# Reducing Leakage in Value Predictors by using Decay Techniques

Juan M. Cebrián, Juan L. Aragón and José M. García

*Resumen*—**Energy-efficient microprocessor designs are currently one of main goals pursued by designers in both high performance and embedded processor domains. Recent processor implementations and research efforts have been focused on dynamic power consumption. However, as process technology advances toward deep submicron –below 90 nm.–, static power becomes a new challenge to address, especially for large on-chip array structures such as caches and prediction tables. Value prediction emerged in the recent past as a natural way of increasing performance by breaking data dependences. The more accurate the predictor is the more performance is obtained, at the expense of becoming an increasing source of power consumption and a thermal hot spot.**

**This paper proposes the design of low-leakage value predictors by applying decay techniques –already used in caches and branch predictors– in order to disable unused entries from the prediction tables. We explore decay strategies for the three most common value predictors (*Stride*, *FCM* and *DFCM*) studying the particular tradeoffs for these prediction structures, that exhibit different pattern access behavior than caches, in order to reduce VP leakage efficiently without compromising VP accuracy nor processor performance. Results show average leakage power reductions of 61%, 76% and 68% for the stride, *FCM* and *DFCM* value predictors of 20 KB respectively.**

*Palabras clave*—**Leakage, Value Prediction, Power and Energy Consumption, Decay.**

## I. Introduction

POWER dissipation and energy consumption are one of the major design concerns when facing the design of a new microprocessor in the high performance domain –server and desktop– and, more dramatically, in the embedded microprocessor domain, especially in the case of battery-operated devices.

There are two sources of power dissipation, *dynamic* power and *static* power (*leakage*). The dynamic power of an structure is the power dissipated each time that structure is accessed, due to the repeated capacitance charge and discharge on the outputs of transistor gates. On the other hand, static power is the power the structure dissipates regardless of activity, especially through *subthreshold* leakage currents and *gate* leakage currents [12][13] that flow even when the transistor is nominally off.

For several generations, leakage has been just a minimum part of the overall power consumption of microprocessors, and it was not considered as a major concern. However, as feature size shrinks to allow greater density and higher performance, supply voltage must be lowered in order to contain power consumption, since dynamic power is proportional to the square of supply voltage. But using smaller geometries (with very small threshold voltages) has the additional effect of increasing leakage loss exponentially, which leads to static power beginning to dominate the overall power consumption as process technology drops below 65 nm. [13][16].

In order to deal with this problem, it can be found in the literature several proposals both at circuit and architecture level for managing leakage energy. Many proposals have focused on reducing the leakage power by switching off unused portions of large array structures. Cache Decay [14] selectively turns individual data cache lines off if they have not been used in a long time, reducing leakage energy at the expense of loosing the content of the cache line. This *non-state preserving* technique has also been successfully applied to branch predictors and *BTB* structures [6][10].

On the other hand, Value Prediction has been proposed [4][5][8][9][15] as an effective way of improving superscalar processor performance by overcoming data dependences which are one of the major performance limitations in current superscalar processors. However, such prediction structures incur in additional dynamic and static power dissipation –despite the speedup provided– and, therefore, their use has not been widely spreaded. Furthermore, in the ultra-low power embedded domain, the use of VP techniques may be prohibitive in terms of both power and area.

In this paper we propose *Value Prediction Decay*, a mechanism able to dramatically reduce the leakage power of traditional value predictors with negligible impact on accuracy, especially for deep-submicron designs (below 90 nm.) by locating and disabling unused entries in the predictor and, therefore, making value prediction a power-performance efficient mechanism for low-power processor designs. It is important to note that value predictors show a significant amount of spatial and temporal locality and, unlike caches, a decayed prediction does not degrade performance as much as a decayed cache line.

