On BTI Aging Rejuvenation in Memory Address Decoders

Cem Gursoy, Cemil; Kraak, Daniël; Ahmed, Foisal ; Taouil, Mottaqiallah; Jenihhin, Maksim ; Hamdioui, Said

DOI
10.1109/LATS57337.2022.9936940

Publication date
2022

Document Version
Final published version

Published in
Proceedings of the 2022 IEEE 23rd Latin American Test Symposium (LATS)

On BTI Aging Rejuvenation in Memory Address Decoders

Cemil Cem Gürsoy[1], Daniel Kraak[2], Foisal Ahmed[1], Mottaqiallah Taouil[2],
Maksim Jenihhin[1], and Said Hamdioui[2]
[1]Department of Computer Systems, Tallinn University of Technology,
19086 Tallinn, Estonia, {cemil.gursoy,foisal.ahmed,maksim.jenihhin}@taltech.ee
[2]Department of Quantum and Computer Engineering, Delft University of Technology,
2628 CD Delft, The Netherlands,{D.H.P.Kraak,M.Taouil,S.Hamdioui}@tudelft.nl

***Abstract*—Memory designs require timing margins to compensate for aging and fabrication process variations. With technology downscaling, aging mechanisms became more apparent, and larger margins are considered necessary. This, in return, means a larger area requirement and lower performance for the memory. Bias Temperature Instability (BTI) is one of the main contributors to aging, which slows down transistors and ultimately causes permanent faults. In this paper, first, we propose a low-cost aging mitigation scheme, which can be applied to existing hardware to mitigate aging on memory address decoder logic. We mitigate the BTI effect on critical transistors by applying a rejuvenation workload to the memory. Such an auxiliary workload is executed periodically to rejuvenate transistors that are located on critical paths of the address decoder. Second, we analyze workloads' efficiency to optimize the mitigation scheme. Experimental results performed with realistic benchmarks demonstrate several-times lifetime extension with a negligible execution overhead.**

***Index Terms*—BTI, aging, rejuvenation, mitigation, memory, address decoder**

I. Introduction

The continuous miniaturization of devices has been the main driver of improvements in the semiconductor industry. On the other hand, reliability threats have become more severe due to this trend [1]. Bias Temperature Instability (BTI) is considered to be the main contributor to the time-dependent variability of nanometer-scale devices [2], [3]. BTI slows down transistors over time, thus potentially creating reliability issues, such as delay faults. Traditionally, designers use guard-banding to tolerate this time-dependent variability, i.e., margins are added to the design to ensure a reliable operation. A downside of this approach is that these margins result in a penalty in area, power, and performance. Alternatively, designers can embed mitigation schemes that reduce the impact of aging into their design. In this work, we propose an aging mitigation scheme for the memory address decoder logic. Memories are a fundamental part and cover a large area of modern Integrated Circuits (ICs). Therefore, they are critical for the overall reliability of a system. In particular, delay faults in the decoder logic contribute to a significant portion of the customer returns [4], [5]. Cumulative delay due to BTI-induced aging together with the delay from process variations on decoder logic may cause a wrong address selection during a read or write operation, and, consequently, read or write failures.

In literature, most work has focused on analyzing and mitigating the impact of aging on the memory cells [6]–[14]. These mitigation techniques are mainly based on balancing ones and zeros that are stored in the memory cells, since this reduces the BTI aging impact. There is significantly less work on the peripheral structures of memories. For example, mitigation schemes have been proposed for the Sense Amplifier [15], [16]. To the best of our knowledge, there are only two works that target address decoders. The authors of [17] introduce a software-based scheme that mitigates aging by periodically running a rejuvenation workload on top of a user workload. However, it does not provide a method to run the scheme, analyzes simplistic workloads and only targets the Negative BTI (NBTI) aging mechanism. The authors of [18] propose a hardware-based mitigation scheme for address decoders that takes advantage of idle cycles to change the decoder's address input, thereby reducing static BTI stress. A downside of that approach comes from its hardware overhead (area, power and delay). In addition, it may even lead to a higher aging-induced degradation if the idle period is too long.

In this paper, we propose a low-cost software-based aging mitigation scheme to extend the lifetime of a memory's address decoder up to several times. The scheme can be applied to existing hardware, and only requires a minor modification to its software. It mitigates aging by running a *rejuvenation* workload *periodically* during the main functional operation of the system. We propose several approaches to generate such workloads. The rejuvenation workload has an opposite effect (in terms of BTI aging) with respect to the functional workload and, thus, it helps transistors on long paths of the decoder to recover from BTI-induced aging. Our experimental results show that it is possible to recover a significant part of BTI aging at a minimal execution overhead. Our contributions in this paper are threefold:

1. It proposes a low-cost aging mitigation scheme to extend the lifetime of a memory's address decoder up to several times.
2. It proposes a design and workload-aware methodology to generate optimized rejuvenation workload which can be used with the scheme.
3. It validates our proposed scheme and proposed rejuvenation workloads using realistic user workloads and two different decoder designs.

