An Architectural Level Energy Reduction Technique for Deep-Submicron Cache Memories

Tohru Ishihara Kunihiro Asada

VLSI Design and Education Center, The University of Tokyo
7–3–1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
E-mail: {ishihara,asada}@vdec.u-tokyo.ac.jp

Abstract— An architectural level technique for a high performance and low energy cache memory is proposed in this paper. The key idea of our approach is to divide a cache memory into several number of cache blocks and to activate a few parts of the cache blocks. The threshold voltage of each cache block is dynamically changed according to an utilization of each block. Frequently accessed cache blocks are woken up and others are put to sleep by controlling the threshold voltage. Since time overhead to change the threshold voltage can not be neglected, predicting a cache block which will be accessed in next cycle is important. History based prediction technique to predict cache blocks which should be woken up is also proposed. Experimental results demonstrated that the leakage energy dissipation in cache memories optimized by our approach can be less than 5% of energy dissipation in a cache memory which does not employ our approach.

1 Introduction

An important class of digital systems includes applications, such as video image processing and speech recognition, which are extremely memory-intensive. In such systems, a much power is consumed by memory accesses. In most of today's microprocessors, a cache memory is one of the main power consumers. The on-chip caches of the 21164 DEC Alpha chip dissipates 25% of the total power of the processor. The StrongARM SA-110 processor from DEC, which specifically targets low power applications, dissipate about 27% of the power in the instruction cache[1]. Thus, employing low-power cache memory can greatly reduce the overall energy dissipation in digital systems.

Historically, dynamic energy dissipation caused by charging and discharging capacitive load has been dominant. Therefore, most designers have relied on scaling down the transistor supply voltage to reduce chip's energy dissipation. However, maintaining high transistor switching speeds requires a down-scaling of the transistor threshold voltage. This leads to a dissipation of significant amount of leakage energy even when the transistor is not switching. In [2], Borker estimates that leakage current per total gate width on the die increases about 5 times each generation. Since dynamic energy remains constant (according to scaling theory), static energy dissipation caused by leakage current will become a significant portion of total energy dissipation. An increase of an on-chip cache size makes this situation worse, because leakage energy is a function of the number of on-chip transistors. In order to reduce this undesirable leakage current, several methods have been reported. There has been proposed some circuit techniques which cut off this undesirable leakage current of logic circuits with multiple threshold CMOS (MT-CMOS) when systems are inactive[3]. However, these techniques are not applicable to memories, because they can not maintain the data in memory when the power source is cut off. Energy reduction technique maintaining the data in memory is required for low power memory. Some other techniques (e.g., variable threshold CMOS[4], auto-backgate-controlled MT-CMOS[5]) utilize a backgate bias effect to reduce the leakage current in the sleep mode by rising up the threshold voltage when the circuit does not operate. These techniques, however, impact circuit performance and are only applicable to circuit sections that are not performance-critical. The key technology to utilize these dynamically variable threshold voltage schemes is to decide which part of memory block should be woken up or put to sleep[6].

There are some papers which explore architectural approach to reduce leakage energy[7, 8, 9]. However there leaves still room for improving the leakage energy by controlling the threshold voltage. In this paper, we propose an architectural level technique to reduce leakage current of deep-submicron cache memories. The key idea of our approach is to divide a cache memory into several number of cache blocks and to activate a few parts of cache blocks. The threshold voltage of each cache block is dynamically changed according to an utilization of each block. Frequently accessed cache blocks are woken up and others are put to sleep by a cache controller. Since time overhead to change the threshold voltage can not be neglected, predicting a cache block which will be accessed is important. History based prediction technique to predict cache blocks which should be woken up is also proposed.

The rest of the paper is organized in the following way. In Section 2, we discuss the motivations for our work and present our concept to reduce leakage energy of the cache memories. We propose a selectively activated cache (SAC) architecture and a technique to reduce a leakage energy dissipation by using the SAC architecture in Section 3. Section 4 presents experimental results and discussion on the effectiveness of our approach. Section 5 concludes this paper.