The rest of the paper is organized as follows. Section II reviews some related works and provides an overview on value prediction. Section III analyzes the dynamic utilization of the prediction tables. The proposed *Value Prediction Decay* mechanism is described in Section IV. Section V shows the experimental methodology and the leakage savings obtained. Finally, Section VI summarizes the main conclusions of the work.

Dpto. Ingeniería y Tecnología de Computadores, Universidad de Murcia, 30071 Murcia (Spain) e-mail: {jcebrian,jlaragon,jmgarcia}@ ditec.um.es.

## II. Related Work

In order to reduce leakage power in processors, several techniques have been proposed both at the circuit level and at the architectural level. At the architectural level, many proposals have focused on reducing the leakage power by switching off unused portions of large array structures such as caches. These techniques have been categorized into *state-preserving* and *non-state preserving* [1][7][17].

Studies by Powell *et al.* [14] proposed *gated-$V_{DD}$* as a technique to limit static leakage power by banking and providing "sleep" transistors which dramatically reduce leakage current by gating off the supply voltage. This technique, known as *decay,* reduces the leakage power drastically but the cell's contents are lost, being necessary to apply it very carefully since the loose of information can deliver into an increase of the dynamic power to retrieve it again. Kaxiras *et al.* [11] successfully applied decay techniques to individual cache lines in order to reduce leakage in cache structures (67% of static power consumption can be saved with a minimal performance loss due to decay induced misses). This technique has also been applied to conditional branch predictors and BTB structures [6][10].

On the other hand, drowsy techniques try to reduce leakage without loosing the cell's information. Drowsy caches [2] use different supply voltages according to the state of each cache line. Those lines in drowsy mode use a low-voltage level, retaining the data, while requiring a high voltage level to access it again. Waking up from the drowsy state is similar to a pseudo-cache miss incurring in some additional penalty cycles (about 7 cycles). Of course, the leakage savings of this mechanism are lower than the decay ones, but the increase of dynamic power consumption due to the loose of information is also lower. Flautner *et al.* [2] showed that a drowsy cache putting to sleep all cache blocks periodically (every one Kcycles) achieves 54% leakage power savings with a negligible performance degradation of about 1%.

Li *et al.* [7] confronted the use of state and non-state preserving techniques in caches. The authors showed that for a fast L2 cache (5-8 cycles latency) decay techniques are superior in terms of both performance loss and energy savings to drowsy ones.

Finally, an alternative to traditional decay is to use quasi-static, four-transistor (4T) memory cells. 4T cells are approximately as fast as 6T SRAM cells, but do not have connections to the supply voltage ($V_{SS}$). Rather, the 4T cell is charged upon each access, whether read or write, and it slowly leaks the charge over time. When the charge in the cell has been depleted, the value stored is lost. In [10] it was proposed to apply decay techniques to branch predictors by using 4T cells. By doing this, some of the drawbacks of using gated-$V_{DD}$ transistors are eliminated, since an access to a 4T cell automatically reactivates the cell, whereas reactivating a 6T cell from the "sleep" mode is somewhat more complex, requiring extra hardware involved in gating the supply voltage.

### A. Value Prediction Overview

The *last value predictor* was introduced by Lipasti *et al.* [9]. This is the most basic prediction mechanism and, basically, it assumes that the next value produced by an instruction will be the same as the previous one.

A generalization of the last value predictor leads to the *stride value predictor* (STP). Introduced by Gabbay *et al.* [4], it uses the last value produced by an instruction plus a stride pattern. In a stride pattern, the difference between two consecutive values is always the same constant. The next predicted value is computed by adding the last value to the stride.

The *finite context method* value predictor (FCM), introduced by Sazeides *et al.* [15], uses the history of recent values, called the *context*, to determine the next value. This is implemented by using two-level prediction tables. The first level stores the context of the recent history of the instruction (VHT). The second level stores, for each possible context, the value which is most likely to follow it (VPT). The value is predicted by using the program counter to access the VHT table and, according to the context hash function, the VPT table is accessed to get the predicted value.

