COSI: A Framework for the Design of Interconnection Networks

Alessandro Pinto
University of California, Berkeley

Alberto Sangiovanni-Vincentelli
University of California, Berkeley

Luca P. Carloni
Columbia University

Editor's note:
This article presents a software framework for communication infrastructure synthesis of distributed systems, which is critical for overall system performance in communication-based design. Particular emphasis is given to on-chip interconnect synthesis of multicore designs.
—Radu Marculescu, Carnegie Mellon University

TIME-TO-MARKET, NONRECURRING engineering charges, and error-free implementation requirements are revealing the increasing importance of composable designs, whereby complex systems are built from possibly predesigned and preverified components and guaranteed to inherit their properties. Distributed embedded systems and even microprocessors are designed today using preexisting, preverified IP. These architectures require great attention to the design of the interconnect infrastructure and communication protocols. Similarly, communication plays a fundamental role in ensuring the correct behavior of distributed embedded-controller designs, such as UAV (unmanned air vehicle) navigation systems, because delay and throughput substantially affect the control algorithm.

Computer scientists and operations researchers have extensively studied optimal network design for data networks and transportation networks. Several approximation and heuristic algorithms are available. Designers can apply these algorithms to the synthesis and optimization of interconnection networks for SoCs or distributed embedded systems, provided they are properly adapted and combined with accurate models of performance and cost of the communication building blocks. Opportunities for designers are noteworthy, but if they must spend time struggling with design software to implement these algorithms, the opportunities will remain largely untapped.

We built the Communication Synthesis Infrastructure (COSI), a public-domain design framework instituted on the platform-based design paradigm,[1,2] so that researchers and designers could contribute, combine, and compare optimization algorithms, communication protocols, partial designs, and models for interconnection design. Specifically, COSI cleanly separates network specification, the library of building blocks that can be instanced and composed to derive the network implementation, the models of performance and cost associated with each of them, and the optimization algorithms used to explore the design space. Adopting this methodology lets designers compare different interconnection topologies and building blocks, freeing them from preconceived ideas about the efficiency of particular interconnection schemes. The COSI software framework relies on principles that abstract and formalize the problem at hand, making it easily customizable and efficient. It was built to be structurally scalable in the sense that it can accommodate a wide variety of building blocks and performance measures.

To show how COSI works, we present its application to the development of design flows for network-on-chip (NoC) design. However, our approach to

0740-7475/08/$25.00 © 2008 IEEE Copublished by the IEEE CS and the IEEE CASS

interconnection network design is general. The generality is due to a formal framework that abstracts the design problem to a level where the algorithms and the methodology are indeed applicable to the design of interconnection networks for other distributed embedded systems, such as automation systems for buildings.[3,4]

A model for communication synthesis

COSI is based on a model consisting of quantities that measure the performance of a communication component, composition rules that govern how to build composite components from existing ones, and communication structures that capture the behavior of composite components. Here, we can only briefly introduce the model; a complete presentation appears elsewhere.[5]

Design constraints and component capabilities (performance figures) are expressed with quantities. A quantity q ranges on a partially ordered domain D_q. We assume that a quantity's domain contains the special value $\perp$, denoting "no value," and it may contain the special value $\top$, denoting "any value." Quantities can be very general. For instance, a component's ports are pairs composed of a tag and an interface specification. Figure 1a shows the domains of the quantities involved in describing an interface that is a tuple of four quantities: the type τ denoting the interface protocol, the width w in number of bits, the speed f in Hz, and the direction io indicating whether an interface is input, output, or bidirectional. A quantity's domain is ordered according to a relation that ranks each value in terms of performance or constraint. For instance, an interface's speed and width follow the ordering of natural numbers, because an interface offering a broader bit parallelism and operating at a faster speed dominates a slower and narrower one. The domain of quantity io is ordered by the following relations: $\perp < in < inout$ and $\perp < out < inout$, but in and out are incomparable. The type domain is unordered. The domain of the tuple of quantities that specify an interface is the cross product of the domains D_τ, D_w, D_f, and D_{io}, which are sorted according to the order induced by the single quantities. Figure 1b shows the ordering relation among some elements of $D_{(w,f,io)} = D_w \times D_f \times D_{io}$.

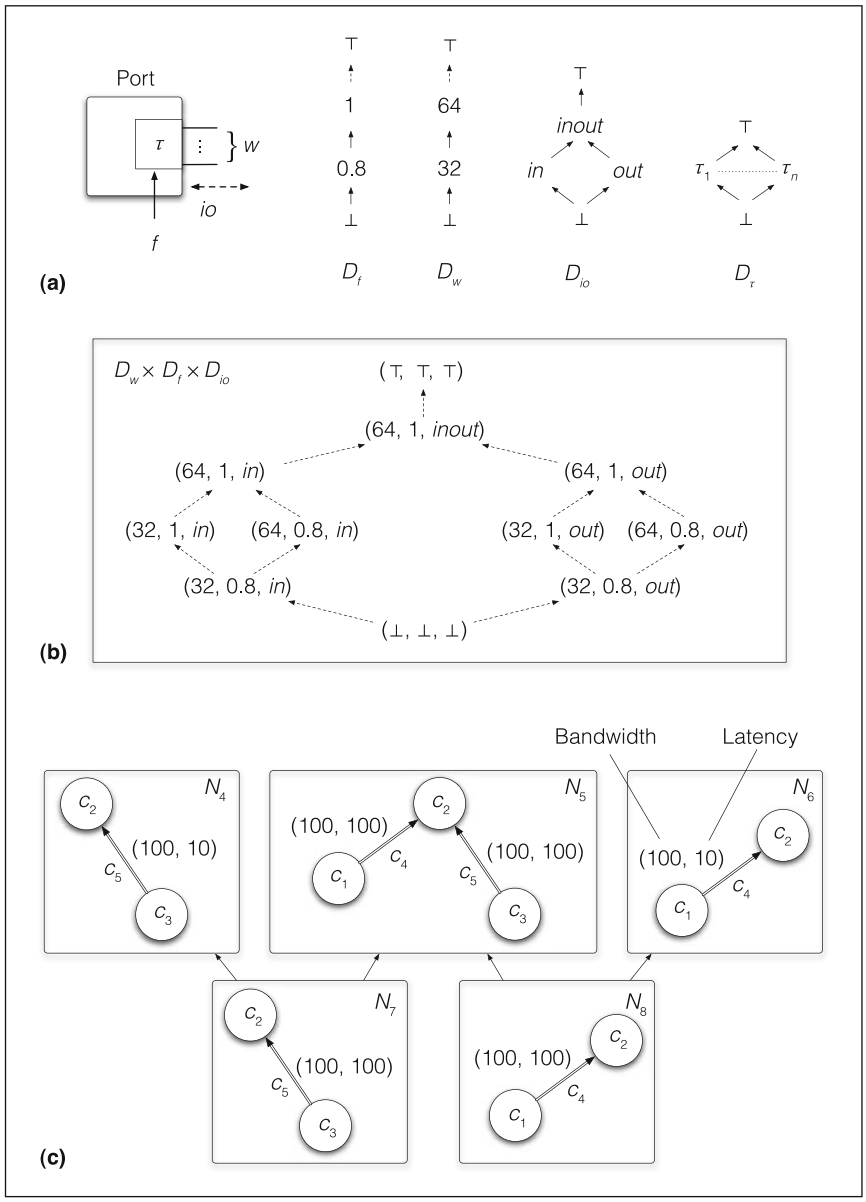

Figure 1. Example of interface description (a), interface quantity domain (b), and communication structure ordering (c).

