Software-Based Mitigation for Memory Address Decoder Aging

D.H.P. Kraak*, C.C. Gürsoy†, I.O. Agbo*, M. Taouil*, M. Jenihhin†, J. Raik†, S. Hamdioui*

*Delft University of Technology, THE NETHERLANDS
{D.H.P.Kraak, I.O.Agbo, M.Taouil, S.Hamdioui}@tudelft.nl

†Tallinn University of Technology, ESTONIA
{Cem, Maksim, Jaan}@ati.ttu.ee

***Abstract*—Integrated circuits typically contain design margins to compensate for aging. As aging impact increases with technology scaling, bigger margins are necessary to achieve the desired reliability. However, these increased margins lead to a reduced performance and lower yield. Alternatively, mitigation schemes can be deployed to reduce the aging. This paper proposes a *software-based* method to mitigate the aging of the memory's address decoder logic due to Bias Temperature Instability. The method is based on periodically applying a rejuvenation application on top of a user application. The goal of the rejuvenation application is to recover aged transistors of the critical paths of the address decoder. The experimental results show that the proposed method significantly reduces aging in cases when applications consist of memory access patterns that result in an unbalanced stress in the address decoder logic. In particular, it reduces the degradation of the address decoder's setup delay by up to 43% with an execution overhead of only 1%.**

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

I. Introduction

The aggressive downscaling of CMOS technology has been the main driver of the improvements in the performance and functionality of Integrated Circuits (ICs) over the past decades. However, due to several challenges, the rate of downscaling and its benefits have started to decrease [1]–[3]. One of the challenges of downscaling is that it worsens the reliability of ICs due to increased time-dependent variability. Time-dependent variability consists of variations that occur during the operational lifetime of the IC. They include environmental variations, such as supply voltage and temperature fluctuations, and aging variations due to, for example, Bias Temperature Instability (BTI) [4]. In order to guarantee a high quality and reliable product at optimal design, it is crucial to assess this variability and provide appropriate mitigation schemes. In this work, we investigate an aging mitigation scheme for the address decoder logic of memories. Several works report that *delay faults* in the decoder logic are a major contributor to the total amount of customer returns [5], [6]. These delay faults, for instance due to aging, may cause a delayed selection of the requested address, the selection of a wrong address, or the selection of multiple addresses, resulting in *read failures* or *write failures*. Hence, understanding the impact of aging on the address decoder logic and providing appropriate mitigation schemes is an important part of designing reliable memories.

This work was supported through the projects TRACE (CATRENE, Grant 16ES0488K-16ES0502, 16ES0737), PRYSTINE (ECSEL, Grant 783190), and RESCUE (EU Horizon 2020, Grant 722325).

Previous works on SRAM aging mitigation have mainly focused on the memory cells; examples are [7]–[11]. Most of these works aim at balancing the probability of writing zeroes and ones to the memory cells, as this minimizes the degradation due to BTI. The proposed schemes are either *hardware-based* or *software-based*. The hardware-based schemes add additional hardware that encodes the data written to the memory cells such that it has a more equal probability of writing either a zero or a one [7]–[9]. The software-based schemes extend the memory's lifetime through smart data allocation at the software-level in order to distribute the stress on the memory cells [10], [11]. In contrast to the memory cells, mitigation schemes for the memory's peripheral circuitry (e.g., sense amplifier, write driver, address decoders) have received significantly less attention. To the best of our knowledge, only one mitigation scheme has been proposed, which targets the sense amplifier [12]. Mitigation schemes for the other peripheral circuitry, such as the address decoder, timing circuit, and write driver, have not yet been researched.

In this work, we propose *software-based* mitigation to reduce the aging of the memory's address decoder. This is achieved by periodically running a *rejuvenation* application on top of a user application. The rejuvenation application mitigates aging by recovering transistors in the critical paths. It is worth noting that the rejuvenation approach has already been successfully applied to mitigate aging in the datapaths of CPUs [13]. Nevertheless, its potential has not yet been investigated for other circuits, such as memories. The contributions of this work are as follows:

1) It proposes *software-based* mitigation to reduce aging in the memory's address decoder logic. It specifically achieves this by periodically running a *rejuvenation* application.
2) It evaluates, as a case study, the effectiveness of rejuvenation to mitigate aging of the data memory's address decoder logic due to BTI.
3) It investigates the impact of different user applications, rejuvenation applications, and execution overheads.