The *differential finite context method* value predictor (DFCM), introduced by Goeman *et al.* [5], joins the best of the two previous predictors in one structure. DFCM works like FCM (two-level prediction tables, VHT and VPT), but it stores the differences between the values instead of the values themselves, plus the last value of the instruction. This allows DFCM to capture stride patterns as if they where constant patterns, containing only one value. For a stride pattern (e.g., 0 1 2 3), the DFCM predictor will remember the last value 3 and the history of differences: 1 1 1. In this way, the DFCM can predict stride patterns by adding the last value to the stride associated to the context 1 1 1. For non-stride patterns, DFCM works just like the FCM predictor.

## III. Utilization Analysis of Value Prediction Structures

As previously seen, power dissipation of value prediction structures is divided into dynamic and static power. The dynamic component depends on the utilization of the value predictor. Values can be predicted at different levels, the most aggressive utilization predicts the output value for all instructions traversing the pipeline. Other approaches found in the literature restrict the use of the value predictor to just a fraction of instructions such as long-latency instructions; load instructions that miss in the L1 or L2 data cache; instructions that belong to a critical path; instructions that have been proven to provide the highest VP hit ratio; or just to predict the effective address of memory instructions to provide early memory disambiguation.

Therefore, restricting the VP utilization to just a fraction of selected instructions, effectively reduces the dynamic power component of this structure. However, the static power component is still present, as the VP structure leaks regardless of utilization with increasing leakage loss as process technology shrinks, as cited before. For this reason, this work is focused on reducing the static power component of the VP structure.

In [11] Kaxiras *et al.* showed that, very frequently, cache lines have an initial active period (known as *live time*) followed by a period of no utilization (known as *dead time*) before they are eventually evicted. They proposed to break the stream of references to a particular cache line into generations. Each generation lasts until the cache line is evicted and replaced by a new one. This generational behaviour also appears in the value predictor structure, although with some particularities: as value predictors are implemented as direct-mapped tables with no tags and allowing destructive interferences, in our proposal, a generation ends when the VP entry is accessed by an instruction with a different PC (see Figure 1). Its *live time* will be the period of accesses with the same PC and its *dead time* will be the period between the last access with an specific PC until an access with a new one.

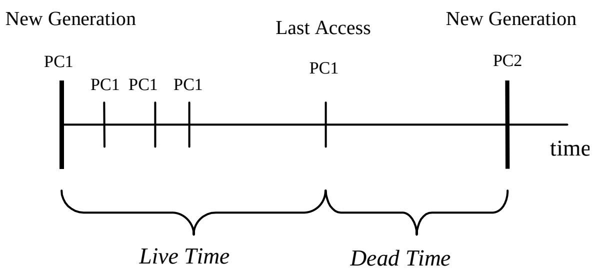

Figure 1. Generations in value predictor entries.

Our first evaluation analyzes the utilization of the value prediction tables, measuring the fraction of time each entry remains in the *dead* state, in order to determine if turning those VP entries off will result in a significant decrement of leakage power.

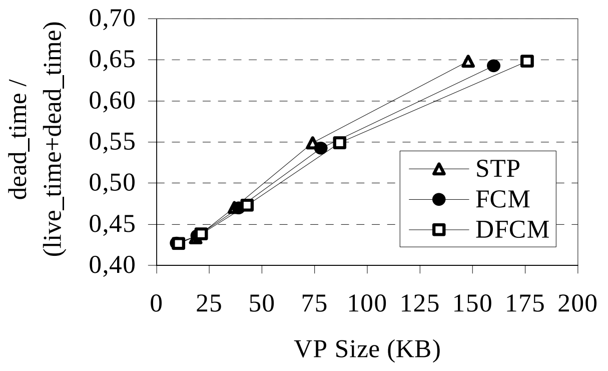

Figure 2. Fraction of time VP entries are in a *dead* state.

