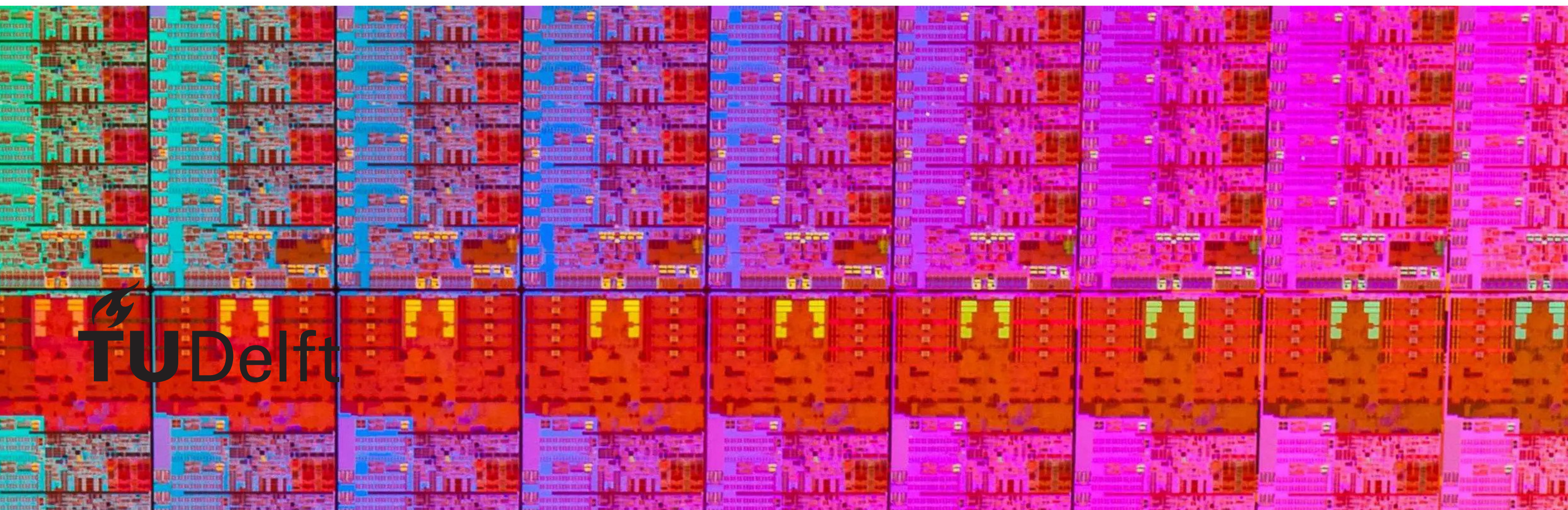
SRAM

Mitigation for BTI ageing in SRAM memories

L.J. Hamburger

Technische Universiteit Delft



BTI in SRAM

Mitigation for BTI ageing in SRAM memories

by

L.J. Hamburger

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday November 25, 2020 at 12:00.

Student number:	4292936	
Project duration:	September 2, 2019 – November 25, 2020	
Thesis committee:	Prof. Said Hamdioui,	TU Delft, supervisor
	Dr.ir Rene Van Leuken,	TU Delft
	Dr.ir Mottaqiallah Taouil,	TU Delft
	Ir Daniel Kraak,	TU Delft

TU Delft

Abstract

The aggressive downscaling of the transistor has led to gigantic improvements in the performance and functionality of electronics. As a result, electronics have become a significant part in our daily lives whose absence would be difficult to imagine. Our cars, for example, now consist of many sensors and small computers each controlling certain parts of the car. A downside of the aggressive downscaling of transistor sizes is that it negatively impacts the reliability and accelerated ageing, and thus a reduced lifetime, of electronics. Nevertheless, to ensure the reliable operation of electronics, it has therefore become essential to assess the reliability of *any* of its embedded components accurately. Conventionally, to combat ageing, designers use guardbanded design; adding design margins. These margins, however, lead to a penalty in area, power, and speed. Alternatively, one may investigate mitigation schemes that aim at reducing the impact of ageing to extend the reliability and lifetime. These mitigation schemes may lead to a higher performance compared with the conventional guardbanded design. This work focuses on an ageing mitigation scheme for SRAMs. SRAMs typically have the highest contribution to the total area of integrated circuits. Therefore, they are highly optimised (i.e. their integration density is the lowest). This also makes them one of the most susceptible components to ageing. Hence, providing appropriate ageing mitigation schemes for SRAMs is essential for the overall reliability of ICs.

Whereas prior work has mainly investigated hardware-based ageing mitigation schemes for SRAMs, this thesis investigates the possibility of mitigating the ageing through *software*. The advantages of this approach include that it does not require circuit changes (and, thus applicable to existing circuits) and it comes at zero area overhead. This study's proposed software-based scheme is based on periodically running a mitigation routine. This mitigation scheme flips the contents of the memory cells to put the transistors into relaxation from BTI stress, the most crucial ageing mechanism in deeply scaled CMOS process. The results show that the software-based scheme can significantly reduce the ageing of the memory at a low overhead. For example, the degradation of the hold SNM metric of the memory cell is reduced with up to 40% at a runtime overhead of only 1.4%. Moreover, the scheme also mitigates the ageing of other components of the memory. For example, the degradation of the offset voltage of the sense amplifier is reduced by nearly 50%. This thesis shows that it is possible to use software to mitigate the ageing effects in the memory components and it is worthwhile to consider implementing it.

Preface

Thank you to all who have been during my thesis or study.

To my closest friends for supporting me and always being there to proofread parts of my thesis, even it meant reading more mistakes than correct words.

To my colleges at Insyde and fellow master students, Profs, PhD'ers, and employers for supporting me and having fun events.

Next to this, I would like to thank the IRIS, many hours have been made to program the software and support the different organisation hosting a rowing competition. It is always great fun to travel to the competition itself and feel the vibe there and make the rowers happy with a well-organised event. The competitions ranged in size from a few hundred people participating in events where thousands of people would participate, and even more people checking the results.

L.J. Hamburger
Delft, November 2020

Contents

List of Figures

List of Tables

Glossary

dark silicon Due to the power budget constraint some parts of a chip cannot always be used and thus are turned off. The parts which are turned off can be used to speed up for exclusive use cases.

infield The environment in which a chip is run during its lifetime.

Acronyms

$\overline{\mathbf{BL}}$ Inverted value of the bit line

BIST Built-in self-test

BL Bit line

BTI Bias Temperature Instability

CFDR Cell Flipping technique with Distributed Refresh phases

CMOS Complementary metal–oxide–semiconductor

CPU Central processing unit

DRAM Dynamic random-access memory

EM Electromigration

EMBC Embedded Microprocessor Benchmark Consortium

HCI Hot Carrier Injection

IC Integrated Circuit

ITL Idle time leverage

IVC Input vector control

LER Line edge roughness

LFSR Linear feedback shift register

LVL Logic-Wear-Levelling

MGG Metal gate granularity

MOSFET Metal-oxide-semiconductor field-effect transistor

NBTI Negative Bias Temperature Instability

NMOS N-channel MOSFET

NOP No operation

OTV Oxide thickness variation

PBTI Positive Bias Temperature Instability

PMOS P-channel MOSFET

PVT Process, voltage, and temperature

RAM Random access memory

RD	Reaction-Diffusion
RDD	Discrete random dopants
RISC-V	Reduced instruction set computer five
RTN	Random Telegraph Noise
SA	Sense Amplifier
SNM	Static noise margin
SRAM	Static random-access memory
SVS	Static voltage scaling
TDDB	Time-Dependent Dielectric Breakdown
WL	Word line

Nomenclature

I_{leak}	Leak current
I_{sc}	Short circuit Current
T_{p_0}	Intrinsic delay of an inverter
t_{ret}	Retention time
t_{sc}	Short circuit Time
V_{dd}	Supply voltage
V_{DSAT}	Saturation voltage
V_t	Threshold voltage

1 Introduction

This chapter will introduce the thesis:

1. Section 1.1 provides motivation and the relevance of SRAM ageing and why its mitigation is useful.
2. Section 1.2 provides a brief overview of the state-of-the-art mitigation schemes and their shortcomings.
3. Section 1.3 will present the contributions this thesis has made.
4. Section 1.4 will give an outline of the thesis.

1.1. Motivation

The downscaling of the complementary metal–oxide–semiconductor (CMOS) technology has improved the performance and functionality of Integrated Circuits (ICs). The size of the transistors is now in the order of 2 nm to 7 nm [7, 8]. Several challenges are currently reducing the rate of this aggressive downscaling. One of the major challenges is that the reliability of transistors reduces due to accelerated ageing effects, which results in a shorter lifetime of the circuit [9]. The bathtub curve illustrates the impact of scaling and, thus, the reduced lifetime of ICs , as shown in figure 1.1. The bathtub curve shows the expected failure rate of a product at a certain point in time. As also highlighted in figure 1.1, the bathtub curve consists of three phases:

- **Early failure**: This is the phase where chips have just been manufactured. In this phase, the failure rate is high. This is because, during production, defects in the chips can occur. These defects will result in a chip not working or failing early. To remove these bad chips from the production line, each chip is tested. By filtering out the bad and the weak chips, the failure rate is reduced into an acceptable rate to apply the chips in-field, done by burn-in testing.
- **Random failure**: This is the phase when the chips are sold and are, thus, put into the field. In this phase, the failure rate is low, and it entails the expected lifetime of the chip. Failures in this phase occurred randomly, caused by single-event upsets.
- **Wear out**: In this phase, the chip reaches the end of its lifetime and ageing effects inside the chip start causing failures. As a result, the failure rate increases.

Figure 1.1 [10] shows various curves. Each of these curves depicts the failure rate for different technology nodes. As can be seen, as the transistor dimensions decrease, the failure rate increases and the lifetime is shortened, since smaller technologies age faster. Hence, the lifetime and reliability of ICs is decreasing.

As the lifetime of ICs is shrinking, proper actions must be taken. Conventionally, designers add margins to the design, called worst-case-design, (i.e. guardbanding). As the impact of ageing increases with technology scaling, higher margins must be added, resulting in a penalty in speed, area, and power consumption. Moreover, the increased area leads to a lower yield. As an alternative to guard banding, *mitigation schemes* can be incorporated into the design. These mitigation schemes aim at reducing the impact of ageing.

Within ICs, SRAM takes up a substantial percentage of the total die area [11]. Hence, the design of this part of the chip must be as optimal as possible. Therefore, the margins of memories are highly optimised to reduce area and power and to improve their performance. A disadvantage of this is that the memory becomes

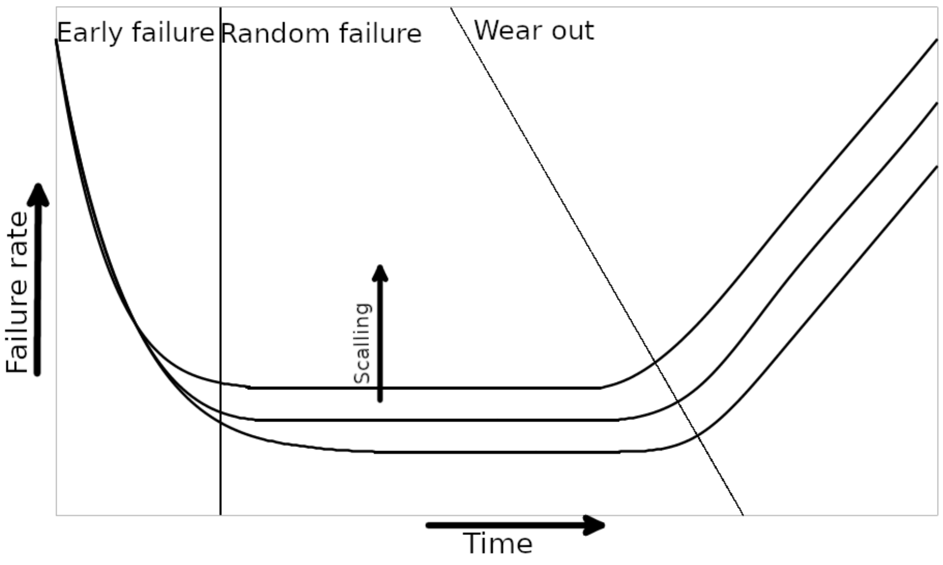


Figure 1.1: A bathtub curves shown depicting tracing the failure rate of different technologies over time. The bathtub has three phases as shown

more susceptible to *ageing*. Hence, why it is vital to incorporate ageing mitigation schemes into the memory. Therefore, this thesis focuses on developing ageing mitigation schemes for SRAMs.

When looking at the state-of-the-art ageing mitigation schemes, most solutions focus on hardware-based mitigation schemes. The disadvantage of these schemes is that they require the original hardware to be altered to accommodate the ageing mitigation scheme, leading to a penalty in area, power, and speed. In this thesis, we research the possibility of using an alternative to hardware-based mitigation: mitigation through software. In this case, mitigation through software means using a software co-routine that mitigates ageing. Software-based mitigation schemes are expected to have the following advantages:

- They can be added to existing ICs; no hardware modifications are needed.
- They come at zero area overhead.
- They can be applied during idle times of the application.
- They can work in conjunction with hardware mitigation schemes.

The research question of this thesis is, therefore, as follows:

Is it possible to mitigate ageing of the whole, or a subset of the memory using software routines?

1.2. State of the Art

This section briefly overviews state of the art in ageing mitigation. First, it provides a classification of the different ageing mitigation schemes. Next, the application of sensors to improve mitigation schemes against ageing is discussed. Finally, it discusses the shortcomings of the state of the art.

1.2.1. Ageing mitigation schemes

Figure 1.2 gives a classification of different ageing mitigation schemes. The mitigation schemes are divided into two categories: mitigation during *design-time* and mitigation during *run-time*. Mitigation schemes take different input into account to decide if the hardware has had too much ageing. This decision can either be done by prediction or with the corporation of sensing or taking no run time information into account and maintain fixed intervals when the mitigation is run.

Mitigation during design time

One approach to mitigate ageing is to take it into account during design; guaranteeing that the chip functions correctly during its required lifetime. The required lifetime of a chip depends on the target application of the chip. For example, chips used in cars or the aerospace industry require a longer lifetime than chips used in mobile phones or other non-critical or less critical hardware. When the design takes ageing into account, it

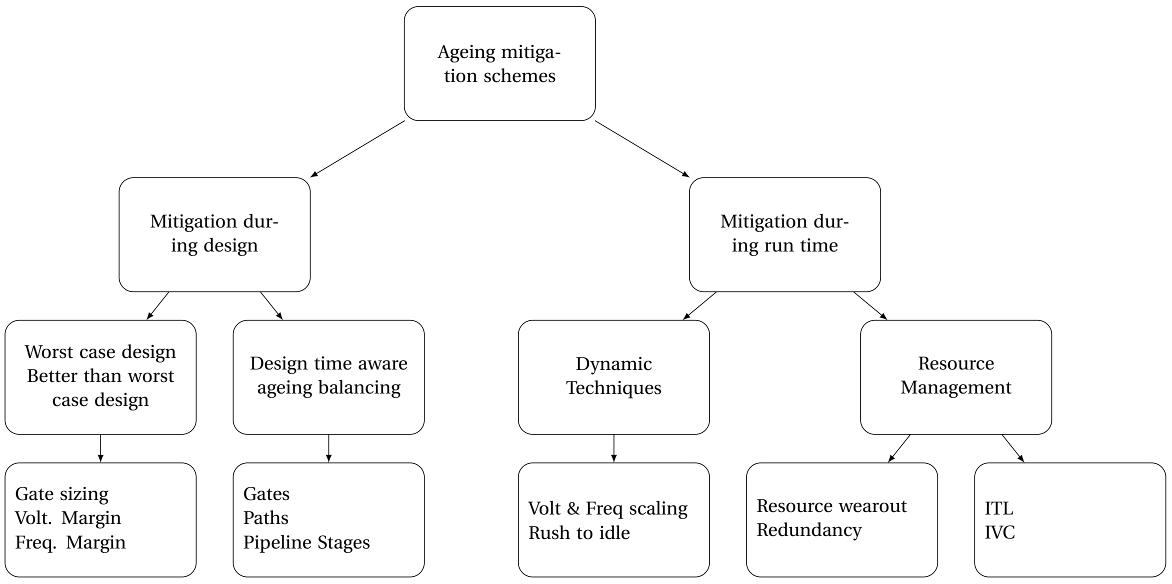


Figure 1.2: Taxonomy of ageing mitigation schemes

needs to add extra margins (and thus more hardware). As can be seen in figure 1.2, the mitigation techniques that are applied during design can be divided into *worst-case design* and *design-time aware ageing balancing*. In the case of worst-case design, worst-case operating conditions and ageing for the transistors are assumed during the design [12, 13]. Because of these worst-case assumptions, a margin is added to the chip to allow for correct operation under ageing. A disadvantage of this technique is that it is pessimistic and results in over-design leading to penalties in area, power, and performance.

An alternative to worst-case based design is *design-time aware ageing balancing*. In contrary to the worst-case-design strategy, information about the workload is used to determine which transistors will age the most and, thus, will fail first. An advantage of this method is that it reduces the hardware overhead compared with worst-case-design [14, 15]. A limitation of this method is that mitigating the ageing effects of transistors running different workloads requires having different library cells for each expected lifetime and load [16]. However as stated in the papers, implementing these different library cells requires too high effort from the designers of these library cells. Besides, the other limitation relates to the design-time aware ageing balancing mitigation, which is workload-specific. The above two points can yield a significant amount of effort. Moreover, the latter point may also cause a reduced lifetime when an error is made in the prediction of the workload that will occur in the field [17]; implementing this scheme requires know what will happen during the lifetime of this chip. The halting problem [18] limits this knowledge; proving it is not possible to predict if a computer program halts given certain inputs. To know which static stresses could occur within a memory component requires knowledge of the execution flow of the program given any input which could occur. As it is not possible to tell upfront if a program will stop, this could mean it would take considerable time to analyse the control-flow program.

Mitigation during run time

Besides the static mitigation schemes that are applied during the design, it is also possible to embed mitigation schemes into the chip that are activated during *run time*. Figure 1.2 illustrates the run-time mitigation schemes consist of two different methods: *dynamic* schemes and schemes based on *resource management*.

Dynamic techniques: One way to ensure an IC works while the transistor performance degrades over time, is by altering the requirements for the transistor over time. In this category, the performance metrics of the chip change steadily. By altering the voltage, frequency, and temperature of a chip can extend the lifetime. Based on sensors placed on the chip as shown in [19, 20] the voltage and frequency can be changed, to keep the chip running. Change of the voltage or frequency comes with a disadvantage that sensing is required, and if the chip is used in a real-time system, the timing constraints might be violated, or the power usage increases. In [21] it is shown it is possible to determine at which point the voltage should scale to keep the chip reliable, called static voltage scaling (SVS). In [22] the SVS method only yields a 7% improvement over guardbanding. Changing the voltage V_{dd} has some disadvantage. Increasing the V_{dd} comes with an increase in power consumption. Lowering the V_{dd} reduces the frequency at which the chip can run as seen in equation (1.1a) and equation (1.1b) based on the formulae from [23].

$$P = C \cdot V_{dd} \cdot f + V_{dd} \cdot I_{leak} + t_{sc} \cdot I_{sc} \cdot V_{dd} \tag{1.1a}$$

$$\begin{aligned} t_{p_{inv}} &\approx t_{p0} \cdot \frac{V_{dd}}{V_{dd} - V_{te}} \\ V_{te} &= V_t + V_{DSAT}/2 \end{aligned} \tag{1.1b}$$

To effectively control the V_{dd} and utilise supply voltage for age mitigation, it must have a control accuracy in the range of 5 mV to 10 mV for the V_{dd}. Having such exact control requires high area overhead, as argued in [24].

It is also possible to use power gating on the chip, turning off parts on the chip to improve the lifetime [25, 26]. Computational sprinting (i.e. rush-to-idle) can yield the longest possible duration of relaxation. In the idle time, power gating can be used to mitigate ageing effects as shown in [27].

Resource Management: Resource management can use the last mitigation technique. Either strategy can be applied to evenly wear-out the different available resources. Adding redundant hardware would allow for an adaptive scheme to alter which part of the hardware is used. This would still require extra hardware to be implemented at the design phase.

A mitigation scheme without any hardware requirements would be either idle time leverage (ITL) or input vector control (IVC). Within the scheme, ITL-software mitigation scheme can utilise the unused computation time [28]. The IVC scheme will not take advantage of idle time in the CPU. It will use an interrupt routine to halt the computation. IVC has been successively used in memories for example by bit flipping the memory cells [29].

