CAPPS: A Framework for Power-Performance Trade-Offs in On-Chip Communication Architecture Synthesis *

Sudeep Pasricha, Young-Hwan Park, Fadi J. Kurdahi and Nikil Dutt

Center for Embedded Computer Systems
University of California Irvine
Irvine, CA 92697-3425, USA
{spasrich, younghwp, kurdahi, dutt}@uci.edu

CECS Technical Report #06-12
November, 2006

Abstract

On-chip communication architectures have a significant impact on the power consumption and performance 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 technical report we present an automated framework (CAPPS) for fast system-level, application-specific, power-performance trade-offs in bus matrix communication architecture synthesis. Our technical report 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 cycle energy error across 180, 130 and 90nm technology libraries. Our early system-level power estimation approach also shows a significant speedup of as much as* ***2000X*** *when compared with detailed gate-level power estimation. Furthermore, our bus matrix synthesis framework generates a tradeoff space with designs that exhibit an approximately* ***20%*** *variation in power and* ***40%*** *variation in performance for an industrial networking MPSoC application, demonstrating the usefulness of our approach.*

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

CAPPS: A Framework for Power-Performance Trade-Offs in On-Chip 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

On-chip communication architectures have a significant impact on the power consumption and performance 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 technical report we present an automated framework for fast system-level, application-specific, Power-Performance trade-offs in bus matrix Communication Architecture Synthesis (CAPPS). Our technical report 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 cycle energy error across 180, 130 and 90nm technology libraries. Our early system-level power estimation approach also shows a significant speedup of as much as **2000X** when compared with detailed gate-level power estimation. Furthermore, our bus matrix synthesis framework generates a tradeoff space with designs that exhibit an approximately **20%** variation in power and **40%** variation in performance for an industrial networking MPSoC application, demonstrating the usefulness of our approach.

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 on-chip communication architectures such as hierarchical shared buses [3] [4], bus matrices [5] and network-on-chip (NoC) [6] 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 [7]. Bus matrix architectures, such as the AMBA AHB bus matrix [5], 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. Such bus matrix architectures are being increasingly used in designs today.

On-chip communication architectures such as the bus matrix typically have customizable topologies and parameters, which creates a vast exploration space [8]. 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 [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. An increase of even 10 ºC in the operating temperature has been shown to not only increase interconnect delay (reducing performance), but also increase electromigration (EM), which significantly increases device failure rate [10]. Since interconnect temperatures can reach as high as 90 ºC [11], reducing power consumption becomes essential to ensure correct operation, and also to 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 reliable power estimation at the system level, which is not a trivial task.

1.1 Report Overview and Contributions

In this technical report, we address the issues highlighted above by proposing an automated synthesis framework (CAPPS) to perform application-specific, system-level power-performance trade-offs, for bus matrix communication architectures. Our key contributions in this technical report are: *(i)* detailed energy macro-models for the bus matrix architecture, which can be used in any cycle-accurate simulation models for power estimation across technology libraries; 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 [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 macro-modeling approach with respect to detailed gate-level estimation using Synopsys PrimePower [14] for 180, 130 and 90nm technology libraries. Moreover, our system-level power estimation approach achieves upto 2000X speedup over gate-level estimation when the macro-models are plugged into the transaction-level CCATB [12] simulation environment in SystemC [13]. Applying our synthesis framework on a networking industrial 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 our approach.

2. RELATED WORK

There is a large body of work dealing with performance-oriented system-level synthesis of hierarchical shared bus [15-17], NoCs [18] and, more recently, bus matrix/crossbar [8] [43] architectures. Some approaches also consider power-aware synthesis for hierarchical shared buses [19] and NoCs [20], 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 [21] and NoCs [22]. 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.

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 communication architectures at the system level, where significant improvements in run time can be achieved, at the cost of estimation accuracy. Some of these approaches extract gate-level power estimates and characterize transaction-level models [26-27]. Although these approaches allow for fast estimation, they can be highly inaccurate and lack reusability across technology libraries. Other approaches have created energy macro-models from gate-level power estimates for the AMBA AHB hierarchical shared bus [10], STBus interconnection network [25] and NoCs [24]. One of the goals of our work is to create such energy macro-models for the bus matrix communication architecture that can then be used in a system-level simulation environment for fast cycle energy/power estimation. We also explore the effect of technology library scaling on system-level energy macro-model based estimation time and accuracy.

