Methodology for Application-Dependent Degradation Analysis of Memory Timing

Daniël Kraak   Innocent Agbo   Mottaqiallah Taouil   Pieter Weckx[1,2]   Stefan Cosemans[1]   Francky Catthoor[1,2]
Said Hamdioui

Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
{D.H.P.Kraak, I.O.Agbo, M.Taouil, S.Hamdioui}@tudelft.nl

[1]imec vzw., Kapeldreef 75, B-3001, Leuven, Belgium
[2]Katholieke Universiteit Leuven, ESAT, Belgium
{Pieter.Weckx, Francky.Catthoor}@imec.be

Abstract—Memory designs typically contain design margins to compensate for aging. As aging impact becomes more severe with technology scaling, it is crucial to accurately predict such impact to prevent overestimation or underestimation of the margins. This paper proposes a methodology to accurately and efficiently analyze the impact of aging on the memory's digital logic (e.g., timing circuit and address decoder) while considering realistic workloads extracted from applications. To demonstrate the superiority of the methodology, we analyzed the degradation of the L1 data and instruction caches for an ARM v8-a processor using both our methodology as well as the state-of-the-art methods. The results show that the existing methods may significantly over- or underestimate the impact (e.g., the decoder margin up to 221% and the access time up to 20%) as compared with the proposed scheme. In addition, the results show that in general the instruction cache has the highest degradation. For example, its access time degrades up to 9% and its decoder margin up to 44%.

Index Terms—Memory, Aging, Timing, Address Decoder

I. INTRODUCTION

The continuous downscaling of CMOS technology has resulted in significantly improved performance and functionality of Integrated Circuits (ICs). However, downscaling worsens the reliability due to the increased *time-zero* and *time-dependent* variabilities [1]. Time-zero variability, often referred to as process variation, is caused by imperfections during production. As a result, fabricated ICs have deviating performance. Time-dependent variability is caused by operational stress during the lifetime of the ICs. They include environmental variations, such as voltage and temperature fluctuations, and aging variations due to, for instance, Bias Temperature Instability (BTI) [2]. To achieve a high quality product in terms of low failure rates at optimal design, it is essential to estimate the impact of these variabilities. In this work, we focus on the degradation of SRAMs. They often dominate the total area of SoCs and microprocessors and, therefore, their area, power, and performance are very optimized. Hence, it is crucial to estimate their degradation.

Previous studies on SRAM reliability have mainly focused on estimating the impact of aging on single memory components (e.g., memory cells). The majority of these works investigated the degradation of the memory cell array [3–7], while fewer works investigated the impact on the peripheral circuitry, such as the sense amplifier [8]. Limited work investigated the impact of aging while considering multiple components [9] or

the complete memory core [10, 11]. The latter provides a more accurate estimation, since they also include the impact of the interaction between the aged components. Nevertheless, the digital logic, such as the timing circuit and address decoder, have received little attention. To the best of our knowledge only two works included the digital circuitry in their analysis [10, 11]. They examine metrics such as the access time (which is dominated by the digital circuit) and decoder margin. However, only a single path of the memory was analyzed to estimate the impact based on *simplified artificial* workloads. For example, the authors in [11] assume that each address is accessed an *equal* amount of times, while in [10] the authors assume that all operations are spread over *only* four addresses. These regular workloads do not reflect the reality; hence, they may lead to inaccurate results. Real applications access the memory in a much less regular pattern, as is shown in this work. Therefore, the logic paths receive different stresses and they all need to be analyzed to get a good estimation of the impact on metrics such as the access time and decoder margin.

To address these shortcomings, this work proposes a methodology to efficiently and accurately analyze the impact of aging on the digital logic (e.g., timing circuit, address decoder) of memories using real applications. This methodology is based on performing static timing analysis. Using this methodology it becomes feasible to efficiently analyze all digital logic paths, such as the ones related to the access time and decoder margin. In short, the contributions of this work are as follows:

1) It proposes a methodology to analyze the impact of real applications on the memory's digital logic.
2) It investigates the accuracy of the proposed method based on real applications versus artificial workloads used in the state-of-the-art.
3) As a case study, it investigates and compares the impact of aging due to real applications on the L1 data and instruction caches of an ARM v8-a processor.
4) All the above work is performed using an *industrial-strength* 14 nm FinFET SRAM design and an *accurate* aging model.

