An Application-Specific Design Methodology for On-Chip Crossbar Generation

Srinivasan Murali, *Student Member, IEEE*, Luca Benini, *Fellow, IEEE*, and Giovanni De Micheli, *Fellow, IEEE*

Abstract*—Designing a power-efficient interconnection architecture for MultiProcessor Systems-on-Chips (MPSoCs) satisfying the application performance constraints is a nontrivial task. In order to meet the tight time-to-market constraints and to effectively handle the design complexity, it is essential to provide a computer-aided design tool support for automating this task. In this paper, we address the issue of "application-specific design of optimal crossbar architecture" satisfying the performance requirements of the application and optimal binding of the cores onto the crossbar resources. We present a simulation-based design approach that is based on the analysis of the actual traffic trace of the application, considering local variations in traffic rates, temporal overlap among traffic streams, and criticality of traffic streams. Our approach is physical design aware, where the wiring complexity of the crossbar architecture is also considered during the design process. This leads to detecting timing violations on the wires early in the design cycle and to having accurate estimates of the power consumption on the wires. We apply our methodology onto several MPSoC designs, and the synthesized crossbar platforms are validated for performance by cycle-accurate SystemC simulation of the designs. The crossbar matrix power consumption values are based on the synthesis of the register transfer level models of the designs, obtained using industry standard tools. The experimental case studies show large reduction in communication architecture power consumption (45.3% on average) and total wirelength (38% on average) for the MPSoC designs when compared with traditional design approaches. The synthesized crossbar designs also lead to large reduction in transaction latencies (up to 7×) when compared with the existing design approaches.

***Index Terms*—Application specific, bus, crossbar, floorplan, Networks-on-Chips (NoCs), SystemC, Systems-on-Chips (SoCs), timing closure.**

I. Introduction

WITH technology scaling, the number of processor and memory cores integrated on a single chip and their speed of operation are increasing. Such MultiProcessor Systems-on-Chips (MPSoCs) combine several different embedded processors, memories, and specialized hardware units to provide a complete integrated system [1]. There are several MPSoCs that are already available in the market, such as the Philips Nexperia digital video platform [2], the TI OMAP platform [4] for third-generation wireless applications, and the ST Nomadik platform [3] for multimedia applications. In the next few years, many more MPSoCs that support a variety of application scenarios are expected to be available in the market [1].

As technology advances, the ratio of wire delay to gate delay increases as wire scaling is not at par with transistor scaling. This, coupled with the fact that the number of communicating components in the chip and their speed of operation are increasing, has led to the scenario where the communication between cores is a major bottleneck for system performance [6]–[9]. Traditional communication architectures such as a single shared bus or bridged buses are inherently nonscalable and will not be able to support the heavy communication traffic [10]. A communication-centric design approach, i.e., Networks-on-Chips (NoCs), has recently emerged as the design paradigm for designing a scalable communication infrastructure for MPSoCs [6]–[9]. The need for scalable communication architectures is reflected in the recent trend that many of the standard bus products such as the Advanced Microcontroller Bus Architecture (AMBA)[1] [from ARM] and the STbus[2] (from STMicroelectronics) have now introduced the capability of designing a crossbar with multiple buses operating in parallel, thus providing a low-latency and high-bandwidth communication infrastructure.

The MPSoC platforms target high-volume markets and need to support aggressive performance under tight power budgets. Many MPSoCs are used in mobile applications such as cellular phones and wireless devices, where the peak power consumption of the system is limited to reduce the amount of heating of the system. Reducing the power consumption[3] of the system is also essential to have a reliable system operation [15]. Thus, it is important to achieve a power-efficient design of the communication architecture. As the MPSoCs support multiple different applications on the same chip, the communication traffic characteristics vary over time, and it is important to consider these variations during interconnect synthesis. Due to the inherent complexity of the problem that has a large

Manuscript received May 29, 2006. This work was supported in part by the U.S. National Science Foundation under Contract CCR-0305718, by Fonds National Suisse (FNS) under Grant 20021-109450/1, and by a grant from STMicroelectronics for DEIS. This paper was recommended by Associate Editor N. Chang.

S. Murali is with the Computer Systems Laboratory, Stanford University, Stanford, CA 94305 USA (e-mail: smurali@stanford.edu).

L. Benini is with the Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, 40126 Bologna, Italy, and also with Ecole Polytecnique Federale de Lausanne, 1015 Lausanne, Switzerland (e-mail: lbenini@deis.unibo.it).

G. De Micheli is with the Laboratoire des Systèmes Intégrés, Ecole Polytecnique Federale de Lausanne, 1015 Lausanne, Switzerland (e-mail: giovanni.demicheli@epfl.ch).

Digital Object Identifier 10.1109/TCAD.2006.888284

[1]More information on AMBA AXI from ARM corporation is available at http://www.arm.com/products/solutions/AMBAHomePage.html.

[2]More information on STBus from STMicrolectronics is available at http://www.st.com/stonline/prodpres/dedicate/soc/cores/stbus.htm.

[3]In the rest of this paper, we use the term power consumption to signify the peak power consumption of the system.

TABLE I
CROSSBAR PERFORMANCE AND COST FOR AN EXAMPLE IMAGE-PROCESSING MPSoC

Type	Average Latency (cycles)	Maximum Latency (cycles)	Size Ratio
shared bus	35.1	51	1
full crossbar	6	9	10.5
partial crossbar	9.9	20	4

solution space, designing an interconnection architecture for MPSoCs that minimizes the power consumption of the design while satisfying the performance constraints is a nontrivial task. In order to meet the tight time-to-market constraints and to effectively handle the design complexity, it is essential to provide a computer-aided design tool support for automating this task.

When designing the communication architecture, it is critical to consider the effect of the physical design measures, such as the wire delays in the communication architecture. In order to achieve timing closure of the design, the wiring complexity of the interconnect architecture should be considered during the communication architecture synthesis phase. Considering the wiring complexity in the early stages of the design will lead to a faster design cycle, reduction in the number of design re-spins, and faster time-to-market. Having accurate wirelength estimate during the architecture synthesis phase is also important to be able to obtain an accurate estimate of the interconnect power consumption.

The communication architecture for the design should closely match the application traffic characteristics and performance requirements. As an example, let us consider an image-processing MPSoC (detailed explanation of the MPSoC and experimental setup is presented later in Section VII) with three different communication architectures used to connect the cores, namely: 1) a shared bus, 2) a full crossbar, and 3) a partial crossbar. In Table I, we present the average and maximum latency incurred for a transaction (transfer of a single data word), obtained from SystemC simulation of the design using different communication architectures. The sizes of the crossbars (in terms of number of components used) normalized with respect to the size of the shared bus are also presented in the table. As seen from the table, as expected, both the average and the maximum transaction latencies are much higher for a single shared bus than the partial or full crossbars. However, it is interesting to note that an optimal partial crossbar gives almost the same performance as a full crossbar, although it uses fewer resources than a full crossbar.

In this paper, we target the automatic design of the "most power efficient crossbar" configuration for an MPSoC, satisfying the performance characteristics of the applications. The proposed design methodology is based on actual functional traffic analysis of the application, and the generated crossbar configuration is validated by cycle-accurate SystemC simulation of the application using that crossbar. Most previous works on bus generation and NoC topology generation (which are some what similar to crossbar generation) are based on either average communication traffic flow between various cores or statistical traffic-generating functions. While the former approaches fail to capture local variations in traffic patterns (as the average bandwidth of communication is a single metric that is calculated based on the entire simulation time), the latter approaches are only based on approximations to the functional traffic. While methodologies that target the design of NoCs are required in the long run, providing design support for the state-of-the-art crossbar-based bus designs poses an immediate and pressing problem. The crossbar-based architectures are already widely deployed in several industrial platforms [3], and a streamlined methodology to design them is still not yet fully developed.

Our design methodology differs from existing approaches (refer to Section II for a survey of existing works) in the fact that it is based on the analysis of simulated traffic patterns in windows. We divide the entire simulation period into a number of fixed-sized windows. Within each window, we guarantee that the application communication requirements (such as the bandwidth requirements) are met. We minimize the overlap among traffic streams mapped onto the same resource, thereby reducing the latency for data transfer. We also consider the criticality and real-time requirements of streams and map overlapping critical streams onto different crossbar resources.

Our methodology spans an entire design space spectrum with the analysis based on the average communication traffic (as done in many previous works [23]–[31]) and the peak bandwidth (as done in [32]), being the two extreme design points. Thus, our methodology also applies to cases where application traces are not available, and only rough estimates of the traffic flows between the various cores are known. The design point in the spectrum is varied by controlling the window size used for traffic analysis and design, which is explained further in Section VII.

We also integrate the setting up of several communication architecture parameters (such as the frequency of operation) with the crossbar synthesis phase. Unlike earlier approaches to crossbar generation (refer to Section II for details), we consider the wiring complexity of the interconnect during the communication architecture synthesis procedure. During the synthesis phase, the floorplan of the design is performed, where the accurate physical locations of the cores and the crossbar matrix are determined. From the resulting floorplan, the wirelengths in the design are obtained. Based on the length of the wires and the operating frequency of the crossbar (which is automatically tuned by the synthesis procedure), any timing violations on the wires are obtained early in the design cycle. Thus, the crossbar architecture generated by the procedure is also validated for timing correctness, which is a key step to bridge the gap between the higher-level architectural models and the actual physical design models of the crossbar architecture. From the wirelength estimates, we also obtain accurate estimates of the power consumption of the interconnect wires. The crossbar matrix power consumption values are based on the synthesis of the register transfer level (RTL) models of the design, obtained using industry standard tools. From the wire and crossbar matrix power consumption, the total communication architecture power consumption is obtained, which is used to guide the synthesis procedure to obtain the most power efficient crossbar architecture.

