Interface and Cache Power Exploration for Core–Based Embedded System Design

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

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

Frank Vahid
Department of Computer Science and Engineering
University of California, Riverside, CA 92521
vahid@cs.ucr.edu

Abstract

Minimizing power consumption is of paramount importance during the design of embedded (mobile computing) systems that come as systems-on-a-chip, since interdependencies of design characteristics like power, performance, and area for various system parts (cores) become increasingly influential. In this scenario, interfaces play a key role since they allow one to control/exploit these interdependencies with the aim to meet design constraints like power. In this paper, we present the first comprehensive approach to explore this impact. We consider a whole system comprising a CPU, caches, a main memory and interfaces between those cores and demonstrate the high impact that an adequate adaptation between core parameters and interface parameters in terms of power consumption has. We especially found that cache parameters and bus configurations of cache buses have a significant impact in this respect. In addition, we made the important observation that optimizing for performance no longer implies that power is optimized as well in deep submicron technologies. Instead, we found out that especially for newer technologies, the relative interface power contribution increases, leading to scenarios where we obtain a real power/performance tradeoff. In summary, our explorations unveiled not yet investigated interdependencies that represent the first step towards future efforts to optimize/adapt interfaces and caches in core–based systems for low power designs.

1 Introduction

The power consumption of electronic devices is becoming an increasingly essential concern when designing embedded systems, especially mobile computing devices. This is because those devices draw their current from batteries that place a limited amount of energy at the system's disposal. Consequently, the lower the average power consumption of those devices, the longer they can operate between two re–charge phases. Hence, their mobility is higher and this is a strong argument for preferring such a device to competitive devices.

From a design point of view, those consumer devices often come in the form of a *SOC* (System–On–a–Chip) in order to keep production and development expenses as low as possible while complying with various design constraints including power consumption, performance, etc. Though currently-deployed semiconductor technologies [1] allow one to integrate more than 100 million transistors onto one single chip, the so-called *design gap* currently hardly allows designs that exceed 10 million transistors to be integrated onto a chip (excluding memory). A solution that helps overcome this design gap (see also [2]) is a design process called *core–based system design.* Rather than developing every sub–system from scratch, the designer now composes a system by integrating various purchased *cores* (or re–uses previously deployed ones). Depending on the form in which the core comes (soft core or hard core, see also [3]) the designer has the leeway to adapt those cores to specific constraints or can use already optimized and synthesized (hard) cores in case they already comply with those constraints.

Still, interfacing those various cores, which may even come from different vendors, remains a key problem. Though organizations like VSIA [4] are working on defining uniform interfaces, eventually the designer has to choose the kind and the configurations of the interfaces that connect their cores best in terms of their design constraints.

The impact of interfaces in core-based designs, especially with respect to low power consumption, has hardly been investigated. Our preinvestigations have shown that a single application under different combinations of cache design parameters (cache size, block size, associativity etc.) and interfaces design parameters (like bus width, bus encoding etc.) may result in solutions that feature as little as only 3% of relative bus power consumption up to as much as 41% relative bus power consumption[1].

Obviously, there is a strong interdependency between the optimum core parameters and the according optimum bus parameters. This adaptation has to be performed *concurrently* in order to achieve optimum results. In addition, it is important to concurrently take into consideration as much as possible relevant design constraints like *power*, *performance* and *hardware effort*.

In this paper, we present the first comprehensive exploration of interface power dissipation in core-based designs, since we explore the interdependencies between various core parameters *and* diverse interface (i.e., bus) configurations. In addition, we take into consideration not only power dissipation but its interdependencies with performance and hardware effort also. As we will discuss in Section 2, other approaches are less comprehensive since they focus only on a limited amount of design constraints or they do not focus on a complete SOC (for example, just investigating bus and cache but not including main memory or CPU).

The rest of this paper is structured as follows. After introducing related work in the subsequent Section 2, Section 3 explains the target architecture on which we focus. The experimental setup to conduct our exploration is summarized in Section 4. Section 5 describes in detail our explorations of the design space through our experiments. Section 6 provides the main conclusions and highlights future work.