The rest of the paper is organized as follows. Section II provides background on the BTI mechanism and address decoders. Section III introduces our proposed mitigation methodology. Section IV presents the experimental setup, performed experiments, and the obtained results. Sections V and VI provide a brief discussion and conclusions for the paper.

II. Background

In this section, we briefly introduce the adopted BTI aging mechanism and the address decoders. Then, we explain how BTI influences an address decoder.

2022 IEEE 23rd Latin American Test Symposium (LATS) | 978-1-6654-5707-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/LATS57337.2022.9936940

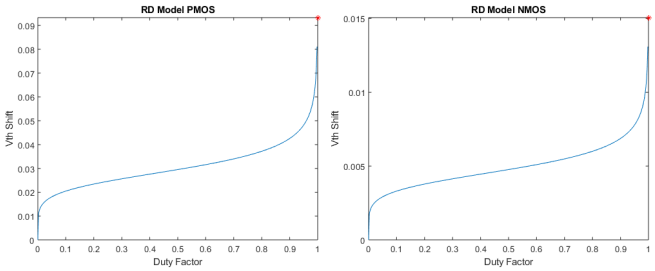


Fig. 1. The adopted BTI aging model for PMOS and NMOS transistors.

A. Bias Temperature Instability model

In this study, we rely on the accurate combined model for Negative BTI in PMOS and Positive BTI in NMOS transistors proposed in [19] for the 28 nm technology and calibrated for 22 nm Predictive Technology Model (PTM) technology as explained in [17]. The resulting dependency of the BTI-induced threshold voltage (Vth) shift depending on the average duty factor (the probability of a transistor to be on) for NMOS and PMOS is presented in Fig. 1.

B. Address Decoder

Address decoders in memories are responsible for accessing the desired cells in the memory cell array. In order to access a particular row and column in the memory cell array, a *wordline decoder* and a *column decoder* are used, respectively. In general the wordline decoder is more critical, since memories typically have more rows than columns. For this reason, we analyze two different 9-to-512 wordline decoder designs in this work, namely a NAND-NOR decoder and an AND-AND decoder.

Fig. 2a shows a simplified schematic of the NAND-NOR decoder. It is a hierarchical design, meaning it consists of a *pre-decoder* and a *post-decoder* stages. The pre-decoder stage is implemented with three 3-to-8 decoders that have the address bits as their input. In our first decoder design, the pre-decoder consists of inverters and NAND-gates. Unique combinations of the original and inverted address bits are fed to the inputs of the NAND-gates. This way, each input combination to the decoder results in one of its outputs becoming low. The post-decoder stage consists of 512 post-decoders that are implemented using NOR-gates. Each post-decoder activates one of the wordlines of the memory cell array. It has as its inputs a unique combination of the outputs from the pre-decoders. In addition, it has as input a *decoder_enable* signal, that is generated by the timing circuit. Using this signal it is possible to control the duration of the wordline activation. The AND-AND wordline decoder is implemented similarly as the NAND-NOR decoder. In this case, the pre-decoders consist of AND-gates and inverters, and the post-decoder stage is implemented using AND-gates.

Fig. 2c shows an important reliability metric of the decoder with respect to timing signals, the *slack* time. It is defined as the time between the pre-decoder outputs being ready and the setup time of the post-decoder inputs. Before the *decoder_enable signal* is activated, the pre-decoder outputs must be ready and stable during *setup time*. If the pre-decoder outputs take too long to settle, the slack becomes negative and the setup time is violated. This may lead to wrong address selection or selection of multiple wordlines and, thus, read or write failures. Increasing timing budget to accommodate more slack time results in more reliable operation, but the memory performance degrades.

