Reducing NBTI-induced Processor Wearout by Exploiting the Timing Slack of Instructions

Fabian Oboril, Farshad Firouzi, Saman Kiamehr and Mehdi B. Tahoori
Chair of Dependable Nano Computing (CDNC), Karlsruhe Institute of Technology (KIT)
Haid-und-Neu-Str. 7
76131 Karlsruhe, Germany
fabian.oboril@kit.edu, firouzi@ira.uka.de, kiamehr@kit.edu, mehdi.tahoori@kit.edu

ABSTRACT

Transistor aging due to Negative Bias Temperature Instability (NBTI) is a major reliability challenge for embedded microprocessors at nanoscale. It leads to increasing path delays and eventually more failures during runtime. In this paper, we propose a novel microarchitectural approach combining aging-aware instruction scheduling with specialized functional units to alleviate the impact of NBTI-induced wearout. To achieve this, the instructions are classified depending on their worst-case delay into critical (i.e. the instructions whose delay is close to the cycle boundary) and non-critical instructions (i.e. those instruction with larger timing slack). Each of these classes uses its own (specialized) functional unit(s). By that means it is possible to increase the idle ratio of the units executing the critical instructions, which can be used to extend lifetime by up to 2.3x in average compared to the usually used balanced scheduling policy.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

Keywords

instruction scheduling, NBTI, functional unit, transistor aging, microarchitecture

1. INTRODUCTION

Aggressive transistor scaling of CMOS technology over the past decades allowed increasing transistor counts and by this means the implementation of more and more features in modern microprocessors. This also enabled the successful entrance of embedded systems in almost all areas of our daily life. Though, many of these systems have very tight reliability constraints. However, with decreasing feature sizes to nanoscale dimensions, reliability is threatened by various challenges. Thereby, a major reliability issue of nanoscale embedded processors is fast transistor aging [6, 10, 24].

CODES+ISSS'12, October 7–12, 2012, Tampere, Finland.

Among several phenomena *Negative Bias Temperature Instability* (NBTI) is the dominant transistor wearout effect [6, 32]. It leads to a shift of the threshold voltage V_{th} of the affected transistors, which manifests in an increasing switching delay of these transistors. Over the time, this results in increasing path delays, which can lead to timing violations.

To combat these runtime degradation issues manufacturers are currently adding *guardbands* to their designs, to ensure that the chips will be functional for a certain lifetime. However, this overdesign increases development and manufacturing costs, which is crucial for the embedded segment where low costs are a primary target. Hence, new approaches are necessary to take further advantage of scaled technology nodes. Therefore, a lot of research is done at various design levels. To name just a few, special NBTI-resilient circuits [3], input vector control [15, 16, 33], power gating [12], adaptive body biasing [29], dynamic voltage and frequency scaling (DVFS) [5, 26] and enhanced instruction and application scheduling techniques [28, 29] are some of the existing aging mitigation methods. The last two techniques try to balance the necessary calculations on the available units (cores) to achieve equal wearout states on all units (cores), which should guarantee a longer lifetime of these parts.

Our investigation of various (embedded) microprocessors shows that the execution stage (i.e. the functional units) in a pipelined microprocessor is among the most timing-critical stages and hence to improve the overall lifetime of the microprocessor, the execution stage must be dealt with. However, as we will show in this paper, balancing the number of executed instructions (workload) over several functional units is not always the best aging mitigation strategy. This is mainly due to the fact that within the same clock cycle boundaries, different instructions executed by the same functional unit have different timing criticality and consequently different time slacks. Although all instructions executed in an ALU have an execution time of one cycle, there are some instructions whose delay is close to one cycle (e.g. arithmetic operations with long operands), while others need much less time (e.g. simple logic operations). Hence, the first group of instruction is called *timing-critical (TC)*, the other one is named *non-timing-critical (NTC)*. In fact, this makes no difference from the performance point of view (always one cycle). However, in terms of aging, when the path delay of the circuit increases, the TC instructions start to fail (i.e. have timing delays) while NTC instructions will continue to be executed correctly for a much longer time. This observation is the main motivation behind the technique presented in this work and illustrated in Figure 1. We plan to ex-

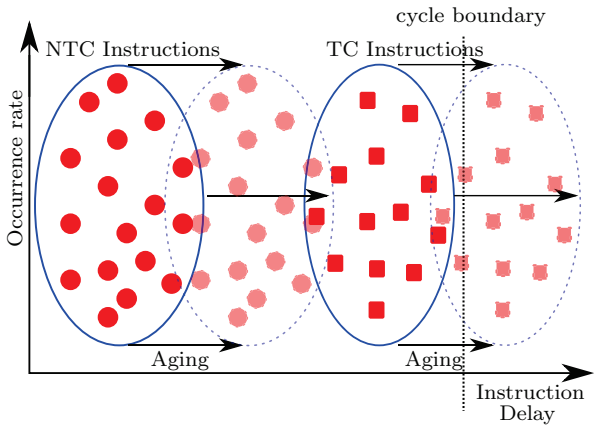

Figure 1: Illustration of TC and NTC instructions

ploit this information for aging-aware scheduling of different instruction classes on different functional units.

If an embedded microprocessor with two functional units of the same type (e.g. ALU) uses a balanced instruction scheduling technique, both units will fail almost after the same time. Now imagine, that one unit is used only for the TC instructions and the other one just for the NTC instructions. Since the latter have a delay which is far less than the TC instructions, the first unit will always fail first and hence determines the lifetime. In addition, as long as the fraction of TC instructions is less than 50 %, the critical unit will execute less instructions than in the first case. Hence, the idle ratio of this unit is increased. This can be used in combination with various architectural techniques such as power gating [12] or input vector control for NOP-operations [16] to reduce the NBTI effect and hence, to increase the lifetime of this unit.