We present experiments on several different MPSoC designs that show large reduction in power consumption of the communication architecture (45.3% on average) and total wirelength of the crossbar buses (38.0% on average) when compared with the traditional full crossbar-based design approaches. Compared with the existing design methods, our proposed methodology results in crossbar platforms that lead to a large reduction in transaction latencies (up to 7×). Our experiments also show that the proposed approach is highly scalable to a large number of cores and to a large number of simulation windows in the design.

II. Previous Work

A component-based design methodology for SoC design is presented in [5]. The synthesis and instantiation of single bus and multiple bridged buses have been explored in many research works [16]–[19]. An approach for mapping the system's communication requirements and optimizing the protocols for a given communication architecture template is presented in [20]. In [21], the use of communication architecture tuners to adapt to the runtime variability needs of a system is presented. A floorplan-aware method for designing point-to-point links and buses is presented in [34] and [35].

The need for a scalable communication architecture has been presented in several research works [6]–[9]. A large body of research focuses on developing design tools and architectures for NoCs [11]–[14]. A detailed survey of many of the NoC research works is presented in [10].

The mapping of the communication requirements of a system onto a fixed set of NoC topologies is explored in [23]–[27]. Design methodologies are presented for application-specific bus design in [28] and for application-specific NoC topology design in [29]–[31]. These works are based on average communication transferred between various cores. In [32], designing application-specific topologies based on actual simulation traces is presented. However, the methodology is based on eliminating contention and can lead to oversizing of network components, as even a small amount of overlap between two traffic streams would result in the need for separate communication resources for them. In [22], the analysis is based on statistical traffic generators and not functional application traffic.

In [36] and [37], a methodology to design an NoC architecture that satisfies the performance constraints of multiple applications is presented. There are several differences between the methods presented there and our proposed method. First, our approach is fine tuned to target the specific problem of crossbar generation, while these earlier works target the design of general NoCs, which cannot be directly extended to address the crossbar design problem. Second, the earlier methods do not consider the wiring complexity and the physical level details, which cannot be ignored when designing crossbar-based architectures. Third, the earlier approaches do not consider several important metrics such as overlap among traffic streams, which are considered here.

In [38], the authors present an exact approach to crossbar synthesis, where they integrate the NoC architecture parameter setting with the synthesis process. Their design approach is again based on average traffic rates between different cores and does not consider the variations in traffic rates and overlap among streams. Also, the synthesis approach does not consider the physical design information for estimating the wiring complexity of the design.

In [33], we had presented a synthesis approach for designing the most cost effective crossbar architecture for MPSoCs. The objective of the design procedure was to minimize the total number of buses used in the design, satisfying the performance constraints of the application. The synthesis procedure was based on the use of the exact integer linear program (ILP) formulation of the problem, which was solved optimally.

In this paper, we significantly augment the synthesis procedure from [33], leading to several important and new contributions.

1) First, we integrate the floorplanning process with the crossbar synthesis procedure. Thus, our new synthesis approach considers the wiring complexity of the design as well.
2) Second, we modify the objective function of the design procedure so that we minimize the power consumption of the design, satisfying the performance constraints of the applications. The design objective of minimizing power consumption is more pragmatic than minimizing the total number of components as power efficiency is becoming increasingly important for most MPSoC platforms. We also integrate the setting up of communication architecture parameters in the synthesis phase.
3) Third, we present a scalable approach to crossbar matrix synthesis. The exact ILP models presented in our earlier work are applicable to small problem instances where there are a few tens of cores in the design and where a few hundred simulation windows are used for analysis. For larger designs or when more simulation windows are used for analysis, the exact formulation becomes infeasible to be applied as it does not provide a solution in reasonable time. This is due to the fact that the problem of optimal crossbar matrix synthesis is an instance of the bin-packing problem [44], which is NP-hard, and the ILP formulation has an exponential time complexity to solve a problem instance. In this paper, we present a fast and efficient heuristic approach for crossbar generation. We validate the quality of the resulting solutions with the optimal solutions from the exact ILP formulation for small problem instances.
4) Finally, in the synthesis procedure presented here, we consider the binding of cores that are masters (those that initiate bus transactions) as well as cores that are slaves (those that reply to the requests of the masters) onto the crossbar buses. In most crossbar designs (such as the AMBA AXI and STbus designs), the masters and shared slaves are usually mapped onto different crossbar resources in order to simplify the interfacing requirements.[1] We explicitly consider this information in the synthesis procedure. Considering both the masters and slaves will lead to smaller crossbar configurations when compared with the scheme where only one of them is considered.

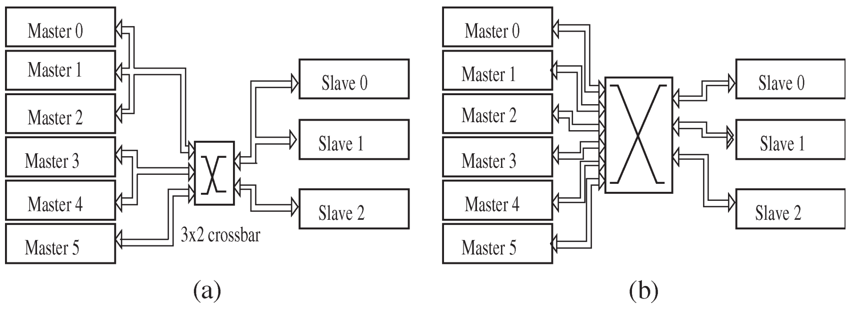

Fig. 1. STbus crossbars. (a) Partial crossbar. (b) Full crossbar.

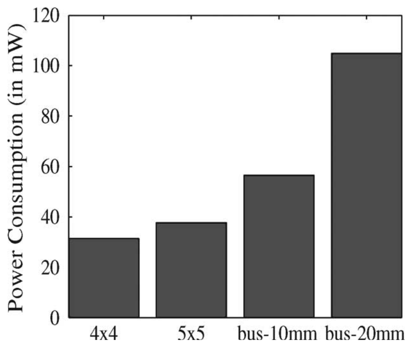

Fig. 2. Power consumption of switch matrix and wires.

Our methodology is also applicable to the multiapplication scenario presented in [36] and [37]. In this case, the simulation characteristics of each application can be treated as a separate simulation window, and the rest of the design process remains the same. As different inputs and application scenarios can lead to different communication characteristics, the entire traffic trace collected from several simulation runs under different scenarios can also be fed to the design procedure.

III. Problem Motivation and Application Traffic Analysis

A. Problem Motivation

There are three possible ways in which a crossbar can be instantiated, namely: 1) as a shared bus; 2) as a partial crossbar; or 3) as a full crossbar. The partial and full crossbars are actually composed of many buses to which the processor/memory cores are connected. Examples of partial and full crossbars are presented in Fig. 1. In the partial crossbar architecture, some of the cores (such as the Master 0 and Master 1) share the same bus, while in the full crossbar, each core is connected to a separate bus. The objective of the crossbar synthesis procedure is to obtain an efficient clustering of the master and slave cores onto the crossbar buses such that a communication architecture with low power consumption is obtained.

When choosing the most power efficient crossbar configuration, it is also important to account for the wiring complexity of the different configurations. As an example, the power consumption of the crossbar components (switch matrix and arbiters) for two different configurations and the power consumption of the wires for two different total wirelengths (assuming a design with 30 cores and data width of 32 bits for the crossbar buses) are presented in Fig. 2. For most MPSoC designs, the total length of the wires of the crossbar buses is of the order of a few tens of millimeters (refer to Section VII-B). For the power consumption values presented in the figure, we assume a 130-nm process technology, an operating frequency of 500 MHz, and an operating voltage of 1.2 V. The methods and assumptions used for estimating the power consumption of the crossbar matrices and wires are presented in detail in Section VII. From the figure, we can infer that the wire power consumption is a significant fraction of the total communication architecture power consumption for crossbar-based systems. Thus, it is important to consider the length of wires during the synthesis process, as the design point can be far from the optimum design point if such information is not accounted for. In order to have accurate wirelength estimates, we need to have accurate floorplan information of the design.

Another point worth noting is that, in many crossbar architectures, the underlying protocol may not support pipelining of the buses (for example, the Type 1 protocol of STbus).[2] In this case, the frequency of operation of the communication architecture is limited by the length of the longest bus in the design. For a chosen frequency point, it is then important to evaluate whether the length of the wires is lower than the threshold limit so that they can be traversed in one clock cycle. We would also require the accurate floorplan and wirelength estimates to apply such feasibility checks.

