

# Fast Cache and Bus Power Estimation for Parameterized System-on-a-Chip Design

Tony D. Givargis, Frank Vahid  
Department of Computer Science and Engineering  
University of California, Riverside, CA 92521  
{givargis, vahid}@cs.ucr.edu

Jörg Henkel  
C&C Research Laboratories, NEC USA  
4 Independence Way, Princeton, NJ 08540  
henkel@cctl.nj.nec.com

## Abstract

We present a technique for fast estimation of the power consumed by the cache and bus sub-system of a parameterized system-on-a-chip design for a given application. The technique uses a two-step approach of first collecting intermediate data about an application using simulation, and then using equations to rapidly predict the performance and power consumption for each of thousands of possible configurations of system parameters, such as cache size and associativity and bus size and encoding. The estimations display good absolute as well as relative accuracy for various examples, and are obtained in dramatically less time than other techniques, making possible the future use of powerful search heuristics.

## Keywords

System-on-a-chip, low power, estimation, intellectual property, cache, on-chip bus.

## 1. Introduction

Silicon capacity continues to increase faster than the ability for designers to use that silicon, resulting in the well-known productivity gap [18]. Many researchers propose extensive reuse of pre-designed intellectual property *cores* to reduce this gap [8], where typical cores include microprocessors, microcontrollers, digital signal processors, encoders/decoders, bus interfaces, and numerous other common peripheral components. Two complementary core-based design approaches are emerging. One approach, based on a traditional *capture-and-simulate* [5] paradigm, assumes that a designer pieces together many cores obtained from various sources [24] (adding some custom logic), simulates extensively, and then generates new silicon implementing the system-on-a-chip. The other approach, which this paper addresses and which we refer to as *configure-and-execute*, assumes the designer starts with a pre-designed system-on-a-chip<sup>1</sup>, and then configures that system (including adding and deleting some cores) before generating new silicon [16][20][21][22]. The configure-and-execute approach has an advantage of enabling software development on real silicon, reducing the need for lengthy hardware/software co-simulations. Several commercial products now support such an approach for various application domains [14][23], such as networks and communications.

<sup>1</sup> Such pre-designed silicon has been referred to as a reference design, fig chip (configurable chip), and silicon platform by various authors.

A key to the success of a configure-and-execute approach is that the pre-designed system's architecture be heavily *parameterized*, so that design metrics like power, performance and size, can be optimized for a particular application's design constraints, by selecting particular parameter values before generating new silicon. We focus in this paper on parameters of the system cache and its associated on-chip buses, the CPU to cache bus, and the cache to main memory bus, as cache and bus have been shown to contribute to a significant percentage of system power. The main contribution of this paper is the creation of a fast cache power/performance estimation method and its coupling with a fast bus estimation method, enabling future heuristics that could simultaneously explore the large design space defined by cache and bus parameters. Such simultaneous exploration was recently shown to be crucial to optimizing deep-submicron designs [7], in which bus power consumption begins to surpass that of cache, and in which the cache and bus parameters must therefore be carefully tuned to one another.

Section 2 highlights the basic idea of parameterized system design. Section 3 describes related work in cache and bus power estimation and optimization. Section 4 describes our overall estimation approach. Section 5 describes the cache model used. Section 6 shows how to couple the cache model with our previously developed bus model. Section 7 describes our experimental results showing the speed and excellent accuracy of our approach. Section 8 provides conclusions.