1.2.2. Sensing

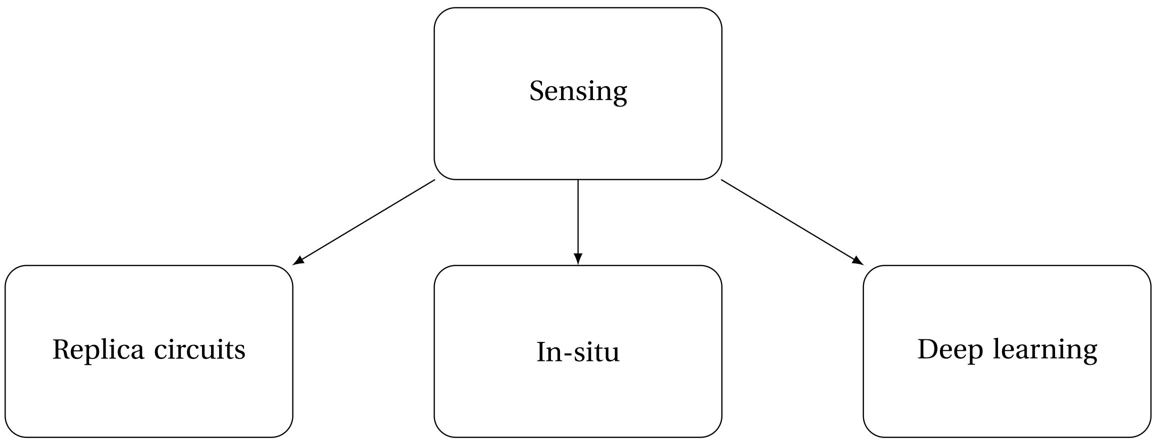


Figure 1.3: Taxonomy of sensing schemes

Sensors are popular tools to measure and subsequently act upon the ageing of the circuit. These sensors supply extra tools to ensure the chip will continue to function correctly over time. In [17] discusses several different ageing sensors; figure 1.3 gives an overview.The sensing scheme must be able to run without extra external hardware to sense the ageing of transistors during operation of the chip. No or little performance impact should be inflicted on running the application, which will result in a trade-off between area, power and accuracy. Looking at [17], several methods of sensing are of interest, divided into *replica circuits* and *in-situ* sensors/monitors.

The current focus of ageing sensors on chips lies on the usage of replica circuits to measure the degradation and thereby the ageing [30, 31]. Replica circuits is an extra piece of hardware not used by the CPU within its logic to compute. Instead, this extra hardware has dummy values passed through to try and simulate the hardware ageing effects. The replica circuit is a lot smaller compared to the complete IC and can then be actively monitored and sensed to estimate the ageing. The disadvantage of replica circuits is that they measure the exact amount of ageing on the circuit less accurately [32]. This limitation is because the ageing of the circuit is not measured, but the ageing of the replica circuit, which does not fully represent the ageing of the primary circuit.

Besides replica circuits, in-situ sensors can be embedded into the chip [33, 34]. These in-situ sensors directly measure the performance of the (sub)circuits to measure their ageing. A disadvantage of in-situ sensors is that they can come with a significant area and power penalties [17]. However, their accuracy is typically higher, compared with the replica circuits.

Deep learning
As sensing the current sensing methods are not that accurate and require and high overhead. Recently, people started research into the usage of deep neural networks to predict the ageing of a complete chip by using a few sensors on the chip [35–37] to reduce this overhead. An advantage of this approach is that it can reduce the overhead of the sensing network significantly [17]. Reducing the required number of sensors within the IC reduces the overhead at which sensing comes. Deep learning can even use the BIST to sense the degradation of the chip [37].

1.2.3. State-of-the-art limitations

The state-of-the-art analysis clearly shows that there is little work on software-based ageing mitigation schemes and in particular for static random-access memories (SRAMs). To the best of our knowledge only one scheme has been proposed for SRAMs [29]. In this work, the authors propose to periodically flip the contents of all memory cells to balance the probability of storing a zero or one in them as this results in their lowest possible degradation. Limitations of the above work and other work on SRAM mitigation are as follows: firstly, they consider only one component (the memory cell). However, as shown in [38], the ageing of different components of the memory affects other parts. Secondly, prior work typically uses SRAM and ageing models which are relatively old and, thus, less accurate and relevant. Thirdly, most papers do not use realistic workloads based on real applications.

Besides the limited work on software-based mitigation schemes for the SRAMs, none of the literature investigates the possible performance advantage of using sensors within the IC compared to time-based routines. However, it can be expected that the use of sensors can also aid in improving the effectiveness of the mitigation since the system can now monitor which parts degrade more and act with an appropriate response (e.g. by putting degraded parts more often into idle mode). Hence, this gives rise to the following sub research question:

Does the use of ageing sensors improve the performance of software-based ageing mitigation compared to software-based mitigation that does not use sensing?

Besides the use of sensors, the built-in self-test (BIST) inside chips may also be an interesting tool to deploy for ageing mitigations. Using BIST to mitigate the hardware ageing, this would reduce the time overhead to mitigate hardware ageing. The BIST can be used to test the chip infield. During this testing, lots of patterns are applied to the circuit [39]. These test patterns have a pseudo-random nature when a linear feedback shift register (LFSR) is used to generate the vectors. Since different vectors are applied to the circuit, the circuit is subject to various workloads. Hence, it may be possible to adapt these vectors to mitigate the circuit ageing in the form of IVC. The literature study, however, yielded no research to adapt the BIST to apply input vector control (IVC) patterns to reduce circuit degradation and to apply Logic-Wear-Levelling (LVL). As the literate study did not yield anything about using BIST for ageing mitigation, it raises a second sub research question:

Can the test patterns in the BIST be altered to allow for mitigation against ageing?

1.3. Contributions

This thesis proposes a *software-based* mitigation scheme for SRAM memories. The scheme is based on periodically flipping the contents of the memory cells reducing the duration of static stress periods is significant, resulting in a lower overall degradation of the SRAM memory. Additionally, this software-based scheme also helps to mitigate the ageing of the memory's Sense Amplifiers. Besides the software mitigation scheme, a hardware mitigation scheme has been created and tested. The hardware mitigation works similarly to the software scheme: it flips the memory cells to reduce the static stress duration. It also changes the average stored value in the memory, such that the probability of storing a one or zero becomes equal. In short, the contributions of this thesis are as follows:

- **Memory access model:** The authors propose an analytical model to predict the memory access patterns and the workloads of the components inside the memory. Use of this model demonstrates that the expected static stress for the SRAM cells is non-negligible and, thus, a mitigation scheme is required (also proposed in this thesis).
- **Software-based mitigation scheme:** This paper proposes a mitigating scheme implemented in software. Using this software scheme makes it possible to reduce the BTI induced ageing effect in the SRAM cell and Sense Amplifier (SA). This mitigation scheme reduces the BTI in the memory cells up to 2.5 times, and in the sense amplifier, the offset voltage is reduced up to 50%.
- **Hardware-based mitigation scheme:** The authors propose a mitigating scheme implemented in hardware, making it possible to reduce the BTI induced ageing effect in the SRAM cell and Sense Amplifier (SA). This mitigation scheme reduces BTI in memory cells up to 45 times and reduces the offset voltage in the sense amplifier up to 67%.
- Validation of mitigation schemes on a RISC-V platform for real applications.
- Improved ageing modelling over state-of-the-art: instantaneous or short-term BTI effect is included.
- Comparison between software-based and hardware-based mitigation schemes.
- The usage of BIST for mitigation of ageing within SRAM is not feasible.
- Using sensing to improve software mitigation scheme is not workable due to inaccurate and run time constraints.

1.4. Outline

1. Chapter 2 provides background on the SRAM design.
2. Chapter 3 discusses the reliability failure mechanisms in CMOS, the prior work on SRAM reliability modelling, and the state of the art of SRAM ageing mitigation.
3. Chapter 4 presents the developed mitigation.
4. Chapter 5 supplies the experimental results. Here, the authors evaluate and compare the performance of both the software- and hardware-based schemes.
5. Chapter 6 gives the conclusion of this work and suggestions for future work.

2
SRAM Design

This chapter provides background on SRAM design.

1. It gives a general overview of memories in computers.
2. It discusses the SRAM organisation and, subsequently, it discusses the most important SRAM metrics.

2.1. Computer memory

There are two categories of computer memories: *volatile* and *non-volatile*. Volatile memories require power to retain their stored data. Conversely, non-volatile memories do not require this; they keep their data even when they are powered down. A disadvantage of non-volatile memories, however, is that they are slower than volatile memories. Therefore, for direct computations, ICs or processors use volatile memories, while for long-term storage they use non-volatile memories.

The two most commonly used volatile memories are DRAM and SRAM. DRAM was the main memory of computers and processors in the early stage of computing. However, the speedup of processors created a *memory gap*. This means that DRAM could not keep up with the speed of the processors [40] as illustrated in figure 2.1. The speed of the processor increased at a faster rate than that of the memory. To combat this memory gap, caches were introduced. These caches are small(er) memories close to the processing unit that cache data for fast access. SRAM typically implement these caches, which is faster than DRAM [41].

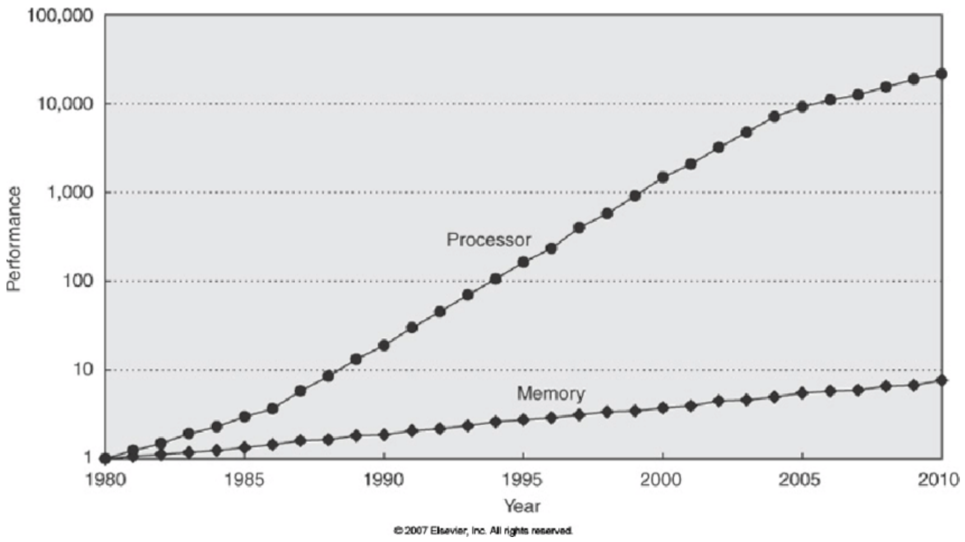


Figure 2.1: CPU-memory performance gap increasing over time, showing the need for faster and improved cache policies. Figure 1 from [1]

Fast memories need to be used, to counter the memory gap, otherwise the processor would be waiting for data. Fast memories are available in the form of SRAM or DRAM. An overview of these memories is shown in table 2.1 [42]. Caches exploit the temporal locality and spatial locality to bridge the memory gap. Temporal locality refers to the fact that if a value has recently been accessed, the chances are high, it will be used again in the near future. Where as spatial locality refers if a certain value is accessed, it is expecting values close to this value will be accessed next (e.g. a for looping traversing an array). As can be seen in the table the faster the memory the more expensive the memory becomes. Moreover, RAM is volatile and does not allow for permanent storage of data, which is done on flash that is much slower.

SRAM creates the L1, L2, L3 caches in the processor. The size of the memory cells limits the cache size. The bigger the cache becomes the slower the access will become [42].

Table 2.1: Comparing the different types of memories used in current computers

Parameters	SRAM	DRAM	Flash
Volatile	Yes	Yes	No
Cell size	Six transistors	Single transistor and a capacitance	Single transistor
Access time	0.5 ns to 2 ns	50 ns to 70 ns	5 µs to 50 µs
Write power	Low	Low	High
Price per gb (2012)	500$	10$	1$

2.2. SRAM organization

This section discusses the organisation and design of SRAMs. SRAM designs consist of multiple components, each taking care of a specific function within the memory. Figure 2.2 diagrams a typical memory design. The inputs consist of control signals to select whether a read or write operation (*r/w*) should occur and an *enable* signal to enable the operation of the memory. Other input and outputs are address lines and the data lines *Data In*. The output signals consist of only data lines, namely *Data Out*. Some designs combine them with the input lines. Combining the input and output lines reduces the number of pins on and the size of the memory chip, which makes the memory component cheaper.

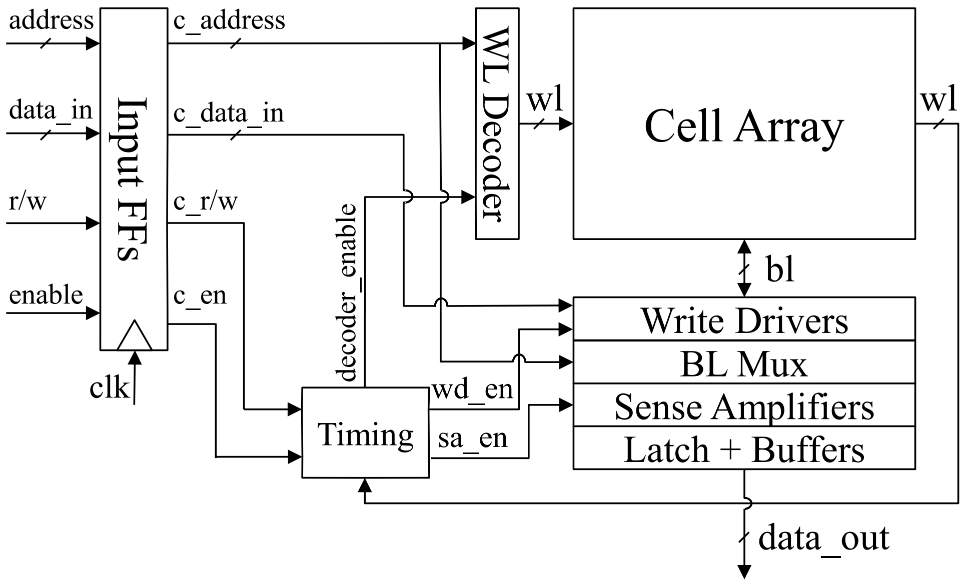


Figure 2.2: Functional model of an SRAM component. Based on figure 1 from [2]

Figure 2.2 also shows the different memory that an SRAM module is typically composed of, which are as follows:

- **Memory cell array:** responsible for storing the bit values of a word.
- **Address decoder:** responsible for selecting the right memory cells during a read or write operation. It typically consists of a column and row decoder.

- **Sense amplifier:** responsible for converting the voltage, returned by the memory cells, to the corresponding bit values.
- **Write driver:** responsible for writing new values to the selected memory cells.
- **Data-Out and Data-In registers:** buffer to stabilise the in-and-output values.
- **Timing:** will generate the control signals to enable the write or read circuit at the right moment in time.

A brief description of each component follows.

2.2.1. Memory cell array

The memory cell array handles storing the data. It is composed of memory cells, which are the essential part of any memory. The SRAM memory cells are based on *bistable* circuits. The conventional SRAM cell uses six transistors (i.e. the 6T SRAM cell) shown in figure 2.3. It consists of a cross-coupled inverter pair (M_1 and M_2 and M_3 and M_4) which serves as the bistable element.

When writing to or reading from the memory cells, the WL must be driven high, ensuring access through pass transistors M_5 and M_6 to the cross-coupled inverters. Writing to a memory cell is then done by driving the values on BL and $\overline{\text{BL}}$. During the write operation the bit lines must be driven with enough power to overcome the cross coupled inverters. As a result, the cross-coupled inverters will flip the stored values and, thus, a new value is successfully written to the cell. Reading data from SRAM is done by first pre-charging the bit lines to a high value. After enabling the word line, the inner nodes Q and $\overline{\text{Q}}$ will be connected to the BL and $\overline{\text{BL}}$. As a result, the memory cell will then pull one of the bit lines low. Afterwards which the read circuit can amplify the read value. The Sense Amplifier performs this amplification; which is discussed in section 2.2.3.

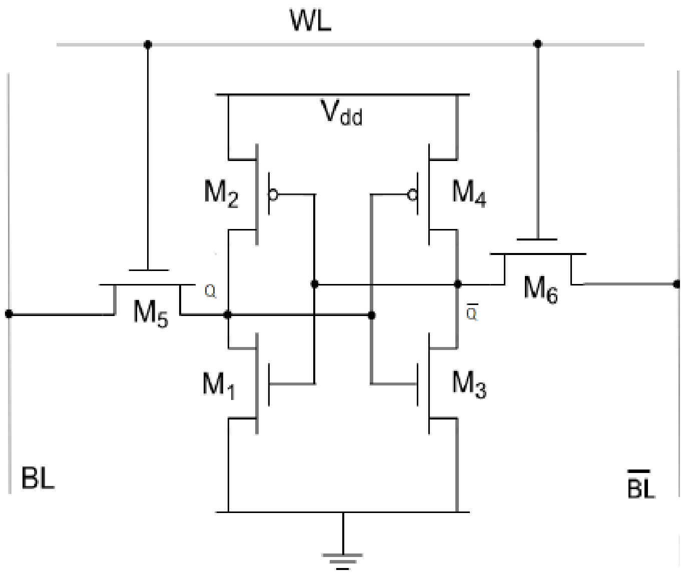


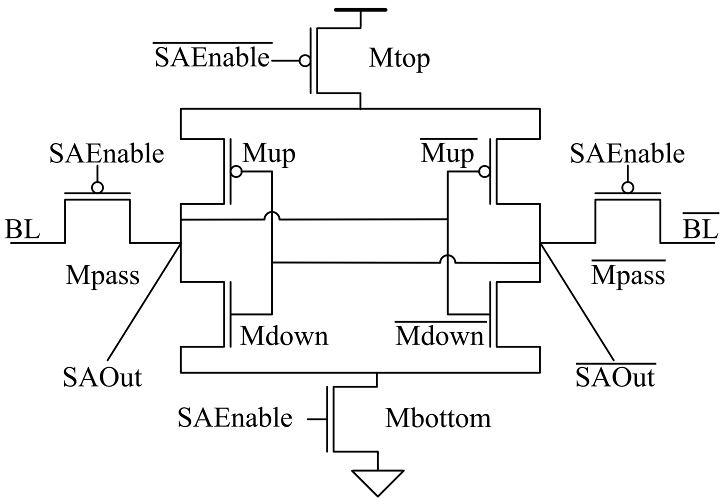
Figure 2.3: A conventional six-transistor SRAM cell

2.2.2. Address decoder

The address decoder decodes the input address, such that the read or write operation is performed for the correct memory cells. Typically, the memory array is divided into selectable rows and columns. A row decoder selects the rows, also often called the WL, and the column decoder selects the columns. Therefore, memory addresses are typically divided into high and low order bits. The advantage of using both a wordline decoder and a column decoder is that the individual decoders require less hardware and are, thus, faster. The higher order bits are typically used by the row decoder and the lower order bits by the column decoder.

2.2.3. Sense amplifier

The Sense Amplifier (SA) is responsible for the memory's read operation. The SA amplifies the small voltage difference at the bit lines generated by the memory cells. Figure 2.4 shows the schematic of the standard latch-type Sense Amplifier, which is a popular design and will, therefore, also be the focus of this work. The operation of the SA consists of two phases. The first phase is the sensing phase, followed by an amplification phase. The sensing phase will read the voltages of the BLs. In this phase, signal SAEnable is low which activates the Mpass and the $\overline{\text{Mpass}}$ to pass through the voltage of the bitlines to the SAOut and $\overline{\text{SAOut}}$. The second phase is started by the timing component which will make sure the SAEnable signal is high when disconnecting the SA from the bitlines and enable the cross-coupled inverters. The cross-coupled inverters will then amplify the voltage difference between SAOut and $\overline{\text{SAOut}}$. After completion the full-swing read value is available on SAOut and its inverse at $\overline{\text{SAOut}}$.


$\overline{\text{SAEnable}}$
Mtop
SAEnable
Mup
$\overline{\text{Mup}}$
SAEnable
BL
$\overline{\text{BL}}$
Mpass
$\overline{\text{Mpass}}$
Mdown
$\overline{\text{Mdown}}$
SAOut
$\overline{\text{SAOut}}$
SAEnable
Mbottom
Figure 2.4: Sense Amplifier

2.2.4. Write driver

The write driver manages the memory's write operation. It achieves this by driving the bit lines. Figure 2.5 shows an example write driver. The first stage of the write driver is receiving input on the Data_in line after which Data_in will pass through one inverter to create an inverted signal $\overline{\text{Data\_in}}$ for the $\overline{\text{BL}}$. This signal will then pass through a second inverter to create the signal for the BL. The control circuit will then finally enable the write driver by toggling the Write_enable signal high. The Write_enable will let the transistors Q1 and Q2 pass the value towards the bit lines which then will propagate to the memory cell. Strong inverters are used within the write drivers to ensure that it can drive the high parasitic capacitance of the bit lines.

2.2.5. Timing circuit

The subcomponents of the memory require several control and timing signals to make the complete component work. The timing circuit controls the memory to allow every subcomponent to work in harmony. These signals are all generated by the timing component.

2.3. SRAM metrics

This section gives a brief overview of the memory's metrics. Figure 2.6 shows a classification diagram of the different SRAM metrics. The SRAM metrics can be divided into *functional* and *parametric* metrics, as proposed in [3]. More detailed subdivisions for both the parametric and functional metrics are given in figures 2.7 and 2.8, respectively. The figures shows the different metrics for every component of the memory. The division into functional and parametric metrics is motivated by [43]. Functional reliability refers to the correctness of the system (i.e. the system gives the correct output values). For example, a bit flip in an SRAM cell is a functional error. In contrast, the parametric metrics evaluate the performance of the memory which does not directly affect the correct functionality of the memory. An example of a parametric metric would be the speed at which the memory cell can discharge one of the bit lines. This study focusses on mitigating the ageing of the memory cell and the sense amplifier. Therefore, the authors will only discuss these metrics.