Based on this idea, in this paper we propose a novel microarchitectural approach combining aging-aware instruction scheduling with specialized functional units to alleviate the impact of NBTI-induced wearout. In our proposed microarchitecture, the instructions are categorized depending on their worst-case delays into classes of critical and non-critical instructions. Each of these classes uses its own (specialized) functional unit(s). By that means it is possible to increase the idle ratio of the units executing the critical instruction, which can be used to extend lifetime of the functional units by applying special mitigation techniques. Our simulation results show that the proposed scheduling technique can achieve this goal and can yield improvements in terms of *Mean Time to Failure (MTTF)* of 1.5x (for power gating) and 2.3x (for input vector control) in average, compared to the standard (i.e. balanced) scheduling.

The rest of this paper is organized as follows. In Section 2 the NBTI phenomenon is introduced followed by the presentation of two special aging mitigation techniques in Section 3. Our proposed microarchitectural solution is provided in Section 4. Afterwards we present in Section 5 the simulation framework we use to investigate the proposed methodology. Also the simulation results will be shown in this section. Finally, the paper is concluded in Section 6.

2. PRELIMINARIES ON NBTI-INDUCED AGING

NBTI occurs in PMOS transistors when the transistor is under negative gate-source bias i.e., $V_{gs} = -V_{dd}$, (stress mode). When a PMOS transistor experiences the stress mode, some Si-H bonds break at the interface $Si - SiO_2$ due to the presence of holes in the channel. The resulting H diffuses away and as a result some positive traps (Si^+) are left causing the magnitude of transistor threshold voltage V_{th} to increase [7]. When the stress mode is removed (recovery mode i.e. $V_{gs} = 0$) some of the created interface traps are healed, and the threshold voltage shift can be partially recovered [7]. This process is shown in Figure 2.

The long-term NBTI-induced threshold voltage shift, ΔV_{th}, can be modeled by the following equation [32]:

$$\Delta V_{th} = A(T) Y^n t^n, \tag{1}$$

where $A(T)$ is a technology dependent function of temperature T, n is a fabrication process dependent constant, t is the total time, and Y is the duty cycle. The duty cycle represents the ratio between stress to total time. To calculate the delay degradation of a gate due to ΔV_{th} imposed by NBTI, the alpha power model can be used [11]:

$$\tau = \frac{K}{(V_{gs} - V_{th})^{\alpha}}. \tag{2}$$

Here K is a technology dependent factor, V_{gs} is the gate-source voltage, and α represents the velocity saturation. If ΔV_{th} is small, by using first order Taylor expansion, the following equation for NBTI-induced delay degradation can be derived from Equation (2):

$$\Delta\tau = \frac{\alpha \Delta V_{th}}{V_{gs} - V_{th0}} \times \tau_0. \tag{3}$$

where ΔV_{th} is the NBTI-induced threshold voltage change, V_{th0} is the original transistor threshold voltage, and τ_0 is the pre-aging delay of the gate. By using Equations (1) and (3), the total NBTI-induced delay degradation can be estimated as follows:

$$\Delta\tau = \frac{A(T)\alpha Y^n t^n \tau_0}{V_{gs} - V_{th}}. \tag{4}$$

It should be noted that, this equation is valid for a simple inverter, which has one transistor in each pull-up/pull-down network. For other gates (e.g. NAND and NOR) the stacking effect has to be considered as well. In stacked structures, the delay of each gate is affected by the threshold voltage change of all the internal transistors. Moreover, the duty

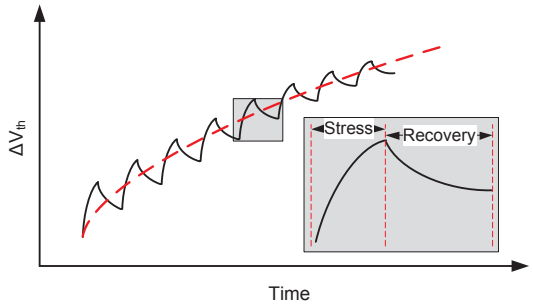

Figure 2: The conceptual illustration of V_{th} change under stress and recovery conditions

cycle of each transistor depends not only on the state of its input, but also on the state of upper and lower transistors [35].

Beside NBTI also *Positive Bias Temperature Instability* (PBTI) is an emerging reliability problem due to the introduction of high-k gate oxides [27]. However, the impact of PBTI on the behavior of NMOS transistors is very similar to the NBTI effect on PMOS transistors and hence our proposed technique can be used for PBTI mitigation as well.

3. SPECIAL AGING MITIGATION TECHNIQUES

The main goal of this work is to increase the idle ratio of the functional units executing aging critical instruction by applying a novel instruction scheduling policy in combination with special aging mitigation techniques. In the following we will introduce two approaches namely input vector control and power gating, that can be used during idle phases of functional blocks to alleviate their NBTI-induced wearout by increasing their relaxation phases.

3.1 Input Vector Control

Duty cycle (the time ratio which transistor is in stress mode to the total time) has a strong impact on NBTI induced wear out. Moreover, the duty cycle of each transistor in the circuit is determined by the input vector and the state of the gate inputs. Therefore, the input vector can affect the NBTI-induced delay degradation and by that means Input Vector Control (IVC) can be used during the standby mode (when the circuit does not perform any operation) to reduce the NBTI effect without any performance degradation.

IVC is a well known technique which has been used for leakage power reduction in conjunction with clock gating [23]. In this approach, the clock of a unit which does not perform any operation is disabled, to avoid dynamic power dissipation due to switching activity. IVC is also combined with other techniques such as Dual V_{th} assignment to minimize the static leakage power [36].

There is also some work to minimize the NBTI effect by exploiting IVC techniques. In the following we briefly present a few of them. The authors in [32] have investigated that the input vector has a strong effect on NBTI and proposed an IVC method based on random vector simulation to mitigate the NBTI effect. IVC in conjunction with internal node control is used in [8] for minimizing the effect of NBTI. This approach uses a concept similar to test point insertion. A linear programming approach to find the best NBTI-aware input vector to be applied during standby is proposed by the authors of [15]. Furthermore, a microarchitectural IVC study using special No-Operations as input vectors for an ALU is presented in [16].