2 Motivations

2.1 Energy and Delay Model

So far, analysis of energy consumption considers only dynamic energy consumption, and most VLSI designer relied on scaling down the supply voltage to reduce chip's energy dissipation. Leakage energy consumption has not been highly significant in the past, but it will be significant in the future, because of the following reasons.

It is well-known fact that lowering the supply voltage causes an increase of circuit delay as shown in (1).

$$t_{pd} \quad \propto \quad \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \qquad (1)$$

t_{pd} denotes propagation delay, V_{dd} and V_{th} denotes supply voltage and threshold voltage of the device, respectively. α is a factor depending on the carrier velocity saturation and is about 1.3 in advanced MOSFETs.

It is clear from (1) that the circuit delay (t_{pd}) in CMOS circuits can be improved by lowering the threshold voltage (V_{th}). However, this causes explosive increase of subthreshold leakage current. The subthreshold leakage energy dissipation can be given by

$$E_{leak} \quad \propto \quad 10^{-\frac{V_{th}}{S}} \cdot V_{dd} \qquad (2)$$

where S is subthreshold factor and its smallest limit at room temperature is 60 mV/dec. Considering these design tradeoffs in CMOS circuits, scaling down the V_{th} of only a small part of cache block is effective way to reduce energy consumption of memory without performance degradation.

2.2 Dual V_{th} SRAM

Figure 1 shows an example of a dual threshold voltage SRAM[5]. This circuit can dynamically change the threshold voltage of SRAM cells and reduce subthreshold leakage current when the memory is in a sleep mode. Here Q1, Q2, Q3, and Q4, which are higher threshold transistors than those for the internal memory circuit, behave as a switch to cut off the subthreshold leakage current. While the memory circuit is operating (which called the "active mode"), Q1, Q2, and Q3 are turned on. The virtual source line, VVDD, becomes 1.7V supplied by the voltage

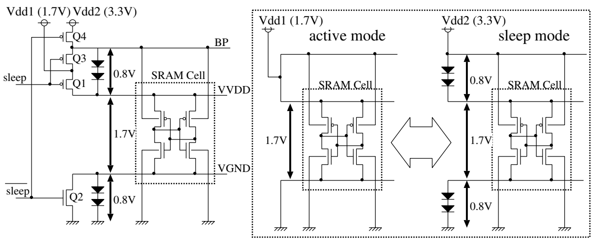

Figure 1: An example of dual V_{th} SRAM structure

source Vdd1 through Q1. The substrate bias, BP, is also forced 1.7V through Q3. Another virtual source line, VGND, is forced to ground level through Q2. In the sleep mode, the VVDD and VGND are connected to Vdd2, and ground respectively, through diodes. The static leakage current, which flows from Vdd2 to ground, decreases significantly compared with that of the active mode, because the threshold voltage of the internal transistors increase by its backgate bias effect. The results of SPICE simulation demonstrate that the leakage current in sleep mode is reduced by about 1/1000 compared with that of the active mode, and we regards the leakage current in sleep mode can be neglected.

2.3 Our Approach

Modern cache architectures are designed to satisfy the demands of the most memory-intensive application. The actual cache utilization, however, varies widely both within and across applications [7]. This means that frequently referred cache lines are dynamically changed according to the condition of the application program and rarely referred cache lines waste large amount of leakage energy for only maintaining data.