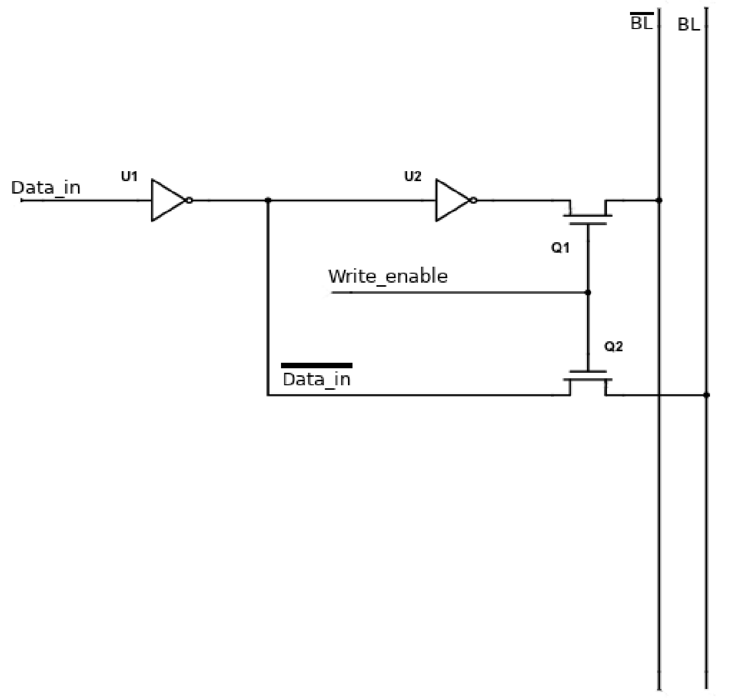


Figure 2.5: Simple write driver showing how enough drive strength can be created to successfully change the value of the bit lines.

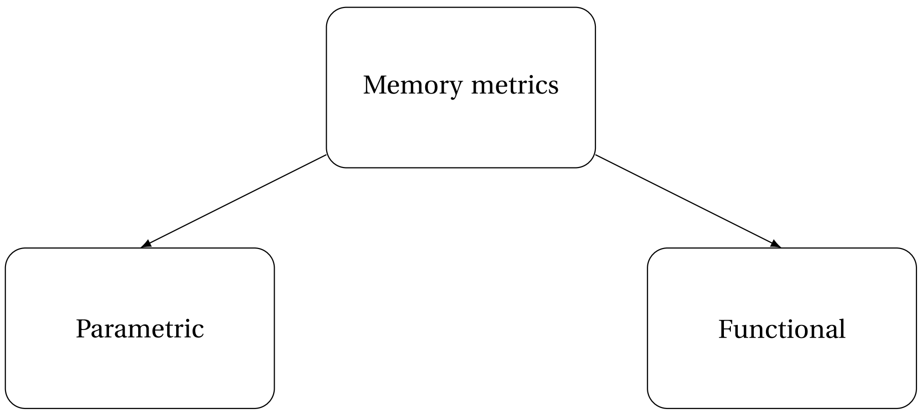


Figure 2.6: Classification diagram of the different memory metrics, based on [3].

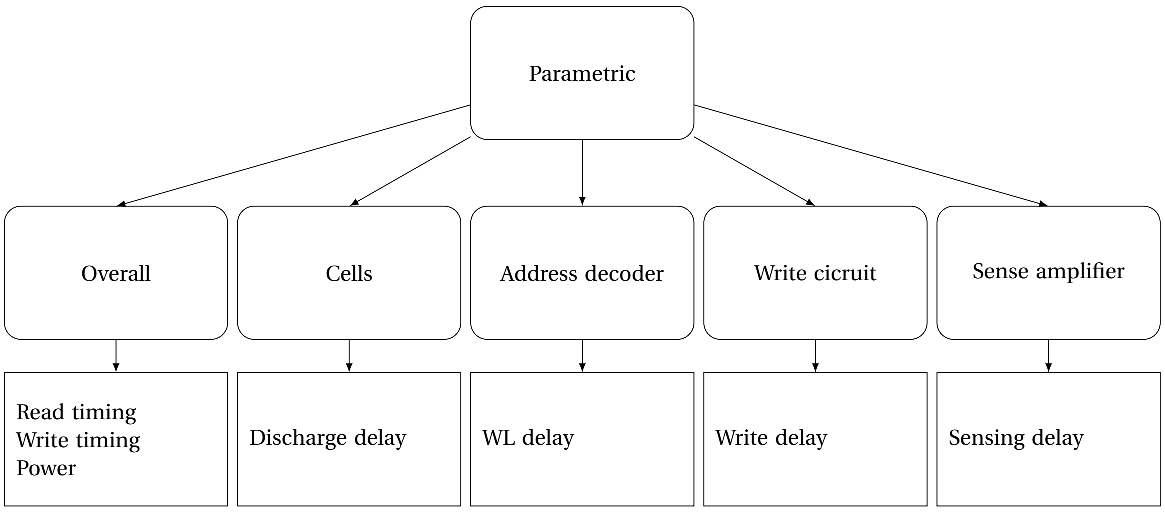


Figure 2.7: Classification diagram of the different parametric memory metrics [3]. The round boxes denote the components, while the square boxes at the bottom list the corresponding metrics.

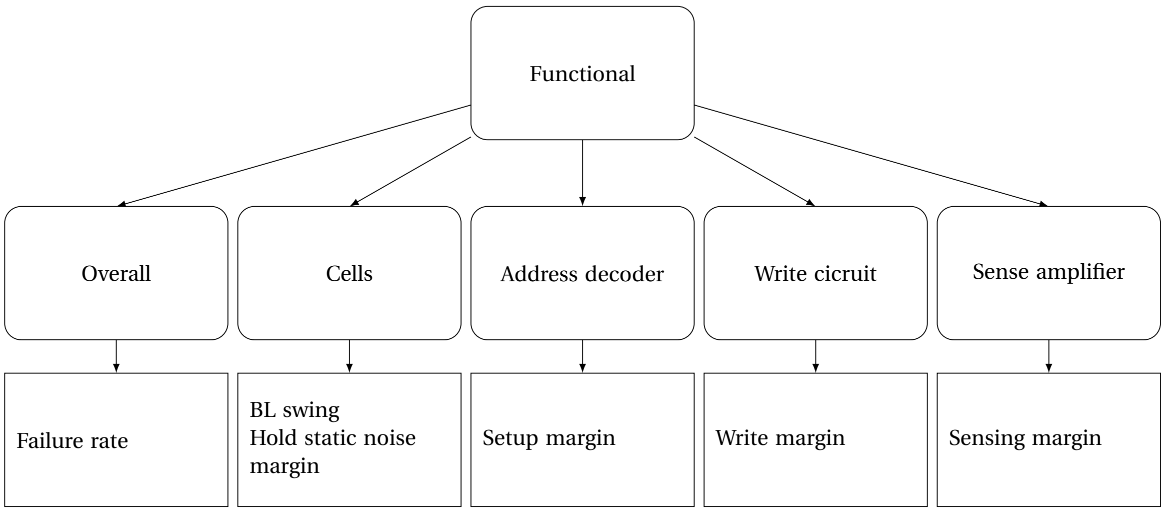


Figure 2.8: Classification diagram of the different functional memory metrics [3]. The round boxes denote the components, while the square boxes at the bottom list the corresponding metrics.

2.3.1. Memory cell
The memory cell is responsible for storing a binary value, which it needs to maintain reliably over time. Additionally, it needs to allow the value to be read without causing any bit flip. Hence, the memory cell needs to be stable. It also needs to generate a sufficient bitline discharge, such that the sense amplifier can amplify this value into a full-swing logic value. Figure 2.7 illustrates that the memory cell's parametric metric is the discharge delay. The functional metrics of the memory cell are its BL swing, its hold-static noise margin, and its read-static noise margin, as shown in figure 2.8. Next, the authors will discuss both parametric and functional metrics of the memory cell.

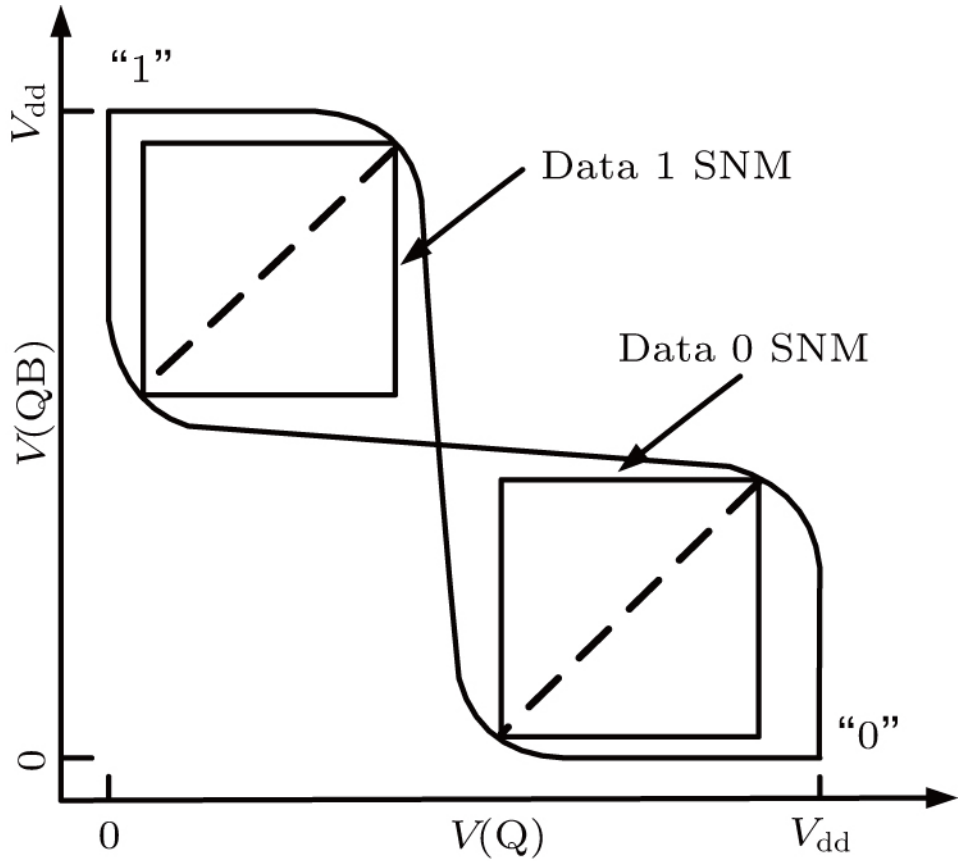


Figure 2.9: Example voltage of a memory cell showing the hold and read static noise margins.

Parametric metrics
The memory cells read and write values into memory. For the memory cells the discharge delay is the metric used to define the parametric metric. This delay defines how long the sense amplifier needs to wait before it can start to process the voltage difference on the BLs. Figure 2.10 shows the discharge delay. The delay denotes the duration after the wordline was activated and one of the two BLs is discharged for 10%.

Functional metrics
The functional metrics for the memory cell is how much noise the memory cell can tolerate during a read operation. The SNM is shown in figure 2.9. The read SNM decreases compared to the hold SNM because of writing the cell value to the BLs, called the *hold SNM.* The write performance of the cell is not taken into account. The memory cell must be able to be written to, which is tested at the production of the memory wherein the write performance improves over time [44]. Since the cell writes performance improves, it is not interesting to measure this during the evaluation of the ageing of the transistors.

2.3.2. Sense amplifier
After the memory cells have slightly discharged one of the two BLs, the sense amplifier will use the voltage difference as an input to convert into digital value to be used as the output of the memory. The sense amplifier

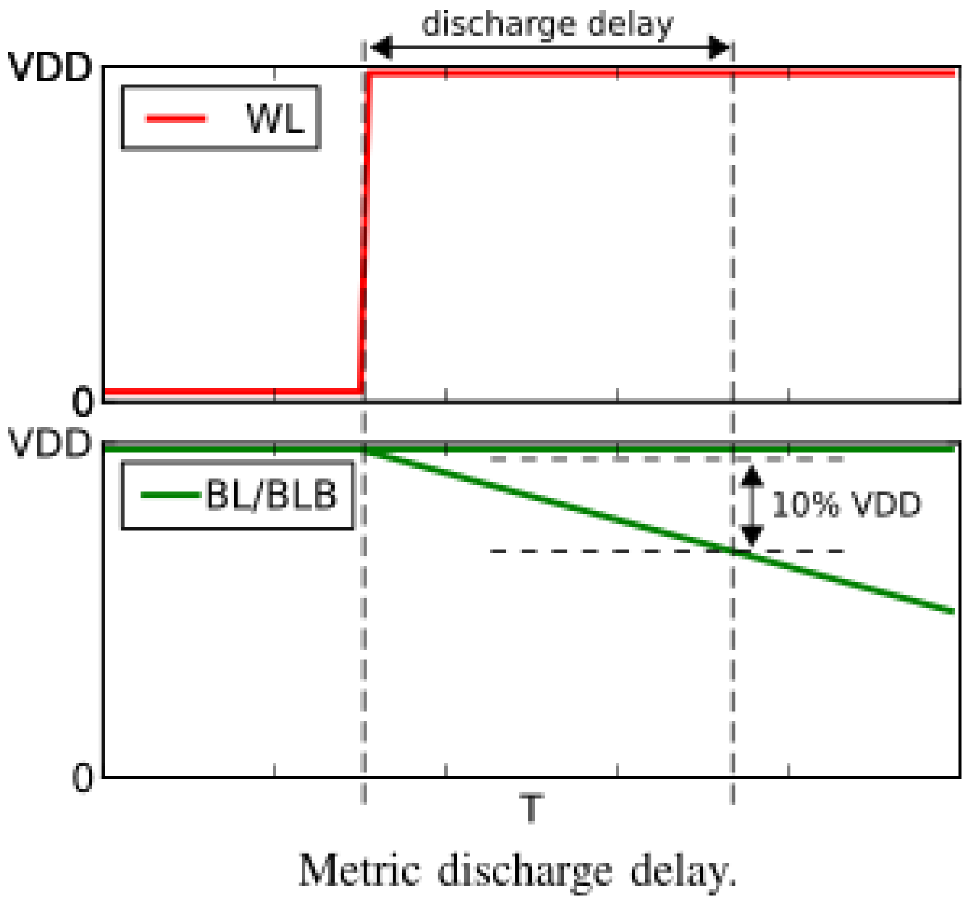


Figure 2.10: Discharge delay of the memory cell.

is, ideally, balanced and thus will convert a negative voltage difference to a zero value and a positive voltage difference to one. However, if the offset voltage of the sense amplifier is not zero, it will have a preference to convert to one of the two bits values. In most cases, the offset voltage is not zero due to ageing and process variations [45]. Figure 2.11 presents the offset voltage effect as an added voltage source on the input line of the BL. The added voltage source will reduce the voltage difference between BL and $\overline{\text{BL}}$. If the voltage difference comes under the voltage threshold which the sense amplifier requires, the result is an incorrect output value.

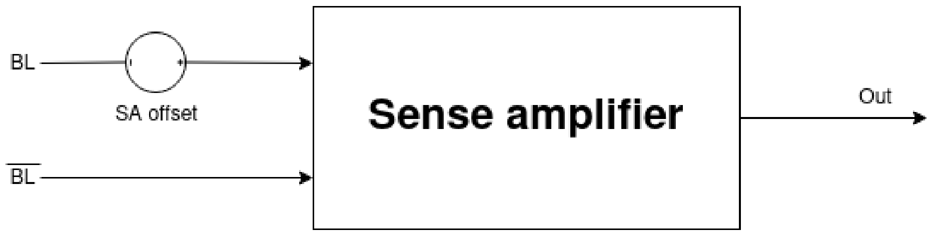
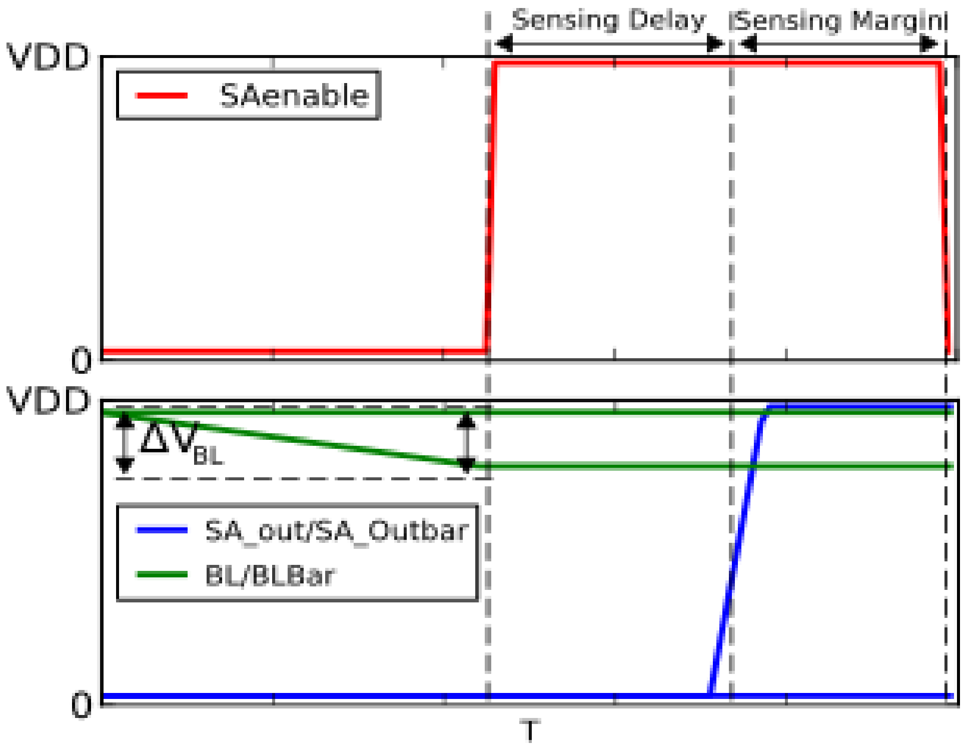


Figure 2.11: Offset voltage model for the sense amplifier an extra voltage source is added on the BL input. The added voltage source can reduce the voltage difference between the two BLs this voltage source can then result into an incorrect read value on the output.

Parametric metrics

The sense amplifier will read the input from the BLs and convert this value into a binary value which the processor can use. For the parametric metric of the sense amplifier, one needs to look at the *sensing delay*. The *sensing delay* is related to many factors. In figure 2.12, the *sensing delay* is shown together with the sensing margin. The sense amplifier is enabled by the timing circuit when the difference between the two bit lines is larger than 10%. The *sensing delay* is then the amount of time it takes to reach half a swing.



Metrics sensing delay, sensing margin, and bitline swing (ΔV_{BL}).

Figure 2.12: Sensing delay and sensing margin of the sense amplifier.

Functional metrics
As the sense amplifier reads a voltage offset from the two BLs which is relatively small, the sense amplifier needs to convert this value to the correct value. If the sense amplifier would have too big of an offset on the decision threshold it might decide to convert the voltage difference to the wrong value. The offset voltage is the most important metric and will be the only metric used for the sense amplifier in this work.

3 Overview on SRAM Reliability Modelling and Mitigation

This chapter discusses reliability issues and application to SRAM. It also explores the state of the art on how to mitigate reliability issues:

1. The authors discuss the different failure mechanisms in section 3.1, followed by a discussion on how these reliability challenges and failure mechanisms can be modelled for SRAMs in section 3.2.
2. Section 3.3, the authors discuss the state-of-the-art mitigation schemes and explore their advantages and disadvantages.

3.1. Failure mechanisms of transistors

In figure 3.1 a classification of the different reliability failure mechanisms is shown, based on the work of Agbo, Innocent in [46]. The reliability failure mechanisms can be divided into two categories: *time-zero* defects and *time-dependent* defects. Process variation causes the time-zero defects. Process variation can be on different scales; for example, a single chip within a wafer has a degraded performance or a complete wafer has degraded performance. Within the thesis, the focus will be on time-dependent failures. Environment-induced failures are out of scope for this thesis as they are the inputs where the system runs and cannot be reliably mitigated on-chip: voltage (for example voltage spikes) and temperature changes. As ageing failures are relatively new, the authors will list the mechanisms first, followed by an in-depth explanation of each. As shown in figure 3.1, the ageing related failure mechanisms are as follow:

- Bias Temperature Instability (BTI)
- Random Telegraph Noise (RTN)
- Time-Dependent Dielectric Breakdown (TDDB)
- Hot Carrier Injection (HCI)
- Electromigration (EM)

3.1.1. Bias Temperature Instability

Bias Temperature Instability (BTI) is an ageing mechanic which increases the absolute V_{th} value and decreased drain current of the transistors [46]. The BTI mechanism is active when the transistor is switched on. There are two different types of BTI: Negative Bias Temperature Instability (NBTI) and Positive Bias Temperature Instability (PBTI). NBTI affects PMOS while PBTI affects NMOS. BTI is the most relevant ageing mechanic in memory [47, 48]. This is because memories perform relatively little switching and, thus, some transistors are turned on (hence, the BTI mechanism is active) for long periods and, thus, degrade significantly.

While the effect of NBTI is the most significant of the two, PBTI is becoming more and more prevalent due to the scaling of the transistors [17]. As said in [14, 15, 44, 49–51], the effects of BTI on the lifetime is a function which takes into account duty cycle, temperature, signal probability (zeros, ones, state switches) and the supply voltage (V_{dd}). BTI consists of two phases a recovery and a stress phase. The effect of these phases are shown in figure 3.2. Over time the BTI effect builds up due to partial recovery.

