Penelope[1]: The NBTI-Aware Processor

Jaume Abella, Xavier Vera, Antonio González
Intel Barcelona Research Center, Intel Labs - UPC
{jaumex.abella, xavier.vera, antonio.gonzalez}@intel.com

Abstract

Transistors consist of lower number of atoms with every technology generation. Such atoms may be displaced due to the stress caused by high temperature, frequency and current, leading to failures. NBTI (negative bias temperature instability) is one of the most important sources of failure affecting transistors. NBTI degrades PMOS transistors whenever the voltage at the gate is negative (logic input "0"). The main consequence is a reduction in the maximum operating frequency and an increase in the minimum supply voltage of storage structures to cope for the degradation. Many PMOS transistors affected by NBTI can be found in both combinational and storage blocks since they observe a "0" at their gates most of the time.

This paper proposes and evaluates the design of Penelope, an NBTI-aware processor. We propose (i) generic strategies to mitigate degradation in both combinational and storage blocks, (ii) specific techniques to protect individual blocks by applying the global strategies, and (iii) a metric to assess the benefits of reduced degradation and the overheads in performance and power.

1. Introduction

Reliability is a key issue in microprocessor design because a given performance must be provided for a given time period (product's lifetime). While technology evolution drives to smaller devices (transistors and wires), the supply voltage does not scale at the same pace, leading to higher current densities (which also produce higher temperatures). The increased current density and temperature accelerate device degradation, and thus, shorten the lifetime of the product. Moreover, the size of the chip does not scale, which implies that in every technology generation there is a larger number of such highly vulnerable devices.

The increasing electric field and temperature make negative bias temperature instability (NBTI) [4][17] emerge as a threat for future technologies. NBTI affects PMOS transistors when negative voltage is applied at the gate (logic input "0"), causing an increase in the threshold voltage, and hence, a lower speed of the transistor.

Degradation due to NBTI has an impact in the power and performance of circuits. The cycle time is impacted because circuits become slower when they are degraded (if degradation is very high they may even fail). The conventional solution to address the decreased speed of circuits is guardbanding, which consists in reducing the operating frequency to account for the degradation that circuits may experience during their lifetime. Large guardbands of 10-20% in the cycle time may be required [1]. Similarly, storage structures observe an increase of their minimum voltage required to keep their contents (Vmin) [3]. This issue is also addressed with guardbanding, which consists in increasing the nominal Vmin by a given voltage to account for the degradation that circuits may experience. For instance, 10% Vmin increase may be required to tolerate 10% threshold voltage (V_{TH}) shifts [1]. Higher Vmin produces higher power because the supply voltage cannot be decreased as much as desired for power savings.

NBTI depends on circuit parameters and data patterns. On one hand, NBTI depends on the geometry of transistors, operating voltage and frequency, and temperature. Such factors affect power, area and delay of circuits, so changing them may have a negative impact in the whole processor design. On the other hand, data patterns are highly biased for some bits causing some PMOS transistors to degrade faster, which leads to larger guardbands. This work focuses on mitigating NBTI by reducing the amount of time that PMOS transistors observe a "0" at their gates (zero-signal probability). This paper presents and evaluates Penelope, an NBTI-aware processor. The main contributions of this paper are:

- Strategies to mitigate NBTI for combinational and memory-like blocks.
- A global approach to protect the whole processor by adapting previous strategies to each concrete block.
- A metric to compare the cost and benefit of different solutions based on how much they mitigate NBTI and how much overhead they incur.

[1] In the Greek mythology, Penelope spent 20 years waiting for her husband Odysseus to return from the Trojan War. In order to refuse marriage proposals during that time, she devised several tricks, one of which was pretending to weave a shroud and claiming she would choose one suitor when she had finished. Every night for three years she undid part of the shroud.

1072-4451/07 $25.00 © 2007 IEEE
DOI 10.1109/MICRO.2007.11

By reducing the degradation due to NBTI the guardband of the different blocks can be reduced to increase their performance. Guardband reductions of 10X have been reported (i.e., from 10-20% to only 1-2%) [1]. Similarly, mitigating NBTI in memory-like structures provides energy savings due to a lower Vmin. Some experiments show V_{TH} shifts one order of magnitude lower for non-biased data patterns (i.e., from 10% V_{TH} to only 1%) [1].

The rest of the paper is organized as follows. The remaining of this section is devoted to illustrate the high bias of data flowing through the pipeline that motivates this work. Section 2 introduces the physics of NBTI. Global strategies to mitigate NBTI are presented in Section 3. Section 4 presents the evaluation framework, specific mechanisms for an adder, register files, schedulers and caches, and a metric to compare the different NBTI-aware techniques. Finally, Section 5 draws the main conclusions of this work.

1.1 Motivation

We have studied data from real world programs (more details about such programs are provided later) and evaluated how biased data are for different structures of the processor such as adders, register files, schedulers and caches.

Adders have a wide variety of PMOS transistors observing different inputs. However, some of them usually observe a "0" at their gate most of the time; for instance, those PMOS transistors whose gate is connected to the carry in of the adder have a high bias because such carry in is typically "0". Our experiments show that such bit is "0" more than 90% of the time consistently across our working set. Therefore, PMOS transistors whose gates are connected to the carry in degrade very quickly.

Patterns for register files and data caches correspond to those of the data being fetched, operated and stored back again. Our experiments for integer and FP data show that zero-signal probability for some bits is very high. For instance, zero-signal probability for the integer register file ranges between 65% and 90% for all bits. Similarly, some fields of the scheduler have almost 100% zero-signal probability.

Overall, it is very common observing highly biased inputs for some PMOS transistors, which will degrade very quickly. We can conclude that it is crucial reducing the maximum amount of time that any PMOS transistor observes a "0" at their gate to mitigate NBTI degradation, which enables shorter guardbands (higher performance) and lower Vmin (lower power)..

2. NBTI Source of Failure

NBTI has emerged as a significant issue for reliability of future technologies. This section illustrates the main mechanisms involved in NBTI degradation of transistors. First, we illustrate the physics behind NBTI. Then, we introduce the self-healing effect of NBTI that allows recovering from degradation.

2.1 NBTI Physics

NBTI breaks progressively silicon-hydrogen bonds at the silicon/oxide interface whenever a negative voltage is applied at the gate of PMOS transistors [13][17]. During negative voltage at the gate Si-H breakages generate more interface traps (N_{IT}), which capture electrons flowing from source to drain, leading to an increase of the threshold voltage (V_{TH}). Therefore, transistors become slower and may not fit timing requirements, especially for those circuits that rely on a given relation between the delay of the pull-up and the pull-down.

Similarly to NBTI, PBTI (positive voltage temperature instability) affects NMOS transistors. While the physics of NBTI on PMOS transistors and PBTI on NMOS ones is basically the same, the degradation is significantly different. State-of-the-art experiments [15] have shown that PBTI degradation in NMOS transistors is practically negligible when compared to NBTI in PMOS transistors. Different parameters have an effect on NBTI:

- *Geometry*. While increasing the length of PMOS transistors increases the degradation due to NBTI [16][17], increasing the width decreases such degradation [7]. Length is typically set to the minimum possible and only the width is changed to fit timing, power and area constraints. As a rule of thumb we can consider that NBTI can be mitigated by using wider transistors [19], but it has an impact in delay, area and power.
- *Voltage*. The higher the operating voltage, the higher the NBTI-degradation is [13][16]. Therefore, lower operating voltage is desired to mitigate NBTI.
- *Frequency*. Some studies show that NBTI is independent of the operating frequency [6], whereas other works show a weak dependence [1][4] where higher frequencies produce lower NBTI degradation. Either way, the relation between frequency and NBTI degradation is low.
- *Temperature*. Research on the area consistently shows that NBTI degradation is higher for higher operating temperatures [8][13].
- *Zero-signal probability*. Different studies have reported a strong dependence between the amount of NBTI degradation and the zero-signal probability [1][4]. The larger the amount of time with input set to "0", the higher the degradation due to NBTI is.