Quantities are attached to the components of a communication network to characterize its properties. In fact, we represent networks by mathematical objects called *communication structures*. A communication structure is a tuple $N(C, \mathbf{q}, L)$, where C is a set of components (nodes and links), $\mathbf{q}$ is a vector of quantity variables, and L is a set of configurations (a

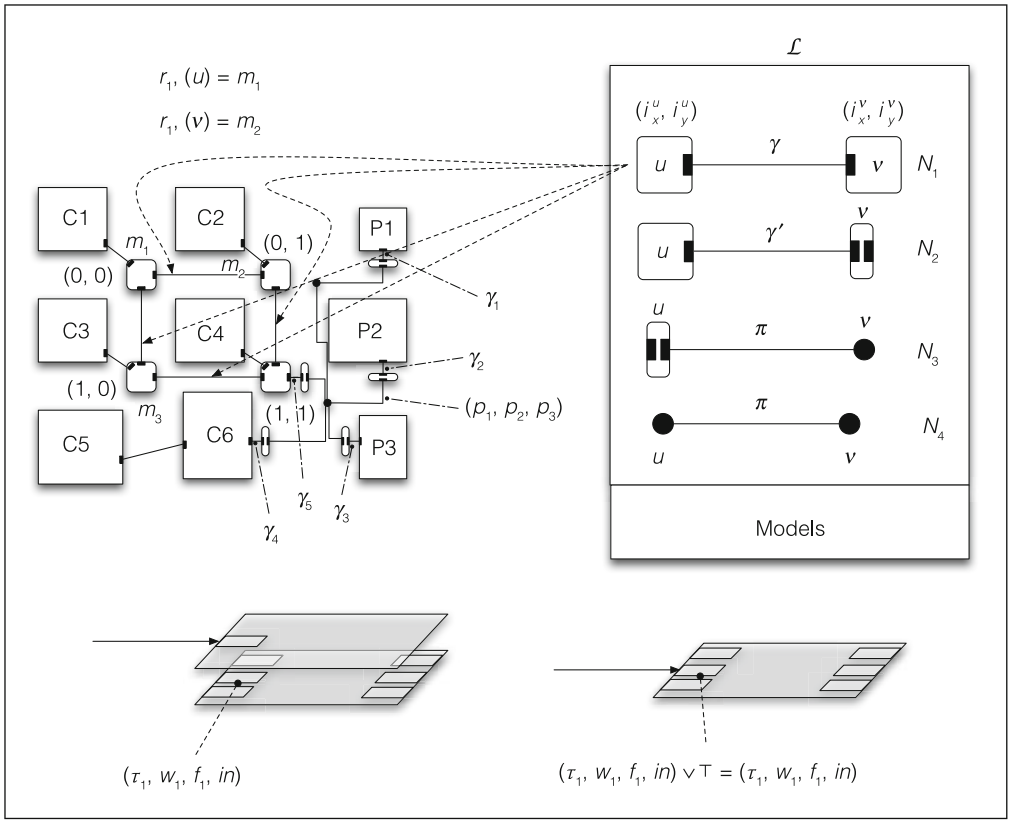

Figure 2. Example library of components, their instantiation, and their composition.

set of functions of the form $l : C \rightarrow D_q$ that associate quantity values to components). The set of all communication structures is also partially ordered by a relation that is induced by the partial order defined on D_q and by component containment. For instance, Figure 1c shows the ordering relation among a subset of communication structures where the vector of quantities has two components: bandwidth and latency. Intuitively, $N_7 \leq N_5$ because N_7 has fewer components, and the components that are in common are configured to perform better in N_5 than in N_7. Instead, N_5 and N_6 are incomparable.

Thus far, the model is generic and does not depend on a particular application domain. Zooming in on the SoC design problem, we capture the specification of a SoC's communication requirements using a communication structure $N_C(C, \mathbf{q}_C, L_C)$, where $\mathbf{q}_C$ contains the quantity variables representing the constraints, and L_C defines their values such as end-to-end bandwidth and latency requirements. The design space—that is, the set of all communication architectures that can be used to implement a communication system in a particular technology—is implicitly defined by a library of communication structures $N_i(C_i, \mathbf{q}_P, L_i) \in \mathcal{L}$, called *library elements*, and by a composition rule denoted by $\|$. The vector $\mathbf{q}_P$ contains the quantity variables representing performance, and L_i defines the performance space of the library element N_i. The composition rule dictates how to assemble the library elements to derive a complex network.

Figure 2 shows an example library containing several types of links. Element N_1 is a mesh link, N_2 connects a component u to a bus interface v, N_3 connects a bus interface to a bus node, and N_4 connects two bus nodes. The quantities attached to the components are the logical coordinates i_x and i_y, the capacity of links γ; and the actual link layout π, which is a list of physical locations on the chip—for instance, (p_1, p_2, p_3) in Figure 2. Library elements can be instantiated by renaming their nodes. For instance, N_1 is renamed by a renaming function r_1 such that vertexes u and v are called m_1 and m_2, respectively. Instances of library elements can be combined to form larger networks. A particular composition of library elements is called a *platform instance*, and the set of all valid platform instances is called a *platform*.

Figure 2 shows one possible composition in which cores C1 to C4 connect by a mesh; C5 connects to C6 by a point-to-point link; and the three peripherals P1, P2, and P3 connect to the rest of the system through a bus. Defining the composition operator can be rather involved and depends on the properties that the composite network is required to satisfy. In general, composition is defined by two rules. One rule defines the configurations of the composite, given the configurations of the two communication structures being composed. For instance, consider a link that belongs to both communication structures. The configurations of these communication structures may in general associate two different vectors of quantities to the link. The composition has to "unify"

these two vectors. In our example, the vector of quantities associated with the link in the composite communication structure can be defined as follows: The link layout is the one with the shortest distance, the capacity is the maximum capacity, and the set of flows is the union of the sets of flows. Another example is the composition of ports, also shown in the bottom part of Figure 2. Consider a node in common between two communication structures being composed. One interface has the value $\top$, meaning "any interface," whereas the other interface has the value (τ_1, w_1, f_1, *in*). When the two nodes are combined, we take the join of the two interfaces with respect to the order shown in Figure 1b (that is, the least common denominator of their features). A second rule defines a set of constraints that the result of the composition must satisfy. For instance, the sum of all capacities of the links between cores and bus interfaces must be less than or equal to the total bus capacity. In the example of Figure 2, $\gamma_1 + \gamma_2 + \gamma_3 + \gamma_4 + \gamma_5 \leq \gamma_{\max}$. For the links instantiated in a mesh, the logical indexes are also constrained such that adjacent nodes differ at most by one unit on one coordinate. All these constraints can be used directly in the formulation of optimization problems to automatically synthesize heterogeneous communication structures. Some rules, such as deadlock freedom, are highly nonlinear, making the development of optimization algorithms particularly difficult.

Moving down from the specification toward the implementation of a communication system, we add more details to the communication structures by augmenting and refining the set of quantities associated with the components. For instance, routing tables are added after the routing algorithm is chosen. Communication structures at different abstraction levels can be related by abstraction functions. For a given implementation N_I, the abstraction $\Pi(N_I)$ returns a point-to-point communication structure capturing all the specifications that N_I can implement. Another abstraction, $\Psi(N_I)$, returns a communication structure defined on the quantities $\mathbf{q}_P$ that characterize the platform. Given a specification N_C and a library $\mathcal{L}$, the model presented here can serve to formulate a general optimization problem for communication synthesis. We can state the problem as follows: Minimize the cost of the implementation $F(N_I)$, where F is a cost function defined on communication structures, such that $N_C \leq \Pi(N_I)$ and $\Psi(N_I)$ is a valid composition of library elements. These constraints can be written in terms of relations on quantities and components of the communication structures and library elements.

Finally, some quantities can be derived from others. We formally define the notion of a model as a function that computes the derived quantity for a given component starting from the value of the quantities associated with it. More generally, given a communication structure and a component belonging to it, a model computes an additional quantity relative to that component. For instance, the delay model of a point-to-point link takes the link configuration (the positions of the extreme nodes and the parameters of the silicon implementation) and returns the value of the delay quantity. Similarly, the I/O delay model of a router takes the router configuration (the values of the input commodities, the routing table, and the parameter of the silicon technology) and returns the value of the delay quantity.