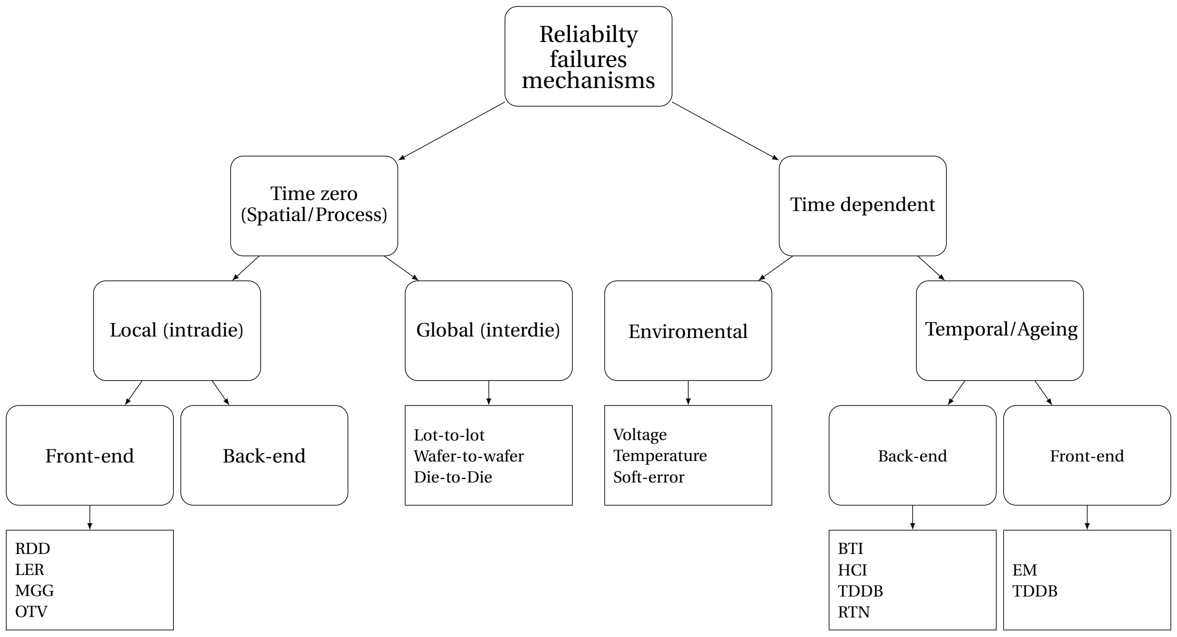


Figure 3.1: Reliability failure mechanisms classification

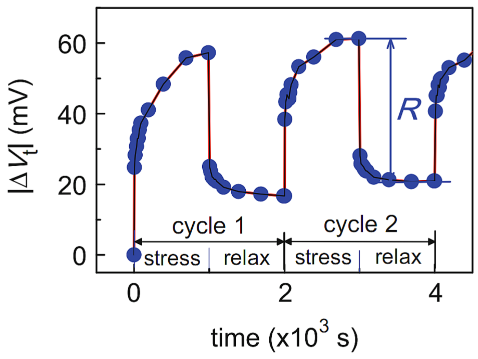


Figure 3.2: Bias Temperature Instability voltage threshold shift during stress and relaxation. Figure based on [4]

In [52], the two models are compared. They show that while the atomistic trap model is more accurate than the Reaction-Diffusion (RD) model, its simulation time is limited to a scale of just seconds. The RD model is less accurate but is good enough to show the ageing over a long timescale. The author suggests using both models in conjunction to allow for faster simulation. The physics of BTI are not fully understood as two models are used to describe the BTI-effect.

Reaction-Diffusion Model

The Reaction-Diffusion (RD) model focuses on the breaking and healing of Si–H bonds [53]. The reaction part of the model focuses on the chemical reaction of the Si–H bonds at the interface. During the stress phase of BTI, these bonds break; during the relaxation phase, these bonds can recover. The second part of the RD model focuses on the diffusion of the hydrogen atoms. The bondage breakages in the Si–H result in dangling bonds in the silicon oxide which will create a threshold voltage shift due to the trapping of charge at the silicon interface [54]. When removing the electrical field, some of the hydrogen atoms will diffuse back, resulting in a reduction of the voltage threshold shift [55].

Atomistic Trap Model

The RD model can be inaccurate in [56]. To improve the accuracy of the BTI modelling the atomistic trap-based model was introduced by [57]. The model is based on the capture and release of single traps. These traps or defects are created because of the production process. Each trap introduces an δV_{th} which can restore itself overtime during the relaxation phase [52].

3.1.2. Random Telegraph Noise

Random Telegraph Noise (RTN) is a stepwise ageing effect due to which carriers are injected into the oxide layer. The RTN occurs as random events over time and it is unpredictable when it will occur. In [58], RTN is described as traps creating leakage currents. Traps formed in the high-k material create this leakage current. As more traps build up over time, a hard breakdown will eventually occur. RTN is a switching phenomenon [59]. Since memories are mostly static, the effects of RTN will be limited.

3.1.3. Time-Dependent Dielectric Breakdown

In [60], the process of Time-Dependent Dielectric Breakdown (TDDB) is described as when the device is under a constant electric field, but less than the material breakdown field strength, the transistor gate-oxide will still breakdown over time. TDDB consists of three-phases [55]. At first, a soft-dielectric breakdown can occur followed by a progressive-dielectric breakdown. At the last stage, a hard-dielectric breakdown occurs. Soft-dielectric breakdown causes partial loss of the dielectric properties, resulting in lower gate currents compared to the next stages. The progressive-dielectric breakdown can be detected by a slow increase in gate current over time. When the hard fault occurs, the gate current rises to the mA at the standard voltage levels [55].

3.1.4. Hot Carrier Injection

HCI is caused by the acceleration of carriers (holes or electrons) under the force of the electrical field at which the momentum is high enough that the carriers break the barriers of surrounding dielectric and get trapped in the gate and sidewall oxides [61]. The acceleration and breaking of bonds are visualised in figure 3.3.

Hot Carrier Injection is an ageing effect which will increase the V_{th}. The V_{th} shifts due to the injection of carriers into the gate. This injection happens when the gate voltage of the transistor switches. In [48] Hot Carrier Injection (HCI) is described in relation to the effects of BTI in memory. This study reveals that HCI happens when a bit is flipped (and, thus, the gate voltage switched), while BTI is a static ageing mechanic. The results of [48] reveal that adding HCI simulation to the model yields a slight improvement in the performance of the memories over time.

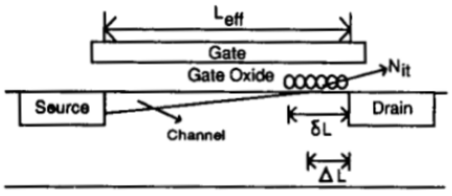


Figure 3.3: Hot Carrier Injection bond breakage towards the end of the channel. Figure from [5]

3.1.5. Electromigration

Electromigration (EM) is an ageing failure mechanism for the interconnects in the chip [62]. Due to the increasing current densities within the wires on the chip, metal ions can be displaced. The displacement of the metal ions results in voids and hillocks. In figure 3.4, an example is shown of the effect of EM, creating the voids and hillocks. The voids within the wire will create open connections while the hillocks can create shorts. In [63] several mitigation schemes for EM are described. Mitigation of electromigration can be done either by taking it into account during the design or by using bi-directional currents [64] during the operation of the IC. A disadvantage of bi-directional power is that the design requires changes to accommodate for this.

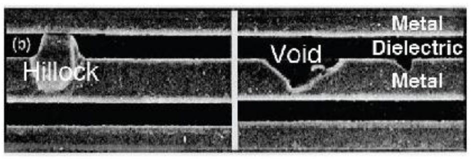


Figure 3.4: The result of hillocks and voids created by electromigration. Figure from [5]

3.2. SRAM Reliability Modelling

This section briefly discusses state-of-the-art in SRAM reliability modelling. This modelling is important as it allows one to estimate the impact of ageing during the design and, proper measures can be taken. This will lead to a more reliable design at reduced costs. Figure 3.5 gives an overview of SRAM reliability modelling. The, SRAM reliability modelling consists of three major parts:

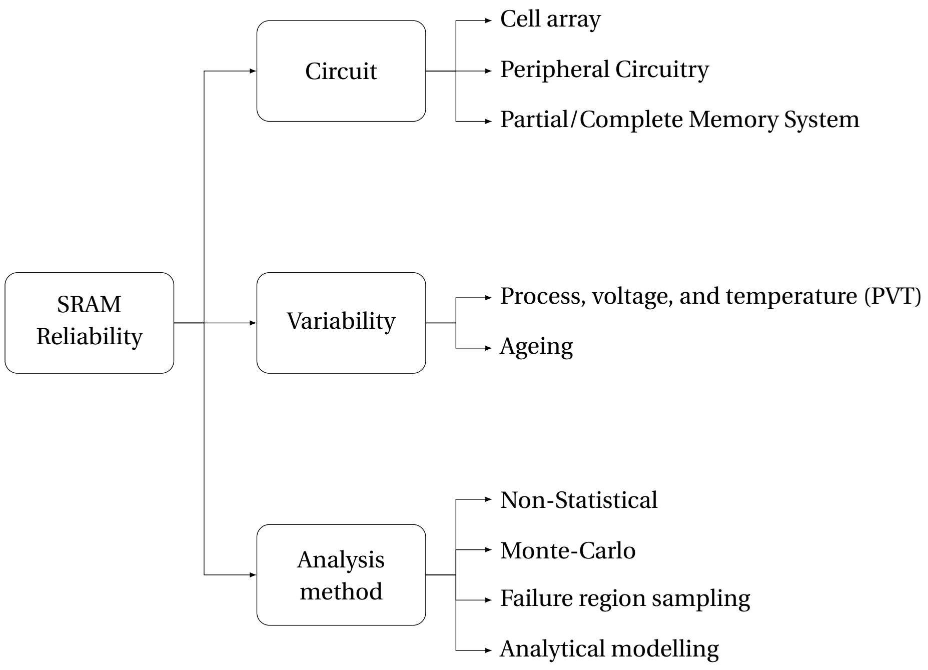


Figure 3.5: Reliability flow of SRAM

- **Circuit:** this can be either a single SRAM cell, one of the peripheral circuitries or a partial or complete memory system.
- **Variability:** several types of variability have an impact on the memory system. These are process, voltage, and temperature (PVT), and ageing.
- **Analysis method:** different methods exist to estimate the reliability of a system. Each model comes with its advantages and disadvantages. The different methods consist of non-statistical methods, Monte Carlo simulations, failure region sampling, and analytical modelling.

This thesis will discuss the memory cell reliability and peripheral circuit containing the read-write circuitry.

3.2.1. Memory cell array

Most of the prior works focused on modelling the reliability of the memory cells, specifically on PVT. The most used simulation method is based on the Monte Carlo approach. Less work exists on ageing modelling of the memory cells. This thesis will focus on the ageing aspect due to BTI, which is the most important ageing effect in transistors. Most studies use non-statistical methods to estimate the impact of BTI. The non-statistical approach usually underestimates the ageing effects. In the papers [65–67] ageing impact is analysed in the memory cells. Of these papers, only [67] analyses the failure impact due to ageing in SRAM cells. They use a non-Monte-Carlo simulation method which looks at the possible V_{th} shifts. This method allows for a more accurate simulation of BTI impact compared to the old statical method of expected V_{th} shift. This allows an accurate representation of when a memory cell could fail if the correct workload is taken into account.

3.2.2. Peripheral circuitry and partial or complete memory system

Significantly less work exists on the reliability modelling of the memory's peripheral circuitry compared with the memory cell. Within the works of [68, 69] ageing of the sense amplifier has been analysed. Within [70, 71] the write driver and the ageing of the timing circuit are analysed.

There is even less research on the reliability modelling of the partial or complete memory system. In [72], an analysis is made how ageing impacts the cell and Sense Amplifier, including their interactions. In [73], a complete SRAM circuit is analysed. These works are limited; by that the authors do not examine how the individual components contribute to the ageing.

3.3. SRAM Mitigation

In section 1.2, the study investigated several mitigation schemes, but it has not been translated to mitigations for SRAM. This section will go into more detail on how mitigation schemes can be applied to SRAM.

An overview of the different mitigation schemes is given in figure 1.2. There are two categories of the mitigations and adaption techniques: mitigation at design-time and mitigation at run-time.

3.3.1. Mitigation at design-time

These schemes are static and can be implemented in different ways. The first way of mitigating the effects of ageing is by adding extra hardware to create a margin for chip degradation [12, 13]. These novel implementations are known as a worst-case design [13] and can be improved upon by looking at [74] and running a better-than-worst-case design. The better-than-worst-case design can be used because chips will not always be under maximal stress.

Worst-case design strategy

The method worst-case-design requires the usage of extra hardware either by resizing the size of transistors or reducing the operating frequency of the chip which creates a margin where the chip can accommodate the effects of BTI. Applying worst-case-design to SRAM is highly inefficient and costly as the memories consume the largest part of the die. Most added silicone would not be used during the lifetime of the chip.

3.3.2. Better-than-worst-case design strategy

The static methods require advanced knowledge of the workload and analysis of the stresses occurring in memory, to improve on the method worst-case-design. In this scheme on memory, while having the most ageing, can be resized either by creating unbalanced cells, so the cell is tailored more towards the storage of a zero or a one-bit value. This technique is introduced in [75] to counter the differential ageing of SRAM-cells. It will reduce the required hardware overhead compared to worst-case-design [14, 15]. Mitigating the ageing effects of transistors running different workloads require having different library cells for each expected lifetime and load [16]. For SRAM, it would require to have different memory cells within the cell array to handle the unbalanced workloads. This will create an inefficient, memory, cell-array structure. If the workload behaves differently from the expected workload, it can create a much faster degradation in the memory cells compared to not having implemented any mitigation scheme. This will be the case if the workload simulation suggests placing smaller, weaker cells at a certain location, which will then result in having a higher than average static stress [17]. As staid in the state of the art, the halting problem does not allow solving the problem, not knowing upfront how the memory stresses will be distributed.

3.3.3. Mitigation during run-time

During the run-time of a chip, the usage of dynamic schemes is possible to reduce the BTI-induced ageing effects within the memory. These dynamic schemes will apply resource management either to reduce the amount of static stress, which occurs or to relax the parts of the memory which are under the most stress. There are different techniques, which can mitigate BTI-induced ageing effects these are:

- Idle time leverage (ITL)
- Input vector control (IVC)
- Controlled resource wear out
- Spatial redundancy

Idle-time leverage and input-vector control

As a chip usually does not use all resources at all times due to stalls (structural-, data- and control -hazards) and a program does not always require maximum performance, analysis of programs is made in [76]. Within the time that a CPU has a stall or is not fully used, recovery schemes can be executed to reversed to mitigate the effect of BTI. Schemes taking advantage of this are of the category ITL. ITL scheme can be extended with the usage of IVC to control the mitigation of, for example, BTI-induced ageing.

Input vector control (IVC) can be an effective technique to reverse and mitigate the effect of BTI in the chips. The mitigation scheme can either run in available idle or by stalling the execution of the program. The zero-value cause in the PMOS transistors creates BTI-wear [77]. To determine which bit value should be used in idle time requires estimating the degradation of transistors. The process of estimating the degradation is hard due to variation in the operating temperature and settings. In [78], artificial intelligence techniques are used to find an optimal program which yields the best BTI rejuvenation program to be run on the CPU.

IVC can be applied by the usage of NOPs within the CPU. The usage of alternative NOP instructions has been analysed in [28] to see if this can reduce the effect of BTI-wear. When using idle instructions in the execution, a part of the work can be offloaded to the compiler, as it can detect some stalls and idle times at compile-time. The usage of IVC by using a different instruction for the NOP is not possible for the case to mitigate stress in memory cells. The NOP instruction is not allowed to have side effects and thus cannot alter a register value or change a memory value. As it is not possible to read a single memory value, relax the cells, restore the cells, and restore the register state within one clock cycle, it is not feasible for memory components.

A different technique of mitigating BTI-induced ageing bit flipping is introduced. Bit flipping is used to reduce the stress in the memory components as described in [29]. Bit flipping is a unique form of IVC as the vectors are based on the current values in memory. Flipping the bits in memory ensures the zero and one-bit value have the same occurrence chance. A software and hardware approach are discussed in [79] for the flipping of bits in SRAM. The limitations are that using the CPU to flip the bits will cost too much time for the L3 cache. To improve the results of bit flipping [80] proposes the Cell Flipping technique with Distributed Refresh phases (CFDR). CFDR reduces the time required for flipping the memory bits by changing the frequency at which individual cells are flipped. Using information from the workload allows for a smaller overhead. The disadvantage is that it requires advanced workload knowledge, and it has to consider the halting problem. It would require the ability to determine the control flow of the program upfront for all the different inputs available, to find the best solution for the mitigation scheme. In 1936, Alan Turing proved that this is not possible.

Controlled resource wear-out and spatial redundancy
As dark silicon is starting to get introduced [81] more and more parts of the chip cannot always be active due to the power wall [82]. Smart schemes can be created to add extra hardware, which can be used to extend the lifetime of chips. In [83, 84] computational sprinting is described, which allows for bursting the speed of a CPU past the thermal power limit by using a different set of cores. Since computational sprinting is used, the task is completed faster than expected. The generated slack can then be used to power-gate several parts of the chip. Power-gating will then allow for the recovery of the BTI effect.
Another option to control the wear-out of the chip is using spatial redundancy. Spatial redundancy adds redundant hardware which can be used via software or on-chip logic. For example, the scheme Logic-Wear-Levelling (LVL) [85] allows software routines to switch critical paths to the redundant paths, which are available to allow the transistors to recover.

Applying computational sprinting within SRAM only, will not work as the memory is only supplying the CPU with value when requested and is already mostly idle. It is not able to mitigate, but by just doing nothing, the bit value will be maintained in the memory cell. Using the advantage of dark silicon within SRAM would be possible. For example, when a part of the memory breaks a smart control logic can replace the faulty word or use a logic scheme to alter the write locations of a word from the main memory to the dark silicon. When making use of the dark silicon, this will not limit the usage of a software mitigation scheme or a hardware mitigation scheme, which will try to mitigate BTI-induced ageing actively.

4 Mitigation Methodology

This chapter will introduce mitigation schemes:

1. Section 4.2, the authors created and analysed a model for static stresses.
2. Section 4.3 uses the results from the modelling of the stresses in the memory to discuss different possible mitigation schemes.
3. Section 4.4 and section 4.5 examines software, and hardware mitigation schemes and the authors calculate how feasible each scheme is.

4.1. Static stress

When looking at SRAM memory, it is important to know which static stresses occur in memory, as a single second of static stress creates a high amount of ageing. Reducing the duration of the static stress slightly already yields a large improvement. In [4], the initial recovery is discussed showing that even a short amount of relaxation will significantly reduce the BTI in the transistors. The transistor should not be stressed too long, as that will greatly increase the required relaxation time, which is needed to recover from the BTI effects. In figure 4.1, the authors conduct a sweep of different duty factors showing the effects this has on the V_{th} shift. Short static stress of 1 s already changes the V_{th} shift with 4 mV. Adding a small relaxation would ensure this stress does not occur.

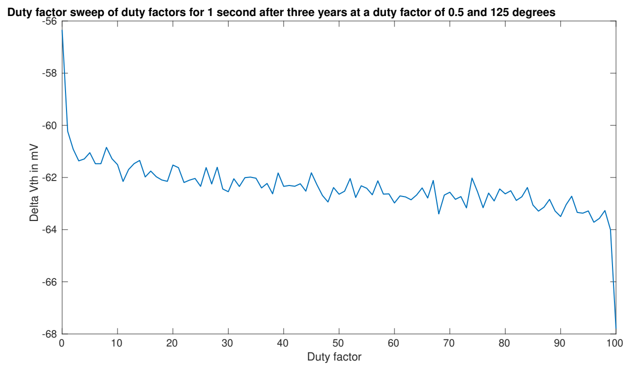


Figure 4.1: A duty factor sweep is conducted after ageing a transistor for three years at 125 °C. The ageing is extended with one year to show the effect of different stress values on the transistors. The duty factors which are at the extreme have a large impact compared to the other duty factors.

4.2. Memory access modelling

As seen in section 4.1, static stress is by far the most important stress to be considered related to the ageing of SRAM. From the literature review, [86–89], a majority of papers in literature find that on average 10% of the instructions are memory writes and that 20% of the instructions are memory reads. Using the Linux tool `perf` allows one to monitor the L1-cache usages. As expected, different workloads have different footprints in memory usage. When using the `perf` tool to monitor Libreoffice (an office suite) [1], the memory instruction is around 10% memory writes, and 20% reads. When `perf` analyses the generation of prime numbers and it is run for a to create a varying amount of prime numbers the memory writes drop to 1.2%.

If static stresses are not taken into account and assume memory writes to be random, it is possible to model this using a Poisson distribution or Markov model, allowing the creation of a model for memory usage. Markov model analyses expected duty factors, while the Poisson model analyses static stresses in the memory.

Markov model
Simulating the memory as a Markov model gives insight into the operations of memory. Figure 4.2 shows the simple cell states with the available transitions.

