A Survey of Digital Design Reuse

Margarida F. Jacome
The University of Texas at Austin

Helvio P. Peixoto
Intel Corp.

The authors survey recent advances in digital design reuse. They stress the need for effective strategies that accommodate reuse throughout the entire design process.

As integrated circuit technologies advance toward higher performance, greater densities, and increasing system complexity, CAD tools and design methodologies struggle to keep pace. Managing the formidable complexity of the design process is one of the main challenges to IC design. Disseminating design reuse is central to bringing the design effort's complexity back to a manageable size. Effective reuse, though, takes more than just gathering predesigned components in a library. Reuse-oriented policies and strategies must permeate the entire design process, from the methodologies themselves to the final designs.

In this article, we provide a brief overview of the state of the art in design reuse for digital systems. We also discuss the challenges posed to this discipline by the recent trend toward integrating processor cores in high-volume application-specific integrated circuits.

Design reuse, particularly in the core-based system-on-chip (SOC) design era, is no longer a discipline in isolation. Reuse cannot be fully separated from simulation and verification, estimation, synthesis, or test. For example, the "Design Section" of the 1999 edition of the *International Technology Roadmap for Semiconductors* emphasized the need for synthesis technology to support the reuse of predesigned blocks with surrounding synthesized logic, thus facilitating intellectual property (IP) reuse.[1] The need for new levels of abstraction to support efficient system-level early design-space exploration, synthesis, and simulation and verification is yet another example of a synergy increase across CAD areas. Early performance estimation is also important so that top-down design methodologies can correctly evaluate design decisions at high abstraction levels, such as at the system or architectural level. Without good estimation techniques, top-down design of deep submicron technologies may be impractical, and CAD tools developed for such design would be useless. The semiconductor and electronic-design-automation (EDA) industries must successfully leverage several decades of accumulated investment in hierarchical, top-down design methodologies. Finally, standardization—namely, for system descriptions and portable synthesis library formats—is yet another critical issue in SOC design.[1]

Figure 1 shows the main industry players for SOC, field-programmable design.

Reuse-oriented design, verification, and test

We now consider two representative contributions for creating highly reusable components and assisting the reuse process. The first contribution typifies the traditional function-specific, hardware-only paradigm; the second considers larger-scale, field-programmable systems.

General-purpose VHDL-based reuse

Preis et al. proposed a domain-independent, VHDL-based reuse methodology for a hypertext-based reuse environment.[2] At the core of this

0740-7475/01/$10.00 © 2001 IEEE

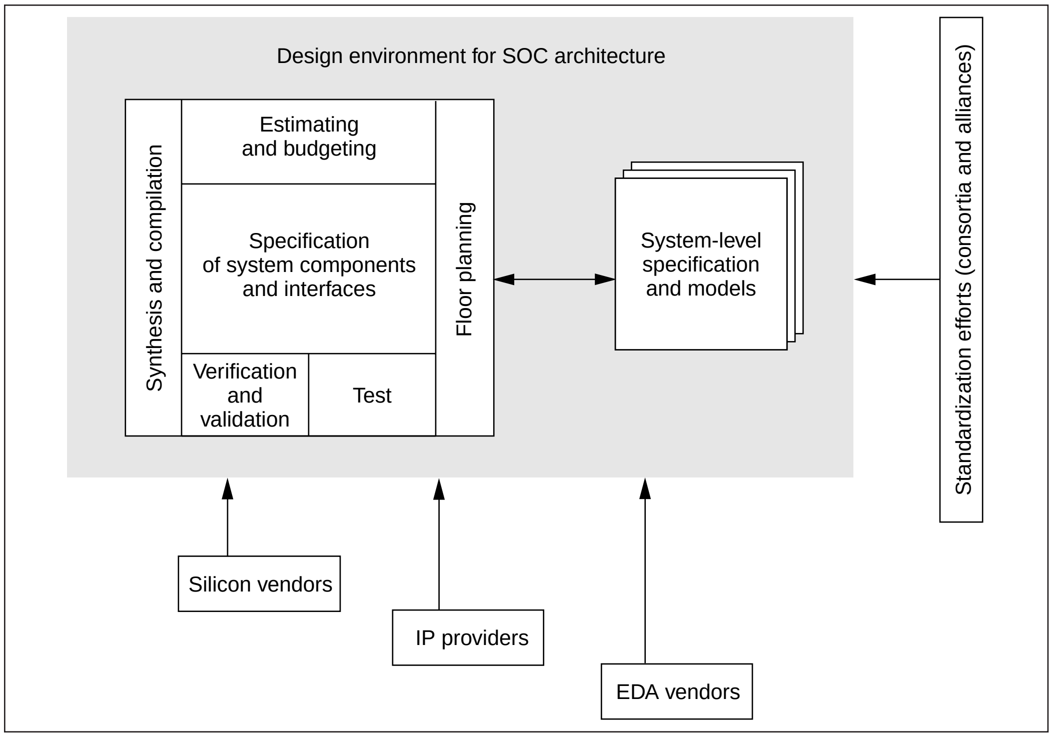
Design environment for SOC architecture
Synthesis and compilation
Estimating and budgeting
Specification of system components and interfaces
Verification and validation
Test
Floor planning
System-level specification and models
Standardization efforts (consortia and alliances)
Silicon vendors
IP providers
EDA vendors

