7

CHAPTER

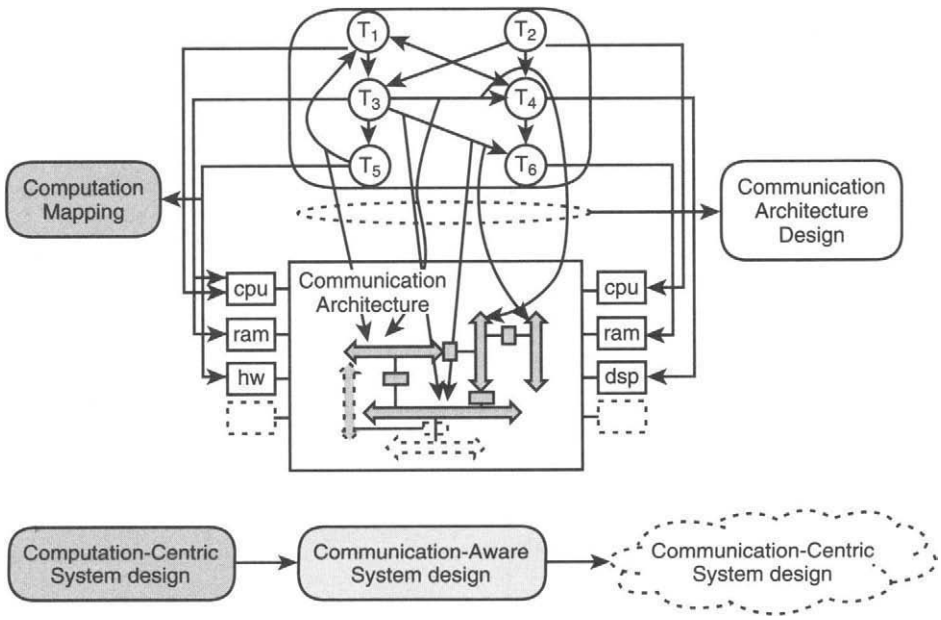
Design of Communication Architectures for High-Performance and Energy-Efficient Systems-on-Chips

Kanishka Lahiri, Sujit Dey, and Anand Raghunathan

7.1 INTRODUCTION

On-chip communication is increasingly being regarded as one of the major hurdles for complex system-on-chip (SoC) designs [274]. As technology scales into the nanometer era, chip-level wiring presents numerous challenges, including large signal propagation delays, high-power dissipation, and increased susceptibility to errors [275,276]. On the other hand, at the system-level, the integration of an increasing number and variety of complex components is resulting in rapid growth in the volume and diversity of on-chip communication traffic, imposing stringent requirements on the underlying on-chip communication resources. As a result, on-chip communication has started to play a significant role in determining several system-wide metrics, including overall system performance, power consumption, and reliability.

The growing importance of on-chip communication adds a new facet to the process of system design, as illustrated in Figure 7-1. Under the traditional notion of system design, a system's functional requirements are refined and mapped onto a well optimized set of computational resources and storage elements. Today, system designers are required to pay increasing attention to the relatively less understood process of mapping a system's communication requirements onto a well-optimized on-chip communication architecture. We have reached a point at which system design practices need to evolve from being computation-centric to being increasingly communication-aware (Fig. 7-1). Recognizing this, new architectures and system design techniques have started to emerge that address



7-1 FIGURE The two dimensions of system design and the evolution of system design methodologies.

on-chip communication. In this chapter, we describe such recent advances, focusing in particular on techniques for designing on-chip communication architectures for high-performance and energy-efficient SoCs.

We address the problem of communication architecture design for SoCs in terms of the following steps: (1) design or selection of an appropriate network topology, which defines the physical sructure of the communication architecture; (2) design of the on-chip communication protocols, which define the mechanisms by which system components exchange data; and (3) communication mapping, which specifies a mapping of system communications (dataflows) to physical paths in the communication architecture.

This chapter describes advances in tools and design methodologies for communication architecture design that are concerned with the above problems. We point out the significant advantages (performance, energy efficiency) that can be obtained by customizing the communication architecture to characteristics of the communication traffic generated by the SoC components. An important requirement for any communication architecture design and customization environment is the availability of automatic techniques for analyzing the impact of

the communication architecture (including the effects of the topology, protocols, and communication mapping) on system performance and power consumption. In this chapter, we provide an overview of alternative approaches for such automatic analysis, describing one such technique in detail. We describe techniques for customizing the design of the communication architecture to exploit the characteristics of the on-chip communication traffic generated by an application.

Several systems exist, in which components may impose time-varying communication requirements on the communication architecture. For such systems, a one-time customization of the communication architecture may often prove ineffective. Hence, techniques are required that can dynamically customize the communication architecture to time-varying requirements. We describe techniques for dynamic (run-time) adaptation of the on-chip communication protocols for achieving high performance in such systems. In this chapter, we also describe techniques for designing communication architectures for energy-efficient systems. We describe two classes of techniques: (1) those that target reducing the energy consumption of the communication architecture itself, and (2) techniques that exploit the communication architecture to improve the energy-efficiency of the entire system. We provide an example of the latter, in which "power aware" on-chip communication protocols are used for system-level power management that aim at improving battery life.

The rest of this chapter is organized as follows. Section 7.2 presents background material on communication architectures, including a survey of typical topologies and communication protocols in use today. Section 7.3 describes automatic system-level analysis techniques for driving communication architecture design. Section 7.4 describes techniques for design space exploration and customization of the communication architecture. Section 7.5 describes techniques that allow adaptive on-chip communication, via dynamic customization of the communication protocols. Section 7.6 describes techniques for communication architecture design for energy-efficient and battery-efficient systems.

7.2 ON-CHIP COMMUNICATION ARCHITECTURES

The on-chip communication architecture refers to a fabric that integrates SoC components and provides a mechanism for the exchange of data and control information between them. The first basis for classifying communication architectures is the network topology, which defines the physical structure of the communication architecture. In general, the topology could range in complexity from a single shared bus to an arbitrarily complex, regular or irregular interconnection of

channels. The second basis for classification is the communication protocols employed by the communication architecture, which specify the exact conventions and logical mechanisms according to which system components communicate over the communication architecture. In addition, these protocols define resource management and arbitration mechanisms for accesses to shared portions of the communication architecture. As mentioned in the previous section, communication mapping refers to the process of mapping the system-level communication events to physical paths in the topology.

In the rest of this section, we survey existing and emerging communication architecture topologies and communication protocols. We also describe recent advances in communication interface design. First, we define some basic terminology that is used in the rest of this chapter.

7.2.1 Terminology

Two kinds of components may be connected to the communication architecture. Masters are system components that can initiate communications (reads/writes). Examples include CPUs, DSPs, DMA controllers, and so on. Slaves are system components (e.g., on-chip memories and passive peripherals) that do not initiate communication transactions by themselves but merely respond to transactions initiated by a master. Bridges or routers allow communication between component pairs that are connected to different communication channels. Bridges may support one-way or two-way communications and may themselves consist of multiple master and slave interfaces. Communication resource management and arbitration algorithms are implemented in centralized or distributed arbiters or controllers. Numerous parameters help define properties of the communication channels and their associated protocols, such as bus widths, burst transfer size, and master priorities.

7.2.2 Communication Architecture Topologies

In this subsection, we describe existing and emerging communication architecture topologies, in order of increasing sophistication and complexity.

Shared Bus

The system bus is the simplest example of a shared communication architecture topology and is commonly found in many commercial SoCs (e.g., PI-bus [277]).

The bus consists of a set of address, data, and control lines shared by a set of masters, that contend among themselves for access to one or more slaves. In its simplest form, a bus arbiter periodically examines accumulated requests from the multiple master interfaces, and grants access to a master using arbitration mechanisms specified by the bus protocol. However, limitations on bus bandwidth (due to increasing load on global bus lines) are forcing system designers to use more advanced topologies.

Hierarchical Bus

In recognition of the problems associated with topologies based on a flat shared bus, topologies consisting of multiple busses have recently begun to appear. This architecture usually consists of several shared busses interconnected by bridges to form a hierarchy. Different levels of the hierarchy correspond to the varying communication bandwidth requirements of SoC components. Commercial examples of such architectures include the AMBA bus architecture (ARM) [278] and the CoreConnect Architecture (IBM) [279]. Different levels of the hierarchy are connected by bridges. Transactions across the bridge involves additional overhead, and, during the transfer, both busses remain inaccessible to other components. However, multiple word communications can proceed across the bridge in a pipelined manner. Such topologies typically offer large throughput improvements over the shared topology, due to decreased load per bus, and the potential for transactions to proceed in parallel on different busses.

Rings

In certain application areas, ring-based communication architectures have been used, such as high-speed ATM switches [280]. In such architectures, each component (master/slave) communicates using a ring interface, which typically implements a token-passing protocol. The advantage of ring-based architectures is that each communication channel is of shorter length (point to point between neighboring components) and therefore can potentially support higher clock speeds.

Packet Switched Fabrics

Recently, the use of more complex topologies has been proposed for SoCs consisting of larger numbers of processing elements. These architectures avoid the use of globally shared busses but rely on switching mechanisms to multiplex communication resources among different master and slave pairs. Although these

architectures have been studied in detail in the context of general-purpose systems such as multiprocessors and network routers, examples of such emerging architectures in the SoC domain include architectures based on crossbars [280], fat-trees [281], octagons [282], and two-dimensional meshes [274,283]. The advantages such architectures are expected to provide include higher on-chip communication bandwidth and predictable electrical properties, due to regularity in the communication architecture topology.