2 Related work

The related work important to our approach can be divided into three categories: work on system-level power optimization in general, architectural power optimization focusing on a single core (like a CPU, cache, main memory etc.) and work on bus issues like power, performance and size.

As for the first group, Dave et al. [5] introduce a co-design methodology that optimizes for power and performance at the task-level. Their procedure for task allocation is based on an *average* power consumption and does not take into consideration data dependencies on the instruction level to estimate/optimize power. In addition, they do not take into consideration cache and bus effects in terms of power and performance. The system–level power optimization approach proposed by Hong et al. [6] has the same limitations regarding the addressing of architectural tradeoffs. Rather, they exploit the technique of variable voltage scaling in order to minimize power consumption.

At the architectural-level for single system components (i.e., not considering any trade-offs between various system parts), high performance microprocessors have been investigated and specific software synthesis algorithms have been derived to minimize power by Hsieh et al. [7]. Tiwari [8] investigated the power consumption at the instruction-level for different CPU and DSP architectures and derived specific power optimizing compilation strategies.

An approach that is system-level based and takes into consideration the interdependencies between various system parts has been proposed by Li et al. [9]. Their target system features a CPU, a data cache, an instruction

[1] The *relative bus power consumption* is defined as the relationship between bus power consumption and the power consumption of all other involved cores like CPU, caches, main memory, etc.

0-7803-5832-X /99/$10.00 ©1999 IEEE.

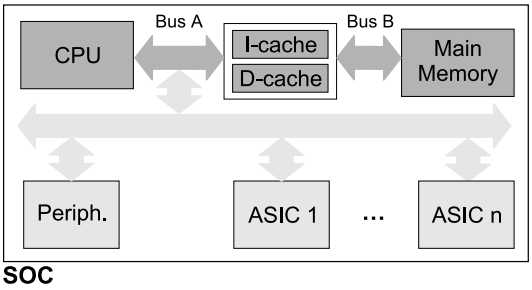

Figure 1: Target architecture.

cache and a main memory. The impact of buses is not accounted for. Fornaciari et al. [10] explore the impact of different bus encoding schemes for embedded systems in terms of power dissipation. Though they study power consumption for various cache sizes they do not explore all relevant interdependencies (e.g., cache line size, associativity, main memory size, etc.). So, their approach basically assumes that most system parameters have already been determined and they focus on finding the best bus encoding scheme.

A key contribution in reducing bus power has been the encoding of data, such as bus-invert [11]. Here, the Hamming distance of two consecutive data words is computed. If this distance is greater than half the word size, the inverted data is sent and a control signal is asserted to signal decoding of the data on the receiver side. Using bus-invert, a theoretical power savings of 25% average and 50% peak is obtainable. Limited-weight codes [12] are generalized encoding schemes that, using more than one control signal, reduce average bit transitions on the bus, resulting in even lower power consumption.

Givargis and Vahid [13], have developed a set of mathematical formulas for rapidly estimating bit switching activities on a bus with a given size and encoding scheme. They have also contributed formulas to estimate bit switching activities used by the encoding/decoding logic. These formulas, combined with the capacitance estimation formulas by Chern et al. [14], can be used to perform a system level exploration of bus size and encoding schemes for low power designs.

Our approach is more comprehensive than approaches proposed so far since we: (a) take into consideration most of the relevant parts of a whole embedded system (CPU, instruction/data cache, main memory and buses), and (b) we explore the interdependencies of different system parameters (like cache sizes, cache associativities etc., main memory size, bus encoding schemes, bus widths), and (c) we consider, besides power, performance and area as well.

3 Target Architecture

Figure 1 shows the block diagram of our SOC target architecture. The whole system may feature a CPU, an instruction cache (I-cache), a data cache (D-cache), a main memory, peripheral units, and cores for diverse applications (like MPEG encoding, for example), as well as various buses to connect all these cores. Our investigations focus on the sub-system CPU, CPU-to-cache bus, I/D-caches, cache-to-main-memory bus and the main memory (highlighted in dark grey). Changes of system parameters within this sub-system primarily have an impact on cores within this sub-system and merely influence the behavior (in terms of power, performance, hardware effort) of cores outside of this group.