NBTI-induced circuit degradation and leakage power are both strongly dependent on the input vector but in different directions. Two different methods are proposed in [33] to find the best input vector resulting in the minimum NBTI-induced wearout and/or leakage power: exhaustive search and probability based algorithm. In the exhaustive method the random input vectors are generated and the best input vector resulting in the minimum NBTI-induced delay degradation is chosen. In the probability based method, first a number of input vectors are generated. Then the best set of input vectors with the minimum wearout due to NBTI is chosen. Based on the 0/1 probability of each input obtained from the selected set, the new random inputs are generated until a convergence is obtained. In [34], IVC is combined with gate replacement technique using a dynamic programming algorithm to co-optimize leakage power and NBTI-induced degradation.

3.2 Power Gating

Power gating is another technique to mitigate NBTI [12, 13]. Since it reduces the power consumption, temperature decreases which in turn alleviates the NBTI effect. Furthermore, the stress time of PMOS transistors is reduced and thereby duty cycle, since a power gated PMOS transistor is in recovery mode ($V_{gs} = 0$). However, it takes some time to power down or up a block of a processor, whereby the time periods depend on the size and amount of the power gate transistors as well as the size of the power-gated block. Recent work has shown, that an ALU can have a wake up time of 3 ns to 10 ns [20, 30]. Hence, the power gated block is not available for a certain amount of time, which can lead to performance losses. Due to the implied performance penalty, power gating is nowadays mainly used in multi-core processors to switch off unused cores to reduce power consumption [21]. However, it also becomes more and more popular in low-power embedded systems [4, 19].

4. AGING-AWARE INSTRUCTION SCHEDULING

In this section we describe our microarchitectural approach to mitigate the impact of NBTI using enhanced instruction scheduling and specialized functional units.

Based on the observations described in Section 2, temperature and duty cycle are two of the key aspects that influence the NBTI effect. Hence, to slow down wearout due to NBTI, it is mandatory to reduce temperature and duty cycle. In a functional unit like an ALU, these two parameters strongly depend on the sequence of executed instructions. Especially the idle periods between two instructions can be used to mitigate the impact of NBTI, for example by applying power gating or input vector control (IVC) for NOP-operations [12, 16]. However, these techniques can only be used efficiently, if the idle periods between two instructions are very long, which is often not the case. Our envisioned microarchitectural technique aims at increasing the idle periods between two instructions using an enhanced instruction scheduler. By that means the benefit that can be taken from power gating or IVC can be increased. In the following we introduce our idea using an exemplary ALU. However, the idea is also applicable to other functional units.

4.1 Instruction Classification

By investigating the delay of various paths in a functional unit such as an ALU in detail, it is observed that different instructions have different execution times (path delays). Some instructions need just a small fraction of a clock cycle to complete, while others need almost the entire clock cycle. The real execution time (i.e. delay) thereby strongly depends on the instruction and the applied operands. For the ALU of IVM [31] synthesized using the Synopsys Design Compiler and the SAED 90 nm library, the worst-case delay for each instruction is depicted together with the occurrence rate of the instructions in Figure 3.

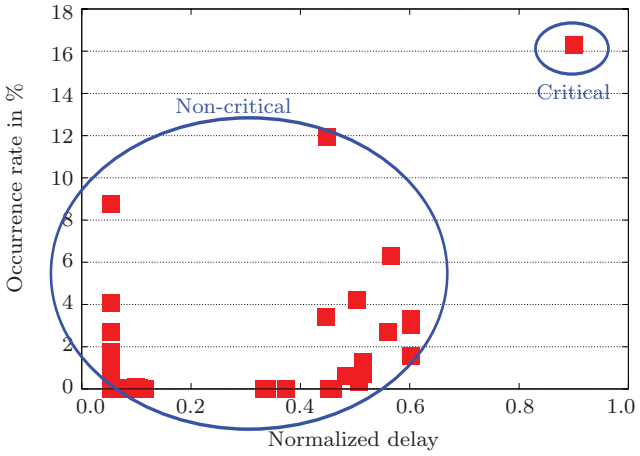

Figure 3: Worst-Case delay distribution and occurrence rate of ALU instructions

From timing perspective, the delay of a functional unit is determined by its longest path delay (slowest instruction) which is obtained from static timing analysis. Once the clock period is set accordingly, from the performance perspective at microarchitecture level, all instructions are considered the same. However, due to aging effects, the delay of various paths in the circuit increases. Finally for some instructions the (intermediate) result will not be computed within the given timing boundaries, i.e. one clock cycle. Thereby, the instructions whose timing slack is very small are the first ones that start to fail. This means that the *timing critical (TC) instructions* (the ones with a real delay close to one cycle) determine the lifetime of the ALU. Moreover, the critical instructions (here it is ADDQ, i.e. add for 64 bit operands) occur much less frequent than the non-critical instructions. Using the performance simulator gem5 [9] we observed that just 16 % of all instructions of various SPEC2000 benchmarks (compiled with GCC 4.3 and *-O3*-optimization) that are executed in the ALU(s) belong to the critical category, while 84 % are *non-timing critical(NTC) instructions* as detailed in Table 1. Based on this observation, our key idea to increase the lifetime of the functional units is to use dedicated unit(s) for each of the two instruction classes, instead of executing both classes in the same unit(s).

Please note that for another ALU or functional unit with a different set of instructions and/or different implementation the timing distribution of various instructions and their occurrence rate will change from what is shown in Figure 3, but the overall classification will still be valid. For instance, logic operations will have less delay than arithmetic operations (with wider word sizes and more sophisticated bit operations). This observation, which basically tells different instructions have different *aging-criticality* can be used for aging-aware scheduling. For example, a functional unit might be aged in a way that it can no longer be used for TC instructions, but it still can execute NTC instructions with no timing failures.