The COSI software architecture

Table 1 shows the COSI software organization. The rows correspond to aspects of the communication synthesis design flows; the columns represent different application domains. The elements that characterize a design flow are

- the quantities and communication structures that define the abstraction levels at which the specification, platform, and implementation are captured;
- the library of communication components, together with their performance and cost models (used to annotate derived quantities and costs), and the composition rules;
- a platform data structure representing the library and the rules so that the synthesis algorithms can operate;
- the environment in which the network operates—for example, floorplanning information in the case of chips, and building geometry in the case of automation systems for buildings; and
- I/O functions such as parsers and code generators that ease the process of specifying the communication problem and analyzing the results.

We present in detail only the first two columns of the matrix. The core package provides basic definitions for widely used quantities such as positions, flows, and ports, as well as data structures and basic

Table 1. Organization of the Communication Synthesis Infrastructure (COSI) software.

Aspects of design flow	Core	On-chip communication	Building automation
Quantities	Ports, bandwidth, flows …	Interface, geometry of an IP core, node implementation parameters	Interface, node implementation parameters, threads
Communication structures	Graphs	Specification, platform instance, implementation	Specification, platform instance, implementation
Library	Abstract classes for nodes and links	Router, link, bus	Sensor, actuator, controller, twisted-pair link
Models	Abstract classes for models	Orion,[6] Ho area model,[7] Ho power model,[7] UCSD model[8]	Token ring, 802.15.4
Rules	NA	Critical length, deadlock	Wiring rule, node position rule
Platforms	NA	Routers and point-to-point links, bus and NoC	Daisy chain buses, wireless tree networks
Environment	NA	Rectangle	Walls, cable ladders
I/O	Dot code generator for graph structure	Parsers, SVG code generator, Parquet interface, SystemC code generator	Building parser, SVG code generator, simulator interface
Algorithms	Shortest path, traveling-salesman problem, spanning tree, facility location, K-median	Degree-constrained shorter path, latency-constrained shortest path, hierarchical-network synthesis	Daisy chain partition, wireless tree

*SVG: scalable vector graphics; UCSD: University of California, San Diego.

algorithms for graph manipulation. The on-chip communication (OCC) package provides specialized definitions for port interfaces (including the number of virtual channels of a router input, the buffer length, and the clock speed), the geometry of a component, and the implementation parameters for nodes and links (for example, global or local interconnects, and shielding). A library of network components includes several models of routers, point-to-point links, and network interfaces. Different area and power models are available for these components, including router models derived with the Orion tool,[6] the analytical model for metal wires presented by Ho et al.,[7] and a more recent set of accurate wire models presented by Carloni et al.[8] All the models are provided for various technology processes, including 90-, 65-, and 45-nm technologies.

The environment package captures the occupied and unoccupied areas on the chip as unions of rectangles. A rich set of input and output functions is also provided. This includes tools to parse the input specification and the synthesis script, given in XML format; tools to generate graphical views of the synthesized network; and tools to produce cycle-accurate SystemC descriptions of the synthesized network. Various algorithms are already available for OCC synthesis.

The COSI software implementation has been engineered to support the orthogonalization of concerns advocated by platform-based design. Figure 3 shows the class diagram relative to the definition of communication structures, components, and platforms. The core package includes the basic data structures for quantities, configurations, and communication structures. COSI users define the quantities, together with their partial order, and attach them to communication structures. For instance, users can define the interfaces of IP cores and routers, pass the interfaces as parameters to ports, and attach ports to components. Other definable quantities include commodities, which represent the flow of packets from a source to a destination; latency figures; implementation parameters for nodes and wires; and the geometry of chip components.

The bottom part of Figure 3 shows how COSI captures components and platforms. The core package defines node and link components as basic objects with ports. Each component in the library must implement two sets of services: instantiation services, which allow generation of component instances, and performance and cost computation services, which expose the metrics of each component through models. An instantiation service returns a communication structure for a given name and

configuration of a library element. The dashed lines in Figure 2 represent this method. Performance and cost models can be developed separately as long as they implement a service that, given the component name and its configuration, returns a derived quantity associated with the component (for example, power, area, or latency). Different models can be attached to the same component, which accesses a model through a standard interface independently from the model's implementation. Similarly, different nodes and links can belong to a platform, which instantiates them through a standard interface independently from the components' implementation.

The COSI framework can help develop design flows for interconnect synthesis. It can also help researchers and designers develop models, library elements, and optimization algorithms; compare different optimization strategies; and evaluate the efficiency of different heuristic algorithms. The "Other component composition frameworks" sidebar provides helpful comparisons with COSI.

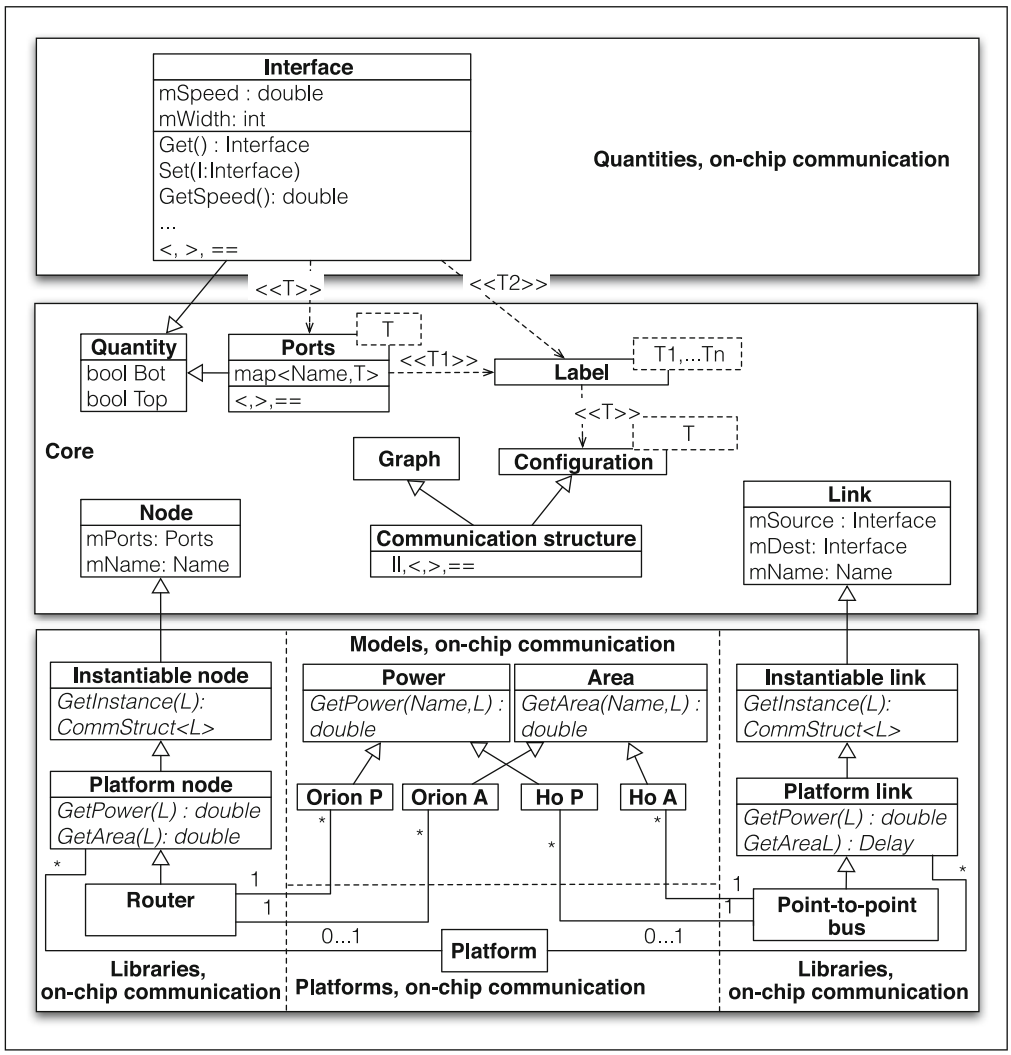

Figure 3. UML (Unified Modeling Language) class diagrams of the data structures implemented in COSI for the network-on-chip (NoC) application domain. (Orion P, A: power and area models derived with the Orion tool[6]; Ho P, A: power and area models presented by Ho et al.[7])

OCC synthesis flow in COSI

Figure 4 shows how COSI-OCC, a design flow for OCC synthesis, was built on top of COSI. The sequence of operations typically performed to synthesize an OCC architecture is represented by numbered arrows. The input to COSI-OCC is a project file containing pointers to the communication specification and to the library. All I/O files are in XML format. The communication specification contains a list of IP cores and intercore communication constraints. The specification is parsed to yield an internal communication structure N_C in which each node represents an IP core and the links represent constraints. The library-model file contains a description of each library element and its associated models. The platform is constructed by taking components from the library and attaching models to them. The platform also contains rules such as topological constraints, a restriction on the position of nodes, and implementation requirements such as deadlock freedom. In addition, the project file includes optimization parameters such as the relative weights of power and area variables in the objective function.