The different actions that can happen every clock cycle are:

- Value 0 is written
- Value 1 is written
- Cell is read
- Cell is left idle

The state of the cell will only change if a value is written, which is the opposite value of the current cell value. When taking into account the number of words in the memory and assuming all memory actions are random based. The bit distribution is estimated to be non-random and favouring a zero over a one-bit value. The Markov model can be run to simulate possible memory stresses.

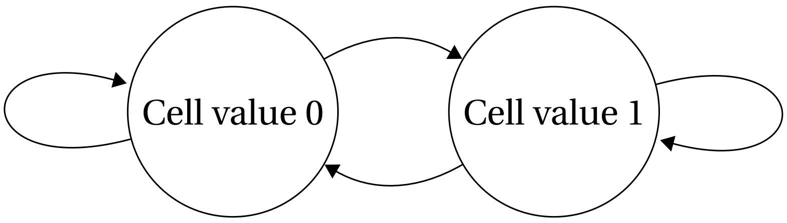


Figure 4.2: Markov model to describe the memory model

However, the Markov model is not exact, and it is computationally expensive to gather results from it. Since we are interested in how long the static stress state can be within a cell, one can look at when transitions occur. The authors suggest to use a Poisson model to calculate the chances for expected static stresses.

Poisson model
The assumption is that memory write addresses, are random, and the written value is sampled from a random distribution. This is possible because of how one uses memory, for which the law of large numbers [90] can be applied.

If one only considers the writes to the memory, this can be modelled as a Poisson process. The example from the prime number generation provides these units to use within the simulations:

- Memory size 32 KiB
- Word size 8 bytes
- Frequency 1 GHz
- $P_{write} = 0.012$
- $P_{flip} = 0.01$

$$P_{write_{cell}} = P_{write} \cdot \frac{Word_{size}}{Memory_{size}} \tag{4.1}$$

$$\lambda = t \cdot P_{write_{cell}} \cdot Freq \cdot P_{flip} \tag{4.2}$$

[1]A open office suite which is an alternative to Microsoft office

$$P_{stress}(t) = e^{-t \cdot P_{write_{cell}} \cdot Freq \cdot P_{flip}} \tag{4.3}$$

This results of an expected static stress duration which can continue up to multiple seconds. Besides the normal static stress, prolonged static stress can also occur due to non-random events in a program. For example, when encrypting a message, the encryption key will stay in the cache during the encryption processes. The extended static stress needs to be measured by using benchmarks which simulated real-world applications. Which is what we analyze in chapter 5.

4.2.1. Concluding

The random approach appears optimistic about the expected static stress duration in the SRAM cells, as non-random events are not considered, which could create static stress for much longer than a few seconds. However, static stresses of multiple seconds can occur, which could be problematic for the lifetime of SRAM. It is expected that there would also be more static values in the memory with certain workflows, such as encryption where the encryption key remains in memory for a longer time. In these situations, mitigation would be of even more importance.

4.3. Concept of the mitigation scheme

Static values induce ageing in SRAM in memory cells. The most optimal memory usage would be a perfect balance between zeros, ones, and a short duration of static stress values.

Reducing the wear and ageing effects can be done by adding mitigation schemes to improve the lifetime of the SRAM. Mitigation schemes will target the duration of static stress. By reducing the maximal static stress in a fixed amount of time, it is possible to utilise the fast initial recovery of BTI effects [57]. Reducing static stress is best done with the usage of IVC patterns, as it has a small overhead in the execution of a program and makes sure the static stress sequence is broken. Due to a small overhead, the duty factor of a memory cell does not change by much. This is shown in equation (4.4).

$$duty\_factor_{mitigated} \approx \frac{duty\_factor \cdot t_{total}}{t_{total} + t_{mitigation}} \approx duty\_factor \tag{4.4}$$

A different approach is to balance the stored value in memory such that the $P_{one} = 0.5$, which is possible by adding a large overhead with a stall or by using hardware which could use pointers to keep track of which parts of the memory are altered. Adding a large execution overhead would require doubling the execution time to reach a $P_{one} = 0.5$. As such, this is not feasible, and only the hardware mitigation scheme can generate a completely balanced memory usage. The hardware scheme can then be implemented as a cyclic flipping scheme which will be described in more detail in section 4.5.

4.3.1. BTI effect in a transistor and mitigation potential

It is important to understand the impact on a single transistor and see if a mitigation scheme will have a positive impact on the transistor, when analysing the effects of BTI. Several different experiments are run to establish the effects of the environment on the transistors. The different mitigation schemes are run on a single transistor, also analysing frequency and simulation accuracy. For the transistor, the study uses the 14 nm model from IMEC, creating the transistor workload is done in the form of CDW format. Within the CDW format the duration, duty factor, and frequency are specified. In MATLAB a modelling platform was created where mitigation algorithm could be created allowing to run several different mitigation schemes:

- Baseline
- Periodic mitigations
  - Periodic insert idle time (software)
  - Periodic flip the transistors states (hardware)

Each of these implementations will come with different amounts of execution and/or hardware overhead. The simulation will be run for three years at 125 °C. Simulating at an high temperature is the similar to a lower temperature over a longer period. Higher temperature induces faster ageing due to the sped up reaction process.

The BTI model from IMEC works in the bracket:

- V_{dd} at 0.8 V ±10%

- Time step between 1×10^{-9} s to 1×10^{8} s
- Frequency 1 GHz is realistic but in a range between 1 Hzand3 × 10^9 Hz
- Temperature −40 °C to 125 °C

Frequency impact

The impact of frequency on the ageing is related to the duration of static stress. When reducing the frequency at which the transistors run, will extend the static stress durations. If the duty factor of the transistor is either zero or one, it does not matter if the operating frequency changes. To verify this assumption, the BTI-model of IMEC is used. Table 4.1 shows the results. In the extreme case where the duty factor is one, the frequency does not matter. When the duty factor is not in the extreme, the frequency does have an impact on the degradation.

Table 4.1: V_{th} shift after three years showing the effect of using different clock frequencies on the degradation of a transistor.

Frequency	Duty factor	Degradation
1 GHz	1	−83.6 mV ± 0.42 mV
1 kHz	1	−83.6 mV ± 0.39 mV
1 GHz	0.9	−68.5 mV ± 0.64 mV
1 kHz	0.9	−71.4 mV ± 0.62 mV

Simulation accuracy

The model of IMEC allows using average duty factors to speed up the calculation of the BTI impact on the transistors. Using the average duty factor will result in degradation, which is similar (on average), but the worst-case performance could be different. The worst-case will decide at which moment the device will fail a timing constraint. This effect can be mitigated by accurately simulating the last part of the workloads. In table 4.2 this effect is calculated in more detail. The simulation accuracy is illustrated in more detail in figure 4.3 shows, which results into a difference of up to 3.6 mV after 1×10^3 s. The graph shows, the effect of fast degradation and fast recovery. To determine the ageing it is required to look at the worst-case performance and not to the average response as a metric violation will happen at the first point in time where the worst case degradation point passed the toleration value.
The MATLAB code, written and created for this project, will find the most optimal and accurate simulation of the workloads as possible.

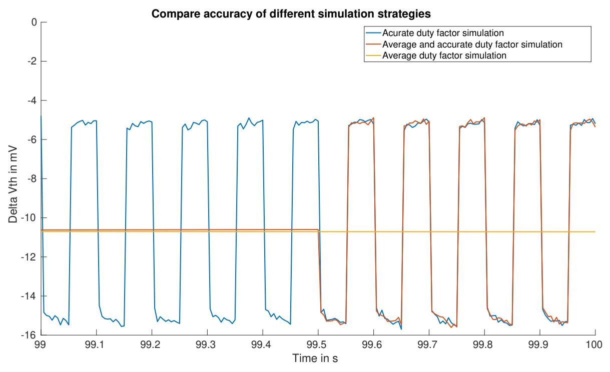


Figure 4.3: Simulating a workload with an average duty factor of 0.5. It is shown that there is a big difference in the average simulation and the other two more accurate simulations. The difference between the two more accurate simulations is due to the stochastic nature of the BTI process

Mitigation scheme performance

After setting the baseline related to the importance of how the MATLAB simulation should run and the importance of frequency, it is important to see if the mitigations schemes can work and reduce the ageing effect induced by BTI. To analyse this the hardware flipping and idle-time injection are ran and compared in table 4.3. Each scheme yields a significant reduction in BTI ageing.

Table 4.2: Maximum V_{th} shift after 100 s depending on how the simulation is run at 125 °C

Simulation	**Degradation**
Average	$-10.62\,\text{mV} \pm 0.26\,\text{mV}$
Average + accurate	$-14.82\,\text{mV} \pm 0.32\,\text{mV}$
Accurate	$-14.9\,\text{mV} \pm 0.33\,\text{mV}$

Table 4.3: V_{th} shift after three years depending on how the simulation is run at 125 °C

Mitigation	**Degradation**
Baseline	$-84.02\,\text{mV} \pm 0.40\,\text{mV}$
Hardware flipping at 200 Hz	$-66.60\,\text{mV} \pm 0.52\,\text{mV}$
Insert idle time at 200 Hz with an execution overhead of 0.01%	$-79.21\,\text{mV} \pm 0.52\,\text{mV}$

4.4. Software mitigations

Several mitigation schemes have been discussed in section 1.2, where the IVC scheme is useful and could be implemented with the usage of software routines. The IVC can be implemented in several ways resulting in different mitigation performance.

For software mitigation schemes, the different mitigation techniques consists is a periodic mitigation scheme to inject idle time either cycling through the memory or looking at the static memory addresses.

Within the software routine, several limitations need to be considered. After the software routine ran the state of the CPU and memory should be the same as before the start of the interrupt. For software, it would be acceptable to run memory mitigation up to 1% of the time. Figure 4.4 software mitigation shows how it will affect the run-time of the application. It is possible to alter the periodic at which the mitigation scheme will run the duration of the forced mitigation time. After the completion of the program, it could be possible that idle time is available for the mitigation scheme to run in. Implementing software mitigation can be done with the algorithm of **Multiple cell mitigation**. This will create a cyclic effect throughout the memory.

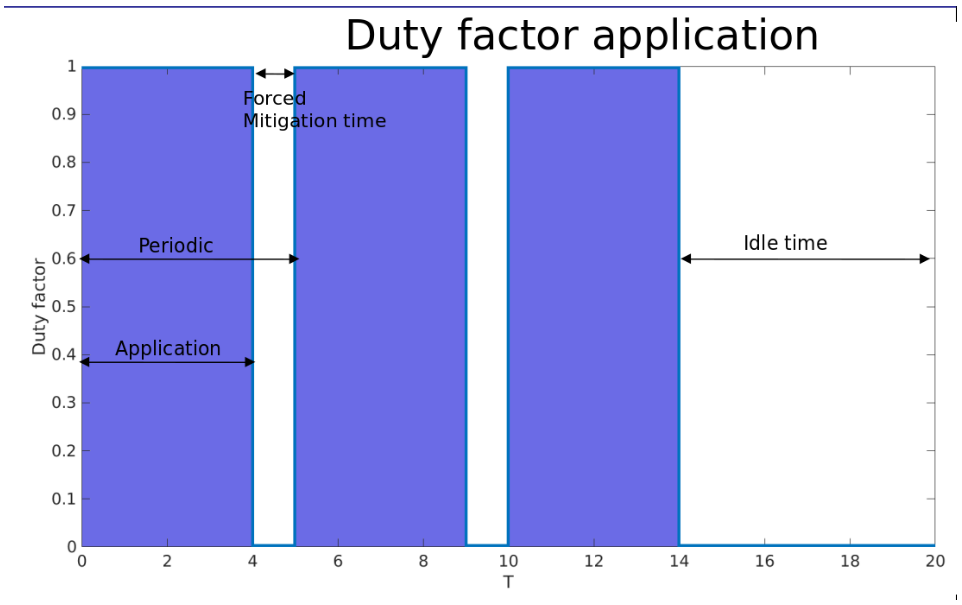


Figure 4.4: Mitigation scheme in software showing what can be fine-tuned.

4.4.1. Multiple cell mitigations

Mitigating of multiple cells during one run of the software routine can improve the performance significantly. It requires the value of the SRAM to be recoverable. This can be done by flipping the value of the cell and then writing this value back. Using the negated value of the cell also balances the overall duty factor. The used algorithm is shown in algorithm 1. The algorithm will start at the first address of the memory. When the software is interrupted. The algorithm will read and write invert values to the memory cell this will be done for half of the execution time. Once completed, flipping values for cells will start at the beginning again to restore the memory values. For the next interrupt, it will move its pointer to start at where it ended in this cyclic.

Algorithm 1 Algorithm to mitigate multiple memory cells by inverting the value and the recovery before returning from the software routine. Idx is the current index at which the routine is while N_{words} is the amount of cells can be relaxed every run.

```
$start \leftarrow idx$
for all $N_{words}$ do
    $^*idx \leftarrow !(^*idx)$                ▷ Write inverted value to memory (Relax)
    $idx \leftarrow idx + 4$                   ▷ Move to next address
end for
repeat Nop
until Cells are relaxed                        ▷ Optional relaxation time
$idx \leftarrow start$                         ▷ Restore the memory
for all $N_{words}$ do
    $^*idx \leftarrow !(^*idx)$                ▷ Write inverted value to memory (Restore)
    $idx \leftarrow idx + 4$                   ▷ Move to next address
end for
if $idx \geq Memory_{words}$ then              ▷ Bounds check to return to the start of memory
    $idx \leftarrow 0$
end if
```

To calculate the overhead and the efficiency of this mitigation scheme, it depends on several variables. At first, it needs to be defined how long the mitigation may stall the CPU. For real-time systems this will be limited. Next, it needs to be defined how long the relaxation should take. The overhead calculation is shown in equation (4.10).

$$N = \text{\# words in memory} \tag{4.5}$$

$$T_{stall} = \text{Maximum allowed CPU stall duration} \tag{4.6}$$

$$T_{relax} = \text{Defined minumum relaxation time} \tag{4.7}$$

$$Cells_{mitigated} = max(\frac{T_{stall} \cdot f}{2 \cdot Cycles_{read-flip-write}}, N) \tag{4.8}$$

$$Mitigation_{runs} = \left\lceil \frac{N}{Cells_{mitigated}} \right\rceil \cdot \left\lceil \frac{T_{relax} \cdot f}{Cycles_{read-flip-write} \cdot Cells_{mitigated} + Idle_{mitigation_{cycles}}} \right\rceil \tag{4.9}$$

$$Overhead = Mitigations_{runs} \cdot T_{stall} \cdot f_{mitigate} \tag{4.10}$$

4.5. Hardware mitigations

When using a hardware-based approach, it is possible to maintain the function of the CPU, even if the cells are in the mitigated state. Different algorithms are available which can be implemented in hardware:

- Cyclic
- Flip on write access
- Flip on read access
- Flip on replace of cache value [80].

Out of the four different algorithms, only cyclic is balanced and works by mitigating the complete memory [80]. The expected performance **Flip on x** instead of cyclic will give two Poisson processes. As stated in [91] summing, two independent Poisson distribution is the same as summing the rates. For flip on write-read access, the static stress will be reduced with a factor of $60 - 100x$. However, these schemes will not mitigate all static stresses within the memory.

4.5.1. Cyclic flipping

The advantage of cyclic flipping, as stated in [80], is a scheme, which will mitigate all the memory cells, whereas the flip on replace or flip on write access will not be able to mitigate static cells.

The flipping algorithm consists of:

- Read cell value
- Flip cell value
- Write flipped cell value

In equation (4.11) the derived formula is given to calculate the frequency at which a cell can be flipped. For an overhead of 1% with 4000 elements in the L1 cache will give a frequency of 400 Hz to 625 Hz.

$$f = \frac{overhead_{fraction} \cdot f_{clock}}{2 \cdot N \cdot Cycles_{read-flip-write}} \tag{4.11}$$

In figure 4.5, the working of the hardware algorithm is visualised how it will pass through the memory. It also shows that within hardware it is only possible to change the speed at which the hardware addresses are flipped.

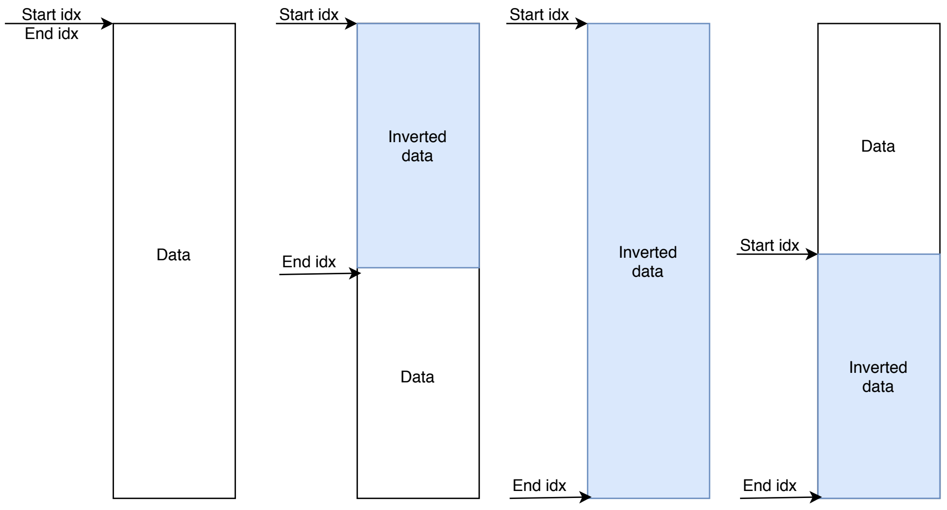


Figure 4.5: The figure gives a high-level overview of how two indexes can be used to keep track of which part of the memory is flipped or is not flipped. The read-write circuit can then use this to make sure the right value is read and written.

It is important to know at which speed this mitigation algorithm can run through the memory. The required time for a single pass is $N \cdot C_{read-flip-write}/f$, which results in to 1.2×10^{-5} s. Also note the execution time of the algorithm scales in $\mathcal{O}(Cycles_{read-flip-write} \cdot N)$. In figure 4.6 general overview is given of how a hardware implementation could roughly look like. The counter allows the flipping control unit to work with out the being triggered by read or write operations.

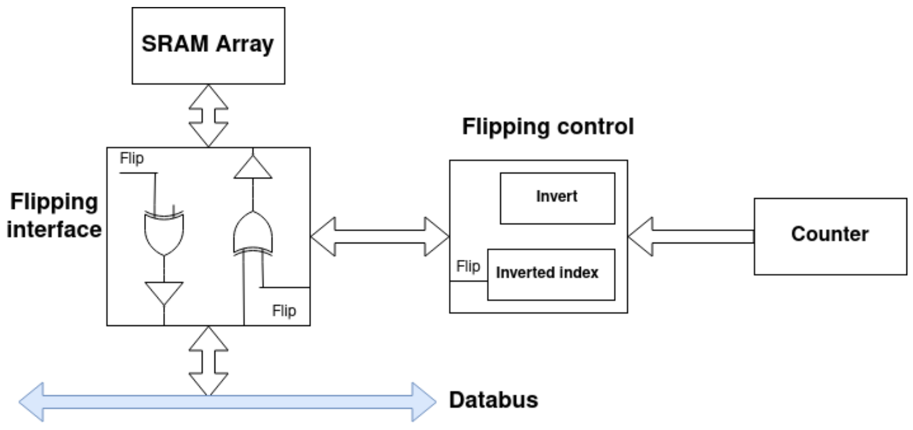


Figure 4.6: The figure gives a high-level overview of how the hardware cyclic flipping can be implemented. The Flipping interface will make use of the flipping control to know if it needs to flip the value which is being written or read. The counter will trigger the invert, which will read and write to a next memory address in line to be flipped.

5
Experimental Results

This chapter performs a case-study to validate the selected mitigation schemes:

1. The authors discuss the experimental setup, followed by what experiments will be executed.
2. The chapter analyses the hardware and software mitigations to see how the cells and SA perform.
3. The authors draw a comparison between the two schemes to see which perform better and how the overhead of each scheme affects the chip.

5.1. Simulation methodology

It needs a platform, to evaluate the performance and quality of the mitigations scheme's, where it can be run, as discussed in section 5.2.1. A further requirement is a series of benchmark; discussed in section 5.2.1. The analysis, shown in figure 5.1, has two main phases: a high-level simulation and a low-level simulation. In the high-level simulation, different benchmarks will be run on the target platform, which is under analysis. The second part is a low-level simulation where simulations will be run using the input from the high-level simulation. The low-level simulation will simulate on a single transistor level to determine the BTI-induced ageing effect.

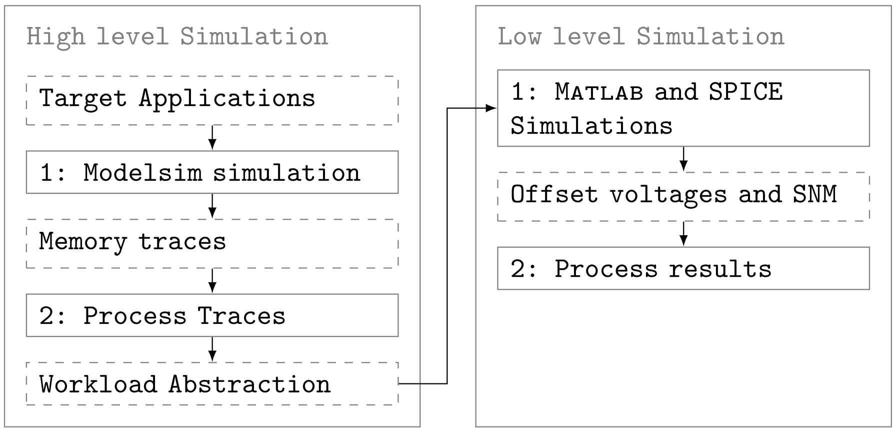


Figure 5.1: A general overview how the simulation setup works within this thesis.