B. Application Traffic Analysis

In this subsection, we explore the traffic characteristics of applications to model the performance constraints to be satisfied by the crossbar designed for the system. As an example, we consider the 21-core image-processing application, as shown in Fig. 3(a). In this example, there are nine ARM cores, 11 on-chip memories, with some of the memories used for interprocessor communication, and an interrupt device. The ARM cores act as masters, and the memory cores act as slaves. The ARM cores run a set of image-processing benchmarks that involve accesses to different memories. We performed a cycle-accurate simulation of the system with a full crossbar design using the STbus crossbar architecture. A small trace of the traffic to three of the cores is shown in Fig. 3(b).

Although the aggregate traffic (measured over the entire simulation period) to the three cores is lower than that can be supported by a single bus, using a single bus to connect all three cores will lead to high average and peak latency due to overlap in traffic patterns during some regions of the simulation.

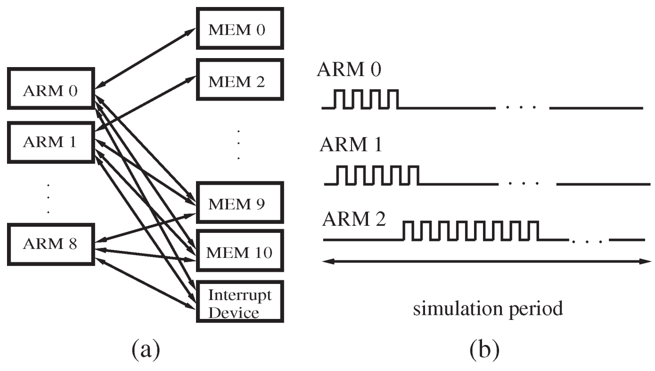

Fig. 3. Application traffic analysis. (a) Application. (b) Traffic trace.

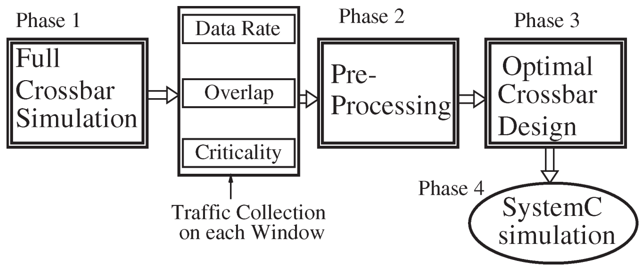

Fig. 4. Crossbar design methodology.

Another related point is that if overlaps are not considered, connecting ARM 0 and ARM 1 on to the same bus is better than connecting ARM 0 and ARM 2 onto the same bus, as the former results in lower bandwidth needs. However, the latter solution will result in better performance (reduced transaction latency) while still satisfying the bandwidth needs. Note that using peak bandwidth instead of the average bandwidth will solve this problem but will lead to an overdesign of the crossbar (in terms of number of buses needed or their frequency of operation). The design methodology needs to consider overlap among various traffic streams and should consider local variations in traffic rates. Also, some of the traffic streams can be critical, and to facilitate providing real-time guarantees, real-time traffic streams that overlap in time should not be mapped onto the same crossbar bus.

IV. Design Methodology

The design flow for the crossbar design is shown in Fig. 4, which consists of four distinct phases. In the first phase, the application is initially designed using a full crossbar communication architecture, and a SystemC simulation of the design is carried out. As the full crossbar architecture is nonblocking in nature (no contention between the cores if they are accessing different cores), it helps in modeling the application traffic requirements under ideal operating conditions. For the simulations, we use the MPARM simulation environment [40] that allows interconnection of ARM cores to several interconnection platforms (such as AMBA, STbus, ...) and to perform cycle-accurate simulations for a variety of benchmark applications.

To effectively capture local variations in traffic patterns and to perform overlap calculations, we define a window-based traffic analysis. The entire simulation period is divided into a

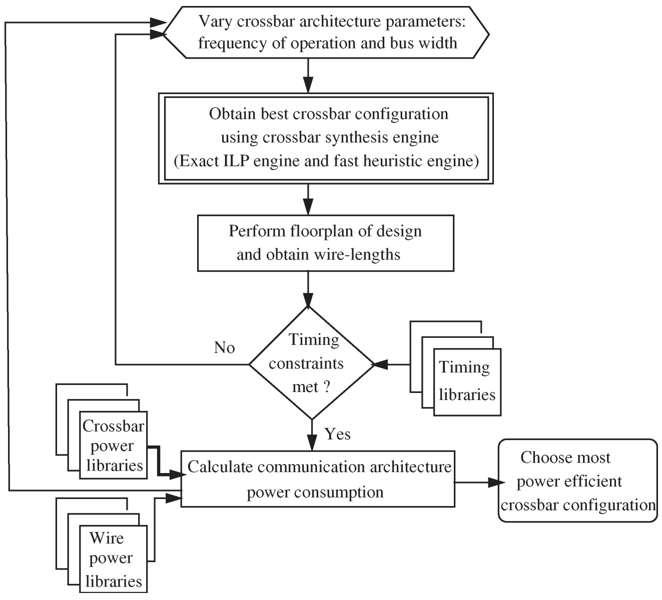
Vary crossbar architecture parameters: frequency of operation and bus width
Obtain best crossbar configuration using crossbar synthesis engine (Exact ILP engine and fast heuristic engine)
Perform floorplan of design and obtain wire-lengths
No
Timing constraints met ?
Timing libraries
Yes
Crossbar power libraries
Calculate communication architecture power consumption
Wire power libraries
Choose most power efficient crossbar configuration

Fig. 5. Crossbar synthesis phase.

number of windows, and the traffic characteristics to the various cores in each window are obtained. The traffic characteristics recorded include the amount of data sent and received by each core in every window, the amount of pair-wise overlap between the traffic streams between different cores in every window, the real-time requirements of traffic streams, etc. Without loss of generality, in the rest of this paper, we assume that all the windows are of equal size, although the methodology also applies to windows with varying sizes. The size of the window is parameterizable and depends on the application characteristics and performance requirements.

After the data collection phase, a preprocessing phase is carried out in which cores that have traffic flows with large overlaps in any window and need to be put on different buses are identified. In this phase, the overlapping critical streams that need to be on separate buses are also identified.

In the next phase, the optimal crossbar configuration for the application, satisfying the performance constraints, is synthesized. To generate the optimal crossbar configuration, we use the traffic information collected in each window and check whether the bandwidth, overlap, and criticality constraints are satisfied in each window. In the final phase, the designed crossbar matrix is instantiated in MPARM environment, and SystemC simulations are carried out.

The details of the crossbar synthesis phase are presented in Fig. 5. In the outer loop of the synthesis process, the communication architectural parameters (such as the frequency of operation and bus width) are varied in several user-defined steps. The interesting range for the parameters is obtained from the user. For each architectural parameter point, the most power efficient crossbar configuration is synthesized. For synthesis, we present two approaches: one approach is based on solving the problem exactly using ILP formulation, which is applicable for small problem instances, and the other is a more scalable approach based on fast and efficient heuristics. In the next step of the synthesis phase, we perform a floorplan of the synthesized design. Floorplanning is the process of determining the exact 2-D positions of the different cores and the switch matrix in the design. For obtaining the floorplan, we use Parquet [39], a fast and accurate floorplanner that minimizes the design area as well as the average wirelength. As the cores in MPSoC are usually predesigned hardware blocks, we realistically assume that the size of the cores (either the width and height or the aspect ratio and area) is provided as an input to the synthesis process.

From the floorplan of the design, the length of the wires (based on the Manhattan distance) and hence the power consumption on the wires are obtained. In the next step, for the chosen frequency point, the wirelengths are checked to see whether the maximum wirelength exceeds the length that the data can traverse in a single clock cycle. In the next step, from the switch matrix power consumption and the wire power consumption, the power consumption of the synthesized communication architecture is obtained. From the set of generated crossbar architectures for each architectural design point, the most power efficient architecture that satisfies the performance and timing constraints is chosen.

V. Exact Approach to Crossbar Synthesis

In this section, we formulate the mathematical models of the crossbar design problem and present the exact ILP formulation to synthesize the most efficient architecture for a chosen architectural parameter design point.

A. Problem Formulation

Definition 1: The set of all cores in the design is represented by the set T. The set of all windows used for traffic analysis is represented by the set W, with the bandwidth available (product of frequency of operation and bus width) in each window represented by WS. The set of buses used in the crossbar is represented by the set B.

Definition 2: The bandwidth requirement of each core t_i $\forall i \in 1 \cdots |T|$ in every window m $\forall m \in 1 \cdots |W|$ is represented by $comm_{i,m}$.[4] The amount of data overlap between every pair of cores (t_i, t_j) in each window m is represented by $wo_{i,j,m}$.

The overlap between every pair of cores t_i and t_j over the entire simulation period is obtained by summing the overlap between them in all the windows and represented by the entries of the overlap matrix OM as

$$om_{i,j} = \sum_m wo_{i,j,m} \; \forall i, j. \tag{1}$$

In the preprocessing phase of the design flow (refer to Fig. 4), those pairs of cores that have overlap exceeding the threshold value (which is parameterizable) in any window are identified. By mapping the traffic flows of such cores onto separate buses, the maximum and average latency of data transmission can be reduced and, in some cases, can also speed up the process of finding the optimal crossbar configuration. Also in this preprocessing step, the real-time traffic streams that overlap with each other in any window are identified. Such cores with overlapping real-time streams should not be placed on the same bus as real-time communication guarantee to the streams cannot be given in this case. Also, as noted earlier, most crossbar architectures do not allow masters and shared slaves of the design to be mapped onto the same bus. We define the set of all cores that cannot be on the same bus by the conflict matrix

$$c_{i,j} = \begin{cases} 1, & \text{if } t_i \text{ and } t_j \text{ should be on different buses} \\ 0, & \text{otherwise} \end{cases} \quad \forall i, j. \tag{2}$$

We model the performance constraints that need to be satisfied by the crossbar configuration in each window as constraints of an ILP.