Figure 2 shows the average fraction of time each generation is in a *dead* state (i.e., the ratio *dead*/(*live*+*dead*)) for the whole SPECint2000 benchmark suite as a function of VP size (see Section V for details about simulation methodology and processor configuration). It can be observed that the three evaluated value predictors –stride, FCM y DFCM– present a similar utilization regardless of their size. For sizes around 20 KB, the average fraction of dead time is 43% and for predictor sizes around 40 KB the average fraction of time the entries spend in their *dead* state is 47%. Therefore, if we were able to take advantage of these *dead times* by detecting them and shutting the entries off, we could reduce the leakage power of the value predictor structure by one half on average.

It is important to note that this is not an upper bound on the leakage power savings that could be achieved by decaying VP entries. Long periods of inactive *live time* could be also detected to early shut the entry off in order to obtain further leakage savings, at the expense of reducing the VP accuracy.

## IV. *Value Prediction Decay* Mechanism

In order to apply decay techniques to the value predictor mechanism we need to detect those VP entries that have been unused for a significant period of time in order to switch them off. But in order to successfully apply decay techniques it is necessary to choose carefully the number of cycles we should wait before shutting an entry off in order to match generational changes.

The proposed *Value Prediction Decay* mechanism is time-based as in [11]. It tracks the accesses to each VP entry in order to detect if a particular entry is accessed very frequently or, otherwise, the entry has been unused for a long period of time, probably entering into a *dead* state. In order to measure the power-efficiency of our proposal, we will explore a wide range of *decay intervals* to precisely detect the dead states while at the same time not degrading the value predictor accuracy.

Note that, if the policy that decides when to turn a VP entry off uses too long decay intervals, the potential leakage savings will be reduced. Conversely, if the time-based policy uses too short decay intervals, the VP hit ratio will be degraded. In any case, a positive effect in *Value Prediction Decay* when compared to *Cache Decay* is that prematurely disabling a VP entry is not so harmful as disabling a cache line: loosing the content of the VP entry will most likely result in a value misprediction on the next access (the entry will be reset to an initial state) but this is exactly what it would have happened if we had a real generation change.

A power-efficient implementation of the time-based decay mechanism requires the use of a hierarchical counter composed of a global counter and two-bit saturated gray-code counters on each value predictor entry[1] as in [11]. Every time the global counter gets to zero, all the local counters will be incremented by one. On the other hand, an access to a VP entry results on a reset of its local counter. When a VP entry remains unused for a long time, its local counter will reach the upper limit eventually, and the corresponding entry will be shut off.

The length of the decay intervals are controlled by the period of the global counter. If we set the period of the global counter to a low value, the VP entries may be disabled prematurely and leakage will be reduced drastically, but so will the hit rate of the predictor. On the other hand, if we disable too late (large global counter periods), the leakage savings won't be as high as they could be.

The VP entries will be shut off by using *gated*-$V_{DD}$ transistors [14]. These "sleep" transistors are inserted between the ground (or supply) and the cells of each VP entry, which reduces the leakage in several orders of magnitude and can be considered negligible. An

[1] Using a hierarchical counter is more power-efficient since it allows accessing the local counters at a much coarser level. Accessing the local counters each cycle would be prohibitive because of the power overhead associated.

alternative to using *gated-$V_{DD}$* transistors consists of using quasi-static 4T transistors [10] in the VP array.

Regarding the power overhead associated to the proposed *Value Prediction Decay* mechanism, we are introducing two additional bits per VP entry plus the dynamic power consumption of updating these bits. That power has been measured to be about 2% of the dynamic power of the VP, which can be considered negligible. As for the static power overhead, all the two-bit local counters and the global counter have been included in the static power model associated to the proposed *Value Prediction Decay* mechanism.

