems," *IEEE Design & Test of Computers*, Sept.-Oct. 2001, pp. 21-31.
23. S. Wuytack et al., "Memory Management for Embedded Network Applications," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, May 1999, pp. 533-544.
24. L. Benini, A. Bogliolo, and G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. VLSI Systems*, June 2000, pp. 299-316.
25. L. Benini and G. De Micheli, "Networks-on-Chips: A New SoC Paradigm," *Computer*, Jan. 2002, pp. 70-78.

Wayne Wolf *is a professor of electrical engineering at Princeton University. His research interests include embedded computing, multimedia, VLSI CAD, and the study of codesign problems. Wolf received a PhD in electrical engineering from Stanford University. He is a member of the IEEE Computer Society and a member of* Computer*'s editorial board. Contact him at wolf@ee.princeton.edu.*

IEEE 5th Int. Symposium on Multimedia Software Engineering (MSE2003)

December 10-12, 2003, Taichung, Taiwan, ROC

http://mse2003.src.ncu.edu.tw/ and http://mse2003.ece.uci.edu/

Sponsored by IEEE Technical Committee on Multimedia Computing

The Fifth International Symposium on Multimedia Software Engineering (MSE2003) is an international forum for multimedia and software engineering researchers to exchange information regarding advancements in the state of the art and practice of multimedia software engineering, as well as to identify the emerging research topics and define the future of multimedia research. The technical program of MSE2003 will consist of invited talks, paper presentations, and panel discussion. MSE2003 will also include a few special tracks dedicated to focus areas. Submission of high quality papers and special tracks describing mature results or on-going work are invited. Topics for submission include but are not limited to:

- Multimedia system architecture & specification lang
- Multimedia streaming, networking, and QoS
- Multimedia meta-modeling techniques and OS
- Signal processing including audio, video, image processing, and coding
- Multimedia software development techniques
- Multimedia tools
- Multimedia user interfaces and interaction models
- Multimedia file systems, databases, and retrieval
- Multimedia collaboration
- Multimedia enabled e-learning
- Rich media enabled E-commerce
- Pervasive, interactive multimedia systems
- Computational intelligence
- Intelligent agents for multimedia content creation, distribution, and analysis
- Internet telephony and hypermedia
- Multimedia security and watermark
- Multimedia systems and applications in education, entertainment, and design
- Distributed multimedia systems and applications
- Bioinformatics

Submissions must be received no later than **May 20, 2003**. The best papers presented in the symposium will be chosen for a special issue in a selected journal. For details, please contact **mse2003@src.ncu.edu.tw**

General Co-Chairs: Wen-Tsuen Chen, *NTHU*, Jeffrey J.P. Tsai, *UIC,* Phillip C-Y Sheu, *UCI*
Program Co-Chairs: Jen-Yao Chung, *IBM*, Stephen J.H. Yang, *NCU*, Rong-Ming Chen, *THMU*