A Low-Power Cache Design for CalmRISC™-Based Systems

Sangyeun Cho, Wooyoung Jung, Yongchun Kim, and Seh-Woong Jeong

Media IP Group, Samsung Electronics Co., Yong-In, KOREA

E-mail: sgyncho@samsung.co.kr

Abstract

Lowering power consumption in microprocessors, whether used in portables or not, has now become one of the most critical design concerns. On-chip cache memories tend to occupy dominant chip area in microprocessors, and it becomes increasingly important to design power-efficient cache memories. This paper describes an experimental low-power on-chip cache system designed for a 32-bit processor core called CalmRISC™-32. A number of architectural optimizations were applied to the instruction and data caches, which significantly decrease the number of tag and data memory accesses and the amount of memory traffic to and from off-chip memory. Implemented in a 0.18μm CMOS technology, the presented instruction and data caches consume 90μA/MHz and 72μA/MHz at 1.8V, respectively.

1 Introduction

Reducing power consumption in microprocessors has become a very critical design issue in recent years, and extensive research efforts are being made toward this goal. Cache memory, an integral part of most high-performance and embedded processors, consumes often a significant portion of the power dissipated by an entire processor. For instance, caches in the StrongARM 110 processor consume more than 40% of the total chip power [11]. As the performance requirements of processors keep increasing, the trend is expected to continue.

There are a number of approaches to cache power reduction: advanced process technology, circuit-level memory optimization, voltage reduction, and micro-architectural optimization. All these techniques should be combined together to design the most competitive product possible. From the viewpoint of an ASIC (Application-Specific Integrated Circuit) designer with a severe time-to-market constraint, however, the available optimization space can be greatly narrowed due to limited direct access to the memory circuits, processor technology, and so on. Under such circumstances, architectural power optimizations can become the only choice for the designer. On the other hand, the technology-independent cache power optimization techniques are very important in themselves as they allow easy and quick design reconfiguration and migration.

This paper describes an experimental cache system designed for a 32-bit embedded processor based on the CalmRISC™-32 core[1] [4]. Several architectural techniques were applied that result in significant reduction in memory accesses both on-chip and off-chip. A *line buffering* technique was implemented that eliminates both tag and data memory access for about 80% of all the instruction cache accesses. Both the caches use small cache lines for reduced memory traffic, with additional provisions to decrease the write traffic generated in the data cache. Our design used only standard library cells and compiled memory macros, for shorter design time as well as higher portability. Each of the implemented instruction and data caches occupies $2.4mm^2$ on silion, and consumes around 90μA/MHz and 72μA/MHz at 1.8V, respectively.

The rest of this paper is organized as follows. Section 2 provides background for the discussions in the paper, by describing the rationale of the design decisions made, previous related works, and the experimental setup. Section 3 presents the details of our cache design, followed by the implementation results in Section 4. Concluding remarks are given in Section 5.

2 Background

2.1 Architectural techniques for low-power cache design

Just as we calculate the *average memory access time* (AMAT), the *average memory access energy* (AMAE) of

[1] CalmRISC™ is a trademark of Samsung Electronics Co.

a single memory access can be calculated as follows:

$$\begin{aligned}\text{AMAE} &= (\text{Hit Rate}_{\text{L1}} \times \text{Energy}_{\text{L1 hit}}) \\ &+ (\text{Hit Rate}_{\text{L2}} \times \text{Energy}_{\text{L2 hit}}) \\ &+ (\text{Miss Rate} \times \text{Energy}_{\text{Miss}})\end{aligned}$$

assuming that there is an on-chip level 2 (L2) cache.

From this simple equation and from common knowledge that cache hit rates are typically high ($(\text{Hit Rate}_{\text{L1}} + \text{Hit Rate}_{\text{L2}}) \gg \text{Miss Rate}$) and miss power is much higher than that of an on-chip cache access ($\text{Energy}_{\text{Miss}} \gg \text{Energy}_{\text{L1}}, \text{Energy}_{\text{L2}}$), two important strategies for power optimization are derived. First one is to reduce the power or energy of an access that hits in an on-chip cache. This is especially important for the instruction cache which is accessed almost every cycle. Second strategy is to decrease the miss rate (or in turn increase the on-chip cache hit rate) to avoid paying high energy penalty of accessing off-chip memories via high capacitance bus lines.

