**Computer Engineering**
Mekelweg 4,
2628 CD Delft
The Netherlands
**http://ce.et.tudelft.nl/**

2014

# MSc THESIS

# Aging Mitigation Schemes for Embedded Memories

**ANTENEH BOGALE GEBREGIORGIS**

**CE-MS-2014-10**

**Abstract**

With the continuous miniaturization of CMOS technology into the nanometer regime, the reliability of SRAM memories is threatened by accelerated transistor aging such as Bias Temperature Instability (BTI), Hot Carrier Injection (HCI) and gate oxide breakdown. Among these mechanisms BTI is known to be the primary aging mechanism in nanoscale devices. The overall effect of BTI is a gradual increase of threshold voltage (Vth). BTI significantly reduces the Static Noise Margin (SNM) of an SRAM cell and makes it more susceptible to failures. To address the impact of BTI in memory array a variety of bit flipping techniques has been proposed. However, all the proposed bit flipping techniques require at least an additional column to store the inversion flag which imposes considerably large area overhead. In this thesis, we propose two techniques to mitigate BTI induced aging in embedded memory: aging-aware instruction encoding and self-controlled bit flipping; both schemes take the workload into consideration.

The aging-aware instruction encoding technique is based on changing the encoding of the Instruction Set Architecture (ISA) in order to balance the occurrence probabilities of 1s and 0s and therefore minimize the impact of BTI in the embedded memory. To evaluate this scheme, we used the Leon2 processor for course case study. A C++ based simulation environment was used to exhaustively search for an encoding resulting in a balanced occurrence probabilities of 0s and 1s. The simulation results revels that on average up to 30% SNM degradation improvement.

Self-controlled bit flipping is based on inverting the content of the memory array during write operation with respect to a specific bit of the written word referred to as flip bit; this bit is left untouched during the write and used as a reference bit to indicate either the written data is inverted or not. Simulation results show that up to 33% SNM degradation improvement can be achieved. The area overhead of the proposed technique is the flip circuitry only. For this reason, our technique saves 64% of the area overhead induced by the periodic flipping techniques.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

# Aging Mitigation Schemes for Embedded Memories

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

ANTENEH BOGALE GEBREGIORGIS
born in ZALANBESSA, ETHIOPIA

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

# Aging Mitigation Schemes for Embedded Memories

by ANTENEH BOGALE GEBREGIORGIS

**Abstract**

With the continuous miniaturization of CMOS technology into the nanometer regime, the reliability of SRAM memories is threatened by accelerated transistor aging such as Bias Temperature Instability (BTI), Hot Carrier Injection (HCI) and gate oxide breakdown. Among these mechanisms BTI is known to be the primary aging mechanism in nanoscale devices. The overall effect of BTI is a gradual increase of threshold voltage (Vth). BTI significantly reduces the Static Noise Margin (SNM) of an SRAM cell and makes it more susceptible to failures. To address the impact of BTI in memory array a variety of bit flipping techniques has been proposed. However, all the proposed bit flipping techniques require at least an additional column to store the inversion flag which imposes considerably large area overhead. In this thesis, we propose two techniques to mitigate BTI induced aging in embedded memory: aging-aware instruction encoding and self-controlled bit flipping; both schemes take the workload into consideration.

The aging-aware instruction encoding technique is based on changing the encoding of the Instruction Set Architecture (ISA) in order to balance the occurrence probabilities of 1s and 0s and therefore minimize the impact of BTI in the embedded memory. To evaluate this scheme, we used the Leon2 processor for course case study. A C++ based simulation environment was used to exhaustively search for an encoding resulting in a balanced occurrence probabilities of 0s and 1s. The simulation results revels that on average up to 30% SNM degradation improvement.

Self-controlled bit flipping is based on inverting the content of the memory array during write operation with respect to a specific bit of the written word referred to as flip bit; this bit is left untouched during the write and used as a reference bit to indicate either the written data is inverted or not. Simulation results show that up to 33% SNM degradation improvement can be achieved. The area overhead of the proposed technique is the flip circuitry only. For this reason, our technique saves 64% of the area overhead induced by the periodic flipping techniques.

**Laboratory** : Computer Engineering
**Codenumber** : CE-MS-2014-10

**Committee Members** :

| | |
|---|---|
| **Advisor:** | Dr.Ir. Said Hamdioui, CE, TU Delft |
| **Chairperson:** | Prof.Dr.Ir. Koen Bertels, CE, TU Delft |
| **Member:** | Dr.Ir. Zaid Alars, CE, TU Delft |
| **Member:** | Dr.Ir. Rene van Leuken, CAS, TU Delft |

*Dedicated to my mom who always believe in me!!*

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**BTI, -** bias temperature instability

**CMOS, -** complementary metal oxide semiconductor

**EM, -** electro-migration

**HCI, -** hot carrier injection

**ISA, -** instruction set architecture

**TDDB, -** time dependent dielectric breakdown

**MIPS, -** millions of instructions per second

**MOS, -** metal oxide semiconductor

**MOSFET, -** metal oxide semiconductor field effect transistor

**MTTF, -** mean time to fail

**NBTI, -** negative bias temperature instability

**NMOS, -** n-type metal oxide semiconductor

**PBTI, -** positive bias temperature instability

**PMOS, -** p-type metal oxide semiconductor

**RTN, -** random telegraph noise

**SNM, -** static noise margin

**SRAM, -** static random access memory

**VLSI, -** very large scale integrated circuits

# Acknowledgements

The study and research work presented in this thesis has been carried out at the Computer Engineering group, Delft University of Technology, Netherlands in collaboration with the Chair of Dependable Nano Computing, Karlsruhe Institute of Technology (KIT), Germany. The research work was sponsored by the European Cooperation in Science and Technology (COST) action. I would like to thank COST action for providing me the financial support required for my research work.

First of all, I would like to express my gratitude to my supervisor, Dr. Ir. Said Hamdioui for the opportunity, continuous guidance and supervision throughout the completion of my project work. His continuous effort of checking my work, motivation and help was a very crucial input for the successful completion of my project. I would also like to thank Dr. Ir. Zaid Al-Ars for his effort to find me a suitable project. In addition to his willingness to discuss on different project ideas, Zaid enriched me with various project startup inputs.

I would like to thank Professor Dr. Ir. Mehdi B. Tahoori, chair of the dependable nano computing research group of KIT for his encouragement, support, valuable comments and suggestions during my stay at KIT. I would also like to thank Mr. Mojtaba Ebrahimi, Fabian Oboril, Saman Kiamehr and other members of the group for their vital inputs for the successful completion of my work. Though my stay at KIT was short but it was full of collaboration and very friendly discussions.

Last but not least I would like to thank all my thesis committee members for their willingness to evaluate my work.

ANTENEH BOGALE GEBREGIORGIS
Delft, The Netherlands
July 7, 2014

# 1 Introduction

For the last 40 years, the performance of microprocessors has shown a tremendous increase from generation to generation. This tremendous increase in microprocessor's performance is a result of the continuous scaling in oxide layer and geometry (width and height) of CMOS devices. Scaling from one generation to the next generation in CMOS technology increases the transistor count per unit area. This increase in transistor count per unit area enables to manufacture high end processors at an affordable cost. In 1965 Gorden Moore, co-founder of Intel, states that the number of transistors in an integrated circuit doubles approximately every two years, which is today commonly known as Moore's law [2]. Even though technology scaling has a vital contribution on the performance of modern microprocessors, there are various consequences that technology scaling brings with it. With scaling in deep sub-micron regime processor power dissipation and temperature have been increasing at an alarming rate [3]. Besides, testability, design productivity, on field and lifetime reliability of newly manufactured devices are becoming more and more challenging. More importantly, scaling accelerates aging induced failures of CMOS devices.

In the nanometer regime the main lifetime reliability challenges due to device aging (wear-out) includes: Time Dependent Dielectric Breakdown (TDDB), Bias Temperature Instability (BTI), Hot Carrier Injection (HCI) and Electromigration (EM). failures due to these mechanisms are shrinking the lifetime of devices from generation to generation. The reliability bathtub curve shown in Figure 1.1 elaborates the shrinking of the useful life time of CMOS devices from generation to generation.

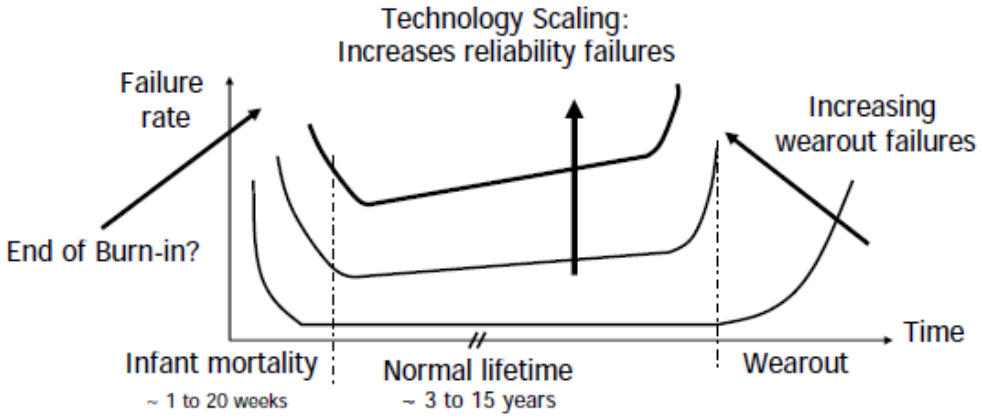



Figure 1.1: Reliability failure classification [1]

In the nano scale devices as the gate oxide thickness decreases BTI becomes more dominant failure mechanism [4]. In this thesis, we propose two techniques to mitigate BTI induced failures of embedded memories: aging-aware instruction encoding and self-controlled bit flipping; both schemes take the workload into consideration.

## 1.1 Motivation

The continuous shrinking of oxide layer and transistor geometry increases the failure probability of transistors. Hence, technology scaling accelerates transistor failure rate. Moreover, the increase in the number of transistors per unit area leads to an increase in the number of failing transistors per unit area. Most of the aging induced failures such as BTI and HCI lead to an increase in transistor's threshold voltage ($V_{th}$), which in turn increases circuit delay. To overcome this $V_{th}$ shift induced delay, designers are using guard banding approach; a process of adding some margin to the timing requirement of their design [5]. Guard banding technique can work well for logic circuits to meet their timing requirements. For memory components delay during read and write operation is not a very critical issue, however, stability of the value stored in the memory cells is highly affected by BTI induced failures. Hence, to address the impact of BTI in memory arrays (caches and register-file) various works have been done by researchers [6, 7, 8, 9].

A memory cell will be under BTI stress when the cell have unbalanced signal probability (probability of storing '0' and '1'). To address this stress driven BTI effect in memory structures various researchers have proposed different techniques to balance the stress time of memory cell by balancing its signal probability [6, 7]. BTI significantly degrades the Static Noise Margin (SNM) of an SRAM cell and makes it more susceptible to failures [7, 8, 10, 9]. SNM is an important reliability metric for SRAM cell stability representing the minimum amount of voltage noise that can lead to shift the value stored in the cell.

Since memory structures occupy most of the area of modern microprocessors, BTI mitigation techniques that does not impose an additional area overhead are highly demanded. In this thesis work we propose two different techniques to fulfill this demand. The first technique is to explore an aging-aware instruction encoding option to mitigate aging of instruction cache. To the best of our knowledge, this work is the first effort to explore an instruction encoding alternatives from instruction cache aging perspective. The second technique is a low cost self-controlled bit flipping technique that addresses BTI induced failure of different cache units and register-file.

## 1.2 Thesis Contribution

Since BTI is the dominant failure mechanism in nano scale devices [4] this thesis contributes to the area of mitigating BTI impact in memory arrays. More specifically, this work investigates an aging-aware instruction encoding that imposes less BTI impact on the instruction cache of a processor and to introduces a low cost self-controlled bit flip-

ping technique that addresses BTI induced aging of different memory structures. Hence, the main objectives of this thesis includes:

- To find an encoding scheme that imposes less BTI impact on the instruction cache with no or considerably less area overhead.
- To implement a low cost self-controlled bit flipping technique that addresses BTI induced failure of different memory arrays.
- To compare the achieved improvement of the proposed techniques with the state-of-the-art solutions in terms of static noise margin degradation improvement.
- To analyze the added area overhead of the proposed techniques in comparison to other state-of-the-art bit flipping techniques.

## 1.3 Thesis Outline

The remainder of the thesis is structured as follows: Chapter 2 discusses various reliability failure mechanisms and their models. The chapter starts by classifying the failure mechanisms into different categories followed by a detail description of the mechanisms in each category. Then it discusses various models of aging induced degradation mechanisms. In this work we use SPARC compliant Leon2 processor which implements SPARC instruction encoding. Hence, in Chapter 3 an analysis of the SPARC instruction encoding is discussed. The chapter describes the instruction formats and their groups followed by an explanation of the instruction encoding and register usage.

To mitigate the impact of BTI in an instruction cache; an aging-aware instruction encoding technique is discussed in Chapter 4. The chapter starts by presenting the main idea of the aging-aware encoding followed by analysis of the improvement opportunities of the SPARC encoding. Then a detailed description of the proposed aging-aware instruction encoding algorithm is presented. Chapter 4 is concluded by discussing the experimental setup and results of the proposed technique. To address the issue of BTI in different memory structures (such as cache units and register-file). Chapter 5 presents a low cost self-controlled bit flipping technique. The chapter starts by presenting a detailed description of the self-controlled bit flipping technique followed by an experimental results and comparison with state-of-the-art techniques. Chapter 5 is concluded by presenting an analysis and comparison of area overhead of the proposed techniques. Finally, chapter 6 concludes the thesis by providing a conclusion and recommendation for future works.

# 2 Reliability Failure Mechanisms

Reliability failure is the inability of a device to perform its proper functionality throughout its intended lifetime. Reliability failure mechanisms are electrical, chemical, physical and metallurgical phenomena stimulated by events such as electric field, electronic charges, semiconductor doping, temperature, radiations or moisture [11]. An existence of single or multiple combination of reliability failure mechanism results in a variation of device parameters, functional or structural damages of the semiconductor devices [11]. Classification of reliability failure mechanisms and different models for each mechanism are discussed in this chapter. Section 2.1 provides the classification of reliability failure mechanisms. Section 2.2 discusses design and manufacturing failure mechanisms. In section 2.3 environmental failure mechanisms are discussed. Degradation failure mechanisms and their models are presented in section 2.4. Finally, the impact of BTI on SRAM cell and its model is presented in Section 2.5.

## 2.1 Classification of Reliability Failures

Identification and classification of reliability failures is an important step in analyzing the root cause of the failures and to apply an appropriate measure to mitigate their impact. Figure 2.1 presents a high level classification of reliability failures based on their initial cause. As shown in the figure reliability failures can be classified into three main categories:

1. Design and manufacturing failures: These are failures that are caused by the errors made during device design or manufacturing time.

2. Degradation failures: These are failures due to aging of the devices.

3. Environmental failures: These are failures that are caused by the radiations and variation of operating environment of a device.

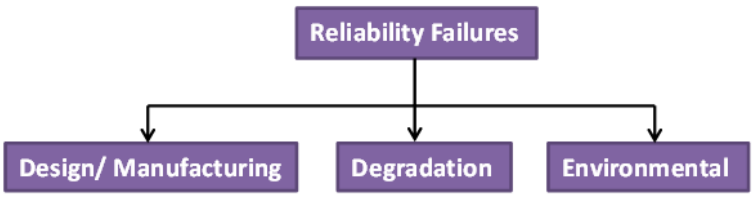



Figure 2.1: Reliability failure classification

## 2.2 Design and Manufacturing Failures

Design and manufacturing failures are failures caused by manual or machine (such as lithography) errors of device parameters during design and manufacturing time. Process variation and crosstalk are the most common and widely studied design and manufacturing failures [12].

### 2.2.1 Process Variation

Manual or lithographic imperfectness during processing of Complementary Metal Oxide Semiconductor (CMOS) devices lead to a variation of device geometry such as length width and oxide layer thickness of the device. These variations of device geometry can lead to a variation in voltage and current characteristics such as shift in threshold voltage of the device [12]. Process variation can lead to a measurable decrease in the output performance of an integrated circuits. Based on the granularity, process variations are classified into two categories [13]:

- Die-to-die (inter-die) variation: are variations resulting from parameter variation of devices in different dies. These variations can be further classified as lot-to-lot and wafer-to-wafer. Die-to-die variations affect all transistors and interconnecting wires on a die uniformly. Die-to-die variations can occur due to different factors such as processing temperature and equipment properties.

- Within-die (intra-die) variation: are variations that occur among devices of single die. Within-die variations can occur when there is a non-deterministic placement of dopant atoms or channel length variations within a single die.

### 2.2.2 Crosstalk

The increase in transistor density per unit area of an integrated circuit has made an increase in the noise level such as coupling capacitance. More importantly, the interconnect wire becomes thicker and narrower which leads to an increase in the interference level of the interconnect wires which is commonly known as crosstalk [14]. Crosstalk is a phenomenon which arises due to the coupling capacitance between the neighboring nets in a coupled interconnects systems, which can impact the functionality, timing, and reliability of circuit. For deep sub-micron technology the wire spacing decreases from generation to generation which increases the signal slew rate, capacitive inductance and cross coupling or crosstalk between adjacent wire lines [14]. Crosstalk lead to an increase in the noise level which could disturb the proper functionality of circuits.

To minimize the impact of crosstalk it is important to identify the dependencies of crosstalk and determine an optimal spacing in interconnect wires. The challenging task is determining the optimal spacing in a way the scaling objective is achieved. Authors in [14] discussed various dependencies of crosstalk.

## 2.3 Environmental Failure Mechanisms

Environmental failures are failures that are driven by an external factor. Environmental failures cause or accelerate the failure of semiconductor devices. Operating temperature, supply voltage and cosmic ray radiations are the main environmental factors that affect device reliability. In the following sub sections we will discusses how this environmental factors contribute to the failure of semiconductor devices.

### 2.3.1 Voltage Variation

Due to various factors such as variations in switching activity, the supply voltage can vary from the designed value during operation time. This fluctuation in supply voltage determine the stability of the supply power of circuits which in turn can drastically affect performance of the circuit [15]. Voltage variation leads to uneven power dissipation which can result to the formation of temperature hot spot and temperature variation.

### 2.3.2 Temperature Variation

Due to the increase in power dissipation operating temperature of CMOS device is also increasing from time to time. The fluctuation in operating temperature is becoming more and more challenging with technology scaling in the nanometer regime [15]. Temperature variation can be categorized into two different categories [13]; temporal and spatial temperature variation. Spatial temperature variation occurs when there is a temperature hot-spot around an active unit sitting nearby a passive or less active unit which causes difference in transistor performance. Temporal temperature variation occurs when the device switches between idle and active time. Temporal variation cause the die temperature to increase or decrease depending upon the ON or OFF state of the device. Overall temperature variation can affect threshold voltage of transistors which can lead to circuit delay.

### 2.3.3 Latch Up

Latch up is a phenomenon that is triggered by the parasitic structure in CMOS devices that inadvertently creates a low-impedance path between power supply lines and causing over-current in the MOSFET circuit [16]. The over-current is latched by the parasitic structure and can only be resolved by power-down. An intense ionization, thermal spike, or voltage overshoot and undershoot can lead to the latch up phenomenon [16]. The impact of latch up includes parametric variation, disrupting proper function of the circuit, and even destructing the device.

### 2.3.4 Soft Errors

Soft Error is a kind of error where a datum is wrongly changed its value due to a noise (electrical and magnetic) or various radiations (e.g. alpha particles, cosmic rays, and thermal neutrons) [17]. Soft errors are random, temporary and non-reproducible.

The impact of soft error includes single bit flip and multiple bit upsets in the storage elements and transient events in logic units [18].

Chip packaging materials contain a small amount of unstable radioactive contaminants which always occurs with alpha particle emission. The positively charged alpha particle can easily penetrate into a semiconductor and generate an electron-hole pairs alongside. The electron-hole pairs are separated by electric field to the opposite polarity nodes, hence disturbing the stored charge distribution. If the charge collected exceeds the critical charge required, the transistor will switch its state without any external voltage [17]. Cosmic rays can also create energetic neutrons and protons, which generate electron-hole pairs while losing energy [18]. Thus cosmic rays have the same mechanism as alpha particles in soft error.

## 2.4 Degradation Failure Mechanisms

Degradation failures are failures that inherently exist within the transistor. However, they are stimulated later during operation time due to the aging of the semiconductor devices. Degradation failures are more pronounced at the nano scale devices. Degradation failures gradually decrease the performance of the device and finally the device will fail to perform its functionality before its intended lifetime is reached. Figure 2.2 shows the main degradation failure mechanisms in CMOS transistors and the interconnect wires.

