System-Level Power-Performance Trade-Offs in Bus Matrix Communication Architecture Synthesis

Sudeep Pasricha, Young-Hwan Park, Fadi J. Kurdahi, Nikil Dutt
Center for Embedded Computer Systems
University of California, Irvine, CA
{spasrich, younghwp, kurdahi, dutt}@uci.edu

ABSTRACT

System-on-chip communication architectures have a significant impact on the performance and power consumption of modern multi-processor system-on-chips (MPSoCs). However, customization of such architectures for an application requires the exploration of a large design space. Thus 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 trade-offs in bus matrix communication architecture synthesis. Our paper makes two specific contributions. First, we develop energy macro-models for system-level exploration of bus matrix communication architectures. Second, we incorporate these macro-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 macro-models incur less than 5% average absolute error compared to gate-level models. Furthermore, our bus matrix synthesis framework generates a tradeoff space with designs that exhibits an approximately 20% variation in power and 40% variation in performance on an industrial networking MPSoC application, demonstrating the utility of our approach.

Categories and Subject Descriptors: J.6 [**Computer-aided Design**]; B.7.2 [**Integrated circuits**]: Design; C.5.4 [**VLSI Systems**]

General Terms: Design, Experimentation, Performance

Keywords: Communication Architectures, Bus Matrix Synthesis, System-on-Chip, Power Estimation, Power-Performance Trade-offs

1. INTRODUCTION

The rapidly increasing complexity of Multi-Processor System-on-Chip (MPSoC) designs, coupled with poor global interconnect scaling in the deep sub-micron era, is making on-chip communication a critical factor affecting overall system performance and power consumption [1]. These communication architectures are not only a major source of performance bottlenecks for data intensive application 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 communication architectures such as hierarchical shared buses [3, 4], bus matrix [6] and network-on-chip (NoC) [7] have been proposed. While hierarchical shared bus architectures, such as those proposed by AMBA [3] and CoreConnect [4] have been extensively used in the past, they are often unable to meet the high bandwidth requirements of emerging applications. Network-on-Chips are able to overcome these bandwidth limitations, but can consume more area and significantly more power than shared bus-based designs in current lithographic processes [8]. Bus matrix architectures, such as the AMBA AHB bus matrix [6], are an evolution of the hierarchical shared bus scheme. They consist of several parallel buses that can provide superior bandwidth response, while keeping the simple, standard interface of bus-based communication architectures, which is essential in promoting IP reuse.

On-chip communication architectures, such as the bus matrix [6], typically have customizable topologies and parameters, which creates a vast exploration space [5]. Different configurations in this space can have vastly different power/energy and performance characteristics. While meeting the performance 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 which run on batteries and have a limited energy budget. 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 and an increase of 10 ºC in the operating temperature of an electronic device results in a 100% increase in its failure rate [9]. Reducing power consumption is therefore essential to ensure correct operation, and also reduce costs for expensive cooling and packaging equipment.

Consequently, designers require a way to effectively traverse the vast communication architecture exploration space during its design/ synthesis phase, to trade off power-performance characteristics. Due to the highly complex nature of modern applications, performing performance and especially power exploration at the lower RTL/gate levels can be excessively time consuming. 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 power estimation at the system level, which is not a trivial task.

1.1 Paper Overview and Contributions

In this paper, we address the issues highlighted above by proposing an automated synthesis framework to perform application-specific, system-level power-performance trade-offs for bus matrix communication architectures. Our key contributions in this paper are: *(i)* detailed energy macro-models for the bus matrix architecture, which can be used in any cycle-accurate simulation models for power estimation; and *(ii)* an automated synthesis framework for the bus matrix architecture, which uses these energy macro-models in the fast and accurate transaction-level CCATB [26] simulation abstraction-based environment in SystemC [25], to efficiently explore the power-performance design space of the generated set of solutions. Experimental results indicate a less than 5% average absolute error when our energy macro-models are compared with detailed gate-level estimations. Applying our synthesis framework on an industrial networking MPSoC application used for data packet processing and forwarding enabled a potential tradeoff of approximately 20% for power, and 40% for performance, for different points in the solution space, showing the effectiveness of the approach. To the best of our knowledge, this is the first piece of work dealing with creating

CODES+ISSS'06, October 22–25, 2006, Seoul, Korea.

energy macro-models and performing power-performance trade-offs for the bus matrix communication architecture.

