COSI: A Public-Domain Design Framework for the Design of Interconnection Networks

Alessandro Pinto
Luca Carloni
Alberto L. Sangiovanni-Vincentelli

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-22
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-22.html

March 20, 2008

COSI: A Public-Domain Design Framework for the Design of Interconnection Networks

Alessandro Pinto, Luca P. Carloni and Alberto L. Sangiovanni-Vincentelli *

The COmmunication Synthesis Infrastructure (COSI), a public-domain design framework for the design exploration and synthesis of interconnection networks, is presented. The framework embodies a methodology based on the platform-based design principles and is used to define specific design flows for a variety of applications. In this paper, we focus on a design flow for on-chip interconnect design.

Time-to-market, NREs, and error-free implementation requirements are exposing the increasing importance of composable designs, whereby complex systems are built out of possibly pre-designed and pre-verified components. Composability implies that the components maintain their properties/behavior when composed; thus interfaces and communication become essential elements for methodologies of this kind. For example, system-on-chip architectures, distributed embedded systems and even microprocessors are designed today using pre-existing IPs. In the case of microprocessors, the classical single-core architecture has been largely abandoned due to the difficulties of increasing clock speed with strict power consumption constraints and of verifying its correctness. Multi-core architectures are the choice for future generations of microprocessors. These architectures require great attention to the design of the interconnect infrastructure and of the communication protocols. In the design of distributed embedded controllers such as UAV navigation systems, communication plays a fundamental role in ensuring the correct behavior of the design since delay and throughput affect the control algorithm in substantial ways. In this paper, we focus on the design of the communication infrastructure for on-chip architectures, albeit our approach can be (and has been) extended to distributed system design. In particular, we are interested in the design of cost-effective *networks-on-chip (NoC)* that must meet a set of given throughput and latency constraints.

The problem of designing an optimal network is not new. It has been extensively studied by computer scientists and operations researchers for the design of data networks and transportation networks [7]. Most network optimization problems are NP-hard, but for many of them approximation algorithms have been proposed. For several important applications, the design problem reduces to determining which network among those belonging to a certain class (e.g. having a particular topology) satisfies a given budget constraint and minimizes a given objective function. These problems, known as bi-criteria network design problems, are also NP-hard. The optimal solution can be approximated with an algorithm running in polynomial time only if the budget constraint is relaxed [12].

The large body of approximation and heuristic algorithms that were developed for network optimization can be applied to the synthesis and optimization of on-chip interconnection networks provided they are properly adapted and combined with accurate models of performance and cost of the building blocks. The opportunities offered to chip designers are noteworthy. However, if the designer has to spend time to implement these algorithms fighting

*A. Pinto and A. Sangiovanni-Vincentelli are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720, USA ({apinto,alberto}@eecs.berkeley.edu). L.P. Carloni is with Department of Computer Science, Columbia University in the City of New York, New York, NY 10027, USA (luca@cs.columbia.edu).

with design software, then these opportunities will remain largely untapped.

COSI (COmmunication Synthesis Infrastructure) is a public-domain design framework constructed to allow researchers and designers to contribute, combine, and compare optimization algorithms, communication protocols, partial designs, and models for interconnection design. COSI embodies a methodology based on the Platform-Based Design paradigm [5, 13]. Specifically, COSI enforces a clean separation among 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 that are used to explore the design space. Adopting this methodology allows comparing different interconnection topologies and different building blocks thus smoothing out preconceived ideas about efficiency of particular interconnection schemes.

At present, COSI includes various utility functions to: floor-plan a chip, parse the floor-plan results and derive an internal representation, analyze the network synthesis results, compute metrics of interest such as cost and performance of the entire network, generate graphical representations of the network as well as SYSTEMC models that can be used for simulation. This framework can be used to provide robust design flows for interconnect synthesis and can also help researchers and designers in the development of models, library elements and optimization algorithms, in the comparison of different optimization strategies and in the evaluation of the efficiency of different heuristic algorithms.

In this paper, we briefly present the model for communication synthesis that is the foundation of COSI. Then, we describe the key aspects of the software engineering of COSI . We show also that it is possible to use COSI to develop design flows for on-chip network synthesis and to evaluate alternative optimization approaches. Finally, we point out that COSI can be applied to the synthesis of network in different application domains, e.g. control networks for building automation.

1 A Model for Communication Synthesis

In this section, we summarize the COSI model that consists of: quantities that "measure" performance of a communication component, communication structures that capture the behavior and the structure of the components and of composition rules that allow to form composite components out of existing ones. A complete presentation of this model is given in [9].