Workload	Instructions	NTC [%]	TC [%]
applu	160,614,011	63.4	36.6
bzip2	293,455,843	80.9	19.1
equake	459,022,375	87.5	12.5
gcc	502,540,388	85.7	14.3
gzip	558,920,018	80.8	19.2
lucas	163,374,308	86.2	13.8
mcf	75,321,231	87.5	12.5
mesa	564,352,647	87.8	12.2
mgrid	80,766,502	93.5	6.5
parser	351,760,701	90.8	9.2
swim	93,599,572	66.7	33.3
twolf	365,459,289	84.8	15.2
wupwise	418,895,837	87.8	12.2
average	**314,467,902**	**83.7**	**16.3**

Table 1: Workloads and their instruction ratios (execution time = 0.5 seconds)

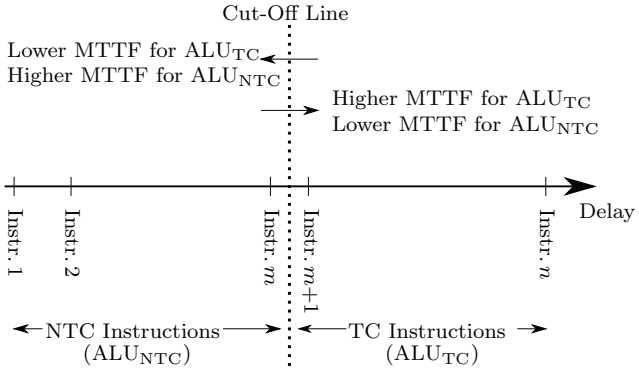

Figure 4: Classification of instructions into TC and NTC and its impact on MTTF

4.1.1 *Optimal cut-off line between TC and NTC instructions*

In the following we present a general approach to find the cut-off line to obtain the highest MTTF considering all units. This approach works for any delay distribution of various instructions and occurrence rates. For simplicity, lets consider that there is one functional unit for the TC instructions (ALU$_{TC}$) and another one for the NTC instructions (ALU$_{NTC}$). As illustrated in Figure 4, the placement of the cut-off line is crucial. If the cut-off line is shifted to the right, more instructions are considered as NTC. Hence MTTF of ALU$_{NTC}$ will decrease (more instructions lead to a lower idle ratio which lowers MTTF; in addition the worst-case delay increases and hence MTTF decreases). However, at the same time MTTF of ALU$_{TC}$ will be improved (less instructions lead to a higher idle ratio and hence higher MTTF). If the cut-off line is moved to the left, MTTF of ALU$_{NTC}$ will increase (less instructions, lower worst-case delay), but MTTF of ALU$_{TC}$ will decrease (more instructions). Hence, finding the right placement of the cut-off line is an optimization problem.

The algorithm, to find the optimal solution is described in Algorithm 1. First, only the instruction with the highest worst-case delay (instruction n) is considered as TC instruction (Step 1). Then, simulations are processed and MTTF of ALU$_{NTC}$ and ALU$_{TC}$ is evaluated (Step 2). Afterwards, the set of TC instructions is expanded by instruction $n-1$ (Step 5) and again MTTF of both units is evaluated (Step 6). This process is continued until the highest MTTF considering both units is found. In our cases the algorithm yielded the result depicted in Figure 3.

4.2 Delay-Aware Scheduling

Normally the information about the instruction delay is not used by the instruction scheduler (or the unit which de-

Algorithm 1 Finding the optimal cut-off line between NTC and TC instructions

// List all instructions according to their worst-case delay,
// i.e. $\text{delay}_{\text{Instruction 1}} \leq \cdots \leq \text{delay}_{\text{Instruction } n}$
1. TC = {Instruction n}
2. Run simulations with this setting and extract:
   $\text{MTTF}_0 = \min(\text{MTTF}_{\text{ALU}_{\text{TC}}}, \text{MTTF}_{\text{ALU}_{\text{NTC}}})$
3. $i = 0$

Do
   4. $i = i + 1$
   5. TC = TC $\cup$ {Instruction $n - i$}
   6. Run simulations with this setting and extract:
      $\text{MTTF}_i = \min(\text{MTTF}_{\text{ALU}_{\text{TC}}}, \text{MTTF}_{\text{ALU}_{\text{NTC}}})$

While ($\text{MTTF}_i > \text{MTTF}_{i-1}$)

cides which functional unit will be used). Instead, if two ALUs are available, the scheduler will balance the workload, so that both ALUs will execute roughly 50% of all the instructions, which is mainly due to performance reasons. Also some previous work on aging-aware instruction scheduling [28], that considers the delay of all instructions to be the same (i.e. 1 clock cycle), shows that the balanced technique is better than unbalanced ones in terms of MTTF. However, as we discussed before, if the accurate timing of instructions is taken into account, a balanced scheduling is not the best choice from the aging perspective. Hence, in our proposed microarchitecture each instruction class has its own dedicated ALU, i.e. one ALU just for the TC instructions and one ALU just for the NTC ones, although this can lead to an unbalanced load among the two ALUs.

The scheduler (or the unit which decides which functional unit will be used) selects the unit, which corresponds to the actual instruction based on the instruction opcode. Hence, no extra bits to encode the classification are necessary. For example in our evaluated scenario the critical ADDQ instruction has the opcode 1020h. Whenever this opcode is detected, the corresponding instruction (ADDQ) is sent to the corresponding ALU, while all other instruction will be sent to the other ALU. However, due to this additional classification of instructions (TC ALU instructions, NTC ALU instructions) a small hardware overhead comes along with our proposed technique. Using this approach the first ALU executes only 16 % of all the instructions, while the other ALU handles the other 84 %. This means, that the first ALU, which deals with the critical instructions, is idle most of the time, which is also shown in Figure 5. Hence, this unit can be power gated very efficiently and also IVC can be applied most of the time. *Please note that the "critical ALU" is the one which determines the lifetime, since the non-critical instructions have a delay which is much smaller than the delay of the critical instructions and even with aging included never surpasses the delay of the critical ones, before these will start to fail.*

Of course, this unbalanced load can lead to a performance loss compared to an architecture with two "full" functional units of the same type. However, the performance loss is relatively small, as we will explain in the next section. Moreover, the functional units in the first case can be more specialized, hence they can be smaller. By this means die size and manufacturing costs can by saved. Furthermore, smaller units will consume less power and by that means stay cooler. This again will help to slow down wearout.