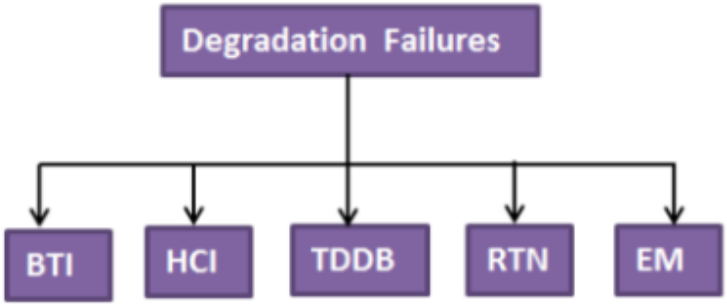



Figure 2.2: Degradation failure classification

Figure 2.3 shows the location of the degradation failure mechanisms in a CMOS transistor and an interconnect wires, i.e BTI and HCI are failures at the Si-$SiO_2$ interface of MOS transistor, TDDB is a failure in the $SiO_2$ layer, RTN is a result of trapping and de-trapping of carriers inside the oxide and electro-migration is a phenomena that takes place in the interconnect wires. A detailed explanation of degradation failure mechanisms, their impact on the device parameters and state-of-the-art models of these mechanisms is discussed in this section.

### 2.4.1 Bias Temperature Instability

BTI is a phenomenon that gradually increases the threshold voltage of transistors over a long period of time, causing the drive current to decrease. BTI has two phases known

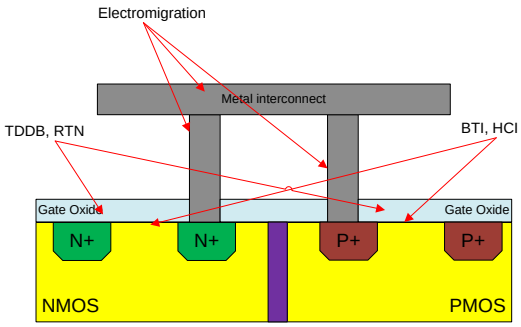
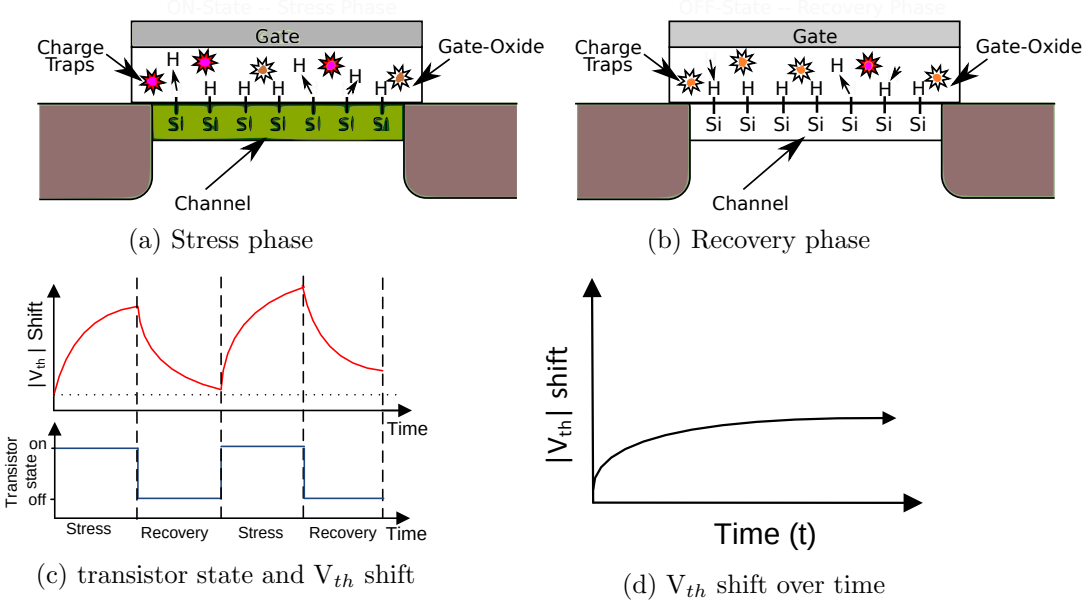



Figure 2.3: Reliability failures on transistor and interconnect

as *stress* and *recovery*, depending on the transistor state (on/off), it is in one of these phases. During the stress phase where the transistor is on, the field-dependent reaction at $Si/SiO_2$ interface generates traps by breaking the Si-H bonds inside the dielectric bulk and the released H species escape away from $SiO_2$ interface (see Figure 2.4a). Furthermore, the generated traps become active and contribute to the increase of the transistor threshold voltage.




(a) Stress phase

(b) Recovery phase

(c) transistor state and $V_{th}$ shift

(d) $V_{th}$ shift over time

Figure 2.4: BTI during stress and recovery phase

When the transistor is off, it enters to the recovery phase and the escaped H species start to diffuse back towards the $SiO_2$ interface and recombine with the Si species to reform the Si-H bonds (Figure 2.4b). As a result, some of the charged traps generated during the stress phase become inactive again. However, during the recovery phase, (as shown in Figure 2.4b) not all the H species are defused back to their original positions and reform the bond, which eventually results in a net $V_{th}$ increment. Figure 2.4c

shows the shift in $V_{th}$ of a transistor. When the transistor is on, the rate of breaking the Si-H bond and generation of charged traps increase which leads to an increase in the transistor $V_{th}$. When the transistor is off, $V_{th}$ decreases due to the healing effect of the recovery phase. The net increase in the transistor threshold voltage over a long period of time is shown in Figure 2.4d.

In general, BTI can be classified as *Negative Bias Temperature Instability* (NBTI) in PMOS transistors and *Positive Bias Temperature Instability* (PBTI) in NMOS transistors. While NBTI was claimed to be the primary reliability concern, however, PBTI is emerged as an important issue with the introduction of high-k materials [19].

**Models of NBTI**

At transistor level various models of NBTI have been specified. In this section we will discuss the widely studied Reaction Diffusion (RD) and disorder-controlled kinetics models.

#### 2.4.1.1 Reaction-Diffusion (RD) Model

Jeppson et al. in [20] proposed the fundamental concept of the Reaction Diffusion (RD) model that prompted many researchers [21] to model NBTI mechanism. The essence of the RD model is that the interface traps are the main cause of NBTI. This model is described in [21] by using the Equations 2.1 - 2.4 given below:

$$\frac{dN_{IT}}{dt} = K_F(N_0 - N_{IT}) - K_R N_H N_{IT}(x = 0) \tag{2.1}$$

$$\frac{dN_{IT}}{dt} = D_H\left(\frac{dN_H}{dx}\right) + \left(\frac{\delta}{2}\right)\frac{dN_H}{dt}(0<x<\delta) \tag{2.2}$$

$$D_H\left(\frac{d^2N_H}{dx^2}\right) = \frac{dN_H}{dt}(\delta<x<T) \tag{2.3}$$

$$D_H\left(\frac{dN_H}{dx}\right) = K_F N_H(X>T) \tag{2.4}$$

Where x=0 denotes the $Si/SiO_2$ interface and x>0 is (in the oxide) towards the gate. $N_{IT}$ is the number of interface traps at any time instance. $N_0$ is the initial number of unbroken Si-H bonds and $N_H$ is the hydrogen concentration. $K_F$ is the oxide field dependent forward dissociation rate constant and $K_R$ is the annealing rate constant. $D_H$ is the hydrogen diffusion coefficient. $\delta$ is the interface thickness. T is the oxide thickness and $K_p$ is the surface recombination velocity at the oxide/poly-silicon interface. Note that none of the terms in Equation 2.3 are field dependent. It means that the diffusion species in R-D model is assumed neutral.

**Properties of the RD model**

1. Stress phase
   Figure 2.5 shows the general solution of RD model in the stress phase consists of five different regimes of time-dependent interface trap generation [22].

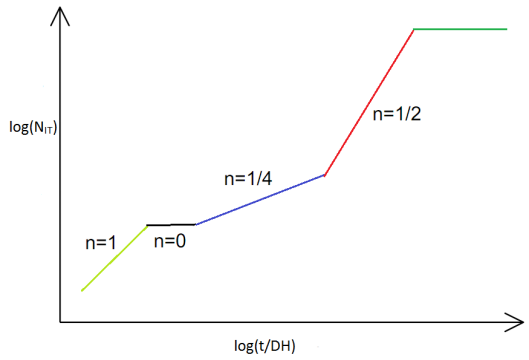



Figure 2.5: Five regimes of time dependent interface-trap generation from RD model

- Regime 1 (n=1, n is time exponent): At the starting point, both $N_{IT}$ and $N_H$ are small and negligible. According to Eq.2.1,

$$N_{IT} \sim K_F N_0 t \tag{2.5}$$

- Regime 2 (n=0): During the second phase, the forward and reverse reactions in Equation 2.1 are both large but approximate equal (i.e., $K_F N_0 \sim K_R N_H N_{IT}(x=0)$) and also all the hydrogen released are still at the interface so that $N_H(x=0) = N_T$. Therefore, according to Equation 2.1,

$$K_{IT} \sim \left(\frac{K_F N_0}{K_R}\right)^{0.5} t^0 \tag{2.6}$$

  However, for both of these two phases, they only last up to a few microsecond, and therefore are not observed in typical NBTI measurement.

- Regime 3 & 4 (n=0.25): In practical NBTI experiments, this regime is typically observed after the hydrogen diffusion, which begins to control the trap generation process. Equation 2.3 suggests $x = (D_H t)^{0.5}$ and by Equation 2.2, $N_H(x=0) \sim \left(\frac{t}{D_H}\right)^{0.5} \cdot \left(\frac{dN_{IT}}{dt}\right)$. Substituting this in Equation 2.1 and assuming that the net trap generation is so slow, thus in Equation 2.1, $\left(\frac{dN_{IT}}{dt} \sim 0.\right)$, we find,

$$N_{IT} \sim \left(\frac{K_F N_0}{K_R}\right)^{0.5} \cdot (D_H t)^{0.25} \tag{2.7}$$

- Regime 5 (n=0.5): Once the hydrogen reaches the oxide/poly-interface, according to Equation 2.1, 2.2 and 2.4, we can get,

$$N_{IT} = A \left(\frac{K_F N_0}{K_R}\right)^{0.5} \cdot (D_H t)^{0.5} \tag{2.8}$$

  Where $A = \left[2\left(\frac{D_H}{K_p} + T\right)\right]^{-0.5}$.

2. Annealing phase:
   Besides stress phase, another important phase of RD model is called annealing phase. Once the stress is removed, the diffused H species will move back and recombine with the Si dangling bonds to form a passive Si-H state.

#### 2.4.1.2 Disorder-controlled Kinetics Model

Kaczer et al., in [23] proposed a model that indicates interface traps as the prime source of NBTI induced degradation in a MOS transistor. This model can be described by the equations shown below [23]:

1. Stress phase:

$$\Delta V_{th} = At^b \propto N_{IT} \tag{2.9}$$

   Where A $\propto \left(\frac{G}{R}\right)^{0.5} \left(\frac{D}{V}\right)^{0.25} \left(\frac{N_C}{N_V}\right)^{0.25}$ and b=$(\frac{\alpha}{4})$=$\frac{K_B T}{4E_0}$ ; $\alpha$ is dispersive parameter and T is temperature; G and R are the generation rate and recombination rate of interface states D and V are temperature independent technology constant. t is the stress time and b is time exponent. $E_0$ is the characteristic density-of-states width. $K_B$ is Boltzmann constant.

2. Recovery phase:
   The model propose that the log(t)-like dependence of recovery arrives from the wide distribution of hopping time typical for dispersive transport. Thus, the relaxation rate is directly linked with the stress phase though the dispersion parameter $\alpha$. It can be described as the equation shown below:

$$N_{IT}(\varepsilon) = \frac{1 - \varepsilon^{\frac{\alpha}{2}}}{1 - \varepsilon^{\alpha}} \cdot N_{IT}(0) \tag{2.10}$$

   where $\varepsilon = \frac{t}{t_{stress}}$ is the dimensionless relaxation time and $N_{IT}(0)$ is the interface state density at the beginning of relaxation.

### 2.4.2 Time Dependent Dielectric Breakdown

Time dependent dielectric breakdown, also termed as gate oxide breakdown, is emerging as one of the most significant sources of degradation in CMOS devices with technology scaling. In addition to the thin gate oxide, the saturating trend in supply voltage scaling results in a large electric field across the thin dielectric layer which forms traps in the dielectric layer [24]. These traps assist the tunneling of carriers leading to the tunneling current. This step can be termed as soft oxide breakdown (SBD). Although these traps may lead to a performance degradation, such as threshold voltage shift and random gate leakage, the device is still functioning [24].

Initially the generated traps are non-overlapping and thus do not conduct. With more traps generated, they start to overlap and this may lead to the formation of a conducting path. As shown in Figure 2.6, once the conduction channel is formed, more traps

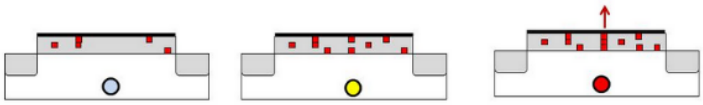


Figure 2.6: Different phases in Time Dependent Dielectric Breakdown

generation causes wider conduction path and hence more current flows which leads to a higher temperature [25]. This thermal runaway condition leads to a catastrophic failure known as the hard oxide breakdown (HBD). After that, the device does not function properly.

During the second phase, the traps accumulate in the oxide and cause degradations. The transition from the beginning of SBD to HBD is not abrupt, and gate leakage current starts to progressively increase long before hard breakdown occurs. TDDB degradation accelerates as the thickness of the gate oxide decreases with continued device scaling. ITRS'03 [26] predicts the equivalent oxide physical thickness for high-performance logic technology to be 1.2nm at 90nm technology node, however, oxide thickness less than 2.5nm will not be able to sustain the operating voltage for their full expected lifetime [27]. Therefore, TDDB is becoming a potential reliability challenge for sub-100nm CMOS technology.

**Models of TDDB**

At transistor level four main models of TDDB have been studied in detail. In this section we will discuss the two most common models. The two models are E-model and 1/E-model.

#### 2.4.2.1 E-model

The E-model, also termed as Thermo-chemical model, suggests a direct correlation between oxide field and the oxide degradation. It was originally proposed by McPherson et al. in [28]. This model indicates that the weak Si-Si bonds within the oxide layer experience heavy strains due to the strong oxide field and some of the bonds may break and create some charged traps after they obtain sufficient thermal energy [28]. These charged traps are accumulated and when critical value is reached they result in the oxide breakdown. There is an exponential relationship between the MTTF and the oxide electric field. This exponential relationship can be numerically expressed by [28]:

$$MTTF = Ae^{(-\gamma E)} \times e^{\left(\frac{E_a}{KT}\right)} \tag{2.11}$$

And

$$E_a = \Delta H_0 - \alpha \times E_{ox} \tag{2.12}$$

Where A is technology constant and E is the applied oxide field. $\Delta H_0$ is enthalpy of activation and $\alpha$ s constant and refers to effective dipole moment. $E_a$ is thermal

activation energy, which has a linear dependence on oxide field and $\gamma$ is a field acceleration parameter.

#### 2.4.2.2 1/E-model

The 1/E-model, also called Anode Hole Injection (AHI) model, is proposed by Duschl in [29] assumes gate oxide breakdown is triggered by the trapping of holes at localized region in the oxide layer, which either enhances the cathode field or leads to the oxide electron trap generation [25]. Thus, there are increasing number of electrons that are injected into the oxide layer. By means of impact ionization, again, it facilitates the trapping of holes and increases the trap density until they reach a critical density, which eventually lead to sudden breakdown of oxide [30]. Similar to the E-model, the reciprocal of oxide field and the MTTF due to TDDB have an exponential relationship [29], which can be analytically represented by using Equation 2.13:

$$MTTF = \tau_0 e^{\left(\frac{\beta}{E}\right)} \times e^{\left(\frac{E_a}{KT}\right)} \tag{2.13}$$

where $\tau_0$ is a constant and $\beta$ is the electric field acceleration factor (with a typical value of 350MV/cm). E is the applied oxide field and $E_a$ is the activation energy.

### 2.4.3 Hot Carrier Injection

Hot carrier injection is a phenomenon in CMOS devices where energetic electrons or holes at the drain end of the channel overcome the potential barrier at the silicon-oxide interface and inject into the gate oxide [31]. The term "hot" refers to the effective temperature used to model carrier density or energy, not to the overall temperature of the device. The impact of HCI in CMOS device includes threshold voltage increment, trans-conductance degradation, saturation current decreasing and leakage current increasing.

The mechanism of HCI can be summarized into three stages. Firstly, the channel carriers becomes "hot" as they shoot out from the source of a MOSFET, accelerate in the channel, and experience impact ionization near the drain junction. The carries could gain energy of up to 1.5 eV by means of high lateral electric field and electron-electron scattering [31]. Secondly, majority of the carriers are collected at the drain to constitute the drain current. However, a few "lucky" carriers continue to gain energy and overcome the surface energy barrier, injecting into the gate oxide. In order to enter the conduction band of $SiO_2$, an electron must gain a kinetic energy of $\sim$3.2 eV. For holes, the valence band offset in this case dictates they must have a kinetic energy of 4.6 eV. Finally, the injected carriers break Si-H bonds at the silicon-oxide interface or $SiO_2$ inside the oxide layer, generating interface states and oxide charges, which are the main mechanisms for degradation of MOSFET parameters [32]. Since typically no recovery is observed, the switching characteristics of the transistor can be permanently changed [31].

The shifts in threshold voltage and trans-conductance are proportional to the average trap density, which in turn is inversely proportional to the effective channel length [32].

Therefore, reducing the channel length will exacerbate hot carrier effect. For future CMOS technologies, even the power supply voltage will be reduced to 1V or below, HCI is still a significant reliability concern due to continuous scaling of device channel length [33].

**Models of HCI**

We will discuss two models to explain the HCI mechanism. The models are Lucky Electron Model (LEM) [34], and the Energy-Driven Dominant Energy Model [35].

#### 2.4.3.1 Lucky Electron Model

Hu et al. in [34] presented the Lucky Electron Model (LEM) view to explain the HCI mechanism. It is believed that the cause of MOSFET degradation is the generation of interface traps (surface states) through hot channel carriers injecting into the oxide [34]. According to this model [34], the generation of interface traps $\Delta N_{it}$ can be expressed as the equation below:

$$\Delta N_{it} = C_4 \left[ t \frac{I_d}{w} \ell^{\left(-\frac{\psi_{it}}{q\lambda E_m}\right)} \right]^n \tag{2.14}$$

Where $C_4$ is a technology dependent constant and W is the channel width. "t" and "n" are parameters that indicate how $\delta N_{it}$ increases with time. $\lambda$ is the hot carrier mean free path and $\psi_{it}$ is the critical energy that a carrier must have to generate an interface trap. $\psi_{it}/qE_m$ is the distance that a carrier has to travel in the channel electric field $E_m$ to gain sufficient energy $\psi_{it}$ and $\ell^{(-\psi_{it}/\lambda qE_m)}$ is the probability of a carrier that can successfully travel this distance without suffering a collision. Thus $I_d \times \ell^{\left(-\frac{\psi_{it}}{q\lambda E_m}\right)}$ is the proportion of hot carriers that finally have energies greater than $\psi_{it}$, this is the "lucky electron".

For the purpose of predicting device lifetime, we should setup a value of $\Delta N_{it}$, indicating that the device starts to fail when such amount of surface charged states are generated. Hence, Equation 2.14 can then be rewritten in the form:

$$MTTF = C_5 \times \frac{W}{I_d} \ell^{\left[\frac{\psi_{it}}{\lambda qE_m}\right]} \tag{2.15}$$

Equation 2.15 represents the HCI induced Mean Time To Failure (MTTF). Where $C_5$ contains $C_4$, n and the chosen $\Delta N_{it}$ value in Equation 2.14.

#### 2.4.3.2 Energy-Driven Dominant Energy Model

As the scaling of MOSFET technology, the Electron-Energy Distribution Function (EEDF) is more and more dependent only on the applied bias because of the effect of quasi-ballistic transport over high-field region. Rauch et al [35] proposed a new concept that the bias dependence of the impact-ionization (II) rate and hot-carrier lifetime is given by the energy dependences of impact-ionization scattering rate $S_{II}(E)$ and an effective interface state generation (ISG) cross section $S_{IT}(E)$ by first order.