Design constraints and component capabilities (i.e. performance figures) are expressed with *quantities*. A quantity q ranges on a partially-ordered domain D_q. We assume that the domain of a quantity 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 the ports of a component are pairs composed of a tag and an interface specification. Figure 1(a) shows the domains of the quantities involved in the description of an interface which 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 if an interface is input, output or bidirectional. The domain of a quantity is ordered according to a relation that ranks each value in terms of performance and/or constraint. For instance, the speed and width of an interface follow the ordering of natural numbers since 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 is sorted according to the order induced by the single quantities. Figure 1 shows the ordering relation among some elements of $D_w \times D_f \times D_{io}$.

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 $N(\mathcal{C},Q,L)$ where $\mathcal{C}$ is a set of components (i.e. nodes and links), Q is a set of quantity variables and L is a set of configurations, i.e a set of functions $l : \mathcal{C} \rightarrow D_Q$ 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

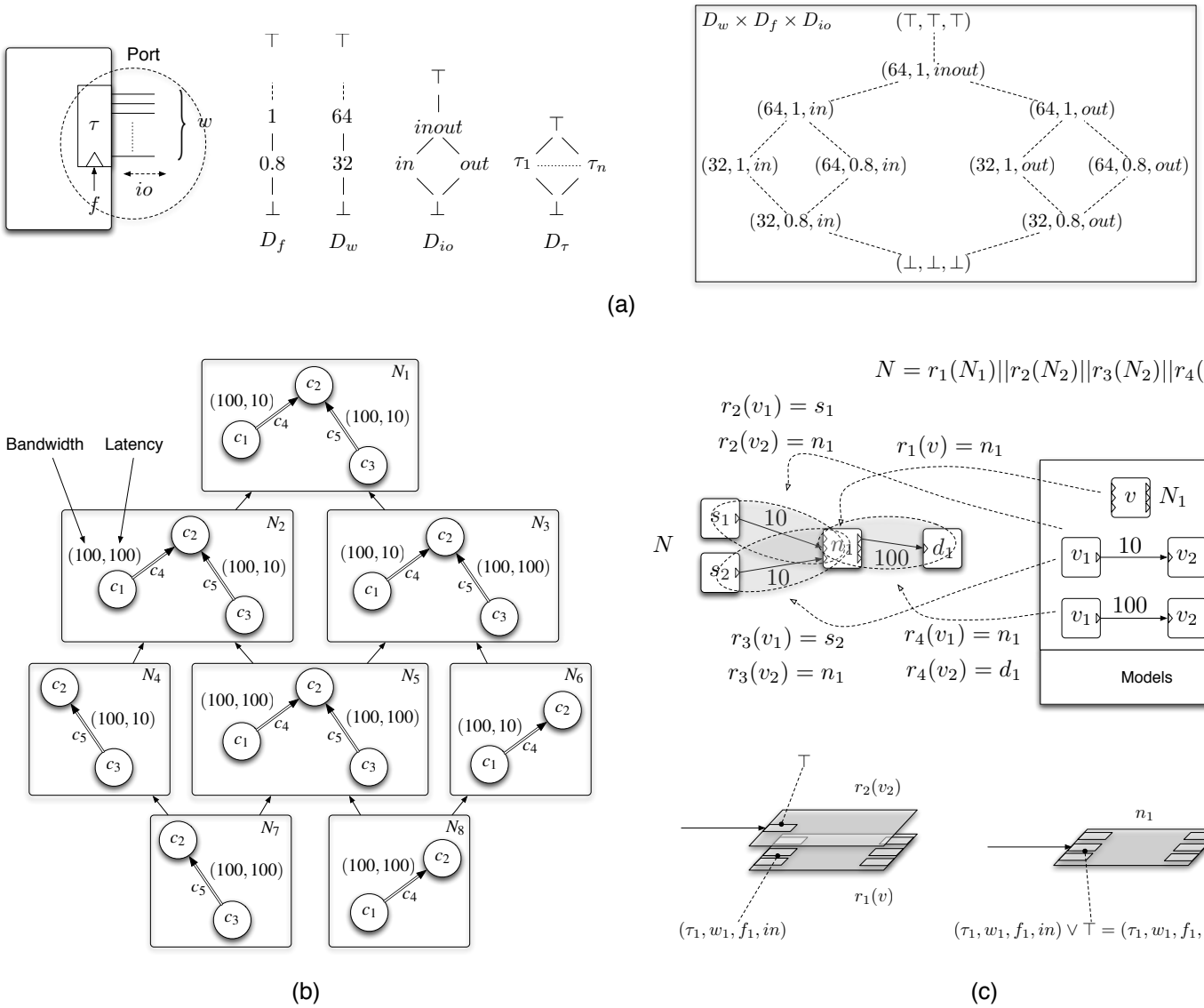

Figure 1: Example of quantity domain (a), communication structure ordering (b) and example of a library of components, their instantiation and their composition (c).

defined on D_Q, and by component containment. For instance, Figure 1(b) shows the ordering relation among a subset of communication structures where the set Q contains two quantities: b that represents bandwidth and t that represents delay. Intuitively, $N_7 \leq N_5$ because N_7 has less components and and the components that are in common are configured to perform better in N_5 than in N_7. N_5 and N_6 are incomparable.