Geometry of transistors as well as the operating voltage and frequency are set considering not only NBTI but power, area and delay of circuits, so changing them may have a negative impact in the whole processor design. Additionally, controlling the processor temperature has similar implications as the previously mentioned parameters. Thus, the focus of this work is mitigating NBTI by reducing the zero-signal probability of PMOS transistors.

2.2 NBTI Self-Healing Effect

The higher the time a PMOS observes a negative voltage at the gate, the farther the hydrogen atoms are dragged. Conversely, when its gate is set to "1" not only it does not degrade but it enjoys from the self-healing effect of NBTI [1][4][6][14]. During such periods, those hydrogen atoms that were dragged away from the interface of the gate are dragged back to the interface filling the holes that they created. The closer to the interface hydrogen atoms are, the faster they are dragged back to the interface. Hence, whenever the input at the gate of a PMOS transistor switches, hydrogen atoms are dragged back and forth providing a variable behavior of the transistor. NBTI degradation (self-healing effect) happens in such a way that the number of N_{IT} created (recovered) in the interface during a given period of time, Δt, is a fraction of the current number of Si-H bonds (H atoms). This behavior is illustrated in Figure 1, where periods of degradation and self-healing alternate (this picture has been taken from [4]).

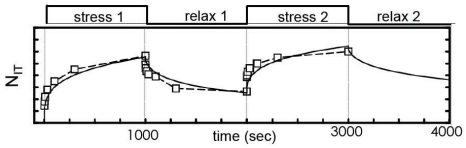

Figure 1. N_{IT} at the gate interface of a PMOS during alternate periods of stress (gate set to "0") and relax (gate set to "1") [4]. Note that V_{TH} shift depends directly on N_{IT}

As shown in the figure, degradation speed decreases as the number of Si-H bonds decreases (and hence, the N_{IT} increases). Recovery happens just the other way around: the higher the number of N_{IT}, the faster the recovery is. Full recovery could only happen after infinite relaxation time. As it can be seen, during relaxation periods degradation does not freeze but decreases, which implies that keeping the gate of PMOS transistors set to "1" extends their lifetime significantly. NBTI is not well understood yet, so only chip testing can report real data on the guardband reduction (or lifetime increase) achieved by reducing the zero-signal probability of PMOS transistors. Nevertheless, some estimates [1] show that guardbands in the cycle time can be reduced by 10X or that lifetime can be increased by a factor of at least 4X [4]. Similarly, there is a lack of data reporting the magnitude of benefits in terms of Vmin that can be achieved if NBTI is mitigated, but V_{TH} shift reductions of 10X have been reported [1].

3. Strategies to Mitigate NBTI

State-of-the-art solutions to NBTI can be considered to remove some of the guardbands:

- Memory-like blocks may operate in inverted mode during half of the time as proposed in [10], which reduces zero-signal probability down to 50%, and hence guardbands can be reduced by 10X [1][4][10]. The cost of such a technique comes from the extra XNOR gates required to invert/deinvert data with the *invert bit* (global signal indicating the current mode), which has an impact in cycle time. Note that inverting is not a suitable solution for combinational blocks because inverted and non-inverted inputs may stress the same PMOS transistors.

Since such solution has significant cost in terms of performance and does not work for combinational blocks, we propose a set of solutions for the different types of structures of processors. Our solutions mitigate NBTI by reducing the zero-signal probability of PMOS transistors without using extra resources, and thus the cost in terms of hardware, performance and TDP is negligible.

3.1 Strategy for Combinational Blocks

Combinational circuits may exhibit different degradation levels in each PMOS transistor because different inputs for the circuit can lead to different inputs at the gate of PMOS transistors. In particular, it may happen that some PMOS transistors degrade practically 100% of the time because they have a "0" at their gates most of the time, whereas some others may hardly degrade because they have a "1" at their gates. Each individual combinational circuit may exhibit different relations between the degradation of their PMOS transistors. An example is shown in Figure 2. We can observe that the PMOS transistor of the inverter will observe D at its gate. D depends on A, B and C. If it is the case that C is "1" most of the time, D will be "1" most of the time, but if all inputs are "0" most of the time, D will be very biased towards "0", and therefore, the PMOS transistor of the inverter will degrade significantly. In general, combinational circuits will degrade more or less depending on their inputs.

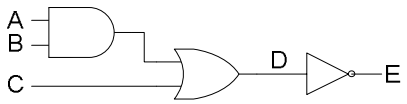

Figure 2. Example of a combinational circuit

As shown in Section 1.1, data may be very biased for combinational blocks. Based on the observation that many combinational blocks are idle a significant fraction of time, we propose using *special inputs* alternatively during idle periods. Note that any given input would always degrade the same transistors, but by alternating several inputs that degrade different PMOS transistors the maximum degradation of any PMOS is reduced with practically no cost.

Several issues must be addressed to implement this technique for a given combinational block:

- First, we must analyze how often the block is idle so we can set special values in its input latches. If the block is idle most of the time (e.g., integer and FP

ALUs), there is room to set special inputs most of the time and degradation is kept low. Conversely, if the block is busy most of the time we can set special inputs during the idle periods and resize those PMOS transistors that are expected to make the block fail before the target lifetime has elapsed, which has a cost in delay, area and power.

- The other main issue is how to choose the inputs to use during idle periods. Based on the knowledge of the circuit we can infer which inputs are most likely to evenly distribute PMOS degradation. Otherwise, we can generate a small set of inputs, identify which PMOS transistors are degraded for each one of them, and choose those inputs that degrade different PMOS transistors to be used alternatively (e.g., in a round-robin fashion). We have observed that with few inputs we can reduce the maximum degradation of any PMOS transistor in the block. Other algorithms to choose the inputs are part of our future work.

Regarding the implementation of the mechanism, the selected inputs need to be hardwired and written into the input latches of the corresponding block when it is idle. Scan ports of latches may be used for that purpose. A simple implementation sets one of such inputs in each idle period in a round-robin fashion. Although idle periods may have different lengths, in the long run all the low-degrading inputs will be used the same amount of time.

3.2 Strategy for Memory-like Blocks

Memory-like structures have a special characteristic. Bit cells consist of two inverters arranged in a ring-manner. Hence, there is always one of the inverters with negative voltage (logic input "0") at its gate, which implies that its PMOS transistor degrades. The best case degradation happens when the value at the output of each inverter is "0" 50% of the time, which means that both PMOS transistors degrade the same. Otherwise, one of such PMOS transistors degrades faster and the memory cell fails earlier. As explained in Section 1.1, it is quite common observing some bits highly biased towards "0".