II and ISG are determined by the integral of EEDF f(E) and interaction cross section or scattering rate S(E) as the following form [35]:

$$Rate = \int f(E)S(E)dE \tag{2.16}$$

The integrand of this rate equation peaks at several points, which are referred to as "dominant energies".

$$\frac{d\left[f(E)\cdot S(E)\right]}{dE} = 0 => \frac{d\ln f}{dE} = -\frac{d\ln S}{dE} \tag{2.17}$$

The dominant energy can be controlled by points of high curvature (knee points) of either ln(f) or ln(S). the field-driven paradigm assumes the dominant energy is controlled by the knee points of ln(S) while the energy-driven paradigm assumes the knee points of ln(f) drive the dominant energies [35].

Many literatures claim that the EEDF has a strong knee near the energy range of importance. Based on this, Rauch proposed this knee determines the dominant energies; hence the II and ISG rates are "energy driven". Put another way, the dominant energies track with bias condition, and the hot carrier bias dependences are primarily due to the energy dependence of the interaction cross section or scattering rate.

Under the energy-driven approximation, hot carrier rates ($MTTF^{-1}$) are proportional to the S functions at the available energy. In the linear regime of the S function, electron-electron scattering effects are neglected due to quasi-ballistic transport over the high-field region. Thus the available energy is $qV_{EFF}$ for II and for ISG.

$$MTTF \approx [A \cdot I_s \cdot S_{IT}(qV_{EFF})]^{-1} \tag{2.18}$$

### 2.4.4 Random Telegraph Noise

Random telegraph noise is a phenomena commonly used to describe resistance fluctuations that show random switching between several, often only two, discrete values [36]. Generally, RTN occurs in a very small devices like MOSFETs, p-n junction or metal contacts. In MOSFETs, this fluctuation can be considered as a form of current modulation [37].
For the mechanism of current modulation, two models have been proposed [37]. One is the carrier's number fluctuation due to the trapping at a defect. By alternative capture and emission of an individual active trap inside the oxide layer, the total number of carriers in channel has a time dependent variation, which results in the discrete modulation of channel current.

Another model is the mobility degradation due to enhanced Coulomb scattering. For MOSFETs with a channel area less than 1 $\mu m^2$, the channel is squeezed to the extent that it is possible to have a single oxide trap in the vicinity of silicon oxide surface, as shown in Figure 2.7. The transport of channel carriers is significantly affected by the trap due to the Coulombic repulsion. And finally causes the discrete current modulation.

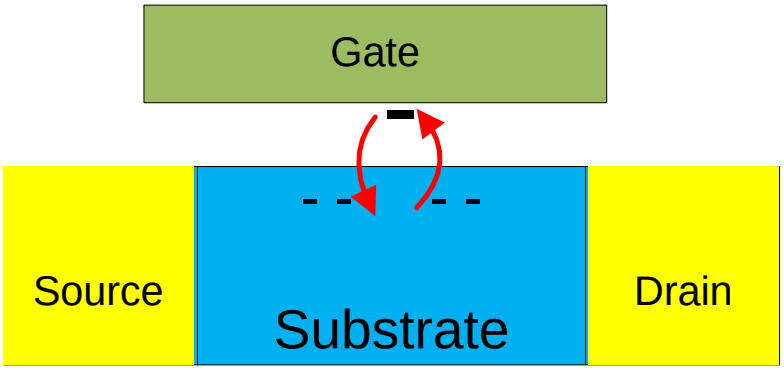



Figure 2.7: Random Telegraph Noise mechanism

### 2.4.5 Electro-migration

Electro-migration is the mass transport of a metal wire due to the momentum exchange between the conducting electrons which move in the applied electric field, and the metal atoms which make up of the interconnecting material [31]. The impact of EM includes producing interconnect resistivity change, and interconnect failure such as short and open faults.

Gradients in temperature, current density or ion-diffusion coefficients will result in the divergence of metal ions flux, while the inhomogeneity in the metal's micro-structure is the most influential factor. Figure 2.8 shows an example of the metal micro-structure's inhomogeneity induced EM failures. This phenomena might happen where two metal layers with different grain sizes intersect, e.g. at contact holes or vias. The two metal layers will exhibit different mass flow levels and hence problems arise at the interconnect points. As shown in Figure 2.8, at the left intersection, an increase in the mass flow will give rise to the void formation. Whereas at the right intersection, mass will accumulate due to a sudden decrease in the mass flow rate, leading to the hillock formation [38].
The interconnecting metal atoms migration has a particularly important effect in applications where high direct current densities are used. Keeping pace with the shrinking of MOSFET physical dimensions leads to higher current density flowing through the interconnects, which exacerbates EM wear out effects on circuit performance and reliability. As a result, EM is the most significant reliability concern in all interconnect failure mechanisms. The original model along with one of its variant will be described in the following subsection.

#### Models of EM

Black's equation is the most common, publicly accepted and widely used model for EM. Hence, Black's equation along with one of its variant will be discussed in this section.

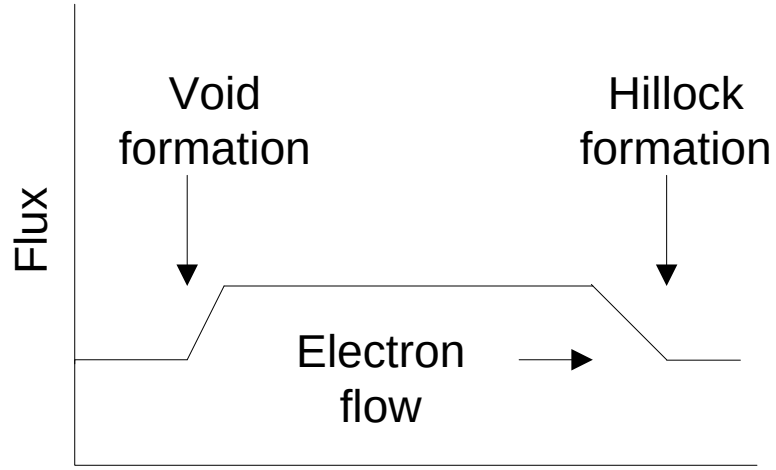



Figure 2.8: Void and hillock formation of EM

#### 2.4.5.1 Original Black's Model

J. R. Black in [39] proposed an empirical formulation for the EM. In this model, Mean Time To Failure depends on the current density and physics of the interconnect materials. Analytical formulation of this model can be derived as the following equation:

$$MTTF = \frac{A}{J^{-2}} exp\left(\frac{E_a}{kT}\right) \tag{2.19}$$

Here A is a constant that depends on the physics of interconnect and its geometry, J is the current density and $E_a$ is the activation energy.

#### 2.4.5.2 Generalized Black's Model

EM failure can be characterized by the "generalized black model" expressed as:

$$MTTF = A\frac{J^n}{T^m} exp\left(\frac{\Delta H}{KT}\right) \tag{2.20}$$

Where n and m depends on the particular failure mechanisms as described in Chapter 2.4 n=1 and m=0 when failure is due to the growth of a pre-existing void in the absence of a high stress or the result of the drift of an edge. n=2 for all nucleation dependent models. Failure due to flux divergences caused by a stress relaxation at a defect would produce n=$\sim$ 2 and m=$\sim$1 and failure due to the presence of sub Blech length grain clusters would have n=m=$\sim$2.

## 2.5 Impact of BTI on SRAM Cell

SRAM is a memory cell which can store a data as long as power is supplied. unlike Dynamic Random Access Memory (DRAM), SRAM doesn't need a periodic refreshment to retain its content. The structure of SRAM cell is formed by two cross coupled CMOS

inverters, in which the output of one of the inverter is feedback as an input to the other inverter and vise versa. This feedback loop of the inverters stabilizes the state of the SRAM cell. The structure of a conventional 6-transistor SRAM cell is given in Figure 2.9.

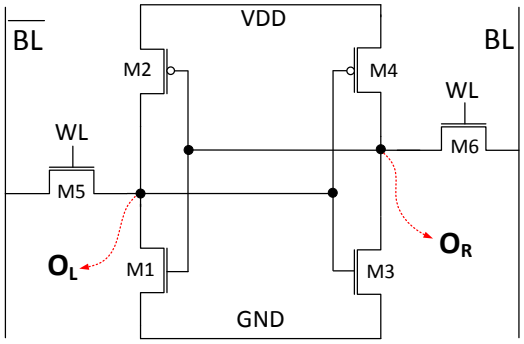



Figure 2.9: Structure of 6-T SRAM cell

In the figure, M1 and M2 are the NMOS and PMOS transistors of the first inverter and M3 and M4 are their equivalent transistors of the second inverter. The access transistors M5 and M6 along with word and bit lines (WL and BL) are used to read and write data from/to the cell.

### 2.5.1 BTI Model for SRAM Cell

As discussed in Section 2.4.1 there are various models of BTI at transistor level. In this section we will discuss a reaction-diffusion (R-D) based model to represent the impact of BTI in SRAM cell. According to [4] the long term $\Delta V_{th}$ shift model is given by Equation 2.21 and Equation 2.22:

$$\Delta V_{th}(t) = \left( \frac{\sqrt{K_v^2 \alpha T_{clk}}}{1 - \beta_t^{\frac{1}{2n}}} \right)^{2n} \tag{2.21}$$

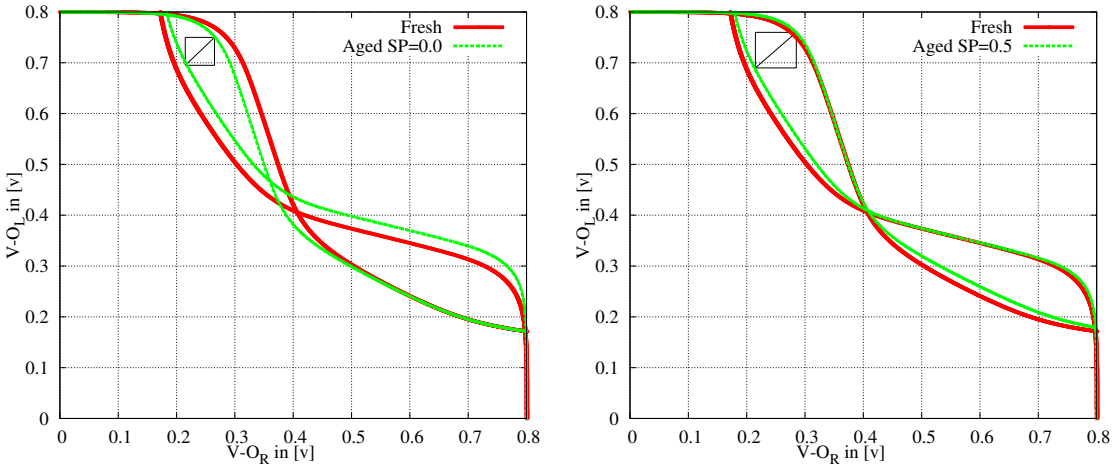
$$\beta_t = 1 - \frac{2\xi_1 t_e + \sqrt{\xi_2 C(1-\alpha)}}{2t_{ox} + \sqrt{Ct}} \tag{2.22}$$

Where $\xi_1$ and $\xi_2$ are constants, $K_v$ and C are temperature dependent coefficient and $t_e$ is the effective diffusion distance. Here $\alpha$ is the input signal probability. This model is used to analyze the impact of BTI in SRAM cells in this work.

### 2.5.2 BTI induced SNM degradation

We have seen that BTI leads to an increase in transistors threshold voltage. The increase in threshold voltage impacts the stability of SRAM cells. This increase in threshold voltage has negligible impact on the write and read delay of SRAM cells; however, it significantly reduce the Static Noise Margin (SNM) of an SRAM cell [40] and make it more susceptible to failures. In order to show the effect of BTI on stability of SRAM cell, we consider the conventional 6T SRAM cell shown in Figure 2.9. SNM is a standard

stability metric of an SRAM cell which is defined as the maximum amount of DC voltage noise that can be tolerated before changing the state of the cell. The stress and recovery




(a) SP=0 (b) SP=0.5

Figure 2.10: The impact of BTI on SNM of a 22 nm SRAM cell

time of an SRAM cell is determined by its signal probability (probability of storing '0' or '1'). Hence, the impact of BTI on an SRAM cell is a strong function of the cell's input signal probability. To explain this issue, we consider two cases, when the $O_L$ node of the SRAM cell shown in Figure 2.9 has a signal probability of 0.0 and 0.5:

- SP($O_L$) = 0.0: In this case, the $M4$ PMOS transistor (see Figure 2.9) is always under NBTI stress and $M1$ NMOS transistor is under PBTI stress which leads to an increase in the threshold voltage of these two transistors. The shift in threshold voltage will eventually degrades the SNM of the cell over time. Figure 2.10a shows the butterfly curve of the SRAM cell for this case.
- SP($O_L$) = 0.5: In this case, $M4$ and $M2$ PMOS ($M3$ and $M1$ NMOS transistors) are under the same amount of NBTI (PBTI) stress. Therefore, SNM is symmetrically reduced for the cases in which $O_L$ value is either '0' or '1' (see Figure 2.10b).

Table 2.1: SNM degradation with different signal probability

| SP | SNM | | |
|---|---|---|---|
| | Fresh | After 3 years | Degradation in % |
| 0.0 | 0.093 | 0.075 | 19.35% |
| 0.5 | 0.093 | 0.088 | 5.37% |

As shown in Table 2.1, for the aged SRAM cell, the SNM degrades much faster when SP=0.0. On the other hand, when SP=0.5 the SNM of the SRAM cell is very close to fresh SNM and it degrades slowly.

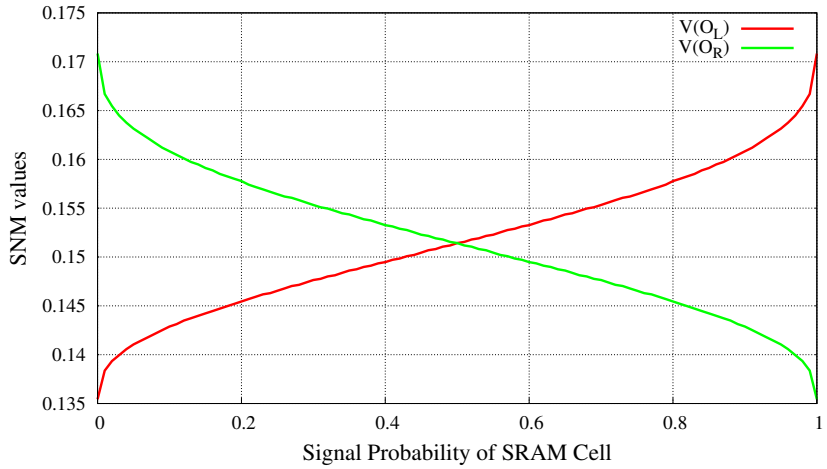



Figure 2.11: SNM of a 45 nm SRAM cell

Figure 2.11 shows SNM of an SRAM cell for different signal probabilities. When the signal probability of the SRAM cell is 0, V-$O_L$ is very small which leads to a smaller SNM value of the left inverter (Figure 2.9), however, V-$O_R$ will be high and the right inverter will have a higher SNM. With increase in the signal probability the SNM of the left inverter increases and SNM of the right inverter starts to decrease. This situation shows the symmetric nature of the SNM degradation of the two cross coupled inverters of an SRAM cell. From SRAM cell stability point of view always the minimum SNM of the two inverters determine the stability of the cell. Hence, the best SNM is obtained when the signal probability of $O_L$ is equal to 0.5, in which both inverters are degrading symmetrically.

# Analysis of SPARC Instruction Encoding

# 3

An instruction is a unique encoding that represents all the informations necessary for a processor to perform a certain task [41]. Instruction set Architecture (ISA) is the implementation of instruction encoding which serves as a boundary between the software and the hardware layers [41]. Based on the amount and complexity of their instructions ISA's can be broadly classified as Complex Instruction Set Architecture (CISC) and Reduced Instruction Set Architecture (RISC). CISC architectures have many complex and variable length instructions. On the other hand RISC architectures have few, simple and fixed length instructions. There are various implementations of CISC (such as x86) and RISC such as MIPS, SPARC, ALPHA which varies on the length of their instructions and their encoding techniques. For example both MIPS and SPARC architectures have a 32-bit instruction length but they vary on the amount of instructions they support and their encoding scheme.

SPARC v8 is the baseline ISA studied in this work. It is a RISC based ISA formulated by Sun Microsystems in 1985 [42]. SPARC architecture supports 32-bit integer registers and a 32, 64 and 128 bit IEEE standard 754 floating point data types. All SPARC instructions are 32-bit in length. Generally speaking, any SPARC compliant processor have an integer unit (IU), floating point unit (FPU) and an optional co-processor (CP) [42]. The instruction format, category, encoding and register window of SPARC ISA are discussed in this chapter.

## 3.1 Instruction Format and Instruction Categories

### 3.1.1 Instruction formats

SPARC v8 have 72 basic instructions. These instructions are 32-bit in length and they are encoded in three main formats:

1. CALL (Function call instruction)
2. SETHI and branch instructions
3. Memory, arithmetic and remaining instructions.

Figure 3.1 shows the three instruction encoding formats and the different fields of the three instruction formats. In Figure 3.1 the blue *OP* (encoding format identifier) field is used to identify one of the three encoding formats. The light blue *RD*, *rs1* and *rs2* fields are the destination and source register identifiers. The displacement and immediate fields are used to store immediate constant values of an instruction. *OP2* and *OP3* are opcode fields that specify the operation type of the instruction. The *i* field indicates if

the instruction is register-register ($i$=0) or register-immediate ($i$=1). Finally, the *Cond* field specifies the condition code and the 'a' field indicates if a branch is taken or not taken.

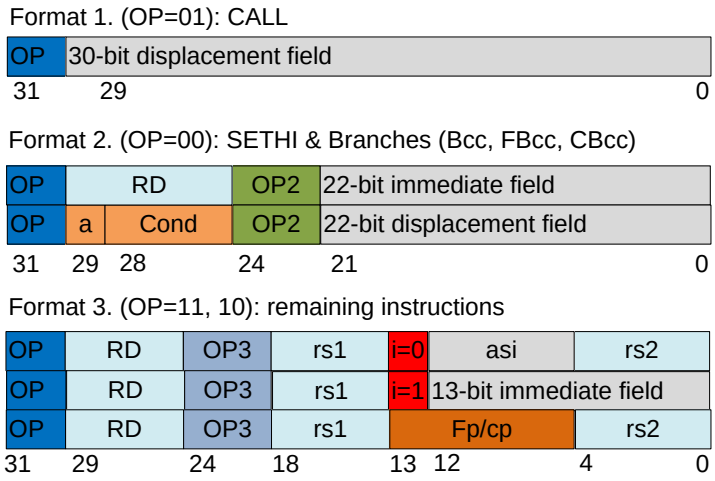



Figure 3.1: SPARC v8 instruction formats

The interpretation of the different fields of the instruction formats is given in Table 3.1 and Table 3.2. Table 3.1 shows the various encoding options of the OP field and their descriptions. Similarly, the explanation of the other fields are presented in Table 3.2.

Table 3.1: *OP* field encoding

| Format | OP value | Instructions |
|---|---|---|
| 1 | 01 | CALL |
| 2 | 00 | Bicc, FBfcc, CBccc, SETHI |
| 3 | 11 | Memory instructions |
| 3 | 10 | arithmetic, logical, shift, and remaining |

### 3.1.2 Instruction Categories

Based on their operation type SPARC instructions are classified into 6 main instruction categories. These categories are:

1. Memory (Load/ Store) instructions: This group consists of all instructions that can read/ write data from/to the memory. No other instruction can access the memory except the instructions in this category.

2. Integer arithmetic (arithmetic and logical shift) instructions: Instructions that perform arithmetic and logical shift operations fall in this category. These instructions perform an arithmetic/ logical operation on the operands specified by register

Table 3.2: Instruction field and their interpretation

| Instruction Field | Interpretation |
|---|---|
| 3-bit op2 | Specifies the type of the instructions in format 2 |
| 6-bit OP3 | encodes all the instructions in format 3 |
| 1-bit a | branch instruction annuls if a branch is taken or not |
| 4-bit Cond | condition codes for branch instructions |
| 22 and 30 bit displacement | sign extended displacements for a call and a branch instructions |
| 8-bit asi | address space identifier for load/store instructions |
| 9-bit fp/op | encodes floating point or co-processor instructions |
| 13-bit immediate | second ALU input operand if i=1 |
| 5-bit RD | destination register |
| 5-bit rs1 | register suorce one |
| 5-bit rs2 | register source two |
| 1-bit i | second input operand identifier |

source one (*rs1*) field and register source two (*rs2*) field if *i* field is zero or the 13-bit immediate field if *i*='1'. Then the result is stored in the destination register (*rd*). Most of the instructions in this category exist in a dual version. The two versions are identified by the condition code (*cc*) suffix. If an instruction have a cc suffix (for example ADDcc) then it performs its arithmetic operation first and then it modifies the integer condition code (cc) of the processor. On the other hand, the non suffixed version performs the arithmetic operation only.