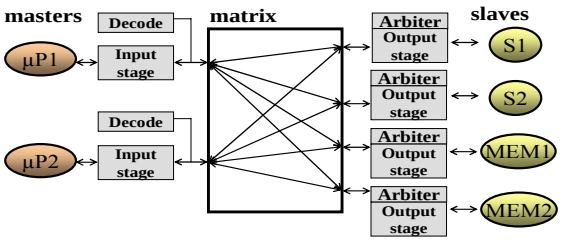

Figure 1. AMBA AHB Full Bus Matrix

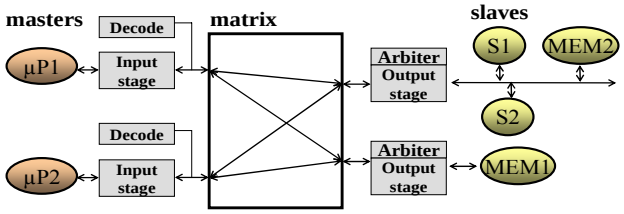

Figure 2. AMBA AHB Partial Bus Matrix

3. AMBA AHB BUS MATRIX OVERVIEW

We use the AMBA AHB (Advanced High Performance) bus matrix [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. 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* 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 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* 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 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 that meet all the performance constraints of the application being considered.

4. BUS MATRIX ENERGY MODELS

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

4.1 Background on Energy Macro Models

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 macro-models that can encapsulate events or factors having a strong correlation to energy consumption for a given component. A macro model 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 macro model for a component can be expressed as:

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

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.

For the purpose of our energy macro-models, 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 macro model relation in Eq. (1). The *data* factor represents data events that 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).

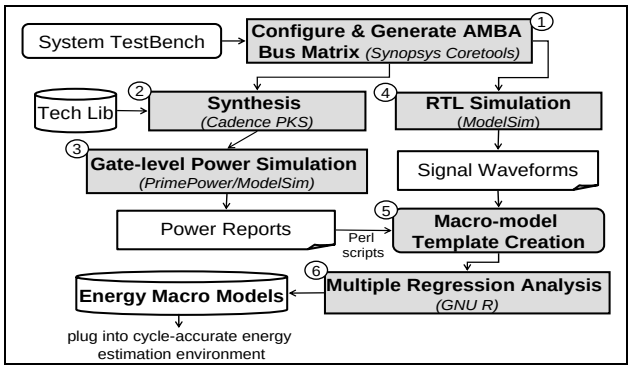

Figure 3. Energy Macro-model Generation Methodology

4.2 Methodology Overview

A high level overview of 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 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 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).

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

Figure 4. Macro-model Template

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 a macro-model template that consists of energy and variable values for each cycle of testbench execution (Step 5). Figure 4 shows an example of a macro-model 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 input components, 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 macro model variables is found, the energy macro models 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.

4.3 Bus Matrix Component Energy Models

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

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

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 macro-models were created for the first four components, 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 coefficient α_0 for each of the components. Based on our experiments, we noticed a fairly linear relationship between cycle energy and macro-model variables for the components. We will now present the energy models for each of the components.

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:

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

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.

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:

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

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:

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

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:

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

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.

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:

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

where V_{dd} is the supply voltage, α is the switching factor representing bit transition activity on the wire, ΣC_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 inter-repeater distance and E_{REP} is the repeater internal energy.

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. Repeater capacitance and static energy is obtained from data sheets. A high-level simulated-annealing based floorplanner [33] is used to generate IP block placement to obtain wire lengths and [34] is used to determine optimal-delay repeater spacing/sizing. Finally, the switching factor (α) is obtained from simulation.

5. CAPPS SYNTHESIS FRAMEWORK

In this section, we describe the CAPPS automated framework for fast system-level, application-specific, power-performance trade-offs in bus matrix communication architecture synthesis. First we describe how performance constraints are represented 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.

5.1 Background

Typically, MPSoC designs have 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 define a **Communication Throughput Graph** *CTG = G(V,A)* that 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 that are in the critical path impacting the maintenance of the throughput. Figure 5 shows a CTG for a 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 Gbps.

