Reducing Execution Unit Leakage Power in Embedded Processors

Houman Homayoun* and Amirali Baniasadi[1]

[1]Electrical and Computer Engineering Department
University of Victoria, Victoria, Canada
houman@houman-homayoun.com,amirali@ece.uvic.ca

Abstract. We introduce low-overhead power optimization techniques to reduce leakage power in embedded processors. Our techniques improve previous work by a) taking into account idle time distribution for different execution units, and b) using instruction decode and control dependencies to wakeup the gated (but needed) units as soon as possible. We take into account idle time distribution per execution unit to detect an idle time period as soon as possible. This in turn results in increasing our leakage power savings. In addition, we use information already available in the processor to predict when a gated execution unit will be needed again. This results in early and less costly reactivation of gated execution units. We evaluate our techniques for a representative subset of MiBench benchmarks and for a processor using a configuration similar to Intels Xscale processor. We show that our techniques reduce leakage power considerably while maintaining performance.

1 Introduction

The goal of this work is to reduce leakage power in embedded processors. In recent years, we have witnessed a rapid complexity increase in the embedded space. As a result, embedded processors power dissipation has become one of the major barriers in their deployment in mobile devices. Meantime, as the semiconductor technology scales down, leakage (standby) power will account for an increasing share of processor power dissipation [1, 2].

In most processors, including embedded processors, computational units power dissipation accounts for a considerable share of total power dissipation. However, and as we show in this work, computational units may be idle for long periods of time depending on the applications required resources. During such idle periods, execution units consume energy without contributing to performance.

We investigate embedded processors and show that there is an opportunity to reduce leakage power dissipated by idle execution units. In particular, we show that execution units may be idle for long periods of time. Identifying such idle periods accurately provides an opportunity to reduce power while maintaining performance.

* The author was with the University of Victoria, Electrical and Computer Engineering Department when this work was done.

To reduce power dissipation, we turn off the voltage supply for execution units that are detected to be in their idle time.

One way to detect idle execution units is to monitor the units and to gate them if they are idle for a consecutive number of cycles [3]. This is referred to as time-based power gating. This approach has two inefficiencies. First, the time overhead associated with this method could be costly. Particularly the energy savings are very sensitive to the time needed to wakeup gated units. Second, as we show in this work, different functional units have different idle time distributions. While some execution units may be idle for long periods there are others that stay idle for short periods. Therefore, a one-size-fits-all approach fails to provide optimal results across all units.

In this work we introduce new heuristics to address both inefficiencies. We improve previously suggested techniques by using different idle time detection thresholds and by using control dependency and decode information to wakeup gated execution units early in embedded processors.

In particular we make the following contributions:

1. We show that there is an opportunity in the embedded space to reduce leakage power by identifying idle execution units.
2. We improve previously suggested leakage reduction techniques as we detect idle periods more effectively. Consequently we increase energy savings.
3. We reactivate gated (but needed) units earlier than the time they are reactivated using previously suggested methods. Consequently, we reduce the performance cost.

Note that there is a timing overhead associated with power gating. We take into account this overhead in this study.

The rest of the paper is organized as follows. In section 2 we discuss related work. In section 3 we discuss power gating in more detail. In section 4 we discuss our motivation and present our techniques. In section 5 we review methodology, present our analysis framework and present performance and power savings results. Finally, in section 6 we offer concluding remarks.

2 RELATED WORK

Leakage power may become a more serious issue in embedded processors where applications may require long periods of inactivity [4, 5]. Accordingly, previous study has introduced many techniques to reduce leakage in different processor units (e.g., [6–10]). Powell et al., explored an integrated architectural and circuit level approach to reducing leakage energy dissipation in instruction caches [6]. Kaxiras et al. proposed an approach to reduce the L1 cache leakage energy by turning off the cache line not likely to be reused [11]. Bai et al optimized several components of on-chip caches to reduce gate leakage power [12]. Kao and Chandrakasan suggested dual-threshold voltage techniques for reducing standby power dissipation while still maintaining high performance in static and dynamic combinational logic blocks [7]. Johnson et al., modified circuits considering state dependence. They identified a low leakage state and inserted leakage control transistors only where needed [8]. Durate et al., studied and compared three leakage power reduction techniques: Input Vector Control, Body Bias Control and Power Supply Gating. They investigated their limits and benefits, in terms of the potential leakage reduction, performance penalty and area and power overhead [9]. Rele et al.,introduced an approach to combine compiler, instruction set, and microarchitecture support to turn off functional units that are idle for long periods of time for reducing static power dissipation by idle functional units using power gating [10]. Our work is different from previous work as it targets embedded processors. We show that there is a motivating opportunity in the embedded space to apply power gating. Moreover, we take advantage of embedded processor characteristics such as in-order execution to improve previously suggested gating techniques.