Previously studied techniques that lower on-chip cache access power include *line buffering* and *cache subbanking* [16, 6, 1]. A line buffer saves one or more cache lines and behaves as a small, low-power cache. A more generalized form of the same idea, called *filter cache*, was studied by Kin *et al*. [8]. Cache subbanking partitions a cache into a number of banks and activates on an access only a single bank, which is smaller and more energy-efficient than a conventional cache. Another specialized technique to this end is a *loop cache* that saves instructions from a program loop, so that once the program runs in the loop there is no need to fetch the instructions from cache [10].

There are a variety of techniques to increase the on-chip cache hit rate. Multi-level caches and set-associative caches are good examples. Victim caching [5] effectively reduces conflict misses in a direct-mapped cache and can help decrease the power consumption due to miss handling. Effective compiler techniques have been developed, to reduce cache miss rates by restructuring a given program and its data structures [12]. It is noted however that achieving high cache hit rates with constrained silicon without adversely affecting access time is a challenging design goal.

In case of the data cache, a cache miss is not the only cause of generating off-chip memory traffic. Write traffic can be of significant quantity if the cache is not sufficiently large or set-associative. Therefore, it is also one of our prime concerns to decrease the write traffic of the data cache. To reduce the energy associated with off-chip accesses, researches have investigated techniques such as compressing instructions to decrease the amount of memory traffic [2] and encoding bus signals in a way that lowers the bit transition rates [14].

2.2 Experimental setup

For cache simulation with varied configurations, we used the SimpleScalar tool set [3]. We derived our cache simulator from the *sim-cache* simulator. Benchmark programs used in the study are selected from the SPEC95 integer suite [15] and the Mediabench suite [9], which are summarized in Table 1. For power analysis, we used *CubicPower*, an in-house gate-level power estimation tool. Inputs to CubicPower include the gate netlist, capacitance informatation for the netlist, power characterization information (part of the library), and toggle counts from a gate-level simulator.

3 Caches for CalmRISC™

3.1 Chip overview

The chip for which the caches were designed is a portable digital audio encoding and decoding chip. A 32-bit low-power processor core called CalmRISC™ [4] and a 24-bit digital signal processor (DSP) as a coprocessor [7] form the main processing block of the chip. The processor core is a 5-stage pipelined RISC processor, which can run at up to 180 MHz (under worst conditions). An in-circuit debugger unit is also implemented that communicates with programmers or programming tools via JTAG or UART.

Equipped with various on-chip peripherals and memories, the total gate count reaches 1.4 million gates or 5.6 million transistors. On-chip memories comprise a 8-KByte mask ROM, an 32-KByte program RAM, and 96 KBytes of data RAM. On-chip pheripherals include timers, interrupt controllers, smart media interface, USB interface, I^2C, I^2S, and so on. Clocks are drawn from three PLLs. The chip is packaged in a 208-QFP package.

3.2 Determining basic cache parameters

While determining important cache parameters such as cache size, set associativity, and line size, simulation results provided valuable hints.

Cache size. Determining the cache size was more a question of the silicon availability. Initially, we were between 8-KByte and 16-KByte caches, and their sizes were around 50% different – far less than 100% – due to the memory implementation. We chose 16 KBytes since there was more than negligible performance/power degradation with 8 KBytes (over 1% of miss rate) in some of the large benchmark programs such as *compress*, *go*, and *perl*.

Set associativity. Since we were unable to optimize the memory modules at circuit level, simple direct-mapped caches were preferred to set-associative caches. Furthermore, a direct-mapped cache is considered faster and cooler, as detailed in the following subsections.