2. RELATED WORK

There is a large body of work dealing with performance-oriented system-level synthesis of hierarchical shared bus [10, l2], NoCs [13] and, more recently, bus matrix [5] architectures. Some approaches also consider power-aware synthesis for hierarchical shared buses [16] and NoCs [17], but power-aware synthesis for bus matrix architectures has not been addressed. There have also been a few pieces of work which have looked at power-performance tradeoffs for segmented buses [18] and NoCs [19]. Our work differs from existing work in that we focus on the bus matrix communication architecture, for which we create an automated synthesis framework to enable exploration of power-performance trade-offs at the system level.

Several approaches [9, 21, 22] have looked at analyzing power early in the design cycle, at the system level, for on-chip communication architectures. Some of these approaches have used the idea of creating energy macro-models from gate-level power/energy estimations, and applied it in the context of the AMBA AHB hierarchical shared bus architecture [9], STBus interconnection network [22] and NoCs [21]. One of the goals of our work is to perform a detailed analysis of the bus matrix communication architecture to create such energy macro-models which can then be used in a system-level simulation environment for fast cycle energy/power estimation.

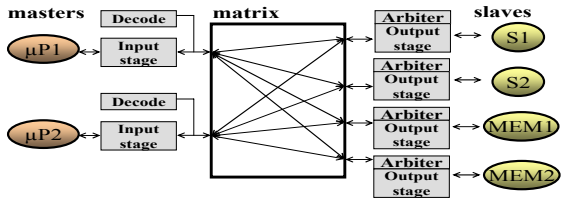

Figure 1. AMBA AHB Full Bus Matrix

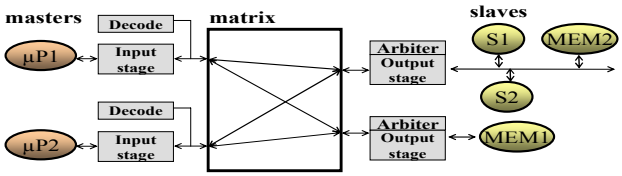

Figure 2. AMBA AHB Partial Bus Matrix

3. AMBA AHB BUS MATRIX OVERVIEW

We use the AMBA AHB (Advanced High Performance) bus matrix [6] 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. Figure 1 shows a 2 master, 4 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* which 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 busses within a bus matrix have the same data bus width, which usually depends on the application.

One drawback of the *full bus matrix* structure shown in Figure 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. Figure 2 shows a *partial bus matrix* which has fewer wires and components (e.g. arbiters, MUXs), which consequently reduces power consumption, as well as area, at the cost of performance. Our synthesis framework (described in Section 5) 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, which meet minimum performance constraints of the application being considered.

4. BUS MATRIX ENERGY MODELS

4.1 Background on Energy Macro Models

In order to obtain the energy consumption of a bus matrix communication architecture, we need to characterize it in terms of energy consumed at different phases of its execution. For this purpose, we create energy macro-models, which can be built with the knowledge of factors that have a strong correlation to energy consumption. A macro model consists of variables, which represent factors influencing energy consumption, and regression coefficients, which capture the correlation of the variables with energy consumption. A general energy macro model for a component can be expressed as:

$$E_{component} = \alpha_0 + \sum_{i=1}^{n} \alpha_i \cdot \Psi_i \qquad .. (1)$$

where α_0 is the energy of the component which is independent of the model variables, and α_i is the regression coefficient for the model variable Ψ_i.

For the purpose of our energy macro-models, we considered three types of model variables or factors influencing energy consumption: *control*, *data* and *structural*. The *control* factor represents control events, involving a control signal, which 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 macro model relation in Eq. (1). The *data* factor represents data events, which trigger energy consumption on data value changes. Data variables take an integer value in Eq. (1) representing the Hamming distance (number of bit-flips) of successive data inputs. 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 Eq. (1).