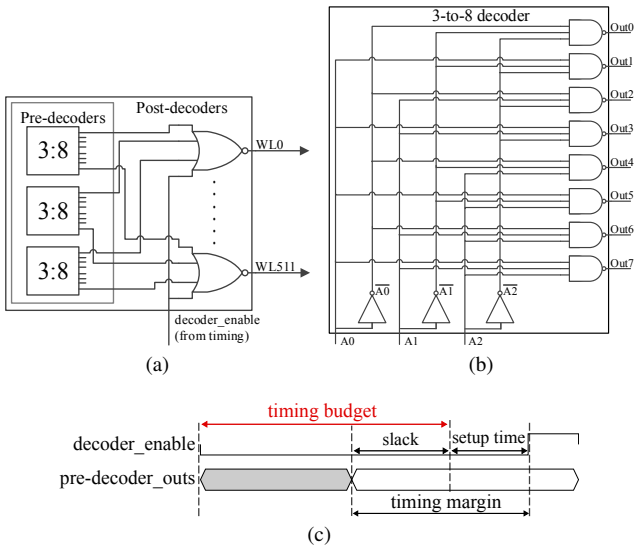


Fig. 2. (a) Schematic of the wordline decoder and (b) a pre-decoder. (c) The slack metric.

C. Aging in Address Decoders

The timing budget is set by the designer, and it is constant over the lifetime of the memory. As the pre-decoder suffers from aging, path delays of the decoder increases and more time required until pre-decoder outputs switch. Hence, less and less time is left for the slack. Eventually, when the slack time goes below zero, timing violations will occur.

Fig. 3a illustrates the delay of the longest path for each output in a 3-to-8 pre-decoder which are obtained by SPICE simulations. Activation delay of a path is the time required to pull the output to high. Likewise, deactivation delay corresponds to 1 to 0 output transition of a path. Fig. 3b illustrates delay increase on decoder's paths due to aging. Since path delays of Out2, Out4, and Out6 reach beyond the timing budget, they cause delay faults in the memory. On the other hand, the other paths still have some slack time.

Transistors suffer from BTI aging only when they are under BTI stress. When a path's input switches, transistors in the path also switch. During activation or deactivation of an output, half of the transistors in the path will switch, thus half of them transitions from the BTI stress state (i.e., the biased condition) to the BTI relief state (i.e., the unbiased condition). The transistors that are in the relief state partially recover from the BTI aging effect. Thus, a path ages the most if workloads do not cause its input to switch [20], since a half of the transistors constantly stays at the BTI stress state and they have no chance to recover. Depending on the workload different addresses are selected and therefore different input combinations are applied to the decoder. Thus, a path's delay increase can be manipulated by workloads which run during memories operation [17].

III. Proposed Mitigation Methodology

The system that we consider as a case study in this work is composed of a CPU and a memory. The address output of the CPU is connected to the memory's address decoders. We analyze the larger (wordline) decoder and the least significant bits of the address signal are connected to the smaller (column) decoder. After the bits of the column decoder, 9 bits of the address signal are connected to the wordline decoder that has three 3-input pre-decoders. It should be noted that while the proposed methodology is

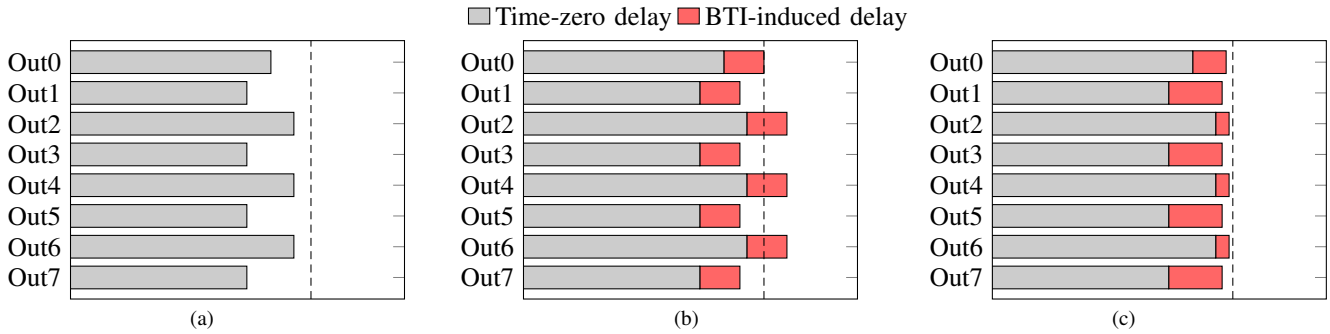


Fig. 3. (a) Time-zero path delays of 3-to-8 pre-decoder. (b) Impact of aging. (c) Aging when rejuvenation workload ran together with main workload.

explained using this specific example, it remains general and applicable for other memory decoder architectures.

