Cache Power Reduction in Presence of Within-Die Delay Variation using Spare Ways

Goudarzi, Maziar
Kyushu University

Matsumura, Tadayuki
Kyushu University

Ishihara, Tohru
Kyushu University

Cache Power Reduction in Presence of Within-Die Delay Variation using Spare Ways

Maziar Goudarzi, Tadayuki Matsumura, Tohru Ishihara
Kyushu University, Fukuoka, Japan
{goudarzi, ishihara}@slrc.kyushu-u.ac.jp   matsumura@c.csce.kyushu-u.ac.jp

Abstract

The share of leakage in cache power consumption increases with technology scaling. Choosing a higher threshold voltage (V_{th}) and/or gate-oxide thickness (T_{ox}) for cache transistors improves leakage, but impacts cell delay. We show that due to uncorrelated random within-die delay variation, only some (not all) of cells actually violate the cache delay after the above change. We propose to add a spare cache way to replace delay-violating cache-lines separately in each cache-set. By SPICE and gate-level simulations in a commercial 90nm process, we show that choosing higher V_{th}, T_{ox} and adding one spare way to a 4-way 16KB cache reduces leakage power by 42%, which depending on the share of leakage in total cache power, gives up to 22.59% and 41.37% reduction of total energy respectively in L1 instruction- and L2 unified-cache with a negligible delay penalty, but without sacrificing cache capacity or timing-yield.

1. Introduction

The share of leakage in total power consumption of cache memories increases considerably with technology scaling since dynamic power reduces and static power increases. The naïve solution is to increase V_{th} and/or T_{ox} to exponentially reduce respectively sub-threshold- and gate-leakage, but this impacts cache access delay. In addition, *within-die variation of access-delay of SRAM cells* is becoming more severe in sub-90nm technologies [1], and are predicted to only rise when further approaching atomic sizes by technology scaling [1]. *Within-die* delay variation means similar SRAM cells on the same die show different access delays. The spatially-correlated component of this variation similarly affects neighboring SRAM cells and is best compensated by redundant rows/columns, but the *uncorrelated random component* randomly affects cell delays in the chip; empirical study [2] shows the latter variation is 3.54% for a single logic element (roughly equivalent to a single SRAM cell) in 90nm FPGAs. At such variation, if the cache is implemented with a higher V_{th}, T_{ox} to reduce leakage, randomly some of the cells (but not all of them) violate the original timing. We propose to use extra cache ways to compensate for them per cache-set.

Our design- and manufacturing-time optimization technique, *(i)* keeps V_{DD} untouched (to avoid its quadratic impact on dynamic power), *(ii)* optimally chooses a higher V_{th} and T_{ox} at cache design time, and *(iii)* adds a few extra cache ways enough to compensate for timing-violating cache lines caused by higher V_{th} and T_{ox}. We use analytical formula to ensure that *(i)* timing-yield is kept intact (except a negligible delay overhead), and *(ii)* each cache-set keeps its original number of delay-meeting cache-lines assuming Gaussian distribution for the uncorrelated random within-die delay variation; consequently, cache capacity is also kept intact (or even increased). The extra cache ways potentially increase dynamic power per access, but simulations of caches implemented using a commercial 90nm process show that in instruction caches (where most accesses are sequential) and in L2 caches (where leakage dominates total power) the total power can still be significantly reduced. The additional cache-ways also imply a wider way-selector multiplexer with negligible delay overhead of 3.12% in a 4-way 16KB cache.

2. Related works

To tolerate slow/faulty cells, programmable address decoders [4] redirect accesses to slow/faulty cache lines to other lines in the same cache-set. A simpler technique [5] uses an unused combination of flag bits to mark the cache-line as faulty and to avoid storing data there. Any such technique can be used in our work to mark and avoid using delay-violating lines.

Ozdemir et. al [6] propose to turn off delay-violating and too leaky cache ways or lines. They also propose to allow different parts be accessed at different latencies. Although improving chip yield, these affect cache capacity or speed, but we keep original yield and capacity with negligible speed impact. We reduce leakage by higher T_{ox} and V_{th}, not by turning parts off.

Within-die leakage variation is considered in [7] to extend *selective cache ways* [8] by starting from the leakiest cache way when disabling those not used by the application. We do not reduce leakage by *disabling* parts of cache. Moreover, we operate at a per-cache-set basis when replacing slow cache lines with spares.