The outline of this paper is as follows: Section II provides the background. Section III presents the proposed methodology. Sections IV discusses the performed experiments and the obtained results. Section V contains a brief discussion. Finally, Section VI concludes this work.
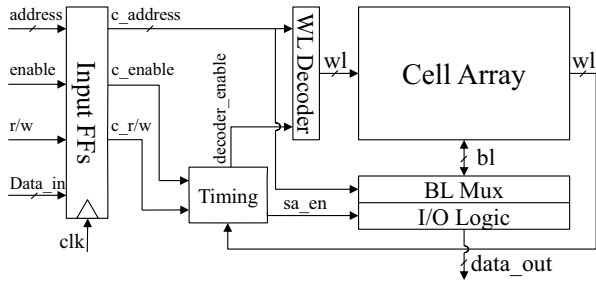
Fig. 1: Block diagram of memory.

II. BACKGROUND

This section first discusses the used memory model and, subsequently, the analyzed metrics.

A. Memory Model

The memory model used in our case study is a high performance 14 nm FinFET SRAM from imec. It has a size of 8 kB and a logical word length of 32 Bytes. Under worst-case conditions, the memory is able to run at a frequency of 2 GHz. Memories with such specs are typically used to implement the L1 data and instruction caches of processors. Fig. 1 shows a diagram of the memory with its relevant components and signals. Each component is described in more detail below.

- **Input Flip-Flops:** the input flip-flops serve as the interface to the memory. The *address* signal contains the memory address for the operation. The *enable* signal enables the operation of the memory. The *r/w* signal specifies the type of operation (read or write).
- **Cell Array:** the cell array consists of 128 rows by 512 columns. It is implemented using high performance cells with a 1:2:2 ratio (1 fin for the pull-up transistor, 2 fins for the pass as well as the pull-down transistors).
- **WL Decoder:** the wordline (WL) decoder selects one of the 128 rows of the cell array based on the input address.
- **BL Mux:** the bitline multiplexer (BL Mux) selects the appropriate columns in the cell array based on the input address. It has a multiplex factor of 2. This means that 256 bits (or 32 Bytes) can be addressed in one operation.
- **Timing:** the timing circuit is the main control circuit of the memory. It provides timed control signals to the other components during operation (e.g., enabling the WL decoder). It uses a WL detect scheme, i.e., it is able to detect the activation of one of the wordlines, as illustrated by the feedback loop in the figure.
- **I/O Logic:** the I/O logic contains the peripheral circuitry to read/write from/to the memory cells, such as the sense amplifiers, write drivers, output latches, and output buffers.

During read operations, the *enable* signal is used to activate the timing circuit. Meanwhile, the address is applied to the WL decoder and BL Mux to select the corresponding rows and columns of the cell array. Subsequently, the timing circuit activates WL decoder using the *decoder_enable* signal. Once the timing circuit detects the activation of the wordline using
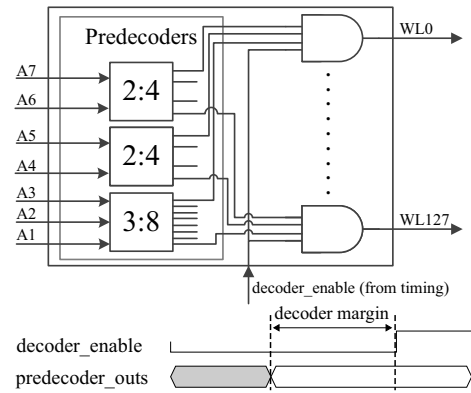

Fig. 2: Metric decoder margin.

its WL detect scheme, it keeps the wordline activated for a certain time to ensure the cells can discharge the bitlines by a sufficient amount. Finally, the timing circuit activates the sense amplifiers using the *sa_en* signal to generate full-swing read values.