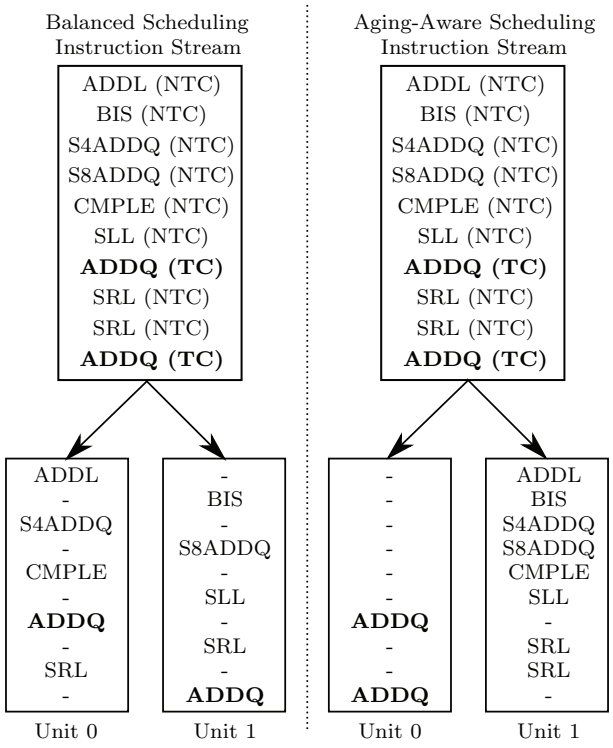

Figure 5: Assignment of instructions to functional units for balanced and aging-aware scheduling

In our case, the specialized critical-ALU needs just to support the ADDQ-instruction (ADD instruction for three 64 bit operands a+b = c). Hence, the feature set can be heavily reduced for this specialized ALU, resulting in a very small size. Using Synopsys Design Compiler and the SAED 90 nm library we found out that the size for such a specialized ALU is just 8 % of a normal, general purpose ALU. This means that using one specialzed ALU together with a normal ALU (aging-ware) results in only 54 % of the size of two normal ALUs (balanced) as it is shown in Table 2. Furthermore, we also investigated the power consumption of these two configurations using Synopys PrimeTime. For this purpose we extracted the instruction sequences for each ALU using the simulation framework detailed in Section 5 and various SPEC2000 workloads. With ModelSim the switching activity behavior for each benchmark and each ALU was extracted and afterwards PrimeTime was used to calculate the power consumption of each ALU for each application. Using the specialized ALU in our proposed microarchitecture, the power consumption of the two ALUs (critical +

Scheduling	normal ALU(s)	special ALU(s)	Area [um^2]	Avg. Power [uW]
Balanced	2x	0	91790 (100 %)	296 (100 %)
Aging-Aware	1x (NTC)	1x (TC)	49546 (54 %)	160 (54 %)
Aging-Aware (Enhanced)	2x (NTC)	1x (TC)	95441 (104 %)	307 (104 %)

Table 2: Different scheduling techniques and combinations of ALUs (normal once and specialized once for TC instructions)

non-critical) can be reduced to 54 % (extracted with Synopsys PrimeTime using real switching activity values) for all executed SPEC200 benchmarks (see Table 2). Due to the performance loss, the small size and the low power consumption of the specialized ALU, another option beside using one critical and one non-critical ALU is to feature two non-critical ALUs and one critical. This will eliminate the performance penalty, as we will show later and the area as well as power overhead is negligible compared to the entire embedded processor.

If an embedded processor uses only one ALU, our proposed technique leads to a small area and power overhead, since a second ALU has to be implemented. However, due to its small size and power consumption both overheads are very minimal. In addition, lifetime will be greatly improved (see Section 5).

4.3 Further Extensions

Beside for a simple ALU, the proposed scheduling technique is also applicable to much more sophisticated functional units such as floating point units or vector execution units. Also these units will feature some instructions (typically multi-cycle instructions) that have a slack close to zero in one of their execution cycles.

Moreover, the concept can be also extended to handle process variation as well. If two functional units of the same type are available, one can be faster than the other one due to process variation, i.e. the timing slack of an instruction executed in the faster unit is larger than in the slower unit. In that case, the TC instructions should be scheduled to the faster unit, while the NTC instructions should use the slower unit.

5. EXPERIMENTAL RESULTS

5.1 Simulation Setup

To validate our microarchitectural approach and to investigate the advantages as well as disadvantages, we used our microarchitectural framework ExtraTime [25]. It is based on the cycle-accurate microarchitectural performance simulator gem5 [9] and includes models for power (based on McPAT [22]), temperature (based on HotSpot [18]) and aging, so that these parameters can be observed during the execution of typical applications. The temperature information in conjunction with information about the usage/activity of different microarchitectural blocks is used by our microarchitectural aging model for NBTI. This is based on the transistor-level model explained in Section 2. For each microarchitectural block (e.g. ALU, instruction decoder, etc.) it is assumed that all transistors behave similarly (i.e. have the same temperature, aging rates, etc.). Hence, a representative transistor can be chosen, for which the current and future V_{th} shift is estimated. Based on that the delay increase using an alpha power law can be calculated and so the current and future delay change of the entire block can be determined. We have validated this representative transistor model with accurate circuit-level implementation and the results show a high accuracy (< 3 % difference).