ILWM technique. Various techniques reduce cache dynamic power [9]. A well-known one applicable to set-associative instruction caches is [10]; since instructions mostly execute sequentially, and several instructions reside in the same cache-line, tag-comparisons can be eliminated and only one cache way activated unless last executed instruction was either a branch or was at the end of a cache-line. We call this *Inter-Line Way Memorization (ILWM)* and use it in experiments.

3. Our approach

Motivational Example. The left-hand side of Fig. 1 shows a 4-way cache implemented with V_{th}=270mv. The enlarged cache-line demonstrates within-die delay variation: SRAM cells have different latencies (for presentation we show 4 bits per cache line). At the right-hand side, V_{th} is 50mv higher (320mv). Fig. 2 gives SPICE simulation results of leakage vs. delay of standard 6T SRAM cells using a commercial 90nm process and shows that delay increases and leakage decreases by raising V_{th},T_{ox}. Thus, in the right-hand side of Fig. 1 cell latency increases (compare the two enlarged parts). To compensate for the delay-violating cache-lines, one spare way is added to ensure at least 4 out of 5 cache lines still meet target delay as before. Thus, the cell-array latency is kept at the original 642ps (way-selector delay increases; not shown for simplicity) and the cache capacity also remains intact. The choice of higher V_{th} reduces leakage by 42% including all 5 cache ways (Section 5 and Table I). For practicality we do not turn off slow cache lines; they are invalidated and locked by software at boot time as [5].

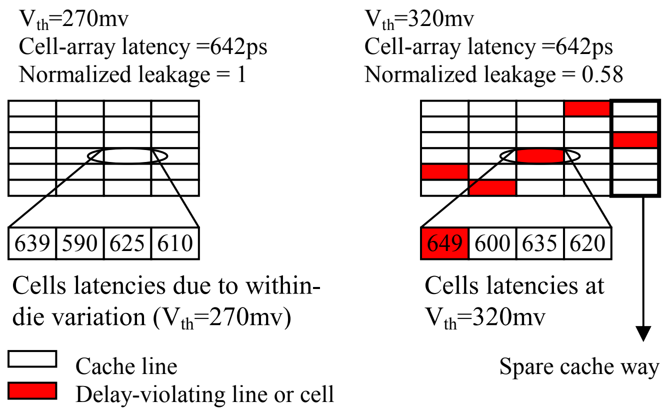
V_{th}=270mv
Cell-array latency =642ps
Normalized leakage = 1
V_{th}=320mv
Cell-array latency =642ps
Normalized leakage = 0.58
639 590 625 610
649 600 635 620
Cells latencies due to within-die variation (V_{th}=270mv)
Cells latencies at V_{th}=320mv
Spare cache way
Cache line
Delay-violating line or cell

Fig. 1. One spare way is added to a 4-way cache to let choose 50mv higher V_{th} without impacting neither cell-array latency nor capacity.

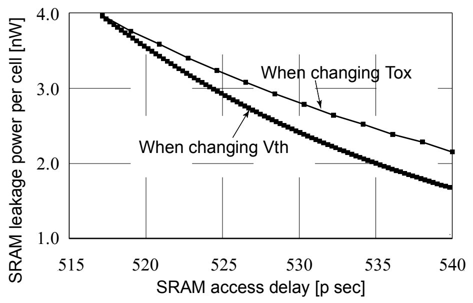

Fig.2. Mean leakage power vs. mean access-delay of a single SRAM cell when raising V_{th} and T_{ox} (from left to right) in a commercial 90nm process in presence of 5% delay variation.

Our Approach. Fig. 3 outlines our proposal. Our technique determines number of extra cache-ways and the manufacturing options of T_{ox} and V_{th} for SRAM transistors. Original cache organization (size, line-size, and number of ways) along with process characteristics (mean and standard-deviation of SRAM cell delay as well as leakage-delay curves of cells at various V_{th} and T_{ox} values) are input to the optimization program. Cache organization is modified according to the results while the chosen T_{ox} and V_{th} are handed over to the manufacturer for chip fabrication. The produced chips are then tested offline to detect and mark cache lines containing slow SRAM cells. If such slow lines per cache-set exceed number of spare ways, the chip is considered faulty and contributes to yield loss. If practical, such slow lines should be turned off but we merely rely on marking them at boot-time after their location is read from an agreed-upon non-volatile storage. Marking of slow cache lines can be done in software by clearing their *valid* bit and setting their *lock* bit [3][5]. Finally, at runtime the cache works as ever without noticing and without using slow lines.