B. Metrics

Read Access Time

One of the investigated metrics is the memory's *read access time*. In this work it is defined as the delay between the rising edge of the clock and the activation of the sense amplifiers (*sa_en* in Fig. 1) during a read operation. Note that the read access time is typically measured as the delay between the rising edge of the clock and the output being ready; hence, it also includes the delay of the sense amplifier, which in turn is affected by the cell. Since this work focuses on the degradation of the memory's digital logic, the sensing delay of the sense amplifier is not included. However, the sensing delay occupies only a very short portion of the total read access time [10, 11] and its contribution is less than 2% in our memory design. Hence, our definition of the read access time still gives an accurate approximation of the real access time.

Decoder Margin

The second used metric that is investigated is the *decoder margin*. It is shown in Fig. 2 and is defined as the minimum time between the predecoders of the WL decoder being ready and the activation of the decoder. The figure shows a simplified diagram of the WL decoder, which only shows the connections for the first and last wordline. The wordline decoder consists of three predecoders. The output combinations of these predecoders are uniquely combined as input to the final stage consisting of AND-gates; the output of the AND-gates are used to select one of the wordlines. In addition, the enable_decoder signal from the timing circuit is also connected to the input of these AND-gates. It controls the wordline activation. In case the delay of the predecoders exceeds the time at which the enable_decoder is activated (i.e., negative margin), a wrong wordline might be activated, see also Fig. 2. Hence, the *decoder margin* is the minimum time between the outputs of the predecoders being ready and the activation of the enable_decoder signal.
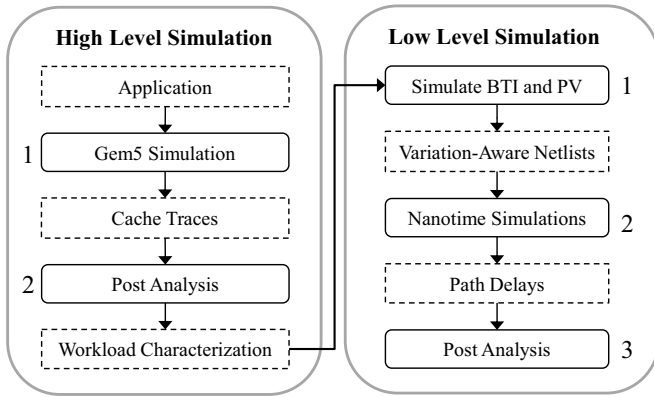
Fig. 3: Proposed methodology.

III. METHODOLOGY

Fig. 3 shows the proposed methodology that is used to investigate the impact of aging of real applications on the digital logic of the memory. As can be seen, it consists of a *high level simulation* and a *low level simulation* part. In the high level simulation part, the workload of the input application is characterized. In the low level simulation part, the characterized workload is used to evaluate the impact of aging. The high and low level simulation parts are discussed next in more detail.

A. High Level Simulation

The high level simulation part consist of two steps, which are described next.

1. Gem5 Simulation: In this step, cache traces for the input application are generated; this is achieved by simulating a CPU architecture using the gem5 simulator [12]. Gem5 has been modified to support the creation of traces of the simulated caches. These traces contain all L1 data and instruction cache operations over time. Per operation the following data is stored: the type of operation (read or write), the cycle in which it is performed, the requested address, and the write data (in case of a write operation).

2. Post Analysis: During the post analysis step, the cache traces are used to characterize the workload. The characterized workload specifies the duty factors at the gates of each transistor of the memory.