7.2.3 On-Chip Communication Protocols

In this section, we describe existing and emerging communication protocols, focusing on the different types of resource management algorithms employed for determining access rights to shared communication channels.[1] In this regard, these on-chip communication protocols are analogous to those used in the "datalink" layer of wide area networks, which determine access rights to shared LAN resources (e.g., CSMA/CA) [284]. As systems continue to grow in complexity, it is expected that other, higher layer protocol concepts from the domain of large scale networking will become increasingly applicable to communication architectures, such as routing, flow control, and quality of service [285].

Static Priority

This is a commonly employed arbitration technique used for shared bus-based communication architectures [277–279]. In this protocol, a centralized arbiter examines accumulated requests from each master, and grants access to the requesting master that is of highest priority. Transactions may be non-preemptive (i.e., once a transaction of multiple bus words begins, it runs to completion, during which time, all other components requiring access to the bus are forced to wait) or preemptive (lower priority components are forced to relinquish the bus if higher priority components are waiting). Although this protocol can be implemented efficiently, it is not suited to providing fine-grained control over the allocation of on-chip communication bandwidth to different system components.

TDMA

In this type of architecture, the arbitration mechanism is based on a timing wheel with each slot statically reserved for a unique master. Techniques are typically

[1] On-chip communication protocols also feature a variety of other functions, such as support for burst mode transfers, split transactions, multithreaded transactions, and so on.

employed to alleviate the problem of wasted slots (inherent in time-division multiple access [TDMA]-based approaches). One approach is to support a second level of arbitration. For example, the second level can keep track of the last master interface to be granted access via the second level and issue a grant to the next requesting master in a round-robin fashion. A commercial example of a TDMA-based protocol is one offered by Sonics [286,287].

Lottery

In this architecture, a centralized lottery manager accumulates requests for ownership of shared communication resources from one or more masters, each of which is (statically or dynamically) assigned a number of "lottery tickets" [288]. The lottery manager probabilistically chooses one of the contending masters to be the winner of the lottery and grants access to the winner for one or more bus cycles, favoring masters that hold a larger fraction of lottery tickets. This architecture provides fine-grained control over the allocation of communication bandwidth to system components and fast execution (low latencies) for high-priority communications, at the cost of more complex protocol hardware.

CDMA

A code division multiple access (CDMA)-based protocol has been proposed for sharing on-chip communication channels [289], to exploit the same advantages that CDMA has been known to provide in sharing the air medium in wireless networks, namely, resilience to noise/interference, and the ability to support multiple, simultaneous data streams [290]. The proposed architecture features synchronous pseudorandom code generators (a code is assigned to a communicating pair of components), modulation and demodulation circuits at the component bus interfaces, and differential signaling.

Token Passing

Token passing protocols have been used in ring-based architectures [280]. In such protocols, a special data word circulates on the ring, which each interface can recognize as a token. An interface that receives a token is allowed to initiate a transaction. If the interface has no pending request, it forwards the token to its neighbor. If it does have a pending request, it captures the token and reads/writes data from/to the ring, one word per cycle, for a fixed number of cycles. When the transaction completes, the interface releases the token.

7.2.4 Communication Interfaces

An important issue in designing components for use in SoC designs is the use of a consistent communication interface (across different on-chip busses and communication architectures) to facilitate plug-and-play design methodologies. Using such interfaces can provide the additional advantage of freeing the SoC component designer from having to be aware of the details of the communication architecture to which the component will be connected. Hence, this approach facilitates the development of innovative communication architectures that are not constrained by the interfacing requirements of system components that it may potentially need to serve. According to an analogy drawn in Zhu and Malik [291], communication interfaces provide an abstraction that is similar to the instruction set architecture of a microprocessor, whose purpose is to make microarchitectural details transparent to the programmer and, at the same time, facilitate advances in microarchitecture design. Currently, the standards being proposed by various industry consortia in an effort to help realize this goal include interfaces based on VSIA's Virtual Component Interface [292], and the Open Core Protocol [293].

In addition to such standardization initiatives, a large body of research has examined various issues in the design and implementation of on-chip communication interfaces, including techniques for interface modeling, simulation, and refinement (or synthesis). These techniques aim at providing ways of exploring different interfaces, and converting high-level protocol descriptions into efficient hardware implementations. Techniques for model refinement of communication interfaces are described in refs. 294 and 295, and techniques for synthesizing interfaces between components that feature incompatible protocols are described in ref. 296. Generic module interfaces ("wrappers") are described in ref. 297 that facilitate simulation of a complex SoC, while providing for the flexibility of mixing multiple levels of abstraction during simulation. Interface synthesis using these wrappers is described in ref. 298.

7.3 SYSTEM-LEVEL ANALYSIS FOR DESIGNING COMMUNICATION ARCHITECTURES

In recognition of the growing role of on-chip communication, recent work has addressed the development of system-level analysis techniques for estimating the

impact of the communication architecture on overall system performance and power consumption. In this section, we describe such techniques, which aim at providing automatic support to drive the process of communication architecture selection, design, or optimization. These techniques can be broadly divided into the following categories.

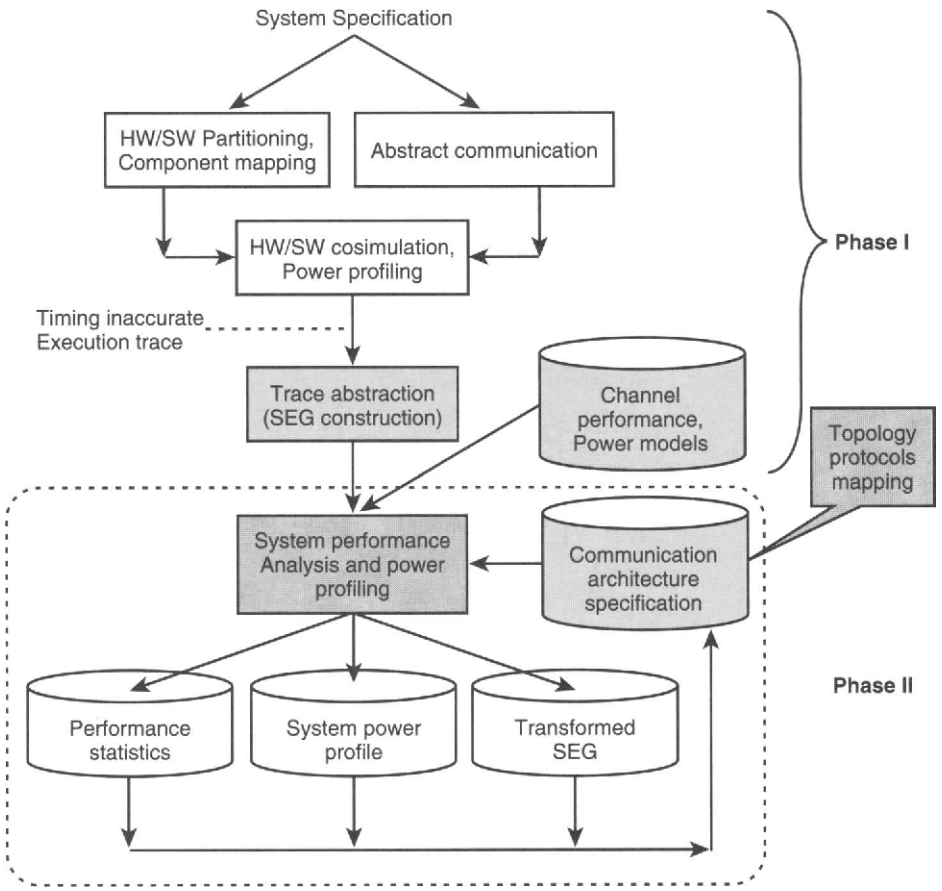
1. System simulation-based techniques. In this class of techniques, the effects of the communication architecture are incorporated by developing suitable simulation models of the communication architecture topology and protocols [291,294,299–303]. Techniques that rely on simulation of the complete system are typically not feasible for exploring large design spaces, such as those offered by existing and emerging communication architectures. Simulation speed up is typically achieved by using abstract models of system components and the communication architecture. However, the use of such models typically trade off accuracy for efficiency.
2. Static estimation-based techniques. This class of techniques makes use of "static" models of the communication latency between system components [304–308] or static characterizations of the power consumption of system tasks, using simple power models for alternate implementations, and intertask communication (e.g., refs. 309 and 310). These techniques often assume systems in which the computations and communications can be statically scheduled. For many systems, using such techniques could result in inaccurate performance and power estimates, since they usually ignore or make simplifying assumptions regarding the occurrence of dynamic effects (e.g., waiting due to bus contention).
3. Trace-based techniques. In this class of techniques, the effects of the communication architecture are incorporated into system-level analysis using a trace-based approach. These techniques derive accuracy from an initial detailed simulation and computational efficiency from fast processing of execution traces. In the next section, we describe one of these techniques in detail. A similar approach has been taken by Lieverse et al. [311,312], in which trace transformation techniques are used for communication refinement of signal processing applications modeled using Kahn process networks. Givargis et al. [313] also use trace-based analysis: although they do consider the effects of the communication architecture, their work is aimed mainly at SoC parameter tuning (e.g., cache parameters, buffer sizes, bus widths), rather than analyzing and exploring different communication architectures.

7.3.1 Trace-Based Analysis of Communication Architectures

In this section we describe a trace-based technique for fast and accurate system-level performance analysis and power profiling, to drive the design of the communication architecture [314,315]. The analysis technique enables evaluation of general communication architecture for use in complex SoC designs. The techniques provide the system designer with the freedom to select a network topology, communication protocols, and communication mapping. The topology could range from a single shared bus to an arbitrary interconnection of dedicated and shared communication channels. For each channel, the designer can select and configure the communication protocols and specify values of channel related parameters. The designer can also specify a communication mapping: i.e., an arbitrary mapping of system communications to paths in the specified topology. For a given specification of the communication architecture, the designer can evaluate system performance, the system power consumption profile, and various statistics regarding the usage of communication architecture resources.