The rest of this paper is organized as follows: Section II provides the background. Section III presents the proposed methodology. Section IV discusses the experimental setup and performed experiments. Section V presents the obtained results. Section VI provides a brief discussion. Finally, Section VII concludes this work.

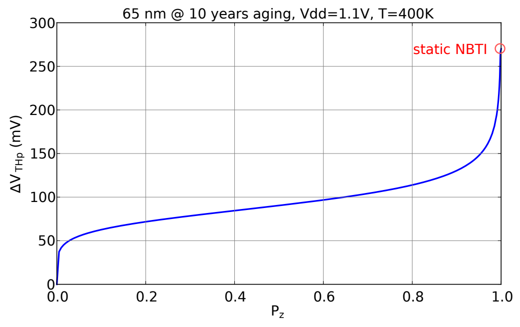


Fig. 1: Threshold voltage shift ΔV_{THp} as a mathematically convenient function of signal probability P_z.

II. Background

This section discusses the used method to model NBTI-induced delays and, subsequently, the address decoder and its metric.

A. Hierarchical Modelling of NBTI-Induced Delays

Bias Temperature Instability (BTI) is an aging mechanism that takes place inside the MOS transistors and causes an increment in the threshold voltage V_{TH} of transistors. It happens under negative gate stress for PMOS transistors, referred to as Negative BTI (NBTI) and under positive gate stress for NMOS transistors, referred to as Positive BTI (PBTI). NBTI is considered to be the most important aging mechanism in deeply scaled CMOS technologies [4]. Therefore, to model aging, we focus on modelling NBTI in this work. The approach for this is discussed next.

In [13], an approach for fast, yet accurate modeling of NBTI-induced delays at the gate level is proposed that can be summarized by the following steps:

- *Step A* (at the transistor level): Obtain a technology and environment dependent curve of the threshold voltage shift as a function of the transistor's gate input signal probability $\Delta V_{THp}(P_z)$.
- *Step B* (at the gate level): Obtain technology and environment dependent curves of the gate delay degradation as a function of the threshold voltage shift $\Delta t(\Delta V_{THp})$ for each gate type in the netlist (e.g. INV, 2NAND, 2NOR). It implies one-time SPICE electrical simulations of the related logic cells.
- *Step C* (at logic paths): Identify NBTI-critical paths at the gate level. This step involves the simulation of application-specific gate signal probabilities and static timing analysis for the nominal and NBTI-induced path delays.

In the NBTI effect analysis, we rely on a reaction-diffusion (R-D) based predictive model for dynamic NBTI presented in [4], [14]. This model predicts the long term threshold voltage V_{THp} degradation due to NBTI at a time $t > 1,000s$ at high frequencies [4]. It captures the dependence of NBTI on a gate input signal probability P_z (probability that pMOS transistor is under stress) in addition to its dependence on other key process and design parameters as presented in [4].

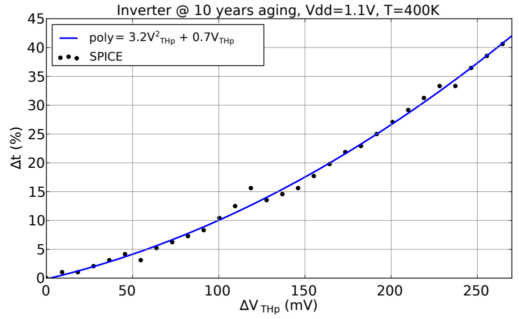


Fig. 2: Dependence of the gate output delay Δt on the ΔV_{THp} in an INV gate. Results of SPICE simulations (black dots) and of the mathematically convenient function (blue curve).