3. Control transfer instructions: These are instructions that can change the value of the program counter and can affect the order of program execution. Conditional branch, unconditional jump, trap, function call and return instructions are examples of the instructions in this category.

4. Floating-Point instructions: These instructions are floating point equivalent of the integer arithmetic instructions.

5. Co-processor instructions: These instructions are instructions to be executed by the attached co-processor (if available). This instructions are dependent on the implementation of the co-processor and they are preceded by the *cp* prefix.

6. Read/Write State Registers: These instructions are special purpose instructions that can access the state and status register of the processor.

## 3.2 Sparc v8 Instruction Encoding

Having the instruction formats and their categories discussed in the previous section, the binary encoding of those instructions, their register usage convention and encoding of the immediate fields are discussed in this section.

### 3.2.1 Opcode Field Encoding

The 6-bit *OP3*, field of Figure 3.1 represents the encoding of the operation type (opcode for short) of an integer arithmetic and load/store instructions. For simplicity in this section the encoding of the most common SPARC instructions will be discussed and few encoding examples of selected SPARC instructions will be provided. However, an encoding of all SPARC instructions is presented in [42].

***Load/store instructions***

In SPARC ISA the load/store instructions are identified by the value '11' in their *OP* field. Table 3.3 shows the opcode (OP3) field encoding of some representative load/store instructions.

Table 3.3: opcode encoding of selected load/store instructions

| OP | Opcode | Binary-encoding (OP3) | Interpretation |
|---|---|---|---|
| integer load/store | | | |
| 11 | LD | 000000 | Load word |
| 11 | LDD | 000011 | Load double word |
| 11 | ST | 000100 | Store word |
| 11 | STD | 000111 | Store double word |
| floating point load/store | | | |
| 11 | LDF | 100000 | Load word |
| 11 | LDDF | 100011 | Load double word |
| 11 | STF | 100100 | Store word |
| 11 | STDF | 100111 | Store double word |
| co-processor load/store | | | |
| 11 | LDC | 110000 | Load word |
| 11 | LDDC | 110011 | Load double word |
| 11 | STC | 110100 | Store word |
| 11 | STDC | 110111 | Store double word |

***Arithmetic and logical shift instructions***

Arithmetic and logical shift instructions are identified by the value of '10' in their *OP* field. In a similar fashion to the load/store instructions, the encoding of common arithmetic and logical shift instructions is given in Table 3.4.

### 3.2.2 OP2 Field Encoding

The 3-bit *OP2* field identifies SETHI and branch instructions such as Bicc, CBcc and FBcc. These instructions have a specific purpose. SETHI instruction is used to place a constant value in to the high-order 22-bit of the register specified by the destination register (rd) field of the instruction. Most of the time SETHI instruction is used just before load/store instruction to store the source or destination memory address in a

Table 3.4: opcode encoding of selected arithmetic/logical instructions

| OP | OPcode | Binary-encoding (OP3) | Interpretation |
|---|---|---|---|
| Arithmetic | | | |
| 10 | ADD | 000000 | Addition |
| 10 | SUB | 000100 | Subtraction |
| 10 | UMUL | 001010 | Unsigned multiply |
| 10 | UDIV | 001110 | Unsigned division |
| Logical shift | | | |
| 10 | AND | 000001 | Logical AND |
| 10 | OR | 000010 | Logical OR |
| 10 | SLL | 100101 | Logical shift left |
| 10 | SRL | 100110 | Logical shift right |

register. Additionally, SETHI instruction can be used before a function call instruction which needs larger value to specify the memory address of the called function. NOP is a variant of SETHI instruction in which the *rd* and 22-bit immediate fields are all zero. The remaining instructions in this category are used to handle branches and increment the processor condition code (cc). Majority of the encoding options of *OP2* represents unimplemented instruction. Table 3.5 shows the values of *OP2* field and the instructions identified by the specific *OP2* field value.

Table 3.5: OP2 field encoding and their instructions

| OP | OP2 | instruction |
|---|---|---|
| 0 | 2 | Bicc |
| 0 | 4 | SETHI |
| 0 | 6 | FBcc |
| 0 | 7 | CBccc |
| 0 | 0,1,3,5 | Unimplemented |

### 3.2.3 Register Window and Register Encoding

#### 3.2.3.1 SPARC Register Window

A SPARC compliant processor can have up to 40-520 general-purpose integer registers, 32 floating point registers, an optional co-processor and up to 12 control registers. Out of the available general-purpose registers only 32 registers are visible to the programmer at any time. These programmer visible registers are grouped in to four different categories; *global (g), local (l), input (i)* and *output (o)* registers in which each group is consisting of 8 registers. In SPARC convention all register names start with a % sign.

To provide a large number of registers, a SPARC implementation uses a sliding register window scheme in which one register window will have 24 registers (*input*, *local* and *output* registers). For any procedure in a user program there will be only one active

window consisting 8 *input* registers to store incoming parameters, 8 *local* registers to store local variables and 8 *output* registers to store an outgoing parameters during procedure calls. The remaining 8 global registers are available for any procedure in a program and does not belong to any window. For data transfer between procedures during procedure call the register windows overlap partially. Hence, the *output* registers of the caller procedure will be *input* registers of the callee and when the callee returns the window slides back. The sliding window process is managed by SAVE and RESTORE instructions.

The process of sliding register window is illustrated in Figure 3.2. The green boxes in the figure shows the shared *input* and *output* registers of the register windows of procedures 1-4. As shown in the figure procedure 1 calls procedure 2 and the calling process continues in the same sequence until procedure 4 is called. During procedure return, procedure 4 will be the first procedure to return and puts its return value in the shared green register window in order to be accessed by the caller procedure (procedure 3) and the process continues until all procedures are executed. The primary advantage of the sliding window scheme is to increase performance by increasing the number of physical registers which in turn decreases the access of the stack during program execution. As shown in Figure 3.2 the caller stores its outgoing parameters in its *output* registers then the callee will get the parameters on its *input* registers. When the callee finishes execution it will store its return value in one of its *input* registers. However, this overlapping register window scheme forces the programmer to save the state of any *input/output* registers to the stack by using the SAVE instruction whenever the programmer needs to use them. Then after usage the programmer should restore the state of the registers by using RESTORE instruction. This saving and restoring of registers can induce potential performance overhead.

Among the 32 programmer visible registers there are few special purpose registers that are used for a specific purpose and the programmer can not use any of these registers. These special purpose registers includes: $g0$ (zero register) which is hardwired to be zero, $i6$ (frame pointer), $o6$ (stack pointer) and $i7$ (return address).

#### 3.2.3.2 Register encoding of a SPARC ISA

SPARC architecture uses 5-bits to encode the 32 programmer visible registers. The SPARC encoding of these registers is straightforward and follows the conventional naming of the registers. For example, the encoding of *r0/g0* is "00000" and encoding of *r31/i7* is "11111". The register window that is currently in use is determined by the Current Window Pointer (CWP). Table 3.6 shows the conventional register name, their groups, and encoding of the active 32 registers.

### 3.2.4 NOP and Immediate Fields Encoding

In modern microprocessors NOP is a commonly used instruction to stall few clock cycles without doing any useful task except incrementing the program counter. NOP is used during hazards (data and structural dependencies) and control dependencies (branch

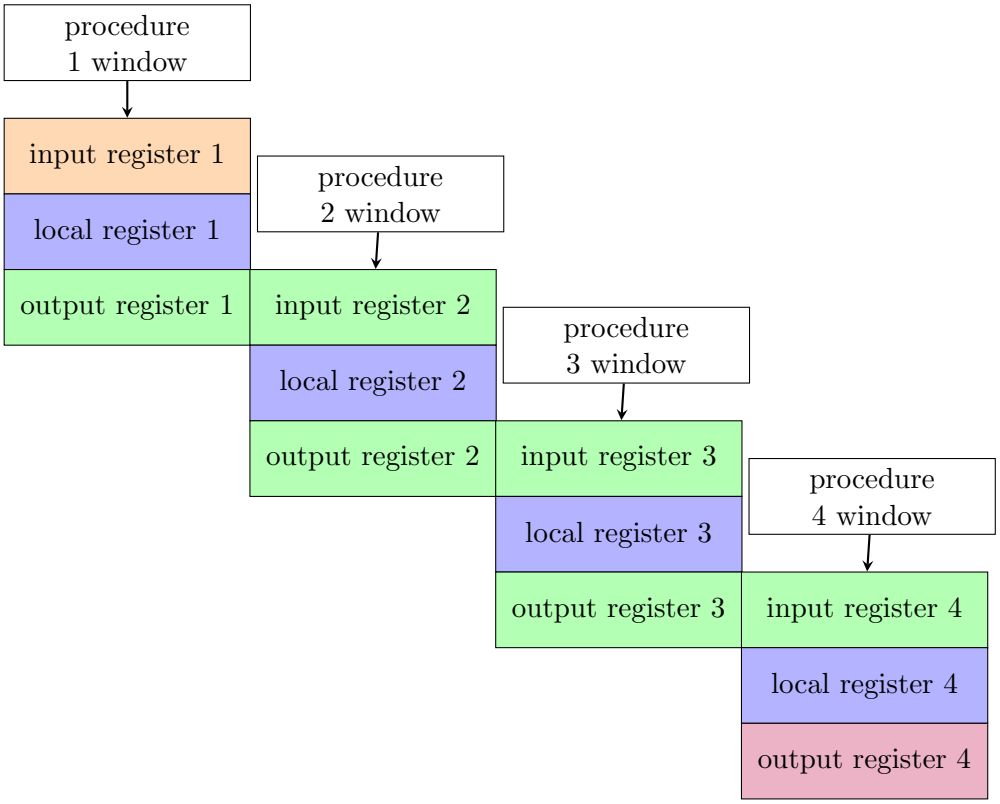



Figure 3.2: SPARC v8 sliding register window

Table 3.6: Register encoding

| Registers | SPARC group | Encoding |
|---|---|---|
| $r0$ - $r7$ | $g0$ - $g7$ | 00000-00111 |
| $r8$ - $r15$ | $o0$ - $o7$ | 01000-01111 |
| $r16$ - $r23$ | $l0$ - $l7$ | 10000-10111 |
| $r24$ - $r31$ | $i0$ - $i7$ | 11000-11111 |

miss predictions). Generally speaking, NOP instruction is used when there is a branch instruction or to solve any potential hazards and fill any delay slots during program execution. In a SPARC architecture NOP instruction is encoded as SETHI($OP$='00' and $OP2$='100') which sets the higher 22 bits of register $r0$ ($g0$) to the value of immediate 22 fields of the instruction. However, since register $r0$ is hardwired to be '0' all the time its value will never be changed. The encoding of NOP instruction is shown below:

| OP | RD | OP2 | immediate 22 |
|---|---|---|---|
| 00 | 00000 | 100 | 0000000000000000000000 |

Immediate fields are fields of an instruction in which their value is determined at run time. the value of an immediate field can be either a data (for arithmetic operation) or an address (for memory, call and branch instructions) that specify the memory location.

Hence, unlike opcode or register fields the value of immediate field is determined by the application for this reason they does not have a specific encoding by the ISA. However, for arithmetic and load/store instructions, when the *i*-field is '0', which means the operands are *rs1* and *rs2* then the 8-bit *asi*-field shown in Figure 3.1 is encoded to be '0' all the time. In Chapter 4 we will discuss how to take advantage of this encoding scheme to generate instruction cache aging-aware encoding.

# Aging-aware Instruction Set Encoding (AISE)

# 4

In this chapter we will discuss the implementation of the proposed aging-aware instruction encoding technique. First the main idea of aging-aware instruction encoding is discussed followed by an analysis of the related works and improvement opportunities of the existing SPARC encoding. Then the working principle of the proposed instruction cache aging-aware encoding algorithm is presented. The limitations and improvement opportunities of the proposed technique are discussed. Finally the chapter is concluded by discussing the experimental setup and simulation results of the new encoding technique followed by a brief comparison of the improvement alternatives of the aging-aware encoding technique.

## 4.1 Concepts of Aging-aware Instruction Encoding

The main idea of the aging-aware instruction encoding technique is to change the encoding of the instructions of a specific ISA, SPARC ISA in our case, so that the impact of BTI on the instruction cache will be minimized. To achieve this goal first the baseline encoding is analyzed and we develop a simulation based application that exhaustively search for an encoding with less BTI impact on the instruction cache from all possible encoding alternatives. The effectiveness of the new encoding is evaluated for different benchmark applications. The main challenge of the aging-aware instruction encoding technique is that an encoding which gives less BTI impact on the instruction cache evaluated for one benchmark application can impose a relatively higher BTI impact when evaluated using other benchmark application. To address this issue we set a reasonable target reliability margin for all of the benchmark applications and we search for an encoding that satisfy the reliability margin across all benchmark applications.

## 4.2 Related Works

Various instruction encoding schemes have been widely studied for low power designs in order to minimize the switching activity, which in turn reduces the dynamic power consumption of embedded microprocessors. Authors in [43] proposed a low power instruction encoding technique which finds an encoding that minimizes the number of bits that change their value over continuous sequence of instruction. This approach reduces the dynamic power by minimizing the switching activity, however, this minimizing the flipping of the bit positions eventually leads to an accelerated BTI impact on the instruction cache. In [44] instruction encoding and compression technique is discussed to minimize power consumption of an instruction cache. In this work instructions are stored in a compressed form to minimize the power consumption of the cache.

Moreover, they employ an encoding technique that minimizes the switching activity of the instruction cache. However, similar to [44] in this work the instruction cache will suffer from BTI induced failures. Moreover this work requires an additional decoder to decompress the compressed instructions which will add a potential area overhead.

Authors in [45] proposed an aging-aware instruction encoding technique which generates a new encoding for the opcode field of an instruction to addresses aging of the decoder unit of a processor. However, their technique does not address all fields of an instruction. Besides, since they are targeting the decoder unit, their encoding of the opcode field can potentially worsen the impact of BTI on the instruction cache. In comparison to the techniques discussed above our aging-aware encoding technique addresses aging of the instruction cache by addressing all the fields of an instruction.

## 4.3 Improvement Opportunities of SPARC Encoding

In the previous chapter we have discussed the instruction encoding, register usage and naming convention of the SPARC architecture. The SPARC encoding was designed to optimize compilers, simplify the implementation of processor pipeline stages and to decrease execution time of applications [42]. Hence, SPARC implementation does not take into account the impact of the encoding of the instructions on reliability of the pipeline stages and instruction cache. However, from BTI induced aging of instruction cache point of view we can find different improvement spots of the existing encoding and we can use those improvement spots to generate a new encoding that imposes less BTI impact on the instruction cache. The opcode, register and constant fields of the existing encoding are the most encoding dependent fields of an instruction. Moreover, these fields have a relatively large encoding alternatives. Hence, we can explore the encoding alternatives of these fields to find an aging optimal encoding. OP and Cond fields of a SPARC instruction, however, have a small size which makes them to have quite limited encoding alternatives. In this section we will discuss how we can use the improvement opportunities of the baseline encoding to generate an aging-aware encoding.

### 4.3.1 Opcode Field Encoding Analysis

Though a SPARC compliant processors have to satisfy the basic requirement of the SPARC ISA, depending on the processor's hardware architecture all the SPARC instructions may not be equally applicable for all SPARC compliant processors. For example, the encoding of the floating point instructions are less relevant for a microprocessor which does not have a floating point hardware. For such kind of processor the compiler will use software emulation technique to generate an equivalent integer instructions. Moreover, the SPARC co-processor instructions are not applicable for a processor that does not have any co-processor attached to it. Again, for such kind of processors the co-processor instructions can be removed and the search space of an aging-aware encoding can be increased.

Let us revisit the load/store instruction encoding Table 3.3 (repeated here in Table 4.1 for convenience). As we can see from Table 4.1 the regular integer instructions are

Table 4.1: opcode encoding of representative load/store instructions

| OP | OPcode | Binary-encoding (OP3) | Interpretation |
|---|---|---|---|
| integer load/store | | | |
| 11 | LD | 000000 | Load word |
| 11 | LDUH | 000010 | Load unsigned half word |
| 11 | LDD | 000011 | Load double word |
| 11 | ST | 000100 | Store word |
| 11 | STD | 000111 | Store double word |
| floating point load/store | | | |
| 11 | LDF | 100000 | Load word |
| 11 | LDDF | 100011 | Load double word |
| 11 | STF | 100100 | Store word |
| 11 | STDF | 100111 | Store double word |
| 11 | STFSR | 100101 | Store FP State Register |
| co-processor load/store | | | |
| 11 | LDC | 110000 | Load word |
| 11 | LDDC | 110011 | Load double word |
| 11 | STC | 110100 | Store word |
| 11 | STDC | 110111 | Store double word |
| 11 | STCSR | 110101 | Store CP State Register |

encoded using lower order numbers (more number of zeros) while the equivalent floating point and co-processor instructions are encoded with less number of zeros. Various studies showed that majority of the instructions in different applications such as MiBench [46], an embedded benchmark suit, are integer related operations. Hence, for an integer intensive applications such as MiBench, the existing encoding scheme will result in a higher BTI impact on the opcode field of an instruction cache. Hence, to decrease this impact we can shuffle the opcode field encoding of the integer, floating point and co-processor instructions so that the BTI induced failure of the opcode field of an instruction cache will be minimized.

Our target microprocessor, Leon2, does not have any co-processor and floating point hardware. It uses software emulation technique to execute any floating point related operations. Software emulation technique is a compiler activity of generating one or more integer equivalent instructions for a floating point instruction. For the compiler to perform software emulation the programmer have to enable the -msoftfloat compiler flag. Then the compiler will do the task of replacing floating point instructions with an equivalent integer instructions during compile time. For such microprocessor with no floating point hardware and no attached co-processor, we can replace the encoding of the integer instructions by the encoding of co-processor or floating point instructions so that the signal probability of the bit positions of the opcode filed of an instruction cache

will be balanced.

### 4.3.2 Register Naming Analysis

In the previous chapter we have seen that SPARC register encoding follows the pattern of the register names (from lower to higher) where *r0/g0* is encoded to be '00000' and *r31/i7* is '11111'. However, since the register usage in an application is quite non-uniform this encoding scheme results in a worst BTI induced aging of the register fields of the instruction cache. To prove this scenario, we have studied the register usage of various applications by generating their assembly code and we found that registers *r0* through *r3*(*g0-g3*) are used most of the time. Hence, the *rd*, *rs1* and *rs2* fields of an instruction cache will be under BTI stress most of the time. To minimize the BTI impact on the register fields of an instruction cache there are two options:

1. **Register renaming:**
   This is a process of changing the register usage in a sequence of instructions. This technique basically aims to achieve a performance improvement of a pipelined processor by reducing the data dependency between sequence of instructions. Besides, this technique can be used to mitigate the impact of BTI on the register fields of an instruction cache. However, it needs to be done manually by inspecting the assembly code of an application. Additionally, the manual inspection is highly application dependent and a time consuming task. Since most of the modern microprocessors have large number of physical registers than the registers that are visible to the programmer, register renaming can be done by using register renaming hardware in the pipeline to achieve instruction level parallelism. However, register renaming at the instruction pipeline does not address the impact of BTI on instruction cache.

2. **Changing the encoding:**
   This approach attempts to change the binary encoding of the registers at the decoder unit of the processor and the assembler unit of the compiler toolset. For example, if the registers *r0* and *r1* encoded as '00000' and '00001' respectively, are intensively used in an instruction sequence then the register fields of the instruction cache will be under BTI stress most of the time. Hence, this technique tries to change the encoding of the *r0* and *r1* registers to have opposite values in most of their bit positions as can as possible. For example, the encoding '00000' for *r0* and '11111' for *r1* will result in a less BTI stress of the register fields of the cache. Unlike register renaming this technique does not have performance improvement, however, its ease of implementation with no area and performance penalty makes it preferable. Hence, we will discusses this approach in detail in Section 4.4.