The I-cache and D-cache each have the following design parameters: cache size, line size, associativity, tag size and index size. The range of values considered are: cache sizes of (32K, 16K, 8K, 4K, 2K, 1K, 512, 256, or 128), line sizes of (8, 16, or 32), and associativity of (2, 4, or 8). Tag and index sizes follow from earlier parameters. Each bus has the following design parameters: number of data lines (32, 16, 8 or 4) with bus invert (on or off).

We assume that the surface area used for routing bus wires is fixed between CPU and cache and between cache and memory. Given some amount of routing area, the spacing between wires will be greater for narrower buses, i.e., 4-bit bus wires will be spaced greater than 32-bit bus wires. Greater spacing of bus wires will result in lower bus capacitance due to reduced coupling capacitance, as described in [14].

In our experiments we focus on evaluating bus and cache parameters together in order to find the best tradeoff between power/performance/hardware effort.

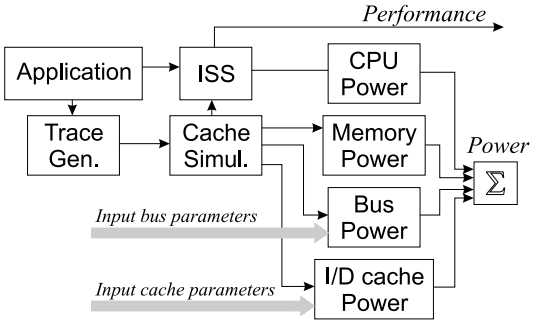

Figure 2: Experimental setup.

4 Experimental setup

For our experiments, we used the setup shown in Figure 2, which allows us to estimate power consumption of the sub–system shown in Figure 1, as well as to estimate its performance for a given set of bus and cache parameters. First, the source code of the application is fed into a trace simulation tool that provides traces to a cache simulator[2]. The cache simulator outputs the number of total cache accesses, the number of cache misses and the number of cache hits. This data is used to calculate the total number of main memory accesses as well as the number of bus transfers via buses A and B, according to Figure 1, such that by means of analytical power models for buses, the caches and the main memory, the specific power consumption is calculated. As for the CPU, a simulation approach is deployed that uses an ISS (instruction set simulator) with an attached, instruction–level based, power model. The performance is obtained as an output of the ISS whereas the power consumption of the whole system results as the sum of the power consumptions of all involved cores.[3] The design space exploration can be accomplished by running an application through this setup with varying cache parameters (cache sizes, associativities, line sizes) and bus parameters (bus width, bus encoding schemes, etc.) Different cache and bus parameters result in a different hardware effort. Distinguishing design points that would otherwise result in similar design points (similar power consumption and similar performance), based on the hardware effort at which those results have been obtained, represents an important criteria for the designer. The area required to implement a given bus/cache configuration was computed as the sum of the area required by the following logic: bus-invert coding/decoding (24 gates * bus-size), data multiplexing/de-multiplexing (2200 + 32 * number of sub-items), cache cell area (.75 * cache size), and cache associativity (96 * (associativity-1)).

For our experiments we deployed four, mostly data–dominated, applications: an algorithm for computing 3D vectors of a motion picture (”*3d-image*”), an MPEGII encoder (”*mpeg*”), a complex chroma–key algorithm (”*ckey*”), and a diesel engine control algorithm (”*diesel*”). (We in fact did a fifth example of a trick animation algorithm, but omit results for presentation briefness since they matched the others). The applications ranged in size from about 5kB to 230kB of C code.