## V. Experimental Results

### A. Simulation Methodology

To evaluate the power-performance efficiency of the proposed *Value Prediction Decay*, we have used the SPECint2000 benchmark suite. All benchmarks were compiled with maximum optimizations (-O4 -fast) by the Compaq Alpha compiler and were run using a modified version of HotLeakage power-performance simulator [18] that includes the static power model for the three evaluated value predictors (stride, FCM and DFCM) as well as the static power overhead of the proposed mechanism. Due to the large number of dynamic instructions in some benchmarks, we reduced the input data set while keeping a complete execution.

TABLE I. SPECint2000 benchmark characteristics.

| Benchmark | Input set | Total # simulated instr. (Mill.) | # skipped instr (Mill.) |
|---|---|---|---|
| bzip2 | input source 1 | 500 | 500 |
| crafty | test (modified) | 437 | - |
| eon | kajiya image | 454 | - |
| gap | test (modified) | 500 | 50 |
| gcc | test (modified) | 500 | 50 |
| gzip | input.log 1 | 500 | 50 |
| mcf | test | 259 | - |
| parser | test (modified) | 500 | 200 |
| twolf | test | 258 | - |
| vortex | test (modified) | 500 | 50 |
| vpr | test | 500 | 100 |

TABLE II. Configuration of the simulated Processor.

| Processor Core | |
|---|---|
| Process Technology: | 70 nanometers |
| Frequency: | 5600 Mhz |
| Instruction Window: | 128 RUU, 64 LSQ |
| Decode Width: | 8 inst/cycle |
| Issue Width: | 8 inst/cycle |
| Functional Units: | 8 Int Alu; 2 Int Mult |
| | 8 FP Alu; 2 FP Mult |
| | 2 Memports |
| Pipeline: | 22 stages (P4-like) |
| **Memory Hierarchy** | |
| L1 Icache: | 64KB, 2-way |
| L1 Dcache: | 64KB, 2-way |
| L2 cache: | 1MB, 4-way, unified |
| Memory latency: | 120 cycles |

Table I shows, for each particular benchmark, the input set, the total number of simulated instructions, and the number of forwarded instructions. Table II shows the configuration of the simulated architecture. The leakage related parameters have been taken from the provided alpha 21264 configuration file using a process technology of 70 nanometers.

### B. Value Prediction Decay Leakage Savings

This section presents a power-performance evaluation of the proposed *Value Prediction Decay* mechanism for the stride, FCM and DFCM value predictors as predictor size varies for several decay interval windows: 256, 1024, 8192 and 65536 cycles.

Figures 3 and 4 show the VP hit ratio and the leakage power savings for the stride value predictor (STP). As we can see in Figure 3, for decay windows below 8192 cycles, the hit ratio is reduced as expected, due to the premature data loss of deactivating entries early; the use of bigger decay windows provides hit ratios very close to the original predictor. Figure 4 shows that our proposal succeeds decreasing power consumption in a wide range of values, from 10% to 91% depending on the configuration. For a predictor of about 20 KB we obtain average leakage power savings of 46% for an 8 Kcycles decay window, with a VP accuracy degradation of just 0.1% and leakage savings of 61% with an accuracy degradation of only 1%. As expected, further leakage savings can be obtained as we increase VP size.

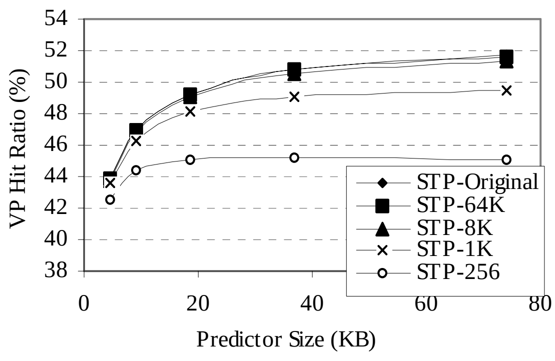

Figure 3. STP value predictor accuracy.