Obtaining these duty factors for a memory trace of millions of operations is computational-wise *impossible* using a single Spice simulation. In order to circumvent this bottleneck, the following method is proposed: 1) For each address, the duty factors are determined for the read operation, the write operation, and the idle cycle using Spice waveforms. 2) For each address, the number of read, write, and idle cycles is determined using the memory trace. 3) The duty factor of each transistor (say $dut(i)$) can now be calculated by combining the previous two steps using the following equation:

$$dut(i) = \frac{\sum_{j=1}^{addresses} reads(j) * duts_{read}(i,j)}{cycles} + $$
$$\frac{\sum_{j=1}^{addresses} writes(j) * duts_{write}(i,j)}{cycles} + \quad (1)$$
$$\frac{\sum_{j=1}^{addresses} idles(j) * duts_{idle}(i,j)}{cycles}$$

Here, $duts_{read}$, $duts_{write}$, and $duts_{idle}$ are tables that contain the duty factors for each transistor for a read, write, and idle cycle, respectively, per accessed address obtained from the first step of the post analysis. $Reads$, $writes$, and $idles$ are tables that contain the number of read, write, and idle cycles, respectively, per address obtained from the second step of the post analysis. $Cycles$ is the total amount of cycles that the trace covers. It is important to note that this method only works for the memory logic that does not depend on the read/write values. For our study we are only interested in the access time and decoder margin, which both do not depend on these.

B. Low Level Simulation

In the low level simulation part, the impact of aging is evaluated using the characterized workload obtained from the high level simulation. Each step is discussed in more detail next.

1. Simulate PV and BTI: During this step, *variation-aware* memory netlists are generated. These netlists incorporate both the effects of time-zero and time-dependent variabilities, by performing Monte Carlo simulations where Process Variation (PV) and aging due to Bias Temperature Instability (BTI) are simulated. For each Monte Carlo iteration a separate netlist is generated. To model the effects of PV, Pelgrom's model is used [13]. To model BTI, the *calibrated* atomistic model from [14] is used; it includes the dependency on the workload, which is modelled by the duty factors of the transistor's gate voltage. These required parameters are obtained from the characterized workload generated during the high level simulation.

2. Nanotime Simulations: In this step, the relevant path delays of the generated variation-aware netlists are measured using Synopsys' Nanotime [15]. Nanotime is a transistor-level static timing analysis tool that is able to efficiently analyze logic paths while providing a high accuracy. The accuracy is within a 5% error margin of Spice [15]. Hence, this tool makes it feasible to accurately analyze all paths. Note that once the worst-case paths (relative to the other paths) are identified with Nanotime, it is possible to get an even higher accuracy by simulating these paths using Spice.

3. Post Analysis: In the final step, post analysis is performed on the measured path delays. For our analysis, the worst-case path delays are analyzed for each Monte Carlo instance, both for the access time and decoder margin.
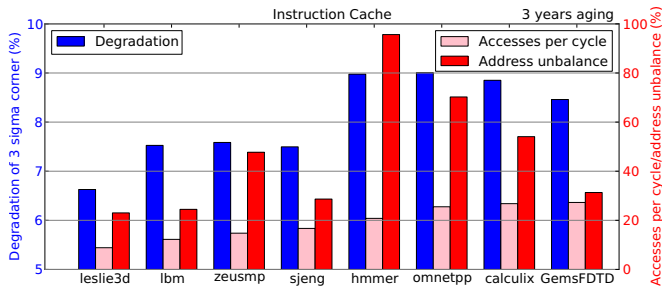
Fig. 4: Degradation of the instruction cache's access time.

Fig. 5: Degradation of the instruction cache's decoder margin.

TABLE I: Gem5 configuration.