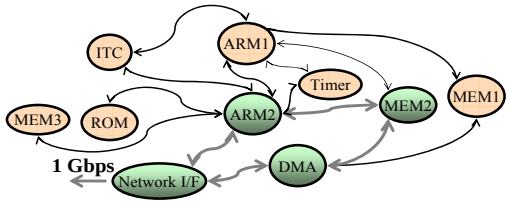

Figure 5. Communication Throughput Graph (CTG)

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 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. 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 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. Figure 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 0.13 µm 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 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 busses 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 re-verification 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, preferably at the system level. To detect and eliminate any possible bus cycle time violations in the bus matrix communication architecture solutions generated by our synthesis framework, we make use of high-level *floorplanning* and *wire delay estimation engines*.

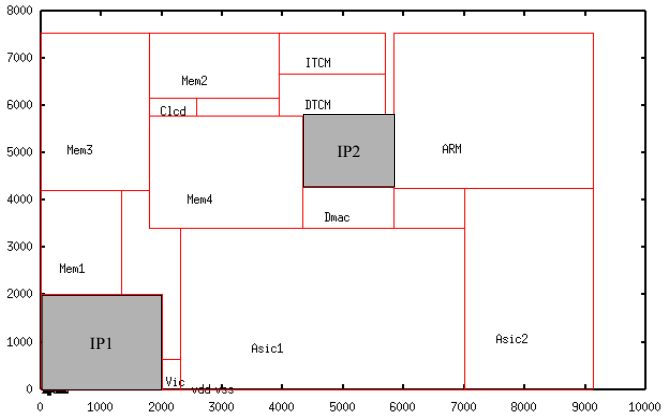

Figure 6. Example floorplan with bus cycle timing violation

The floorplanning stage in a typical design flow arranges arbitrarily shaped, but usually rectangular blocks representing circuit partitions, into a non-overlapping 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 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

$$Cost = w_1.Area + w_2.Bus_{WL} + w_3.Total_{WL} \qquad \ldots (8)$$

where *Area* is the area of the chip, Bus_{WL} is the wire length corresponding to wires connecting components on a bus, $Total_{WL}$ is total wire length for all connections on the chip (including inter-bus connections) and w_1, w_2, w_3 are adjustable weights that are used to bias the solution. The floorplanner outputs a non overlapping 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 [36-37], for optimal wire sizing 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 *(i)* the wire lengths from the floorplanner, *(ii)* 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]), *(iii)* technology library dependent parameters (from [40]) and *(iv)* the capacitive loads (C_L) of component output pins (obtained from RTL synthesis).

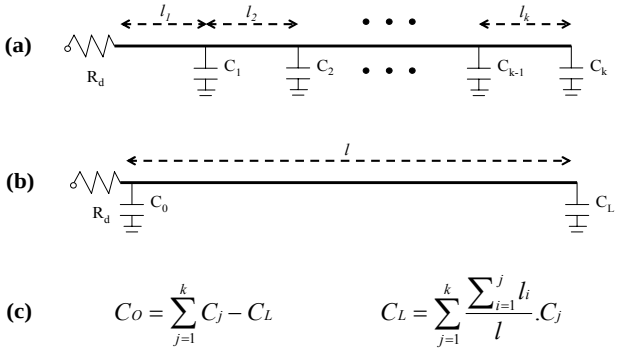

Figure 7. Transforming multiple pin net into two pin net

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

$$T_{OWS} = R_d C_o + \left(\frac{\alpha_1 l}{W^2(\alpha_2 l)} + \frac{2\alpha_1 l}{W(\alpha_2 l)} + R_d c_f + \sqrt{R_d r c_a c_f l} \right) . l$$
$$= t_{driver} + T'_{ows}(R_d, l, C_L) \qquad \ldots (9)$$

where $\alpha_1 = \frac{1}{4} r c_a$, $\alpha_2 = \frac{1}{2} \sqrt{\frac{r c_a}{R_d C_L}}$ 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_O and C_L are capacitive loads that are calculated as shown in Figure 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 fF/μm^2 and c_f is unit fringing capacitance in fF/μm (defined to be the sum of fringing and coupling capacitances).

