2 Extending CAD tools and techniques

Giovanni De Micheli, Stanford University

Most digital systems consist of a hardware component and software programs that execute on the hardware platform. Obviously, these systems can deliver higher performance when the hardware design is tuned to its software applications and vice versa. Computer designers have exploited the synergism between hardware and software for many years, for example, in defining hardware architectural support for operating systems.

Renewed interest in hardware-software codesign can be traced to progress in computer-aided design. We now understand many hardware CAD problems and have design tools to solve them. The emphasis in CAD has risen from the physical-design to the logic-synthesis level, where hardware models in register-transfer languages (for example, Verilog, VHDL, UDL/I) are compiled into logic circuits and optimized. High-level synthesis techniques support the design of hardware from behavioral descriptions, whose abstractions are similar to those used in software programs. Hence, design tools may soon be able to handle some aspects of the concurrent design of hardware and software modules.

Problem taxonomy. To be realistic about the possibility of supporting codesign with CAD tools, we must first make a coarse-grained taxonomy of the problem. Codesign is part of system-level design, where a system is a physical unit that can deliver a service (for example, a computer workstation or a manufacturing robot). System-level design requires solving mechanical, electrical, and software problems.

There are three major classes of system design: (1) large systems that include several computing units and broad software functions, such as an aircraft or a continental telephone network; (2) general-purpose computing systems that include several layers of software in the operating system; and (3) embedded systems that have processors dedicated to specific functions and different degrees of programmability.

The problems of designing large systems include reliability and availability, which are often achieved through redundancy. Another problem, verification of correct system operation under different environmental conditions, is addressed mainly through software management techniques. Computer-aided software engineering (CASE) tools are currently more applicable to these problems than CAD programs are.

In general-purpose computing systems, two codesign problems have been investigated extensively: caches and pipelines. The design and sizing of memory caches require a match between circuit performance and the updating algorithm and its parameters. Most cache designs are based on validating the design assumptions through simulation with specialized tools.

The design and control of processor pipelines require removing pipeline hazards. Either hardware or software techniques can solve the problem. In the former case, the pipeline control unit can flush the pipe; in the latter, the compilers can reorder the instructions or insert "no operations." The choice affects overall performance, and estimating performance is not trivial. It requires appropriate models for both hardware and software. Computing the most effective number of pipe stages for a given architecture is

(Continued at the bottom of p. 86)

3 Codesign and concurrent engineering

Klaus Buchenrieder, Siemens AG

The surge of interest in hardware-software codesign is driven by advances in the technologies that support a unified approach. In particular, these include system-level specification and simulation environments, soft-prototyping techniques, formal design and verification methods, and high-level synthesis and framework technology.

A unified approach continuously relates the hardware and software development cycles so that decisions made in either activity significantly affect other operations. Ideally, several design teams develop system components concurrently, so codesign also refers to conjoint team cooperation.

Framework and management structure. Systems from portable CD players to complex flight controls can benefit from this design approach. The field of application is so wide that it is impossible to devise a single method or tool to serve all needs. Therefore, much research has concentrated on developing integrated design methods.

One approach is to stress system-level design over specific hardware and software issues. Industrial systems, however, contain mainly time-discrete and time-continuous components for which formal design methods and design tools exist. It is possible to move to higher levels of abstraction in the design process and still take advantage of lower level tools, but it requires a formal model at project start for both the system under design and the codesign process itself. The work flow can then be mapped onto the model so that tool support for data and task management can be established, and threads of action can be assigned in parallel. Some practical codesign environments employ framework technology for tool encapsulation, sequencing, scheduling, and auditing.

As a team effort, codesign needs "coaching" to be successful. A coaching staff consists of technical specialists and managers who meet regularly from project start to end. They uncover and remove barriers, and act as mediators — possibly between remote development sites. In the startup phase, they devise threads of action for all design disciplines and equip each thread with appropriate tools. The key issue is to establish a smooth transition between development sites, avoiding "ball over the wall" or "island design" problems. The coaching staff can maintain communications between designers through informal reviews and formal consistency-checking systems embedded in the framework.

Partitioning and integration. As in all fields of engineering, measurements guide decision processes; hence, codesign must employ metrics on the design, system, and hardware-software level. Design-level metrics are very similar to those employed for analysis of parallel computer systems. This is because modules that implement time-discrete or time-continuous functions are continuously active in parallel.

System-level metrics address architectural issues, reliability, serviceability, and system performance. Designers must measure parameters and

compare them to estimates for full hardware or full software implementations. All this boils down to the most famous question in hardware-software codesign: How well is the system partitioned into hardware and software?