Processor	Single-core @ 3 GHz, out-of-order, 4-issue
L1-Cache	64 KByte, 3 cyc latency
L2-Cache	2 MByte, 15 cyc latency
Execution Units	2x ALU, 2x CALU, 2x FPU
Expected wearout	$\Delta^{rel}d = 10$ % in 3 years, i.e MTTF = 3 years (90 °C)
Conditions	T_{start} =57 °C, $V_{dd} = 1.0$ V, $V_{th} = 0.21$ V
SPEC2000 benchmarks	applu, bzip2, equake, gcc, gzip, lucas, mcf, mesa, mgrid, parser, swim, twolf, wupwise
Runtime	0.5 seconds excluding initialization

Table 3: Configuration details for the experiments

Details on the processor configuration modeled and the simulated workloads (SPEC2000 benchmarks) can be found in Table 3. As functional unit we have again chosen the ALU detailed in the previous section.

As said before, our proposed scheduling technique aims at increasing the idle ratio of the ALU executing the TC instructions. By that means it is possible to increase the Mean Time To Failure (MTTF) of these functional units by exploiting NBTI mitigation techniques. Please note that it is always the ALU executing the TC instructions, which will fail first. Hence this unit is mainly responsible for the achieved MTTF values. Moreover, it is worth to note that usually the execution stage is the pipeline stage with the smallest timing slack, i.e. highest delay. This means that a lifetime extension of the functional units which are part of the execution stage is crucial, if lifetime of the entire microprocessor should increase. The aging mitigation techniques for the functional units we used in this paper are input vector control and power gating. The details for both are provided in the following subsections.

5.1.1 Applied Input Vector Control Technique

Usually input vector control (IVC) is used at gate-level to mitigate the influence of NBTI and/or leakage (see Section 3). However, IVC can be also used at microarchitecture-level. At that level, input vectors are applied at the primary inputs of the entire microarchitectural blocks, such as functional units. For an ALU, the input vector consists of an instruction opcode and two instruction operands. Since, the NBTI-minimizing input vector should not affect the program execution running on the processor, it has to be a No-Operation (NOP) [17]. In the selected superscalar out-of-order processor the NBTI-minimizing input vector can be applied at the ALU inputs every cycle the ALU is idle, i.e. not executing an instruction. Since it takes some time, until all internal gates of the functional unit are in the state resulting in the minimal NBTI-induced wearout, the first cycle in every idle period is considered as a normal operational cycle with higher degradation rate.

To identify the aging rates of the ALU, when the NBTI-minimizing input vector is applied, we did the following: First, the gate-level description of the IVM processor [31] (similar to the core modeled in gem5) is extracted using Synopsys Design Compiler and the SAED 90 nm library. In the same step, all critical paths of the functional units are extracted as well. Afterwards, the signal probabilities (probability of a signal being 1) of all internal nodes are obtained with Modelsim. Using these signal probabilities the duty cycle of all transistors inside the ALU can be derived and by that means the aging rate of these transistors (Equation 4). Then, a linear programming solver is used to extract the input vector resulting in the minimum NBTI-induced degradation similar to [16]. Finally, the wearout rate for

	Balanced 2x normal ALU		Aging-Aware 1x TC ALU, 1x or 2x NTC ALU	
	power gating	input vector	power gating	input vector
applu	4.5	5	8.5	11
bzip	5	5.5	5	13
equake	5	5	7	10.5
gcc	4	5.5	6.5	14
gzip	4.5	4.5	4.5	13
lucas	4.5	5.5	4.5	12.5
mcf	5	5	8	10
mesa	5	5	5.5	9
mgrid	4.5	5	13	8.5
parser	4	5.5	8	14.5
swim	5	4.5	6	11
twolf	4.5	5.5	7.5	14
wupwise	4	5	4.5	12.5
Average	**4.6 (+1.5x)**	**5.1 (+1.7x)**	**7 (+2.3x)**	**11.8 (+3.9x)**

Table 4: MTTF in years for several SPEC2000 benchmarks (with power gating or input vector control)

this NBTI-minimizing input vector is calculated and used in our microarchitectural aging model for the representative transistor. For the normal ALU of IVM our extracted input vector resulting in the minimum NBTI-induced degradation leads to a relative delay increase of 6.25 % in 3 years. This means, that the duty cycle of our representative transistor in case this special input vector is applied (i.e. in idle cycles) is just 0.15 (see Equation 4). In non-idle cycles, i.e. in cycles the ALU is executing "normal" operations, the duty cycle of the representative transistor estimates the average duty cycle of the transistors in the critical paths [25].

5.1.2 Applied Power Gating Technique

The chosen power gating technique is relatively simple. After an idle period of at least 200 cycles a functional unit can be power gated. The following power down time until the ALU is completely power gated is set to 3 ns [20, 30]. Until no new instruction is incoming, the unit then remains power gated. When a new incoming instruction is detected, the unit is woken up. The wake up time is set to 7 ns [20, 30] and afterwards it can execute the new instruction(s). In our microarchitectural simulations only in the period, in which the unit is completely power gated, the duty cycle of all transistors in this block is considered as 0. In the other cycles the duty cycle of the representative transistors estimates the average duty cycle of the transistors in the critical paths [25].

5.2 Results

As said before, our proposed scheduling technique aims at increasing the idle ratio of the ALU executing the TC instructions. By that means it is possible to increase the Mean Time To Failure (MTTF) of these functional units by exploiting NBTI mitigation techniques such as power gating or input vector control. Since the units executing the TC instructions, are the ones that will fail first, these functional units also determine the lifetime of the entire processor. The corresponding MTTF and performance results for these techniques can be found in Table 4 and Table 5.

The previously introduced power gating scheme can greatly improve MTTF. Using power gating in combination with the aging-aware scheduling, MTTF can be increased to 7 years in average as shown in Table 4. Since the standard scheduling policy yields only 4.6 years, this translates into a benefit of 52 % of the aging-aware scheduling over the standard policy. However, in some cases, both approaches deliver comparable results. This is due to the fact. that a unit cannot be power gated in every idle period. Compared to the situation without power gating, in which MTTF is just 3 years, the improvements are even better and underline the efficiency of our proposed scheduling technique.

