Design Space Exploration for Optimizing On-Chip Communication Architectures

Kanishka Lahiri, Anand Raghunathan, and Sujit Dey

***Abstract*—Rapid growth in the complexity of system-on-chips is being accompanied by increasing volume and diversity of on-chip communication traffic, which in turn, is driving the development of advanced system-level communication architectures. While these architectures have the potential to improve system performance, they pose significant new challenges to the system designer, owing to the complex design space defined by the availability of numerous network topologies, communication protocols, and mapping alternatives for system communications. In this paper, we address the problem of mapping a system's communication requirements to a given communication architecture template. We illustrate the nature of the communication architecture design space, and describe an exploration methodology that uses efficient algorithms to help automate the process of mapping the system communications to the selected template. In addition, we demonstrate the importance of simultaneously optimizing the on-chip communication protocols in order to maximize system performance. Experiments conducted on example systems, including a cell forwarding unit of an ATM switch, indicate that the proposed techniques aid in automatically constructing communication architectures that have high performance. For the systems we considered, the solutions generated using our methodology had 53% superior performance (on average), over those based on conventional architectures and mapping approaches. The algorithms used in the proposed methodology are computationally efficient, and scale well with increasing communication architecture complexity.**

***Index Terms*—Bus architectures, communication synthesis, network-on-chip, on-chip communication, system-level design, system-on-chip.**

I. Introduction

Electronic system design is being revolutionized by the widespread adoption of the system-on-chip (SoC) paradigm. The benefits of the SoC approach are numerous, including improvements in system performance, cost, size, power dissipation, and design turn-around-time. In order to exploit these advantages to the fullest, system design methodologies need to adequately address two dimensions of system design. First, it is essential to optimize the mapping of a system's computation and storage requirements onto a set of high-performance components, like CPUs, DSPs, application-specific cores, memories, etc. Second, it is equally important to optimize the mapping of the system's communication requirements onto a set of selected communication resources. The focus of this paper lies on this latter aspect of system design.

The on-chip communication architecture is a fabric that integrates the various SoC components, and provides them with a mechanism for the exchange of data. While trends in system complexity indicate increasing demands placed by the on-chip communication traffic on the communication architecture, technology scaling trends indicate that the performance and energy characteristics of global interconnect are rapidly deteriorating, relative to logic components. These trends, taken together, make the communication architecture an increasingly critical determinant of several system-wide metrics, including overall system performance and power consumption [1]–[4]. As a result, traditional architectures for on-chip communication, such as those based on a single shared bus, often fail to satisfy stringent performance requirements. This is driving the development of advanced SoC communication architectures, ranging from derivatives of buses to complex networks of communication channels, with associated communication protocols [5], [6]. When customizing such architectures toward the requirements of an application (or application domain), careful attention needs to be paid to several aspects, including: 1) selection of an appropriate network topology; 2) selection of appropriate communication protocols, along with configuration of various protocol parameters; and 3) mapping of the system communications to physical paths in the topology. Each of these steps by themselves pose numerous design alternatives. Additionally, as shown in this paper, these steps often interact, leading to tradeoffs that are very difficult to identify or evaluate in the absence of automatic tools. The aim of this work is to provide system designers with automatic tools to enable rapid exploration of such tradeoffs, and techniques to efficiently implement a given system's communication requirements as an on-chip communication architecture.

A. Paper Overview and Contributions

In this paper, we describe a design-space exploration methodology for optimizing system-level on-chip communication architectures. The methodology takes as inputs, a system description that has been partitioned into hardware (HW) and software (SW), and mapped onto appropriate components, and the selected communication architecture template. The methodology uses efficient algorithms to generate an optimized mapping of the system's communication requirements onto a selected communication architecture template. The techniques also help in statically customizing the on-chip communication protocols.

The methodology is based on identifying, through efficient performance analysis, the characteristics of system-level communication traffic, including an accurate analysis of potential contentions for shared channels in the architecture. The exploration and optimization methodology consists of: 1) a clustering algorithm that helps determine an initial mapping of the SoC communications to the network topology and 2) an iterative improvement strategy that takes into account dynamic effects in order to improve upon the quality of the initial solution. The result is an optimized mapping of system communications, along with suitably configured communication protocols.

In Section II, we provide background on SoC communication architectures. In Section III, we study the nature and size of the communication architecture design space, evaluate the potential advantages of accurate design space exploration, and illustrate, using examples, key issues that need to be considered by the design space exploration methodology. In Section IV, we describe the proposed exploration and optimization methodology, and detail the various steps in Section V. In Section VI, we present results of experiments that evaluate the methodology, by considering its application to several example systems.

B. Related Work

A large body of work exists on system-level synthesis of application-specific architectures through HW/SW partitioning and mapping of the application tasks onto predesigned cores and application-specific hardware [7]–[11]. While most previous research has focussed on optimizing computation and storage during the mapping process, only recently has the problem of mapping a system's communication requirements onto an underlying on-chip communication architecture started to receive interest.

Manuscript received October 2, 2002. This work was supported in part by NEC Laboratories America and in part by the California MICRO program. This paper was recommended by Associate Editor R. Camposano.

K. Lahiri was with the Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA 92093 USA. He is now with NEC Laboratories America, Princeton, NJ 08540 USA (e-mail: klahiri@nec-labs.com).

A. Raghunathan is with NEC Laboratories America, Princeton, NJ 08540 USA.

S. Dey is with the Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA 92093 USA.

Digital Object Identifier 10.1109/TCAD.2004.828127

Research on synthesis of on-chip communication architectures mostly deals with the generation of optimized network topologies [12]–[16]. In addition, several new communication protocols for high performance on-chip communication have recently been proposed [17]–[20]. While topology selection and protocol design are each critical steps in the design of a communication architecture, in this work, we address a complementary problem. We assume the network topology and protocols are predetermined via selection of a communication architecture template, which are becoming increasingly available from interconnect core providers [17], [21]–[23]. We address the problem of how to optimize the mapping of system communications to provided templates, while at the same time, suitably configuring the on-chip communication protocols.

To combat problems associated with long, capacitive wires typically faced by bus-based architectures, researchers have proposed the use of communication architectures based on the concept of a network-on-chip consisting of regular interconnect topologies and associated protocols capable of supporting large numbers of communicating elements [5], [24]. These complex networks create significant challenges in determining the best way to map system communications to an underlying network topology.

Fast and accurate system-level performance analysis is the key to a practical design space exploration methodology. Research on system-level performance analysis of on-chip communication include approaches based on simulation of the entire system, which model communication at varying levels of abstraction [25], [26]. However, the high computational cost of simulation-based techniques make these techniques infeasible when exploring a large design space. More efficient static performance-estimation techniques include [12]–[14], and [27], but they do not model dynamic effects like bus contention accurately enough to drive an optimization methodology. A third category of performance-analysis techniques that account for on-chip communication include trace-based techniques [28], [29]. In this work, we adopt the trace-based performance-analysis framework described in [28], which provides for accurate modeling of dynamic effects such as bus contention, while at the same time being far more efficient than simulation-based techniques.

Recent advances that facilitate the process of optimizing the communication architecture include development of latency insensitive design techniques [30]. Latency insensitive design refers to a methodology in which the functional correctness of a system consisting of a set of communicating components is guaranteed, even though the delay associated with system-level communications may vary. The use of latency insensitive system specifications allows system designers to explore numerous alternative communication architectures, without incurring the large cost of verification at each step.

