A Methodology for Constraint-Driven Synthesis of On-Chip Communications

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

***Abstract*—We present a methodology and an optimization framework for the synthesis of on-chip communication through the assembly of components such as interfaces, routers, buses and links, from a target library. Models for functionality, cost, and performance of each element are captured in the library together with their composition rules. We develop a mathematical framework to model communication at different levels of abstraction from the point-to-point input specification to the library elements and the final implementation.**

***Index Terms*—Communication synthesis, System-on-chip, Interconnect synthesis, Performance optimization.**

I. Introduction

With the advances of IC technology, global interconnects have become the dominant factor in determining chip performance: they are not only becoming responsible for a larger fraction of the overall delay and power dissipation but exacerbate also design problems such as noise coupling, routing congestion, and timing closure, thereby imposing severe limitations on design productivity [1], [2]. Because of these characteristics, most VLSI circuits can be considered *distributed systems*, a fact that challenges traditional design methodologies and the electronic design automation tools that are based on them [3]. Systems-on-Chip (SoCs) are typically designed by assembling intellectual property (IP) components from different vendors and/or different divisions of the same company in the attempt of reducing time-to-market by reusing pre-designed and pre-verified elements. However, since these components are designed independently, the assembly step is often a challenging problem that requires the design of communication interfaces to match different protocols and data parallelism, and the routing of global interconnect wires to meet the constraints imposed by the target clock period.

The Open Core Protocol (OCP) [4] tackles this problem by defining a standard open-domain interface with which IP cores should comply to allow fast integration using appropriate interconnect architectures. While there is no intrinsic limitation on the interconnect architecture for OCP, most designers rely on traditional bus architectures so that pre-designed components can be used. In this domain, proprietary protocols such as the ARM AMBA BUS and the IBM CORECONNECT are popular among SoC designers making the adoption of a universal standard difficult at best.

We argued that SoCs are distributed systems. For this reason, bus architectures may not be always ideal; in fact, a set of seminal papers has proposed scalable, multi-hop, packet-switched Networks-on-Chip (NoCs) as a solution for the integration of IP components as an interesting alternative [5]–[7]. Borrowing from the communication networks literature, an NoC can be built through the combination of heterogeneous elements such as interfaces, routers, and links. The NoC design is a challenging problem because there are many degrees of freedom (e.g. network topologies, routing protocols, flow-control mechanisms, positions of the communication components and core interfaces) as well as multiple optimization goals (e.g. performance, power, area occupation and reliability). Hence, the problem had been simplified by limiting the number and types of components considered, by focusing on a subset of the relevant objectives, by constraining NoC topology and components positions, and by dividing the optimization process in successive stages. Limiting the degrees of freedom has also the important side effect of reducing implementation and layout complexity.

In [8] Bertozzi *et al.* propose NETCHIP, a synthesis flow to derive an *application-specific* NoC by mapping the application cores on standard topologies (e.g torus, mesh, hypercube) in an optimal way. In [9], Hu and Marculescu perform mapping and routing on the NoC with optimal energy and performance. Lahiri *et al.* use standard topologies consisting of sets of channels (point-to-point links or shared busses) connected by bridges [10]. Ogras *et al.* propose a perturbation method that starting from the mapping of an application on a standard topology optimizes performance and cost by inserting custom long links between routers [11]. In [12] Murali *et al.* synthesize NoCs that, albeit being more general than the approaches that start from a regular topology, are still constrained to be "two-level structures", where star topologies are connected by links to satisfy inter-cluster communication requirements. In [13] Srinivasan *et al.* synthesize an application-specific NoC without assuming any pre-existing interconnection fabric. The synthesis problem is linearized and solved via integer linear programming (ILP) that, due to its complexity, yields

This work was partially supported by the GSRC Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and by the National Science Foundation (Award #: 0644202).

A. Pinto is with United Technologies Research Center, East Hartford, CT, most of this work was carried out while at the Dept. of EECS, U.C. Berkeley, CA 94720, (apinto@eecs.berkeley.edu).

L.P. Carloni is with Department of Computer Science, Columbia University New York, NY 10027 (luca@cs.columbia.edu).

A. Sangiovanni-Vincentelli is with the Dept. of EECS, U.C. Berkeley, CA 94720, (alberto@eecs.berkeley.edu). Manuscript received November 15, 2007; revised April 28, 2008.

running time of the order of several hours even for relatively small instances. In [14] the same authors propose an efficient approximation algorithm that is strongly tied to the cost model and that does not consider constraints on the router size (i.e. number of inputs and outputs).

While a rich set of interesting results exists in the literature, few are the examples of practical applications of NoCs. In fact, the debate between those who favor standard bus architectures or variations thereof and those who advocate the adoption of NoC approaches ranging from constrained architectures to custom ones is vibrant. We do not take sides even though the NoC approach has undisputable fundamental merits that may make it successful in the long run. Instead, *we propose a general methodology for the design of on-chip communication that can explore a large number of alternatives including as special cases NoCs, bus architectures and hybrid ones*. Thanks to its generality our approach can be used to build a framework where different constrained solutions are compared using a number of evaluation factors.

We address the synthesis of optimal heterogeneous networks by assembling components from a fine-grained library without enforcing any constraint on their topology other than the ones formally captured in the library. In particular, the network that we obtain need not be direct and not even connected if these constraints are not captured in the composition rules of the communication components.

Our approach is detailed in the rest of the paper as follows: In Section II, we introduce formally the *SoC design specification* (i.e. the function), the target technology process with the library of communication components and the final communication implementation. At a first glance, the formalism used in this section may seem overly complex. However, in our opinion, the benefits it offers in terms of generality (the same formalism applies independent of the communication synthesis problem being investigated) outweigh its complexity. In Section III, we show how to use this formal framework to formulate a general optimization problem for a general class of libraries. In Section IV, we use our framework to formulate the communication synthesis problem in the specific case of NoCs. and provide a heuristic algorithm to solve the resulting complex integer optimization problem. The algorithm is independent from the specific input constraints and the target platform. We do report a customization of the algorithm that takes into account bandwidth and latency constraints, expressed as hop count, to synthesize a minimal-power NoC. The general algorithmic framework can be customized in several other ways by changing the cost function and constraints.

The material presented in this paper is the theoretical foundation of COSI-OCC , a design flow for on-chip communication synthesis design that is part of the COmmunication Synthesis Infrastructure (COSI). COSI is a public-domain design framework for the analysis and synthesis of interconnection networks [15]. Our goal has been to provide an infrastructure that can be used by researchers and designers as a basis for developing new design flows by integrating additional models, library elements, analysis tools and synthesis tools [1]. In Section V, we briefly describe COSI-OCC together with the results we obtain by applying it to a number of test cases for NoC design. We present more details on COSI and COSI-OCC in [17] and we provide a detailed comparison of our approach with other on-chip communication design tools in [18].

II. The Methodology and its Mathematical Representation

A. The Methodology

The general approach is based on Platform-Based Design (PBD) [19] where the design specification and the implementation alternatives are kept separate. The methodology is recursive: the functional specification is implemented on a particular architecture through a series of refinement steps. At each step, which corresponds to a specific level of abstraction, the implementation alternatives are characterized by a set of components, called *library*, that can be instantiated, configured, and assembled according to specific rules, to derive a more complex structure. The set of components together with their compositional rules define a *platform* which is a family of admissible solutions. The task of the synthesis process is then to select one out of this family (a *platform instance*) and a mapping of the specification onto the components that satisfy the requirements and possibly optimize the objectives of the design. The implementation refines both requirements and platform instance and is defined at a lower level of abstraction.

In this process, it is essential to formalize how requirements are specified, how the library is described, and how the composition rules are defined and applied to generate the space of admissible solutions. The composition rules can be used to encode constraints related to the topology that the designer wishes to consider while the components in the library determine which kind of "nodes" can be selected. To select a platform instance using an optimization algorithm we must associate to each library component (and to the hierarchical composition of two or more of them) a "characterization" in terms of cost, performance, power, and "type" (e.g., number of ports and interface type of a router) that allows us to evaluate metrics associated with the objectives and constraints of the design.

To illustrate our approach, consider, for instance, the simplified *Set-Top Box System* shown in Fig. 1. This design will serve as an example throughout the paper. The SoC *specification* contains six IP cores that exchange messages through a dozen of point-to-point channels and interact with the external environment through four major I/O connections (pads). The data input stream is processed by the demux core (`dem`) that sends an audio stream to the audio decoder and a video stream to the video decoder. The video decoder accesses the external memory through a memory controller. The memory is used both as an intermediate storage and to send the decoded stream to the display controller and HDTV encoder. Finally, a master

[1]This approach is similar to the one our group followed in developing MIS that has been used for years as a platform to invent and test new logic synthesis algorithms [16].

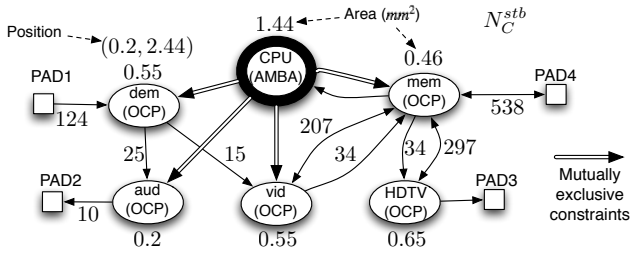

Fig. 1. The system-level specification of a simplified Set-Top Box. Each core in the specification is annotated with and area in mm^2 and each arrow is annotated with a bandwidth constraint in MB/s.

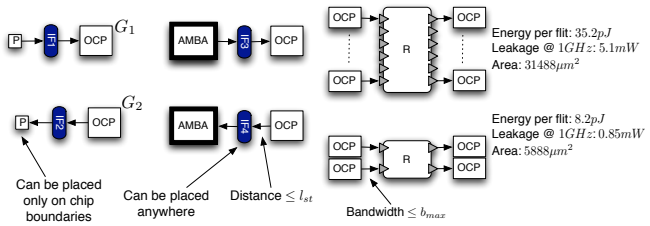

Fig. 2. A library of predefined on-chip communication components.

CPU controls the operation of all the blocks and handles the interaction with the environment. Additional non-functional constraints are often part of the specification: e.g, the `dem` core must occupy position $(0.2, 2.44)$ (in millimeter); the `cpu` communicates with the other cores, one at the time.

Fig. 2 shows a *library of on-chip communication components* that contains a set of communication templates including interfaces `IF1` and `IF2` to connect pads with OCP cores, and interface `IF3` and `IF4` to connect AMBA cores with OCP cores. The library also contains various OCP routers that differ by the number of I/O ports. Each component is characterized by performance metrics, cost functions, and composition rules. Possible characterizations include: a link in a given metal layer can sustain up to a certain bandwidth b_{max} and span a distance no greater than l_{st}; a parameterized synthesizable router may not have more than a maximum number of I/O ports, and an IP core may feature only a specific protocol interface.