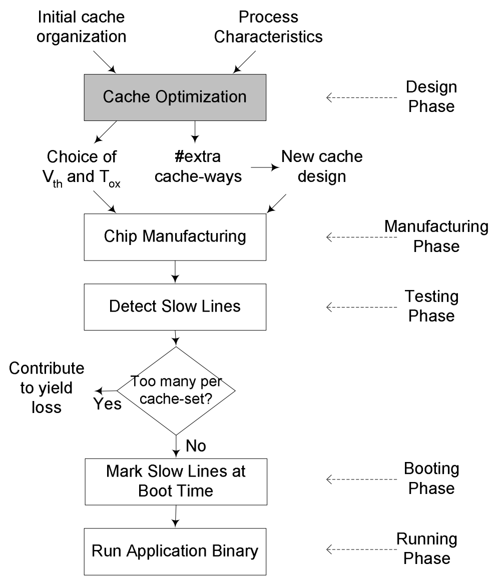

Fig.3. Big picture of our proposed approach.

4. Problem formulation and algorithm

Using the following notations, we define the problem:
***μ_d*:** original mean delay of SRAM cells.
***σ_d*:** original standard deviation of delay of SRAM cells.
D: target delay of the cache.
***N*:** number of additional cache ways.
***Y*:** original timing-yield of the cache.
***Y_{cell}*:** timing-yield of a single SRAM cell.
***w*:** original number of ways in the cache.
b: number of bits per cache line (including tag bits).
s: number of cache-sets.
***L*:** leakage power of the cache.
***V_{th}*:** optimal value for V_{th} of cache SRAM transistors.
***T_{ox}*:** optimal value for T_{ox} of cache SRAM transistors.
Problem: *"For a given process technology (μ_d, σ_d), cache organization (w, b, and s), and timing-yield (Y), minimize the leakage power of cache (L) by setting V_{th}, T_{ox}, and N such that target delay, D , is kept unchanged (ignoring delay penalty of way-selector multiplexer that is measured and considered separately)."*

Algorithm. The following algorithm takes the cache organization and process technology as input and provides the best choice of V_{th}, T_{ox}, and N if successful.

```
Algorithm 1: OptimizeCacheDesign()
Inputs: ($\sigma_d$,$\mu_d$: process tech. characteristics),
        (w,b,s: original cache configuration),
        (Y:     Target timing-yield of cache)
Output: set of (N, $V_{th}$, $T_{ox}$) triples.
1 set answers_set = empty_set
2 compute D based on Y, $\sigma_d$, $\mu_d$.
3 compute L (leakage power) of original cache.
4 for N=1 to w/2 do
4.1 compute new $\mu_d$ in presence of N extra
    cache-ways such that D and Y are kept intact.
4.2 compute $V_{th}$ and $T_{ox}$ corresponding to new $\mu_d$
4.3 compute L' (leakage power) of new cache
4.4 if L'<L then add (N,$V_{th}$,$T_{ox}$) to answers_set
```

Below we show the relation between N and the nominal V_{th} and T_{ox} of SRAM transistors. The algorithm simply assumes various values for N up to roughly 50% area overhead (line 4), computes the corresponding V_{th} and T_{ox} (lines 4.1 and 4.2), and then checks whether they sufficiently reduce the leakage so that the static power of the cache with the extra way(s) is less than before or not (lines 4.3 and 4.4).

Uncorrelated random within-die delay variation can be modeled by Gaussian distribution [3]. Thus, probability Y_{cell} that a SRAM cell delay is less than target-delay D, is given by the area below probability density function (PDF) of Gaussian distribution up to D:

$$Y_{cell} = \Pr[x \le D] = \int_{-\infty}^{D} f(x)dx \quad f(x) = \frac{1}{\sigma_d\sqrt{2\pi}} e^{\frac{-(x-\mu_d)^2}{2\sigma_d^2}} \tag{1}$$

where *f(x)* is the PDF of Gaussian distribution. Now the original yield of the entire cache, Y, is:

$$Y = Y_{cell}^{\,b\times w\times s} \tag{2}$$

With N extra ways, each set is still fine as long as it contains 0 to N slow cache lines. Thus, total yield is:

$$Y = \left(\sum_{i=0}^{N} \binom{w+N}{i} \times \left(Y_{cell}^{\,b}\right)^{w+N-i} \times \left(1-Y_{cell}^{\,b}\right)^{i}\right)^{s} \tag{3}$$