Statistically, holding 50% of the time values inverted would produce 50% degradation for each PMOS in the bit cell [1][4]. However, operating in inverted mode 50% of the time may be expensive in terms of delay because a XNOR gate must be introduced in the read/write data paths to invert/deinvert data when operating in inverted mode [12]. Such extra delay may pay off for some slow structures (e.g., 2nd level caches), but may harm performance for some fast structures (e.g., register files, schedulers, 1st level caches, etc.).

We propose mechanisms to write *special values* in empty entries so that on average each bit cell stores "0" and "1" 50% of the time each. The different situations that may arise for a block or its different fields are as follows:

I. Entries are available more than 50% of the time on average (e.g., 1st level caches). In this case special values would be inverted values and they will be written when needed to keep 50% of the entries inverted on average. The effect would be the same as operating 50% of the time in inverted mode. Writing actual inverted values would require reading actual values, inverting them and writing them back. Sampling is an efficient solution to avoid the read operation. Regular values can be sampled and inverted periodically, and used to update those entries that must be inverted. Sampling produces near-optimal balancing in the long run. Our mechanism uses a special register for each structure, which is referred to as *RINV*, to store inverted sampled values. *RINV* is updated periodically with the inversion of any value being stored in the block. For instance, we can update *RINV* with the value flowing through a given write port of the block every one million cycles.

II. There are less than 50% of the entries available on average but no bit stores either "0" or "1" more than 50% of the time (overall time). That means that by writing the proper value during idle periods perfect balancing can be achieved without harming performance. For instance, if a given bit cell is busy 75% of the time and holds a "0" 67% of the time, it means that 50% of the time it holds a "0", 25% a "1" and 25% it is idle. Therefore, we can store a "1" during idle time for perfect balancing.

III. There are less than 50% of the entries available on average and at least one of the bits stores either "0" or "1" more than 50% of the overall time. In this case, whatever we write in such bit during idle periods perfect balancing is unfeasible. Therefore, guardband savings will be lower than in the case of perfect balancing. Alternatively, we can resize those bit cells, but such solution has some cost in terms of power and area.

IV. The entries are always busy. In this case nothing can be done because there are not idle periods.

V. The contents of the entries are self-balanced. For instance, if values stored are uniformly distributed or completely random values, the bias of each bit cell will be the ideal one (50%) in the long run.

In order to reduce the hardware overhead of write operations for inverted values, existing ports can be used when available in such a way that extra write ports are not required. In those cases when there is no write port available and updates are delayed one or two cycles, the impact on NBTI degradation is negligible because entries in different blocks remain either inverted or non-inverted for tens, thousands or even millions of cycles depending on the block.

Techniques to decide what to write and when for different types of memory-like structures may change depending on the characteristics of such structures. Memory-like structures can be classified into two categories depending on the way that their entries are deallocated: cache-like and explicitly managed structures. The following subsections describe both categories.

3.2.1 Cache-like Blocks. Entries in cache-like structures (e.g., caches, branch predictor, etc.) are evicted when an available entry is required. Based on the observation that most of the cache contents correspond to useless data (they will be evicted before being reused [2][9]), we propose to keep a fraction of the cache contents, including both data and tags, invalidated and with inverted values so that the degradation of the PMOS transistors is balanced.

Next we describe (i) the granularity at which the cache contents can be invalidated and overwritten, (ii) the fraction of the cache that stores special values and (iii) some implementation issues.

Granularity. The mechanism based on invalidating and inverting (we refer to it as inversion) can be applied at different granularities:

- *Set*. A given number of cache sets can be chosen for inversion (typically half of them) and the cache capacity is effectively halved, so there is some performance loss. The actual cache sets inverted are selected in a round-robin fashion at coarse time periods to minimize the extra cache misses.
- *Way*. Similarly, we can choose a given number of ways for inversion. The actual cache ways inverted are selected in a round-robin fashion. The cache works as if it had lower associativity and smaller size, so some performance loss is introduced.
- *Line*. Individual cache lines from different sets or ways can be chosen for inversion. It can be implemented by keeping a given ratio of cache lines inverted (and invalid). When an inverted cache line is refilled with valid data, a different valid cache line is inverted and invalidated to keep the ratio of cache lines inverted constant. To select the cache line to be inverted, we can use the information provided by the replacement policy (LRU, pseudoLRU, ...) and pick those cache lines that will be replaced earlier (LRU position). This approach is likely to have a minimal performance penalty considering that most of the cache access hits occur in the most recently used (MRU) position of cache sets (e.g., our simulations for a 32KB 8-way DL0 cache show that 90% of the hits occur in the MRU position, 7% in the MRU+1 position, and 3% in the remaining 6 positions). One possible implementation would use a counter (*INVCOUNT*) that tracks the number of inverted cache lines in the whole cache. Whenever *INVCOUNT* is below a given threshold (*INVTHRESHOLD*) and there is a write port available, a valid cache line from a random set is invalidated and inverted as explained above. Then, *INVCOUNT* is incremented. If there is no valid cache line in the selected set or there is not any available write port, *INVCOUNT* is not updated, and therefore, another try will be done in the future because *INVCOUNT* will remain below *INVTHRESHOLD*. Note that the valid/state bits indicate whether the cache line is valid and non-inverted, or invalid and inverted.

Fraction of Invalid Cache Contents. The fraction of the cache contents that are kept invalid and inverted can be chosen depending on the amount of NBTI-recovery that we want to achieve. For perfect balancing we would need 50% of the cache contents inverted on average.

The given fraction of the cache contents to be inverted (K) is a parameter of the proposed mechanisms, and can be either set a priori (*fixed*) or dynamically adjusted (*dynamic*). Each alternative has advantages and drawbacks:

- *Fixed*. Using a fixed invert ratio requires a simpler implementation, but may harm performance for those programs that make an effective use of all or most cache space. For perfect balancing we would choose K=50%.
- *Dynamic*. Using an invert ratio that dynamically changes can further improve performance while achieving close to perfect balancing. The idea is to select low K values for programs that use most of the cache and high K values for programs using a small fraction of the cache space.

Implementation Issues. To implement a dynamic invert ratio we need a mechanism to detect whether inverting and invalidating some cache contents impacts performance of a program. We have considered that the current program is run for some instructions to measure the performance impact that the inversion would have without actually performing it. Depending on whether the performance loss is below or above a given threshold, the mechanism is activated or deactivated respectively. This action must be repeated periodically to decide whether the mechanism is activated or deactivated during the next period. Our simulations show that the induced extra miss rate is a good performance indicator. Obtaining such miss rate is done by adding a bit per cache line that indicates whether cache lines would have been inverted if the mechanism was activated. Whenever a hit happens in such cache lines, it is counted as an induced extra miss. After the test step we decide which value of K to use.

3.2.2 Explicitly Managed Blocks. The main difference between explicitly managed and cache-like blocks lies on the fact that an entry can be inverted (and invalidated) when needed in a cache-like block, whereas entries in explicitly managed blocks can be used to store inverted (or special) values only when they have been released. Different situations may arise depending on their occupancy and the contents of the bit cells during busy periods as described before. Each situation requires a different strategy.