A communication structure that serves as the communication backbone for an SoC is constructed by *instantiating* communication templates (i.e. components from the library) and *composing* them. For example, `PAD4` is Fig. 3 is connected to the memory controller by instantiating templates G_1 and G_2. Fig. 3 shows two alternative NoC implementations of the same specification. Network G_P^1 is obtained by instantiating the necessary interfaces plus one 8×8 router while G_P^2 is obtained by instantiating only 2×2 routers. The performance and cost of the communication structure depend on the performance metrics and the cost functions of each component.

B. Basic Definitions

The basic element of our formal framework is the *communication structure*. A communication structure is a set of interconnected components with associated *quantities* such as latency, bandwidth and position. A quantity q takes on values

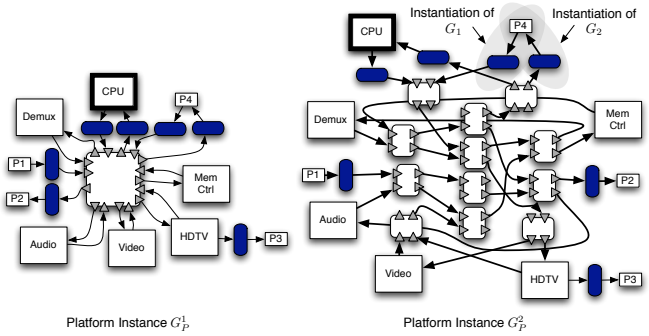

Fig. 3. Two NoC instances obtained by instantiation and composition of communication components.

from a domain D_q that is partially ordered by a relation $\preceq_q$. The ordering relation captures the notion of a value being "better" than another value. We assume that $\perp$, which denotes no values, always belongs to the domain of a quantity D_q. Also, $\perp \preceq_q \nu$ for all $\nu \in D_q$. A quantity q is finite if D_q is a finite set, and it is bounded if there exists an element $\bar{\nu} \in D_q$ such that $\nu \preceq_q \bar{\nu}$ for all $\nu \in D_q$. Bandwidth, for instance, is modeled by a quantity b. Its domain D_b can either be the set of natural numbers, or it can be a discrete set of values like $D_b = \{10, 100\}$ (in MB/s). Ordering relation $\preceq_b$ is the same as the ordering relation $\leq$ defined on natural numbers. The domain D_h of the quantity h representing latency can be defined as a finite set of integer numbers, but the ordering relation $\preceq_h$ is now reversed, i.e. $100(ns) \preceq_h 10(ns)$.

Given a vector of quantities $\mathbf{q} = (q_1, \ldots, q_k)$, the domain of $\mathbf{q}$ is the cross product $D_{q_1} \times \ldots \times D_{q_k}$. It is partially ordered by a relation $\preceq_{\mathbf{q}}$ point-wise induced by the relations $\preceq_{q_i}$. We use the notation $\perp^n$ to denote a n-tuple of $\perp$ values. $[X \to Y]$ denotes the set of all functions from set X to set Y.

Definition 1. *A* communication structure *is a tuple* $N(\mathcal{C}, \mathbf{q}, L)$ *where* $\mathcal{C} = \{c_1, \ldots, c_n\}$ *is a set of components,* $\mathbf{q} = (q_1, \ldots, q_k)$ *is a vector of quantities, and* $L \subseteq [\mathcal{C} \to D_{\mathbf{q}}]$ *is a set of communication configurations. Set* $\mathcal{C}$ *is partitioned into the set of nodes* $V \subseteq U_V$ *and the set of links* $E \subseteq V \times V$.

The set L of communication configurations captures the different ways in which quantities can be associated to components. The set U_V is called the *node universe*. Similarly, the *component universe* is $U_{\mathcal{C}} = U_V \cup U_V^2$, and the *configuration universe* is $U_{\mathbf{q}} = \cup_{\mathcal{C} \subseteq U_{\mathcal{C}}} [\mathcal{C} \to D_{\mathbf{q}}]$, the union of all possible configurations for any subset of components. Let $\mathcal{G}_{\mathbf{q}}$ be the set of all communication structures with quantities $\mathbf{q}$.

For a given subscript σ, and vector of quantities $\mathbf{q}$, let $N_\sigma \in \mathcal{G}_{\mathbf{q}}$ be a communication structure. Then, we use $\mathcal{C}_\sigma$,V_σ, E_σ and L_σ to denote the sets of components, nodes, links, and configurations of N_σ, respectively.

Example 1. (Communication structure): Consider the vector of quantities $\mathbf{q} = (x, y)$ representing the horizontal and vertical coordinates of a component. The domain $D_{\mathbf{q}}$ is the set of points where nodes can be placed. This domain can be described, for instance, by a discrete set of points or by union of rectangles. If there are no preferred positions, the elements of $D_{\mathbf{q}}$ are not comparable, therefore the order $\preceq_{\mathbf{q}}$ is a flat one, with $\perp$ being the minimum element. Given

a communication structure $N(\mathcal{C}, \mathbf{q}, L)$, the set of configurations L captures all the admissible placements of the nodes in V. Since we do not assign any position to the links, for all $l \in L$ and for all links $e \in E$, $l(e) = \perp$[2]. The additional constraint that no two nodes occupy the same position requires that for all $l \in L$, and for all pair of nodes $u, v \in V$, $l(u) \neq l(v)$.

We introduce two scoping operators on configurations. Given a communication structure $N(\mathcal{C}, \mathbf{q}, L)$, the restriction of a configuration $l \in L$ to a subset of components $\mathcal{C}' \subseteq \mathcal{C}$, denoted by $l|_{\mathcal{C}'}$, is a function $f : \mathcal{C}' \rightarrow D_{\mathbf{q}}$ such that $f(c) = l(c)$ for all $c \in \mathcal{C}'$. In particular, $l|_V$ and $l|_E$ are the restrictions of a configuration l to the set of nodes and links, respectively. Given a vector $\mathbf{q}'$ obtained from $\mathbf{q}$ by projecting away some of the quantities, the projection of a configuration onto $\mathbf{q}'$ is denoted by $l[\mathbf{q}']$, and corresponds to ignoring the quantities not in $\mathbf{q}'$. We naturally extend these operators to sets of configurations, e.g. $L[(x)]|_V$ denotes the possible assignments of horizontal positions to nodes in Example 1.

We use communication structures to capture three important and related concepts in our framework: the specification of an on-chip communication synthesis problem, the collection of alternatives to implement the communication (the platform instances), and the final communication implementation. These three structures correspond to different abstraction levels. In Section II-F we establish precise relations among them to define when an implementation refines a platform instance and supports a specification. It is often necessary to compare specifications, platform instances and implementations; e.g. it is important to be able to order different specifications depending on how stringent the constraints are. Similarly, it is important to compare platform instances depending on their performance. Therefore, we define an ordering relation $\leq_{\mathbf{q}}$ on the set of communication structures $\mathcal{G}_{\mathbf{q}}$ as follows:

Definition 2. *Given two communication structures $N_1, N_2 \in \mathcal{G}_{\mathbf{q}}$, $N_1 \leq_{\mathbf{q}} N_2$ if and only if $\mathcal{C}_1 \subseteq \mathcal{C}_2$, and for all $l_1 \in L_1$ there exists $l_2 \in L_2$ such that for all $c \in \mathcal{C}_1$, $l_1(c) \preceq_{\mathbf{q}} l_2(c)$.*

C. Communication Specification

We express the specification of an on-chip communication synthesis problem as a communication structure $N_C \in G_{\mathbf{q}_C}$, where $\mathbf{q}_C = (x, y, a, \tau, b, h)$. Nodes represent IP cores (that can be sources and/or destination of a communication) and have an associated position (x, y) in the Euclidean plane, an area a, and a type τ denoting the supported interface protocol. Links represent distinct inter-core communications. Each link is associated with two quantities: a minimum average bandwidth b and a maximum latency h. Each configuration $l \in L_C$ represents a possible combination of the positions and interfaces of the cores, and bandwidth and latency requirements for the communication among them (e.g., to capture different communication scenarios or different chip floor-planning).

Example 2. (Communication specification): In the set-top box example of Fig. 1, the position of the *dem* core is fixed at coordinates $(0.2, 1.44)$. Hence, each configuration $l \in L_C^{stb}$ must be such that $l(dem) = (0.2, 1.44, 0.55, OCP, \perp, \perp)$. Since there are no other floor-planning constraints, the position of the other IP cores can be determined during the synthesis process. The double arrows indicate that the constraints between the CPU and the IP cores are mutually exclusive, i.e. the CPU can only communicate with one core at the time: i.e. for all $l \in L_C^{stb}[(b)]$, only one among $l((CPU, dem))$, $l((CPU, aud))$, $l((CPU, vid))$, $l((CPU, mem))$ can be different from zero.

Since the performance and cost of the network depend on the core positions, an important step in our design flow is to restrict the possible configurations of a specification by fixing the position of the ports of each core. In COSI-OCC we rely on the PARQUET floor-planner [20] to obtain these positions.

D. Communication Structures Instantiation and Composition

To allow the incremental design of complex on-chip communications, we introduce two operations: *renaming* and *parallel composition*. The identifiers of two nodes in different sub-nets can be renamed to be the same to indicate that either one IP implements both or an implicit connection is present between the two sub-nets at these nodes. A *renaming function* $r : U_V \rightarrow U_V$ is a bijection on the vertex universe. R denotes the set of all renaming functions. Given a communication structure N and a renaming function r, with abuse of notation we use $r(N)$ to denote a new communication structure where the components have been renamed according to r.

The *composition* of two communication structures N_1 and N_2, denoted by $N_1 \| N_2$, results in a new communication structure N that contains the set of components $\mathcal{C}_1 \cup \mathcal{C}_2$. We define the operator $\|$ by two rules. The first rule establishes how the configurations of the components being merged contribute to the formation of the ones of the combined entity. The rule is expressed by the binary operator $\oplus_{\mathbf{q}}$ that is commutative and associative so that the composition of communication structures also satisfies these properties. This is important since we want the result of the composition to be independent of the order in which communication structures areinstantiated and composed. Further, if $l_1 : \mathcal{C}_1 \rightarrow D_{\mathbf{q}}$ and $l_2 : \mathcal{C}_2 \rightarrow D_{\mathbf{q}}$, then $l = l_1 \oplus_{\mathbf{q}} l_2$ must be such that $l : \mathcal{C}_1 \cup \mathcal{C}_2 \rightarrow D_{\mathbf{q}}$. This operator is defined on sets of configurations as follows: let $L_1 \subseteq [\mathcal{C}_1 \rightarrow D_{\mathbf{q}}]$ and $L_2 \subseteq [\mathcal{C}_2 \rightarrow D_{\mathbf{q}}]$, then $L_1 \oplus_{\mathbf{q}} L_2 = \{l_1 \oplus_{\mathbf{q}} l_2 | l_1 \in L_1 \wedge l_2 \in L_2\}$. A second rule restricts the legal compositions by forcing the composed structure to satisfy certain properties. This rule, that defines a class of communication structures the result of the composition must belong to, is given by a relation between the components and the configurations and it is denoted by $\mathcal{R} \subseteq 2^{U_C} \times U_{\mathbf{q}}$.