In line 2 of the algorithm, D is computed using Eq. 1 and 2 for the given μ_d, σ_d, and Y. Leakage-delay curves (Fig.2) are used to compute the cache total leakage in line 3 (and also later in line 4.3). In line 4.1, we numerically compute new μ_d using Eq. 1 and 3. Transistors in spare cache-ways also contribute to leakage, and hence, if their overhead is too high, leakage may not be improved. Thus, if leakage decreases (line 4.4), the corresponding new V_{th} and T_{ox} are reported. These new V_{th} and T_{ox} (line 4.2) are available from SPICE simulations done to produce leakage-delay curves (V_{th} and T_{ox} values are not shown in Fig.2, but are recorded when running SPICE simulations for generating Fig.2).

5. Experimental results

We designed SRAM cells using SPICE transistor models of a real 90nm process technology (undisclosed due to NDA) with 1V supply voltage and implemented other cache circuitry using Synopsys Design Compiler synthesis tool and a 90nm standard cell library. Leakage and dynamic power and delay were obtained by SPICE simulations for SRAM cells, and by Synopsys tools for cache periphery. SRAM cells dominate cache leakage and also cache area. Thus, we focus on variation in delay of SRAM cells since cache periphery occupies a small area and can be properly sized to minimize variation. In the rest of this paper, "cache leakage" means "SRAM cells leakage". Monte Carlo simulation was used at various delay variations to obtain leakage-delay curves for SRAM cells (Fig.2) with their corresponding V_{th} and T_{ox} (not shown in Fig.2).

Results and overheads. Results obtained on 16KB, 4-way cache with 256 data and 20 tag bits per line are given in Table I; we save more when targeting a higher yield as well as at higher delay variation. Algorithm execution time is just a fraction of second on a Xeon 3.80GHz processor with 2MB of cache and 3.5GB of memory, but it took a week to run all SPICE simulations on the same machine to obtain the tables (see Fig.2) used in steps 3 and 4.3 of the algorithm; but it doesn't matter since this is a one-off task.

Table I. Obtained leakage reduction on a 16KB 4-way cache

Delay variation	Target Yield	Cache leakage power (μW)			Saving (%)	
		ORG	N=1	N=2	N=1	N=2
3%	90%	76.16	69.18	72.99	9.16	4.16
	95%	80.03	70.70	74.85	11.65	6.47
	99%	90.80	76.73	78.34	15.50	13.72
5%	90%	552.4	350.58	319.90	36.54	42.09
	95%	616.2	376.26	336.81	38.94	45.34
	99%	752.1	438.98	377.04	41.63	49.87

Table II gives resulting changes of static power at

3% and 5% delay variation, dynamic energy per access, and also delay of cache (negative value shows an improvement); target timing-yield is 99%. By way-scaling, delay increases due to widened way-selector which also increases dynamic power (further increased by more tag-comparisons caused by extra ways except for single-way accesses, enabled by ILWM [10]).

Table II. Change of static power (99% timing-yield), dynamic energy, and delay of 16KB 4-way cache after applying our technique.

		Change (%)	
		1 extra way	2 extra ways
Leakage Power	3% delay variation	-15.50	-13.72
	5% delay variation	-41.63	-49.87
Dynamic Energy per access	Hit (Single-way access)	7.74	15.47
	Hit (Full-way access)	24.89	49.78
	Miss	15.10	30.20
Total cache access delay		3.12	5.67

Total power reduction. We tradeoff dynamic- for static-power, and hence, final outcome depends on the share of them in total power. We analyzed the ratio of static to dynamic power in an embedded processor, M32R-II, when running a number of benchmarks. The processor runs at 200MHz and has separate L1 instruction and data caches: a 16KB 4-way set-associative cache with 32-byte lines for instructions and the same organization but 16-byte cache-lines for data. It is also equipped with a 16KB unified L2 cache with the same organization as the L1 instruction-cache. We used four applications from MiBench and compiled them once with no compiler option and once with '-O3' full-optimization option. HDL model of the processor was simulated for 1 million instructions of each benchmark to gather number of clock cycles and cache access statistics (i.e. number of hits, misses, and also single-way accesses and full-way accesses for the instruction cache). (Reports omitted for lack of space.) Fig.4 shows breakdown of L1 instruction-cache energy consumption assuming 5% delay variation and 99% target timing-yield; the bars are respectively original cache, cache with one extra way (N=1), and cache with two extra ways (N=2). The results include ILWM [10].