Figure 1. IP reuse can significantly affect design productivity, beginning with the critical system-level design phase. Key players involved in system-on-chip design include embedded application developers, electronic-design-automation vendors, intellectual property providers, and silicon vendors, most of which are currently committed to standardization efforts. The top box represents the fundamental tasks required during early design-space exploration, and the myriad models needed to support such tasks.

approach is a library of hardware components specially designed for reuse. Three fundamental pieces of information define each reusable library component: the reuse documentation, component model, and verification support.

Reuse documentation includes a checklist-based reuse guideline to assist designers through the reuse methodology steps for each library component. Such steps include

- refinement, through parameter specification, of the reusable component model, to meet design requirements;
- functional verification, supported by built-in unit test benches with standard test cases;
- synthesis, automated by scripts; and
- hardware testing, supported by built-in self-test (BIST) structures, whenever appropriate.

A reusable component's basic modeling style is a model that can be fully parameterized—one where parameters completely determine all functional and structural variants for the component's retargetable aspects. Such parameterization may include

- aspects of the component structure (inserting, selecting, and arranging subcomponents such as register files, data width, and BISTs); and
- aspects of the component functionality (characterizing logic-level behavior by specifying active signal levels, or addressing other higher-level functional issues such as reset types and signal coding).

When the number of parameters in the component exceeds a manageable size, Preis et al. propose an alternative template-based modeling style. A template is a construction kit to build complex models from base models that can be fully parameterized. Checklists guide the designer through the construction process, but only serve to prevent or reduce composition errors; they do not guarantee correctness.

In the proposed methodology, simulation supports verification,[3] with a unit test bench supporting each reusable component. For components with models that can be fully parameterized, the test bench is also fully parameterized; the model's parameters automatically adapt simulation and validation to each possible instance of the reusable component (a component instance is a component with a value assigned to each of its parameters). Similarly, for reuse components built using the template approach, the unit test bench is also prepared as a template. Preis et al. suggest concentrating on highly reusable test benches (minimizing work through reuse) rather than guaranteeing that the test bench architecture will fit the needs of all possible applications (maximizing accuracy and completeness).

Reuse-based rapid prototyping for signal processing

To stress the importance of continuous product improvement in meeting short development schedules (three to 12 months), the Defense Advanced Research Project Agency's Rapid Prototyping of Application Specific Signal Processors (RASSP) program introduced the Model of the Year Architecture (MYA) approach, focusing on the complex evolution of large-scale embedded signal-processing applications.[4] The MYA approach is an incremental refinement methodology; rapid prototyping reuse process and reuse libraries help develop Model of the Year upgrades. When developing the MYA approach, the RASSP program placed significant emphasis on facilitating reusability and regular low-cost technology upgrades, as well as on developing a supporting base. This base includes not only automation tools and reuse libraries but also enterprise integration capabilities and standards. The basic MYA elements are

- the functional architecture,
- encapsulated library components, and
- design guidelines and constraints.