Definition 3. *Given a binary operator $\oplus_{\mathbf{q}}$ and a composition rule $\mathcal{R}$, and two communication structures N_1 and N_2 belonging to $\mathcal{G}_{\mathbf{q}}$, their composition is $N_1 \|^{\mathcal{R}}_{\mathbf{q}} N_2 = N \in \mathcal{G}_{\mathbf{q}}$, where $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$, $L = \{l \in L_1 \oplus_{\mathbf{q}} L_2 | (\mathcal{C}, l) \in \mathcal{R}\} \neq \emptyset$; the composition is not defined if $L = \emptyset$.*

Example 3. (Composition of communication specifications): We want to add an extra video channel to our set-top box chip by *reusing* the already instantiated IP cores. In Fig. 4, N_{vch} is a communication structure capturing the communication requirements of a set-top-box video channel. To reuse the same IP cores, we rename the nodes according to a renaming function r such that $r(d) = dem$, $r(m) = mem$, $r(v) = vid$ and $r(dec) = HDTV$. Since the new video channel must be displayed on the same device, $r(P2) = PAD3$ forces the same output pad to be reused. For the demodulator input, though, we need an additional pad. We also add a new pad to

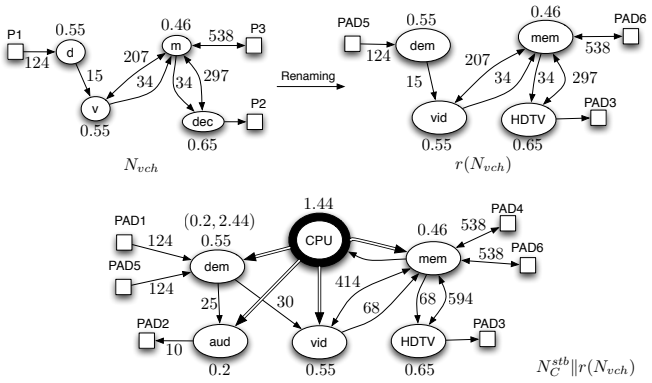

Fig. 4. Example of parallel composition of networks: the set-top box is expanded by adding a video channel and an extra off-chip memory bank.

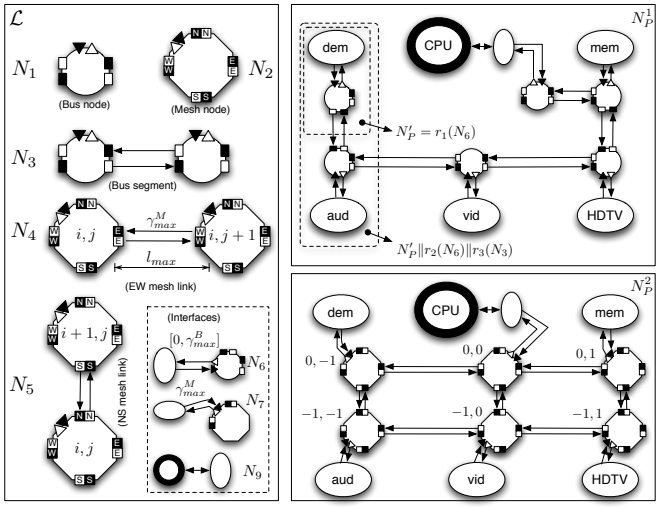

Fig. 5. Example of a library $\mathcal{L}$ and two alternative implementations for the set-top box based on composing elements instantiated from $\mathcal{L}$.

connect a second memory bank to the memory controller. Fig. 4 shows the result of the composition $N_C^{stb} \|_{\mathbf{q}_C}^{\mathcal{R}_C} r(N_{vch})$. Intuitively, we have added the bandwidths of common requirements and we have restricted the position of the dem core. More precisely, we need to define the operator $\oplus_{\mathbf{q}_C}$. Given two communication structures structures $N_1,\ N_2 \in \mathcal{G}_{\mathbf{q}_C}$, let $l_1 \in L_1$ and $l_2 \in L_2$ be two configurations. The configuration $l = l_1 \oplus_{\mathbf{q}_C} l_2$ is defined as follows:

- there is no "interference" between components not shared by N_1 and N_2, i.e $l(c) = l_1(c)$ for all $c \in \mathcal{C}_1 \setminus \mathcal{C}_2$, and $l(c) = l_2(c)$ for all $c \in \mathcal{C}_2 \setminus \mathcal{C}_1$;
- common nodes must be "compatible", meaning that they must agree on the positions and interfaces:

$$\forall c \in V_1 \cap V_2,\ l(c) = \begin{cases} l_1(c) & if\ l_1(c) = l_2(c) \\ \perp^6 & if\ l_1(c) \neq l_2(c) \end{cases}$$

  (notice that it is sufficient to have some compatible configurations for the composition to be defined);
- for all $c \in E_1 \cap E_2$, $l[(b)](c) = l_1[(b)](c) + l_2[(b)](c)$ and $l[(h)](c) = \min\{l_1[(h)](c), l_2[(h)](c)\}$.

We now define the composition rules. First, we specify that each node has an assigned position and interface protocol: $\mathcal{R}_C^v = \{(\mathcal{C}, l) \in 2^{U_C} \times U_{\mathbf{q}_C} | \forall v \in \mathcal{C},\ \forall q \in \{x, y, a, \tau\},\ l[(q)](v) \neq \perp\}$. A second rule may depend on the area budget ν_a for the IP cores on the chip:

$$\mathcal{R}_C^a = \left\{ (\mathcal{C}, l) \in 2^{U_C} \times U_{\mathbf{q}_C} \,\middle|\, \sum_{c \in V} l[(a)](c) \leq \nu_a \right\}$$

The two rules are combined as $\mathcal{R}_C = \mathcal{R}_C^v \cap \mathcal{R}_C^a$. We give examples of other rules in Section II-E.

E. Libraries and Platforms

A platform is the set of all *valid* compositions that can be obtained by assembling the components from a given communication library. These components either have a corresponding implementation that is ready to be used or can be synthesized by tools operating at a lower level of abstraction.

A *communication library* $\mathcal{L}$ is a collection of communication structures, i.e. $\mathcal{L} \subset \mathcal{G}_{\mathbf{q}}$. The elements of a communication library are templates that can be instantiated and composed to obtain more complex communication structures. The vector of quantities that characterize our platform is $\mathbf{q}_P = (x, y, \tau, in, out, \gamma)$ where each node has an associate position (x, y), a type τ, two multisets in and out of input and output port interfaces, respectively. Each link is associated with a *capacity* γ, i.e. the maximum bandwidth that it can sustain (Section II-E). Differently from $\mathbf{q}_C$, vector $\mathbf{q}_P$ represents the *capabilities* of a component; e.g., quantities x and y in $\mathbf{q}_C$ denote the coordinates where a component *must* be located, whereas the same variables in $\mathbf{q}_P$ denote the coordinates where a component *can* be located.

The definition of composition $\|_{\mathbf{q}_P}^{\mathcal{R}_P}$ captures the set of valid communication architectures (i.e. communication platform instances) that can be obtained out of the communication library. The definition of the rules is more involved than in the case of Example 3 and depends on the design space of interest. The following example shows the flexibility that our framework provides in defining the set of communication structures that can be obtained by composition of library elements.

Example 4. Composition rules: Consider a communication library whose elements are nodes and links. Fig. 5 shows a communication library $\mathcal{L}$ and two possible platform instances N_P^1 and N_P^2. Library $\mathcal{L}$ contains the following set of components: a bus node and a bidirectional bus-segment connecting two bus nodes; a mesh node and two mesh links for East-West connection and North-South connection, respectively. It contains also a set of interface communication structures to connect IP cores to bus nodes and mesh nodes. Each node has an associated multi-set of input interfaces in and output interfaces out (depicted as filled and non-filled shapes attached to nodes in Fig. 5). A link connects an output interface of a node to an input interface of another node. Mesh links have an associated maximum capacity γ_{max}^M while bus-segments (including the link between an IP core and a bus node) have an associated interval of capacities $[0, \gamma_{max}^B]$ corresponding to different configurations. We introduce two more quantities i_x and i_y for mesh structures that are the row and column index of a node. Now, we state a set of composition rules such that the only platform instances that are valid in this platform are either busses or meshes:

1) The number of bus nodes can be at most the number of bus segments minus one. This ensures that the topology of a bus is a collection of trees. Also, since a bus node has only two bidirectional ports to connect to other bus nodes,each bus is a chain of IP cores (as shown by the platform instance N_P^1).
2) An East-West mesh link can connect two mesh nodes (u, v)

only if $l[(i_x, i_y)](u) = (i, j)$ and $l[(i_x, i_y)](v) = (i, j+1)$; a North-South mesh link can connect two mesh nodes (u, v) only if $l[(i_x, i_y)](u) = (i, j)$ and $l[(i_x, i_y)](v) = (i+1, j)$ (as shown by the platform instance N_P^2).

3) A bus configuration l forces the sum of the capacities of the links connecting the cores to the bus to be less than γ_{max}^B. This restricts the possible bus organizations and models the sharing of the bus capacity among all connected IP cores.

These three rules define $\mathcal{R}_P$ for this specific platform.

The platform is obtained by taking all possible compositions of instantiated library elements (if the composition is valid):

Definition 4. *The* communication platform *generated by the communication library $\mathcal{L}$ under composition $\|_{\mathbf{q}}^{\mathcal{R}}$ is*

$$\langle \mathcal{L} \rangle = \{ N(\mathcal{C}, \mathbf{q}, L) = r_1(N_1) \|_{\mathbf{q}}^{\mathcal{R}} \dots \|_{\mathbf{q}}^{\mathcal{R}} r_m(N_m) \mid r_i \in R',\ N_i \in \mathcal{L}, L \neq \emptyset, m \geq 1 \}$$

where $R' \subseteq R$ is a set of valid renaming functions. An element $N \in \langle \mathcal{L} \rangle$ is called a communication platform instance.

The set of renaming functions is restricted because generally there may be constraints on the possible ways in which components are instantiated (see Section IV for an example of definition of R').