3 POWER GATING

Power dissipation in a CMOS circuit can be classified to dynamic and static. Dynamic power dissipation is the result of switching activity while static power dissipation is due to leakage current. Among all factors influencing the static power the subthreshold leakage is considered to be an important contributor. Subthreshold leakage current ($I_{leakage}$) flows from drain to source even when the transistor is off (see figure 1(a)). Static power dissipation can be computed using the following:

$$P_{static} = V_{cc}.I_{leakage} = V_{cc}.N.K_{design}.K_{tech}.10^{\frac{-V_T}{S_t}} \tag{1}$$

The parameters in equation 1 are divided to two categories: technology dependent and design dependent. V_{cc}, N and K_{design} are technology independent and may be varied independently targeting a specific design model. V_T is a technology dependent parameter. As the technology scales down, V_T decreases which results in an increase in static power.

We use power gating to block V_{cc} and reduce leakage power to zero. In figure 1(b) we present how power gating is achieved using a header transistor to block voltage supply from reaching a circuit unit. The power gate detection circuit decides when is the appropriate time to turn off the voltage supply. Once the sleep signal is generated, and after a transition period, the V_{cc} signal will be blocked from reaching the functional unit.

Applying power gating comes with timing overhead. To explain this in more detail in figure 2 we present transition states associated with power gating.

As presented, the power gating process includes three separate intervals. We refer to the first interval as the active to sleep transit period (ASP). ASP starts the moment we decide to power gate a unit and ends when the voltage supply is completely blocked. We refer to the second interval as the deep sleep period or DSP. This is the period where the functional unit is gated and therefore does not dissipate power. Power dissipation reduction depends on how often and for how

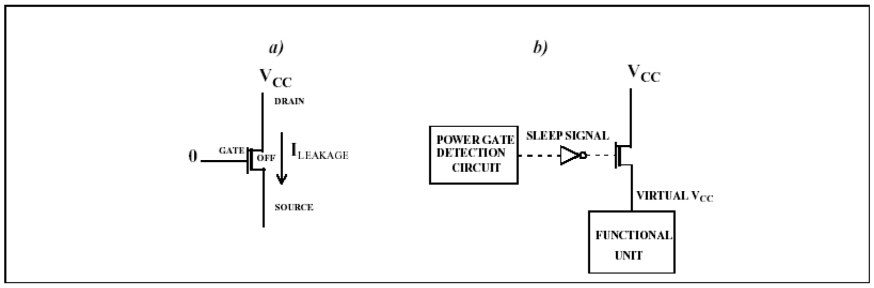

Fig. 1. a) Turned off transistor dissipating leakage power b) Schematic showing major blocks exploited in power gating.

long units stay in DSP. We have to wakeup a unit as soon as its idle period ends. For example, in the case of integer ALU, this is when an instruction requires the unit to execute. Turning on the voltage supply to wakeup a unit takes time. The third interval presented in figure 2 represents this timing overhead and is the time needed to reactivate a unit. We refer to this period as the sleep to active transition period (SAP).

While saving leakage power during ASP and SAP is possible, in this study we assume that power reduction benefits are only achievable when a unit is in DSP. As such we refer to ASP and SAP as timing overheads associated with power gating. Hu et. al, provide a detailed explanation of the three intervals [3].

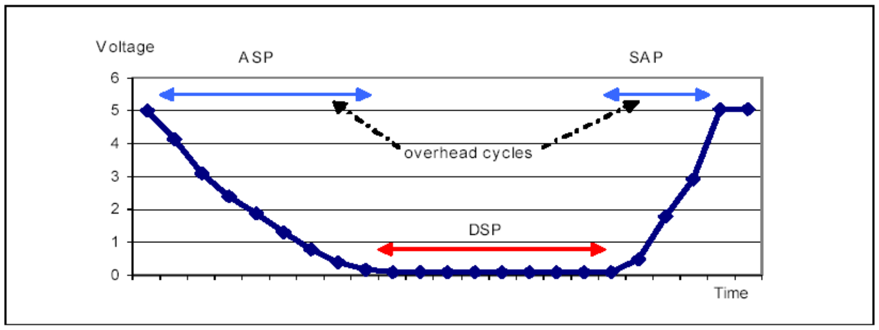

Fig. 2. Transition states in power gating.