Processor	ARM v8-a, single-core, out-of-order @ 1.8GHz
L1 Data & Instruction Cache	32 kB, 4-way set associative, 32 B linesize
L2 Cache	1 MB, 8-way set associative, 32 B linesize

IV. RESULTS

A. Performed Experiments

We use the methodology of Fig. 3 to investigate the impact of aging on the access time and decoder margin of the L1 instruction and data caches. In the high-level simulation step, we simulate eight different applications from the SPEC2006 Benchmark suite [16] on an ARM v-8a processor using gem5. Details of the gem5 configuration can be found in Table I. In order to make the simulation feasible, we simulate a sample of one billion instructions per application. In the gem5 simulation, we assume that the data and tag sections of each set are implemented using separate memories. Hence, two memory traces are generated per cache set (one for the data section and one for the tag section). Due to similar accesses to the data and tag sections within a set and also to the different sets within the cache, we limit our analysis to only the first set's data section. In the low-level simulation, we assume that this data section is implemented using the memory model discussed in Section II. Using this setup, we perform the following experiments:

1) **Application dependency**: we investigate the impact of aging on the access time and the decoder margin of the L1 instruction cache for different applications. This will be used as a baseline to compare our approach with the state-of-the-art.

2) **Impact of workload characterization**: we investigate and compare the accuracy of our proposed method to the state-of-the-art methods [10, 11]. To perform this comparison, we also characterize the same applications with similar artificial workloads used in the state-of-the-art. For each application, we define a *best-case* workload based on [11] and a *worst-case* workload based on [10]. For these best-case and worst-case workloads we assume that the amount of performed operations is the same as in the application. However, for the best-case the accesses are uniformly spread over all addresses, while for the worst-case they are uniformly spread over only the first four addresses.

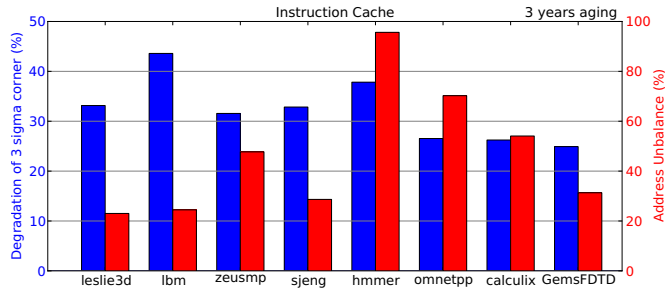3) **Cache dependency**: we investigate the impact of aging on the access time and the decoder margin of the L1 data

cache for different applications and compare it with the degradation of the L1 instruction cache.

In each experiment above, 1000 Monte Carlo simulations are performed per application/workload. We assume that the memory is aged for three years at 85°C (junction temperature) and a nominal supply voltage of $0.8\,\mathrm{V}$. For each set of Monte Carlo simulations, the $3\,\sigma$ corners are calculated based on the average and spread of the measured access times and decoder margins. These are then compared with the $3\,\sigma$ corners at time-zero (which is only affected by process variation).

B. Results

Application Dependency

Access Time: Fig. 4 shows the degradation of the $3\,\sigma$ corner of the access time for the L1 instruction cache for the 8 applications. In addition to the access time degradation, two additional metrics are included that provide information about the access patterns: the *operations per cycle* and *address unbalance* metrics. The *accesses per cycle* metric shows the amount of memory operations (read and write) that the application performs. For example, a value of 25% means that the applications performs on average every four cycles a memory operation. The *address unbalance* metric gives information on the distribution of the memory accesses. It presents the percentage of the accesses that are performed on the ten most accessed addresses as compared with the total accesses. The scale of both these metrics is shown on the right axis.

The figure reveals that the degradation is strongly dependent on the application. For example, the lowest degradation is ∼6.6% (for 'leslie3d'), while the highest degradation is ∼9.0% (for 'omnetpp'). First of all, this strong dependency is caused by the different access patterns of the applications. Typically, a higher number of *accesses per cycle* results in a higher degradation of the access time because the common logic paths are stressed more. For example, 'lbm' (∼12% accesses per cycle) has a degradation of ∼7.5%, while 'GemsFDTD' (∼27.2% accesses per cycle) has a degradation of ∼8.5%. In addition, the distribution of the accessed addresses also causes a different degradation between the applications. Applications with a high address unbalance have a higher degradation. For instance, 'omnetpp' has the highest degradation, while it does not have the highest amount of accesses. This is due to the fact that it has a high address unbalance. *This shows the importance of using the correct workload for accurate degradation estimation.* Applications with an unbalanced access pattern
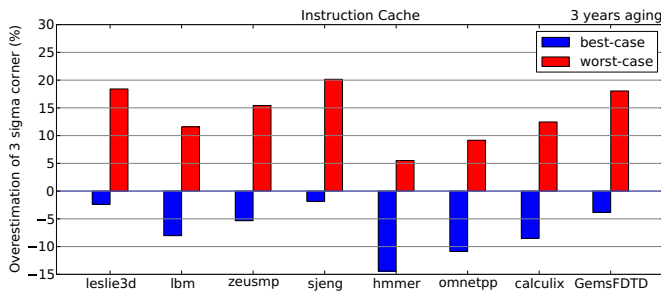
Fig. 6: Overestimation of the access time degradation.