5.1.1. High-level overview

The high-level simulation consists of several steps. Before the simulation can start it must create the target applications and prepare the target device. At this stage, the configuration of the type of experiment to run is done. The three different mitigation schemes are:

- **Hardware mitigation:** In this mitigation scheme, it is only possible to set up the mitigation speed.

- **Software mitigation:** In this mitigation scheme, it is possible to configure the chunk size and the interval at which the co-routine is run.
- **Software mitigation with added idle time:** In this mitigation scheme, it is chosen to fix the interval and chunk size during the run-time of the benchmarks. It will only allow configuration of the overhead, which the mitigation scheme can use, after and benchmark finishes.

After the configuration is set up, the first simulation, (i.e. the Modelsim simulation) can run. It will return as memory traces consisting of all the read and write operations to the memory. The next stage will process the memory traces to create an analysis to find the memory cells with the most static stress, and the most extreme duty factor. Next to the cell analysis, the sense amplifier is carefully analysed, and the bit distribution is calculated for each bit which is converted to duty factors using the code supplied by IMEC. The workload abstraction uses the calculated duty factors, which provide input for the low-level simulation phase.

5.1.2. Low-level simulation

The high-level simulation phase supplies the inputs for the low-level simulation phase. Two different workload abstractions have been created: one for the memory cell and one for the sense amplifier. For the memory cells, the BTI-induced ageing will be analysed in MATLAB, after which the SNM will be simulated in SPICE. A Monto-Carlo simulation analyses the sense amplifier directly in SPICE, to determine the offset voltage of the sense amplifier. After calculating the offset voltage and the SNM, the researchers can process the results to see which configuration yields the best results for the given circumstances.

5.2. Performed experiments

This section will explain the setup of the experiments and the different effects of BTI-induced ageing in the memory cells.

5.2.1. Setup

The setup of the experiments requires several different components:

- A platform on which the benchmark and mitigation scheme can run (i.e. PULPINO).
- Benchmarks which are used to create memory traces.

Benchmark platform

After introducing the overview of the simulation, it is now required to introduce the platform, which is used to run the benchmarks on PULPINO. It is required to have an implementation target device. For this, the PULPINO is used the details of the platform and implementation are discussed in [92].

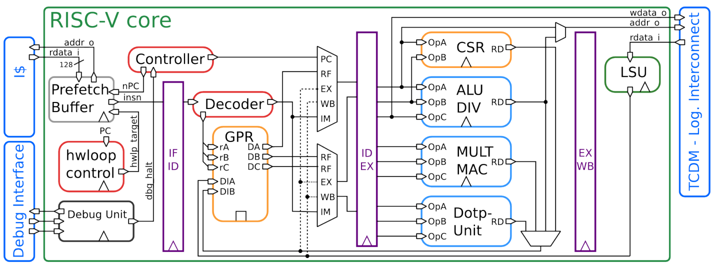


Figure 5.2: Schematic overview of the PULPINO RISCY core. Image from [6]

The PULPINO comes with instructions and a separate dataram, each being 32 KiB consisting of a word size of 32 bit resulting into 8192 words. The processor is a four-stage pipeline, and its memory map is shown in figure A.1. The PULPINO reassembles a microcontroller architecture. Within microcontrollers, the program is stored in flash memory. Flash memories are not subjected to ageing as with that the data RAM.

Benchmarks

As the PULPINO core has only 32 KiB amount of RAM available it can only run a limit set of benchmarks. The available benchmarks are shown in table 5.1, and a few key metrics are given. Next, an analysis is made of the chunk of the memory cells are subjected to long static stress. For all benchmarks, except the FFT-benchmark, all memory addresses have cells with static values for the complete benchmark.

Table 5.1: Overview of used benchmarks, duration, and memory usage.

Benchmark	Duration in clock cycles
PI 800 digits calculation	$15.4 \cdot 10^6$
Coremark 175 iterations	$61.8 \cdot 10^6$
Basicmath (MiBench)	$14.9 \cdot 10^6$
Bitcount (MiBench)	$24.3 \cdot 10^6$
FFT (MiBench)	$22.4 \cdot 10^6$
Picojpeg (MiBench)	$9.1 \cdot 10^6$
String search (MiBench 5)	$6.3 \cdot 10^6$

The used benchmarks come from various sources. The PI benchmark is written by Dik T. Winter in and is one of the most compact algorithms to calculate the digits of PI. This benchmark uses a spigot algorithm. Coremark is a benchmark made by EMBC to analyse the performance of microcontrollers and CPUs in embedded systems [93]. The other benchmarks are from the collection of MiBench created by the University of Michigan. These benchmarks try to simulate real-world workloads [94]. Using this setup in combination with the mitigation schemes from section 3.3. The following experiments have been performed.

- Software-based mitigation.
- Software-based mitigation with idle time.
- Hardware-based mitigation.

Within the mitigation scheme, it is possible the change the frequency the mitigation runs. For the software scheme, it is possible to change the mitigation size and optionally add extra idle time to the execution of the program. In the mitigation scheme, it can continuously run. For software mitigation, three different intervals will be analysed. Three different sizes and overhead up to 50% for the hardware mitigation scheme at three different intervals, will be analysed to give an impression of the results. At first we will introduce the baselines to be used to see how well mitigation schemes. To do this these questions need to be answered.

- **Static stress times:** For every mitigation scheme, the maximum static stress duration.
- **Overhead:** execution time overhead, power overhead and an indication of area overhead on the chip.
- **Duty factor:** How much does the duty factor change.

After the basic analyse has been conducted to answered the questions. The BTI impact will be analysed for the Sense Amplifier and the memory cells also the required baselines will be defined.

Software mitigation implementation

Implementing the software mitigation scheme requires it to be created as a coroutine, which can be invoked as an interrupt handler. The interrupt will be invoked periodicity and will stop the execution of the current process on the processor. he software coroutine must be written in assembly to maintain control over the executed code and ensure the value of the registers is correctly maintained. A small example of the coroutine is shown in listing 5.1.

Listing 5.1: RISC V assembly implementation of the mitigation scheme. Only the core part of the algorithm is shown. Initialisation, storing, and restoring register values are not shown.

```
mv t0, %0 ;Load the mitigating index
li t1, 0  ;t1 is and register to be used store the amount of itteration has been done.
mv t2, %1  ;Load the target offset value
j check
flip:
addi t1, t1, 1 ;Increment itteration
lw  t3, 0(t0) ;Load first memory value
not t3, t3 ;Invert the value
sw  t3, 0(t0) ;Store first memory value start relaxation phase
lw  t3, 0(t0) ;Load second memory value
not t3, t3 ;Invert the value
sw  t3, 0(t0) ;Store first memory value start relaxation phase
addi t0, t0, 8 ;Increment mitigation index
check:
ble t1, t2, flip ;Check if the required amount of mitigation itterations has passed
mv t0, %0 ;Restore the mitigating index back to the orginal starting location
li t1, 0  ;t1 is and register to be used store the amount of itteration has been done.
j check2
revert:
addi t1, t1, 1 ;Increment itteration
lw  t3, 0(t0) ;Load first memory value
not t3, t3 ;Invert the value
```

```
sw  t3, 0(t0) ;Store first memory value start relaxation phase
lw  t3, 0(t0) ;Load second memory value
not t3, t3 ;Invert the value
sw  t3, 0(t0) ;Store first memory value start relaxation phase
addi t0, t0, 8 ;Increment mitigation index
check2:
ble t1, t2, revert
```

5.2.2. BTI effect in the cell

For the memory cell, BTI effects will be analysed by the usage of the cell shown in figure 2.3. It consists of six transistors; four are subjected to static stress. The transistors M_1 and M_2 store the inverted value in the cell, while the transistors M_3 and M_4 store the normal value. Due to this balance in the cell bit values, zero and one will each create static stress and unbalancing the ageing in the cell. The transistors M_5 and M_6 are used to read or write values from and to the cell.

The BTI effect in the cell will degrade the SNM. The degrading of the cell is the strongest if the cell stores static values. For the extreme cases of zero or one-bit,value, stored for three years in the cell, the graph of the ageing of the transistors is shown in figure 5.3. This is also compared to the ideal case where the duty factor of the transistors M_1 up to M_4 is 0.5. When the memory cell is ageing perfectly balanced the SNM does not change, whereas when the duty factor reaches the extreme values, the BTI effect accelerates. Using the calculated voltage shift in the transistors the ageing effect in the memory cell can be calculated using SPICE in Cadence. Table 5.2 shows the different cases to illustrate the effect the duty factors have on the ageing of an SRAM-cell.

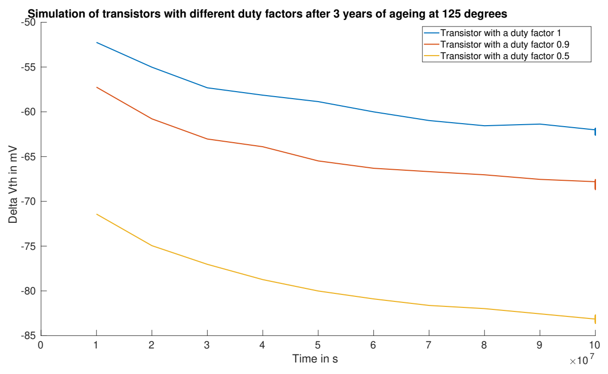


Figure 5.3: Comparing different duty factors on the transistors. If the duty factor is slightly reduced from 1.0 to 0.9, it has a relatively larger impact than when changing the duty factor from 0.9 to 0.5.

Table 5.2: Overview of different static stress durations depending on the running mitigation. Comparing the different mitigation scheme size to the interval at which they are run.

Duty factor	static noise margin
No ageing	354 mV
0.0	328 mV
0.1	350 mV
0.5	355 mV
0.9	350 mV
1.0	328 mV

5.2.3. BTI effect in the Sense Amplifier

Analysing the BTI in the SA is much more difficult as it is required to know the distribution of bits being read and the active time. The distribution of bits and active time will be calculated by running the different benchmarks.
To analyse the SA Monto Carlo simulation is used which has been provided by IMEC. A Python script has been

created to analyse the results from the simulation. The Monto Carlo simulation consists of 2000 iterations. For the SA it is only possible to define the start value of the offset voltage which is in ideal cases 0 mV.

5.3. Software Mitigation Results

At first the results of software mitigation are discussed. The implemented algorithm is the multiple cell mitigation, resulting in the lowest execution overhead compared to the other implementation, as discussed in section 4.4.

A python code has been designed to create the different routines variating the sizes and speed of the software mitigation, which will craft the assembly code to inject into the benchmarks.

5.3.1. Cell performance

The performance will be measured and shown for each benchmark looking at the memory cells and sense amplifier. Figures 5.4 to 5.6 shows the SNM degradation of the memory cells, running different mitigation scheme configuration, with and without mitigation. The observations are as follow:

- The SNM degradation is not application dependent. The requirement is that the benchmark simulation is long enough to complete full mitigation of the memory.
- The degradation of the SNM is reduced significantly, when adding a mitigation scheme, allowing the mitigation to have a higher overhead with a lower return on investment.
- The mitigation scheme reduces the degradation of the SNM significantly (up to 2.5x).

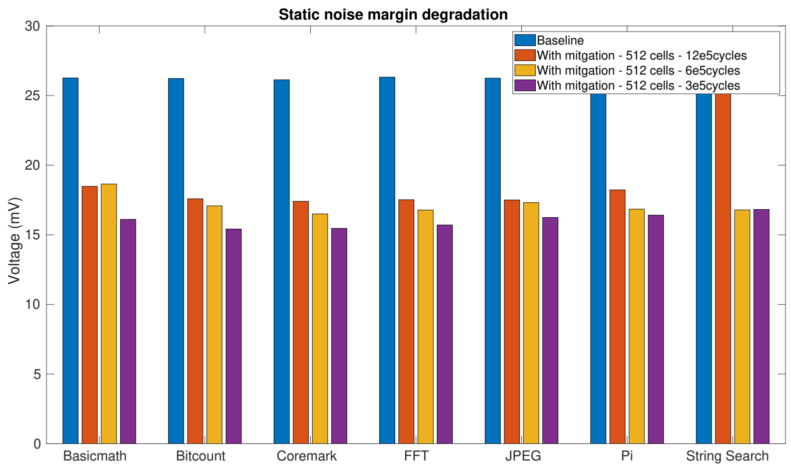


Figure 5.4: The degradation of the SNM running different benchmarks with and without mitigation scheme. The software scheme will relax **512** memory cells at different intervals. The experiment is run at 125 °C for three years.

Table 5.3 shows that the cell performance is application-independent, as derived from the graph to compare the V_{th} shift. When looking at the degradation of the SNM at a block size of 2048 and an interval at 1.2×10^6 cycles and to the 512 block size and an interval of 3×10^5 cycles. The latter option has a bigger degradation while the overhead of both mitigation schemes is the same. The memory cell should have a slightly longer relaxation and static stress than a shorter relaxation and duration when stress is present.

Table 5.3: Overview SNM of cells with different mitigation scheme settings running. The longer and the faster the mitigation is run, the better the mitigation scheme performs. It can also be seen that reduction of the length of static stress has a smaller impact on the SNM than running the mitigation scheme longer.

Block size	1.2×10^6	6.0×10^5	3.0×10^5
512	17.4 mV	16.5 mV	15.4 mV
1024	15.1 mV	14.7 mV	13.2 mV
2048	13.3 mV	12.0 mV	10.9 mV

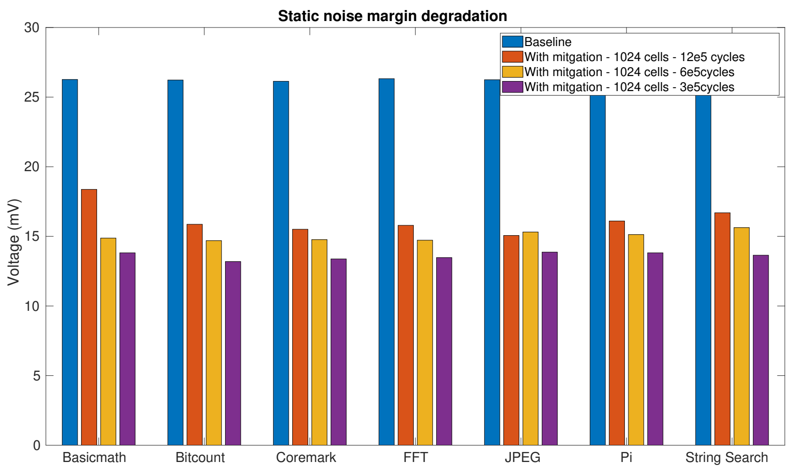


Figure 5.5: The degradation of the SNM running different benchmarks with and without mitigation scheme. The software scheme will relax **1024** memory cells at different intervals. The experiment is run at 125 °C for three years.

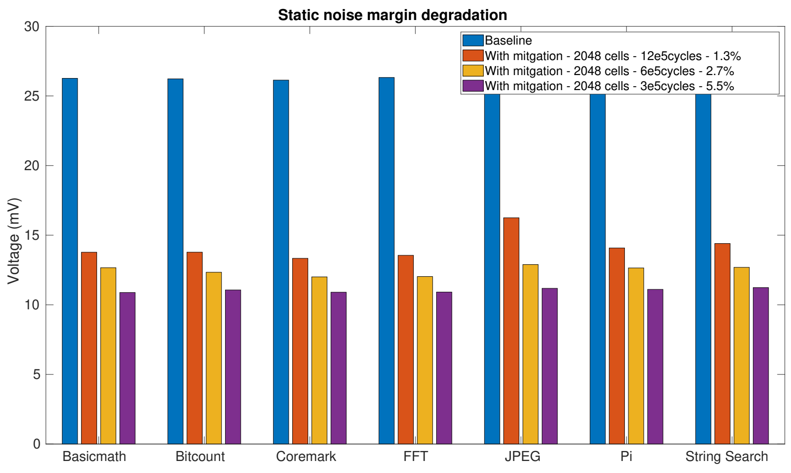


Figure 5.6: The degradation of the SNM running different benchmarks with and without mitigation scheme. The software scheme will relax **2048** memory cells at different intervals. The experiment is run at 125 °C for three years.

Figure 5.7 shows the SNM degradation of the memory cells running the software mitigation differently, with added idle time. The SNM is shown with and without mitigation. The following can be observed from the figure:

- The SNM degradation is not application dependent. The requirement is that the benchmark simulation is long enough to complete full mitigation of the memory.
- The degradation of the SNM is reduced significantly, when adding a mitigation scheme, allowing the mitigation to have a higher overhead results in a lower return on investment.
- The mitigation scheme reduces the degradation of the SNM (up to 2.5x)
- Adding idle time into the simulation reduces the ageing of the memory cells.

After analysing the mitigation scheme without injecting idle time, the next part will analyse the SNM degradation if idle time is used to mitigate static stress in memories. Adding idle time to the simulation of the mitigation scheme reduces of the degradation on the SNM as shown in table 5.4. The biggest improvement is when adding a slight overhead into the simulation. This is expected, and the more overhead that is added, the better the result.

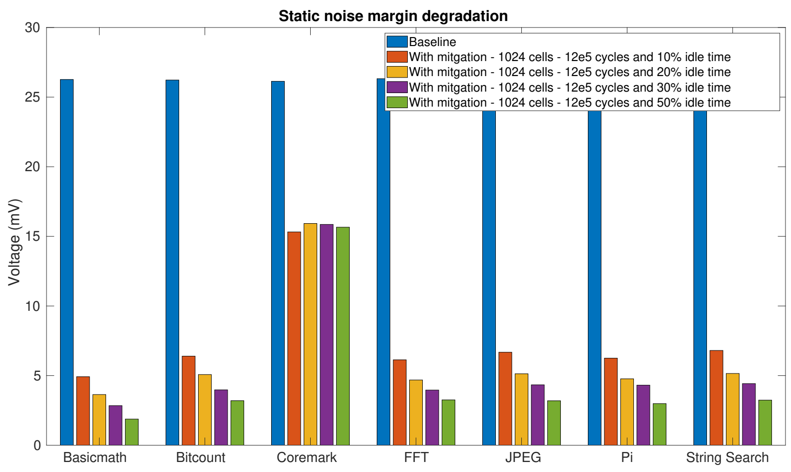


Figure 5.7: The degradation of the SNM running different benchmarks with and without mitigation scheme. The software scheme will relax **1024** memory cells at a fixed interval and add a varying amount of idle time. The experiment is run at 125 °C for three years.

Table 5.4: Overview SNM degradation of the cells after adding idle time to the simulation of the benchmarks.

Idle time overhead	SNM offset voltage
10%	4.9 mV
20%	3.6 mV
30%	2.8 mV
50%	1.9 mV

5.3.2. Sense amplifier

For the sense amplifier, it is not possible to group the results as for the cells due to different bit distribution, and sense amplifier activation between every benchmark.

In figures 5.8 to 5.10 the offset voltage sense amplifier is plotted. The different software mitigations configuration has been run. The offset voltage is shown with and without mitigation. The following can be observed from the figure:

- The offset voltage of the sense amplifier is application-dependent. Each benchmark has an optimal mitigation scheme to reduce the offset voltage.
- The offset voltage can increase if the amount of memory operation increases to much and cause more stresses in the sense amplifier. Across the run benchmarks, the overall offset voltage improves over the worst benchmark.
- The mitigation scheme can reduce the offset voltage with nearly 50%.

Figures 5.8 to 5.10 show the results of the different software mitigation schemes without the simulation of idle time. The benchmark, JPEG, can even have a higher degradation as compared to running no mitigation scheme. The worst-case performance significantly improves over the different benchmarks. While the benchmark, Coremark, defines the result.

Figure 5.11 shows the results from adding idle time to the mitigation scheme. There is a balance in the mitigation scheme and the usage of the sense amplifier:

- The offset voltage of the sense amplifier is application dependent. Each benchmark has an optimal mitigation scheme to reduce the offset voltage.
- The offset voltage can increase if the amount of memory operation increases, which causes more stresses in the sense amplifier. In all the run benchmarks, the overall offset voltage always improves over the worst possible benchmark.
- Using idle time to mitigate the offset voltage increase, reduces the performances.

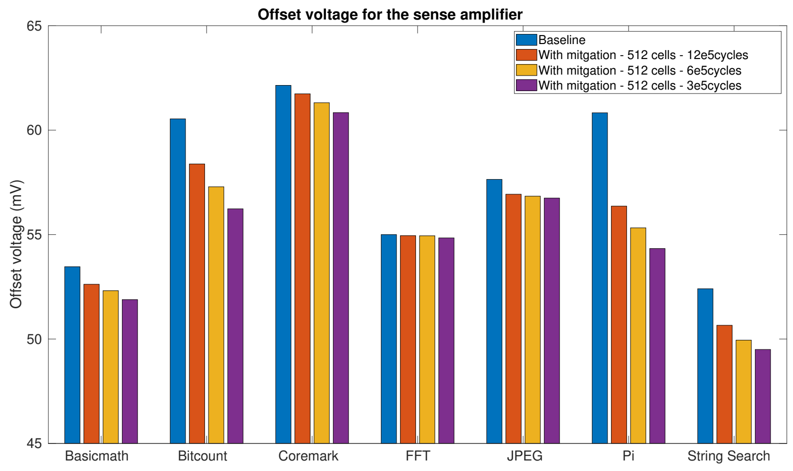


Figure 5.8: The degradation of the sense amplifier running different benchmarks with and without mitigation scheme. The software scheme will relax **512** cells at different intervals. The faster the mitigation scheme runs, the more balanced the sense amplifier will be. It will also be used more. The more usage of the sense amplifier can result in extra degradation of the offset voltage spec. The experiment is run at 125 °C for three years.