The values of the involved technology and environmental parameters can be summarized by a parameter γ in the following form:

$$|\Delta V_{THp}| = \gamma(\frac{P_z}{1-P_z})^n \tag{1}$$

Note that Equation 1 is valid only for dynamic stress, as ΔV_{THp} becomes infinite when P_z reaches the value 1. Therefore, the upper limit of ΔV_{THp} is defined by static NBTI models [4]. Equation 1 represents a convenient mathematical function of the threshold voltage V_{THp} degradation dependence on the signal probability for the gate input signal $P_z(x_i)$ of a pMOS transistor. In the equation, $n = 1/6$ represents the variety of the dominant diffusion species (H or H_2) expressed by the time exponent parameter and $\gamma = 0.0904$ represents a parameter that incorporates the selected technology and environmental variables. In Fig. 1, the corresponding dependence is illustrated for PTM 65 nm technology [15] after 10 years of NBTI-induced degradation at constant temperature $T = 400K$ with supply voltage $V_{DD} = 1.1V$. The calculated value of ΔV_{THp} for static NBTI is 0.27V. The model allows fast estimation of NBTI-induced V_{THp} shifts.

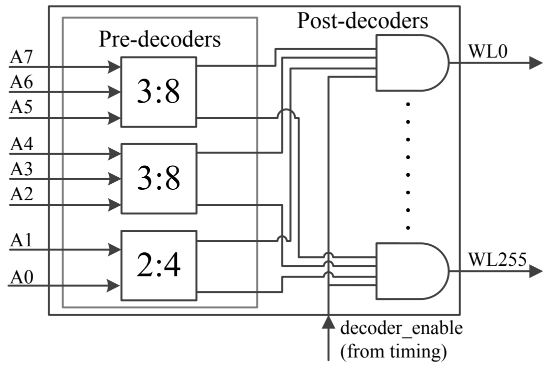
A set of SPICE simulations for each logic cell is used to create a polynomial curve to model the gate delay degradation (see Fig. 2):

$$\Delta t_{gate} = \lambda * \Delta V_{THp}(X_i) + \mu * (\Delta V_{THp}(X_i))^2 \tag{2}$$

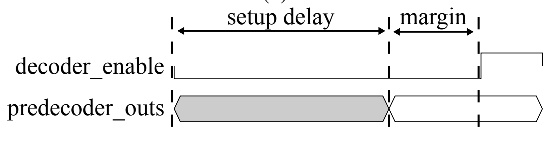
Here, Δt_{gate} is the gate output delay increase (in percentage) compared to the nominal gate delay, $\Delta V_{THp}(x_i)$ is the change of V_{THp} for the stressed pMOS transistor at the gate input x_i, while λ and μ are technology dependent constants. For example, in our experiments λ and μ are set to 0.7 and 3.2 for the INV gate.

In case a logic gate consists of multiple cascaded pMOS transistors, both their physical location relative to the output node and their $1 \rightarrow 0$ input transition impact the gate delay degradation. Each combination of gate input values is modelled by different values of the constants λ and μ in Equation 2.

The path delays are calculated by processing the gate-level netlist, gate by gate, from inputs to outputs, both with and without aging.



(a)



(b)

Fig. 3: (a) Schematic of the wordline decoder. (b) Setup delay metric of the wordline decoder.

B. Address Decoder

Memories typically contain logic to access certain rows in the memory cell array, referred to as the *wordline decoder*, and logic to select the respective column, referred to as the *column decoder*. In general, the wordline decoder is more critical, as memories typically have more rows than (selectable) columns. Therefore, for this study, we limit our analysis to the wordline decoder.

In this work, we consider a 8-to-256 wordline decoder. Fig. 3a shows a simplified diagram of its design. It is implemented using a hierarchical architecture that consists of a *pre-decoder* stage and a *post-decoder* stage. The pre-decoder stage is implemented using two 3-to-8 decoders and one 2-to-4 decoder. The post-decoder stage consists of 256 post-decoders that are implemented using AND-gates. Their output is used to activate one of the wordlines of the memory cell array. Each set of inputs to these AND-gates is a unique combination of the outputs from the pre-decoders. Hence, only one wordline can be activated at a time. In addition, an 'enable_decoder' signal is connected to the input of these AND-gates. The period during which the wordline is activated can be controlled using this signal. In general, it is driven by a timing circuit in the memory.