F. Mapping

The same specification can be implemented by many platform instances. On the other hand, the same platform instance can implement a variety of different specifications. For a given platform instance, deriving an implementation of a given specification is called *mapping* in the platform-based design terminology. The implementation is a refinement of both the specification and the platform instance. Being a refinement means that the implementation contains more details, that are captured not only by the number of components of the communication structure defining the implementation, but also by the vector of quantities. In our example, the implementation of a communication specification is a communication structure derived from a platform instance by adding the information regarding the routing of packets and the latency. Routing is captured by a quantity ρ called *transfer table*. To define ρ, we introduce another quantity λ with domain D_λ representing a name attached to each component. To simplify the notation, we assume that this quantity implicitly belongs to any vector of quantities. For a component c, we denote its name with $\lambda(c)$. The name of a component c is different from its *identifier* which is denoted by the symbol c itself. In particular, the renaming function does not change the name of a component but only its identifier (this is the main reason to distinguish them). The domain of ρ is $D_\rho = 2^{D_\lambda \times D_\lambda \times D_{out}}$. Therefore, a transfer table is a set of triples $(\lambda_s, \lambda_d, o)$ where λ_s and λ_d are the names associated with the source and destination of the packets, respectively, and o is an output interface of a node. Each triplet specifies the output interface o for each packet that arrives at the node from a given source in its transit to a given destination. For routers, the transfer table is also called routing table. Latency is captured by quantity h as introduced in Section II-B. Hence, an implementation is a communication structure $N_I(\mathcal{C}_I, \mathbf{q}_I, L_I)$ where $\mathbf{q}_I = (x, y, \tau, in, out, \rho, b, \gamma, h)$ (which contains the quantities coming from the specification and from the platform instance).

The latency information associated to the components of an implementation depends on the actual network traffic which is known only after mapping. This quantity is *derived* from the others. However, if it is measured in number of hops, then it is an independent quantity and each link has a latency equal to one while each node has a latency equal to zero. Another example of derived quantity is the bit error rate over wireless communication links that depends on the interference from other nodes in a communication structure. These quantities depend on the abstraction of the specific protocol that is used at the network level and at the lower level of abstraction (e.g., Layer 2 of the OSI protocol stack [21]). For example, packets traveling on a bus incur in different latencies if the protocol is AMBA rather then OCP. To compute derived quantities, that are often used to model specification dependent metrics, we formally introduce the notion of a *model*. Let q denote a derived quantity. Two cases can arise. If the configurations of a component c of a communication structure contain enough information to determine the value of q, then the quantity is *directly derived* from a function $m_q : D_{\mathbf{q}} \rightarrow D_q$, and we call m_q a *direct model* for q. For example, the power dissipated on a link is directly derived from its communication bandwidth. If the computation of the value of q depends not only on the configuration but also on the other components and how they are configured in the communication structure, then the quantity is *indirectly derived* from a function $m'_q : \mathcal{G}_{\mathbf{q}} \times U_{\mathcal{C}} \rightarrow D_q$, and we call m'_q an *indirect model* for q. During the refinement process, some quantities can be determined by models (like latency in our example) while independent quantities are computed by optimization algorithms (like transfer tables in our example).

Example 5. Transfer tables and latency: Fig. 6 shows a bus-based implementation of the set-top box example of Fig. 1. The light-gray arrows represent paths in the communication structures. The paths are implicitly defined by the transfer tables of each bus-node. For example, the transfer table of node v_2 contains an element $(\lambda_{CPU}, \lambda_{dem}, o_3)$ meaning that a packet from the CPU core to the dem core must be sent to output interface o_3. The transfer table information can be used at a lower abstraction level to optimize the bus circuitry (e.g. decoders and multiplexers) or even to segment the bus and insert bus bridges.

The latency to access the bus for each IP core depends on the actual set of components and the bus configuration. When refining the platform instance N_P^1 shown in Fig. 5 into the implementation N_I^2, shown in Fig. 6, a range of latencies $[h_{min}, h_{max}]$ is first considered for the access link (dem, v_1). This range can be computed by a best and worst case analysis of a bus. An indirect model m'_h is used to restrict the range of latencies depending on the actual specification mapped on the implementation. Therefore, the indirect model becomes part of a composition rule that can be state ad follows:

$$\mathcal{R}_I^h = \{ (\mathcal{C}, l) \in 2^{U_{\mathcal{C}}} \times U_{\mathbf{q}_I} \mid l[(h)](c) = m'_h((\mathcal{C}, \mathbf{q}_I, \{l\}), c),\ \forall c \in \mathcal{C} \}$$

The latency of an end-to-end communication is the sum of the latencies of all components in the path. Notice that in this example of bus model we lump the latencies on the access link to the bus and assign a latency equal to zero to each bus segment.

Assuming a 128 bit-wide bus and $200Mhz$ clock frequency, the maximum theoretical throughput is $1.6GB/s$. Hence, we can assign capacities to the links connecting the cores to the bus nodes. Given the capacity assignment, the communication implementation can support a larger set of specifications than the one in Fig. 1. For example, the

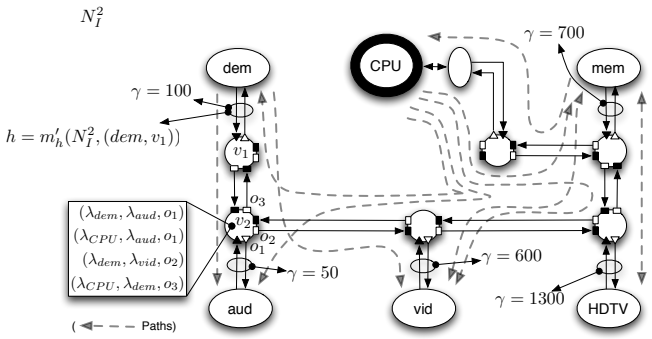

Fig. 6. Example of communication implementation for the set-top box.

throughput of the dem core can be increased up to $100MB/s$. In the rest of this section we define precisely the set of specifications that an implementation can support.

Other examples of composition rules are the following. For each configuration l of a communication structure $N_I(\mathcal{C}, \mathbf{q}_I, L)$, the bandwidth on each link must be less than or equal to the capacity of the link, i.e. $l[(b)] \leq l[(\gamma)]$, A possible additional rule is *deadlock freedom*, which requires the channel dependency graph of N_I to be acyclic [22].

A synthesis problem for on-chip communication is a constrained optimization problem where the decision variables are the components that form the communication implementation together with their configurations, and the constraints come from the specification N_C and the platform (the constraints are detailed in Section III). Therefore, we need to related an implementation to the set of specifications that it can correctly implement, and to a platform instance. We define these relations by abstraction functions as follows.

Given an implementation N_I, a path of length n is a sequence of n links $\pi = (e_1, \ldots, e_n)$ such that $e_i = (v_{i-1}, v_i)$. Even if the topology is such that a path can be found between two nodes in N_I, packets may not be able to flow through the path simply because a node may not have routing capabilities, that are captured by transfer tables. A *real path* from a source node s to a destination node d according to a configuration $l \in L_I$ is such that $v_0 = s$, $v_n = d$ and $\forall e_i \in \pi,\ \exists(\lambda_s, \lambda_d, o) \in l[(\rho)](v_{i-1})$ and $l[(out)](e_i) = o$.

The communication specification characterizing the set of specifications that a communication implementation N_I can correctly implement is given by the *path abstraction* $\Pi : \mathcal{G}_{\mathbf{q}_I} \rightarrow \mathcal{G}_{\mathbf{q}_C}$, which is defined by the following construction:

- the nodes of N_C are the IP cores also present in N_I
- there exists a link $(s, d) \in \mathcal{C}_C$ if and only if there exists a real path from s to d in N_I according to some configuration $l_I \in L_I$;
- a configuration l_C belongs to L_C if an only if there exists a configuration $l_I \in L_I$ such that the following conditions are satisfied:
  1) $l_C[(x, y, \tau)](c) = l_I[(x, y, \tau)](c)$ for all $c \in \mathcal{C}_C$
  2) for all links $e \in \mathcal{C}_I$

$$\sum_{(s,d) \in \mathcal{C}_C : e \in \pi(s,d)} l_C[(b)](s, d) = l_I[(b)](e)$$

3)

$$\sum_{e \in \pi(s,d)} l_I[(h)](e) = l_C[(h)](s, d)$$

To relate implementations and platform instances we introduce the abstraction relation $\Psi : \mathcal{G}_{\mathbf{q}_I} \rightarrow \mathcal{G}_{\mathbf{q}_P}$ that removes the transfer tables and the latency quantities, i.e. given an implementation N_I it returns $\Psi(N_I) = N_P(\mathcal{C}_I, \mathbf{q}_P, L_I[\mathbf{q}_P])$. Given a specification N_C and a platform $\langle \mathcal{L} \rangle$, implementation N_I must satisfy two constraints: $N_C \leq_{\mathbf{q}_C} \Pi(N_I)$ and $\Psi(N_I) \in \langle \mathcal{L} \rangle$. When the implementation is constrained to have a specific topology such as a mesh or a torus, an additional condition $\Psi(N_I) = N_P$ must be satisfied where N_P is the platform instance capturing the specific topology.

III. Formulation of the Optimization Problem

Our objective is to find an implementation N_I that minimizes a given cost function $F : \mathcal{G}_{\mathbf{q}_I} \rightarrow R_+$. We assume that the cost function is monotonic, i.e. $N_1 \leq_{\mathbf{q}_I} N_2 \Rightarrow F(N_1) \leq F(N_2)$. This is a reasonable assumption since a less performing communication structure should also cost less. First, we formulate the problem of configuring a platform instance N_P to implement a specification. The communication synthesis problem can be stated as follows:

$$\begin{aligned}
\texttt{PR1}(N_P): \quad & \min_{\mathcal{C}_I, L_I} && F(N_I) && \\
& \textit{subject to} && N_C \leq_{\mathbf{q}_C} \Pi(N_I), && (1)\\
& && \Psi(N_I) \in \langle \mathcal{L} \rangle && (2)\\
& && \Psi(N_I) \leq_{\mathbf{q}_P} N_P && (3)\\
& && (C_I, l_I) \in \mathcal{R}_I,\ \forall l_I \in L_I && (4)
\end{aligned}$$

Constraints 1 and 2 require N_I to implement the specification and to be a refinement of a platform instance. Constraint 3 requires the implementation to be contained in the performance envelope of the given platform instance N_P and Constraint 4 requires the implementation to satisfy the rules defined at the implementation level (like for instance deadlock freedom). This formulation of the communication synthesis problem has been used in the optimization of NoC with fixed topologies where, for instance, N_P is a mesh [9].

Let Alg be a hypothetical algorithm that solves problem PR1 exactly. Given a library $\mathcal{L}$, platform $\langle \mathcal{L} \rangle$ can be explored by using Alg to solve problem PR1 for each $N_P \in \langle \mathcal{L} \rangle$. In [23], the optimization problem is solved for many instances corresponding to meshes, tori, butterflies and other regular topologies. In [24], the optimization technique explores the isomorphic-free set of all regular topologies and in [25] the authors assume that one N_P is given as input to their algorithm. The following lemma relates the cost of the solution to problem PR1 for different platform instances. [2]

Lemma 1. *Let N_C be a specification, $N_{P,1}$ and $N_{P,2}$ two platform instances such that $N_{P,1} \leq_{\mathbf{q}_P} N_{P,2}$. Let $N^*_{I,1}$ and $N^*_{I,2}$ be the implementations found by Alg for platform instances $N_{P,1}$ and $N_{P,2}$, respectively. Then $F(N^*_{I,2}) \leq F(N^*_{I,1})$.*

[2]The proofs of the lemma and proposition are given in [18].