The functional architecture specifies the abstract architectural components and standard functional interfaces among these components—that is, the manner in which the component's interfaces must be defined to ensure that the design is upgradeable and to facilitate *technology insertion* (moving from very abstract models toward more specific, detailed models that incorporate implementation technology issues). The functional architecture does not specify the signal-processing architecture's topology or configuration; nor does it stipulate specific processor types or system-level standards for interfaces external to the processor. The functional architecture is thus defined at a higher abstraction level—a starting point in the requirements definition for application-specific architecture selection. In addition, this architecture enforces a standard reference approach for selecting and implementing open interfaces, and establishes guidelines for verification and test.

Libraries of hardware and software elements—encapsulated library components—facilitate reuse and upgrades. Functional wrappers, to support the abstract functional architecture, encapsulate these architectural reuse library components. Specifically, by hiding or abstracting implementation details, these functional wrappers ensure library element interoperability and technology independence.

Finally, the design guidelines and constraints specify how to use the functional architecture framework. They describe the general use of encapsulated libraries and provide procedures and templates to encapsulate new library components.

Three abstraction levels of modeling are used to define reuse library elements.

- *Performance, uninterpreted, and architectural models* provide sequence and timing-only behavior for processor nodes, buses, and interconnects by specifying the number and type of processors and the type and topology of the network. (Uninterpreted models don't actually execute the tasks that the modeled application must perform; they only specify the expected time to fulfill these tasks, address general synchronization issues, and so on.)

- *Abstract behavioral models* provide fully functional behavior, including those at both the algorithm level and the instruction-set-architecture level.
- *Fully functional and interface models* provide full functionality at the signal level, and timing fidelity at the signal and clock level, including register transfer level (RTL) and logic models, which perform all computations in the same way as the final design.

Finally, the representation of architectural elements as objects may include not only hardware representations in the form of VHDL models but also the behavior defined by the software libraries associated with the hardware.

The reuse-based rapid prototyping methodology starts with the creation and refinement of dataflow graphs, which model the intended behavior of the application's data and signal groups. These graphs drive both the architecture design and the software generation for the target signal processors. Specifically, each node in the data flow goes to either software or hardware. Software generation then provides executable threads to be run on the selected digital signal processors. Verification proceeds during the many abstraction levels.

Some of these key ideas and concepts have been successfully incorporated into commercial products—for example, the ALTA tools of Cadence and the Inventra from Mentor Graphics. Synopsys and Mentor Graphics recently created a partnership to develop a reuse-based design methodology.[5]

Design reuse and high-level synthesis

Several research groups have investigated reuse to improve the quality of the designs produced by state-of-the-art high-level synthesis tools. (The "Evolution of design reuse" sidebar discusses the history of design reuse.) Some recent efforts have focused on automating the synthesis of interfaces between IP blocks (reusable components). Here we discuss representative contributions in these areas.

High-level library mapping

Jha and Dutt proposed a high-level mapping technique to let architectural and logic synthesis tools directly reuse technology-specific RTL data path components—in much the same way as traditional logic synthesis tools map components into standard cells.[6] Jha and Dutt identified RTL components, such as arithmetic-logic units (ALUs) and register files, as good candidates for RTL libraries. This was because designers often hand-optimize or create these components using module generators rather than the traditional technology-mapping techniques implemented within high-level synthesis tools.

Jha and Dutt developed and validated the high-level library mapping technique using several optimized ALUs drawn from different libraries. The mapping draws on the functional specifications of the source component (one of the reusable ALUs in the library) and the target component (the ALU under design). The mapping derives from comparing these two functional specifications with a canonical functional specification of the ALU.

The canonical specification of an ALU or other data path component is based on several functions, each defining a relationship between the component's inputs and outputs. Given the canonical ALU specification, each library component is described by its corresponding subset of canonical functions and by the Boolean relationship between the ports of the library component and of the canonical component. Those last relationships represent required glue logic.