### 4.3.3 Immediate and Constant Fields Analysis

Immediate fields are fields of an instruction in which their value is determined at run time and they do not have a specific encoding. This makes it difficult to address BTI impact by applying the proposed technique. The constant fields on the other hand, are

fields that are hard coded by the encoding to be zero all the time without depending on the nature of the application. The encoding of a constant field can be changed to achieve the targeted reliability margin. NOP instruction and register-register instructions (where i-field='0') are instructions that employ constant fields. For NOP instruction the 22 bit positions are hard coded to be zero all the time. However, to mitigate BTI induced aging of an instruction cache we can modify the encoding of NOP instructions to make the 22 bit fields all 1's. In a similar fashion the *asi* field of a register-register instruction where the two operands are *rs1* and *rs2* can be encoded to be all 1 instead of all 0.

## 4.4 Implementation of Aging-aware Instruction Encoding

Before we discuss the implementation of instruction cache aging-aware encoding technique let us refresh our mind on the flow of program execution. To explain the execution flow of a user program in a computer system we will elaborate the gcc compiler toolset. In the program execution flow first an include-files and macros of a C/C++ source program will be preprocessed using the preprocessing tool. Then the compiler will take the output of the preprocessor and the C/C++ source files to generate an assembly code. After that the assembler will take the assembly code and generate the equivalent binary code and store the result as an object file. Finally, the linker will take different object files to generate the final executable file. Figure 4.1 shows the compilation flow

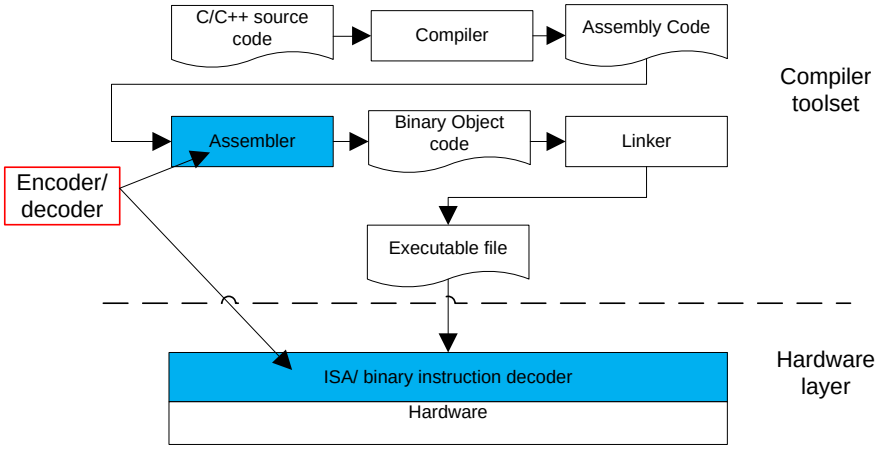



Figure 4.1: Compilation steps and gcc toolset

and gcc compiler toolset along with the underlaying ISA and hardware layer. The assembler generates the binary encoding of instructions and then the binary instructions are interpreted in the decoder unit of the processor according to the implemented ISA.

To generate an aging-aware encoding both the assembler (from the compiler toolset) and the decoder unit that implements the ISA (from the hardware layer of Figure 4.1) should be modified so that the new encoding will function properly. To achieve this goal we propose an aging-aware instruction encoding technique (AISE) that imposes no or

considerably less area overhead in the decoder unit. The main idea behind AISE is to change the existing binary encoding of the SPARC instructions so that the duty cycle of an instruction cache will be balanced (close to 0.5).

AISE searches for an aging-aware encoding by exhaustively checking all possible non redundant encoding options. Then the algorithm selects an encoding that imposes less BTI impact on the instruction cache.

### 4.4.1 Aging-aware Encoding Generation

The proposed technique tries to efficiently utilize all the possible improvement opportunities of the baseline encoding. Hence, to generate an aging-aware encoding first the default encoding (SPARC encoding) is stored in a lookup table (LUT) for further usage by the algorithm. The instructions written to the instruction cache are extracted to determine the aging impact of the default encoding. Once the instruction trace is available the baseline signal probability is calculated. After that, the algorithm will iteratively check for all possible encoding options from the lookup table by shuffling the encoding of different instructions to obtain a better encoding. This general flow is depicted in Figure 4.2.

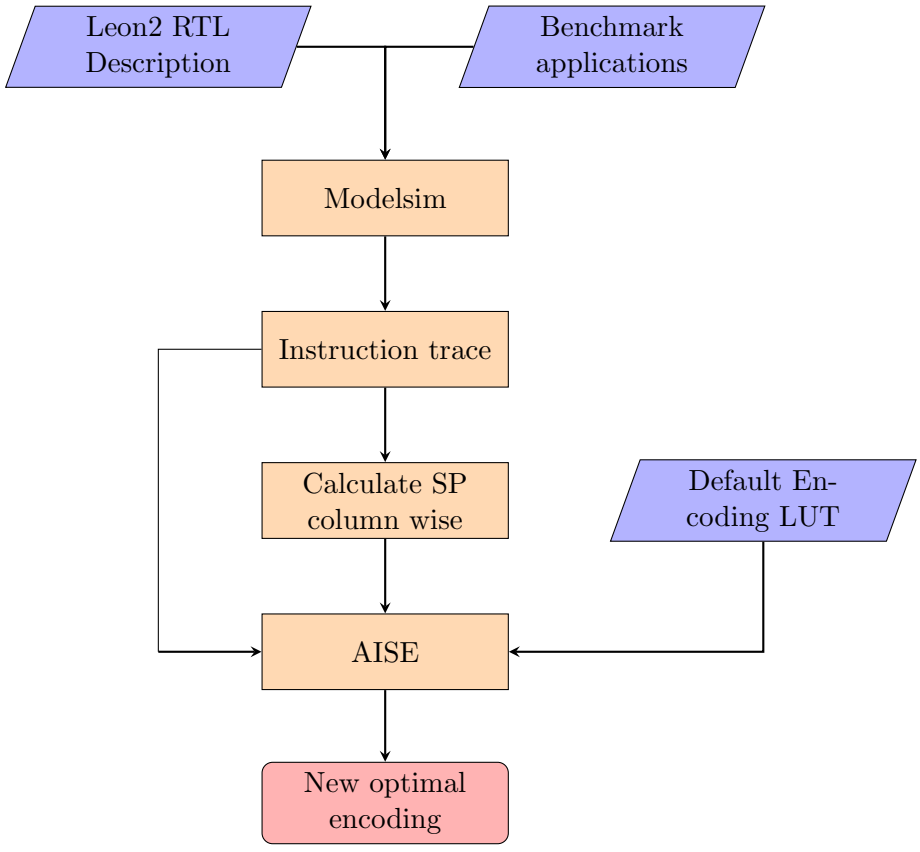



Figure 4.2: General flow of aging-aware encoding generator

AISE follows divide and conquer approach, in which the instruction encoding is divided into three module:

- Opcode field encoding: This module focuses on finding an optimal encoding for the opcode field of an instruction.
- Register field encoding: This module finds an optimal encoding of the source and destination register fields.
- immediate and constant field encoding: This module of the algorithm addresses the encoding of the hard codded constant fields such as the NOP instruction and *asi* field of a register-register operation instruction.

#### 4.4.1.1 Module 1: Opcode Field Encoding

Since majority of the SPARC instructions are arithmetic and load/store instructions the opcode field encoding module of AISE focuses on the encoding of the opcode (*OP3*) field of a load/store and arithmetic instruction categories. The algorithm generates a new encoding from the lookup table by incrementing the index of the encoding of the opcode field of the instructions.

**Algorithm 1** Opcode field encoding

1: *Use baseline encoding from LUT as input*
2: **for** *i=0 to N* **do**
3: *Select new encoding from LUT by incrementing the index*
4: **for** *j=0 to 6* **do**
5: *calculate* $SP_j$ *for bit j*
6: $Variance_j := |SP_j - 0.5|$
7: **End For**
8: $worstcase_i := \max_{0 \le j \le 6} Variance_j$
9: **End For**
10: *optimal encoding* $= \min_{0 \le i \le N} wostcase_i$

Using the new encoding the signal probability of each bit position will be calculated. Then for each encoding the variance of the signal probability (variation from 0.5) of each bit position will be calculated and saved in an indexed array (indexed by the iteration number). This process will continue until all possible encoding options are analyzed. Table 4.2 shows the process of selecting a new encoding from the default encoding lookup table. Algorithm 1 shows the process of selecting an optimal encoding for the opcode field used by AISE. As shown in Algorithm 1 on each iteration all opcode will have a new encoding and a new signal probability according to the new encoding. Then the variance of each opcode bit position will be calculated again and stored in an array indexed by the iteration number. Finally the opcode field encoding module selects an opcode field

Table 4.2: Opcode encoding of generation

| Opcode | encoding at different iterations | | | | | |
|---|---|---|---|---|---|---|
| | i=0 | i=1 | i=2 | i=3 | i=4 | i=5 |
| LD | 000000 | 000100 | 100011 | 100111 | 110011 | 110111 |
| ST | 000100 | 100011 | 100111 | 110011 | 110111 | 000000 |
| LDDF | 100011 | 100111 | 110011 | 110111 | 000000 | 000100 |
| STDF | 100111 | 110011 | 110111 | 000000 | 000100 | 100011 |
| LDDC | 110011 | 110111 | 000000 | 000100 | 100011 | 100111 |
| STDC | 110111 | 000000 | 000100 | 100011 | 100111 | 110011 |

encoding that have smaller signal probability variance in majority of the bits positions of the opcode field across all the selected benchmark applications (Equation 4.1).

$$encoding := \min_{0 \leq i \leq N} Variance_i \tag{4.1}$$

Where N is the search space (size of the lookup table) and Variance$_i = |0.5 - SP_i|$.

#### 4.4.1.2 Module 2: Register Fields Encoding

Register field encoding uses similar technique as Algorithm 1. However, register encoding differs from the opcode field encoding in a way that it encodes three different fields (*rd*, *rs1* and *rs2*) of an instruction. To achieve this objective the algorithm is designed to have one loop for each field that performs the task of Algorithm 1. Register field encoding module uses different offset for each register field to generate a new encoding from the LUT. Algorithm 2 shows the flow of register field encoding. The algorithm returns a single register encoding that have less BTI impact across all register fields of the instruction cache. There could exist a situation that an encoding can result in less BTI impact in one of the register fields but worst BTI impact on the other fields. However, such an encoding will not be considered and an encoding that imposes less aging impact across all the register fields will be selected.

#### 4.4.1.3 Module 3: Immediate and Constant Field Encoding

This module of AISE focuses on the encoding of NOP instruction and *asi* field of register-register instructions by changing the static encoding of all '0' to be '1' or hybrid of '0' and '1'. Figure 4.3 shows the general flow of AISE and the concurrency of the three modules of the algorithm.

### 4.4.2 Limitations and Improvements of AISE

AISE does not address the encoding of the 2-bit *OP* field, 1-bit *i* and *a* fields and the 3-bit *Cond* field of an instruction. The main reason for the algorithm to focus only on opcode, register and constant fields but not addressing the other fields of a SPARC instruction is due to the fact that the other fields have a small bit length which narrows the search space of the encoding options. From our experiment we noticed

**Algorithm 2** Opcode field encoding

1: *Use baseline encoding from LUT as input*
2: *offsetrd=0; offsetrs1=1;offsetrs2=2;*
3: **for** *i=0 to N* **do**
4: *Select three offseted new encoding from LUT by adding offsets to loop index*
5: **for** *j=0 to 5* **do** *//rd encoding*
6: *calculate* $rd - SP_j$ *for bit j*
7: $\textit{rd-Variance}_j := |SP_j - 0.5|$
8: **End For**
9: **for** *k=0 to 5* **do** *//rs1 encoding*
10: *calculate* $rs1 - SP_k$ *for bit k*
11: $\textit{rs1-Variance}_k := |SP_k - 0.5|$
12: **End For**
13: **for** *l=0 to 5* **do** *//rs2 encoding*
14: *calculate* $rs2 - SP_l$ *for bit l*
15: $\textit{rs2-Variance}_l := |SP_l - 0.5|$
16: **End For**
17: $\textit{rd-worstcase}_i := max\ \textit{rd-Variance}_j$
18: $\textit{rs1-worstcase}_i := max\ \textit{rs1-Variance}_k$
19: $\textit{rs2-worstcase}_i := max\ \textit{rs2-Variance}_l$
20: **End For**
21: *optimal encoding* $= \min_{0 \le i \le N} wostcase_i$

that the encoding of the three fields (*i*,*a* and *cond*) have less contribution on aging due to the fact that arithmetic and load/store instructions are dominant in an application domain. However, the encoding of *OP* field have a very crucial impact since all SPARC instructions have an *OP* field.

In order to mitigate the BTI impact of the baseline encoding of the *OP* field, AISE can be extended by using one of the following techniques:

1. Applying the same encoding technique as opcode (Op3) or register fields.
2. Implementing a bit flipping technique which periodically flips the value of *OP* field in an instruction cache.

This two options have their own advantages and disadvantages. The advantage of using the same technique as opcode field encoding is there will be no area overhead added. However, in addition to the small encoding option (four options using 2-bit) the nature of the application also determines the effectiveness of this method. In the bit flipping technique the data written to the *OP* field of an instruction cache will be inverted after a certain time. Bit flipping technique requires an additional flip circuitry which includes a counter to determine when to flip, an additional bit that indicates if the data is inverted

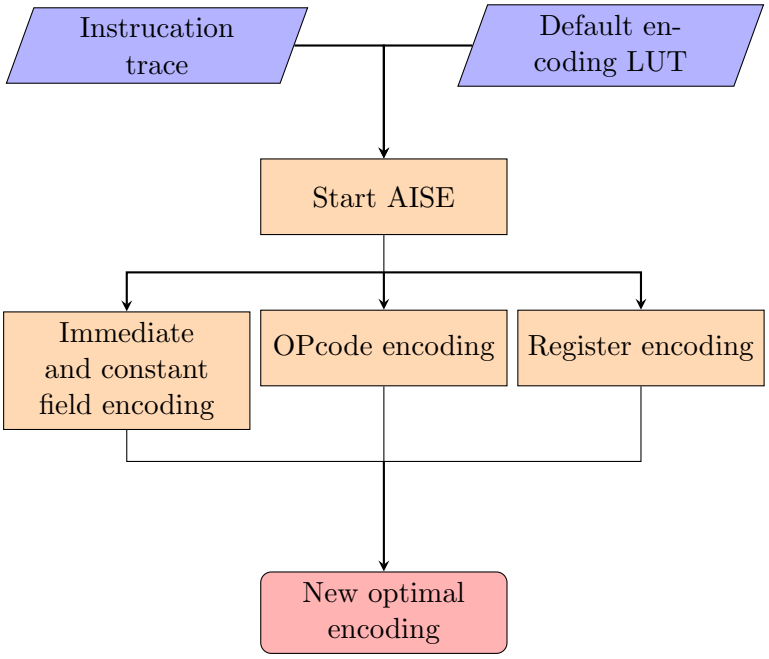



Figure 4.3: General flow of AISE

or not and four XOR gates to perform the inversion process at the write and read port of an instruction cache. In the cost of a small area overhead using bit flipping approach is more effective in balancing the signal probability of the *OP* field of the instruction cache than the first option (changing the encoding of the OP field). The area overhead induced by the bit flipping technique can be further optimized by using a self-controlled bit flipping technique which will be presented in Chapter 5.

## 4.5 Experimental Results

In this section the experimental setup and simulation environment of the project is presented first. After that, the new aging-aware encoding generated by using AISE is presented and sample opcode field encoding along with register encoding is discussed. Finally, the achieved SNM degradation improvement of the new encoding is discussed for various technology nodes. This section also address the impact of scaling and temperature on accelerating BTI induced aging along with the simulation results for a different technology nodes and operating temperature.

### 4.5.1 Experimental Setup

In this section we discuss the Hardware and software environments used in the project. In order to show the efficiency of the project, we apply this technique to the Leon2 processor model. Figure 4.4 shows the Leon2 processor model which is a 32-bit in-order processor that implements a SPARC hardware architecture which have, an integer unit with 5-stage pipeline, multiply, divide and MAC units and separate 32-bit width

instruction and data cache. Leon2 has a register-file with 256 registers and a 25-bit width data and instruction tag caches. To trace the data written to different memory

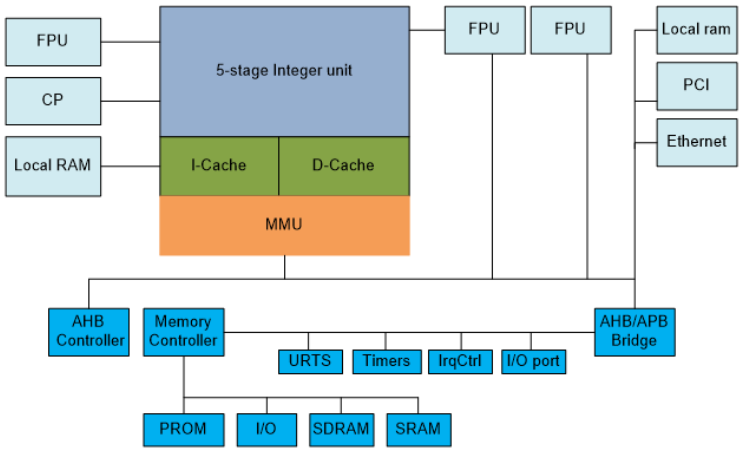



Figure 4.4: Leon2 processor model

structures of the Leon2 processor model we first modify the register transfer level (RTL) description the model in a way that it dumps the data and the clock period of the write operation to a trace file whenever the write enable signal is enabled. The modifications made to the Leon2 model is given in Appendix A.

Once the trace is available we use the iterative AISE algorithm to find an optimal encoding first. After that the signal probability of the instruction cache is calculated to evaluate the aging impact of the new encoding. To generate an aging-aware encoding we use a C++ tool that implements aging-aware encoding technique. We evaluated our technique using benchmark applications from MiBench [46].

### 4.5.2 Aging-aware Encoding

As discussed in Section 4.3 AISE generates a new opcode and register encoding. The main motivation to focus on this parts of the ISA includes:

1. Majority of the SPARC instructions are format 3 (arithmetic or load/store).

2. Majority of the fields in those instructions are always encoding dependent which makes them elegant from encoding point of view.

3. The search space and encoding alternative of the opcode field of arithmetic and load/store instructions is much wider than the other instructions.

4. Registers are used by most of the SPARC instructions and they can occupy up to 50% of the bit positions of an instruction.

#### 4.5.2.1 Opcode Field Encoding

AISE generates a new encoding for each benchmark application. However, the aging impact of an encoding generated for one application could worsen the aging impact when evaluated for different application. Hence, AISE is modified to store all possible

Table 4.3: Arithmetic opcode encoding

| Arithmetic operation | | | | | |
|---|---|---|---|---|---|
| Opcode | Baseline | Aging-aware | Opcode | Baseline | Aging-aware |
| IADD | 000000 | 111101 | IAND | 000001 | 111110 |
| IOR | 000010 | 111111 | IXOR | 000011 | 000000 |
| ORN | 000110 | 000011 | IXNOR | 000111 | 000100 |
| ADDX | 001000 | 000101 | UMUL | 001010 | 000111 |
| SMUL | 001011 | 001000 | SUBX | 001100 | 001001 |
| UDIV | 001110 | 001011 | SDIV | 001111 | 001100 |
| ADDCC | 010000 | 001101 | ANDCC | 010001 | 001110 |
| ORCC | 010010 | 001111 | XORCC | 010011 | 010000 |
| SUBCC | 010100 | 010001 | ANDNCC | 010101 | 010010 |
| ADDXCC | 011000 | 010101 | UMULCC | 011010 | 010111 |
| SMULCC | 011011 | 011000 | UDIVCC | 011110 | 011011 |
| SDIVCC | 011111 | 011100 | ISLL | 100101 | 100010 |
| ISRL | 100110 | 100011 | JMPL | 111000 | 110101 |
| SAVE | 111100 | 111001 | RESTORE | 111101 | 111010 |

encoding and their signal probability for different benchmark applications. Then we conduct an analysis on the signal probabilities of the different encoding alternatives to select an encoding that have less aging impact across all benchmark applications. However, for an application specific microprocessors, where the target applications are predetermined, an encoding generated by AISE can give a promising result to mitigate BTI induced aging of an instruction cache.

Based on the nature of the selected benchmark applications, the aging-aware instruction encoding algorithm generates a new opcode field and register encoding. Few examples of the newly generated opcode field encoding of arithmetic and load/store instructions are presented in Table 4.3 and 4.4.

#### 4.5.2.2 Register Encoding

To generate a new register encoding the register usage frequency of various applications is analyzed first. We use the Leon2 disassembler, which is a tool that is used to disassemble the binary instruction to an assembly code. The assembly code of the applications reveals that there is a non-uniform register usage frequency in which the global registers are used most of the time. This uneven register usage frequency leads to an accelerated BTI induced aging of the various register field entries of the instruction cache.