## 2. Parameterized system design

Our long-term goal is to develop an environment supporting

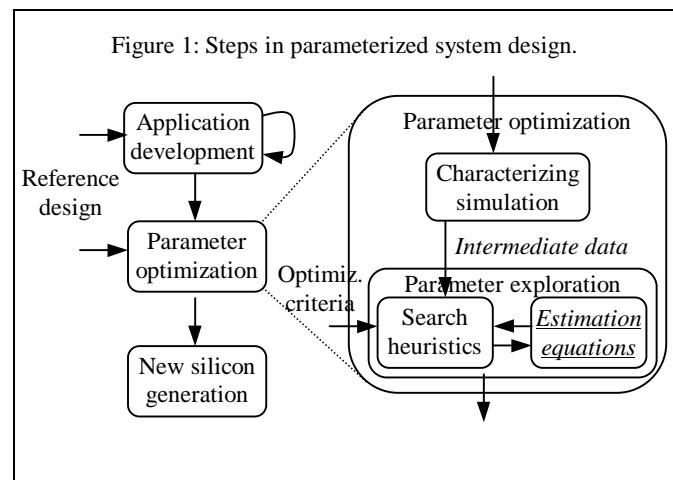
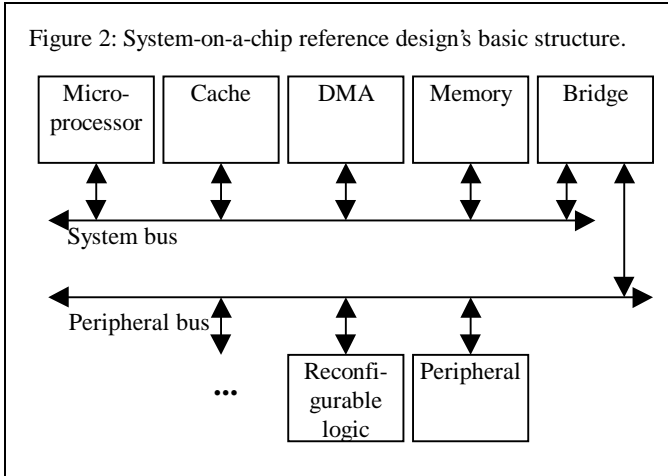


Figure 2: System-on-a-chip reference design's basic structure.



a *parameterized system design* approach. Such an approach consists of three main steps, as illustrated in Figure 1.

1. *Application development* begins with a commercially available "reference design," implemented on a configurable prototyping system-on-a-chip ("fig chip"). Figure 2 illustrates a typical reference design system-on-a-chip [24] consisting of a microprocessor core, cache, main memory, and direct-memory access (DMA) controller, all connected via a system bus. Also on that bus is a bridge to a set of peripheral cores, which differ depending on the class of intended applications (e.g., networking), and to reconfigurable logic or to add-on chips. The desired application is developed on this fig chip, which supports in-circuit emulation and hence at-speed application execution, overcoming the problem of prohibitively long simulation time for systems-on-a-chip. Some additional cores could be added (using the reconfigurable logic) and unneeded ones shut off. Numerous system-on-a-chip developers have begun to emphasize the importance of starting with such a reference design rather than composing cores from scratch [16][20][21][22].

2. *Parameter optimization* occurs once the application has been developed with the aid of the fig chip. The architecture's parameters are optimized for that application and its accompanying power, performance and size optimization criteria. Critical architectural parameters may include bus parameters like data size, address and data encoding techniques, multiplexing, etc., cache parameters like cache size, associativity, write-back techniques, block size/line size, etc., DMA parameters, and parameters relating to specific peripheral cores, like buffer sizes, resolutions, compression levels, etc. While the first step above has already had manifestations in commercial products (e.g., [23]), this second step still requires extensive research, and is beginning to receive some attention (such as work in [2], which optimizes a parameterized virtual memory system). Because the parameters are highly interdependent (for example, cache size impacts bus traffic), as well as strongly dependent on each application's features, this is a hard problem.

3. *New silicon generation* then results in a new chip implementing the optimized architecture, including any added

cores and excluding any shut-off ones. For example, while the reference design may have had a 32K cache and 32-bit unencoded bus, the optimized architecture for a particular application may have a 4K cache and an 8-bit bus using bus-invert encoding to reduce power. Ideally, the silicon is correct on the first-pass due to the extensive in-circuit emulation already performed.

Note the difference between a parameterized system design approach and the traditional system-level synthesis approach. In the traditional approach, an application is first described behaviorally, and then an architecture is synthesized for that application (processors, memories and buses are instantiated), and the behavior is then mapped to the architecture. In the parameterized system design approach, the basic architecture is pre-defined, and the application is developed on that architecture. The "system synthesis" going on is really a fine-tuning of the original architecture through selection of values for the architecture's parameters.

### 3. Related work

Numerous techniques for high-level power estimation and optimization have evolved recently; an overview can be found in Raghunathan et al [17].

Much attention has been given to developing detailed models of cache internals to accurately predict a cache's latency [25] as well as power consumption [3] for a given parameter configuration. Such detailed models would be used to estimate power in cache simulators, which we use as described later.

Attention has also been given to exploring various cache configurations in terms of power, performance and size. Su and Despain [19] evaluate several cache design techniques with respect to power and performance. Henkel [11] used exhaustive trace-driven cache simulations to show that the best cache configuration, in terms of power, performance and size, differed greatly for different applications.