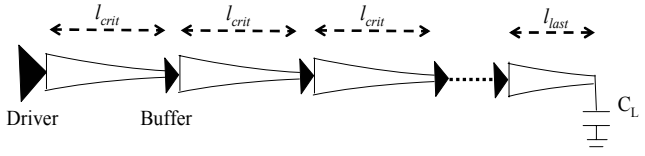

Figure 8. Wire model for BIWS optimization

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 Eq. (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 Figure 8. Except for the first and last stages, each stage has equal length l_c and an equal delay T_{crit} given by

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

where t_{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_{first} given by

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

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

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

Then the delay for a wire of length l, with optimal Buffer Insertion and Wire Sizing (BIWS), shown in Figure 8, can be summarized as

$$T_{BIWS} = T_{OWS} \quad \text{for } l < l_c \quad \text{... (13)}$$

$$= T_{first} + T_{last} \quad \text{for } l_c < l < 2l_c \quad \text{... (14)}$$

$$= T_{first} + n_b.T_{crit} + T_{last} \quad \text{for } l > 2l_c \quad \text{... (15)}$$

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

5.4 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 technical report 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 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 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.

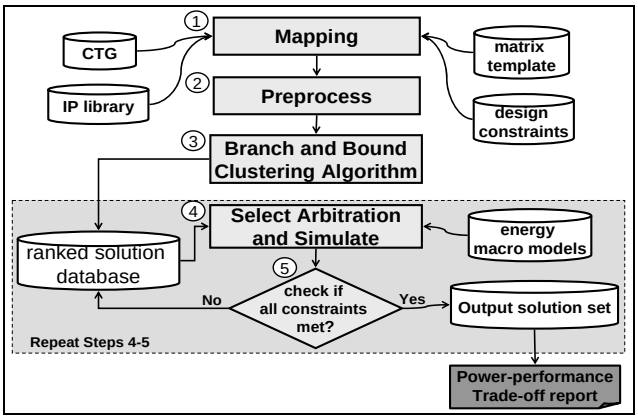

Figure 9. CAPPS Bus Matrix Synthesis Framework

Figure 9 shows the major steps in the CAPPS 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, that allow a 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 that meet 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 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 *(i)* only beneficial clusterings are allowed that 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 [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: *(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 [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) trade-off graphs.

5.4.1 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 [12]. It incorporates bus matrix energy macro-models for energy estimation. The equations from Section 4.3 are inserted in the code, at points where a change in the value of an energy consuming event (i.e. dependent variable) can occur.

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 4.3, 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 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.

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}

6. EXPERIMENTS

6.1 Energy Macro-model Generation

In order to generate the energy macro-models for the AHB bus matrix communication architecture, we first selected a diverse set of bus matrix based system testbenches, having different number of master and slave components, bus widths, and data traffic profiles that activate the components in the bus matrix in different states. We used these to generate the energy macro-models and obtained the component regression coefficients using the methodology from Figure 3, for the TSMC 180nm standard cell library. The regression coefficients for the macro-models of each of the components of the bus matrix (Section 4.3) are shown in Table 2. Also shown are the R squared (R^2) values which measure the goodness of fit for each regression [31]. The value of R2 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 R2 values are close to 1.0, thus enabling reliable prediction of energy at the system-level.

Table 2. Coefficients for Energy Macro-Models (180nm)

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			

Next, we targeted the same system testbenches to the 130nm and 90nm 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 shown in Table 3. 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 α_{inp3} is less than $1/3^{rd}$ the value of the coefficient for α_{inp4} at 180nm, but this ratio reduces to 1/2 for the 90nm library.

Table 3. 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

A major advantage of our energy macro-model 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 macro-model template shown in Figure 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.*

6.2 Accuracy of Energy Macro-Models

We performed experiments to determine the macro-model accuracy for different technology libraries, by comparing our macro-model 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 macro models, and had different bus matrix structures and traffic characteristics: *(i)* a 2 master, 3 slave bus matrix with 32 bit data bus width (2x3_32b), *(ii)* a 3 master, 4 slave bus matrix with 32 bit data bus width (3x4_32b), *(iii)* a 4 master, 5 slave bus matrix with 32 bit data bus width (4x5_32b) and *(iv)* a 2 master, 3 slave bus matrix with 64 bit data width (2x3_64b). We synthesized these architectures and estimated cycle energy values using PrimePower [14] at the gate-level, for each of the 180, 130 and 90nm TSMC standard cell libraries. In parallel, we characterized the model variables and coefficient values in the energy macro-models for each of the components in the matrix, to obtain estimated energy values.