While the CPU executes its functional workload, memory operations (read/write) alters the address signal, hence the decoder's input. Fig. 3b shows the path delays after aging while the CPU executes a given functional workload. It is clear that some paths in pre-decoder are longer compared to others. Furthermore, depending on the given workload, some paths have higher delay increase due to aging than others. As in [17], we aim to mitigate aging by changing the workload that affects the memory. This change is done by executing an dedicated small auxiliary workload periodically. We call such workload as a *rejuvenation workload*, because it puts stressed transistors in long paths to the relief state to recover from the BTI aging. Fig. 3c illustrates the resulting path delays after a rejuvenation workload is applied. Compared to Fig. 3b, the longest delays of Out2, Out4, and Out6 are reduced, thus total pre-decoder's delay stays within the specified timing budget (shown as the vertical dashed line).

Our mitigation approach consists of three parts explained in following subsections. In III.A), we present the steps taken to calculate aging for a given workload. In III.B), three different rejuvenation workloads are presented with a method to generate them. In III.C), our mitigation scheme that combines both main and rejuvenation workloads together is explained.

A. Aging Modeling and Assessment

In order to analyze aging in the decoder we implement a flow that contains a high-level and a low-level setup. The high-level setup is responsible for generating a cycle-accurate *memory trace* out of a given workload. The generated trace file must contain a cycle number and selected address pairs during the execution of the workload. The high-level setup can be realized with a RTL simulation of a CPU implementation or a cycle accurate instruction set simulation. the low-level setup has to be run at the transistor level and it is needed to obtain path delays in pre-decoders. The low level setup also includes the aging model.

The flowchart in Fig. 4 is explained below. The steps that are inside of the gray area requires a transistor-level simulation, and the green area corresponds to an RTL simulation. The rest of the steps are automated with scripts.

Step 1: Extract raw paths of the decoder from the decoder design. This step is executed once. (In particular, for the case-study design, 144 path files in transistor level are generated. Specifically, for each 3-to-8 pre-decoder, 24 activation and 24 deactivation paths exist.)

Step 2: Identify and save transistor values in the decoder for each address input with transistor level simulation with transistor level simulation. This step is also executed once and transistor-value pairs are recorded for each address input.

Step 3: Obtain nominal delay of each path with transistor level simulation.

Step 4: Obtain memory access traces by running RTL simulation.

Step 5: Calculate signal probabilities for each transistor.

Step 6: Add average signal values of transistors to each path file.

Step 7: Obtain delay of each path after aging by running transistor level simulation with aging model. Note, since the critical path may change after aging, we simulate all paths in this step.

Step 8: Calculate aging percentage using the results from Step 4 and 7.

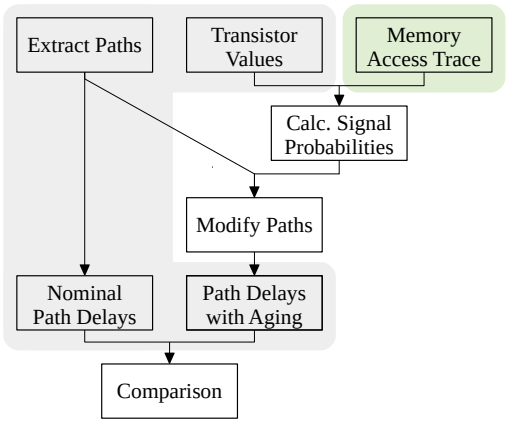
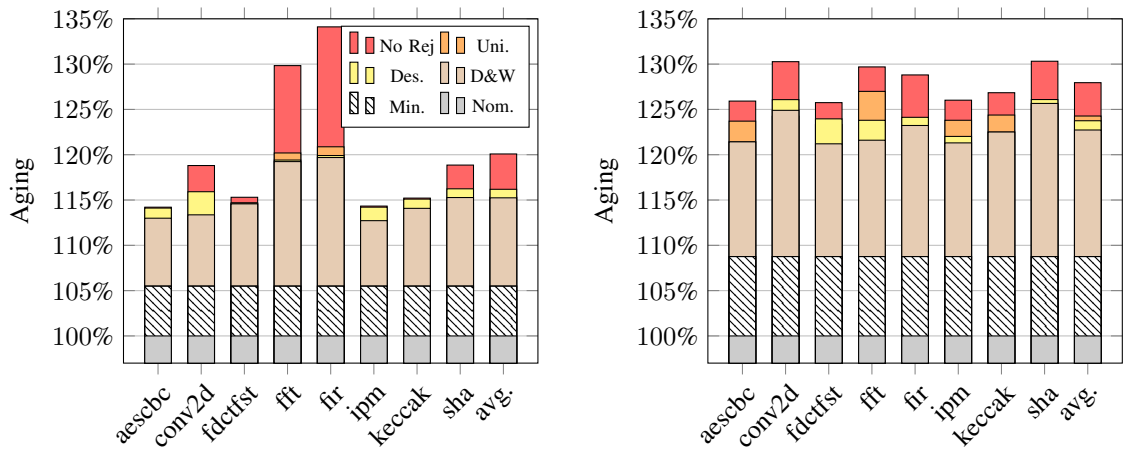


Fig. 4. The flow that is utilized to calculate aging in the decoder.