The mapping algorithm has two main steps. First, it implements the source ALU onto the canonical ALU using only those functions present in the target ALU. The algorithm performs this implementation, using predefined mapping rules, which also contain entries specifying the connectivity between the ports of the source and canonical ALUs and the necessary additional glue logic. The mapping process is formulated as a dynamic programming problem that incrementally constructs mappings of function subsets—partial solutions. The mapping process completes when a set of rules is found, one for each source function, that minimizes the cost of extra hardware, measured in gate count or delay. The algorithm's second step maps the canonical ALU onto the target component—that is, connects their ports together.

Evolution of design reuse

The traditional requirement for a reusable design is that the function it realizes is approximately duplicated or is common to several applications. Arithmetic circuits, such as adders and multipliers, are examples of common building blocks needed in many different applications. Components implementing standards such as an Ethernet interface or an Integrated Service Data Network protocol are also natural candidates for reuse across multiple designs. Design evolution itself provides yet another form of common functionality of major strategic importance: Over time, groups of products are updated and adapted to increase functionality or performance, possibly taking advantage of new implementation technologies to increase their market competitiveness.[1]

The introduction of hardware description languages (HDLs) led to a major leap in design reusability. The various forms of common design functionality could then be represented in a technology-independent form that was easier to understand and customize to specific applications. The subsequent HDL-based generations of high-level synthesis and verification tools consolidated functional reuse. They automated process retargetability and facilitated the exploration of area, speed, and power trade-offs in the physical implementation of the reusable components. HDL-based tools let companies compile libraries of technology-independent functions, which could be reused in each new design generation.[1]

Most HDLs also provide built-in parameterization features for creating reusable components that designers can tailor to support different applications. For example, you can use an *N*-bit adder or multiplier to achieve parameterization at the structural level. At the functional level, you can design a modem that operates at several different frequencies and signaling schemes; then you can tie such variants to actual operating mode parameters defined at the HDL level. Building designed-for-reuse, parameterized components is difficult, but you can use them in many applications, thus increasing the potential return on your investment.

Since the arrival of HDLs, mainstream research in design reuse has focused mostly on modeling and design tools for creating highly reusable, function-specific components and for assisting the reuse process. In the last few years, though, the increasing importance of field programmability has irreversibly challenged this function-specific reuse paradigm. Indeed, if performance and power-consumption requirements allow, you can use microprocessors and field-programmable gate arrays to directly implement most of the functionality of complex systems-on-chip (SOCs). Such field programmability gives system developers valuable flexibility. For example, you can create entire families or lines of products that share a common hardware platform, with the various products differentiated mainly, or completely, by features implemented in software. Moreover, you can significantly reduce the cost of accommodating changes in a product's specification. Reflecting the attractiveness of such solutions, field-programmable systems are already pervasive in many embedded application areas, including telecommunication line cards, cellular and cordless phones, video processing, and automotive real-time control.

The increasing trend toward integrating the number of intellectual property cores—complex components developed by a third party—and, in particular, processor cores, in high-volume application-specific integrated circuits is irreversible (see http://www.design-reuse.com). *The Economist*, for example, forecasted that by the year 2002 the sale of systems containing embedded processor cores will nearly match that of PCs and thereafter greatly exceed them.[2] Increasingly, such cores are application-specific instruction-set processors—that is, processors whose architecture and instruction set are customized to a specific application. Compared to more general-purpose processors, the architecture and

Increasing data path reusability

High-level synthesis tools traditionally do not directly consider physical design effects. The performance predicted by such tools must be recalculated after the physical design performance information of the RTL design is back-annotated with timing information. In designing data path circuits, for example, you might completely resynthesize the data path by running the synthesis tools again with the back-annotated physical design information. However, redoing scheduling and allocation with the new physical design information may generate a completely different data path for which the previously back-annotated physical design information is useless.[7] Similarly, retargeting an existing data path to a new standard cell library may cause suboptimal execution delays when the new instruction-set specialization of ASIPs yields better area/performance and power/performance ratios.