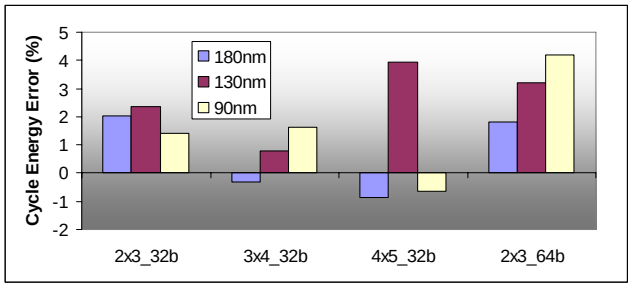

Figure 10. Average cycle energy estimation errors

Figure 10 compares the average error in energy estimated at each cycle by the macro-models at the early system-level, compared to gate-level estimation, for the different standard cell libraries. It can be seen that the energy macro-models allow highly accurate cycle energy estimation that is unaffected by the scaling of technology libraries towards deep sub-micron (DSM). *The maximum average estimation error for the macro-models is only 4.19%.*

Next, we plugged the energy macro-models 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 which capture signal details, without sacrificing accuracy [12]. The simulation model incorporates bus matrix energy macro-models for cycle energy estimation. The equations from Section 4.3 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.

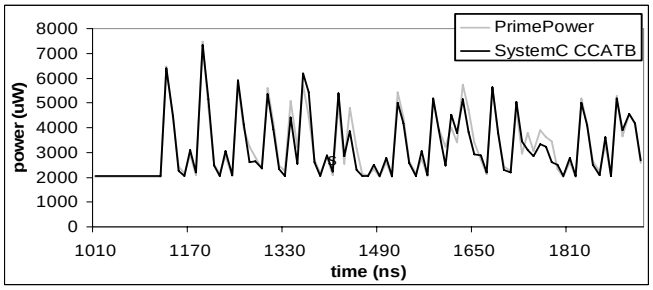

Figure 11. Predicted and measured power waveforms

Figure 11 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 the 2x3_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 4. 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

Table 4 compares the CPU time taken for the PrimePower simulation (for gate-level cycle energy estimation) with the 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. *As can be seen from the table, the system-level power estimation gives a substantial speedup close to **2000X** 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 macro-models across technology libraries, we believe that our energy macro-modeling 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.

6.3 Power-Performance Trade-offs

To validate the CAPPS bus matrix synthesis framework, we applied it to an industrial MPSoC application from the networking domain, used for data packet processing and forwarding. Figure 12 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 5.

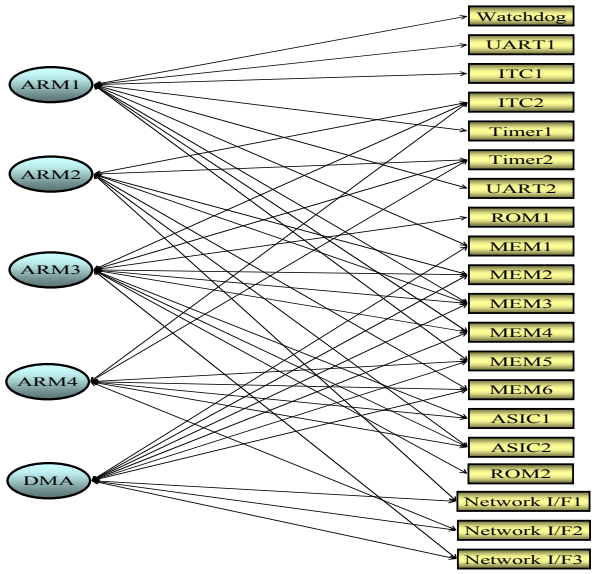

Figure 12. CTG for networking MPSoC application

Table 5. 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

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*).

The application and the target AMBA AHB bus matrix architecture were captured at the CCATB abstraction [12] in SystemC [13], 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 as described in Section 4.3.