Definition 3: The set X represents the set of binding variables $x_{i,k}$ such that $x_{i,k}$ is 1 when core t_i is connected to the bus b_k and 0 otherwise.

In the crossbar design, each core has to be connected to a single bus (while a single bus can connect multiple cores). This is implemented by the constraint

$$\sum_k x_{i,k} = 1 \; \forall i. \tag{3}$$

In every window used for traffic analysis, the individual buses of the crossbar have to support the traffic through them in that window. By evaluating the bandwidth constraints over a smaller sample space of a window (which is typically a few hundred or thousand cycles) instead of the entire simulation sample space (which can be millions of cycles), we are better able to track the local variations in the traffic characteristics.

This window-based bandwidth constraint is represented by

$$\sum_i comm_{i,m} \times x_{i,k} \leq WS \; \forall k, m. \tag{4}$$

Definition 4: The set SB represents the set of sharing variables $sb_{i,j,k}$ such that $sb_{i,j,k}$ is 1 when cores t_i and t_j share the same bus b_k and 0 otherwise. The set S represents the set of sharing variables $s_{i,j}$ such that $s_{i,j}$ is 1 when cores t_i and t_j share any of the buses of the crossbar and 0 otherwise.

$sb_{i,j,k}$ can be computed as the product of $x_{i,k}$ and $x_{j,k}$. However, this results in nonlinear (quadratic) equality constraints. To break the quadratic equalities into linear inequalities, we use

$$\begin{aligned} & sb_{i,j,k} \in \{0, 1\} \\ & x_{i,k} + x_{j,k} - 1 \leq sb_{i,j,k} \\ & 0.5x_{i,k} + 0.5x_{j,k} \geq sb_{i,j,k} \; \forall i, j, k \end{aligned} \tag{5}$$

and $s_{i,j}$ is computed using

$$s_{i,j} = \sum_k sb_{i,j,k} \; \forall i, j. \tag{6}$$

The condition that certain cores are forbidden to be on the same bus, obtained from (2), is represented by

$$c_{i,j} \times s_{i,j} = 0 \; \forall i, j. \tag{7}$$

[4]In the rest of this paper, we follow the convention that variables i and j are defined for $1 \cdots |T|$, variable k is defined for $1 \cdots |B|$, and m for $1 \cdots |W|$.

The fact that all the integer variables introduced above take values of either 0 or 1 only is represented by

$$x_{i,k}, s_{i,j}, c_{i,j} \in \{0, 1\}\ \forall i, j, k. \tag{8}$$

B. Exact Crossbar Synthesis Algorithm

The exact algorithm for the crossbar design has two major steps. The first is to find the best crossbar configuration that satisfies the performance constraints (which were presented in the above subsection). The second step is to find the optimal binding of the cores to the chosen crossbar configuration.

In order to find the best crossbar configuration, we vary the number of buses in the design, from the maximum number (equal to the number of cores in the design, modeling a full crossbar) to one (modeling a single shared bus), in a binary search manner. For each configuration of bus count, we check whether a feasible solution that satisfies the constraints of the ILP [formed by the set of inequalities from (3)–(8)] exists. Once the minimum number of buses has been identified from applying the ILP, possibly multiple times, we separate the buses used by the masters and slaves of the design, thereby generating the optimal crossbar configuration.

Once the best crossbar configuration is obtained, in the next step, the optimal binding of the cores onto buses of the crossbar is obtained. A binding of cores to the buses that minimizes the amount of overlap of traffic on each bus will result in lower average and peak latency for data transfer.

For this, the above ILP is solved with the objective of reducing the maximum overlap on each of the bus and satisfying the performance constraints as

$$\begin{aligned} &\min : maxov \\ &\text{s.t.} \sum_i \sum_j om_{i,j} \times sb_{i,j,k} \leq\ maxov\ \forall k \end{aligned} \tag{9}$$

and subject to (3)–(8).

By splitting the problem into two ILPs, we speed up the execution time of the algorithm as solving ILP 1 for feasibility check is usually faster than solving ILP 2 with the objective function and additional constraints. The ILPs are solved using the CPLEX package [42].

VI. Heuristic Approach to Crossbar Synthesis

As the exact ILP approach is not scalable to large problem instances, either when the number of cores in the design is large or when the number of simulation windows used for the analysis is large, in this section, we present a fast and efficient heuristic approach for crossbar synthesis.

The problem of assigning cores to the minimum number of buses, subject to the performance constraints, is a special instance of the general problem of "constrained bin packing" [43]. There are several efficient heuristics that have been developed for the bin-packing problem [43]. In this paper, we use an approach that is based on the "first-fit" heuristic to bin packing. We chose this heuristic for several reasons. When the performance constraints are removed, the heuristic procedure is theoretically guaranteed to provide solutions that are within two times the optimum solution that would be obtained by an exact algorithm [43]. Practically, we found that the solutions obtained by the heuristic are close to the optimum solution possible for experiments on several SoC benchmarks. Moreover, the heuristics are relatively simple to implement and have a very low run time complexity, making the approach scalable to large designs and allowing the use of a large number of simulation windows for analysis.

The heuristic algorithm for crossbar synthesis is presented in Algorithm 1. In the first step of the algorithm, the bandwidth available in each simulation window is calculated. In the next step, all the cores are initialized as unmapped, as they are yet to be mapped onto buses. Then, the number of buses in the crossbar is initialized to zero (step 5). In steps 6 to 25, the assignment of the cores onto the buses of the crossbar is performed. The basic approach used is the following: We try to map as many cores as possible onto a single bus. While mapping the cores, from the set of all cores that satisfy the bandwidth and conflict constraints, we choose the one that minimizes the pair-wise traffic overlap with the cores that have been already mapped onto the current bus. When no more cores can be assigned to the current bus, either because the bandwidth of the bus in any of the simulation window has been saturated or because of conflicts with the cores already mapped onto the bus, a new bus is instantiated. The process is repeated until all the cores in the design have been mapped onto a bus.

```
Algorithm 1 Heuristic-synthesis($frequency$, $buswidth$)
 1: Bandwidth available in each window, $WS = frequency \times buswidth$
 2: for $i = 1 \cdots |T|$ do
 3:   mapped($i$) = false
 4: end for
 5: Initialize number of buses used, $k$ to 0.
 6: while $\exists i \in 1 \cdots |T|$, such that mapped($i$) = false do
 7:   Increment the bus count $k$ by 1 and instantiate new
      bus. Initialize bandwidth available on bus on all windows: $BW(k,m) = WS, \forall m \in 1 \cdots |W|$
 8:   Choose unmapped core $i$, $\forall i \in 1 \cdots |T|$, with maximum bandwidth requirements on any window and
      map it onto bus $k$.
 9:   Initialize the set $chosen\_set$ to $\phi$
10:   for $i = 1 \cdots |T|$ do
11:     if mapped($i$) = false and core $i$ does not have conflicts with cores already mapped onto bus $k$ then
12:       $bw\_satisfied$ = true
13:       for $m = 1 \cdots |W|$ do
14:         if $BW(k,m) < comm_{i,m}$ then
15:           $bw\_satisfied$ = false
16:         end if
17:       end for
18:       if $bw\_satisfied$ = true then
19:         $chosen\_set = chosen\_set \bigcup i$
20:       end if
21:     end if
22:   end for
```

TABLE II
COMMUNICATION REQUIREMENTS OF EXAMPLE SYSTEM

Name	Type	Bandwidth (window 1) MB/s	Bandwidth (window 2) MB/s
core_0	Master	300	180
core_1	Master	200	270
core_2	Master	80	210
core_3	Slave	60	110
core_4	Slave	150	70

TABLE III
AMOUNT OF TRAFFIC OVERLAP BETWEEN CORES (IN MEGABYTES PER SECOND) OF EXAMPLE SYSTEM

	core_0	core_1	core_2	core_3	core_4
core_0	x	30	10	x	x
core_1	30	x	27	x	x
core_2	10	27	x	x	x
core_3	x	x	x	x	15
core_4	x	x	x	15	x

```
23:   Choose core $i, \forall i \in 1 \cdots |chosen\_set|$, with minimum
      overlap with cores mapped onto bus $k$ and map it to bus
      $k$. Update available bus bandwidth as: $BW(k,m) =$
      $BW(k,m) - comm_{i,m}, \forall m \in 1 \cdots |W|$.
24:   Repeat steps 9–23 until $chosen\_set$ is empty.
25: **end while**
26: Separate the buses onto which masters and slaves are
    mapped and generate the crossbar configuration.
```

From the resulting number of buses, we separate those buses onto which masters are attached and those onto which slaves are attached. From this, the efficient crossbar configuration for the design is obtained.