We will make use of the *RINV* register to update idle entries. For structures with multiple fields, each one is treated as if it was an independent structure, and hence, independent *RINV* registers and strategies are used for each field. Figure 3 describes the casuistic to choose the technique to use. The different techniques to be used work as follows:

- *ALL1 (0)*: the contents of *RINV* are always set to 1 (0). This technique is used in situation III (section 3.2).
- *ALL1-K% (0)*: the contents of *RINV* are set to 1 (0) *K%* of the time, and the rest of the time *RINV* is set to 0 (1). Note that *ALL1 (0)* is a special case of *ALL1-K% (0)* when *K*=100%. This technique is used in situation II (section 3.2).
- *ISV*: the contents of *RINV* are updated with inverted sampled values (*ISV*), but the entries in the block are updated only 50% of the overall time. To measure how long entries hold inverted or non-inverted contents we can use timestamps. Whenever an entry has hold non-inverted contents longer than inverted ones, such entry is updated with inverted contents. The update may happen at release time. Statistically, all entries will spend the same time inverted, and thus, tracking all entries or any entry gives the same results. Thus, we sample a single entry to decide when to write inverted contents. Such entry can be a fixed one, or one chosen by random selection, round-robin, etc. In our case we choose a fixed entry for the sake of simplicity. This technique is used in situation I (section 3.2).

```
IF (occupancy > 50%) THEN
    IF (occupancy x bias to 0 > 50%) THEN
        use ALL1
    ELSE IF (occupancy x bias to 1 > 50%) THEN
        use ALL0
    ELSE IF (bias to 0 > bias to 1) THEN
        use ALL1-K%
    ELSE
        use ALL0-K%
    ENDIF
ELSE
    use ISV
ENDIF
```

Figure 3. Casuistic to choose the proper technique for a given field

3.3 Strategy for Latches

Although latches are memory-like blocks because they consist of bit cells, they are a special case because we cannot set inverted or special values easily. Latches feed other blocks and we may need to set some values to mitigate NBTI in such blocks, which may not provide perfect balancing for the bit cells of latches.

Fortunately, transistors in latches are usually quite large because they have large fan-outs and do not have sense amplifiers to accelerate their reading. Therefore, their lifetime can be long enough even if their contents are highly biased. If it is the case that lifetime of some latches is not long enough and large guardbands are required, mechanisms to mitigate NBTI in latches must be used. Such mechanisms should trade between the proper inputs to mitigate NBTI in the blocks they feed and the proper inputs to mitigate NBTI in the latches themselves.

4. Penelope: The NBTI-Aware Processor

This section presents case studies of the strategies described in Section 3, the evaluation framework and a new metric to measure the cost and benefit of any NBTI-aware mechanism. Finally, a global view of the whole processor is provided.

The case studies considered for the Penelope processor are a combinational block (Ladner-Fischer adder), an explicitly managed block with large idle time (register file), an explicitly managed block with short idle time (scheduler), and two cache-like blocks (first level data cache (DL0) and data TLB (DTLB)).

4.1 Evaluation Framework

Results provided for register files, schedulers and caches have been collected from an IA32 trace-driven Intel® production simulator. Our workload consists of 531 traces of 10 million consecutive IA32 instructions each, which were obtained from different programs presented in Table 1. The processor configuration resembles the Intel® Core™ Microarchitecture, although our techniques can be used in any kind of processor.

Aging simulations for the adder have been performed with an Hspice-like Intel® production simulator for aging at electrical level using 65nm technology. Inputs for the adder have been sampled from the traces in Table 1. Idle time for the adder has been obtained with the same IA32 trace-driven simulator used for the rest of experiments assuming that there is an adder in each integer and address generation port.

Table 1. Workloads

Benchmark suite	# traces	Description
Encoder	62	Audio/video encoding
SpecFP2000	41	Floating-point specs
SpecINT2000	33	Integer specs
Kernels	53	VectorAdd, FIRs
Multimedia	85	WMedia, photoshop
Office	75	Excel, Word, Powerpoint
Productivity	45	Internet contents creation
Server	55	TPC-C
Workstation	49	CAD, rendering
SPEC2006	33	Specs

4.2 NBTI Metric

Several factors must be considered to decide whether a solution for NBTI is worth or not. *Delay* (or performance) is a key metric. Delay is the product of two factors: (i) the number of cycles of execution and (ii) the cycle time. While energy is especially important in the portable market segment, *TDP* is a key metric in all market segments. TDP is measured as the maximum amount of power that the cooling system is required to dissipate. Any technique requiring a higher TDP implies a modification of the processor design or more expensive cooling solutions. Any NBTI-aware technique requiring extra *area* has an impact either in performance or in TDP. For

the sake of simplicity, we assume that area impacts linearly TDP although other transformations of area overhead into delay could be considered instead. Finally, any technique aimed to mitigate NBTI provides some benefit in terms of *NBTI guardband* reduction. We will report benefits of guardband reduction in the cycle time, so this factor will impact directly the delay.

All factors are combined in one metric (see equation (1)) that we use to compare different techniques. Similarly to PD^3 (ED^2) [5], which weights delay and power (energy) in high-performance processors, delay is cubed in our metric. We state without proof that the best techniques to mitigate NBTI are those with lowest *NBTIefficiency* in equation (1). Although absolute values can be used for all the parameters, we will use relative values in the remaining of the paper.

$$NBTIefficiency = (Delay \cdot (1 + NBTIguardband))^3 \cdot TDP \quad (1)$$

Equation (1) can be used for any block very easily. However, obtaining the different parameters for the whole processor may be a bit trickier. We show in equations (2), (3) and (4) how delay, TDP and NBTI guardband are obtained for a processor. The delay of the whole processor is the product of the number of cycles per instruction (CPI) and the cycle time. While the cycle time is the maximum cycle time imposed by any block, the CPI produced by the different blocks cannot be combined directly and requires full simulation of all mechanisms together to consider the cross-impact between different mechanisms. TDP is the accumulation of the TDP of each block. Finally, the NBTI guardband of the processor is the maximum guardband required by any block because we assume that all paths of the different blocks have been adjusted to fit the cycle time to save power.

$$Delay_{processor} = CPI \cdot \underset{i=1}{\overset{Numblocks}{MAX}} CycleTime_i \quad (2)$$

$$TDP_{processor} = \sum_{i=1}^{Numblocks} TDP_i \quad (3)$$

$$NBTIguardband_{processor} = \underset{i=1}{\overset{Numblocks}{MAX}} NBTIguarband_i \quad (4)$$

In order to illustrate the new metric we evaluate the baseline solution to mitigate NBTI presented in Section 3: inverting data periodically. We also consider the case where we pay the whole guardband (we assume 20% guardband in the cycle time [1]).

- The baseline case pays the whole 20% delay guardband to tolerate NBTI. Our metric provides the following result:

$$NBTIefficiency = (1 \cdot (1 + 0.2))^3 \cdot 1 = 1.73$$

- If the block is a memory-like structure we can consider a design that operates in inverted mode half of the time. Such design requires introducing XNOR gates in all data-paths as explained in Section 3. The overhead in area and TDP is negligible, but there is some impact in the cycle time. For instance, we can consider that XNOR gates have the delay of 1 FO4 and the cycle time is 10 FO4. Then, the impact in delay is 10%. In this example we can assume that by inverting the guardband is reduced by 10X [1]. Overall, the efficiency of such a solution would be as follows:

$$NBTIefficiency = (1.1 \cdot (1 + 0.02))^3 \cdot 1 = 1.41$$

Inverting would be the most efficient solution for memory-like blocks, whereas paying the whole guardband would be the only solution for combinational blocks. We can observe that there is significant margin for improvement by further reducing delay, TDP and NBTI guardband overheads.