The specification of the communication requirements of an SoC is captured by a communication structure $N_C(C, Q_C, L_C)$ where 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, i.e. 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 $N_i(C_i, Q_P, L_i) \in \mathcal{L}$, called *library elements*, and by a composition rule denoted by $||$. The set 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 1(c) shows a simple example of library of communication structures that contains a node with three inputs and three outputs and two links, each able to sustain a different bandwidth level. Library elements can be instantiated by *renaming* their nodes. For instance, N_1 is renamed by a renaming

function r_1 such that vertex v ends up being called n_1. Instances of library elements can be composed together to form larger networks such as the communication structure N in Figure 1(c). The definition of the composition operator can be rather involved and depends on the properties that the composite network is required to satisfy. In general, the composition of two communication structures contains the union of their components and combines the configurations appropriately (e.g., the set of flows on a common link is the union of the set of flows associated with that link in the communication structures being composed). A particular composition of library elements is called *platform instance*, and the set of all valid platform instances is called *platform*.

An example of how quantities are combined is shown in Figure 1(c). Node v_2 belonging to the link of capacity 10 is renamed as n_1 and so is node v. The interface of v_2 in the library has value $\top$ meaning that the link is able to connect any two interfaces while the interface of n_1 is the tuple (τ_1, w_1, f_1, in). When the two nodes are combined together we take the join of the two interfaces with respect to the order shown in Figure 1(a), (i.e. the "least common denominator" of their features).

When marching down from the specification towards the implementation of a communication system, more details are added to the communication structures by augmenting and refining the set of quantities associated with the components. For instance, routing tables are added after the choice of the routing algorithm is made. Communication structures at different abstraction levels can be related by abstraction functions.

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

2 The Software Infrastructure

The COSI software has a matrix organization (Figure 2) where the columns correspond to aspects of the communication synthesis design flows while the rows represent different application domains. The elements that characterize a design flow are:

1. the quantities and communication structures that define the levels of abstraction at which specification, platform, and implementation are captured;

2. the library of communication components and the performance and cost models used to annotate derived quantities and compute costs, as well as the composition rules;

3. a platform data structure representing the library and the rules so that the synthesis algorithms can operate;

4. the environment where the network operates, e.g. floor-planning information in the case of chips, and building geometry in the case of building automation systems;

5. the input/output functions such as parsers and code generators that ease the process of specifying the communication problem and analyzing the results.

We present in detail the first two rows of the matrix organization.

The *core* package provides basic definitions for widely-used quantities such as positions, flows, and ports. It also provides the definitions of model graphs and a set of basic algorithms running on weighted graphs. The on-chip communication package provides specialized definitions for port interfaces (including, for instance, the number of virtual channels of a router input, the buffer length, and the clock speed), the geometry of a component