Some of the issues posed by field-programmable core-based designs are similar to those faced in traditional digital design reuse except that, because of the extreme complexity of the reusable components (the processor cores), the supporting technology is stressed to its limits (the tools are too slow, they malfunction, and so on). The need for well-defined and documented interfaces is one such critical issue.[3] At an even more fundamental level, field-programmable core-based designs challenge the paradigm underlying most state-of-the-art contributions in reuse. Such a paradigm, centered on function-specific reusable components, does not directly apply to a programmable component such as a processor core. Because HDLs let you define technology-independent yet function-specific building blocks, new modeling mechanisms are needed to accommodate the heterogeneous nature of the main architectural components that comprise field-programmable systems.

The previous HDL-based reuse paradigm separated function from implementation technology. This new scenario requires decoupling, for as long as possible, the specification of a system's intended behavior from decisions about the system components' final implementation style—for example, function-specific hardware building blocks and software programs executing on core processors. The challenges posed by field programmability are still very new, and their effect on reuse, as well as on synthesis and verification, are still basically open research issues. Hardware/software codesign, a recently created CAD discipline, aims at realizing the full potential of field-programmable system design.[3,4]

We have thus far focused on the challenges posed by SOC design from a system-level perspective—that is, at the highest level of design abstraction. To compound the problem, important challenges are also faced during design planning and, later, at the physical level—the lowest abstraction level. First of all, the ability to integrate increasingly complex systems on a single chip is obviously quite attractive from the point of view of important physical figures of merit, such as performance and power consumption. This ability is also advantageous from a cost and reliability point of view. Unfortunately, the high chip-transistor densities, which made possible such SOC designs, are now challenging the very foundations of top-down, hierarchical design.

Today it is possible to create CMOS transistors with feature sizes of 180 nm and smaller—now referred to as deep-submicron technologies. With these advances toward larger and denser circuits, the interconnect has emerged as a critical factor limiting performance; most of the on-chip signal delay will actually occur in the interconnect.[5] Therefore, in the deep-submicron era, designers can no longer reliably achieve such physical figures without considering the physical level. This problem is crucial not only to the end developers of application-specific systems but also to the actual core developers themselves. Another recently created CAD discipline, timing-driven design (also called design for interconnectivity) focuses on developing mechanisms to reliably estimate the delay of future transistors and interconnects and to account for all such delays early in the design process.

References

1. E. Girczyc and S. Carlson, "Increasing Design Quality and Engineering Productivity through Design Reuse," *Proc. ACM/IEEE Design Automation Conf.*, ACM Press, New York, 1993, pp. 48-53.
2. "After the PC," *The Economist*, 12 Sept. 1998, pp. 79-81.
3. G. De Micheli and M. Sami, eds., *Hardware/Software Codesign*, Kluwer Academic Publishers, Norwell, Mass.,1996.
4. D. Gajski et al., *Specification and Design of Embedded Systems*, Prentice Hall, Upper Saddle River, N.J., 1994.
5. K. Keutzer and D. Sylvester, "Chip-Level Assembly Is the Key to DSM Design," *Proc. ACM/IEEE Int'l Workshop Timing Issues in the Specification and Synthesis of Digital Systems* (TAU 99), ACM Press, New York, 1999, pp. 23-24.

physical delay values are applied to a data path optimized for different physical delays.

Jha proposed a technique to address such problems in the context of data path designs.[7] Specifically, he resynthesized the data path controller using a technique called reclocking; the controller redesign increased the particular data path's reusability by improving its performance. Logic synthesis tools can easily synthesize controllers using standard-cell libraries. So the data path and the connectivity between the controller and the data path are preserved, and only the controller is redesigned.