Noticing that large tradeoffs are possible by configuring a cache for an application, but that cache simulations are slow, many researchers have focused on speeding up cache simulations. Kirovski et al [9] reduces the *number* of trace-driven cache simulations necessary for exploring different cache configurations, by establishing bounds and hence pruning numerous inferior configurations without having to simulate them. Wu and Wolf [26] order the search of different cache configurations such that, after each cache simulation, they can reduce the *size* of a given input trace by removing redundant information ("trace stripping"), thus speeding up subsequent simulations of other configurations. Our work differs in that we couple cache parameters with bus parameters (and possibly other parameters in the future), resulting in an enormous design space and thus seemingly excluding any approach based on repeated simulation. While one-pass cache-simulation [12] is a common technique, in which numerous cache configurations are evaluated simultaneously during one simulation, incorporating the myriad of other parameters that we wish to consider (bus, DMA, peripheral cores, etc.) into such an approach would likely become prohibitively complex.

With the advent of deep-submicron technology and the accompanying increase in the bus' contribution to system power consumption, recently researchers have begun to focus on reducing bus power [17], and more closely related to our work, on the inter-relationship of cache and bus power consumption. Compiler-level approaches, like that by Panda and Dutt [15], seek to generate executable code that minimizes power on a given cache/bus architecture. An architectural approach was presented by Fornaciari et al [4], who investigated the power consumption of different bus encodings for various cache configurations. Li and Givargis et al [6] developed a fast bus power model and then coupled [7] this model with Li and Henkel's cache simulations to show that, in deep submicron technologies, the best cache and best bus configurations are tightly interdependent and thus should be sought simultaneously. Our work is an improvement to this work in that the long cache simulations can be replaced by the fast models in this paper to reduce the time to evaluate all cache/bus configurations for a given application from days/weeks to seconds/minutes.

#### 4. Approach overview

Because a parameterized system design approach will have numerous interdependent parameters, an approach requiring simulation for each configuration would be computationally infeasible due to the exponential number of configurations. Thus, we instead use a two-step approach to parameter optimization, as shown in Figure 1. *Characterizing simulation* involves simulating the application with typical input vectors once or a small number of times that is just enough to provide enough intermediate data to characterize the application for the second step. The second step, *parameter exploration*, uses heuristics to traverse the design space of possible parameter configurations, coupled with fast estimation equations that use the intermediate data to provide power, performance and size values for a given configuration. These equations evaluate in constant-time, so can deal with huge numbers of possible configurations.

We have chosen to focus initially on developing parameter optimization for a system's cache and bus sub-systems, because these typically consume a significant percentage of system power (we plan to soon extend our approach to also consider DMA). We have already developed an approach for buses [6] involving definition of the intermediate data (bus traffic), estimation equations for power, performance and size as a function of bus parameters (size and encoding) and traffic, and an exhaustive search heuristic. We now describe the intermediate data and estimation equations necessary for cache parameter optimization, followed by a description of a method for coupling the cache methods with that previously developed for buses. We showed earlier [7] that the tight interdependency of cache and bus parameters requires such a coupling in order to find the best cache and bus configurations optimizing power, performance and size.

#### 5. Cache performance and power estimation

In this section, we discuss the technique that we have employed for rapidly estimating cache metrics. We define the problem as follows. Given a trace of memory references

(referred to hereafter as a *trace-file*), we are to compute the number of cache misses<sup>2</sup>, denoted  $N$ , for all different caches. Two caches are different if they differ in their total cache size, line size (block size) or degree of associativity. We limit each of these three distinguishing parameters to a finite range:

$$\begin{aligned} S &= \{2^i : i = S_{Min} \cdots S_{Max}\} \\ L &= \{2^i : i = L_{Min} \cdots L_{Max}\} \\ A &= \{2^i : i = A_{Min} \cdots A_{Max}\} \end{aligned}$$

Note that, for practical purposes, we only consider values that are powers of two for each of these parameters. Given a trace-file, we must define a function:

$$f : (S \times L \times A) \rightarrow N.$$

to compute the number of cache misses for any cache configuration. We assume that, with the aid of a cache-simulator, we are able to compute the above function, for any value from the sets  $S$ ,  $L$  and  $A$ , in linear time with respect to the size of the trace-file. Intuitively, our approach works as follows. We know that at low cache sizes, higher line size and associativity have a greater positive effect than they do at high cache sizes. For example, doubling the line size when cache size is 512B may reduce cache miss rate by 30%, however, when the cache size is 8K, it may not reduce the miss rate at all. Thus, we are interested in finding these improvement ratios at both low and high cache sizes, so that, by line fitting, the improvement ratio for any cache size can be estimated. This assumes a smooth design space between these points. We next describe our approach for estimating this function for all range values.