Table 4.4: Memory opcode encoding

| Memory operation | | | | | |
|---|---|---|---|---|---|
| Opcode | Baseline | Aging-aware | Opcode | Baseline | Aging-aware |
| LD | 000000 | 111111 | LDUB | 000001 | 000000 |
| LDSB | 001001 | 001100 | LDSH | 001010 | 001001 |
| LDA | 010000 | 001101 | LDUBA | 010001 | 010000 |
| LDUHA | 010010 | 010001 | LDDA | 010011 | 010010 |
| LDSBA | 011001 | 011000 | LDSHA | 011010 | 011001 |
| LDSTUBA | 011101 | 011100 | SWAPA | 011111 | 011110 |
| LDF | 100000 | 011111 | LDFSR | 100001 | 011110 |
| LDC | 110000 | 101111 | LDDC | 110011 | 110010 |
| ST | 000100 | 000011 | STB | 000101 | 000100 |
| STA | 010100 | 010011 | STBA | 010101 | 010100 |
| STHA | 010110 | 010101 | STDA | 010111 | 010110 |
| STF | 100100 | 100011 | STFSR | 100101 | 100100 |
| STDFQ | 100110 | 100101 | STDF | 100111 | 100110 |
| STC | 110100 | 110011 | STCSR | 110101 | 110100 |
| STDCQ | 110110 | 110101 | STDC | 110111 | 110110 |

Hence, we change the encoding of the registers such that the bit positions of the most frequently used registers will have alternative values of '0' and '1'. The new register encoding generated by AISE along with its baseline encoding is shown in Table 4.5.

Table 4.5: New register encoding

| Register id | Baseline | Aging-aware | Register id | Baseline | Aging-aware |
|---|---|---|---|---|---|
| R0/g0 | 00000 | 01010 | R16/l0 | 10000 | 11010 |
| R1/g1 | 00001 | 01011 | R17/l1 | 10001 | 11011 |
| R2/g2 | 00010 | 01100 | R18/l2 | 10010 | 11100 |
| R3/g3 | 00011 | 01101 | R19/l3 | 10011 | 11101 |
| R4/g4 | 00100 | 01110 | R20/l4 | 10100 | 11110 |
| R5/g5 | 00101 | 01111 | R21/l5 | 10101 | 11111 |
| R6/g6 | 00110 | 10000 | R22/l6 | 10110 | 00000 |
| R7/g7 | 00111 | 10001 | R23/l7 | 10111 | 00001 |
| R8/o0 | 01000 | 10010 | R24/i0 | 11000 | 00010 |
| R9/o1 | 01001 | 10011 | R25/i1 | 11001 | 00011 |
| R10/o2 | 01010 | 10100 | R26/i2 | 11010 | 00100 |
| R11/o3 | 01011 | 10101 | R27/i3 | 11011 | 00101 |
| R12/o4 | 01100 | 10110 | R28/i4 | 11100 | 00110 |
| R13/o5 | 01101 | 10111 | R29/i5 | 11101 | 00111 |
| R14/o6 | 01110 | 11000 | R30/i6 | 11110 | 01000 |
| R15/o7 | 01111 | 11001 | R31/i7 | 11111 | 01001 |

### 4.5.3 Reliability Improvement

To emphasize the impact of technology scaling on BTI induced failure, the project is evaluated using three different technology nodes. The reliability improvement is expressed in terms of SNM degradation improvement. We use an industry standard 45nm, and a predictive 32 and 22nm technology node SRAM cells as our experimental model. The cells are evaluated under three different temperature levels. To extract the BTI induced SNM degradation of the SRAM cells we perform a fresh/golden SPICE simulation and a simulation in the presence of 10% BTI induced aging after 3 years. A detail technology specification for the selected cell libraries is provided in Table 4.6. For BTI modeling we

Table 4.6: Simulated transistor technology specification

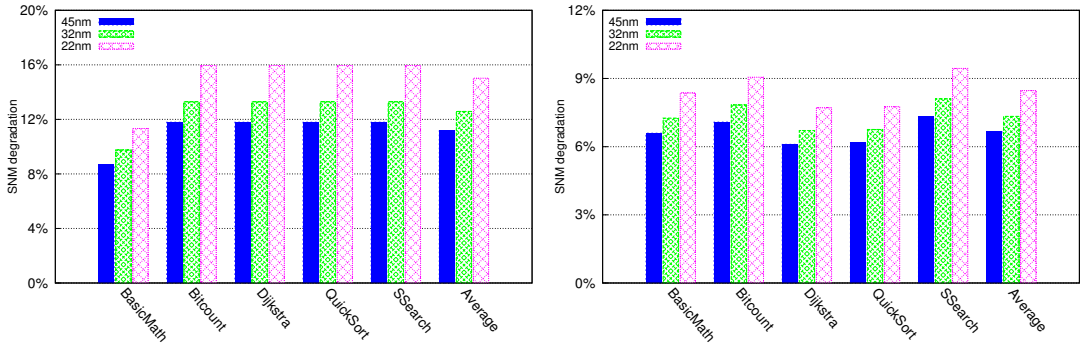
| Spec | Technology models | | |
|---|---|---|---|
| | 45-nm | 32-nm | 22-nm |
| Gate Length | 45nm | 32nm | 22nm |
| Gate Material | Metal | Metal | Metal |
| Gate Dielectric | High-K | High-K | High-k |
| $V_{dd}$ | 1.0V | 0.9V | 0.8V |
| Fresh SNM | 0.157V | 0.124V | 0.093V |

use the long term BTI model given by Equation 2.21 in Chapter 2. The BTI induced aging is evaluated for three different operating temperature levels to analyze the impact of temperature on accelerating BTI induced aging. Before we present the aging improvement of the proposed technique it is important to analyze the impact of scaling and operating temperature in accelerating the BTI induced SNM degradation of an SRAM cell.

#### 4.5.3.1 Scaling Induced SNM Degradation

In chapter 2 we have discussed that aging induced degradation increases when we move to smaller technology nodes. To prove this argument we analyze the impact of BTI induced aging for three different technology nodes. Figure 4.5 shows the impact of technology scaling on the SNM degradation of an instruction cache built using 45, 32 and 22nm SRAM cells for the selected benchmark applications. In memory array BTI affects the metastability of the memory which is determined by the worst case SNM degradation. Metastability determines the stability of the value stored in all cells of a memory [47]. Hence, improving the worst case SNM degradation is very important to improve the stability of an SRAM based memory array. However, from soft error point of view average SNM degradation determines the total number of cells of a memory that fails due to BTI impact [48]. Hence, it is important to analyze the impact of BTI in memory array from metastability and soft-error perspective. In this work the impact of BTI on metastability and soft-error is analyzed and the results are reported as a worst and average case SNM degradation improvements. As we can see from Figure 4.5 there is an average of 15% increase in the worst case SNM degradation by moving from one technology node to the next node. Generally speaking, the SNM degradation increases

by 34% for scaling from 45nm to 22nm technology. This shows how technology scaling is threatening the lifetime reliability of devices.




(a) Worst case (b) Average case

Figure 4.5: Worst and average case SNM degradation of SRAM cell

#### 4.5.3.2 Impact of Temperature on BTI

With scaling of device geometry, the power dissipation is also increasing at a faster rate resulting in an increase in the operating temperature. This increase in the operating temperature eventually leads to an increase in transistors threshold voltage which accelerates the aging of devices. Hence, the increase in device operating temperature produces another big reliability challenge in deep sub-micron regime. Figure 4.6 shows the impact

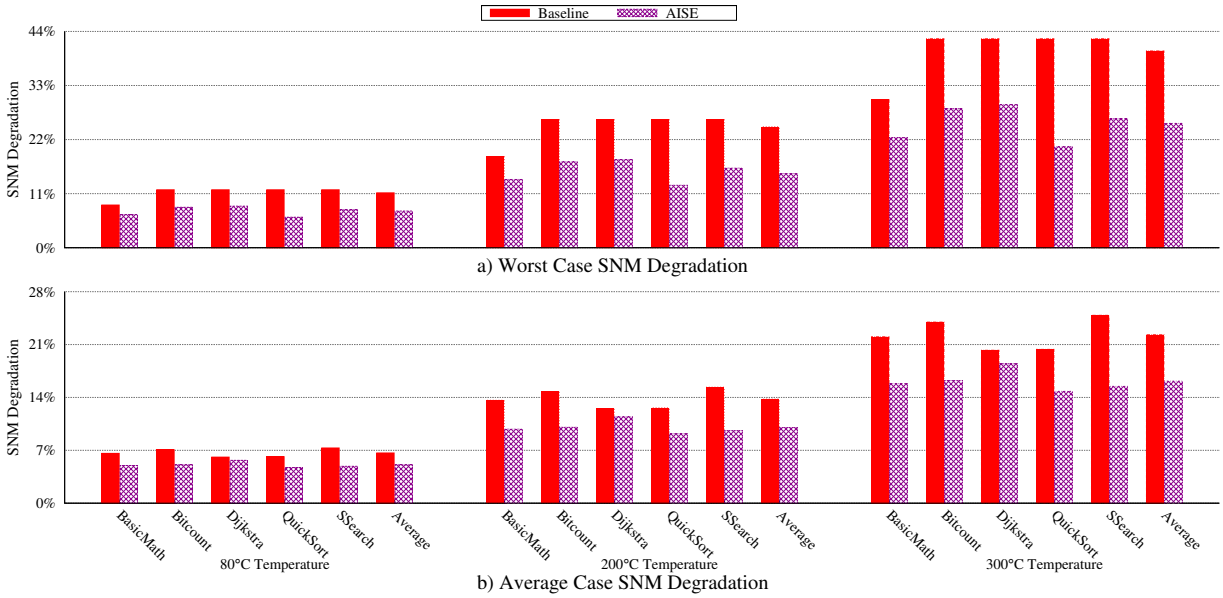



Figure 4.6: Impact of temperature on BTI induced SNM degradation for a 45-nm SRAM cell

of temperature on BTI induced SNM degradation of a 45-nm SRAM cell. As shown in Figure 4.6a the impact of BTI induced aging for an SRAM cell at 200 °C operating temperature is double than a cell under 80 °C operating temperature. The acceleration of BTI induced aging even becomes worst with a higher operating temperature such as 300 °C. On average there is about 1.5X increase in BTI induced aging (Figure 4.6b). The impact of temperature is pronounced with scaling to a smaller technology nodes which can put a reliability limit for further scaling.

#### 4.5.3.3 Reliability Improvement of Aging-aware Encoding

In this section we will present the experimental results of the aging-aware instruction encoding technique. Figure 4.7 shows the worst and average case SNM degradation improvement of AISE over the baseline encoding. From Figure 4.7 we can see on average up to 30% SNM degradation improvement is achieved for a 22 and 45nm technologies and 34% improvement for the 32nm technology. For the 45nm technology the SPARC

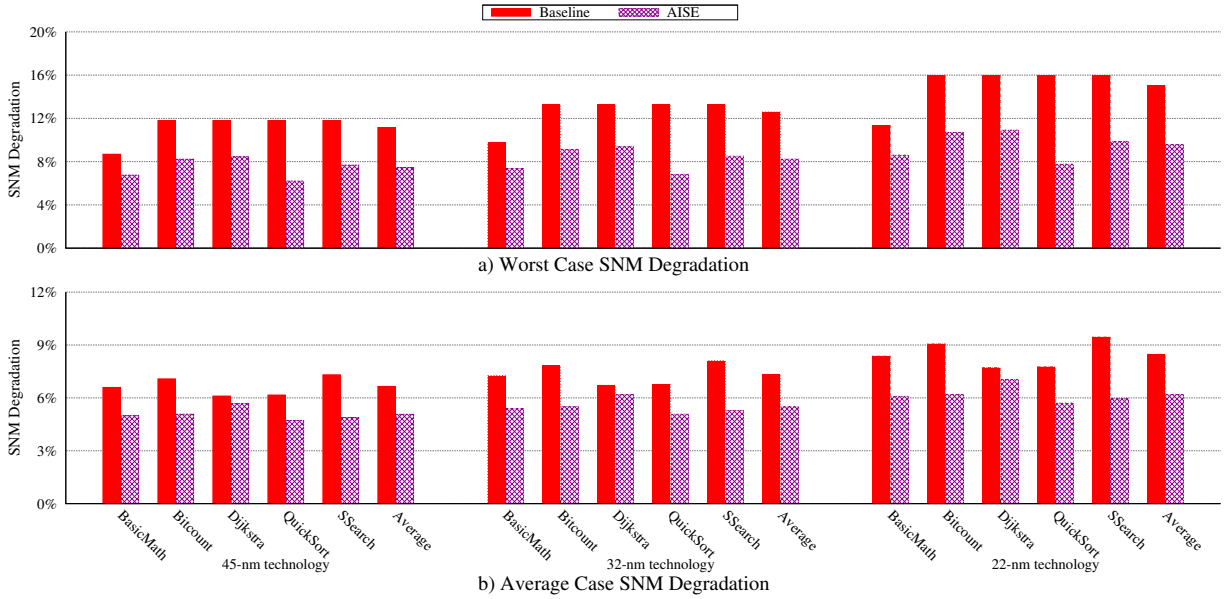



Figure 4.7: Worst and average case SNM improvement using AISE

encoding have an average worst case SNM degradation of 12%. However, using AISE encoding the average worst case SNM degradation is reduced to be 7.5%. This shows that the worst case SNM degradation is improved by 38%.

## 4.6 Comparison of AISE Improvement Alternatives

In section 4.3.2 we have seen that AISE did not address certain fields such as *OP* field of a SPARC instruction. However, two improvement options have been discussed to address the BTI impact on the *OP* field of the instruction cache. Table 4.7 compares the improvements on the *OP* field achieved by the two improvement alternatives in

terms of worst case SNM degradation improvement for a 45nm SRAM cell.

Table 4.7: Comparison of AISE extensions in terms of SNM degradation

| Application | Baseline AISE | Extended AISE | AISE + Bit flipping |
|---|---|---|---|
| Basic math | 7.67% | 4.11% | 5.42% |
| Bit count | 11.79% | 11.79% | 4.34% |
| Dijkstra | 5.27% | 6.43% | 5.66% |
| Quick sort | 6.05% | 6.05% | 4.42% |
| String search | 6.9% | 5.42% | 5.27% |
| Average | 7.54% | 6.76% | 5.02% |

From Table 4.7 we can see that the AISE + Bit flipping approach gives better SNM degradation improvement. However, it induces 2.8% area overhead while the Extended AISE have zero area overhead in the decoder unit. Hence, one can make trade-off between the target reliability and area overhead during instruction encoding to achieve target reliability margin.

To optimize the area overhead of AISE + Bit flipping technique we can use the self-controlled bit flipping technique which will be discussed in the next chapter. Self-controlled bit flipping technique will eliminate the usage of counter and flip control bit. Hence, the area overhead of AISE + Bit flipping can be decreased further to 0.08%.

# Self-controlled Bit Flipping Technique (SCF)

# 5

Depending on the workload the impact of BTI varies from one memory array to other memory array. The technique discussed in the previous chapter address BTI effect in instruction cache only. However, in a modern processor there are various memory components such as data cache, register-file, data and instruction tag caches. To address the impact of BTI induced aging in those memory elements various variants of bit flipping technique such as periodic bit flipping technique, are proposed. In this chapter, we will discuss the implementation of a low cost self-controlled bit flipping technique to address BTI induced aging of different memory arrays. First the concept of self-controlled bit flipping is discussed by using a sample running example. After that an analysis of the state-of-the-art solutions is discussed followed by a detail description of the implementation of the proposed technique. Then the experimental results in comparison with the periodic bit flipping technique is presented. Finally, a comparison of the different techniques and area overhead analysis are discussed.

## 5.1 Concept of Self-controlled Bit Flipping

As discussed in Section 2.5, BTI has a direct impact on SNM degradation of an SRAM cell. Furthermore, SNM is more affected when a cell stores either '0' or '1' for a long period of time i.e. when the signal probability of the cell is very close to '0' or '1'. Hence, balancing the signal probability of the cell is an important task to mitigate BTI induced SNM degradation. In order to balance the signal probability of an SRAM cell, we propose a low cost self-controlled bit flipping technique which selects a bit position that have a signal probability close to 0.5 and use that bit position to flip the other bit positions.

The main idea of self-controlled bit flipping technique is to search for a bit position with signal probability close to 0.5, called flip bit, and flip the content of the memory array during write operation with respect to that bit while that specific bit remains untouched. This means the flip bit is a bit from the input data and depending on the value of that bit, the flipping process will be determined. This technique allow us to implicitly use one of the bit positions as a flip control signal and diminish the requirement of additional column to store flip signal which is implemented by other techniques [6, 49]. During memory read operation, the original value can be obtained by flipping the content of the memory depending on the value of the flip bit.

To explain the proposed technique let us consider the example shown in Figure 5.1(b). In this example, the width of the memory array is 12 bits, and bit position 3 is used as a flip bit. Let us consider two different scenarios: one input with flip bit value '1' and

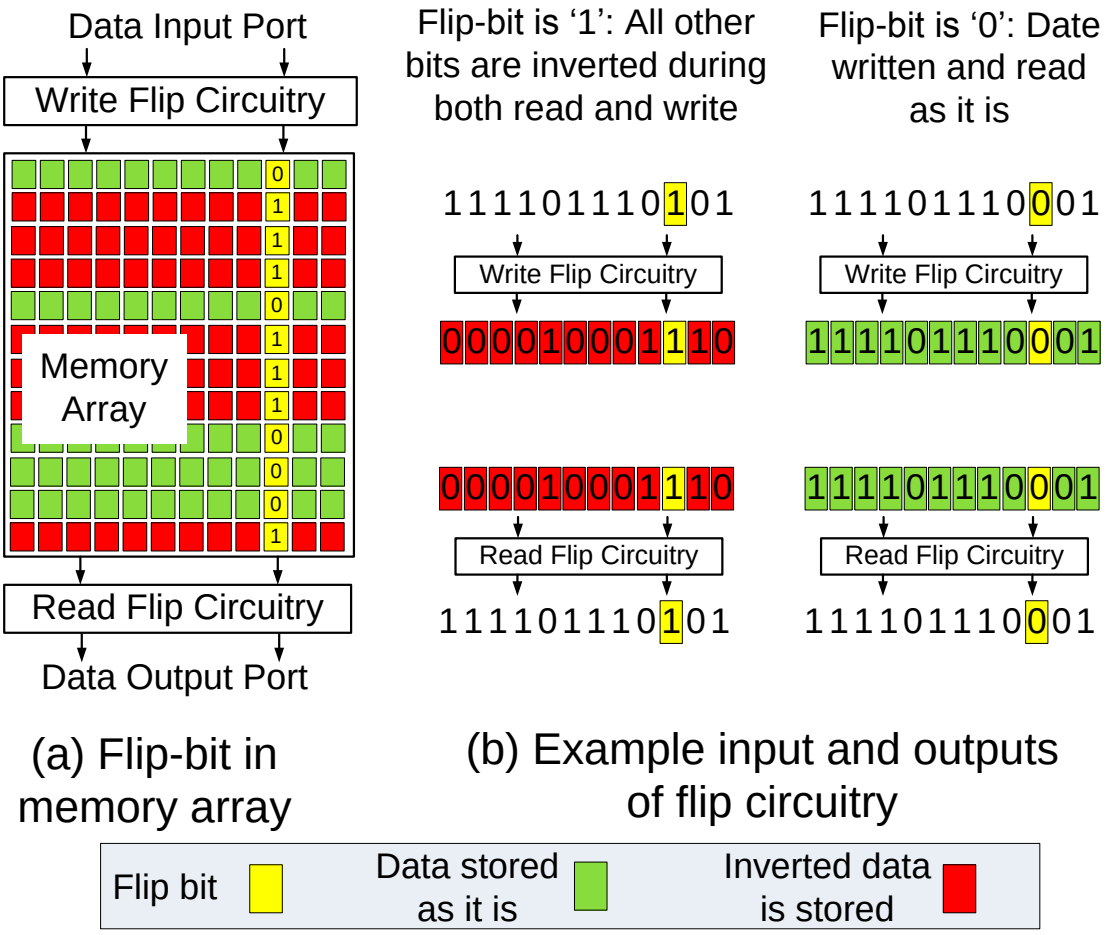



Figure 5.1: Example of self-controlled bit flipping

other input with flip bit value '0'. Figure 5.1(a) shows a memory array along with the added flip circuitry. As we can see from Figure 5.1(a) when the flip bit is '0' the data stored in the memory array is in its original form (Green cells). On the other hand, when the flip bit is '1' the data will be stored in an inverted form (Red cells).