Benchmark	Description	Input	Inst. Count	Load/Store (%)
go	[Spec95] A *go* game.	train	554M	21.4/7.9
compress	[Spec95] A UNIX utility.	ref	503M	21.2/12.4
li	[Spec95] A *lisp* interpreter.	ref	537M	27.4/19.2
perl	[Spec95] A *perl* interpreter.	ref	525M	27.4/17.3
jpeg.c	[Media] A *jpeg* coder.	monalisa.ppm	339M	23.9/7.6
jpeg.d	[Media] A *jpeg* decoder.	monalisa.jpg	579M	18.8/7.2
mpeg	[Media] An *mpeg* decoder	mei16v2.m2v	171M	15.5/3.8
gsm.c	[Media] A *gsm* coder.	s_16_44.pcm	370M	17.3/4.9
gsm.d	[Media] A *gsm* decoder.	s_16_44.gsm	116M	7.3/4.0
g721.c	[Media] A *g.721* coder.	s_16_44.pcm	439M	13.3/4.1
g721.d	[Media] A *g.721* decoder.	s_16_44.g721	407M	13.8/4.7
rasta	[Media] A noise/distortion filter.	A sample wave file	504M	20.1/9.8

Table 1. Summary of the benchmark programs used. Input, dynamic instruction count, and percentage of dynamic load and store instructions in each benchmark program are presented. Percentage of load or store instructions is relative to the total instruction count.

Line size. The question was to employ a 16-Byte or a 32-Byte line. In terms of the average memory access time (AMAT), an instruction cache with 32-Byte lines outperformed one with 16-Byte lines by less than 2%, while a 16-Byte line data cache had a better AMAT than a 32-Byte line data cache, again by around 2%. If one looks at the *average memory transfer rate* (AMTR), a metric more closely related with energy consumption, adopting a 16-Byte line size gives much better results: 50% and 80% less traffic in the instruction and data caches (0.07 Bytes/access versus 0.11 Bytes/access and 0.28 Bytes/access versus 0.51 Bytes/access), respectively. The line size of 16 Bytes was selected accordingly. One thing to note here is that lowering the high AMTR of the data cache (0.28 Bytes/access compared to the instruction cache's 0.07 Bytes/access) will be very beneficial in saving power.

3.3 Instruction cache design

Instead of a set-associative cache that can lead to higher hit rates, we implemented the simplest direct-mapped instruction cache for several reasons, first of which being its lower power consumption. A normal cache access incurs in a direct-mapped cache an access to data array, an access to tag array, and a tag comparison; On the other hand, multiple memory arrays are accessed, and a number of tag comparisons are done in parallel in a set-associative cache. Furthermore, a direct-mapped cache usually has a shorter access time due to its simpler datapath, in turn providing an opportunity for further power optimization, given a tight timing margin in the target system. It is well-known, however, that a direct-mapped cache suffers from conflict misses unless it is large enough. Experiments showed us that at least 50% of overall instruction cache misses are due to conflicts in programs having relatively high miss rates, such as *go*, *lisp*, *perl*, and *rasta*. To remedy this situation, we implemented a 32-entry miss cache that saves replaced cache lines, similar to the victim cache [5]. At lease 40% and on average 70% of the conflict misses are removed by this addition in the above programs.

Unlike the original victim cache, the implemented miss cache is looked up only after detecting a cache miss in the first cache access cycle. The tag of the fully-associative miss cache is implemented using a *content-addressable memory* (CAM), and accessing it every cycle (thus disspating power) to save a single cycle penalty on a CAM hit is deemed prohibitive. When the access hits in the miss cache, data is accessed in the next cycle, further saving power on miss cache misses. As a result, an access that hits in the miss cache becomes a 3-cycle access. Considering the sub-1% average miss rate of a 16-KByte instruction cache, this access latency is acceptable.

The largest power savings in the instruction cache is coming from *line reuse buffer* (LRB, Figure 1). The LRB is a small buffer that keeps a single cache line so that when the line is accessed the next time, data is provided from this buffer while the actual cache is disabled. Unlike previous works on line buffering [6, 1], our implementation of the LRB eliminates both the tag and data accesses on an LRB hit, by utilizing the sequential access information that comes one cycle early from the CalmRISC^{TM}-32 core. When it is known that the LRB provides the data to the processor, the cache stays in a stand-by mode, and clocking to the cache is blocked. The LRB does not change the timing of a normal cache access and is hidden from software and other hardware components. Hit rates of the LRB reach up to around 85% with an average of 80%.