Finally, a body of work on interface synthesis and design [31]–[37] addresses issues related to implementation of specified protocols in hardware, and enables component reusability. Once the mapping of the system components to a communication architecture has been performed, these techniques help generate the interface logic between pairs of communicating components. In order to promote design reuse, recently, standards have started to evolve to define specifications for communication interfaces, or wrappers [38], [39]. Such initiatives aim at standardizing the signaling conventions at the component-communication architecture interface. These efforts will facilitate customization of the communication architecture topology, mapping and protocol parameters.

II. Background: On-Chip Communication Architectures

The design of on-chip communication architectures must address several issues including: 1) definition of an appropriate network *topology*; 2) selection and configuration of the communication *protocols*; and 3) definition of a *mapping* of the system communications.

The communication architecture topology defines the physical structure of the communication architecture. Numerous topology alternatives exist, ranging from those based on a single shared bus (which connects all system components), to more complex architectures, such as bus hierarchies, token rings, crossbars, or custom networks. When the topology has multiple communication channels (or buses), *bridges* are used to interconnect the necessary channels. Components connected to the topology include: 1) *master* components, (e.g., CPUs, DSPs), which are capable of initiating communication transactions and 2) *slave* components (e.g., memories, peripherals), which merely respond to transactions initiated by a master. In this paper, we consider on-chip communication architectures ranging from simple bus-based architectures to those consisting of arbitrary networks of shared and dedicated channels. We assume the on-chip communication architecture is memoryless, i.e., data cannot be stored in the communication architecture.

Since multiple masters often share communication channels in the communication architecture, communication protocols are used to manage access to each channel. These protocols implement arbitration algorithms such as round-robin access, TDMA [17], and priority-based selection [23], [40]. In addition, these protocols are responsible for defining the logical conventions that govern communication transactions. For example, typical protocols provide for the availability of burst transfers, wherein a master is given the right to use the bus for multiple cycles, without being required to negotiate repeatedly with the arbiter. The protocol may specify maximum burst lengths to prevent a master from monopolizing a bus. The protocols also define other functionality, such as pipelined transactions, split transactions, word sizes, endianness, etc.

Communication mapping refers to the process of associating abstract system-level communications with physical communication paths in the communication architecture topology. This step is trivial for communication architectures based on a single shared bus (where all communications are mapped to the bus). However, as shown later in this paper, when the communication architecture has numerous channels, the number of possible mappings grows exponentially, making simple exhaustive search impractical. Additionally, the interaction between communication protocols and communication mappings adds further to the complexity of this design space, calling for exploration techniques based on accurate analysis to derive architectures that are well optimized for the given application.

III. Characteristics of the Communication Architecture Design Space

We formulate the problem of designing the on-chip communication architecture for a partitioned and mapped HW/SW system as consisting of three steps: 1) defining a topology consisting of a network of channels (each serving either as a dedicated point-to-point link, or as a shared bus) interconnected by bridges; 2) customizing the communication protocol for each channel; and 3) mapping the system's communications onto paths in the network topology (by mapping components onto channels). In this work, we focus on the latter two steps, assuming that the designer has selected an architectural template for the communication architecture. The communication architecture template specifies: 1) the network topology, which could be any arbitrary interconnection of shared and dedicated communication channels and 2) a definition of the communication protocols used on each channel, with available configurable parameters.

The reasons we chose this approach are twofold. First, several such templates are commercially available to the designer today, covering

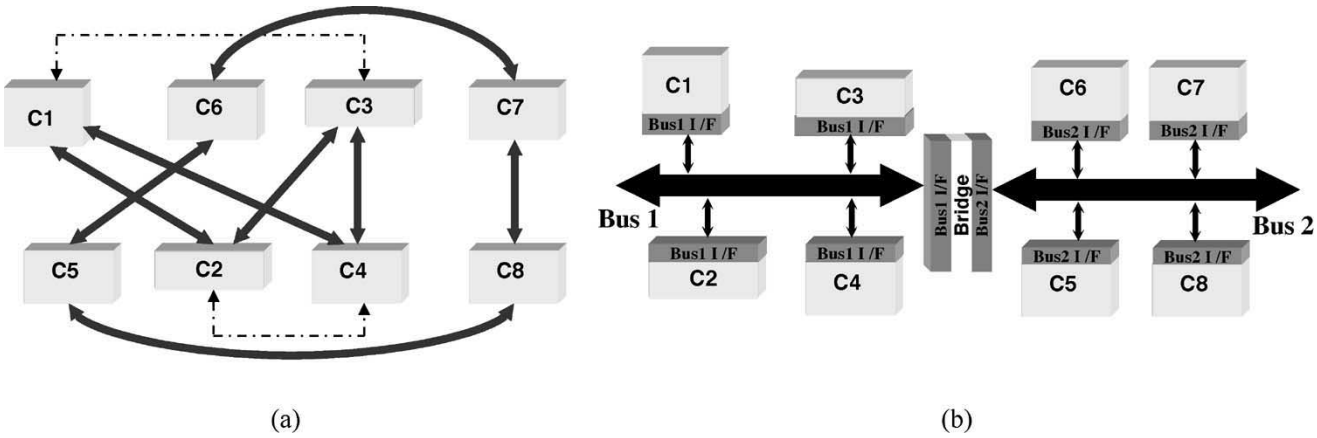

Fig. 1. Example system of communicating components: (a) logical view of intercomponent communication, (b) a possible mapping of system communications to a selected network topology, called `Arch1`.

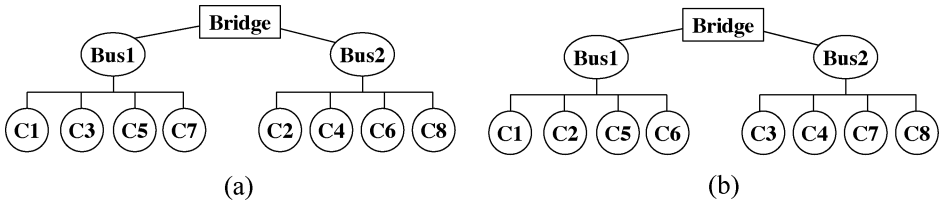

Fig. 2. Alternative mappings for example system: (a) `Arch2` and (b) `Arch3`.

a range of topologies and communication protocol alternatives [17], [21]–[23], [41]. Second, being faced with such choices, we believe the designer should be empowered by automatic tools in order to evaluate alternative templates for a given system. Our techniques aim at providing these tools, using which the designer can optimize the mapping of system communications to each template in turn, customize the template, and evaluate the resulting solutions using fast system-level performance analysis.

In this section, using examples, we illustrate the key issues that arise in the process of mapping a system's communications to a given template. We first show that the design space comprising alternative communication mappings and communication protocol configurations is significantly large, making techniques based on exhaustive search impractical with existing tools. Next, we show that the variation in performance across the design space is large enough to motivate careful exploration, due to the performance gains that may be possible. We show that an approach that merely clusters frequently communicating components does not necessarily provide the best solution, due to the additional complexity introduced by temporal properties of system communications. Finally, we show that selecting a particular mapping of components and communications to physical paths without considering the effects of the on-chip communication protocols can result in significantly suboptimal designs.