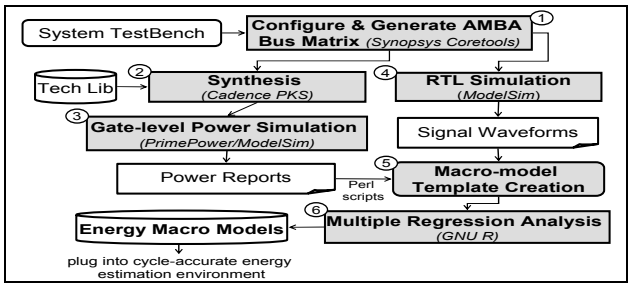

Figure 3. Energy Macro-model Generation Methodology

4.2 Methodology Overview

The methodology used to create energy macro models in this work is shown in Figure 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 which exercises the matrix under different conditions. Synopsys Coretools [27] is used to configure the bus matrix (specify data bus width, number of masters and slaves etc.) and generates a synthesizable RTL description of the bus matrix architecture (Step 1). This description is synthesized to the gate-level with Cadence Physically Knowledgeable Synthesis (PKS) [14], for

the target standard cell library (Step 2). The gate-level netlist is then used with PrimePower [27] to generate power numbers (Step 3).

In parallel with the synthesis flow, we perform RTL simulation to generate signal waveform traces for important data and control signals (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. The selected data and control events become the variables in our macro-model template (Step 5). This template consists of variable and energy values for every cycle of testbench execution, and is used as an input to the GNU R tool [23], which performs multiple regression analysis to find coefficient values for the chosen variables (Step 6).

Steps 1-6 can be repeated for testbenches having different structural attributes such as data bus widths and number of input components, to identify structural factors/variables that may influence cycle energy consumption. Once a good fit between cycle energy and macro model variables is found, the energy macro models are generated in the final step. These models can then be plugged into any cycle accurate energy estimation environment, to get energy consumption values for the AMBA AHB bus matrix communication architecture.

4.4 Bus Matrix Component Energy Models

We partitioned the energy macro-model of the entire AMBA AHB bus matrix communication architecture into sub-components, and then used our energy macro-model generation methodology to create macro-models for each of the sub-components. The total energy consumption of a bus matrix can be given as:

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

where E_{INP} and E_{DEC} are the energy for the input and decoder components for all the masters in 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 macro-models were created for the first four factors, with E_{WIRE} being calculated separately.

The energy macro-models are essentially of the form shown in Eq. (1). Leakage and clock energy, which are the major sources of independent energy consumption are considered as part of the static energy [1] coefficient α_0 for each of the components. Based on our experiments, we noticed a good linear correlation between cycle energy and macro-model variables for the components. We will now present these energy macro-models. Due to lack of space, the energy model for the *bus wires* (which also takes into account repeater energy) is presented in more detail in our technical report [24].

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:

$$E_{INP} = \alpha_{inp0} + \alpha_{inp1}.\Psi_{load} + \alpha_{inp2}.\Psi_{desel} + \alpha_{inp3}.\Psi_{HDin} + \alpha_{inp4}.\Psi_{drive}$$

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.

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

$$E_{DEC} = \alpha_{dec0}+\alpha_{dec1}.\Psi_{slavesel}+\alpha_{dec2}.\Psi_{respsel}+\alpha_{dec3}.\Psi_{HDin}+\alpha_{dec4}.\Psi_{sel}$$

where $\Psi_{slavesel}$ and $\Psi_{respsel}$ 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 which transitions when the decoder is selected or deselected.

Output Stage: Every slave is connected to the bus matrix through the output stage, which handles multiplexing of 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:

$$E_{OUT} = \alpha_{out0}+\alpha_{out1}.\Psi_{addrsel}+\alpha_{out2}.\Psi_{datasel}+\alpha_{out3}.\Psi_{HDin}+\alpha_{out4}.\Psi_{noport}$$

where $\Psi_{addrsel}$ and $\Psi_{datasel}$ 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.

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:

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

where Ψ_{arb} and Ψ_{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 $\Psi_{desel+1}$ 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 MATRIX SYNTHESIS FRAMEWORK

5.1 Background

Typically, MPSoC designs have performance constraints, which are dependent on the nature of the application, and must be satisfied by the underlying communication architecture. The *throughput* of communication between components is a good measure of the performance of a system [10]. To represent performance constraints in our approach, we define a **Communication Throughput Graph** $CTG = G(V,A)$ which 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 sub-graph of a CTG, consisting of a single component for which data throughput must be maintained, and other masters, slaves and memories which are in the critical path that impacts the maintenance of the throughput. Figure 4 shows a CTG for a network 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 Gbps.

