CAPPS: A Framework for Power–Performance Tradeoffs in Bus-Matrix-Based On-Chip Communication Architecture Synthesis

Sudeep Pasricha, *Member, IEEE*, Young-Hwan Park, *Student Member, IEEE*, Fadi J. Kurdahi, *Fellow, IEEE*, and Nikil Dutt, *Fellow, IEEE*

***Abstract*—On-chip communication architectures have a significant impact on the power consumption and performance of emerging chip multiprocessor (CMP) applications. However, customization of such architectures for an application requires the exploration of a large design space. Designers need tools to rapidly explore and evaluate relevant communication architecture configurations exhibiting diverse power and performance characteristics. In this paper, we present an automated framework for fast system-level, application-specific, power–performance tradeoffs in a bus matrix communication architecture synthesis (CAPPS). Our study makes two specific contributions. First, we develop energy models for system-level exploration of bus matrix communication architectures. Second, we incorporate these models into a bus matrix synthesis flow that enables designers to efficiently explore the power–performance design space of different bus matrix configurations. Experimental results show that our energy macromodels incur less than 5% average cycle energy error across 180–65 nm technology libraries. Our early system-level power estimation approach also shows a significant speedup ranging from 1000 to 2000× when compared with detailed gate-level power estimation. Furthermore, on applying our synthesis framework to three industrial networking CMP applications, a tradeoff space that exhibits up to 20% variation in power and up to 40% variation in performance is generated, demonstrating the usefulness of our approach.**

***Index Terms*—Communication architecture performance, communication architecture power estimation, digital systems, high-level synthesis.**

I. Introduction

THE rapidly increasing complexity of chip multiprocessor (CMP) designs, coupled with poor global interconnect scaling in the deep-submicrometer (DSM) era, is making on-chip communication a critical factor affecting overall system performance and power consumption [1], [45]. These communication architectures are not only a major source of performance bottlenecks for data-intensive domains, such as multimedia and networking, but recent research has shown that they also consume as much power as other well-known primary sources of power consumption, such as processors and caches [2]. Designers must therefore give special emphasis to the selection and design of on-chip communication architectures early in the design flow, preferably at the system level.

Several different types of on-chip communication architectures, such as hierarchical shared buses [3], [4], bus matrices [5], and network-on-chips (NoC) [6] have been proposed. Hierarchical shared bus architectures, such as those proposed by AMBA [3] and CoreConnect [4] have been extensively used in the past, but are often unable to meet the high-ba ndwidth requirements of emerging applications. Bus matrix architectures, such as the AMBA AHB (Advanced High-performance Bus)bus matrix [5], are an evolution of the hierarchical shared bus scheme. They consist of several parallel buses that can provide superior bandwidth response, and are being increasingly used in designs today. While packet-switched NoCs are promising candidates for interconnection fabrics in future designs and lithographic processes [7], our focus in this paper is on bus matrix architectures that are being used in a large number of CMP designs today.

Bus matrix architectures are characterized by customizable topologies and protocol parameters, which creates a vast exploration space that cannot be explored exhaustively in a reasonable amount of time [8]. Different configurations in this space can have vastly different power/energy and performance characteristics. While meeting the minimum performance (e.g., minimum throughput) constraints of an application is certainly an important criterion for the choice of a particular configuration, minimization of energy consumption is just as relevant, especially for mobile devices that run on batteries and have a limited energy budget [9]. Even more important are the thermal implications of power consumption. A large portion of the energy consumed from the power supply is converted into heat. A rise of even 10 °C in the operating temperature has been shown to not only increase interconnect delay (reducing performance), but also increase electromigration (EM), which significantly raises device failure rate [10]. Since interconnect temperatures can reach as high as 90 °C [11], reducing power consumption becomes essential to ensure correct operation, and also to reduce costs for expensive cooling and packaging equipment.

Designers are thus faced with the challenge of quickly and effectively traversing the vast bus matrix architecture exploration space during its design/synthesis phase, to tradeoff power–performance characteristics. Due to the highly complex nature of emerging applications, performing performance and especially

Manuscript received September 18, 2007; revised February 07, 2008. First published April 14, 2009; current version published January 20, 2010. This work was supported in part by grants from SRC (2005-HJ-1330) and a CPCC Fellowship.

S. Pasricha is with Colorado State University, Fort Collins, CO 80523-1373 USA (e-mail: sudeep@engr.colostate.edu).

Y. Park, F. J. Kurdahi, and N. Dutt are with the University of California, Irvine, CA 92617-3425 USA (e-mail: younghwp@uci.edu; kurdahi@cecs.uci.edu; dutt@cecs.uci.edu).

Digital Object Identifier 10.1109/TVLSI.2008.2009304

power exploration at the lower register transfer level (RTL) /gate levels can be excessively time-consuming [23]. There is a need to raise the exploration abstraction to the system level, where not only can the speed of exploration be improved, but design decisions also have a greater impact on power and performance, than at the lower levels. However, such an effort requires reliable power estimation at the system level, which is not a trivial task.

A. Paper Overview and Contributions

In this paper, we address the issues highlighted above by proposing an automated synthesis framework (CAPPS) to perform application-specific, system-level power–performance tradeoffs, for bus matrix communication architectures. Our key contributions in this paper are: 1) developing detailed energy macromodels for the bus matrix architecture, which can be used in any cycle-accurate simulation models for power estimation across technology nodes and 2) an automated synthesis framework for the bus matrix architecture, which uses these energy macromodels in the fast and accurate transaction-level CCATB [12] simulation environment in SystemC [13], to efficiently explore the power–performance design space of the generated set of solutions. Our experimental results demonstrate estimation accuracy of average cycle energy to within 5% using our energy macromodeling approach with respect to detailed gate-level estimation using Synopsys PrimePower [14] across 180–65 nm technology nodes. Moreover, our system-level power estimation approach achieves between 1000 and 2000 $\times$ speedup over gate-level estimation when the macromodels are plugged into a transaction-level simulation environment (CCATB [12]) in SystemC [13]. Applying our synthesis framework on three industrial networking multiprocessor systems-on-chip (MPSoC) applications used for data packet processing and forwarding enabled a tradeoff of up to 20% for power, and up to 40% for performance, showing the effectiveness of our approach.

The rest of the paper is organized as follows. Section II presents related work. Section III gives an overview of the AMBA AHB bus matrix architecture. Section IV describes the energy model creation methodology and also presents the created energy models for the AMBA AHB bus matrix architecture. Section V describes the CAPPS synthesis framework in detail. Section VI presents experiments to determine the accuracy of our energy models and evaluate the efficacy of our synthesis framework. Finally, Section VII presents the conclusion and directions for future work.

II. Related Work

There is a large body of work dealing with performance-oriented system-level synthesis of hierarchical shared bus [15]–[17], NoCs [18], [46] and, more recently, bus matrix or crossbar [8], [43] architectures. These works attempt to improve application performance by customizing the communication architecture topology and protocol parameters such as arbitration schemes. Some approaches have addressed power-aware synthesis of communication architectures. In [19], a heuristic was presented for topology customization of the AMBA AHB hierarchical shared bus to reduce power consumption. In [20], [47]–[50] techniques for low-power design of NoCs were presented. However, power-aware synthesis for bus matrix architectures has not been addressed to date. A few pieces of work have looked at power–performance tradeoffs during on-chip communication architecture design. In [21], a technique to tradeoff energy and performance using high-level layout optimization was proposed for segmented buses. In [22], the Orion power–performance simulator was described, to enable architects to explore interconnect power and performance tradeoffs. In [34], different NoC topologies such as mesh and ring are analyzed to explore power–performance tradeoffs for a cryptographic application. Our study differs from existing work in that we focus on the bus matrix communication architecture, for which we create an automated, simultaneous topology and protocol parameter synthesis framework, to enable exploration of power–performance tradeoffs at the system level.

Power modeling and estimation is typically performed either at the transistor, gate, or register-transfer levels [23]. In practice, most commercial design flows use register-transfer or gate-level power estimation tools. However, due to the highly complex nature of modern applications, these approaches are too inefficient for early power estimation. To overcome this drawback, recent approaches [10], [24]–[29] analyze power for on-chip communication architectures at the system level, where significant improvements in run time can be achieved, at the cost of estimation accuracy. In [28], a high-level framework for NoC energy consumption is proposed. The approach uses link utilization as the unit of abstraction for network utilization and energy consumption, but is aimed at early coarse-grained power estimates, and not on accurate cycle power/energy estimation. In [29], an energy estimation flow for different types of NoC routers is proposed. The approach however ignores leakage and clock power. In [26], gate-level power observations are used to create a simple lookup table for read/write transfers on the IBM Coreconnect bus. The lookup table is used to annotate transaction-level models (TLMs) and obtain very high-level power estimates for communication on the bus. A similar gate-level table lookup technique is used in [27] to obtain power estimates for cycle-accurate and cycle-approximate TLM models. Although these TLM-based approaches allow for fast estimation, they can be highly inaccurate [44].

Other approaches have used gate-level estimates to create energy macromodels for on-chip communication architectures. In [10], macromodels for the arbiter and decoder components of the AMBA AHB hierarchical shared bus architecture are presented. However, the authors do not describe a model generation methodology, or account for power contribution due to leakage, clock, and wires. A coarse-grained single macromodel for the entire STBus communication architecture is presented in [25]. However it suffers from significant and unpredictable error under different traffic/loading conditions. In [24], an energy macromodel for an NoC router is presented. However, it is not clear how easily the methodology can be ported across multiple technology nodes. Our study is different from these in that we create energy macromodels for the bus matrix communication architecture, which enable accurate cycle-level energy/power estimation in a system-level simulation environment. Unlike previous work, we also explore the effect of technology scaling on power estimation accuracy.