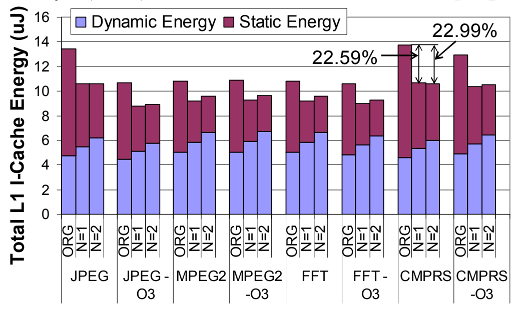

Fig.4. Total energy in L1 Instruction-cache.

In data cache, due to smaller size (and hence less leakage) and inapplicability of ILWM (since accesses are mostly not sequential), our technique is not as useful and in some cases even marginally increases total power by up to 1.10% (diagram omitted for space). Leakage energy comprises a bigger part in L2 caches since such caches are less frequently accessed. Thus, our technique is more effective here (Fig.5).

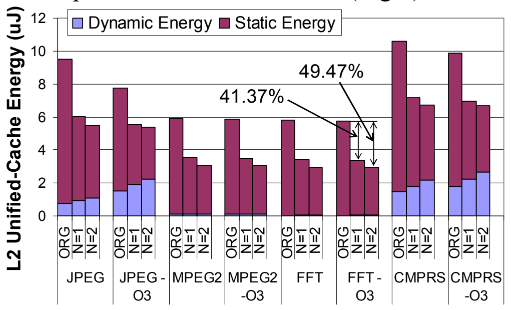

Fig.5. Total energy in L2 unified cache.

6. Summary and conclusion

We reduced cache leakage by using higher V_{th} and T_{ox} while adding extra cache ways; the former reduces leakage while the latter compensates for the resulting delay-violating cache lines. Total power reduction depends on the balance of static to dynamic power, but we showed over 40% reduction in a 16KB L2 cache.

Acknowledgments. This work is supported by VDEC, The University of Tokyo with collaboration of STARC, Panasonic, NEC Electronics, Renesas Technology, and Toshiba. It is also supported by CREST project of JST. We are grateful for their support.

7. References

[1] International Technology Roadmap for Semiconductors—Design, 2006 Update, http://www.itrs.net

[2] P. Sedcole, P.Y.K. Cheung, "Within-die delay variability in 90nm FPGAs and beyond," Proc. Field-Programmable Technology (FPT), 2006.

[3] Agarwal, B.C. Paul, H. Mahmoodi, A. Datta and K. Roy, "A process-tolerant cache architecture for improved yield in nanoscale technologies," *IEEE Trans. on VLSI,* vol. 13, no. 1, pp. 27-38, 2005.

[4] P.P. Shirvani and E.J. McCluskey, "PADded cache: a new fault-tolerance technique for cache memories," *Proc. IEEE VLSI Test Symp.*, pp. 440-445, April 1999.

[5] T. Ishihara, F. Fallah, "A cache-defect-aware code placement algorithm for improving the performance of processors," *Proc. Int'l Conference on Computer-Aided Design*, pp. 995-1001, 2005.

[6] S. Ozdemir, D. Sinha, G. Memik, J. Adams, H. Zhou, "Yield-aware cache architectures," *Int'l Symp. on Microarchitecture*, 2006.

[7] K. Meng, R. Joseph: "Process variation aware cache leakage management," *Int'l Symp. Low Power Elec. and Design*, 2006.

[8] D. Albonesi, "Selective cache ways: on-demand cache resource allocation," *Proc. Int'l symp. on Microarchitecture*, 1999.

[9] V.G. Moshnyaga, K. Inoue, "Low-Power Cache Design," in Low-Power Electronics Design, C. Piguet Eds., CRC Press, 2005.

[10] R. Panwar and D. Rennels, "Reducing the frequency of tag compares for low-power I-cache design," *Proc. Int'l Symp. Low Power Electronics and Design*, pp. 57-62, 1995.

[11] Y. Tsukamoto, et al., "Worst-case analysis to obtain stable read/write DC margin of high density 6T-SRAM-array with local V_{th} variability," *Proc. Int'l Conf. Computer-Aided Design (ICCAD)*, 2005.