Example 1: Let us consider a small example with five cores, with three of them being masters and the rest being slaves. For illustrative purposes, let us assume that two simulation windows are used for analysis (although in real systems usually several thousand windows are used). The communication traffic rates for each of the cores (in megabytes per second) for the two simulation windows are presented in Table II, and the amount of traffic overlap between the different cores over all the windows is presented in Table III. Let us assume that the current frequency design point is 100 MHz and the bus width is 32 bits, which are automatically tuned by the crossbar synthesis procedure (as presented in Fig. 5). In the first step of the heuristic algorithm, the bandwidth of the bus in each simulation window is calculated to be 400 MB/s (frequency $\times$ data width). Initially, a single bus is instantiated, and core_0 is chosen to be mapped onto the bus, as it has the maximum bandwidth requirements of the different cores, across all the simulation windows (see Fig. 6).

Then, from the set of all cores, those cores that satisfy the bandwidth and conflict constraints are chosen. As cores that are masters and slaves are not allowed to be mapped onto the same bus (specified as part of the conflict constraints), the sets of assignable cores to the bus are core_1 and core_2. From these two, core_2 is chosen as it has the minimum overlap with the cores already mapped onto the bus (i.e., with core_0) and assigned onto this bus (Fig. 7). When no more cores can be assigned to the current bus, a new bus is instantiated. The different steps of the procedure for the five-core example are presented in Figs. 6–11. At the end of the procedure, those buses that are used by the masters and those that are used by the slaves are separated, which gives the best crossbar configuration. In this example, we have two buses used by the masters and one used by the slaves, resulting in a 2$\times$ 1 crossbar design, as shown in Fig. 11.

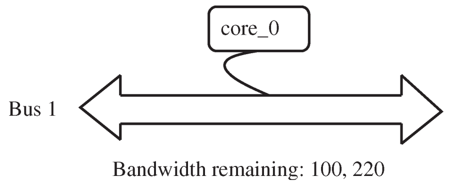

Fig. 6. Step 1. The core_0 is chosen and mapped onto the bus. The bandwidth remaining in each of the simulation windows (in megabytes per second) after mapping core_0 is also presented.

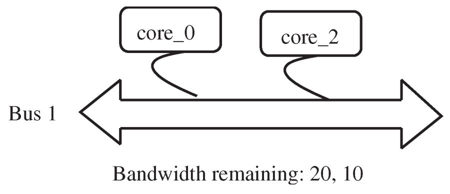

Fig. 7. Step 2. From the remaining cores, core_2 is chosen and mapped.

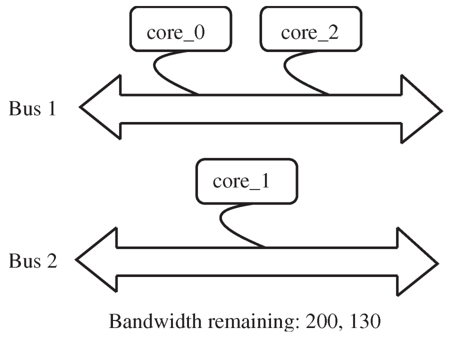

Fig. 8. Step 3. A new bus is instantiated, and core_1 is mapped.

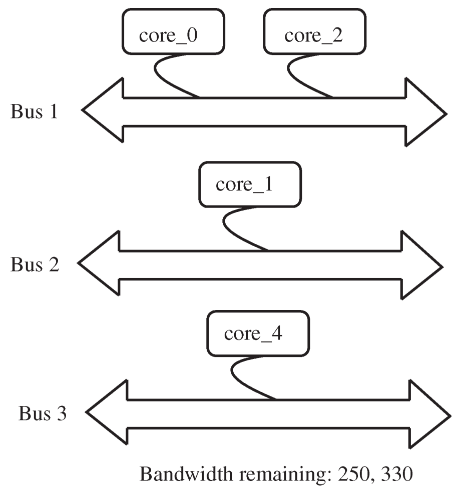

Fig. 9. Step 4. Another bus is instantiated, and core_5 is mapped.

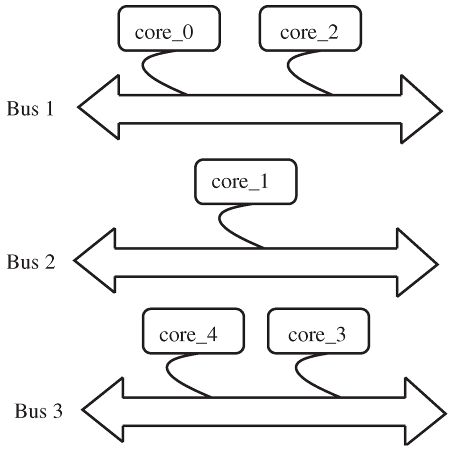

Fig. 10. Step 5. The remaining core core_4 is mapped.

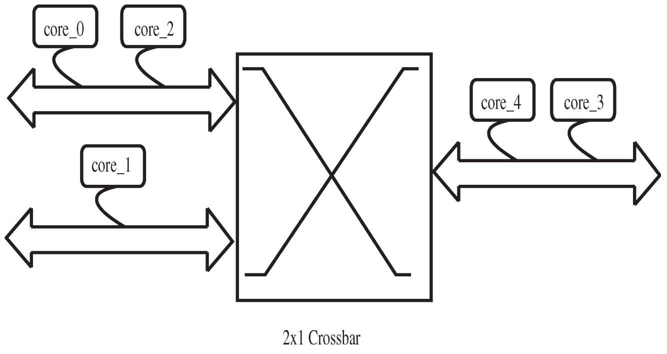

Fig. 11. Step 6. The crossbar configuration (2× 1) is obtained from the instantiated buses.

VII. Experiments and Case Studies

In this section, we present the experimental case studies performed to validate the proposed crossbar design methodology.

A. Experimental Platform and Power Models

For performing the SystemC simulations on MPSoC benchmarks, we use the MPARM simulation platform [40]. The platform is representative of a large class of multiprocessor SoC platforms and consists of a configurable number of 32-bit ARM processors, memory cores, hardware devices or traffic generators, and a hardware interrupt unit. The platform allows the use of different interconnect architectures such as AMBA and STbus to interconnect various hardware cores. It also supports a variety of MPSoC benchmarks that have been efficiently parallelized to run on ARM cores.

For power consumption estimations of the switch matrix, we implemented several configurations of the AMBA multilayer crossbar, varying the number of input and output ports of the matrix. The different configurations were implemented using the AMBA DesignWare libraries obtained from the Synopsys CoreAssembler tool [45]. The tool generates RTL codes of the different configurations, which were then synthesized using Synopsys Design Compiler [45]. For synthesis, we utilize a 130-nm process technology, an operating voltage of 1.2 V, and an operating frequency of 500 MHz. Based on the power consumption values obtained from the synthesis process, we built

TABLE IV
Traffic Characteristics of IMP2

Core	win. 1 (MB/s)	win. 2 (MB/s)
ARM 0	810	210
ARM 1	740	234
MEM 0	790	150
MEM 1	730	220
MEM 9	180	50
MEM 10	180	50

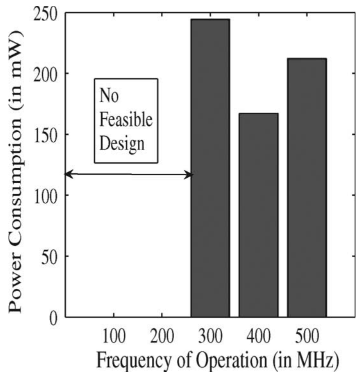

Fig. 12. Power consumption for different crossbar frequencies.

analytical models for the switch matrix power consumption using linear regression. During the crossbar design process, the power numbers from the analytical models are linearly scaled based on the crossbar operating frequency (which is automatically tuned by the design process). We estimate the wiring capacitance and wire power consumption based on the models from [41]. The power consumption values of some of the crossbar components were presented earlier in Fig. 2.

B. Application Benchmark Analysis

We apply our crossbar design methodology on several SoC designs implemented using the MPARM platform: IMage Processing design 1 (IMP1-25 cores), IMage Processing design 2 (IMP2-21 cores), fast Fourier transform (FFT)-based SoC (FFT-29 cores), Data Processing SoC (DP-15 cores), and SoC implementing a DES encryption system (DES-19 cores). The traffic characteristics of the applications were scaled to project the traffic requirements of future MPSoCs, as presented in [12]. For traffic analysis, we use 1000 simulation windows for the different designs, with each simulation window accounting for a few hundred simulation cycles. The tuning of the window size (and hence the number of windows) is further explored in Section VII-D.