In this paper we propose a novel power management cache architecture, a selectively activated cache (SAC) architecture, which can dynamically activate a small number of cache lines and put to sleep the remaining cache lines. The key idea of our approach is to

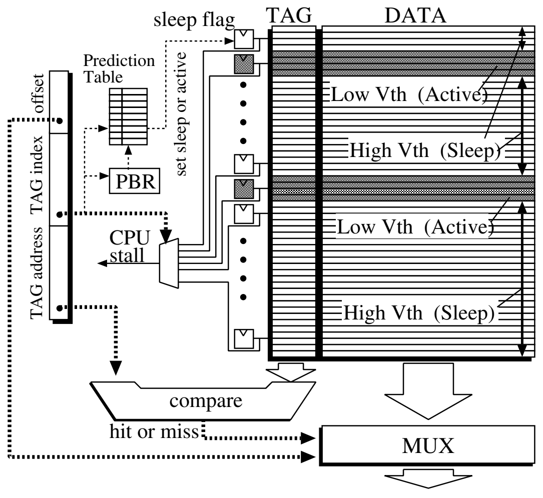

Figure 2: The selectively activated cache architecture

partition a cache memory into several number of cache blocks and to activate only a few parts of cache blocks. The threshold voltage of each cache block is dynamically changed according to an utilization of each cache block. To vary the threshold voltage of cache blocks, we employ the variable V_{th} structure as shown in Figure 1.

3 Selectively Activated Cache Architecture

At first, we present assumptions for our proposed cache architecture. Next, we formally define a problem of leakage energy reduction in deep-submicron cache memories.

3.1 Target System

Our work targets systems which assume the following.

1. The target system consists of a processor and a main memory. The processor has a CPU, and an instruction cache (i-cache) and a data cache (d-cache).

2. The number of clock cycles per memory (main memory) accesses is 1.3.

3. Cache miss penalty for each of a read miss and a write miss is 10 cycle.

4. Cache memory is partitioned into several blocks and each of them can independently selects an active mode or a sleep mode.

5. The number of clock cycles per read access to active cache blocks is 1.

6. Transition time to wake up the slept cache blocks is less than a clock cycle time. Thus, a prediction miss penalty is 1 cycle.

7. The number of clock cycles per read access to slept cache blocks are 2, because it needs 1 clock cycle to wake up and needs one more cycle to read.

8. The number of clock cycles per write access to both slept and active cache blocks are 1, because write time depends on the drive capability of driver circuits.

Feasibility of transition time to wake up the slept cache block is verified by SPICE simulation. The simulation result is shown in Figure 3. We have designed a 4bits × 128word SRAM circuit with $0.5\mu m$ CMOS technology. The SPICE simulation demonstrates that it takes about 9 nanoseconds (which is less than a clock cycle) to wake up the slept memory block.

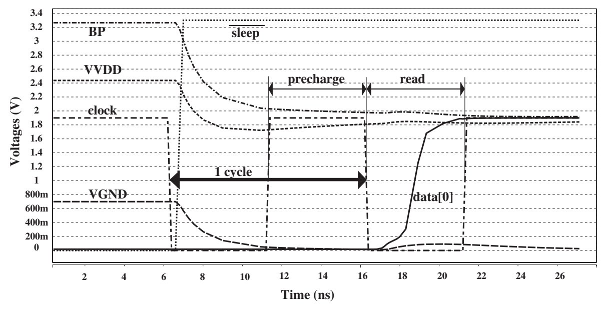

Figure 3: Transition time to wake up a slept memory block

3.2 Wake-up Block Prediction

Let us introduce a new term, R_{hit}. The R_{hit} can be defined as (3).

$$R_{hit} = \frac{\#\ of\ accesses\ to\ the\ active\ blocks}{A\ total\ cache\ access\ count} \quad (3)$$

If a cache access to the sleep blocks is occurred frequently, performance is terribly degraded. Therefore, it is necessary to increase R_{hit} for the SAC architecture to improve performance. To increase the R_{hit}, we employ a prediction table as shown in Figure 4. Cache blocks which are registered in the prediction table are activated and others are put to sleep. For example in Figure 4, when the CPU is reading data from a cache block 0 (current block), block 32, block 1, $\cdots$, and block 19 are woken up. The entries of the prediction table are decided by history information. As the number of entries (ne) in the prediction table is increased, R_{hit} will increase, and performance is improved. However this causes increase of leakage energy, because the number of the active blocks is increased. Therefore the number of the entries of the prediction table (ne) should be decided considering tradeoffs between performance and leakage energy consumption.