An important metric of the wordline decoder is its *setup delay*. It is illustrated in Fig. 3b for our wordline decoder design; it is defined as the maximum propagation delay of the pre-decoders and, thus, it equals the critical path of the pre-decoder logic. In addition, the *margin* of the decoder is shown in Fig. 3b. It is defined as the time between the pre-decoder outputs being ready and the activation of the decoder through its decoder_enable port. In case the setup delay is too high (the margin is negative), wrong signals are applied to the inputs of the post-decoder's AND-gates. This may result in a delayed activation of the wordline, the selection of a wrong wordline, or the selection of multiple wordlines, potentially causing a *read failure* or a *write failure*. Hence, a lower setup delay results in a more reliable memory.

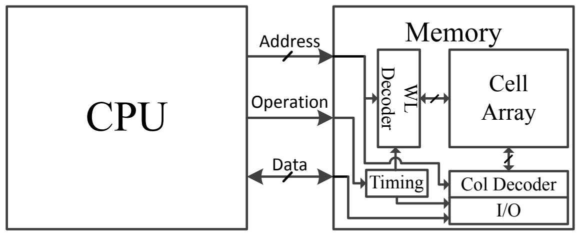


Fig. 4: Schematic of a CPU and memory.

III. Proposed Methodology

In this section, we first present the concept of mitigating the degradation of the memory's address decoder through additional software programs and, subsequently, our approach to evaluate the effectiveness of the proposed rejuvenation method.

A. Concept

Fig. 4 shows a simplified schematic of a CPU and memory and the bus through which they are connected. During the execution of any application, CPUs fetch instructions from the memory and they also execute memory-related instructions. In both cases, this triggers read/write operations to/from the memory. These memory operations determine the workload of the memory and, thus, they strongly impact its aging [16], [17]. The idea proposed in this paper is to optimize the memory instructions issued by applications in order to mitigate the memory's aging. *Thus, the aging of the memory is mitigated at the software level.*

This software-based memory mitigation can be applied as follows: a *rejuvenation* application is periodically run at a certain execution overhead of an original *user* application [13]. This rejuvenation application mitigates the memory aging by performing instructions that put critical transistors in the memory into relaxation. The rejuvenation application can, for instance, be applied during idle times of the user application.

B. Approach

Fig. 5 illustrates the used flow to evaluate the potential of rejuvenating the address decoder. As can be seen, it consists of four steps, which are described next.

1. Simulate user application: In the first step, the signal probabilities of the inputs of the logic gates of the memory address decoder are determined for a user-application. This is achieved by simulating the execution of a user application on a register-transfer (RTL) level CPU and a gate-level netlist of the memory's address decoder logic. This gate-level netlist of the address decoder is connected to the address bus of the RTL CPU; this ensures that it receives the correct input stimuli, which are necessary to determine its workload. Then, during simulation, the signal probabilities of the logic gates of the address decoder are measured, which will be used in later steps.

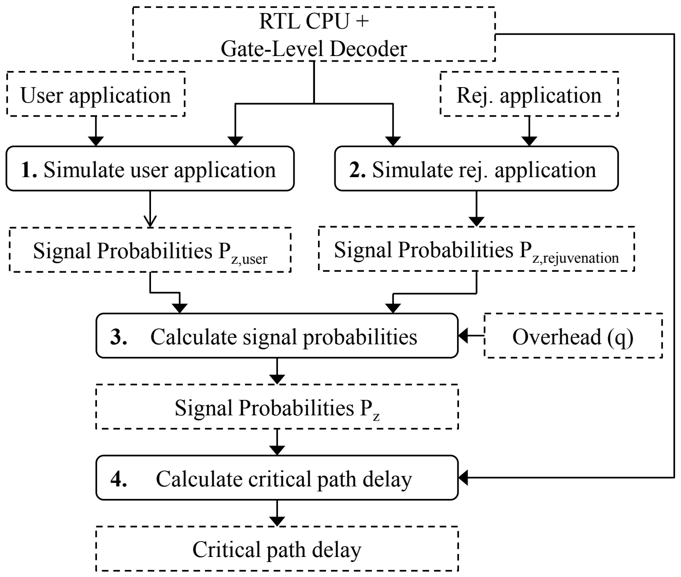

RTL CPU + Gate-Level Decoder
User application
Rej. application
1. Simulate user application
2. Simulate rej. application
Signal Probabilities $P_{z,user}$
Signal Probabilities $P_{z,rejuvenation}$
3. Calculate signal probabilities
Overhead (q)
Signal Probabilities P_z
4. Calculate critical path delay
Critical path delay

Fig. 5: Used flow to evaluate rejuvenation of the memory address decoder logic.