5 Design-space explorations

5.1 Overview

For each of the four applications, we ran the cache simulation tool for all 712 different cache configurations, and obtained the cache, main-memory, and CPU power consumption values as well as the CPU-to-cache and cache-to-main-memory bus traffic traces. We then input the bus traffic traces to a bus power estimation tool [13] previously shown to be accurate to within 1% of a trace driven estimator, for all 64 possible bus configurations. We thus obtained total power (cache plus main-memory plus CPU plus bus) consumption for $712 * 64 = 45568$ possible configurations for each example. Generating this power data for all four examples required roughly one week of computation time. We did this procedure for two distinct technologies, an older technology (0.8μ), shown in Figure 3, and a newer technology (0.18μ), shown in Figure 4. Plots represent the *3d-image*, *mpeg*, *ckey* and *diesel* examples, starting from upper-left and going clockwise. The wire-to-gate capacitance ratios for old and new technologies were 3 and 100, respectively, based on [15], also supported by data in [16] showing rapidly increasing ratios.

[2] Trace generator and cache simulator are third party tools.

[3] For more details on our analytical power models, please refer to [13].

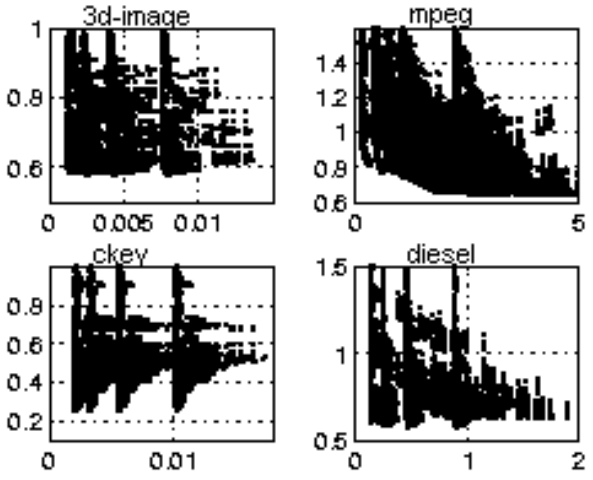

Figure 3: Power vs execution-time plots for older technology – X-axis is execution-time in seconds, Y-axis is total power in watts.

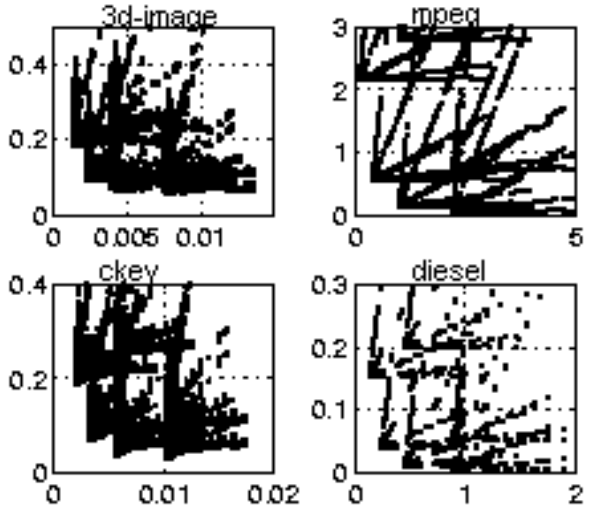

Figure 4: Power vs execution-time plots for newer technology – X-axis is execution-time in seconds, Y-axis is total power in watts.

Each plot represents several ten thousands (see numbers above) of configurations, each configuration represents a point in the plot[4]. In addition, many inferior data points have been cropped to further improve the presentation.

5.2 Power and performance tradeoffs in newer technologies

Figure 3 provides plots of execution-time (X-axis) and total power (Y-axis) for each example in the old technology. Total power includes CPU, cache, memory and buses. Note that in 3 of the 4 examples, the configuration with the lowest execution-time corresponded with the configuration having the lowest power consumption. Only in 1 example was there a real tradeoff between execution-time and power.

Figure 4 provides plots in the newer technology. Upon examination, we find that *all four examples exhibit a tradeoff between execution-time and power*. The reason for this behavior is that the bus power is a significant component of total power in newer technologies, and bus power is inversely related to execution-time. In particular, fewer wires implies less wire capacitance and hence less power, but also implies more bus transfers and hence a higher execution-time.

[4] Please note that due to the scale/resolution, many different design points appear as only one point in the plots.