Jha defined the reclocking problem as follows: Given an initial schedule for the operations in the design's behavior and the updated delays—back annotated or from a new library—a clock width leading to minimal execution time is first determined for the various paths in that schedule. The controller, a finite state machine that specifies the operations in each state, is then rescheduled and resynthesized based on this new clock width. Jha showed that the optimal clock width lies on an integer division of the largest combinational delays of each state.[7] The algorithm for finding optimal clock width has a runtime complexity of $O([n + m]^2)$, where n is the number of states in the input, and m is the number of multicycle operations.

Synthesis of interfaces

Passerone, Rowson, and Sangiovanni-Vicentelli proposed an algorithm for automatic synthesis of interfaces between cores or IP blocks with different signaling protocols.[8] The synthesized interface converts from one set of signaling conventions or protocols to another. The approach's key assumptions follow:

- The communication is point to point. Each module has a set of ports, data, and control, over which the data transfer occurs.
- The internal storage provided in the interface is always sufficient to store the entire data type.
- The IPs exchanging data are driven by the same clock—that is, they are fully synchronous. Moreover, the synthesis can consider only a single data transfer.

The input to the algorithm is a description of the protocol used by the two modules. The protocol is the legal sequence of values that may appear on the ports from the onset to the end of the data transfer. The output of the proposed algorithm is a finite state machine and a data path consisting of the interface's internal registers.

Ports are first ordered in an arbitrary way. Those representing buses are bundled together and assigned a single identifier, or name. A protocol symbol is a tuple with ports listed in assigned order. A protocol is a set of strings of symbols—a language in the alphabet of all the values that a symbol may assume. Such a language is described through regular expressions. A synthesis algorithm generates the correct interface, if one exists, between the two protocols. This interface is a finite state machine that recognizes a given regular language on the producer module (the module sending the data) and generates a proper string contained in the regular language of the consumer module (the module receiving the data).

Industry initiatives and standardization efforts

The 1999 *International Technology Roadmap for Semiconductors* recommended new standards for system descriptions and portable synthesis library formats.[1] The Silicon Integration Initiative responded by launching several technology programs actively addressing the need for such standards. These programs include the Chip Hierarchical Design System Technical Data Standard, the Electronic Component Information Exchange, and the Delay and Power Calculation Program (http://www.si2.org).

The System Level Design Language (SLDL) is an ongoing worldwide standards initiative aimed at developing an interoperable language environment for the specification and system-level design of single- and multichip silicon-based embedded systems (see http://www.inmet.com). Bringing systematic verification and validation methodologies to the initial phases of SOC design is crucial. SLDL describes system behavior and design constraints before hardware/software partitioning, thus enabling system-level verification and validation using appropriated tools and techniques.

The Virtual Socket Interface alliance reflects yet another important industry-organized response to the challenges posed by core-based SOC designs (see http://www.vsi.org). The objectives of the recently created VSI alliance include

- selecting appropriate design representations—formats—for virtual component deliverables; and
- defining required and recommended design practices for virtual components, covering logical design, physical design, test, and bus interfaces.

Bringing the object-oriented modeling paradigm to system design may also help reduce design complexity. Object-oriented languages support the so-called reactive computation model, which is significantly more abstract than that of typical hardware description languages (HDLs). This model is useful during early design stages to verify the correctness of fundamental high-level design decisions before investing additional effort in the detailed design. The IEEE Design Automation Standards Committee Object Oriented-VHDL study group is working on a standard for object-oriented VHDL based on the Objective VHDL extension.[9]

The irreversible trend toward IP reuse is clear from the large number of industry contributions, including those from EDA vendors and ASIC providers, to the annual IP-based workshop.[10] This workshop addresses both business IP-protection and technology issues of IP design, use, and reuse.[11,12]

Design space layer

We have studied naturally merging reuse and early design-space exploration using techniques to estimate performance, silicon area, and other figures of merit; and defined new reuse library layers to support IP-based designs.

We can abstractly characterize the design space and, as needed, make the set discrete as a set of behavioral and structural properties or features.[13] We define properties as higher-level abstractions—specifically, metadata—to organize and categorize the myriad design data created and manipulated during the design process. Such properties include

- behavioral and structural descriptions to define the structure and intended behavior of design objects at various levels of design composition (algorithmic, RTL, logic, and so on);
- constraints to specify the design requirements on performance, area, and so on; and
- design decisions or restrictions during conceptual design regarding critical design issues, such as the system architecture and the implementation technology for the various system components.