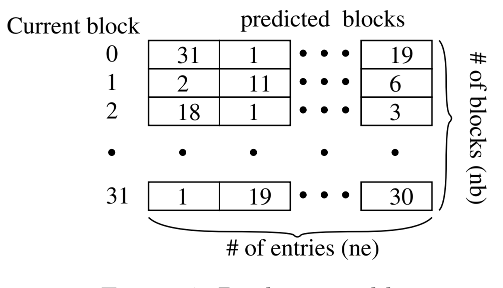

Figure 4: Prediction table

Figure 5 shows a timing diagram of a read access for the prediction table. The prediction table has an address register, named *a previous block address register* (PBR) which keeps previously accessed block address. And also, each cache block has *a sleep flag* which is set to 1 when the corresponding cache block is activated (see Figure 2). If the current memory address is different from the value of the PBR, the sleep flag of the currently accessed block is set to 1, and this block starts to wake up if it was slept. The value of the PBR is also set to the current address. In the same cycle, data of the prediction table is read out and the sleep flag is set to 1. Predicted cache blocks start to wake up in the next cycle, if they were slept.

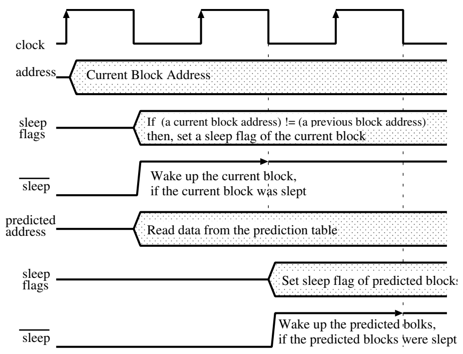
clock
address
Current Block Address
sleep flags
If (a current block address) != (a previous block address) then, set a sleep flag of the current block
sleep
Wake up the current block, if the current block was slept
predicted address
Read data from the prediction table
sleep flags
Set sleep flag of predicted blocks
sleep
Wake up the predicted bolks, if the predicted blocks were slept

Figure 5: Timing diagram of a read access

3.3 Problem Formulation

In this section, we present a problem formulation for a leakage energy reduction problem. Firstly, we give notations used in the formulation. Next, we present a problem formulation.

- ne_i : The number of entries of the prediction table for the i-cache (see Figure 4)
- ne_d : The number of entries of the prediction table for d-cache memory (see Figure 4)
- nb_i : The number of divided blocks of the i-cache memory (see Figure 4)
- nb_d : The number of divided blocks of the d-cache memory (see Figure 4)
- $PM_i(ne_i, nb_i)$: Prediction miss count of the i-cache memory. This variable is a function of ne_i and nb_i.
- $PM_d(ne_d, nb_d)$: Prediction miss count of the d-cache memory. This variable is a function of ne_d and nb_d.
- LS $(= 256\ bits)$: Cache line size.
- IX $(= 128)$: Cache index size.
- TW $(= 20\ bit)$: Bit width of the TAG memory.

$$OBJ = \frac{(LS+TW)\cdot IX}{nb_i}\cdot(ne_i+1)+ne_i\cdot nb_i\cdot log(nb_i)$$
$$+\frac{(LS+TW)\cdot IX}{nb_d}\cdot(ne_d+1)+ne_d\cdot nb_d\cdot log(nb_d) \quad (4)$$

$$PM_i(ne_i, nb_i) + PM_d(ne_d, nb_d) \leq N_{const} \quad (5)$$

Table 1: Description of benchmark programs

Benchmark	Program size in words
Arithmetic Calculator	11,451
TV Remote Controller	15,360
Espresso	62,156

Table 2: Active data size (byte)