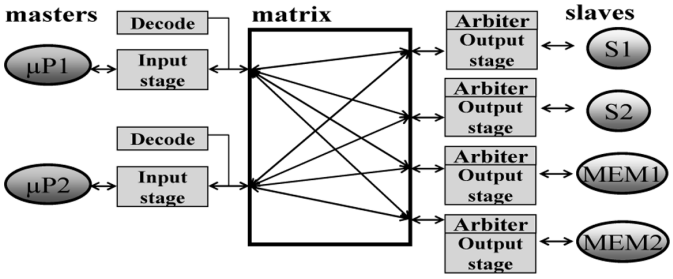

Fig. 1. AMBA AHB full bus matrix.

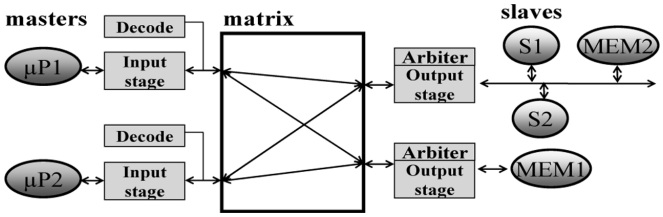

Fig. 2. AMBA AHB partial bus matrix.

III. AMBA AHB Bus Matrix Overview

We use the AMBA AHB matrix [5] as a representative of bus matrix communication architectures. The AHB [3] bus protocol supports pipelined data transfers, allowing address and data phases belonging to different transactions to overlap in time. Additional features, such as burst transfers and transaction SPLIT support, enable high data throughputs. A bus matrix configuration consists of several AHB buses in parallel, which can support concurrent high bandwidth data streams.

Fig. 1 shows a two-master, four-slave AMBA AHB full bus matrix. The *input stage* is used to handle interrupted bursts, and to register and hold incoming transfers if receiving slaves cannot accept them immediately. The *Decoder* generates select signals for slaves, and also selects which control and read data inputs received from slaves are to be sent to the master. The *Output Stage* selects the address, control, and write data to send to a slave. It calls the *Arbiter* that uses an arbitration scheme to select the master that gets to access a slave, if there are simultaneous requests from several masters. Unlike in traditional hierarchical shared bus architectures, arbitration in a bus matrix is not centralized, but distributed so that every slave has its own arbitration. Also, typically, all buses within a bus matrix have the same data bus width.

One drawback of the *full bus matrix* structure shown in Fig. 1 is that it connects every master to every slave in the system, resulting in a large number of wires and logic components in the matrix. While this configuration ensures high bandwidth for data transfers, it is certainly not optimal as far as power consumption is concerned. Fig. 2 shows a *partial bus matrix* that has fewer wires and components (e.g., arbiters, MUXs), which consequently reduces power consumption, as well as area, at the cost of performance (due to an increase in the number of shared links). Our synthesis framework (described in Section V) starts with a full bus matrix, and aims to generate a set of partial bus matrix configurations (with different number of buses and logic components) that meet all the minimum performance constraints of the application being considered.

IV. Bus Matrix Energy Models

In this section, we present details of our bus matrix energy macromodel creation methodology. We first describe the basics of energy macromodeling. Then, we present the macromodel generation methodology. Finally, we present the energy models for all the components in the bus matrix, including bus wires.

A. Background on Energy Macromodels

The energy consumption of a bus matrix can be obtained by identifying events that cause a noticeable change in its energy profile. For this purpose, we create energy macromodels that can encapsulate events or factors having a strong correlation to energy consumption for a given component. A macromodel consists of variables that represent factors influencing energy consumption, and regression coefficients that capture the correlation of each of the variables with energy consumption. A general energy macromodel for a component can be expressed as

$$E_{\text{component}} = \alpha_0 + \sum_{i=1}^{n} \alpha_i \Psi_i \tag{1}$$

where α_0 is the energy of the component that is independent of the model variables, and α_i is the regression coefficient for the model variable Ψ_i. Note that we consider a linear energy model even though quadratic models can theoretically provide higher accuracy. The motivation for choosing a linear model is that if the linear model can provide us with high estimation accuracy, there is no need to consider more complex quadratic models.

For the purpose of our energy macromodels, we considered three types of model variables representing factors influencing energy consumption: *control*, *data*, and *structural*. The *control* factor represents control events, involving a control signal that triggers energy consumption either when it transitions from 1 to 0 or 0 to 1, or when it maintains a value of 0 or 1 for a cycle. Control variables can either have a value of 1 when a control event occurs, or 0 when no event occurs, in the energy macromodel relation in (1). The *data* factor represents data events that trigger energy consumption on data value changes. Data variables take an integer value in (1) representing the Hamming distance (number of bit-flips) of successive data inputs [10]. Finally, *structural* factors, such as data bus widths and number of components connected to the input also affect energy consumption of a component. They are represented by their integer values in (1).

B. Methodology Overview

A high-level overview of the methodology used to create energy macromodels in this study is shown in Fig. 3. We start with a system testbench, consisting of masters and slaves interconnected using the AMBA AHB bus matrix fabric. The testbench generates traffic patterns consisting of single and burst transactions of varying sizes, and different modes (e.g., SPLIT/RETRY, locked bus, etc.) that exercise the matrix under different operating conditions. Synopsys Coretools [14] is used to configure the bus matrix (specify data bus width, number of masters and slaves etc.) and generate a synthesizable RTL description of the bus matrix architecture (step 1). This description is synthesized to the gate-level with the Cadence physically knowledgeable

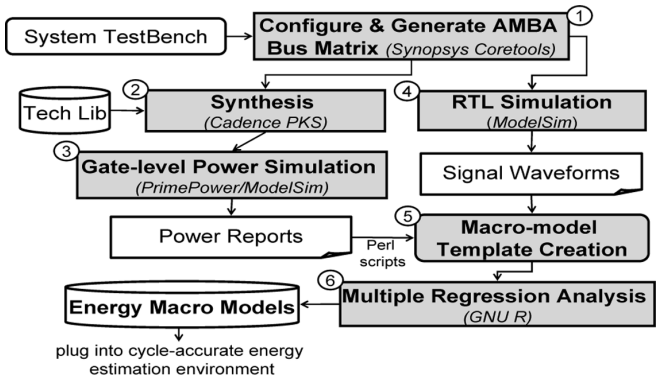

Fig. 3. Energy macromodel generation methodology.

cycle	cycle_energy	S_load	S_desel	HD_addr	S_drive
10	0.54802	0	0	0	0
20	0.54802	0	0	0	0
30	0.54802	0	0	0	0
40	0.54802	0	0	0	0
50	0.54802	0	0	0	0
60	0.54802	0	0	0	0
70	0.54802	0	0	0	0
80	0.54802	0	0	0	0
90	0.54802	0	0	0	0
100	0.54802	0	0	0	0
110	0.54802	0	0	0	0
120	0.54802	0	0	0	0
130	1.632607	1	0	3	1
140	0.961418	1	0	0	1
150	0.56406	0	0	0	0
160	0.560536	0	0	0	0
170	0.601455	0	0	3	0
180	0.547972	0	0	0	0
190	1.721611	1	0	6	1
200	0.946274	1	0	0	1
210	0.56392	0	0	0	0
220	0.5604	0	0	0	0
230	0.611902	0	0	3	0

Fig. 4. Macromodel template.

synthesis (PKS) [42] tool, for the target standard cell library (step 2). PKS pre-places cells and derives accurate wire length estimates during logic synthesis. In addition, it generates a clock tree including clock de-skewing buffers. The gate-level netlist is then used with Synopsys PrimePower [14] to generate power numbers (step 3).

In parallel with the synthesis flow, we perform RTL simulation to generate signal waveform traces for data and control signals of the bus matrix (step 4). These signal waveforms are compared with cycle energy numbers, obtained after processing PrimePower-generated power report files with Perl scripts, to determine which data and control signals in the matrix have a noticeable effect on its energy consumption. For our analysis, we consider signals that change in value when an increase in bus matrix energy consumption of at least 0.01% is observed over the base case (i.e., no data traffic). Note that a finer-grained selection criteria can be chosen (e.g., 0.001%), which would result in even more accuracy, but at the cost of more complex macromodels that take longer to create and evaluate. The selected data and control events become the variables in a macromodel template that consists of energy and variable values for each cycle of testbench execution (step 5). Fig. 4 shows an example of a macromodel template for one of the components of the bus matrix. The template consists of energy values (*cycle_energy*) and variable values (*S_load*, *S_desel*, *HD_addr*, *S_drive*) for each cycle of testbench execution. This template is used as an input to the GNU R tool [30] that performs multiple linear regression analysis to find coefficient values for the chosen variables (step 6). Steps 1–6 are repeated for testbenches having different structural attributes such as data bus widths and number of masters and slaves, to identify structural factors (variables) that may influence cycle energy.

Statistical coefficients such as *Multiple-R*, *R-square*, and *standard deviation for residuals*[31] are used to determine the goodness of fit and the strength of the correlation between the cycle energy and the model variables. Once a good fit between cycle energy and macromodel variables is found, the energy macromodels are generated in the final step. These models can then be plugged into any system-level cycle-accurate or cycle-approximate simulation environment, to get energy consumption values for the AMBA AHB bus matrix communication architecture.