4 MOTIVATION AND HEURISTICS

Through this study we report for a representative subset of MiBench benchmarks [13] and for a processor similar to that of Intels XScale processor (more on this later in section 5).

In figure 3 we present energy savings achievable by ideal (but not practical) power gating. We assume that the percentage of execution units idle cycles indicates maximum leakage power reduction possible by using power gating. We also assume that the timing overhead with power gating is zero. As a result the data presented in figure 3 serves as an upper bound for our leakage power savings. Bars from left to right report average savings for integer ALU, integer multiplier/divider, memory ports, floating point ALU and floating point multiplier/divider.

In figure 3, and as an indication of potential leakage power savings, we report how often each of the five units used in the Intels XScale are idle. On average, three of the units, i.e., integer multiplier/ divider, floating point ALU and floating point multiplier/divider are idle more than 95% of cycles. Average idle period is least for integer ALU (40%). We conclude from figure 3 that there is motivating opportunity in embedded processors to exploit idle times and to power gate execution units to reduce leakage power dissipation. However, identifying idle times early enough is a challenging problem. Moreover, reactivating the gated execution units soon enough is critical since stalling instruction execution could come with a performance penalty.

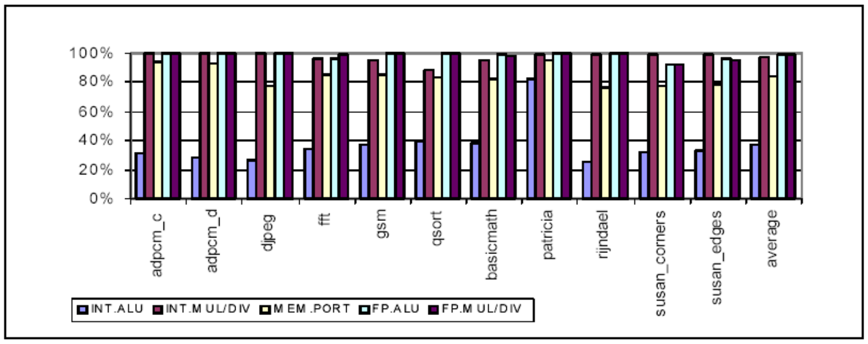

Fig. 3. Leakage power reduction achieved by ideal power gating.

As explained earlier time-based power gating monitors the state of each execution unit and turns it off after the number of consecutive idle cycles exceeds a pre-decided threshold. We refer to this threshold as the idle detect threshold (IDT). In the following sections we extend time-based power gating to reduce leakage power further.

4.1 Multiple IDTs (MIDT)

In figure 4 we report how changing the idle detect threshold or IDT impacts power gating. We assume that the active to sleep period is 3 cycles. We also assume that returning an execution unit from sleep to active takes 5 cycles [3].

In 4(a) bars from left to right report average percentage of cycles each execution unit is gated for the benchmarks studied here for IDT values 5, 10, 20, 50, 100 and 150.

In 4(b) we report performance cost for the benchmarks studied here for different IDT values. Average performance slowdown is 10.9%, 4.1%, 1.9%, 0.9%, 0.3%, 0.3%, for IDT values 5, 10, 20, 50, 100 and 150 respectively.

A closer look at figure 4 reveals that none of the IDT values provide acceptable results across all execution units and for all applications. Lower IDT values (i.e., 5, 10 and 20) provide high power savings but come with high performance cost. Higher IDT values (i.e., 50 and 100), on the other hand, maintain performance but can reduce power savings dramatically. This is particularly true for integer ALU and memory port.