Example 1: Fig. 1(a) shows a system consisting of a set of eight components, each of which execute a set of computation tasks and also perform data and control communications. Bold lines indicate transfer of data, while dotted lines indicate exchange of control or synchronization signals. Note that these lines indicate the logical view of intercomponent communication, and not physical paths in the communication architecture. Fig. 1(b) shows a communication architecture where all the communications generated by the system components are mapped to one of the two shared buses connected by a bridge. For example, communications between C_1 and C_2 are mapped to $\mathrm{Bus1}$, while those between C_5 and C_6 are mapped to $\mathrm{Bus2}$. However, for this template, there could be other mappings as well, such as the ones shown symbolically in Fig. 2(a) and (b). For example, Fig. 2(a) shows an architecture in which communications between C_1 and C_4 are mapped to a path of buses: $\mathrm{Bus1} \rightarrow \mathrm{Bus2}$.

TABLE I
PERFORMANCE VARIATION OVER DIFFERENT POINTS IN THE DESIGN SPACE

Cases	Bus 1	Bus 2	Performance (clock cycles)
Arch1	C1 C2 C3 C4	C5 C6 C7 C8	11723
Arch2	C1 C3 C5 C7	C2 C4 C6 C8	15314
Arch3	C1 C2 C5 C6	C3 C4 C7 C8	9242

Table I reports the performance of the system under different mappings as measured by the performance-analysis tool described in [28]. Each row represents a distinct mapping of components to buses in the communication architecture. For example, in `Arch1`, components C_1–C_4 are grouped onto one bus, and components C_5–C_8 are on another bus [Fig. 1(b)]. In this case, the system takes 11 723 cycles to process a fixed sequence of input stimuli. `Arch1` is 23% faster than `Arch2` [Fig. 2(a)], which takes 15 314 cycles to process the same sequence of input stimuli. However, `Arch1` is 27% slower than `Arch3`, which takes 9242 cycles to complete processing the stimuli. ■

The above example illustrates the following issues.

- The number of alternative mappings can be very large for a system with numerous components and channels. For the small example in Fig. 1, (8 components and 2 channels), under the assumption that the two buses are identical, and that the components must be equally partitioned between them, the number of possible mappings is $\binom{8}{4}$, or 70.

  For an arbitrary set of n components and k distinct channels the number of possible mappings is k^n. In practice, though, the design space is smaller, since there typically exist limits on the

number of components that may be assigned to each channel. However, as shown later in this section, for each choice of mapping, there exist important performance critical choices when configuring the communication protocols for each channel. Combined with the problem of choosing the best communication mapping, this results in a highly complex design space.

- The performance variation across this design space is significant. The total time taken by the system to complete execution varies by up to 65%, depending on the selected communication architecture. This motivates the need for systematic exploration of the design space.
- Exploiting the characteristics of intercomponent communication traffic can lead to more optimized solutions. For example, note that `Arch1` outperforms `Arch2`. The reason behind this is that the mapping of components to buses in `Arch1` is derived by clustering components that exchange large volumes of performance critical data. This reduces the number of communications that span multiple buses, and incur the overhead of communicating over the bridge.

In the next example, we demonstrate that using simple metrics such as the volume of communication traffic to drive the communication architecture optimization does not necessarily yield the best solution.

Example 2: Let us consider again the system discussed in Example 1, and examine the alternative mappings in a little more detail. Table I reported that system performance under `Arch1` is superior to that under `Arch2`. The mapping in `Arch1` provides superior performance because: 1) the number of communication transactions that span multiple buses are minimized and 2) components that exchange large amounts of performance critical data are attached to the same bus. Both of these factors result in low transmission latencies. However, as we see next, this approach does not necessarily yield the best architecture. If we change the mapping of components to channels to that of `Arch3` [Fig. 2(b)], then we find that the system completes its task in 9242 cycles, a further improvement of 21% over `Arch1`. The reason for the improvement is that the new mapping separates components that have largely overlapping communication lifetimes (C_1 from C_3 and C_5 from C_7), resulting in an implementation that causes fewer conflicts and hence enhanced concurrency in the system's execution. However, the penalty paid is that the number of communication transactions going across the bridge is no longer minimum, since component C_1 communicates with component C_4 and C_3 with C_2 [Fig. 1(a)]. For this system, the benefit of reduced conflicts outweighs the penalty of transactions that span multiple buses. Hence, `Arch3` is the architecture that best recognizes and takes advantage of the characteristics of the application's communication traffic. The example shows a case where a simple clustering heuristic fails to generate the best solution. ■

Example 3: Finally, we illustrate that communication architecture design tools should incorporate the influence of the on-chip communication protocols in order to accurately assess the quality of a candidate solution. Using the same example from Fig. 1, recall that the performance-analysis results indicated that `Arch3` was the best solution due to reduced conflict level on each bus. However, the performance estimate for `Arch3` in Table I was derived assuming that optimized bus-protocol parameters were used for each of the two buses.

Suppose that, while examining alternative solutions, we ignore the effect of the bus protocol, and assume fixed protocol parameters across different mapping alternatives. To illustrate the danger of such an approach, we report on the following experiment. Each bus was assigned a static priority-based arbitration protocol, with fixed protocol parameters (we consider the bus access priorities of each component, and the maximum burst transfer size, as the protocol parameters).

TABLE II
EFFECT OF COMMUNICATION PROTOCOLS IN THE DESIGN SPACE

Case	Bus 1 Protocol	Bus 2 Protocol	Performance (cycles)
Arch3 Sub-opt	C1>C2>C5>C6 BURST= 5	C3>C4>C7>C8 BURST = 10	12504
Arch3 Opt	C1>C6>C2>C5 BURST = 10	C3>C7>C4>C8 BURST = 10	9242

In the first experiment, we evaluated *Arch3* [Fig. 2(b)], leaving the protocol parameters unchanged from `Arch2`, (as shown in row 1 of Table II). The system took 12 504 cycles to complete the task, a degeneration of 6.7% over `Arch1`. However, after we generated optimized bus protocol parameters for `Arch2` (row 2 of Table II), the best performance result of 9242 cycles was obtained. ■

The above example illustrates the following important point: while examining alternative candidate communication architectures, the effects of the on-chip protocols cannot be ignored. When the set of components mapped to a channel is changed, the traffic characteristics on that channel change, and hence necessitate re-examination of previously configured protocols. This suggests that the problems of selecting an optimized communication mapping, and choosing the best set of protocols, are interrelated, and that solving each independently could easily lead to suboptimal solutions. In the example, such an approach would have made us overlook `Arch3`. Hence, in order to make an accurate comparison between two alternative communication mappings, it is necessary to: 1) derive optimized protocols for each and 2) conduct performance analysis of each architecture while taking into account the effects of both the selected communication mapping as well as the selected protocol configuration.

IV. DESIGN-SPACE EXPLORATION AND OPTIMIZATION METHODOLOGY

In this section, we present an overview of our communication architecture design methodology and highlight the important steps. In the next section, we describe how some of the important steps are conducted in greater detail.

The overall methodology is shown in Fig. 3. The inputs (shown shaded in the figure) consist of: 1) a system specification that has been partitioned into HW and SW, and mapped to appropriate cores or custom HW and 2) a communication architecture template consisting of a network topology (defined by a set of shared/dedicated communication channels interconnected by bridges), and associated communication protocols. Note that the methodology applies to topologies that are memoryless. This implies that data transfers, once successfully initiated, must complete before another transfer can occur, since data cannot be stored internally in the communication architecture. The algorithms automatically generate an optimized mapping of system components to specific channels in the target architecture, as well as optimize the communication protocols for each channel by customizing the protocol parameters.