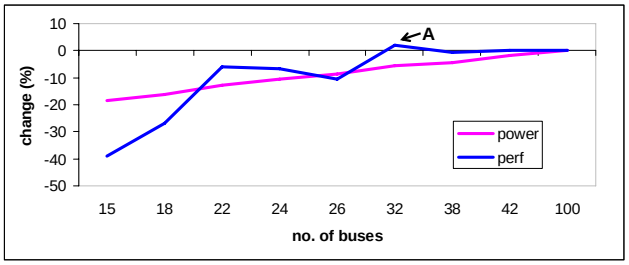

Figure 13. Power-Performance Trade-off graph

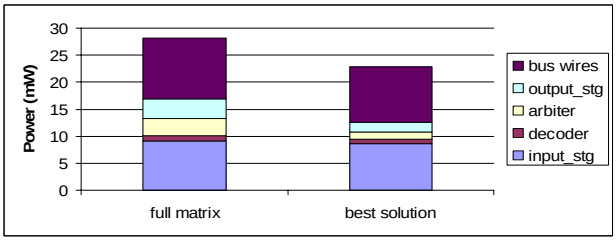

Figure 14. Component-wise Power Comparison

Figure 13 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 14 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 increasingly clustered slaves.

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 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 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 15 shows the corresponding total energy vs. application runtime trade-off 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.

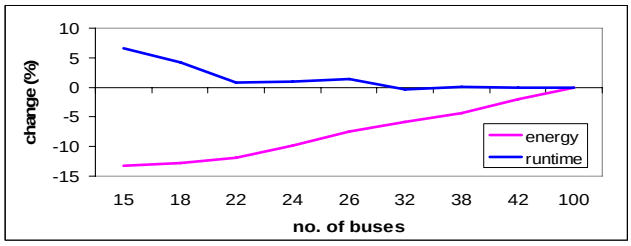

Figure 15. Total Energy vs. Runtime Trade-off graph

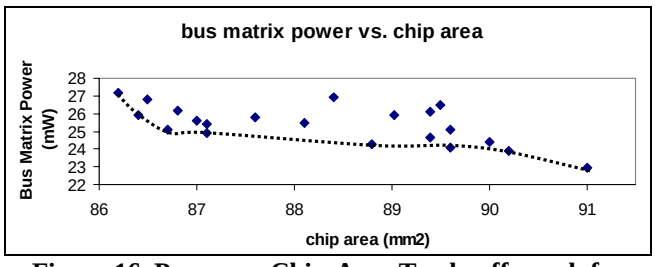

Figure 16. Power vs. Chip-Area Trade-off graph for MPSoC

It is also possible to perform other design space trade-offs within the CAPPS framework. For instance, Figure 16 shows the pareto-optimal trade-off 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.

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. There is also a possible trade-off of upto 10% for runtime and upto 15% for total energy consumed, which is a useful trade-off metric when the application is running on a fixed energy budget, which is the case for mobile battery-driven devices. The CAPPS 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. CONCLUSION

We presented the CAPPS 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. These energy macro-models were shown to have an average cycle energy estimation error of less than **5%** across the 180, 130 and 90nm technology libraries, compared with gate-level estimation. Plugging these macro-models in a system-level simulation framework allows a substantial speedup of almost **2000X** for cycle energy estimation, over gate-level estimation. These macro-models were used in the CAPPS synthesis framework for power estimation. Experimental results of applying CAPPS on an industrial networking MPSoC application generated results in a matter of a few 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. 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] AMBA AHB Interconnection Matrix, *www.synopsys.com/products/designware/amba_solutions.html*