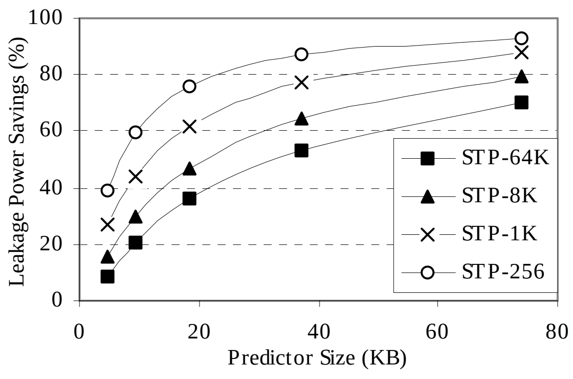

Figure 4. STP value predictor leakage power savings.

Regarding the FCM value predictor, as it is a two-level predictor (with the relevant and bigger part of the data stored in the second level table), we will be disabling both level tables. Figures 5 and 6 show the VP hit ratio and the leakage power savings for the FCM value predictor. Leakage power savings go from 0% to 91% depending on the configuration. For a predictor of about 20 KB we obtain average leakage power savings of 76% for a 256-cycle decay window, with a VP accuracy degradation of just 0.65%.

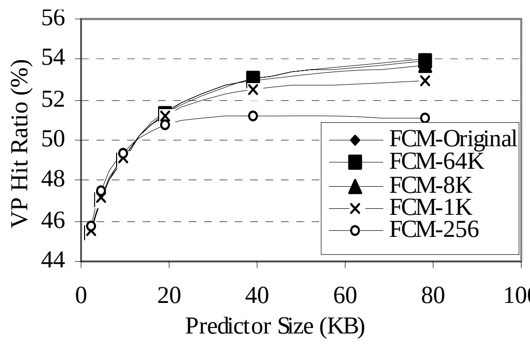

Figure 5. FCM value predictor accuracy.

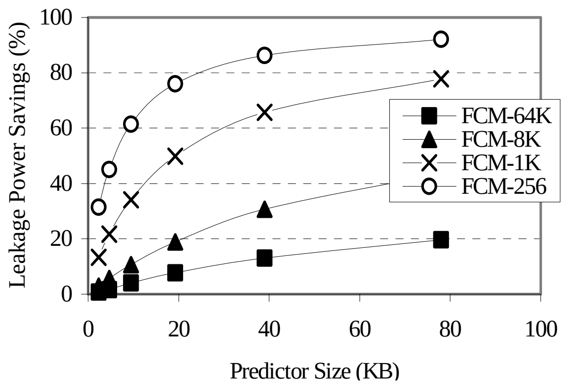

Figure 6. FCM value predictor leakage power savings.

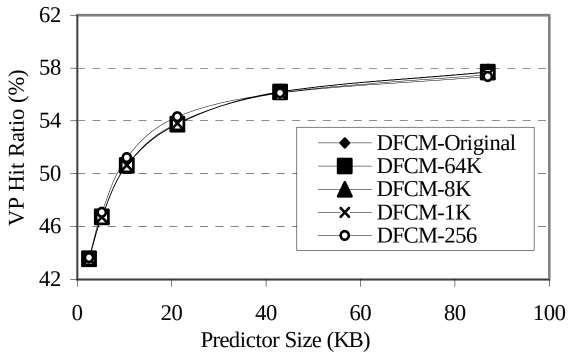

Figure 7. DFCM value predictor accuracy.

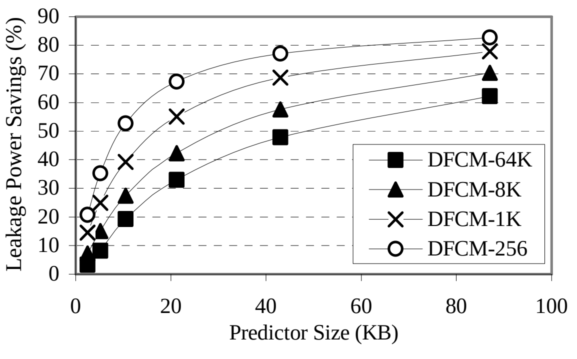

Figure 8. DFCM value predictor leakage power savings.