In the first step, HW/SW cosimulation of the partitioned/mapped system description is performed, with communication modeled using instantaneous, conflict-free exchange of communication events. Execution traces are collected and stored in a compact representation called a symbolic execution graph (SEG), which captures the abstracted system behavior (including computation, communication, and intercomponent synchronization) over the entire execution trace [28]. Using the analysis algorithms detailed in [28], Step 3 generates various statistics about the system performance and intercomponent communication traffic. Based on these statistics, and a specification of the com-

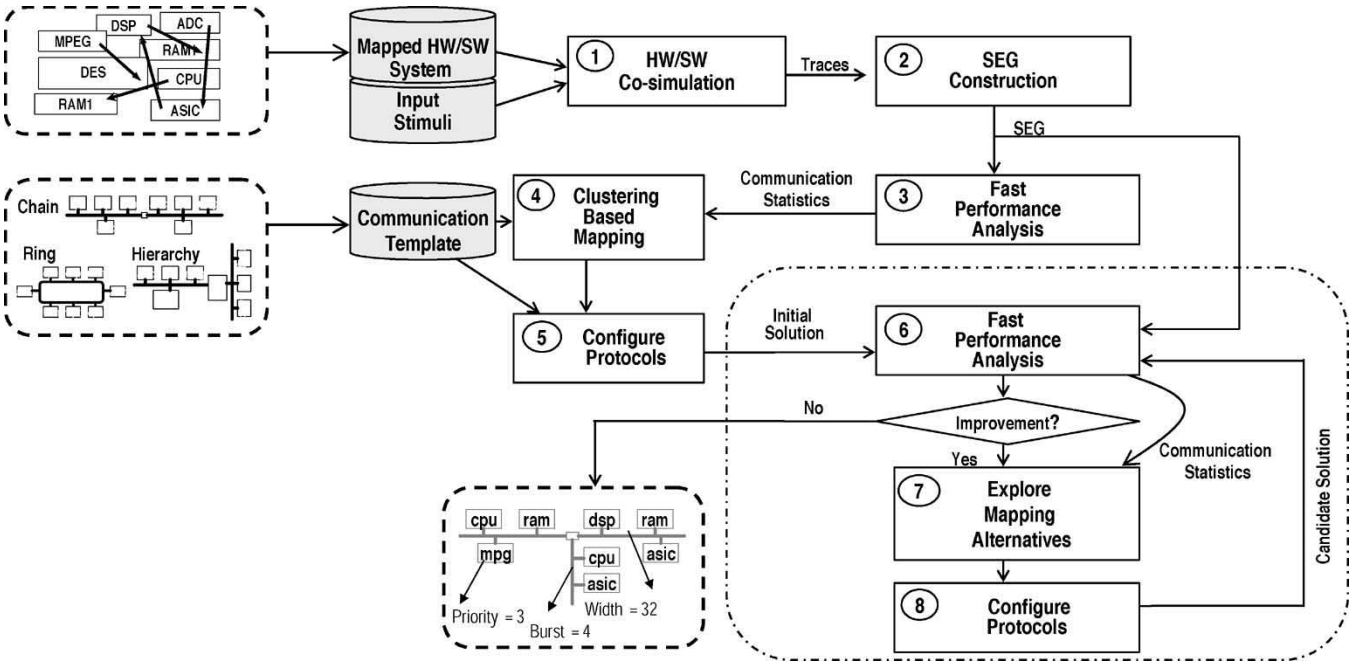

Fig. 3. Methodology for design space exploration of on-chip communication architectures.

munication architecture template, clustering techniques (Step 4) are used to generate an initial mapping of components to the target architecture. Step 5 determines a set of optimized protocol parameters for each channel. The results of Steps 4 and 5 together constitute an initial solution.

As demonstrated in Section III, an approach that stopped with the initial solution may result in poor system performance for many systems. Hence the need for the second, iterative improvement part of our methodology. In Step 6, the performance-analysis tool is re-invoked, to consider effects of the generated communication architecture. The tool re-evaluates system performance and gathers new communication statistics. Based on these statistics, Step 7 explores alternative solutions by calculating the potential performance gains of moving already assigned components from one channel to another, and chooses the best set of candidate "moves" to construct a new solution. The output of Step 7 is a new mapping of components to the target architecture. Step 8 reconfigures the protocol parameters, for reasons illustrated in Section III. The new solution is re-evaluated, and the iterative procedure (Steps 6–8) is repeated till no further improvement in performance is obtained.

V. Algorithms

In this section, we first describe details of the algorithms used by the methodology of Fig. 3. First, we describe how the given system, its intercomponent communication statistics, as well as the given communication architecture template are modeled. We next describe the techniques used to compute the initial solution, and then consider how the iterative part of the methodology improves on that solution.

A. Modeling Intercomponent Communication Statistics

Statistics generated by the performance analysis of Step 3 are represented in an intercomponent communication graph. The communication graph is a directed graph consisting of one vertex for each component, and an edge (C_i, C_j) when there exists communications between component C_i and C_j. The direction of the edge is dependent on the master-slave relationship between them.[1] An edge (C_i, C_j) is annotated with properties of the communication transactions seen between components C_i and C_j, including the number of transactions, distribution of their sizes (mean, variance), critical path information, (expressed as the distribution of their timing slacks), the number of transactions with zero slack (critical transactions). While the various parameters on each edge may be used in several different ways, in our implementation, we chose to derive a single weight for each edge by taking the product of the average size (in number of bus words) and the number of transactions between C_i and C_j, and scaling it by the average slack (in cycles) (Fig. 4). This takes into account frequency, volume and criticality of transactions that occur between components C_i and C_j.

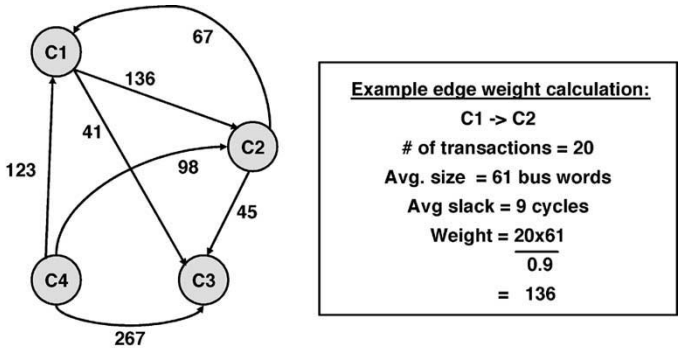

Fig. 4. Communication graph: example.

By examining the Communication Graph, Step 4 calculates for each component, a measure of the load it imposes on the communication architecture. For component C_i this is the sum of the weights of the outgoing edges from C_i in the Communication Graph. It then arranges the components in descending order of load.

[1]A master initiates a communication transaction, while a slave only responds to transaction requests.

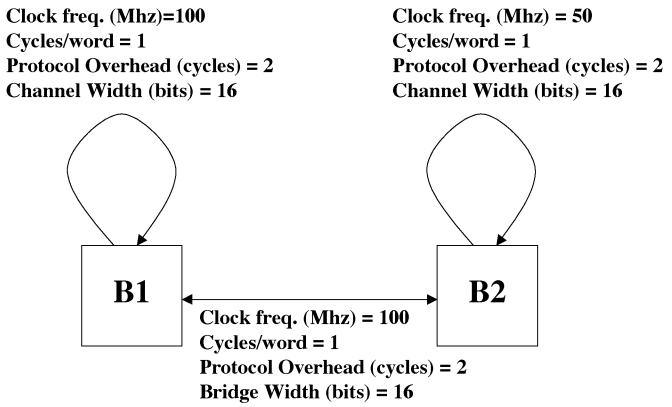

Fig. 5. Topology graph: example.