The complete methodology is shown in Figure 7-2. The first phase of this methodology constitutes a preprocessing step in which simulation of the entire system is performed, without considering the detailed effects of the communication architecture. In this phase, communication is modeled in an abstract manner, through the exchange of events or tokens. The output of this step is a timing-inaccurate system execution trace. The second phase (enclosed within the dotted box in Fig. 7-2) is responsible for analyzing system performance and generating a system power consumption profile over time, including the effects of a selected communication architecture. The second phase operates on a symbolic execution graph (SEG), which is constructed from the traces obtained in the first phase, and captures the computations, communications, and synchronizations observed during simulation of the entire system. The system designer specifies the communication architecture, in terms of the network topology, communication protocols, and communication mappings. Analysis algorithms manipulate the SEG to incorporate effects of the selected communication architecture. The outputs, all of which account for the effects of the communication architecture, include (1) a transformed version of the SEG, (2) system performance estimates, (3) communication architecture usage statistics, such as channel utilization, resource conflicts, and so on, (4) system critical path (or paths), and (5) a profile of the system power consumption over time, as well as power profiles of individual components.

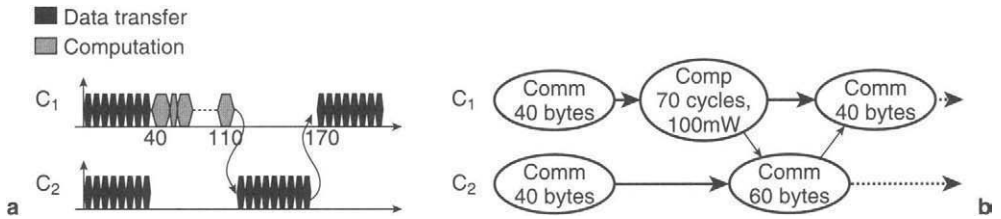
The advantage of this approach is that the time-consuming simulation step need not be performed iteratively, and alternative communication architectures



7-2 FIGURE Trace-based analysis methodology for communication architecture design.

can be evaluated using the second phase of the methodology alone. The second phase, being fast and accurate (for reasons discussed later in this section), can provide the designer with important feedback regarding the impact of the communication architecture within very short analysis times, allowing efficient evaluation of many alternative architectures.

In the following subsections, we illustrate each of the shaded steps of Figure 7-2. These include (1) abstracting information from the simulation trace and



7-3 FIGURE Trace abstraction via SEG construction. (a) Traces generated by HW/SW co-simulation. (b) Corresponding SEG.

constructing the SEG, and (2) analyzing the system performance under the selected communication architecture.

Trace Abstraction

Simulation traces can be highly complex and detailed, making them undesirable as a starting point for analysis. Hence, only information that is necessary and sufficient to perform accurate analysis and to incorporate the effects of the communication architecture is extracted from the traces. The extraction step selectively omits unnecessary details (such as values of internal variables, communicated data, and so on). In addition, the extraction procedure groups contiguous computation and communication events into clusters, to construct an SEG. The SEG, being an abstract and compact representation of the traces, can be efficiently analyzed to obtain a variety of useful information.

Example 1. *Figure 7-3a shows a set of execution traces as two components, C_1 and C_2; each executes computations, communication with slaves, and synchronization between each other. Figure 7-3b illustrates the SEG that corresponds to the traces of Figure 7-3a. The graph has two types of vertices—computation and communication—and a set of directed edges representing timing dependencies, which arise due to the sequential component control-flow, or due to intercomponent synchronization and communication. Note that the SEG is acyclic because it is constructed from simulation traces in which all dependencies have been unrolled in time.*

The example illustrates that the SEG accurately captures the dynamic control-flow behavior of the HW/SW partitioned and mapped system. However, it should

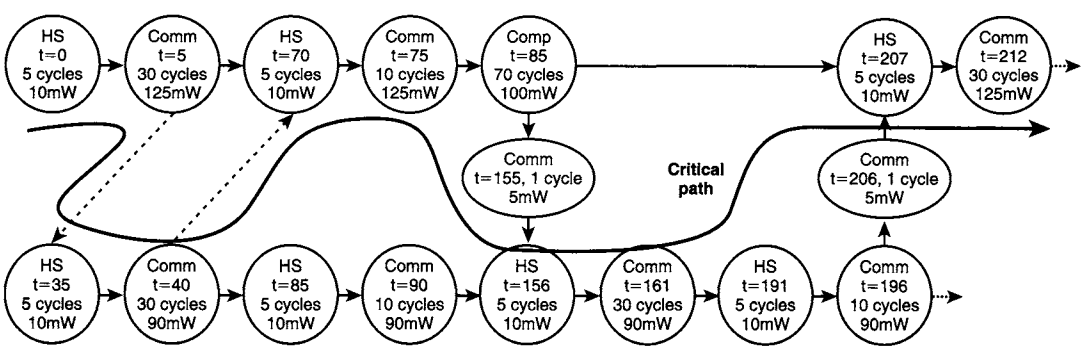
be noted that the SEG is "timing-inaccurate," since communication between system components was modeled in an abstract manner during simulation for trace collection.

We next describe how the analysis techniques incorporate the effects of a communication architecture to provide the required accuracy.

Analyzing Abstract Traces

The analysis technique is based on manipulating the SEG to incorporate the effects of the selected communication architecture, which includes a specification of the network topology, protocols, and communication mapping. The analysis technique makes use of designer-provided information and executes a pass of the SEG, adding new vertices and edges, splitting existing vertices, and calculating time stamps and power estimates for each vertex. Next, we illustrate the execution of the analysis algorithm with an example. We consider the processing of the SEG of Figure 7-3b, in order to incorporate the effects of a communication architecture consisting of a static priority-based shared bus. Note that although we use a shared bus for ease of explanation, the technique is applicable to arbitrary topologies and protocols.

Example 2. *Figure 7-4 illustrates the SEG that results from the execution of the analysis algorithms. We indicate example transformations that the algorithms introduce*



7-4 FIGURE Transformed SEG after incorporating effects of a priority-based shared bus.

while incorporating effects of the communication architecture. Intrinsic protocol overheads, such as a connection setup, or handshaking, precede every data transfer. This is incorporated by the insertion of new vertices of type HS in the SEG, prior to each communication vertex. The protocol specifies a maximum permissible burst size. Hence, some communication vertices are split into multiple vertices, none of which exceed the burst size (30 cycles, in this example). The SEG of Figure 7-3b contains communication vertices from different masters that overlap. Since the communication architecture in this example is a shared bus, this represents a conflict, and one of the components must wait. The transformation that incorporates this conflict is shown in Figure 7-4, in which the analysis tool "grants" access to C_1, in accordance with designer-specified priorities ($C_1 > C_2$). As a result, C_2 waits, and an additional dependency (dotted arrow) is introduced to enforce the mutually exclusive nature of access to the shared bus. SEG vertices are annotated with time stamps, which are obtained by examining time stamps of predecessor vertices and calculating, or looking up, vertex execution times. For communication vertices, the execution times are calculated using provided raw channel bandwidth and latency information. Lookup techniques are used for HS vertices. For the computation vertices, the execution times are preserved from the original SEG. The analysis algorithms also annotate each vertex in the transformed SEG with average power estimates. For the communication and HS vertices, analytical power models of the communication channels, master and slave interfaces, are used, whereas for the computation vertices, again, the estimates are obtained from the original SEG.

From the resulting SEG, various useful statistics can be derived, to provide feedback to the designer of the communication architecture. For example, by examining time stamps on specific SEG vertices, system performance metrics can be calculated (time taken to complete a specific task, number of missed deadlines, and so on). As shown in Figure 7-4, the technique allows generation of the system critical path(s). Also, since each vertex is annotated with a power estimate, the technique can efficiently generate the system power consumption profile versus time under the selected communication architecture. Statistics regarding the usage of communication architecture resources, such as channel utilization, conflicts, and so on, are provided to the designer and can be used to guide automatic design space exploration techniques.

Case Study: Exploration of Communication Architecture Parameters for a TCP/IP Subsystem

The trace-based analysis technique has been applied to a range of system designs and communication architectures. To illustrate the utility of the analysis technique, we present an example in which an exhaustive search of the design space