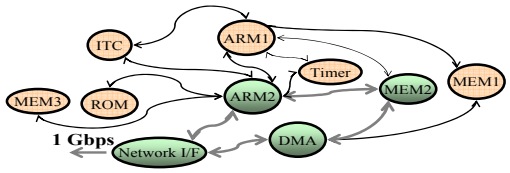

Figure 4. Communication Throughput Graph (CTG)

5.2 Assumptions and Goals

We are given an MPSoC application which 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 which 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, which meet the minimum performance constraints of the

[1] In the rest of the paper, the term *static energy* will be used to refer to the sum of leakage energy and clock energy

application. The output of the framework is a power-performance trade-off 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.

5.3 Framework Overview

We now describe our bus matrix synthesis framework in more detail. This framework has been derived from our previous work on bus matrix synthesis, which generated a single, optimal partial bus matrix solution having the least number of buses [5]. This paper extends the framework by adding energy macro-models, and enabling power-performance trade-offs 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, which 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 having different number of components and buses, without violating application performance constraints, 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 trade-off graph for the solution set will then allow a designer to select a configuration having the desired power and performance characteristics for a particular application.

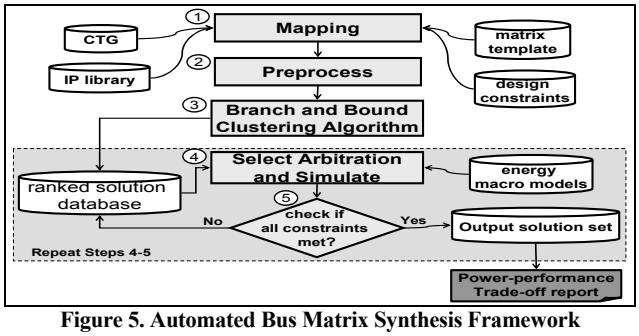

Figure 5. Automated Bus Matrix Synthesis Framework

Figure 5 shows the major steps in our synthesis framework. The inputs are *(i)* the application-specific *Communication Throughput Graph* (CTG), *(ii)* a library of IP components consisting of masters, slaves and memories, *(iii)* a template of the target full bus matrix communication architecture (e.g. AMBA AHB bus matrix), *(iv)* energy macro-models for the bus matrix, and *(v)* designer constraints, which allow the designer to set the bus matrix bus clock frequency, data bus width (assumed to be uniform for all buses in the matrix) and arbitration schemes. The output is a set of solutions which meet minimum application performance constraints, and a power-performance trade-off report for these solutions.

We start by first mapping components in the IP library onto the full bus matrix template (Step 1). Next, we perform preprocessing on this matrix structure by *(i)* removing unused buses with no data transfers on them, according to the CTG, and *(ii)* migrating components which are accessed exclusively by a master, to its local bus, 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, which 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 *(i)* only beneficial clusterings are allowed, which result in component or wire savings and *(ii)* performance constraints are still valid after clustering. The interested reader can refer to our previous work in [5] for a more detailed treatment of this algorithm.

The output of the algorithm is a set of solutions which 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: *(i)* fixes arbitration schemes for each slave cluster in the matrix, and *(ii)* performs simulation, to gather power and performance statistics. The arbitration scheme gives priority to higher *Throughput Constraint Paths* (TCPs) from the CTG. Simulation is performed using a fast, transaction-level CCATB [26] simulation model in SystemC [25]. It allows us to verify if all 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) trade-off graphs.

5.3.1 Linking Energy Macro-Models with CCATB Models

The CCATB simulation abstraction uses function/transaction calls instead of signals, to speed up the simulation-based estimation process [26]. It incorporates bus matrix energy macro-models for energy estimation. The equations from Section 4.4 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 1. 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 macro-models, we present an example of a single write transaction and show how it activates the macro-models as it propagates through the bus matrix. Table 1 gives the dependent variables from Section 4.4, for each of the components in the matrix, which are triggered during the write transaction in the model, assuming no slave delays and a single cycle overhead for the arbitration. Every change in a dependent variable for a component triggers its corresponding energy macro-model 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 trade-off analysis.