Fig. 7: Overestimation of the decoder margin degradation.

stress the circuitry related to particular addresses more often (e.g., wordline driver). As a result, the paths of these addresses become slower and their access time increases as compared to the other addresses.

Decoder Margin: Fig. 5 shows the degradation of the 3σ corner of the decoder margin for the L1 instruction cache for the different applications; the degradation of the decoder margin is also strongly dependent on the application and even severe than that of the access time. The degradation of the decoder margin goes up to 44% (for 'lbm'), while this is only ~9.0% for the access time. The higher application dependency and higher decoder margin degradation can both be explained by the fact that certain paths in the address decoder are constantly stressed even when the memory is not performing operations. Finally, comparing the degradation with the address unbalance reveals that it is not straightforward to identify any correlation between them. For example, 'lbm' and 'leslie3d' have a similar address unbalance. Nevertheless, 'lbm' has a ~30% higher relative degradation. This is due to the fact that even though applications may access a wide variety of addresses, there is no guarantee that the paths of the predecoders receive a similar stress. For example, if an application mainly accesses the first 128 out of 256 addresses, then the most-significant address bit for the decoder is low most of the time. Hence, the predecoder (i.e, the top 2:4 precoder in Fig. 2) that processes this address bit will have a very unbalanced stress and, therefore, some of its paths have a high degradation.

Impact of Workload Characterization

Access Time: Fig. 6 compares the 3σ corner degradation of the access time of the best-case and worst-case characterizations relative to the actual degradation base-line for each application; the actual degradation is taken from Fig. 4. The figure reveals that the best-case characterizations underestimate the degradation, while the worst-case characterizations overestimate the degradation. The best-case characterizations underestimate the actual degradation, because they assume that the memory operations are equally spread over all addresses, while the worst-case characterizations have an overestimation due to the assumption that the memory operations are spread over only the first four addresses. Depending on the application, the underestimation can reach ~15%, while the overestimation can reach ~20%; this illustrates again the importance of appropriate modelling the behavior of a *real*

application for optimal design.
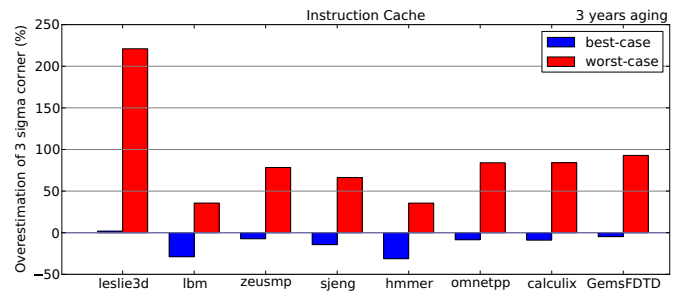
Decoder Margin: Similarly, Fig. 7 compares the 3σ corner degradation of the decoder margin of the best-case and worst-case characterizations relative to the actual degradation (from Fig. 5). Clearly, the worst-case characterizations are way too pessimistic at estimating the degradation; the overestimations are between 35% and 221%. These high values are due to the fact that the worst-case characterizations only stress the first four addresses. As a result, the most significant address bits never change their value. This means that certain paths of the predecoders are constantly stressed. On the contrary, the best-case characterizations are relatively closer to the actual degradation. The highest observed underestimation is ~31% as the access patterns of the real applications are closer to the patterns described by the best-cases. Moreover, the best-case characterization of 'leslie3d' even shows a slight over-estimation of 2%, meaning that the application accesses the different memory addresses very evenly. This application has a bias towards addresses that are activated using short decoder paths, meaning a slightly lower degradation as compared with its best-case characterization.