For short and predictable response time of critical pro-

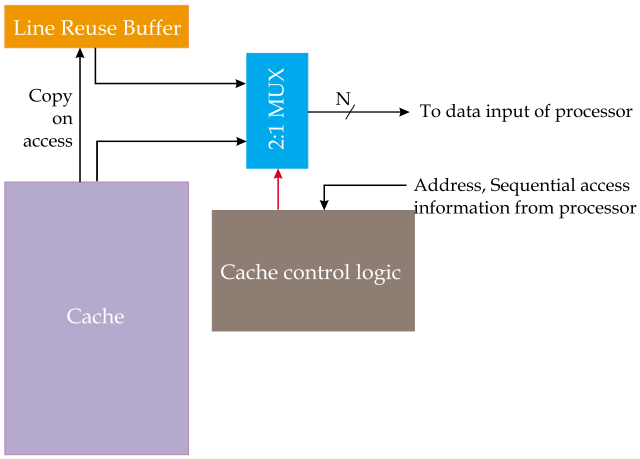

Figure 1. Line Reuse Buffer (LRB) operations. Sequential access information is provided from the processor to the cache control logic, which then determines which of the two, the LRB or the cache, provides the data. While the LRB services the processor, the cache is in the stand-by mode.

gram areas, cache line locking mechanism is provided. Locked lines survive being replaced on conflicting cache accesses. In order not to continuously penalize the conflicting accesses due to already locked cache entries, the cache lines corresponding to those accesses are allocated in the miss cache on their introduction. With help from the LRB, sequential accesses to a line in the miss cache become single-cycle accesses after paying the one-time fee of 3 cycles.

For cache management, means to perform the whole cache invalidation or single line invalidation is provided. In order not to lose performance during miss handling, hot-half-word-first linefill is done with an early processor start capability.

3.4 Data cache design

From the same design considerations, the data cache is also direct-mapped and has 16-Byte lines. A cache line is allocated on any type of misses, read and write. Either write-back or write-through policy is activated by *memory management unit* (MMU) for each memory page accessed, or by setting a control bit in the *data cache control register* (DCCR). To reduce the cache miss latency, the missed word (also called *hot word*) is fetched first regardless of its location in the line to be brought in, and the processor starts on arrival of the hot word without waiting for the whole cache line to fill the cache.

Unlike an instruction cache, a data cache generates write traffic toward off-chip memories. Write-back caches usually have less write traffic than a write-through cache in a similar configuration. To further reduce the write-back traffic, each line in our data cache is associated with two dirty bits, one per each half line. Figure 2 shows that this addition of a single bit, together with help from the miss cache, could reduce the off-chip traffic due to write back by 54% on average. For smooth write traffic generation, a 2-entry write-back buffer and a 4-entry write-through buffer are implemented.

For data cache management, invalidation, synchronization, and flushing are done at the whole cache or at each cache line. Cache line locking is provided as in the instruction cache.

4 Results

4.1 Implementation

We implemented the described cache system in a $0.18\mu m$ CMOS technology. In doing so, we only used a standard cell library (*i.e.*, no custom cells) and compiled SRAM modules [13]. Although there is room for further space/time optimization, we took this approach for shorter design time as well as for higher portability. Each cache occupies around $2.4mm^2$ on silicon. Under worst conditions, the chip with these caches turned on runs at 130 MHz.

The performance of the caches, represented by the average memory access time (AMAT) calculated for the benchmark programs, is 1.003 for the instruction cache and 1.176 for the data cache. This sub-ideal (1-cycle access is ideal) cache performance causes about 6% of the execution delay (CPI is 1.06) if there is no other source for additional execution delay.