4.3 Case Study for Combinational Blocks: Ladner-Fischer Adder

This section validates our strategy to mitigate NBTI in combinational blocks. We have implemented a 32-bit Ladner-Fischer adder [11], whose layout has been generated for 65nm technology. Ladner-Fischer adder is a high-performance adder that speedups the addition at the expense of some hardware cost.

Accordingly with the strategy described in Section 3.1, we have studied the utilization of the adders for our 531 traces and found out that (i) if additions are allocated to adders with priorities, the utilization of the adders ranges between 11% and 30%, but (ii) if additions are distributed uniformly across adders, the utilization of adders is 21%.

The second step consists in choosing the proper inputs (synthetic inputs) to use during idle periods. Inputs during idle periods are referred to as *InputA*, *InputB* and *CarryIn*. Whenever we indicate that *InputA* or *InputB* are 0 (1), it means that all their bits are 0 (1). The inputs we have chosen are the eight combinations given by setting *InputA*, *InputB* and *CarryIn* either to 0 or 1. These synthetic inputs have been chosen because they are very likely to propagate either "0" or "1" to all PMOS transistors. Besides, some of these inputs stress all carry propagation circuits whereas some others do not. Other algorithms to choose the proper inputs are part of our future work.

Results for the actual input data as well as for each one of the eight synthetic inputs have been collected with the aging electrical simulator. Actual inputs have been sampled from our 531 traces (inputs remain unchanged during idle periods). As expected, some PMOS transistors are degraded most of the time for actual input data. Similarly, some PMOS transistors are degraded all the time for each of the synthetic inputs. Fortunately, different inputs degrade different transistors, so we have combined all pairs of synthetic inputs to identify the pair that requires the shortest guardband. Combination has been performed in a round-robin fashion. Results for each one of the combinations of synthetic inputs are shown in Figure 4. Inputs <*InputA*, *InputB*, *CarryIn*> have been numbered from 1 to 8 in ascending order (input 1 corresponds to <0,0,0>, input 2 <0,0,1>, and so on). Note that by combining two different inputs in a round-robin fashion the zero-signal probability for any transistor is 0%, 50% or 100%. As we can see, the best combination

corresponds to inputs (1) and (8), thus, <0,0,0> and <1,1,1>. The round-robin combination of such inputs ensures that narrow PMOS transistors have 0% or 50% zero-signal probability, and only few wide PMOS have 100% zero-signal probability. Fortunately, such PMOS do not suffer from NBTI significantly [19] (our simulator shows that wide PMOS with 100% zero-signal probability degrade less than narrow PMOS with 50% probability). Other input pairs require resizing at least some transistors to ensure that large guardbands are not required.

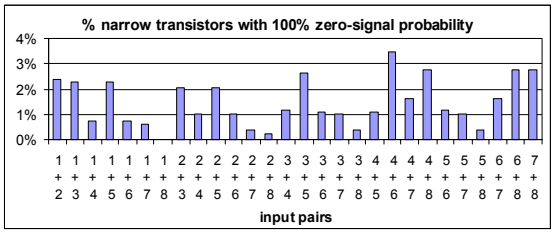

Figure 4. Narrow transistors with 100% zero-signal probability w.r.t. the total number of transistors

Finally, we have obtained the degradation of the adder for the three different scenarios where actual inputs are used during 11%, 21% and 30% of the time respectively, and the input pairs 1+8 are used the rest of the time. As explained in Section 3.1, for the sake of simplicity we can set one of such inputs in each idle period in a round-robin fashion. Results are depicted in Figure 5 in terms of guardband required. Note that without our technique the guardband required is 20% whereas a 50% zero-signal probability reduces such guardband to 2% (10X reduction [1]). Guardband can be reduced from 20% to 5.8% without any cost if additions are uniformly distributed across adders (21% utilization), whereas it is reduced to 7.4% if additions are allocated to adders with priorities. Note that by alternating the selected pair of inputs during idle periods, latches hold similar amounts of time opposite values, which is good to mitigate NBTI in such latches accordingly with the observations in section 3.3.

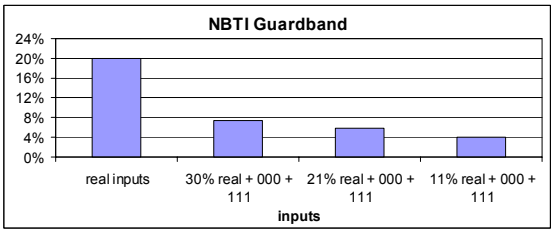

Figure 5. Guardband requirements for different inputs and utilization of the adders

If we measure the efficiency of our solution using equation (1), we observe that the overhead in terms of area and TDP to store the two input sets used during idle periods is negligible. Some extra activity (and thus, power) is caused in the combinational block when injecting synthetic inputs, but it happens only when the block is idle, and thus, TDP is not increased. The benefits in terms of NBTI guardband are significant even for the worst-case usage of any adder (30% of the time). Hence, the *NBTIefficiency* of our solution is 1.24, which is much better than that of the baseline (1.73). Note that inverting periodically is not suitable for combinational blocks.

$$NBTIefficiency_{roundrobin-inputs} = (1 \cdot (1 + 0.074))^3 \cdot 1 = 1.24$$

4.4 Case Study for Explicitly Managed Blocks with Large Idle Time: Register File

This section presents the case study for the register file, which is an explicitly managed block whose entries are idle most of the time (see Section 3.2.2). Figure 6 (*baseline*) shows the bit bias for the integer and FP registers. The Y-axis shows the bias towards "0". It can be seen that the worst-case for any bit shows a bias of 89.9% for integer data and 84.2% for FP data.

On average, 54% (69%) of the time integer registers (FP registers) are free (time between release and the next write operation), so accordingly with the casuistic detailed in Figure 3, we must use the *ISV* technique because they are free more than 50% of the time.

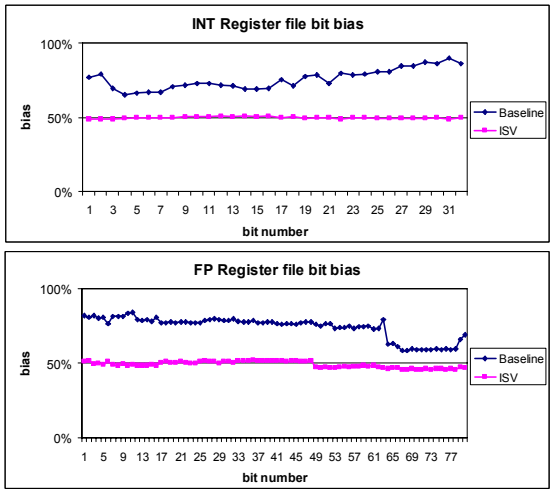

Figure 6. Balancing of bit cells contents for the different bits of the integer and FP register files. Y-axis shows the bias towards "0"

Registers are updated with *RINV* (see *ISV* mechanism, Section 3.2.2) when they are released and there is an available write port. Any update that cannot be done when the register is released because of lack of idle ports is discarded. Available ports are found 92% (86%) of the times for integer (FP) register files. Thus, discarding updates happens very rarely, so its impact in NBTI degradation is negligible.