C. Bus Matrix Component Energy Models

To obtain the energy consumption for the entire AMBA AHB bus matrix communication architecture, we used our energy macromodel generation methodology to create macromodels for each of its components. The total energy consumption of a bus matrix can be expressed as

$$E_{\mathrm{MATRIX}} = E_{\mathrm{INP}} + E_{\mathrm{DEC}} + E_{\mathrm{ARB}} + E_{\mathrm{OUT}} + E_{\mathrm{WIRE}} \quad (2)$$

where E_{INP} and E_{DEC} are the energy for the input and decoder components for all the masters connected to the matrix, E_{ARB} and E_{OUT} are the energy for arbiters and output stages connecting slaves to the matrix, and E_{WIRE} is the energy of all the bus wires that interconnect the masters and slaves. Energy macromodels were created for the first four components, with E_{WIRE} being calculated separately.

The energy macromodels are essentially of the form shown in (1). Leakage and clock energy, which are the major sources of independent energy consumption are considered as part of the static energy coefficient α_0 for each of the components. Based on our experiments, we observed an approximately linear relationship between cycle energy and macromodel variables for the components. We will now present the energy models for each of the components.

1) Input Stage: Every master connected to a bus matrix has its own input stage, which buffers address and control bits for a transaction, if a slave is busy. The input stage model can be expressed as

$$E_{\mathrm{INP}} = \alpha_{\mathrm{inp0}} + \alpha_{\mathrm{inp1}}\Psi_{\mathrm{load}} + \alpha_{\mathrm{inp2}}\Psi_{\mathrm{desel}} + \alpha_{\mathrm{inp3}}\Psi_{\mathrm{HDin}} + \alpha_{\mathrm{inp4}}\Psi_{\mathrm{drive}} \quad (3)$$

where Ψ_{load} and Ψ_{drive} are control signals asserted when the register is loaded, and when the values are driven to the slave respectively; Ψ_{desel} is the control signal from the master to deselect the input stage when no transactions are being issued; and Ψ_{HDin} is the Hamming distance of the address and control inputs to the register.

2) Decoder: A decoder component is connected to every master, and consists of logic to generate the select signal for a slave after decoding the destination address of an issued transaction. It also handles multiplexing of read data and response sig-

nals from slaves. The decoder energy consumption model can be formulated as

$$E_{\mathrm{DEC}} = \alpha_{\mathrm{dec0}} + \alpha_{\mathrm{dec1}}\Psi_{\mathrm{slavesel}} + \alpha_{\mathrm{dec2}}\Psi_{\mathrm{respse\ l}} + \alpha_{\mathrm{dec3}}\Psi_{\mathrm{HDin}} + \alpha_{\mathrm{dec4}}\Psi_{\mathrm{sel}} \quad (4)$$

where Ψ_{slavesel} and Ψ_{respsel} are control signals asserted in the cycle in which the slave select and the data/response MUX select signals are generated, respectively; Ψ_{HDin} is the Hamming distance of the read data and response signals from the slave; and Ψ_{sel} is a control signal that transitions when the decoder is selected or deselected.

3) Output Stage: Every slave is connected to the bus matrix through the output stage that multiplexes address and control bits from the masters. It also calls the arbiter to determine when to switch between accessing masters. The energy consumption for the output stage is given by

$$E_{\mathrm{OUT}} = \alpha_{\mathrm{out0}} + \alpha_{\mathrm{out1}}\Psi_{\mathrm{addrsel}} + \alpha_{\mathrm{out2}}\Psi_{\mathrm{datasel}} + \alpha_{\mathrm{out3}}\Psi_{\mathrm{HDin}} + \alpha_{\mathrm{out4}}\Psi_{\mathrm{noport}} \quad (5)$$

where Ψ_{addrsel} and Ψ_{datasel} are control signals asserted when address and data values are selected after a call to the arbiter results in a change in the master accessing the slave; Ψ_{HDin} is the Hamming distance of address and data inputs; and Ψ_{noport} is a control signal from the arbiter, which goes high when no masters access the slave in a cycle.

4) Arbiter: The arbiter is invoked by the output stage, and uses an arbitration scheme to grant access to one of the potentially several masters requesting for access to the slave. The cycle energy model for the arbiter is calculated as

$$E_{\mathrm{ARB}} = \alpha_{\mathrm{arb0}} + (\alpha_{\mathrm{arb1}} + n\alpha_{\mathrm{arb2}})\Psi_{\mathrm{arb}} + \alpha_{\mathrm{arb3}}\Psi_{\mathrm{arb+1}} + (\alpha_{\mathrm{arb4}} + n\alpha_{\mathrm{arb5}})\Psi_{\mathrm{desel}} + \alpha_{\mathrm{arb6}}\Psi_{\mathrm{desel+1}} \quad (6)$$

where Ψ_{arb} and $\Psi_{\mathrm{arb+1}}$ are control signals representing the cycle when arbitration occurs, and the subsequent cycle when the master select signal is generated; Ψ_{desel} and $\Psi_{\mathrm{desel+1}}$ are control signals representing the cycle when the arbiter is not selected by any master, and the subsequent cycle when it generates the *noport* signal for the output stage; and n represents the number of masters connected to the arbiter.

5) Bus Wires: The bus wires that connect masters, slaves, and logic components in the bus matrix dissipate dynamic power due to switching, and leakage power due to the repeaters inserted in long wires to reduce signal delay. The expression for energy consumption of a bus wire from [32] is extended to include the effect of repeaters (to reduce wire delay), and is given as

$$E_{\mathrm{WIRE}} = 0.5V_{\mathrm{dd}}^2\left(\Sigma C_L + \left(\frac{l}{d}\right)C_{\mathrm{REP}} + l\left(C_G + 2C_C\right)\right)\alpha + \left(\frac{l}{d}\right)E_{\mathrm{REP}} \quad (7)$$

where V_{dd} is the supply voltage, α is the switching factor representing bit transition activity on the wire, Σ_L is the sum of load capacitances of all the components connected to the wire, including the driver and receiver, C_{REP} is the capacitance of a repeater, C_G is the wire-to-ground capacitance per unit length, C_C is the coupling capacitance per unit length of the wire to its adjacent wires, l is the length of the wire, d is the interrepeater distance and E_{REP} is the repeater internal energy.

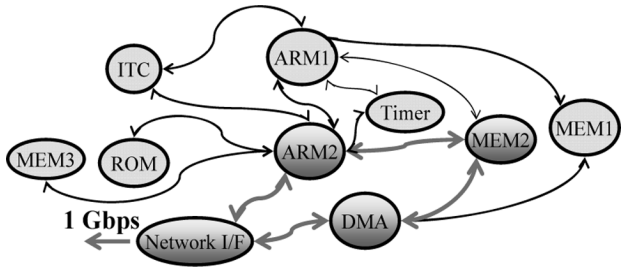

Fig. 5. CTG.

This single bus wire model is extended for an N bit bus. The Berkeley predictive technology model (PTM) [41] is used to estimate values for ground (C_G) and coupling (C_C) capacitances. The static energy of the repeater (E_{REP}) and its capacitance (C_{REP}) are obtained from data sheets. The component load capacitance on the wire (C_L) is obtained after component synthesis. A high-level simulated annealing-based floorplanner [33] is used to generate IP block placement to obtain wire lengths and is used to determine optimal-delay repeater spacing/sizing [36]. Finally, the switching factor (α) is obtained from simulation.

V. CAPPS Synthesis Framework

In this section, we describe the automated CAPPS framework for fast system-level, application-specific power–performance tradeoffs in bus matrix CAPPS. First we describe how performance constraints are encapsulated in our approach, followed by our assumptions and goals. Next we present the high-level floorplanning and wire delay estimation engines used to detect and eliminate clock cycle timing violations at the system level, as well as determine bus wire lengths. Finally, we describe the flow for the CAPPS framework in detail.

A. Background

Typically, CMP designs have certain minimum performance constraints that are dependent on the nature of the application, and must be satisfied by the underlying on-chip communication architecture. The *throughput* of communication between components is a good measure of the performance of a system [15]. To represent performance constraints in our approach, we use a *communication throughput graph* (CTG) $= G(V, A)$, which was first introduced in [16] and is briefly described here. A CTG is a directed graph, where each vertex v represents a component in the system, and an edge a connects components that need to communicate with each other. A *throughput constraint path* (TCP) is a subgraph of a CTG, consisting of a single component for which data throughput must be maintained, and other masters, slaves, and memories that are in the critical path impacting the maintenance of the throughput. Fig. 5 shows a CTG for a system on-chip (SoC) subsystem, with a TCP involving the ARM2, MEM2, DMA and ''Network I/F'' components, where data packets must stream out of ''Network I/F'' at a rate of at least 1 Gb/s.

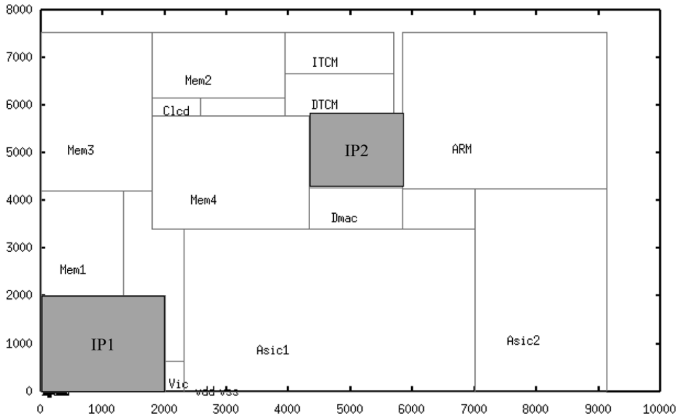

Fig. 6. Example floorplan with bus cycle timing violation.