Beside power gating, also input vector control (IVC) can be very helpful to mitigate NBTI-induced aging as explained before (see Table 4). If this special input vector is applied at the primary inputs of the investigated ALU in all idle cycles

	Balanced 2x normal ALU		Aging-Aware 1x TC ALU, 1x NTC ALU		Aging-Aware (Enhanced) 1x TC ALU, 2x NTC ALU	
	w/o power gating	w/ power gating	w/o power gating	w/ power gating	w/o power gating	w/ power gating
applu	0.93	0.93	0.93	0.92	0.93	0.92
bzip2	0.78	0.77	0.74	0.73	0.77	0.76
equake	0.74	0.74	0.74	0.73	0.74	0.74
gcc	0.83	0.83	0.80	0.79	0.83	0.82
gzip	1.25	1.25	1.20	1.20	1.26	1.25
lucas	0.72	0.72	0.71	0.71	0.72	0.72
mcf	0.24	0.24	0.24	0.24	0.24	0.24
mesa	1.18	1.18	1.15	1.15	1.18	1.17
mgrid	0.67	0.67	0.67	0.67	0.67	0.67
parser	0.70	0.70	0.68	0.67	0.70	0.68
swim	0.22	0.22	0.22	0.22	0.22	0.22
twolf	0.68	0.68	0.67	0.65	0.68	0.66
wupwise	1.05	1.05	1.02	0.99	1.05	1.03
Average	**0.77 (100 %)**	**0.77 (100 %)**	**0.75 (97 %)**	**0.74 (96 %)**	**0.77 (100 %)**	**0.76 (99 %)**

Table 5: Performance (IPC) evaluation of different scheduling techniques

MTTF can be extended to 12 years using the aging-aware scheduling. This is an improvement of almost 4x compared to the case without IVC and still 2.3x compared to the standard scheduling policy with IVC.

In addition, the performance penalty of the proposed scheduling technique is almost negligible. As it is illustrated in Table 5, the average performance loss of the aging-aware instruction scheduling using a (non)-critical classification is 4 % in average and at most 5 %. In case the performance impact is not acceptable, we suggest to use two normal ALUs, which handle the NTC instructions according to a balanced workload distribution and add a third specialized ALU, which deals only with the TC instructions. Since the specialized ALU is very small (see Section 4.2), the additional overhead is negligible and hence the additional manufacturing costs. In addition, this approach does not just incur any performance overhead compared to the standard technique, but it can also further reduce the aging rates in the "normal" ALUs, due to a lower load (i.e. more idle cycles). Please note that the performance results for IVC are not presented in Table 4, since IVC has no impact on performance. Hence, performance with IVC is the same as the performance without power gating.

Now, the important question is whether the applied approach can improve the lifetime of the entire microprocessor. In other words, the key question is whether the lifetime of the entire microprocessor is determined by the lifetime the functional unit due to time criticality of the execution unit compared to the other stages. To answer this question, we synthesized various processor cores using the SAED 90 nm library and Synopsys Design Compiler. For IVM [31] and FabScalar [14] (both superscalar, out-of-order) as well as OpenRisc 1200 [2] and miniMIPS [1] (both in-order), we investigated the delay of each pipeline stage. The results confirm that the execution stage always contains the longest paths, i.e. has the highest delay. That means the wearout of the functional units has to be addressed in order to extend the lifetime of the entire microprocessor. Hence, our proposed technique which extends the lifetime of the functional units can help to increase the MTTF of the entire processor as well.

6. CONCLUSION

Embedded microprocessors at nanoscale are exposed to various reliability issues, which include a more rapid aging of all components. This will eventually reduce the Mean Time to Failure (MTTF), a crucial point for embedded systems that have often very tight reliability constraints (e.g. long mission times). To alleviate the impact of transistor aging due to Negative Bias Temperature Instability (NBTI) we proposed a new delay-aware scheduling methodology in this paper. Instructions are categorized dependent on their timing slacks within clock cycle boundaries and then scheduled to separate functional units. In combination with specialized functional units and efficient mitigation techniques MTTF can be extended in average by 1.5x using power gating and 2.3x using input vector control compared to the standard (i.e. balanced) scheduling approach. At the same time performance is not compromised.

7. REFERENCES

[1] miniMIPS. Opencores: `http://opencores.org/project,minimips`. [Online; accessed July 2012].

[2] OpenRisc 1200. Opencores: `http://opencores.org/project,or1200_hp`. [Online; accessed July 2012].

[3] J. Abella, X. Vera, and A. Gonzalez. Penelope: The NBTI-Aware Processor. In *Proc. of the Int'l Symp. on Microarchitecture*, pp. 85–96, Dec. 2007.

[4] ARM Limited. *Cortex-A8 Technical Reference Manual*, 2010.

[5] M. Basoglu, M. Orshansky, and M. Erez. NBTI-Aware DVFS: A New Approach to Saving Energy and Increasing Processor Lifetime. In *Proc. of the Int'l Symp. on Low Power Electronics and Design*, pp. 253–258, Aug. 2010.

[6] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance CMOS variability in the 65-nm regime and beyond. *IBM Journal of Research and Development - Advanced silicon technology*, 50, pp. 433–449, July 2006.

[7] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula. Predictive modeling of the NBTI effect for reliable design. In *Proc. Custom Integrated Circuits Conf.*, pp. 189–192, 2006.

[8] D. Bild, G. Bok, and R. Dick. Minimization of NBTI performance degradation using internal node control. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pp. 148–153, Mar. 2009.

[9] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 26(4), pp. 52–60, July 2006.

[10] S. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6), pp. 10–16, Nov. 2005.

[11] K. A. Bowman, B. L. Austin, J. C. Eble, X. Tang, and J. D. Meindl. A Physical Alpha-Power Law MOSFET Model. In *Proc. of the Int'l Symp. on Low Power Electronics and Design*, pp. 218–222, Aug. 1999.