B. Network-Topology Modeling

The network topology of the architectural template is modeled using a *Topology Graph*. An example of a Topology Graph is shown in Fig. 5, corresponding to the template shown in Fig. 1(b). The graph consists of one vertex for each channel, and bidirectional edges between vertices for channels that are interconnected by a bridge. The information on such edges contain parameters that describe the properties of the bridge—the overhead of transmitting a single word over it, its frequency of operation etc. Each vertex also has a self-looping edge, which describes properties of the channel it represents. For each channel P in the topology, a *connectivity* metric is calculated by estimating the time taken to transfer a fixed amount of data from a component on channel P to each of the other channels in the topology. This estimate is obtained using the topology model described above and includes: 1) *Protocol Overheads* and 2) *Transmission Delay*. *Protocol Overheads* are contributed by handshaking conventions on each channel/bridge, and *Transmission Delay*, is determined by the channel bandwidth. The result is used to derive a connectivity rank for channel P. Note that this calculation ignores the possibility of conflicts. The intuition behind the ranking procedure is that channels which have high bandwidth, and are "well connected" to the rest of the network topology, have low delays and, hence, obtain a high rank.

C. Clustering Procedure for Initial Solution

The clustering procedure picks the highest ranked component that is not yet assigned to a channel and then tries to decide which channel to assign it to. For example, suppose channel P has components C_i and C_j assigned to it and component C_k is being considered for assignment. The interaction level of component C_k with channel P is measured by summing the weights of the edges (C_k, C_j), (C_k, C_i), (C_i, C_k), (C_j, C_k) in the communication graph. This is repeated for each channel, and the one for which the result is maximum is the target channel for component C_k. If this level exceeds a threshold, then the assignment is made. If not, it implies that too few components have been assigned to make an informed decision using the above technique. In this case, the list of components is scanned for a component which has not yet been assigned, and has maximum interaction with the current component C_k. The ranks of the two components, the number of components assigned to each channel so far, and the ranks of the available channels are used to assign the component pair to an appropriate channel. If there still exist choices among a few alternatives, a channel is randomly chosen.

1) Assigning Protocol Parameters: The procedure for optimizing the protocols is specific to the exact set of configurable parameters that are available in the communication architecture template. We explain the procedure for an architectural template that uses a static priority-based protocol for each channel, since this is the most commonly used protocol in practice today e.g., [23], [40]. The parameters that we consider are the priorities of each component, and the maximum permissible burst transfer size.

- Priorities are assigned to components sharing a channel by examining the ranks of each component in the sorted list of components generated earlier.
- The maximum burst transfer size for a channel k is calculated as the weighted average of the size of transactions mapped to the channel, where the weight incorporates the criticality, derived from the average values of the timing slack. This favors large burst transfer sizes when large transactions lie on the system critical path.

D. Iterative Improvement Incorporating Dynamic Effects

In this section, we describe how we construct a sequence of moves (or transformations) to yield a solution that improves the system's performance, while taking into account the effects of runtime conflicts in the communication channels. Given an assignment of components to channels, with a set of protocol parameters for each channel, Step 7 first efficiently computes the potential gain of moving component C_i from the channel to which it is currently mapped (P) to another channel (Q). This is repeated for every combination of component and potential destination channels. The rest of the algorithm uses well-known iterative techniques to determine a subsequence of moves whose cumulative gain is maximum, in order to construct an improved solution [42]. At each step in the iterative procedure, the move that is chosen is the one producing the maximum gain. Note that the move producing the maximum gain may result in a deterioration (gain may be negative). However, a sequence of moves with a net positive gain may be enabled by considering individual moves with negative gain. Thus, the exploration technique provides for a degree of hill-climbing in order to avoid local maxima.

1) Estimating the Potential Gain of a Move: The potential gain of a move is efficiently estimated using the procedure described in Fig. 6, which makes use of additional information that is generated by the performance-analysis tool for the architecture under consideration. We first describe the principle behind the procedure through the use of an example. Under a given architecture, the lifetime of a communication transaction is made up of three parts: 1) waiting time arising out of protocol overhead; 2) waiting time arising out of simultaneous access attempts to the shared channel i.e., conflicts; and 3) time taken to transfer the data. The performance-analysis tool generates, for each pair of components (C_i, C_j), the number of cycles for which the lifetimes of communication transactions involving C_i overlaps with transaction lifetimes involving C_j. Fig. 7(a) shows a set of overlapping communication transaction lifetimes, and Fig. 7(b) shows the resulting *communication conflict graph*.

Given a certain mapping of components to channels, the *conflict level* for a channel P is calculated by accumulating the edge weights for every (C_i, C_j) in the communication conflict graph, where C_i and C_j represent components mapped to channel P. When considering a move of component C_i from channel P to channel Q, the algorithm calculates the potential decrease in the conflict level on channel P and the potential increase in the conflict level on channel Q, and estimates the merit of performing such a move. To illustrate this, consider the example shown in Fig. 7(b), where the components are shown grouped into two buses, and potential gain of moving component C_3 from $\mathrm{Bus2}$ to $\mathrm{Bus1}$ is being evaluated. From Fig. 7(b), the conflict level on $\mathrm{Bus2}$ is calculated as 19 (6 between C_1 and C_2, 5 between C_2 and C_3, and 8 between C_1 and C_3). After moving C_3, the conflict level on $\mathrm{Bus2}$ reduces to 6 (6 between C_1 and C_2), while the conflict level on $Bus1$ increases from 2 to 9. Fig. 7(c) shows the conflict levels on each bus before and after moving vertex C_3 from $\mathrm{Bus1}$ to $\mathrm{Bus2}$.

To calculate the potential gain of moving component C_i from channel P to Q, the pseudocode of Fig. 6 is executed. The first loop

```
Evaluate_gain
inputs: COMPONENT i, BUS P, BUS Q
outputs: FLOAT gain
begin
    for each v∈P, u∈Q
        old_delay = old_delay +
            calculate_comm_delays(v, old_speeds);
        old_delay = old_delay +
            calculate_comm_delays(u, old_speeds);
    end for
    l_P' = conflict_level(P);
    l_Q' = conflict_level(Q);
    for each component j such that
        j∈Q && overlap_cycles(i, j) ≠ 0:
        l_Q = l_Q + overlap_cycles(i, j);
    end for
    for each component k such that
        k∈P && overlap_cycles(i, k) ≠ 0:
        l_P = l_P − overlap_cycles(i, j);
    end for
    Q → new_speed = (l_Q − l_Q') / l_Q' × (Q → old_speed);
    P → new_speed = (l_P' − l_P) / l_P' × (P → old_speed);
    send(i, P, Q);
    for each v∈P, u∈Q:
        new_delay = new_delay +
            calculate_comm_delays(v, new_speeds);
        new_delay = new_delay +
            calculate_comm_delays(u, new_speeds);
    end for
    gain = old_delay − new_delay;
    return gain;
end
```

Fig. 6. `Evaluate_gain` procedure to compute gain of each move.

accumulates the communication delays associated with communications involving components on channels P and Q, using the method described in Section V-B. The result is stored in old_delay. Then the conflict levels on the each channel are calculated as described earlier and saved in $l_{P'}$ and $l_{Q'}$. The second loop calculates the new (increased) conflict level l_Q on channel Q, and the third loop calculates the new (decreased) conflict level l_P on channel P. Then the speed of each channel is symbolically scaled by the conflict level on each channel. The time taken for all communications involving components on channels P and Q are recalculated (new_delay), and compared with the previous value. The difference is the potential gain of performing the move.