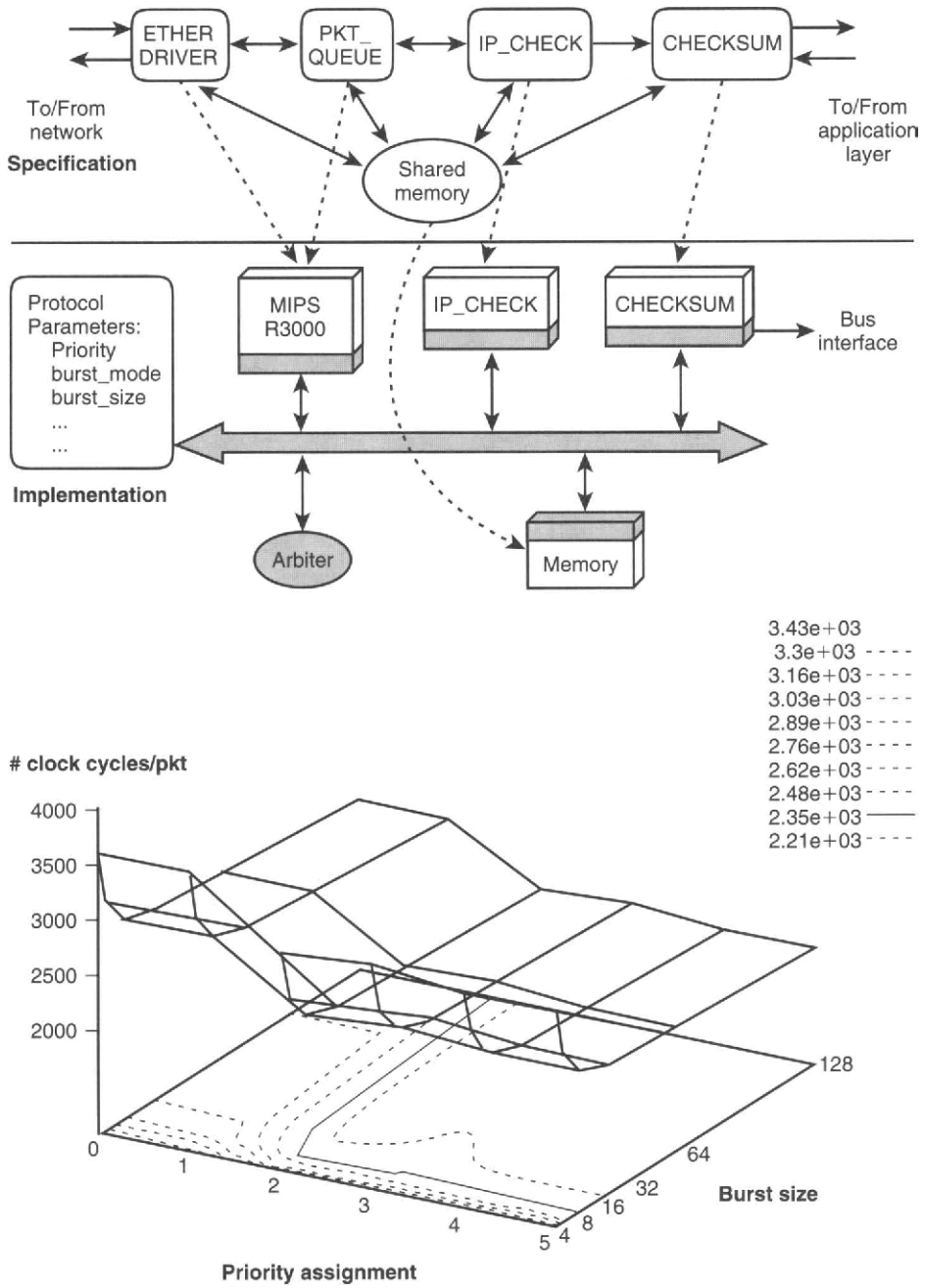
of a particular communication architecture is conducted to obtain a configuration of the communication architecture that maximizes system performance.

Example 3. *Consider a system consisting of three components and a shared memory, which together implement a subsystem of the TCP/IP protocol, relating to computation and verification of the TCP checksum. The parameter space of the communication architecture consists of all possible priority assignments and available burst sizes. Results of the analysis for each configuration are presented in Figure 7-5, which shows the performance of the TCP/IP system when one is processing 10 packets of size 512 bytes under 36 different design points. The best performance is obtained when the burst size is 128 and priorities are assigned so the three components Ether_Driver, IP_Chk, and Chksum are in descending order of priority. The curves in the x-yplane are isoperformance contours. The system performance is observed to vary between extremes of 2077 cycles and 3570 cycles. The entire space exploration experiment took less than 1 second of CPU time.*

The above example illustrates that (1) it is possible to perform thorough and fast exploration of the communication architecture design space using the described trace-based techniques, and (2) obtaining the optimum communication architecture parameters is a nontrivial problem, calling for design space exploration algorithms for customization of the communication architecture. In addition to demonstrating over two orders of magnitude improvements in efficiency over simulation-based approaches, this technique results in imperceptible loss of accuracy, with respect to both system performance (on average 2%) and generated power profiles (on average 4%).

Summary

In this section, we discussed techniques that help to analyze the impact of communication architecture design choices on overall system performance and power consumption. We described a technique providing designers with accuracy that is comparable to that of complete system simulation and yet is orders of magnitude faster. The technique derives its efficiency from the fact that it operates on a highly abstract representation of the system execution traces. The technique derives its accuracy from two factors. First, the traces are derived from simulation, which ensures that control flow (loop iterations, branches taken) is fully determined and is available during analysis. Second, since communications are not isolated from the rest of the system (computations and synchronizations), it is possible to account for the indirect effects that the communication architecture has on the timing and power profile of the system components. These two features make the analysis technique a useful addition to communication architecture design environments.



7-5 FIGURE

Design space exploration of a bus-based communication architecture. (a) TCP/IP subsystem. (b) Performance versus priority and burst size for TCP/IP subsystem.

7.4 DESIGN SPACE EXPLORATION FOR CUSTOMIZING COMMUNICATION ARCHITECTURES

The design of a communication architecture calls for careful attention to several issues in order to meet all the desired design goals. These include (1) selection of an appropriate network topology; (2) selection of appropriate communication protocols, along with careful configuration of protocol parameters; and (3) optimization of the mapping of the system communications to physical paths in the topology. As communication architectures grow in complexity, each of these steps potentially presents system designers with a large number of alternatives. In addition (as we shall see later in this section) they may interact, leading to tradeoffs that are difficult to identify or evaluate without automatic design space exploration tools. In this section, we describe techniques for exploring this design space, focusing on techniques for customizing the communication architecture to best suit the characteristics of the communication traffic generated by an application [316].

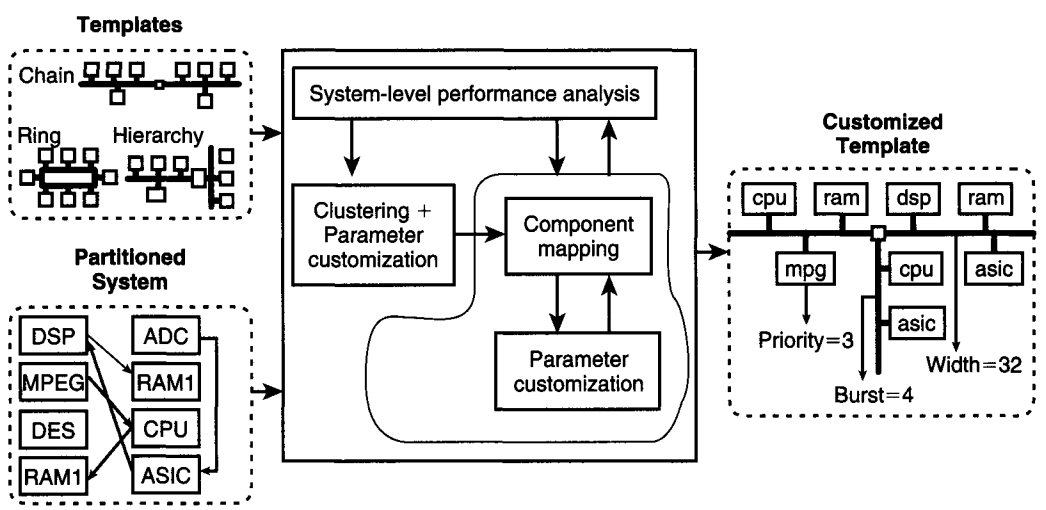
7.4.1 Communication Architecture Templates

Existing techniques for design space exploration and customization of communication architectures mostly deal with generation of network topologies [304,305,308,317,318]. Although topology selection is a critical step in the design of a communication architecture, we believe that future communication architecture designs will be based on one of several architectural "templates" that can be customized for the needs of an application. A few such templates have started to become available from interconnect IP providers, examples of which include hierarchical busses [278], shared micronetworks [286], and mesh-based networks [283]. Modern templates often feature standard topologies, while providing designer configurable parameters for the communication channels (e.g., bus widths) and protocols (e.g., priorities, split transactions, and so on). As available templates increase in complexity and configurability, the parameter space of the communication architecture, together with the number of alternative ways in which the system's communications can be mapped to the template, will make exhaustive design space exploration computationally intractable. As shown in this section, even for relatively simple communication architectures (e.g., consisting of just two busses), naive use of the communication architecture could result in poor system performance. Hence, it is crucial that the underlying communication architecture template be optimized for, or customized to, the specific characteristics of the on-chip communication traffic generated by the application. Since the

design space offered by such templates is complex, efficient methodologies are required for template customization. In the next subsections, details of such a methodology are presented.

7.4.2 Communication Architecture Template Customization

Figure 7-6 presents a design space exploration and optimization technique for communication architecture customization. The methodology takes as inputs (1) a partitioned and mapped system, and (2) a communication architecture template, consisting of a fixed network topology, configurable protocol parameters, and support for arbitrary mapping of system communications to paths in the topology. The methodology uses efficient algorithms to generate (1) an optimized mapping of the system's communications onto the communication architecture topology, and (2) optimized values for communication protocol/channel parameters. The methodology consists of two interacting parts: (1) a fast and accurate performance analysis tool, and (2) a design space exploration framework. For a given point in the communication architecture design space, the analysis tool efficiently estimates system performance, profiles intercomponent communication



7-6 FIGURE Design space exploration and customization of communication architectures.

traffic, and provides accurate estimates of contentions for shared resources. This information is used to guide the design space exploration algorithms for communication architecture customization.