Our approach consists of three steps. First we simulate the trace-file for some selected  $S$ ,  $L$  and  $A$  values and obtain the corresponding cache misses. Then we calculate a linear equation, using the least square approximation method. Last we use our linear equations to compute  $N$  for all cache parameters. We first simulate the following points in our domain space:

$$\begin{aligned} f(S_{Min}, L_{Min}, A_{Min}) &= N_1 \\ f(S_{Max}, L_{Min}, A_{Min}) &= N_2 \\ f(S_{Min}, L_{Max}, A_{Min}) &= N_3 \\ f(S_{Min}, L_{Min}, A_{Max}) &= N_4 \\ f(S_{Max}, L_{Max}, A_{Min}) &= N_5 \\ f(S_{Max}, L_{Min}, A_{Max}) &= N_6. \end{aligned}$$

Then we compute the following ratios:

$$\begin{aligned} R_1 &= N_1 / N_3, \quad R_2 = N_1 / N_4 \\ R_3 &= N_2 / N_5, \quad R_4 = N_2 / N_6. \end{aligned}$$

<sup>2</sup> Other metrics, e.g., number of write backs, can be estimated, using our approach, in a similar manner.

Here,  $R_1/ R_2$  denotes the improvement we obtain by using maximum line-size/associativity when cache size is at its minimum. Likewise  $R_3/ R_4$  denote the positive improvement we obtain by using maximum line-size/associativity when the cache size is at its maximum. Given these ratios we estimate  $N$  for a given cache size, line size and associativity as follows:

$$\begin{aligned}
s &= (S_i - S_{Min}) / S_{Max} \\
l &= (L_j - L_{Min}) / L_{Max} \\
a &= (A_k - A_{Min}) / A_{Max} \\
t_1 &= s(N_2 - N_1) + N_1 \\
t_2 &= l(R_3 - R_1) + R_1 \\
t_3 &= a(R_4 - R_2) + R_2 \\
f(S_i, L_j, A_k) &\approx t_1(1 - t_2 - t_3).
\end{aligned}$$

The first three equations,  $s$ ,  $l$  and  $a$ , normalize our parameters to be within a unit range. The next equation,  $t_1$ , estimates cache misses using lowest line size and associativity, by computing a linear line through the points  $N_1$  and  $N_2$ . If more simulation data is available, the least square approximation is used to compute  $t_1$ . The next two equations,  $t_2$  and  $t_3$ , estimate the expected improvement gained from higher line size or associativity. The last equation combines the previous equations to estimate cache miss rate.

## 6. Combined cache/bus estimation

In this section, we describe how to extend the cache data into bus data for simultaneous cache/bus design space exploration. The technique described in the previous section allows us to rapidly estimate the number of cache misses,  $N$ , for a given cache parameter setting. This number,  $N$ , is a measure of cache to main-memory bus traffic. Likewise, the total number of cache accesses, i.e., the size of the trace-file, is a measure of CPU to cache bus traffic. Given this traffic, and assuming data of random nature, we can use equations [6] to compute the bit switching activity on the bus and use it, along with wire capacitance models, to compute power consumption of our system. In this work, we consider varying the number of data bus wires, e.g., 16 or 32-bits, and data encoding, e.g., binary or bus-invert.

For our bus model, we assume that there are  $m$ ,  $n$ -bit items transmitted per unit time on a bus of width  $k$  using binary encoding. (Here  $m$  denotes the traffic on the bus and is obtained by estimating cache misses as described above.) The following equation gives power consumption for the data bus:

$$\begin{aligned}
P_{bus} &= (C_{bus}) \left( \left[ \frac{n}{k} \right] \text{transfer/item} \right) (k \text{ bit/transfer}) \\
&\quad (m \text{ item/second}) \left( \frac{1}{2} \text{ transition/bit} \right) \\
&= \frac{1}{2} (C_{bus}) \left[ \frac{n}{k} \right] (k)(m) \text{ transition/sec}
\end{aligned}$$