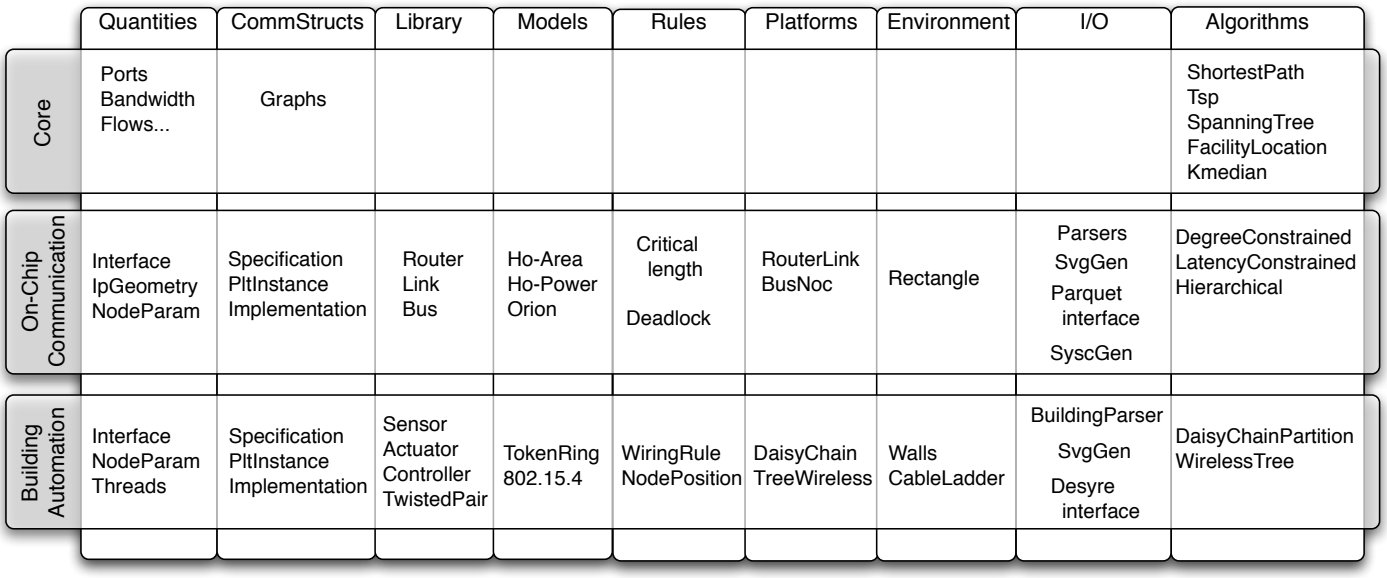

Figure 2: Organization of the COSI software.

and implementation parameters for nodes and links (e.g. global or local interconnect, 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 that were derived with the Orion tool [14], the analytical model for metal wires presented by Ho *et al.* [6], as well as a more recent set of accurate wire models presented in [3]. All the models are provided for various technology processes, including 90, 65, and 45*nm* technology.

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, which are given in XML format, to generate graphical views of the synthesized network, and to produce cycle-accurate SYSTEMC descriptions of the synthesized network. Various algorithms are already available for on-chip communication synthesis; some of these are presented in Section 4.

The software implementation of COSI has been engineered to support the orthogonalization of concerns advocated by the Platform-Based Design methodology. 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. A users of COSI defines the quantities together with their partial order and attaches them to communication structures. For instance, the user can define the interfaces of IP cores and routers, pass them as parameters to ports, and attach ports to components. Other quantities that can be defined include: commodities, which represent the flow of packets from a source to a destination, latency figures, implementation parameters for nodes and wires, and geometry of components on the chip.

The bottom part of Figure 3 shows how components and platforms are captured in COSI. The core package defines a node and link component as basic objects with ports. Each component in the library must implement two sets of services: instantiation services, which allow to generate 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. This method corresponds to the dashed lines of Figure 1(c). 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 (e.g. power, area or latency). Different models can be attached to the same component that is not aware of the model implementation. Similarly, different nodes and links can belong to a platform that is not aware of the components' implementation.