2. Simulate rejuvenation application: In this step, the signal probabilities of the inputs of the logic gates of the address decoder are determined for the rejuvenation application. This step is similar to the first step. The only difference is that, here, the execution of a rejuvenation application is simulated instead of a user application.

3. Calculate signal probabilities: In this step, the signal probabilities of the previous two steps are combined, i.e., the signal probabilities are calculated for the case where the rejuvenation application is executed after the user application. A new signal probability (say P_z) is derived using the following equation:

$$P_z = P_{z,user} \cdot (1 - q) + P_{z,rejuvenation} \cdot q \qquad (3)$$

Here, $P_{z,user}$ is the signal probability of a gate input signal when the user application is executed while $P_{z,rejuvenation}$ when the rejuvenation application is executed, and q the execution overhead of the rejuvenation application. This formula is applied for all the gate input signals of the address decoder. It is worth noting that this approach ignores the overhead of switching between the user and rejuvenation applications. However, this switching overhead is very small and, thus, has a negligible impact.

4. Calculate critical path delay: In the final step, the critical path delay of the aged address decoder is calculated. This is achieved by analyzing the gate-level description of the decoder using the hierarchical NBTI modelling method discussed in Section II. Here, the signal probabilities from the previous step are used as the workload for the aging.

IV. Experimental Setup

In order to evaluate the potential of rejuvenation for address decoders, we perform a case study on the wordline decoder of the data memory of the open-source Plasma processor [18] using the flow from Fig. 5. The Plasma is an RTL-description of a 32-bit CPU with a MIPS instruction set. The key configuration parameters of the Plasma setup are listed in Table I. In addition, we implemented the wordline decoder of Fig. 3a at the gate-level with Inverters and 2-input NAND gates using $65\,\text{nm}$ PTM technology [15]. Buffers were added at the outputs of the pre-decoders to compensate for their high fan-out to the post-decoders. This gate-level decoder is connected to the most significant address bits of the Plasma's data memory. The execution of different user- and rejuvenation applications is simulated using ZamiaCAD [19]. Using ZamiaCAD's scripting support, the signal probabilities of all gate inputs are measured.

TABLE I: Configuration of Simulation Setup.

Processor	MIPS Plasma @ $200\,\text{MHz}$
Instruction Memory	$8\,\text{kB}$ (8192 addresses)
Data Memory	$8\,\text{kB}$ (8192 addresses)

The user- and rejuvenation-applications are implemented as assembly programs. They consist of read operations at different memory addresses. Note that it is sufficient to include only read operations due to the fact that we are interested in the impact on the decoder's setup delay. The setup delay corresponds to the delay of the pre-decoders, whose workload is determined by the requested addresses (see Fig 3a). Hence, it is sufficient to simulate the relative amount of accesses between the addresses to correctly model the workload of the address decoder. Two applications for rejuvenation are implemented: a *universal* rejuvenation application and an *unused addresses* rejuvenation application. The universal rejuvenation application iterates over all memory addresses, while the unused addresses application only iterates over the addresses that are not accessed by the user application. Both rejuvenation applications ensure that all possible address combinations are applied to the decoder and, thus, all PMOS transistors recover from aging for a certain amount of time.

Using this setup, we perform the following experiments:

1) **Impact of Workload:** We investigate the effect of rejuvenation for several different user workloads. Each user workload accesses a different range of addresses. We assume that all addresses within this range are accessed an equal amount of times. These workloads mimic different applications with varying memory access behavior. We include both the *universal* method, as well as the *unused addresses* method for the rejuvenation sequences. We keep the execution overhead (i.e., q in Equation 3) at a fixed number of 1%.
2) **Impact of Execution Overhead:** We investigate the effect of rejuvenation for several execution overheads for the universal rejuvenation sequence. The used execution overheads are 0.1%, 1%, and 10%. For this experiment we use the same user workloads as in the first experiment.