Regarding the DFCM value predictor, it experiences both the greater leakage power savings and the less VP hit ratio reduction. Figures 7 and 8 show the VP hit ratio and the leakage power savings for the DFCM value predictor. For a predictor size of about 20 KB, we obtain average leakage power savings of 68% for a 256-cycle decay window, with no reduction on the predictor accuracy.

As we can see in Figure 7, there is a positive side effect when shutting DFCM entries off due to the reduction of aliases and destructive interferences. It causes the predictor to maintain its accuracy even for small decay intervals. This effect was also reported in [6] for branch predictors. In the DFCM predictor case, resetting the entries to zero makes patterns between generations from different entries match the same second level entry, something that would not happen if patterns from the previous generation were still inside the predictor.

Finally, Figure 9 shows the leakage power savings breakdown for a predictor size of 20 KB and a decay interval of 1024 cycles. It can be observed that a significant amount of leakage savings are obtained when disabling VP entries during its live time period. As commented in Section III, there are many cases where even though an entry is live, the next access will be far in the future (more than 1024 cycles ahead in this experiment). In such cases, short decay intervals can obtain even further leakage savings by early disabling those entries. Figure 9 shows that, on average, half of the leakage power savings comes from disabling entries during their live time and the other half comes from disabling entries during a dead time. Note also that the three evaluated predictors obtain a very similar leakage savings breakdown since they all are indexed in the same way, i.e., using the instruction PC.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper we propose *Value Prediction Decay*, a mechanism able to dramatically reduce the leakage power of value predictors with negligible accuracy reduction, especially for deep-submicron microprocessor designs. Our proposal dynamically tracks the accesses to each value predictor entry in order to determine if the entry has been unused for a significant period of time, and in that case, it switches the entry off, avoiding leakage loss.

Experimental results have shown that both FCM and DFCM seem to be the most power-efficient predictors achieving average leakage power savings of 76% and 68%, respectively, for a predictor size of around 20 KB with negligible VP accuracy reduction when considering a decay interval window of 256 cycles. We have also shown that leakage power savings are not limited by only detecting *dead times,* since value predictors are structures that exhibit long periods of inactivity during an entry's *live time* which allows to early shut the entry off in order to obtain further leakage savings.

Finally, the use of low-power value prediction structures could make value prediction a power-performance efficient mechanism suitable for low-power processor designs.

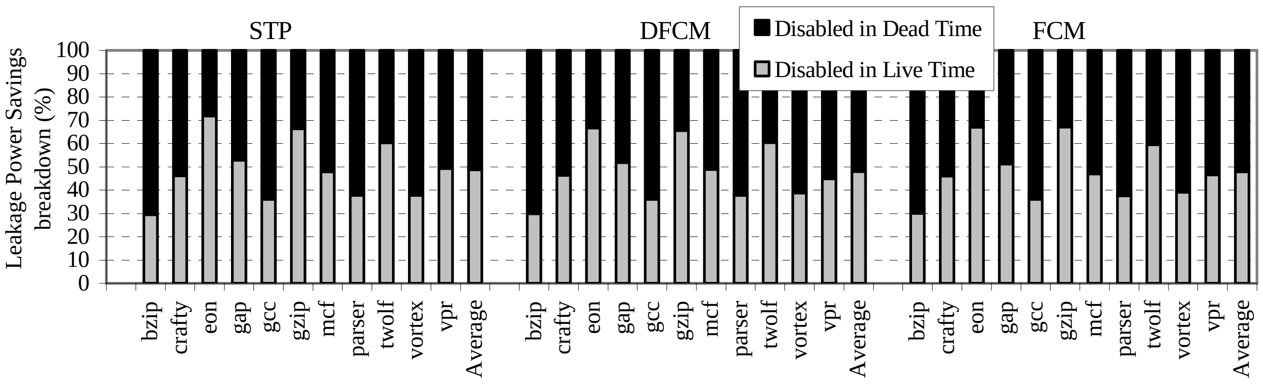

Figure 9. Leakage power savings breakdown for a 20 KB predictor with a decay interval of 1024 cycles.