According to Lemma 1, if we can find the greatest element $\overline{N_P}$ of $\langle\mathcal{L}\rangle$ with respect to the ordering relation $\leq_{\mathbf{q}_P}$, then the solution of problem PR1 with $N_P = \overline{N_P}$ is the best communication structure among all possible platform instances. Unfortunately, such greatest element is not guaranteed to exist in any given platform. Hence, instead of looking for it, we can look for an upper bound $N_P^{\langle\mathcal{L}\rangle}$ of $\langle\mathcal{L}\rangle$ (which is not required to belong to the platform). The existence of an upper bound is related to the platform being finite (i.e. containing a finite number of platform instances). A communication structure is finite if the set of its components is finite. A library is finite if it is a finite set of finite communication structures.

Proposition 1. *Given a vector of quantities* $\mathbf{q}$ *such that each quantity is either finite or bounded, a finite library* $\mathcal{L} \subseteq \mathcal{G}_{\mathbf{q}}$ *and a finite set of valid renaming function* $R' \subset R$*, for any composition rule* $\mathcal{R}$ *and operator* $\oplus_{\mathbf{q}}$*, there exists an upper bound* $N^{\langle\mathcal{L}\rangle} \in \mathcal{G}_{\mathbf{q}}$ *of the communication platform* $\langle\mathcal{L}\rangle$ *with respect to the ordering relation* $\leq_{\mathbf{q}}$*.*

Assuming that the upper bound can be constructed, it follows from Lemma 1 and Proposition 1 that in order to solve the communication synthesis problem we need to solve the optimization problem PR2 $\equiv$ PR1$(N_P^{\langle\mathcal{L}\rangle})$. In general the upper bound $N_P^{\langle\mathcal{L}\rangle}$ does not satisfy the composition rules $\mathcal{R}_P$ (in fact, these rules are not taken into account by the constructive proof of the upper bound itself). Constraint 2 makes sure that the final implementation is a refinement of a platform instance. The solution N_I^* to problem PR2 is the best communication structure that implements the specification among all possible implementations that can be constructed from $\mathcal{L}$ through composition. Notice that, once the upper bound has been found, Constraint 2 is the only constraint that depends on the library $\mathcal{L}$. Thus, the properties of the optimization problem can depend on $\mathcal{L}$ and consequently, an algorithm that solves efficiently the communication synthesis problem could also depend on $\mathcal{L}$.

IV. Application to Network-on-Chip Synthesis

In this section we apply our methodology to the synthesis of NoCs. We present the library of communication components and the models that we use to characterize them. The models include the cost of each element in terms of area and power consumption. We define the composition rules and develop an efficient heuristic algorithm to solve problem PR2, which generally is not a linear program.

A. The Communication Library and the Composition Rules

The nodes of our library are routers and network interfaces. Fig. 7(a) shows the internal architecture of an input-queued router. Given a target technology process, the area and energy dissipation of a router depend on five parameters: number of inputs i, flit-width fw, number of lanes (i.e. virtual channels) vc, queue length l, and number of outputs o. For each configuration of these paramenters, we characterize a router by an energy-per-flit metric $E(i, o, fw, vc, l)$ and an area metric $A(i, o, fw, vc, l)$ that are estimated with Orion [26]. that we obtain not through an analytical model, but by running a series of simulations with Orion [26]. The table in Fig. 7(b) reports the energy values across different router configurations and technology processes. Network interfaces are directly connected to cores. Their characterization in terms of power and area is the same as for the routers. However, their performance in terms of throughput and latency can be very different because they need to provide extra services such as protocol conversion, flit-width adjustment, and packetization.

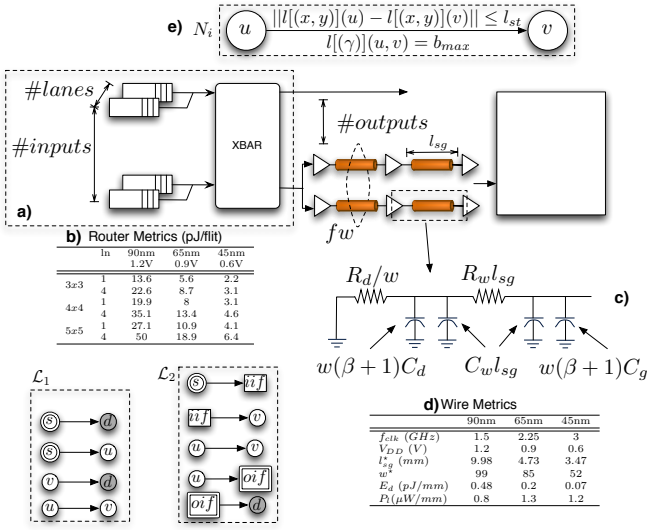

Fig. 7. Modeling the NoC components.

A link is a bundle of wires connecting the output port of a node with the input port of another node. Fig. 7(c) shows the first-order RC model of a buffered wire. For a detailed description we refer the reader to [1], [27], [28]. We use optimally buffered interconnects. For any given technology, the *critical sequential length* is the maximum distance l_{st} that a signal can travel in a target clock period $1/f$. Fig. 7(d) summarizes the metrics of interests for the purpose of NoC synthesis. In particular, each link is characterized by an energy dissipation per bit per unit length E_d/l and an area per bit per unit length, which includes the wiring and buffer areas [3]

Fig. 7(e) shows the basic NoC component N_i, i.e. a link. The set of configurations of a component contains all assignments of positions to the two nodes such that their distance is not greater than the maximum distance l_{st}. The capacity of a link is equal to b_{max} and the latency is equal to one hop. The capacity b_{max} is different from the clock frequency. In fact, in order to avoid router congestion, the capacity of a link should be set in such a way that the routers' injection rate is far from saturation. Otherwise, the actual communication latency would grow exponentially.

In Fig. 7, $\mathcal{L}_1$ and $\mathcal{L}_2$ are two possible communication libraries. There are many types of nodes: s is a source node (without any input), d is a destination node (without any output), u and v are routers, iif is an input interface and oif is an output interface. Since each component in $\mathcal{L}_1$ has the same interface, this library allows establishing direct connections

[3]More details on these models are available in the COSI-OCC manuals [15].

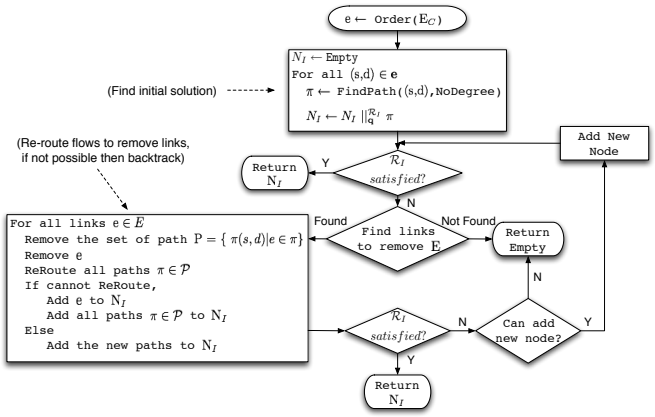

Fig. 8. High-level description of the heuristic algorithm.

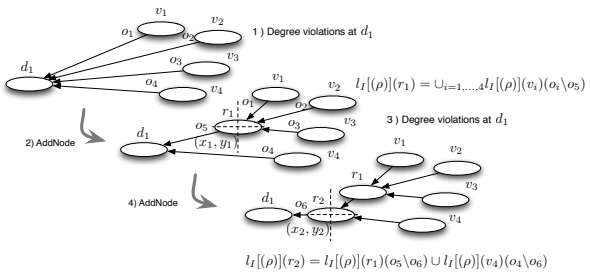

Fig. 9. Procedure for adding a new router to the NoC implementation. For an expression exp, we denote by $exp(x\backslash y)$ the same expression where variable x has been replaced by y.

between a source and a destination. Instead, library $\mathcal{L}_2$, where the source interface iif is different from the destination interface oif, supports a design flow where there are dedicated sockets to connect the cores to the NoC.

Two important composition rules are considered. At the platform level, rule $\mathcal{R}_P$ allows only communication structures where the number of input and output links of a node does not exceed the number of input and output ports, respectively. At the implementation level, rule $\mathcal{R}_I$ allows only deadlock-free communication structures by forcing the channel-dependency graph of an implementation to be acyclic. Moreover, the bandwidth on any channel cannot exceed its capacity.

B. Solution to the Optimization Problem

We use the results of Section III to solve the optimization problem. We begin the optimization process by assigning a fixed position to each core with a floor-planner. We assume that placing the network over the cores is not allowed. Hence we identify the area A_C on the chip that is available to lay out the network because it is not occupied by the IP cores [4].

An automatic procedure takes care of computing and discretizing A_C so that the set of positions $D_{(x,y)}$ is finite. Quantity τ is also finite and quantity γ is bounded, moreover we assume to have a finite library. Because we want $l_I[(x,y,\tau)]$ to be injective (i.e. only one component of a specific type can be installed in a particular location), the maximum number of nodes in any platform instance is limited to $|D_{(x,y,\tau)}|$. Thus, let $U'_V \subset U_V$ be a set of nodes such that $|U'_V| = |D_{(x,y,\tau)}|$. The set of valid renaming functions R' is such that each node of the library elements is renamed to one of the nodes in U'_V. Hence, it is possible to find an upper bound $N_P^{\langle\mathcal{L}\rangle}$ following the construction of Proposition 1.

Problem PR2 is non-linear and discrete. The cost of nodes is a non-linear function of the number of inputs and outputs and not all of the constraints are linear. We could try to linearize the problem and solve it using Integer Linear Programming (ILP). To illustrate this approach, consider library $\mathcal{L}_1$ of Fig. 7. First, we have to linearize the cost function by assuming that the cost of a router is the sum of the cost of each input port plus the cost of each output port. Then, we define the energy per flit as $\min_{i,j}[E(i+1,j) - E(i,j)]$ for an input port and as $\min_{i,j}[E(i,j+1) - E(i,j)]$ for an output port (and, similarly, the leakage power and area occupation). This linearized cost function is a lower bound of the real cost of the network. Hence solving the ILP with this cost function returns a solution that is optimistic. Using one binary variable for each installation site denoting whether a router is installed at that site, one binary variable for each link that can be installed between two sites, and one binary variable to denote that a constraint is routed through a link, the number of variables of the ILP problem becomes very large. It is equal to $|U'_V|^2 \cdot |E_C| + |U'_V|^2 + |U'_V|$ where the first term is the square of the number of installation sites times the number of constraints. For the simple example of Fig. 1 with 70 installation sites, the number of binary variables is $93,170$. These many variables cause an ILP solver to run very slow. Moreover, some composition rules (e.g deadlock freedom) cannot be included in the ILP since they are highly non linear. Because of these difficulties, we devised a heuristic approach to solve problem PR2. In Section V, we compare the results obtained by the heuristic with a lower bound provided with a further optimistic approximation of the ILP formulation.