Assumptions and Goals

The input to our framework is a CMP application that has certain minimum performance constraints that need to be satisfied. The application has several components (IPs) that need to communicate with each other. We assume that hardware–software partitioning has taken place and that the appropriate functionality has been mapped onto hardware and software IPs. These IPs are standard "black box" library components that cannot be modified during the synthesis process. The target standard bus matrix communication architecture (e.g., AMBA AHB bus matrix) is also specified.

The goal of our bus matrix synthesis framework then, is to automatically generate a set of bus matrix configurations for the given application that meet the minimum performance constraints of the application. These configurations exploit the *slack* available after meeting the minimum performance constraints, to optimize the design for power or performance. The output of the framework is a power–performance tradeoff graph for the generated set of valid bus matrix solutions, allowing a designer to pick a solution with the desired combination of power and performance characteristics.

B. Floorplan and Wire Delay Estimation Engines

It is possible that after a bus matrix architecture has been synthesized at the system-level, it might not be physically realizable due to *bus cycle timing violations*[35]. This can be illustrated as follows. Fig. 6 shows a floorplan for a system, where IP1 and IP2 are connected to the same bus as ASIC1, Mem4, ARM, VIC, and DMA components, and the bus has a clock frequency of 333 MHz. This implies that the bus cycle time is 3 ns. For a 130-nm process, a floorplanner determines a wire length of 9.9 mm between pins connecting the two IPs to the bus. Using output pin load capacitance values for IPs obtained from logic synthesis, the wire delay is calculated from formulations presented in [36], [37], and found to be 3.5 ns, which clearly violates the bus clock cycle time constraint of 3 ns. Typically, once such violations are detected at the physical implementation stage in the design flow, designers end up pipelining the buses by inserting latches, flip-flops, or register slices on the bus, in order to meet bus cycle time constraints. However, in our experience, such pipelining of the bus can not only have an adverse effect on critical path performance, but also requires tedious manual reworking of RTL code and extensive reverification of the design that can be very time-consuming [35]. It is therefore important to detect and eliminate such violations as early as possible in the design flow at the system level. To detect and eliminate any possible bus cycle timing violations in the bus matrix architecture solutions generated by our synthesis framework, we make use of high-level *floorplanning* and *wire delay estimation engines*.

The floorplanning stage in a typical design flow arranges arbitrarily shaped, but usually rectangular blocks representing circuit partitions, into a nonoverlapping placement while minimizing a cost function, which is usually some linear combination of die area and total wire length. Our *floorplanning engine* is adapted from the simulated annealing based floorplanner (PARQUET) proposed in [33]. The input to the floorplanner is a list of components and their interconnections in the system. Each component has an area associated with it (obtained from RTL synthesis). Dimensions in the form of width and height (for "hard" components) or bounds on aspect ratio (for "soft" components) are also required for each component. Additionally, maximum die size and fixed locations for hard macros can also be specified as inputs. Given these inputs, our floorplanner minimizes the cost function

$$\mathrm{Cost} = \mathrm{w}_1\mathrm{Area} + \mathrm{w}_2\mathrm{Bus}_{\mathrm{WL}} + \mathrm{w}_3\mathrm{Total}_{\mathrm{WL}} \tag{8}$$

where Area is the area of the chip, $\mathrm{Bus}_{\mathrm{WL}}$ is the wire length corresponding to wires connecting components on a bus, $\mathrm{Total}_{\mathrm{WL}}$ is total wire length for all connections on the chip (including interbus connections) and w_1, w_2, w_3 are adjustable weights that are used to bias the solution. For our experiments, we use the values $w_1 = 0.2$, $w_2 = 0.4$, $w_3 = 0.4$ to create a floorplan that has minimal wire length, at the cost of a less compact floorplan. The floorplanner outputs a nonoverlapping placement of components from which the wire lengths can be calculated by using half-perimeter of the minimum bounding box containing all terminals of a wire (HPWL) [38].

Once the wire lengths have been calculated, the *delay estimation engine* is invoked. The wire delay is calculated based on formulations proposed in [36] and [37], for optimal wire sizing (OWS) with buffer (or repeater) insertion. Both wire sizing and buffer insertion are techniques that have been found to reduce wire delay [39]. The inputs to this engine are: 1) the wire lengths from the floorplanner; 2) buffer details (resistance, capacitance, and delay, the values of which are obtained from data sheets; size and its corresponding critical length for buffer insertion, which is obtained from [37]); 3) technology library dependent parameters (from [40]); and 4) the capacitive loads (C_L) of component output pins (obtained from RTL synthesis).

We can simplify the multiple pin net problem (which is representative of a bus line) depicted in Fig. 7(a) to multiple two-pin net problems, as shown in Fig. 7(b). First, let us consider the wire delay for a wire without buffer insertion. The delay for a wire of length l, with OWS [37], is given as

$$\begin{aligned} T_{\mathrm{OWS}} &= R_d C_o + \left(\frac{\alpha_1 l}{W^2(\alpha_2 l)} + \frac{2\alpha_1 l}{W(\alpha_2 l)} \right. \\ &\qquad \left. + R_{dCf} + \sqrt{R_{drCaCf} l} \right) l \\ &= t_{\mathrm{driver}} + T'_{\mathrm{ows}}(R_d, l, C_L) \end{aligned} \tag{9}$$

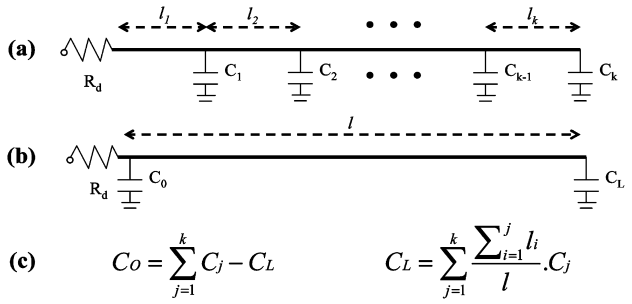

Fig. 7. Transforming multiple-pin net into two-pin net.

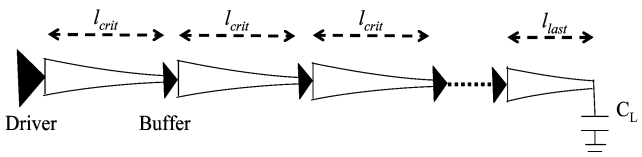

Fig. 8. Wire model for buffer insertion and wire sizing (BIWS) optimization.

where $\alpha 1 = (1/4)rC_a$, $\alpha 2 = (1/2)\sqrt{(rC_a/R_dC_L)}$ and $W(x)$ is Lambert's W function defined as the value of w, which satisfies $we^w = x$.R_d is the resistance of the driver, l is the wire length and $C_{\rm O}$ and C_L are capacitive loads that are calculated as shown in Fig. 7(c). The rest of the parameters are dependent on the technology library used, and are obtained from [40] $-r$ is the sheet resistance in Ω/sq, c_a is unit area capacitance in $\mathrm{fF}/\mu\mathrm{m}^2$ and $c_{\rm f}$ is unit fringing capacitance in $\mathrm{fF}/\mu\mathrm{m}$ (defined to be the sum of fringing and coupling capacitances).

Now let us consider wire delay, with buffer insertion. Let l_c be the critical length for buffer insertion, for the buffer size used, from [37]. If the length of a wire l is less than l_c, then no buffers need to be inserted, and the wire delay is given by (9). If however $l > l_c$, we need to insert buffers. The total number of buffers to be inserted, $n_b = \lfloor l/l_c \rfloor$. The wire is thus divided into $n_b + 1$ stages as shown in Fig. 8. Except for the first and last stages, each stage has equal length l_c and an equal delay $T_{\rm crit}$ given by

$$T_{\rm crit} = t_{\rm buffer} + T'_{\rm ows}(R_b, l_c, C_b) \tag{10}$$

where $t_{\rm buffer}$, R_b and C_b are the delay, resistance, and capacitance of the buffer used, respectively. The first stage also has a length of l_c, but a delay $T_{\rm first}$ given by

$$T_{\rm first} = t_{\rm driver} + T'_{\rm ows}(R_d, l_c, C_L). \tag{11}$$

The length of the last stage $l_{\rm last} = l - n_b.l_c$, and its delay $T_{\rm last}$ is given by

$$T_{\rm last} = t_{\rm buffer} + T'_{\rm ows}(R_b, l_{\rm last}, C_L). \tag{12}$$

Then the delay for a wire of length l, with optimal buffer insertion and wire sizing (BIWS), shown in Fig. 8, can be summarized as

$$T_{\rm BIWS} = T_{\rm OWS} \text{ for } l < l_c \tag{13}$$

$$= T_{\rm first} + T_{\rm last} \text{ for } l_c < l < 2l_c \tag{14}$$

$$= T_{\rm first} + n_b.T_{\rm crit]} + T_{\rm last} \text{ for } l > 2l_c. \tag{15}$$

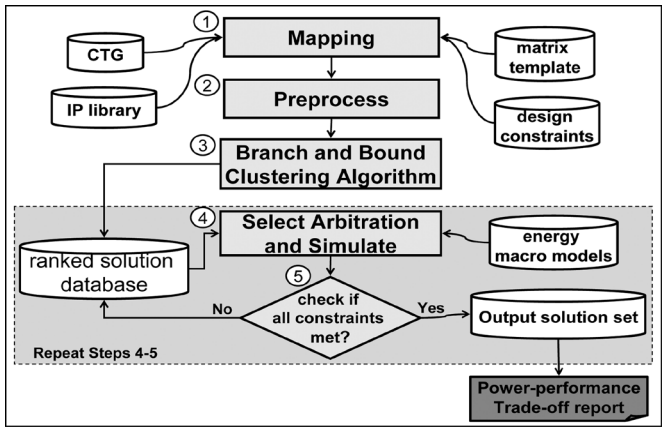

Fig. 9. CAPPS bus matrix synthesis framework.