## ACKNOWLEDGEMENTS

This work has been supported by the Ministry of Education and Science of Spain under grant TIC2003-08154-C06-03.

## REFERENCES

[1] J.A. Butts and G. Sohi. “A static power model for architects”. In *Proc. of the 33rd Int. Symp. on Microarchitecture*, 2000

[2] K. Flautner *et al*. “Drowsy Caches: Simple Techniques for Reducing Leakage Power”. In *Proc. of the 29th Int. Symp. on Computer Architecture*, 2002.

[3] M.J. Flynn and P. Hung. “Microprocessor Design Issues: Thoughts on the Road Ahead”. In *IEEE Micro*, vol. 25, no. 3, pp. 16-31, May/Jun, 2005.

[4] F. Gabbay and A. Mendelson. “Speculative execution based on value prediction”. Technical Report 1080, Technion – Israel Institute of Technology, 1997.

[5] B. Goeman, H. Vandierendonck and K. de Bosschere. “Differential FCM: Increasing Value Prediction Accuracy by Improving Table Usage Efficiency”. In *Proc. of the 7th Int. Symp. on High-Performance Computer Architecture*, 2001.

[6] Z. Hu *et al*. “Applying Decay Strategies to Branch Predictors for Leakage Energy Savings”. In *Int. Conf. on Computer Design*, Sep. 2002.

[7] Y. Li *et al*. “State-Preserving vs. Non-State-Preserving Leakage Control in Caches,” In *Proc. of the DATE Conference*, Feb. 2004.

[8] M. Lipasti and J. Shen. “Exceeding the dataflow limit via value prediction”. In *Proc. of the 29th Annual International Symposium on Microarchitecture,* Dec. 1996.

[9] M. Lipasti, C. Wilkerson and J. Shen. “Value locality and load value prediction”. In *Proc. of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1996.

[10] P. Juang *et al*. “Implementing Branch-Predictor Decay Using Quasi-Static Memory Cells”. In *ACM Transactions on Architecture and Code Optimization*, Vol. 1, No. 2, June 2004.

[11] S. Kaxiras, Z. Hu and M. Martonosi. “Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power”. In *Proc. of the 28th Int. Symp. on Computer Architecture*, 2001.

[12] A. Kesharvarzi. “Intrinsic iddq: Origins, reduction, and applications in deep sub-micron low-power CMOS IC’s”. In *Proc. of the IEEE International Test Conference*, 1997.

[13] N.S. Kim, T. Austin *et al*. "Leakage Current: Moore’s Law Meets Static Power". In *Proc. of IEEE Computer*, 2003.

[14] M.D. Powell *et al*. “Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories”. In *Proc. of the Int. Symp. on Low Power Electronics and Design*, 2000.

[15] Y. Sazeides and J.E. Smith. “The predictibility of data values”. In *Proc. of the 30th Annual International Symposium of Microarchitecture*, Dec 1997.

[16] Semiconductor Industry Association. “The international technology roadmap for semiconductors”. Available at http://public.itrs.net/Files/2001ITRS/Home.htm, 2001.

[17] S. Yang *et al*. “An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-Caches”. In *Proc. of the 7th Int. Symp. on High-Performance Computer Architecture*, 2001.

[18] Y. Zhang, D. Paritkh, K. Sankaranarayanan, K.Skadron and M. Stan. “HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects”. Technical report, Dept. of Computer Science, Univ. of Virginia, 2003.