4.2 Power consumption

We calculated the average energy consumption of a cache hit and a miss cache hit from the results of running several evaluation programs generating memory accesses that always hit in the cache or in the miss cache. An instruction access that hits in the L1 cache consumes on average $2.8 \times 10^{-10}J$. A miss cache hit followed by the L1 instruction cache miss altogether burns $23.0 \times 10^{-10}J$. In case of the data cache, $7.3 \times 10^{-10}J$ ($13.7 \times 10^{-10}J$) and $23.4 \times 10^{-10}J$ ($24.5 \times 10^{-10}J$) were consumed for a read (write) access that hits in the L1 cache or in the miss cache.

Figure 3 shows the average memory access energy (AMAE) consumed for each memory access toward the instruction and data caches. It is shown that the average AMAE for instructions is only about 10% larger than the cache hit energy thanks to high hit rates. Some programs, such as *go*, *compress*, *li*, and *perl*, exhibit higher data miss rates than others, attributing larger portions of the AMAE

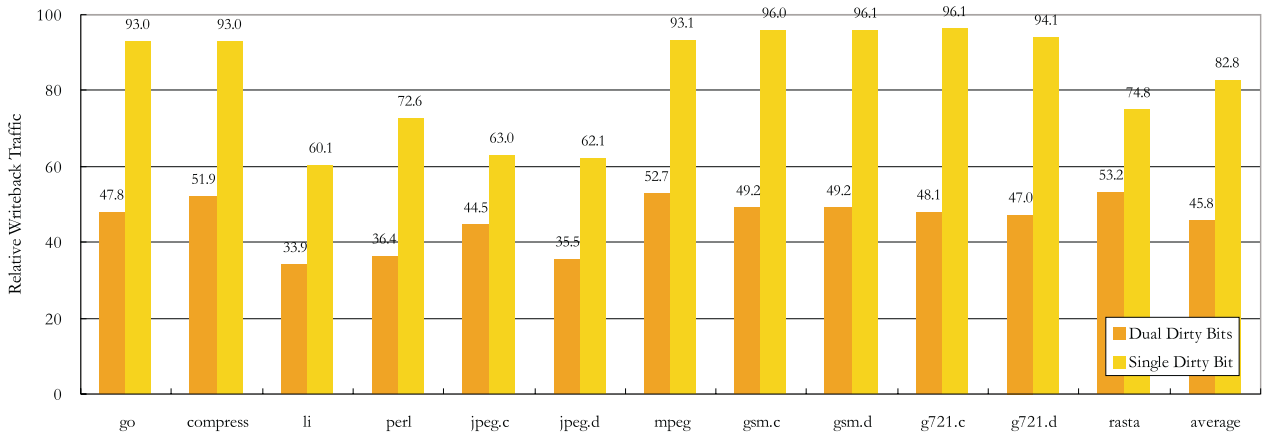

Figure 2. Write-back traffic in data cache. Numbers are relative to a baseline 16-KByte data cache with no miss cache. Employing a 32-entry miss cache reduced the traffic by 17% on average (see right bars). Addition of a second dirty bit to each line cuts the traffic by another 37% (left bars).

to the miss energy. *g721.c* has a lower AMAE than *g721.d* since it has a relatively smaller number of store instructions.

We also measured the currents flowing in the CalmRISC^{TM}-32 core, the instruction cache, and the data cache, while running a compiled quicksort code. After a short warm-up time, the code and the data of the program reside completely in the caches, leaving the processor in a state that observes high power consumption. Measured currents are 82μA/MHz, 90μA/MHz, and 72μA/MHz, from the respective blocks.

5 Concluding Remarks

On-chip cache memory with a low energy-delay product is key to building an efficient low-power system on a chip. This paper describes an experimental on-chip cache system designed for a 32-bit embedded processor. Considerations for low power consumption led us to design rather simple direct-mapped instruction and data caches, each backed up by a small fully-associative miss cache. To decrease the power consumed by frequent accesses that hit in the instruction cache, *line reuse buffer* is devised and implemented, which eliminates both the tag and data accesses in the cache for around 80% of all the accesses. In the case of data cache, efforts were made to decrease the amount of traffic between the cache and off-chip memory, by employing a small cache line and dual dirty bits per line.