Each plot displays at least four distinct "regions." In the older technology, each region appears as a spike. In the newer technology, each region appears in an L–shape. Upon investigation of the data, we found that each region corresponds to a particular CPU-to-cache (see Figure 1) bus size of either 32, 16, 8 or 4 (i.e., the four sizes we evaluated). The leftmost region corresponds to a CPU-to-cache bus of 32 bits (giving the smallest execution time), the rightmost corresponds to 4 bits. The bottom-most point of each region corresponds to the optimal cache/bus configuration for that region based only on power and execution-time (not size). The many points above and often to the right of the optimal correspond to different cache and cache-to-main-memory bus configurations. Within a region, there is no tradeoff between power and execution-time – the lowest execution time configuration is also the lowest power configuration in any given region. Thus, we observe that *the CPU-to-cache bus size is the major component in determining the system's power and execution-time metrics, and there is a tradeoff among those two metrics.* We conclude from this data that future automated search heuristics should begin by selecting a CPU-to-cache bus size that first puts one in the appropriate region based on power and execution-time constraints and/or cost function for a given application, and second seeks the near-optimal power and execution-time points in that region.

As an additional note, some plots show more than four regions, with a region X appearing directly above another region Y (actually all plots originally showed four pairs of regions before being cropped). These X and Y regions differ in that Y uses bus-invert for the CPU-to-cache bus while X does not. We see that bus-invert is crucial for low-power in newer technologies.

5.3 Size and differing cache/bus configurations

At the bottom of each region in the new technology, we see a very dense set of points. These points correspond to the optimal or near-optimal power and performance configurations for that region, each point representing a different cache configuration and cache-to-main-memory bus configuration. We obviously want to choose the configuration with the smallest size that is near the optimal within some tolerance.

We therefore examined all configurations within a couple percent of the optimal power and execution-time of each region to find the minimum size configuration in this sub-region. One might expect that the best cache configuration in one region would be the best in the other regions, for a single example. In some examples, this was indeed the case. But in other examples, the best configuration was *different* in different regions. Table 1 illustrates this difference for two examples. The column headings are as follows: execution time (sec), power (watts), silicon area (gates), CPU-to-cache bus width (wires), CPU-to-cache bus encoding (0=binary, 1=bus-invert), cache-to-memory bus width (wires), cache-to-memory bus encoding (0=binary, 1=bus-invert), instruction cache size (bytes), instruction cache associativity, instruction cache tag (bits), data cache size (bytes), data cache associativity and data cache tag (bits). The first three rows correspond to the best configuration in the 32, 16 and 8-bit CPU-to-cache regions for the *mpeg* example and the second three rows for the *diesel* example. Notice that the best configurations for the first example involve variations in cache sizes, associativity, and tag sizes, and in one case bus invert is not used on the cache-to-memory bus. In the second example, associativity and tag sizes vary, as does the cache-to-memory bus size and the use of bus invert.

Slight changes in these parameters result in significant penalties. For example, changing the tag in the fifth row from 25 to 24 (to match that of the last row) results in a performance penalty of 28%!

The key conclusion from the above discussion is that there is no one best cache configuration independent of bus configuration, and vice-versa.

Ex	Pwr	Size	Cb	Cb-i	Mb	Mb-i	I	I-a	I-t	D	D-a	D-t
.086	43.6	199888	32	1	32	0	16k	4	20	16k	4	20
.389	11.4	204568	16	1	32	1	16k	8	21	32k	8	21
.995	3.4	303672	8	1	32	1	32k	8	20	16k	8	21
.002	.19	15496	32	1	32	1	1k	4	24	512	4	25
.003	.07	17096	16	1	32	1	1k	4	24	512	4	25
.005	.02	17656	8	1	4	0	1k	2	23	512	2	24

Table 1: The best cache configuration differs for different CPU-to-cache bus size regions.