E. Accuracy and Efficiency Issues

In this section, we explain why the performance-analysis tool needs to be invoked inside the body of the iterative procedure (in Step 6 of Fig. 3). The reason behind this strategy is that the intercomponent-communication statistics gathered through performance analysis can be divided into two categories: 1) statistics that are independent of the communication architecture, which include the number of communication transactions that occur over the period of the trace between each pair of components, and the sizes of these communications (number of bytes per transaction) and 2) statistics that are sensitive to the exact implementation of the communication architecture, which include information about the timing of communication transactions, time spent by a component waiting for access to a shared channel, the execution time of each communication transaction, the critical transactions exchanged between a pair of components, and the available timing slack on each transaction. For each new mapping of system communications, the architecture-dependent statistics (the latter category) need to be re-evaluated. This requires the analysis tool to be re-invoked at the beginning of each iteration.

Taking this further, it may be argued that in order to achieve even greater accuracy, it should be necessary to run the analysis tool even deeper in the iterative procedure, namely in Step 7, when the potential gain of moving a component from one channel to another is calculated. However, this could result in significantly increased runtimes for the exploration tool. Additionally, in our experiments, we observed that the benefit may be limited for templates based on complex network topologies, where the effect of moving a single component from one channel to another is often local, with little effect on the majority of the communication transactions in the system. Individual local perturbations can be evaluated using simpler, more efficient procedures, such as the one described in Section V-D1.

VI. Experimental Results

In this section, we present results of experiments conducted on example systems, and evaluate the benefits of using the exploration techniques presented in this paper. In the first set of experiments, we compare the performance of communication architectures generated using our exploration techniques with ones based on conventional bus architectures. In addition, we report on the CPU time consumed by the exploration tool while generating these solutions. In the second set of experiments, we evaluate the use of our exploration technique to evaluate tradeoffs in selecting communication architecture templates.

A. Experimental Methodology

The experiments were conducted on two example systems. The first is a cell forwarding unit of an output queued ATM switch, depicted in Fig. 8. The `ATM` system consists of 8 output ports, each with a local queue of cell headers. The system also has three shared memory banks, to store the arriving cell payloads. Each port periodically polls its queue to detect presence of a cell. When nonempty, it issues a *dequeue* signal to its local queue, extracts the relevant cell from the appropriate shared memory and sends it onto its output link. The second system (`SYS`) is the one described in Fig. 1.

Each system was specified as a set of concurrent communicating tasks, with communication modeled as the exchange of abstract communication events. HW/SW partitioning and mapping was performed using the POLIS [11] framework, and system-level simulation was carried out in PTOLEMY [43]. The resulting simulation traces were used in the subsequent communication architecture analysis and exploration algorithms. For the `ATM` example, the network topology consisted of three buses interconnected by 2 bridges. For the `SYS` example, the network topology consisted of two buses connected by a bridge with specified parameters (width, speed, etc.) as shown in Fig. 1(b). In each case, the final architecture generated using the described analysis and exploration methodology was evaluated using detailed HW/SW cosimulation to confirm the accuracy of the predicted performance estimates. The performance estimation error was found to be less than 2% in all cases.

B. Effectiveness of the Exploration Techniques

Tables III and IV report on the performance of the `SYS` and `ATM` examples, respectively. In each table, the performance of the system is

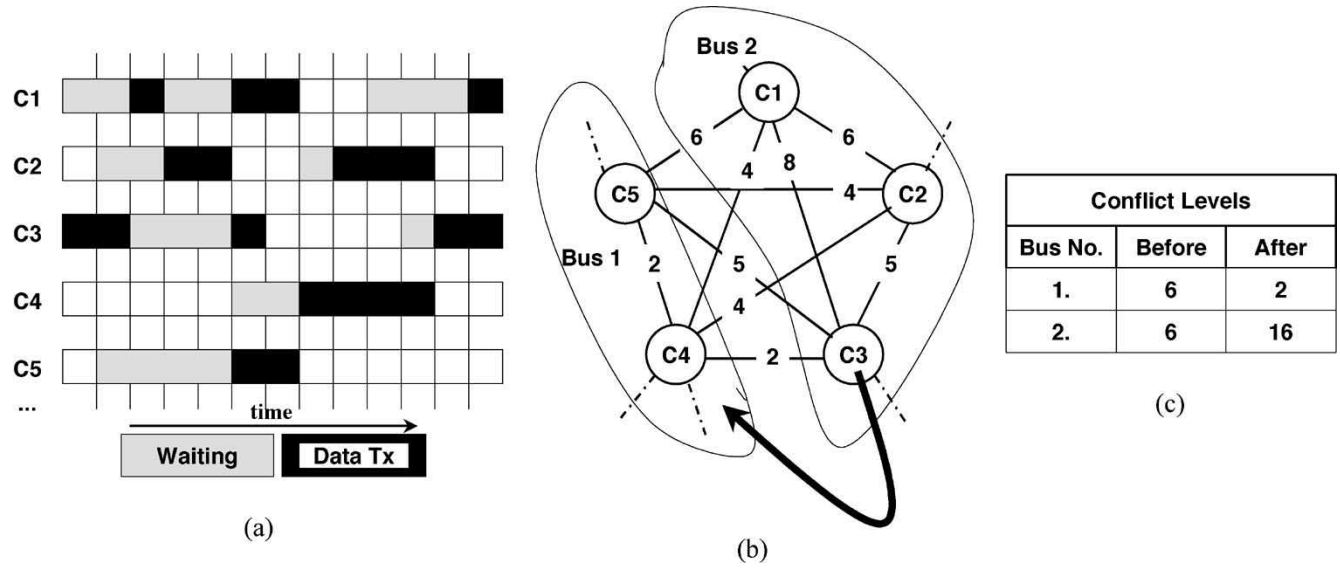

Fig. 7. Taking into account conflicts during estimation of gain of performing a move. (a) Execution trace showing communication lifetimes; (b) Communication Conflict Graph; (c) Conflict levels when considering moving component $C3$ from $Bus1$ to $Bus2$.

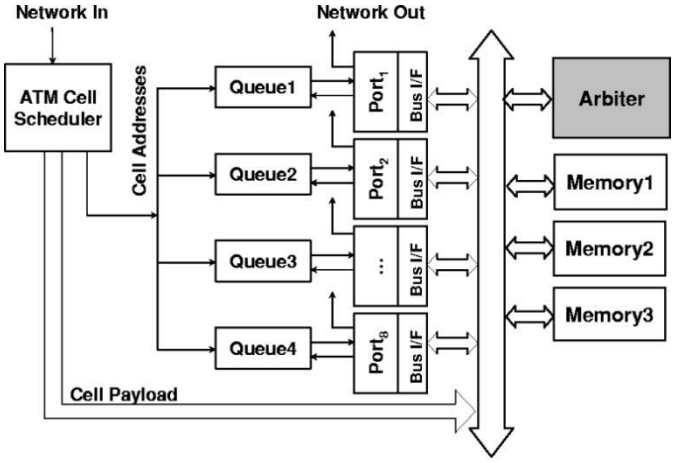

Fig. 8. Example system: ATM.

reported under different communication architectures. The rows correspond to the following architectures: in row 1, all components are mapped to a single shared system bus; in row 2, components are randomly mapped to the selected communication architecture topology; in row 3, the mapping and protocols are determined by the clustering-based initial solution; in row 4, the mapping and protocols are those determined by the complete exploration procedure; in row 5, the communication architecture is based on an ideal one, which allows for infinite concurrency and bandwidth (this provides a lower bound on the system execution time).