B. Rejuvenation Workload Generation

Here, we propose several rejuvenation workloads and provide a method to generate them.

- **Universal:** This rejuvenation workload sequentially selects all addresses for equal amount of clock cycles as in [17]. This is the simplest workload out of the three, and can be generated without any analysis of decoder design or memory access trace.
- **Design-Aware:** This rejuvenation workload takes path delays into account, and selects some of the addresses for more cycles than others to balance path delays. Fig. 3c shows the path delays of a pre-decoder after design aware workload run for the specified period (e.g., three years in the case study). To generate this



(a) AND-AND (b) NAND-NOR

Fig. 5. Comparison of eight workloads with different mitigation strategies (at 1% execution overhead and 3 years of aging).

workload several iterations of aging assessment has to be made. In the given example, 3-bit pre-decoder inputs can be controlled by 8 addresses. Initially all 8 addresses are selected by equal amount of time. After the first run, we take a note of the longest path and its input value. Then for the second run we increase cycles for address that corresponds to that input value. This process is repeated until all of the long paths reach the same amount of delay and no single path has a higher delay than others. At the end of this process we obtain n *weights* (constant values) for each of the n addresses. To achieve lowest aging, selection ratio of addresses (in cycles) must match the weights which are obtained for the design. It is important that, this rejuvenation workload is generated once for a given decoder design.

- **Design-&-Workload-Aware:** This rejuvenation workload uses the weights that are calculated for Design-Aware workload and combines it with the main functional workload's memory trace. Based on its memory trace, we calculate signal probability of each pre-decoder input, and generate a rejuvenation workload that balances it. For instance, most workloads use the last addresses of the decoder more frequently as those corresponds to the stack portion of the memory. We generate a workload such that those highly utilized addresses are selected less while the rejuvenation workload is running. This process yields the best rejuvenation for a given time overhead. However, it requires the knowledge of both the design and the main workload ahead of time.

C. On Aapplying the Rejuvenation Workloads

After deciding and generating the rejuvenation workload, it must be mixed to the main functional workload. In our mitigation scheme, the rejuvenation workload is added to a interrupt routine to be applied periodically. The routine is called with a timer interrupt and by adjusting its count limit, the routine can be called at the desired frequency and execution overhead. The routine can be placed in either *crt0* file, so that it can be compiled without modifying the main code, or it can be attached to the main functional workload.

IV. Experimental Results

A. Experimental Setup

Our experimental setup consists of two parts. The high-level part, which is used to generate memory access traces, is implemented with some modifications to Pulpino [21] open-source single core microcontroller project. The RISCY CPU in the project configured to implement RISC-V RV32F instruction set. We increased the observability of the memory, since we are also interested in memory state when the CPU is at sleep state. We used the benchmarks in the design repository, some of them originating from the MiBench library [22]. The unrelated benchmark statistics code and UART messages were removed to observe the workload only. The interrupt routine was implemented in C and the rejuvenation workload was implemented as an assembly code. The interrupt period is adjusted based on the length of both the rejuvenation and the main workloads to achieve desired execution overhead. At the end of RTL simulation, we obtain a memory trace file that contains memory state during the execution of the workload, i.e. read/write operations, the selected address and the corresponding clock cycle.

As mentioned in Section II, we consider two 9-to-512 decoder designs using 22nm PTM technology and the aging model [19] on the low-level setup. We used Spectre [23] to do SPICE simulations for extracting paths of the pre-decoders, obtaining transistor values for a given address input and at the end of the flow, to obtain path delays with or without aging. We also have a Python script to automate the flow. It takes memory trace as an input, calculates signal probabilities, adds the signal probability values to the transistors on a the paths and runs SPICE simulations for each path. We assume that simulations with the aging model are run at 125°C and a nominal supply voltage of 0.95V. The duration of the aging is 3 years except the last experiment, where simulations are run for 1 to 10 years with one year increments.

The assembly code for the universal rejuvenation workload is implemented manually. However, they are generated with a script for the other methods.

B. Performed Experiments

Using the setup above, the following experiments are performed:

1) Dependency on Functional Workload: We first investigate how much the functional workload affects the mitigation potential to see the best and the worst cases. On the later experiments, we use averages of all workloads or a single one. Fig. 5 shows all the functional/rejuvenation workload combinations for 3 years of aging at 1% execution overhead.