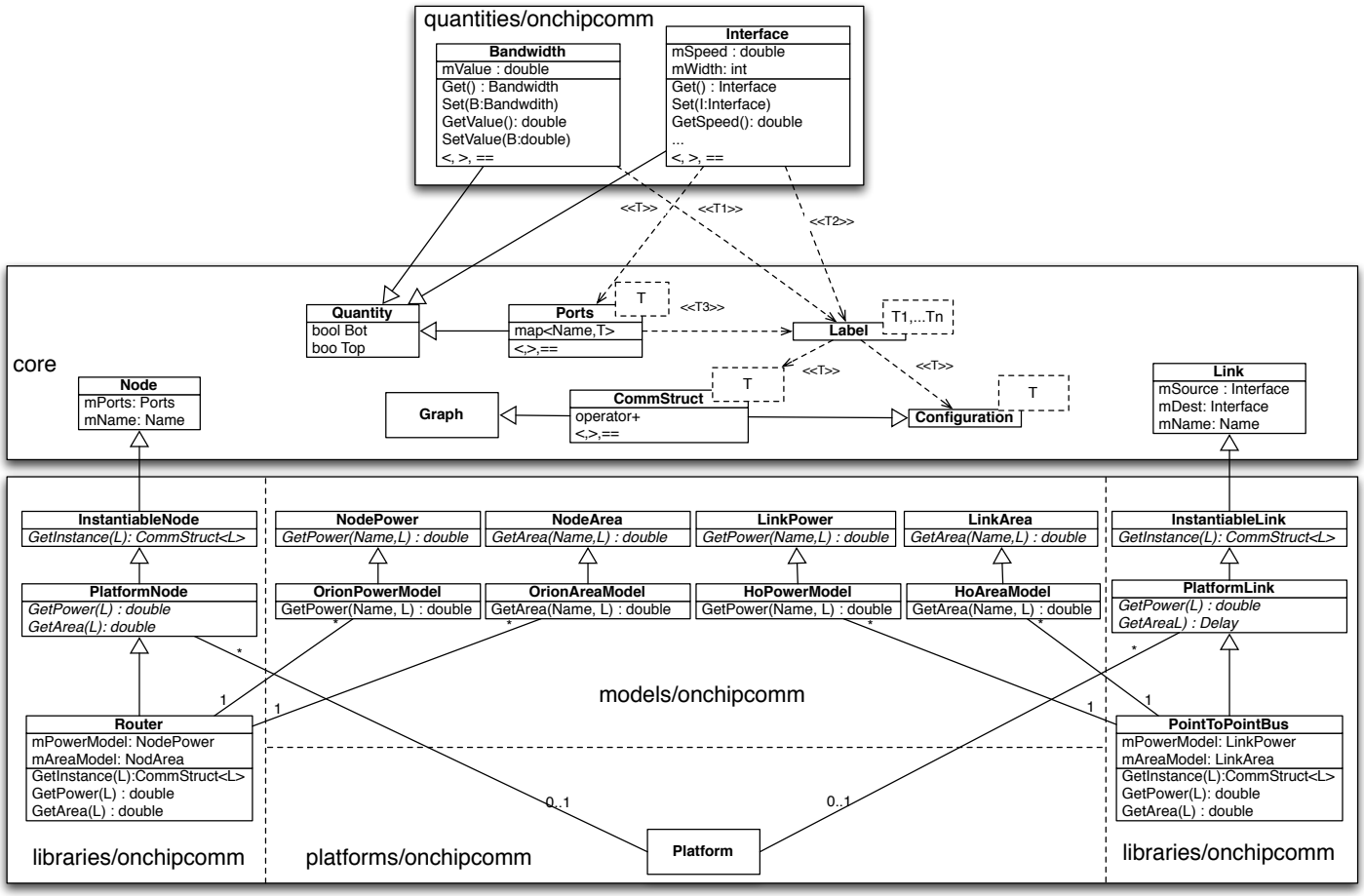

Figure 3: UML class diagrams of the data structures implemented in COSI for the NoC application domain.

3 An On-Chip Communication Synthesis Design Flow in COSI

Figure 4 shows how COSI-OCC, a design flow for on-chip communication synthesis, was built using COSI. The sequence of operations that are typically performed to synthesize an on-chip communication architecture are represented by numbered arrows. The input to COSI-OCC is a project file that contains pointers to the communication specification and to the library. All input/output files are in XML format. The communication specification contains a list of IP cores and inter-core communication constraints. The specification is parsed to yield an internal communication structure N_C where each node represents an IP core and the links represent constraints. The library/model file contains the description of each library element and the models associated with them. The platform is constructed by taking components from the library and attaching models to them. The platform also contains rules such as: restriction on the position of nodes, topological constraints, and requirements on the implementation like deadlock freedom. The project file includes also the optimization parameters such as the relative weight of power and area cost.

If there are unplaced IP cores, PARQUET [1] is used to floor-plan the chip (Step 2). The third step of a typical flow consists of 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. The outputs that are then used for analysis are generated in Step 4. COSI-OCC includes a set of code generators to produce an SVG graphical representation and a DOT 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 that is contained in `SysCLib`, also part of the COSI-OCC distribu-