Figure 6 (*ISV*) shows that near-optimal balancing is achieved with our technique. The worst-case degradation is reduced from 89.9% (39.9% from the optimal) to 48.5% (1.5% from the optimal) for the integer register file. For

the FP register file, degradation reduces from 84.2% (34.2% from the optimal) to 45.5% (4.5% from the optimal). Note that FP results are slightly worse than integer ones because integer traces start the simulation with an empty non-inverted FP register file and hardly use FP registers. In the real case, the worst-case bias will be much closer to 50% because integer programs will find a variety of inverted and non-inverted values in the FP register file.

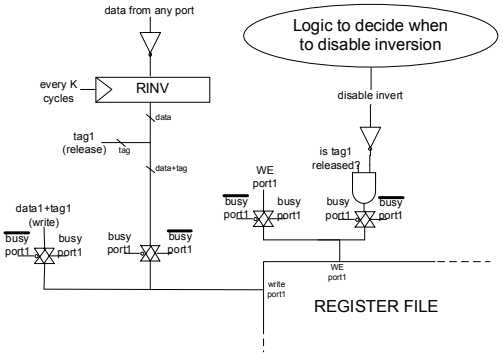

Figure 7. Design of the NBTI-aware register file

Our approach is extremely efficient because TDP remains practically unchanged since we add a single register per register file (*RINV*) and timestamps for a single register. Roughly speaking this is below 1% overhead for 128-entry highly-ported register files. Inverted values are written through actual write ports, so TDP is not increased. Delay is not impacted because neither the number of ports nor the critical paths are changed with respect to the baseline (see Figure 7). Finally, degradation is reduced significantly because bit bias reduces from 89.9% (84.2%) to 48.5% (45.5%) for the Integer (FP) register file. We use equation (1) to evaluate our proposal and the scheme where data is inverted periodically. Although we neglect it, such a solution would need some extra circuitry to read actual values, invert (deinvert) them and write them back when changing to inverted (non-inverted) mode. We use 3.6% guardband for our proposal, which corresponds to the FP register file bias (the worst one), whereas minimum guardband (2%) is assumed for the periodic inversion. In such a scheme TDP is hardly impacted and delay may grow around 10% (i.e. from 10 FO4 to 11 FO4). By inverting registers at release the overhead is much lower than by inverting the whole register file periodically (1.12 for our mechanism vs. 1.41 for periodic inversion as shown in Section 4.2). Furthermore, inverting at release does not need the circuitry to change the current mode.

$$NBTIefficiency_{invert-at-release} = (1 \cdot (1 + 0.036))^3 \cdot 1.01 = 1.12$$

4.5 Case Study for Explicitly Managed Blocks with Short Idle Time: Scheduler

Schedulers are complex structures to protect because they have a large number of fields, each one of them exhibiting different usage and data patterns. The description of the different fields is provided in Table 2. Activity patterns for the scheduler show significant imbalance because some of the bits are "0" (or "1") most of the time, producing much higher degradation in one of the PMOS transistors of such bit cells. Figure 8 (*baseline*) shows the value balancing for all the bits of the scheduler in the same order as in Table 2 but the opcode ones. Opcode bits are not shown because they depend strongly on the implementation, but by smartly encoding the opcodes of the uops, large imbalances can be avoided (IA32 instructions are split into microoperations also known as uops). In the figure the Y-axis shows the fraction of time that bits store "0". It can be seen that the worst-case for any bit shows almost a 100% bias for some *flags*, *shift* bits and *latency* bits.

The occupancy of the scheduler entries is 63%, although some fields (*SRC1 data*, *SRC2 data* and *immediate*) are available 70-75% of the time on average because they remain unused beyond the allocation or are not used at all for some instructions. Thus, based on the usage of each field and its bias, we apply techniques in Figure 3. Note that there are write ports available most of the time (on average 77% of the ports from allocate are available) so the very most of the updates of entries with *RINV* contents will be performed.

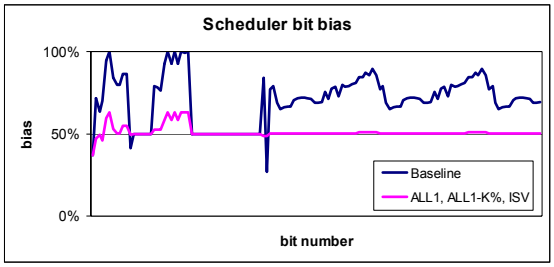

Figure 8. Balancing of bit cells contents for the different bits of the scheduler. Y-axis shows the bias towards "0"

Table 2. Description of the fields of the scheduler

Field	Bits	Description
Valid	1	Slot is valid
Latency	5	Latency of the uop
Port	5	Port for issue (loads and stores are not in the scheduler)
Taken	1	The branch is taken
MOB id	6	Memory Order Buffer identifier
tos	3	Top of stack position for FPs
Flags	6	Flags for the uop
shift1	1	Source 1 must be shifted (AH, BH, CH and DH)
shift2	1	Source 2 must be shifted (AH, BH, CH and DH)
DST tag	7	Destination register
SRC1 tag, SRC2 tag	7 each	Source 1 and source 2 registers
ready1, ready2	1 each	Source 1 and source 2 are ready for issue
SRC1 data, SRC2 data	32 each	Source 1 and source 2 data for data capture schedulers
Immediate	16	Immediate data field
Opcode	12	Opcode for the uop. Not shown in Figure 8

For the sake of fairness, selection of *K* for each field has been done based on the profiling information obtained from 100 random traces out of the 531 ones available. Then, such information is used for the remaining 431 traces used in our evaluations. *K* is computed as the value that would give us ideal balancing for the 100 traces used for profiling purposes. The classification of fields is as follows:

- *ALL1* fields: *latency* (bits 4 and 5), *port*, *flags*, *shift1* and *shift2*.
- *ALL1-K%* fields: *latency* bit 1 (*K*=95%), *latency* bit 2 (*K*=75%), *latency* bit 3 (*K*=95%), *taken* (K=50%), *tos* (*K*=50%), *ready1* (*K*=60%) and *ready2* (*K*=60%). Note that *ready1* and *ready2* use the same value for *K* because we assume that both source operands can be used alternatively to hold the first operand. Otherwise, the first operand usage would be higher and values for *K* would change, although our technique would work normally.
- *ISV* fields: *SRC1 data*, *SRC2 data* and *immediate*. Again, we assume that source operands 1 and 2 can be used alternatively to hold the first operand. If this is not possible, then *SRC1 data* and *SRC2 data* would need independent timestamps to decide when they must be updated with inverted contents. Sampled values for the corresponding fields of *RINV* can be taken from the register file when read or from bypasses for *SRC1 data* and *SRC2 data*, whereas *immediate* values are taken directly from the instruction.
- Nothing must be done to repair register tags (*DST tag*, *SRC1 tag* and *SRC2 tag*) as well as the *MOB id* because their activity is self-balanced because register file entries and MOB slots are used evenly.
- Nothing can be done for the *valid* bit because its contents are always useful, so we cannot update their contents with NBTI repairing data at any time.

Figure 8 shows the balancing for all the bits of the scheduler but the opcode ones when our set of techniques is used. Regarding the opcode, by choosing properly the encoding of the different uops we can avoid huge imbalance in all bits of such opcode and any of our techniques can be used to achieve near-optimal balancing. It can be seen in the plot that only those bits with very high bias in the baseline show still some bias after using our techniques. Those bits correspond to the ones where *ALL1* is used and the *valid* bit, which cannot be protected. The worst-case bias decreases from 100% to 63.2% (13.2% from the optimal solution).