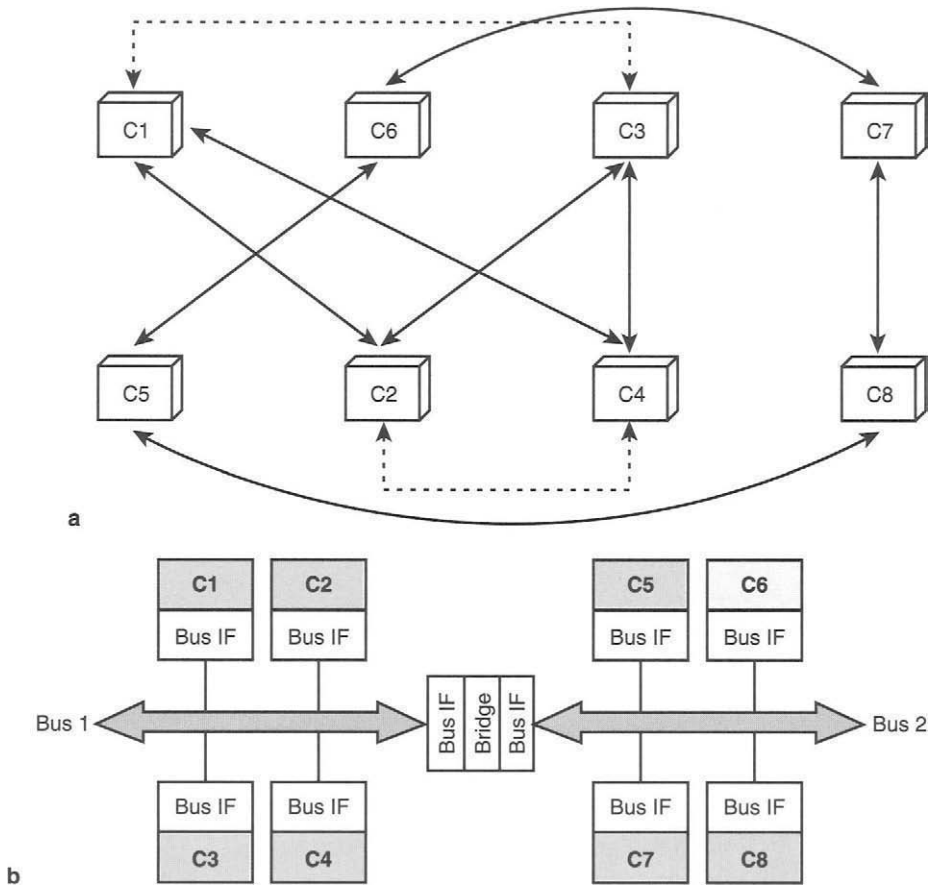
In the next subsections, we present details of two techniques used by the methodology: a clustering-based initial solution construction and a subsequent design space exploration procedure. We use examples to illustrate some of the issues that are taken into consideration by the different algorithms.

Clustering-Based Initial Solution Construction

The first step toward customizing the communication architecture for the application uses the principle of clustering frequently communicating components closer in the topology. The intuition is that clustering reduces the number of communications that undergo high-latency, multi-hop communications (e.g., those spanning multiple busses connected by bridges). A list of components is constructed in decreasing order of communication bandwidth requirements, based on statistics obtained via performance analysis. Each component from the list is considered in turn and is mapped to a channel in the template topology. Metrics are used to estimate the "affinity" of the component with different channels, based on measurements of the volume of traffic it exchanges with other (already assigned) components. Wherever possible, components are assigned to channels with which they have maximum affinity, subject to constraints on the maximum number of components per channel.

Example 4. *Figure 7-7a shows a system consisting of a set of masters that execute computation tasks and exchange data and control as indicated. (Solid lines indicate data transfer, and dotted lines indicate synchronization.) Application of the clustering-based mapping strategy to a template consisting of two shared busses connected by a bridge results in Arch1 (Fig. 7-7b). In this mapping, communications between C_1 and C_2 are mapped to Bus1, and those between C_5 and C_6 are mapped to Bus2. This mapping minimizes the volume of transactions that go across the bridge, by clustering frequently communicating components on the same bus. Figure 7-8a symbolically represents Arch2, a naive mapping to the template (chosen at random). In Arch2, communications between C_1 and C_4 are mapped to a path of busses: Bus1 → Bus2. Table 7-1 reports system performance under each of these architectures. From the table, we observe that Arch1 is 24% faster than Arch2.*

The above example illustrates that better system performance can be obtained by customizing the communication mapping to the traffic characteristics of the application. In this case, simple clustering of frequently communicating components in the communication architecture helped improve system performance



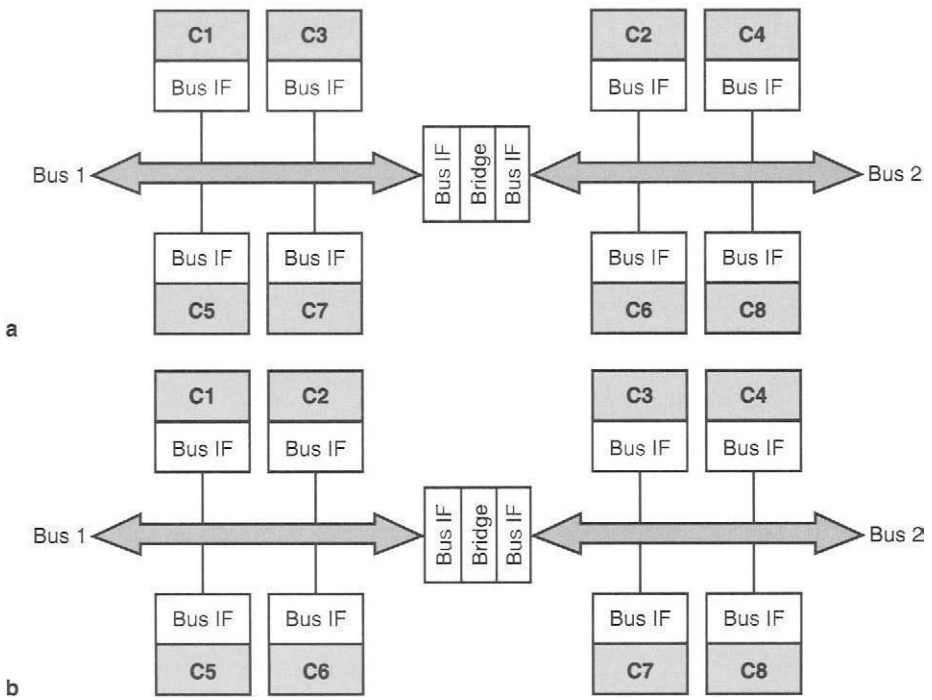
7-7 FIGURE

Example system of communicating components. (a) Logical view of intercomponent communication. (b) Candidate communication mapping to the template topology–Arch1.

compared with a random assignment. However, as we shall see next, this approach does not necessarily yield the best solution.

Analysis-Guided Iterative Refinement

The second step of the exploration framework aims at improving system performance beyond what can be achieved via simple clustering. This step starts with an initial solution (consisting of a communication mapping and a configuration of the protocol parameters for each channel) and uses efficient exploration



7-8 FIGURE Alternative mappings for the example system. (a) Arch2. (b) Arch3.

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

7-1 TABLE Performance for Different Communication Mapping Alternatives.

algorithms that consider the benefits of moving components from currently mapped channels to other channels, while simultaneously reconfiguring the protocol parameters. The exploration technique is based on a well-known iterative improvement strategy for partitioning that maximizes the cumulative gain of a sequence of moves [319]. The procedure employs efficient analytical techniques to estimate the potential gain of moving a component from its currently mapped channel to another, which takes into account communication characteristics of the component, properties of the channels, and their utilization levels, as well as resource conflicts. To illustrate the advantage of the iterative refinement step, we return to the previous example.

Example 5. *Consider again the example system of Figure 7-7a. Applying the exploration procedure with Arch1 as a starting point results in Arch3 (shown in Fig. 7-8b). Analysis of Arch3 indicates that the system completes its execution in 9242 cycles, an improvement of 27% over Arch1 (which was deemed optimal from a clustering viewpoint). Arch3 consists of a mapping that is superior to Arch1 because it is conscious not only of the volume of data communicated between components but also the timing and concurrency of different communications. In particular, it separates components that have largely overlapping communication lifetimes (C_1 from C_3 and C_5 from C_7), resulting in fewer resource conflicts and hence enhanced concurrency at the system level. The price paid is increased traffic that spans multiple busses, since component C_1 communicates with component C_4 and C_3 with C_2 (Fig. 7-7a).*

The example illustrates that in order to derive the best communication mapping, it is important to consider all the characteristics of the application's communication traffic, including its temporal properties. The exploration procedure described above is capable of accounting for such characteristics and outperforms techniques that are based solely on the volume of data exchanged by components.

Interdependency Between Communication Mapping and Protocol Customization

We next illustrate how the problems of optimizing the communication mapping and optimizing the parameters of the communication protocols are interdependent. In other words, a strategy that first optimizes the communication mapping, and then optimizes the protocol parameters could result in significantly suboptimal system performance. The methodology of Figure 7-6 addresses this interdependence by interleaving refinement of protocol parameters with refinements to the communication mapping. To illustrate the importance of considering this interdependence, we return to the example of Figure 7-7 and evaluate an exploration strategy whereby the communication mapping is first optimized, assuming

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

TABLE 7-2 Effect of Protocol Parameters on Design Space Exploration.

a fixed set of protocol parameters, and the protocol parameters are subsequently tuned.

Example 6. *Consider again the example system of Figure 7-7. Each bus in the template is a static priority based shared bus supporting burst mode transfers. The protocol parameters (the maximum burst transfer size, bus access priorities) were fixed during exploration of alternative communication mappings. After obtaining a solution for the communication mapping, the protocol parameters were optimized as a separate step. This yields Arch1, the same solution as that obtained via clustering. This solution, as reported earlier, is 27% worse than the best solution. In this approach, Arch3, the best solution, gets overlooked. This is because when the exploration algorithms consider the communication mapping of Arch3 with the a priori fixed selection of protocol parameters, performance estimates show that the system takes 12,504 cycles to complete (indicated in Table 7-2 by Arch3-subopt). Since this represents a performance degradation of of 6.6% compared with Arch1, the communication mapping of Arch3 is discarded in favor of that of Arch1. In contrast, when communication mapping and protocol customization are performed simultaneously (using the methodology of Fig. 7-6), the optimal architecture (Arch3) is obtained.*

The above example demonstrates that it is important to consider and exploit the interdependence between the protocol parameters and the communication mapping while customizing a communication architecture template.