Column 2 of Tables III and IV reports the actual performance measurement in terms of the number of clock cycles taken to accomplish a given task. In the case of SYS this was the time taken to process 2000 input stimuli, and for the ATM example, it was the time taken to process 1000 cells. In column 3, each table reports performance of each configuration relative to the case where all communication goes through a shared bus (row 1). Column 5 reports CPU times with the following interpretations. In rows 1 and 2, (shared bus and random), the reported CPU is that spent in choosing optimized protocol parameters (the mapping is predetermined) and a single evaluation of system performance using the tool presented in [28]. In row 3 the CPU time measurement includes the time spent in constructing the initial solution, protocol configuration, and performance analysis. In row 4 (optimized solution), it represents the time spent in analysis, construction of the initial solution, protocol configuration, and the iterative procedure. Finally, for the last row, CPU time indicates the time required to perform HW/SW cosimulation of the entire system, to generate the initial system execution traces.

From these tables we observe the following:

- The clustering-based approach by itself provides significant performance improvements for both systems, as compared to the shared bus solution, and the random solution. For the SYS system, clustering results in improvements of 52% and 23% over the shared bus and random solutions respectively. For the ATM system, clustering provides corresponding improvements of 38% and 22%.
- Performance of each system under the optimized mapping is superior to any of the other solutions. In particular, for the SYS example, the optimized solution is 63% (2.67 times) faster than the shared bus solution, 40% faster than the random solution, and 21% faster than one obtained by simply clustering frequently communicating components. Similar results for the optimized ATM architecture indicate improvements of 44%, 29%, and 9%, respectively.
- The CPU time consumed by the exploration algorithms are dominated by the time spent in performance analysis. In rows 1–3, the times reported are roughly the same, and correspond to the time take for a single invocation of the analysis tool. The CPU time reported in row 4 is twice the CPU time reported in the previous rows, owing to an extra invocation of the analysis tool during the iterative improvement step.
- Using cosimulation as a performance-analysis tool for design space exploration methodology is clearly infeasible, since the large computational cost of a simulation is potentially encountered at every point in the design space. This is borne out by the time taken to conduct HW/SW cosimulation for each system, which, in spite of abstract communication modeling, takes over 2 min for each system (row 5 in each table). In comparison, our performance-analysis technique is over an order of magnitude faster.

TABLE III
EXPERIMENTAL RESULTS—EXAMPLE SYSTEM SYS

Communication Architecture	Performance (cycles)	Speedup	Required Algorithms	CPU time (seconds)
Single shared bus architecture, Trivial Mapping	24654	1.00	Analysis + protocol configuration	10.3
Multiple bus architecture, Random mapping	15314	1.61	Analysis + protocol configuration	11.3
Multiple bus architecture, Clustered mapping (initial solution)	11723	2.10	Analysis + clustering + protocol configuration	12.1
Multiple bus architecture, Optimized mapping	9242	2.67	Analysis + clustering + protocol configuration + iterative improvement	23.5
Ideal architecture (Zero communication time)	4992	4.94	HW/SW co-simulation	138

TABLE IV
EXPERIMENTAL RESULTS—EXAMPLE SYSTEM ATM

Communication Architecture	Performance (cycles)	Speedup	Required Algorithms	CPU time (seconds)
Single shared bus architecture, Trivial Mapping	32328	1.00	Analysis + protocol configuration	6.8
Multiple bus architecture, Random mapping	25593	1.26	Analysis + protocol configuration	7.0
Multiple bus architecture, Clustered mapping (initial solution)	19998	1.62	Analysis + clustering + protocol configuration	6.7
Multiple bus architecture, Optimized mapping	18139	1.78	Analysis + clustering + protocol configuration + iterative improvement	11.8
Ideal architecture (Zero communication time)	9988	3.24	HW/SW co-simulation	134

C. Evaluating Tradeoffs in Communication Architecture Template Selection

Next, we illustrate how the described techniques can be utilized to evaluate tradeoffs in selecting or designing an architectural template for an on-chip communication architecture. For these experiments, we consider an extended version of the system described in Fig. 1. The extended system consists of 16 communicating components, with inter-component communication characteristics defined by stochastic traffic generators (communication requests and interrequest separations followed Gaussian distributions). The architectural template considered was based on a variable-length chain of buses.[2] In this experiment, the previously described techniques were used to optimize the mapping of system communications to templates consisting of 2–8 buses. Performance of the architecture in each case was measured in terms of the total number of cycles required to complete the processing of a fixed number of input stimuli. For each template, three performance numbers are reported. Fig. 9 depicts, for each template, performance under: 1) a random assignment of components to buses; 2) a clustering-based assignment (based on the initial solution constructed in our methodology); and 3) an optimized assignment (derived using the complete methodology described in this paper). For example, when the template consists of two buses, system performance with a random mapping is 48 849 cycles. With a clustering-based assignment, it is 36 525 cycles, a 25% improvement. With the optimized mapping, performance is 30 693 cycles, a 37% improvement. From the graph, we make the following observations.

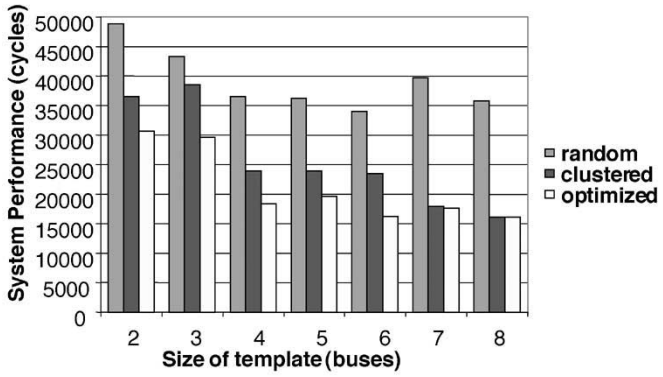

Fig. 9. Tradeoffs in template selection: performance variation with increasing number of buses under different mapping strategies.

- While the general trend indicates improving performance for the optimized mapping with increasing number of buses, there are several cases where forcibly adding more buses has negligible, or even detrimental effect on overall performance (e.g., template sizes 5,7,8). The reason behind this is that addition of a new bus increases parallelism for certain communications on one hand, while, on the other, it introduces additional overhead for com-

[2]These experiments illustrate the evaluation of different instances within a class of templates. However, the approach could also be used to compare performance and evaluate tradeoffs across different template classes, e.g., token rings, bus hierarchies, crossbars, etc.

munications that encounter bus bridges when spanning multiple buses. Note that for these experiments, we forced the optimization algorithm to use all the available buses, rather than isolate and discard buses that do not benefit performance.

- The proposed techniques are capable of generating high-performance solutions across different communication architecture templates. The proposed methodology is successful in generating solutions that have significantly better performance than even those that are based on clustering alone. The optimized communication architectures provide up to 1.45x improvements over the clustering solutions, and up to 2.25x improvements over the random solutions.
- The presented methodology can help designers choose the most appropriate template for a given design. For example, the results indicate that for this system, a template consisting of four buses may provide sufficient performance. Adding more buses brings about incremental performance improvements that may not justify the additional hardware and design cost.