1) Structure of the Algorithm: Fig. 8 shows the high-level structure of the heuristic algorithm. In the first step we find an initial solution with the same technique that is used in algorithms for global routing: the end-to-end constraints in E_C are first ordered by decreasing bandwidth. One path in $N_P^{\langle\mathcal{L}\rangle}$ is derived for each constraints one at a time (the actual implementation of procedure FindPath depends on the composition rules). During this step, possible rule constraining the maximum degree of a network node are not taken into account; however, if we are lucky, N_I may still satisfy the degree rule, in which case the algorithm returns N_I and stops. Otherwise, we activate an iterative procedure to remove the degree constraint violations.

This procedure implements a rip-up and reroute approach one link at a time. The links connected to the output of nodes with output degree violations and links connected to the input of nodes with input degree violations are the ones that are considered for rip-up and re-route. For each link, all

[4]In COSI-OCC this area can be defined in two ways as input to our tool. The user can decide to reserve a small amount of area around each core that is made available for the communication architecture. Alternatively, the user can define "virtual" communication cores and place them on the chip to reserve space for the installation of communication components.

Procedure `Reach` $(N_I, v, \mathcal{R}, \mathcal{L})$

$N_R \leftarrow$ empty communication structure ;
forall $N_i \in \mathcal{L}$ *with* $C_i = \{v_i, u_i, (u_i, v_i)\}$ **do**
  forall $l_i \in L_i$ **do**
1 **if** $l_I(v) = l_i(u_i)$ **then**
2 let $N_i'(\mathcal{C}_i, \mathbf{q}_P, \{l_i\})$;
3 $N_i'' \leftarrow r(N_i)$, with $r(u_i) = v, r(v_i) = id(l_i(u_i))$;
4 **if** $\Psi(N_I) \|_{\mathbf{q}}^{\mathcal{R}} N_i''$ *is defined* **then**
  $N_R \leftarrow N_R \|_{\mathbf{q}}^{\mathcal{R}} N_i''$
return N_R ;

source-destination paths containing that link are re-routed by procedure `FindPath` that now takes into account the degree constraint rule. If a path cannot be removed, the algorithm back-tracks by reinserting the link and all the paths. Otherwise, the new paths are added to the communication implementation. If the re-routing procedure finds an implementation that satisfies the composition rules, the algorithm ends with success.

Otherwise we try adding a new node (router) to yield a feasible solution. 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 (Fig. 9). However, if no node can be added (e.g., because delay constraints would be violated) the algorithm ends with an empty implementation implying that no solution was found. Fig. 9 shows how new nodes are added. First, among all nodes with input/output degree violations, the one with the highest number of input/output links is selected. All input/output links to/from the node are candidates for the merge operation. A subset of them is chosen with a criterion that depends on the optimization goal. The source and target nodes of the selected links are connected to the new router, which is instantiated in a position that minimizes the cost of the links. The transfer table of the router is set according to the new paths flowing through it. As it executes this local transformation the algorithm makes sure that link capacities and degree constraints are not violated. Note that the merge operation does not change the number of nodes with degree violations.

2) The `FindPath` *procedure:* This procedure is available in different forms to search for the "best" path between a source and a destination core depending on the given composition rules. If a delay model must be taken into account to check delay constraints, the best path is discovered by a labeling algorithm (`SpLabeling`) that finds the minimum-cost constrained shortest path between two nodes; a modified version of Dijkstra's shortest path algorithm is used otherwise. If deadlock freedom is included in the set of rules $\mathcal{R}_I$, then `FindPath` runs on the channel-dependency graph of the communication implementation to make sure that it remains acyclic. The degree constraints of the nodes can be taken into account by adding a rule denoted $\mathcal{R}_{dg}$.

`FindPath` explores the upper bound $N_P^{\langle\mathcal{L}\rangle}$ without building an explicit representation. In fact $N_P^{\langle\mathcal{L}\rangle}$ is explored locally at run-time by procedure `Reach`. This procedure takes as input parameters the current communication implementation N_I, a node $v \in \mathcal{C}_I$, the composition rules $\mathcal{R}$, and the platform

Procedure `SpLabelling` $(s, d, N_I, \mathcal{L}, \mathcal{R}, \mathcal{R}_I)$

1 $D[s] \leftarrow \{(0,0)\}$, $D[v] \leftarrow \emptyset$, $\forall\, v \in U_V' \setminus \{s\}$;
2 $Q \leftarrow (s, (0,0))$;
while $Q \neq \emptyset$ **do**
3 $(v, D_v) \leftarrow$ `ExtractMin` (Q) ;
4 $N_\pi \leftarrow$ path from $(s, (0,0))$ to (v, D_v) ;
   $N_R \leftarrow$ `Reach` $(N_I \|_{\mathbf{q}_I} N_\pi, v, \mathcal{R}_P, \mathcal{L})$;
5 **forall** $(v, u) \in \mathcal{C}_R$ **do**
   $l \leftarrow$ `Configure` $(L_I, (v, u))$;
   define $N'(\{(u, v)\}, \mathbf{q}_I, \{l\})$;
6 **if** $N_I \|_{\mathbf{q}_I}^{\mathcal{R}_I} N'$ *is defined* **then**
7 $f \leftarrow$ Compute incremental area and power ;
8 $D_u = (D_v.H + 1, D_v.C + f)$;
   if $\nexists D \in D[u]$ *s.t.* $D < D_u$ **then**
9 $D[u] \leftarrow D[u] \cup \{D_u\}$;
10 `Insert` $(Q, (u, D_u))$;
11 $\pi[(u, D_u)] \xleftarrow{N'} (v, D_v)$;
12 **if** $D[d] = \emptyset$ **then**
  return $\emptyset$;
else
  return `ToGraph` (π) ;

library $\mathcal{L}$. `Reach` checks which links can be instantiated with the source node v (Line 1). For each link, an instance is generated by renaming the nodes appropriately (Lines 2 and 3). Function id associates a unique identifier to a node depending on its type and position. If the new link can be composed with the communication implementation without violating the composition rules (Line 4), then it is added to the reachable communication structure N_R (note that $N_R \in \mathcal{G}_{\mathbf{q}_P}$, therefore the set of rules $\mathcal{R}$ must contain $\mathcal{R}_P$).

Procedure `SpLabelling` is a particular implementation of `FindPath`. It solves the constrained shortest-path problem [29] using a labeling algorithm [30]. We use the number of hops as a model for latency. A distance label is a tuple $D = (H, C)$ associated to a node v where H is the number of hops of the path from the source s to v with minimum cost C. A distance label D is dominated by D', written $D < D'$ if $D.H \leq D'.H$, $D.C \leq D'.C$, and $D.H \neq D'.H \vee D.C \neq D.C'$. A set of distance labels $D[v]$ is associated to each node v. The queue Q contains pairs (v, D) where v is a node and $D \in D[v]$ is a distance label of v. Distance labels in the queue are ordered by number of hops and, for the same number of hops, by cost. The procedure starts with an empty set of distance labels for all nodes but the source, which has the distance label $\{0, 0\}$. The pair $(s, \{0, 0\})$ is the only element in the queue (Line 2). The minimum distance label node is extracted from the queue (Line 3) and the set of possible links departing from the node (computed by procedure `Reach`) is processed (Lines 4 and 5). Each link is first configured by selecting one possible configuration, then composition rules are checked (Line 6). If this distance label is not dominated by any other already present at u, then it is added to the set of distance labels of u (Line 9), the new pair is added to the queue (Line 10), and the predecessor tree is updated (Line 11). A path from s to d that satisfies the hop constraint exists if the set of distance labels at d is not empty (Line 12). If this

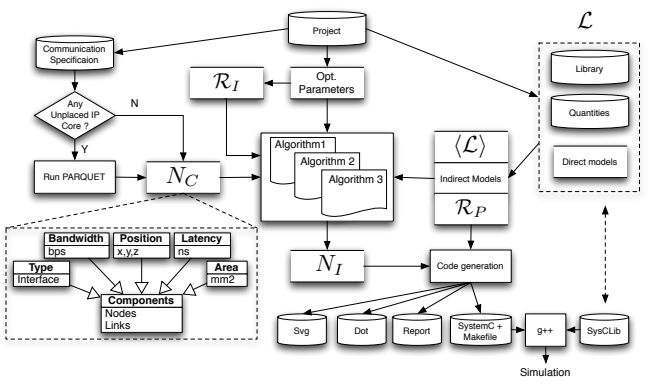

Fig. 10. The COSI-OCC design environment built on the COSI infrastructure.

is the case, the path with minimum cost C is selected and returned. During the construction of the initial solution, the composition rules $\mathcal{R}$ and $\mathcal{R}_I$ do not contain rule $\mathcal{R}_{dg}$, which is added during the re-routing procedure instead.

V. Implementation and Experimental Results

In this section, first we briefly describe COSI-OCC, the on-chip communication design flow that we developed, them we present some results that we obtained by applying it to a number of test cases for NoC design.

A. The COSI-OCC Design Flow

COSI-OCC is an open-domain software package that implements the methodology presented in the previous sections. Since our methodology is quite general, COSI-OCC allows for the use of different libraries, compositional rules, and synthesis algorithms. In this paper, we focus on the use of COSI-OCC to solve NoC design problems using the formulation and the heuristic algorithm presented in Section IV. COSI-OCC is part of the COSI project [15], [17], [18]. COSI belongs to the class of component composition frameworks (CCFs) such as BALBOA [31], LSE [32], SPARTACAS [33], and MCF [34]. A detailed comparison is available in [18].

Fig. 10 shows the software organization of COSI-OCC. The input to COSI-OCC is a project file that contains pointers to the communication specification file and to the library file. The former contains a list of IP cores and inter-core communication constraints. If there are unplaced IP cores, PARQUET is used to floor-plan the chip [20]. The library file contains the description of each library element, the quantities attached to them, and the parameters needed to compute the value of directly derived quantities. The library is used to construct the platform data structure that contains the composition rules including models for indirectly derived quantities. The project file includes also the optimization parameters such as the relative weights of power and area cost. The communication specification and the platform are passed to the synthesis algorithm that derives the network implementation N_I.

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 corresponding SYSTEMC-view of each element instantiated from the library that is contained in `SysCLib`, also part of the COSI-OCC distribution. The generation of the SYSTEMC netlist is a further refinement of N_I that requires the binding of each port of the nodes to links, the generation of the routing tables, and the computation of the weights for the *weighted fair queuing* algorithm, which is used by the routers for flit scheduling. The COSI-OCC distribution includes a set of algorithms to solve some variants of the communication synthesis problem, e.g. an algorithm that generates deadlock-free networks. Our approach to solve this problem is different from the one proposed in [12] that is based on prohibited turns. In COSI-OCC the optimization algorithm operates directly on the channel-dependency graph of the communication structure and at run-time checks that such graph is kept acyclic (i.e. it checks that the corresponding composition rule is satisfied).