6. EXPERIMENTS

6.1 Accuracy of Energy Macro-Models

We compared the accuracy of our bus matrix energy macro-models with gate-level energy models. First, we selected a diverse set of bus matrix based system testbenches, which activate the components in the bus matrix in different states. Then we generated the macro-models (Section 4.4) and obtained their regression coefficients using the methodology from Figure 3, for the TSMC 0.18µm standard cell library. Due to lack of space, detailed regression coefficient values for the macro-models are presented in our technical report [24].

Next, we selected 4 system testbenches having different structures and

traffic characteristics: *(i)* a 2x3 bus matrix with 32 bit data bus width (2x3_32b), *(ii)* a 3x4 bus matrix with 32 bit data bus width (3x4_32b), *(iii)* a 4x5 bus matrix with 32 bit data bus width (4x5_32b) and *(iv)* a 2x3 matrix with 64 bit data width (2x3_64b). We synthesized these applications using the TSMC 0.18µm standard cell library and estimated cycle energy using PrimePower, at the gate-level. In parallel, we characterized each of the energy macro-model variables with their calculated coefficient values. Figure 6 compares the error in energy estimated by the macro-models compared to gate-level estimation, for the selected testbenches. From the figures, it can be seen that the maximum average error is 2.03%, while the maximum absolute average error (average of absolute error values at each cycle) is 4.89%.

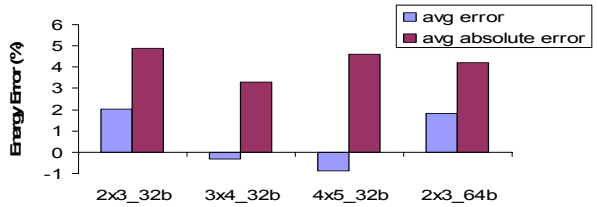

Figure 6. Energy macro-model estimation errors

Next, we plugged the energy macro-models into a SystemC transaction-level CCATB simulation environment [26]. Figure 7 shows a snapshot of the power waveform generated by gate-level simulation using PrimePower, and the SystemC CCATB simulation using our energy macro-models for a 2x3 bus matrix system testbench (2x3_32b in Figure 6). The SystemC CCATB model allows extremely accurate power profiling with a maximum absolute average estimation error of approx. 5%, and a more than **1000x** speedup over PrimePower based estimation. As can be seen, the power estimates from the CCATB simulation 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 an important but becoming increasingly difficult to do in DSM.

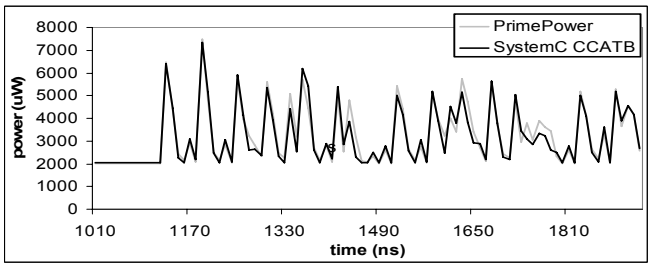

Figure 7. Predicted and measured power waveforms

6.2 Power-Performance Trade-offs

To validate our automated bus matrix synthesis framework, we applied it to an industrial MPSoC application from the networking domain, used for data packet processing and forwarding. Figure 8 shows the Communication Throughput Graph (CTG) for the application, with its minimum performance constraints that need to be satisfied, represented by Throughput Constraint Paths (TCPs), presented separately in Table 2.

ARM1 is a protocol processor (PP), while ARM2 and ARM3 are network processors (NP). 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 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*).