For AND-AND design there is much larger deviation from the average value. The main difference between the two designs is on the AND-AND design, lowest outputs of the pre-decoder has longer delay, and higher ones have lower delays. This is the opposite on the NAND-NOR design. Therefore, to mitigate aging with the Design-Aware rejuvenation, we allocate more clock cycles for the high address range of the memory with the AND-AND design. The *fir* and the *fft* workloads heavily utilize low address ranges of the memory but they rarely use the stack for function calls, hence the 3rd pre-decoder that connected to most significant three bits of the address signal stays at ”000” input for 95% of the time. Due to this imbalance, these two benchmarks have the highest aging on AND-AND design and the *aescbc* has already 60% less aging compared to *fir*. By adding 1% rejuvenation overhead we reduce this difference to ∼20%.

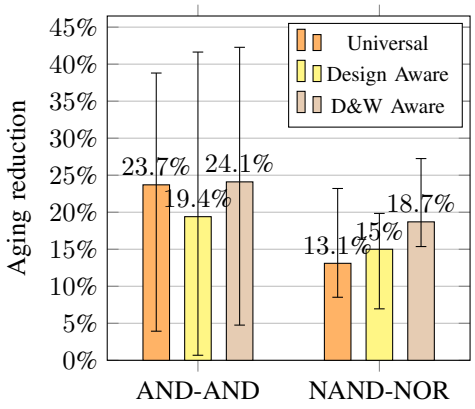


Fig. 6. Aging reduction of three rejuvenation workloads compared to no rejuvenation after 3 years.

2) Dependency on Rejuvenation Workload: Here, we look at the same data as the previous one, but we compare the impact of rejuvenation workloads.

Fig. 6 shows the average aging reduction for three rejuvenation workloads. The error bars show the results with best and worst case considering the eight main workloads. We achieved the best results with the Design-&-Workload-Aware rejuvenation, which yield to a greater benefit over the others on the average as well as best/worst cases. It has reduced aging by 42% in the best case and by 5% in the worst one. We did not observe a clear advantage with Design-Aware workload over the Universal.

3) Saturation of The Rejuvenation Effect: In this experiment we investigate the saturation of the rejuvenation effect as we increase the execution overhead. The Fig. 7 shows the average aging considering all main workloads versus the execution overhead. It is clear that we get the highest amount of benefit with the first few percentages of the rejuvenation execution overhead. This expected as the transistors suffer from extreme static BTI effect when the input signal probability is close to 1.0 or 0.0, but the effect quickly decreases (see Fig. 7). The crosses mark the lowest possible aging (when running only the Design-Aware workload).

4) Dependency on Target Lifetime: Lastly, we performed an experiment to estimate the amount of the system's lifetime extension by our mitigation method. We ran cases from 1 year of aging to 10 years with 1 year increments with and without Universal rejuvenation. The mitigation scheme applied assuming *fir* as the functional workload with 1% overhead. The Fig. 8 shows the amount of aging while

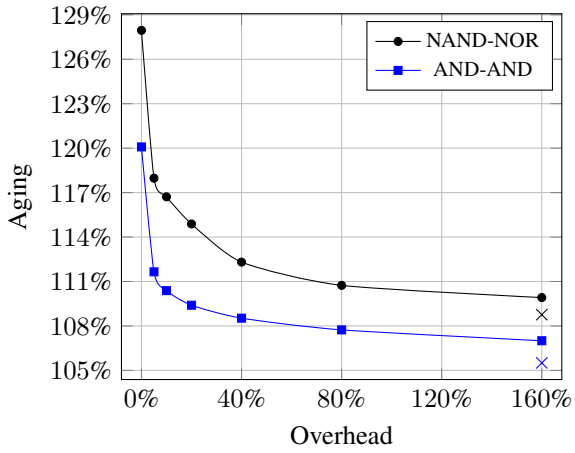


Fig. 7. Average aging in 3 years vs. execution overhead. The cross marks show the minimum aging possible.