Name	$\lvert V_C \rvert$	$\lvert E_C \rvert$	Area (mm^2)	Total Bw. ($Gbps$)	Ref.
MWD	12	13	3×4	8.96	[8]
MPEG4	12	27	3×2.35	27.8	
VOPD	12	15	1.53×1.18	27.9	
dVOPD	26	34	2×2.23	66.6	[35]
tVOPD	38	51	2.78×2.37	98.84	
VProc	42	69	8×6	78.2	

TABLE I
CHARACTERISTICS OF THE SELECTED SOCS APPLICATIONS.

B. Test Cases and Experimental Results

Table I lists the SoCs that we used in our experiments. We selected the test cases based on several criteria:

- the number of IP cores $|V_C|$, ranging from 12 to 42, and the size of the chip, as large as $48mm^2$;
- the total bandwidth requirement, defined as the sum of the bandwidth requirements over all end-to-end constraints E_C, and ranging from 9 to 99 $Gbps$;
- the maximum input degree of a destination core and the maximum output degree of a source core, ranging from 2 to 25 depending on the SoC application.

The goal of our experiments is to study the impact of these application features on the synthesized NoC. Specifically we are interested in the following metrics: the power and area breakdown, the maximum and average input and output degree of the nodes, the maximum and average number of hops among all source-destination paths, and the maximum and average latency. The latency measurements are obtained by simulating the SYSTEMC implementation of the NoC generated by COSI-OCC. The maximum latency is the largest end-to-end delay experienced by any packet over the entire simulation, i.e. the time that elapses from the generation of the head flit to the delivery of the tail flit to the destination. The average latency is computed by dividing the sum of the latencies of each packet over the simulation run by the total number of flits sent. The SYSTEMC model of the NoC implements wormhole routing and weighted round robin packet scheduling. Moreover, each packet has one header flit, one tail flit, and four payload flits.

1) Impact of the Application Characteristics: The SoC applications used in this experiment were: a Multi-Window

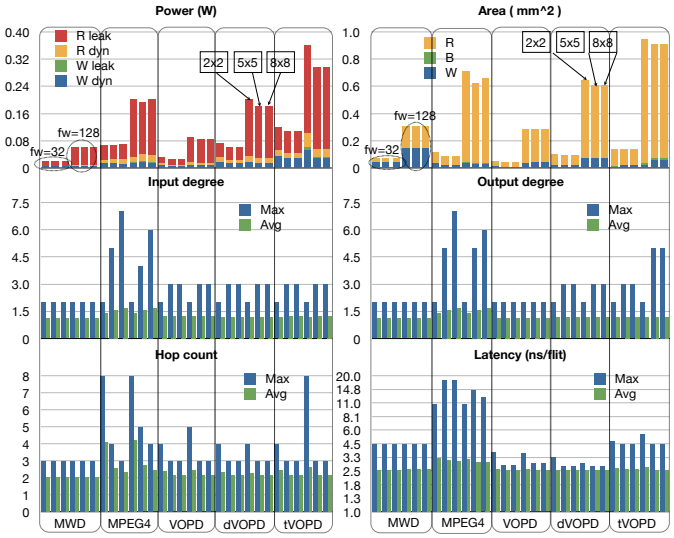

Fig. 11. Properties of the synthesized NoCs for the MWD, MPEG4, VOPD, dVOPD and tVOPD applications. Power is expressed in $Watts$, area in mm^2 and latency in $ns/flit$. We used the following notation: `R` for routers, `W` for wires, `B` for sequential buffers. Latency is reported on a logarithmic scale.

Displayer (MWD), an MPEG4 decoder (MPEG4), a Video Object Plane Decoder (VOPD) as well as two applications, called dVOPD and tVOPD obtained by instantiating two and three VOPDs, respectively sharing a common memory. We assumed a $90nm$ technology and a target clock frequency $f = 1.5GHz$ [5]. The link capacity b_{max} was set to $1.12GBps$. We used six libraries of communication components differing for the flit-width of the data path (32 and 128 bits corresponding to $280 \cdot 10^6$ and $70 \cdot 10^6$ flits per second, respectively) and the size of the largest switch available in the library (2×2, 5×5 and 8×8). The results are reported in Fig. 11. Each histogram is divided into five zones, one for each application. Each zone contains six bars, one for each library.

The power consumption and the area occupied by the NoC are increasing functions of the total bandwidth requirement. Most test cases do not need the instantiation of large routers. For example, the number of input and output ports on each router in the NoCs supporting the MWD and the VOPD applications is no greater than two since each core is a source and/or destination of few communication constraints. These NoCs are basically a set of dedicated point-to-point links with very little sharing and the synthesis algorithm avoids the use of costly routers. Hence, the difference between the maximum and the average latency is small.

The dVOPD and tVOPD applications show the effect of merging different communications into a common link. In these applications, a central memory is shared among a few cores. Since the memory has only one input and one output port, one or more routers are needed to merge concurrent memory accesses via time multiplexing. Allowing the installation of larger routers provides two advantages: (1) the total power consumption is reduced (by 14% and 12% for dVOPD and tVOPD, respectively) thanks to a reduced hop count, and (2) the end-to-end latency (both the maximum and average value) is reduced. The latency decrease is modest compared to the reduced number of hops because the time spent for contention among the input queues grows with the router size. Generally, however, for these applications the reduced number of hops counterbalances the negative effect due to contention.

In the MPEG4 application, the SDRAM is shared among many more cores than in the case of the dVOPD and tVOPD applications. Hence, to use larger routers does not give the same benefits. Despite the significant differences between the maximum number of hops in the 2×2 and 8×8 cases, there are no gains in terms of maximum latency, which in fact is even worse for larger routers (a 73% increase with respect to the 2×2 case). Here, port contentions cannot be counterbalanced by the reduced hop count, as opposed to the dVOPD and tVOPD cases where routers have no more than five inputs.

This set of test cases shows that the power consumption and the area occupied by the NoC implemented with 32-bit links is much smaller than in the 128-bit implementation. The latter case gives smaller link utilization (i.e. flit rate), which reduces the latency due to contention. This, however, is a minor gain and does not justify the use of wider data parallelism, which should be limited to cases when the required bandwidth cannot be achieved with narrower links.

2) Effect of Technology Scaling: For the second set of experiments we selected the VProc SoC as a representative embedded system application and we studied the impact of scaling the technology on the performance and cost of the synthesized NoC. VProc features a central memory that serves 25 different cores. Each core requires a write and read bandwidth of $960Mbps$ (in each direction). In this case we expect the algorithm to use routers because the central memory has only one read and one write port. Technology scaling generally enables higher transistor densities and clock frequencies. Hence, as we scale the technology we double the bandwidth requirements from each core to the central memory while keeping the core size fixed. This choice mimics the fact that two cores can fit in the area of one, as the transistor density doubles with the new process generation.

We used a total of nine libraries obtained as the combination of three different technology processes with three different router designs: specifically, we used $90nm$, $65nm$, and $45nm$ technologies while the routers' maximum size was set equal to 2×2, 5×5 and 8×8, respectively. Again the targer clock frequency was $f = 1.5Ghz$ at $90nm$. Since the total memory bandwidth is $3GBps$, we set the flit width to 128 bits to achieve a link capacity of $3.2GBps$ with a maximum flit rate of $200 \cdot 10^6$ per input port of the routers. We increased f to $2.25GHz$ and $3GHz$ and the link capacities to 6.4 and $12.8GBps$ for the 70 and $45nm$ process, respectively. The synthesis constraints were set as follows: the density of the installation sites was fixed to 20, which gives a total of 364 different possible locations to install the routers; the synthesis goal is minimum latency with a constraint that each path be no longer than 10 hops. The results are reported in Fig. 12.

The critical sequential length drops from $9.98mm$ at $90nm$ down to $3.47mm$ at $45nm$ due to the different electrical parameters of the wires and also to the increased clock frequency

[5]The clock frequency is not the result of hardware synthesis but it is a constraint for the NoC synthesis. In [36] we also characterize interconnect elements using low level implementations.

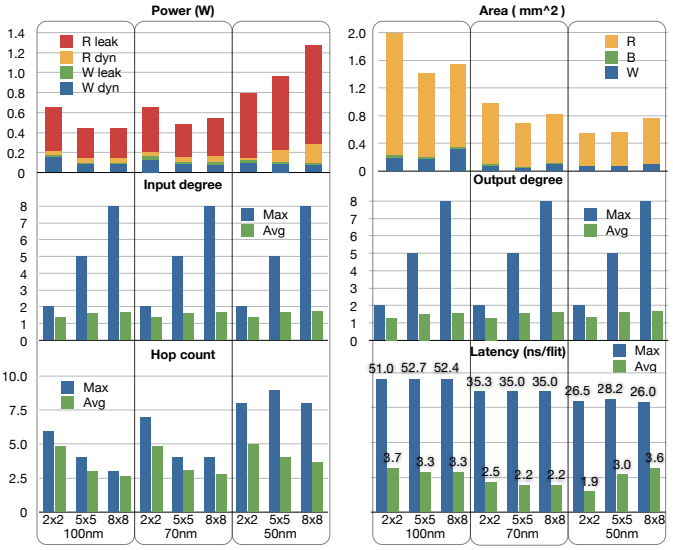

Fig. 12. Properties of the synthesized NoCs for the VProc applications.

(from 1.5 to 3 GHz). Since the chip size is $8 \times 6mm^2$, the entire chip can be spanned in one clock cycle at $90nm$ while 3 cycles are needed at $45nm$. Hence, it is not surprising that the maximum number of hops at $45nm$ does not change much across the various router configurations because intermediate stateful repeaters such as relay stations [37] are needed to segment long interconnection links. The use of larger routers does not help to reduce the number of hops while it increases the chance of contention. Therefore, contrary to the $90nm$ case, the reduced number of hops does not balance the higher contention probability and, ultimately, the average latency obtained by simulation actually increases.

In terms of power consumption, the advantage of using larger routers does not persist as we scale from $90nm$ to $45nm$ due to the higher target clock frequency and leakage power dissipation. If stateful repeaters are needed to span large distances among the cores, it is more efficient to spatially distribute small routers on the chip. Finally, the results highlight the need for more accurate timing models for synthesis rather than the simple measure based on hop count.