- Flip bit ='1': As shown in Figure 5.1a, first the flip circuitry will invert all the bit positions except the flip bit. After that the flipped values are written to the memory array. The flip bit will be used again to flip the memory content again during read operation to obtain the original input data.

- Flip bit ='0': In this case, the input data will not be inverted during the write operation and the flip bit will be used to control the flipping process during read operation (Figure 5.1b).

## 5.2 Related Works

To address the impact of BTI in memory structures various researchers have proposed a variety of bit flipping based solutions. Authors in [6] proposed a periodic bit flipping technique that periodically invert the data written to a cache memories. Periodic bit flipping use a flip control bit to flip data during read and write operations. Hence, the periodic bit flipping technique requires an additional bit to control the flipping process and an additional counter that determines the flipping period. An NBTI resiliency technique for register-file by classifying the registers into frequent and infrequent registers is discussed in [7]. For the frequent registers the contents of the upper and lower half are swapped or '1' is written in the upper half, and for the infrequent registers they use a periodic bit flipping technique. For the frequent registers three additional flag bits are used to swap the upper and lower half or to write '1' in the upper half. Hence, for systems with a large number of frequent registers the induced area overhead of this technique is higher than the work in [6]. Authors in [49] proposed a short term bit flipping technique which requires a flip signal, and a counter to perform the flip process. Authors in [50] discussed Colt, an approach to balance the stress time of register-file by balancing their usage. Colt uses complemented execution (periodically flipping) to mitigate aging of register-file. Even though Colt uses the flipped register contents without flipping back, it still requires a flip bit and inverters to flip the input data to a register-file. A microarchitectural technique, periodic register rotation (RR) and bit level rotation (BLR), to mitigate the impact of BTI in register-file is discussed in [8]. Both RR and BLR require a counter to determine when to rotate. In comparison to the periodic bit flipping technique, RR induces more area overhead. Moreover, if most of the fields of the register are all '0' or '1' BLR will not be effective. Authors in [51] proposed a technique to writes '0' or '1' to he upper half of the register. Though, this work does not need a flip circuitry, it still requires an additional bit to determine when to write this values. Moreover, this approach addresses only the upper half of the registers.

All of the techniques discussed above require an additional bits either as a counter or as a flip bit to balance the signal probability. Self-controlled bit flipping technique exceeds the stat-of-the-art solutions by minimizing the area overhead without losing the target reliability.

## 5.3 Implementation of Self-controlled Bit Flipping

There are two main steps in the operation of self-controlled bit flipping technique. The first step is the process of selecting the flip bit. Once the flip bit is identified, the second step is the implementation of flip circuitry to invert the data during write operation and invert back during memory read operation depending on the value of the flip bit.

### 5.3.1 Flip Bit Selection

In the proposed technique there are two conditions to be satisfied by a flip bit. First the flip bit need to have an acceptable signal probability (signal probability close to 0.5). However, due to the impact of correlation all bit positions with balanced signal probability may not be effective. Hence, in addition to balanced signal probability the flip bit should not strongly correlate with the other bit positions of the memory array.

**Algorithm 3** Flip control bit selection algorithm

```
1: Building SNM degradation vs. signal probability look-up table
2: Trace data written to each memory array
3: Compute baseline signal probabilities and SNM degradation
4: for bit position i=1 to n do
5:     Select bit i as flip bit
6:     Compute SPs for the current flip-bit
7:     Compute SNM degradation for each bit cell based on its signal probability
8:     Find average SNM degradation
9: end for
10: Select bit position i which have minimum worst-case SNM degradation
```

If the flip bit does not have balanced signal probability then it will lead to unbalanced flipping frequency which can limit the effectiveness of the bit flipping technique. Moreover, if a flip bit is correlated with other bit positions even if it has a good signal probability, the correlation could possibly lead to a noneffective flipping process. For example, there is high degree of correlation between bit positions of an instruction cache. However, in a data cache and register-file the list significant bit (LSB) has a very small degree of correlation with other bit positions.

The flip bit selection process is shown in Algorithm 3. In the beginning, the SNM degradation for different signal probabilities are obtained from SPICE simulation and a lookup table is constructed. Then,the trace of the data written to a memory array during the execution of the workloads in the training set is extracted. Based on the obtained trace, the baseline (i.e., before applying SCF) signal probabilities are computed. Afterwards, the best candidate for the flip bit is selected by exhaustively checking all bit positions acting as the flip bit. In the exhaustive search process, each bit position is selected as a flip bit and signal probabilities are computed by assuming that data written to the memory is inverted when the flip bit position has the value '1'. The next step is to calculate the impact on SNM degradation using the lookup table. In this regard, the extracted signal probabilities are translated to SNM degradation by interpolation of the nearest signal probability samples in the lookup table. Finally, the best flip bit which has the minimum worst-case degradation SNM degradation is selected as flip bit.

To select an optimal flip bit Algorithm 3 is applied to different workloads in a training

set and a bit position that gives smaller worst-case SNM degradation for all applications is selected. Figure 5.2 shows the worst-case SNM degradation of Leon2 register-file by using different flip bit positions for six workloads form the Powerstone benchmark suite. Overally, bit position 2 gives a smaller SNM degradation than the other bit positions. Hence, it will be used as a flip bit for this memory array in our SCF technique.

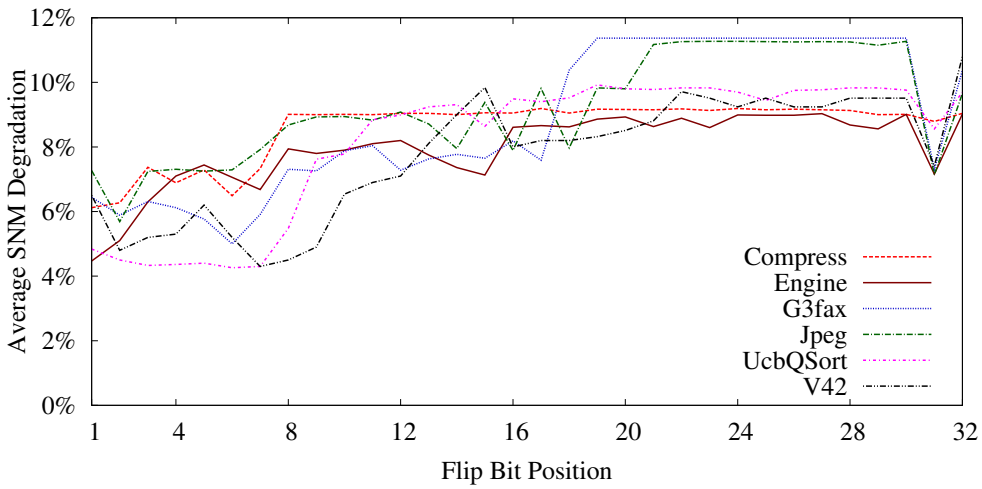



Figure 5.2: SNM improvement using different flip bit positions for register-file of Leon2

### 5.3.2 Flip Circuitry

Once the flip bit is identified the next step of the self-controlled bit flipping technique is to invert the data written to a memory array when the flip bit has value of '1'. To perform this operation an additional flip circuity is required at the write port of the memory array. In order to minimize the area and performance overhead, the additional flip circuitry is implemented by using XOR gates. In order to re-invert the data to its original version during memory read operation, the same flip circuitry should be duplicated at the read port of the memory. Figure 5.3 shows the structure of a memory array along with the additional flip circuitry on the read/write port of the array.
Since the flip bit is used to invert the content of the memory array during read operation, its value should be preserved throughout the process. Hence, for an N-bit memory array our technique requires N-1 XOR gates to invert the N bit input data. In comparison to [49] our work saves two XOR gates in the read/ write flip circuitry of each memory array.

The process of alternatively storing true and inverted data on the memory array balances the signal probability of the memory array which eventually lead to less BTI induced aging of the memory structure. This technique can be easily applied to any memory array by adding flip circuitry in both read and write port without the need of any additional hardware component.

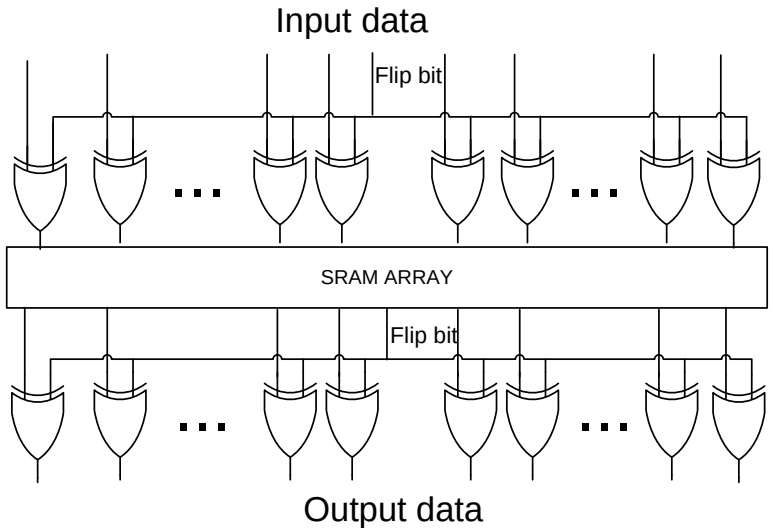



Figure 5.3: Schematic of write flip circuitry; along with flip bit

### 5.3.3 Adaptive SCF

Although we select the flip bit of each memory array with respect to the training workload set at design time, instead of hard-wiring the flip bit, we can add a register to flip circuitry to specify the flip bit. Then, the flip bit can change from time to time during the lifetime based on the application type and expected input data characteristics. This register can be modified on-demand by the firmware, for instance on the boot-up.

## 5.4 Experimental Results

To analyze the reliability improvement of self-controlled bit flipping technique we use the experimental setup discussed in Section 4.4.1. To compare our work with the state-of-the-art solutions we implement a periodic bit flipping technique that inverts the content of the memory array periodically with a flip interval of 128 cycles long. This means that for the write operations occurring in the first 128 cycles, the original data is stored in the memory array while for the second 128 cycles, the inverted value is written. Both self-controlled and periodic bit flipping techniques are applied to different memory structures of the Leon2 processor model. In order to analyze the BTI and soft error impacts, the experimental results presented in this section are expressed in terms of worst case and average case SNM degradation improvement.

In order to show the independency of the flip bits form the training set of workloads, we chose the flip bits according to the results of the six workloads from the Powerstone benchmark suite [52] namely *Compress, Engine, G3fax, Jpeg, UcbQSort, and V42.* Then, we evaluated our technique using a seperate evaluation set of workloads from the MiBench embedded benchmark suite [46] namely *Basicmath, Bitcount, CRC32, Dijkstra, Quicksort, and Stringsearch.* To trace the contents of the caches and register-file, we modified the VHDL model of Leon2 to dump the value and time of each write operation in each memory array. This is very time consuming process since every

memory related activity needs to be recorded. Once the trace is obtained, we exploit our tool to perform the flip bit selection and analyze the signal probability distribution according to Algorithm 3.

After obtaining the signal probabilities, the SNM degradation results are extracted based on SPICE simulations of an SRAM cell designed with a 45 nm PTM technology. First, a fresh SRAM cell is simulated and its SNM is obtained to be 0.157 V. Then, SNM degradation due to BTI is computed for different signal probabilities.

### 5.4.1 Flip Bit Position

Flip bits selected by analyzing training set workloads are reported in Table 5.1. An important point to notice is that in all memory arrays, the flip bit is one of the least significant bits. In the register file and data cache, these bit positions store data and do not strongly correlate with other bit positions. In tag arrays, these bits belong to the least significant bits of the address which are changing more frequently than other bit positions. In instruction cache, these are used to store register addresses or immediate operands.

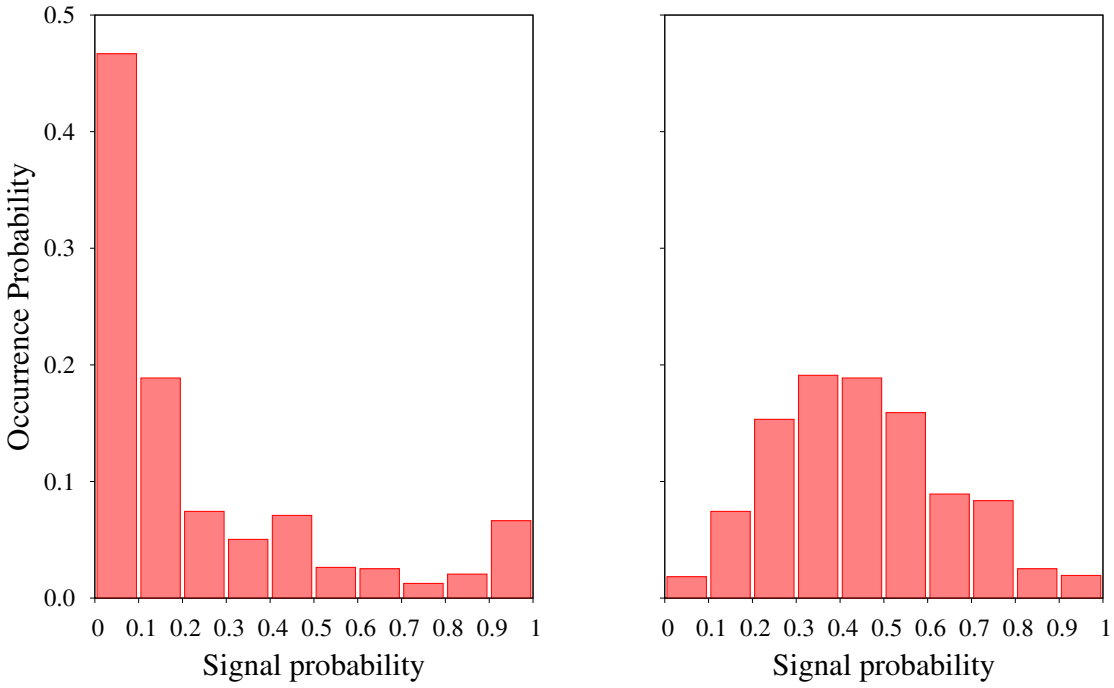
### 5.4.2 Reliability Improvement of Self-controlled Bit Flipping

As discussed in section 2.5, from SNM degradation perspective, it is important to not have cells with signal probability very close to 0 or 1. However, signal probability distribution analysis of SRAM cells in the Leon2 processor during execution of the evaluation workloads shows that majority of the cells have such signal probability (see Figure 5.4a). Applying SCF technique significantly changes signal probability distribution resulting in more desired signal probabilities from aging point of view (see Figure 5.4b).

For the register-file and data cache using Algorithm 3 bit position 1 is found to be the flip control bit. While for the instruction cache bit position 4 is the flip control bit. Bit position 2 and 4 are the flip control bit for data and instruction tag caches respectively. In comparison to the periodic bit flipping technique, our approach achieves almost the same reliability improvement with considerably less area overhead.

Execution time of self-controlled bit flipping algorithm depends on the size of the application and for the selected benchmark applications the maximum execution time of the algorithm is 4.9 Seconds.

Figure 5.5 shows average and worst-case SNM degradation for each memory array of Leon2 processor with respect to different workloads. The proposed SCF technique reduces average SNM degradation of five memory arrays on average by 33.6%, while periodic bit-flipping can reduce it on average by 31.0%. Furthermore, SCF and periodic bit-flipping improve the worst-case SNM degradation on average by 29.7% and 30.6%, respectively. It is clear from this results that these two techniques provide almost




(a) Before applying SCF

(b) After applying SCF

Figure 5.4: Impact of SCF on signal probability distribution of SRAM cells during execution of evaluation set workloads

same level of reliability, however, as we demonstrate later, SCF has smaller area overhead.

Please note that the reported flip control bit position for the memory elements are positions that yield a better result across all the benchmark applications. However, for a specific application we can found other bit position which can give better SNM degradation improvement. Hence, for a memory arrays in an application specific processor the proposed technique can result in a much better SNM degradation improvement.

## 5.5 Area Overhead Analysis

In order to extract area overhead, we employ a UMC memory compiler [53]. This is an industrial tool which provides an abstract information about the latency, area, and power for an arbitrary memory configuration. Using this tool, we build all the memory arrays of the Leon2 processor and for each of which its area is obtained.

As discussed in Section 5.2 the induced area overhead of our technique is the flip circuitry only. Table 5.1 shows the memory area, area of the flip circuitry and the area overhead in percent. From Table 5.1 we can see that the total area overhead induced by the periodic bit flipping is 4.8% and the total area overhead of self-controlled bit flipping technique is 1.7%.

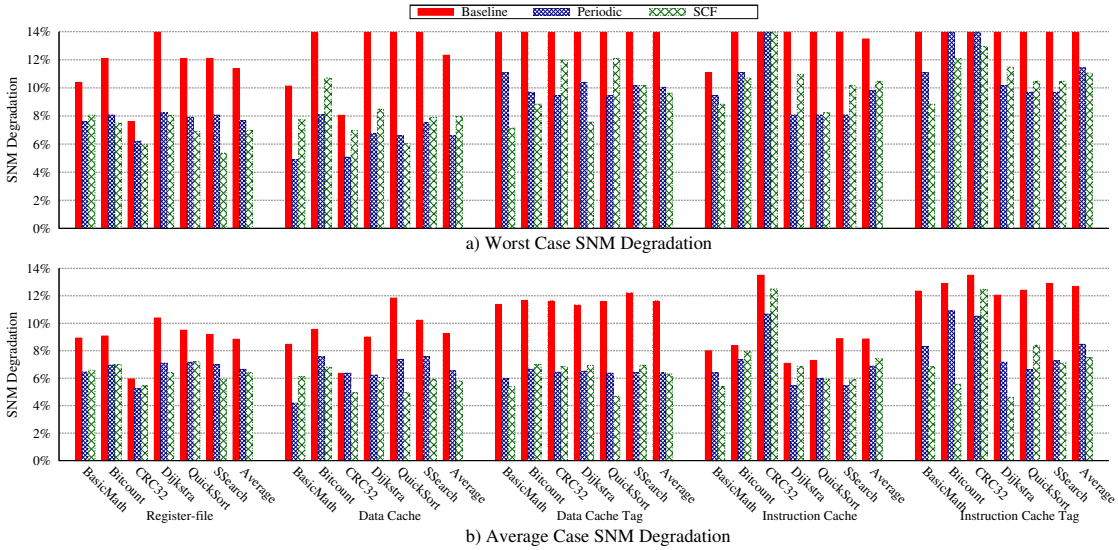



Figure 5.5: Average and worst case SNM degradation of memory arrays; Average and worst-case represents average and maximum SNM degradation of all cells in the memory array

Table 5.1: Area overhead of SCF and periodic bit-flipping technique; Area unit is $\mu^2 m$

| Memory component | Size | Flip bit | Baseline | Periodic bit-flipping | | | Self-controlled bit-flipping | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Memory area | Memory area | Flip circuitry | Over-head | Memory area | Flip circuitry | Over-head |
| Register file | 256×32 | 2 | 13,457 | 13,781 | 397 | 5.4% | 13,457 | 322 | 2.4% |
| Data cache | 1,024×32 | 3 | 26,811 | 27,486 | 3,967 | 4% | 26,811 | 322 | 1.2% |
| Instruction cache | 1,024×32 | 4 | 26,811 | 27,486 | 397 | 4% | 26,811 | 322 | 1.2% |
| Data tag | 256×25 | 2 | 8,557 | 8,819 | 324 | 6.8% | 8,557 | 249 | 2.9% |
| Instruction tag | 256×25 | 5 | 8,557 | 8,819 | 324 | 6.8% | 8,557 | 249 | 2.9% |
| **Total area** | | | 84,196 | 86,392 | 1,838 | **4.8%** | 84,196 | 1,466 | **1.7%** |

Overall self-controlled bit flipping is able to achieve up to 64% improvement of the area overhead induced by the conventional periodic bit flipping technique while having comparable SNM degradation improvement.

## 5.6 Performance Overhead

Both SCF and the periodic bit-flipping techniques increase the memory read and write access latency as they insert a flip circuitry at the inputs and output ports of the memory array. In order to accurately estimate the impact of SCF on the processor performance, we synthesized the original implementation as well as the modified implementation (i.e., with flip circuitry at the input and output ports of the memory arrays) of the Leon2 processor to obtain the minimum possible clock period. The timing analysis results show that applying SCF to this processor increases the minimum possible clock period from 1.241 ns to 1.255 ns. This means that SCF increases the processor clock period

by 1.13%. The synthesis result of Leon2 processor with periodic bit-flipping technique shows exactly the same performance penalty which is reasonable as it has a similar flip circuitry structure.