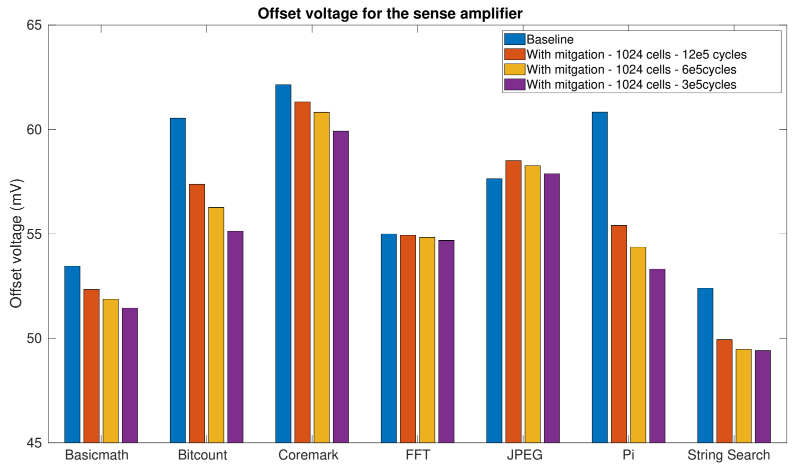


Figure 5.9: The degradation of the sense amplifier running different benchmarks with and without mitigation scheme. The software scheme will relax **1024** cells at different intervals. The faster the mitigation scheme runs, the more balanced the sense amplifier will be. It will also be used more. The more usage of the sense amplifier can result into extra degradation of the offset voltage spec. The experiment is run at 125 °C for three years.

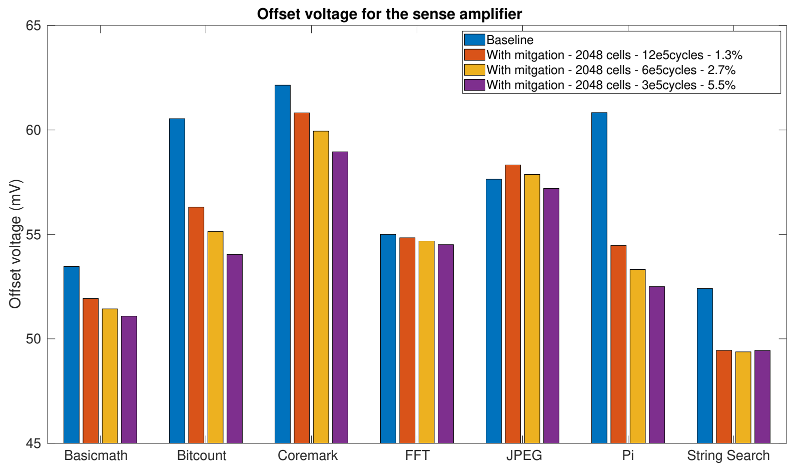


Figure 5.10: The degradation of the sense amplifier running different benchmarks with and without mitigation scheme. The software scheme will relax **2048** cells at different intervals. The faster the mitigation scheme runs, the more balanced the sense amplifier will be. It will also be used more. The more usage of the sense amplifier can result into extra degradation of the offset voltage spec. The experiment is run at 125 °C for three years.

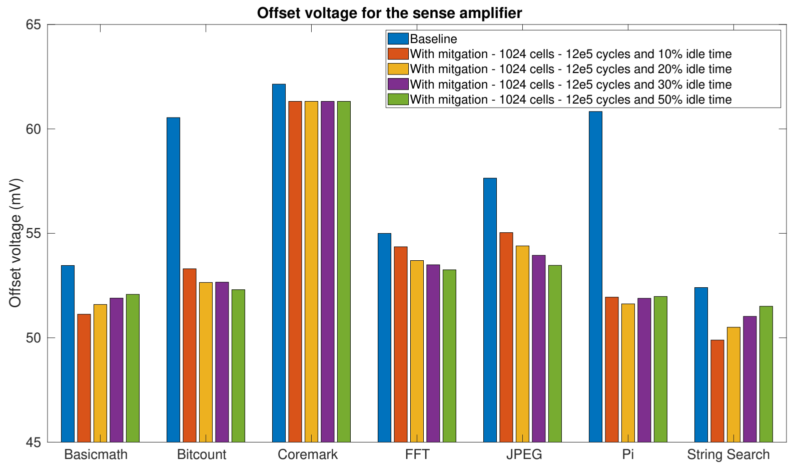


Figure 5.11: The degradation of the sense amplifier running different benchmarks with and without mitigation scheme. The software scheme will relax **1024** cells at an interval of every 1.2×10^6 cycles and incorporate idle time for extra mitigation. The more usage of the sense amplifier can result into extra degradation of the offset voltage spec. The experiment is run at 125 °C for a years.

5.4. Hardware Mitigation Results

The software results have been discussed. Next step is to look at the hardware implementation. The chosen algorithm is the cyclic flipping, as shown in section 4.5. This algorithm will have the best mitigation performance. The implementation is done in Verilog and will allow configuring at which interval the hardware mitigation will run.

5.4.1. Cell performance

The hardware mitigation scheme will make sure the average duty factor of every cell is balanced at the value of 0.5. The average duty factor might stay higher if the hardware mitigation scheme does not run fast enough to complete a cyclic pass through the memory within the duration of the benchmark. In figure 5.12 the SNM is shown for the different mitigation scheme settings. Some benchmarks have outliers because they are too short to run a balanced mitigation scheme using the hardware mitigation scheme settings. From this figure, the following observation have been made:

- The SNM degradation is slightly application dependent. The requirement is that the benchmark simulation is long enough to complete full mitigation of the memory. The other requirement is that memory values should not be accidentally moved to longer static stress due to flipping of the value.
- The degradation of the SNM is reduced significantly, when adding a mitigation scheme allowing the mitigation to have a higher overhead with a lower return on investment.
- The mitigation scheme reduces the degradation of the SNM (up to 45x).

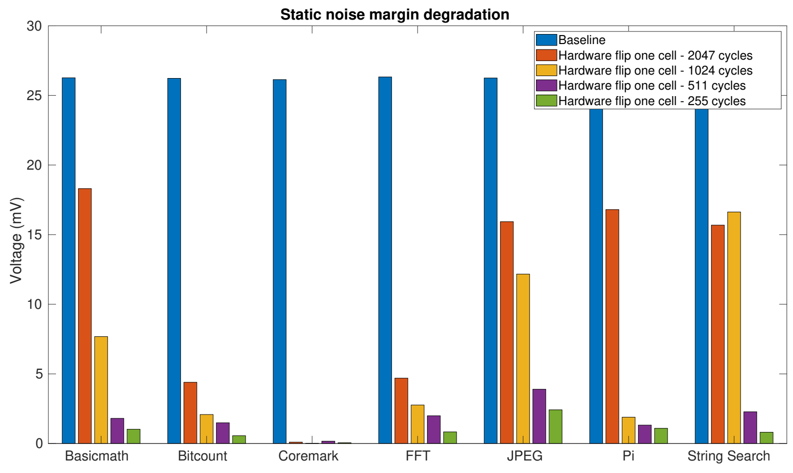


Figure 5.12: The degradation of the memory cell running different benchmarks with and without mitigation scheme. Hardware mitigation allows for better balancing if the benchmark is long enough. The ageing effect due to BTI nearly disappears. The experiment is run at 125 °C for three years.

5.4.2. Sense amplifier performance

There is a significant difference depending on the benchmark: how many read, write operation, bit distribution. This difference is less with hardware mitigation as shown in figure 5.13:

- The offset voltage of the sense amplifier is application dependent. Each benchmark has an optimal mitigation scheme to reduce the offset voltage.
- The offset voltage can increase if the amount of memory operation increases slightly in all the run benchmarks. The overall offset voltage always improves over the worst possible benchmark.
- The mitigation scheme can reduce the offset voltage with nearly 67%.

The hardware mitigation scheme balances the sense amplifier much better compared to the software mitigation scheme where the result still is very much dependant on the running benchmark.

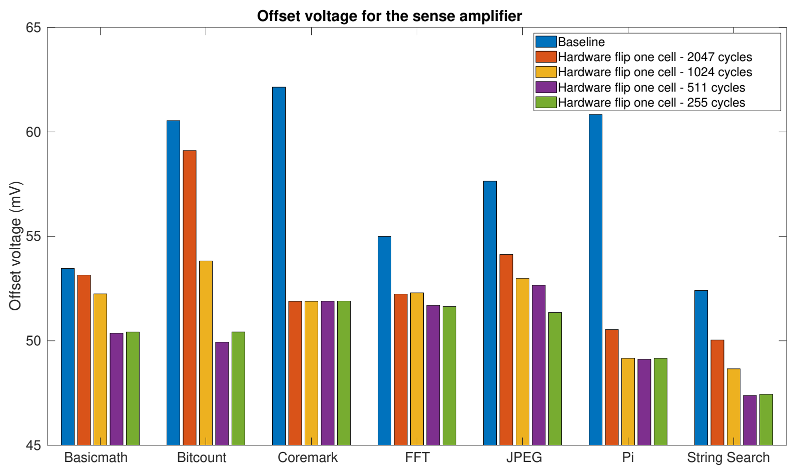


Figure 5.13: The degradation of the sense amplifier running different benchmarks with and without mitigation scheme. Hardware mitigation allows for better balancing of the sense amplifier compared to the software scheme but even still if the sense amplifier gets used to much extra degradation can be seen. The experiment is run at 125 °C for a period of three years.

5.5. Hardware versus Software

The last step is to establish which of the two mitigation schemes performs best. For this, the different trade-offs will be listed as well as the advantages of each option.

5.5.1. Comparison

Comparing the two different methods, hardware versus software it is interesting to look at what the overheads and achievements of both schemes are.

Software mitigation performance overhead

Table 5.5 shows the results. A total of 130 cycles overhead is spent in the function call and handling the interrupt, while the flipping of a single memory cell takes four clock cycles, which is one cycle more than expected at first. The four cycles relate to PULPINO's pipeline, which is four stages long.

Table 5.5: Execution time of software routine showing the function call time.

Block size	Execution time (cycles)	Cell relax time (cycles)	Function call time	Interrupt overhead
4	164	16	132	90%
8	193	32	129	83%
16	257	64	129	75%

As discussed, the performance will be assessed for the memory cells and the sense amplifier. When running the mitigation scheme, without idle time, the duty factors barely change. Only the duration of static stress is significantly shorter. Table 5.6 shows the longest static stress durations are for each mitigation setting, which has been run. The mitigation scheme introduces performance overhead during the run-time of a benchmark: the measure values shown in table 5.7.

Table 5.6: Overview of different static stress durations depending on running mitigation. Comparing the different mitigation scheme size to the interval at which they are run.

Block size	1.2×10^6	6.0×10^5	3.0×10^5
512	1.92×10^7	9.60×10^6	4.80×10^6
1024	9.60×10^6	4.80×10^6	2.40×10^6
2048	4.80×10^6	2.40×10^6	1.20×10^6

Table 5.7: Overview of different static stress durations depending on running mitigation. Comparing the different mitigation scheme size to the interval at which they are run.

Block size	1.2×10^6	6.0×10^5	3.0×10^5
512	0.4%	0.7%	1.4%
1024	0.7%	1.4%	2.8%
2048	1.4%	2.8%	5.5%

When adding the idle time, which is a fraction of the execution time of the benchmark, will result in a significant change of the duty factor in table 5.8.

Table 5.8: Overview of different of the duty factor which can be found in the cells depending on the different amounts of available idle time.

Overhead	Low duty factor	High duty factor
10%	0.05	0.95
20%	0.08	0.92
30%	0.11	0.89
40%	0.14	0.86
50%	0.17	0.83

After analysing the mitigation scheme impact, the next step is to inspect the effect on the BTI-ageing.

Hardware mitigation performance overhead
The hardware implementation is in Verilog and will only use the memory when the IC does not require access to the memory. The hardware implementation will, therefore, not add any overhead in the form of execution time. The overhead will consist of extra hardware, which will be required, and extra usage of the memory, thus, more power usage and a larger chip.

Looking at how the static stress duration is handled within the hardware mitigation, this study does the same analysis within the software mitigation scheme. These results are shown in table 5.9.

Table 5.9: Overview of different static stress durations depending on running mitigation. Comparing the different intervals at which the hardware mitigation can run.

Block size	Static stress duration in cycles
255	2.09×10^6
511	4.19×10^6
1023	8.38×10^6
2047	1.68×10^7

5.5.2. Memory overhead

Table 5.10 shows the memory-overhead usage for the different mitigation scheme, which has been analysed. The hardware mitigation scheme requires a lot less memory usage compared to the software mitigation scheme.

5.5.3. Conclusion

Software mitigation scheme introduces execution overhead and memory usages, whereas the hardware mitigation scheme introduces memory usage overhead and requires extra hardware. Table 5.11 highlights key points of the effect of hardware and software mitigation scheme, which summarise the implementation. The hardware mitigation scheme brings greater improvement compared to the software mitigation scheme, as it changes the average duty factor.

Table 5.10: Comparing the memory overhead of the different scheme which has been created and run.

Benchmark	Memory overhead usage compared to the baseline
Hardware mitigation - 255 cycles	101.6%
Hardware mitigation - 511 cycles	101%
Hardware mitigation - 1024 cycles	100.5%
Hardware mitigation - 2047 cycles	100.2%
Software mitigation - 1024 cells - 12e5 cycles	101.7%
Software mitigation - 1024 cells - 3e5cycles	107%
Software mitigation - 1024 cells - 6e5cycles	103.5%
Software mitigation - 512 cells - 12e5cycles	100.9%
Software mitigation - 512 cells - 6e5cycles	101.8%
Software mitigation - 512 cells - 3e5cycles	103.5%
Software mitigation - 2048 cells - 12e5cycles	103.4%
Software mitigation - 2048 cells - 6e5cycles	106.9%
Software mitigation - 2048 cells - 3e5cycles	113.8%

Table 5.11: An overview of the different impacts of the hardware and software mitigation scheme on the memory and IC.

Item	Hardware	Software
Execution overhead	No impact on execution time	Increases execution time
Memory usages	Slight increase	Increase with up to 15%
Idle time	-	Can utilise idle time
Impact on duty factor	Average moves to 0.5	Duty factor does not change
Hardware overhead	Requires extra transistors	None
Implementation	During chip design	Can be implemented afterwards
SNM improvement		
Offset voltage improvement		

6 Conclusion

6.1. Conclusion

This thesis has shown that mitigating BTI induced ageing in SRAM is possible by adding software coroutines. Both the memory cells and the Sense Amplifier saw a significant reduction in BTI-induced ageing at very low overhead. The memory cell ageing was reduced with up to 40% at a run-time overhead of only 1.4%. In addition, the degradation of the SA was reduced by up to 50%. This reduced degradation leads to a more reliable memory and an increased lifetime. Moreover, when adding idle time to the simulated workload, software mitigation scheme could make a significant improvement.

In addition, this thesis showed that the implemented hardware mitigation scheme is more efficient compared to the software mitigation scheme. This is due to the fact that the hardware mitigation scheme can better balance the overall duty cycles of the memory cell to 50%. As a result, the hardware mitigation scheme also yields a much higher reduction of BTI-induced ageing at the cost of adding extra hardware to the memory. Nevertheless, software mitigation schemes can be easily added to the compilers. Hence, the software-based approach can even be added to existing hardware *without requiring modifications*. This addition will extend the lifetime of the chip at the costs of extending the required run-time of a program. Additionally, software mitigation could also make use of run-time information, while this would be much harder to implement in hardware. Hence, both software mitigation and hardware mitigation have their merits. For example, the software-based scheme is interesting for systems that require a low area, while the hardware-based scheme is interesting when this restriction is not present. The usage of BIST for mitigation of BTI-induced ageing within the memories is not feasible, as the best mitigation scheme will flip the cells values compared to the value they are storing.

6.2. Future work

The work presented two different mitigation schemes against the effect of BTI-induced ageing. The research within the thesis can be extended by looking at:

- **Circuit model**
  The suggested mitigation scheme can be applied to a newer, more modern SRAM design. The thesis used a 14 nm. It would be interesting to find out how much of an improvement these mitigation schemes can provide on a smaller technology node. This is due to the fact that smaller technology nodes are more affected by ageing.

- **Sensing**
  Adding sensors within the memory could allow for smarter mitigation schemes. The state of the art showed that the results of this are not promising. An in-depth comparison could be made between mitigation schemes with and without sensors. Sensors could be combined with AI to be able to make smarter sensors.

- **Different platforms**
  With this thesis, the work focused on the PULPINO, which has a RISC-V architecture. In future re-

search, different platforms could be analysed with different architects and allowing more advanced benchmarks.

- **Improve ageing model**
  This research could also be extended with a more accurate ageing model. For example, add HCI, TDDB, and RTN to the ageing model. Furthermore, if a complete memory design is available EM degradation could be included.

- **Extending the analysis**
  This research could be extended in depth of the analysed components. First of all, the effect on more memory components can be investigated, such as the address decoder and timing circuit. It is expected that both the proposed software- and hardware-based mitigation schemes will reduce the ageing of the address decoder. This is due to the fact that they both ensure all memory addresses are traversed and, thus, static stresses will be reduced in the address decoder. This is already briefly demonstrated in [95]. Second, future work could also analyse more metrics, such as the write and hold SNM of the memory cell. This work analysed the read SNM, as it is the most critical metric. However, for a reliable design *any* metric should stay within its specifications even under ageing.

A

PULPino

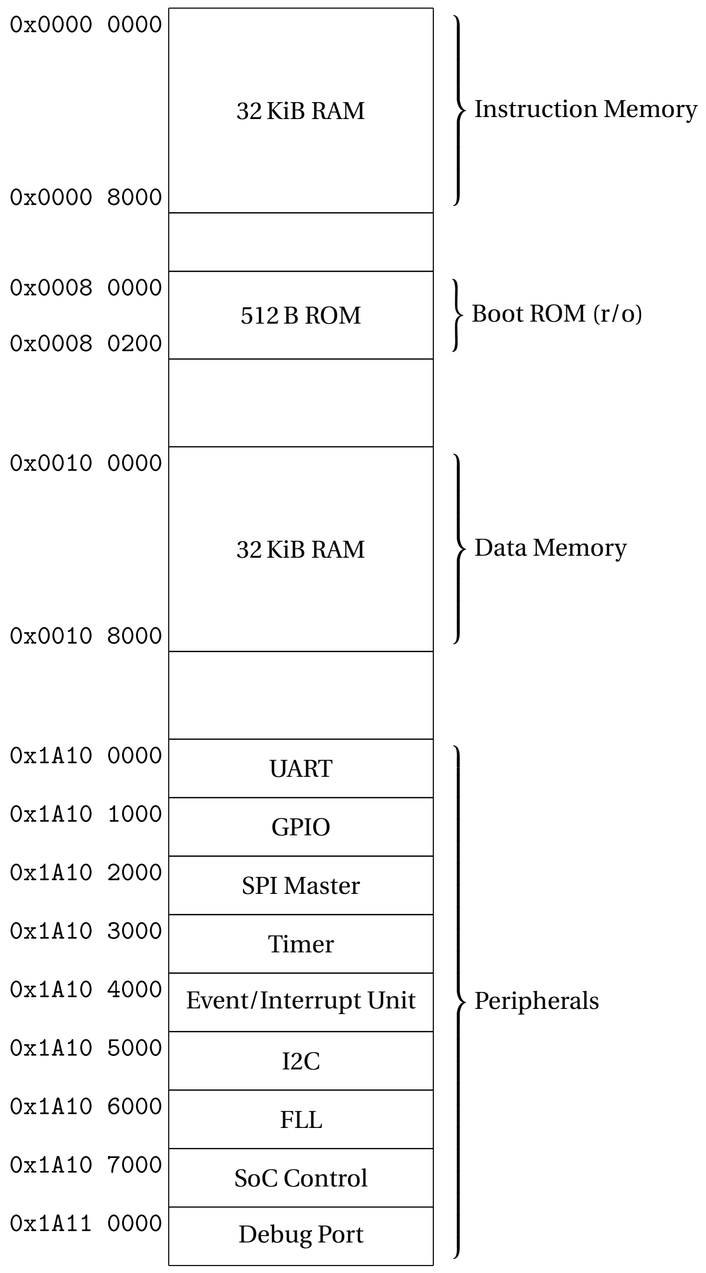


Figure A.1: The addressing of the memory with in the PULPINO. This can be used to write a assembly routine which can mitigate the memory of the core. Image from [6]

Bibliography

[1] A. Nowak, “Opportunities and choice in a new vector era,” *Journal of Physics: Conference Series*, vol. 523, no. 1, 2014.

[2] 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,” *Proceedings of the IEEE VLSI Test Symposium*, vol. 2015-January, pp. 1–6, 2015.