VII. Conclusion

In this paper, we presented new techniques to help designers optimize the mapping of a system's communication requirements to an on-chip communication architecture. We illustrated the issues in automating this process, and presented a methodology and its constituent algorithms for design space exploration. Experimental results conducted to evaluate the effectiveness of the proposed techniques indicate that the methodology performs well, generating solutions that provide significantly better performance over conventional communication architectures.

References

[1] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, pp. 490–504, Apr. 2001.

[2] D. Sylvester and K. Keutzer, "A global wiring paradigm for deep submicron design," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 242–252, Feb. 2000.

[3] L. Benini and G. D. Micheli, "Powering networks on chips," in *Proc. Int. Symp. Syst. Level Synthesis*, 2001, pp. 33–38.

[4] V. Raghunathan, M. B. Srivastava, and R. K. Gupta, "A survey of techniques for energy-efficient on-chip communication," in *Proc. Design Automation Conf.*, June 2003, pp. 900–905.

[5] W. J. Daily and B. Towles, "Route packets not wires: On-chip interconnection networks," in *Proc. Design Automation Conf.*, June 2001, pp. 684–689.

[6] L. Benini and G. D. Micheli, "Networks on chips: A new SoC paradigm," *IEEE Comput.*, vol. 35, pp. 70–78, Jan. 2002.

[7] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, *Specification and Design of Embedded Systems*. Englewood Cliffs, NJ: Prentice Hall, 1994.

[8] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design Test Mag.*, vol. 10, pp. 64–75, Dec. 1993.

[9] T. B. Ismail, M. Abid, and M. Jerraya, "COSMOS: A codesign approach for a communicating system," in *Proc. IEEE Int. Workshop Hardware/Software Codesign*, 1994, pp. 17–24.

[10] P. H. Chou, R. B. Ortega, and G. B. Borriello, "The CHINOOK hardware/software cosynthesis system," in *Proc. Int. Symp. Syst. Level Synthesis*, 1995, pp. 22–27.

[11] F. Balarin, M. Chiodo, H. Hsieh, A. Jureska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, *Hardware-Software CoDesign of Embedded Systems: The POLIS Approach*. Norwell, MA: Kluwer, 1997.

[12] T. Yen and W. Wolf, "Communication synthesis for distributed embedded systems," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 288–294.

[13] J. Daveau, T. B. Ismail, and A. A. Jerraya, "Synthesis of system-level communication by an allocation based approach," in *Proc. Int. Symp. System Level Synthesis*, Sept. 1995, pp. 150–155.

[14] M. Gasteier and M. Glesner, "Bus-based communication synthesis on system level," *ACM Trans. Design Automation Electron. Syst.*, vol. 4, no. 1, pp. 1–11, 1999.

[15] R. B. Ortega and G. Borriello, "Communication synthesis for distributed embedded systems," in *Proc. Int. Conf Computer-Aided Design*, Nov. 1998, pp. 437–444.

[16] A. Pinto, L. P. Carloni, and A. Sangiovanni-Vincentelli, "Constraint-driven communication synthesis," in *Proc. Design Automation Conf.*, June 2002, pp. 783–788.

[17] Sonics Integration Architecture Available: http://www.sonicsinc.com [Online]

[18] R. Yoshimura, K. T. Boon, T. Ogawa, S. Hatanaka, T. Matsuoka, and K. Taniguchi, "DS-CDMA wired bus with simple interconnection topology for parallel processing system LSIs," in *Proc. Int. Solid-State Circuits Conf.*, 2000, pp. 370–371.

[19] K. Lahiri, A. Raghunathan, G. Lakshminarayana, and S. Dey, "Communication architecture tuners: A methodology for the design of high performance communication architectures for system-on-chips," in *Proc. Design Automation Conf.*, June 2000, pp. 513–518.

[20] K. Lahiri, G. Lakshminarayana, and A. Raghunathan, "LOTTERYBUS: A new communication architecture for high-performance system-on-chip designs," in *Proc. Design Automation Conf.*, June 2001, pp. 15–20.

[21] CoreConnect Bus Architecture [Online]. Available: http://www.chips.ibm.com/products/coreconnect/

[22] B. Cordan, "An efficient bus architecture for system-on-a-chip design," in *Proc. Custom Integrated Circuits Conf.*, 1999, pp. 623–626.

[23] AMBA 2.0 Specification [Online]. Available: http://www.arm.com/armtech/AMBA

[24] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, "SPIN: A scalable, packet switched, on-chip micro-network," in *Proc. Design Automation Test Eur. (DATE) Conf.*, 2003, pp. 70–73.

[25] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface based design," in *Proc. Design Automation Conf.*, June 1997, pp. 178–183.

[26] K. Hines and G. Borriello, "Optimizing communication in embedded system cosimulation," in *Proc. Int. Symp. Hardware/Software Codesign*, Mar. 1997, pp. 121–125.

[27] P. Knudsen and J. Madsen, "Integrating communication protocol selection with partitioning in hardware/software codesign," in *Proc. Int. Symp. Syst. Level Synthesis*, Dec. 1998, pp. 111–116.

[28] K. Lahiri, A. Raghunathan, and S. Dey, "System-level performance analysis for designing on-chip communication architectures," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 768–783, June 2001.

[29] P. Lieverse, P. V. D. Wolf, K. Vissers, and E. Deprettere, "A methodology for architecture exploration of heterogeneous signal processing systems," *J. VLSI Signal Process.*, vol. 29, no. 3, 2001.

[30] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 1059–1076, Sept. 2001.

[31] S. Narayanan and D. D. Gajski, "Interfacing incompatible protocols using interface process generation," in *Proc. Design Automation Conf.*, June 1995, pp. 468–473.

[32] P. H. Chou, R. B. Ortega, and G. B. Borriello, "Interface cosynthesis techniques for embedded systems," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 280–287.

[33] J. Oberg, A. Kumar, and A. Hemani, "Grammar-based hardware synthesis of data communication protocols," in *Proc. Int. Symp. Syst. Level Synthesis*, 1996, pp. 14–19.

[34] R. Passerone, J. A. Rowson, and A. Sangiovanni-Vincentelli, "Automatic synthesis of interfaces between incompatible protocols," in *Proc. Design Automation Conf.*, June 1998, pp. 8–13.

[35] J. Smith and G. De Micheli, "Automated composition of hardware components," in *Proc. Design Automation Conf.*, June 1998, pp. 14–19.

[36] K. Anjo, A. Okamura, T. Kajiwara, N. Mizushima, M. Omori, and Y. Kuroda, "NECoBus: A high end SoC bus with a portable and low latency wrapper based interface mechanism," in *Proc. Custom Integrated Circuits Conf.*, May 2002, pp. 315–318.

[37] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya, "Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip," in *Proc. Design Automation Conf.*, June 2001, pp. 518–523.

[38] On-Chip Bus Attributes Version 1 (OCB 1.2x [Online]. Available: http://www.vsi.org/resources/VSIASpecifications.htm

[39] Open Core Protocol International Partnership (OCP-IP) [Online]. Available: http://www.ocpip.org

[40] *OMI Standards Draft, Siemens AG, OMI 324 PI Bus, Rev 0.3d*, 1994.

[41] Crossbow Technologies [Online]. Available: http://www.crossbowip.com

[42] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Technical J.*, vol. 49, pp. 291–307, 1970.

[43] J. Buck, S. Ha, E. A. Lee, and D. D. Masserchmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," *Int. J. Comput. Simul.*, vol. 4, pp. 155–182, 1994.