Summary

In this section, we described a design space exploration strategy for customizing a communication architecture template to best suit the communication traffic characteristics of an application. The techniques described demonstrate the advantages of customizing the template architecture while taking into account spatial locality of the communication traffic, temporal properties of the communication traffic, and the interdependence between the problems of protocol customization and communication mapping. These techniques lead to the design of

customized communication architectures that are better suited to handling the requirements imposed by an application, resulting in significant gains in overall performance.

7.5 ADAPTIVE COMMUNICATION ARCHITECTURES

The previous section described techniques for statically customizing the communication architecture toward specific characteristics of the on-chip communication traffic generated by an application. In several systems, the traffic generated by SoC components could exhibit significant temporal variation in the requirements they impose on the communication architecture. This may occur due to variability in the exact functions being executed by the system components or the properties of the data being processed. For such systems, a mere static customization of the communication architecture (using approaches such as the ones described in the previous section) may prove insufficient. In other words, the communication architecture should be capable of detecting variations in the communication requirements of system components over time and of adapting the underlying communication resources to best meet those requirements. In recognition of this need, commercial bus architectures have recently started provisioning for the ability to configure communication protocol parameters dynamically. Examples include programmable arbiters in the AMBA bus [278] and dynamically variable slot reservations in the Sonics micronetwork [286]. Although hardware support for dynamic configurability is becoming available, there is a lack of established methodologies that enable designers to take advantage of such configurability. In this section, we describe the use of communication architecture tuners, a technique that enables the on-chip communication protocols to recognize and adapt to varying requirements of the application's traffic [320].

7.5.1 Communication Architecture Tuners

Communication architecture tuners (CATs) constitute a layer of circuitry that surrounds a communication architecture topology and provides mechanisms for communication protocols to adapt to run-time variations in the communication needs of system components. The CATs monitor the internal state of each component, analyze the generated communication transactions, and "predict" the relative importance of communication transactions in terms of their impact on system-level performance metrics. The results of the prediction are used to

configure available communication protocol parameters to best suit each component's changing communication needs[2]. For example, more performance-critical data transfers may be handled differently, leading to lower communication latencies. The CATs approach yields improved utilization of the on-chip communication bandwidth and consequently significant improvements in overall system performance (e.g., the ability to meet real-time deadlines).

In this section, we first describe the overall architecture of a CAT-based system, along with basics of the tuner circuitry that needs to be added to SoC components. We then illustrate the functioning of a CAT with an example and finally provide an overview of a methodology for designing CAT-based communication architectures.

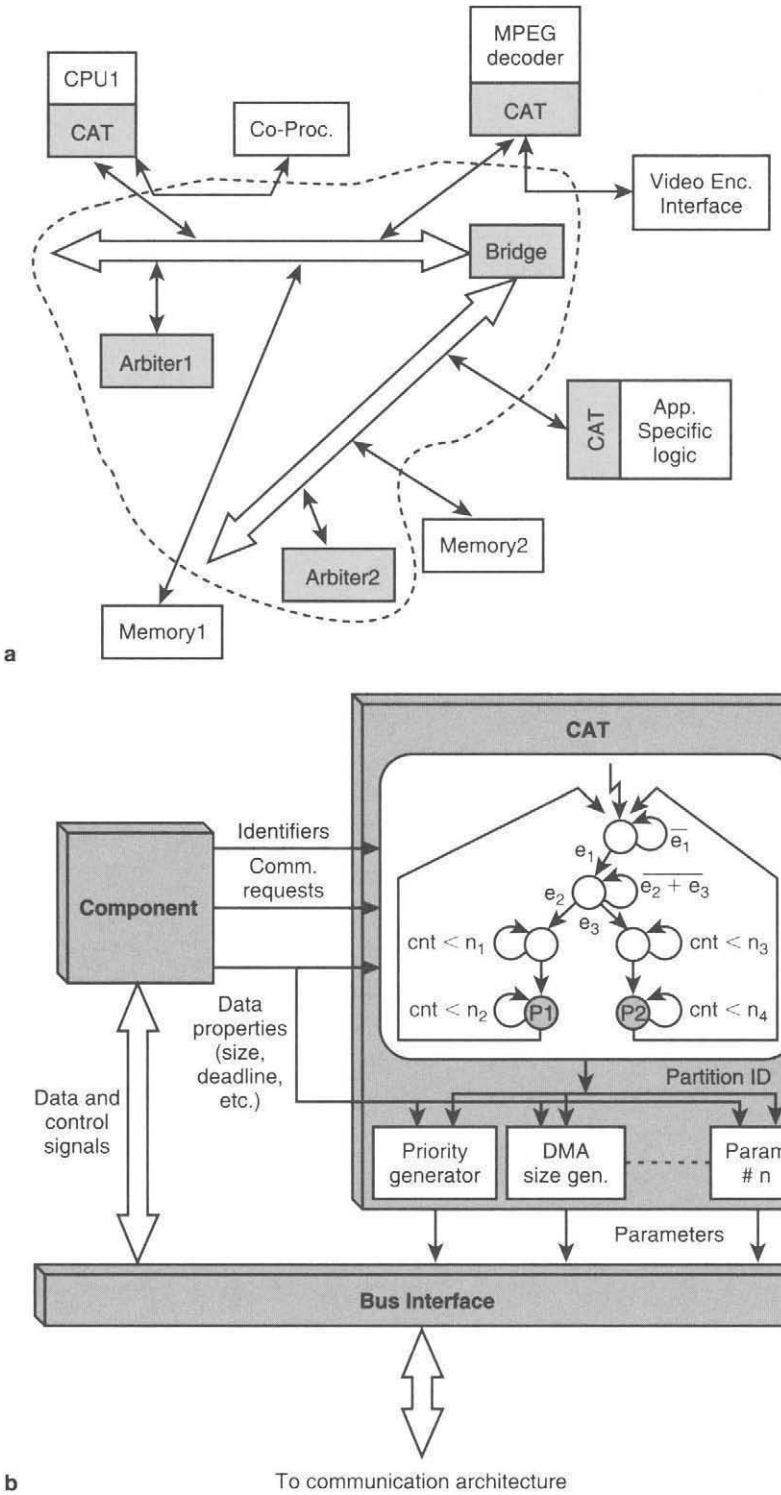
CAT-Based Communication Architectures

A typical system with a CAT-based communication architecture is shown in Figure 7-9a. The system contains several components, including a processor core, memories, and peripherals. The topology (consisting of a combination of shared and dedicated channels) is enclosed in the dotted boundary. The portions of the system that are added or modified as a result of the CATs methodology are shown shaded in Figure 7-9a. A detailed view of a component with a CAT is shown in Figure 7-9b. The CAT consists of a partition detector circuit and parameter lookup tables (LUTs).

1. Partition detector. A communication partition is a subset of the communication transaction instances generated by the component during system execution. For each component, the methodology defines a set of partitions[3] and the conditions that need to be satisfied by a communication instance for it to be classified as belonging to a specific partition. The partition detector circuits monitor and analyze the following information generated by the component:
   - Identifiers: identifiers t_1, t_2, ... are emitted by a component when the history of its control flow satisfies associated expressions defined over a

[2] Note that the addition of CATs does not change the topology of the communication architecture but allows dynamic variation of the parameters governing the on-chip communication protocols.

[3] The term "partition" is often used to refer to the decomposition of a set into a set of disjoint subsets, as well as to one of the individual subsets. In this section, we used the term "partition" with the latter connotation.



7-9 FIGURE

(a) An example SoC with a CAT-based communication architecture. (b) Detailed view of a master component enhanced with a CAT.

set of control flow events $e_1 \ldots e_n$. The component specification is enhanced to generate these identifiers purely for the purpose of the CAT.

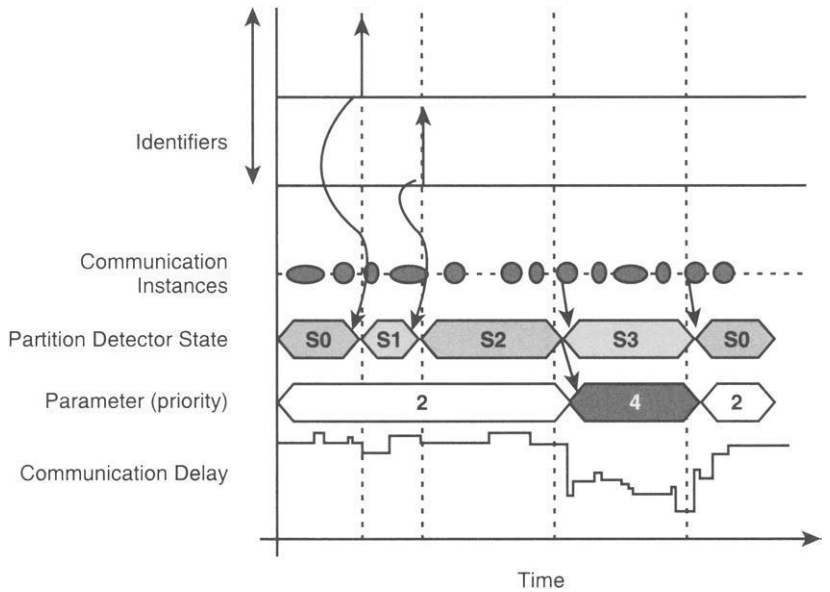
- The communication requests that are generated by the component.
- Any other application-specific properties of the communicated data (e.g., fields in the data that indicate its relative importance).

The partition detector uses heuristic techniques based on observing identifiers (t_1, t_2, . . .), communication instances (c_1, c_2, . . .), and data properties to classify communication instances into partitions. These heuristics take the form of regular expressions. For example, the expressions $t_1 \cdot c^4$ and $t_1 \cdot c^8$ when associated with a specific partition indicate that the fourth to seventh communication instances that are generated following the identifier event t_1 will be classified as belonging to that partition.