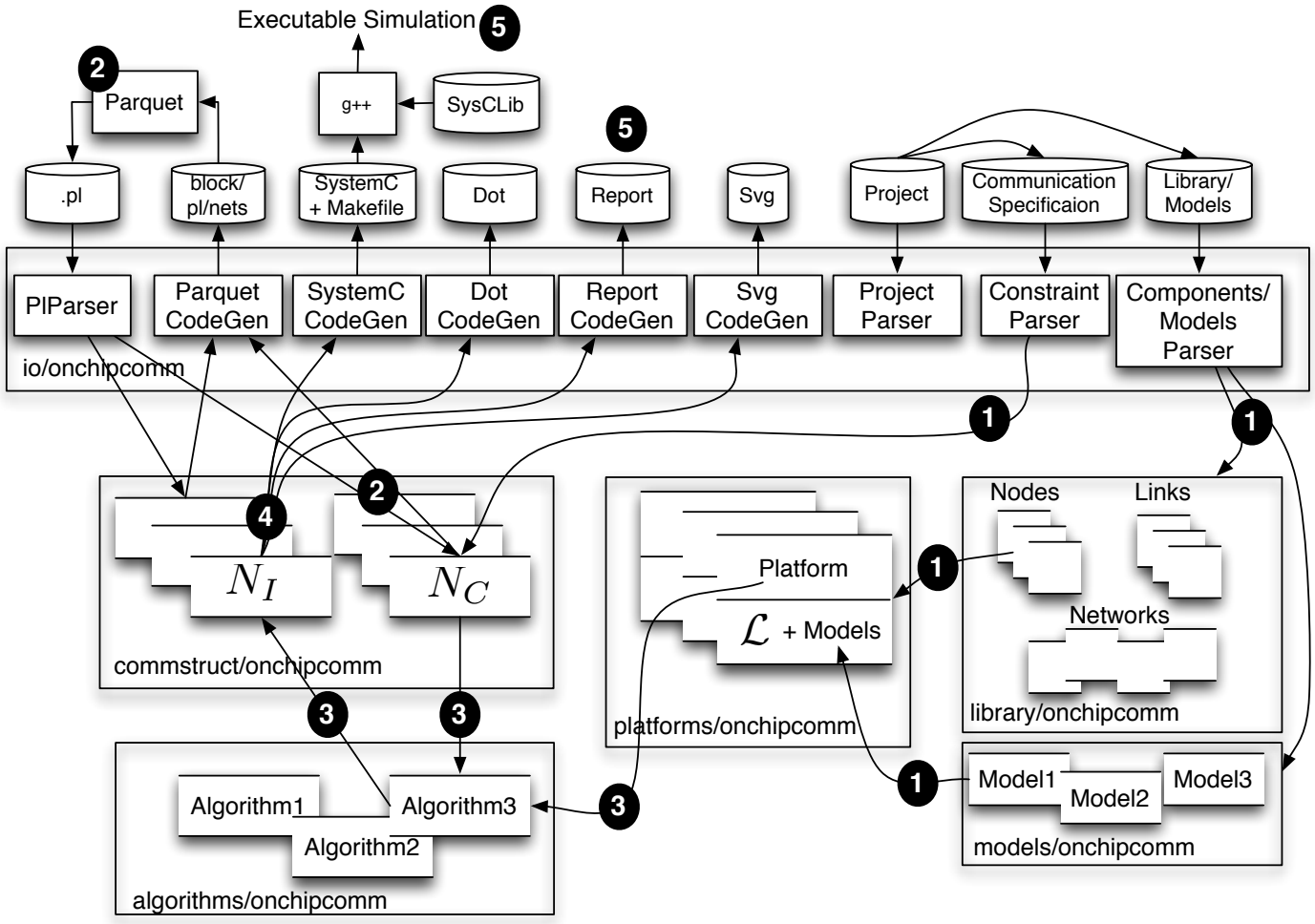

Figure 4: The COSI infrastructure used to set up a complete design flow for on chip communication.

tion. The generation of the SYSTEMC netlist is a further refinement of N_I that requires to set up protocol-related information like the weights for the *weighted fair queuing* algorithm, which is used by the routers to schedule flits. More information on COSI-OCC is available on the COSI project web site [4].

4 COSI Capabilities

In this section we illustrate the capabilities of the COSI infrastructure. Due to lack of space, we only report a set of results concerning the implementation and test of four network optimization algorithms, and refer the reader to previous papers that show how COSI has been used also for other purposes. In particular, in [9] we conducted a study on the power, area and performance trade-offs of a NoC as a function of router sizes at different technology nodes. We also interfaced COSI with an integer linear programming solver to evaluate the quality of our heuristic algorithm (called heuristic $H1$ later in this section). In [3] we evaluated the impact of the accuracy of wire models on system-level design choices. We integrated accurate delay models for 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, in [8] we showed how the infrastructure can be used to solve the communication synthesis problem for bus-based building automation networks.

Here we compare the performance achieved by four different algorithms for NoC optimization. The first algorithm considered here, which was described in detail in [9]), solves the *degree constrained multi-commodity flow problem*. The algorithm is composed of two main steps. After parsing the specification of the SoC and running the floor-planner, the first step consists in finding an initial solution without taking into consideration constraints on the node degrees. In the second step an iterative procedure removes degree violations by deleting links and/or adding routers. The initial solution is reached with the same technique that is used by algorithms for global routing. A path is found for each constraint one at a time (the actual implementation of the procedure depends on the composition rules). If the implementation network satisfies the degree constraints, the algorithm stops returning 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, all source-destination paths containing that link are attempted to be re-routed and replaced in the communication implementation with new paths. However, if one of these paths cannot be removed due, for instance, to bandwidth constraints, the algorithm back-tracks by reinserting the link and all the paths. If the re-routing procedure finds an implementation that satisfies the composition rules, the algorithm ends with success. If the procedure fails, a new attempt to reach a feasible solution is made after adding a new node (router). The idea is that when a new node is added, multiple links entering/exiting a node can be merged/split into/from one link, thereby reducing the degree of the node. However, if no node can be added (e.g., because delay constraints would be violated) the algorithm ends with an empty implementation, thus implying that no solution was found. We call this heuristic algorithm $H1$. An alternative heuristic $H2$, a variation of $H1$, is obtained by combining the two steps. The communication constraints that are part of the specification are considered one at the time as in $H1$. However, differently from $H1$, degree violations are now corrected as soon as they appear. After routing a constraints along a path, the algorithm checks if there are nodes with degree violations and attempts to remove them using the same procedure adopted in the second step of heuristic $H1$.

Both $H1$ and $H2$ solve the problem using a synthesis approach where the network is derived in a constructive way by adding components as needed after having completed the floor-planning of the SoC. A different approach is to map the SoC cores onto a pre-defined regular network topology such that the number of hops between communicating cores is minimized. To minimize the number of hops corresponds to minimizing power consumption and delay. We implemented an optimal mapping algorithm along the lines of the one presented in [2], that iteratively improves an initial mapping on a regular mesh topology. The initial mapping is computed by placing cores that communicate at high rate into mesh nodes that are topologically close to each other. 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, we remove the unused network resources (i.e. links, ports and routers) and run a floor-plan of the entire chip including the NoC. We call this algorithm *Mesh*.