Benchmark	Data1	Data2	Data3
Arithmetic Calculator	14,816	15,240	15,164
TV Remote Controller	19,232	19,216	19,220
Espresso	29,260	71,192	151,072

An object function and constraint of this optimization problem are (4) and (5), respectively. The variables to be determined are nb_i, ne_i, nb_d, and ne_d. The object function represents total number of total active bits. We neglect leakage energy of sleep block. Thus, total leakage energy dissipation is in proportional to the number of total active bits. We assume that the prediction table always dissipates leakage current, because it is always active. The dynamic energy dissipation of the prediction table is also negligible, because the prediction table is rarely accessed and its energy is small enough compared with that of the cache memory. The first term of the object function represents total active bits of the i-cache. The second term of the object function is the total bits of the prediction table for the i-cache. The third and fourth terms represent the total bits of d-cache and the total bits of the prediction table for d-cache, respectively. Formula (5) represents that the total prediction miss count must be equal or less than the N_{const}.

The *leakage energy reduction problem* is formally defined as follows. "For a given constraint N_{const}, find ne_i, nb_i, ne_d, and nb_d, which minimize OBJ under the constraint".

4 Experimental Results

We use three benchmark programs: Arithmetic calculator, TV remote controller, and Espresso (a boolean function optimizer). Descriptions of the benchmark programs are as shown in Table 1. The benchmark programs are compiled by gcc-dlx compiler which is based on GNU CC Ver. 2.7.2 for DLX architecture [10]. We got address traces by using *fast* simulator Ver. 0.97 [11] for DLX architecture. Table 2 shows description of three kinds of sample data used as input for each benchmark program. Active data size means the number of accessed addresses. The address size of CPU, cache line size, cache index size, and TAG bit width are 32 bits, 256 bits (8 words), 128, and 20 bits, respectively. A direct mapped cache architecture is assumed in this paper.

At first, we evaluate OBJ, $PM_i(nb_i, ne_i)$, and $PM_d(nb_d, ne_d)$ for different nb_i and nb_d. Only *Data1* is used for each benchmark program. The results are shown in Table 3. Leakage energy, and execution time of both an i-cache and a d-cache are presented. Since leakage energy dissipation is independent of the kinds of application programs and input data in our model, only a single table for leakage energy is appeared in Table 3. The results shown in Table 3 are normalized to the results of the conventional cache architecture which does not employ SAC architecture. The results show that our approach can reduce leakage energy drastically with a small performance degradation. Especially for the i-cache, leakage energy can be reduced by 1/15 at the sacrifice of only 3% of performance.

Table 3: Experimental results for different nb_i and ne_i

Leakage Energy				
nb_i	$ne_i = 0$	$ne_i = 1$	$ne_i = 2$	$ne_i = 3$
8	12.5%	25.07%	37.64%	50.20%
16	6.25%	12.68%	19.11%	25.54%
32	3.13%	6.70%	10.26%	13.86%

Execution Time (instruction cache)				
Arithmetic Calculator				
nb_i	$ne_i = 0$	$ne_i = 1$	$ne_i = 2$	$ne_i = 3$
8	104.10%	102.12%	101.54%	101.17%
16	105.41%	102.80%	102.12%	101.59%
32	107.56%	103.07%	102.12%	101.58%
TV Remote Controller				
nb_i	$ne_i = 0$	$ne_i = 1$	$ne_i = 2$	$ne_i = 3$
8	102.21%	101.18%	100.74%	100.27%
16	103.13%	101.61%	101.05%	100.52%
32	104.73%	101.94%	101.12%	100.62%
Espresso				
nb_i	$ne_i = 0$	$ne_i = 1$	$ne_i = 2$	$ne_i = 3$
8	102.12%	101.13%	100.81%	100.53%
16	102.97%	101.39%	100.96%	100.70%
32	104.74%	101.92%	101.40%	100.92%