2. Parameter LUTs. These LUTs contain precomputed values for communication protocol parameters (e.g., priorities, burst sizes, and so on), which are indexed by the output of the partition detector, and other application-specific data properties specified by the system designer. These parameter values are sent to the communication architecture, which may result in a change in the performance offered by the communication architecture to the SoC component.

Example 7. *The functioning of a CAT-based communication architecture is illustrated using symbolic waveforms in Figure 7-10. The first two waveforms represent identifiers generated by the component. The third waveform represents the communication instances generated by the component. The fourth waveform shows the state of the partition detector circuit. The state of the partition detector circuit changes first from S0 to S1, and later from S1 to S2, in reaction to the identifiers generated by the component. The fourth communication instance generated by the component after the partition detector reaches state S2 causes it to transition into state S3. All communication instances that occur when the partition detector FSM is in state S3 are classified as belonging to partition CP. The fifth waveform shows the output of the priority generation circuit, which assigns a priority level of 2 to all communication instances that belong to partition CP. This increase in priority leads to a decrease in the latency associated with the communication instances that belong to partition CP.*

The above example illustrates how CATs provide the mechanisms for run-time detection of time-varying communication requirements and appropriate adaptation of the communication protocols.

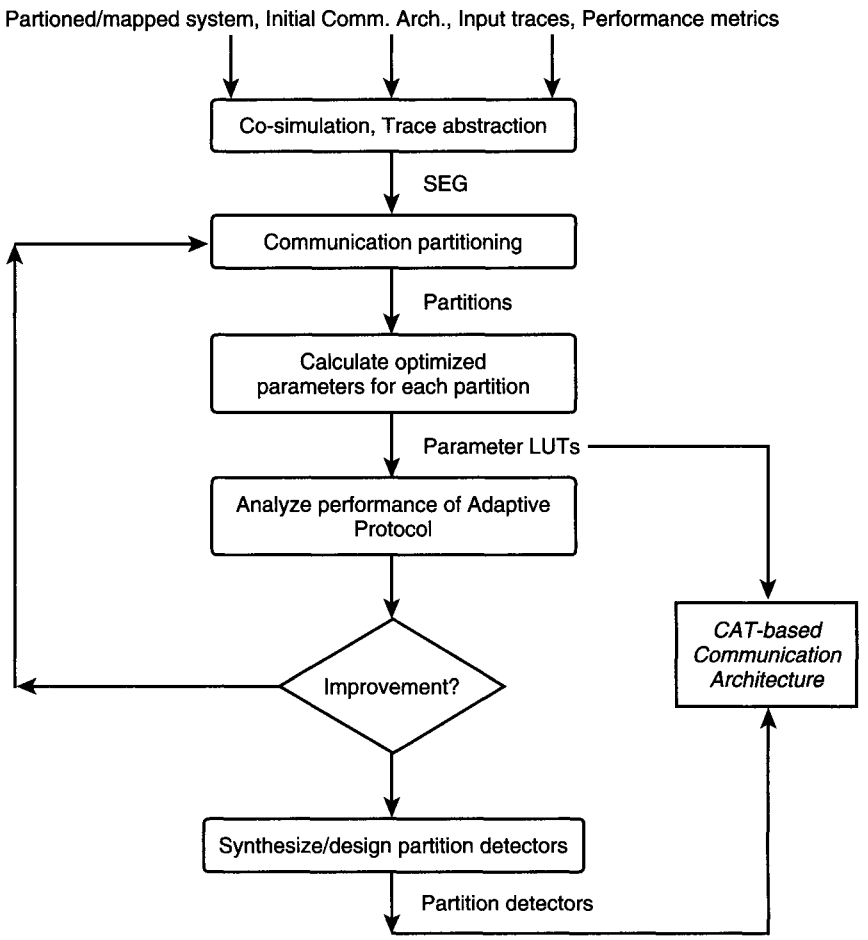


7-10 FIGURE Symbolic illustration of the execution of a CAT-enhanced component.

Design Methodology for CAT-Based Systems

The overall methodology for designing CAT-based communication architectures is shown in Figure 7-11. The methodology takes as inputs a simulatable partitioned/mapped system description, a selected network topology and protocols, typical input stimuli, and a set of performance objectives (e.g., time taken to complete a specific task, number of real-time deadlines met or missed). The output is a set of optimized communication protocols for the target system. From a hardware point of view, the system is enhanced through the addition of CATs wherever necessary and through the modification of the controllers/arbiters for the various channels in the communication architecture.

In step 1, simulation of the partitioned/mapped system is carried out, with a statically configured communication architecture. The resulting traces are converted into a compact representation (SEG) that facilitates fast analysis. (Details of the SEG and the techniques used to analyze it have been described in Section 7.3.) In step 2,the SEG communications are classified into a set of distinct "communication partitions," each partition consisting of communications that impose similar performance requirements on the communication architecture. Communications belonging to a particular partition may benefit from a unique



7-11 FIGURE Methodology for CAT-based communication architecture design.

specification of the communication protocol parameters. In step 3, the communication partitions are analyzed to determine an optimized set of parameters for each communication partition. The output of step 3 is a set of lookup tables that map partitions to parameter values. Step 4 reevaluates system performance, taking into account the parameter assignments derived in step 3. Steps 1 to 4 are repeated until no further performance improvement is obtained.

Note that the off-line determination of communication partitions (step 2) is relatively easy, due to the availability of the complete execution trace. However, since knowledge of future events is not available at run time, dynamic partition

detection is performed using heuristic strategies, based on component control-flow history and data properties. Step 5 deals with generation of such partition detector circuits. Parameters for the detected partitions are obtained from LUTs generated in Step 3. These LUTs together with the partition detector circuits constitute the CAT hardware that is used to enhance the communication architecture.

It bears mentioning that the notion of associating protocol parameters to communication partitions is quite general. For example, a static priority-based protocol represents a special case of such an approach, wherein all communications generated by a component belong to a single partition, and there are as many partitions as components.

7.6 COMMUNICATION ARCHITECTURES FOR ENERGY/BATTERY-EFFICIENT SYSTEMS

In this section, we turn our attention to the design of communication architectures for energy-efficient and battery-efficient systems. For battery-efficient systems, it is important not only to reduce the total energy consumption of the system but also to tailor the manner in which energy is drawn to specific characteristics of the battery.

We consider two categories of techniques that address the design of the communication architecture, keeping energy/battery-efficiency in mind. The first category consists of techniques that attempt to reduce the energy consumption of the communication architecture itself, by reducing the power consumed by the wires constituting the communication architecture topology. The second category consists of techniques that are concerned with the impact of the communication architecture on the energy consumption of the rest of the system. We describe the details of an approach that uses "battery-aware" on-chip communication protocols to regulate the total system power consumption dynamically, as opposed to merely reducing energy consumption of the communication architecture. This communication-based power management technique is battery-driven and hence can be used to maximize battery life.

7.6.1 Minimizing Energy Consumed by the Communication Architecture

We first examine techniques that help reduce the energy consumption of the communication architecture itself. We describe techniques for reducing the power

consumed while transferring data across busses in deep submicron designs and then describe recent work on optimizing the topology for energy minimization.

Energy-Efficient DSM Interconnects

Low-power design of DSM interconnects is an active area of research that aims at coping with the new sources of power consumption associated with global interconnects in deep submicron (DSM) technologies. These effects (e.g., cross-coupling between neighboring wires) play a significant role in determining the power consumption of the communication architecture, due to the combined effects of large wire lengths, high aspect ratios, and small interwire spacing. Analysis of technology trends suggests that the power consumed by such DSM communication architectures could constitute a significant fraction of total system power, in some cases, reaching 40 to 50% [276,321]. Although in the past, numerous techniques have been proposed for minimizing the power consumed on bus lines (e.g., refs. 322 and 323), these techniques were largely concerned with off-chip busses, for which it was sufficient to minimize the total switching activity on the bus. However, it is increasingly clear that such approaches provide limited advantage for on-chip busses, in which DSM effects further complicate the problem.

Several power models have recently been developed for DSM interconnects [324–327]. These models, being more accurate than traditional approaches (based on signal transition counts), have been applied to developing design techniques for low-energy data communication on DSM busses. These techniques are largely based on encoding techniques that transform the communicated data so as to reduce the impact of DSM effects [326,328–331]. Specialized techniques are used for address busses to exploit the high correlation between successive bus words. Another approach that avoids power consumption due to deep submicron effects includes modifying the organization of bus lines. Shin and Sakurai [332] describe an approach whereby the ordering of parallel bus lines is optimized (based on a profile of the data values transferred) so as to minimize power consumption. Machiarullo et al. [333] describe a technique for optimizing the spacing between adjacent address bus wires (also using value profiling), to help reduce power consumption, while satisfying design rules. Recently, low swing signaling techniques have been proposed for on-chip busses [334] that make use of dynamic voltage scaling techniques to reduce energy consumption.

Energy-Efficient Communication Architecture Topologies

Techniques described in refs. 335 and 336 illustrate the potential savings in energy consumption that can be achieved via optimizations in the communication

architecture topology. In each of these approaches, shared bus-based architectures are considered. These techniques evaluate the benefit of splitting globally shared bus lines into smaller, segmented busses (e.g., by separating them with tristate buffers, or pass transistors). It has been demonstrated that when the communications between system components exhibit high spatial locality, splitting the shared medium into segments can result in large energy savings. This is because, in the split architecture, local communications can be performed by charging/discharging shorter, segmented busses, which have lower parasitic load and hence require less energy.