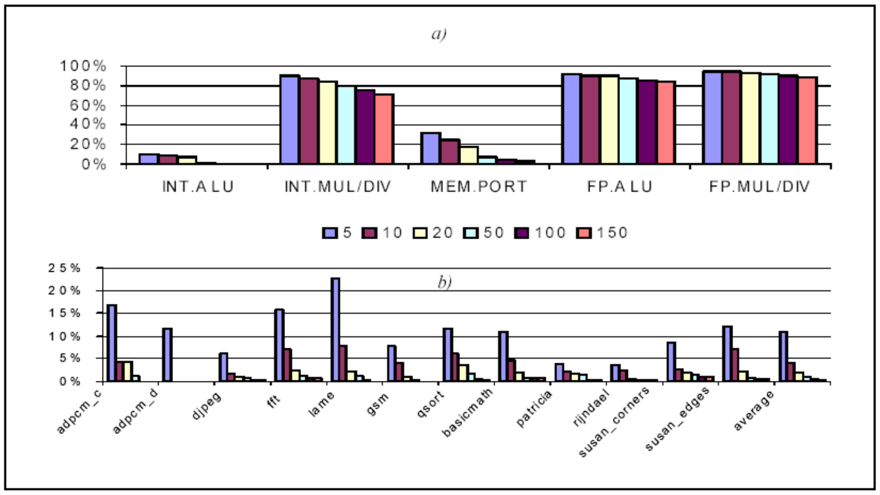

Fig. 4. a) Average leakage power savings achieved by power gating for different IDT values for ASP=5 and SAP=3. Higher is better. b) Performance cost associated with power gating for different IDT values for ASP=5 and SAP=3. Lower is better.

To provide better insight in figure 5 we report idle time distribution for each execution unit. As presented, idle time distribution is quite different from one execution unit to another. As such using a single IDT for all execution units is inefficient. To address this issue we use a different IDT for each execution unit. We refer to this method as multiple IDT or MIDT. To pick the right IDT for every execution unit we took into account many factors including how often the execution unit becomes idle and how long it stays idle. After testing many alternatives we picked IDT values 20, 80, 40, 100 and 140 for integer ALU, integer multiplier/divide, memory ports, floating point ALU and floating point multiplier/ divider respectively. Note that multiple IDT could be implemented easily by using programable registers.

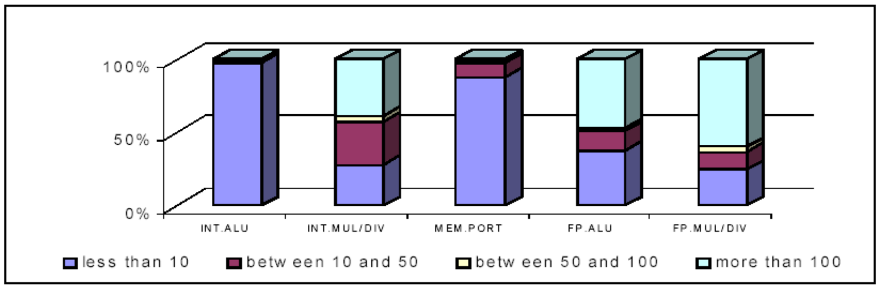

Fig. 5. Idle time distribution for different execution units.

4.2 Early Wakeup

In figure 6 we report how changes in SAP impact performance. Note that SAP is the time required to reactivate an execution unit by turning on the power supply. SAP depends on the circuit parameters and may change from one design to another. We assume that IDT and ASP are 20 and 5 respectively. Bars from left to right report for SAP values of 1, 2, 3 and 4 respectively. As expected the longer it takes to reactivate a gated execution unit the higher the performance penalty would be. Average performance cost is 0.5%, 1.2%, 1.9% and 2.7% for different SAP values.

We conclude from figure 6 that long wakeup periods can harm performance seriously. One way to reduce performance cost is to reactivate gated units as early as possible. To reactivate gated execution units sooner we suggest two methods:

First, we use control dependencies to wakeup execution units in advance. We refer to this method as the branch-aided wakeup or BAW technique.

Second, we use information available at the decode stage to wakeup the needed execution units at least one cycle earlier than when they become active in conventional power gating. We refer to this technique as the decode-aided wakeup or DAW.

Branch-Aided Wakeup (BAW) Note that embedded processors such as Intels XScale use inorder issue. As such once a branch instruction is issued, the

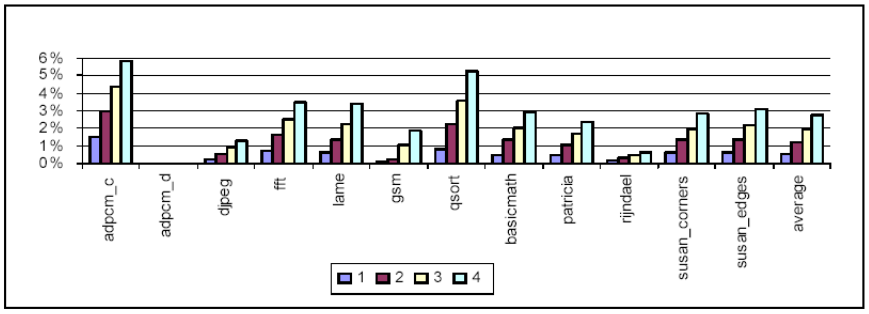

Fig. 6. Performance cost associated with power gating for different SAP values for IDT=20 and ASP=5. Lower is better.

following basic block should issue sequentially. To take advantage of inorder instruction issue we store information regarding whether integer ALU and memory ports are used inside a basic block. We limit the stored information to the these two execution units since our study shows that long wakeup periods for the two units impact performance more seriously compared to other execution units. Moreover,limiting the technique to the two units will reduce the overhead associated.