In our methodology, the interesting ranges of operating frequencies and bus data widths are obtained as inputs from the designer. Practically, the data width of the bus is set up based on the data widths of the different processors in the design. In our SoC designs, all the processors have the same data width (of 32 bits), and hence we feed this value as input to the synthesis engine. We define the interesting range of operating frequencies to be between 100 and 500 MHz, with each frequency point being a multiple of 100 MHz. With this set up,

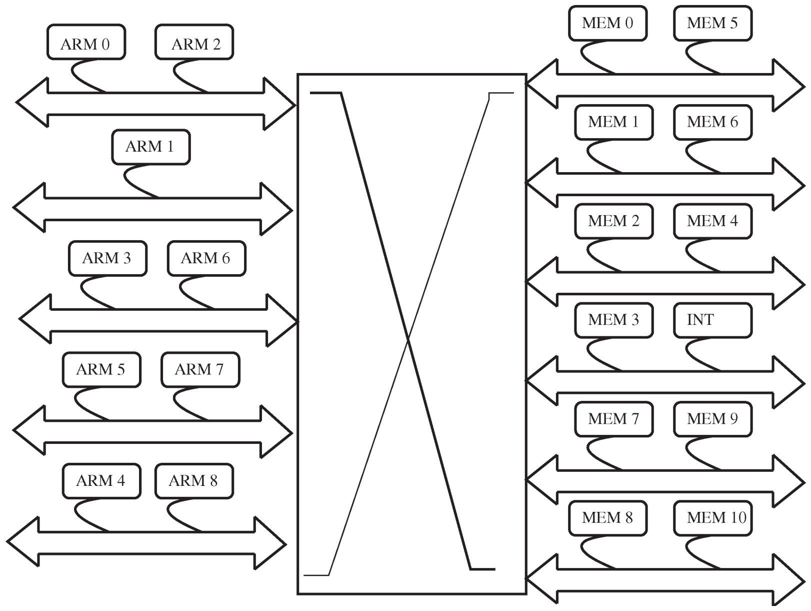

Fig. 13. Synthesized crossbar for the IMP2 SoC design.

we apply our heuristic synthesis engine to design the crossbar architecture for the designs.

We first briefly analyze the quality of the crossbar design obtained for the IMP2 SoC design. The communication between the cores of the IMP2 design was presented earlier in Fig. 3(a). The communication requirements of some of the cores for the first few simulation windows are presented in Table IV. In this benchmark, there are nine ARM cores (ARM 0 to ARM 8), 11 memory cores (MEM 0 to MEM 10), and an interrupt device (INT). The ARM cores act as masters, and the others are slave cores that respond to the requests of the masters. There is substantial temporal overlap between traffic flows from the various ARM cores to the memories as the ARM cores perform similar computations and thus access their memories at almost the same time. The power consumption of the synthesized crossbar designs for the different frequency design points is plotted in Fig. 12. As the maximum bandwidth requirements of most of the cores were above 800 MB/s, the minimum frequency design point that gives a feasible solution is 300 MHz (at 200 MHz, the available bus bandwidth of 800 MB/s cannot support the requirements of most cores). At lower operating frequencies (such as 300 MHz), a larger crossbar configuration is required to satisfy the bandwidth constraints. A larger crossbar configuration usually also leads to an increased wiring complexity. These two factors coupled together result in larger power consumption for the communication architecture. At very high operating frequencies, the power consumption of the communication architecture is higher as the power consumption increases linearly with the operating frequency of the system. For the IMP2 design, the crossbar architecture with the lowest power consumption is obtained at 400 MHz.

The synthesized crossbar architecture (a 5× 6 crossbar) for the IMP2 design is presented in Fig. 13. In order to satisfy the window bandwidth constraints, only few of the cores can share a single bus, and thus each of the buses used in the crossbar has at most two cores attached to them. The bindings are such that the cores with highly overlapping streams are placed on different buses. As a result, the designed crossbar has acceptable performance (in terms of average and maximum latency constraints) with 1.9× reduction in the number of buses used when compared to a full crossbar. The floorplan of the IMP2 SoC with the designed crossbar, as obtained from the Parquet floorplanner, is presented in Fig. 14.

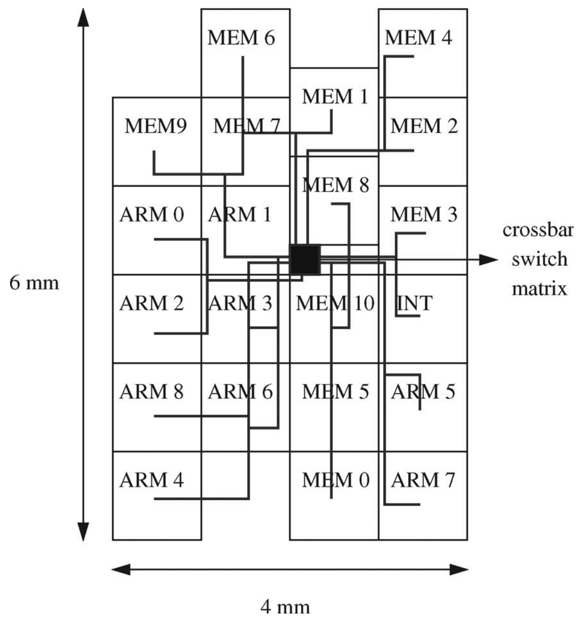

Fig. 14. Generated floorplan and buses for IMP2 SoC design.

The size and power consumption of the synthesized crossbar architectures for the different SoC designs and for full crossbar configurations are reported in Table V. The power consumption of both the switch matrix and the crossbar bus wires is shown in the table. Our methodology results in a large reduction in crossbar architecture power consumption (45.3% on average) when

TABLE V
CROSSBAR SIZE AND POWER CONSUMPTION FOR SOC DESIGNS

Design	Full Cross -bar Size	Synthesized Cross -bar Size	Full Crossbar Power Consumption (mW)			Synthesized Crossbar Power Consumption (mW)		
			matrix	wire	total	matrix	wire	total
IMP1	11x14	6x7	156.7	228.0	384.74	60.2	146.1	206.3
IMP2	9x12	5x6	128.4	198.2	326.6	45.2	125.0	170.2
FFT	13x16	7x8	175.1	301.4	476.5	75.9	191.8	276.7
DP	6x9	3x5	38.7	51.3	90.0	12.1	36.9	49.0
DES	8x11	4x6	56.0	82.1	138.1	18.8	54.1	72.9

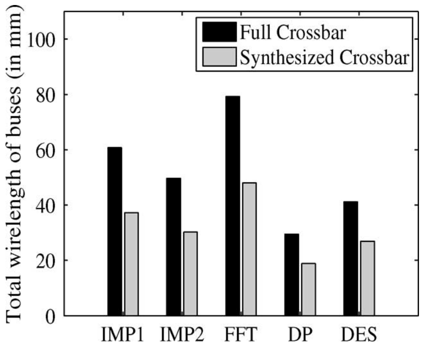

Fig. 15. Average wirelength of the crossbar buses for the designs.

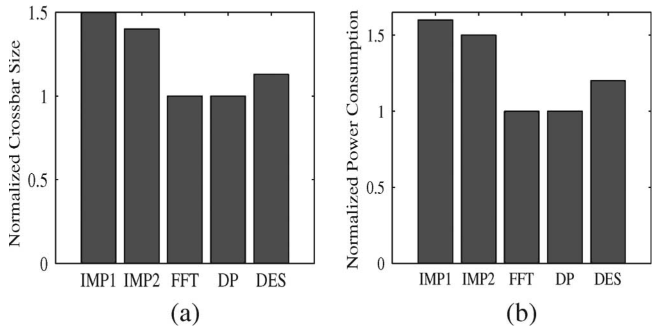

Fig. 17. Comparisons of heuristic engine versus exact engine. (a) Crossbar size. (b) Crossbar power consumption.

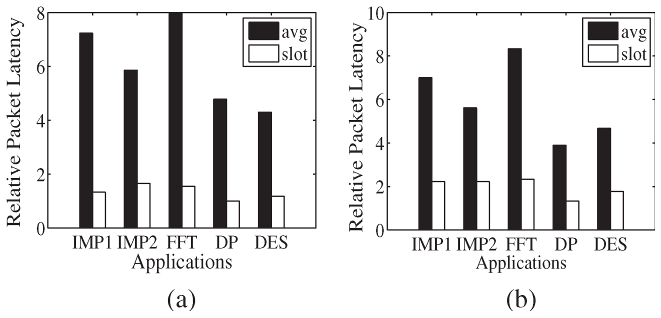

Fig. 16. Application relative latencies. (a) Average latency. (b) Maximum latency.

TABLE VI
HEURISTIC PROCEDURE RUN TIME FOR FFT DESIGN

Number of Windows	Run-time (in s)
1000	4.85s
10000	5.46
50000	8.31
100000	11.91
500000	41.4

compared with traditional full crossbar-based systems. The synthesized crossbar configurations also lead to a large reduction in the total length of the buses used in the design (38.0% on average, refer to Fig. 15), as there are fewer buses in the design. Reducing the wiring congestion is essential to having a faster physical design process and to achieve faster design closure.

The normalized average and maximum read/write transaction latencies (to read or write one data word) for the designs obtained using the methodology based on average traffic flows and using our proposed methodology (referred to as "slot" in the figures, signifying the use of our proposed slot or window-based methodology) are presented in Fig. 16(a) and (b). As seen from the figures, the latencies incurred by crossbar designs based on average traffic flows are 4× to 7× higher than the crossbars designed using our scheme. Also, the latencies incurred in the designs generated by our scheme are within acceptable bounds from the minimum possible latencies (of a full crossbar). Moreover, depending on the design objective, crossbar size–performance tradeoffs can be explored in our approach by tuning the analysis parameters (such as the window size, overlap threshold, etc.), as explained in further subsections.