If there are unplaced IP cores, Parquet[9] is used for chip floorplanning (step 2 in Figure 4). Step 3 of a

Other component composition frameworks

The Communication Synthesis Infrastructure (COSI) provides a synthesis-oriented infrastructure for the design of interconnection networks. Complex interconnections are built by composing elements from a library, and COSI provides the framework to facilitate the definition of library elements and composition rules. From this point of view, COSI-OCC (on-chip communication) belongs to the class of component composition frameworks (CCFs) such as Balboa,[1] Liberty Simulation Environment (LSE),[2] the metamodeling-based CCF (MCF),[3] Spartacas,[4] and Abrie.[5]

We can compare COSI with other CCFs at the component-composition language (CCL) level. CCL is the language the CCF uses to define a design's structural aspect. The criteria selected for comparison are the formal framework underlying the CCL, what the CCL captures, the supported library of components, the supported composition rules, and the features provided by the CCF. Balboa and LSE are not based on a rigorous formal model (as also noted by Mathaikutty and Shukla[3]). Both frameworks assume that the library of components is described in an imperative language such as C or C++ and that the components are exported using an interface description language. The only composition rule is that the types of connected components must be compatible. Both frameworks can perform type inference.

The metamodeling-driven CCF (MCF) is based on a metamodel captured in UML (Unified Modeling Language). Like Balboa and LSE, MCF supports components described at the RTL or at the transaction level. The two levels can communicate through transactors. Composition rules are captured using the Object Constraint Language, which can express constraints on entities (components) and relationships (connections). MCF is geared toward SystemC IP libraries and can perform consistency checking and type inference.

Unlike the previous tools, Spartacas captures the architectural specification and the library components using the same language, Rosetta.[6] This is because Spartacas defines the composition rules on the set of input and output values of components, and it features automated component adaptation. In this context, the problem specification is the desired I/O relation, and the adaptation selects components from the library to match the problem specification.

The Abrie framework has been developed mainly for software reuse.[5] A system is defined by components and connections. Each component has a type, a role, and a set of ports. There are constraints on the ports that can be connected to certain roles. Abrie also provides automated component selection.

COSI has been developed for the synthesis of communication infrastructures. It is based on a formal model centered on a mathematical object called a *communication structure*, which captures components in terms of nodes and links. In a departure from the other CCFs, we attach performance metrics to components and define ordering relations on the basis of those metrics. Because we target synthesis rather than simulation, our approach to model libraries resembles that of Spartacas. A library is a set of communication structures and is captured in the same way as the problem specification. In contrast to the other CCFs, COSI lets users define the composition rules such that different systems can be obtained using the same library. Composition rules can be defined in a general way as relations between components and their properties. Finally, COSI features automatic synthesis of communication structures.

Remarkably, COSI's output could very well be a description of a communication structure in one of the other CCFs. At the same time, once a system is simulated in a CCF, the communication requirement on each connection among components can be passed to COSI, which uses synthesis to automatically refine the connections into a more sophisticated communication structure.

References

1. F. Doucet et al., ''Balboa: A Component-Based Design Environment for System Models,'' *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 12, Dec. 2003, pp. 1597-1612.
2. M. Vachharajani, N. Vachharajani, and D. August, ''The Liberty Structural Specification Language: A High-Level Modeling Language for Component Reuse,'' *Proc. Conf. Programming Language Design and Implementation* (PLDI 04), ACM Press, 2004, pp. 195-206.
3. D.A. Mathaikutty and S.K. Shukla, ''MCF: A Metamodeling Based Component Composition Framework—Composing SystemC IPs for Executable System Models,'' *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 7, July 2008, pp. 792-805.
4. B. Morel, ''Spartacas: Automating Component Reuse and Adaptation,'' *IEEE Trans. Software Eng.*, vol. 30, no. 9, Sept. 2004, pp. 587-600.
5. Y. Chen and B.H.C. Cheng, ''Facilitating an Automated Approach to Architecture-Based Software Reuse,'' *Proc. 12th IEEE Int'l. Conf. Automated Software Engineering* (ASE 97), IEEE CS Press, 1997, pp. 238-245.
6. P. Alexander, D. Barton, and C. Kong, *Rosetta Usage Guide*, tech. report, Information and Telecommunications Technology Center, Univ. of Kansas, 2000.