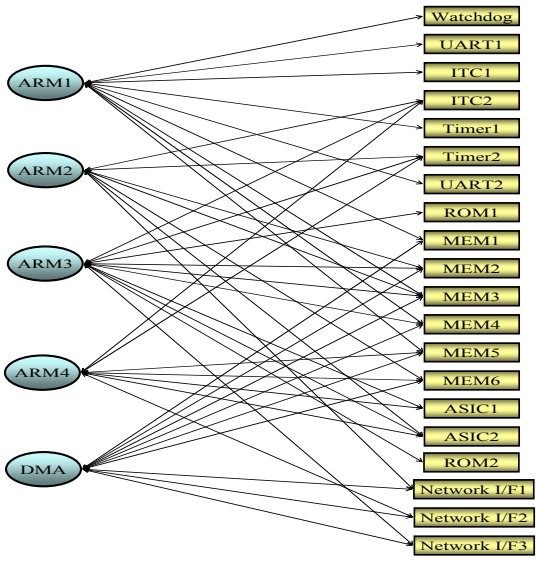

Figure 8. CTG for networking MPSoC application

Table 2. Throughput Constraint Paths (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

The application and the target AMBA AHB bus matrix architecture were captured at the CCATB abstraction in SystemC [25, 26], with the bus matrix energy macro-models 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 0.18µm standard cell library. We also included the energy contribution of bus wires. The Berkeley Predictive Technology Model (PTM) [11] is used to estimate capacitance values, a high-level simulated-annealing based floorplanner [20] is used to generate IP block placement to obtain bus wire lengths and optimal-delay repeater spacing/sizing is obtained from [15]. More details can be found in our technical report [24].

Figure 9 shows the power-performance curves output by the framework for the MPSoC application. The *x-axis* shows solutions in the output set having different number of buses, while the *y-axis* shows the % change in power and performance, using the original full bus matrix as the base case. It can be seen that *reducing the number of buses reduces the average power dissipation*, because of a smaller number of arbiters, output stages and bus wires in the matrix, which results in less static and dynamic power consumption. Figure 10 highlights this trend, showing 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 clustered slaves.

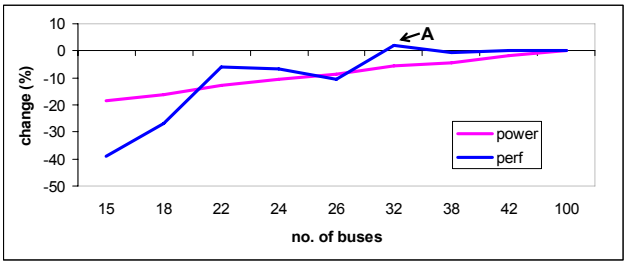

Figure 9. Power-Performance Trade-off graph

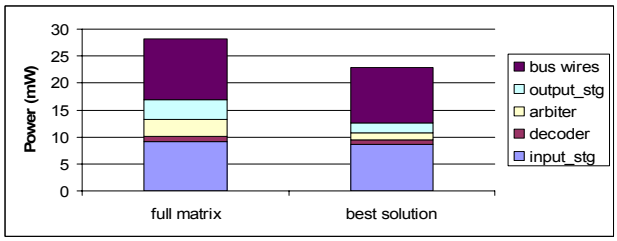

Figure 10. Component-wise Power Comparison

As far as the performance change is concerned, (measured in terms of average % 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 Figure 9 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 re-arbitration delays for masters. In addition to the average % change in throughput, another useful metric to gauge application performance is its total runtime. Figure 11 shows the corresponding total energy vs. application runtime trade-off graph for the MPSoC, which is useful for cases where the application must execute within a given time or energy budget, such as for mobile battery-driven applications.

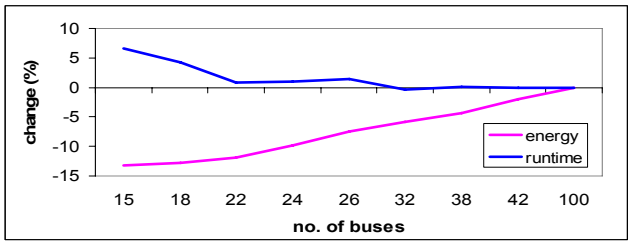

Figure 11. Total Energy vs. Runtime Trade-off graph

Overall, the power-performance curves show a possible trade-off of upto approximately 20% for power and upto 40% for performance, enabling a designer to select the appropriate point in the solution space which meets the desired power and performance characteristics. The automated system-level synthesis framework generated the solution set and statistics in a matter of a few hours, instead of days or even weeks it would have taken with a gate-level estimation flow. Such a framework is invaluable for designers early in the design flow, for quick and reliable communication architecture design space exploration, to guide design decisions at the system level.