3) Quality of the Solution: Ideally, we would like to compare the *exact* solution of problem PR2 with the one found by our heuristic algorithm. However, the best we can do is to compare our results with a lower bound since even the relaxed ILP algorithm sketched in Section IV-B has prohibitive running times for our test cases. We relaxed the problem further by converting the ILP into a Linear Program (LP), which we solved with CPLEX [38]. We then computed the ratio of the power consumption of the solution found by CPLEX over the one of our heuristic algorithm across various benchmarks. Table II reports these results together with the number of positions $|D_{(x,y)}|$, the computation time 't_{cpu} LP' of CPLEX and the computation time 't_{cpu} H' of our heuristic.

In most cases our heuristic algorithm is 2-3 orders of magnitude faster than solving the LP (a remarkable fact since the LP does not find a feasible solution). The power of the NoC found by the heuristic is within 2x from the power found by CPLEX that is very optimistic for the change in the cost function and for the relaxation of the integer constraints.

Name	$\lvert D_{(x,y)}\rvert$	t_{cpu} LP	t_{cpu} H	Ratio
MWD 2x2	94	4.76	0.11	1
MWD 5x5	94	4.83	0.11	1
MWD 8x8	94	4.78	0.11	1
MPEG4 2x2	117	434	5.29	0.49
MPEG4 5x5	117	479	1.46	0.55
MPEG4 8x8	117	394	1.32	0.48
VOPD 2x2	63	1.98	0.13	0.73
VOPD 5x5	63	0.87	0.13	0.78
VOPD 8x8	63	0.85	0.13	0.78
dVOPD 2x2	147	130	1.8	0.69
dVOPD 5x5	147	60	1.66	0.66
dVOPD 8x8	147	60	1.65	0.66
tVOPD 2x2	150	438	4.54	0.71
tVOPD 5x5	150	423	3.32	0.66
tVOPD 8x8	150	426	3.34	0.66

TABLE II
EVALUATING THE HEURISTIC ALGORITHM OF FIG. 8.

VI. CONCLUSIONS AND FUTURE WORK

We presented a design methodology with a supporting tool infrastructure that follows the Platform-Based Design paradigm and relies on a solid mathematical foundation to model, compose, and optimize communication networks. The communication specification is given as a point-to-point network. A mathematical formalism is used to model the platform that supports on-chip communication (OCC) design. The platform captures all possible communication structures that can be built by assembling the components from the target communication library. We formulated a general optimization problem for OCC synthesis that applies to a wide class of libraries. Then we applied the methodology to the NoC synthesis problem and we proposed an efficient heuristic to solve it. Finally, we presented two sets of experiments made with COSI-OCC, an on-chip communication synthesis design flow that we developed as part of the COmmunication Synthesis Infrastructure (COSI) to support the proposed methodology.

REFERENCES

[1] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 490–504, April 2001.
[2] J. D. Meindl, "Interconnect opportunites for gigascale integration," *IEEE Micro*, 2003.
[3] L. P. Carloni and A. L. Sangiovanni-Vincentelli, "Coping with latency in SOC design," *IEEE Micro*, vol. 22, no. 5, pp. 24–35, Sep-Oct 2002.
[4] OCP-IP. [Online]. Available: http://www.ocpip.org/home
[5] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Vberg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era," in *Proc. of the IEEE NorChip Conference*, Nov. 2000.
[6] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. of the Design Automation Conf.*, June 2001.
[7] L. Benini and G. D. Micheli, "Networks on chip: A new SoC paradigm," *IEEE Computer*, 2002.
[8] 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*, vol. 16, no. 2, pp. 113–129, Feb. 2005.
[9] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 551–562, Nov. 2005.
[10] K. Lahiri, A. Raghunathan, and S. Dey, "Design space exploration for optimizing on-chip communication architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 23, no. 6, pp. 952–961, Dec. 2004.

[11] U. Ogras and R. Marculescu, "Application-specific network-on-chip architecture customization via long-range link insertion," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov. 2005.
[12] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," in *Proc. Intl. Conf. on Computer-Aided Design*, Nov. 2006, pp. 355–362.
[13] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 4, pp. 407–420, Apr. 2006.
[14] ——, "Application specific network-on-chip design with guaranteed quality approximation algorithms." in *ASPDAC*, January 2006.
[15] "http://embedded.eecs.berkeley.edu/cosi/."
[16] R. K. Brayton, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. CAD-6, no. 6, pp. 1062–1081, Nov. 1987.
[17] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli, "COSI: A framework for the design of interconnection networks," *IEEE Design & Test of Computers*, vol. 25, no. 5, Sep-Oct 2008.
[18] A. Pinto, L. Carloni, and A. L. Sangiovanni-Vincentelli, "A methodology for constraint-driven synthesis of on-chip communications," Department of EECS, University of California at Berkeley, Tech. Rep., 2008.
[19] A. Sangiovanni-Vincentelli, "Defining platform-based design," *EEDesign of EETimes*, February 2002.
[20] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning : Enabling hierarchical design," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 1120–1135, December 2003.
[21] *ISO/IEC 7498-1, Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, 1994.
[22] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, no. 5, pp. 547–553, 1987.
[23] S. Murali and G. D. Micheli, "SUNMAP: A tool for automatic topology selection and generation for NOCs," in *Proc. of the Design Automation Conf.*, June 2004, pp. 914–919.
[24] Y. Hu, H. Chen, Y. Zhu, A. A. Chien, and C.-K. Cheng, "Physical synthesis of energy-efficient networks-on-chip through topology exploration and wire style optimization," in *ICCD*, 2005, pp. 111–118.
[25] A. Hansson, K. Goossens, and A. Rădulescu, "A unified approach to constrained mapping and routing on network-on-chip architectures," in *CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2005, pp. 75–80.
[26] 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*, Nov. 2002, pp. 294–305.
[27] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*. Reading,MA: Addison-Wesley, 1990.
[28] S. Heo and K. Asanovic, "Replacing global wires with an on-chip network: a power analysis," in *Proc. of the Intl. Symp. on Low Power Electronics and Design*, 2005, pp. 369–374.
[29] G. Handler and I. Zang, "A dual algorithm for the constrained shortest path problem," *Networks*, 1980.
[30] M. Desrochers and F. Soumis, "A generalized permanent labelling algorithm for the shortest path problem with time windows," *Information Systems and Operations Research*, vol. 26, no. 3, pp. 191–212, 1988.
[31] F. Doucet, S. K. Shukla, M. Otsuka, and R. K. Gupta, "Balboa: a component-based design environment for system models," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 12, pp. 1597–1612, 2003.
[32] M. Vachharajani, N. Vachharajani, and D. August, "The liberty structural specification language: A high-level modeling language for component reuse," in *Proc. of the Conf. on Programming Language Design and Implementation*, June 2004, pp. 195–206.
[33] B. Morel and P. Alexander, "Spartacas automating component reuse and adaptation," *IEEE Trans. Softw. Eng.*, vol. 30, no. 9, pp. 587–600, 2004.
[34] D. A. Mathaikutty and S. K. Shukla, "MCF: A metamodeling based component composition framework – composing SystemC IPs for executable system models," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 7, pp. 792–805, 2008.
[35] 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.
[36] L. Carloni, A. B. Kahng, S. Muddu, A. Pinto, K. Samadi, and P. Sharma, "Interconnect modeling for improved system-level design optimization," in *Proc. of the Asia and South Pacific Design Automation Conference*, 2008, pp. 258–264.
[37] L. P. Carloni, K. L. McMillan, A. Saldanha, and A. L. Sangiovanni-Vincentelli, "A methodology for "correct-by-construction" latency insensitive design," in *Proc. Intl. Conf. on Computer-Aided Design*. IEEE, Nov. 1999, pp. 309–315.
[38] "CPLEX." [Online]. Available: http://www.ilog.com/products/cplex/

Alessandro Pinto is a Research Scientist in the Embedded Systems and Networks group at the United Technologies Research Center, East Hartford, Connecticut. His research interests are in the field of networked embedded systems with particular emphasis on formal models, design methodologies and tools. Dr. Pinto received a Ph.D. degree in Electrical Engineering and Computer Sciences from the University of California at Berkeley in 2008, and a M.S. degree in Electrical Engineering in 2003 from the same university. He holds a Laurea degree from the University of Rome "La Sapienza". In 1999, Dr. Pinto spent one year as a consultant at Ericsson Lab Italy in Rome, Italy, working on the design of system-on-chips. He consulted for the same company from 2000 to 2001, developing system-level design flows for wireless access networks. He is a member of the IEEE.

Luca P. Carloni received the Laurea degree (summa cum laude) in electrical engineering from the Università di Bologna, Italy, in 1995, and the M.S. and Ph.D. degrees in electrical engineering and computer sciences from the University of California, Berkeley, in 1997 and 2004, respectively.

He is currently an Assistant Professor with the Department of Computer Science, Columbia University, New York, NY. He has authored over 50 publications and is the holder of one patent. His research interests are in the area of design tools and methodologies for integrated circuits and systems, distributed embedded systems design, and design of high-performance computer systems.

Dr. Carloni received the Faculty Early Career Development (CAREER) Award from the National Science Foundation in 2006 and was selected as an Alfred P. Sloan Research Fellow in 2008. He is the recipient of the 2002 Demetri Angelakos Memorial Achievement Award "in recognition of altruistic attitude towards fellow graduate students." In 2002, one of his papers was selected for "The Best of ICCAD", a collection of the best IEEE International Conference on Computer-Aided Design papers of the past 20 years. He is a member of the IEEE and the IEEE Computer Society.

Alberto L. Sangiovanni-Vincentelli (Fellow, IEEE) holds the Buttner Chair of Electrical Engineering and Computer Sciences at the University of California at Berkeley. He was a cofounder of Cadence and Synopsys, the two leading companies in the area of electronic design automation. He is the chief technology adviser of Cadence. He is a member of the board of directors of Cadence, UPEK (a company he helped spin off from ST Microelectronics), Sonics, and Accent (an ST Microelectronics-Cadence joint venture he helped found). He was a member of the HP Strategic Technology Advisory Board and is a member of the Science and Technology Advisory Board of General Motors. He consulted for many companies, including Bell Labs, IBM, Intel, United Technology, COMAU, Magneti Marelli, Pirelli, BMW, Daimler-Chrysler, Fujitsu, Kawasaki Steel, Sony, and Hitachi. He is the founder and Scientific Director of PARADES, a European Group of Economic Interest supported by Cadence and ST Microelectronics. He is a member of the High-Level Group and of the steering committee of the EU Artemis Technology Platform. In 1981, he received the Distinguished Teaching Award of the University of California. He received the worldwide 1995 Graduate Teaching Award of the IEEE for "inspirational teaching of graduate students." In 2002, he was the recipient of the Aristotle Award of the Semiconductor Research Corporation. In 2001, he was given the prestigious Kaufman Award of the Electronic Design Automation Council for pioneering contributions to EDA. He is an author of more than 800 papers and 15 books in the area of design tools and methodologies, large-scale systems, embedded controllers, hybrid systems and innovation. Dr. Sangiovanni-Vincentelli has been a fellow of the IEEE since 1982 and a member of the National Academy of Engineering since 1998.