5.4 Tradeoffs between design constraints: old vs. new technology

The following experiments have been conducted to demonstrate the different behavior of the old and the new technology in terms of power con-

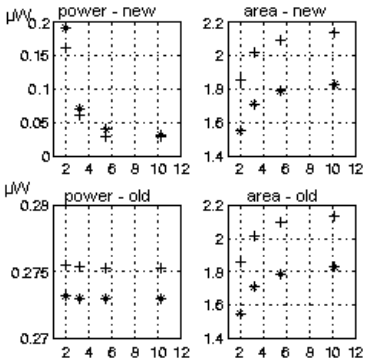

Figure 5: Power/performance tradeoffs for two specific configurations, for new and old technologies – X-axis is execution-time in seconds, Y-axis is power/gates for caches and bus logic only (CPU and main-memory are constant).

sumption. In Figure 5, the plot in the upper left corner shows the power consumption (vertical axis) vs. execution time (horizontal axis). Each point of one set (the set denoted by "+" has a smaller data cache size than set denoted by "*") corresponds to different bus sizes (from left to right: 32, 16, 8, 4). As we can see, the graphs show a strong power/performance tradeoff in dependency of the CPU-to-cache bus. Even if we choose half the cache size ("+"), the qualitative behavior is the same due to the CPU-to-cache bus' dominance. The plot in the lower left corner also shows power vs. performance (for the same application) deploying the same set of parameters as mentioned for the plot above, except that this is for the older technology. However, a significant power/performance tradeoff with respect to bus sizes cannot be observed (i.e., all corresponding points are almost located on a horizontal line). This is because of the already mentioned lower wire/transistor capacitance relationship for older technologies.

In terms of area/performance tradeoff, shown in the upper right corner, (horizontal axis shows execution time, vertical axis shows area), we again have a significant tradeoff for the new technology. But since area is the same (in terms of transistor counts), in this case the old technology shows the same tradeoff (in terms of the shape of the graph; the absolute numbers of the area are different due to different feature sizes, of course).

Obviously, the question whether there is a tradeoff or not depends on the technology as well as on the constraint that is of concern. These are key observations when performing design space explorations. Area and power obviously behave differently, either showing a tradeoff in conjunction with performance or not. A *Branch–and–Bound* technique, for example, could use this knowledge for a fast and efficient search of the design space.

5.5 Simultaneous optimization of buses and caches

The implication of the above data is that cache and bus cannot be optimized independently; they must be optimized in some combined manner. Any approach that tries to separate their optimization may produce very inferior results (especially when newer technologies are deployed). For example, consider a straightforward heuristic that first optimizes cache without considering bus (assuming standard 32-bit buses with negligible capacitance), which essentially represents earlier cache power optimization work done for older technologies, and that second optimizes the bus. While the cache configuration may represent the best power, execution-time and size in some regions, it may represent a rather inferior configuration in other regions. We applied this heuristic to the same two examples above. The heuristic achieves the best power and performance within each region, but does very poorly in terms of size compared to the best sizes in Table 1. For the first example, the sizes obtained for the three regions were 301272, 304472, and 306072, representing size penalties of nearly 50% in the first two regions. The sizes for the second example were 26272, 27872, and 26872, representing size penalties of 70%, 63%, and 52%, respectively. Thus, we see the need for new heuristics that simultaneously optimize cache and bus.

6 Conclusions

In this paper, we presented our explorations on interface (i.e., bus) and cache power consumption in embedded data-intensive systems. Our experimental setup allowed us to explore the complete design space of a system comprising a CPU, instruction/data caches, a main memory and the buses connecting these cores. We conducted experiments on two different technologies for five data–dominated applications, while varying various cache and bus parameters. Our observations can be summarized as follows: (1) In older technologies, there is hardly a tradeoff between power and execution times, i.e., a small execution times implies low power consumption and a large execution time implies high power consumption. (2) Newer technologies do feature a real tradeoff, since bus power consumption becomes more significant due to a higher wire/gate capacitance ratio for smaller feature sizes. (3) The dominating source for power consumption in newer technologies is the CPU-to-cache bus. The selection of this bus' size therefore is the major factor in making the tradeoff between a system's power and performance. (4) For a given CPU-to-cache bus configuration, there is an optimal configuration of remaining cache/bus parameters that minimizes both power and performance. However, there is no optimum set of cache parameters across different CPU-to-cache bus configurations, and therefore the bus parameters of the CPU-to-cache bus parameters must be adapted to one another. (5) Regarding size, for a given CPU-to-cache bus configuration, there is a near-optimal configuration of remaining cache/bus parameters that minimizes power, performance and size. Again, this CPU-to-cache configuration differs for different CPU-to-cache bus configurations, and minor changes to the configuration can yield large penalties in performance or size.