Such properties need not be entirely independent of one other—for example, the designer may make a design decision that later proves to be inconsistent with the performance requirements of the system under design.

This formalism is not a substitute for any of the more detailed HDL models discussed previously. Rather, the formalism rests atop such models, properly abstracting their content. The formalism defines a level of abstraction that is higher than typically provided by HDLs, object-oriented HDLs, and other behavioral models or modeling languages. When refining the design, you should still use the appropriate models and languages for describing the specific system behavior (reactive control or computation intensive, deterministic or nondeterministic, and so on).[14] Naturally, you may need to support a certain degree of model heterogeneity between model pieces and across several design abstraction levels. This is particularly true when different models can better describe different parts of the system behavior and constraints.[15,16]

The design formalism generalizes many key concepts and principles explored in previous reuse models and methodologies. Perhaps the most powerful generalization is specialization, implemented at each design abstraction level. Specifically, at each level, the formalism lets you organize properties or features into a hierarchy of increasingly specialized design object classes. This feature is critical for supporting systematic early design-space exploration of large design spaces. Therefore, to control complexity, designers should merge—logically generalize—algorithm groups, hardware- and software-oriented implementation styles, and so on, during the initial exploration phases. By doing so, these items become selectable as viable families of design alternatives, and can then be incrementally discriminated as the design process progresses.

Specialization is realized at the property level. The use of generalized properties—merging families of alternatives—should precede the more detailed properties whenever beneficial for evaluating or determining achievable ranges of performance, energy consumption, and so on. Such properties are then organized into a hierarchy of increasingly specialized classes of design objects (groups of reusable designs). This hierarchical organization, the design discipline hierarchy,[13] formally defines the design space—the range of all feasible implementations—of the design objects represented in it.

The discipline hierarchy also provides the basic schema for specifying, indexing, and identifying component families residing in reuse libraries. Such reusable designs—design objects of a certain class—are thus no longer arbitrarily and monolithically represented and accessed. Instead, while traversing the discipline hierarchy, designers can discriminate or select between object families, defining desired ranges of performance, power consumption, and so on. Designers can also access single properties of individual design objects, such as HDL behavioral or structural models. Thus, reuse naturally merges with the design methodologies themselves. The discipline hierarchy not only minimizes the need for redesign but also assists conceptual design and early estimation.

We proposed a new library layer, the design space layer, to implement discipline hierarchies and support early design-space exploration and estimation for IP-based system-level design.[17] We hope to help designers systematically consider relevant alternative implementations for SOC architecture's various components. This library layer also allows quick and transparent selection from reuse libraries of IP cores that are good candidates for implementing these components.

SPACE LIMITATIONS precluded us from providing an exhaustive coverage of work on design reuse. We have thus presented a small set of representative contributions on several important topics, in some cases omitting other relevant work. Moreover, we focused only on digital design reuse. A relevant topic missing from this discussion is analog design reuse. However, the widespread development and use of analog HDLs began only recently, and automatic synthesis and process retargetability are still quite limited in this domain. Systematic reuse and automation for analog design lags several generations behind digital design.

Functional reuse, enabled by high-level synthesis and verification tools, has taken digital design a long way from the technology-dependent, design-from-scratch approach still very much practiced in analog design. As the levels of density and heterogeneity increase—requiring consideration of higher design abstraction levels—reuse, synthesis, verification, and test hardware and software communities need once again to work synergistically to respond to these new challenges with creative and effective solutions. ■

Acknowledgments

A National Science Foundation CAREER Award (MIP-9624231), a Texas Higher Education Coordination Board Grant (ATP-003658-088), and a donation from Rockwell supported this work. We thank the reviewers for their valuable comments and suggestions.