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

[7] F. Angiolini et al., “Contrasting a NoC and a Traditional Interconnect Fabric with Layout Awareness”, *DATE 2006*
[8] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Constraint-Driven Bus Matrix Synthesis for MPSoC", *ASPDAC 2006*
[9] K. Lahiri et al., “Battery driven system design: A new frontier in low power design”, *ASPDAC 2002*
[10] M. Caldari et al. “System-level power analysis methodology applied to the AMBA AHB bus”, *DATE 2003*
[11] L. Shang et al., “Thermal Modeling, Characterization and Management of on-chip networks”, *MICRO 2004*
[12] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Fast Exploration of Bus-based On-chip Communication Architectures", *CODES+ISSS 2004*
[13] SystemC initiative, *www.systemc.org*
[14] Synopsys CoreTools, PrimePower *www.synopsys.com*
[15] M. Gasteier, M. Glesner, “Bus-based communication synthesis on system level”, *ACM TODAES, January 1999*
[16] S. Pasricha, N. Dutt, E. Bozorgzadeh, M. Ben-Romdhane, "Floorplan-aware Automated Synthesis of Bus-based Communication Architectures", *DAC 2005*
[17] M. Drinic et al. “Latency-guided on-chip bus network design”, *ICCAD 2000*
[18] A. Pinto, L. P. Carloni, A. L. Sangiovanni-Vincentelli, “Efficient Synthesis of Networks On Chip,” *ICCD 2003*
[19] N.D. Liveris, P. Banerjee, “Power aware interface synthesis for bus-based SoC designs”, *DATE 2004*
[20] U. Ogras, R. Marculescu, “Energy and Performance-Driven NoC Communication Architecture Synthesis using a Decomposition Approach”, *DATE 2005*
[21] J. Guo et al., “Energy/area/delay trade-offs in the physical design of on-chip segmented bus architecture”, *SLIP 2006*
[22] H-S. Wang et al., “Orion: a power-performance simulator for interconnection networks”, *MICRO 2002*
[23] A. Raghunathan et al., “High level Power Analysis and Optimization”, *Kluwer Academic Publishers, 1998*
[24] J. Chan et al., “NoCEE: energy macro-model extraction methodology for network on chip routers”, *ICCAD 2005*
[25] A. Bona et al., “System level power modeling and simulation of high-end industrial network-on-chip”, *DATE 2004*
[26] N. Dhanwada, et al., “A power estimation methodology for systemC transaction level models”, *CODES+ISSS 2005*
[27] U. Neffe et al. “Energy estimation based on hierarchical bus models for power-aware smart cards”, *DATE 2004*
[28] N. Eisley, L. Peh, “High level power analysis of on-chip networks”, *CASES 2004*
[29] T.T. Ye, L. Benini, G. De Micheli, “Analysis of power consumption on switch fabrics in network routers” *DAC 2002*
[30] GNU R, *http://www.gnu.org/software/r/R.html*
[31] J.J. Faraway, “Linear Models with R”, *CRC Press, 2004*
[32] C. Kretzschmar, et al., “Why transition coding for power minimization of on-chip buses does not work”, *DATE 2004*
[33] S. N. Adya, I. L. Markov, "Fixed-outline Floorplanning: Enabling Hierarchical Design", *IEEE TVLSI, Dec. 2003*
[34] J. Cong, D. Z. Pan, “Interconnect Performance Estimation Models for Design Planning”, *IEEE TCAD, June 2001*
[35] S. Pasricha, N. Dutt, E. Bozorgzadeh, M. Ben-Romdhane, "FABSYN: Floorplan-aware Bus Architecture Synthesis," *IEEE TVLSI March 2006*
[36] J. Cong, D. Z. Pan, “Interconnect Performance Estimation Models for Design Planning”, *IEEE TCAD, June 2001*
[37] J. Cong, Z. Pan, “Interconnect delay estimation models for synthesis and design planning”, *ASPDAC 1999*
[38] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov, A. Zelikovsky, “On Wirelength Estimations for Row-based Placement”, *In IEEE Trans. on ICCAD, vol.18, (no.9), IEEE, Sept. 1999*
[39] J. Cong, Z. Pan, L. He, C-K. Koh, K-Y. Khoo, “Interconnect design for deep submicron ICs”, *ICCAD 1997*
[40] Semiconductor Industry Association, “National Technology Roadmap for Semiconductors”, *SIA 1997*
[41] Berkeley Predictive Technology Model, U.C. Berkeley, *http://www-devices.eecs.berkeley.edu/~ptm/*
[42] Cadence PKS, *www.cadence.com/datasheets/pks_ds.pdf*
[43] S. Murali, G. De Micheli, “An Application-Specific Design Methodology for STbus Crossbar Generation”, *DATE 2005*