Under these circumstances, a full search through the design space is not typically feasible due to large space and the strong interdependencies between bus and cache parameters. Though we performed a full search, we limited the range of cache and bus parameters to a fraction only, and we still required simulation times of up to four days for a single application. Many more parameters are possible, such as address-bus encoding techniques, data-bus limited-weight codes, and multiplexed address and data buses, thus making the design space exponentially larger. Therefore, further work is needed on heuristics that exploit the peculiarities of the design space. Based on the findings reported in this paper, such heuristics would likely first determine the appropriate region of the power/performance curve through CPU-to-cache bus configuration, and then search the region for the near-optimal power/performance/size cache/bus configurations in that region. We are currently examining such heuristics and are performing research on related problems as part of the Dalton project [17] at University of California, Riverside, in collaboration with NEC USA, Princeton.

References

[1] *TI's 0.07 Micron CMOS Technology Ushers In Era of Gigahertz DSP and Analog Performance*, Texas Instruments, Published in the Internet, http://www.ti.com/sc/docs/news/1998/98079.htm, 1998.

[2] M. Keaton, P. Bricaud, *Reuse Methodology Manual For System–On–A–Chip Designs*, Kluwer Academic Publishers, 1998.

[3] R.K. Gupta, Y. Zorian, *Introducing Core-Based System Design*, IEEE Design & Test of Computers Magazine, Vol. 13, No. 4, pp. 15–25. 1997.

[4] *Virtual Socket Interface Association, Architecture Document*, http://www.vsi.org, 1997

[5] B.P. Dave, G. Lakshminarayana, N.K. Jha, *COSYN: Hardware-Software Co-Synthesis of Embedded Systems*' Proc. DAC'97, pp.703-708, 1997.

[6] I. Hong, D. Kirovski, M. Potkonjak, *Potential-Driven Statistical Ordering of Transformations*, Proc. DAC'97, pp.347-352, 1997.

[7] Ch.Ta Hsieh, M. Pedram, G. Mehta, F.Rastgar, *Profile-Driven Program Synthesis for Evaluation of System Power Dissipation*, IEEE Proc. of 34th. Design Automation Conference (DAC97), pp.576-581, 1997.

[8] V. Tiwari, *Logic and system design for low power consumption*, PhD thesis, Princeton University, Nov. 1996.

[9] Y.Li, J.Henkel, *A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems*, IEEE Proc. of 35th. Design Automation Conference (DAC98), pp.188-193, 1998.

[10] W.Fornaciari, D.Sciuto, C.Silvano, *Power Estimation for Architectural Explorations of HW/SW Communication on System–Level Buses*, To be published at HW/SW Codesign Workshop, Rome, May 1999.

[11] M.R. Stan, W.P. Burleson, *Bus-Invert Coding for Low Power I/O*, IEEE Transactions on VLSI, March 1995.

[12] M.R. Stan, W.P. Burleson, *Limited-Weight Codes for Low Power I/O*, Int. Workshop on Low Power Design, April 1994.

[13] T. Givargis, F. Vahid, *Interface Exploration for Reduced Power in Core-Based Systems*, International Symposium on System Synthesis, December 1998.

[14] Jue-Hsien Chern et. al., *Multilevel Metal Capacitance Models for CAD Design synthesis Systems*, IEEE Electron Device Letters, vol. 13, no. 1, pp.32-34, January 1992.

[15] N.H.E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, 1998.

[16] *National Technology Roadmap for Semiconductors*, Semiconductor Industry Association, 1997.

[17] *The UCR Dalton Project*, http://www.cs.ucr.edu/dalton