The last algorithm proceeds by partitioning the cores in the specification in n parts according to their communication requirements such that the total communication bandwidth among cores belonging to different parts is minimized. Each partition is implemented by a star topology and the n stars are connected as part of a higher level network. This network is first assumed to be completely connected. Then, optimal routes are computed for each source-destination pair. Finally, we remove the unused network resources (i.e. links, ports and routers) and run a floor-plan of the entire chip including the NoC. We call this algorithm $Hstar(n)$ where n is the number of partitions.

The development, debugging, and performance assessment of new algorithms in COSI requires minimal effort since all the services offered by the COSI infrastructure share the same underlying model, i.e. communication structures. We report that the implementation of *Mesh* took approximately one day and the implementation of $Hstar(n)$ took only half a day (since many procedures are shared with the *Mesh* algorithm).

The results are shows in Figure 5 where the power, area and average number of hops are compared across the four algorithms. We used a $90nm$ technology, the Ho [6] model for wires, and the ORION [14] model for routers. The platform is configured to contain routers with at most five input and five output ports for $H1$, $H2$ and *Mesh*, and at most ten input and ten output ports for $Hstar(n)$. The flit-width is fixed to 32 bits and the clock frequency to $1GHz$. Notice 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 a large number of hops due to topological constraints. The two heuristics $H1$ and $H2$ return highly customized NoCs that match the number of hops of the one returned by *Hstar* while being more efficient in terms of power dissipation and area occupation.

5 Conclusions and Future Work

The design of communication infrastructures is a major aspect of the design of complex systems that range from SoCs to embedded distributed controllers. The cost-effective design of these infrastructures requires to solve optimization problems that are in general of high computational complexity. While given a problem domain, the actual implementation of the optimization algorithms is different, the structure of these optimization problem is remarkably similar so that a unified framework could be built to ease the construction of specialized design flows. In this paper, we presented COSI, one such software framework. To be easily customizable and efficient, the framework has to be based on solid principles that require the formalization of the essence of the problem at hand. In addition, it has to be built so that it is scalable and can accommodate a variety of building blocks and performance measure. To demonstrate its use in defining a design flow and in comparing a number of optimization algorithms, we presented an on-chip communication synthesis design flow, COSI-OCC, that uses COSI and includes a rich collection of libraries, models, algorithms and auxiliary tools. We expect that these resources together with the opportunity of augmenting them will promote collaboration among NoC researchers and designers with complementary skills. While today COSI-OCC targets NoCs, it can (and will) be extended to other on-chip interconnect structure such as busses, cross-bars and direct connections to allow SoC and microprocessor designers to choose the most effective infrastructure with no bias towards one or the other solution.

COSI can also be used to define design flows for other application domains by (a) changing the quantities that characterize communication constraints, cost, and performance, and (b) modeling the library elements to be used in that application. In particular, we used the COSI infrastructure to build a design flow for the synthesis of wired and wireless networks for building automation systems [8, 10].