In this equation, bus capacitance is calculated using models developed by Chern et al [1]. Our equation is expanded to take into account bus-invert encoding. This method uses an extra

control line and extra circuit logic to compute the Hamming distance (bit transitions) between two consecutive data items. If the Hamming distance is greater than  $\frac{1}{2}$  the bus width, then the control line is asserted and the inverted data is send over the bus [12]:

$$PI_{bus} = (C_{bus}) \left( \left[ \frac{n}{k} \right] \left( \frac{1}{2^k} + \frac{2}{2^k} + \dots + \frac{\left[ \frac{k}{2} \right]}{2^k} \right) \left[ \frac{k}{2} \right] \right)$$

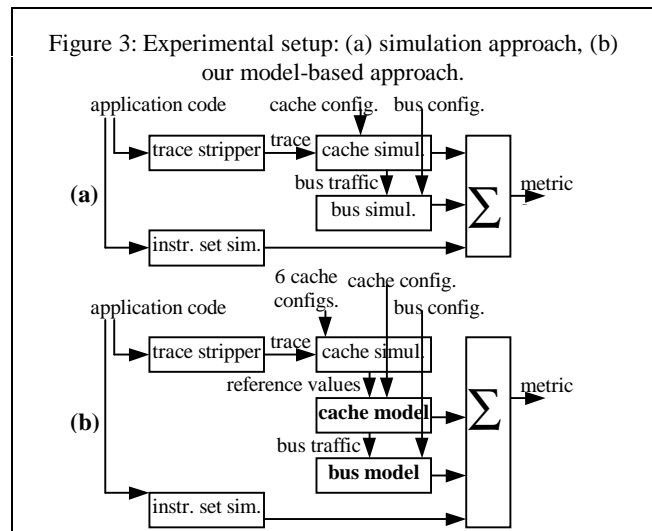
(m) transitionsec

Given the traffic  $m$  on a bus, power can be quickly estimated using analytical models as described above. Likewise, similar analytical models can be applied to compute cache and memory power, (and performance). These have been extensively modeled by [3].

## 7. Experiments

In order to verify our approach, we performed the following experiments. We used two applications written in C, a diesel engine controller (Diesel), and an encryption algorithm (Key). We explored power and performance for mapping each application to a system architecture including three parameterized parts: cache, CPU-cache bus, and cache-memory bus. The cache parameters and their possible values were: cache size of 128, 256, 512, 1K, 2K, 4K, 8K or 32K; cache line of 8, 16 or 32; and associativity of 2, 4 or 8. The parameters for each bus were: data width of 4, 8, 16 and 32; and bus invert encoding either enabled or disabled.

We compared this paper's fast cache/bus estimation technique with the simulation approach of [7]. For the simulation approach, illustrated in Figure 3(a), we ran the C application through a trace stripper to generate a trace of memory references. Then, for each cache configuration, we



ran the trace through a cache simulator to obtain cache power, cache performance, and bus traffic, and this traffic was then fed into a bus simulator [6] for each bus configuration, which provided bus power and performance. We also ran the C application through an instruction-set simulator to obtain CPU power and performance values. The metrics were then combined to provide final metric values for the CPU, cache and bus sub-system. Obtaining these values for all possible cache/bus configurations required 4.8 hours of CPU time to carry out the simulations.

For the fast estimation approach, illustrated in Figure 3(b), we again ran the trace stripper. But then, we ran the cache simulator only 6 times for the reference cache configurations described earlier. We then fed the power, performance and hit rate information from these simulations into our cache power/performance models, and then evaluated the models for all cache configurations. For each such configuration, we also obtained bus traffic data, and fed this data into our bus model for all bus configurations. Obtaining these values for all possible cache/bus configurations required only 2.3 minutes of CPU time, a speedup of 125 times.

While we obtained data for all of the 45,568 possible cache/bus configurations, we present data for just a small subset of 10 configurations in Table 1. These configurations have been selected to reflect worst, average and best case estimates. Figure 4 provide performance and power respectively for Diesel and Key applications. The light-gray bars are actual measurements, and the dark-gray bars estimated measurements.