Execution Time (data cache)				
Arithmetic Calculator				
nb_d	$ne_d = 0$	$ne_d = 1$	$ne_d = 2$	$ne_d = 3$
8	117.60%	106.58%	101.08%	100.88%
16	117.76%	106.66%	101.45%	101.09%
32	118.17%	107.39%	102.16%	101.66%
TV Remote Controller				
nb_d	$ne_d = 0$	$ne_d = 1$	$ne_d = 2$	$ne_d = 3$
8	113.12%	105.95%	101.87%	101.09%
16	114.32%	107.99%	103.73%	102.35%
32	115.70%	108.58%	104.69%	102.78%
Espresso				
nb_d	$ne_d = 0$	$ne_d = 1$	$ne_d = 2$	$ne_d = 3$
8	112.89%	107.66%	104.59%	102.43%
16	114.30%	109.85%	106.89%	104.76%
32	116.43%	111.82%	109.24%	107.73%

Comparatively, performance degradation of d-cache is large. This is because a reference locality of data memory is smaller than that of instruction memory. However, the prediction table effectively works to improve the performance. For example, if the ne_d is increased form 0 to 1, performance degradation is reduced by half. This demonstrates a high prediction hit ratio.

Table 4: The results of performance degradation

Instruction cache			
Arithmetic Calculator			
Predicted for input data	Data1	Data2	Data3
Data1	101.58%	101.77%	101.59%
Data2	101.92%	101.73%	101.81%
Data3	101.76%	101.76%	101.66%
TV Remote Controller			
Predicted for input data	Data1	Data2	Data3
Data1	100.62%	100.68%	100.68%
Data2	100.65%	100.72%	100.72%
Data3	100.64%	100.71%	100.71%
Espresso			
Predicted for input data	Data1	Data2	Data3
Data1	100.92%	101.07%	101.03%
Data2	101.33%	100.95%	101.41%
Data3	101.00%	101.29%	100.69%

Data cache			
Arithmetic Calculator			
Predicted for input data	Data1	Data2	Data3
Data1	101.66%	101.70%	101.69%
Data2	102.18%	102.05%	102.04%
Data3	102.04%	101.91%	101.91%
TV Remote Controller			
Predicted for input data	Data1	Data2	Data3
Data1	102.78%	102.85%	102.78%
Data2	102.30%	102.33%	102.30%
Data3	102.53%	102.60%	102.53%
Espresso			
Predicted for input data	Data1	Data2	Data3
Data1	107.73%	108.91%	110.99%
Data2	112.62%	110.49%	114.61%
Data3	121.53%	120.41%	115.67%

Next, we evaluate performance of cache memories with different prediction tables' entries. The results are shown in Table 4. In the previous experiment, entries of the prediction tables are determined by the history information of *Data1*, and the prediction hit counts are also evaluated by using *Data1*. However, determining optimal entries of the prediction table is one of the key points of our technique. The second line of each sub-table in Table 4 represents the input data which are used for determining the entries of the prediction tables. The results shown in Table 4 demonstrate that the performances of cache memories weakly depend on the kinds of data which are used for determining the entries of the prediction tables. This means that we can successfully determine the entries of prediction tables by a trace information of randomly selected input data.

Finally, we evaluated leakage energy of cache memories under performance constraints. The results are shown in Figure 6. The leakage energy results are mean values of the leakage energy of the i-cache and the d-cache. All execution time results shown in Table 3 and Table 4 are calculated by (6).

$$\frac{T_c + PM_i(ne_i, nb_i) + PM_d(ne_d, nb_d)}{T_c} \times 100(\%) \quad (6)$$

Here, T_c denotes execution time of conventional system which does not employ SAC architecture. The $PM_i(ne_i, nb_i)$ and $PM_d(ne_d, nb)$ are explained in Section 3.3. The results demonstrated that the leakage energy dissipation in cache memories optimized by our approach can be less than 5% of energy dissipation in cache memory which does not employ our approach.