■ References

1. *International Technology Roadmap for Semiconductors*, Sematech, Austin, Tex., 1999.
2. V. Preis et. al, "Reuse Scenario for the VHDL-Based Hardware Design Flow," *Proc. European Design Automation Conf. with EURO-VHDL 95*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 464-469.
3. M. Schuetz, "How to Efficiently Build VHDL Testbenches," *Proc. European Design Automation Conf. EURO-VHDL 95*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 554-559.
4. J. Pridmore et al, "Model-Year Architectures for Rapid Prototyping," *J. VLSI Signal Processing*, vol. 15, no. 1/2, 1997, pp. 83-96.
5. M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip Designs*, Kluwer Academic Publishers, Norwell, Mass., 1998.
6. P. Jha and N. Dutt, "High-Level Library Mapping for Arithmetic Components," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 4, no. 2, June 1996, pp. 157-169.
7. P. Jha, S. Parameswaran, and N. Dutt, "Reclocking Controllers for Minimum Execution Time," *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, vol. E78-A, no. 12, Dec 1995, pp. 1715-1721.
8. R. Passerone, J. Rowson, and A. Sangiovanni-Vicentelli, "Automatic Synthesis of Interfaces between Incompatible Protocols," *Proc. 35th ACM/IEEE Design Automation Conf.*, ACM Press, New York, 1998, pp. 8-13.
9. M. Radetzki, W. Putzke-Roming, and W. Nebel,

"A Unified Approach to Object-Oriented VHDL," *J. Information Science and Engineering*, vol. 14, no. 3, 1998, pp. 523-545.

10. *Proc. Int'l Workshop IP Based Synthesis and System Design*, IEEE CS Press, Los Alamitos, Calif., 1998.
11. A. Kahng et al., "Watermarking Techniques for Intellectual Property Protection," *Proc. IEEE/ACM Design Automation Conf.*, ACM Press, New York, 1998, pp. 190-195.
12. A. Oliveira, "Robust Techniques for Watermarking Sequential Circuits," *Proc. IEEE/ACM Design Automation Conf.*, ACM Press, New York, 1999, pp. 837-842.
13. M. Jacome and S. Director, "A Formal Basis for Design Process Planning and Management," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 10, Oct. 1996, pp. 1197-1211.
14. D. Gajski et al., *Specification and Design of Embedded Systems*, Prentice Hall, Upper Saddle River, N.J., 1994.
15. W. Chang, A. Kalavade, and E. Lee, "Effective Heterogeneous Design and Cosimulation," *Hardware/Software Co-design*, G. DeMicheli and M. Sami, eds., NATO ASI Series vol. 310, Kluwer Academic Publishers, Norwell, Mass., 1996.
16. E. Lee and A. Sangiovanni-Vincentelli, "A Framework for Comparing Models of Computation," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 12, 1998, pp. 1217-1229.
17. H.P. Peixoto et al., "The Design Space Layer: Supporting Early Design Space Exploration for Core Based Designs," *Proc. ACM/IEEE Design, Automation and Test in Europe* (DATE 99), ACM Press, New York, 1999, pp. 676-683.

Margarida F. Jacome is an associate professor in the Department of Electrical and Computer Engineering at the University of Texas at Austin. Her research interests include CAD for hardware/software codesign of embedded systems, retargetable compilation for VLIW application-specific instruction-set processors, and IP reuse. Jacome has a BS in electrical engineering and MS in electrical and computer engineering, both from the Technical University of Lisbon; and a PhD in electrical and computer engineering from Carnegie Mellon University. She is a member of the IEEE Computer Society and the ACM.

Helvio P. Peixoto is a systems engineer at Intel. His research interests include hardware/software codesign of embedded systems, CAD algorithms, and operations research. Peixoto has a BS in computer science from the Federal University of Uberlandia, Brazil; MSc in computer science from State University of Campinas, Brazil; and PhD in electrical and computer engineering from the University of Texas at Austin.

■ Direct questions and comments about this article to Margarida F. Jacome, Electrical and Computer Engineering Dept., Univ. of Texas at Austin, Austin, TX 78712-1084; jacome@ece.utexas.edu.