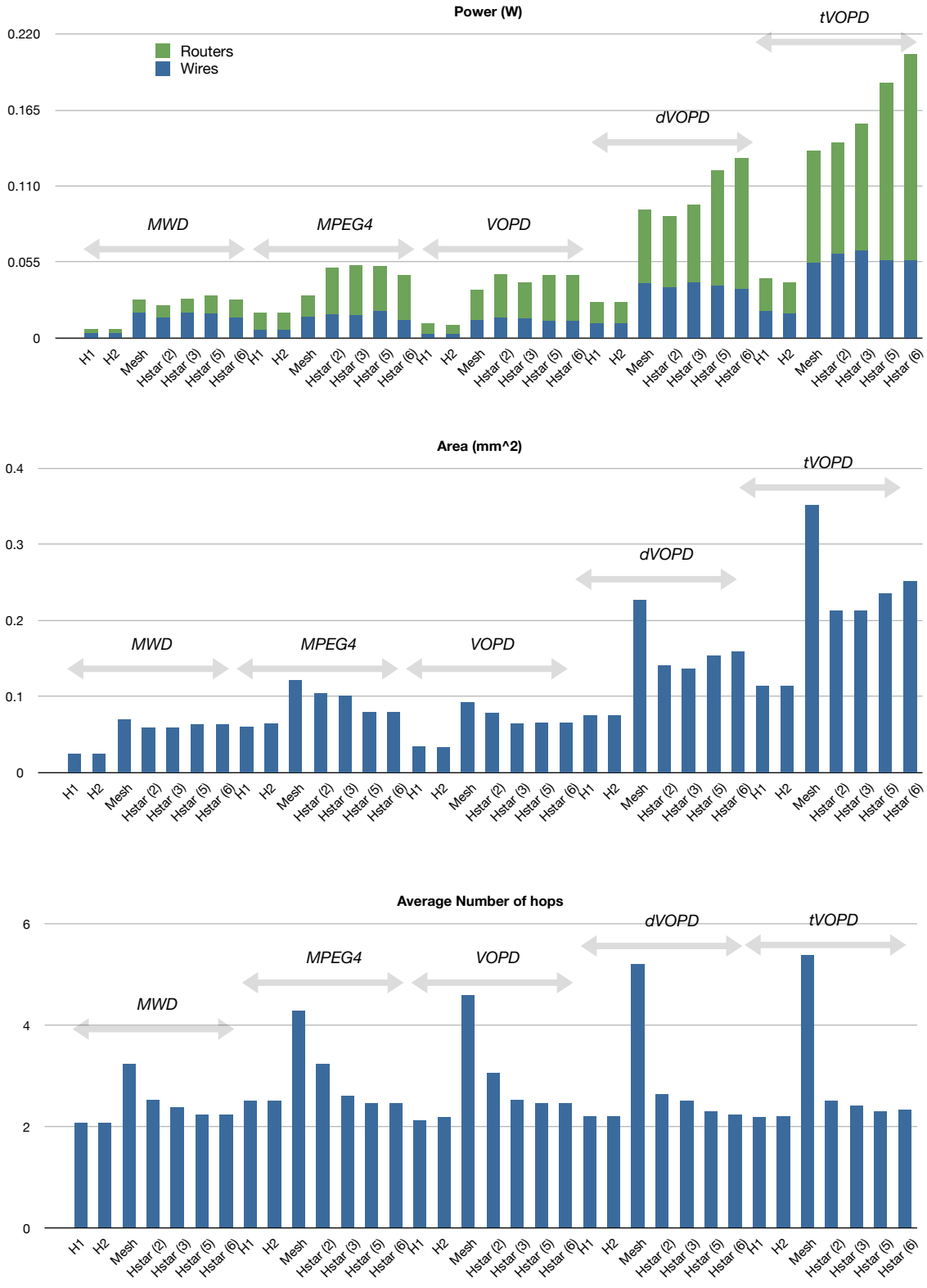

Figure 5: Comparison of the power, area and average number of hops for different algorithms on a set of benchmarks taken from the literature [11, 2]: a Multi Window Displayer (MWD), a Video Object Plane Decoder (VOPD), two VOPDs sharing a memory (dVOPD) and three VOPDs sharing two memories (tVOPD).

References

[1] S. N. Adya and I. L. Markov. Fixed-outline floorplanning : Enabling hierarchical design. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 11(6):1120–1135, December 2003.

[2] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. D. Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. on Parallel and Distributed Systems*, 16(2):113–129, Feb. 2005.

[3] L. Carloni, A. B. Kahng, S. Muddu, A. Pinto, K. Samadi, and P. Sharma. Interconnect modeling for improved system-level design optimization. In *Proceedings of the 2008 Asia South Pacific Design Automation Conference*, 2008.

[4] T. C. S. I. (COSI). http://embedded.eecs.berkeley.edu/cosi/.

[5] A. Ferrari and A. L. Sangiovanni-Vincentelli. System design: Traditional concepts and new paradigms. In *Proceedings of the International Conference on Computer Design*, pages 1–12, Oct. 1999.

[6] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, pages 490–504, April 2001.

[7] M. Minoux. Network synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, (19):313–360, 1989.

[8] 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. In *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 21–29, New York, NY, USA, 2007. ACM.

[9] A. Pinto, L. P. Carloni, and A. L. S. Vincentelli. A methodology and an open software infrastructure for the constraint-driven synthesis of on-chip communications. Technical Report UCB/EECS-2007-130, University of California, Berkeley, November 2007.

[10] A. Pinto, M. D'Angelo, C. Fischione, E. Scholte, and A. Sangiovanni-Vincentelli. Synthesis of embedded networks for building automation and control. To appear in the Proceedings of the 2008 Americal Control Conference, 2008.

[11] A. Pullini, F. Angiolini, P. Meloni, D. Atienza, S. Murali, L. Raffo, G. D. Micheli, and L. Benini. 65 nm NoC design: Opportunities and challenges. *Proc. of the 1st Intl. Symp. on Networks-on-Chips*, 2007.

[12] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. H. III. Approximation algorithms for degree-constrained minimum-cost network-design problems. *Algorithmica*, 31(1):58–78, 2001.

[13] A. L. Sangiovanni-Vincentelli. Quo vadis sld: Reasoning about trends and challenges of system-level design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[14] H. S. Wang, X. Zhu, L. S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In *Proc. of the 35th Intl. Symp. on Microarchitecture*, pages 294–305, Nov. 2002.