Config.	Bus1	Bus2	Isize	DSize	Line	Assoc.
0	8/0	4/1	128	512	8	2
1	32/1	32/1	512	16K	8	2
2	32/0	16/0	8K	2K	8	2
3	4/0	8/1	32K	16K	8	2
4	16/0	32/1	512	4K	16	2
5	8/0	8/1	8K	512	16	2
6	8/1	16/0	4K	16K	32	2
7	8/1	4/0	1K	8K	8	4
8	16/0	16/0	1K	256	16	4
9	32/1	8/1	1K	1K	32	4

Table 1: Design space configuration parameters

While hundreds of times faster, our cache estimation approach resulted in an average error of only 2%, with the worst case being 18%, over the entire solution space of thousands of cache/bus configurations (and not just the 10 configurations presented here). It should be noted that the CPU power consumption was about 44% and 65% (for Diesel and Key respectively) of total power consumption.

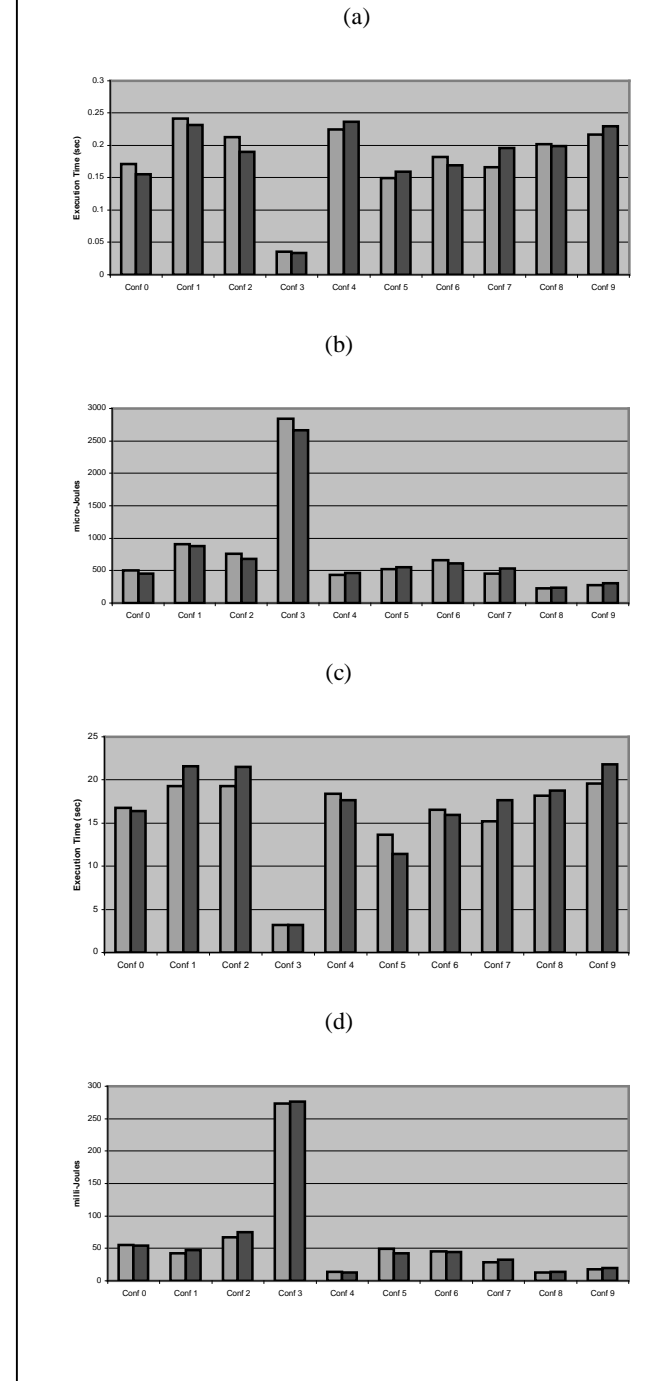
Perhaps even more important than the accuracy reported above is the *relative accuracy*, or fidelity, of the estimates. We see from the charts that our fast approach orders the various configurations the same as the simulation approach – thus, we have the ability to still pick the best parameter configuration, which is the most important aspect of the approach.

## 8. Conclusions

We have presented a technique for rapidly estimating the power and performance of the cache memory and bus sub-system of a system-on-a-chip. The technique is orders of

magnitude faster than simulation-based approaches, but yields good accuracy. The technique therefore enables the extensive exploration of the many configurations possible in a parameterized system-on-a-chip architecture.

Figure 4: Experimental results: (a) Diesel application’s performance, (b) Diesel application’s energy, (c) Key application’s performance, (d) Key application’s energy. Light gray is actual measurement, and dark-gray denotes estimated measurements.



## 9. Acknowledgement

This work was supported by the National Science Foundation (CCR-9811164) and a Design Automation Conference Graduate Scholarship.