We use an 8-entry table to record the required information. The table is indexed using the branch instruction address associated with the basic block. Each entry stores two bits. The first bit records if any instruction within the basic block uses the integer ALU and the second bit records if any instruction uses the memory port.

At fetch, and in parallel with accessing the branch predictor, we probe the 8-entry table. We reactivate the execution units if the table indicates that they will be needed by the following basic block. We take no action if the execution units are already active. Note that possible misspeculations are not costly from the performance point of view since all they do is to reactivate an execution unit which will not be used.

We use a 3-bit register to store the index associated with the latest fetched branch. For example, when an issued instruction is dispatched to the integer ALU, we update the entry associated with the last branch fetched, i.e., we set the first bit in the entry to 1 indicating that the integer ALU will be used by the basic block. We use the 3-bit register to find and update the table entry associated with the last fetched branch.

The area overhead associated with this techniques is very small. We use a 3-bit register and an 8- entry table which contains eight 2-bit fields. The total area requirement is equivalent to 19 bits which is negligible.

Decode-Aided Wakeup (DAW) In this method we start the activation process of the gated execution units at decode and immediately after recognizing the opcode. Note that in conventional power gating execution units are activated when a ready to execute instruction is detected. DAW, on the other hand, uses the already available information at decode and starts reactivation at least one cycle before the instruction becomes ready. This in turn reduces the timing overhead associated with power gating.

5 METHODOLOGY AND RESULTS

In this section we report our analysis framework and simulation results. To evaluate our optimization techniques we report performance and leakage power reduction. We use a subset of MiBench benchmark suite [13] compiled for MIPS instruction set. In this work, similar to earlier studies [3], we assume that the percentage of cycles a functional unit stays in DSP indicates net leakage savings achieved by using power-gating.

Table 1. Configuration of the processor model

Issue Width	In-Order:2	Inst/Data TLB	32-entry,full-associative
Functional Units	1 I-ALU, 1 F-ALU,1 I-MUL/ DIV 1 F-MUL/DIV	L1 - Instruction/ Data Caches	32K, 32-way SA, 32-byte blocks, 1 cycle
BTB	128 entries	L2 Cache	None
Branch Predictor	Bimodal, 128 entries	Register Update Unit	8 entries
Main Memory	Infinite, 32 cycles	Load/Store queue	8 entries

For simulation purpose we used a modified version of simplescalar v3.0 toolset [14]. We modeled a single issue in-order embedded processor with an architecture similar to Intels XScale core. Table 1 shows the configuration we used.

In figure 7 we report how our optimizations impact performance and leakage power savings. We also report results for a combined technique where multiple IDT, BAW and DAW are applied simultaneously. We refer to the combined technique as CMB. For the sake of comparison we also include results achieved when all execution units use the same IDT. Bars from left to right report for IDT values 5, 10, 20, 50, 100, 150, multiple IDT (MIDT), BAW, DAW and CMB.

We limit our discussion to comparing the CMB method to methods where a single IDT is used across all execution units. Nonetheless, it is important to note that similar comparisons could be made for each of the three optimizations. Note that as explained earlier single IDT results in either high performance cost (for low IDTs) or low energy savings (for high IDTs). CMB, however, maintains performance for all benchmarks (see figure 7(a)). While average performance costs are 10.9% and 4.2% for IDT values 5 and 10, average performance cost is reduced to 0.3% for CMB.

Average leakage energy savings are 0.1% and 3% for integer ALU and memory ports for IDT values 100 and 150 respectively. For CMB, average savings for integer ALU and memory port are increased to 6.5% and 11% respectively (see figure 7(b)). Note that CMB maintains high energy savings for other execution units.