Implemented in a $0.18\mu m$ technology, each cache occupies on silicon around $2.4mm^2$ of space. Only standard library cells and compiled memories were used for shorter design time and greater portability, at the inevitable expense of a moderate loss in area and operating speed. Gate-level power simulations show that the designed caches consume 90μA/MHz and 72μA/MHz at 1.8V, respectively.

Acknowledgment

Sanghyun Park, who integrated the chip, has provided valuable feedbacks on this work. Seongshin Paeng and Jachul Koo did the physical design of the chip, and our thanks go to them also.

References

[1] R. I. Bahar, G. Alberta, and S. Manne. "Power and Performance Tradeoffs using Various Caching Strategies," *Proc. of the Int'l Symp. on Low Power Electronics and Design*, pp. 64 – 69, Aug. 1998.

[2] L. Benini, A. Macci, E. Macci, and M. Poncino. "Selective Instruction Compression for Memory Energy Reduction in Embedded Systems," *Proc. of the Int'l Symp. on Low Power Electronics and Design*, pp. 206 – 211, Aug. 1999.

[3] D. Burger and T. M. Austin. "The SimpleScalar Tool Set, Version 2.0," *Computer Sciences Department Tech. Report*, No. 1342, Univ. of Wisconsin, June 1997.

[4] S. Cho, S. Park, S. Kim, Y. Kim, S.-W. Jeong, B.-Y. Chung, H.-L. Roh, C.-H. Lee, H.-M. Yang, S.-H. Kwak, and M.-K. Lee. "CalmRISC^{TM}-32: A 32-Bit Low-Power MCU Core," *Proc. of the 2nd IEEE Asia-Pacific Conf. on Application-Specific Intergrated Circuit*, pp. 100 – 110, Aug. 2000.