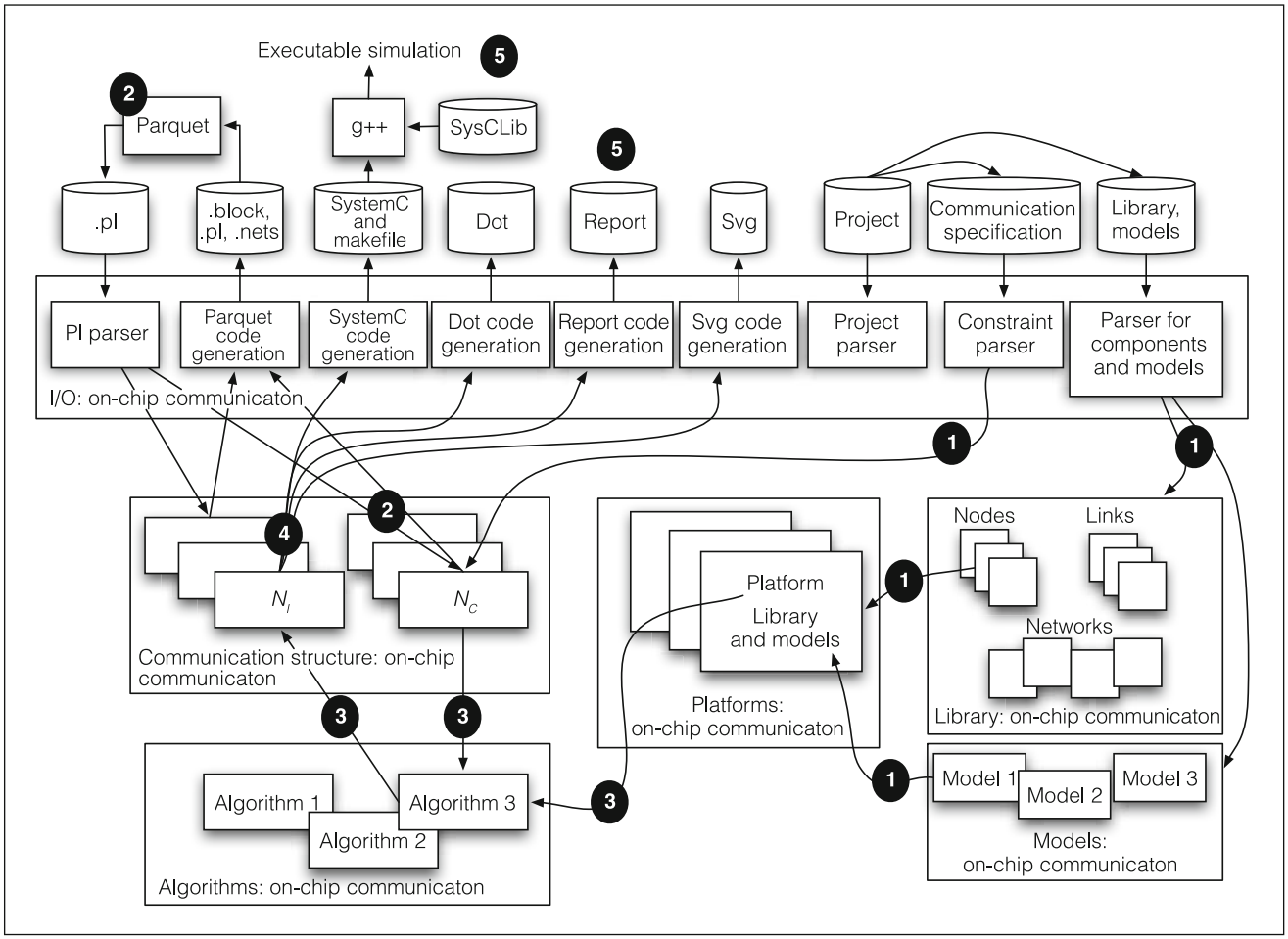

Figure 4. The use of COSI to set up a complete design flow for on-chip communication (OCC). (Numbers indicate steps in one of the design flows implemented in COSI-OCC.)

typical flow involves selecting an algorithm that takes the specification and the platform description and derives a communication implementation N_I. The COSI-OCC distribution includes a set of algorithms to solve some variants of the communication synthesis problem. Step 4 generates the outputs that are then used for analysis. COSI-OCC includes a set of code generators to produce a scalable vector graphics (SVG) representation and a Graphviz DOT language logical representation of N_I. A SystemC netlist can be generated from N_I by assembling the SystemC view of each element, which can be instanced from the library contained in SysCLib, also part of the COSI-OCC distribution. Generation of the SystemC netlist is a further refinement of N_I that requires establishing the computation of protocol-related information such as the weights for the weighted fair-queuing algorithm, which the routers use to schedule flits. The "Other OCC design frameworks" sidebar briefly compares some other approaches. (For more information on COSI-OCC, see http://embedded.eecs.berkeley.edu/cosi.)

Experimental results for COSI-based NoC design

Because of space limitations, we report only a set of results concerning the implementation and test of four network optimization algorithms. Examples of how COSI has been used for other applications are available elsewhere. In particular, we studied the power consumption, area, and performance trade-offs for an NoC as a function of router size at different technology nodes.[5] We also interfaced COSI with an integer-linear-programming solver to evaluate the quality of a heuristic algorithm that we developed called heuristic H1. Elsewhere, we evaluated how the accuracy of wire models affects system-level design choices.[8] We integrated accurate delay models for

Other OCC design frameworks

Comparing the Communication Synthesis Infrastructure (COSI) on-chip communication with other methodologies and tools for OCC design, we limit our attention to design frameworks rather than NoC implementation techniques. A fundamental difference between our approach and the work reviewed here is that we do not aim at solving one particular OCC problem. Instead, we provide an infrastructure to manage any communication design problem, including the specification, the library elements and performance metrics associated with them, and the composition rules for building complex networks.

The NetChip design environment provides a complete solution for designing, simulating, and synthesizing NoCs.[1] The specification is captured by a core graph—a communication structure that associates bandwidth requirements with end-to-end constraints. A platform instance is captured by a communication structure whose links offer a specified capacity. In NetChip, the library contains a set of predefined topologies (meshes, tori, and so forth). NoC optimization works by solving one optimization problem for each topology in the library. Once an implementation is found, other tools generate a SystemC executable simulation.

Gerstlauer et al. have developed a methodology that distinguishes two abstraction levels: the network level and the link level.[2] The specification exists as a set of processing elements and a set of channels between them. The first step is assigning channels to buses. Users can rely on a library of buses, masters, slaves, and bus bridges, which transfer data from one bus to another. The second step is optimizing the link design. Although this approach accounts for a large vector of quantities, including the protocol behavior, its present focus is on bus-based communications, and it does not support automatic synthesis of generic networks.

Carnegie Mellon University's SlicNet project (http://www.ece.cmu.edu/~sld/research/soc.php) has led to the development of various analysis and optimization techniques for NoC design. In general, this project focuses on NoCs with predefined topologies. For instance, Ogras and Marculescu propose a method to improve the performance of an NoC with a regular mesh topology through the careful insertion of additional long links.[3] Other contributions to the SlicNet project include direct and indirect statistical models for critical NoC metrics as well as FPGA prototyping of NoCs.

Srinivasan et al. discuss a formulation of the multicommodity flow problem with degree constraints that is very similar to the one solved by the H1 algorithm, but for a specific library of communication components that contains routers and links.[4] The problem is formulated as an integer linear program; therefore, highly nonlinear constraints, such as deadlock freedom, cannot be taken into account. Also, the cost function is considered to be linear in the value of the bandwidth.

The QNoC architecture focuses on quality of service.[5] This NoC provides different service classes for network traffic, which corresponds to considering the domain of bandwidth quantity as a set of sets of commodities, where each commodity is a source-destination flow with its own quality of service. QNoC provides tools to map an application onto a regular mesh and then remove unused nodes and links.

The Apsra methodology computes deadlock-free routing functions such that the routing algorithm's adaptiveness is not compromised.[6] Routing is modeled much like the definition of routing tables in COSI-OCC.[7] Moreover, as in COSI-OCC, the definition of deadlock is based on the channel dependency graph. For a given platform instance, the algorithm provided in Apsra solves an optimization problem in which the decision variables are the routing tables for each node, and the objective function is a measure of the routing protocol's adaptiveness. The composition rule, which becomes a constraint in the optimization problem, is that the network must be deadlock free.

GeNoC, instead of focusing on NoC optimization, provides a formal approach to NoC verification: given the description of an NoC implementation in Common Lisp, the ACL2 theorem prover verifies that the system implements the required specification.[7]

References

1. D. Bertozzi et al., ''NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip,'' *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 2, Feb. 2005, pp. 113-129.
2. A. Gerstlauer et al., ''Automatic Layer-Based Generation of System-on-Chip Bus Communication Models,'' *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, Sept. 2007, pp. 1676-1687.
3. U. Ogras and R. Marculescu, ''Application-Specific Network-on-Chip Architecture Customization via Long-Range

Link Insertion," *Proc. Int'l Conf. Computer-Aided Design* (ICCAD 05), IEEE CS Press, 2005, pp. 246-253.
4. K. Srinivasan, K.S. Chatha, and G. Konjevod, "Linear-Programming-Based Techniques for Synthesis of Network-on-Chip Architectures," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 4, Apr. 2006, pp. 407-420.
5. E. Bolotin et al., "QNoC: QoS Architecture and Design Process for Network on Chip," *J. System Architecture*, vol. 50, nos. 2-3, Feb. 2004, pp. 105-128.
6. S.-V.M. Palesi and R. Holsmark, *Apsra: A Methodology for Design of Application Specific Routing Algorithms for NoC Systems*, tech. report DIIT-TR-01-060406, Dept. of Computer Science and Telecommunication, Univ. of Catania, Italy, 2006.
7. D. Borrione et al., "A Generic Model for Formally Verifying NoC Communication Architectures: A Case Study," *Proc. 1st Int'l Symp. Networks-on-Chip* (NOCS 07), IEEE CS Press, 2007, pp. 127-136.

wires that rely on the characterization of intermediate and global metal lines and minimum-size inverters at different technology nodes. In particular, it was possible to evaluate the sensitivity to modeling errors of system-level optimization techniques. Finally, we showed how the infrastructure can be used to solve the communication synthesis problem for bus-based building automation networks.[3]

To show the flexibility of the tools within the COSI environment, we describe the implementation of four different algorithms for NoC optimization: H1, H2, HStar, and Mesh.