C. Comparisons of Heuristic Engine With the Exact Engine

In this subsection, we explore the quality of the solutions produced by the heuristic engine with respect to the exact ILP engine. As the exact engine takes several hours to compute solutions for designs with more than a few hundred windows, we reduced the number of windows to 100 for the designs and applied the two engines for the SoC designs. The size (total number of buses) of the crossbar synthesized by the heuristic engine normalized with respect to the size of the crossbar synthesized by the exact engine for the different designs is presented in Fig. 17(a). The normalized power consumption of the synthesized crossbar designs for the different SoC designs is presented in Fig. 17(b). Compared with the exact solutions, the solutions obtained by the heuristic engine incur only a modest increase in crossbar size (1.21× on average) and power consumption (1.26× on average).

The total run time of the heuristic engine (including the time for performing floorplanning) for the biggest SoC design (the FFT SoC) for different numbers of window sizes is presented in Table VI. The experiments were performed on a Linux workstation with 3.2-GHz processor and 4-GB RAM. The run time also includes the time to perform the sweep over the architectural parameters (frequency of operation and bus width)

TABLE VII
RUN TIME FOR DIFFERENT NUMBER OF CORES

Number of cores	Run-time (in s)
29	41.4s
40	67.22
50	96.73
60	130.03

of the crossbar design. As seen from the table, the algorithms have a very low run time complexity even for large designs and when a large number of windows are used for analysis. On the other hand, the exact ILP procedure did not produce results in reasonable time for the design when more than a few hundred windows were used for analysis. To show the scalability of the heuristic procedure with the number of cores in the design, we produced synthetic benchmarks based on scaled versions of the FFT SoC. The execution times of the engine for the different benchmarks (with the number of windows set to 500 000) are presented in Table VII. From the table, we see that even for a very large design (60 cores with 500 000 windows used for analysis), the heuristic process completes in a few minutes, thereby showing the scalability of the procedure.

D. Window Sizing

The size of the window used during the design process is an important parameter that determines the efficiency of the design methodology to capture the application performance parameters. A small window size results in a much finer control of the application performance parameters, and the resulting crossbars have lower latencies. However, a very small window size will lead to an overdesign of the network components. On the other hand, a large window size results in lesser control over the performance parameters of the application but results in a more conservative design approach where higher transaction latencies can be tolerated.

To illustrate these effects, we applied our design methodology with different window sizes for a synthetic benchmark with 20 cores. Please note that we use a synthetic benchmark for this experiment (instead of real SoC designs) so that we can vary the burst sizes (we refer to a burst as a stream of words generated by the same core) in the application to study its impact on the crossbar synthesis process. The typical burst sizes for the benchmark are initially set to 100 cycles. When the window size is much smaller than the burst size, the size of the crossbar generated is very close to that of a full crossbar (refer Fig. 18). When the window size is around a few times that of the burst size (from one to four times), the synthesized crossbar has a much smaller size (typically around 25%) and acceptable latencies (around 1.5×) of that of a full crossbar. For aggressive designs, the window size can be set closer to the burst size, and for conservative designs (where larger transaction latencies can be tolerated), the window size can be set to a few times the typical burst size. The acceptable window sizes for various burst sizes is presented in Fig. 19. It can be seen from the plot that window size varies almost linearly with burst size, consolidating the above arguments.

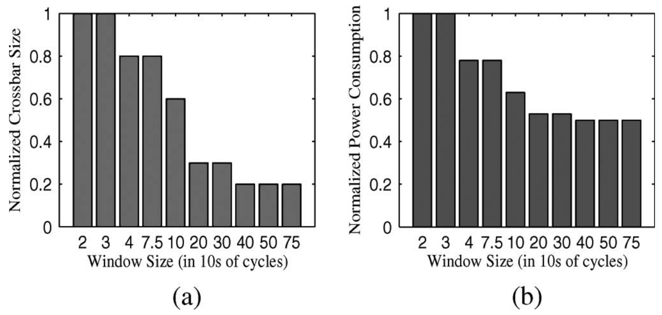

Fig. 18. Effect of window size on crossbar size and power consumption. (a) Crossbar size. (b) Crossbar power consumption.

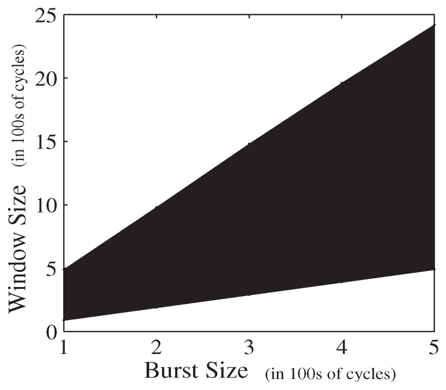

Fig. 19. Burst versus window size.

E. Real-Time Streams and Effect of Binding

In each simulation window, the critical traffic streams that require real-time guarantees are recorded. During the preprocessing step of the design flow (refer to Fig. 4), the real-time traffic streams that overlap with each other in any window are identified. In order to provide real-time guarantees to such streams, in our methodology, the cores with critical streams that have temporal overlap are placed onto separate buses of the crossbar. Experimental results on the benchmark applications show a very low transaction latency (almost equal to the latency of perfect communication using a full crossbar) for such streams. Please note that in order to provide hard real-time guarantees, the underlying crossbar architecture should also provide support for having priorities for the different traffic streams so that real-time streams are given higher priorities over other streams. In many crossbar architectures, such as the STbus, such support is provided in the crossbar architecture by utilizing priority-based arbitration mechanisms.

After finding the best crossbar configuration, we do an optimal binding of the cores onto the buses of the crossbar, minimizing the total overlap on each bus. By minimizing the overlap on each bus, the transaction latencies reduce significantly. To illustrate this effect, we compare the crossbars designed using our approach with two binding schemes: random binding of cores onto the buses, satisfying the design constraints [(3)–(8)], and optimal binding that minimizes overlap on each bus, satisfying the design constraints. The average latency incurred by the random binding scheme for the benchmark applications was on average 2.1× higher than that incurred by the optimal binding scheme.

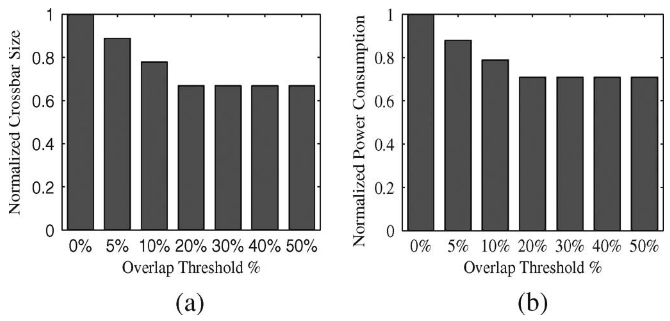

Fig. 20. Effect of overlap threshold parameter. (a) Crossbar size. (b) Crossbar power consumption.

F. Overlap Threshold Setting

By varying the two parameters, i.e., window size and overlap threshold, the crossbar can be designed such that the average and the maximum transaction latencies incurred in the design are acceptable. The effects of the overlap threshold parameter on the size and power consumption of the crossbar generated for the synthetic benchmark are presented in Fig. 20(a) and (b). The crossbar size and power numbers are normalized with respect to the case when the overlap threshold is set to 0%, which leads to a full crossbar configuration (as no two cores can share a bus in this case). The plots end at 50% overlap between cores because, if the pair-wise overlap between two cores exceeds 50% of the window size (in any of the windows), then the window bandwidth constraints cannot be satisfied. So, the maximum value of the overlap parameter can be set at 50% of the window size. This will also speed up the process of finding the best crossbar configuration as such overlapping cores will be identified in the preprocessing phase (refer to Fig. 4) and will be forbidden to be on the same bus of the crossbar. From experiments, we found that for aggressive designs (where there are tight requirements on the maximum latencies), the threshold can be set to around 10%, and for conservative designs, the threshold can be set to 30%–40% of the window size.

VIII. Conclusion and Future Work

To accommodate the growing communication demands of MPSoCs, scalable communication architectures and related design methodologies are needed. In this paper, we have presented a design methodology for designing the optimal crossbar configuration for an application and for binding the cores onto the crossbar resources. Our design approach is based on a simulation window-based analysis of the application traces, considering the local variations in traffic rates, temporal overlap among traffic patterns, and criticality of traffic streams. The methodology has several parameters that can be tuned to explore the design space of the crossbar design and to match the application characteristics. The proposed method is physical design aware, where the wiring complexity of the crossbar architecture is also considered during the design process. Several experimental studies show large reduction in latency and crossbar power consumption when compared with traditional design approaches. In the future, we plan to analyze the effect of using variable simulation window sizes for the design for guaranteeing quality-of-service to applications.