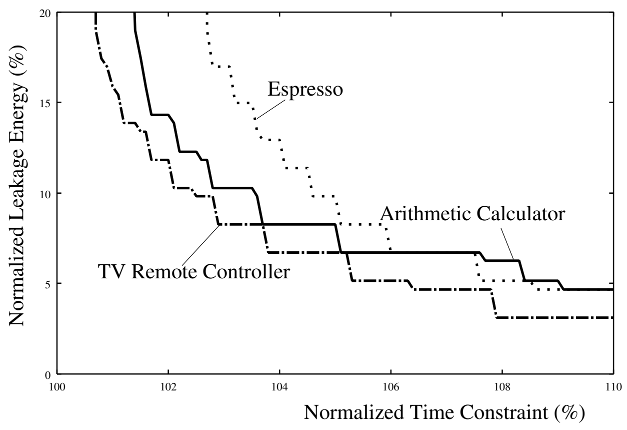

Figure 6: Leakage energy results under time constraints

5 Conclusion

As device size is shrunk, supply voltage of CMOS circuits also is scaled down. In near future, 1.5V or 1.0V supply voltage will become common. In such era, scaling the threshold voltage becomes to have strong impacts on both energy consumption and circuit delay. In this paper, we have proposed a selectively activated cache (SAC) architecture, and a technique to reduce leakage energy dissipation by using the SAC architecture. Experimental results demonstrated that the leakage energy dissipation in cache memories optimized by our approach can be less than 5% of energy dissipation in cache memory which does not employ our approach. Our approach will become more important for complex and low-power system on a chip (SOC) design in near future, because current semiconductor technology enables integrating larger memories on a single chip. This results in the need for a technique to reduce leakage energy dissipation.

Our future work will be devoted to extend the proposed technique considering general cache models such as a set associative cache memory.

References

[1] Nikolaos Bellas, and Ibrahim Hajj,. “Architectural and Compiler Support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors”. In *Proc. of ISLPED’98*, pages 70–75, 1998.

[2] S. Borker. “Design Challenges of Technology Scaling”. *IEEE Micro*, 19(4):23–29, July 1999.

[3] S. Mutoh S. Date, N. Shibata and J. Yamada. “1-V, 30-MHz Memory-Macrocell-Circuit Technology with a 0.5μm Multi-threshold CMOS”. In *Proc. of IEEE Symposium on Low Power Electronics*, pages 90–91, 1994.

[4] T. Kuroda and T. Sakurai. “Threshold-voltage control scheme through substrate-bias for low-power highspeed CMOS LSI design”. *Kluwer J. of VLSI signal processing, special issues on technologies for wireless computeiog*, 1996.

[5] K. Nii, H. Makino, Y. Tujihashi, C. Morishima, and Y. Hayakawa. “A Low Power SRAM using Auto-Backgate-Controlled MT-CMOS”. In *Proc. of Int’l Symposium on Low Power Electronics and Design*, pages 293–298, 1998.

[6] T. Ishihara and K. Asada. “A System Level Memory Power Optimization Technique Using Multiple Supply and Threshold Voltages”. In *Proc. of ASPDAC’01*, pages 456–461, Jan. 2001.

[7] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T.N. Vijaykumar. “Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories”. In *Proc. of ISLPED’00*, pages 90–95, 2000.

[8] S.-H. Yang, M. Powell, B. Falsafi, K. Roy, and T.N. Vijaykumar. “An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches”. In *Proc. of HPCA’01*, Jan. 2001.

[9] S. Kaxiras, Z. Hu and M. Martonosi. “Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power”. In *Proc. of ISCA’01*, pages 240–251, 2001.

[10] J. L. Hennessy and D. A. Patterson. *“Computer Architecture: A Quantitative Approach”*. Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.

[11] http://www-mount.ece.umn.edu/˜okeefe/mcerg/fast-dlx/.