Partitioning actually starts with the system-design modeling step, in which the designer expresses the system's behavior formally with, for example, parallel random-access machines that encapsulate state diagrams and time-continuous transfer functions. Depending on the underlying theoretical model, abstraction level, and integration strategy, several estimation and analysis methods are available. Deterministic estimation, for example, requires a fully specified model with all data dependencies removed and all component costs known. This method leads to very good partitions, but fails whenever data items are unavailable. Statistical estimation, based on the analysis of similar systems and certain design parameters, is then required. Profiling techniques rely on the examination of control flow and data flow within an architecture to determine computationally "expensive" parts prone for realization in hardware.

In the strategic area, the issue is early versus late binding. Early binding is the preferred strategy in the industry because it supports complete planning of the development cycle to guide design decisions. In contrast, late binding can help in finding better solutions to product performance issues and in addressing "moving target" problems such as change requests from consumers.

After parallel implementation of assigned team modules, all design threads are joined for system integration. Clearly, successful integration depends on the quality of the partitions. Since both steps complement each other, interfaces must be specified during system partitioning and prepared for automatic synthesis and system test.

State of the art. Codesign is slowly gaining acceptance in the industry — partly because meeting the design-to-cost bounds on the development of increasingly complex systems under rigid time-to-market constraints requires new approaches. Confidence that hardware-software codesign can solve some of these problems stems from successful automotive and telecommuncations applications. In addition, many small-scale codesigned systems do exist.[1,2]

Critics bring forward the issue of a missing common model or standard for unified, exchangeable design representation. Even though research labs are working in this area, we are far from a standardized representation.

Partitioning starts with the system-design modeling step.

Another issue is that few tools are designed for hardware-software cross specification, development, simulation, integration, and test. Most design tools apply to a single domain, making it difficult to observe the sequence of operations or connect such tools to framework environments. Design flow management, operation monitoring, or task cross-checking becomes very hard if not impossible. Tool vendors must support hardware-software codesign efforts by supplying mechanisms for framework interfacing. Frameworks should also be improved with respect to communication, cosimulation, auditing, and checkpointing capabilities between physically distant development sites linked via networks.

Because estimation and analysis tools are in their infancy, automatic system partitioning is not yet possible. At best, users get some hints from estimations to guide their intuitions. This is not sufficient for industrial applications. Much attention must be focused on partitioning methods and tools for industrial use.

There is plenty of room for research and experimentation. The field is evolving rapidly. A book based on the First International Workshop for Hardware-Software Codesign in Grassau, Germany, held in the spring of 1992, and the follow-up in Estes Park, Colorado, in late September, is scheduled for publication early in 1993. Interest is increasing among industrial labs and production sites in the US, Europe, and Asia. The Computer-Aided Software-Hardware Engineering Codesign Workshop, scheduled for May 1993, and special sessions of the Design Automation Conference and the European Design Automation Conference will increase the community interested in the topic as well.

References

1. Collected papers from the First International Workshop on Hardware-Software Codesign, sponsored by IFIP WG 10.5 in cooperation with WG 10.2, Grassau, Germany, May 19-21, 1992.
2. Workshop handouts from the First International Workshop on Hardware-Software Codesign, sponsored by ACM and IEEE, Estes Park, Colo., 1992.

Klaus Buchenrieder is research manager at Siemens AG, Corporate R&D, Base Technologies, Software and Engineering, ZFE BT SE 52, Otto-Hahn-Ring 6, W-8000 Munich 83, Germany.

Extending CAD *(continued from p. 85)*

thus a hardware-software codesign problem with possibly multiple solutions. CAD tools can explore the trade-off and suggest a convenient implementation. The Piper synthesis program is an example of a codesign tool that addresses this problem.[1] It provides pipe stage partitioning and pipeline scheduling, and also determines the instruction reorder that the corresponding back-end compiler should use to avoid hazards.

Embedded systems are computing systems dedicated to an application. The most restrictive view of an embedded system is a microcontroller or a processor running a fixed program. This model can be broadened to a general-purpose processor, assisted by application-specific hardware and memory, that performs a dedicated function. Sensors and actuators allow the system to communicate with the environment.

Embedded systems often fall into the class of reactive systems. They are meant to react to the environment — executing functions in response to specific stimuli. In some cases, their functions must execute within predefined time frames. Hence, they are called real-time systems. Examples of reactive real-time systems are pervasive in the automotive field (for example, engine combustion control), the manufacturing industry (robot controllers), and the consumer and telecommunication industries (portable telephones).

Embedded systems and ASICs. Embedded systems can be thought of as a generalization of application-specific integrated circuits (ASICs), where a processor coupled with its software program can be viewed as a system resource. The similarities between embedded systems and ASIC design

make it likely that CAD systems will evolve to support embedded system design in the near future. On the other hand, the heterogeneity of embedded systems requires the development of design systems that take both the hardware and software component requirements into account. Frameworks that support the cospecification and cosimulation of heterogeneous systems are particularly attractive for hardware-software codesign.