## 10. References

- [1] Chern et al., Multilevel metal capacitance models for CAD design synthesis systems, *IEEE Electron Device Letters*, vol. 13, Jan. 1992.
- [2] J.L. da Silva Jr, F. Catthoor, D. Verkest and H. De Man. Trading-off Power versus Area through a Parameterizable Model for Virtual Memory Management, *IEEE VOLTA*, 1999.
- [3] R. J. Evans, P.D. Franzon. Energy Consumption Modeling and Optimization for SRAMs, *IEEE Journal of Solid-State Circuits*, Vol. 30, No. 5, pp. 571-579, 1995.
- [4] W. Fornaciari, D. Sciuto, C. Silvano. Power Estimation for Architectural Exploration of HW/SW Communication on System-Level Buses, *International Workshop on Hardware/Software Codesign*, pp. 152-156, 1999.
- [5] D. Gajski, N. Dutt, C. Wu, S. Lin. *High-Level Synthesis*, Kluwer Academic Publishers, 1992.
- [6] T. Givargis and F. Vahid. Interface Exploration for Reduced Power in Core-Based Systems, *International Symposium on System Synthesis*, 1998, pp. 117-122.
- [7] T. Givargis, J. Henkel, and F. Vahid. Interface and Cache Power Exploration for Core-Based Embedded System Design. *ICCAD 1999*.
- [8] R. Gupta and Y. Zorian. Introducing Core-Based System Design, *IEEE Design & Test*, Vol. 14, No. 4, Oct-Dec 1997, pp. 15-25.
- [9] D. Kirovski, C. Lee, M. Potkonjak, and W. Mangione-Smith. Synthesis of Power Efficient Systems-on-Silicon, *ASP-DAC 1998*, pp. 557-562.
- [10] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, A. Sangiovanni-Vincentelli. Efficient Power Estimation Techniques for HW/SW Systems, *IEEE VOLTA*, 1999.
- [11] Y. Li and J. Henkel. A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems, *Design Automation Conference*, pp.188-193, 1998.
- [12] Mircea R. Stan and Wayne P. Burleson, Bus-Invert Coding for Low Power I/O, *IEEE Transactions on VLSI*, March 1995.
- [13] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation Techniques for Storage Hierarchies, *IBM Systems Journal*, 9(2), pp. 78-117, 1970.
- [14] S. Ortiz. New Chips Move Networking onto Silicon. *IEEE Computer*, Feb 1999.
- [15] P. Panda and N. Dutt, Reducing address bus transition for low power memory mapping, *European Design and Test Conference*, pp. 63-68, 1996.
- [16] B. Payne. Rapid Silicon Prototyping: Paradigm for Custom System-on-a-Chip Design, <http://www.vlsi.com/velocity>, 1998.
- [17] A. Raghunathan, N.K. Jha and S. Dey. *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, Norwell, MA, 1998.
- [18] Semiconductor Industry Association Roadmap 1997, <http://notes.sematech.org/ntrs/PublNTRS.nsf>.
- [19] C. Su, A. Despain. Cache design trade-offs for power and performance optimization: a case study. *International Symposium on Low Power Design*, pp. 63-68, 1995.
- [20] F. Vahid, T. Givargis, The Case for a Configure-and-Execute Paradigm. *International Workshop on Hardware/Software Codesign*, 1999.
- [21] P. van der Wolf, P. Lieveise, M. Goel, D.L. Hei, K. Vissers. An MPEG-2 Decoder Case Study as a Driver for a System Level Design Methodology, pp. 33-37, *International Workshop on Hardware/Software Codesign*, 1999.
- [22] J. van Meerbergen, A. Timmer, J. Leijten, F. Harmsze, M. Strik. Experiences with System Level Design for Consumer ICs, *VLSI'98*, pp 17-22.
- [23] Velocity product information, VLSI Technology Inc., <http://www.vlsi.com/velocity>.
- [24] Virtual Socket Interface Association, Architecture Document, <http://www.vsi.org>, 1997.
- [25] S.J.E. Wilton and N.P. Jouppi. CACTI: An Enhanced Cache Access and Cycle Time Model, *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 5, pp. 677-688, 1996.
- [26] Z. Wu and W. Wolf. Iterative Cache Simulation of Embedded CPUs with Trace Stripping. *International Workshop on Hardware/Software Codesign*, pp. 95-99, 1999.