RINV has fewer bits than a scheduler slot because it does not hold self-balanced fields (*DST tag*, *SRC1 tag* and *SRC2 tag*). Its fields are set accordingly with the previous description of the technique used for each field. This means that *ALL1* fields are set always to "1", *ALL1-K%* fields are set to "1" *K%* of the time and to "0" the rest of the time, and *ISV* fields are set to inverted sampled values always. Fields that do not need to be balanced are not written in the slots when they are released. *RINV* contents for *ISV* fields must be updated periodically (i.e., every some thousands or millions of cycles) to provide a good balancing in the scheduler.

Bias towards "0" is reduced from up to 100% to 50% approximately for most of the bits. The remaining bits (10% of the total bits) have an imbalance of up to 63% and must be resized to ensure the same guardband as we would have with perfect balancing. Since such resizing has a cost in power, area and delay, we use the guardband required for 63% bias (6.7% guardband). Our techniques have low overhead in terms of area and TDP because *RINV* has almost the same number of bits as a single slot of the scheduler, but it may be smaller because it has neither CAM cells nor as many ports as the scheduler. Some small counters can be used to implement *ALL1-K%* mechanism (4 small counters of up to 5 bits each for the different *K* values: 50%, 60%, 75% and 95%) and timestamps for *ISV* (2 timestamps of 10 bits each suffice for *SRC1 data* and *SRC2 data* fields, which share the same timestamp, and for *immediate* field). Overall, *RINV*, the counters and timestamps may take less than 2% of the scheduler area (less than 2 entries size in terms of number of bits, but smaller bit cells than the 32 entries of the scheduler), so 2% is a pessimistic TDP overhead. Similarly to the previous structures, inverted values are written through available write ports, and therefore, TDP is not increased due to port requirements. On the other hand, inverting periodically has a delay overhead around 10% as shown before. Recalling equation (1) we can observe that our set of techniques is more efficient (1.24 *NBTIefficiency*) than inverting for such a critical component like the scheduler (1.41 *NBTIefficiency*).

$$NBTIefficiency_{all1,all1-K\%,isv-K\%} = (1 \cdot (1 + 0.067))^3 \cdot 1.02 = 1.24$$

4.6 Case Study for Cache-like Blocks: DL0 and DTLB

This subsection presents the performance evaluation of our strategy for cache-like structures when applied to the first level data cache (DL0) and the data TLB (DTLB). In order to validate and illustrate the effect in performance of the proposed mechanism (see Section 3.2.1), different possible schemes have been evaluated:

- *SetFixed50%*. 50% consecutive sets are invalid and inverted at any time. The cache effectively operates as if it had half the size.
- *LineFixed50%*. 50% of the cache lines are invalid and inverted at any time. Whenever an inverted (and invalid) cache line becomes valid, the set of the cache line to be inverted is selected randomly.
- *LineDynamic60%*. 60% of the cache lines are inverted at any time. The program is run for some time to warm up the cache (200K cycles for the DL0 and DTLB), then we measure the number of misses that our mechanism would introduce if activated during some time (other 200K cycles for the DL0 and DTLB), and if the number of misses that the mechanism would

cause is higher than a threshold (for the DL0, 2% for 32KB, 3% for 16KB and 4% for 8KB; for the DTLB 0.5% for 128 entries, 1% for 64 entries and 2% for 32 entries) the mechanism is deactivated. Such test is done periodically (in our experiments every 10M cycles). Our results show that on average the number of cache lines inverted is slightly above the desired 50%. If the target is keeping the invert ratio at 50%, we can track the amount of time that the mechanism is active and disable it for some time if the current invert ratio is well above 50%.

In order to evaluate the performance impact of our proposal six DL0 cache configurations have been evaluated (8KB, 16KB and 32KB caches for 4-way and 8-way set-associative), as well as three DTLB configurations (128, 64 and 32 entries, all of them 8-way set-associative). Table 3 summarizes the average performance loss for the different implementations of our mechanism. Results show that *LineDynamic60%* achieves the 50% invert ratio with the lowest performance degradation. Furthermore, the performance of fewer programs is impacted, because the dynamic scheme allows disabling the inversion for those programs that fully utilize the cache. For instance, the fraction of programs that lose more than 5% (10%) performance for the 16KB 8-way DL0 is 7.0% (2.8%) for *SetFixed50%*, 7.2% (2.5%) for *LineFixed50%*, and only 4.4% (1.1%) for *LineDynamic60%*.

Table 3. Average performance loss for the different mechanisms

		SetFixed50%	LineFixed50%	LineDynamic60%
DL0 8-way	32KB	0.75%	0.53%	0.45%
	16KB	1.30%	1.14%	0.69%
	8KB	1.60%	1.60%	0.96%
DL0 4-way	32KB	0.83%	0.67%	0.45%
	16KB	1.29%	1.50%	0.78%
	8KB	1.73%	2.31%	1.02%
DTLB 8-way	128 ent.	0.32%	0.34%	0.14%
	64 ent.	0.55%	0.47%	0.32%
	32 ent.	1.31%	1.18%	0.97%

Similarly to the technique for register files, our proposals for DL0 and DTLB reduce the bias towards "0" from 90% to roughly 50%. We use equation (1) to evaluate *LineFixed50%* scheme for the 32KB 8-way DL0 and the simple solution where the whole contents can be inverted periodically. For the periodic inversion we ignore the overhead of flushing the whole cache when changing from non-inverted (inverted) mode to inverted (non-inverted) mode, which is against our technique. In such a scheme the impact in performance would be around 10% (i.e. from 10 FO4 to 11 FO4). Another alternative would be increasing the latency (number of cycles) of the cache, but that might imply modifying some other parts of the pipeline and increasing the pressure in the scheduler due to hit-speculated instructions. Moreover, there would be some performance loss due to the extra DL0 latency. Regarding our scheme we only need an extra cache line (the 32KB DL0 cache has 512 of them) and a counter (*INVCOUNT*) tracking the number of inverted cache lines. Overall, the hardware overhead is below 1% that we will include into the TDP cost. As we can see in the equations, our scheme is more efficient (1.09 *NBTIefficiency*) than inverting periodically (1.41 *NBTIefficiency*).

$$NBTIefficiency_{linefixed\,50\%} = (1.0053 \cdot (1 + 0.02))^3 \cdot 1.01 = 1.09$$

4.7 Summary

Once our techniques for different blocks in the processor have been described and evaluated, we present an overall view of what kind of technique is more suitable for each component depending on their benefit and overheads, and the *NBTIefficiency* for the Penelope processor. The alternatives analyzed are our custom techniques (Penelope processor), as well as the alternative solution presented in section 3 (inverting periodically).

Table 4 presents a description of the trends for the different parameters (coverage, NBTI guardband, delay, TDP and NBTI efficiency) for each mechanism.