[3] 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, June 2019.

[4] A. Campos-Cruz, G. Espinosa-Flores-Verdad, A. Torres-Jacome, and E. Tlelo-Cuautle, “On the Prediction of the Threshold Voltage Degradation in CMOS Technology Due to Bias-Temperature Instability,” *Electronics*, vol. 7, no. 12, p. 427, dec 2018. [Online]. Available: http://www.mdpi.com/2079-9292/7/12/427

[5] K. N. Quader, C. Li, R. Tu, E. Rosenbaum, P. Ko, and C. Hu, “A new approach for simulation of circuit degradation due to hot-electron damage in NMOSFETs,” *Technical Digest - International Electron Devices Meeting, IEDM*, vol. 1991-Janua, no. 7, pp. 337–340, 1991.

[6] A. Traber and M. Gautschi, *PULPino: Datasheet Imperio: The first PULPino ASIC*, 2017.

[7] M. T. Bohr and I. A. Young, “CMOS Scaling Trends and beyond,” *IEEE Micro*, vol. 37, no. 6, pp. 20–29, 2017.

[8] 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. [Online]. Available: http://ieeexplore.ieee.org/document/1566551/

[9] R. Aitken, E. H. Cannon, M. Pant, and M. B. Tahoori, “Resiliency challenges in sub-10nm technologies,” *Proceedings of the IEEE VLSI Test Symposium*, vol. 2015-January, pp. 1–4, 2015.

[10] E. Suhir, “Aging-related failure rate obtained from bathtub curve data,” in *2015 IEEE Aerospace Conference*, 2015, pp. 1–8.

[11] H. F. Dadgour and K. Banerjee, “A built-in aging detection and compensation technique for improving reliability of nanoscale cmos designs,” 2010, pp. 822–825.

[12] K. Jeong, A. B. Kahng, and K. Samadi, “Impact of guardband reduction on design outcomes: A quantitative approach,” pp. 552–565, 2009.

[13] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, and D. Sylvester, “Underdesigned and opportunistic computing in presence of hardware variability,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 8–23, 2013.

[14] K. Wu and D. Marculescu, “Aging-aware timing analysis and optimization considering path sensitization,” in *2011 Design, Automation Test in Europe*, March 2011, pp. 1–6.

[15] M. Ebrahimi, F. Oboril, S. Kiamehr, and M. B. Tahoori, “Aging-aware logic synthesis,” in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD ’13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 61–68. [Online]. Available: http://dl.acm.org/citation.cfm?id=2561828.2561840

[16] S. Kiamehr, F. Firouzi, M. Ebrahimi, and M. B. Tahoori, “Aging-aware standard cell library design,” *Proceedings - Design, Automation and Test in Europe, DATE*, vol. 3, pp. 1–4, 2014.

[17] N. Khoshavi, R. A. Ashraf, R. F. DeMara, S. Kiamehr, F. Oboril, and M. B. Tahoori, “Contemporary cmos aging mitigation techniques: Survey, taxonomy, and methods,” *Integration*, vol. 59, pp. 10 – 22, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167926017301876

[18] M. Sipser, *Introduction to the Theory of Computation*, 3rd ed. Boston, MA: Course Technology, 2013, pp. 216, 217.

[19] E. Mintarno, J. Skaf, R. Zheng, J. B. Velamala, Y. Cao, S. Boyd, R. W. Dutton, and S. Mitra, “Self-tuning for maximized lifetime energy-efficiency in the presence of circuit aging,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 760–773, May 2011.

[20] O. Khan and S. Kundu, “A self-adaptive system architecture to address transistor aging,” *Proceedings -Design, Automation and Test in Europe, DATE*, pp. 81–86, 2009.

[21] L. Zhang and R. P. Dick, “Scheduled voltage scaling for increasing lifetime in the presence of NBTI,” *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pp. 492–497, 2009.

[22] T. B. Chan, J. Sartori, P. Gupta, and R. Kumar, “On the efficacy of NBTI mitigation techniques,” *Proceedings -Design, Automation and Test in Europe, DATE*, pp. 932–937, 2011.

[23] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits (2nd Edition)*. Pearson, 2003. [Online]. Available: https://www.amazon.com/Digital-Integrated-Circuits-2nd-Rabaey/dp/0130909963?SubscriptionId=AKIAIOBINVZYXZQZ2U3A{&}tag=chimbori05-20{&}linkCode=xm2{&}camp=2025{&}creative=165953{&}creativeASIN=0130909963

[24] F. Oboril and M. B. Tahoori, “Reducing wearout in embedded processors using proactive fine-grain dynamic runtime adaptation,” *Proceedings - 2012 17th IEEE European Test Symposium, ETS 2012*, pp. 1–6, 2012.

[25] D. Rossi, V. Tenentes, S. Yang, S. Khursheed, and B. M. Al-Hashimi, “Reliable Power Gating with NBTI Aging Benefits,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 8, pp. 2735–2744, 2016.

[26] X. Guo and M. R. Stan, “Work hard, sleep well-Avoid irreversible IC wearout with proactive rejuvenation,” *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, vol. 25-28-Janu, pp. 649–654, 2016.

[27] S. Gupta and S. S. Sapatnekar, “GNOMO: Greater-than-NOMinal V dd operation for BTI mitigation,” *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pp. 271–276, 2012.

[28] F. Firouzi, S. Kiamehr, and M. B. Tahoori, “Nbti mitigation by optimized nop assignment and insertion,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE ’12. San Jose, CA, USA: EDA Consortium, 2012, pp. 218–223. [Online]. Available: http://dl.acm.org/citation.cfm?id=2492708.2492763

[29] A. Gebregiorgis, M. Ebrahimi, S. Kiamehr, F. Oboril, S. Hamdioui, and M. B. Tahoori, “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.

[30] E. Karl, P. Singh, D. Blaauw, and D. Sylvester, “Compact in-situ sensors for monitoring negative-bias-temperature-instability effect and oxide degradation,” in *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, Feb 2008, pp. 410–623.

[31] P. Lu and K. A. Jenkins, “A built-in bti monitor for long-term data collection in ibm microprocessors,” in *2013 IEEE International Reliability Physics Symposium (IRPS)*, April 2013, pp. 4A.1.1–4A.1.6.

[32] S. Wang, J. Chen, and M. Tehranipoor, “Representative critical reliability paths for low-cost and accurate on-chip aging evaluation,” in *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2012, pp. 736–741.

[33] H. F. Dadgour and K. Banerjee, “A built-in aging detection and compensation technique for improving reliability of nanoscale cmos designs,” in *2010 IEEE International Reliability Physics Symposium*, May 2010, pp. 822–825.

[34] Z. Qi, J. Wang, A. Cabe, S. Wooters, T. Blalock, B. Calhoun, and M. Stan, “Sram-based nbti/pbti sensor system design,” in *Design Automation Conference*, June 2010, pp. 849–852.

[35] A. Koneru, A. Vijayan, K. Chakrabarty, and M. B. Tahoori, “Fine-grained aging prediction based on the monitoring of run-time stress using dft infrastructure,” in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015, pp. 51–58.

[36] F. Firouzi, F. Ye, K. Chakrabarty, and M. B. Tahoori, “Aging- and variation-aware delay monitoring using representative critical path selection,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 3, pp. 39:1–39:23, Jun. 2015. [Online]. Available: http://doi.acm.org/10.1145/2746237

[37] F. Firouzi, F. Ye, A. Vijayan, A. Koneru, K. Chakrabarty, and M. B. Tahoori, “Re-using bist for circuit aging monitoring,” in *2015 20th IEEE European Test Symposium (ETS)*, May 2015, pp. 1–2.

[38] D. Kraak, I. Agbo, M. Taouil, S. Hamdioui, P. Weckx, S. Cosemans, and F. Catthoor, “Degradation analysis of high performance 14nm finfet SRAM,” *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, DATE 2018*, vol. 2018-January, pp. 201–206, 2018.

[39] W. L. Wang and K. J. Lee, “An efficient deterministic test pattern generator for scan-based BIST environment,” *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 18, no. 1, pp. 43–53, 2002.

[40] C. Carvalho, “The gap between processor and memory speeds,” *Icca*, pp. 27–34, 2002. [Online]. Available: http://gec.di.uminho.pt/discip/minf/ac0102/1000gap{_}proc-mem{_}speed.pdf

[41] F. Wang and M. Hamdi, “Matching the speed gap between SRAM and DRAM,” *2008 International Conference on High Performance Switching and Routing, HPSR 2008*, pp. 104–109, 2008.

[42] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*, 5th ed. Morgan Kaufmann Publishers Inc., 2013.

[43] D. Rodopoulos, G. Psychou, M. M. Sabry, F. Catthoor, A. Papanikolaou, D. Soudris, T. G. Noll, and D. Atienza, “Classification framework for analysis and modeling of physically induced reliability violations,” *ACM Computing Surveys*, vol. 47, no. 3, pp. 1–33, 2015.

[44] Kunhyuk Kang, Sang Phill Park, Kaushik Roy, and M. A. Alam, “Estimation of statistical variation in temporal nbti degradation and its impact on lifetime circuit performance,” in *2007 IEEE/ACM International Conference on Computer-Aided Design*, Nov 2007, pp. 730–734.

[45] I. Agbo, M. Taouil, S. Hamdioui, P. Weckx, S. Cosemans, P. Raghavan, F. Catthoor, and W. Dehaene, “Quantification of sense amplifier offset voltage degradation due to zero-and run-time variability,” *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, vol. 2016-September, pp. 725–730, 2016.

[46] I. O. Agbo, “Reliability Modeling and Mitigation for Embedded Memories,” Ph.D. dissertation, Delft University of Technology, 2018. [Online]. Available: https://doi.org/10.4233/uuid:ce7b3290-9e0f-406b-93ee-7bfb7c9a8430

[47] N. Kimizuka, T. Yamamoto, T. Mogami, K. Yamaguchi, K. Imai, and T. Horiuchi, “The impact of bias temperature instability for direct-tunneling ultra-thin gate oxide on mosfet scaling,” in *1999 Symposium on VLSI Technology. Digest of Technical Papers (IEEE Cat. No.99CH36325)*, June 1999, pp. 73–74.

[48] T. Liu, C. C. Chen, J. Wu, and L. Milor, “SRAM stability analysis for different cache configurations due to Bias Temperature Instability and Hot Carrier Injection,” *Proceedings of the 34th IEEE International Conference on Computer Design, ICCD 2016*, pp. 225–232, 2016.

[49] D. Lorenz, M. Barke, and U. Schlichtmann, “Aging analysis at gate and macro cell level,” in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD ’10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 77–84. [Online]. Available: http://dl.acm.org/citation.cfm?id=2133429.2133444

[50] F. Firouzi, S. Kiamehr, M. Tahoori, and S. Nassif, “Incorporating the impacts of workload-dependent runtime variations into timing analysis,” in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2013, pp. 1022–1025.

[51] S. Karapetyan and U. Schlichtmann, “Integrating aging aware timing analysis into a commercial sta tool,” in *VLSI Design, Automation and Test(VLSI-DAT)*, April 2015, pp. 1–4.

[52] H. Kükner, S. Khan, P. Weckx, P. Raghavan, S. Hamdioui, B. Kaczer, F. Catthoor, L. Van der Perre, R. Lauwereins, and G. Groeseneken, “Comparison of reaction-diffusion and atomistic trap-based bti models for logic gates,” *IEEE Transactions on Device and Materials Reliability*, vol. 14, no. 1, pp. 182–193, March 2014.

[53] S. Verweij, *Fundamentals of Bias Temperature Instability in MOS Transistors*, ser. Springer Series in Advanced Microelectronics, S. Mahapatra, Ed. New Delhi: Springer India, nov 2016, vol. 52, no. 6. [Online]. Available: http://doi.wiley.com/10.1002/sres.2247http://link.springer.com/10.1007/978-81-322-2508-9

[54] D. A.Neamen, *Semiconductor Physics and devices*, 4th ed. Cambridge: Mc Graw Hill, 2012. [Online]. Available: http://www.optima.ufam.edu.br/SemPhys/Downloads/Neamen.pdf

[55] G. Gielen, P. De Wit, E. Maricau, J. Loeckx, J. Martín-Martínez, B. Kaczer, G. Groeseneken, R. Rodríguez, and M. Nafría, “Emerging yield and reliability challenges in nanometer CMOS technologies,” *Proceedings -Design, Automation and Test in Europe, DATE*, pp. 1322–1327, 2008.

[56] T. Grasser, B. Kaczer, W. Goes, H. Reisinger, T. Aichinger, P. Hehenberger, P. J. Wagner, F. Schanovsky, J. Franco, M. Toledano Luque, and M. Nelhiebel, "The paradigm shift in understanding the bias temperature instability: From reaction-diffusion to switching oxide traps," *IEEE Transactions on Electron Devices*, vol. 58, no. 11, pp. 3652–3666, 2011.

[57] B. Kaczer, T. Grasser, P. J. Roussel, J. Franco, R. Degraeve, L. A. Ragnarsson, E. Simoen, G. Groeseneken, and H. Reisinger, "Origin of NBTI variability in deeply scaled pFETs," *IEEE International Reliability Physics Symposium Proceedings*, pp. 26–32, 2010.

[58] S. S. Chung, "The understanding of breakdown path in both high-k metal-gate CMOS and resistance RAM by the RTN measurement," *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology, ICSICT 2016 - Proceedings*, pp. 428–431, 2017.

[59] C. M. Chang, S. S. Chung, Y. S. Hsieh, L. W. Cheng, C. T. Tsai, G. H. Ma, S. C. Chien, and S. W. Sun, "The observation of trapping and detrapping effects in high-k gate dielectric mOSFETs by a new gate current random telegraph noise (IG-RTN) approach," *Technical Digest - International Electron Devices Meeting, IEDM*, no. 2, pp. 8–11, 2008.

[60] J. W. McPherson, "Time dependent dielectric breakdown physics - Models revisited," *Microelectronics Reliability*, vol. 52, no. 9-10, pp. 1753–1760, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.microrel.2012.06.007

[61] J. Fang and S. S. Sapatnekar, "Incorporating hot-carrier injection effects into timing analysis for large circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 12, pp. 2738–2751, 2014.

[62] J. R. Black, "Electromigration—A Brief Survey and Some Recent Results," *IEEE Transactions on Electron Devices*, vol. 16, no. 4, pp. 338–347, 1969.

[63] J. Lienig and M. Thiele, *Mitigating Electromigration in Physical Design.* Cham: Springer International Publishing, 2018, pp. 99–148. [Online]. Available: http://link.springer.com/10.1007/978-3-319-73558-0{_}4

[64] J. Xie, V. Narayanan, and Y. Xie, "Mitigating electromigration of power supply networks using bidirectional current stress," *Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI*, pp. 299–302, 2012.

[65] A. Bansal, R. Rao, J. J. Kim, S. Zafar, J. H. Stathis, and C. T. Chuang, "Impacts of NBTI and PBTI on SRAM static/dynamic noise margins and cell failure probability," *Microelectronics Reliability*, vol. 49, no. 6, pp. 642–649, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.microrel.2009.03.016

[66] S. Khan, I. Agbo, S. Hamdioui, H. Kukner, B. Kaczer, P. Raghavan, and F. Catthoor, "Bias Temperature Instability analysis of FinFET based SRAM cells," *Proceedings -Design, Automation and Test in Europe, DATE*, 2014.

[67] P. Weckx, B. Kaczer, H. Kukner, J. Roussel, P. Raghavan, F. Catthoor, and G. Groeseneken, "Non-Monte-Carlo methodology for high-sigma simulations of circuits under workload-dependent BTI degradation-application to 6T SRAM," *IEEE International Reliability Physics Symposium Proceedings*, pp. 1–6, 2014.

[68] 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)*, 2012, pp. 342–346.

[69] 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.

[70] I. Agbo, M. Taouil, S. Hamdioui, P. Weckx, S. Cosemans, and F. Catthoor, "Bti analysis of sram write driver," in *2015 10th International Design Test Symposium (IDT)*, 2015, pp. 100–105.

[71] X. Wang, W. Xu, and C. H. Kim, "Sram read performance degradation under asymmetric nbti and pbti stress: Characterization vehicle and statistical aging data," in *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, 2014, pp. 1–4.

[72] I. Agbo, M. Taouil, S. Hamdioui, P. Weckx, S. Cosemans, F. Catthoor, and W. Dehaene, "Read path degradation analysis in sram," in *2016 21th IEEE European Test Symposium (ETS)*, 2016, pp. 1–2.

[73] 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.

[74] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Designing a processor from the ground up to allow voltage/reliability tradeoffs," *Proceedings - International Symposium on High-Performance Computer Architecture*, pp. 1–11, 2010.

[75] X. Zuo and S. K. Gupta, "Asymmetric sizing: An effective design approach for SRAM cells against BTI aging," *Proceedings of the IEEE VLSI Test Symposium*, 2017.

[76] K. Flautner, R. Uhlig, S. Reinhardt, and T. Mudge, “Thread-level parallelism and interactive performance of desktop applications,” *SIGPLAN Notices (ACM Special Interest Group on Programming Languages)*, vol. 35, no. 11, pp. 129–138, 2000.

[77] Y. Wang, X. Chen, W. Wang, V. Balakrishnan, Y. Cao, Y. Xie, and H. Yang, “On the efficacy of input vector control to mitigate NBTI effects and leakage power,” in *Proceedings of the 10th International Symposium on Quality Electronic Design, ISQED 2009*, 2009, pp. 19–26.

[78] F. Pellerey, M. Jenihhin, G. Squillero, J. Raik, M. S. Reorda, V. Tihhomirov, and R. Ubar, “Rejuvenation of nbti-impacted processors using evolutionary generation of assembler programs,” *Proceedings of the Asian Test Symposium*, vol. 0, pp. 304–309, 2016.

[79] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, “Impact of nbti on sram read stability and design for reliability,” in *Proceedings of the 7th International Symposium on Quality Electronic Design*, ser. ISQED ’06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 210–218. [Online]. Available: https://doi.org/10.1109/ISQED.2006.73

[80] S. Duan, B. Halak, and M. Zwolinski, “Cell Flipping with Distributed Refresh for Cache Ageing Minimization,” *Proceedings of the Asian Test Symposium*, vol. 2018-October, pp. 98–103, 2018.

[81] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, “Dark silicon and the end of multicore scaling,” *IEEE Micro*, vol. 32, no. 3, pp. 122–134, may 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6175879/

[82] T. Kuroda, “CMOS design challenges to power wall,” *2001 International Microprocesses and Nanotechnology Conference, MNC 2001*, pp. 6–7, 2001.

[83] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. Martin, “Computational sprinting,” *Proceedings - International Symposium on High-Performance Computer Architecture*, no. Hpca, pp. 249–260, 2012.

[84] U. R. Karpuzcu, B. Greskamp, and J. Torrellas, “The BubbleWrap many-core: Popping cores for sequential acceleration,” *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, pp. 447–458, 2009.

[85] R. A. Ashraf, N. Khoshavi, A. Alzahrani, R. F. DeMara, S. Kiamehr, and M. B. Tahoori, “Area-energy tradeoffs of logic wear-leveling for BTI-induced aging,” *2016 ACM International Conference on Computing Frontiers - Proceedings*, pp. 37–44, 2016.

[86] A. H. Ibrahim, M. B. Abdelhalim, H. Hussein, and A. Fahmy, “Analysis of x86 instruction set usage for Windows 7 applications,” *ICCTD 2010 - 2010 2nd International Conference on Computer Technology and Development, Proceedings*, no. Icctd, pp. 511–516, 2010.

[87] A. Akshintala, B. Jain, C. C. Tsai, M. Ferdman, and D. E. Porter, “X86-64 instruction usage among C/C++ applications,” *SYSTOR 2019 - Proceedings of the 12th ACM International Systems and Storage Conference*, pp. 68–79, 2019.

[88] A. H. Ibrahim, M. B. Abdelhalim, H. Hussein, and A. Fahmy, “An analysis of x86-64 instruction set for optimization of system softwares,” *International Journal of Advanced Computer Science*, vol. 1, no. 4, pp. 152–162, 2011.

[89] I. J. Huang and T. C. Peng, “Analysis of x86 instruction set usage for DOS/Windows applications and its implication on superscalar design,” *IEICE Transactions on Information and Systems*, vol. E85-D, no. 6, pp. 929–939, 2002.

[90] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaa, and L. E. Meester, *A Modern Introduction to Probability and Statistics.*, 2005.

[91] R. D. Yates and D. J. Goodman, *Probability and Stochastic processes : a friendly introduction for electrical and computer engineers*, 2nd ed. Wiley India, 2012.

[92] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, “Near-Threshold RISC-V core with DSP extensions for scalable IoT endpoint devices,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[93] S. Gal-on and M. Levy, “Exploring CoreMark™ - A Benchmark Maximizing Simplicity and Efficacy,” *The Embedded Microprocessor Benchmark Consortium (EEMBC)*, 2012. [Online]. Available: www.eembc.org

[94] 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,” *2001 IEEE International Workshop on Workload Characterization, WWC 2001*, pp. 3–14, 2001.

[95] D. H. Kraak, C. C. Gursoy, I. O. Agbo, M. Taouil, M. Jenihhin, J. Raik, and S. Hamdioui, “Software-based mitigation for memory address decoder aging,” *LATS 2019 - 20th IEEE Latin American Test Symposium*, 2019.