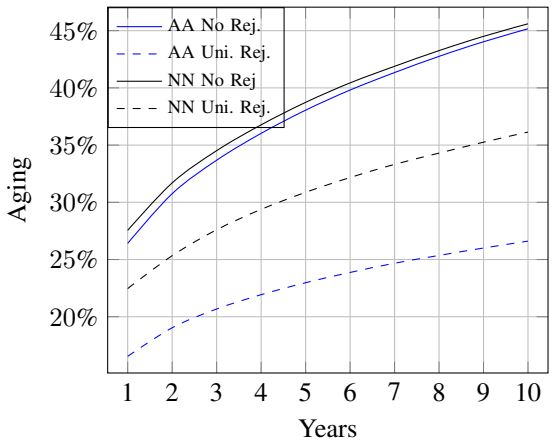


Fig. 8. Aging of the workload fir with and without the universal rejuvenation at 1% overhead, simulated for 1 to 10 years range.

duration of aging increased. Blue and black lines represents AND-AND and NAND-NOR decoder design respectively. The results demonstrate that, for AND-AND design, the same amount of aging accumulated after 9 years of aging with mitigation and 1 year of aging without mitigation. It is possible to observe a similar benefit with NAND-NOR design, where we see the same amount of aging at 9 and 3 years with and without mitigation respectively. the actual extended lifetime strongly depends on the initially allocated slack, but generally it may be up to several times as in the discussed case-study.

V. Discussion

In this section, we make following points on the performed experiments.

A. *Improved Reliability* Our experimental results show that we achieved 21% aging reduction on average, up to 42% in some cases. We also reflected this on the life time of decoder and showed that it is possible to extend duration of its reliable operation by three times or more.

B. *Cost of the scheme* Since our mitigation scheme implemented as a software addition to the original code, we do not need a new hardware design, and therefore there is no area or path delay overhead. However, the scheme allocates CPU time and has an execution time overhead.

In our experiments, we set this overhead to only 1% and achieved significant benefits. In addition, we showed that it is possible to gain most of the rejuvenation benefit in first few percentages of execution overhead. Due to the execution overhead, our scheme has also a power overhead.

Our scheme is also transparent from the software point of view, and easy to integrate. Although a new compilation of the software necessary, it is possible to add it to the compiler and keep the existing programs the same.

C. *Choosing the rejuvenation workload* In our experiments the NAND-NOR decoder design reached 43% more aging reduction with Design and Workload Aware rejuvenation workload compared to the Universal one. However, we did not observed this on the other decoder design. This is mostly depends on the pre-decoder paths, since the AND-AND had long paths connected to lowest bits, and the opposite with the NAND-NOR design.

Choosing the rejuvenation workload depends on the system and also the workloads that the CPU needs to execute. We observed that most workloads use first and last part of the memory, and they can benefit from a workload aware rejuvenation. In most systems however, it will not be possible to use a workload aware method if an operating system is present, or on a multi-core system. In such systems, Design-Aware or Universal methods may be applied.

D. *Potential Improvements* In our experiments, we assume that the CPU busy 100% of the time. In reality, there can be idle periods that can be used to run our mitigation scheme. This may also reduce or eliminate the execution overhead. We did not explore this, since it is too dependant on the system, but presented the data so that the benefit can be calculated. Our results showed that we can gain almost half of maximum possible aging reduction with a 5% of overhead and it quickly saturates as we increase the overhead percentage.

VI. Conclusion

In this work, we presented a low-cost aging mitigation scheme, which can be applied to existing hardware to mitigate aging on memory address decoder logic. We mitigate the BTI effect on critical transistors by applying a rejuvenation workload to the memory. Such an auxiliary workload is executed periodically to rejuvenate transistors that are located on critical paths of the address decoder. Second, we analyze workloads' efficiency to optimize the mitigation scheme. Experimental results performed with realistic benchmarks demonstrate several-times lifetime extension with a negligible execution overhead.

Acknowledgments

This work was partly supported by the European Union through the European Social Fund in the frames of the "Information and Communication Technologies (ICT) programme" ("ITA-IoIT" topic), by the Estonian Research Council grant PUT PRG1467 "CRASHLESS", and by the RESCUE funded from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowaska-Curie grant agreement No. 722325.

References

[1] S. Hamdioui, D. Gizopoulos, G. Guido, M. Nicolaidis, A. Grasset, and P. Bonnot, "Reliability challenges of real-time systems in forthcoming technology nodes," in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 129–134.

[2] 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*, vol. 50, no. 4.5, pp. 433–449, 2006.

[3] K. K. Kim, W. Wang, and K. Choi, "On-chip aging sensor circuits for reliable nanometer mosfet digital circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 10, pp. 798–802, 2010.

[4] W. Needham, C. Prunty, and Eng Hong Yeoh, "High volume microprocessor test escapes, an analysis of defects our tests are missing," in *Proceedings International Test Conference 1998 (IEEE Cat. No.98CH36270)*, 1998, pp. 25–34.