H1 is a two-step heuristic algorithm that solves the degree-constrained multicommodity flow problem.[5] After SoC specification parsing and running of the floorplanner, the first step consists of finding an initial solution without considering constraints on the node degrees. In the second step, an iterative procedure removes degree violations by deleting links or adding routers. The same technique used by algorithms for global routing finds the initial solution. The procedure finds a path for each constraint, one at a time. (Actual implementation of the procedure depends on the composition rules.) If the implementation network satisfies the degree constraints, the algorithm stops and returns a solution. Otherwise, the second step of the algorithm uses a "rip-up and reroute" approach to remove one link at a time. For each link connected to nodes with degree violation, the algorithm attempts to reroute all source-destination paths containing that link and replace them in the communication implementation with new paths. However, if one of these paths cannot be removed, say because of bandwidth constraints, the algorithm backtracks by reinserting the link and all the paths. If the rerouting procedure finds an implementation that satisfies the composition rules, the algorithm ends successfully. If the procedure fails, a new attempt to reach a feasible solution is made after the addition of a new node (router). The idea is that when a new node is added, multiple links entering or exiting a node with degree violations can be merged into or split from one link using the new node, thereby reducing the degree of the node. However, if no node can be added (say, because delay constraints would be violated), the algorithm ends with an empty implementation, thus implying that no solution was found.

The two steps can be combined in a different way to produce an alternative heuristic H2. In this heuristic algorithm, the communication constraints that are part of the specification are considered one at a time, as in H1. However, in a departure from H1, degree violations are now corrected as soon as they appear. After routing constraints along a path, the algorithm checks whether there are nodes with degree violations and if so attempts to remove them using the same procedure adopted in the second step of heuristic H1.

Both H1 and H2 solve the NoC optimization problem using a synthesis approach whereby the network is derived in a constructive way by adding components as needed after SoC floorplanning is complete. A different approach is to map the SoC cores onto a predefined regular network topology such that the number of hops between communicating cores is minimized. Such minimization corresponds to minimizing power consumption and delay. We implemented an optimal mapping algorithm along the lines of the one presented by Murali and De Micheli.[10] This algorithm, called Mesh, iteratively improves an initial mapping on a regular mesh topology by placing cores that communicate rapidly into mesh nodes that are topologically close to one another. At each iteration, two cores are swapped in the mapping and new paths are selected in the mesh network such that the number of hops between sources and destinations is minimized. Each time the total communication cost decreases, the solution is saved as the current optimal solution. Finally, the unused network resources (links, ports, and routers)

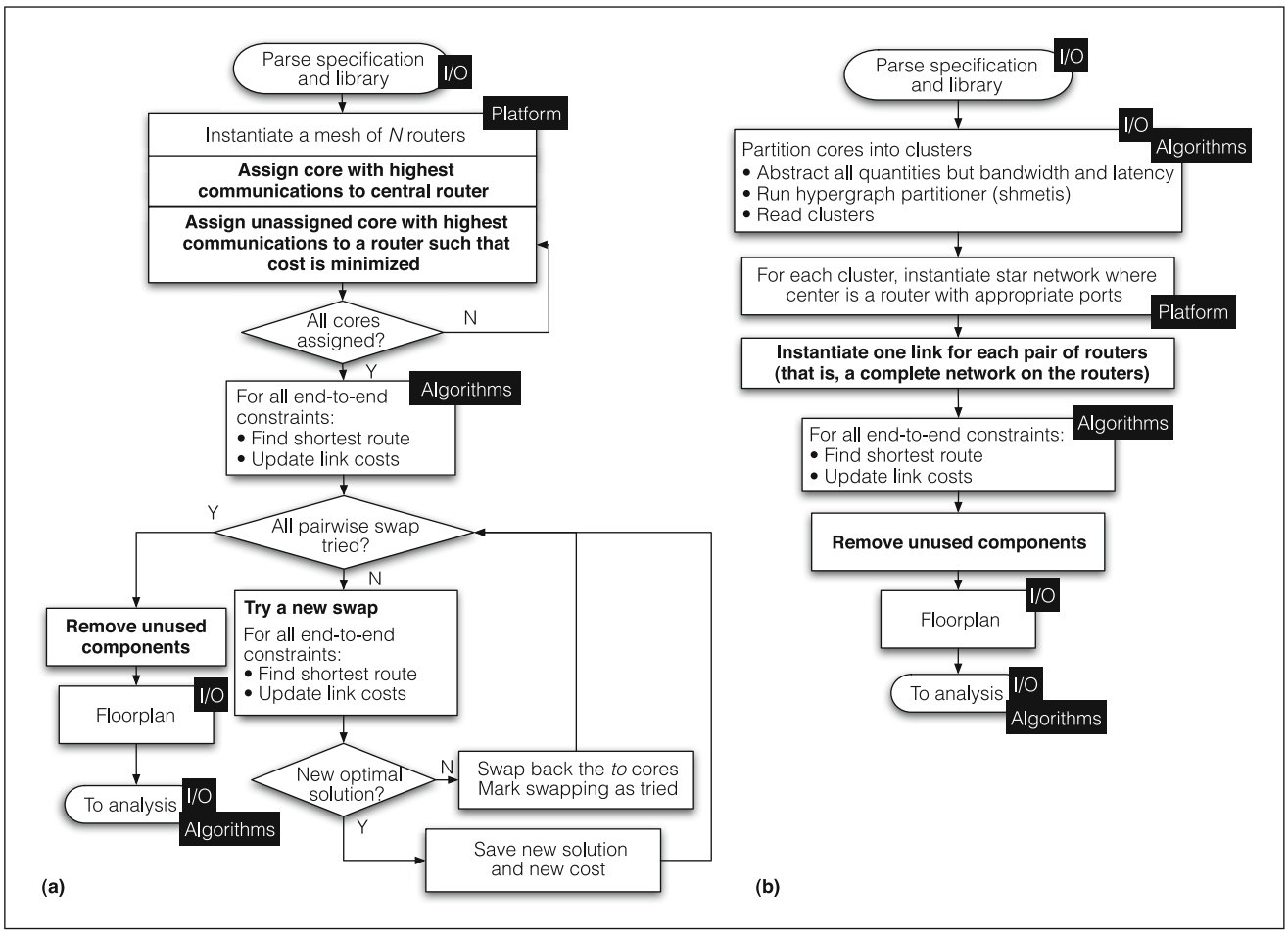

Figure 5. Flowcharts of the Mesh (a) and HStar (b) algorithms.

are removed before the algorithm runs a floorplan of the entire chip, including the NoC.

The Hstar algorithm is an example of the many clustering-based NoC optimization approaches proposed in the literature. It partitions the cores in the specification into n clusters according to their communication requirements such that the total communication bandwidth between cores belonging to different clusters is minimized. A star topology serves to implement each partition, and the n stars are connected as part of a higher-level network. This network is first assumed to be completely connected. Then, the algorithm computes optimal routes for each source-destination pair. Again, the unused network resources are removed before the algorithm runs a floorplan of the entire chip. We call this algorithm Hstar(n), where n is the number of partitions.

Figure 5 shows the flowcharts of the Mesh and HStar algorithms, and Figures 6 through 8 show the optimal NoC implementation that each implemented algorithm returns for a given SoC benchmark. In each flowchart in Figure 5, we use bold text to indicate the parts we had to develop anew for the given algorithm. Indeed, the development, debug, and performance assessment of a new algorithm in COSI requires a relatively small effort, because the developer can take advantage of the services offered by COSI that share the same underlying model—that is, the communication structures. For example, our implementation of Mesh took approximately one day, but implementing Hstar(n) took only half a day because many procedures are shared with the Mesh algorithm.

The bar charts in Figure 9 compare the results from running the four NoC optimization algorithms on a set of benchmarks from the literature.[11,12] We used a 90-nm technology, the Ho model for wires,[7] and the Orion model for routers.[6] The platform is configured to contain routers with at most five input and five output ports for H1, H2, and Mesh; and at most 10 input and 10 output ports for Hstar(n). Flit width is fixed at 32

bits, and clock frequency at 1 GHz. Note that the number of hops between a source and a destination is structurally bounded to three in the case of Hstar(n), at the price of higher power consumption. Mesh networks are resource hungry. Moreover, communicating cores can be separated by many hops because of topological constraints. Heuristics H1 and H2 return highly customized NoCs that match the number of hops of the NoC returned by HStar, while being more efficient in terms of power dissipation and area occupation.