As we will demonstrate in Section V-D, our synthesis framework makes use of the described *floorplanning and wire delay estimation engines* to detect (and eliminate) bus cycle time violations.

C. CAPPS Framework Overview

We now describe the CAPPS bus matrix synthesis framework in more detail. This framework has been derived from our previous work on bus matrix synthesis that generated a single, optimal partial bus matrix solution having the least number of buses [8]. This paper extends the framework by adding energy macromodels, and enabling power–performance tradeoffs on an output solution set having possibly several diverse bus matrix configurations.

The basic idea is to start with a full bus matrix configuration, and iteratively cluster slaves together to reduce the logic components and wires in the matrix that will intuitively allow a reduction in power consumption. But this will come at the cost of performance, which degrades with decreasing number of buses, because of bottlenecks at the slave end due to increased traffic (and consequently extended arbitration delays). The final solution set will consist of bus matrix configurations that do not violate minimum performance constraints and have different number of logic components and buses, lying between the extremes of a full bus matrix and an optimally reduced, partial bus matrix having the least number of buses and components possible. The power–performance tradeoff graph for the solution set will then allow a designer to select a configuration from the space within these extremes, having the desired power and performance characteristics for a particular application.

Fig. 9 shows the major steps in the CAPPS synthesis framework. The inputs are: 1) the application-specific (CTG), 2) a library of IP components consisting of masters, slaves, and memories, 3) a template of the target full bus matrix communication architecture (essentially a simulatable bus matrix model for a chosen standard such as AMBA AHB, that is extendable to interconnect as many masters and slaves as required in an application), 4) energy macromodels for the bus matrix, and 5) designer constraints, which allow a designer to set the bus matrix clock frequency, data bus width (assumed to be uniform for all buses in the matrix), and arbitration schemes (currently our methodology

supports static priority, round robin, and TDMA schemes). The output is a set of solutions that meet application performance constraints, and a power–performance tradeoff report for these solutions.

We start by first mapping components in the IP library on to the full bus matrix template (Step 1). Next, we perform preprocessing on this matrix structure by: 1) removing unused buses with no data transfers on them, according to the CTG, and 2) migrating components that are accessed exclusively by a master, to its local bus, in order to reduce components (arbiters, output stage) and wire congestion in the matrix (Step 2). The subsequent step involves static analysis to further reduce the number of components and buses in the matrix, by using a *branch and bound based hierarchical clustering algorithm* to cluster slave components (Step 3). Note that we do not cluster masters because it adds two levels of contention (one at the master end and another at the slave end) in a data path that can drastically degrade system performance.

The branching algorithm starts out by clustering two slave clusters at a time, and evaluating the gain from this operation. Initially, each slave cluster has just one slave. The total number of clustering configurations possible for a bus matrix with n slaves is given by $(n! \times (n-1)!)/2^{(n-1)}$. This creates an extremely large exploration space, which cannot be traversed in a reasonable amount of time. In order to consider only useful clustering configurations, we make use of a *bounding* function, which ensures that: 1) only beneficial clusterings are allowed that result in component or wire savings and 2) performance constraints are still valid after clustering. The interested reader can refer to our previous work in [8] for a more detailed treatment of this algorithm.

The output of the algorithm is a set of solutions that are ranked and stored in a database according to the number of buses. The next step (Step 4) selects a solution from the ranked database, and does the following: 1) fixes arbitration schemes for each slave cluster in the matrix, and 2) performs simulation, to gather power and performance statistics. The arbitration scheme gives priority to higher (TCPs) from the CTG. Simulation is performed using a fast, transaction-level CCATB [12] simulation model in SystemC [13]. It allows us to verify if all performance (TCP) constraints are satisfied, in the next step (Step 5), since it is possible for dynamic factors, such as traffic congestion, and buffer overflows etc., to invalidate the statically predicted solution, in some cases.

Steps 4 and 5 are repeated for the desired number of times, based on the number of solutions required in the output set. The designer can specify different strategies to select solutions from the ranked database, such as selecting solutions from the top and the bottom of the rankings, and then selecting solutions at regular intervals between them, to get a wider spread on the power–performance characteristics. Finally, we use the energy and performance statistics obtained from simulation in Step 4, for each of the solutions in the output set, to generate power–performance (and energy-runtime) tradeoff graphs.

1) Linking Energy Macromodels With CCATB Models: The CCATB simulation abstraction uses function (i.e., transaction level) calls instead of signals, to speed up the simulation-based estimation process [12]. It incorporates bus matrix energy macromodels for energy estimation. The equations from Section IV-C are inserted in the code, at points, where a change in the value of an energy consuming event (i.e., dependent variable) can occur.

TABLE I
DEPENDENT VARIABLE ACTIVATIONS FOR A WRITE TRANSACTION

Component	Cycle n	Cycle n+1	Cycle n+2
Input Stage	Ψ_{load}, Ψ_{HDin}	Ψ_{drive}	-
Decoder	$\Psi_{slavesel}$, Ψ_{HDin}	$\Psi_{respsel}$	-
Arbiter	Ψ_{arb}	Ψ_{arb+1}	-
Output Stage	-	$\Psi_{addrsel}$, Ψ_{HDin}	$\Psi_{datasel}$, Ψ_{HDin}

To illustrate the linking of a transaction in the CCATB model with the energy macromodels, we present an example of a single write transaction and show how it activates the macromodels as it propagates through the bus matrix. Table I gives the dependent variables from Section IV-C, for each of the components in the matrix that are triggered during the write transaction in the model, assuming no slave delays and a single cycle overhead for arbitration. Every change in a dependent variable for a component triggers its corresponding energy macromodel equation, which calculates the energy for the event and updates the cycle energy value. Static energy values are updated at the end of every cycle. Energy values can be recorded at each cycle, for every component if needed; or in a cumulative manner for the entire bus matrix, for use in the final power–performance tradeoff analysis.

VI. EXPERIMENTS

A. Energy Macromodel Generation

In order to generate the energy macromodels for the AHB bus matrix communication architecture, we first selected a diverse set of bus matrix-based system testbenches, having different number of masters and slaves (2–40 components), bus widths (32–128 bits), and data traffic profiles (single and variable sized burst transactions) that activate the logic components in the bus matrix in different states. Using the methodology in Fig. 3, we combine the RTL simulation results of these testbenches and populate the macromodel template (Fig. 4). The methodology is then used to generate the energy macromodels and obtain the component regression coefficients, for the TSMC 180 nm standard cell library. The regression coefficients for the macromodels of each of the components of the bus matrix (Section IV-C) are shown in Table II. Also shown are the R squared (R^2) values that measure the goodness of fit for each regression [31]. The value of R^2 is a fraction between 0.0 and 1.0. A value of 0.0 indicates that knowing the variable values will not allow us to predict the energy values, whereas a value of 1.0 indicates that knowing the variable values will allow us to predict the value of energy perfectly. From the table it can be seen that the R^2 values are close to 1.0, thus enabling reliable prediction of energy at the system-level.

Next, we targeted the same system testbenches to the 130 nm, 90 nm and 65 nm TSMC standard cell libraries and repeated the regression coefficient generation process for the components in the bus matrix. The regression coefficients for the *input stage* bus matrix component, for each of the standard cell libraries, are

TABLE II
COEFFICIENTS FOR ENERGY MACRO-MODELS (180 NM)

Component	Variable	Energy coeff. (pJ)	Variable	Energy coeff. (pJ)	R^2
Input Stage	α_{inp0}	7.78	α_{inp3}	0.96	0.97
	α_{inp1}	3.81	α_{inp4}	3.27	
	α_{inp2}	2.60			
Decoder	α_{dec0}	0.47	α_{dec3}	0.13	0.90
	α_{dec1}	3.04	α_{dec4}	0.38	
	α_{dec2}	2.17			
Output Stage	α_{out0}	0.72	α_{out3}	0.14	0.94
	α_{out1}	2.61	α_{out4}	1.48	
	α_{out2}	1.53			
Arbiter	α_{arb0}	0.65	α_{arb4}	0.34	0.86
	α_{arb1}	0.76	α_{arb5}	0.48	
	α_{arb2}	0.30	α_{arb6}	0.52	
	α_{arb3}	0.60			

TABLE III
COEFFICIENTS FOR INPUT STAGE BUS COMPONENT (IN PJ)

Standard cell library	α_{inp0}	α_{inp1}	α_{inp2}	α_{inp3}	α_{inp4}
180 nm	7.78	3.81	2.60	0.96	3.27
130 nm	1.33	0.66	0.04	0.24	0.56
90 nm	1.23	0.44	0.02	0.20	0.40
65 nm	0.54	0.29	0.02	0.09	0.18

shown in Table III. It can be seen from the table that as we reduce the standard cell size, the coefficient values decrease, which is indicative of a decrease in overall power consumption. However, note that the reduction in coefficient values is not linear and the table indicates a change in relative importance of model variables with a change in standard cell library. For instance, the value of the coefficient for model variable $\alpha_{\rm inp3}$ is less than one-third the value of the coefficient for $\alpha_{\rm inp4}$ at 180 nm, but this ratio reduces to 1/2 for the 65 nm library.