Cache Dependency

Access Time: Fig. 8 shows the degradation of the access time for the L1 data cache for several applications. In case we compare the degradation of the data cache with that of the instruction cache (Fig. 4), we observe that the data cache in general has a lower degradation. Its access time degradation is between ~5.6% and ~7.4%, while that for the instruction cache is between ~6.6% and ~9.0%. The lower degradation for the data cache is due to the applications performing less accesses to the data cache and, therefore, its read paths receive less stress.

Decoder Margin: Fig. 9 shows the degradation of the decoder margin for the L1 data cache for several applications. Comparing the degradation of the decoder margin of the data cache with that of the instruction cache (Fig. 5) reveals that they both have a similar degradation; the degradation is between 32% and 43% for the data cache and for the instruction cache it is between 24% and 44%. This similarity is due to the fact that both the data and instruction caches have applications that access the memory's addresses in a fairly distributed way (and, therefore, give a lower degradation) and applications that access the addresses in a less distributed way (and give a higher degradation).
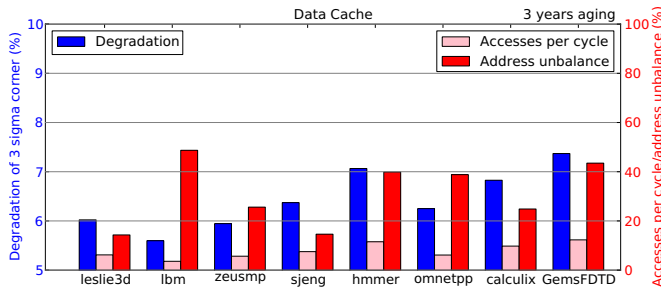
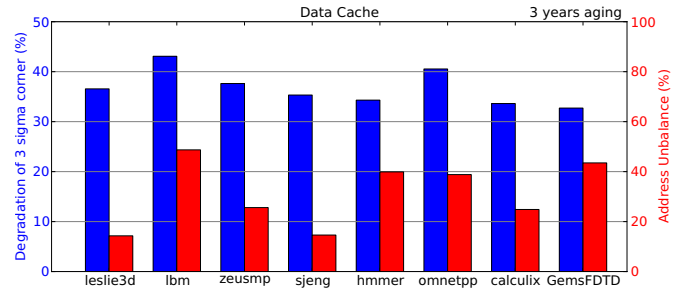Fig. 8: Degradation of the data cache's access time.

Fig. 9: Degradation of the data cache's decoder margin.

V. Discussion

The main messages of this study are the following:

Methodology: it is of great importance to have a methodology that efficiently (in terms of simulation time) and accurately (in terms of selecting the real behavior of applications) estimates the impact of aging of memory's digital circuits; like our methodology. Our work reveals that using artificial workloads to mimic the application behavior is very misleading and can overestimate e.g., the decoder margin with up to 221%, resulting in unnecessary overdesign (yield loss).

Access Time versus Decoder Margin: the decoder margin is relatively more sensitive to aging than the memory access time (up to 5x). This could clarify why most of the customer returns are due to timing issues in the cache's decoder [17–19]. Hence, special attention should be given to address decoder timing.

Cache Dependency: although their decoder margin degradation is quite similar, the access time of the instruction cache is more sensitive to aging than that of the data cache. Nevertheless, most of the published work in this field focuses on data caches, e.g., [20, 21]. Hence, more effort should be put not only to better understand the instruction cache aging, but also to provide appropriate mitigation schemes.