[5] N. Jouppi. "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative

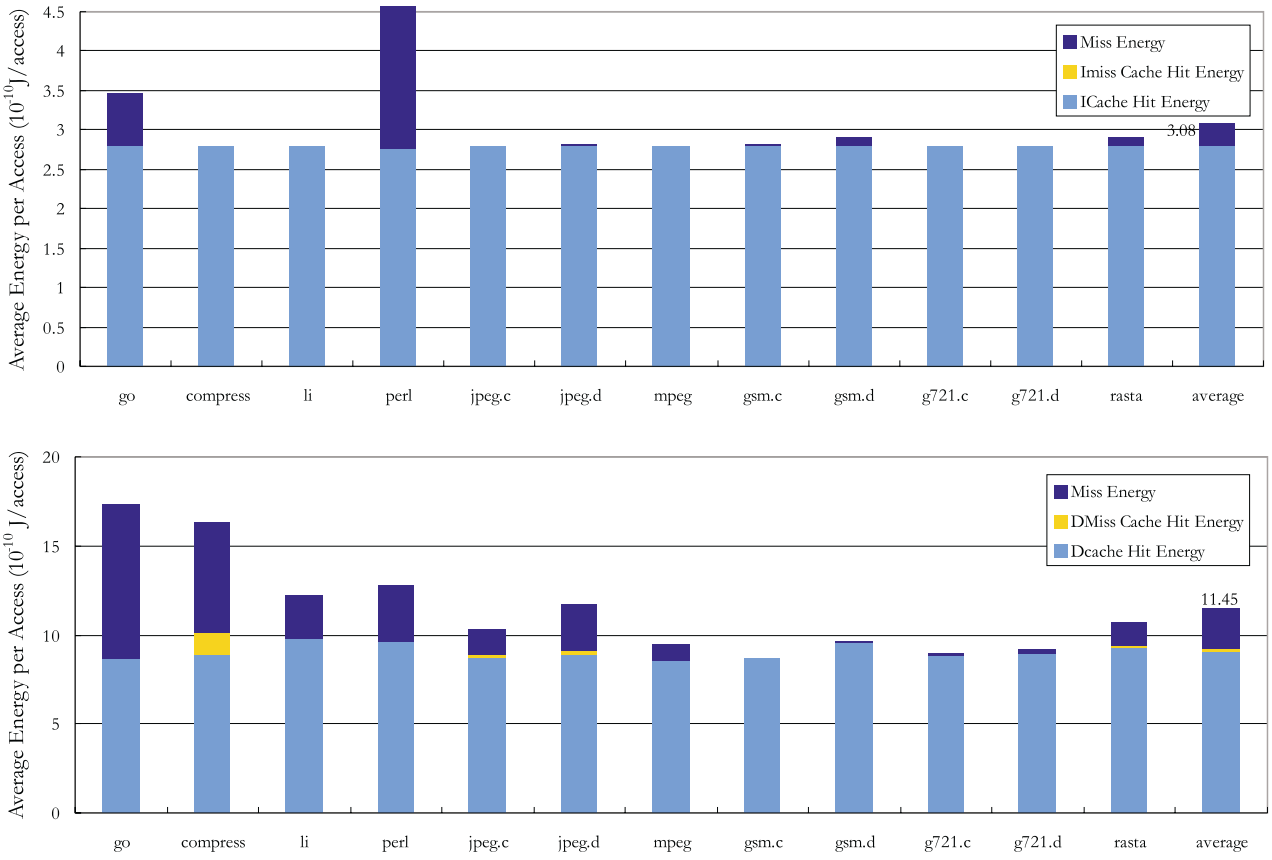

Figure 3. Breakdown of the average energy consumption per memory access. The upper graph is for for instruction access, and the lower graph for data access. The energy for an on-chip cache miss is assumed to be $200.0 \times 10^{-10} J$.

Cache and Prefetch Buffers," *Proc. of the 17th Int'l Symp. on Computer Arch.*, pp. 364 – 373, May 1990.

[6] M. B. Kamble and K. Ghose. "Analytical Energy Dissipation Models For Low Power Caches," *Proc. of the Int'l Symp. on Low Power Electronics and Design*, pp. 143 – 148, Aug. 1997.

[7] H.-K. Kim, K.-M. Lim, Y.-C. Kim, S.-W. Jeong, E.-M. Kim, Y.-H. Kim, B.-Y. Chung, and H.-L. Roh. "A Customizable DSP Coprocessor Extension in Calm-RISC(TM) Microcontroller," *Proc. of the Int'l Conf. on Signal Processing Appl. & Tech.*, Nov. 1999.

[8] J. Kin, M. Gupta, and W. H. Mangione-Smith. "The Filter Cache: An Energy-Efficient Memory Structure," *Proc. of the 30th Int'l Symp. on Microarchitecture*, pp. 184 – 193, Dec. 1997.

[9] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. "MediaBench: A Tool for Evaluating Multimedia and Communications Systems," *Proc. of the 30th Int'l Symp. on Microarchitecture*, pp. 218 – 227, Dec. 1997.

[10] L. H. Lee, Bill Moyer, and J. Arends. "Instruction Fetch Energy Reduction Using Loop Caches For Embedded Applications with Small Tight Loops," *Proc. of the Int'l Symp. on Low Power Electronics and Design*, pp. 267 – 269, Aug. 1999.

[11] J. Montanaro *et al.* "A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor," *IEEE J. of Solid-State Circuits*, Vol. 31, Issue 11, pp. 1703 – 1714, Nov. 1996.

[12] S. S. Muchnick. *Advanced Compiler Design and Implementation*, Morgan Kaufmann Publishers, 1997.

[13] Samsung Electronics Co. *STD130 Databook*, 2001.

[14] M. R. Stan and W. P. Burleson. "Bus-Invert Coding for Low-Power I/O," *IEEE Trans. on Very Large Scale Integration (VLSI)*, Vol. 3, Issue 1, pp. 49 – 58, March 1995.

[15] Standard Performance Evaluation Corporation. `http://www.specbench.org`.

[16] C.-L. Su and A. M. Despain. "Cache Design Trade-offs for Power and Performance Optimization: A Case Study," *Proc. of the Int'l Symp. on Low Power Design*, pp. 63 – 68, April 1995.