Table 4. Summary of the characteristics of the different alternatives to mitigate NBTI

	Invert periodically	Penelope processor
Coverage	Only memory-like blocks	All
NBTI guardband	Very low	Very low
Delay	Some impact	None or small impact
TDP	Negligible	Negligible or small impact
NBTI efficiency (global view)	Only for memory-like blocks where delay impact is low like 2nd level caches	Suitable for any block with low cost

The Penelope processor covers any kind of structure. NBTI guardband is reduced for all structures with negligible hardware cost. Only some cache-like structures may require some extra hardware overhead (although low) to mitigate some performance loss. In general, the performance loss introduced for few of the structures is very low. Therefore, our proposals are very suitable for any kind of structure in the processor.

Measuring the exact cost in terms of delay, TDP and NBTI guardband for the processor requires designing and fabricating the whole processor for each one of the alternatives, which is out of the scope of this paper. However, we illustrate how it must be done for the five blocks analyzed in this paper: the adder, the register file, the scheduler, the DL0 and the DTLB. For the sake of this example we assume that each one of the five blocks has the same weight in terms of TDP. Results in terms of delay, TDP and NBTI guardband for each component have been presented in the corresponding sections. The only result missing is the combined CPI for the DL0 and the DTLB. The combined normalized CPI is 1.007 when *LineFixed50%* is used for both blocks simultaneously. As we can see, only the mechanism for the DL0 and the DTLB impacts the CPI, whereas none of the mechanisms impacts the cycle time. The impact in TDP of the different

mechanisms is low. Finally, the guardband of the different blocks is also combined. Note that the highest guardband is that of the adder.

$$Delay_{processor} = CPI \cdot \overset{Numblocks}{\underset{i=1}{MAX}} CycleTime_i = 1.007 \cdot 1 = 1.007$$

$$TDP_{processor} = \sum_{i=1}^{Numblocks} TDP_i = \frac{1}{5} + \frac{1.01}{5} + \frac{1.02}{5} + \frac{1.01}{5} + \frac{1.01}{5} = 1.01$$

$$NBTIguardband_{processor} = \overset{Numblocks}{\underset{i=1}{MAX}} NBTIguarband_i =$$

$$= MAX(0.074, 0.036, 0.067, 0.02, 0.02) = 0.074$$

Finally, the *NBTIefficiency* for the Penelope processor is as shown in the equation below. It can be seen that delay and TDP degradation are very low, whereas the NBTI guardband is significantly lower than the 20% of the baseline. Overall, *NBTIefficiency* for the Penelope processor is 1.28 whereas the baseline *NBTIefficiency* is 1.73. Similarly, enabling an invert mode would be quite expensive in terms of delay and could not be used for combinational blocks.

$$NBTIefficiency_{Sisyphus} = (1.007 \cdot (1 + 0.074))^3 \cdot 1.01 = 1.28$$

5. Conclusions

Conventional processors leave significant performance and power savings on the table due to NBTI guardbands and high Vmin in memory-like structures. Conventional solutions to NBTI like enabling an invert mode, which does not cover all types of blocks, have significant cost in terms of delay, TDP and/or area. In this paper we propose the Penelope processor, which consists of global strategies as well as concrete mechanisms to protect all types of structures in the processor. In particular we propose:

- Strategies to protect any memory-like block (both cache-like and explicitly managed ones) and combinational blocks.
- Custom mechanisms to mitigate NBTI degradation in a Ladner-Fischer adder, integer and FP register files, a scheduler, a DL0 cache and a data TLB.
- A metric to compare solutions to NBTI combining delay, TDP and NBTI guardband in the cycle time.

The benefits of the proposed techniques are (i) their practically negligible cost in hardware (TDP and area), (ii) their low delay impact (if any), and (iii) the significant NBTI guardband reduction. By mitigating NBTI, the proposed Penelope processor allows reducing the guardband due to NBTI degradation for any structure in the chip, and hence, the operating frequency may be boosted or the complexity of such structures (e.g., number of ports, size, etc.) increased without impacting the cycle time. Our results show guardband reductions between 12.6% and 18% for the different blocks without impacting any critical path. Furthermore, Vmin does not increase as much in memory-like structures by mitigating NBTI, hence leading to higher power efficiency of such structures.

6. Acknowledgements

The authors would like to thank Chris Wilkerson and Nam Sung Kim for sharing experimental data, and Alex Piñeiro for the cache workload characterization and for reviewing drafts of this paper. We would also like to thank Javier Carretero and Pedro Chaparro for reviewing drafts of this paper. This work has been partially supported by the Spanish Ministry of Education and Science under grant TIN2004-03702 and Feder Funds.

References

[1] W. Abadeer, W. Ellis. Behavior of NBTI under AC Dynamic Circuit Conditions. In IRPS 2003.

[2] J. Abella, A. González. Heterogeneous Way-Size Cache. In ICS 2006.

[3] M. Agostinelli et al. Erratic Fluctuations of SRAM Cache Vmin at the 90nm Process Technology Node. In IEDM 2005.

[4] M.A. Alam. A Critical Examination of the Mechanics of Dynamic NBTI for PMOSFETs. In IEDM 2003.

[5] D.M. Brooks et al. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. In IEEE Micro, November-December 2000.

[6] G. Chen et al. Dynamic NBTI of PMOS Transistors and Its Impact on Device Lifetime. In IRPS 2003.

[7] S.S. Chung et al. Impact of STI on the Reliability of Narrow-Width pMOSFETs with Advanced ALD N/O Gate Stack. In IEEE Transactions on Device and Materials Reliability, vol. 6, no. 1, March 2006.

[8] P. Hulbert. High Throughput Gate Dielectric Reliability Testing: Digging Out from the Backlog. Keithley Instruments Inc. technical note, 2004.

[9] S. Kaxiras, Z. Hu, M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In ISCA 2001.

[10] S.V. Kumar, C.H. Kim, S.S. Sapatnekar. Impact of NBTI on SRAM Read Stability and Design for Reliability. In ISQED 2006.

[11] R.E. Ladner, M.J. Fischer. Parallel Prefix Computation. In the Journal of the ACM, vol. 27, no. 4, October 1980.

[12] N.R. Mahapatra, S.V. Garimella, A. Tareen. An Empirical and Analytical Comparison of Delay Elements and a New Delay Element Design. In the Workshop on VLSI 2000.

[13] S. Mahapatra, P.B. Kumar, M.A. Alam. Investigation and Modeling of Interface and Bulk Trap Generation During Negative Bias Temperature Instability of p-MOSFETs. In IEEE Transactions on Electron Devices, vol. 51, no. 9, September 2004.

[14] S. Rangan, N. Mielke, E.C.C. Yeh. Universal Recovery Behavior of Negative Bias Temperature Instability. In IEDM 2003.

[15] V. Reddy et al. Impact of Negative Bias Temperature Instability on Digital Circuit Reliability. In IRPS 2002.

[16] C. Schlünder et al. On the Degradation of P-MOSFETs in Analog and RF Circuits under Inhomogeneous Negative Bias Temperature Stress. In IRPS 2003.

[17] D.K. Schroder, J.A. Babcock. Negative Bias Temperature Instability: Road to Cross in Deep Submicron Silicon Semiconductor Manufacturing. In the Journal of Applied Physics, vol. 94, no. 1, July 2003.

[18] D. Tarjan, S. Thoziyoor, N.P. Jouppi. CACTI 4.0. HP Technical Report HPL-2006-86.

[19] X. Xuan. Analysis and Design of Reliable Mixed-Signal CMOS Circuits. Phd. thesis at Georgia Institute of Technology, December 2004.