[12] A. Calimera, E. Macii, and M. Poncino. NBTI-Aware Power Gating for Concurrent Leakage and Aging Optimization. In *Proc. of the Int'l Symp. on Low Power Electronics and Design*, pp. 127–132, Aug. 2009.

[13] T. Chan, J. Sartori, P. Gupta, and R. Kumar. On the Efficacy of NBTI Mitigation Techniques. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pp. 1–6, Mar. 2011.

[14] N. Choudhary, S. Wadhavkar, T. Shah, H. Mayukh, J. Gandhi, B. Dwiel, S. Navada, H. Najaf-abadi, and E. Rotenberg. FabScalar: Automating Superscalar Core Design. *IEEE Micro*, 32(3), pp. 48–59, May 2012.

[15] F. Firouzi, S. Kiamehr, and M. B. Tahoori. A Linear Programming Approach for Minimum NBTI Vector Selection. In *Proc. of the Great Lakes Symp. on VLSI*, pp. 253–258, May 2011.

[16] F. Firouzi, S. Kiamehr, and M. B. Tahoori. NBTI Mitigation by NOP Assignment and Insertion. In

Proc. of the Conf. on Design, Automation and Test in Europe, pp. 218–223, Mar. 2012.

[17] J. L. Hennessy, D. A. Patterson, and D. Goldberg. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, 2003.

[18] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan. HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design. *Trans. on Very Large Scale Integration Systems*, 14(5), pp. 501–513, May 2006.

[19] R. Islam, A. Sabbavarapu, and R. Patel. Power Reduction Schemes in Next Generation Intel ATOM Processor Based SoC for Handheld Applications. In *Symp. on VLSI Circuits*, pp. 173–174, June 2010.

[20] S. Kim, S. V. Kosonocky, and D. R. Knebel. Understanding and Minimizing Ground Bounce During Mode Transition of Power Gating Structure. In *Proc. of the Int'l Symp. on Low Power Electronics and Design*, pp. 22–25, Aug. 2003.

[21] R. Kumar and G. Hinton. A Family of 45nm IA Processors. In *Proc. of the IEEE Int'l Solid-State Circuits Conf.*, pp. 58 –59, Feb. 2009.

[22] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proc. of the Int'l Symp. on Microarchitecture*, pp. 469–480, Dec. 2009.

[23] S. Naidu and E. Jacobs. Minimizing Stand-by Leakage Power in Static CMOS Circuits. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pp. 370–376, Mar. 2001.

[24] V. Narayanan and Y. Xie. Reliability Concerns in Embedded System Designs. *Computer*, 39(1), pp. 118–120, Jan. 2006.

[25] F. Oboril and M. B. Tahoori. ExtraTime: Modeling and Analysis of Wearout due to Transistor Aging at Microarchitecture-Level. In *Proc. of the Int'l Conf. on Dependable Systems and Networks*, June 2012.

[26] F. Oboril and M. B. Tahoori. Reducing Wearout in Embedded Processors using Proactive Fine-Grained Dynamic Runtime Adaptation. In *Proc. of the European Test Symp.*, pp. 68–73, May 2012.

[27] S. Pae, M. Agostinelli, M. Brazier, R. Chau, G. Dewey, T. Ghani, M. Hattendorf, J. Hicks, J. Kavalieros, K. Kuhn, M. Kuhn, J. Maiz, M. Metz, K. Mistry, C. Prasad, S. Ramey, A. Roskowski, J. Sandford, C. Thomas, J. Thomas, C. Wiegand, and J. Wiedemer. BTI Reliability of 45 nm High-K + Metal-Gate Process Technology. In *Int'l Reliability Physics Symp.*, pp. 352–357, May 2008.

[28] T. Siddiqua and S. Gurumurthi. A Multi-Level Approach to Reduce the Impact of NBTI on Processor Functional Units. In *Proc. of the Great Lakes Symp. on VLSI*, pp. 67–72, May 2010.

[29] A. Tiwari and J. Torrellas. Facelift: Hiding and slowing down aging in multicores. In *Proc. of the Int'l Symp. on Microarchitecture*, pp. 129–140, Nov. 2008.

[30] K. Usami, T. Shirai, T. Hashida, H. Masuda, S. Takeda, M. Nakata, N. Seki, H. Amano, M. Namiki, M. Imai, M. Kondo, and H. Nakamura. Design and Implementation of Fine-Grain Power Gating with Ground Bounce Suppression. In *Int'l Conf. on VLSI Design*, pp. 381–386, Jan. 2009.

[31] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. patel. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In *Proc. of the Int'l Conf. on Dependable Systems and Networks*, pp. 61–71, June 2004.

[32] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Y. Cao. The Impact of NBTI Effect on Combinational Circuit: Modeling, Simulation, and Analysis. *IEEE Trans. on Very Large Scale Integration Systems*, 18(2), pp. 173–183, Feb. 2010.

[33] Y. Wang, X. Chen, W. Wang, V. Balakrishnan, Y. Cao, Y. Xie, and H. Yang. On the efficiancy of Input Vector Control to mitigate NBTI effects and leakage power. In *Proc. of the Int'l Symp. on Quality of Electronic Design*, pp. 19–26, Mar. 2009.

[34] Y. Wang, X. Chen, W. Wang, Y. Cao, Y. Xie, and H. Yang. Leakage Power and Circuit Aging Cooptimization by Gate Replacement Techniques. *IEEE Trans. on Very Large Scale Integration Systems*, (99), pp. 1–14, 2011.

[35] K. Wu and D. Marculescu. Joint Logic Restructuring and Pin Reordering against NBTI-induced Performance Degradation. In *Proc. of the Conf. on Design, Automation and Test in Europe*, pp. 75–80, Mar. 2009.

[36] L. Yuan and G. Qu. Simultaneous Input Vector Selection and Dual Threshold Voltage Assignment for Static Leakage Minimization. In *Proc. of the Int'l Conf. on Computer-Aided Design*, pp. 548–551, 2007.