WE EXPECT THAT COSI, by providing an extensible and flexible framework, will promote collaboration among NoC researchers and designers with complementary skills. Although COSI-OCC currently targets NoCs, it will be extended to other on-chip interconnect structures such as buses, crossbars, and direct connections, allowing SoC and microprocessor designers to choose the most effective infrastructure without bias toward a particular solution. Designers can also use COSI to define design flows for other application domains by changing the quantities that characterize communication constraints, cost, and performance, and by modeling the library elements to be used in a particular application. We've used COSI to build a design flow for the synthesis of wired and wireless networks for building automation systems.[3,4] ■

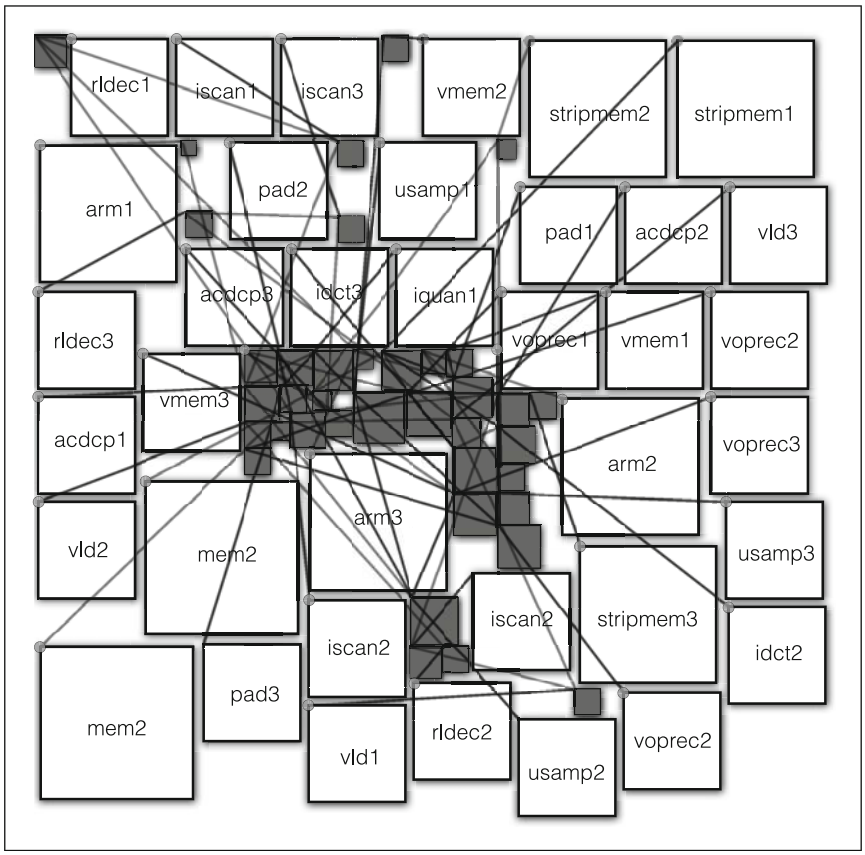

Figure 6. Floorplan of the video object plane decoder (tVOPD) SoC after synthesis by Mesh.

Acknowledgments

This research is partially supported by the Gigascale Systems Research Center (GSRC), one of five research centers funded under the Focus Center Research Program (FCRP), a program of the Semiconductor Research Corp.

■ References

1. A. Ferrari and A.L. Sangiovanni-Vincentelli, "System Design: Traditional Concepts and New Paradigms," *Proc. Int'l Conf. Computer Design* (ICCD 99), IEEE CS Press, 1999, pp. 1-12.
2. A. Sangiovanni-Vincentelli, "Quo Vadis SLD? Reasoning about Trends and Challenges of System Level Design," *Proc. IEEE*, vol. 95, no. 3, Mar. 2007, pp. 467-506.
3. A. Pinto, L.P. Carloni, and A.L. Sangiovanni-Vincentelli, "A Communication Synthesis Infrastructure for Heterogeneous Networked Control Systems and Its Application to Building Automation and Control," *Proc. 7th ACM and IEEE Int'l Conf. Embedded Software* (EMSOFT 07), ACM Press, 2007, pp. 21-29.
4. A. Pinto et al., "Synthesis of Embedded Networks for Building Automation and Control," *Proc. 2008 American Control Conf.*, http://www.eecs.berkeley.edu/~fischion/Publications/acc2008.pdf.
5. A. Pinto, L.P. Carloni, and A.L. Sangiovanni-Vincentelli, *A Methodology and an Open Software Infrastructure for the Constraint-Driven Synthesis of On-Chip Communications*, tech. report UCB/EECS-2007-130, Dept. of Electrical Engineering and Computer Science, Univ. of California, Berkeley, 2007.
6. H.S. Wang et al., "Orion: A Power-Performance Simulator for Interconnection Networks," *Proc. 35th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (Micro 35), IEEE CS Press, 2002, pp. 294-305.

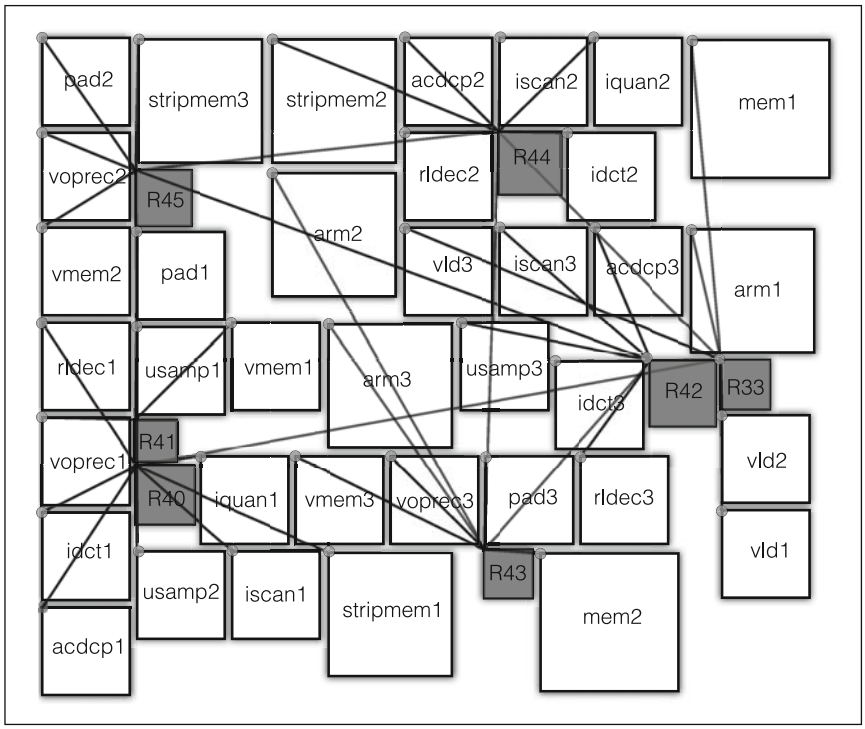

Figure 7. Floorplan of the tVOPD SoC after synthesis by HStar.