For each experiment, we measure the *decoder's setup delay* while assuming it has been aged for 10 years at $400\,\text{K}$ and a nominal supply voltage of $1.1\,\text{V}$. Next, it is compared to the setup delay at time-zero (i.e., no aging).

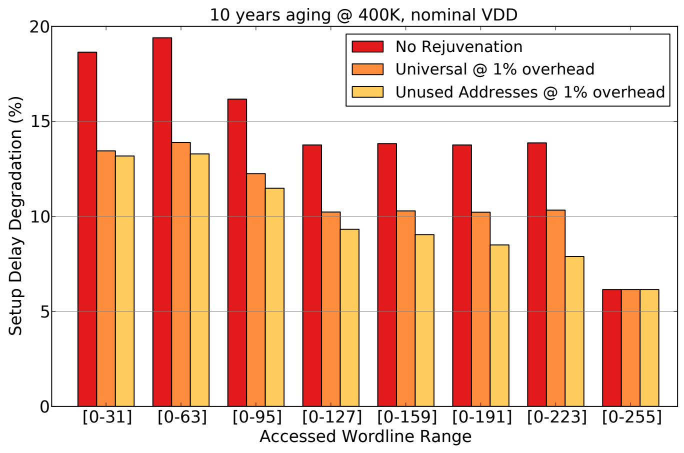


Fig. 6: Degradation and rejuvenation of the wordline decoder.

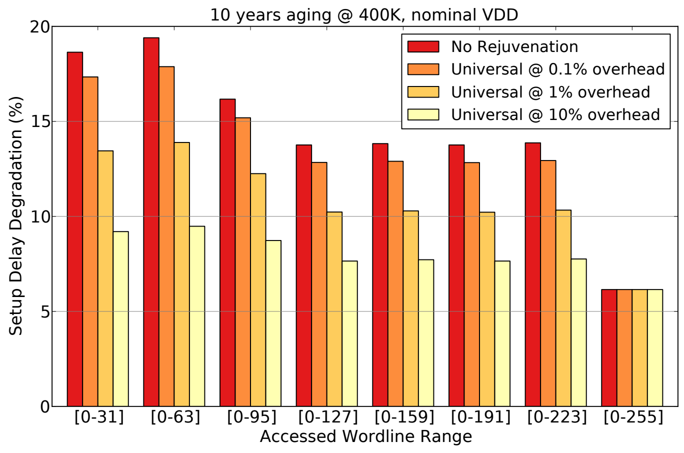


Fig. 7: Impact of rejuvenation overhead on wordline decoder.

V. Experimental Results

A. *Impact of Workload*

Fig. 6 shows the degradation of the setup delay of the wordline decoder. Along the x-axis, different user workloads are shown. They are modelled by different ranges of accessed wordlines. For example, range $[0-31]$ means that the user application only accesses the first 32 wordlines of the memory (first 32 rows of the memory cell array). Hence, it mimics an application that only uses a small portion of the available memory. For each range, it is assumed that the wordlines within it are accessed an equal amount of times. In the figure, the degradation is shown for the baseline case without rejuvenation and for cases where the *universal* and *unused addresses* rejuvenation sequences are applied with a 1% overhead. The following observations can be made from the figure:

- The degradation of the decoder's setup delay strongly depends on the workload. In general, the degradation is higher for workloads that access smaller address ranges and it becomes lower as the workload accesses bigger address ranges. For instance, the degradation is up to 18.64% for the user workload with the range $[0-31]$, while it is only 6.15% for the user workload with the range $[0-255]$. The workloads with a smaller access range give a higher degradation due to the fact that a lot of the logic gates will be stuck at the same value. Hence, these gates experience a high stress. The lower the accessed range, the more gates will experience a high stress and, therefore, there is a higher probability that gates that contribute to the critical path are affected.
- Applying rejuvenation results in a significantly lower degradation in most cases. For example, the degradation is 13.87% without rejuvenation for the user workload with range [0-223], while it is only 7.89% in case the unused addresses sequence is applied; the relative degradation is up to 43% lower at only a 1% run-time overhead. *This shows the potential of mitigating the degradation of the memory's address decoder by applying low overhead rejuvenation sequences.* The rejuvenation is effective mostly due to static NBTI stress in many paths. Static stress results in an accelerated increase of the threshold voltage (as shown in Fig. 1). Hence, putting these transistors into relaxation for a short period results already in a significantly lower degradation and, therefore, the degradation of the critical path delay decreases accordingly. The only case in which the rejuvenation sequences have no impact is for the user workload with address range [0-255]. This is due to the fact that the workload already accesses all memory addresses and, therefore, the decoder's paths are stressed evenly.
- The unused addresses sequence mitigates the degradation more than the universal sequence. For example, the degradation is 12.22% for the unused addresses sequence for the user workload with range [0-127], while it is 12.84% for the universal sequence. The unused addresses sequence has a lower degradation due to the fact that it only accesses addresses that are not accessed by the user-workload. Therefore, the transistors that are under a high stress, are put into relaxation for a longer amount of time compared to the universal mitigation method.
- The benefit of the unused addresses mitigation sequence over the universal sequence increases as the user-workload accesses a wider range of addresses. For instance, the degradation is ∼5% lower for the unused addresses sequence as compared to the universal sequence in case of the user workload with range [0-31], while it is ∼24% lower in case of the user workload with the range [0-223]. This is due to the fact that when the unused addresses sequence needs to access a smaller range of addresses, these addresses can be selected for a longer period of time (at the same execution overhead). Therefore, the critical transistors are put into relaxation for a longer period. A drawback of the unused addresses sequence is, however, that it must be user application-aware; it should know which addresses are accessed by the user application.

B. *Impact of Execution Overhead*

Fig. 7 shows the degradation of the setup delay of the wordline decoder for the unused addresses rejuvenation sequence at different execution overheads. Once again, different user workloads modelled by different ranges of accessed wordlines, are shown along the x-axis. The figure reveals that the degradation of the address decoder is lower in case mitigation is applied at higher overheads. For instance, the degradation is 17.34% for a 0.1% overhead for the user workload with the range [0-31], while it is 13.45% and 9.2% for overheads of 1% and 10%, respectively. The higher rejuvenation overheads result in a lower degradation due to the fact that the critical transistors are put into relaxation for a longer period. Therefore, the degradation of the critical path delay is lower.

VI. Discussion

Based on this study, we conclude the following:

Improved Reliability: our experimental results reveal that the application of rejuvenation sequences significantly reduces the degradation of the address decoder. In the best case the degradation of the decoder's setup delay was reduced with up to 43% at only a 1% execution overhead. Rejuvenation of memory's address decoder logic gives the biggest benefits in case the address decoder has an unbalanced workload and only a limited number of rows is accessed. This is due to the fact that unbalanced workloads result in a static stress for transistors along critical paths, which results in accelerated aging. Hence, putting these transistors in recovery for a short period gives a significantly lower degradation. This reduced degradation of the address decoder leads to a memory with a *higher reliability* and a *prolonged lifetime*.

Rejuvenation application: optimized rejuvenation applications, that take into account the user application, give a higher mitigation. The reason for this is that they are able to put the critical transistors into recovery for a longer period. This is why the unused addresses rejuvenation sequence resulted in a lower degradation (up to 24%) than the universal sequence. Hence, if it is possible to profile the user application, it is beneficial to deploy dedicated rejuvenation applications.

Costs: finally, the costs in terms of area, power, and performance should be evaluated for our proposed mitigation scheme. First, our proposed mitigation scheme does comes at no area overhead due to the fact that it is fully software-based. On the other hand, our proposed scheme results in a power overhead due to the fact that a rejuvenation application is executed periodically. The evaluated rejuvenation applications are currently based on performing read operations. Such operations typically have a relatively high power consumption due to the (dis)charging of the memory's bitlines [20]. The address decoder's workload is, however, only determined by the applied address stimuli. Hence, to reduce the power consumption, a custom instruction could be implemented that only updates the address register.