A major advantage of our energy macromodel approach is the ease with which it can be retargeted to different standard cell libraries. Typically retargeting to a new standard cell library is a very time-intensive effort. However, our approach simply requires the generation of gate-level cycle energy numbers with the new library for the small system testbenches; these cycle energy numbers are used to update the cycle energy column in the macromodel template shown in Fig. 4, after which the regression analysis is repeated. Once the gate-level cycle energy numbers are available for the standard cell library, it only takes a few minutes to obtain the new coefficients for each of the components in the bus matrix. *This enables rapid retargeting of models for new cell libraries and results in an order-of-magnitude reduction in development time*.

B. Accuracy of Energy Macromodels

Next, we performed experiments to determine the macromodel accuracy for different standard cell libraries, by comparing our macromodel energy estimates with detailed gate-level energy estimates. We selected four system testbenches that were separate and distinct from the ones used for characterization and generation of the energy macromodels,

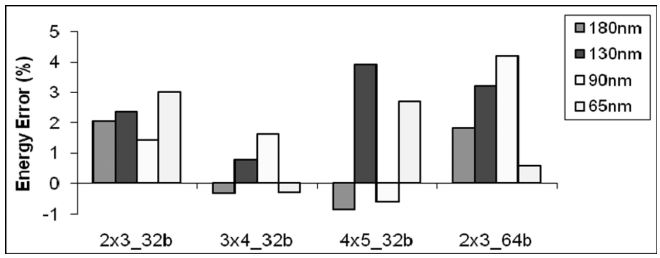

Fig. 10. Average cycle energy estimation errors.

and had different bus matrix structures and traffic characteristics: 1) a 2-master, 3-slave bus matrix with 32 bit data bus width (2×3)_32b, 2) a 3-master, 4-slave bus matrix with 32 bit data bus width (3×4_32b), 3) a 4-master, 5-slave bus matrix with 32 bit data bus width (4×5_32b) and 4) a 2-master, 3-slave bus matrix with 64 bit data width (2×3_64b). We synthesized these architectures and estimated cycle energy values using PrimePower [14] at the gate level, for each of the 180, 130, 90, and 65 nm TSMC standard cell libraries. In parallel, we characterized the model variables and coefficient values in the energy macromodels for each of the components in the matrix, to obtain estimated energy values.

Fig. 10 compares the average error in energy estimated at each cycle by the macromodels at the early system level, compared to gate-level estimation, for the different standard cell libraries. It can be seen that the energy macromodels allow highly accurate cycle energy estimation that is unaffected by the scaling of technology libraries towards DSM. *The maximum average estimation error for the macromodels is only 4.19%*. The variation in error for different test cases is due to the approximation introduced by curve fitting during the regression phase. This error can be further reduced if more variables (events) are included in the energy macromodel characterization phase.

Next, we plugged the energy macromodels into a fast transaction-level bus cycle-accurate simulation environment (CCATB [12]) in SystemC [13]. The CCATB simulation abstraction captures the bus matrix at a cycle-accurate granularity and uses function/transaction calls instead of signals to obtain simulation speed over bus-cycle accurate (BCA) models whichthat capture signal details, without sacrificing accuracy [12]. The simulation model incorporates bus matrix energy macromodels for cycle energy estimation. The equations from Section IV-C are inserted in the code for each component, at points where a change in the value of an energy consuming event (i.e., dependent variable) can occur.

Fig. 11 shows a snapshot of the power waveform generated by gate-level simulation using PrimePower, and the SystemC CCATB simulation using our energy macromodels for the 2×3_32b bus matrix system testbench. As can be seen, the power estimates using the system-level CCATB simulation model are highly correlated to the actual power consumption. This high correlation highlights *an additional benefit of the methodology in estimating peak energy* (as opposed to average energy). Peak energy is important in the planning of power grids, which is becoming increasingly difficult to do in DSM.

Table IV compares the CPU time taken for the PrimePower simulation (for gate-level cycle energy estimation) with the

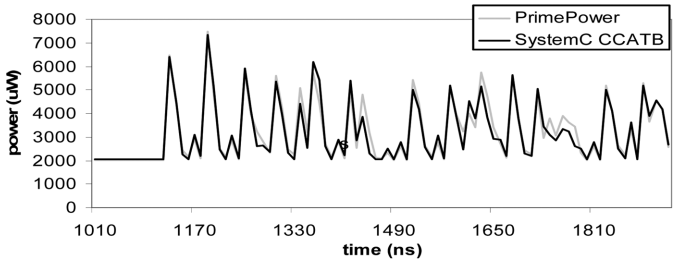

Fig. 11. Predicted and measured power waveforms.

TABLE IV
COMPARING TIME TAKEN FOR POWER ESTIMATION

Test bench	PrimePower Gate-level CPU time	SystemC T-BCA level CPU time	Speedup (X times)
2x3_32b	2.58 hrs	9.1 sec	1021
3x4_32b	7.91 hrs	18.8 sec	1515
4x5_32b	27.04 hrs	49 sec	1987
2x3_64b	3.10 hrs	9.9 sec	1127

system-level CCATB-based prediction, for the four system testbenches described earlier. The time taken for the gate-level and system-level power estimation does not depend on the technology library used, and is independent of it. *It can be seen from the table that the system-level power estimation gives a substantial speedup ranging from* ***1000****to* ***2000*** $\times$ *over gate-level power estimation using PrimePower*. Note that in our comparison we have not included time taken for value change dump (VCD) file generation and parasitic RC extraction file generation needed for PrimePower-based estimation, which takes from several minutes to hours for each of the testbenches. From these speedup results and our earlier observation on the high estimation accuracy of the macromodels across technology libraries, we believe that our energy macromodeling approach can be very useful for designers interested in fast and accurate communication architecture power estimation, early in the design flow, at the system level.

C. Power–Performance Tradeoffs

To validate the CAPPS bus matrix synthesis framework, we applied it to three proprietary industrial CMP applications (*PckSwitch, Datafilter, Nethub*) from the networking domain, used for data packet processing and forwarding. We present results for the *PckSwitch* application below, and summarize the results for the remaining two applications at the end of the section.

Fig. 12 shows the CTG for the *PckSwitch* application, with its minimum performance constraints that need to be satisfied represented by TCPs presented separately in Table V. ARM1 is a protocol processor (PP), while ARM2 and ARM3 are network processors (NPs). The ARM1 PP is mainly responsible for setting up and closing network connections, converting data from one protocol type to another and generating data frames for signaling, operating, and maintenance. The ARM2 and ARM3 NPs directly interact with the network ports and are used for assembling incoming packets into frames for the network connections, network port packet/cell flow control, assembling incoming packets/cells into frames, segmenting outgoing frames

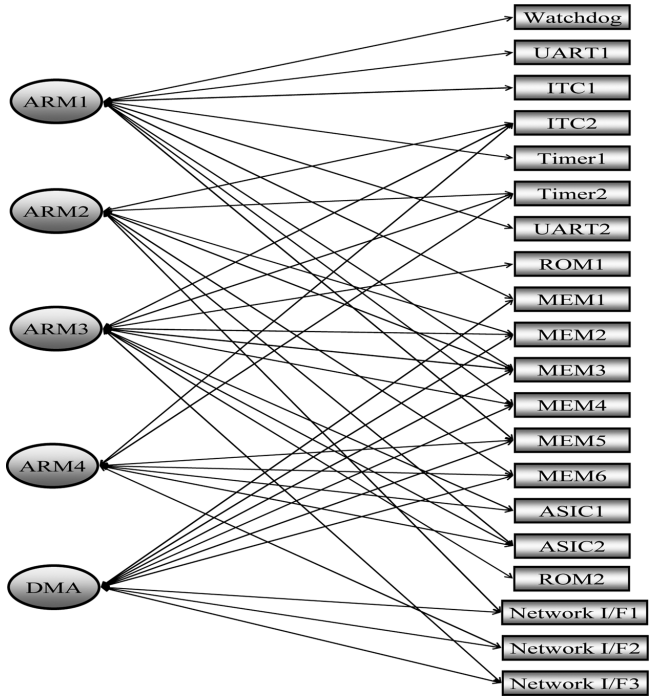

Fig. 12. CTG for *PckSwitch* networking CMP application.

TABLE V
TCPs (TCPs)

IP cores in Throughput Constraint Path (TCP)	Throughput Requirement
ARM1, MEM1, DMA, MEM3, MEM5	320 Mbps
ARM1, MEM3, MEM4, DMA, Network I/F2	240 Mbps
ARM2, Network I/F1, MEM2	800 Mbps
ARM2, MEM6, DMA, MEM8, Network I/F2	200 Mbps
ARM3, ARM4 , Network I/F3, MEM2, MEM7	480 Mbps

into packets/cells, keeping track of errors, and gathering statistics. ARM4 is used for specialized data processing involving data encryption. The DMA is used to handle fast memory to memory and network interface data transfers, freeing up processors for more useful work. Besides these master cores, the application also has a number of memory blocks, network interfaces, and peripherals such as interrupt controllers (*ITC1, ITC2*), timers (*Watchdog, Timer1, Timer2*), UARTs (*UART1, UART2*) and data packet accelerators (*ASIC1, ASIC2*).