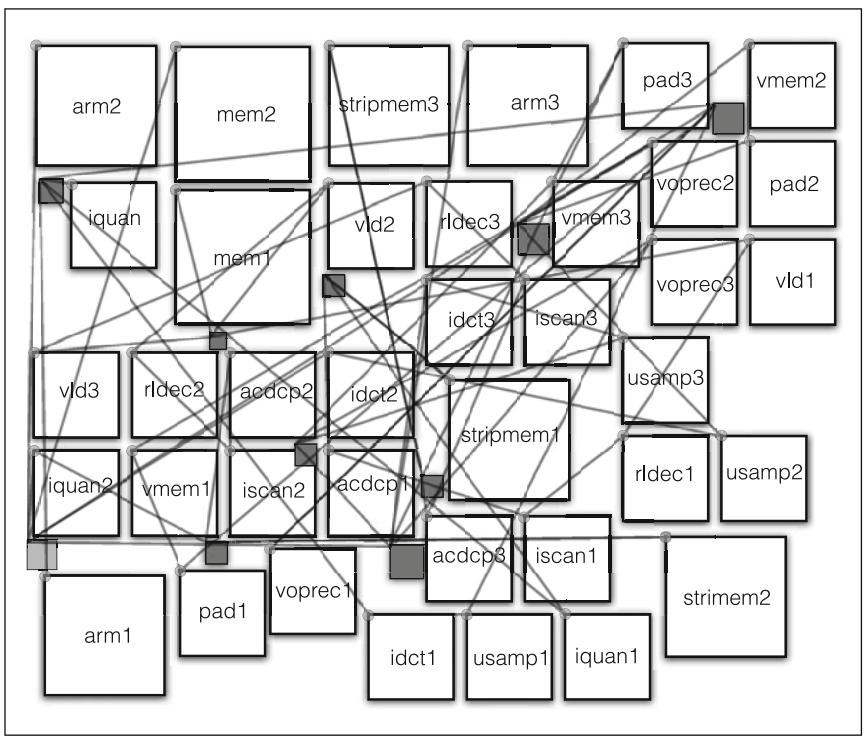

Figure 8. Floorplan of the tVOPD SoC after synthesis by H1.

7. R. Ho, K.W. Mai, and M.A. Horowitz, "The Future of Wires," *Proc. IEEE*, vol. 89, no. 4, Apr. 2001, pp. 490-504.
8. L. Carloni et al., "Interconnect Modeling for Improved System-Level Design Optimization," *Proc. 13th Asia and South Pacific Design Automation Conf.* (ASPDAC 08), IEEE CS Press, 2008, pp. 258-264.
9. S.N. Adya and I.L. Markov, "Fixed-Outline Floorplanning: Enabling Hierarchical Design," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, Dec. 2003, pp. 1120-1135.
10. S. Murali and G. De Micheli, "SUNMAP: A Tool for Automatic Topology Selection and Generation for NOCs," *Proc. 41st Design Automation Conf.* (DAC 04), ACM Press, 2004, pp. 914-919.
11. A. Pullini et al., "NoC Design and Implementation in 65 nm Technology," *Proc. 1st Int'l Symp. Networks-on-Chips* (NOCS 07), IEEE CS Press, 2007, pp. 273-282.
12. D. Bertozzi et al., "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 2, Feb. 2005, pp. 113-129.

Alessandro Pinto is a research scientist in the Embedded Systems and Networks Group at the United Technologies Research Center. His research interests include networked embedded systems, with particular emphasis on design methodologies and tools. He has an MS and a PhD in electrical engineering and computer sciences from the University of California, Berkeley. He is a member of the IEEE.

Luca P. Carloni is an assistant professor in the Department of Computer Science at Columbia University, New York. His research interests include design tools and methodologies for integrated circuits and systems, distributed embedded-systems design, and design of high-performance computer

systems. He has an MS in engineering and a PhD in electrical engineering and computer sciences from the University of California, Berkeley. He is a member of the IEEE and the ACM.

Alberto Sangiovanni-Vincentelli holds the Buttner Endowed Chair of Electrical Engineering and Computer Science at the University of California, Berkeley. He is a cofounder of Cadence Design Systems and Synopsys. His research interests include system-level design, embedded and hybrid systems, and electronic design automation. He has a Dr Eng in electrical engineering and computer sciences from Politecnico di Milano. He is a Fellow of the IEEE, and is a member of the National Academy of Engineering and the ACM.

■ Direct questions and comments about this article to Alessandro Pinto, apinto@eecs.berkeley.edu.

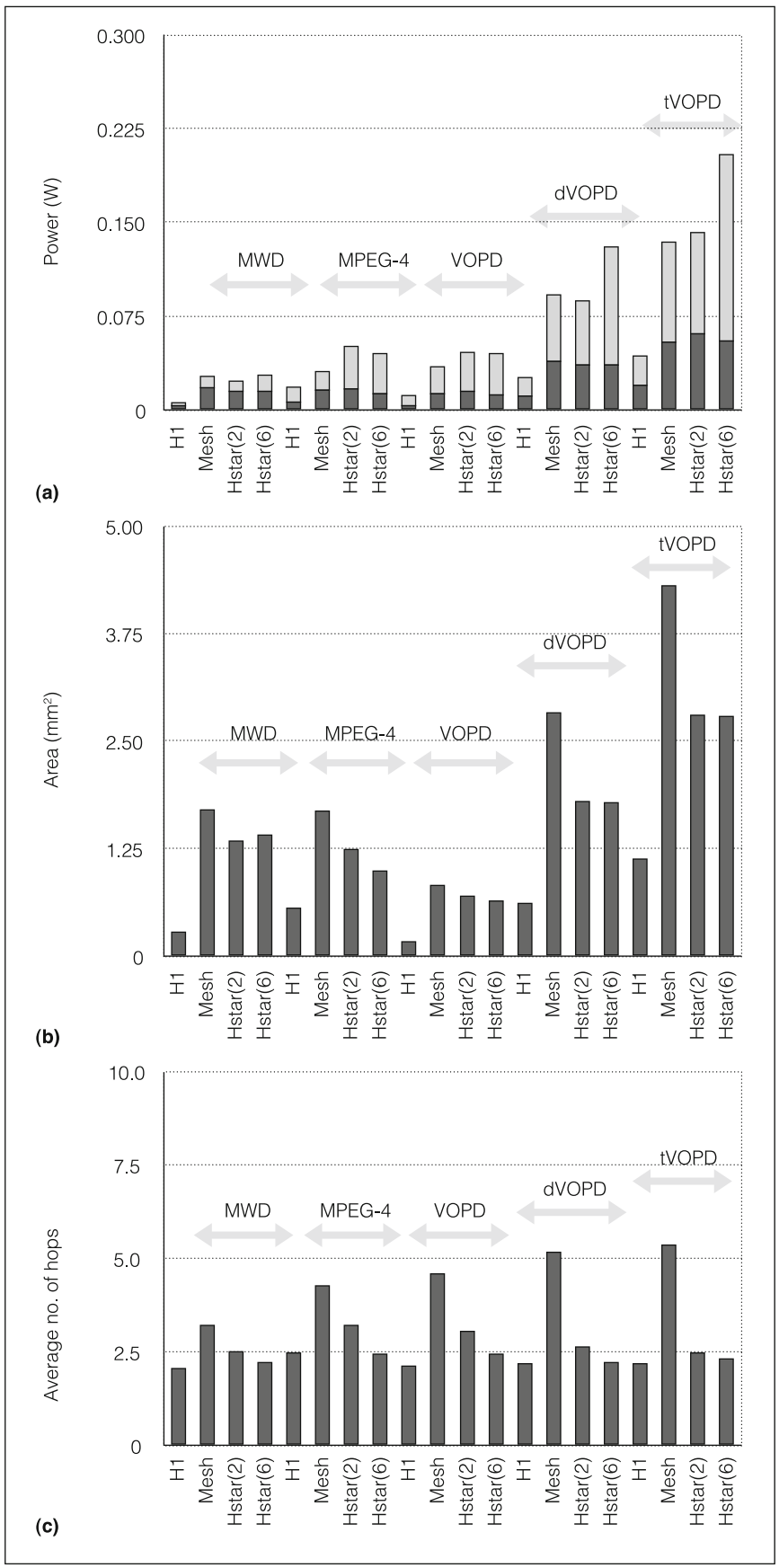

Figure 9. Results from running various NoC algorithms developed in COSI on a set of benchmarks taken from the literature:[11,12] a multiwindow display (MWD), MPEG-4, a video object plane decoder (VOPD), two VOPDs sharing a memory (dVOPD), and three VOPDs sharing two memories (tVOPD). The charts compare power (a), area (b), and average number of hops (c).