7.6.2 Improving System Battery Efficiency Through Communication Architecture Design

In this section, we describe the importance of the communication architecture in affecting the power consumption characteristics of the entire system, rather than merely considering the energy consumed by the communication architecture itself. In addition, the technique we describe in this section helps improve system battery efficiency, as opposed to just reducing energy consumption. Battery-efficient design aims at maximizing battery life by tailoring the run-time power consumption profile of a system to battery discharge characteristics. A large body of work has shown that such techniques can yield significant improvements in battery life, over and beyond what can be accomplished by merely minimizing total energy consumption (see ref. 337 for a survey of the area).

Communication-based power management (CBPM) is a system-level power management methodology that uses new "battery-aware" on-chip communication protocols to regulate the execution of system components dynamically, in order to achieve battery-friendly system-level power profiles. CBPM differs significantly from conventional power management techniques, which aim at minimizing power drawn during idle periods, or voltage scaling techniques, which mainly target reducing average power. In contrast, CBPM exercises proactive control over system components, delaying components that may be executing less critical operations, even if they are not idle. There are several advantages to using the communication architecture for battery-driven, system-level power management. These include:

- Low hardware overhead: the communication architecture physically and logically integrates all the components of the system. Embedding power management functionality in the communication architecture saves the cost of dedicated hardware and wiring for power management units or controllers.

- System-wide visibility: to exercise dynamic, regulatory control over the system power profile, it is important to gain access to system-wide information (e.g., current execution states of components). The communication architecture provides system-wide connectivity, allowing (1) effective monitoring of component execution states, and (2) efficient delivery of power management decisions to system components.
- Control over system power: the communication architecture directly controls the timing of intercomponent communications and hence, if properly exploited, can regulate component execution as well. CBPM features communication protocols that exercise this control, regulating the occurrence of component idle/active states, which allows shaping of the overall system power profile.

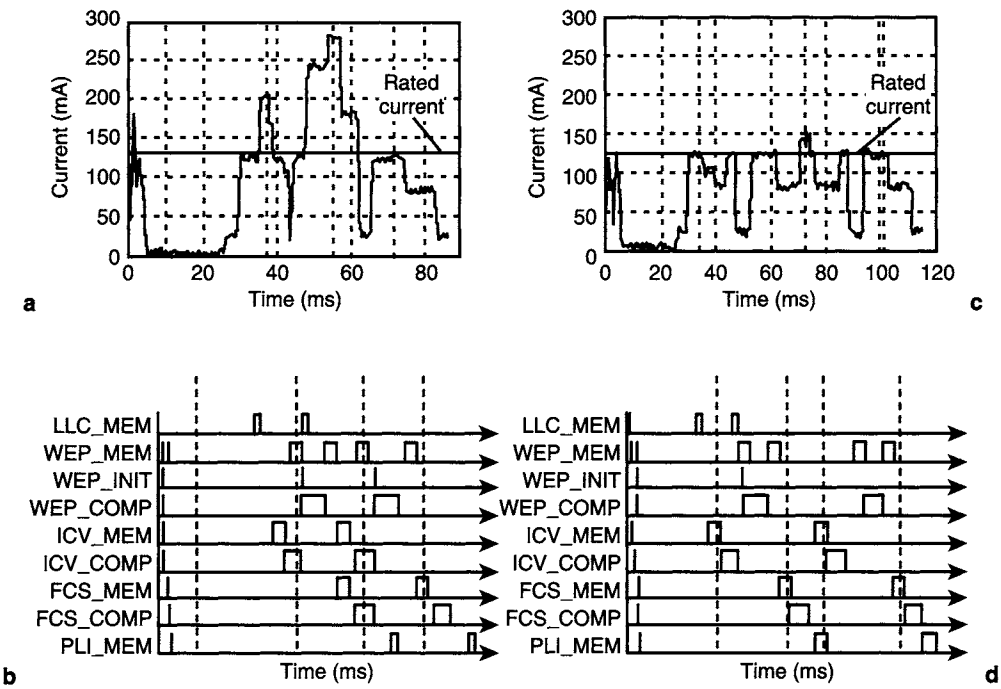
We illustrate how CBPM works by considering its application to an IEEE 802.11 MAC processor design. Next, we provide a brief description of a methodology for designing CBPM-based systems, and finally we describe the operation and implementation of a CBPM-based communication protocol.

CBPM: IEEE 802.11 MAC Processor Example

The IEEE 802.11 MAC processor architecture that we consider consists of two embedded processors, three coprocessors, and a hierarchical bus-based communication architecture. It implements the MAC layer functionality of a device connected to an IEEE 802.11b wireless LAN. Although for this example we consider only the system power profiles and execution traces of the processor, details of the design are available in ref. 338.

Example 8. *We compare the power profiles and battery efficiency of (1) an original, performance-optimized MAC processor and (2) the same design, with a CBPM-enhanced communication architecture. Figure 7-12a shows a snapshot of the current profile for the original MAC processor as it processes three MAC frames.*[4] *This snapshot is part of a longer profile, consisting of several thousand MAC frames processed while one is streaming video over an IEEE 802.11b wireless LAN. For the profile of Figure 7-12a, the energy delivered by a 250-mA/h lithium-ion battery was estimated at 154.3 mA/h over a lifetime of 3209.2 s [339]. The poor efficiency (62%) is attributable to regions of Figure 7-12a, in which it can be seen that the current drawn is significantly higher than the battery's rated current.*

[4] Current profiles are obtained by HW/SW cosimulation-based power estimation followed by division by the supply voltage (assumed constant).



7-12 FIGURE

Current discharge profiles for an IEEE 802.11 MAC processor (a) without CBPM and (b) with CBPM. Symbolic execution traces of component macrostates (c) without CBPM and (d) with CBPM.

To understand how CBPM improves battery efficiency, we consider a macrostate execution trace (Fig. 7-12c), which corresponds to the profile of Fig. 7-12a. Macrostates correspond to specific subtasks that are performed by system components at different times. For example, the component wep could be in one of three macrostates: wep_init, which corresponds to the subtask of initializing a state array for subsequent use during encryption; wep_mem, which corresponds to the subtask of reading/writing MAC frame data from/to memory; and wep_comp, which corresponds to the subtask of encrypting frame data while accessing the state array. From Figure 7-12a and c, it is clear that the current drawn can vary significantly with time, depending on the macrostate combination. For example, Figure 7-12a indicates that, at t = 55 ms, while two hardware components are engaged in lengthy iterative computations (icv_comp and fcs_comp), and a third component accesses memory over a shared bus (wep_mem), the total drawn current is 280 mA, which is well in excess of the battery's rated current [340].

Next, we consider the same sequence of frames being processed by a CBPM-enhanced MAC processor. Figure 7-12b and d shows the current profile and the corresponding macrostate execution trace for the optimized design. A comparison of Figure 7-12c and d shows that CBPM delays the execution of certain macrostates (shaded) to keep the system within the rated current. For example, in Figure 7-12d CBPM delays the occurrence of macrostate wep_mem to t = 42 ms, to avoid overlap with macrostate icv_comp. From the current profiles, it is clear that the occurrence of macrostate combinations that violate the rated current have been greatly reduced. The battery life of the new system was observed to be 10,199 s (a 3.2 × improvement), and battery capacity was 225.4 mA/h (a 1.5 × improvement).

The above example illustrates that the communication architecture can play a significant role in affecting the power profiles of components that are connected to it and can thereby regulate the power consumption of the entire system, improving battery efficiency.

CBPM Design Methodology

In order to design a CBPM-based system, the specification of each component is decomposed into a set of macrostates whose execution is under the control of the communication architecture. Efficient performance analysis techniques are used to prioritize identified macrostates, and power profiling tools are used to estimate the average power consumption of each macrostate. Once the macrostates have been identified, the specification of each component is enhanced to generate special requests at each macrostate transition. These requests convey the component's destination macrostate to the communication architecture. A component with a pending macrostate transition request must wait until the (CBPM-enhanced) communication architecture issues it a grant. These grants are issued in accordance with a predetermined but configurable CBPM policy that is implemented in the on-chip communication protocols.

CBPM-Enhanced Communication Protocol

A CBPM-based system consists of a communication architecture that is enhanced with new battery-aware communication protocols. The CBPM protocols implement policies that regulate macro state execution. The CBPM protocol periodically polls for accumulated pending macrostate requests; it then grants as many pending macrostates as possible, starting with the macrostate of the highest priority, while making sure that a specified constraint on the total power consumption is not violated. This constraint helps control the extent to which the run-time power profile is tailored towards the battery, by regulating the occurrence of

macrostate combinations that violate the battery's rated current. These policies are implemented as extensions to the existing communication protocol hardware. CBPM-enhanced communication architectures can be configured through specification of parameters (including the power constraint), t trade off performance versus battery efficiency statically or dynamically.

7.7 CONCLUSIONS

As SoCs become increasingly complex, the on-chip communication architecture is expected to become more and more critical from the point of view of system performance, energy consumption, and battery life. In this chapter, we provided an overview of techniques for designing communication architectures for high-performance, energy-efficient system-on-chips. We described techniques for fast and accurate system-level performance/power analysis to drive communication architecture design and automatic techniques for customizing the communication architecture to the characteristics of an application's communication traffic, through both static customization (of communication architecture templates) and dynamic customization (using CATs). We also described a system power management methodology (CBPM) that utilizes new, battery-aware on-chip communication protocols to enhance the overall battery efficiency of a system. We believe that in the near future, in order to combat the increasing challenges posed by on-chip communication, such communication-aware design methodologies will be widely integrated into design practice.

ACKNOWLEDGMENTS

The text of this chapter is partly based on material that has been published (or is pending publication) in the following conference proceedings and journals: the International Conference on VLSI Design (2000), the IEEE/ACM Design Automation Conference (2000, 2001, 2002), the Intl. Conf. on Computer-Aided Design (1999, 2000), the International Symposium on HW/SW Co-Design (2002), the IEEE Transactions on Computer-Aided Design of Circuits and Systems (vol. 20, no. 6), and the IEEE Design and Test of Computers (vol. 19, no. 4).