## 5.7 Power Overhead

Both SCF and the periodic bit-flipping techniques increase the energy consumption of the memory array to achieve a balanced signal probability. In order to investigate the power overhead of these two techniques, we extracted the number of read and write accesses as well as the average switching activity during write operations from the trace files. Then, this information is combined with the per access dynamic power and static power consumption reported by UMC memory compiler to obtain the overall power consumption.
Figure 5.6 shows the power overhead of SCF and the periodic bit-flipping technique to each on-chip memory array of Leon2. As it can be seen, SCF reduces the power overhead imposed by periodic bit-flipping technique on average by 27.6% as it does not require an additional bit per word to store the inversion flag.

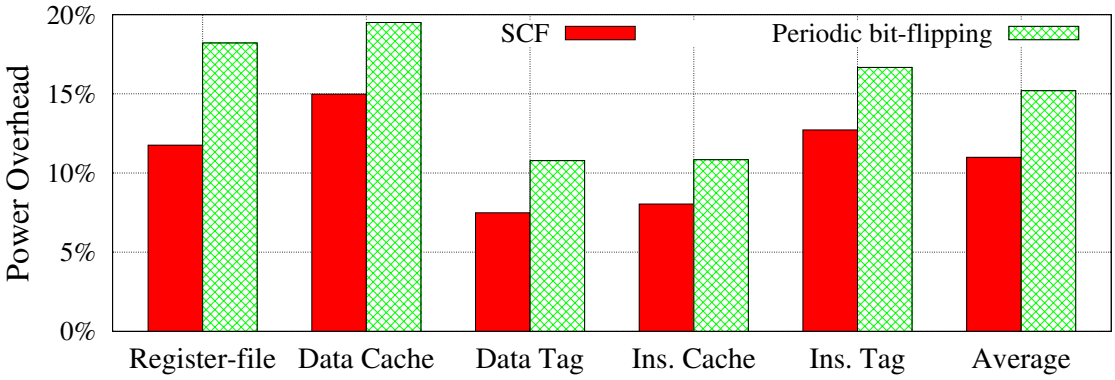



Figure 5.6: Power overhead of SCF and periodic bit-flipping

# Conclusion and Recommendations

# 6

With further scaling in CMOS devices lifetime reliability of microprocessors is becoming more and more challenging. The lifetime reliability challenges of a nano scale devices are rooted from the aging induced failures such as BTI and HCI. Additionally, on field failures soft errors and variations are increasing from one generation to the next generation. Hence, scaling to a nano scale regime will finally lead to unacceptable devices reliability. Today various researchers are working hard to overcome these scaling induced lifetime reliability challenges and they suggest different design alternatives such as over designing, redundancy of functional units and guard banding. However, all the above mentioned techniques are used to extend the lifetime of devices not to mitigate the failure mechanisms. Since storage elements occupy majority of the area of modern microprocessors BTI induced aging have significant impact on memory elements. In memory structures BTI leads to SNM degradation, which makes the memory less stable and increase its failure rate. BTI is a workload dependent which leads to a non-uniform degradation rate of the memory arrays.

## 6.1 Conclusion

In this thesis we showed two different approaches to address the impact of BTI induced aging in embedded memory arrays. The first approach which is first of its kind is to identify an instruction encoding that imposes less BTI impact on instruction cache. This technique is implemented using a C++ tool that iteratively checks different encoding options and analyze their impact on aging of the instruction cache and then finally select an aging optimal encoding among the possible options. Once the aging-aware encoding is identified both the assembler from the compiler toolset and the decoder unit of the microprocessor need to be modified to implement the new aging-aware encoding. Using this technique we showed that on average up to 30% SNM degradation improvement can be achieved by imposing a considerably less area overhead.

In Chapter 5 we discussed a low cost self-controlled bit flipping technique to address the impact of BTI induced aging in all embedded memory arrays. This technique mitigates the impact of BTI by flipping the data written to a memory arrays depending on the value of the flip bit. Since the flip bit is selected from the available bit positions no additional bit is required to store the flip bit. By using a self-controlled bit flipping technique we achieve up to 33% SNM degradation improvement. In comparison to the stat-of-the-art periodic bit flipping technique our approach have 64% less area overhead.

In conclusion, with scaling in to deep sub-micron regime BTI induced aging is increasing which leads to an increased memory failures. BTI impact is also accelerated by various

environmental factors such as operating temperature. We have demonstrated the impact of operating temperature on accelerating BTI induced aging by a factor of 1.5. Hence, the design of robust systems is becoming more and more challenging.

## 6.2 Contribution of The Results

An aging-aware instruction encoding can be applied to an instruction cache of any embedded processor whether a single core or multi/many core with shared or distributed memory architecture. The only requirement for applying aging-aware encoding is the assembler and decoder unit, which implements the SPARC ISA, need to be modified. In a similar fashion the same technique can be applied to other ISA in order to identify an aging-aware encoding of the desired ISA.

Self-controlled bit flipping technique on the other hand, is easy to implement and it does not have any compatibility requirements. In contrast to the aging-aware encoding this technique impose higher area overhead, however, unlike the aging-aware encoding which is applicable only to instruction cache, this technique can be applied to any memory structures. In addition to the area overhead, self-controlled bit flipping technique also have some performance overhead due to the requirement of conditionally inverting the data on every read/write access of the memory array.

## 6.3 Recommendation and Future Work

To the best of our knowledge AISE is the first attempt to address BTI induced aging of instruction cache by changing the instruction encoding of an ISA. To achieve a better aging improvement AISE can be further extended to cover all SPARC instructions regardless of their usage frequency. Generally speaking, AISE can be improved in the following ways:

1. Covering all SPARC instructions: Currently due to the nature of the application and width of the instruction fields AISE generates an aging-aware encoding for the opcode and register field encoding of an arithmetic and load/store instructions. However, AISE can be easily modified to address all the SPARC instructions and generate an entirely new encoding for all SPARC instructions.

2. Applying AISE for other ISA: To address the impact of BTI in embedded memories of other architectures such as MIPS and ALPHA; AISE can be easily applied to generate their equivalent aging-aware instruction encoding. However, depending on the instruction format of the target architecture the implementation of AISE can be modified but the concept remains the same.

For self-controlled bit flipping technique on the other hand, the ISA implementation is not an issue. However, the existence of correlation between the bit positions of a memory array puts a limit on the effectiveness of the algorithm. This problem was demonstrated by applying the self-controlled bit flipping technique to an instruction cache in which the fields of the instruction are highly correlated with each other. The correlation barrier

can be addressed by using multi-flip control bit technique in the cost of small area overhead. In the current implementation of self-controlled bit flipping approach only one bit position of the memory array was selected as a flip control bit. However, in a multi-flip control bit implementation two different bit positions will be selected as a flip control bit. However, with this approach an additional multiplexer is required at the write and read port of the memory array to control the flip signal based on the flip control bits.

# Bibliography


[1] S. K. . S. Hamdiou, *Semiconductor Reliability, Lecture Slides.*

[2] "Moor's Law," http://www.mooreslaw.org/, 2013, [Online; accessed 15-January-2014].

[3] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The impact of technology scaling on lifetime reliability," in *Dependable Systems and Networks, 2004 International Conference on.* IEEE, 2004, pp. 177–186.

[4] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula, "Predictive modeling of the nbti effect for reliable design," in *Custom Integrated Circuits Conference, 2006. CICC'06. IEEE.* IEEE, 2006, pp. 189–192.

[5] F. Oboril and M. B. Tahoori, "Extratime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on.* IEEE, 2012, pp. 1–12.

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

[7] H. Amrouch, T. Ebi, and J. Henkel, "Stress balancing to mitigate nbti effects in register files," in *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on.* IEEE, 2013, pp. 1–10.

[8] S. Kothawade, K. Chakraborty, and S. Roy, "Analysis and mitigation of nbti aging in register file: An end-to-end approach," in *Quality Electronic Design (ISQED), 2011 12th International Symposium on.* IEEE, 2011, pp. 1–7.

[9] M. Bagatin, S. Gerardin, A. Paccagnella, and F. Faccio, "Impact of NBTI aging on the single-event upset of SRAM cells," *Transactions on Nuclear Science*, vol. 57, no. 6, pp. 3245–3250, 2010.

[10] A. Bansal, R. Rao, J. Kim, S. Zafar, J. Stathis, and C. 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.

[11] S. Khan, "Bias temperature instability analysis, monitoring and mitigation for nano-scaled circuits," Ph.D. dissertation, Delft University of Technology, 2013.

[12] S. Bhunia, S. Mukhopadhyay, and K. Roy, "Process variations and process-tolerant design," in *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on.* IEEE, 2007, pp. 699–704.

[13] O. S. Unsal, J. W. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin, "Impact of parameter variations on circuits and microarchitecture," *Micro, IEEE*, vol. 26, no. 6, pp. 30–39, 2006.

[14] S. N. M. Pedram, "Crosstalk-affected propagation delay in nanometer technologies."

[15] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings of the 40th annual Design Automation Conference*. ACM, 2003, pp. 338–342.

[16] "Latch-up in CMOS Designs," http://www.ece.drexel.edu/courses/ECE-E431/latch-up/latch-up.html, 2013, [Online; accessed 10-June-2014].

[17] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 389–398.

[18] M. Ohring, *Reliability and failure of electronic materials and devices*, 1998.

[19] R. Degraeve, M. Aoulaiche, B. Kaczer, P. Roussel, T. Kauerauf, S. Sahhaf, and G. Groeseneken, "Review of reliability issues in high-k/metal gate stacks," in *Physical and Failure Analysis of Integrated Circuits, 2008. IPFA 2008. 15th International Symposium on the*. IEEE, 2008, pp. 1–6.

[20] K. O. Jeppson and C. M. Svensson, "Negative bias stress of mos devices at high electric fields and degradation of mnos devices," *Journal of Applied Physics*, vol. 48, no. 5, pp. 2004–2014, 1977.

[21] M. A. Alam and S. Mahapatra, "A comprehensive model of pmos nbti degradation," *Microelectronics Reliability*, vol. 45, no. 1, pp. 71–81, 2005.

[22] M. Alam, B. Weir, and P. Silverman, "The prospect of using thin oxides for silicon nanotransistors," in *Gate Insulator, 2001. IWGI 2001. Extended Abstracts of International Workshop on*. IEEE, 2001, pp. 30–34.

[23] B. Kaczer, V. Arkbipov, R. Degraeve, N. Collaert, G. Groeseneken, and M. Goodwin, "Disorder-controlled-kinetics model for negative bias temperature instability and its experimental verification," in *Reliability Physics Symposium, 2005. Proceedings. 43rd Annual. 2005 IEEE International*. IEEE, 2005, pp. 381–387.

[24] R. Rodriguez, J. Stathis, B. Linder, S. Kowalczyk, C. Chuang, R. Joshi, G. Northrop, K. Bernstein, A. Bhavnagarwala, and S. Lombardo, "The impact of gate-oxide breakdown on sram stability," *Electron Device Letters, IEEE*, vol. 23, no. 9, pp. 559–561, 2002.

[25] J. McPherson and H. Mogul, "Underlying physics of the thermochemical e model in describing low-field time-dependent dielectric breakdown in sio 2 thin films," *Journal of Applied Physics*, vol. 84, no. 3, pp. 1513–1523, 1998.

[26] I. T. R. map for Semiconductors, *International Technology Road-map for Semiconductors - 2003 Edition.* ITRS, 2003, vol. 03, last checked: 05.02.2014. [Online]. Available: http://public.itrs.net/

[27] R. Degraeve, B. Kaczer, and G. Groeseneken, "Reliability: a possible showstopper for oxide thickness scaling?" *Semiconductor Science and Technology*, vol. 15, no. 5, p. 436, 2000.

[28] J. McPherson and D. Baglee, "Acceleration factors for thin gate oxide stressing," in *Reliability Physics Symposium, 1985. 23rd Annual.* IEEE, 1985, pp. 1–5.

[29] R. Duschl and R.-P. Vollertsen, "Is the power-law model applicable beyond the direct tunneling regime?" *Microelectronics Reliability*, vol. 45, no. 12, pp. 1861–1867, 2005.

[30] J. C. Lee, C. Ih-Chin, and H. Chenming, "Modeling and characterization of gate oxide reliability," *Electron Devices, IEEE Transactions on*, vol. 35, no. 12, pp. 2268–2278, 1988.

[31] C. H. K. John Keane, *transistor-aging.* IEEE, 2011, vol. 1, last checked: 05.02.2014. [Online]. Available: spectrum.ieee.org/semiconductors/processors/transistor-aging

[32] F.-C. Hsu and S. Tam, "Relationship between mosfet degradation and hot-electron-induced interface-state generation," *Electron Device Letters, IEEE*, vol. 5, no. 2, pp. 50–52, 1984.

[33] D. K. Slisher, R. Filippi, D. W. Storaska, and A. H. Gay, "Scaling of si mosfets for digital applications," *Rensselaer Polytechnic Institute*, 1999.

[34] C. Hu, S. C. Tam, F.-C. Hsu, P.-K. Ko, T.-Y. Chan, and K. W. Terrill, "Hot-electron-induced mosfet degradation-model, monitor, and improvement," *Solid-State Circuits, IEEE Journal of*, vol. 20, no. 1, pp. 295–305, 1985.

[35] S. E. Rauch III and G. La Rosa, "The energy-driven paradigm of nmosfet hot-carrier effects," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 4, pp. 701–705, 2005.

[36] P. Fantini, A. Ghetti, A. Marinoni, G. Ghidini, A. Visconti, and A. Marmiroli, "Giant random telegraph signals in nanoscale floating-gate devices," *Electron Device Letters, IEEE*, vol. 28, no. 12, pp. 1114–1116, 2007.

[37] A. Ohata, A. Toriumi, M. Iwase, and K. Natori, "Observation of random telegraph signals: Anomalous nature of defects at the si/sio¡ inf¿ 2¡/inf¿ interface," *Journal of applied physics*, vol. 68, no. 1, pp. 200–204, 1990.

[38] K. B. Klaassen and J. C. L. Van Peppen, "System reliability," *Edward Arnold*, 1989.

[39] J. R. Black, "Electromigrationa brief survey and some recent results," *Electron Devices, IEEE Transactions on*, vol. 16, no. 4, pp. 338–347, 1969.

[40] S. Khan, I. Agbo, S. Hamdioui, H. Kukner, B. Kaczer, P. Raghavan, and F. Catthoor, "Bias temperature instability analysis of finfet based sram cells," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 31.

[41] "Instruction set," http://www.computerhope.com/jargon/i/instset.htm, 2014, [Online; accessed 28-May-2014].

[42] *The SPARC Architecture Manual, Version 8*. Menlo Park, CA: Prentice-Hall, Inc, 1992.

[43] S. Woo, J. Yoon, and J. Kim, "Low-power instruction encoding techniques," in *SOC Design Conference*. Citeseer, 2001.

[44] I. Kadayif and M. T. Kandemir, "Instruction compression and encoding for low-power systems," in *ASIC/SOC Conference, 2002. 15th Annual IEEE International*. IEEE, 2002, pp. 301–305.

[45] F. Oboril and M. Tahoori, "Arise: Aging-aware instruction set encoding for lifetime improvement."

[46] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*. IEEE, 2001, pp. 3–14.

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

[48] D. Rossi, M. Omaña, C. Metra, and A. Paccagnella, "Impact of aging phenomena on soft error susceptibility," in *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 18–24.

[49] Y. Kunitake, T. Sato, and H. Yasuura, "Signal probability control for relieving nbti in sram cells," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*. IEEE, 2010, pp. 660–666.

[50] E. Gunadi, A. Sinkar, N. Kim, and M. Lipasti, "Combating aging with the colt duty cycle equalizer," in *International Symposium on Microarchitecture*, 2010, pp. 103–114.

[51] S. Wang, T. Jin, C. Zheng, and G. Duan, "Low power aging-aware register file design by duty cycle balancing," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2012, pp. 546–549.

[52] J. Scott, L. H. Lee, A. Chin, and B. Moyer, "Designing the m· core tm m3 cpu architecture," in *Computer Design, 1999.(ICCD'99) International Conference on*. IEEE, 1999, pp. 94–101.

[53] "UMC Memory Maker," www.umc.com, 2014, [Online; accessed 20-May-2014].

# List of definitions

# Leon2 VHDL model modification

# A

This appendix presents the main vhdl codes that are added to the vhdl description of Leon2 processor to trace the data written to different memory array. The function bit_to_integer is added to convert the address of memeory array from std_logic_vector to integer. The process clk_cnt is used to count the number of clocks and determine the clock cycle that the data is written to memory array. This process and the bit_to_integer function are global to all memory arrays. Each memory array have its own Trace process that keep trace the data written to the memory array.

```
-----------------------------------------------------------------------
-- Additional packages for file processing
-----------------------------------------------------------------------
use std.textio.all;
use ieee.std_logic_textio.all;

-----------------------------------------------------------------------
--        Signals and file pointer declaration
-----------------------------------------------------------------------
signal clk_counter : integer := 0;  -- counter used to count the clock
signal address_int : integer;
FILE output_data   : text OPEN WRITE_MODE IS "memorytrace.txt";

-----------------------------------------------------------------------
--function to convert address from std_logic_vector to integer
-----------------------------------------------------------------------
function bit_to_integer(input : in std_logic_vector) return integer is
   variable i           : integer := 0;
   variable sum         : integer := 0;
   variable product : integer := 1;
 begin
        while(i < input'length) loop
          if(input(i) = '1') then
                sum := sum + product;
          end if;
          product := product * 2;
          i := i + 1;
        end loop;
 return sum;
end bit_to_integer;
```

```
-------------------------------------------------------------------
--                      process to trace clock
-------------------------------------------------------------------
 clk_cnt: process (clk)
  begin
    if rising_edge(clk) then
         clk_counter <= clk_counter + 1;
    end if;
  end process;
```

The process given below is sensitive to write enable signal of the memory array and a rising clock transition because data is written or read to/ from memory array always on positive clock edge.

```
-------------------------------------------------------------------
-- process to trace the data's written to memory array
-------------------------------------------------------------------
Trace: process (idwrite,clk)
    variable write_line : LINE;
    begin
         if rising_edge(clk) then
          if (idwrite(i)='1') then
                report "The process is icache being executed! ";
                address_int <=bit_to_integer(address);
                write(write_line, address_int);
                write(write_line, ' ');
                write(write_line, iddatain);
                write(write_line, ' ');
                write(write_line, clk_counter);
                writeline(output_data, write_line);
            end if;
         end if;
    end process;
```

# Publishable papers

# B

## B.1 Paper 1: Aging Mitigation in Memory Arrays Using Self-controlled Bit-flipping Technique

Abstract:
downscaling CMOS technologies into the nanometer regime, the reliability of SRAM memories is threatened by accelerated transistor aging mechanisms such as Bias Temperature Instability (BTI). BTI leads to a considerable degradation of SRAM cell Static Noise Margin (SNM), which increases the memory failure rate. Since BTI is workload dependent, the aging rates of different cells in a memory array are quite nonuniform. To address this issue, a variety of bit-flipping techniques has been proposed to decrease the SNM degradation by balancing the signal probabilities of the cells. However, existing bit-flipping techniques impose too much overhead as at least an additional column is required to store the inversion flags. In this paper, we propose a low cost self-controlled bit-flipping technique which inverts all bit positions with respect to an existing bit. This technique is applied to a register-file and cache units of an embedded microprocessor. Our simulation results show that the reliability of the proposed technique is similar to that of existing bit-flipping techniques, while imposing 64% less area overhead.

## B.2 Paper 2: Instruction Cache Aging-aware Instruction Encoding

Abstract:
The reliability of small geometry device is threatened by accelerated transistor aging. Bias Temperature Instability (BTI) is the dominant aging mechanism in a nanoscale devices. In memory arrays BTI degrades the Static Noise Margin (SNM), which in turn accelerates memory failure rate. To address BTI induced aging of memory structures, various bit flipping based hardware and software solutions has been proposed to balance the signal probability of SRAM cells which in turn will decrease the SNM degradation. However, all of the techniques either require an additional hardware or impose considerable performance overhead. In this pa- per, we propose an instruction encoding technique which finds an aging optimal instruction encoding to mitigate BTI induced aging of instruction cache. First, the aging impact and improvement opportunities of the baseline encoding is analyzed. Afterwards, a new encoding is identified to balance the signal probability of the SRAM cells in an instruction cache. Our simulation results show that the proposed scheme achieves up to 33% instruction cache SNM degradation improvement.