The *PckSwitch* application and the target AMBA AHB bus matrix architecture were captured at the CCATB abstraction [12] in SystemC [13], with the bus matrix energy macromodels plugged into the simulation environment. The bus matrix design constraints specified a data bus width of 32 bits, a clock frequency of 100 MHz and a static priority-based arbitration scheme. Power numbers are calculated for the TSMC 180 nm standard cell library. We also included the energy contribution of bus wires as described in Section IV-C.

Fig. 13 shows the power–performance curves output by the framework for the *PckSwitch* application. The x-axis shows solutions in the output set having different number of buses, while the y-axis shows the percentage change in power and performance, using the original full bus matrix as the base case. It can be seen that a reduction in the number of buses leads to a corresponding reduction in the average power dissipation, because

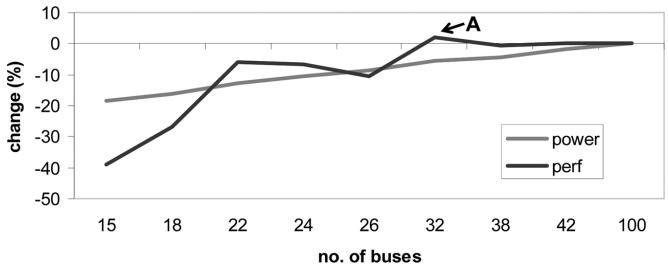

Fig. 13. Power-performance tradeoff graph.

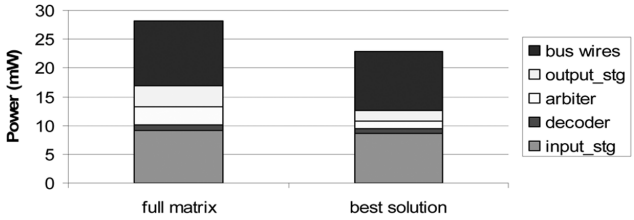

Fig. 14. Component-wise power comparison.

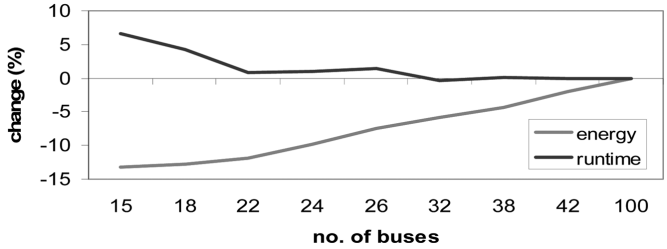

Fig. 15. Total energy versus runtime tradeoff graph.

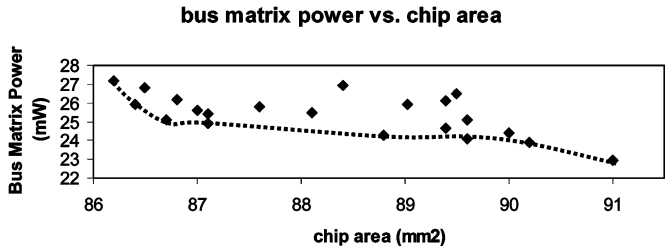

Fig. 16. Power versus chip-area tradeoff graph for MPSoC.

of fewer arbiters, output stages, and bus wires in the matrix, which results in less static and dynamic power consumption. Note that the number of buses cannot be reduced beyond 15 because it would lead to a violation of the minimum performance constraints of the application. Fig. 14 shows a comparison of the component-wise power consumption for the full bus matrix and the solution having the least number of buses, (which also consumes the least power). While the power consumed by the input stage initially decreases when the number of buses is decreased, due to reduced port switching, this trend is soon offset as traffic congestion increases arbitration delays, requiring additional buffering of transactions, for solutions with lesser number of buses. Similarly, for bus wires, the initial power reduction due to reduced number of wires in the matrix is soon offset by the need for longer shared wires to connect the increasingly clustered slaves.

As far as the performance change is concerned, (measured in terms of average percentage change in data throughput of constraint paths) a reduction in the number of buses increases traffic congestion and reduces performance due to an increase in arbitration wait time. However it is interesting to note that there are certain points (e.g., point A) in Fig. 13, where reducing the number of buses actually improves performance! This happens because of a reduction in port switching at the master end as slave clusters grow, reducing rearbitration delays for masters. In addition to the average percentage change in throughput, another useful metric to gauge application performance is its total runtime. Fig. 15 shows the corresponding total energy versus application runtime tradeoff graph for the MPSoC that is useful for cases, where the application must execute within a given time or energy budget, such as for mobile battery-driven applications.

It is also possible to perform other design space tradeoffs within the CAPPS framework. For instance, Fig. 16 shows the pareto-optimal tradeoff curve between bus matrix power consumed and the total area of the chip. To obtain this curve, we iterated through several different floorplans in the floorplanning stage. Since different floorplans have different wire routing arrangements, every floorplan has a different value for power consumption of the bus wires. Typically, as the allowed chip area is increased, the floorplanner can optimize the floorplan better, by reducing bus wire length, which consequently reduces bus power consumption.

TABLE VI
COMPARING TIME TAKEN FOR POWER ESTIMATION

MPSoC application	# of cores	Power/ Performance tradeoff	Energy/ Runtime tradeoff	Synthesis Time (hours)
PckSwitch	25	20% / 40%	14% / 8%	8.5
Datafilter	33	16% / 22%	12% / 5%	9.8
Nethub	49	15% / 34%	11% / 7%	11.1

Finally, Table VI summarizes the number of cores, maximum achievable power–performance and energy-runtime tradeoffs, and synthesis time for the *PckSwitch*, as well as the *Datafilter* and *Nethub* applications. For *PckSwitch*, according to the power–performance curves, there is a possible tradeoff of up to 20% for power and up to 40% for performance, enabling a designer to select the appropriate point in the solution space, which meets the desired power and performance characteristics. There is also a possible tradeoff of up to 8% for runtime and up to 14% for total energy consumed, which is a useful tradeoff metric when the application is running on a fixed energy budget, which is the case for mobile battery-driven devices. Table VI also shows the time taken for synthesis for the applications, in order to generate the tradeoffs curves. The synthesis time can be seen to be dependent on the complexity of the application, and increases as the number of cores in the CMP design increases. Overall, the CAPPS-automated system-level synthesis framework generates the solution set and tradeoff statistics in a matter of hours, instead of days or even weeks it would have taken with a gate-level estimation flow. Such a framework is very useful for designers early in the design flow, for quick and reliable communication architecture exploration, to guide design decisions at the system level.

VII. Conclusion and Future Work

We presented the CAPPS-automated framework for fast system-level application-specific power–performance tradeoffs in bus matrix CAPPS. Detailed energy macromodels for the bus matrix communication architecture were created using multiple regression analysis. These energy macromodels were shown to have an average cycle energy estimation error of less than 5% across the 180–65 nm standard cell libraries, compared with gate-level estimation. Plugging these macromodels in a system-level simulation framework allows a substantial speedup ranging from 1000 to 2000 $\times$ for cycle energy estimation, over gate-level estimation. These macromodels were used in the CAPPS synthesis framework for power estimation. Experimental results of applying CAPPS on three industrial networking CMP applications generated results in a matter of hours, indicating a potential tradeoff between power and performance, energy and runtime, and chip area and power, for different points in the solution space, which shows the effectiveness of our approach. Our ongoing work is dealing with applying this approach to other types of communication architectures such as ring buses and NoCs, and handling multimode applications. Our future work will evaluate our approach on designs with multithreshold synthesis.

References

[1] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.

[2] K. Lahiri and A. Raghunathan, "Power analysis of system-level on-chip communication architectures," in *Proc. CODES + ISSS*, 2004, pp. 236–241.

[3] ARM Inc., San Diego, CA, "AMBA Specification and Multi Layer AHB Specification (rev2.0)," 2001. [Online]. Available: http://www.arm.com

[4] IBM, Armonk, NY, "On-chip CoreConnect Bus Architecture," 2009. [Online]. Available: www.chips.ibm.com/products/coreconnect/index.html

[5] Synopsys Inc., Mountain View, CA, "AMBA AHB Interconnection Matrix," 2009. [Online]. Available: www.synopsys.com/products/designware/amba_solutions.html

[6] L. Benini and G. D. Micheli, "Networks on chips: A new SoC paradigm," *IEEE Comput.*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

[7] F. Angiolini, "Contrasting a NoC and a traditional interconnect fabric with layout awareness," in *Proc. DATE*, 2006, pp. 1–6.

[8] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Constraint-driven bus matrix synthesis for MPSoC," in *Proc. ASPDAC*, 2006, pp. 30–35.

[9] K. Lahiri, "Battery driven system design: A new frontier in low power design," in *Proc. ASPDAC*, 2002, pp. 261–267.

[10] M. Caldari, "System-level power analysis methodology applied to the AMBA AHB bus," in *Proc. DATE*, 2003, pp. 32–37.

[11] L. Shang, "Thermal modeling characterization and management of on-chip networks," in *Proc. MICRO*, 2004, pp. 67–78.

[12] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Fast exploration of bus-based on-chip communication architectures," in *Proc. CODES + ISSS*, 2004, pp. 242–247.

[13] Open SystemC Initiative (OSCI), "SystemC," 1999. [Online]. Available: www.systemc.org

[14] Synopsys, Inc., Mountain View, CA, "Synopsys coreTools, PrimePower," 2007. [Online]. Available: www.synopsys.com

[15] M. Gasteier and M. Glesner, "Bus-based communication synthesis on system level," *Proc. ACM TODAES*, vol. 4, no. 1, pp. 1–11, Jan. 1999.