Finally, our scheme also comes at a performance penalty due to the fact that rejuvenation applications are applied at a certain execution overhead. However, the results showed that significant benefits are achieved already at only 1% execution overhead. In addition, it is likely that the execution overhead can be masked in a lot of cases by scheduling the rejuvenation during idle times of user applications. In this case, the performance penalty becomes negligible.

Therefore, it can be concluded that the proposed scheme comes at zero to low overheads.

VII. Conclusions

This work clearly demonstrated the potential of periodically running rejuvenation sequences to mitigate the degradation of the memory's address decoder. These rejuvenation sequences are able to improve the reliability and increase the lifetime at low overheads. They are especially interesting for cutting-edge technology, which suffers from reduced lifetime. A possible future direction of this work is to investigate the potential of rejuvenation for other memory components.

References

[1] J. Srinivasan, S.V. Adve *et al.*, "The impact of technology scaling on lifetime reliability," in *International Conference on Dependable Systems and Networks, 2004*, June 2004, pp. 177–186.

[2] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov 2005.

[3] S. Hamdioui, D. Gizopoulos *et al.*, "Reliability challenges of real-time systems in forthcoming technology nodes," in *DATE*, March 2013, pp. 129–134.

[4] S. Bhardwaj, W. Wang *et al.*, "Predictive modeling of the nbti effect for reliable design," in *IEEE Custom Integrated Circuits Conference 2006*, Sept 2006, pp. 189–192.

[5] W. Needham, C. Prunty, and E.H. Yeoh, "High volume microprocessor test escapes, an analysis of defects our tests are missing," in *Proceedings International Test Conference 1998*, Oct 1998, pp. 25–34.

[6] 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 Conference on Test*, Oct 2004, pp. 114–123.

[7] 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)*, March 2006, pp. 6 pp.–218.

[8] A. Gebregiorgis, M. Ebrahimi *et al.*, "Aging mitigation in memory arrays using self-controlled bit-flipping technique," in *The 20th Asia and South Pacific Design Automation Conference*, Jan 2015, pp. 231–236.

[9] A. Valero, N. Miralaei *et al.*, "On microarchitectural mechanisms for cache wearout reduction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 857–871, March 2017.

[10] C. Ferri, D. Papagiannopoulou *et al.*, "Nbti-aware data allocation strategies for scratchpad memory based embedded systems," in *2011 12th Latin American Test Workshop (LATW)*, March 2011, pp. 1–6.

[11] D. Papagiannopoulou, P. Prasertsom, and I. Bahar, "Flexible data allocation for scratch-pad memories to reduce nbti effects," in *International Symposium on Quality Electronic Design*, March 2013, pp. 60–67.

[12] D. Kraak, I. Agbo *et al.*, "Mitigation of sense amplifier degradation using input switching," in *DATE*, March 2017, pp. 858–863.

[13] M. Jenihhin, G. Squillero *et al.*, "Identification and rejuvenation of nbti-critical logic paths in nanoscale circuits," *Journal of Electronic Testing*, vol. 32, no. 3, pp. 273–289, Jun 2016.

[14] W. Wang, V. Reddy *et al.*, "Compact modeling and simulation of circuit reliability for 65-nm cmos technology," *IEEE Transactions on Device and Materials Reliability*, vol. 7, no. 4, pp. 509–517, Dec 2007.

[15] Y. Cao and W. Zhao, "Predictive technology model for nano-cmos design exploration," in *2006 1st International Conference on Nano-Networks and Workshops*, Sept 2006, pp. 1–5.

[16] D. Kraak, I. Agbo *et al.*, "Degradation analysis of high performance 14nm finfet sram," in *DATE*, March 2018, pp. 201–206.

[17] I. Agbo, M. Taouil *et al.*, "Integral impact of bti, pvt variation, and workload on sram sense amplifier," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 4, pp. 1444–1454, April 2017.

[18] "Open cores plasma cpu project," http://opencores.org/project,plasma.

[19] A. Tsepurov, G. Bartsch *et al.*, "A scalable model based rtl framework zamiacad for static analysis," in *VLSI-SoC*, Oct 2012, pp. 171–176.

[20] A. Karandikar and K.K. Parhi, "Low power sram design using hierarchical divided bit-line approach," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*, Oct 1998, pp. 82–88.