References

[1] W. Wolf, "The future of multiprocessor systems-on-chips," in *Proc. DAC*, Jun. 2004, pp. 681–685.
[2] S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A multiprocessor SOC for advanced set-top box and digital TV systems," *IEEE Des. Test. Comput.*, vol. 18, no. 5, pp. 21–31, Sep./Oct. 2001.
[3] A. Artieri, V. D. Alto, R. Chesson, M. Hopkins, and M. C. Rossi. (2003). "Nomadik open multimedia platform for next-generation mobile devices," *STMicroelectronics Technical Article TA305*. [Online]. Available: http://www.st.com
[4] J. Helmig, *Developing Core Software Technologies for TI's OMAPTM Platform*, 2002, Dallas, TX: Texas Instruments. [Online]. Available: http://www.ti.com
[5] W. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jerraya, and M. Diaz-Nava, "Component-based design approach for multi-core SoCs," in *Proc. DAC*, Jun. 2002, pp. 789–794.
[6] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
[7] D. Wingard, "MicroNetwork-based integration for SoCs," in *Proc. DAC*, Jun. 2001, pp. 673–677.
[8] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. DAC*, Jun. 2001, pp. 684–689.
[9] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vencentelli, "Addressing the system-on-a-chip interconnect woes through communication-based design," in *Proc. DAC*, Jun. 2001, pp. 667–672.
[10] A. Jantsch and H. Tenhunen, *Networks on Chip*. Norwell, MA: Kluwer, 2003.
[11] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip," in *Proc. DATE*, Mar. 2003, pp. 350–355.
[12] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet switched interconnections," in *Proc. DATE*, Mar. 2000, pp. 250–256.
[13] S. Kumar, A. Jantsch, M. Millberg, J. Oberg, J.-P. Soininen, M. Forsell, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *Proc. ISVLSI*, 2002, pp. 105–112.
[14] D. Bertozzi, A. Jalabert, S. Murali, R. Tamahankar, S. Stergiou, L. Benini, and G. De Micheli, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 113–129, Feb. 2005.
[15] T. Simunic, K. Mihic, and G. De Micheli, "Optimization of reliability and power consumption in systems on a chip," in *Proc. PATMOS*, Sep. 2005, pp. 237–246.
[16] T. Yen and W. Wolf, "Communication synthesis for distributed embedded systems," in *Proc. ICCAD*, Nov. 1995, pp. 288–294.
[17] J. Daveau, T. Ismail, and A. Jerraya, "Synthesis of system-level communication by an allocation based approach," in *Proc. ISSS*, Sep. 1995, pp. 150–155.
[18] M. Gasteier and M. Glesner, "Bus-based communication synthesis on system level," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 4, no. 1, pp. 1–11, 1999.
[19] K. Ryu and V. Mooney, "Automated bus generation for multiprocessor SoC design," in *Proc. DATE*, Mar. 2003, pp. 282–287.
[20] K. Lahiri, A. Raghunathan, and S. Dey, "Design space exploration for optimizing on-chip communication architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 6, pp. 952–961, Jun. 2004.
[21] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey, "Design of high-performance system-on-chips using communication architecture tuners," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 5, pp. 620–636, May 2004.
[22] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *J. Syst. Archit.*, vol. 50, no. 2/3, pp. 105–128, Feb. 2004.
[23] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based NOC architectures under performance constraints," in *Proc. ASPDAC*, Jan. 2003, pp. 233–239.
[24] J. Hu and R. Marculescu, "Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures," in *Proc. DATE*, Mar. 2003, pp. 10 688–10 693.
[25] S. Murali and G. De Micheli, "Bandwidth constrained mapping of cores onto NoC architectures," in *Proc. DATE*, Feb. 2004, pp. 20 896–20 902.
[26] S. Murali and G. De Micheli, "SUNMAP: A tool for automatic topology selection and generation for NoCs," in *Proc. DAC*, Jun. 2004, pp. 914–919.

[27] A. Hansson, A. Radulescu, and K. Goossens, “A unified approach to constrained mapping and routing on network-on-chip architectures,” in *Proc. ISSS*, 2005, pp. 75–80.
[28] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelli, “Constraint-driven communication synthesis,” in *Proc. DAC*, Jun. 2002, pp. 783–788.
[29] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelli, “Efficient synthesis of networks on chip,” in *Proc. ICCD*, Oct. 2003, pp. 146–150.
[30] T. Ahonen, D. Sigüenza-Tortosa, H. Bin, and J. Nurmi, “Topology optimization for application specific networks on chip,” in *Proc. SLIP*, Feb. 2004, pp. 53–60.
[31] K. Srinivasan, K. Chatha, and G. Konjevod, “An automated technique for topology and route generation of application specific on-chip interconnection networks,” in *Proc. ICCAD*, Nov. 2005, pp. 231–237.
[32] W. H. Ho and T. M. Pinkston, “A methodology for designing efficient on-chip interconnects on well-behaved communication patterns,” in *Proc. HPCA*, Feb. 2003, pp. 377–388.
[33] S. Murali and G. De Micheli, “An application-specific design methodology for STBus crossbar generation,” in *Proc. DATE*, Mar. 2005, pp. 1176–1181.
[34] J. Hu, Y. Deng, and R. Marculescu, “System-level point-to-point communication synthesis using floorplanning information,” in *Proc. ASPDAC*, Jan. 2002, pp. 573–578.
[35] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane, “Floorplan-aware automated synthesis of bus-based communication architectures,” in *Proc. DAC*, Jun. 2005, pp. 65–70.
[36] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli, “Mapping and configuration methods for multi-use-case networks on chips,” in *Proc. ASPDAC*, Jan. 2006, pp. 146–151.
[37] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli, “A methodology for mapping multiple use-cases onto networks on chips,” in *Proc. DATE*, Mar. 2006, pp. 118–123.
[38] S. Pasricha, N. Dutt, and M. Ben-Romdhane, “Constraint-driven bus matrix synthesis for MPSoC,” in *Proc. ASPDAC*, Jan. 2006, pp. 30–35.
[39] S. N. Adya and I. L. Markov, “Fixed-outline floorplanning: Enabling hierarchical design,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 6, pp. 1120–1135, Dec. 2003. Tool URL: http://vlsicad.eecs.umich.edu/BK/parquet/
[40] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon, “Analyzing on-chip communication in a MPSoC environment,” in *Proc. DATE*, Feb. 2004, pp. 20 752–20 757.
[41] R. Ho, K. Mai, and M. Horowitz, “The future of wires,” *Proc. IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.
[42] [Online]. Available: http://www.ilog.com/products/cplex/
[43] V. Vaizirani, *Approximation Algorithms*. New York: Springer-Verlag, Mar. 2004.
[44] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
[45] [Online]. Available: http://www.synopsys.com

Srinivasan Murali (S'02) received the B.S. degree (with a gold medal) in computer science and engineering from the University of Madras, Chennai, India, in 2002. He is currently working toward the Ph.D. degree in electrical engineering at Stanford University, Stanford, CA.

His research interests include reliable and efficient design methods for networks-on-chips and systems-on-chips.

Mr. Murali received the Best Paper Award in the DATE conference in 2005.

Luca Benini (S'94–M'97–SM'04–F'06) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1997.

He is currently a Professor with the University of Bologna, Bologna, Italy. He also holds a visiting faculty position with Ecole Polytecnique Federale de Lausanne (EPFL), Lausanne, Switzerland. He has published more than 250 papers in peer-reviewed international journals and conferences, three books, several book chapters, and two patents. His research interests are in the design of systems for ambient intelligence, from multiprocessor systems-on-chip/networks-on-chip to energy-efficient smart sensors and sensor networks.

Dr. Benini is a member of the IST Embedded System Technology Platform Initiative (ARTEMIS) working group on design methodologies, a member of the Strategic Management Board of the ARTIST2 Network of Excellence on embedded system, and a member of the Advisory Group on computing systems of the IST Embedded Systems Unit. He has been a member of the 2003 MEDEA+ EDA Roadmap Committee. He has been Program Chair and Vice-Chair of Design Automation and Test in Europe Conference. He has been a member of the technical program committee and organizing committee of several technical conferences, including the Design Automation Conference, International Symposium on Low Power Design, and the Symposium on Hardware–Software Codesign. He is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF CIRCUITS AND SYSTEMS and of the ACM *Journal on Emerging Technologies in Computing Systems*.

Giovanni De Micheli (S'79–M'79–SM'89–F'94) is currently a Professor and the Director of the Integrated Systems Centre with Ecole Polytecnique Federale de Lausanne (EPFL), Lausanne, Switzerland, and President of the Scientific Committee of CSEM, Neuchatel, Switzerland. He was a Professor of electrical engineering with Stanford University, Stanford, CA. He is, or has been, a member of the technical advisory board of several companies, including Magma Design Automation, Coware, Aplus Design Technologies, Ambit Design Systems, and STMicroelectronics. His research interests include several aspects of design technologies for integrated circuits and systems, with particular emphasis on synthesis, system-level design, hardware/software codesign, and low-power design. He is author of *Synthesis and Optimization of Digital Circuits* (McGraw-Hill) and coauthor and/or coeditor of six other books and of over 300 technical articles.

Dr. De Micheli is a Fellow of the Association for Computing Machinery. He was the President of the IEEE CAS Society in 2003. He is currently the President Elect of the IEEE Council on EDA and chairing the IEEE Product Package Committee. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON CAD/ICAS from 1987 to 2001. He was the Program Chair and General Chair of the Design Automation Conference (DAC) in 1996, 1997 and 2000, respectively. He is the Program Chair of the pHealth and VLSI SOC conferences in 2006. He received two Best Paper Awards at the Design Automation Conference in 1983 and in 1993, the 1987 D. Pederson Award for the best paper on the IEEE TRANSACTIONS ON CAD/ICAS, and a Best Paper Award at the DATE Conference in 2005. He also received the Golden Jubilee Medal for outstanding contributions to the IEEE CAS Society in 2000. He was the recipient of the 2003 IEEE Emanuel Piore Award for contributions to computer-aided synthesis of digital systems.