7. ACKNOWLEDGMENTS

This research was partially supported by grants from SRC (2005-HJ-1330) and a CPCC fellowship.

8. CONCLUSION

We presented an automated framework for fast system-level application-specific power-performance trade-offs in bus matrix communication architecture synthesis. Detailed energy macro-models for the bus matrix communication architecture were created using multiple regression analysis, and shown to have a maximum average absolute error of less than 5%, compared to gate-level estimation. These macro-models were plugged into our bus matrix synthesis framework, at the system level. Experimental results of applying our synthesis framework on an industrial networking MPSoC application generated results in a matter of a few hours, indicating a potential tradeoff of upto 20% for power, and 40% for performance, for different points in the solution space, which shows the effectiveness of our approach. Future work will deal with applying this approach to other types of communication architectures.

REFERENCES

[1] R. Ho, K. W. Mai, M. A. Horowitz, "The Future of Wires", *Proc. IEEE, vol. 89, April 2001*

[2] K. Lahiri, A. Raghunathan, "Power Analysis of system-level on-chip communication architectures", *CODES+ISSS 2004*

[3] ARM AMBA Specification and Multi layer AHB Specification, (rev2.0), *http://www.arm.com, 2001*

[4] "IBM On-chip CoreConnect Bus Architecture", *www.chips.ibm.com/products/coreconnect/index.html*

[5] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Constraint-Driven Bus Matrix Synthesis for MPSoC", *ASPDAC 2006*

[6] AMBA AHB Interconnection Matrix, *www.synopsys.com/products/designware/amba_solutions.html*

[7] L.Benini, G.D.Micheli, "Networks on Chips: A New SoC Paradigm", *IEEE Computers, Jan. 2002*

[8] F. Angiolini et al., "Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness", *DATE 2006*

[9] M. Caldari et al. "System-level power analysis methodology applied to the AMBA AHB bus", *DATE 2003*

[10] M. Gasteier, M. Glesner, "Bus-based communication synthesis on system level", *ACM TODAES, January 1999*

[11] Berkeley Predictive Technology Model, U.C. Berkeley, *http://www-devices.eecs.berkeley.edu/~ptm/*

[12] S. Pasricha, N. Dutt, E. Bozorgzadeh, M. Ben-Romdhane, "Floorplan-aware Automated Synthesis of Bus-based Communication Architectures", *DAC 2005*

[13] A. Pinto, L. P. Carloni, A. L. Sangiovanni-Vincentelli, "Efficient Synthesis of Networks On Chip," *ICCD 2003*

[14] Cadence PKS, *www.cadence.com/datasheets/pks_ds.pdf*

[15] J. Cong, D. Z. Pan, "Interconnect Performance Estimation Models for Design Planning", *IEEE TCAD, June 2001*

[16] N.D. Liveris, P. Banerjee, "Power aware interface synthesis for bus-based SoC designs", *DATE 2004*

[17] U. Ogras, R. Marculescu, "Energy and Performance-Driven NoC Communication Architecture Synthesis using a Decomposition Approach", *DATE 2005*

[18] J. Guo et al., "Energy/area/delay trade-offs in the physical design of on-chip segmented bus architecture", *SLIP 2006*

[19] H-S. Wang et al., "Orion: a power-performance simulator for interconnection networks", *MICRO 2002*

[20] S. N. Adya, I. L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design", *IEEE TVLSI, Dec. 2003*

[21] J. Chan et al., "NoCEE: energy macro-model extraction methodology for network on chip routers", *ICCAD 2005*

[22] A. Bona et al., "System level power modeling and simulation of high-end industrial network-on-chip", *DATE 2004*

[23] GNU R, *http://www.gnu.org/software/r/R.html*

[24] S. Pasricha, Y. Park, F. Kurdahi, N. Dutt, "CAPPS: A Framework for Power-Performance Trade-Offs in On-Chip Communication Architecture Synthesis", CECS Technical Report, *Nov 2006*

[25] SystemC initiative, *www.systemc.org*

[26] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Fast Exploration of Bus-based On-chip Communication Architectures", *CODES+ISSS 2004*

[27] Synopsys CoreTools, PrimePower *www.synopsys.com*