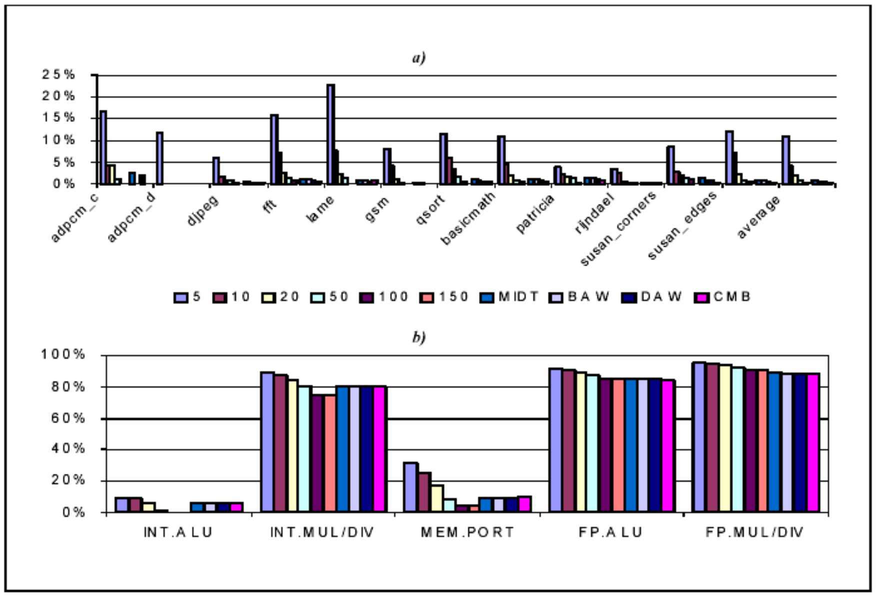

Fig. 7. a) Performance cost. b) Leakage power reduction for the methods discussed.

6 CONCLUSION

In this paper we analyzed how power gating could be exploited in embedded processors to reduce leakage power. We extended previous work by introducing three optimization techniques to reduce leakage power while maintaining performance. Our techniques used control dependency, instruction decode information and idle time distribution. We showed that it is possible to reduce leakage power while maintaining performance for an embedded processor similar to Intels Xscale and for a representative subset of MiBench benchmarks.

References

1. Borkar, S.: Design challenges of technology scaling. IEEE Micro **19** (1999) 23–29

2. Butts, J.A., Sohi, G.S.: A static power model for architects., In Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (December 2000)
3. Hu, Z., Buyuktosunoglu, A., Srinivasan, V., Zuyuban, V., Jacobson, H., Bose, P.: Microarchitectural techniques for power gating of execution units., In proceedings of ISLPED (2004)
4. Unsal, O.S., Koren, I.: System-level power-aware design techniques in real-time systems. Volume 91, NO. 7., In proceedings of the IEEE (July 2003)
5. Jejurikar, R., R., G.: Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems., In proceedings of ISLPED (2004)
6. Powell, M., Yang, S., Falsafi, B., Roy, K., Vijaykumar, T.: Gated-vdd: A circuit technique to reduce leakage in deepsubmicron cache memories., In proceedings of ISLPED (2000)
7. Kao, J., Chandrakasan, A.: Dual-threshold voltage techniques for low-power digital circuits. IEEE Journal of Solid State Circuits **35** (2000)
8. Johnson, M., Somasekhar, D., Cheiou, L., Roy, K.: Leakage control with efficient use of transistor stacks in single threshold cmos. IEEE Transactions on VLSI Systems **10** (2002)
9. Durate, D., Tsai, Y.F., Vijaykrishnan, N., Irwin, M.J.: Evaluating run-time techniques for leakage power reduction., ASPDAC (2002)
10. Rele, S., Pande, S., Önder, S., Gupta, R.: Optimizing static power dissipation by functional units in superscalar processors., In International Conference on Compiler Construction (2002)
11. Kaxiras, S., Hu, Z., Martonosi, M.: Cache decay: exploiting generational behavior to reduce cache leakage power., In proceedings of ISCA (2001)
12. Bai, R., Kim, N., Sylvester, D., Mudge, T.: Total leakage optimization strategies for multi-level caches., ACM Great Lakes Symposium on VLSI (2005)
13. Guthaus, M., Ringenberg, J., Ernst, D., Austin, T., Mudge, T., Brown, R.: Mibench: A free, commercially representative embedded benchmark suite, IEEE 4th Annual Workshop on Workload Characterization (WWC-4) (December 2001)
14. Burger, D., Austin, T.M., Bennett, S.: Evaluating Future Microprocessors: The SimpleScalar Tool Set.Technical Report CS-TR-96-1308, University of Wisconsin-Madison. (July 1996)