[16] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane, "Floorplan-aware automated synthesis of bus-based communication architectures," in *Proc. DAC*, 2005, pp. 565–570.

[17] M. Drinic, "Latency-guided on-chip bus network design," in *Proc. ICCAD*, 2000, pp. 2663–2673.

[18] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli, "Efficient synthesis of networks on chip," in *Proc. ICCD*, 2003, pp. 146–146.

[19] N. D. Liveris and P. Banerjee, "Power aware interface synthesis for bus-based SoC designs," in *Proc. DATE*, 2004, pp. 20864–20864.

[20] U. Ogras and R. Marculescu, "Energy and performance-driven NoC communication architecture synthesis using a decomposition approach," in *Proc. DATE*, 2005, pp. 352–357.

[21] J. Guo, "Energy/area/delay tradeoffs in the physical design of on-chip segmented bus architecture," in *Proc. SLIP*, 2006, pp. 75–81.

[22] H.-S. Wang, "Orion: a power-performance simulator for interconnection networks," in *Proc. MICRO*, 2002, pp. 294–305.

[23] A. Raghunathan, *High level Power Analysis and Optimization*. Norwell, MA: Kluwer, 1998.

[24] J. Chan, "NoCEE: energy macro-model extraction methodology for network on chip routers," in *Proc. ICCAD*, 2005, pp. 254–260.

[25] A. Bona, "System level power modeling and simulation of high-end industrial network-on-chip," in *Proc. DATE*, 2004, pp. 1–6.

[26] N. Dhanwada, "A power estimation methodology for systemC transaction level models," in *Proc. CODES + ISSS*, 2005, pp. 142–147.

[27] U. Neffe, "Energy estimation based on hierarchical bus models for power-aware smart cards," in *Proc. DATE*, 2004, p. 30300.

[28] N. Eisley and L. Peh, "High level power analysis of on-chip networks," in *Proc. CASES*, 2004, pp. 104–115.

[29] T. T. Ye, L. Benini, and G. D. Micheli, "Analysis of power consumption on switch fabrics in network routers," in *Proc. DAC*, 2002, pp. 524–529.

[30] Gnu R. [Online]. Available: http://www.gnu.org/software/r/R.html

[31] J. J. Faraway, *Linear Models With R*. Boca Raton, FL: CRC Press, 2004.

[32] C. Kretzschmar, "Why transition coding for power minimization of on-chip buses does not work," in *Proc. DATE*, 2004, pp. 10512–10512.

[33] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning: Enabling hierarchical design," *IEEE Trans. Very Large Scale (VLSI) Integr. Syst.*, vol. 11, no. 12, pp. 1120–1135, Dec. 2003.

[34] G. Palermo and C. Silvano, "PIRATE: A framework for power/performance exploration of network-on-chip architectures," in *Proc. PATMOS*, Sep. 2004.

[35] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane, "FABSYN: Floorplan-aware bus architecture synthesis," *IEEE Trans. Very Large Scale (VLSI) Integr. Syst.*, vol. 14, no. 2, pp. 241–253, Mar. 2006.

[36] J. Cong and D. Z. Pan, "Interconnect performance estimation models for design planning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 6, pp. 729–752, Jun. 2001.

[37] J. Cong and Z. Pan, "Interconnect delay estimation models for synthesis and design planning," in *Proc. ASPDAC*, 1999, pp. 97–100.

[38] A. E. Caldwell, "On wirelength estimations for row-based placement," *Proc. IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 9, pp. 4–11, Sep. 1999.

[39] J. Cong, Z. Pan, L. He, C.-K. Koh, and K.-Y. Khoo, "Interconnect design for deep submicron ICs," in *Proc. ICCAD*, 1997, pp. 478–485.

[40] Semiconductor Industry Assoc. (SIA), San Jose, CA, "National Technology Roadmap for Semiconductors," 1997.

[41] Univ. California, Berkeley, "Berkeley Predictive Technology Model," 2008. [Online]. Available: http://www.eas.asu.edu/~ptm/

[42] Cadence Design Systems, Inc., San Jose, CA, "Cadence PKS," 2008. [Online]. Available: www.cadence.com

[43] S. Murali, "An application-specific design methodology for on-chip crossbar generation," *IEEE Trans. Comput-Aided Design Integr. Circuits Syst.*, vol. 26, no. 7, pp. 1283–1296, 2007.

[44] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Fast exploration of bus-based communication architectures at the CCATB abstraction," in *Proc. CODES + ISSS*, Feb. 2008, vol. 7, pp. 1–32.

[45] ITRS, Geneva, Switzerland, "International Technology Roadmap for Semiconductors," 2005.

[46] U. Y. Ogras and R. Marculescu, "It's a small world after all: NoC performance optimization via long-range link insertion," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 7, pp. 693–706, Jul. 2006.

[47] Y. Hu, "Communication latency aware low power NoC synthesis," in *Proc. DAC*, 2006, pp. 574–579.

[48] K. Srinivasan, K. S. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 4, pp. 407–420, Apr. 2006.

[49] F. Li, "Compiler-directed proactive power management for networks," in *Proc. CASES*, 2005, pp. 137–146.

[50] D. Bertozzi, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 113–129, Feb. 2005.

Sudeep Pasricha (M'02) received the B.E. degree in electronics and communication engineering from the Delhi Institute of Technology, Delhi, India, in 2000, and the M.S. and Ph.D. degrees in computer science from the University of California, Irvine, in 2005 and 2008, respectively.

He has four years of work experience in semiconductor companies including STMicroelectronics and Conexant. He is currently an Assistant Professor in the Electrical and Computer Engineering Department, Colorado State University, Fort Collins. He is the author or coauthor of more than 30 research articles in peer-reviewed conferences and journals and has presented several tutorials in the area of on-chip communication architecture design at leading conferences. He recently coauthored the book *On-chip Communication Architectures* (Morgan Kauffman/Elsevier, 2008). His current research interests include the areas of multiprocessor embedded systems design, networks-on-chip (NoCs) and emerging interconnect technologies, system-level modeling languages, and design methodologies and computer architecture.

Dr. Pasricha was the recipient of a Best Paper Award at ASPDAC 2006, a Best Paper Award nomination at DAC 2005, and several fellowships and awards for excellence in research. He is currently on the Program Committee of the ISQED and VLSID conferences.

Young-Hwan Park (M'07) received the B.S. degree in electronics from Kyunghee University, Seoul, Korea, in 1997, and the M.S. degree from the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, in 2004. He is currently working toward the Ph.D. degree in the Electrical Engineering and Computer Science Department, University of California, Irvine.

Before coming to the United States, he spent five and half years at the Computer Division of Samsung Company, Korea, as a Hardware Engineer, and he has experience in NVIDIA and QUALCOMM as an Intern during his Ph.D. work. His current research interests include the areas of system-level design exploration methodology considering performance as well as power consumption, multimedia system on chip (SoC) design, computer architecture, and multiprocessor embedded systems.

Mr. Park was the recipient of the Best Paper Award at the International Symposium on Quality Electronic Design (ISQED) Conference in 2006 and was supported by the Semiconductor Research Corporation (SRC) from 2005 to 2008.

Fadi J. Kurdahi (M'87–SM'03–F'05) was born in Beirut, Lebanon. He received the B.Eng. degree from the American University of Beirut, Beirut, Lebanon, in 1981, and the M.S.E.E. and Ph.D. degrees from the University of Southern California, Los Angeles, in 1982 and 1987, respectively.

Since 1987, he has been a Faculty Member in the EECS and ICS Departments at the University of California at Irvine, where he is engaged in research on VLSI CAD and reconfigurable computing with applications in communication and multimedia systems.

Dr. Kurdahi served as the Program Chair and General Chair of the 1999 and 2000 International Symposium of System Synthesis, He also served on executive committees, steering committees, and technical program committees of numerous IEEE conferences. He was an Associate Editor for the IEEE TRANSACTIONS ON CICRUITS AND SYSTEMS II 1993–1995, and Topic Area Editor for reconfigurable computing for IEEE DESIGN AND TEST (2000–present). He was the corecipient of the 2001 IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATED (VLSI) SYSTEMS Best Paper Award and the 2006 ISQED Best Paper Award, as well as three distinguished paper awards at DAC, ASPDAC, and EuroDAC.

Nikil D. Dutt (M'84–SM'96–F'08) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign in 1989.

He is currently a Chancellor's Professor at the University of California, Irvine, with academic appointments in the CS and EECS Departments. His current research interests include embedded systems, electronic design automation, computer architecture, optimizing compilers, system specification techniques, distributed embedded systems, and formal methods.

Prof. Dutt currently serves as Editor-in-Chief of the *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, and an Associate Editor of the *ACM Transactions on Embedded Computer Systems (TECS)* and the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATED (VLSI) SYSTEMS. He was an ACM SIGDA Distinguished Lecturer during 2001–2002, and an IEEE Computer Society Distinguished Visitor for 2003–2005. He has served on the steering, organizing, and program committees of several premier CAD and Embedded System Design Conferences and workshops. He serves or has served on the Advisory Boards of the ACM SIGBED and ACM SIGDA, and previously served as Vice-Chair of the ACM SIGDA and of IFIP WG 10.5. He was the recipient of the Best Paper Awards at CHDL89, CHDL91, VLSIDesign2003, CODES + ISSS 2003, CNCC 2006, and ASPDAC 2006. He is an ACM Distinguished Scientist and the recipient of an IFIP Silver Core Award.