[5] A. J. van de Goor, S. Hamdioui, and R. Wadsworth, "Detecting faults in the peripheral circuits and an evaluation of sram tests," in *2004 International Conferce on Test*, 2004, pp. 114–123.

[6] S. V. Kumar, K. H. Kim, and S. S. Sapatnekar, "Impact of nbti on sram read stability and design for reliability," in *7th International Symposium on Quality Electronic Design (ISQED'06)*, 2006, pp. 6 pp.–218.

[7] I. Agbo, S. Khan, and S. Hamdioui, "Bti impact on sram sense amplifier," in *2013 8th IEEE Design and Test Symposium*, 2013, pp. 1–6.

[8] S. Khan, M. Taouil, S. Hamdioui, H. Kukner, P. Raghavan, and F. Catthoor, "Impact of partial resistive defects and bias temperature instability on sram decoder reliablity," in *2013 8th IEEE Design and Test Symposium*, 2013, pp. 1–6.

[9] S. Khan, I. Agbo, S. Hamdioui, H. Kukner, B. Kaczer, P. Raghavan, and F. Catthoor, "Bias temperature instability analysis of finfet based sram cells," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–6.

[10] I. Agbo, M. Taouil, S. Hamdioui, H. Kukner, P. Weckx, P. Raghavan, and F. Catthoor, "Integral impact of bti and voltage temperature variation on sram sense amplifier," in *2015 IEEE 33rd VLSI Test Symposium (VTS)*, 2015, pp. 1–6.

[11] J. Ding, D. Reid, P. Asenov, C. Millar, and A. Asenov, "Influence of transistors with bti-induced aging on sram write performance," *IEEE Transactions on Electron Devices*, vol. 62, no. 10, pp. 3133–3138, 2015.

[12] I. Agbo, M. Taouil, D. Kraak, S. Hamdioui, H. Kükner, P. Weckx, P. Raghavan, and F. Catthoor, "Integral impact of bti, pvt variation, and workload on sram sense amplifier," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1444–1454, 2017.

[13] J. Kinseher, L. Heiß, and I. Polian, "Analyzing the effects of peripheral circuit aging of embedded sram architectures," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 852–857.

[14] D. Kraak, M. Taouil, I. Agbo, S. Hamdioui, P. Weckx, S. Cosemans, and F. Catthoor, "Parametric and functional degradation analysis of complete 14-nm finfet sram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1308–1321, 2019.

[15] ——, "Impact and mitigation of sense amplifier aging degradation using realistic workloads," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 12, pp. 3464–3472, 2017.

[16] I. O. Agbo, M. Taouil, and S. Hamdioui, "Reliability modeling and mitigation for embedded memories," in *2019 IEEE International Test Conference (ITC)*, 2019, pp. 1–10.

[17] D. H. P. Kraak, C. C. Gürsoy, I. O. Agbo, M. Taouil, M. Jenihhin, J. Raik, and S. Hamdioui, "Software-based mitigation for memory address decoder aging," in *2019 IEEE Latin American Test Symposium (LATS)*, 2019, pp. 1–6.

[18] D. Kraak, I. Agbo, M. Taouil, S. Hamdioui, P. Weckx, S. Cosemans, and F. Catthoor, "Hardware-based aging mitigation scheme for memory address decoder," in *2019 IEEE European Test Symposium (ETS)*, 2019, pp. 1–6.

[19] G. Rzepa, J. Franco, B. O'Sullivan, A. Subirats, M. Simicic, G. Hellings, P. Weckx, M. Jech, T. Knobloch, M. Waltl *et al.*, "Comphy—a compact-physics framework for unified modeling of bti," *Microelectronics Reliability*, vol. 85, pp. 49–65, 2018.

[20] M. Jenihhin, G. Squillero, T. S. Copetti, V. Tihhomirov, S. Kostin, M. Gaudesi, F. Vargas, J. Raik, M. Sonza Reorda, L. Bolzani Poehls, R. Ubar, and G. C. Medeiros, "Identification and rejuvenation of nbti-critical logic paths in nanoscale circuits," *Journal of Electronic Testing*, vol. 32, no. 3, pp. 273–289, Jun 2016.

[21] A. Traber, F. Zaruba, S. Stucki, A. Pullini, G. Haugou, E. Flamand, F. K. Gurkaynak, and L. Benini, "Pulpino: A small single-core risc-v soc," in *3rd RISCV Workshop*, 2016.

[22] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4 (Cat. No. 01EX538)*. IEEE, 2001, pp. 3–14.

[23] K. Kundert, *The designer's guide to spice and Spectre®*. Springer Science & Business Media, 2006.