Ptolemy is a design environment that uses an object-oriented programming paradigm to support signal-processing and communication system design.[2] It provides for heterogeneous cospecification by supporting several modeling styles, including synchronous and dynamic data-flow and discrete-event models. Ptolemy is modular and can be extended with other representation styles. By allowing each system component to be modeled in its natural way, Ptolemy supports effective capture of different facets of a digital system. In addition, it supports concurrent simulation of the system under development and the generation of assembly code for some programmable digital-signal-processing cores.

Computer-aided synthesis of embedded systems, called here *cosynthesis*, is the natural evolution of existing hardware high-level synthesis methods to support system architectures that contain both a software-programmable and an application-specific component. For generality, we can assume that the software-programmable component is executing either on a general-purpose microprocessor or a processor core.

A working hypothesis for cosynthesis is that the overall system can be modeled consistently and partitioned, either manually or automatically, into a hardware and a software component. The hardware component can then be implemented in application-specific hardware circuits by using existing hardware-synthesis tools. The software component can be automatically generated by a program that implements the function to which the processor is dedicated. Cosynthesis must support a means for interfacing and synchronizing the functions implemented in these components.

Benefits. In addition to balancing the performance of customized hardware units with the programmability of software components, mixed hardware-software systems may have the advantage of evolving more efficiently by allowing software programs to undergo upgrades in subsequent releases. By the same token, assigning most (if not all) functionality to the software component can ease product development by reducing circuit fabrication in the prototyping stage.

CAD may support codesign of embedded systems in the near future.

Partitioning into hardware and software components affects overall system cost and performance. At one end of the spectrum, hardware solutions may provide higher performance by supporting parallel execution of operations, but they require the expense of fabricating one (or more) ASICs. At the other end of the spectrum, software solutions may run on high-performing processors available at low cost due to high-volume production. However, operation serialization and lack of specific support for some tasks can decrease performance.

System-level partitioning into hardware and software components has been the object of novel investigations. Ernst and Henkel[3] targeted codesign of systems that were originally modeled as C software programs but had potential for speeding up critical functions through a hardware component. A partitioning program identified the computational bottlenecks and migrated the corresponding functions to application-specific hardware. For example, the program identified a critical loop that took 90 percent of the software runtime for an HDTV chromakey algorithm running on a Sparc processor. By fabricating an ASIC with 17,000 equivalent gates, they achieved a factor-of-three speedup.

Research at Stanford University[4] uses a complementary approach. Systems are modeled in a hardware description language: HardwareC has a C-like syntax, so a purely hardware implementation is available using the Olympus synthesis tools. By shifting noncritical functions from hardware to software program fragments running on a programmable processor such as an I8086 or an R3000, size and cost of the hardware implementation can be reduced without sacrificing performance. The system model specifies performance requirements in terms of latency and data-rate constraints, and when feasible a partitioner satisfies the requirements. Stanford researchers have used this approach to partition a full hardware implementation of an Ethernet coprocessor and save 20 percent of equivalent gates.

Open issues. CAD tools for codesign are still in their infancy, but potential payoffs make them an attractive area for research and development.

Among other problems that impede growth in the field is the lack of well-defined abstract models for hardware-software systems and consistent languages for expressing them. Possible solutions range from the extension of existing hardware and software languages to the use of heterogeneous paradigms. While the latter approach may be more natural to the user, it leaves the burden of determining coarse-grained system structure to the designers and limits the applicability of system partitioning tools.

A second problem lies in the remoteness of abstract models from physical implementations. This complicates the cost and performance evaluations that play an important role in partitioning and synthesis decisions. And finally, methods for validating hardware-software systems are very important. Cosimulation provides a simple way of tracing the input/output (and internal) system behavior; however, it may be insufficient for the design properties of large systems. Extending formal verification techniques to the hardware-software domain would thus be desirable.

References

1. I. Huang and A. Despain, "High-Level Synthesis of Pipelined Instruction Set Processors and Back-end Compilers," *Proc. DAC*, IEEE CS Press, Los Alamitos, Calif., Order No. 2822, 1992, pp. 135-140.
2. J. Buck et al., "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," to be published in *Int'l J. Computer Simulation*, special issue on simulation software development, 1993.
3. R. Ernst and J. Henkel, "Hardware-Software Codesign of Embedded Controllers Based on Hardware Extraction," handout from First Int'l Workshop on Hardware-Software Codesign, sponsored by ACM and IEEE, Estes Park, Colo., 1992.
3. R. Gupta, C. Coelho, and G. De Micheli, "Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components," *Proc. DAC*, IEEE CS Press, Los Alamitos, Calif., Order No. 2822, 1992, pp. 225-230.

Giovanni De Micheli is associate professor of electrical engineering and, by courtesy, of computer science at Stanford University, Center for Integrated Systems, Stanford, CA 94305.