VI. Conclusion

This work proposed a superior methodology that efficiently and accurately analyzes the degradation of the memory's digital logic for real applications. To demonstrate the methodology, it analyzed the degradation of the L1 instruction and data caches for an ARM v8-a processor. The results showed that the proposed method estimates the impact significantly more accurate compared to the existing methods. In addition, they underline the importance of paying enough attention to the degradation of the decoder margin for both caches and to the access time for the instruction cache.

References

[1] S. Borkar, "Microarchitecture and design challenges for gigascale integration," in *Intl. Sympos. on Microarchitecture*, Dec 2004, pp. 3–3.

[2] B. Kaczer, S. Mahato, V.V. de Almeida Camargo *et al.*, "Atomistic approach to variability of bias-temperature instability in circuit simulations," in *2011 International Reliability Physics Symposium*, April 2011, pp. XT.3.1–XT.3.5.

[3] V. Huard, C. Parthasarathy, C. Guerin *et al.*, "Nbti degradation: From transistor to sram arrays," in *2008 IEEE International Reliability Physics Symposium*, April 2008, pp. 289–300.

[4] 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.

[5] B. Cheng, A.R. Brown, and A. Asenov, "Impact of nbti/pbti on sram stability degradation," *IEEE Electron Device Letters*, vol. 32, no. 6, pp. 740–742, June 2011.

[6] S. Pae, J. Maiz, C. Prasad, and B. Woolery, "Effect of bti degradation on transistor variability in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 8, no. 3, pp. 519–525, Sept 2008.

[7] S. Khan, I. Agbo, S. Hamdioui *et al.*, "Bias temperature instability analysis of finfet based sram cells," in *2014 Design, Automation Test in Europe Conference Exhibition*, March 2014, pp. 1–6.

[8] R. Menchaca and H. Mahmoodi, "Impact of transistor aging effects on sense amplifier reliability in nano-scale cmos," in *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, March 2012, pp. 342–346.

[9] I. Agbo, M. Taouil, S. Hamdioui *et al.*, "Read path degradation analysis in sram," in *2016 21th IEEE European Test Symposium (ETS)*, May 2016, pp. 1–2.

[10] 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, 2017*, March 2017, pp. 852–857.

[11] D. Kraak, I. Agbo, M. Taouil *et al.*, "Degradation analysis of high performance 14nm finfet sram," in *2018 Design, Automation Test in Europe Conference Exhibition*, March 2018, pp. 201–206.

[12] N. Binkert, B. Beckmann, G. Black *et al.*, "The gem5 simulator," vol. 39, pp. 1–7, 08 2011.

[13] M.J.M. Pelgrom, A.C.J. Duinmaijer, and A.P.G. Welbers, "Matching properties of mos transistors," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 5, pp. 1433–1439, Oct 1989.

[14] B. Kaczer, T. Grasser, P.J. Roussel *et al.*, "Origin of nbti variability in deeply scaled pfets," in *2010 IEEE International Reliability Physics Symposium*, May 2010, pp. 26–32.

[15] I. Synopsys, "Nanotime - transistor-level static timing analysis solution for custom designs," https://www.synopsys.com/content/dam/synopsys/implementation&signoff/datasheets/nanotime-ds.pdf.

[16] J.L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737

[17] 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*, Oct 2004, pp. 114–123.

[18] 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 (IEEE Cat. No.98CH36270)*, Oct 1998, pp. 25–34.

[19] S. Hamdioui, R. Wadsworth, J.D. Reyes, and A.J. van de Goor, "Memory fault modeling trends: A case study," *Journal of Electronic Testing*, vol. 20, no. 3, pp. 245–255, Jun 2004. [Online]. Available: https://doi.org/10.1023/B:JETT.0000029458.57095.bb

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

[21] S. Ganapathy, R. Canal, A. Gonzlez, and A. Rubio, "irmw: A low-cost technique to reduce nbti-dependent parametric failures in l1 data caches," in *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, Oct 2014, pp. 68–74.